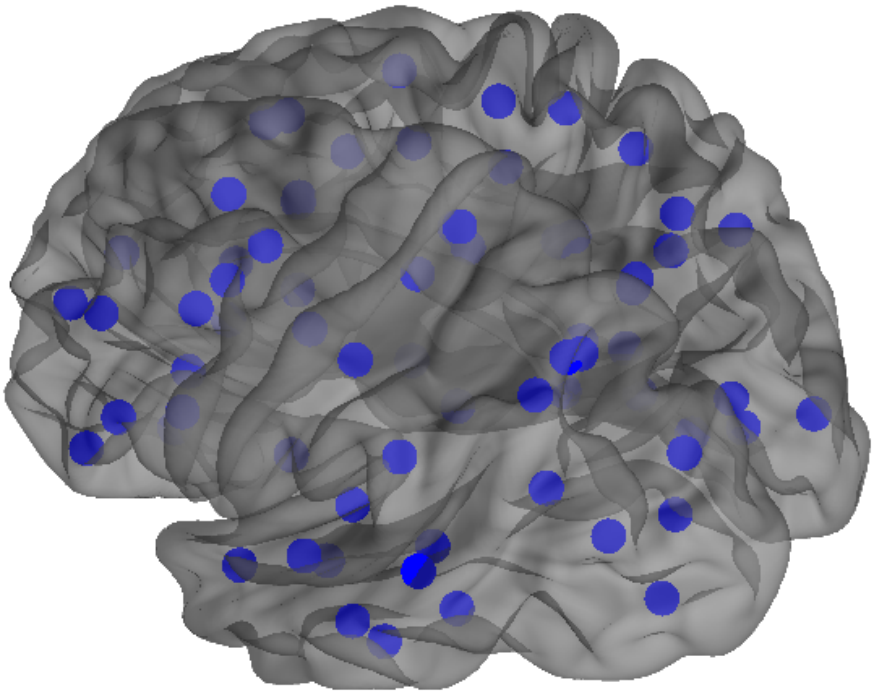
CHALMERS
UNIVERSITY OF TECHNOLOGY



Graph Convolutional Neural Networks for Brain Connectivity Analysis

Master's thesis in Complex Adaptive Systems

Lars Jansson & Tobias Sandström

Department of Physics
CHALMERS UNIVERSITY OF TECHNOLOGY
Gothenburg, Sweden 2020

Master's thesis 2020

Graph Convolutional Neural Networks for Brain Connectivity Analysis

Lars Jansson & Tobias Sandström

CHALMERS
UNIVERSITY OF TECHNOLOGY

Department of Physics
Chalmers University of Technology
Gothenburg, Sweden 2020

Graph Convolutional Neural Networks for Brain Connectivity Analysis
Lars Jansson & Tobias Sandström

Supervisor: Jonas Andersson, Syntronic AB
Examiner: Giovanni Volpe, Department of Physics GU

Master's Thesis 2020
Department of Physics
Chalmers University of Technology
SE-412 96 Gothenburg
Telephone +46 31 772 1000

Cover: Brain Atlas generated in BRAPH 1.0.0 (`http://braph.org/`).

Typeset in LaTeX
Printed by Chalmers Reproservice
Gothenburg, Sweden 2020

Graph Convolutional Neural Networks for Brain Connectivity Analysis
Lars Jansson & Tobias Sandström
Department of Physics
Chalmers University of Technology

Abstract

We explore the potential and limitation of Graph Convolutional Neural Networks (GCNs) for classification of brain graphs derived from MRI measurements of subjects with Alzheimer's disease (AD). GCNs differ from regular Artificial Neural Networks (ANNs) in that they operate directly on graph structures by defining convolutional operators in a non-Euclidean space. We show that GCNs perform well on graph-structured data, where regular ANNs typically fail due to the arbitrary ordering of nodes. Different GCN architectures are examined and compared to a Fully Connected Feedforward Neural Network. Tests are initially performed on simulated graphs mimicking the human brain. The simulated brain graphs were generated by following an algorithmic approach parameterized by graph measures known to be important for characterizing brain graphs. We demonstrate that GCNs are vital in accurately classifying the simulated brain graphs. The GCNs' performance is evaluated on structured MRI-data, displaying cortical thicknesses for 68 regions in the brain of patients with Alzheimer's disease and a healthy control group. On the structured brain data, both GCNs and regular ANNs are shown to be able classifiers. Finally, we show that the performance of regular ANNs is completely dependent on a fixed ordering imposed by the brain graph derivation from the MRI-data. In contrast, we show that GCNs perform well independent of node order.

Keywords: Machine Learning, Graph Neural Networks, Graph Convolutional Neural Networks, Supervised Learning, Brain Connectivity, Alzheimer's Disease

Acknowledgements

First of all we would like to thank our supervisors at Syntronic AB, Jonas Andersson and Alice Deimante Neimantaite. Jonas, without our many discussions and frenetic white board sessions, this would never have been possible. And Alice, without your guidance, the neurological parts of this thesis would most certainly have been lost in confusion. Also, a big thanks to Joana Pereira, *KI*, for providing us with the MRI-data, and to Giovanni Volpe, GU, for being the captain of this successful collaboration. Finally, we would like to thank Syntronic AB, and all the people at the Gothenburg office. Not only for providing the project in the first place, but also for making us feel at home during these six months, despite the ongoing global pandemic.

Lars Jansson & Tobias Sandström, Gothenburg, June 2020

Declaration of contributions

Data used in the preparation of this thesis were obtained from the Alzheimer's Disease Neuroimaging Initiative (ADNI) database (`http://adni.loni.usc.edu`). Led by the principal investigator Michael W. Weiner, MD, the ADNI was launched in 2003 as a public-private partnership. The primary goal of ADNI has been to test whether serial *Magnetic Resonance Imaging* (MRI), *Positron Emission Tomography* (PET), other biological markers, and clinical and neuropsychological assessment can be combined to measure the progression of *Mild Cognitive Impairment* (MCI) and early AD. A complete listing of ADNI investigators can be found at: `http://adni.loni.usc.edu/wp-content/uploads/how_to_apply/ADNI_Acknowledgement_List.pdf`. All participants included in this thesis underwent 3T MRI scanning using a T1-weighted *Magnetization Prepared Rapid Gradient Echo* (MPRAGE) sequence. T1-weighted images were preprocessed with the FreeSurfer software (`https://surfer.nmr.mgh.harvard.edu/`), which provided the cortical thickness values of 68 cortical regions from the Desikan atlas (Desikan et al.[1]).

Contents

List of Figures

List of Tables

1

Introduction

In recent years, graph-structured data has received more and more attention from the machine learning community.[2][3][4] Graphs are applicable in a wide variety of research fields, due to their immense expressive power as a data structure. Some examples of research fields where graphs naturally emerge are social network theory[5][6], molecular chemistry[7][8], and natural language processing[9][10]. Another field where graphs naturally emerge is neurology. Connectivity between different brain regions is conveniently described in a *brain graph*, where nodes represent regions and edges connections between them. Neurodegenerative diseases such as *Alzheimer's disease* (AD) and *Parkinson's disease* (PD) are both characterized by a loss of neurons and synapses in the brain.[11][12] With the use of modern brain imaging techniques, neurodegenerative diseases can be studied from a graph theoretical point of view.[13][14][15][16]

Many different tasks involve learning from graph-structured data. For *link prediction* tasks, the goal is to predict missing connections in a graph, whereas for *node classification* tasks, the goal is to predict unknown node values. Both these problems are usually approached using *unsupervised* or *semi-supervised* learning techniques, by inferring patterns and structures from known parts of the graph.[17][18][19][20] In this thesis, we will focus on *Graph classification*, which is the task of distinguishing graphs of different classes. The classification task is approached in a *supervised* setting on simulated data mimicking the human brain, as well as on AD patient data derived from MRI.

1.1 Graph Neural Networks

An *Artificial Neural Network* (ANN) is typically designed to operate on tensor structured data, i.e., on an ordered container of numbers. The fact that this ordering isn't arbitrary is often crucial for the ANNs' ability to learn. As an example, say an ANN has been successfully trained to classify images of cats and dogs. If the pixels in each of the input samples were shuffled, the ANN would no longer be able to tell the two categories of animals apart. This implies that not only is information contained in the individual pixel values in the image, but also in the pixels' position relative to one another.

In the most general sense, graph-structured data lacks order. A graph is commonly represented as a tensor in the form of an *adjacency matrix*, see Section 2.1. However, as a tensor is ordered by definition, some kind of order needs to be established in an adjacency matrix representation of a graph. As the established order is arbitrary, any representation of a graph by its adjacency matrix is non-unique. To solve classification problems on graphs, a possible solution is to include every permutation of all adjacency matrices in the training set and treat the problem as an image classification task. However, a single graph with as few as 100 nodes could have as many as $100! \approx 10^{157}$ unique adjacency matrix representations, why any such strategies are infeasible. Therefore, an ANN needs to be node order invariant by design. *Graph Neural Networks* (GNNs) is a class of ANNs that addresses such problems. Early attempts by Scarselli et al.[21][22] are based on recurrent neural networks. Since then, numerous variants have been suggested.[2][3][4]

In this thesis we focus on *Graph Convolutional Neural Networks* (GCNs), a class of GNNs with roots in spectral graph theory.[23][24] Motivated by *Convolutional Neural Netowrks'* (CNNs) ability to construct highly expressive representations by extracting local features from structured inputs, GCNs attempt to generalize CNNs to the graph domain. Section 2.3 describes how this can be done by defining spectral filters in the Fourier domain of a graph.

All GCN architectures implemented in this thesis are inspired by the work of Kipf et al.[18] and Zhang et al.[25]. Kipf et al.[18] proposes a scalable approach for semi-supervised learning on graph-structured data for node classification, whereas Zhang et al.[25] proposes a novel GCN architecture accepting graphs of arbitrary size and structure for graph classification.

1.2 Aim of the project

The main focus of this thesis is to investigate the potential of GCNs as a tool in *brain connectivity analysis*. The aim of this thesis is to form a foundation on which future work concerning GNN analysis of brain graphs can be built. Moreover, principal parts of this thesis generalize beyond brain connectivity analysis to other fields where graph analysis is used.

We analyze and evaluate the performance of GCNs on the task of classifying brain graphs constructed from AD patient data derived from MRI, as well as on simulated brain graphs. As described in Section 3.1, we develop a method for simulating graphs exhibiting properties similar to those observed in human brain networks.

2

Theory

2.1 General graph theory

A graph $G = (V, E)$ is defined as the sets of nodes (or vertices) V and edges E. The number of nodes in a graph will be denoted by $N = |V|$, and the number of edges by $M = |E|$. If edges in G are *directed*, $(i, j) \not\equiv (j, i)$, the graph is called directed. Else if edges in G are *undirected*, $(i, j) \equiv (j, i)$, the graph is called undirected. For simplicity, a shorthand notation e_{ij} of edge (i, j) will sometimes be used. Furthermore, a graph G can be either *binary* or *weighted*. In the case of weighted graphs, each edge is defined by a triplet (i, j, w_{ij}), where w_{ij} is the weight associated with edge (i, j).

If there exists a continuous sequence of edges connecting every node in a graph, the graph is considered *connected*. In a connected graph, any given node is able to reach all other nodes in the graph. If all pairs of nodes in a graph are connected to one another, the graph is said to be *complete*.

We also define the neighborhood of a node $i \in V$ as the set of nodes $N_i = \{j \ : \ e_{ij} \in E\}$. In other words, the neighborhood of a node i is the set of all nodes adjacent to i. The size of N_i is referred to as the *degree* of node i.

Edges describe how information is allowed to propagate in the graph, and can be used to describe any kind of relation between nodes. For example, weights can denote the *strength* or *capacity* of connections between nodes, or they can relate the *cost* of traversing from one node to another. Similarly, if nodes exist in a topological space, weights can be used to denote the *distance* between pairs of nodes. In terms of expressive power, this versatility makes graphs a powerful data structure. Unless stated otherwise, weights are to be considered a measurement of connection strength for all graphs in this thesis.

A binary graph is conveniently represented by its *adjacency matrix*, $A \in \mathbb{R}^{N \times N}$, defined as

$$A_{ij} = \begin{cases} 1 & \text{if } e_{ij} \in E \\ 0 & \text{if } e_{ij} \notin E \ . \end{cases} \tag{2.1}$$

Note that for undirected graphs, A is always symmetrical. If the graph is weighted,

elements in A are simply replaced by their corresponding weights, i.e. $A_{ij} = w_{ij}$.

2.2 Graph measures

Graph measures are used to quantify various characteristics or features of a graph. Different measures can refer to either *global* or *local* properties. Global measures refer to properties of the entire graph, while local measures refer to properties on a nodal level. Since local measures can be used to form distributions over all nodes, they as well can be used to describe global properties of a graph.

In this section we present a selection of graph measures commonly used to describe properties of the human brain.[26][27][28] In section 3.1, we present a method for simulating brain graphs exhibiting these characteristics to various degrees.

2.2.1 Centrality

The concept of centrality is used to describe the "importance" of individual nodes in a graph. Thus, indicators of centrality are always local measures. Even though all centrality measures attempt to answer the same question, they approach it in different ways. In this section we briefly mention two different ways of indicating centrality.

The first one has already been mentioned, namely *degree centrality*. If a node has many connections, this should indicate that the node plays a central role in the network. In the case of a weighted graph, the degree of a node is instead referred to as *strength*, which is calculated as the sum of the weights of all edges incident to and from the node.

Another indicator of centrality is the *betweenness centrality* of a node. In order to understand betweenness centrality, the concept of a *shortest path* must first be defined. The shortest path $P_s(i, j)$ is the shortest possible sequence of nodes with edges connecting nodes i and j. Thus, $P_s(i, j)$ is not necessarily unique, since multiple paths of equal length might connect nodes i and j. If $P_s(i, j)$ is known for all pairs of nodes in a graph, the betweeness centrality of a node i is defined as the fraction of all shortest paths traversing node i. By this definition, a node with high betweeness centrality serves as a *hub* in the graph, and is vital for an effective flow of information.

These kinds of hubs are assumed to play an important role in human brain networks.[26]

2.2.2 Clustering and transitivity

Other important features of human brain networks are high degrees of *clustering* and *transitivity*.[26] This means that nodes in a brain network have a high tendency to form highly connected clusters. This property can be described either locally by calculating the *local clustering coefficients* for individual nodes in the network, or

globally by calculating the *transitivity* of the network. For an undirected graph G, the local clustering coefficient of a node i is defined as

$$C(i) = \frac{2 \mid \{(j,k) \ : \ j,k \in N_i, (j,k) \in E\} \mid}{k_i \left(k_i - 1\right)} \ , \tag{2.2}$$

where k_i is the degree of node i. If G is directed, the factor 2 is simply removed in Eq. 2.2. In other words, $C(i)$ amounts to the degree to which *triplets* of nodes form closed loops. A triplet of nodes consists of three nodes that are connected by either two (*open* triplet) or three edges (*closed* triplet).

Clustering can also be calculated on a global level as the transitivity of the graph, sometimes also referred to as the *global clustering coefficient*. The transitivity, T, takes the total number of open and closed triplets in the graph into account:

$$T = \frac{\text{Number of closed triplets}}{\text{Total number of triplets}} \ , \tag{2.3}$$

which holds for both directed and undirected graphs. Global clustering is sometimes defined as an average of the local clustering coefficients in a graph,

$$C_{avg} = \frac{1}{N} \sum_{i=1}^{N} C(i) \ . \tag{2.4}$$

Note that this definition is not equivalent to Eq. 2.3. The average clustering coefficient puts more weight on low degree nodes, while transitivity puts more weight on high degree nodes.

One way of thinking about transitivity is in terms of network permeability. A network with high transitivity is easily maneuvered, since the shortest path between any two nodes is rarely unique. The removal of any single node would, therefore, not have a drastic impact on the flow information.

2.2.3 Modularity

Modularity measures the degree to which the nodes in a graph are divisible into distinct modules. Modules, or *communities*, are defined as the set $S_i \subset V$, where the nodes in S_i are interconnected to a greater extent than with the rest of the graph. Calculations of modularity require a pre-determined division of the nodes into communities, typically found through some type of clustering algorithm. Therefore, the *modularity* of a graph is not uniquely determined. Given a partitioning $\{S_i\}$ of the nodes in a graph G, the modularity of G is defined as

$$\frac{1}{M} \sum_{i=1}^{N} \sum_{j=1}^{N} \left[A_{ij} - \frac{k_i k_j}{M} \right] \delta_{ij}, \tag{2.5}$$

where δ_{ij} equals 1 if i and j belongs to the same community, and 0 otherwise.

This kind of partitioning of the human brain into highly interconnected communities has not only been observed on the scale of individual *neurons*, but also on a larger scale of specialized brain regions considered with brain imaging techniques like MRI.[26]

2.2.4 Small-worldness

The concept of small-worldness is somewhat more diffuse than what has been discussed so far. In a network exhibiting small-world properties, most nodes are not connected to each other. However, every node can be reached from any other node with a small number of intermediate steps. In a small-world network, any two neighbours of a node have a high probability of being neighbours themselves. In terms of graph measures, these properties are characterized by high levels of clustering and short average path lengths.

There are many different ways of quantifying these characteristics in a network. Most of them are based on comparing the network to random graphs with the same size and density. In this thesis *small-worldness* of a graph G is defined as

$$\sigma_{sw} = \frac{C/C_r}{L/L_r}, \tag{2.6}$$

where C and C_r are the average clustering coefficients of G and a random graph respectively, while L and L_r denote the *characteristic path lengths* of the same graphs. The characteristic path length of a graph is the average length of all shortest paths.

The property of small-worldness is one of the main characteristics observed in human brain networks, and is assumed to be crucial for healthy brain function.[29]

2.3 Convolutions in the graph domain

Classical convolutions as performed in a CNN only make sense given a topological space, e.g. 2D or 3D images, where the neighborhood of a pixel is naturally defined. However, for graph-structured data, different nodes might have different degrees, why it is no longer trivial to define convolutional filters that are generally applicable to all local neighborhoods in the graph. In this section we describe how spectral graph theory enables a generalization of the classical notion of a convolution, by defining convolutional filters in the Fourier domain of a graph.[23][24]

Consider a general undirected graph $G = (V, E)$, where nodes, $V \in \mathbb{R}^N$, and edges, $E \in \mathbb{R}^{M \times 2}$, are encoded in the adjacency matrix $A \in \mathbb{R}^{\mathbb{N} \times \mathbb{N}}$. Since G is undirected, A is symmetric. Also, a matrix D with elements

$$D_{ij} = \begin{cases} 0 & \text{if } i \neq j \\ \sum_j A_{ij} & \text{if } i = j, \end{cases} \tag{2.7}$$

is defined as the degree matrix of G. For a function $f : \mathbb{R} \to \mathbb{R}$, the Fourier transform of f is defined as the inner product of f and $e^{2\pi i \xi x}$ as

$$\hat{f}(\xi) = \langle f(x), e^{2\pi i \xi x} \rangle = \int_{-\infty}^{\infty} f(x) e^{-2\pi i \xi x} dx, \tag{2.8}$$

where $e^{2\pi i \xi x}$ are the eigenfunctions of the one dimensional Laplace operator. In other words, the Fourier transform of f is the expansion of f in terms of the eigenfunctions

of the Laplace operator. Completely analogously, the *graph Fourier transform* of a graph signal $X \in \mathbb{R}^N$ is defined as the (spectral) coefficients of the linear expansion of X in terms of the eigenvectors of the graph Laplacian. The graph Laplacian L is defined as

$$L = D - A \ . \tag{2.9}$$

Furthermore, a normalized graph Laplacian can be defined as

$$L = D^{-\frac{1}{2}} L D^{-\frac{1}{2}} = I_N - D^{-\frac{1}{2}} A D^{-\frac{1}{2}}, \tag{2.10}$$

where I_N is the $N \times N$ identity matrix. From this stage on, L is assumed to be normalized according to Eq. 2.10. The definition of the graph Laplacian in Eq. 2.9 can be intuitively understood. First, recall that in Euclidian space, the Laplacian of a function f is defined as the divergence of the gradient of f, written as $\Delta f = \nabla^2 f$. In the one dimensional case this is simply the second derivative of f. More generally, the gradient ∇f is a vector-valued function giving the derivative of f along *each* direction. Thus, Δf is a scalar-valued function corresponding to the sum of the second derivatives of f along each direction. In a graph, the analogue for directions is edges. The gradient of f defined on a graph is then an array of differences in function value along each edge. More formally, consider a function $f(i)$ defined on the nodes of an undirected graph G with edges $e_{ij} \in E$. For simplicity G is assumed to be unweighted (binary). By fixating an arbitrary direction for the edges e_{ij}, the gradient of f can be defined as

$$\nabla f(e_{ij}) = f(i) - f(j) \ \in \ \mathbb{R}^M. \tag{2.11}$$

Let $B \in \mathbb{R}^{N \times M}$, called the *incidence matrix*, be defined as

$$B_{ie} = \begin{cases} +1 & \text{if edge } e \text{ exits node } i \\ -1 & \text{if edge } e \text{ enters node } i \\ 0 & \text{otherwise.} \end{cases} \tag{2.12}$$

Eq. 2.11 can then intuitively be expressed as

$$\nabla f = B^{\mathrm{T}} f. \tag{2.13}$$

If the gradient is thought of as describing flow, the divergence of ∇f can be described in an equally intuitive way. The divergence of ∇f at a node i is the net flow leaving i. The net flow for all nodes can be calculated according to

$$\Delta f = B \nabla f = B B^{\mathrm{T}} f, \tag{2.14}$$

where $\Delta f \in \mathbb{R}^N$. Hence, a second definition of the discrete graph Laplacian has been derived as $L = BB^{\mathrm{T}}$. To show that this equals the definition in Eq. 2.9, we compare the to definitions as follows. Given $L = BB^{\mathrm{T}}$, element L_{ij} is the scalar product of the i^{th} row of B with the j^{th} column B^{T}. Clearly, the j^{th} column of B^{T} is equal to the j^{th} row of B. Looking back on the definition of B in Eq. 2.12, we see that L_{ij} will be equal to -1 if and only if there is a connection from node i to

node j, and equal 0 otherwise. For $i = j$, on the other hand, L_{ij} simply becomes the sum of all connections entering or exiting node i. Hence, $BB^{\mathrm{T}} = D - A$.

We can immediately conclude a few things about the graph Laplacian L. First of all, notice that L is invariant under the choice of edge orientation when computing the incidence matrix B. Secondly, since $L = BB^{\mathrm{T}}$, L is positive semi-definite.[30] This is important because it implies that L only has non-negative eigenvalues. Thirdly, L is singular (has at least one eigenvalue equal to zero). Since all rows in L sum to zero, any constant vector $\vec{v} = [c, c, c, ..., c]^{\mathrm{T}} \in \mathbb{R}^N$ must be an eigenvector of L with eigenvalue $\lambda = 0$. This is best understood by visualizing the graph as a network describing flow. If all nodes have the same potential, the net flow would be zero on all edges of the graph. Finally, from both definitions separately, L must be symmetric. This is one of the most important features of the graph Laplacian for the purposes of this thesis, since it allows for a definition of the graph Fourier transform in a natural way.

Since L is a symmetric matrix with only real entries, the *spectral theorem* tells us that it can be diagonalized by an orthogonal matrix U. Hence, a spectral decomposition of L can be written as

$$L = U\Lambda U^{\mathrm{T}}, \tag{2.15}$$

where the columns of U are the eigenvectors φ_i of L, and Λ is a diagonal matrix with Λ_{ii} equal to the i^{th} eigenvalue λ_i of L. Now, for any signal $X \in \mathbb{R}^N$ defined on the nodes of G, the graph Fourier transform of X is naturally defined as

$$\hat{X} = \langle X, \varphi_i \rangle = \sum_{i=1}^{N} X\varphi_i^* = U^{\mathrm{T}}X, \tag{2.16}$$

where φ_i^* denotes the conjugate transpose of φ_i. The inverse transform is defined accordingly as

$$X = U\hat{X} \ . \tag{2.17}$$

Since convolutions in the time domain reduce to multiplication in the spectral domain, the convolution of a signal X and a filter $g_\theta = diag(\theta)$, parameterized in Fourier space, can be defined as

$$g_\theta * X = U g_\theta U^{\mathrm{T}} X. \tag{2.18}$$

The evaluation of Eq. 2.18 might, however, become prohibitively expensive for large scale graphs, as the decomposition of L has time complexity $\mathcal{O}(N^3)$ for any non-approximative algorithm.[31] A natural way to interpret g_θ is as a function of Λ, i.e. the eigenvalues of L. In Hammond et al.[23], a truncated K^{th} order approximation was suggested as an expansion of $g_\theta(\Lambda)$ in terms of Chebyshev polynomials $T_k(x)$, as

$$g_{\theta'}(\Lambda) \approx \sum_{k=0}^{K} \theta'_k T_k(\tilde{\Lambda}), \tag{2.19}$$

where Λ has been rescaled to $\tilde{\Lambda} = \frac{2}{\lambda_{\max}}\Lambda - I_N$. Here, $\lambda_{\max}$ denotes the largest eigenvalue of L. The Chebyshev polynomials $T_k(x)$ are defined recursively as

$$T_k(x) = 2xT_{k-1}(x) - T_{k-2}(x), \tag{2.20}$$

with $T_0 = 1$ and $T_1 = x$. Insertion of Eq. 2.19 into Eq. 2.18 gives

$$g_{\theta'} * X \approx U \left(\sum_{k=1}^{K} \theta'_k T_k(\tilde{\Lambda}) \right) U^{\mathrm{T}} X. \tag{2.21}$$

Finally, by the fact that $L^k = (U\Lambda U^{\mathrm{T}})^k = U\Lambda^k U^{\mathrm{T}}$,

$$g_{\theta'} * X \approx \sum_{k=1}^{K} \theta'_k T_k(\tilde{L}) X, \tag{2.22}$$

where $\tilde{L} = \frac{2}{\lambda_{\max}} L - I_N$. Since Eq. 2.22 is a K^{th} order polynomial in $\tilde{L}$, it is a K-localized expression for convolutions of $g_{\theta'}$ and signal X.[23] The time complexity of Eq. 2.22 is $\mathcal{O}(M)$, i.e. it is linear with respect to the number of edges in G.[18]

2.3.1 Layer-wise propagation rule

Following the work outlined by Kipf et al.[18], we define a linear layer-wise propagation rule for a GCN. The linearity comes from the fact that the layer-wise convolution is limited to $K = 1$, i.e. Eq. 2.22 becomes a linear function with respect to L. This linearity allows for deeper models, which can improve the modeling capacity in a wide variety of domains. With $K = 1$, Eq. 2.22 is reduced to

$$g_{\theta'} * X \approx \theta'_0 X + \theta'_1 \left(\frac{2}{\lambda_{\max}} L - I_N \right) X \ . \tag{2.23}$$

To simplify things further, the largest eigenvalue of L is fixed at $\lambda_{\max} = 2$. This only changes the scale, and it is thus reasonable to assume that the network parameters will be able to adapt to this change during training. Hence,

$$g_{\theta'} * X \approx \theta'_0 X + \theta'_1 \left(L - I_N \right) X = \theta'_0 X - \theta'_1 D^{-\frac{1}{2}} A D^{-\frac{1}{2}} X \ . \tag{2.24}$$

Further reduction of the number of parameters is done by the introduction of a single filter parameter $\theta = \theta'_0 = -\theta'_1$. Resulting in the approximation

$$g_{\theta'} * X \approx \theta \left(I_N + D^{-\frac{1}{2}} A D^{-\frac{1}{2}} \right) X \ . \tag{2.25}$$

The factor $I_N + D^{-\frac{1}{2}} A D^{-\frac{1}{2}}$ has eigenvalues in range the $[0, 2]$. Building a multi-layer convolutional neural network based on this propagation rule can lead to problems with exploding or vanishing gradients during backpropagation. To remedy this, a renormalization trick is introduced,

$$I_N + D^{-\frac{1}{2}} A D^{-\frac{1}{2}} \ \rightarrow \ \tilde{D}^{-\frac{1}{2}} \tilde{A} \tilde{D}^{-\frac{1}{2}} \ , \tag{2.26}$$

where $\tilde{A} = A + I_N$ and $\tilde{D}_{ii} = \sum_j \tilde{A}_{ij}$. By generalizing Eq. 2.25 to a signal $X \in \mathbb{R}^{N \times F}$ and a layer with C convolutional filters, the following layer-wise propagation rule is defined:

$$H = \sigma \left(\tilde{D}^{-\frac{1}{2}} \tilde{A} \tilde{D}^{-\frac{1}{2}} X W \right), \tag{2.27}$$

where $W \in \mathbb{R}^{F \times C}$ is a matrix of filter parameters, $H \in \mathbb{R}^{N \times C}$ is the convolved signal matrix, and σ is any element-wise applied activation function.

As described in Kipf et al.[17], a featureless propagation rule can be established by replacing X with the identity matrix I. Eq. 2.27 thus reduces to

$$H' = \sigma\left(\tilde{D}^{-\frac{1}{2}}\tilde{A}\tilde{D}^{-\frac{1}{2}}W\right). \tag{2.28}$$

In contrast to the standard GCN, where filters in W are applied to node specific feature vectors, i.e. rows in X, it becomes apparent that filters are now instead applied directly to the normalized adjacency matrix. This is a fundamental difference between a featureless GCN and a standard GCN. For isomorphic graphs, the standard GCN will always produce the same set of filtered feature vectors, regardless of adjacency matrix representation. This is not the case for a featureless GCN however, as no feature vectors are present. Instead, "feature vectors" correspond to rows in the adjacency matrix. As isomorphic graphs can have different adjacency matrix representations, filters will no longer produce the same set of "filtered feature vectors".

3

Method

In this chapter, integral parts of the data simulation and network architectures are broken down to the essentials and reviewed. Both simulated and experimental data were used to investigate the potential and limitation of GCNs, specifically the applicability of GCNs for classification of undirected brain graphs by means of supervised learning. Throughout all GCN implementations, no explicit graph features were given as input to the networks.

The first experiment involved simulated binary graphs. The main purpose of this experiment was to investigate the performance of GCNs on the task of learning from *unordered* graph data.

The second experiment involved weighted graphs generated from AD patient data derived from MRI. Data from a single patient contained recordings of *cortical thickness* values of 68 labeled brain regions. The purpose of this experiment was to investigate structural differences in brain graphs derived from either a healthy control group or a group of AD patients.

All network implementations were done in Python using Tensorflow[32].

3.1 Simulated data

Two different approaches were tested on the simulated data. A featureless approach, see Eq. 2.28, as well as a semi-featureless approach where a matrix of ones was used as input, see 2.27. The latter will be referred to as *dummy features*. This way, filters can still be applied to a "feature matrix", without having to rely on graph-specific feature matrices.

Two-class classification was performed on simulated graphs consisting of 100 nodes. Each sample graph was simulated by imposing graph structures commonly observed in the human brain. By imposing different structure to the two classes, the ability of GCNs to separate them was explored.

3.1.1 Graph simulation procedure

All samples were initialized as identical small-world graphs using the *Watts-Strogatz small-world model*[33] with zero rewiring probability. Small-world structure is frequently observed in a wide range of human brain graphs, and is vital for healthy brain function.[34][35] An intuitive way to visualize a small-world graph is to imagine the nodes in a ring lattice, where each node is connected to its k nearest neighbours, see Fig. 3.1. Throughout all simulations, we choose $k = 4$. As all graphs considered were undirected, they initially contained 200 edges.

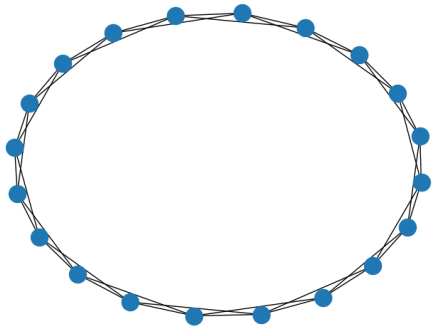


Figure 3.1: Initial Watts-Strogatz small-world graph. Each node is connected to its 4 nearest neighbors in the ring lattice.

The two classes were differentiated by class-specific parameters. These parameters defined a probability distribution over different ways of adding new edges. Edges could be added in one out of four ways: By preferential attachment, by completing triangles, by forming communities, or by adding random connections. Four hundred additional edges were added sequentially to each sample. For each new edge to be added, one of the four ways was selected independently at random from the probability distribution by means of *roulette wheel selection*. The probability distributions for each class were generated randomly. We now proceed to describe these four ways in detail.

Adding edges with preferential attachment

A random source node s was selected with a probability proportional to its degree. A random target node t was then chosen uniformly at random from the set of nodes not already connected to s. Since nodes with a high degree have a high probability of being selected, adding multiple edges this way will eventually result in the formation of *hubs*, see Fig. 3.2.

In terms of graph measures, adding edges with preferential attachment will affect various indicators of centrality, e.g. degree centrality and betweeness centrality, see Section 2.2.1.

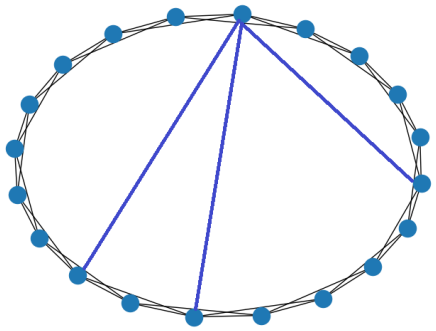


Figure 3.2: Increasing the centrality of certain nodes (blue edges).

Completing triangles

A random source node s was selected from the set of all nodes V. A target node t was then selected randomly from the set $T \subset V$, where T is the set of all nodes two edges away from s, such that $t \notin N_s$. In other words, T is the set of all nodes that together with s constitute end nodes of an open triplet. Notice that by adding edge (s, t) to E, more than one open triplet might be closed at the same time, see Fig. 3.3.

By completing more and more triangles this way, local clustering and transitivity in the graph were increased, see Section 2.2.2.

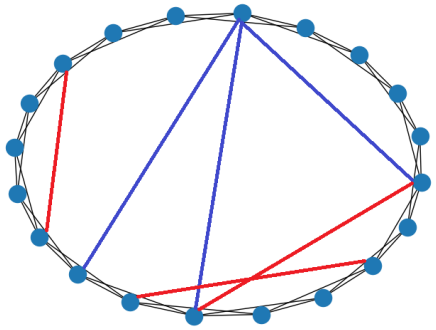


Figure 3.3: Increasing clustering and network transitivity (red edges).

Forming communities

Nodes were partitioned into five distinct subsets, S_i, of equal size. When forming communities, a subset S_i was chosen at random, from which both source node s and target node $t \in S_i \setminus N_s$ were randomly chosen, see Fig. 3.4.

By adding more and more edges this way, distinct communities will start to form, thus increasing the modularity, see Section 2.2.3.

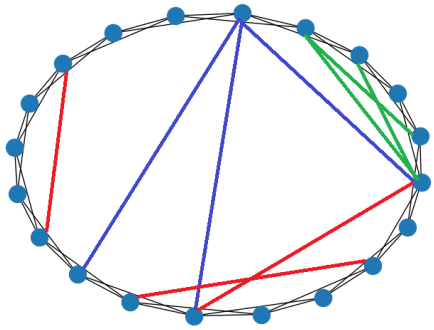


Figure 3.4: Increasing the modularity of the network (green edges).

Adding random connections

A source node s as well as a target node t was selected at random from the complete set of nodes V. The only restriction was that $t \notin N_s$, see Fig. 3.5.

By adding random connections, additional irregularity was introduced to the graphs. This also served as a way of adding noise to the data.

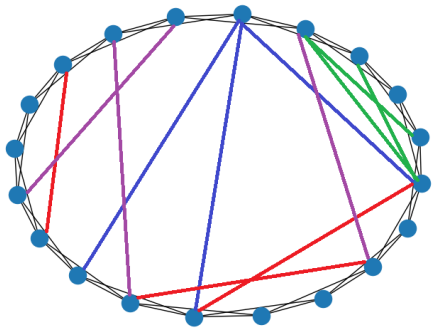


Figure 3.5: Increasing the stochasticity in the network (purple edges).

A pseudo code summary describing the process of generating a data set of simulated graphs is found in Alg. 1. A data set consisted of two classes with distinct class parameters. Ten thousand sample graphs were generated for each class separately. We chose to investigate graphs with 100 nodes and 600 edges as this resulted in densities observed in human brain graphs.[34][35][36]

3.1.1.1 Shuffling of node ordering

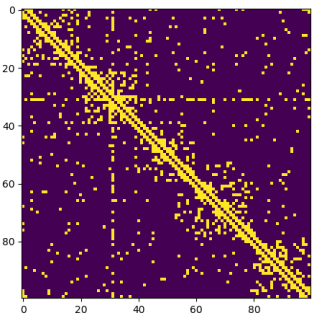
Because of the way samples were generated, graph structures are noticeable in the adjacency matrices. Even though each sample graph was generated independently, certain structures such as modularity and small-worldness followed a coherent indexation, see Fig. 3.6. As the main purpose of the experiments on simulated data was to investigate the performance of GCNs for the task of learning from unordered graph data, the node ordering of each sample graph was shuffled independently. By

Algorithm 1: Generation of a data set of simulated graphs.

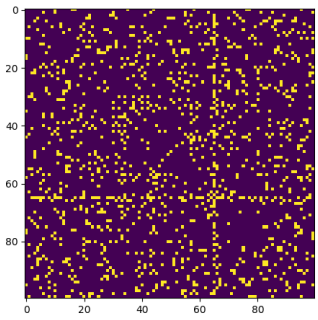
```
for each class do
    sample class parameters P randomly
    for each sample graph do
        generate Watts-Strogatz small-world network S
        for each new edge do
            select rule from P using roulette-wheel selection
            add a new edge to S according to selected rule
        end
    end
end
```

shuffling the node order, any notion of node labeling is effectively gone, leaving only the actual graph structures intact.

Figure 3.6: Example of adjacency matrix representations of a simulated graph before and after shuffling of node ordering.



(a) Before shuffling there are five detectable communities, one hub (node 30) and a small-world structure. The Watts-Strogatz small-world structure is indicated by the diagonal lines above and below the main diagonal.



(b) After shuffling all structure is obscured except for the hub. Note, however, that the hub has received a new indexation in the matrix.

3.2 Experimental data

Two-class classification was also performed on experimental data derived from MRI. The classification was performed using a featureless GCN, i.e. the feature matrix was replaced by the identity matrix, see Eq. 2.28.

The experimental data consisted of cortical thickness values in 68 cortical regions from the Desikan atlas[1]. Cortical thickness values were recorded for three groups of AD patients labeled according to disease progression, as well as for a healthy control group of 110 patients. The AD patients included 57 patients labeled *mild*,

238 patients labeled *medium* and 115 patients labeled *severe*. The three groups of AD patients were combined to form a single class. Thus, no distinction was made in terms of disease progression for the AD patients. Classification was therefore performed on the two classes of subjects, the control class, and the AD class.

Under the assumption that the cortical thickness is related to connectivity, sample graphs were generated by means of correlation between cortical regions. The correlation was calculated on sampled subsets of each of the two classes separately. These subsets will be referred to as sampled *cohorts*.

3.2.1 Sampling cohorts

The patients were split into three batches, where 40% of the control class and AD patient class was reserved for training, 30% was reserved for validation and 30% was reserved for testing. This was done before any cohorts were sampled, see Fig. 3.7. Note that the ratios between splits in Fig. 3.7 are only figurative.

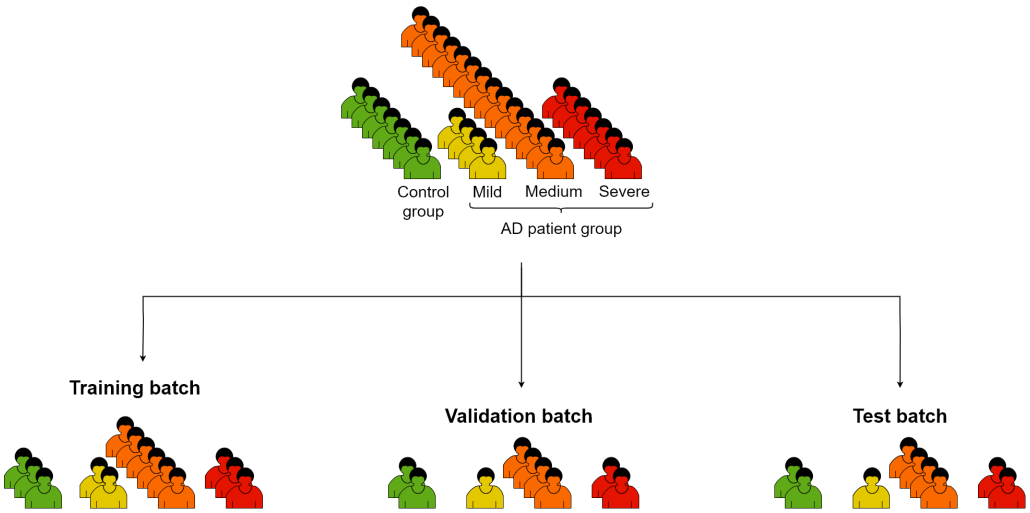


Figure 3.7: Splitting subjects into training, validation and test batches. Each subject contains recordings of the cortical thickness in 68 labeled cortical regions.

For each of the three batches, cohorts were sampled from the control group and AD patient group separately, see Fig. 3.8. Each cohort consisted of ten randomly sampled subjects. Cohorts were sampled with replacement, i.e. subjects were allowed to appear in more than one cohort.

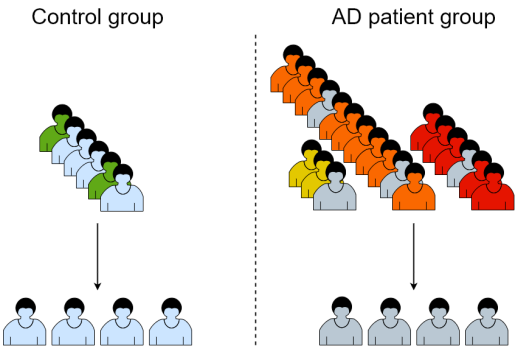


Figure 3.8: Example of two cohorts extracted, one from each patient group. No single subject was allowed to appear more than once in the same cohort.

3.2.2 Constructing adjacency matrices from cohorts

For each sampled cohort, the *Pearson correlation coefficient*,

$$\rho_{ij} = \frac{\mathrm{cov}(i,j)}{\sigma_i \, \sigma_j}, \tag{3.1}$$

was calculated between all pairs of cortical regions i and j. The correlation coefficients were used to indicate the strength of connections between cortical regions, represented in an adjacency matrix with elements

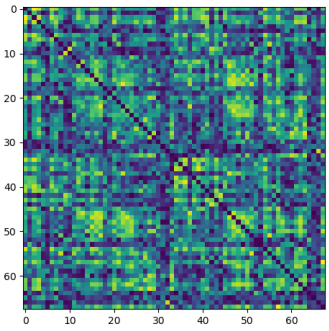
$$A_{ij} = \begin{cases} |\rho_{ij}| & \text{if } i \neq j \\ 0 & \text{if } i = j. \end{cases} \tag{3.2}$$

The elements in A were taken as the absolute value of ρ_{ij}, i.e. positive and a negative correlation between two brain regions were treated as equally strong indicators of connectivity.

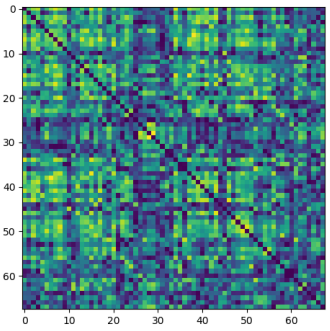
Two sample adjacency matrices are presented in Fig 3.9. These adjacency matrices no longer constitute representations of brain graphs on an individual level, but are instead estimations of the connectivity in the "collective brain" of an entire group of individuals.

Separate training, validation and test sets were constructed. The training set consisted of 25,000 adjacency matrices generated from each class. Both the validation and test set consisted of 5,000 adjacency matrices from each class respectively. The complete process is described in Alg. 2.

Figure 3.9: Example heat maps of adjacency matrices based on correlation.



(a) A random sample from the healthy control group.



(b) A random sample from the AD patient group.

Algorithm 2: Generation of a data set from cortical thickness data.

```
randomly split subjects into training, validation and test batches
for each batch do
    for each class do
        for each cohort do
            extract cohort of 10 random subjects from the class
            generate correlation matrix ρ from extracted cohort
            set diagonal elements in ρ to zero
            save sample as |ρ|
        end
    end
end
```

3.3 Network architectures

The GCNs considered in this thesis can be seen as composite networks of modules: A *graph convolutional module*, a *1D-convolutional module* and a *sorting module*, see Section. 3.3.1. By the inclusion and removal of modules, three different GCN architectures were implemented that we label: *GCN-base*, *GCN-conv*, *GCN-sort*. For baseline comparisons, we also performed experiments using a *Fully Connected Feedforward Neural Network* (FFNN). The output of all implemented networks was a 2×1-vector corresponding to one-hot encodings of class predictions.

3.3.1 Network modules

Three stand-alone modules were used as building blocks for all implemented GCNs. In this section, individual schematics are presented and reviewed for each module.

3.3.1.1 Graph convolutional module

The graph convolutional module consisted of four *hyperbolic tangent* graph convolutional layers (GC-layers), where the output of each GC-layer was concatenated into a single matrix of multi-scale nodal feature embeddings.[25][24] The propagation rule of a GC-layer is given in Eq. 2.27. For GC-layer i, the output H_i is passed as input to the subsequent GC-layer

$$H_{i+1} = \sigma\left(\tilde{D}^{-\frac{1}{2}}\tilde{A}\tilde{D}^{-\frac{1}{2}}H_iW\right). \tag{3.3}$$

In each GC-layer, spectral filters are applied to local neighborhoods of nodes in the graph. The output H_1 of the first GC-layer thus contains information on the 1-hop neighborhoods of all nodes. As the output H_i of each GC-layer is passed as input to the next, H_i will effectively contain information on the i-hop neighborhoods of all nodes. Thus, by stacking multiple GC-layers on top of one another, we effectively convolve local neighborhoods on increasing sub-scales.[25]

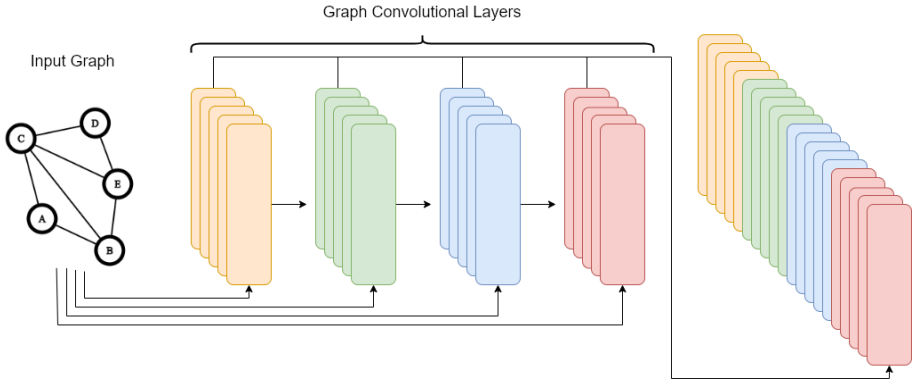


Figure 3.10: *Graph convolutional module*: The module consists of four graph convolutional layers, where the output H_i of each layer is passed as input to the next. All outputs H_i are concatenated into a single matrix of multi-scale nodal feature embeddings, which is the output of the module.

3.3.1.2 1D-convolutional module

The 1D-convolutional module consisted of a single 1D-convolutional layer followed by *Max pooling*. The 1D-convolutional layer consisted of 16 filters with *ReLU* activations. Convolutions were performed node-wise, i.e. the stride and window size were set such that each filter outputted a single scalar value for each node's respective feature embedding. This was followed by *Max pooling* with a window size of 2×1, thus reducing the size by a factor two, see Fig. 3.11.

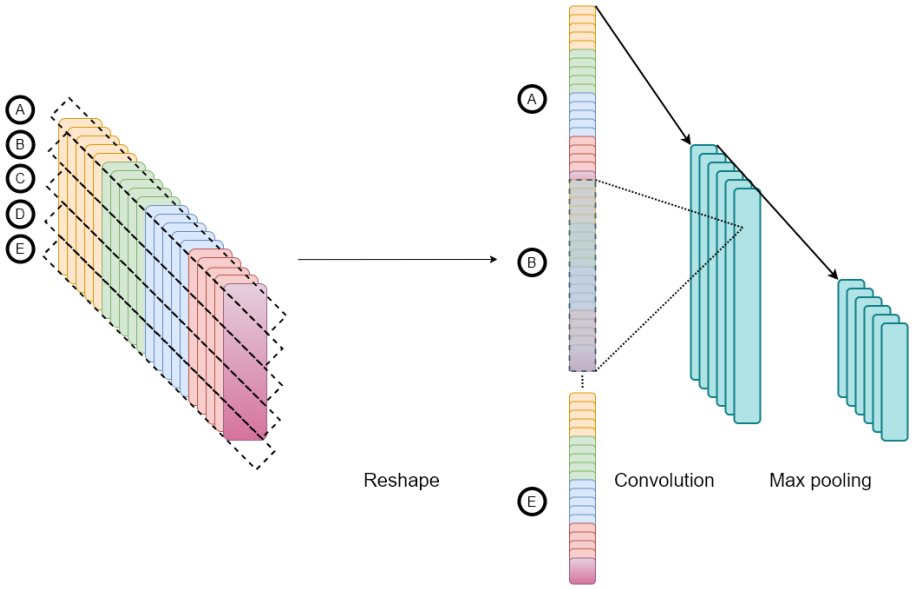


Figure 3.11: *1D-convolutional module*: The matrix of multi-scale feature embeddings is reshaped to a one dimensional array by stacking the nodal feature arrays on top of one another. 1D-convolutional filters are then applied to nodal feature embeddings individually, i.e. filters are not applied *between* nodes. This is followed by a 2×1 *Max pooling*.

3.3.1.3 Sorting module

In the sorting module, the node ordering, i.e the row indexation in the matrix of nodal feature embeddings, was rearranged with respect to a single feature, see Fig. 3.12. In Zhang et al.[25], sorting was introduced to produce a consistent node ordering in the matrix of nodal feature embeddings. This increases subsequent layers ability to generalize from a fixed node ordering. After sorting, only the row indexation in the feature matrix has changed.

3.12.

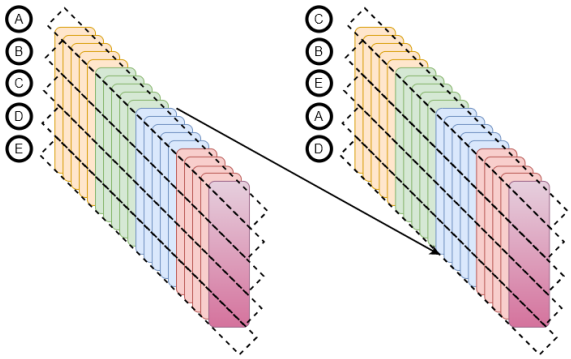


Figure 3.12: *Sorting module*: The matrix of multi-scale feature embeddings is sorted with respect to a single feature (highlighted in purple). The dashed rectangles represent the feature embedding for an individual node.

3.3.2 FFNN

The FFNN was composed of a single fully connected *softmax* layer with two output neurons, see Fig. 3.13. As no increase in generalization for deeper models was observed, no hidden layers were used. For details on network parameters in the implemented FFNN, see Tab. B.4 & B.1 in Appendix B.

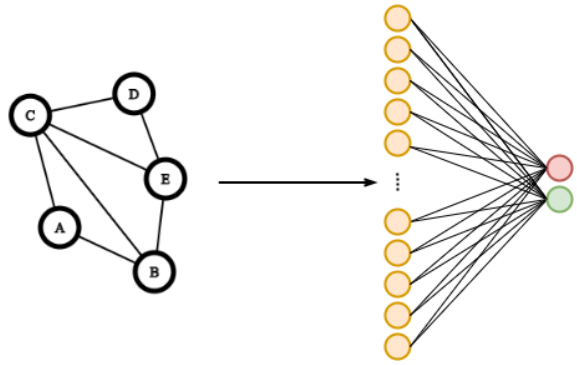


Figure 3.13: Architecture of FFNN: The adjacency matrix is passed directly to a fully connected output layer with two *softmax* neurons.

3.3.3 GCN-base

The GCN-base was composed of a single graph convolutional module, see Fig. 3.10. The concatenated feature matrix was passed directly to a fully connected output layer with two *softmax* neurons, see Fig. 3.14. For details on network parameters, see Tab. B.5 & B.2 in Appendix B.

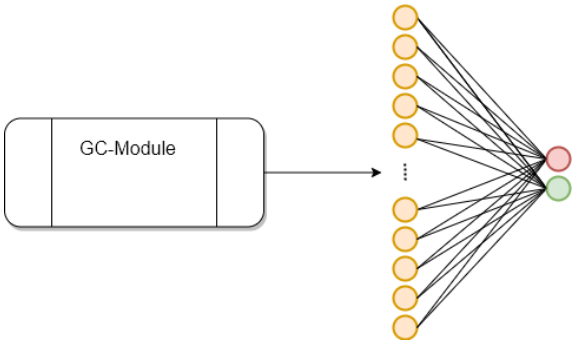


Figure 3.14: Schematic architecture of the GCN-base network.

3.3.4 GCN-conv

GCN-conv has the same network architecture as GCN-base, with the addition of a 1D-convolutional module, see Fig. 3.11. The output from the 1D-convolutional module was passed to a fully connected output layer with two *softmax* neurons, see Fig. 3.15. For details on network parameters, see Tab. B.7 & B.3 in Appendix B.

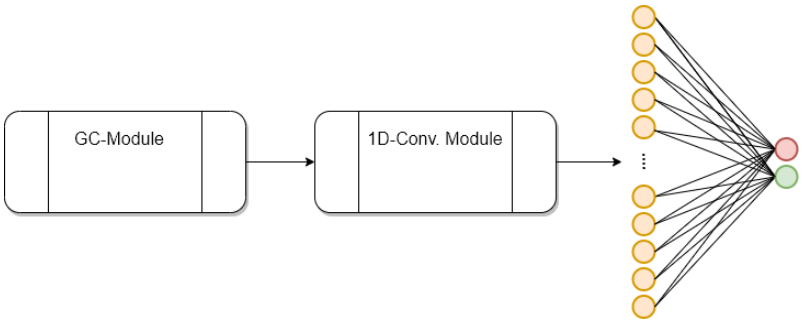


Figure 3.15: Schematic architecture of the GCN-conv network.

3.3.5 GCN-sort

GCN-sort has the same network architecture as GCN-conv, with the addition of a sorting module, see Fig. 3.12. The sorted feature matrix was passed to a 1D-convolutional module, followed by a fully connected output layer with two *softmax* neurons, see Fig. 3.16. For details on network parameters, see Tab. B.7 in Appendix B.

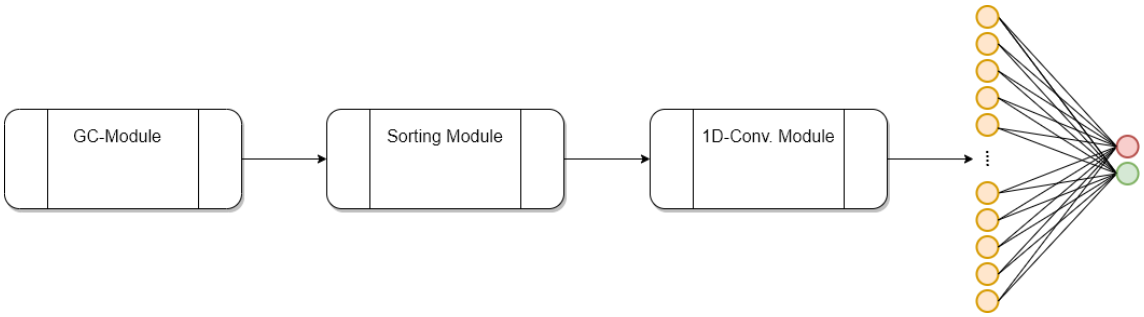


Figure 3.16: Schematic architecture of the GCN-sort network.

4

Results

For the task of classifying the simulated brain graphs, see Section 3.1, five different ANNs were tested and compared. Tests performed using the GCN-base network and the GCN-conv network only included the featureless approach, see Eq. 2.28. For the GCN-sort network, tests performed also included a semi-featureless approach, i.e. *dummy features* were used as input, see Eq. 2.27. This network is denoted *GCN-dummy*. For a baseline comparison, tests were also performed using the FFNN network. Schematics of individual network architectures are found in Section 3.3.

For the task of classifying graphs derived from the experimental data, see Section 3.2, three ANNs were tested and compared. Tests were performed using the GCN-base network, GCN-conv network and the FFNN network. For the implemented GCNs, tests were only performed using the featureless approach on the experimental data. Also, no sorting on nodal features, see Fig. 3.12, was performed as a fixed node order was imposed from the cortical regions.

Throughout, performance is measured in terms of predictive accuracy on the test sets. The loss function *Binary cross-entropy* was used for all networks.

4.1 Simulated Data

Ten data sets consisting of two classes each were generated according to Section 3.1. The parameter configurations defining the two classes in each data set were generated randomly, as described in Section 3.1. For each data set, two sets of class specific parameters were generated randomly. Each data set consisted of 20,000 sample graphs in total, with 10 000 samples of each class. For each class, 70% was used for training, 20% for validation and 10% for testing. All class parameters, along with class distributions for all graph measures discussed in Section 2.2, are found in Appendix A.

Tab. 4.1 presents network performance in terms of predictive accuracy on binary classification for the ten data sets of simulated graphs. The simulated graphs reflect different parameter configurations for each data set. Hence, the results in Tab. 4.1 should not be compared between different data sets, only between different network architectures. As such, any comparison between data sets has to consider

the underlying differences in graph structure. Therefore, no mean values over data sets are presented.

Table 4.1: Network performance on 10 randomly generated data sets. The networks were trained for 500 epochs on each data set. Early stopping was used with a patience of 50 epochs (enforced by validation loss). The numbers show accuracy and standard deviation for each network on the test sets, averaged over 10 runs with random splits, weight initializations and node orderings.

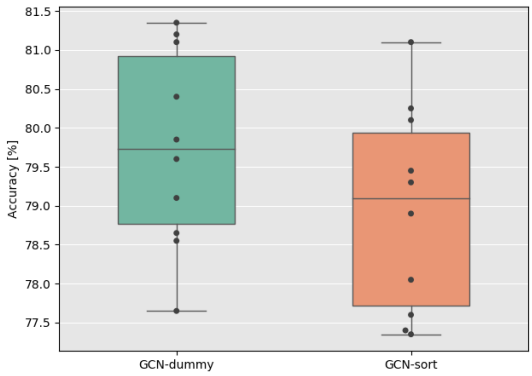
	GCN-dummy	GCN-sort	GCN-conv	GCN-base	FFNN
D1	79.65 (±0.96)	90.26 (±1.31)	**91.37** (±1.02)	84.77 (±1.37)	49.74 (±1.03)
D2	78.10 (±1.85)	98.88 (±0.24)	**99.08** (±0.31)	97.97 (±0.21)	50.04 (±0.71)
D3	**79.75** (±1.27)	78.95 (±1.32)	53.96 (±2.03)	51.67 (±1.16)	50.47 (±1.03)
D4	67.41 (±1.83)	97.50 (±0.78)	**98.11** (±0.40)	97.96 (±0.44)	49.76 (±0.85)
D5	68.71 (±2.46)	92.67 (±2.01)	**94.81** (±0.76)	92.02 (±0.67)	50.27 (±0.97)
D6	64.27 (±0.85)	81.55 (±3.36)	**83.05** (±1.31)	79.76 (±1.09)	50.03 (±1.06)
D7	73.87 (±4.36)	**74.65** (±1.04)	50.73 (±0.97)	50.39 (±0.61)	49.80 (±1.25)
D8	91.13 (±0.61)	**91.40** (±1.37)	87.79 (±0.96)	60.07 (±8.54)	49.63 (±1.40)
D9	73.84 (±0.98)	88.88 (±3.17)	**91.65** (±0.87)	87.66 (±0.59)	49.35 (±0.78)
D10	71.24 (±1.06)	86.47 (±0.92)	**89.41** (±0.66)	78.73 (±0.74)	50.47 (±0.96)

All graph neural networks outperformed the FFNN network. In fact, the FFNN was never observed to perform better than random predictions on any of the simulated data sets. From Tab. 4.1, it is also clear that the addition of a convolutional module to the general GCN architecture is beneficial.

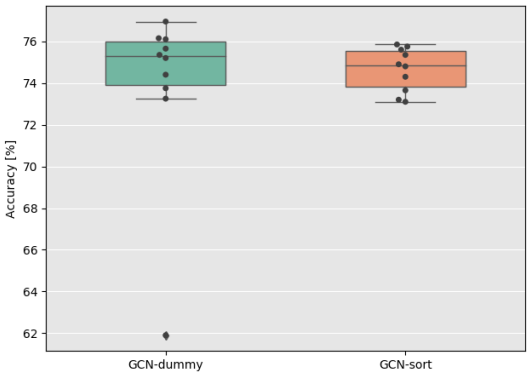
As discussed at the end of Section 2.3.1, the node order independence of the GCNs is compromised in the featureless approach. However, the featureless GCNs are still able to construct nodal feature embeddings separable in Euclidian space.

Node sorting turned out to be imperative to distinguish the classes in Data sets D3 and D7. This is presumably explained by a significant overlap between classes in all graph measure distributions investigated, see Fig. A.3 & A.7 in Appendix A. However, differences in extreme values for the centrality measures between classes were observed. This could indicate that differences in local node measures between classes are harder to detect with an arbitrary node ordering, whereas global attributes are less dependent on node ordering. Even though GCN-sort outperformed GCN-dummy on most of the simulated data sets, it is not entirely clear from Tab. 4.1 whether dummy features are worth pursuing or not. In Fig. 4.1, box plots of test scores for individual runs on data set D3 and D7 are presented for the two networks that implemented sorting. We see that on D3 and D7, the node order independence of GCN-dummy generally resulted in higher test scores.

Figure 4.1: Box plot of test scores for individual runs on data set D3 and D7 for GCN-dummy and GCN-sort.



(a) D3: On average, slightly higher test scores are observed for GCN-dummy.



(b) D7: Without the one outlier, GCN-dummy, on average, achieves higher test scores.

4.2 Experimental data

Network performance was evaluated on ten data sets generated according to Section 3.2. Each data set was generated from different random splits of the subjects into training, validation and test batches, where 40% of the subjects was reserved for training, 30% for validation and 30% for testing. From the training batch, 25,000 sample graphs were generated for each patient class. From the validation and test batch respectively, 5,000 sample graphs were generated for each class.

Network performance on the AD patient data, evaluated over ten random 40/30/30 training/validation/test splits, is presented in Tab. 4.2. The box plot in Fig. 4.2 displays a concise summary of the same results.

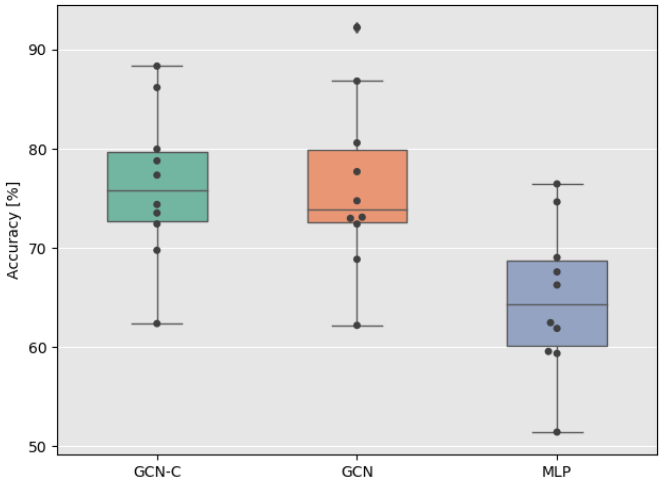


Figure 4.2: Box plot of average test scores for all network architectures on the AD patient data.

Table 4.2: Network performance presented on 10 random splits of all patients with Alzheimer's disease (*severe+medium+mild*) vs Control. Networks were trained for a maximum of 500 epochs, implemented with early stopping with a patience of 10 epochs (enforced by validation loss). Fifty percent dropout was used between consecutive layers in the training. The numbers show accuracy and standard deviation for each network on the test sets, averaged over 10 runs with random weight initializations.

	GCN-conv	GCN-base	FFNN
Split 1	**69.77** (±2.24)	68.86 (±2.83)	61.89 (±3.31)
Split 2	74.39 (±3.10)	**74.76** (±4.81)	59.37 (±3.55)
Split 3	**77.35** (±3.80)	72.99 (±3.61)	69.04 (±3.39)
Split 4	**78.79** (±4.17)	77.70 (±3.16)	66.27 (±3.10)
Split 5	86.18 (±2.65)	**86.83** (±2.73)	74.65 (±6.43)
Split 6	79.97 (±3.14)	**80.60** (±2.98)	67.59 (±6.19)
Split 7	88.34 (±4.07)	**92.25** (±2.57)	76.46 (±8.34)
Split 8	72.43 (±6.03)	**73.11** (±3.64)	62.47 (±8.86)
Split 9	**62.38** (±3.69)	62.20 (±3.99)	51.44 (±1.11)
Split 10	**73.53** (±2.95)	72.42 (±3.99)	59.57 (± 3.25)
Total	**76.31** (±7.65)	76.17 (±8.68)	64.88 (±7.53)

The GCN-base and GCN-conv networks outperformed the FFNN network for all splits. No significant between GCN-base and GCN-conv in average accuracy over all ten splits was observed, which suggest that the added complexity of a convolutional module may be unnecessary for the AD patient data.

Experiments performed on the AD patient data were mostly consistent with those performed on the simulated data, see Tab. 4.1. The main difference compared to the simulated data is that the FFNN performs better on the AD patient data. This is expected since the order of nodes is fixed, which is central for the FFNN network's ability to learn general patterns in the data, see Section 4.1.

The trained networks displayed a bias towards predicting class 2, as shown in Tab. 4.3. This tendency was observed for most splits. The FFNN network was particulary biased, which resulted in a significant number of subjects from the healthy control group being falsely classified as AD patients. Although the same tendency was observed for the GCN-base and GCN-conv networks, it was to a lesser degree. Split 7 stand out, in that the bias towards class 2 is no longer evident for the GCN-base and GCN-conv networks, see Tab. 4.4. This reflects the difference in strategy between GCNs and the FFNN, as the bias is still apparent for the FFNN network. Note that split 7 also produced the best overall accuracy for all of the tested ANNs, see 4.2.

There is considerable variance in performance ability between different splits, indicative of generalization difficulties for some splits. This could be due to the limited amount of subjects in the AD patient data. This was corroborated by the fact that an increase in the number of subjects used for training resulted in higher valida-

Table 4.3: Confusion matrices on predictions for split 5. Class 1 corresponds to samples generated from the healthy control group, and class 2 to samples from the combined set of all AD patients.

	Predicted 1	Predicted 2
Class 1	2590.6	2409.4
Class 2	125.3	4874.7

(a) FFNN predictions.

	Predicted 1	Predicted 2
Class 1	3882.4	1117.6
Class 2	199.1	4800.9

(b) GCN-base predictions.

	Predicted 1	Predicted 2
Class 1	3866.1	1133.9
Class 2	248.4	4751.6

(c) GCN-conv predictions.

Table 4.4: Confusion matrices on predictions for split 7. Class 1 corresponds to samples generated from the healthy control group, and class 2 to samples from the combined set of all AD patients.

	Predicted 1	Predicted 2
Class 1	2915.9	2084.1
Class 2	269.6	4730.4

(a) FFNN predictions.

	Predicted 1	Predicted 2
Class 1	4666.9	333.1
Class 2	442	4558

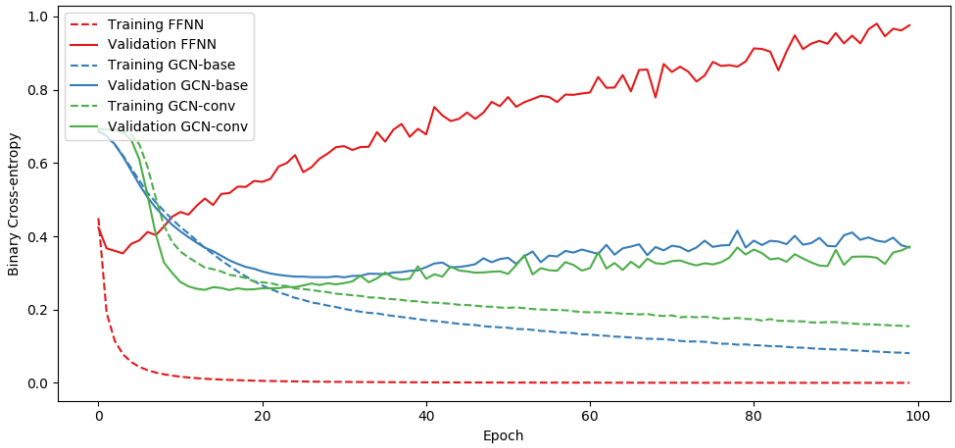
(b) GCN-base predictions.

	Predicted 1	Predicted 2
Class 1	4381.4	618
Class 2	547.6	4452.4

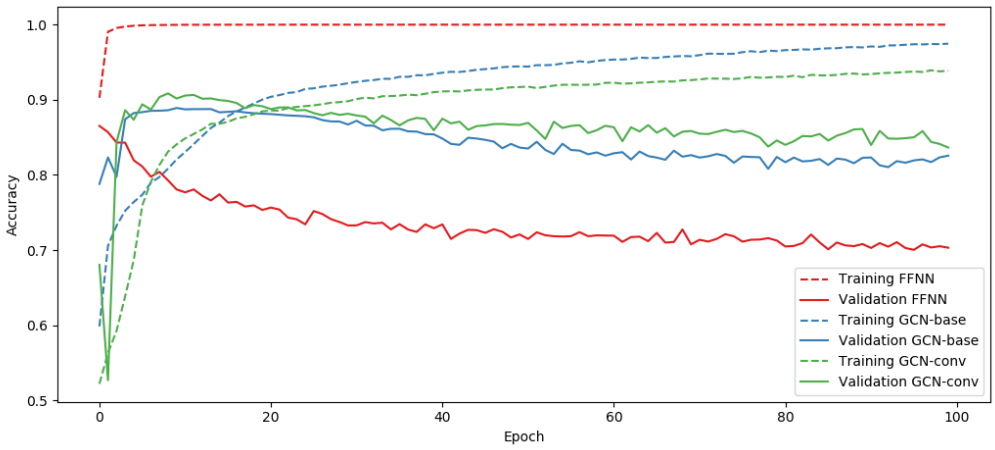
(c) GCN-conv predictions.

tion accuracies. A 70/30 split of subjects into training and validation was used to investigate the progression of training and validation accuracy during training. In general, considerably higher validation accuracies were recorded for the 70/30 splits than for a 40/30/30 split, see Fig. 4.3. Typically, a validation accuracy around 90% was reached. This indicates that the accuracy in Tab. 4.2 is sub-optimal due to the limited amount of subjects in the AD patient data. The 70/30 split was problematic for evaluation of network performance, since no separate test set was available. As such, no performance is presented for 70/30 splits, except for the characteristic learning curves in Fig. 4.3.

Figure 4.3: Network performance of a training session for a 70/30 split of subject into training and validation. Dropout with a factor of 0.5 and L2-regularization with a regularization rate of 1e-4 was applied to every layer in the networks in an attempt to reduce the risk of overfitting.



(a) Training and validation loss progression for 100 epochs.



(b) Training and validation accuracy progression for 100 epochs.

5

Discussion

The main focus of this thesis is classification of brain graphs using GCNs. The capabilities of GCNs are wholly dependent on the existence of structural differences between classes of brain graphs. Given the existence of structural differences, they should be detectable in graph measures. This was observed in a lot of the presented distributions of graph measures for the simulated data in Appendix A. However, it can be difficult to know which graph measures to look for. Also, classification by means of graph measures possibly require complex combinations of multiple graph measures, why it is not always easy to hand pick which measures to analyze. Artificial neural networks show great potential on such tasks, and this was also observed in the distributions in Appendix A. Especially in Fig. A.3 & A.7, where separation of classes based on graph measures seems daunting due to the major overlap in all presented graph measures. Still, GCNs were able to perform separation to a high degree.

Regarding how GCNs are able to classify graphs, a comparison with image recognition using regular CNNs is appropriate. First we want to highlight the fact that a shuffling of node ordering does not imply random permutations of elements in the adjacency matrix, only a reordering of rows and corresponding columns. Secondly, as suggested in Section 2.3, the meaning of a neighborhood in a graph is information on connections for individual nodes, i.e. rows in the adjacency matrix. Therefore a shuffling of node order only implies a shuffling of neighborhoods. For an image this corresponds to a reordering of areas in an image and not a random permutation of pixels. In a sense, a GCN is therefore able to recognize graphs with an arbitrary node ordering in the same way as a regular CNN is able to recognize a cat, regardless of location in an image.

For both the simulated and the experimental data, we chose to present results as averages over ten training sessions. The networks were mostly robust between training sessions on the simulated data, as observed in the small standard deviations in Tab. 4.1. This was not the case for the experimental data, where substantially higher standard deviations were observed, see Tab. 4.2. As previously discussed, the large variance in accuracy between splits presumably stems from the limited amount of patient data. However, there is also variance in the results of individual splits. Since only ten training sessions were performed due to time limitations, a question of statistical reliability of the results therefore exist. Combined with the

fact that the networks have to be considered somewhat unoptimized, the differences in predictive ability, or lack thereof, between GCN-conv and GCN-base is not fully investigated. This argumentation is also valid for the simulated data, although to a lesser degree, since the observed variance within individual data sets was much smaller.

Highlighted by the difference in predictive ability of the GCNs on the simulated data sets as well as for the experimental data, different graph structures present unique problems in terms of class separation. Graphs are also a very general data structure, why problems involving graphs covers a wide range of different areas. In much the same way as analysis of time series and image recognition benefits from different artificial neural network architectures, e.g. recurrent neural networks and convolutional neural networks, different graph neural network architectures are advantageous for different tasks on graph-structured data. As such, GCNs should not be viewed as a magical black box that can solve any problem involving graph-structured data, but one of many methods available when performing analysis on graphs.

A brief interpretation of the effects of stacking multiple graph convolutional layers was given in Section 3.3.1.1, where it was stated that each successive graph convolutional layer performs convolutions on larger and larger sub-scales. For clarification, this does not mean that the neighborhood of a node is expanded to include more nodes, the neighborhoods are fixed. What it does imply however is that the output from a graph convolutional layer contains information not only of a single node but of the entire neighborhood of that node. This output is passed on as a feature matrix to the following graph convolutional layer and so on. This brings forth a reasonable tactic of how to chose the number of graph convolutional layers in a model. Assume that the investigated graphs have characteristic path lengths of k. In using k consecutive graph convolutional layers, the last layer in the model will perform convolutions on information in approximately the entire graph. The choice of four graph convolutional layers was made by repeated testing on the simulated data, as no apparent increase in performance was observed when adding more layers. This could indicate that characteristic path lengths can be used to determine the number of graph convolutional layers utilized in a GCN, as the simulated data typically had characteristic path lengths of two to three. As this claim is only verified for the aforementioned simulated data, this is only a speculation on our part.

In terms of scalability, GCNs potentially have an advantage over fully connected neural networks. Where fully connected, deep neural networks with large input graphs quickly grows in terms of number of neurons, GCNs map graphs to an embedded space prior to the final separation done in a fully connected layer. As the classification step in GCNs, i.e. the final fully connected layer, is applied to embeddings of the graph, the number of neurons needed is proportional to the dimension of the embedded space, which can be much less than for a fully connected neural network with a minimum of N^2 neurons. This can drastically reduce time consumption as well as memory usage. However, the one dimensional convolutions used for the GCN-conv, GCN-sort and GCN-dummy can be time consuming, why there is

a potential problem with scalability. That said, the one dimensional convolutions are also performed in the embedded space and not applied to the adjacency matrix. Hence, if classes of graphs can be separated in a low dimensional embedded space, convolutions will not be overly expensive.

Only the featureless GCN approach and a novel approach using dummy features were investigated in this thesis. This was done for two reasons: To let the networks themselves extract meaningful embeddings, and to reduce bias in the simulated data. The bias comes from the fact that we had knowledge of how the simulated graphs differed, why graph-specific features that best separate the examined classes could have been given. This would be inherently biased, and not a test on how well GCNs are able to classify graph-structured data.

In the brief review of differences between a featureless GCN and a regular GCN, found in the last paragraph of Section 2.3.1, the featureless approach was established to have the drawback of mapping isomorphic graphs to different sets of embeddings. This was a concern when shuffling the ordering of sample graphs in the simulated data, and the motivation behind the use of dummy features. However, our results indicate that the GCNs were still able to produce relevant and separable embeddings for classification tasks on graphs.

It is not evident if the use of dummy features was beneficial or not. GCN-dummy resulted in the highest test score for data set 3, and was observed to outperform GCN-conv and GCN-base for data sets 7 & 8, see Tab. 4.1. However, the difference in test scores between GCN-dummy and GCN-sort for the aforementioned data sets are small, why it is reasonable to assume that the sorting trick (also implemented for the GCN-dummy) was the major contributor to the high test scores for classification on data sets 3, 7 & 8. That said, when removing an outlier in the results for data set 7, GCN-dummy was observed to perform better that GCN-sort for that data set also, which suggest an advantage in using dummy features for problems of a specific nature, see Fig. 4.1. In feeding dummy features to the GCNs, the network is guaranteed to produce the same set of embeddings for isomorphic graphs. This property is undoubtedly useful in classification tasks. At the same time, dummy features were in general observed to produce low test scores for most of the simulated data sets, why there are apparent drawbacks with using dummy features. A probable reason for the low test scores of GCN-dummy is the fact that every node has an initial feature vector of ones, why the multiplication of the adjacency matrix by the dummy feature matrix results in normalized node degree vectors of sorts. This could influence the network to be biased towards separation based on degree, and downplay differences in other important graph structures present.

There are two major differences between the simulated brain graphs and the graphs derived from the experimental data. Firstly, the simulated graphs were sparse, in the sense that the number of edges included is considerably less than the maximum number of edges possible in the graph, whereas graphs derived from experimental data were generally complete, i.e. all pairs of nodes were connected. Secondly the simulated graphs were binary, in contrast to the graphs derived from the experi-

mental data which were weighted. We see potential problems with applying GCNs to dense graphs as convolutions performed in the graph domain will include almost all nodes in the entire graph. This could lead to local information being blurred out, since convolutions are performed over more or less the same neighborhood for all nodes. This effect is lessened by the fact that the dense graphs derived from the experimental data are weighted. As such, even though the neighborhood of all nodes are equal, the strength of connections will vary. It would be interesting to investigate whether an increase in test accuracy could be achieved for the experimental data by binarization of the adjacency matrices. For instance, this could be done by the introduction of a threshold parameter, inferring binary edges if and only if the correlation between two cortical regions was stronger than the threshold value. This would serve as an initial filter on the experimental data, and lead to graphs more alike to the simulated graphs. On a positive side, noise would be removed and graph structures could be more easily detectable, but on a negative side, descriptive information could potentially be lost. Due to time limits, this was only briefly investigated, and no increase in the predictive ability for the GCNs was observed. We can therefore only speculate if higher accuracies could have been reached by tweaking the threshold parameter.

All samples of simulated graphs were equally dense, i.e. they had the same number of edges. This was important since we wanted the GCNs to perform classification on structures other than degree distribution. As an example, consider two graphs that are separable by simple summation of non-zero elements in the respective adjacency matrices. Firstly, the need for a GCN to perform classification is unnecessary, since separation is easily accomplished by hand. Secondly, tests performed on graphs that differed in density resulted in extremely high accuracies for both the GCNs and the fully connected neural networks after only a couple of training epochs. By keeping the density constant for all classes, the neural networks were forced to consider graph structures other than density present in the data.

The shuffling of node order, performed on the simulated data, was motivated by the fact that we did not want the artificial neural networks to perform classification based on index specific values. As an example, if nodes constituting communities such as those present in Fig. 3.6a always have the same fixed indices, an ANN would be compelled to only look for community structure on nodes sharing that same indexation. In order to constrain the ANNs to learn graph structure, a random permutation of node ordering was assigned to each sample. By shuffling the node order, any notion of node labeling is effectively gone, leaving only the actual graph structures intact. As such, GCNs are more likely to recognize the adjacency matrices in Fig. 3.6 as isomorphic graphs, even though index specific values are considerably different.

In Section 2.3.1, the featureless GCN approach was shown to be dependent of a consistent node order in order to map isomorphic graphs to the same set of values. This fact can be critical when implementing sorting. As isomorphic graphs with different adjacency matrix representations will not result in the same output embeddings, a sorting of the output will also be done differently. It is therefore not at all obvious

that sorting will lead to an increase in performance for a featureless approach. That said, a considerable increase in performance was observed in using sorting for data sets 3 & 7, see Tab. 4.1. The distributions of graph measures for these two data sets stand out in that there is significant overlap in all graph measures presented. We argue that a possible explanation of why sorting was profitable for these data sets is that differences between classes lie not on a global level, but on a nodal level. Global measures, such as transitivity and modularity, are not dependent on the ordering of nodes, why sorting does not effect the outcome. If classes differs only locally, e.g. if the maximum degree of each sample of one class is significantly higher than for the other, classification benefits from a fixed node ordering in the embedded space. Sorting can therefore be advantageous if a GCN can construct meaningful embeddings, reveling local differences. Although not entirely clear in the graph measures presented, a difference in extreme values in the centrality measures was observed in the rug plots for both data sets 3 & 7. This could indicate that GCN-sort indeed was better at recognizing differences in local measures. However, the modest differences in centrality measures can not alone explain the increase in performance for GCN-sort for these two data sets. Also, as only a selected few graph measures are presented, separation is conceivably possible in other graph measures not included in the analysis, why we can only hypothesize.

A big problem for the experimental data was the limited amount of patient recordings available. This was also verified by the fact that an increase in the number of patients included in the training set resulted in higher validation accuracies, see Fig. 4.3. The small number of patients in the experimental data also restricted the size of cohorts, why only ten subjects were included in each cohort. With more available data, the size of cohorts could be increased, resulting in more statistically reliable correlation coefficients between cortical regions in the "collective brain" of patient groups (AD and control). A significant amount of variation between cohorts from the same patient group was also observed, and the example adjacency matrices in presented in Fig. 3.9 are not representative of the full set of samples, but are presented merely to show examples of correlation between brain regions.

We also want to highlight that even though an arbitrary number of cohorts can be sampled from the set of patients, the information present in samples are inherently limited by the number of patients available. Depending on how large we select the cohorts to be, the diversity between adjacency matrices generated from aforementioned cohorts will vary. As such, the size of cohorts have a significant effect on the results. We spent little time in exploring different sizes of cohorts, and a size of ten was selected under the hypothesis that smaller sizes would be statistically unreliable, while larger would be bad for generalization. As an extreme example, all subjects put aside for training could have been included in every training cohort, resulting in identical adjacency matrices. This would most certainly lead to bad network generalization for the validation and test set.

A bias towards class two was observed in the predictions of the experimental data. Firstly we we want to stress the fact that an equal number of training samples were used for both classes, why the bias does not stem from an unfair distribution of

samples between classes. We did not have adequate time to investigate in detail where this bias comes from, why we can only speculate. One major difference between the two classes (AD and control) is the number of patients available. The control group consists of 110 patients in its entirety, whereas the combined AD patient group consists of 410 patients (57 mild, 238 medium and 115 severe). This brings forth two apparent qualities in the AD patient data. Firstly, the larger set of patients provides a basis of extracting different sample cohort to a higher degree than for the control group. Secondly, there is potentially also more diversity in the adjacency matrices derived from AD patients, as the AD patient data include patients of different disease progressions. By this reasoning, samples derived from AD patients are likely to be more spread out in the embedded space where the networks perform classification. This could make the network susceptible towards overfitting on outliers in the AD patients class and result in bad generalizability.

In combining all AD patients into a single group, a question whether samples derived from cohorts that by random happen to include many subjects in milder stages of AD are more similar to samples derived from the healthy control group is raised. This could partly explain why certain training/validation/test splits were more or less successful than others. As subjects in different stages of disease progression are included into training, validation and test by random, the loss in accuracy in some splits could be explained by the fact that certain test or training splits contain different number of subjects in the milder stages of disease progression.

Noteworthy for the experiments on the experimental data is that the GCNs does not receive any information on cortical thickness values in absolute numbers, only the connectivity between cortical regions. As differences in the overall cortical thickness values between the AD patients and the healthy control group may be present, potentially useful information is overlooked in the classification process. Whether such differences were present was however not investigated in this thesis, neither did we investigate how to incorporate information on cortical thickness values into our model networks.

In the forward propagation of the GCNs the adjacency matrix is normalized by

$$\tilde{D}^{-\frac{1}{2}}\tilde{A}\tilde{D}^{-\frac{1}{2}},$$

see Section 2.3.1. For the simulated data, the adjacency matrices are binary, why normalization is performed on the number of connections. For the experimental data, the adjacency matrices are however weighted. The degree matrix is therefore replaced by a strength matrix, i.e. a diagonal matrix with the sum of all strengths of connections incident to each respective node. This is yet another difference between the simulated and experimental data. In normalizing the strengths of connections for the experimental data, some information of how strong cortical regions correlate to one another is lost, and it is uncertain whether this is profitable for the networks. There could be differences in the magnitude of correlations between cortical regions between AD patients and the healthy control group that gets blurred out by normalization. We did not have the time to explore alternatives or possible drawbacks of the normalization, or if the GCNs would perform better without normalization

altogether.

The novel approach using dummy features was not explored for the experimental data. This was mostly due to the fact that a fixed node ordering could be imposed on the experimental data from labels of the cortical regions. With a fixed node ordering, all isomorphic graphs will have identical adjacency matrices, why there is no problem with filters (or weights) in the network being applied to arbitrary elements in the input matrices. As the use of dummy features were motivated by this precise fact, it seemed redundant to perform experiments using dummy features on the experimental data. Any expectation of an increase in predictive ability implementing dummy features on the experimental data was also small, since little to none improvement was observed for experiments on the simulated data.

6

Outlook

All experiments in this thesis were based on a featureless approach, i.e. no network received any predefined graph-specific features as input. In feeding explicit graph measures as input, networks can become biased towards looking only on the specified measures. This was inherently problematic for the simulated data where a preconceived notion on how classes differed was inevitable. This is perhaps not a problem for experimental data, but a choice still has to be made on which specific features to feed to the networks. On the other hand, if a notion exist of which graph measures characterises the classes, there might not be a need to train a GCN to perform the separation in the first place. Another idea might be to instead extract features from the graphs by some statistical scheme. There are many algorithms that construct nodal embeddings based on random walks, e.g. Node2Vec[37], DeepWalk[20]. However, the use of non-deterministic methods might become problematic for supervised learning tasks, since the graph mappings are not uniquely defined. In order to utilize random walks for the generation of node features in a supervised setting, it might be necessary to design a random walk scheme that guarantees some form of convergence. That said, a GCN approach using graph-specific features for brain connectivity analysis is definitely worth exploring going forward.

We also see potential in applying GCNs to functional brain data, e.g. fMRI and EEG, for two reasons. Firstly, there are natural ways of constructing brain graphs from functional data. Analogously to the correlation-based adjacency matrices analyzed in this thesis, connections can be established on an individual level since there are multiple measurements for each distinct brain region. Secondly, given a time series, recurrent neural networks could be utilized to extract features on a nodal level, later passed as input features to a GCN. For instance, *Long Short-Term Memory*[38] networks have been applied to a multitude of problems involving time series data with promising results.[39][40] GCNs have also been used for link prediction tasks with promising results.[19][17] Applications of link prediction on brain graphs could for instance involve completion of graphs derived from measurement techniques with some uncertainty associated with them. Link prediction with GCNs could also be used to analyze a potential correlation between the structural and functional brain, or infer edges in a structural brain graph by means of analysis on the functional brain.

Another interesting approach is to include graphs derived from multiple brain imag-

ing techniques. Theoretically, many different graphs of varying sizes could be passed as input to a GCN. This could be done by combining all adjacency matrices into a single block-diagonal matrix. In doing this, no information will "move between graphs", while still including more information for the GCNs to work with.

Finally, as the focus of this thesis was not to optimize network architectures by tuning hyper-parameters, much work can be done on improving network performance in terms of absolute numbers. As such, all results in this thesis only constitute a lower bound.

References

[1] Rahul S Desikan, Florent Ségonne, Bruce Fischl, Brian T Quinn, Bradford C Dickerson, Deborah Blacker, Randy L Buckner, Anders M Dale, R Paul Maguire, Bradley T Hyman, et al. An automated labeling system for subdividing the human cerebral cortex on mri scans into gyral based regions of interest. *Neuroimage*, 31(3):968–980, 2006.

[2] Jie Zhou, Ganqu Cui, Zhengyan Zhang, Cheng Yang, Zhiyuan Liu, Lifeng Wang, Changcheng Li, and Maosong Sun. Graph neural networks: A review of methods and applications. *arXiv preprint arXiv:1812.08434*, 2018.

[3] Ziwei Zhang, Peng Cui, and Wenwu Zhu. Deep learning on graphs: A survey. *IEEE Transactions on Knowledge and Data Engineering*, 2020.

[4] Zonghan Wu, Shirui Pan, Fengwen Chen, Guodong Long, Chengqi Zhang, and S Yu Philip. A comprehensive survey on graph neural networks. *IEEE Transactions on Neural Networks and Learning Systems*, 2020.

[5] Stephen P Borgatti, Ajay Mehra, Daniel J Brass, and Giuseppe Labianca. Network analysis in the social sciences. *science*, 323(5916):892–895, 2009.

[6] Jens Krause, Darren P Croft, and Richard James. Social network theory in the behavioural sciences: potential applications. *Behavioral Ecology and Sociobiology*, 62(1):15–27, 2007.

[7] David K Duvenaud, Dougal Maclaurin, Jorge Iparraguirre, Rafael Bombarell, Timothy Hirzel, Alán Aspuru-Guzik, and Ryan P Adams. Convolutional networks on graphs for learning molecular fingerprints. In *Advances in neural information processing systems*, pages 2224–2232, 2015.

[8] Steven Kearnes, Kevin McCloskey, Marc Berndl, Vijay Pande, and Patrick Riley. Molecular graph convolutions: moving beyond fingerprints. *Journal of computer-aided molecular design*, 30(8):595–608, 2016.

[9] Yoav Goldberg. A primer on neural network models for natural language processing. *Journal of Artificial Intelligence Research*, 57:345–420, 2016.

[10] Maximilian Nickel, Kevin Murphy, Volker Tresp, and Evgeniy Gabrilovich. A review of relational machine learning for knowledge graphs. *Proceedings of the IEEE*, 104(1):11–33, 2015.

[11] Gary L Wenk et al. Neuropathologic changes in alzheimer's disease. *Journal of Clinical Psychiatry*, 64:7–10, 2003.

[12] Charles Anthony Davie. A review of parkinson's disease. *British medical bulletin*, 86(1):109–127, 2008.

[13] Mite Mijalkov, Ehsan Kakaei, Joana B Pereira, Eric Westman, Giovanni Volpe, Alzheimer's Disease Neuroimaging Initiative, et al. Braph: a graph theory software for the analysis of brain connectivity. *PloS one*, 12(8), 2017.

[14] Emily S Finn, Xilin Shen, Dustin Scheinost, Monica D Rosenberg, Jessica Huang, Marvin M Chun, Xenophon Papademetris, and R Todd Constable. Functional connectome fingerprinting: identifying individuals using patterns of brain connectivity. *Nature neuroscience*, 18(11):1664, 2015.

[15] Joaquín Goñi, Martijn P van den Heuvel, Andrea Avena-Koenigsberger, Nieves Velez de Mendizabal, Richard F Betzel, Alessandra Griffa, Patric Hagmann, Bernat Corominas-Murtra, Jean-Philippe Thiran, and Olaf Sporns. Resting-brain functional connectivity predicted by analytic measures of network communication. *Proceedings of the National Academy of Sciences*, 111(2):833–838, 2014.

[16] Xiaoxiao Li, Nicha C. Dvornek, Yuan Zhou, Juntang Zhuang, Pamela Ventola, and James S. Duncan. Graph neural network for interpreting task-fmri biomarkers. *CoRR*, abs/1907.01661, 2019.

[17] Thomas N Kipf and Max Welling. Variational graph auto-encoders. *arXiv preprint arXiv:1611.07308*, 2016.

[18] Thomas N Kipf and Max Welling. Semi-supervised classification with graph convolutional networks. *arXiv preprint arXiv:1609.02907*, 2016.

[19] Muhan Zhang and Yixin Chen. Link prediction based on graph neural networks. In *Advances in Neural Information Processing Systems*, pages 5165–5175, 2018.

[20] Bryan Perozzi, Rami Al-Rfou, and Steven Skiena. Deepwalk: Online learning of social representations. In *Proceedings of the 20th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 701–710, 2014.

[21] Marco Gori, Gabriele Monfardini, and Franco Scarselli. A new model for learning in graph domains. In *Proceedings. 2005 IEEE International Joint Conference on Neural Networks, 2005.*, volume 2, pages 729–734. IEEE, 2005.

[22] Franco Scarselli, Marco Gori, Ah Chung Tsoi, Markus Hagenbuchner, and

Gabriele Monfardini. The graph neural network model. *IEEE Transactions on Neural Networks*, 20(1):61–80, 2008.

[23] David K Hammond, Pierre Vandergheynst, and Rémi Gribonval. Wavelets on graphs via spectral graph theory. *Applied and Computational Harmonic Analysis*, 30(2):129–150, 2011.

[24] Michaël Defferrard, Xavier Bresson, and Pierre Vandergheynst. Convolutional neural networks on graphs with fast localized spectral filtering. In *Advances in neural information processing systems*, pages 3844–3852, 2016.

[25] Muhan Zhang, Zhicheng Cui, Marion Neumann, and Yixin Chen. An end-to-end deep learning architecture for graph classification. In *Thirty-Second AAAI Conference on Artificial Intelligence*, 2018.

[26] Ed Bullmore and Olaf Sporns. Complex brain networks: graph theoretical analysis of structural and functional systems. *Nature reviews neuroscience*, 10(3):186–198, 2009.

[27] Olaf Sporns, Giulio Tononi, and Rolf Kötter. The human connectome: a structural description of the human brain. *PLoS computational biology*, 1(4), 2005.

[28] Martijn P Van Den Heuvel and Hilleke E Hulshoff Pol. Exploring the brain network: a review on resting-state fmri functional connectivity. *European neuropsychopharmacology*, 20(8):519–534, 2010.

[29] Danielle Smith Bassett and ED Bullmore. Small-world brain networks. *The neuroscientist*, 12(6):512–523, 2006.

[30] Nicholas J Higham. *Analysis of the Cholesky decomposition of a semi-definite matrix*. Oxford University Press, 1990.

[31] Victor Y. Pan, Zhao Q. Chen, and Ailong Zheng. The complexity of the algebraic eigenproblem, 1998.

[32] Martín Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Greg S. Corrado, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Ian Goodfellow, Andrew Harp, Geoffrey Irving, Michael Isard, Yangqing Jia, Rafal Jozefowicz, Lukasz Kaiser, Manjunath Kudlur, Josh Levenberg, Dandelion Mané, Rajat Monga, Sherry Moore, Derek Murray, Chris Olah, Mike Schuster, Jonathon Shlens, Benoit Steiner, Ilya Sutskever, Kunal Talwar, Paul Tucker, Vincent Vanhoucke, Vijay Vasudevan, Fernanda Viégas, Oriol Vinyals, Pete Warden, Martin Wattenberg, Martin Wicke, Yuan Yu, and Xiaoqiang Zheng. TensorFlow: Large-scale machine learning on heterogeneous systems, 2015. Software available from tensorflow.org.

[33] Strogatz S. Watts D. Collective dynamics of 'small-world' networks. *Nature*, 393:440–442, 1998.

[34] Lazaros K Gallos, Hernán A Makse, and Mariano Sigman. A small world of weak ties provides optimal global integration of self-similar modules in functional brain networks. *Proceedings of the National Academy of Sciences*, 109(8):2825–2830, 2012.

[35] Danielle S Bassett, Edward T Bullmore, Andreas Meyer-Lindenberg, José A Apud, Daniel R Weinberger, and Richard Coppola. Cognitive fitness of cost-efficient brain functional networks. *Proceedings of the National Academy of Sciences*, 106(28):11747–11752, 2009.

[36] Volpe G Mijalkov M, Pereira JB. Delayed correlations improve the reconstruction of the brain connectome. *PLoS ONE*, 15(2), 2020.

[37] Aditya Grover and Jure Leskovec. node2vec: Scalable feature learning for networks. In *Proceedings of the 22nd ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 855–864, 2016.

[38] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural computation*, 9:1735–80, 12 1997.

[39] Shuohang Wang and Jing Jiang. Learning natural language inference with lstm. *arXiv preprint arXiv:1512.08849*, 2015.

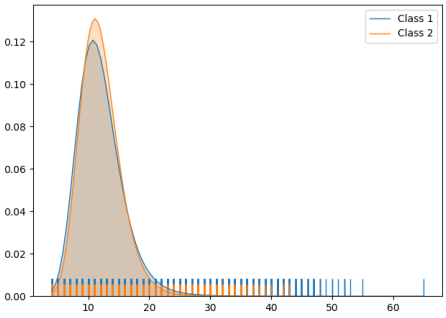
[40] Fazle Karim, Somshubra Majumdar, Houshang Darabi, and Shun Chen. Lstm fully convolutional networks for time series classification. *IEEE access*, 6:1662–1669, 2017.

A

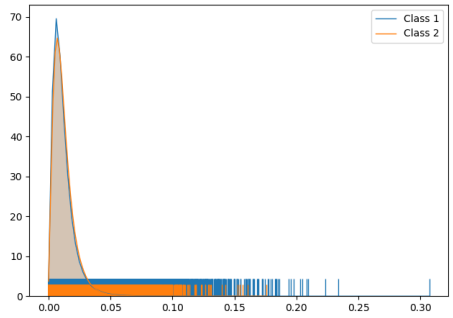
Distributions of graph measures for simulated data

In this chapter, we present class distributions for a selection of graph measures, calculated according to Section 2.2, for the 10 simulated data sets (D1-D10) described in Section 3.1. For the local measures, i.e the centrality measures along with the local clustering coefficients, distributions are formed from individual node features. It is therefor not a distribution over samples, but a distribution over all nodes in all sample graphs combined. For each data set, we present which randomly initialized class specific parameters were used in the simulation. The four parameters, corresponding to probabilities of adding a new edge in a certain way, are denoted p_p (preferential attachment), p_t (completing triangles), p_m (forming communities) and p_u (random connections). For details on these four ways of adding edges, see Section 3.1.1. In order to make extreme values more visible, we also include a *rugplot* together with all distributions.

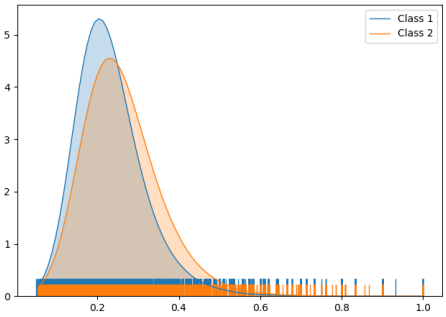
Figure A.1: Graph measure distributions for data set 1 (D1).
Class 1: $p_p = 0.314,\ p_t = 0.183,\ p_m = 0.249,\ p_u = 0.253.$
Class 2: $p_p = 0.236,\ p_t = 0.035,\ p_m = 0.459,\ p_u = 0.270.$



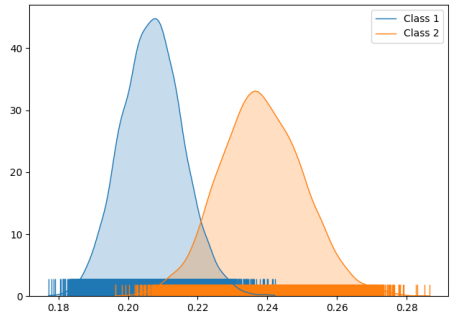
(a) Degree centrality.



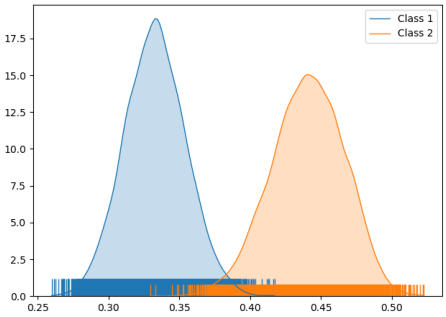
(b) Betweenness centrality.



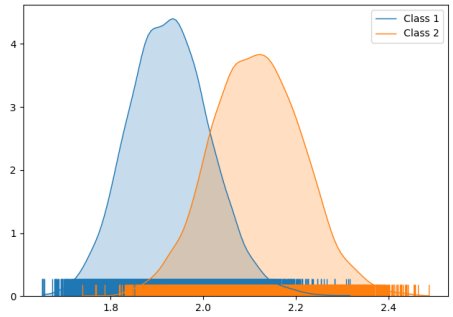
(c) Local clustering coefficient.



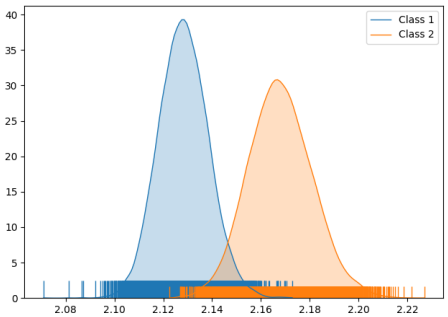
(d) Transitivity.



(e) Modularity.

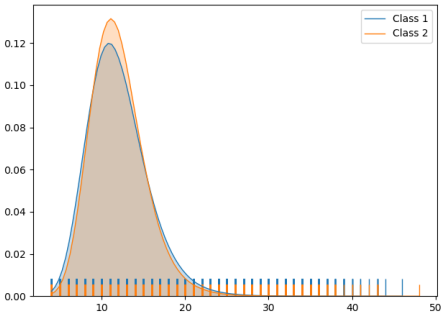


(f) Small-worldness.

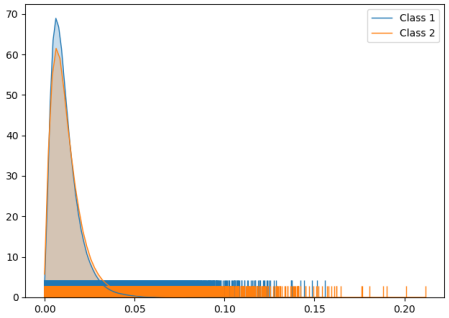


(g) Characteristic path length.

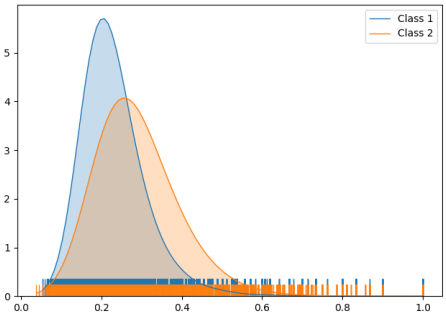
Figure A.2: Graph measure distributions for data set 2 (D2).
Class 1: $p_p = 0.209,\ p_t = 0.394,\ p_m = 0.057,\ p_u = 0.340.$
Class 2: $p_p = 0.252,\ p_t = 0.021,\ p_m = 0.534,\ p_u = 0.270.$



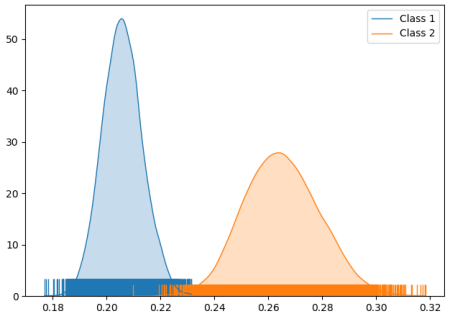
(a) Degree centrality.



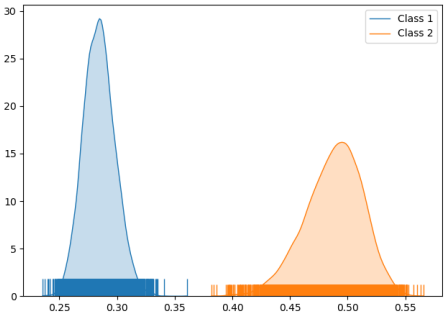
(b) Betweenness centrality.



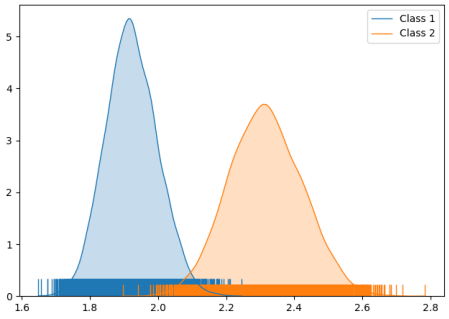
(c) Local clustering coefficient.



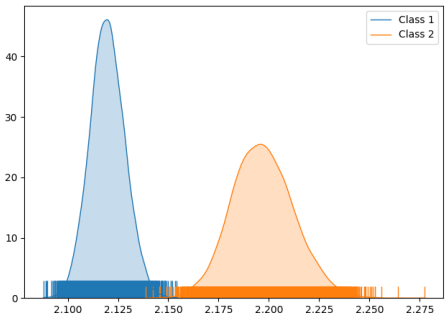
(d) Transitivity.



(e) Modularity.

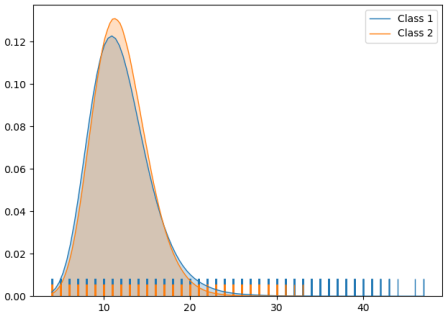


(f) Small-worldness.

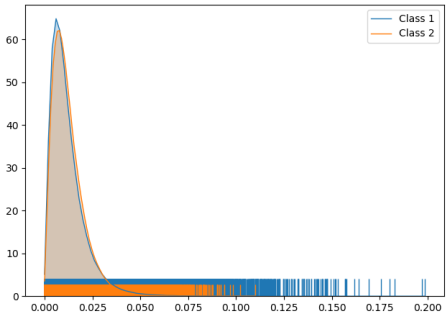


(g) Characteristic path length.

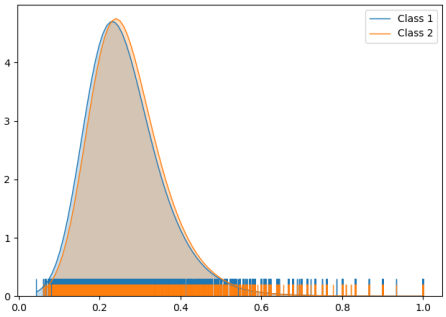
Figure A.3: Graph measure distributions for data set 3 (D3).
Class 1: $p_p = 0.257,\ p_t = 0.229,\ p_m = 0.335,\ p_u = 0.178.$
Class 2: $p_p = 0.100,\ p_t = 0.281,\ p_m = 0.347,\ p_u = 0.271.$



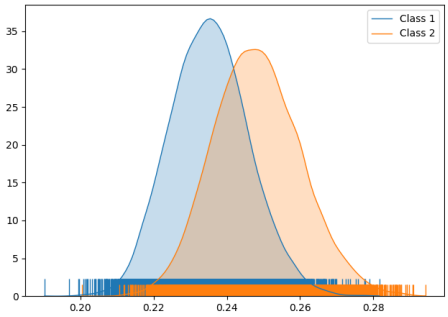
(a) Degree centrality.



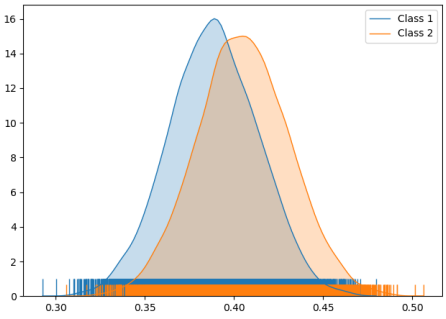
(b) Betweenness centrality.



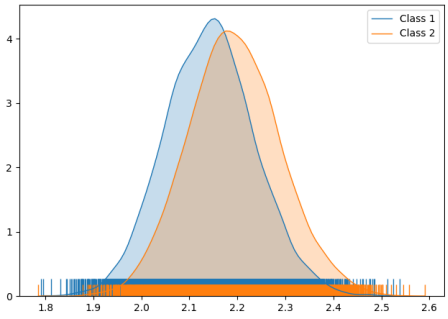
(c) Local clustering coefficient.



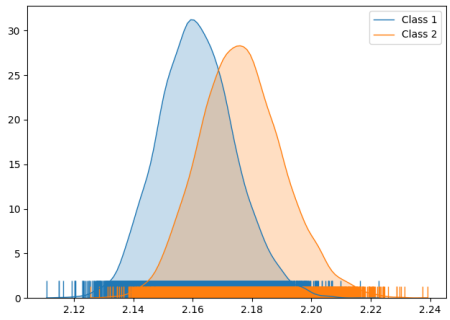
(d) Transitivity.



(e) Modularity.

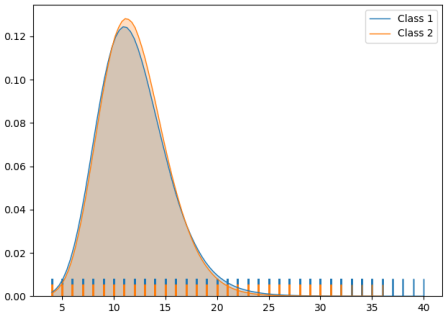


(f) Small-worldness.

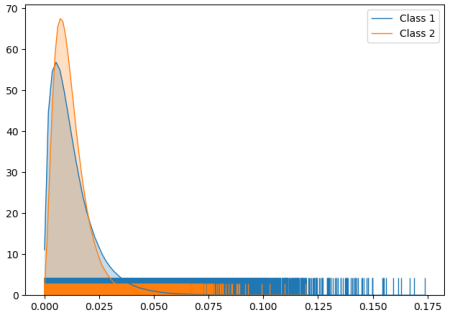


(g) Characteristic path length.

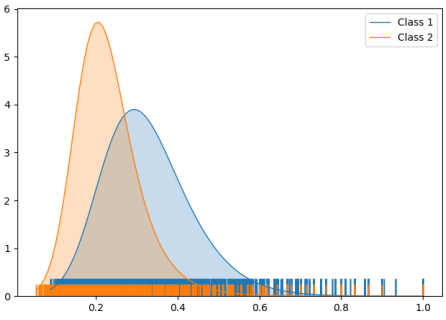
Figure A.4: Graph measure distributions for data set 4 (D4).
Class 1: $p_p = 0.173,\ p_t = 0.424,\ p_m = 0.365,\ p_u = 0.038.$
Class 2: $p_p = 0.114,\ p_t = 0.293,\ p_m = 0.199,\ p_u = 0.394.$



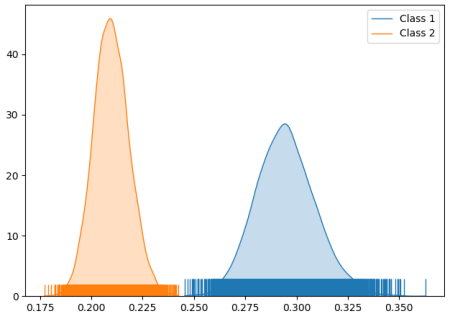
(a) Degree centrality.



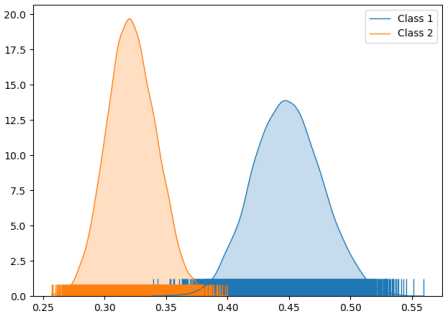
(b) Betweenness centrality.



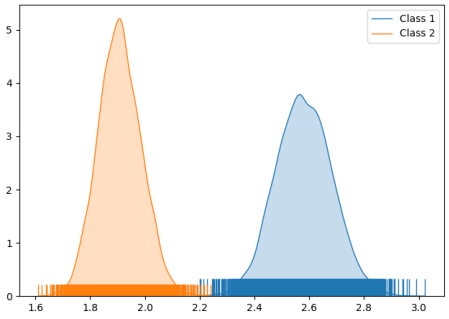
(c) Local clustering coefficient.



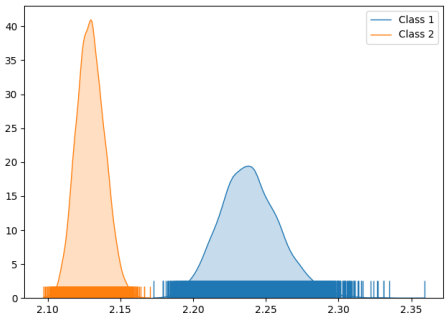
(d) Transitivity.



(e) Modularity.

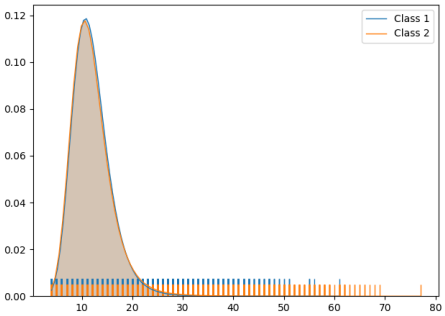


(f) Small-worldness.

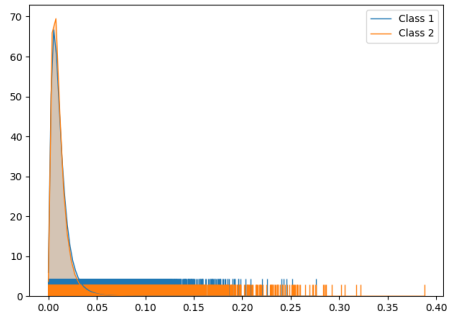


(g) Characteristic path length.

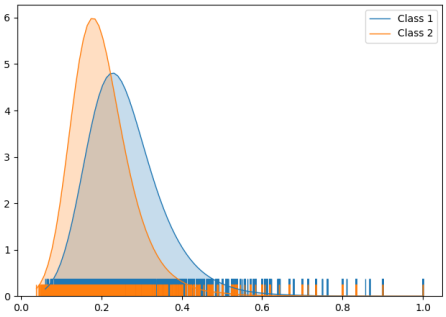
Figure A.5: Graph measure distributions for data set 5 (D5).
Class 1: $p_p = 0.297,\ p_t = 0.304,\ p_m = 0.259,\ p_u = 0.140.$
Class 2: $p_p = 0.409,\ p_t = 0.104,\ p_m = 0.118,\ p_u = 0.369.$



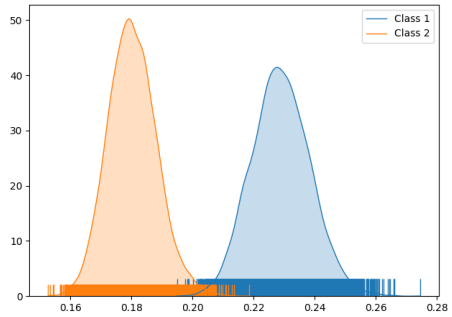
(a) Degree centrality.



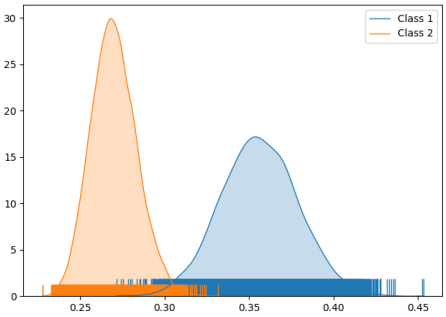
(b) Betweenness centrality.



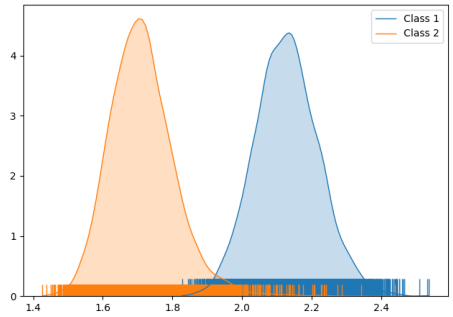
(c) Local clustering coefficient.



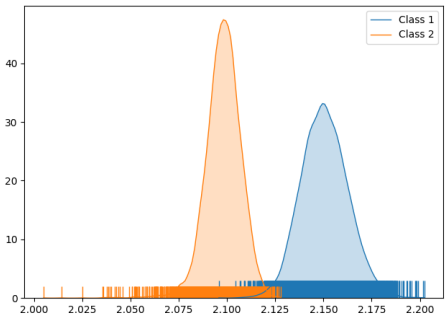
(d) Transitivity.



(e) Modularity.

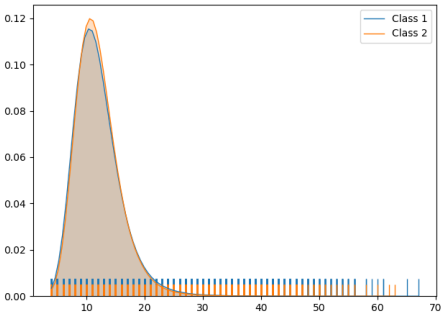


(f) Small-worldness.

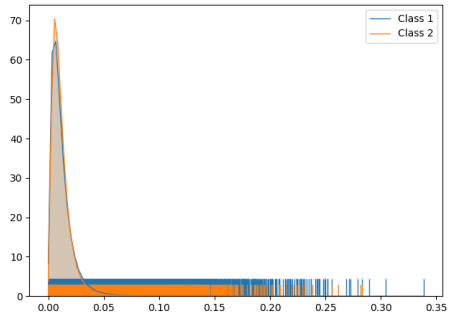


(g) Characteristic path length.

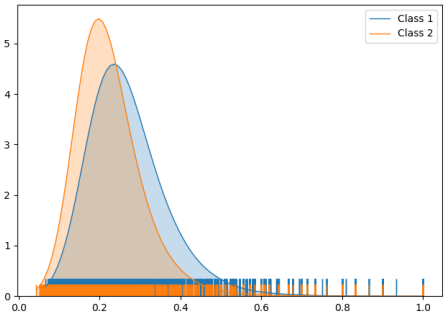
Figure A.6: Graph measure distributions for data set 6 (D6).
Class 1: $p_p = 0.345,\ p_t = 0.340,\ p_m = 0.245,\ p_u = 0.070$.
Class 2: $p_p = 0.332,\ p_t = 0.173,\ p_m = 0.217,\ p_u = 0.278$.



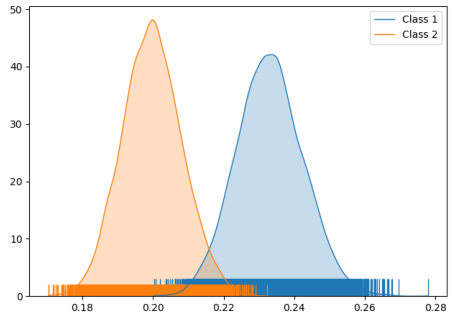
(a) Degree centrality.



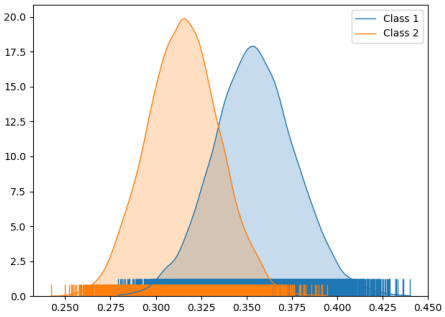
(b) Betweenness centrality.



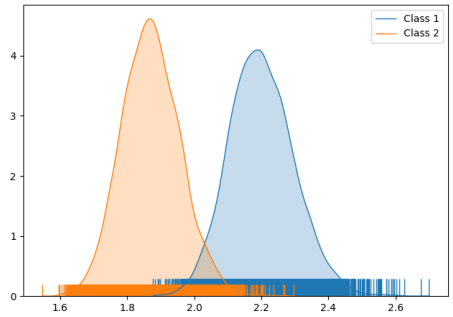
(c) Local clustering coefficient.



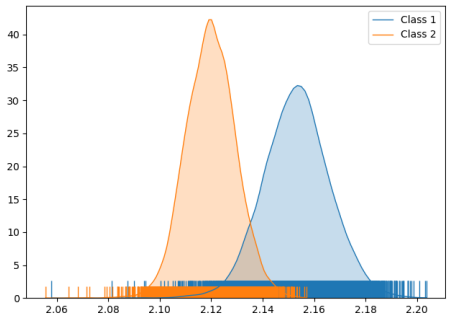
(d) Transitivity.



(e) Modularity.

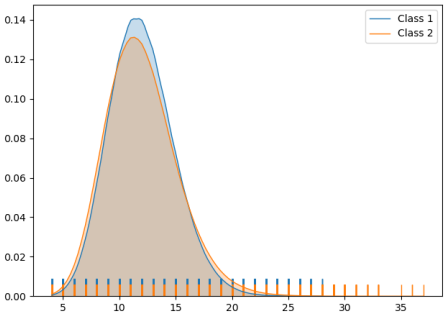


(f) Small-worldness.

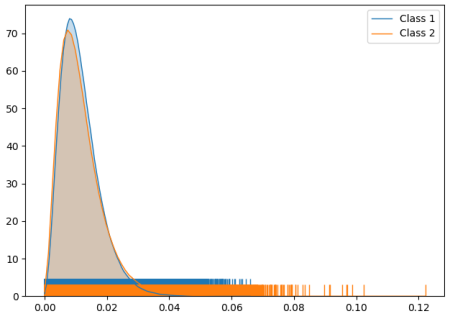


(g) Characteristic path length.

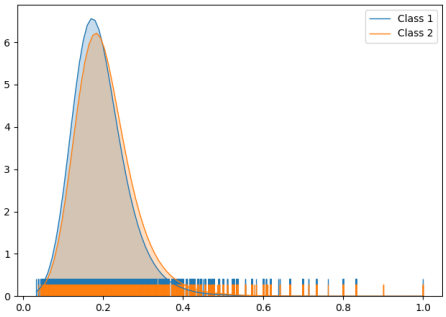
Figure A.7: Graph measure distributions for data set 7 (D7).
Class 1: $p_p = 0.039,\ p_t = 0.019,\ p_m = 0.264,\ p_u = 0.678$.
Class 2: $p_p = 0.150,\ p_t = 0.102,\ p_m = 0.234,\ p_u = 0.514$.



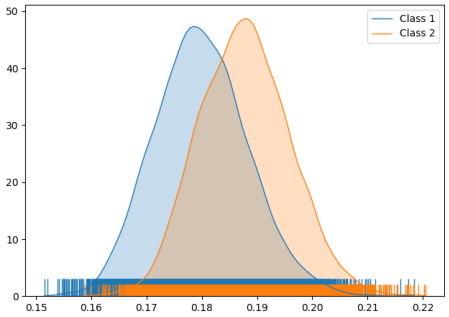
(a) Degree centrality.



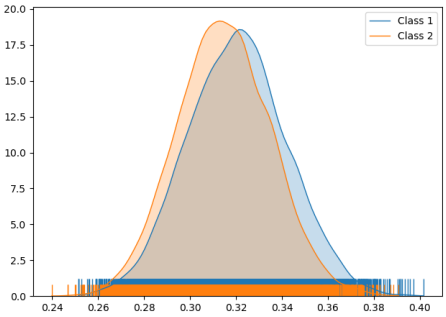
(b) Betweenness centrality.



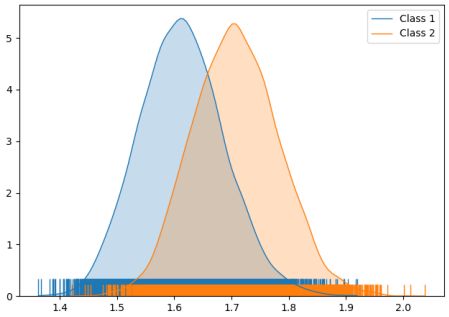
(c) Local clustering coefficient.



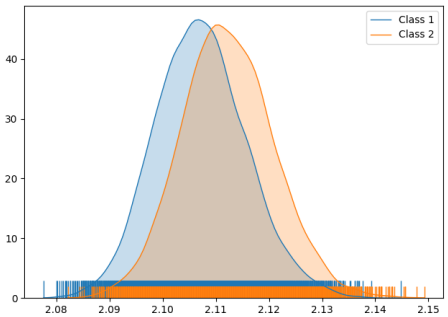
(d) Transitivity.



(e) Modularity.

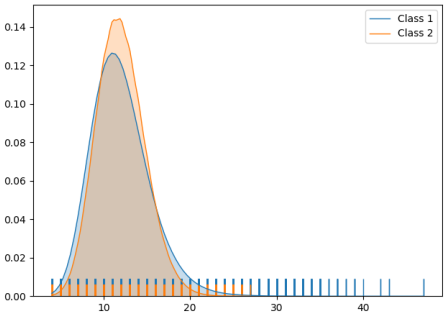


(f) Small-worldness.

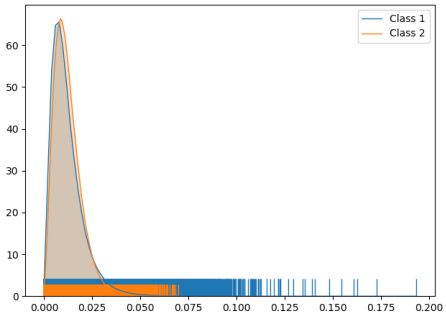


(g) Characteristic path length.

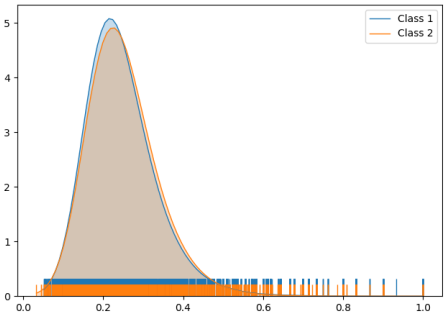
Figure A.8: Graph measure distributions for data set 8 (D8).
Class 1: $p_p = 0.210,\ p_t = 0.191,\ p_m = 0.319,\ p_u = 0.279.$
Class 2: $p_p = 0.011,\ p_t = 0.043,\ p_m = 0.452,\ p_u = 0.494.$



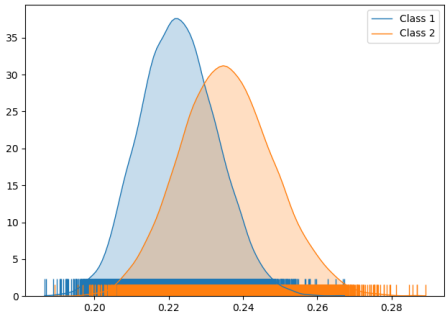
(a) Degree centrality.



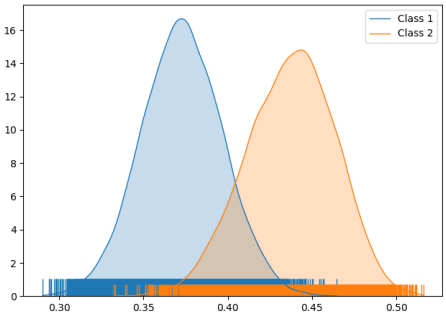
(b) Betweenness centrality.



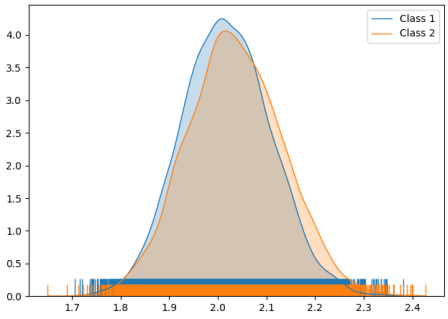
(c) Local clustering coefficient.



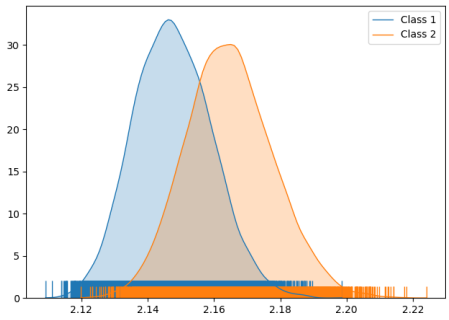
(d) Transitivity.



(e) Modularity.

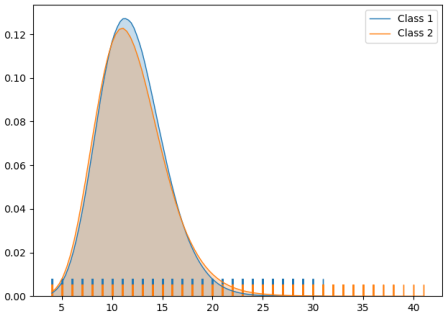


(f) Small-worldness.

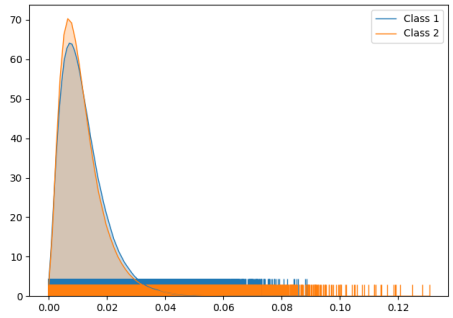


(g) Characteristic path length.

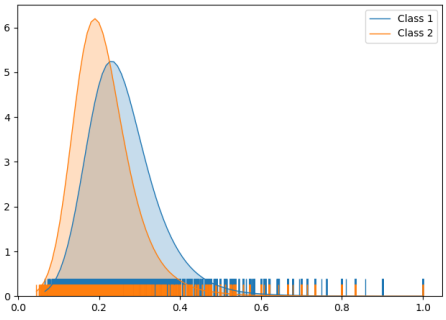
Figure A.9: Graph measure distributions for data set 9 (D9).
Class 1: $p_p = 0.047,\ p_t = 0.478,\ p_m = 0.165,\ p_u = 0.311.$
Class 2: $p_p = 0.193,\ p_t = 0.304,\ p_m = 0.054,\ p_u = 0.449.$



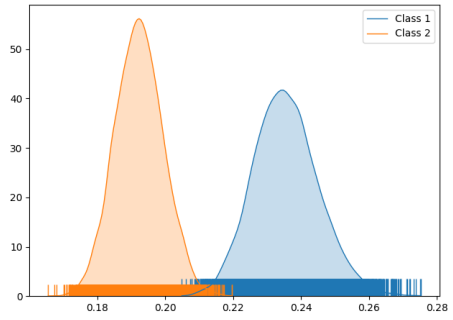
(a) Degree centrality.



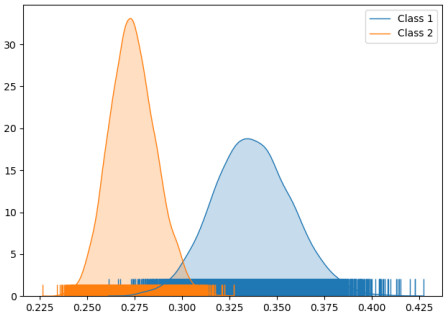
(b) Betweenness centrality.



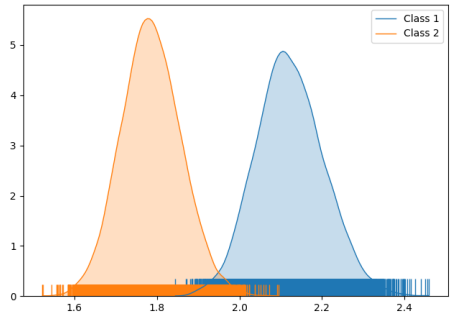
(c) Local clustering coefficient.



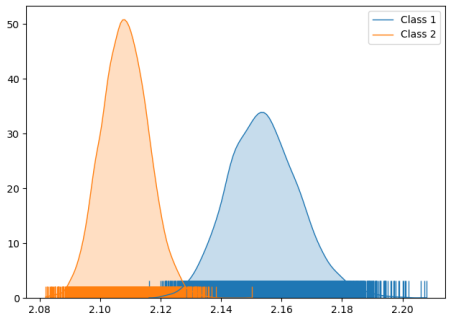
(d) Transitivity.



(e) Modularity.

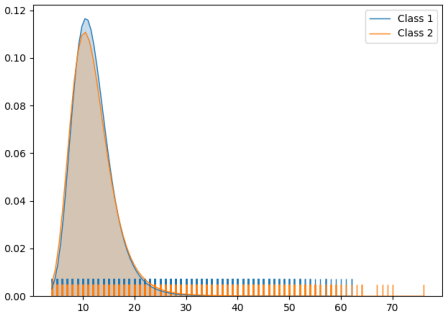


(f) Small-worldness.

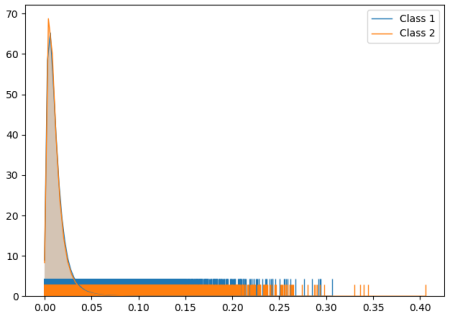


(g) Characteristic path length.

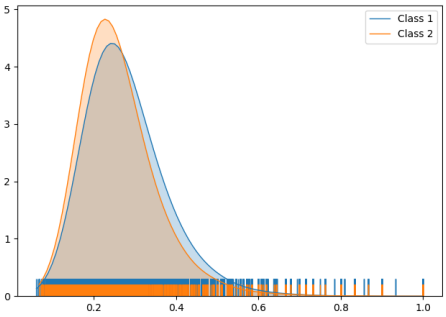
Figure A.10: Graph measure distributions for data set 10 (D10).
Class 1: $p_p = 0.334,\ p_t = 0.321,\ p_m = 0.293,\ p_u = 0.052.$
Class 2: $p_p = 0.363,\ p_t = 0.463,\ p_m = 0.066,\ p_u = 0.109.$



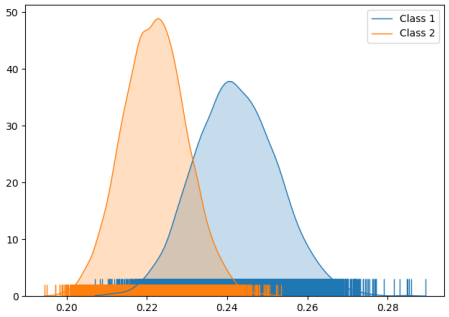
(a) Degree centrality.



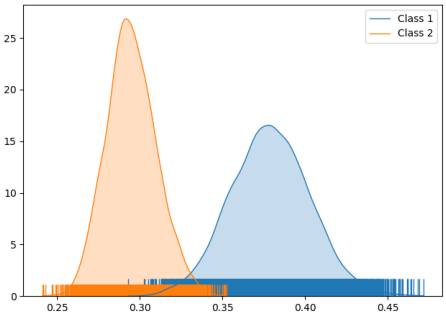
(b) Betweenness centrality.



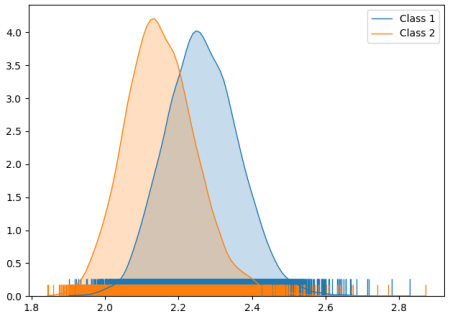
(c) Local clustering coefficient.



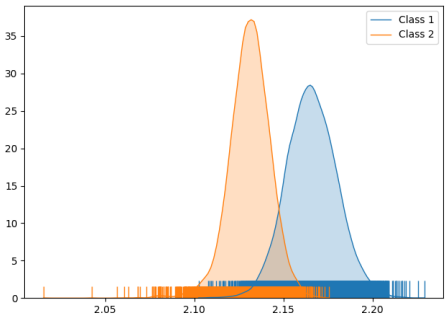
(d) Transitivity.



(e) Modularity.



(f) Small-worldness.



(g) Characteristic path length.

B

Network parameters

In this chapter, we present details on the architectures of all implemented artificial neural networks. Between the simulated and the experimental data, networks only differed due to the fact that the input graphs were of different sizes. All simulated graphs consisted of 100 nodes, whereas all graphs derived from MRI-data consisted of 68 nodes.

Table B.1: FFNN: Experimental data.

	Layer	Size	Activation	#Parameters
1	Fully Connected Layer	2 ×1	Softmax	9,250

Table B.2: GCN-base: Experimental data.

	Layer	Size	Activation	#Parameters
1	GC Layer	68 ×20	tanh	1,380
2	GC Layer	68 ×20	tanh	420
3	GC Layer	68 ×20	tanh	420
4	GC Layer	68 ×20	tanh	420
5	Concatenate	68 ×80	-	0
6	Reshape	4624 ×1	-	0
7	Fully Connected Layer	2 ×1	Softmax	10,882
				13,522

Table B.3: GCN-conv: Experimental data.

	Layer	Size	Activation	#Parameters
1	GC Layer	68 ×20	tanh	1,380
2	GC Layer	68 ×20	tanh	420
3	GC Layer	68 ×20	tanh	420
4	GC Layer	68 ×20	tanh	420
5	Concatenate	68 ×80	-	0
6	Reshape	4624 ×1	-	0
7	Conv. Layer	68 ×16	ReLU	992
8	Max Pooling	34 ×16	-	0
9	Reshape	544x 1	-	0
10	Fully Connected Layer	2 ×1	Softmax	1,090
				4,722

Table B.4: FFNN: Simulated data

	Layer	Size	Activation	#Parameters
1	Fully Connected Layer	2 ×1	Softmax	20,002

Table B.5: GCN-base: Simulated data

	Layer	Size	Activation	#Parameters
1	GC Layer	100 ×20	tanh	2,020
2	GC Layer	100 ×20	tanh	420
3	GC Layer	100 ×20	tanh	420
4	GC Layer	100 ×20	tanh	420
5	Concatenate	100 ×80	-	0
6	Reshape	4624 ×1	-	0
7	Fully Connected Layer	2 ×1	Softmax	16,002
				19,282

Table B.6: GCN-conv: Simimulated data

	Layer	Size	Activation	#Parameters
1	GC Layer	100 ×20	tanh	2,020
2	GC Layer	100 ×20	tanh	420
3	GC Layer	100 ×20	tanh	420
4	GC Layer	100 ×20	tanh	420
5	Concatenate	100 ×80	-	0
6	Reshape	8000 ×1	-	0
7	Conv. Layer	100 ×16	ReLU	1296
8	Max Pooling	50 ×16	-	0
9	Reshape	800 ×1	-	0
10	Fully Connected Layer	2 ×1	Softmax	1,602
				6,178

Table B.7: GCN-sort: Simulated data

	Layer	Size	Activation	#Parameters
1	GC Layer	100 ×20	tanh	2,020
2	GC Layer	100 ×20	tanh	420
3	GC Layer	100 ×20	tanh	420
4	GC Layer	100 ×20	tanh	420
5	Concatenate	100 ×80	-	0
6	Sort Layer	100 ×80	-	0
7	Reshape	8000 ×1	-	0
8	Conv. Layer	100 ×16	ReLU	1,296
9	Max Pooling	50 ×16	-	0
10	Reshape	800 ×1	-	0
11	Fully Connected Layer	2 ×1	Softmax	1,602
				6,178