Research Internship (PRE)

Field of Study: STIC
Scholar Year: 2018-2019

Characterization of cortical folding patterns by machine learning on graphs

Confidentiality Notice

Non-confidential report and publishable on Internet

Author:
Yaroslav Mavliutov

ENSTA ParisTech Tutor:
Alexandre Chapoutot

Promotion:
2020

Host Organism Tutor:
Guillaume Auzias
Sylvain Takerkart

Internship from 16/05/2019 to 11/08/2019

Name of the host organism: Institut de Neurosciences de la Timone
Address: Faculté de Médecine
27 Boulevard Jean Moulin
Marseille, France

Confidentiality Notice

This present document is not confidential. It can be communicated outside in paper format or distributed in electronic format.

Abstract

Deep learning (DL) has revolutionized many tasks in recent years, ranging from video processing to machine translation to audio recognition. Data of these problems are usually presented in Euclidean space.

This work describes alternative DL models that are adapted to data are generated from non-Euclidean domains. These data are represented as graphs that are derived from a topography, where the deepest parts of a cortical sulci are nodes of the graph.

In this project, we implemented GraphSage, DCNN, GAT to solve the problem of binary node classification. In a number of experiments on real dataset we demonstrate that selected methods outperform proposed trivial model. All models have been implemented in Python.

L'apprentissage profond a révolutionné de nombreuses tâches au cours des dernières années, allant du traitement vidéo à la traduction automatique en passant par la reconnaissance audio. Les données de ces problèmes sont généralement présentées dans l'espace Euclidien.

Ce travail décrit des modèles alternatifs d'apprentissage profond adaptés à des données générées à partir de domaines non-Euclidiens. Ces dernières sont représentées sous forme de graphes dérivés de la topographie où les parties les plus profondes de la sulci cortical sont des noeuds d'un graphe.

Dans ce projet, nous avons implémenté GraphSage, DCNN, GAT pour résoudre le problème de la classification binaire des noeuds. Dans un certain nombre d'expériences sur des ensembles de données réelles, nous démontrons que les méthodes sélectionnées surpassent le modèle trivial proposé. Tous les modèles ont été implémentés en Python.

Acknowledgment

The internship opportunity I had with the Timone Institute of Neurosciences (INT) was a great chance for learning and professional development. I am grateful for having a chance to meet professionals.

I am using this opportunity to express my deepest gratitude and special thanks to my tutors, Guillaume Auzias and Takerkart Sylvain, researchers at the Timone Institute of Neurosciences, for their help, giving necessary advices and guidance, generosity to share their knowledge.

I thank the MeCA team for their welcome support for the success of my goals. At INT I met wonderful people who arranged all facilities to make life easier.

I perceive as this opportunity as a big milestone in my career development. I will strive to use gained skills and knowledge in the best possible way, and I will continue to work on their improvement, in order to attain desired objectives.

Contents

Introduction

Recently in the neuroscience research field, interest increased regarding a characterization of the topography of the cortical surface, due to its high potential for finding biomarkers of cerebral pathologies or for the identification of new features associated with functional specificities [1][2][3].

More recently, researchers have begun to pay special attention to the deepest part of the sulci, and began intensive development of theoretical models of cortical anatomy. An important contribution to the development of cortical anatomy was provided by Lohmann in his fundamental work, which is especially important in defining the concept of sulcal pits [4]. This provides a more subtle spatial representation than those offered by sulci, because a sulcus can contain several sulcal pits.

It was found that the modeling of local patterns of sulcal pits as graphs was proved to be effective in finding relationships among genetic factors or characteristics of groups of patients [1][4]. These studies are based on statistical analysis between subjects of the comparison of pit-graphs. These comparisons use a carefully designed measure of similarity [1]. However, this approach has several limitations. First, the efficiency measure of similarity strongly depends on the choice of values of hyperparameters, which required a large set of specific experiments. Second, it is necessary to have strong a priori hypotheses regarding the spatial extent and size of the putative effect [5].

In this paper, we consider the problem of classifying graph nodes, where labels are binary using DL models. Standard approaches to solving the problem include engineering features of input graphs, graph kernels, and methods that define features of the graph in terms of random walks [6][7]. More closely related to our goal in this work are methods that learn graph features, such as Graph Neural Networks, spectral networks, Graph Convolutional Networks [8][9].

This survey provides an overview of these methods and, based on their comparison, selected models that are more suitable for solving the problem of binary classification will be implemented.

Part I

Technological and neurological context

I.1 The brain: a complex object of study

The brain is an important organ that has one of the key roles of the body, serves as the center of the nervous system.

The main function of the brain is to exert centralized control over other organs of the body. The brain affects other parts of the body by generating patterns of muscle activity and controlling the secretion of chemicals called hormones.

The human brain is the regulator of the whole life of the body from the smallest movements to the highest function of thinking. Parts of the brain and their functions include processing signals from receptor mechanisms [10].

The shape and size of the brain may vary depending on species (Fig. I.1). However, there are a number of principles of the brain architecture that apply to a wide range of species [11].

Figure I.1: The classic cerebral cortex

The cerebral cortex is the outer layer of the nervous tissue of the brain. In most mammals, apart from small mammals that have small brains, the cerebral cortex is folded, providing a greater surface area in the confined volume of the cranium. The cortical folding are crucial for the conduct of the brain and its functional organization [12].

The cerebral cortex is the largest site of neural integration in the central nervous system. It plays an important role in attention, perception, memory, speech and consciousness [10]. A ridge in the cortex is called a gyrus and a groove is called a sulcus (Fig. I.2).

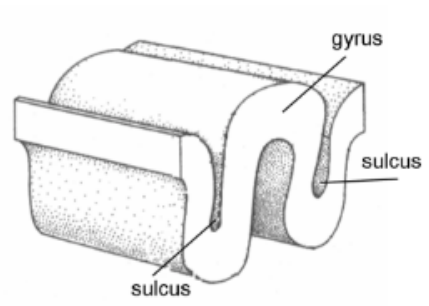

Figure I.2: Gyrus and sulcus

I.2 Deep sulcal landmarks

Charaterizing the cortical organization remains a major problem in neuroimaging [13]. The first step consists in extracting a 3D triangular mesh from an anatomical magnetic resonance image. We first perform the segmentation of the cortical ribbon and the cortical reconstruction using the freesurfer image analysis [5].

Recently, points of maximum depth within the folds, denoted as "sulcal pits" have been proposed as reliable cortical landmarks [1][4]. Conceptually, the "sulcal pits" refer to the inseparable atomic components of the entities located in the deeper parts of the sulci [14].

A surface-based approach has been proposed for the extraction of the sulcal pits from a reconstructed cortical surface tessellation [1]. Based on this method, several studies provided evidence of the interest of the sulcal pits as remarkable macro-anatomical features.

The sulcal pits extraction algorithm consists of two main steps: an assessment of a sulcal depth map and sulcal pits extraction using a watershed algorithm [15]. In order to filter noisy sulcal pits, the watershed is applied on a smoothed depth map until the water reaches a depth threshold and, if necessary, adjacent basins are drained. The decision to merge is based on the following features, when two basins meet at a ridge (Fig. I.3) [13]:

1. the ridge height (R) is the height difference between the shallowest pit and the ridge point (mm);

2. the basin area (A) is calculated by summing the Voronoi region areas of the vertices (mm^2);

3. the distance between the two pits (D) is the geodesic distance between the pits (mm).

During the flooding, the shallowest basin is merged with the deepest one if: $R < ThR$ and ($D < ThD$ or $A < ThA$), where ThR, ThD and ThA are three threshold values set to [$ThR = 2.5$ mm, $ThD = 15$ mm, $ThA = 30$ mm^2] respectively following the approach proposed in [13]. This filtering strategy aims to remove false extremes associated with anatomically irrelevant variations on the depth map.

The extraction almgorithm results in a set of selected vertices on the cortical surface corresponding to the filtered sulcal pits, and a set of corresponding sulcal basins. A natural way to formally represent the local folding pattern of a given part of the cortex is to construct a region adjacency graph from the set of sulcal pits and basins.

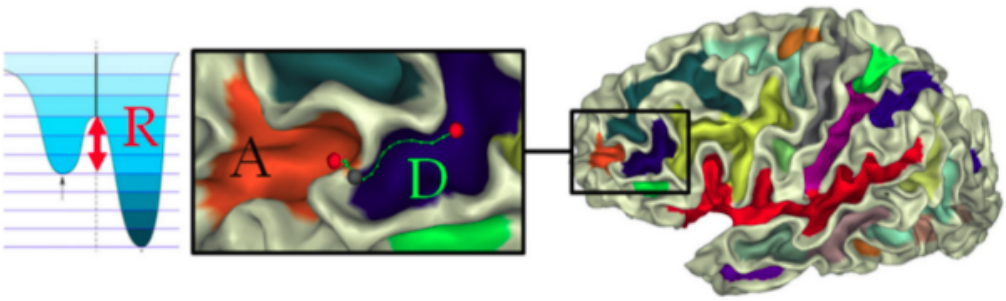

Figure I.3: Illustration of the merging procedure and the corresponding parameters: A is the area of the smallest basin, R is the height of the ridge, D is the geodesic distance between the two pits (red points)

I.3 Structural graph-based morphometry

In order to perform automatic morphometry based on the local organization of the cortex, high-level objects such as cortical sulci are studied. This has led to the development of Object-Based Morphometry (OBM), in which each sulcus is described by a large set of features [16]. Sulcal pits provide a more subtle spatial representation than those offered by sulci, because a single sulcus can contain several sulcal pits. The formal representation of sulcal pits as graph presented hereafter was introduced in [5].

The sulcal pits and corresponding basins are extracted for each individual cortical surface independently. To compare the pits between different subjects of the population S to be studied, we need to bring them into a common space. The authors use a "cortical registration" algorithm to map each individual folded cortical surface onto a cannonical sphere. Then, the information about the convexity of the folded surface is used to align major folds across individuals, on the spherical domain.

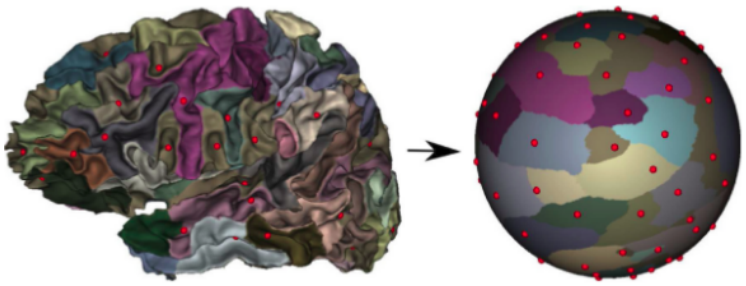

Figure I.4: Left: sulcal pits of one subject, represented on the cortical sheet. Right: after cortical registration to the template and projection to the common spherical space

This sphere, onto which all subjects are aligned, will serve as the common space onto which the authors project the sulcal pits and basins of every subject, as illustrated on Fig. I.4 [5]. For each subject $s \in S$, we obtain the set of sulcal pits $P^s = \{\pi_i^s\}_{i=1}^{i=N^s}$ on the sphere. Note that the number of pits N^s and corresponding basins $\{\beta_i^s\}_{i=1}^{i=N^s}$ can vary across subjects. A natural way to formally present the local folding pattern of a given part of the cortex is to construct a region adjacency graph from the set of sulcal pits and basins located within this region [1]. Given a subset $P \subset P^s$ of N pits ,$N \leq N^s$ each pit $\pi_i^s \in P$ defines as a node of the graph. The graph edges are then given by the spatial adjacency of their associated basins: this defines a binary adjacency matrix $A = (a_{ij}) \in R^{NxN}$ ($a_{ij} = 1$ if β_i and β_j are

adjacent, and 0 otherwise), that encodes the spatial organization of the folding pattern. We can then add attributes to each graph node in order to better characterize the folding pattern. Let $D = \{d_i\} \in R_N$ be the vector of depth values and $X = \{X_i\} \in R^{Nx3}$ be the matrix of coordinates of all graph nodes. Any pattern of sulcal pits can be represented by an attributed graph defined as $G = (P, A, D, X)$. Examples of graph based on sulcal pit are shown on Fig. I.5 [5].

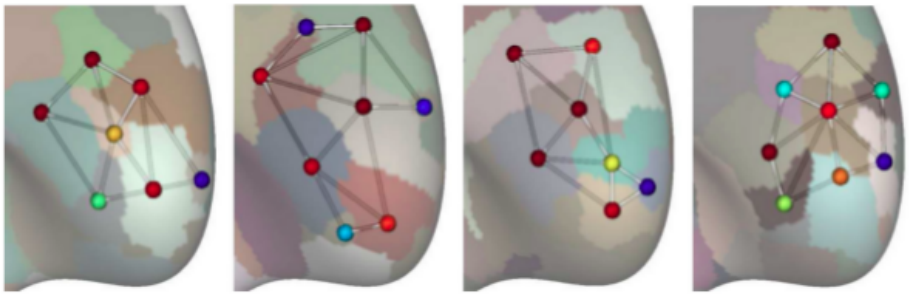

Figure I.5: Sulcal pit-graphs in the right frontal lobe for four subjects, displayed on the inflated cortex. The node color encodes the depth of the pit, from blue (shallow) to red (deep)

In [5], the authors study local patterns of sulcal pits by comparing them using proposed graph kernel that provides a new similarity measure between pit-graphs, in order to define a classification-based searchlight scheme that overcomes the limitations of ROI-based approaches.

In the current project, we will implement DL approaches to enable statistical learning on these complex graphs.

I.4 Graph Neural Networks

The recent success of neural networks has increased research on image recognition and intelligent data analysis. Many machine learning tasks, such as object detection, machine translation and speech recognition have become revolutionary as a result of different paradigms of DL. Taking an example of image analysis, an image can be represented as a regular grid in the Euclidean space. The Convolutional Neural Network (CNN) is capable of exploiting the shift invariance, local connectivity and complexity of image data, and as a result, CNN is able to extract local, meaningful features that are shared with all datasets for various image analysis [17].

DL has achieved great success when data can be represented as a vector in Euclidean space. When data are represented in non-Euclidean domain, some aspects of usual DL algorithm need to be adapted. An example of difference between Euclidean and non-Euclidean domains is shown on Fig. I.6.

The complexity of graph data has caused problems with existing machine learning algorithms. This is due to the fact that graph data is irregular. Each graph has a variable size of unordered nodes, and each node in a graph has a different number of neighbors, resulting in important operations that are easy to calculate in the image area, but are not directly applicable to the graph domain.

Wu and others proposed a new taxonomy of Graph Neural Networks (GNNs) [17]. In this taxonomy, GNNs are categorized into five groups:

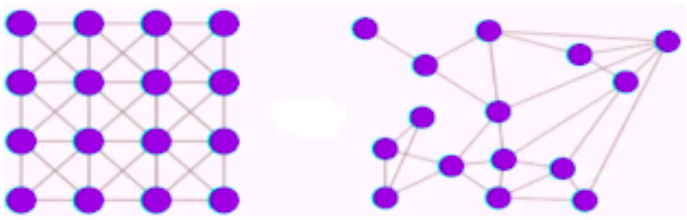

Figure I.6: 2D Euclidean domain vs. 2D Non-Euclidean domain. Left: Euclidean space. Right: Non-Euclidean space

1. Graph Convolution Networks (GCNs)
   GCNs generalize the operation of convolution from traditional data to graph data. The key is to learn a function f to generate a node v_i's representation by aggregating its own features X_i and neighbors features X_j, where $j \in N(v_i)$.

2. Graph Attention Networks (GATs)
   GATs are similar to GCNs and seek an aggregation function to fuse the neighboring nodes, random walks. The key difference is that GATs use attention mechanisms that assign greater weights to more important nodes, or walks.

3. Graph Auto-Encoders
   Graph Auto-encoders are unsupervised learning models which aim to learn low dimensional node vectors via an encoder, and then reconstruct the graph data via a decoder. Graph auto-encoders are a popular approach to the study of embedding of graphs, both for simple graphs without attribute information, and for graphs with features.

4. Graph Generative Networks (GGNs)
   GGN is designed to generate plausible structures from data. Generating graphs according to the empirical distribution is fundamentally difficult, mainly because graphs are complex data structures. To solve this problem, researchers have investigated to determine the generation process as a formation of nodes and edges alternatively to use generative adversarial training.

5. Graph Spatial-Temporal Networks
   Graph Spatial-temporal Networks aim to explore unseen patterns from spatial-temporal graphs. The basic idea of graph spatial-temporal networks is to take into account both the spatial dependence and the temporal dependence.

I.5 Objectives

The main goal of this project is to implement a DL solution for enabling statistical learning on the graphs of sulcal pits presented above, for which classical approaches are not adapted. To reach this main goal, the research plan consisted of the following objectives :

1. Preliminary processing of graphs.
   We need to define a label for each node, choose a Python package for manipulating of networks, and a format for saving graphs to disk.

2. Review of Deel Learning models on graphs.
   Theoretical study of methods to understand how they work.

3. Implementation of selected models.
   As a result, we must choose the most adapted methods to our task and implement them. Show results.

Part II

Objective 1: Preliminary processing of graphs

II.1 Dataset

134 right-handed healthy subjects were selected from the Open Access Series of Imaging Studies (OASIS) database. The OASIS cross-sectional dataset has a collection of 416 subjects aged from 18 to 96, including older adults with dementia. From this dataset, we defined two groups of 67 and 67 young adult healthy subjects (aged 18–34 years) matched in age, gender, cortical surface area and intracranial volume. For each subject, the sulcal pits and corresponding graphs representation were extracted as presented above.

More precisely, the graph of each subject is characterized by a matrix of adjacency, a vector of depth of nodes and a vector of coordinates of nodes. Fig. II.1 shows an example of 3D visualization of one of these graphs.

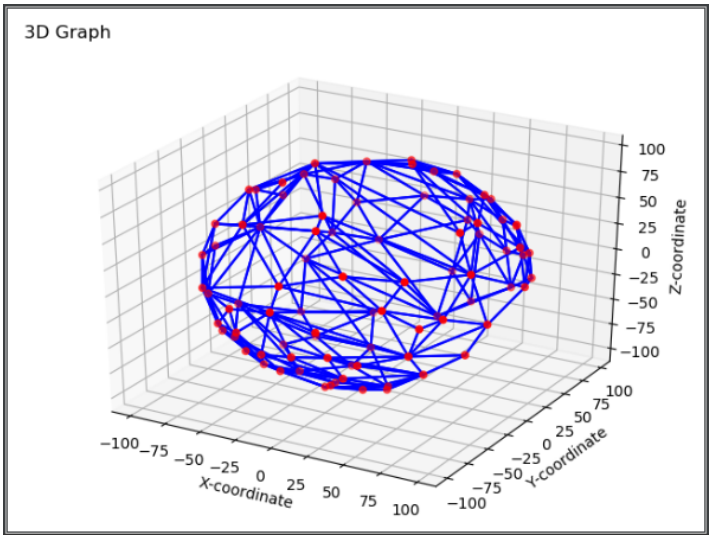

Figure II.1: 3D Graph from one individual of our dataset

In order to better understand the variation of these graphs we can calculate trivial statistics. The result is shown on Fig. II.2, Fig. II.3 and the trivial statistics in Tab. II.1, Tab. II.2.

We can assert that the graphs have different numbers of nodes. In our dataset, the smallest number of nodes is 77, the largest is 101. This makes the learning task more general, since it is necessary to use an algorithm that is applicable to a graph of any size.

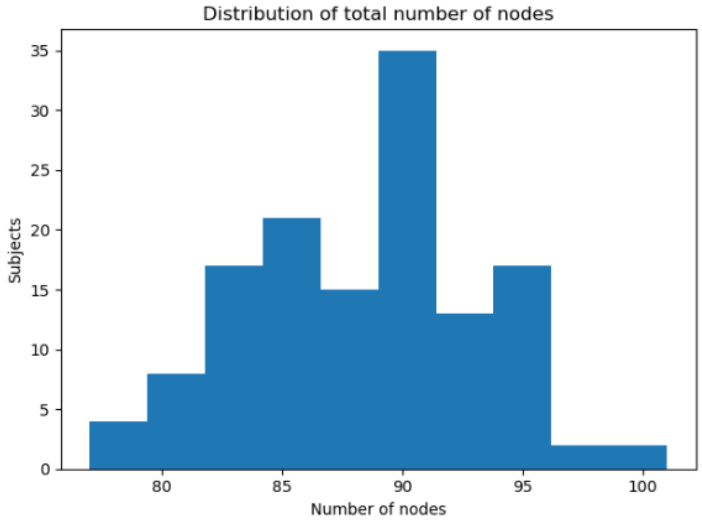

Figure II.2: Distribution of total number of nodes

Statistic	134 graphs
Min	77
Max	101
Mean	88.29
STD	4.76

Table II.1: Statistics of the number of nodes

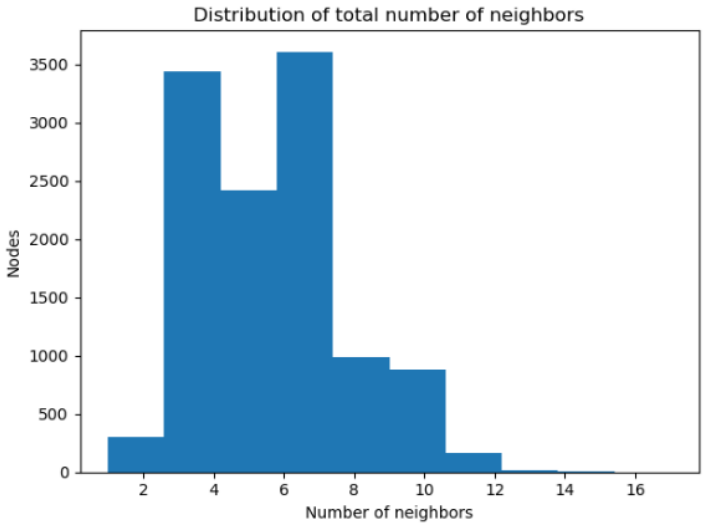

Figure II.3: Distribution of total number of neighbors

Statistic	134 graphs
Min	1
Max	17
Mean	5.64
STD	2.009

Table II.2: Statistics of the number of neighbors

Given the result, we can assert that our graphs are nonregular. A nonregular graph is a graph where each vertex has the different number of neighbors, i.e. every vertex has the different degree or valency.

II.2 Comparison of Python packages for manipulation of complex networks

The manipulation of our dataset of 134 graphs necessitate to set up appropriate tools in Python. We will compare three popular graph libraries available in Python: graph-tool, igraph and NetworkX.

- NetworkX is a pure-python implementation for creation, manipulation, and studying of structures, dynamics, and functions of complex networks.
- Graph-tool is an efficient Python module for manipulation and statistical analysis of graphs where algorithms are implemented in C++.
- igraph is a set of network analysis tools with a focus on efficiency, mobility and ease of use implemented in C.

To show a performance comparison between these packages we based on the research proposed by the authors of Graph-tool (Fig. II.4) [18]. In this research was used a directed graph, with N $= 39,796$ vertices and E $= 301,498$ edges.

Algorithm	graph-tool (4 cores)	graph-tool (1 core)	igraph	NetworkX
Single-source shortest path	0.004 s	0.004 s	0.012 s	0.152 s
PageRank	0.029 s	0.045 s	0.093 s	3.949 s
K-core	0.014 s	0.014 s	0.022 s	0.714 s
Minimum spanning tree	0.040 s	0.031 s	0.044 s	2.045 s
Betweenness	244.3 s (~4.1 mins)	601.2 s (~10 mins)	946.8 s (edge) + 353.9 s (vertex) (~ 21.6 mins)	32676.4 s (edge) 22650.4 s (vertex) (~15.4 hours)

Figure II.4: Comparison between the packages

NetworkX is slower than the others. This is largely due to the pure implementation of Python, which is generally much slower than C/C++. The performance of Graph-tool results in a saving of time and memory during compilation. If you are using an operating system for which no pre-compiled binary files are available, this is an additional load to be taken into account by the user [18]. NetworkX is comparatively inefficient, but it is trivial to install requiring no compilation at all, since it is pure python. The speed may not be a problem if one is dealing with very small graphs. Our graphs do not have a large number of nodes (about 80). We decided use the networkx package for easy manipulation of graphs, besides, with such size we will not lose much time to calculate.

II.3 Python formats to store the graph data

After we have finished analysis graphs, we need to save data in a special format for use those in implementation of DL models. There are many different file formats for graphs. The capabilities of these file formats range from a simple adjacency list to complex hierarchies that can store information on nodes and their relationships with each other. We will consider at several common formats of storing graphs, such as Pickle, GML, SparseGraph6, JSON.

- Pickle

The pickle module implements binary protocols for serializing and de-serializing a object structure. "Pickling" is the process whereby a object hierarchy is converted into a byte stream, and "unpickling" is the inverse operation, whereby a byte stream is converted back into an object hierarchy [19]. By default, the pickle data format uses a compact binary representation. The pickle module is more effective if you need optimal size characteristics, you can compress pickled data.

In Python several libraries can be used to write/read a Python object structure as pickle data. Note that NetworkX graphs can contain any hashable Python object as node. In that case using the Python pickle to store graph data can be used.

- GML

GML is a proposal for a portable file format for graphs [20]. The main features of GML are portability, simple syntax, extensibility and flexibility. A GML file consists of hierarchical lists of key-value. Graphs can be annotated with arbitrary data structures.

A GML file is a 7-bit ASCII file. This makes it simple to write files through standard routines. Parsers are easy to implement, either by hand or with standard tools. Also, files are text files, they can be exchanged between platforms without special converters.

Graphs can be represented by the keys graph, nodes and edges.

- SparseGraph6

graph6 and sparse6 are formats for storing undirected graphs in a compact manner, using ASCII characters [21]. Files in these formats have text type and contain one line per graph.

graph6 is suitable for small graphs, or large dense graphs. sparse6 is more space-efficient for large sparse graphs.

In these graph formats, each graph occupies one text line. You can manipulate them using any of your system tools for text files, such as sorting, concatenating, etc.

- JSON

JSON is a lightweight data-interchange format [22]. It is based on a subset of the JavaScript Programming Language. JSON is a text format that is completely language independent, but uses conventions that are familiar to programmers of the C-family of languages. These properties make JSON an ideal data-interchange language.

JSON is built on two structures: a collection of name/value pairs, and an ordered list of values. These are universal data structures. Virtually all modern programming languages support them in one form or another. It makes sense that a data format, which is interchangeable with programming languages, should also be based on these structures. Using Python JSON to generate and parse the NetworkX graph can be used.

Features	Pickle	GML	SparseGraph6	JSON
Encoding	Byte stream	7-bit ASCII	7-bit ASCII	UTF-8
Portability	is Python-specific	can be easily exchanged between platforms	can be easily exchanged between platforms	widely used outside of the Python ecosystem
Human-readable	No	Yes	Yes	Yes
Compatibility	Compatible	Compatible	Compatible	Compatible

Table II.3: Comparison of formats

Let us compare the selected formats. The following Tab. II.3 compares the features of these file formats.

If we estimate this comparison globally, then, first of all, we compare two classes: text serialization formats and a binary serialization format. If we have a large set of graphs, then text serialization formats are less effective, unlike the pickle format that encodes data as a byte stream. If we want to use data across different programming languages, pickle is not recommended. However, in our implementation we will use only Python, then the pickle module is still a good choice for its ease of use and ability to reconstruct complete Python objects. In addition, the user does not need to read the file, as this data will be used to train and test a DL model.

Based on the comparative table (Tab. II.3) and data storage judgment, it was chosen to use Pickle format.

II.4 Learning task

Our dataset is composed of 134 graphs of sulcal pits defined as $G = (P, A, D, X)$ where $D = \{d_i\} \in R^N$ is the vector of depth values, $X = \{X_i\} \in R^{Nx3}$ is the matrix of coordinates of all graph nodes, P is the number of nodes and A is the binary adjacency matrix $A = (a_{ij}) \in R^{NxN}$.

The learning task that we want to adress using DL is a simple binary node classification, where each sulcal pit in a graph can be labeled as "primary" or "secondary". Recent findings from the team suggest that these two different classes of sulcal pits co-exist in each individual brain. In [13], the link between the reproducibility of the sulcal pits across individuals and their depth was quantified, see Fig. II.5. On average, deeper sulcal pits showed a higher reproducibility than the shallower ones. However, some shallower pits were also highly reproducible across individuals.

The results from [23] suggested a dual dynamic of cortical growth featuring deep, early sulcal pits that are stable in number after birth and superficial, later pits that continue to appear during childhood, involving a more complex and singular gyrification. Overall, "primary" and "secondary" sulcal pits are supposed to be dissociable based on a combination of features:

1. "Primary" sulcal pits are deeper than "secondary" sulcal pits.
2. "Primary" sulcal pits are more stable across individuals than "secondary" sulcal pits.
3. "Primary" sulcal pits are located in major sulci that appear very early in life, during antenatal brain maturation, while "secondary" sulcal pits appear later, including after birth.

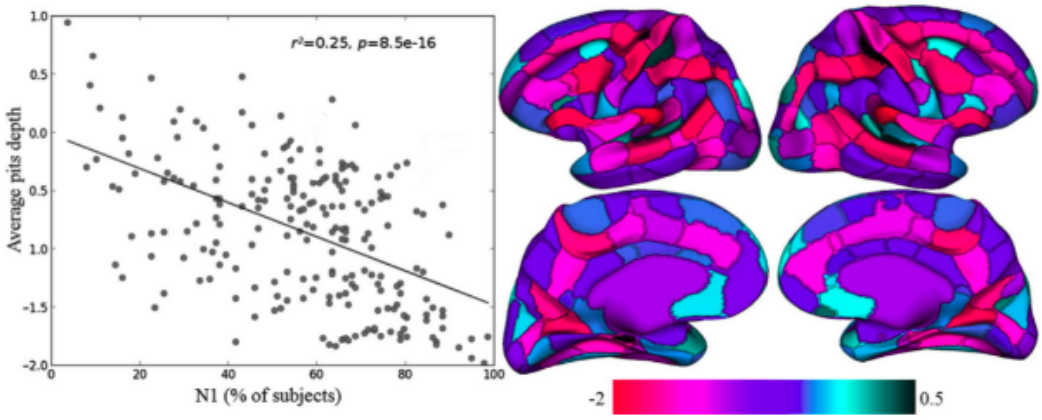

Figure II.5: Left: Scatter plot of the mean depth across all the group clusters and the regression line illustrating the linear relationship between these variables. Right: The map of the average pits depth in each cluster.

Beyond these concepts, the distinction between these two classes of sulcal pits has to be refined through implementation. In other words, we do not have at hand a labelled dataset that could serve as groung-truth for training in our learning framework. Our approach was to explore different ways to implement and learn the pits labelling, with the aim of better understanding the properties of the nodes and their relationships.

II.4.1 A trivial prediction model based on the depth

In this section, we will try to identify the node labels using their depth following [13], which means that we identify "primary" pits as deep and "secondary" as shallow ones.

First, we need to define a threshold depth value. In order to divide the set of nodes into two classes with approximately the same number of instances of the class, we define the threshold for the ith graph as follows

$$t_i = mean(D^{G_i}) \tag{II.1}$$

It should be noted that the threshold is selected for each of the graphs individually. The right panel of Fig. II.6 shows the distribution of the depth across all the nodes from our 134 graphs. Then, we can calculate the threshold distribution of all graphs to see the spread of values (on the left panel of Fig. II.6).

In order to determine the label for the jth node of the ith graph, we use the following system of inequalities

$$y_{ij} = \begin{cases} 0, & \text{if } D_i^{G_j} < t_i \\ 1, & \text{if } D_i^{G_j} \geq t_i \end{cases} \tag{II.2}$$

As a result, we get labeled data and we can estimate the number of representatives of each class in the dataset, as shown in Tab. II.4.

Thus, we have 50.3% of first-class nodes and 49.7% of second-class nodes.

We can calculate the depth distribution of the pits one subject to see the definition of the label from the depth (on Fig. II.7).

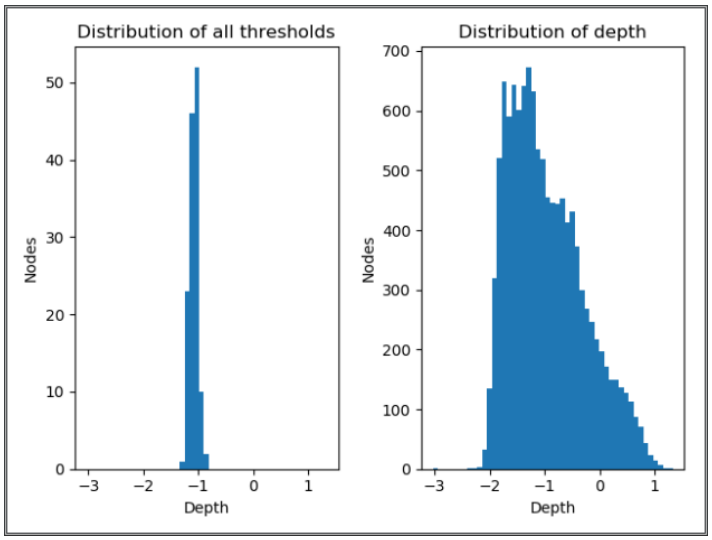

Figure II.6: Distribution of the pits depth

Number of graphs	Total number of nodes	Number of deep nodes	Number of shallow nodes
134	11832	5955	5877

Table II.4: Estimation the number of representatives of each class

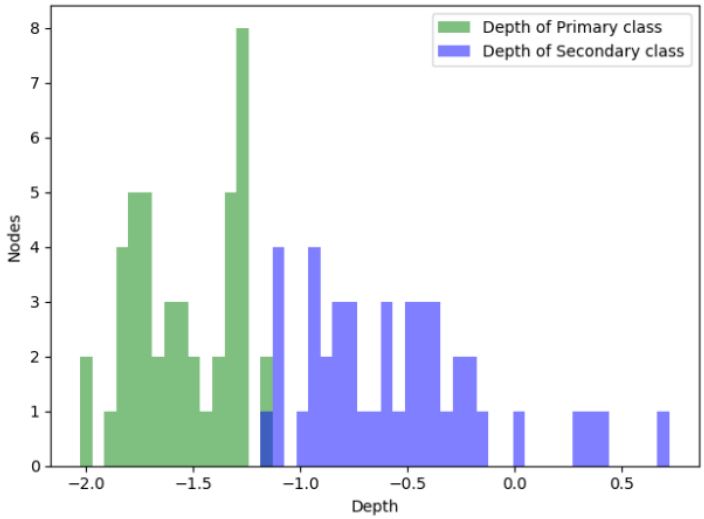

Figure II.7: Distribution of the depth of two classes from one subject

Once we have defined our "ground-truth" labelling, we can set up our learning framework. Before trying DL approaches, we first implement a trivial prediction model based on the simple average of depth threshold across the subjects included in the train set. We expect this trivial model to yield moderate accuracy, which would then be outperformed by DL models thanks to their ability to exploit relational information across the nodes.

We construct the trivial prediction model of the node class based on the depth as follows: We have a set of T graphs $G = \{G_t | t \in 1...T\}$. Each graph $G_t = (V_t, E_t)$ is composed of vertices V_t and edges E_t. The vertices are collectively described by an $N_t \times F$ design matrix X_t of features, where N_t is the number of nodes in G_t and F is the number of features. The

nodes have labels Y associated with them.

We divide our data into two datasets: a training dataset and a test dataset, which are used for different purposes.

The training dataset $\{G_1, G_2, \ldots, G_k\}$ is the initial dataset used to train the model. It includes both input data and the corresponding labelling.

For each graph, we define an individual threshold t_i. In order to determine the threshold of the model t, take the average value of all the individual thresholds of the graphs in the train set as

$$t = \frac{\sum_{i=1}^{k} t_i}{k} \tag{II.3}$$

The test dataset $\{G_{k+1}, G_{k+2}, \ldots, G_T\}$ is used to assess how well your algorithm was trained with the training dataset. To predict the class, we use the following system

$$\forall i \in \{G_{k+1}, G_{k+2}, \ldots, G_T\}, \forall j \in G_i$$
$$y_{ij} = \begin{cases} 0, & \text{if } D_i^{G_j} < t \\ 1, & \text{if } D_i^{G_j} \geq t \end{cases} \tag{II.4}$$

Then, we can estimate the accuracy that has the following definition

$$\text{Accuracy} = \frac{\text{Number of correct predictions}}{\text{Total number of predictions}} \tag{II.5}$$

When we applied this trivial model to our data, we got an accuracy of 96%, which is much higher than we expected. This accuracy is so high that a putative DL model would have very limited possibility to outperform the trivial model, which would make this experiment of little interest.

We thus decided to change the labelling procedure.

II.4.2 Determining of the labels by the coordinates

The Fig. II.8 from [13] shows the individual sulcal pits extracted for the cortical surfaces with the lowest (left) and largest (right) number of pits across the 134 hemispheres. The red arrows indicate places where the presence of sulcal pits may be desirable.

The idea of this approach is to distinguish "primary" and "secondary" sulcal pits based on their coordinates (spatial location) instead of their depth. The "primary" pits will be those located in deep large folds as defined from a standard anatomical nomenclature [24]. Fig. II.9 shows the two regions corresponding to "primary" (red) and "secondary" (green) folds. In this way, we lower the dependency between the definition of the label and the depth since some shallow sulcal pits are also present in the region in red.

As a result, we get labeled data and we can estimate the number of representatives of each class in the dataset, as shown in Tab. II.5.

Thus, we have 43.8% of first-class nodes and 56.2% of second-class nodes.

We can calculate the depth distribution of the pits one subject to see the independence of the definition of the label from the depth (on Fig. II.10).

Let's try to evaluate this method of determining the label with the help of the proposed trivial model. The accuracy of the trivial model showed the result of 56%.

Consequently, this approach to determining the label is partially independent of the value of depth. Now, we can try to apply a DL models to get better accuracy.

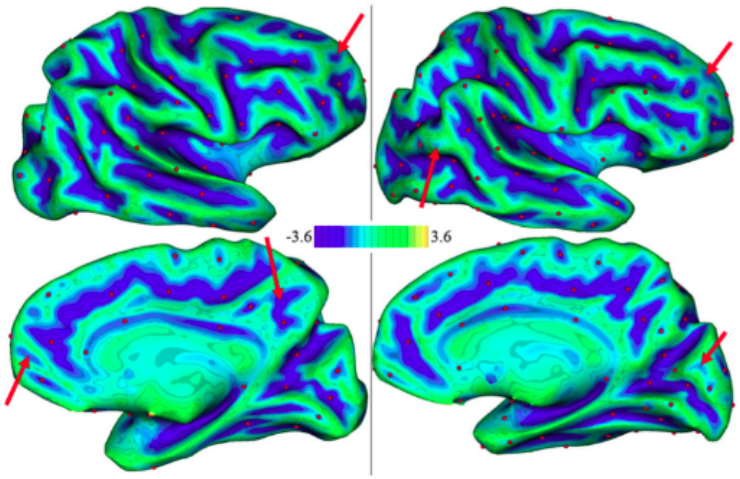

Figure II.8: The sulcal pits (in red) for the cortical surfaces with the lowest and largest number of pits across the entire group of individuals. Note that the location of most of the sulcal pits is almost the same. Arrows are pointing to missing pits

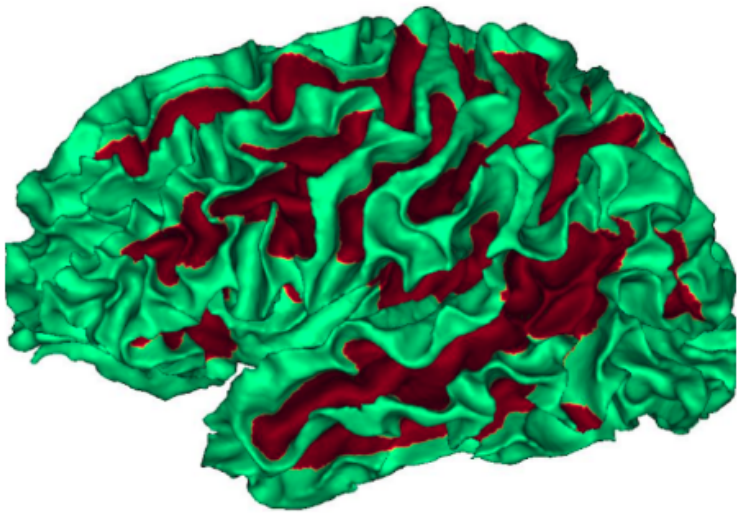

Figure II.9: Extraction of the deepest brain sulcuses (in red)

Number of graphs	Total number of nodes	Number of deep nodes	Number of shallow nodes
134	11832	5192	6640

Table II.5: Estimation the number of representatives of each class

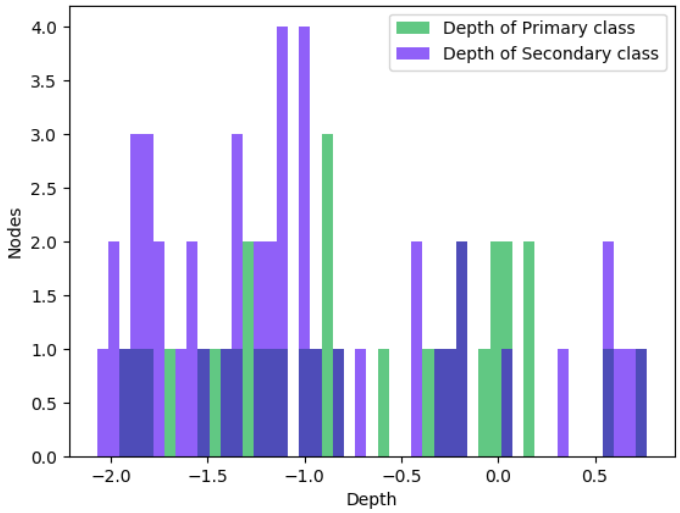

Figure II.10: Distribution of the depth of two classes from one subject

Part III

Objective 2: Review of Deep learning models on graphs

In [17], the authors presented a complete overview of Neural Networks graphs (GNN) in the areas of data mining and machine learning. GNN methods are divided into five groups: Graph Convolution Networks, Graph Attention Networks, Graph Auto-Encoders, Graph Generative Networks, and Graph Spatial-Temporal Networks.

Due to the topology of graphs and nodes information, GCN outputs can focus on different tasks:

- Classification of nodes.
- Classification of arcs and forecasting of connections.
- Classification of graphs.

Since we are interested in the classification of nodes, we will consider in more detail methods which can be applied to solve this problem.

III.1 Overview of models

III.1.1 Semi-Supervised Classification with Graph Convolutional Networks (1stChebNet)

This architecture is represented by the localized approximation of the first-order spectral graph [25]. With the help of the spectral theory of a graph, we can define convolution and pooling on non-Euclidean space. Then, with the help of Chebyshev's polynomials we can approximate expressions.

As a result, we can define a multi-layer Graph Convolutional Network (GCN) with the following layer-wise propagation rule

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

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

Each row of the output is a hidden representation of each node obtained by a linear transformation of aggregated information from the node itself and its neighboring nodes with weights given by the row of A.

On examples of this simple model works so much better than several baselines that is surprising because the model used is just a two-layer model in most experiments, that is very local, and the output of a node can only be affected by nodes in a 2-hop neighborhood.

This method is easy in implementation. For using this method we have the necessary input data, but this model does not show successful results with graphs of different sizes. Another disadvantage is that it has to load the entire graph into memory to perform a convolution, which is not effective in processing large graphs.

III.1.2 The Gaph Nural Network model (GNN)

This model combines two existing models. GNN is an extension of Recursive Neural Networks (RNNs) and random walk models, and it retains the characteristics of both models [8].

A feature that GNN extends RNNs is the ability to process a more general class of graphs.

The main idea of recurrent-based methods is to recursively update the latent representation of a node until a stable fixed point is reached. GNNs are based on an information diffusion mechanism. A graph is processed by a set of units, each of which corresponds to a node graph that is linked according to the graph connectivity. The units update their states and exchange information until they reach a stable equilibrium. The output of a GNN is then calculated locally on each node based on the unit state.

This GNN model can process different types of graphs, such as acyclic, cyclic, directed and undirected.

Authors assume a supervised learning framework with the learning set

$$L = \{(G_i, n_{ij}, t_{ij}) | G_i = (N_i, E_i), n_{ij} \in N_i, t_{ij} \in R^m, 1 \leq i \leq p, 1 \leq j \leq q_i)\} \tag{III.2}$$

where t_{ij} is the desired target associated to n_{ij}.

Then we can attach a state $x_n \in R^m$ to each node n that is based on the information contained in the neighborhood of n (Fig. III.1) [8].

The authors defined x_n and output o_n as

$$\begin{cases} x_n = f_w(l_n, l_{\propto[n]}, x_{ne[n]}, l_{ne[n]}) \\ o_n = g_w(x_n, l_n) \end{cases} \tag{III.3}$$

where f_w is local transition function; g_w is local output function; $l_n, l_{\propto[n]}, x_{ne[n]}, l_{ne[n]}$ are the label of n, the labels of its edges, the states, and the labels of the nodes in the neighborhood of n.

To implement the model it is necessary to provide the following items 3 steps:

1. a method to solve previous system;
2. a learning algorithm to adapt f_w, g_w and using examples from the training dataset;
3. an implementation of f_w and g_w.

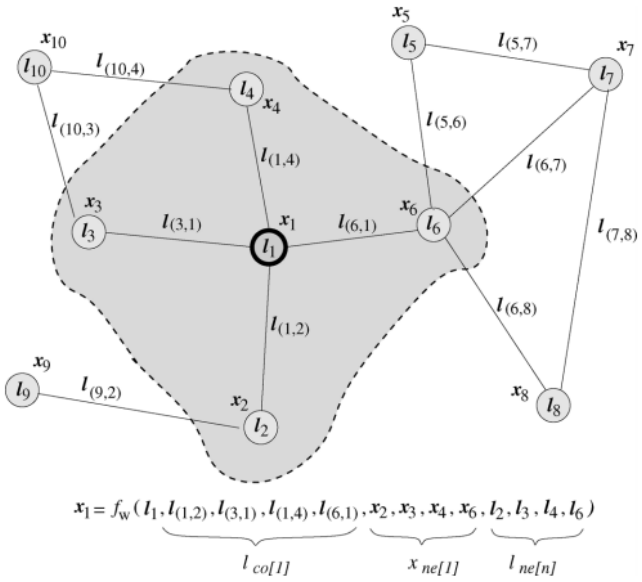

Figure III.1: Graph and the neighborhood of a node

Banach's fixed point theorem allows to iteratively calculate the previous system, then it will as

$$\begin{cases} x_n^{(t+1)} = f_w(l_n, l_{\propto[n]}, x_{ne[n]}^{(t)}, l_{ne[n]}) \\ o_n^{(t)} = g_w(x_n^{(t)}, l_n) \end{cases} \tag{III.4}$$

The training is carried out using the Almeida-Pineda algorithm, which works by running propagation to convergence and then computing gradients based on the convergent solution.

This model has an advantage of not needing to store intermediate states in order to compute gradients. The model extends RNNs since it can process a more general class of graphs including cyclic, directed, and undirected graphs, and it can deal with node without any pre-processing steps.

III.1.3 Gated Graph Sequence Neural Networks (GGNNs)

This model is based on the previous Graph Neural Network, which the authors modify to use gated recurrent units and modern optimization methods. The biggest modification of GNNs is that they use gated recurrent units and backpropagation through time to calculate gradients [26].

GGNNs allows us to include node labels as additional input. To distinguish these node labels, which are used as input data previously entered, the authors call them node annotations and use the x vector to denote these annotations.

One example of the task of this model is predict "Can we go from 1 to 4?" (see on Fig. III.2).

There are two problem-related special nodes, 1 and 4. To mark these nodes as special, we give them an initial annotation. The first node 1 gets the annotation $x_1 = [1, 0]^T$, and the second node 4 gets the annotation $x_4 = [0, 1]^T$. All other nodes v have their initial annotation set to $x_v = [0, 0]^T$.

The basic recurrence of the propagation model is

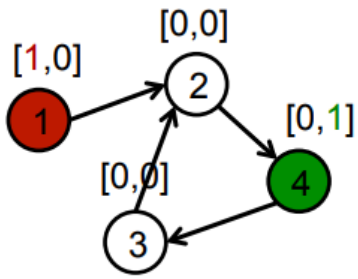

Figure III.2: One example of the model: "Can we go from 1 to 4?"

$$h_v^{(1)} = [x_v^T, 0]^T \tag{III.5}$$

$$a_v^{(t)} = A_v^{(1)}[h_1^{(t-1)T} ... h_{|V|}^{(t-1)T}]^T + b \tag{III.6}$$

$$z_v^t = \sigma(W^x a_v^{(t)} + U^x h_v^{(t-1)}) \tag{III.7}$$

$$r_v^t = \sigma(W^r a_v^{(t)} + U^r h_v^{(t-1)}) \tag{III.8}$$

$$\widetilde{h}_v^{(t)} = tanh(W a_v^{(t)} + U(r_v^t \odot h_v^{(t-1)})) \tag{III.9}$$

$$h_v^{(t)} = (1 - z_v^t) \odot h_v^{(t-1)} + z_v^t \odot \widetilde{h}_v^{(t)} \tag{III.10}$$

GGNNs support node selection tasks by making $o_n(t) = g(h_v^{(t)}, x_v)$ for each node v output node scores and applying a softmax over node scores.

Gated Graph Sequence Neural Networks (GGS NNs) is several GGNNs operate in sequence to produce an output sequence $o^{(1)} ... o^{(K)}$.

One example of the task of this model is predict "Shortest path from A to B" (see on Fig. III.3).

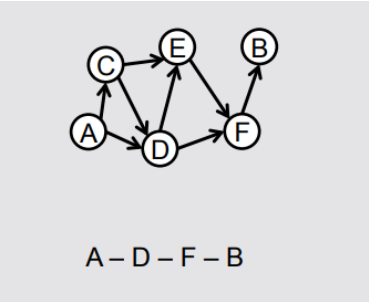

Figure III.3: One example of the model: "Shortest path from A to B"

This model shows a good result with tasks that test basic forms of reasoning such as deduction, induction, pathfinding. Each entity is mapped to a node, and each relationship is mapped to an edge with an edge label given by the relation. Each step of the prediction produces an ouput and creates new annotations of the nodes for the next step.

In the task that was observing in the paper, GGNNs showed better result than RNNs/LSTMs. This shows that using exploiting structures in tasks can make things much easier. This model is not adapted for large graphs, because GGNNs needs to run the recursive function multiple times over all nodes, and intermediate states of all nodes need to be stored.

Thus, this model is more adapted to logical tasks and requires outputting logical formulas, which we formulate as a sequential output problem.

III.1.4 Learning Steady-States of Iterative Algorithms over Graphs (SSE)

Authors proposed an embedding representation for iterative algorithms over graphs, and have developed a learning method which alternates between updating the embeddings and projecting them onto the steady state constraints [27]. Each node in the graph support an embedding vector, and these embedding vectors is updated using a parameterized operator τ_θ, where the parameters θ is learned.

The training dataset consists of the input graph $G = (V, E)$, and the output of the algorithm for a subset of nodes

$$D = \left\{ f_v^* := f(h_v^*) | h_v^* = \tau[\{h_u^*\}_{u \in N(v)}], v \in V \right\} \tag{III.11}$$

The goal is to learn a parameterized algorithm A_θ, such that the output of the algorithm can mimic the output of the original algorithm τ.

The problem of learning the algorithm can be formulated in the following optimization problem

$$\begin{aligned} min_\theta \sum_{v \in V} l(f_v *, \widehat{f}_v) \\ \left\{ \widehat{f}_v \right\}_{v \in V} = A_\theta[G] \end{aligned} \tag{III.12}$$

This approach associates each node in the graph with an unknown vector embedding representation $\widehat{h}_v$, and the core of our algorithm is a parameterized operator, τ_θ, for enforcing steady-state relations between these embeddings. Given a link function $\widehat{f}(h_v)$, this model makes predictions on the output of the algorithm as follows

$$\begin{aligned} output : \left\{ \widehat{f}_v := \widehat{f}(\widehat{h}_v) \right\}_{v \in V} \\ \widehat{h}_v = \tau_\theta[\left\{ \widehat{h}_u \right\}_{u \in N(v)}] \end{aligned} \tag{III.13}$$

The τ_θ operator provides a steady state of the anchorages based on information about the local neighborhood of 1-hop.

For the g prediction function, it takes the node embeddings as input and predicts the corresponding algorithm outputs

$$g(\widehat{h}_v) = \sigma(V_2^T ReLU(V_1^T h_v)) \tag{III.14}$$

To ensure convergence to steady states, the recurrent function of SSE is defined as a weighted average of the historical states and new states.

Thus, SSE recursively evaluates the hidden view of the nodes and updates the parameters using sampled batch data.

III.1.5 Neural Message Passing for Quantum Chemistry (MPNN)

This model learn a message passing algorithm and aggregation procedure to compute a function of input graph. Authors reformulate dexisting models into a single common framework called Message Passing Neural Networks (MPNNs) [28].

A generalization of the convolution operator to irregular domains is usually expressed as a neighborhood aggregation or message passing scheme. With $x_i^{(k-1)} \in R^F$ denoting node features of node i in layer $(k-1)$ and $e_{ij} \in R^D$ denoting edge features from node i to node j, message passing graph neural networks can be described as

$$x_i^{(k)} = \gamma^{(k)}(x_i^{(k-1)}, \tau_{j \in N(i)} \phi^{(k)}(x_i^{(k-1)}, x_j^{(k-1)}, e_{ij})) \tag{III.15}$$

where $\tau_{j \in N(i)}$ denotes a function, such as sum, mean or max, and γ and ϕ denote differentiable functions, such as MLPs (Multi Layer Perceptrons).

The authors presented that several other GCNs fall into their framework by assuming different forms of γ and ϕ:

- Gated Graph Neural Networks (GGNN);
- Interaction Networks;
- Molecular Graph Convolutions;
- Deep Tensor Neural Networks;
- Laplacian Based Method.

MPNNs outperforms several powerful baselines, and this approach eliminates the need for complex function engineering.

III.1.6 Inductive Representation Learning on Large Graphs (GraphSage)

GraphSAGE is an inductive model that uses information about how to efficiently create embedded nodes for previously invisible data [29]. The algorithm can be applied to graphs without node features.

Instead of training a separate embedding vector for each node, this algorithm trains a set of aggregator functions that learn how to aggregate information about an object from node's local neighborhood (see on Fig. III.4). Each aggregator function aggregates information from a different number of hops or search depth that is remote from the node.

The graph convolution operation is defined as

$$h_v^t = \sigma(W^t \times aggregate_t(h_v^{t-1}, \left\{h_u^{t-1}, \forall u \in N(v)\right\})) \tag{III.16}$$

GraphSage learning process consists of three steps. First, it selects a node's local k-hop neighborhood with a fixed-size. Second, it derives the central node's final state by aggregating its neighbors feature information. Finally, it uses the central node's final state to predict and propagate errors.

GraphSage is intended for use on large graphs (>100,000) nodes. The overhead of sub-sampling will begin to outweigh its benefits on smaller graphs.

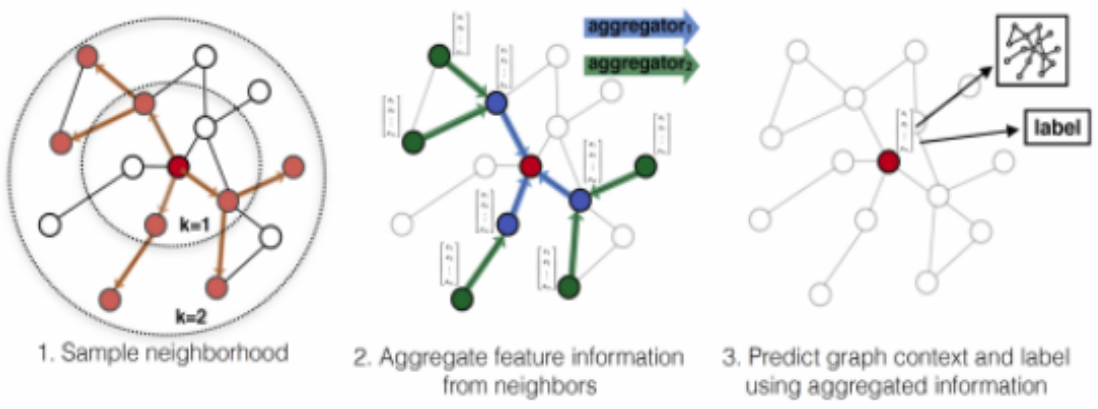

Figure III.4: GraphSAGE sample with aggregate approach

III.1.7 Diffusion-Convolutional Neural Networks (DCNN)

In this paper, the authors had used a diffusion-convolution operation, and showed how diffusion-based representations can be learned from graph structured data and used to efficiently classify nodes [30].

If we have a set of T graphs $G = \{G_t | t \in 1 \dots T\}$. Each graph $G_t = (V_t, E_t)$ is composed of vertices V_t and edges E_t. The vertices are collectively described by an $N_t \times F$ design matrix X_t of features, where N_t is the number of nodes in G_t, and the edges E_t are encoded by an $N_t \times N_t$ adjacency matrix A_t, from which we can compute a degree normalized transition matrix P_t that gives the probability of jumping from node i to node j in one step. No constraints are placed on the form G_t; the graph can be weighted or unweighted, directed or undirected. Either the nodes, edges, or graphs have labels Y associated with them.

DCNN takes set G as input and returns either a hard prediction for Y or a conditional distribution $P(Y|X)$. Each entity of interest is transformed to a diffusion-convolutional representation, which is a $H \times F$ real matrix defined by H hops of graph diffusion over F features, and it is defined by an $H \times F$ real-valued weight tensor W^c and a nonlinear differentiable function f that calculates the activations.

For node classification tasks the diffusion-convolutional representation of graph t, Z_t, will be a $N_t \times H \times F$ tensor, as illustrated on Fig. III.5. For graph or edge classification tasks, Z_t will be a $H \times F$ matrix or a $M_t \times H \times F$ tensor respectively, as illustrated on Fig. III.5.

Since we are interested in classification of nodes, we will consider this problem in more detail.

We have a node classification task where a label Y is predicted for each input node in a graph. If we let P_t^* be an $N_t \times H \times N_t$ tensor containing the power series of P_t, the diffusion-convolutional activation Z_{tijk} for node i, hop j, and feature k of graph t is given by

$$Z_{tijk} = f(W_{jk}^c \sum_{l=1}^{N_t} P_{tijl}^* X_{tlk}) \tag{III.17}$$

The activations can be expressed more concisely using tensor notation as

$$Z_t = f(W^c \odot P *_t X_t) \tag{III.18}$$

where the $\odot$ operator represents element-wise multiplication.

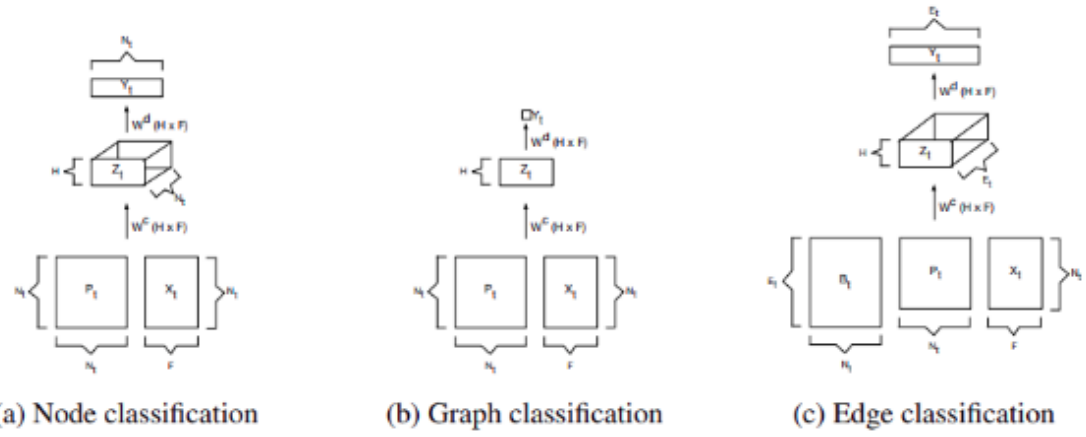

Figure III.5: DCNN model definition for node and graph classification tasks

The model is completed by a dense layer that connects Z to Y. A prediction for Y, denoted $\widehat{Y}$, can be obtained by taking the maximum activation and a conditional probability distribution $P(Y|X)$ can be found by applying the softmax function

$$\begin{aligned} \widehat{Y} &= argmax(f(W^d \odot Z)) \\ P(Y|X) &= softmax(f(W^d \odot Z)) \end{aligned} \tag{III.19}$$

The model only entails $O(H \times F)$ parameters that make the size of the latent diffusion-convolutional representation independent of the size of the input. Computing very large graphs can lead to out-of-memory errors on the GPU. DCNNs can be easily applied to graphs from tens to hundreds of thousands of nodes, but not to graphs with millions to billions of nodes.

III.1.8 Large-Scale Learnable Graph Convolutional Networks (LGCN)

In CNNs, the number of neighboring units in graphs is not fixed, which prevents the use of convolution operations. Gao and other authors solve these problems by proposing a learnable graph convolutional layer (LGCL) [31]. LGCL automatically selects a fixed number of neighboring nodes for each feature based on ranking of values.

The layer-wise propagation rule of LGCL is formulated as

$$\begin{aligned} \widetilde{X_l} &= g(X_l, A, k) \\ X_{(l+1)} &= c(\widetilde{X_l}) \end{aligned} \tag{III.20}$$

where A is the adjacency matrix, $g(-)$ is an operation that performs the k-largest node selection to transform graphs to data of grid-structures, and $c(-)$ denotes a regular 1-D CNN that aggregates neighboring information and outputs a new feature vector for each node.

Suppose $X_l \in R^{N \times C}$ with row vectors $x_l^1, x_l^2, \ldots, x_l^N$, representing a graph of N nodes, where each node has C features. We are given the adjacency matrix $A \in R^{N \times N}$ and a fixed k. Consider i is a specific node, whose feature vector is x_l^i and it has n neighboring nodes. Then, we can obtain the indices of these adjacent nodes based on a simple look-up operation in A, say $i_1, i_2, \ldots, i_n$. Concatenating the feature vectors $x_l^{i_1}, x_l^{i_2}, \ldots, x_l^{i_n}$ outputs a matrix $M_l^i \in R^{n \times C}$. If $n < k$, we can pad M_l^i using columns of zeros.

In LGCNs, we first apply a layer of embedding graphs to obtain low-dimensional representations of nodes. The graph embedding layer is essentially a linear transformation in the first layer expressed as

$$X_1 = X_0 W_0 \tag{III.21}$$

where $X_0 \in R^{N \times C_0}$ represents the high-dimensional input and $W_0 \in R^{C_0 \times C_1}$ changes the dimension of feature space from C_0 to C_1. As a result, $X_1 \in R^{N \times C_1}$ and $C_1 < C_0$.

After the graph embedding layer, we stack multiple LGCLs, depending on the complexity of the data graph. To improve model performance and facilitate learning, we can use skipped connections to combine LGCL inputs and outputs. Finally, the fully-connected layer is used before the softmax function for final predictions.

Consequently, this transformation is done using through a novel k-largest node selection process, which uses the a ranking between node feature values.

III.1.9 Graph Attention Networks (GAT)

One of the advantages of attention mechanisms is that they allow to work with variable-size input data, focusing on the most important parts of the the input for decision-making.

The authors introduced an attention-based architecture to perform a classification of the nodes of graph structured data [32]. The idea is to compute the hidden representations of each node in the graph, by attending over its neighbors, following a self-attention strategy.

The input to attentional layer is a set of node features $h = \left\{ \overrightarrow{h}_1, \overrightarrow{h}_2, \ldots, \overrightarrow{h}_N \right\} \overrightarrow{h}_i \in F'$, where N is the number of nodes, and F is the number of features in each node. The layer produces a new set of node features $h' = \left\{ \overrightarrow{h}'_1, \overrightarrow{h}'_2, \ldots, \overrightarrow{h}'_N \right\}$, $\overrightarrow{h}'_i \in R^{F'}$, as its output.

We then perform self-attention on the nodes, a shared attentional mechanism $a : R^{F'} \times R^{F'} \rightarrow R$ computes attention coefficients with a weight matrix, $W \in R^{F' \times F'}$

$$e_{ij} = a(W\overrightarrow{h}_i, W\overrightarrow{h}_j) \tag{III.22}$$

that indicate the importance of node js features to node i.

The attention mechanism a is a single-layer feedforward neural network, parametrized by a weight vector $\overrightarrow{a} \in R^{2F'}$, and applying the $LeakyReLU$ nonlinearity

$$a_{ij} = \frac{exp(LeakyReLU(\overrightarrow{a}^T[W\overrightarrow{h}_i || W\overrightarrow{h}_j]))}{\sum_{k \in N_i} exp(LeakyReLU(\overrightarrow{a}^T[W\overrightarrow{h}_i || W\overrightarrow{h}_k]))} \tag{III.23}$$

The obtained normalized attention coefficients are used to calculate the linear combination of the corresponding features to serve as the final output features for each node

$$\overrightarrow{h}'_i = \sigma(\sum_{j \in N_i} a_{ij} W \overrightarrow{h}_j) \tag{III.24}$$

To stabilize the learning process of self-attention, the authors stressed that extending this approach to use multi-headed attention would be useful. K independent attention mechanisms perform a preliminary transformation, and then their features are concatenated, resulting in the following output feature representation

$$\overrightarrow{h}'_i = ||_{k=1}^{K} \sigma(\sum_{j \in N_i} a_{ij}^k W^k \overrightarrow{h}_j) \tag{III.25}$$

where || represents concatenation.

The authors have shown by examples that the graph attentional layer, which is used in these networks is computationally efficient.

III.1.10 GaAN: Gated Attention Networks for Learning on Large and Spatiotemporal Graphs (GAAN)

Unlike the traditional multi-head attention mechanism, GaAN uses a convolutional sub-network to control the importance of each head of attention [33].

Given a node i and its neighboring nodes N_i, a graph aggregator is a function γ in the form of $y_i = \gamma_\theta(x_i, \{z_{N_i}\})$, where x_i and y_i are the input and output vectors of the center node i. $z_{N_i} = \{z_j | j \in N_i\}$ is the set of the reference vectors in the neighboring nodes and θ is the learnable parameters of the aggregator.

The detailed formulation of the multi-head attention aggregator is as follows

$$y_i = FC_{\theta_o}(x_i \oplus ||_{k=1}^{K} \sum_{j \in N_i} w_{ij}^{(k)} FC_{\theta_k^{(k)}}^{h}(z_j)) \tag{III.26}$$

$$w_{ij}^{(k)} = \frac{exp(\phi_w^{(k)}(x_i, z_j))}{\sum_{l=1}^{|N_i|} exp(\phi_w^{(k)}(x_i, z_l))} \tag{III.27}$$

$$\phi_w^{(k)}(x, z) = \langle FC_{\theta_k^{(k)}}^{h}(x) | FC_{\theta_k^{(k)}}^{h}(z) \rangle \tag{III.28}$$

Here, K is the number of attention heads. $w_{ij}^{(k)}$ is the kth attentional weights between the center node i and the neighboring node j, which is generated by applying a softmax to the dot product values. $\theta_{xa}^{(k)}$, $\theta_{za}^{(k)}$ and $\theta_v^{(k)}$ are the parameters of the kth head for computing the query, key and value vectors.

The difference between this aggregator and that in GAT is that they taken the attention mechanism of the key value and the dot product attention while GAT does not compute additional value vectors and uses a fully-connected layer to calcule $\phi_w^{(k)}$.

We calculate an additional soft gate between 0 (low importance) and 1 (high importance) to assign a different importance to each head. In combination with the multi-head attention aggregator, we obtain the formulation of the gated attention aggregator

$$\begin{aligned} y_i = FC_{\theta_o}(x_i \oplus ||_{k=1}^{K}(g_i^{(k)} \sum_{j \in N_i} w_{ij}^{(k)} FC_{\theta_k^{(k)}}^{h}(z_j))), \\ g_i = [g_i^{(1)}, ..., g_i^{(K)}] = \phi_g(x_i, z_{N_i}) \end{aligned} \tag{III.29}$$

where $g_i^{(k)}$ is a scalar, the gate value of the kth head at node i.

The authors combine average pooling and max pooling to construct a network that takes a central node and neighboring node features to create the gate values

$$g_i = FC_{\theta_g}^{o}(x_i \oplus max_{j \in N_i}(\{FC_{\theta_m}(z_j)\}) \oplus \frac{\sum_{j \in N_i} z_j}{|N_i|}) \tag{III.30}$$

The authors have shown by examples that the GaAN beats previous state-of-theart algorithms in inductive node classification and traffic speed forecasting

III.2 Summary

After reviewing the existing graph processing methods, we decided that the following models are the most adapted to our problem:

- GNN;
- DCNN;
- GaAN;
- GraphSage;
- GAT.

In Tab. III.1 is shown a brief description of each selected model. The details for each method are given in Part III.

Method	Description
GNN	GNNs recursively update node latent representations until convergence that means each node exchanges information with its neighbors until equilibrium is reached. GNN can process most of the practically useful types of graphs, such as acyclic, cyclic, directed, and undirected.
DCNN	DCNN proposed a graph convolution network which encapsulates a graph diffusion process. This method based on a degree-normalized transition matrix P that gives the probability of jumping from node i to node j in one step. The diffusion-convolutional activations of two isomorphic input graphs will be the same, because DCNN parameters are tied according to search depth rather than their position.
GaAN	GaAN uses the multi-head attention mechanism in updating a node's hidden state. GAAN introduces a self-attention mechanism which computes a different weight for each head, rather than assigning equal weight to each head.
GraphSage	GraphSAGE is an inductive model that leverages node attribut information to efficiently generate representations on previously invisible data. We learn a function that generates embeddings by sampling and aggregating features from a node's local neighborhood. GraphSAGE effectively trades off performance and runtime by sampling node neighborhoods.
GAT	The idea is to compute a hidden representations of each node in by attending over its neighbors, following a self-attention strategy. One of the advantages of attention mechanisms is that they allow for dealing with variable sized inputs, focusing on the most relevant parts.

Table III.1: Selected models

Part IV

Objective 3: Implementation & Result

IV.1 Implementation of models

For implementing DL methods, which are the most adapted to our problem we chose the following methods:

- DCNN;
- GraphSage;
- GAT.

A major reason why we chose these methods among others is the ease of implementation. In addition, each of these methods uses different technology that is interesting to compare: GraphSage learn a function that generates embeddings by sampling and aggregating features from a node's local neighborhood; DCNN performes a mapping from nodes and their features to the results of a diffusion process that uses parameters are tied to search depth rather than their position; GAT compute the hidden representations of each node by attending over its neighbors, following a self-attention strategy. We will show the results of several GraphSage implementation, each using a different aggregation function (sum, mean, max).

As described in section II.4.2, the learning task is to classify between "primary" and "secondary" nodes of each graph, given the "groud truth" labelling based on their localization inside or outside the red region on Fig. II.9.

We report results on a 6-fold cross-validation experiment on the OASIS datasets using all labels. The optimal parameters were selected for the models:

1. GraphSage
   For training, the Adam optimizer with a learning rate of 0.01, the Stochastic Gradient Descent (batch size is 1), 1 hidden layer, 16 hidden nodes, 50 epochs, L2 Regularization are used.

2. DCNN
   For training, the Adam optimizer with a learning rate of 0.05, the Stochastic Gradient Descent (batch size is 1), 11 hops, 30 epochs, L2 Regularization are used.

3. GAT
   For training, the Adam optimizer with a learning rate of 0.01, 2 hidden layers, 16 hidden nodes, the same dropout rates – 5% dropout for all hidden units, the Stochastic Gradient Descent (batch size is 1), 30 epochs, 15 multi-head-attentions for all hidden units, L2 Regularization are used.

Features	Method	Accuracy
adjacency matrix	GAT	0.56
	GraphSage(mean)	0.56
	GraphSage(sum)	**0.63**
	GraphSage(max)	0.56
	DCNN	0.5
adjacency matrix of graph depth of nodes	GAT	0.61
	GraphSage(mean)	0.61
	GraphSage(sum)	**0.65**
	GraphSage(max)	**0.63**
	DCNN	0.5
adjacency matrix of graph coordinates of nodes	**GAT**	**0.76**
	GraphSage(mean)	0.71
	GraphSage(sum)	**0.77**
	GraphSage(max)	0.74
	DCNN	0.51
adjacency matrix of graph depth of nodes coordinates of nodes	**GAT**	**0.77**
	GraphSage(mean)	0.73
	GraphSage(sum)	**0.77**
	GraphSage(max)	0.73
	DCNN	0.52

Table IV.1: Node classification

For each DL method, we consider 4 learning cases with different input data:

1. adjacency matrix of a graph;
2. adjacency matrix of a graph + depth of nodes;
3. adjacency matrix of a graph + coordinates of nodes;
4. adjacency matrix of a graph + depth of nodes + coordinates of nodes;

which will allow us to interpret the potential higher performence with respect to the added value of each type of feature.

The implementation was performed in Python using PyCharm Professional Edition [34]. DCNN was implemented in PyTorch [35]. GraphSage and GAT were implemented in PyTorch-Geometric that is a geometric deep learning extension library for PyTorch. All code provided at GitHub repository [36].

IV.2 Results and discussion

The results of our comparative evaluation experiments are summarized in Tab. IV.1.

In the experiments demonstrated here, not all selected methods for binary node classification outperforms proposed trivial method.

A method based on diffusion-convolutional representation is most likely limited due to their invariance with respect only to node index without his position. This method implements the latent representation from diffusion processes that begin at each node, but we can not use spatial dependencies between individual nodes.

A method based on on embedding nodes, as GraphSage, and a method based on self-attention strategy, as GAT, showed the best accuracy. We can explain this because these methods study characteristics of neighboring nodes. GAT can overcome DCNN's limitation by learning the structural information of each node's neighborhood, on the other hand are limited by performing masked attention only for adjacent nodes. GraphSage studies the topological structure of each node's neighborhood as well as the distribution of node features in the neighborhood.

Given the results, propagation of feature information from neighboring nodes in every layer improves classification performance.

There are several potential improvements and extensions that could be addressed as future research based on this paper, such as implementation other selected methods, experiments with larger batch sizes, and a concatenation of different methods.

A particularly interesting research direction would be implementing a method based on the Recurrent Neural Network to solve this problem.

General conclusion

During my internship I could contribute to research topography of graphs that were presented as graphs. I explored different Deep Learning models and implemented GraphSage, DCNN, GAT. I demonstrated accuracy of this models on real dataset.

On my side, this internship noticeably broadened the list of my competences and allowed me to:

- work with a new machine learning library for me lastorch;
- deepen my knowledge of Deep Learning models on non-eEclidean data;
- gain an experience of working with professionals.

I appreciated the variability of work in an INT and the variety of required knowledge and competences. The most I enjoyed working in a group of professional researchers who respect their work. These are people who would gladly explain if there is something unclear and help if there is something difficult. To sum up, I would like to say that working at INT gave me only positive experience and allowed me to grow as an independent researcher.

Bibliography

[1] Pienaar R. Lee J.-M. Seong J.-K. Choi Y.Y. Lee K.H. Grant P.E. Im, K. Quantitative comparison and analysis of sulcal patterns using sulcal graph matching: a twin study. *NeuroImage*, 57 (3):1077–1086, 2011.

[2] Viellard M. Takerkart S.-Villeneuve N. Poinso-F. Da Fonséca D. Girard N. Deruelle C. Auzias, G. Atypical sulcal anatomy in young children with autism spectrum disorder. *NeuroImage*, 4:593–603, 2014.

[3] Klöppel S. Rivière D.-Perrot M. Frackowiak-R.S.J. Siebner H. Mangin J.-F. Sun, Z.Y. The effect of handedness on the shape of the central sulcus. *NeuroImage*, 60 (1):332–9, 2012.

[4] von Cramon D.Y. Colchester-A.C.F. Lohmann, G. Deep sulcal landmarks provide an organizing framework for human cortical folding. *Cereb. Cortex*, 18 (6):1415–1420, 2008.

[5] Auzias G. Lucile B.-Coulon O. Takerkart, S. Structural graph-based morphometry: A multiscale searchlight framework based on sulcal pits. *NeuroImage*, 35:32–45, 2017.

[6] Tsuda K. Inokuchi A. Kashima, H. Kernels for graphs. *Kernels and Bioinformatics*, page 155–170, 2014.

[7] Al-Rfou R. Skiena-S. Perozzi, B. Deepwalk: Online learning of social representations. page 701–710, 2014.

[8] Gori-M. Tsoi A.C.-Hagenbuchner M. Monfardini-G. Scarselli, F. The graph neural network model. *NeuroImage*, 20:61–80, 2009.

[9] Zaremba-W. Szlam A.-LeCun Y. Bruna, J. Spectral networks and locally connected networks on graphs. *Proceedings of International Conference on Learning Representations*, 2014.

[10] Saladin. Human anatomy. page 416, 2011.

[11] Sporns. Networks of the brain. page 143, 2010.

[12] Schleicher-A. Omran H.-Curtis M. Zilles-K. Armstrong, E. The ontogeny of human gyrification. *Cereb. Cortex*, 5 (1):56–63, 1995.

[13] Brun-L. Deruelle C.-Coulon O. Auzias, G. Deep sulcal landmarks: algorithmic and conceptual improvements in the definition and extraction of sulcal pits. *NeuroImage*, 111:12–25, 2015.

[14] Mangin-J.-F. Frouin-V. Sastre F.-Peragut J.C. Samson Y. Regis, J. Generic model for the localization of the cerebral cortex and preoperative multimodal integration in epilepsy surgery. *Funct. Neurosurg*, 65 (1-4):72–80, 1995.

[15] Han-X.-Xu C.-Prince J.L. Rettmann, M.E. Automated sulcal segmentation using watersheds on the cortical surface. *NeuroImage*, 15 (2):329–344, 2002.

[16] Rivière-D. Cachia-A. Duchesnay E.-Cointepas Y. Papadopoulos-Orfanos D.-Collins D.L. Evans A.C. Régis J. Mangin, J.-F. Object-based morphometry of the cerebral cortex. *NeuroImage*, 23 (8):968–982, 2004.

[17] Shirui P.-Fengwen C.-Guodong L. Chengqi-Z. Yu P.S. Zonghan, W. A comprehensive survey on graph neural networks. 2019.

[18] Retrieved from graph tool.skewed.de. Graph-tool performance comparison. https://graph-tool.skewed.de/performance.

[19] Retrieved from https://www.python.org/. pickle — Python object serialization. https://docs.python.org/3/library/pickle.html.

[20] Michael Himsolt. GML: A portable Graph File Format. http://www.networkdynamics.org/static/zen/html/api/io/gml.html.

[21] B. McKay. graph formats. http://users.cecs.anu.edu.au/~bdm/data/formats.html.

[22] D. Crockford. Introducing JSON. https://www.json.org/.

[23] Auzias-G.-Viellard M.-Villeneuve N. Girard-N. Poinso F.-Da Fonseca-D. Deruelle C. Brun, L. Localized misfolding within broca's area as a distinctive feature of autistic disorder. *NeuroImage*, 1:160–168, 2016.

[24] Fischl-B.-Dale A.-Halgren E. Destrieux, C. Automatic parcellation of human cortical gyri and sulci using standard anatomical nomenclature. *NeuroImage*, 53 (1):1–15, 2010.

[25] Welling-M. Kipf, T.N. Semi-supervised classification with graph convolutional networks. 2016.

[26] Tarlow-D.-Brockschmidt M.-Zemel R. Li, Y. Gated graph sequence neural networks. 2015.

[27] Kozareva-Z.-Dai B.-Smola A.J. Song-L. Dai, H. Learning steady-states of iterative algorithms over graphs.

[28] Schoenholz-S.S.-Riley P.F.-Vinyals O. Dahl-G.E. Gilmer, J. Neural message passing for quantum chemistry. 2017.

[29] Ying-R.-Leskovec J. Hamilton, W.L. Inductive representation learning on large graphs. 2017.

[30] Towsley-D. Atwood, J. Diffusion-convolutional neural networks. 2015.

[31] Wang-Z.-Ji S. Gao, H. Large-scale learnable graph convolutional networks. 2018.

[32] Cucurull-G.-Casanova A.-Romero A. Liò-P. Bengio Y. Veličković, P. Graph attention networks. 2017.

[33] Shi-X.-Xie J.-Ma H. King-I. Yeung D.Y. Zhang, J. Gaan: Gated attention networks for learning on large and spatiotemporal graphs. 2018.

[34] PyCharm. https://www.jetbrains.com/pycharm/.

[35] PyTorch. https://pytorch.org/.

[36] GitHub. https://github.com/yaroslavmavliutov/Characterization-of-cortical-folding-patterns-by-machine-learning-on-graphs.

Glossary

DL Deep Learning

INT Institut de Neurosciences de la Timone

OBM Object-Based Morphometry

CNN Convolutional Neural Network

GNN Graph Neural Network

GCN Graph Convolution Network

GAT Graph Attention Network

GCN Graph Generative Network

RNN Recursive Neural Network

GGNN Gated Graph Neural Network

GGS NN Gated Graph Sequence Neural Network

SSE Stochastic Steady-state Embedding

MPNN Message Passing Neural Network

MLP Multi Layer Perceptron

DCNN Diffusion-Convolutional Neural Network

LGCN Large-Scale Learnable Graph Convolutional Network

GAAN Gated Attention Networks

List of Tables

List of Figures