REPRODUCIBILITY IN A.I. FOR BIOMEDICAL RESEARCH: AN APPLICATION TO PARKINSON'S DISEASE

by

CHRISTIAN LEE MCDANIEL

(Under the Direction of Shannon Quinn)

ABSTRACT

Deep learning-based data analysis techniques have found many uses in biomedical research. Recent expansion of open source databases and computational tools has fostered distributed and explorative research. Under these conditions, reproducibil- ity and experimental rigor must be ensured. Developing explicit analysis pipelines exposes the scientific process and yields reproducible results. In this thesis, we look at the case of deep learning-based data analysis for Parkinsons disease (PD) research. We develop end-to-end pipelines in two PD-related fields: accelerometer data analysis and neuroimage analysis. First, we construct a simple yet robust recurrent neural network for classifying motor activity from accelerometer data alone; this has applications for identifying the mo- tor symptoms of PD. Next, we propose a novel graph convolutional network architecture for distinguishing PD patients vs. healthy controls from multimodal neuroimage data. Our pipelines standardize the data preprocessing and analysis steps, fostering reproducibility and deliberate progression of their respective fields.

INDEX WORDS: artificial intelligence, data science, healthcare, Parkinson's disease, deep learning, long short-term recurrent neural network, graph convolutional network, accelerometer, neuroimaging, reproducibility

REPRODUCIBILITY IN A.I. FOR BIOMEDICAL RESEARCH: AN APPLICATION TO PARKINSON'S DISEASE

by

CHRISTIAN LEE MCDANIEL

B.S., The University of Georgia, 2016

A Thesis Submitted to the Graduate Faculty of The University of Georgia in Partial Fulfillment of the Requirements for the Degree

MASTER OF SCIENCE

ATHENS, GEORGIA

2019

REPRODUCIBILITY IN A.I. FOR BIOMEDICAL RESEARCH: AN APPLICATION TO PARKINSON'S DISEASE

by

CHRISTIAN LEE MCDANIEL

Major Professor: Shannon Quinn

Comittee: Tianming Liu
Frederick Maier

Electronic Version Approved:

Suzanne Barbour
Dean of the Graduate School
The University of Georgia
August 2019

DEDICATION

To Jake, for leading the way.

ת ם ס ל

ACKNOWLEDGEMENTS

A very sincere thanks to those who have helped me along the way. My mom and dad offered tremendous support and leadership. My other family, friends, and my dog Dexter kept me moving forward in one piece. My advisor Dr. Quinn has been an exemplary role model. Caesar, Logan, and many nameless machines allowed the completion of this work. Thanks to the UGA professors and faculty members who offered mentorship and guidance in and out of the classroom. Finally, this work would not be possible without the incredible open source Python community and other open science resources, such as UC Berkley's Machine Learning Repository and the Parkinson's Progression Markers Initiative database.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

CHAPTER 1

INTRODUCTION AND LITERATURE REVIEW

This thesis examines the creation and use of data science pipelines as a partial solution to the reproducibility problem in deep learning-based biomedical research; publishing reproducible work yields open science practices in this domain. Our efforts are aligned with the Center for Open Science [1] , which states "Show Your Work. Share Your Work. Advance Science. That's Open Science." We define *data science* to encompass all aspects of data processing and analysis required for data-centered research. This includes data cleaning and formatting, model construction and training, and performance evaluation. We categorize the development of *data science pipelines* as the delineation of steps taken in data-centered research.

In Chapter 1, we contextualize the problem of reproducibility in healthcare-related research, discuss the use of deep learning-based methods for modeling biomedical data, and introduce the data science pipeline. We then reintroduce the problem in the context of Parkinson's disease (PD) research, before continuing to Chapters 2 and 3, each of which designs an end-to-end pipeline for a specific method of deep learning-based PD research. Through these two experiments, we demonstrate real-world examples of the reproducibility issue in biomedical research, and provide tangible contributions to the domains on which they focus. Finally, we include our concluding remarks in Chapter 4.

Data Science in Healthcare

The field of healthcare is described as being "information rich" but "knowledge poor" [1]. Despite the quantity of information, or data, that is available, the complexity of the data, difficulty of its acquisition, and scrutiny required of insights made from it create significant

[1] `https://cos.io`

barriers to successful use of that information. As the prevalence of and accessibility to biomedical data increases [2, 3], data sets have been made *open source*, or publicly available online. Analysis techniques have both advanced in modeling capacity and become easier to implement thanks to free and user-friendly computer softwares and libraries.

In fact, large corpuses of biomedical data have allowed researchers to borrow from other fields and make use of rapidly developing deep learning-based techniques [4]. Deep machine learning (DL) algorithms are a subgroup of machine learning algorithms with characteristic "deep" architectures. In their simplest form, DL architectures resemble many iterations of logistic regression. Logistic regression is a classical machine learning algorithm which uses a weight matrix to transform input data. Through multiplication between the data and the weight matrix, logistic regression transforms the input data such that it distinctly resembles inputs from one particular *class*, or grouping. A nonlinear *activation function*, the sigmoid function, is applied to the transformed data to encourage the separation of class groupings [5].

DL algorithms group many such transformations, termed *units*, side-by-side into a single *layer*, and then stack many of these layers in succession to form "deep" architectures. Multiple units per layer allow unique transformations and combinations of the inputs to a given layer, and stacked layers perform hierarchical grouping to isolate underlying patterns in the data. These algorithms, collectively termed *deep neural networks*, are particularly well-suited for the noisy and unstructured nature of biomedical data [3, 6].

In Chapters 2 and 3 of this thesis we will explore in detail two specific deep learning-based algorithms and examine their use for biomedical research. Specifically, Chapter 2 will discuss the long short-term memory (LSTM) recurrent neural network (RNN), its use for classifying human motor activity from accelerometer time series data, and why it is better suited for this data over traditional feed-forward neural networks or vanilla (i.e., regular) RNNs. Next, Chapter 3 will introduce the graph convolutional network (GCN) and graph attention network (GAT) and their use for multi-modal neuroimage data.

As analysis tools and biomedical data sets on which to use them become increasingly available through open source practices, research in the field has shifted from tightly controlled, hypothesis-driven experiments to a process more distributed and exploratory in nature [2]. The ability for data analysts from other fields to apply their skills to healthcare research is beneficial to balancing the "information" vs. "knowledge" scales. More research increases the chances of medical and technological breakthroughs [3]. There is great potential in healthcare-related analytics to reduce the cost of healthcare in the U.S., which is twice that of many other developed nations [7]. The field also seeks to reduce bias and human error in diagnosis, and make medical expertise more readily available [1].

Reproducibility in Healthcare Analytics

Peng 2015 explains reproducibility as "the ability to recompute data analytic results, given an observed data set and knowledge of the data analysis pipeline" [8]. Reproducibility is distinct from replicability, which is the ability to obtain the same results with different data and/or methodologies. Replicability is often difficult to achieve in biomedical research due to the resources required to compile data and perform experiments. As such, reproducibility is an alternate, albeit minimal, standard for assessing a study's claims [9].

In her *Reproducibility PI Manifesto*, Barba 2012 pledges eight principles for upholding reproducibility as a PI. They are as follows: "I will teach my graduate students about reproducibility," "All our research code (and writing) is under version control," "We will always carry out verification and validation," "For main results in a paper, we will share data, plotting script and figure under CC-BY," "We will upload the preprint to arXiv at the time of submission of a paper," "We will release code at the time of submission of a paper," "We will add a 'Reproducibility' declaration at the end of each paper," and "I will keep an up-to-date web presence" [10]. These action statements utilize trusted open science partners to ensure reproducibility. Some of them may take time to incorporate into a lab's

standard procedure, but the contribution to the advancement of science makes them worth the effort.

A central focus of open science, reproducibility is paramount to the progression of scientific research; it cultivates confidence in results [11] and paves the way for future research by allowing minor tweaks to and continuations of previous work [8, 2]. Still, reproducibility has long been a major issue in biomedical research. A 2012 *Nature* report found "widespread deficiencies" in descriptions of experimental design, such as the omission of settings used for potentially bias introducing parameters [12]. The journal *Science* echoed similar concern in 2014, announcing its addition of statisticians to its reviewer panel in response to irreproducible statistical findings in domain-specific papers [11].

The complexity of the systems being modeled is a particular challenge to reproducibility in biomedical research [11]. Reporting, and even knowing, all of the details of an experiment is a cumbersome task. Biomedical data often requires many steps before it is prepared for analysis, and it can be difficult to know which steps need reporting. Confidentiality also poses a challenge. Biomedical data is often subject to stringent regulation; making it fully public often requires a concerted effort to remove all sensitive and identifying information and obtain official approval for release. Many data sets remain proprietary or restricted from dissemination, hindering replication of a study which uses such data.

We discussed in the previous section the benefits of making data and analysis tools open source: publicly available resources accelerate the progress of biomedical research by allowing researchers who lack the means to collect data or the knowledge to construct their own studies to enter the field. However, if not utilized and reported correctly, open source resources can work against open science, exacerbating the reproducibility issue. Quick implementation allows for iterative tweaking of procedural steps and documentation may suffer. Open science also encourages design flexibility by providing an easy way to tweak previous studies and combine ideas. Increases in the number of researchers in a field and experiment flexibility have been shown to contribute to false positives in scientific

research [13]. Many of the researchers entering biomedical analytics lack the training to publish reproducible experiments [8, 14]. Combined with the increased burden on journal reviewers to catch mistakes [8] — especially when statistics is not their domain of expertise [11] — and coupled with existing bias in published research for positive novel results, this issue results in more false positives and non-reproducible experiments entering the literature.

Many reports which analyze biomedical data — e.g., a study which uses deep learning to identify lung cancer in chest x-rays — focus on the application of a novel algorithm to outperform results obtained by a previous study which used a different algorithm. Often, experimental details are informally reported, if reported at all [14]. We found this to be particularly prevalent in papers applying novel deep learning algorithms to biomedical domains. Since the main point of such reports is often to compare results with algorithms from other areas of machine learning, such as support vector machines or logistic regression, the comparison of results tends to be the focus, and experimental details are neglected. We found that model hyperparameter and architecture choices often lack explanation or justification. While there have been reports designed to edify the design of specific deep learning algorithms (e.g., recurrent neural networks for text classification [15] and Restricted Boltzmann Machines [16]), most other reports are exploratory in nature, and there is a lack of groundwork to develop evidence-based guidance of their decisions. Without the inclusion of experimental details or discussion of decisions made, the field cannot incorporate new discoveries in a deliberate fashion.

We discuss these issues in Chapters 2 and 3, and propose remedying strategies. For example, in Chapter 2 we optimize our LSTM network over an expansive hyperparameter search space to unify disjoint implementation choices made in literature. In Chapter 3, we provide an in-depth explanation of our neuroimage preprocessing pipeline and we examine the choices we made when constructing our novel GCN architecture. We provide our code

for both experiments [2] [3] . Finally, we include a minimum of six categories of information in the contents of our report, which are outlined later in this chapter and can be remembered by the acronym "PRIMAD".

Biomedical data is typically complex: tricky procedures are often required to obtain it, noise and individual differences introduce variance, metadata and contextual information often accompany a given measurement. There are caveats to its use that researchers from other fields may not be trained to handle [8, 14]. For example, data analysts may neglect the fact that biomedical data sets are often quite small and collected from a single study. Findings from these data sets may not generalize well, and this should be reported as a limitation of the results. Explicitly stating a study's limitations increases the benefits of its findings and inspires future work [12]. While the ability for deep learning researchers and data analysts to apply their skills to biomedical research is beneficial to the field overall, steps must be taken to ensure the gap between the two specialties is bridged as seamlessly as possible. In this way, reproducibility in biomedical research is essential to enabling trust in analyses [8] [14].

The Data Science Pipeline

Finding errors in one's work is primarily the responsiblity of the researcher [8]. Documentation is a chance for an author to review and repair mistakes in his or her code [14]. Going further, the development of trusted reproducible processing pipelines fosters deliberate methodologies that can be reused and revised [8]. This practice is particularly useful in highly technical areas, such as functional connectomics research, which aims to find functional networks of brain regions. Researchers in this field are working to compile a central data processing platform through which to conduct their research and share their results [17].

[2] `https://github.com/xtianmcd/accelstm`
[3] `https://github.com/xtianmcd/GCNeuro`

Although producing user-friendly generalizable code may exceed the limited resources allotted to a research project, there are a few incentives for spending the extra time. The journal *Biostatistics* uses tags on their papers such as "R" to denote reproducible work, and "D" and "C" if the authors provide the data or code, respectively [9]. Sharing code and data seems particularly applicable to open science, where sites such as GitHub [4] and Bitbucket [5] allow reproducible work to be shared and used with ease. Additionally, Freire, Fuhr, and Rauber 2016 argue that when a study's methods can be reproduced, that study benefits from "increased impact, recognition, and citation rates" [14]. These small measures offer some recognition and encouragement for the extra steps required for reproducible work.

As a general rule of thumb, an independent researcher should be able to repeat the analysis solely from details included in the paper [18]. We categorize the delineation of steps taken in data-centered research as the development of data science pipelines for a particular domain. In 2016, an international group of computer science and domain experts gathered to discuss reproducibility in data-oriented "e-science", i.e., they discussed open science. They teamed together to generate the following message for journal editors and conference chairs:

> Transparency, openness, and reproducibility are vital features of science. Scientists embrace these features as disciplinary norms and values, and it follows that they should be integrated into daily research activities. These practices give confidence in the work; help research as a whole to be conducted at a higher standard and be undertaken more efficiently; provide verifiability and falsifiability; and encourage a community of mutual cooperation. They also lead to a valuable form of paper, namely, reports on evaluation and reproduction of prior work. Outcomes that others can build upon and use for their own research, whether a theoretical construct or a reproducible experimental result, form a foundation on which science can progress. . . .

[4] `https://github.com`
[5] `https://bitbucket.org`

[14]. The authors also report a set of key variables to include in any report, which we use as guidance in developing our data science pipelines. Each of these variables are key points which contribute to the results of a study, and are points of modification for inspiring future research.

The set of variables can be remembered as PRIMAD. "P" relates to the platform and execution environment, including the hardware and software stack. "R" refers to the research objectives. "I" represents the implementation, or code, which should be made available whenever possible. "M" stands for the methods used, such as specific algorithms. "A" are the actors, the authors of the study. "D" is the data, including the raw input data that is analyzed and the parameter settings used in analysis tested throughout the experiment. By reporting each of these variables within published research papers, authors create a reproducible data science pipeline. By identifying these variables in other studies, we can measure the robustness of results and the reproducibility of methods.

An Application to Parkinson's Disease

The issue of reproducibility first arose in our lab during participation in an online challenge aimed at Parkinson's disease (PD) research. Affecting more than 1% of the United States population over the age of 60, Parkinson's disease (PD) is the second-most prevalent age-related neurodegenerative disease following Alzheimer's disease [19]. PD diagnosis has traditionally relied on clinical assessments with some degree of subjectivity [20], often missing early-stage PD altogether [21]. Benchmarks for delineating PD progression or differentiating between similar conditions are lacking [22, 23]. As such, many efforts have emerged to identify quantitatively rigorous methods through which to distinguish PD.

Titled the Parkinson's Disease Digital Biomarker DREAM Challenge, the task of the online challenge was to develop a data analysis model capable of predicting the presence of PD from accelerometer data. Accelerometer data measures motion along the x-, y-, and z-axes over time, producing triaxial time series data. Given the characteristic motor symp-

toms of PD, and the increased prevalence of smartphones and wearable devices that contain accelerometers, the use of accelerometer data in PD diagnosis is an attractive research endeavor [6, 24].

Early studies seeking to classify so-called human activity recognition (HAR) tasks from accelerometer data used classical machine learning algorithms such as support vector machines and logistic regression [25]. While these methods often performed quite well, they required intensive preprocessing of data into de-noised hand-crafted features before use. Researchers began investigating recurrent neural networks (RNNs) as an alternative, given their reported success for capturing temporal dependencies in text data [15].

Given these efforts, we turned to the literature to explore the use of RNNs for modeling accelerometer data. We found that in trying to replicate the methods used by many of the published studies in literature, many implementation details were missing or lacking explanation. The lack of traceable and justifiable experiment design choices led us to conduct our own experiment aimed at filling some of these gaps. Unfortunately, the PD-related accelerometer data from the online challenge had not been released for use outside the challenge, so we used a publicly available HAR accelerometer data set instead. We hope to see our efforts applied to the accelerometer data upon its release, to compare our simple LSTM-based pipeline with the leading results of the online challenge.

In *Developing a Start-to-Finish Pipeline for Accelerometer-Based Activity Recognition Using Long Short-Term Memory Recurrent Neural Networks*, outlined in Chapter 2 of this thesis, we first introduce the problem, expanding upon the discussion in the preceding paragraphs. In the Background section, we outline the feed forward neural network, the recurrent neural network, and the long short-term memory cell; we examine the differences between them and situations when one may be preferred over the others. Next, we performed a review of literature related to the use of LSTM RNNs for HAR tasks in the Related Works section. We compiled the various implementation choices made, the gaps needing to be addressed, and the shortcomings that might be improved upon.

We used this investigation to guide our subsequent delineation of a data science pipeline for this domain. Our pipeline, outlined in the Experimental Setup section, performs very little preprocessing and utilizes a very simple LSTM architecture, optimizing its hyperparameters to fit the task. This way, we isolate the LSTM cell's ability to classify accelerometer data and provide a baseline for more complex models and processing pipelines.

Paying close attention to decisions made during preprocessing and performance evaluation, we sought to prevent statistical errors found in the literature and improve the robustness of our results. We utilized the tree-structured Parzen estimator (TPE) Bayesian optimization algorithm to conduct an expansive hyperparameter search over all hyperparameters tested in literature. This guided the construction of our final LSTM model which we trained and tested on six-class HAR accelerometer data. The report in Chapter 2 demonstrates the efficacy of our simple LSTM-based pipeline by performing comparably with benchmark studies, as we show in the Results section and discuss in the Discussion section.

Our experiences using accelerometer data led us to examine another data modality useful for PD research. As a neurodegenerative disease, PD etiology originates in the brain [26]. Abnormalities in the production of dopamine result in downstream affects throughout the brain and subsequently throughout the body (e.g., the motor symptoms targeted by accelerometer-based PD research). Researchers are hopeful that characterizing the neurophysiology of PD will aid in early detection and subtyping of the disease [21]. As such, neuroimage research has caught the interest of many PD researchers.

In Chapter 3 of this thesis, we discuss our work developing a pipeline for multi-modal deep learning-based neuroimage analysis for PD research; the work is titled *Developing a Graph Convolution-Based Analysis Pipeline for Multi-Modal Neuroimage Data: An Application to Parkinson's Disease*. The first step in our report is to introduce neuroimage analysis for PD, explain why deep learning-based methods are justified, and show how such efforts have evolved over time. The Introduction and Related Works sections accom-

plish these goals. We continue to the Discussion of the Preprocessing Pipeline section. Neuroimage data is particularly complex and noisy, and requires extensive preprocessing before it can be used. The preprocessing steps are highly technical and insufficiently defined in many previous works; decisions made during preprocessing can greatly affect the outcome of analysis [27, 28]. Furthermore, while various neuroimaging modalities capture unique aspects of a patient's neurophysiological state, the means of combining multiple modalities for composite analysis are poorly defined in literature, preventing reproduction of existing studies. We seek to foster reproducibility and save time for others conducting similar research by explicitly stating the steps we use and discussing any major difficulties or nuances found.

Our analysis is based on previous works, which utilize the graph convolutional network (GCN) as a means for defining multimodal neuroimage data in a common data space. We construct a novel GCN architecture and train it to predict the presence of PD vs. healthy controls. The Methods section delineates the specific model implementations we used and outlines the experiment through which we demonstrated the efficacy of our pipeline.

The following chapters contain our published experiments to develop data science pipelines for two PD-related data analysis domains. Both of our experiments were somewhat exploratory in nature and were part of the learning process. In each experiment, we discuss the underlying math and previous research behind the techniques employed therein. In identifying missing details in other works, we do not seek to diminish the results of those experiments, but rather enhance their findings by facilitating reproduction and unification of methods in the field. We hope that our discussions and findings are useful for researchers in these fields and are helpful in improving reproducibility in biomedical data analysis. Our discussions are continued in the final chapter of this thesis.

CHAPTER 2

DEVELOPING A START-TO-FINISH PIPELINE FOR ACCELEROMETER-BASED ACTIVITY RECOGNITION USING LONG SHORT-TERM MEMORY RECURRENT NEURAL NETWORKS

[1]

[1] McDaniel, C.L. and S. Quinn. 2018. *Proceedings of the 17th Python in Science Conference (SciPy 2018)* 31-40. Reprinted here with permission of the publisher.

Abstract

Increased prevalence of smartphones and wearable devices has facilitated the collection of triaxial accelerometer data for numerous Human Activity Recognition (HAR) tasks. Concurrently, advances in the theory and implementation of long short-term memory (LSTM) recurrent neural networks (RNNs) has made it possible to process this data in its raw form, enabling on-device online analysis. In this two-part experiment, we have first amassed the results from thirty studies and reported their methods and key findings in a meta-analysis style review. We then used these findings to guide our development of a start-to-finish data analysis pipeline, which we implemented on a commonly used open-source data set in a proof of concept fashion. The pipeline addresses the large disparities in model hyperparameter settings and ensures the avoidance of potential sources of data leakage that were identified in the literature. Our pipeline uses a heuristic-based algorithm to tune a baseline LSTM model over an expansive hyperparameter search space and trains the model on standardized windowed accelerometer signals alone. We find that we outperform other baseline models trained on this data and are able to compete with benchmark results from complex models trained on higher-dimensional data.

Introduction

Human Activity Recognition (HAR) is a time series classification problem in which a classifier attempts to discern distinguishable features from movement-capturing on-body sensors [29]. The most common sensor for HAR tasks is the accelerometer, which measures high-frequency (30-200Hz) triaxial time series recordings, often containing noise, imprecision, missing data, and long periods of inactivity between meaningful segments [30, 31, 32]. Consequently, attempts to use traditional classifiers typically require significant preprocessing and technical engineering of hand crafted features from raw data, resulting in a barrier to entry for the field and making online and on-device data processing impractical [33, 34, 35, 30, 32].

The limitations of classical methods in this domain have been alleviated by concurrent theoretical and practical advancements in artificial neural networks (ANNs), which are more suited for complex non-linear data. While convolutional neural networks (CNNs) are attractive for their automated feature extraction capabilities during convolution and pooling operations[36, 37, 38, 39, 40, 33, 32, 35], recurrent neural networks (RNNs) are specifically designed to extract information from time series data due to the recurrent nature of their data processing and weight updating operations [41]. Furthermore, whereas earlier implementations of RNNs experienced problems when processing longer time series (tens to hundreds of time steps), the incorporation of a multi-gated memory cell in long short-term memory recurrent neural networks (LSTMs) [42] along with other regularization schemes helped alleviate these issues.

As RNN usage continues, numerous studies have emerged to address various aspects of understanding and implementing these complex models, namely regarding the vast architectural and hyperparameter combinations that are possible [43, 44, 45, 15, 46]. Unfortunately, these pioneering studies tend to focus on tasks other than HAR, leaving the time series classification tasks of HAR without domain-specific architecture guidance.

In a meta-analysis style overview of the use of LSTM RNNs for HAR experiments across 30 reports (discussed below), we found a general lack of consensus regarding the various model architectures and hyperparameters used. Often, a given pair of experiments explored largely or entirely non-overlapping ranges for a single hyperparameter. Key architectural and procedural details are often not included in the reports, making reproducibility impossible. The analysis pipelines employed often lack detail; sources of data leakage, where information from the testing data is exposed to the model during training, appear to be overlooked in certain cases. Without clear justifications for model implementations and deliberate, reproducible data analysis pipelines, objective model comparisons and inferences from results cannot be made. For these reasons, the current report seeks to summarize the previous implementations of LSTMs for HAR research available in literature and

outline a structured data analysis pipeline for this domain. We implement a truncated version of our pipeline, optimizing a baseline LSTM over an expansive hyperparameter search space, and obtain results on par with benchmark studies. We suspect that our efforts will encourage scientific rigor in the field going forward and initiate more granular exploration of the field as we understand these powerful data analysis tools within this domain.

Background

This section is intended to give the reader a digestible introduction to ANNs, RNNs, and the LSTM cell. The networks will be discussed as they relate to multi-class classification problems as is the task in HAR.

Artificial Neural Networks. The first ANN architecture was proposed by Drs. Warren McCulloch and Walter Pitts in 1943 as a means to emulate the cumulative semantic functioning of groups of neurons via propositional logic [47, 48]. Frank Rosenblatt subsequently developed the Perceptron in 1957 [49]. This ANN variation carries out its step-wise operations via mathematical constructs known as linear threshold units (LTUs). The LTU operates by aggregating multiple weighted inputs and feeding this summation u through an activation function $f(u)$ or step function step(u), generating an interpretable output $\tilde{y}$ (e.g. 0 or 1) [48].

$$\begin{aligned} \tilde{y} &= f(u) \\ &= f(w \cdot x) \end{aligned}$$

where $\cdot$ is the dot product operation from vector calculus. x is a single instance of the training data, containing values for all n attributes of the data. As such, w is also of length n, and the entire training data set for all m instances is a matrix X of dimensions m by n (i.e., m x n).

A 2-layer ANN can be found in Figure 2. Each attribute in instance x_i represents a node in the perceptron's input layer, which simply provides the raw data to the output layer - where the LTU resides. To represent k target classes, k LTU nodes are included in the output layer, each corresponding to a single class in y. Each LTU's prediction $\tilde{y}$ indicates the predicted probability that the training instance belongs to the corresponding class. The LTU output with the largest value - $\max(\tilde{y})$ - is taken as the overall predicted class for the instance of the data being analyzed. Taken over the entire data set, each LTU has a prediction vector $\tilde{y}_k$ length m and the entire output layer produces a prediction matrix $\tilde{Y}$ with dimensions m x k. Additionally, each LTU contains its own weight vector w_k of length n (i.e., a fully-connected network), resulting in a weight matrix W of dimensions n x k.

ANNs often contain complex architectures with multiple layers, which allow for non-linear transformations of the data and increase the flexibility and robustness of the model. If we look at a simple three-layer neural network (see Figure 2 B), we see input and output layers as described above, as well as a layer in the middle, termed a *hidden layer*. This layer acts much like the output layer, except that its outputs z for each training instance are fed into the output layer, which then generates predictions $\tilde{y}$ from z alone. The complete processing of all instances of the dataset, or all instances of a portion of the dataset called a *mini-batch*, through the input layer, the hidden layer, and the output layer marks the completion of a single *forward pass*.

For the model to improve, the outputs generated by this forward pass must be evaluated and the model updated in an attempt to improve the model's predictive power on the data. An error term (e.g., sum of squared error (sse)) is calculated by comparing individual predictions $\tilde{y}_k$ to corresponding ground truth target values in y_k. Thus, an error matrix E is generated containing error terms over all k classes for all m training instances. This error matrix is used as an indicator for how to adjust the weight matrix in the output layer so as to yield more accurate predictions, and the corrections made to the output layer give an

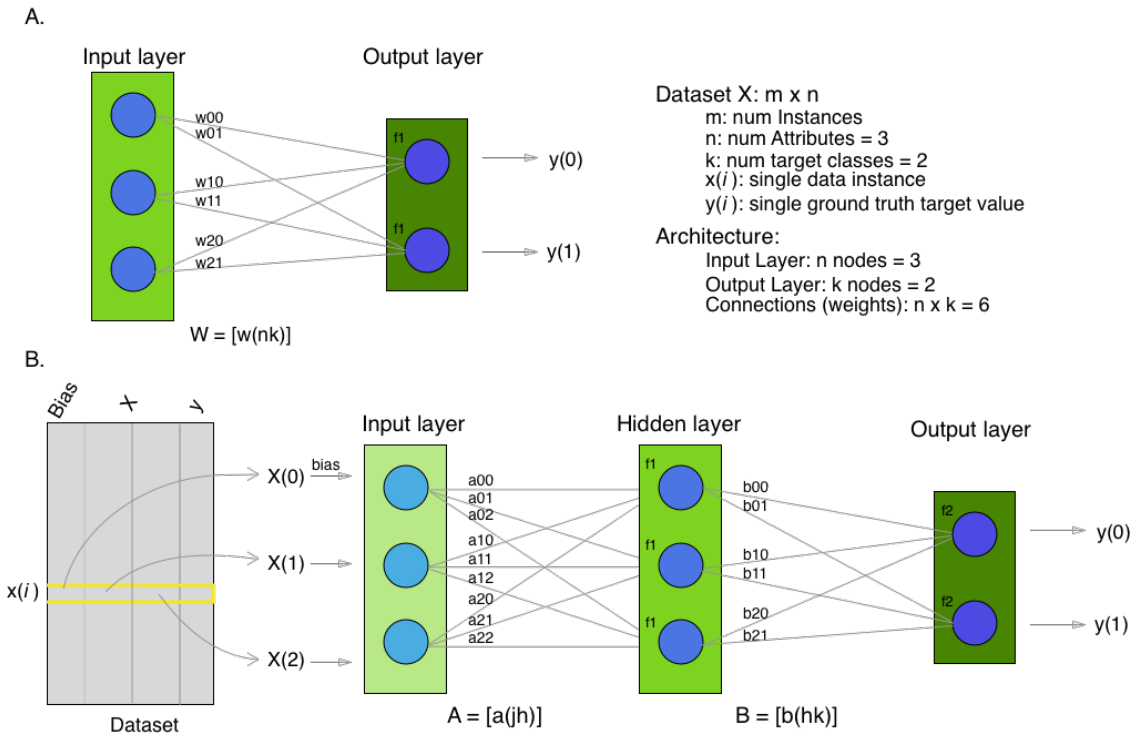

Figure 2.1: **A.** A two-layer network and associated dimensions of the components. **B.** A three-layer network showing a single data instance x_i being fed in as input.

indication of how to adjust the weights in the hidden layer. This process of carrying the error backward from the output layer through the hidden layer(s) is known as *backpropagation*. One forward pass and subsequent backpropagation makes up a single *epoch*, and the training process consists of many epochs repeated in succession to iteratively improve the model.

The iterative improvements to the model are known as *optimization*, and many methods exist to carry this process out. The common example is stochastic gradient descent (SGD), which calculates the gradient of the error - effectively the steepness of E's location as it "descends" toward lower error - and adjusts the weight matrices at each layer in a direction opposite this gradient. The change to be applied to the weight matrices is mediated via a learning rate η [50].

$$E = Y - f(XW)$$

Optimization:

$$\min_W \|E\|_F$$

$$hsse_W = \frac{1}{2}\sum_{c=0}^{k-1}(y_c - f(X \cdot w_c)) \cdot (y_c - f(X \cdot w_c))$$

$$\begin{aligned}\frac{\partial hsse}{\partial w_k} &= X * [f\prime(X \cdot w_k) * e_k] * \eta \\ &= -X * \delta_k * \eta\end{aligned}$$

where $f(...)$ represents the activation function, min_W represents the objective function of minimizing with respect to W, and $\|E\|_F$ stands for the Frobenius norm on the error matrix E. hsse_W represents the halved (for mathematical convenience) sum of squared error, calculated for all k nodes in the output layer. $f\prime(...)$ represents the derivative of the activation function over the term in the parentheses.

Looking at our three-layer neural network depicted in Figure 2, a single epoch would proceed as follows:

1. Conduct a forward pass, compute $\tilde{y}$ and compare with y to generate the error term:

$$z_h = f_1(a_h \cdot x)$$

$$\tilde{y}_k = f_2(b_k \cdot z)$$

$$e_k = y_k - \tilde{y}_k$$

2. Backpropagate the error regarding the correction needed for $\tilde{y}$.

3. Backpropagate the correction to the hidden layer.

4. Update weight matrices A and B via δ^y and δ^z:

$$\begin{aligned} b_{hk} &= b_{hk} - z_h \delta_k^y * \eta \\ &= b_{hk} - \frac{\partial hsse}{\partial b_{hk}} * \eta \end{aligned}$$

$$\begin{aligned} a_{jh} &= a_{jh} - x_j \delta_h^z * \eta \\ &= a_{jh} - \frac{\partial hsse}{\partial a_{jh}} * \eta \end{aligned}$$

sse is commonly used as the error term for regression problems, whereas squared error or *cross entropy* is typical for classification problems.

$$\text{cross entropy} = -\sum_{i=1}^{m} \sum_{c=1}^{k} y_i c * log(f_c(x_i))$$

The high flexibility of neural networks increases the chances of overfitting, and there are various ways to avoid this. *Early stopping* is a technique that monitors the change in performance on a validation set (subset of the training set) and stops training once improvement slows sufficiently. *Weight decay* helps counter large updates to the weights during backpropagation and slowly shrinks the weights toward zero in proportion to their relative sizes. Similarly, the *dropout* technique "forgets" a specified proportion of the outputs from a layer's neurons by not passing those values on to the next layer. *Standardizing* the input

is important, as it encourages all inputs to be treated equally during the forward pass by scaling and mitigating outliers' effects [48, 50].

Other hyperparameters tend to affect training efficiency and effectiveness and tend to differ with different datasets and types of data. Hammerla, et. al. found *learning rate* η to be an important hyperparameter in terms of its effect on performance [51]. Too small a learning rate and the model will exhibit slow convergence during training, while too large a value will lead to wild oscillations during optimization [50]. Hammerla, et. al. also find the *number of units* per layer n to be important, and Miller adds that too many hidden units is better than too few, leading to sparse layers of weight matrices versus restricting flexibility of the model, respectively. *Bias* helps account for irreducible error in the data and is implemeneted via a node whose inputs are always 1's (top node in the input layer of Figure 2 A). Reimers and Gurevych emphasize the importance of weight initialization for model performance in their survey of the importance of hyperparameter tuning for using LSTMs for language modeling [44]. Jozefowicz, et. al. cite the initialization of the forget gate bias to 1 as a major factor in LSTM performance [52].

Recurrent Neural Networks (RNNs). The recurrent neuron, developed by Drs. Ronald Williams and David Zipser in 1989 [41], is extremely useful in training a model on sequence data. Recurrent neurons address temporal dependencies along the temporal dimension of time series data by sending their outputs both forward to the next layer and "backward throught time," looping the neuron's output back to itself as input paired with new input from the previous time step. Thus, a component of the input to the neuron is an accumulation of activated inputs from each previous time step. Figure 2 depicts a recurrent neuron as part of a recurrent layer. Recurrent layers are placed between input layers and output layers and can be used in succession with densely connected and convolutional layers.

Instead of a single weight vector as in ANN neurons, RNN neurons have two sets of weights, one (wx) for the new inputs x_t and one (wy) for the outputs of the previous time

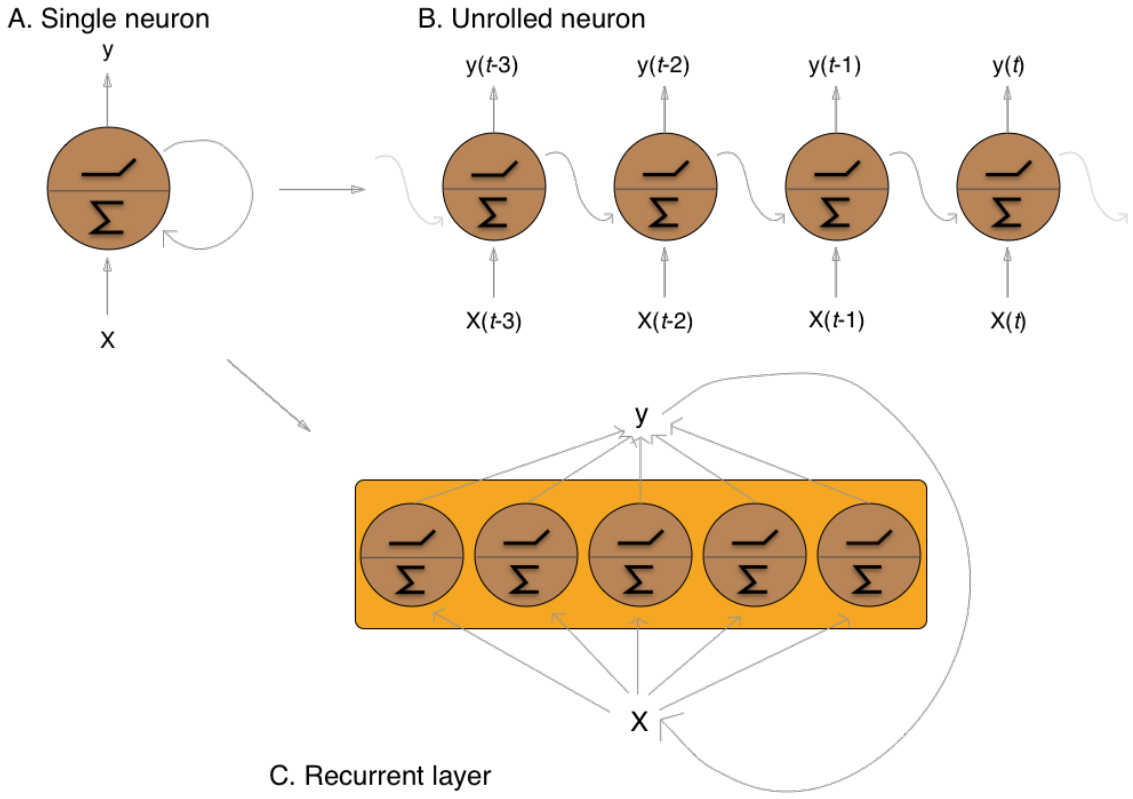

Figure 2.2: The recurrent neuron from three perspectives. **A.** A single recurrent neuron, taking input from X, aggregating this input over all timesteps in a summative fashion and passing the summation through an activation function at each timestep. **B.** The same neuron unrolled through time, making it resemble a multilayer network with a single neuron at each layer. **C.** A recurrent layer containing five recurrent nodes, each of which processes the entire dataset X through all time point.

step $y_{(t-1)}$, yielding matrices W_x and W_y when taken over the entire layer. The portion of the neuron which retains a running record of the previous time steps is the *memory cell* or just the *cell* [48].

Outputs of the recurrent layer:

$$y_{(t)} = \phi(W_x \cdot x_{(t)} + W_y \cdot Y_{(t-1)} + b)$$

where ϕ is the activation function and b is the bias vector of length n (the number of neurons).

The *hidden state*, or the *state*, of the cell ($h_{(t)}$) is the information that is kept in memory over time.

To train these neurons, we "unroll" them after a complete forward pass to reveal a chain of linked cells the length of time steps t in a single input. We then apply standard backpropagation to these links, calling the process backpropagation through time (BPTT). This works relatively well for very short time series, but once the number of time steps increases to tens or hundreds of time steps, the network essentially becomes very deep during BPTT and problems arise such as very slow training and exploding and vanishing gradients [48]. Various hyperparameter and regularization schemes exist to alleviate exploding/vanishing gradients, including *gradient clipping* [53], *batch normalization*, dropout, and the long short-term memory (LSTM) cell originally developed by Sepp Hochreiter and Jurgen Schmidhuber in 1997 [42].

Long Short-Term Memory (LSTM) RNNs. The LSTM cell achieves faster training and better long-term memory than vanilla RNN neurons by maintaining two state vectors, the short-term state $h_{(t)}$ and the long-term state $c_{(t)}$, mediated by a series of inner gates, layers, and other functions. These added features allow the cell to process the time series in a deliberate manner, recognizing meaningful input to store long-term and later extract when needed, and forget unimportant information or that which is no longer needed [48].

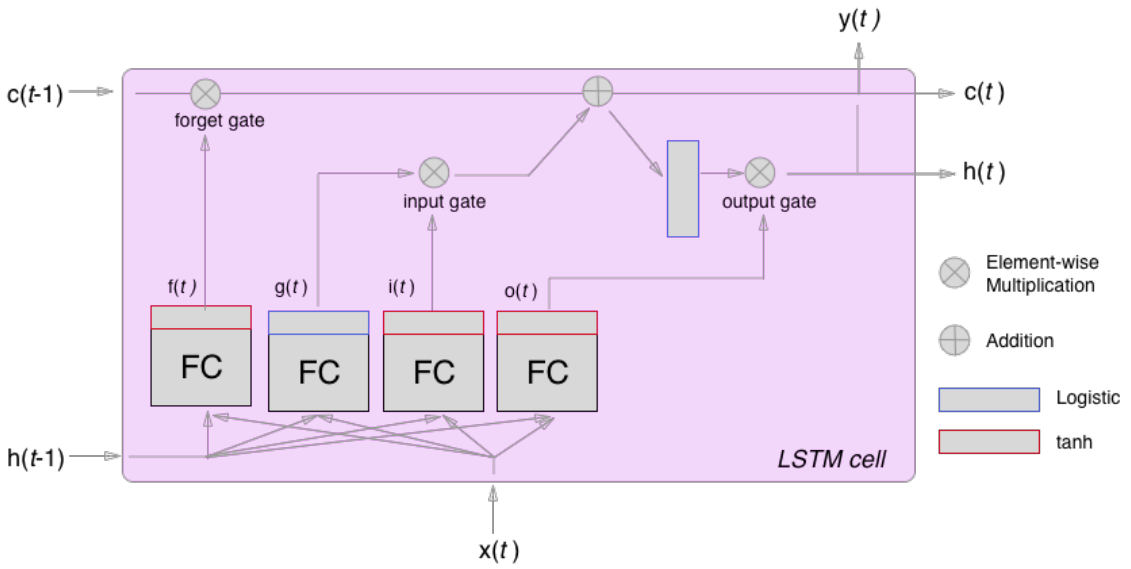

Figure 2.3: The inner mechanisms of an LSTM cell. From outside the cell, information flows similarly as with a vanilla recurrent cell, except that the state now exists as two parts, one for long-term memory ($c_{(t)}$) and the other for short-term memory ($h_{(t)}$). Inside the cell, four different sub-layers and associated gates are revealed.

As can be seen in Figure 2, when the forward pass advances by one time step, the new time step's input enters the LSTM cell and is copied and fed into four independent fully-connected layers (each with its own weight matrix and bias vector), along with the short-term state from the previous time step, $h_{(t-1)}$. The main layer is $g_{(t)}$, which processes the inputs via $tanh$ activation function. In the basic recurrent cell, this is sent straight to the output; in the LSTM cell, part of this is incorporated in the long-term memory as decided by the *input gate*. The input gate also takes input from another layer, $i_{(t)}$, which processes the inputs via the sigmoid activation function σ (as do the next two layers). The third layer, $f_{(t)}$, processes the inputs, combines them with $c_{(t-1)}$, and passes this combination through a *forget gate* which drops a portion of the information therein. Finally, the fourth fully-connected layer $o_{(t)}$ processes the inputs and passes them through the *output gate* along with a copy of the updated long-term state $c_{(t)}$ after its additions from $f_{(t)}$, deletions by the forget gate, further additions from the filtered $g_{(t)}$-$i_{(t)}$ combination and a final pass through

a $tanh$ activation function. The information that remains after passing through the output gate continues on as the short-term state $h_{(t)}$.

$$i_{(t)} = \sigma(W)xi \cdot x_{(t)} + W_{hi} \cdot h_{(t-1)} + b_i$$

$$f_{(t)} = \sigma(W)xf \cdot x_{(t)} + W_{hf} \cdot h_{(t-1)} + b_f$$

$$o_{(t)} = \sigma(W)xo \cdot x_{(t)} + W_{ho} \cdot h_{(t-1)} + b_o$$

$$g_{(t)} = \sigma(W)xg \cdot x_{(t)} + W_{hg} \cdot h_{(t-1)} + b_g$$

$$c_{(t)} = f_{(t)} \otimes c_{(t-1)} + i_{(t)} \otimes g_{(t)}$$

$$y_{(t)} = h_{(t)} = o_{(t)} \otimes \tanh(c_{(t)})$$

where $\otimes$ represents element-wise multiplication [48].

Related Works

The following section outlines the nuanced hyperparameter combinations used by 30 studies available in literature in a meta-analysis style survey. Published works as well as pre-published and academic research projects were included so as to gain insight into the state-of-the-art methodologies at all levels and increase the volume of works available for review. It should be noted that the following summaries are not necessarily entirely exhaustive regarding the specifications listed. Additionally, many reports did not include explicit details of many aspects of their research.

The survey of previous experiments in this field provided blueprints for constructing an adequate search space of hyperparameters. We have held our commentary on the findings of this meta-study until the Discussion section.

Experimental Setup. Across the 30 studies, each used a unique implementation of LSTMs for the research conducted therein. Data sets used include the OPPORTUNITY Activity Recognition dataset [32, 54, 33, 55, 56, 57], UCI HAR dataset [58, 55], PAMAP2 [32, 59, 57, 60], Skoda [32, 57], WISDM [61, 58], and various study-specific and/or internally-collected datasets [62]. Activity classes include Activities of Daily Life (ADL; e.g., opening a drawer, climbing stairs, walking, or sitting down), smoking [63], cross-country skiing [37], eating [64], nighttime scratching [65], driving [66], and so on.

Data analysis pipelines employed include cross validation [67], repeating trials [68], and various train-validation-test splitting procedures [36, 69, 70]. Most studies used the Python programming language and implemented LSTMs via third-party libraries such as Theano Lasagne, RNNLib, and Keras with TensorFlow.

Preprocessing. Some reports kept preprocessing to a minimum, e.g., linear interpolation to fill missing values [32], per-channel normalization [32, 70], and standardization [61, 55]. Zhao, et. al. standardized the data to have 0.5 standard deviation [55] as opposed to the typical unit standard deviation, citing Wiesler, et. al. as supporting this nuance for deep learning implementations [71].

More advanced noise reduction strategies include kernel smoothing [33], removing the gravity component [65], applying a low-pass filter [67], removing the initial and last 0.5 seconds [70]. Moreau, et. al. grouped together segments of data from different axes, tracking the dominant direction of motion across axes [65].

For feeding the data into the models, the sliding window technique was commonly used, with window sizes ranging from 32 [62] to 5000 [55] milliseconds (ms); typically 50% of the window size was used as the step size [37, 36, 56, 32]. Guan and Plotz ran an ensemble of models, each using a random sampling of a random number of frames with

varying sample lengths and starting points. This method is similar to the bagging scheme of random forests and was implemented to increase robustness of the model [57].

Architectures. Numerous architectural and hyperparameter choices were made among the various studies. Most studies used two LSTM layers [32, 61, 64, 54, 58, 55, 57, 70, 62], while others used a single layer [69, 56, 68, 66, 72, 60, 39], three layers [72], or four layers [73].

The number of units (i.e., nodes, LSTM cells) per layer range from 3 [65] to 512 [59]. Several studies used different numbers of units for different circumstances e.g., three units per layer for unilateral movement (one arm) and four units per layer for bilateral movement (both arms) [65] or 28 units per layer for the UCI HAR dataset (lower dimensionality) versus 128 units per layer for the Opportunity dataset [55]. Others used different numbers of units for different layers of the same model e.g., 14-14-21 for a 3-layer model [72].

Almost all of the reports used the sigmoid activation for the recurrent connections within cells and the tanh activation function for the LSTM cell outputs, as these are the activation functions used the original paper [42]. Other activation functions used for the cell outputs include ReLU [55, 70] and sigmoid [60].

Several studies designed or utilized novel LSTM architectures that went beyond the simple tuning of hyperparameters. Architectures tested include the combination of CNNs with LSTMs such as ConvLSTM [33], DeepConvLSTM [32, 36, 56], and the multivariate fully convolutional LSTM network (MLSTM-FCN) [74]; innovations regarding the connections between hidden units including the bidirectional LSTM (b-LSTM) [37, 56, 65, 67, 51], hierarchical b-LSTM [75], deep residual b-LSTM (deep-res-bidir LSTM) [55], and LSTM with peephole connections (p-LSTM) [37]; and other nuanced architectures such as ensemble deep LSTM [57], weighted-average spatial LSTM (WAS-LSTM) [60], deep-Q LSTM [39], the multivariate squeeze-and-excite fully convolutional network AL-STM (MALSTM-FCN) [74], and similarity-based LSTM [38]. Note that the term deep

indicates the use of multiple layers of hidden connections - generally three or more LSTM layers qualifies as “deep”.

The use of densely-connected layers before or after the LSTM layers was also common. Kyritsis, et. al. added a dense layer with ReLU activation after the LSTM layers, Zhao, et. al. included a dense layer with tanh activation after the LSTMs, and Musci, et. al. used a dense layer before and after its two LSTM layers [64, 72, 62]. The WAS-LSTM, deep-Q LSTM, and the similarity-based LSTM used a combination of dense and LSTM hidden layers.

Training. Weight initialization strategies employed include random orthogonal initialization [32, 36], fixed random seed [59], the Glorot uniform initialization [56], random uniform initialization on [-1, 1] [65], or using a random normal distribution [70]. For mini-batch training, reported batch sizes range from 32 [54, 59] to 450 [63] training examples (e.g., windows) per batch.

Loss functions for monitoring training include categorical cross-entropy [32, 73, 61, 36, 64, 59, 56, 70, 60], F1 score loss [57], mean squared error (MSE) [66], and mean absolute error [72]. During back propagation, various updating rules e.g. RMSProp [32, 59, 56], Adam [73, 64, 56, 70, 60], and Adagrad [68, 51] and learning rates 1e-7 [68], 1e-4 [36, 57], 2e-4 [65], 5e-4 [67], and 1e-2 [32] are used.

Regularization techniques employed include weight decay of 90% [32, 36]; update momentum of 0.9 [65], 0.2 [67], or the Nesterov implementation [68]; dropout (e.g., 50% [32, 36] or 70% [72]) between various layers; batch normalization [55]; or gradient clipping using the norm [55, 70, 60]. Broome chose to test the stateful configuration for its baseline LSTM [56]. In this configuration, unit memory cell weights are maintained between each training example instead of resetting them to zero after each forward pass.

The number of epochs specified ranged from 100 [56] to 10,000 [70]. Many studies chose to use early stopping to prevent overfitting [76]. Various patience schemes, speci-

fying how many epochs with no improvement above a given threshold the model should allow, were chosen.

Performance Measures. Various performance measures were used to assess the performance of the model, including the F1 score - used by most [32, 56, 33, 55, 56], classification error [37], accuracy [36, 59], and ROC [65, 70]. As this overview has shown, there are many different model constructions being employed for HAR tasks. The work by the aforementioned studies as well as others have laid the groundwork for this field of research.

Experimental Setup

We implemented a truncated version of our Pipeline, and have made code available for running the entire Pipeline on the UCI HAR Dataset at

`https://github.com/xtianmcd/accelstm`.

Data. Although many studies use the gyroscope- and magnetometer-supplemented records from complex inertial signals, accelerometer data is the most ubiquitous modality in this field and training models on this data alone helps illuminate the robustness of the model and requires lower computational complexity (i.e., more applicable to online and on-device classifications). As such, this report trains its models on triaxial accelerometer data alone.

The primary dataset used for our experiments is the Human Activity Recognition Using Smartphones Data Set (UCI HAR Dataset) from Anguita, et. al. [77].

UCI HAR Dataset. Classes (6) include walking, climbing stairs, descending stairs, sitting, standing, and laying down. Data was collected from built-in accelerometers and gyroscopes (not used in our study) in smartphones worn on the waists of participants.

A degree of preprocessing was applied to the raw signals themselves by the data collectors. The accelerometer data (recorded at 50Hz) was preprocessed to remove noise by applying a third order low pass Butterworth filter with corner frequecy of 20Hz and a median filter. A second filter was then applied to the total accelerometer signal (T) to remove the

gravity component, leaving the isolated body accelerometer signal (B). The accelerometer signals for both B and T were provided as pre-split single-axis windowed signals divided into separate files; see Figure 2.4 A. Windows contained 2.56 seconds (128 time steps) of data and had a step size of 50% of the window size. A 70:30 train-to-test split was used, splitting one of the participants between the two sets.

Preprocessing. We kept preprocessing to a minimum. We first attempted to undo as much of the preprocessing already performed on the data and reformat the data for feeding it into the network. We did this to establish a baseline format for the data at the start of the Pipeline so that data from different datasets can be used. The code for this procedure can be found in the GitHub repository linked above in the file

`accelstm/src/data/HAR_get_data.py`.

First, we re-combined the training and testing sets (Figure 2.4 B). We effectively removed the windows by concatenating together time points from every other window, reforming contiguous time series (Figure 2.4 C). We then combined each axis-specific time series to form the desired triaxial data format, where each time point consists of the accelerometer values along the x-, y-, and z-axes as a 3-dimensional array (Figure 2.4 D). We generated one-hot labels in that step as well. We kept track of the participant to which each record belonged (Figure 2.4 E) so that no single participant was later included in both training and testing sets.

We used an 80:20 training-to-testing split (Figure 2 A-D), and *subsequently* standardized the data by first fitting the standardization parameters (i.e., mean and standard deviation) to the training data and then using these parameters to standardize the training and testing sets separately (Figure 2 E1). This sequenced procedure prevents exposing any summary information about the testing set to the model before training, i.e., data leakage. Finally, a fixed-length sliding window was applied (Figure 2 E2), the windows were shuffled to avoid localization during training (Figure 2 F), and the data was ready to feed into the LSTM neural network.

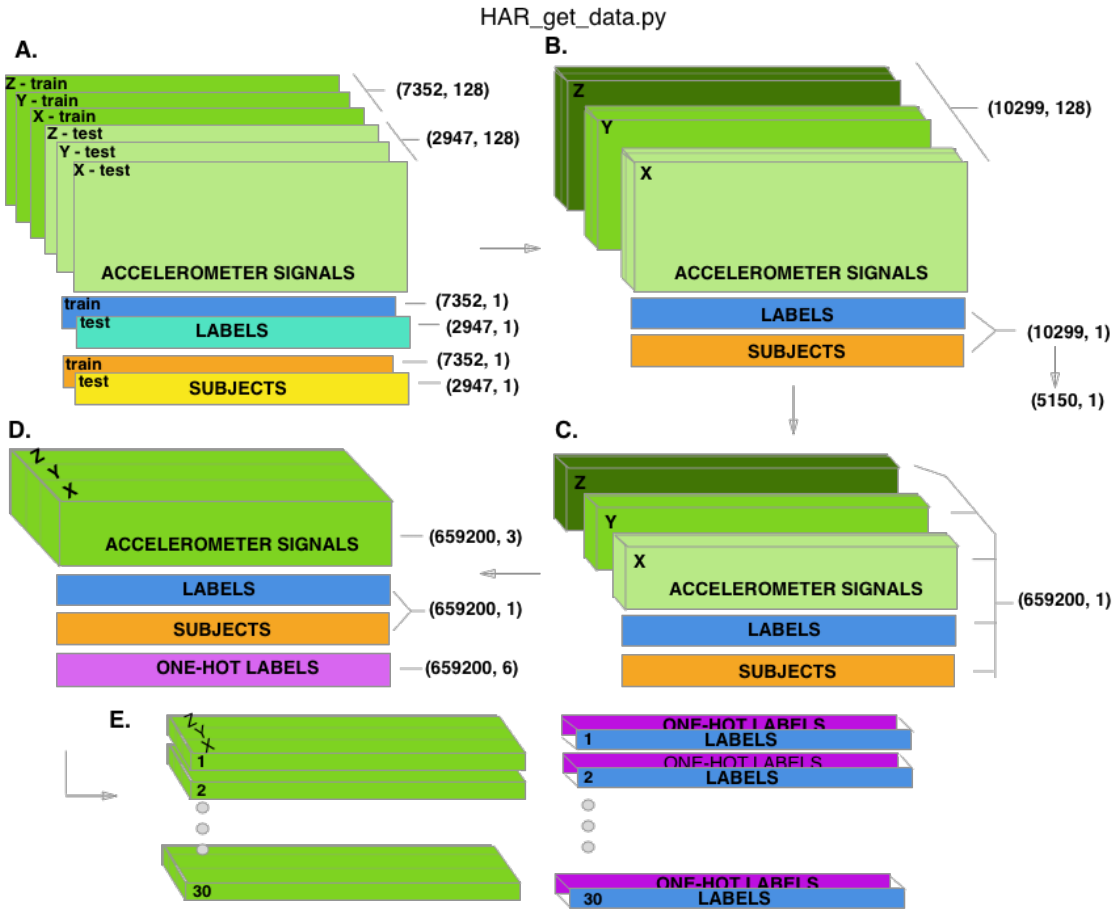

Figure 2.4: Depiction of the "undoing" procedure to return the data in the UCI HAR Dataset to its unprocessed form. **A.** Data is provided as train/test-split single-axis windowed accelerometer signals. **B.** Combine train and test sets. **C.** Remove windows; reformat labels and subject include's accordingly. **D.** Axes are combined into a three-dimensional time series; one-hot labels are generated. **E.** 3-D time series and labels are grouped by subject to emulate subject-wise data acquisition.

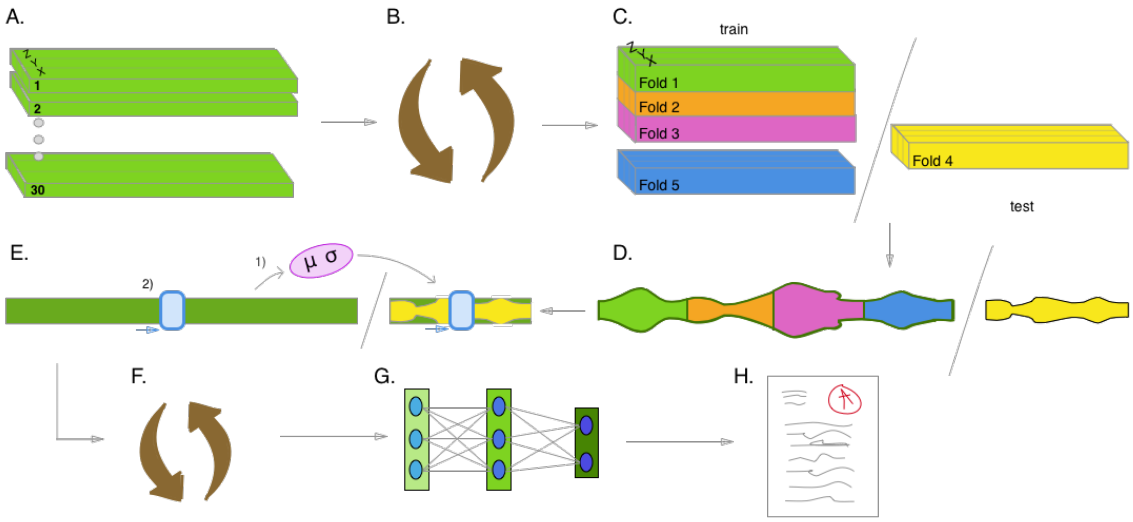

Figure 2.5: Outline of the proposed data analysis pipeline. **A.** The data should start as raw tri-axial data files separated into individual records; one record per individual. **B.** Shuffle the records. **C.** Partition the records into k equal groupings for the k-fold cross validation. **D.** Concatenate the records end-to-end within the train and test sets (for feeding in to the LSTM). **E.** Standardize the data, careful to avoid data leakage; subsequently window the data. **F.** Shuffle the windowed data sets. **G.** If in Part 1 of the Pipeline, optimize the model's hyperparameters; if in Part 2, train the optimized model on the training data. **H.** Predict outcomes for the testing data using the trained model and score the results.

Training. All model training code can be found in the GitHub repository linked above in the folder `accelstm/src/models`. Training the model was broken up into two sections, the first of which consisted of hyperparameter optimization. We employed a heuristic-based search, namely the tree-structured Parzen (TPE) expected improvement (EI) algorithm, in order to more efficiently navigate the vast hyperparameter search space. EI algorithms estimate the ability of a supposed model x to outperform some performance standard y^*, and TPE aims to assist this expectation by heuristically modeling the search space without requiring exhaustive exploration thereof. TPE iteratively substitutes equally-weighted prior distributions over hyperparameters with Gaussians centered on the examples seen over time. This re-weighting of the search space allows TPE to estimate $p(y)$ and $p(x|y)$ - regarding the performance y from suggested model x - ultimately allowing the EI algorithm to estimate $p(y|x)$ of model M via Bayes Theorem [78].

$$EI_{y^*}(x) := \int_{-\infty}^{\infty} \max(y^* - y, 0) p_M(y|x) dy$$

becomes

$$\begin{aligned} EI_{y^*}(x) &= \int_{-\infty}^{y^*} \max(y^* - y, 0) p_M(y|x) dy \\ &= \int_{-\infty}^{y^*} \frac{p(x|y)p(y)}{p(x)} dy \\ &= \frac{\gamma y^* l(x) \int_{-\infty}^{y^*} p(y) dx}{y l(x) + (1-\gamma) g(x)} \\ &\propto (\gamma + \frac{g(x)}{l(x)}(1-\gamma))^{-1} \end{aligned}$$

where

$$\gamma = p(y^* < y)$$

Table 2.1: The hyperparameters included in the search space, and their respective ranges.

Category	Hyperparameter	Range
Data Processing	Window Size	24, 48, 64, 128, 192, 256
	Stride	25%, 50%, 75%
	Batch Size	32, 64, 128, ..., 480
Architecture	Units	2, 22, 42, 62, ..., 522
	Layers	1, 2, 3
Forward Processing	Activation Function (unit, state)	softmax, tanh, sigmoid, ReLU, linear
	Bias	True, False
	Weight Initialization (cell, state)	eros, ones, random uniform dist., random normal dist., constant (0.1), orthogonal, Lecun normal, Glorot uniform
Regularization	Regularization (cell, state, bias, activation)	None, L2 Norm, L1 Norm
	Weight Dropout (unit, state)	uniform distribution (0, 1)
	Batch normalization	True, False
Learning	Optimizers	SGD, RMSProp, Adagrad, Adadelta, Nadam, Adam
	Learning Rate	$10^{-7}, 10^{-6}, 10^{-5}, 10^{-4}, 10^{-3}, 10^{-2}, 10^{-1}$

$$p(x|y) = l(x) \text{ if } y < y^*$$
$$= g(x) \text{ if } y \geq y^*$$

and $p(a|b)$ is the conditional probability of a given event b.

The ranges of hyperparameters were devised to include all ranges explored by the various reports reviewed in the above section of this paper, as well as any other well-defined range or setting used in the field, yielding an immense search space with trillions of possible combinations. The hyperparameters included in the search space are listed in Table **??**. Due to constraints in the Python package used for hyperparameter optimization (i.e., `hyperas` from `hyperopt`), a subsequent tuning of the window size, stride length and number of layers needed to be performed on the highest performing combination of all other hyperparameters via randomized grid search. This step was omitted in the current proof of concept experiment, but the code for carrying out the grid search can be found in the file `accelstm/src/models/more_opt.py`. Thus, for initial optimization and the final cross validation (detailed below), data was partitioned using a window size of 128 with 50% stride length and fed into a 2-layer LSTM network.

For the second portion of the experiment, the Pipeline is completed via 5-fold cross validation, where the folds were made at the participant level so that no single participant's data ended up in both training and testing sets.

Languages and Libraries. All models were written in the Python programming language. The LSTMs were built and run using the Keras [2] library and TensorFlow [3] as the backend heavy lifter. Hyperas [4] from Hyperopt [5] was used to optimize the network. Scikit learn provided the packages for cross validation, randomized grid search, and standardization of data. Numpy and Pandas were used to read and reformat the data among various other operations.

Results

During preliminary testing, we found that the model performed better on the total raw accelerometer signal (T) compared to the body-only data with the gravity-component (B) removed. As such, we used the total accelerometer signal (T) in our experiment.

The hyperparameter optimization explored a search space with trillions of possible parameter combinations. Due to time constraints, we stopped the search after six full days (hundreds of training iterations), during which time the suggested models' accuracies on test sets had ranged from 12.66% to 94.96%. The algorithm found several high-performing models and had used at least once all the values possible for each activation function, initialization strategy, regularization strategy, learning rate, and optimizer in the search space. The algorithm had tested models that both used and omitted batch normalization and bias, and it had tested dropout values between 0.005 and 0.991, batch sizes between 35 and 441 samples per batch, and from 10 to 508 units at both of the two layers.

[2] https://keras.io
[3] https://www.tensorflow.org
[4] https://github.com/maxpumperla/hyperas
[5] http://hyperopt.github.io

Table 2.2: Results table including results from baseline LSTM models trained on all 9 features in the data set - total accelerometer signals (T), body accelerometer signals (gravity component removed, B), gyroscope signals (G). One of the baseline LSTM's did not explicitly specify the number of features used. We provide results from Part 1 (P1, Hyperparameter Optimization) and Part 2 (P2, Cross-Validation) of our Pipeline. P2 scores include Accuracies (percentages) and F1 scores (decimals).

Model	**Performance**	**Num. Features**
Baseline LSTM 1	90.77%	9 (T,B,G)
Baseline LSTM 2	85.35%	3-9 (?)
Pipeline P1 (Best)	93.47%	3
Pipeline P2 (CV)	90.97%	3
	0.910	3
Pipeline P2 (Best)	95.25%	3
	0.957	3

Due to limited time to run our experiments, we conducted part two of the experiment concurrently with part one using a baseline LSTM architecture we felt would be a good starting point based on notes throughout the literature. The hyperparameter settings used in the model are as follows: window size, 128 time steps; step size, 50% of window size; number of layers, 2; units (layer1), 128; units (layer2), 114; batch size, 64; cell activation, tanh; recurrent activation, sigmoid; dropout, 0.5; weight initialization, Glorot Uniform; regularization, None; optimizer, RMSProp; bias, yes. We ran 5-fold CV on the model and computed the overall and class-wise F1 scores and accuracies. Cross validation yielded an average accuracy of 90.97% and F1 score of 0.90968, with a single best run of 95.25% accuracy and 0.9572 F1 score. We include the single best run for comparison with other reports, many of which do not report evidence of using cross validation or repeated trials.

Discussion

The execution of HAR research in various settings from the biomedical clinic early on [79, 30, 80] to current-day innovative settings such as the automobile [66], the bedroom [65], the dining room [64], and outdoor sporting environments [37] justifies the time spent expand-

ing this area of research. As LSTM models are increasingly demonstrated to have potential for HAR research, the importance of deliberate and reproducible works is paramount.

Review of Previous Works. A survey of the literature revealed a lack of cohesiveness regarding the use of LSTMs for accelerometer data and the overall data analysis pipeline. We grew concerned with possible sources of data leakage. Test set data should come from different participants than those used for the training data [5], and no information from the test set should be exposed to the model before training.

We were surprised to see some of the more advanced preprocessing techniques being employed. Much of the appeal of non-linear models such as neural networks is their ability to learn from raw data itself and independently perform smoothing and feature extraction on noisy data through parameterized embeddings of the data. For example, Karpathy's 2015 study of LSTMs for language modeling showed specific neurons being activated when quotes were opened and deactivated when the quotes were closed, while others were activated by parenthetical phrases, marked the end of sentences, and so on [15]. Additionally, these preprocessing methods are more computationally expensive and less realistic for online and on-device implementations than is desired. The improved performance of the model on the total accelerometer signal (T) versus the body-only signal (B) with the gravity component removed demonstrates the promising potential of non-linear data-dependent models for classifying complex noisy data and supports our claim that extensive preprocessing is not necessary.

We do feel standardization is justified for this data due to its complexity and poor signal-to-noise ratio. Standardization is often important for data-dependent models such as LSTMs since the presence of outliers and skewed distributions may distort the weight embeddings [76].

Hyperparameter Optimization and Data Analysis Pipeline. We structured our experiments with the objective of maintaining simplicity, relying as much as possible on the

baseline model itself, maximizing generalizability and reproducibility of our methods and results, and unifying the existing methods and results in literature.

We saw very promising results from the hyperparameter optimization portion of the experiment. The TPE algorithm, although not run to completion in this experiment, was able to navigate the search space and find several well-performing models. We chose to err on the side of caution by using very granular ranges over the numerical hyperparameters, and as a result we ran out of time even using the heuristic-based TPE algorithm. We suggest further experiments to reduce the search space by using less granular ranges over the numeric hyperparameters, and exploring more advanced heuristic search methods. Doing so will decrease the search time and allow completion of the entire Pipeline in a more reasonable amount of time. Nonetheless, the TPE's so-far-best model at the time of termination and our baseline model from Part 2 outperformed other baseline LSTMs trained on higher dimensional data from the same dataset [58, 55]; see Table **??**.

We also compare our performance with other benchmark experiments on the UCI HAR dataset. Compared with more complex LSTMs trained using more features, our averaged cross validation results scored competitively with the b-LSTM (91.09%), the residual LSTM (91.55%), and the deep res-bidir-LSTM (93.57%) all from Zhao, et. al. [55]. As we found no evidence of cross validation in these other reports, we compare our single best-performing test's accuracy of 95.25% and F1 score of 0.9572 and find it to compete with the highest scoring models found in literature: 4 layer LSTM (96.7% accuracy, 0.96 F1score) [73], MLSTM-FCN and MALSTM-FCN (96.71% accuracy) [74], and one-vs-one (OVO) SVM (96.4% accuracy, 551 features) [25].

Conclusions and Future Work

We demonstrate the ability for a baseline LSTM model trained solely on raw triaxial accelerometer data (without gravity component removed) to perform competitively with

classical models trained on hundreds of hand-crafted features and with other more complex LSTM models trained on higher dimensional sensor data.

We demonstrate the ability to optimize a data-centric model over an expansive hyperparameter search space and train it end-to-end within a scientifically rigorous and deliberate Data Analysis Pipeline. The code used in this project can be found at

`https://github.com/xtianmcd/accelstm`.

Going forward, we would like to repeat this experiment to average performances from different models returned by the TPE algorithm; we would also like to repeat this experiment on other HAR datasets. Further exploration should be done to analyze why the algorithms selections are indeed superior, how different data affect these choices, and how the LSTM cells within the models themselves are representing this type of data as has been done with LSTMs in other domains.

We hope that this Pipeline will serve useful in producing explicit and reproducible experiment results and in pushing the field forward in a methodical way.

CHAPTER 3

DEVELOPING A GRAPH CONVOLUTION-BASED ANALYSIS PIPELINE FOR MULTI-MODAL NEUROIMAGE DATA: AN APPLICATION TO PARKINSON'S DISEASE

[1]

[1] McDaniel, C.L. and S. Quinn. 2019. *Proceedings of the 18th Python in Science Conference (SciPy 2018)* 31-40. Reprinted here with permission of the publisher.

Abstract

Parkinson's disease (PD) is a highly prevalent neurodegenerative disease originating in subcortical areas of the brain and resulting in progressively worsening motor, cognitive, and psychiatric (e.g., depression) symptoms. Neuroimage data is an attractive research tool given the neurophysiological origins of the disease. Despite insights potentially available in magnetic resonance imaging (MRI) data, developing sound analytical techniques for this data has proven difficult. Principally, multiple image modalities are needed to compile the most accurate view possible; the process of incorporating multiple image modalities into a single holistic model is both poorly defined and extremely challenging. In this paper, we address these issues through the proposition of a novel graph-based convolutional neural network (GCN) architecture and present an end-to-end pipeline for preprocessing, formatting, and analyzing multimodal neuroimage data. We employ our pipeline on data downloaded from the Parkinson's Progression Markers Initiative (PPMI) database. Our GCN model outperforms baseline models, and uniquely allows for direct interpretation of its results.

Introduction

Affecting more than 1% of the United States population over the age of 60, Parkinson's disease (PD) is the second-most prevalent age-related neurodegenerative disease following Alzheimer's disease [19]. PD diagnosis has traditionally relied on clinical assessments with some degree of subjectivity [20], often missing early-stage PD altogether [21]. Benchmarks for delineating PD progression or differentiating between similar conditions are lacking [22, 23]. As such, many efforts have emerged to identify quantitatively rigorous methods through which to distinguish PD.

Neuroimage data is an attractive tool for PD research. Magnetic resonance imaging (MRI) in particular is safe for patients, highly diverse in what it can capture, and decreasing

in cost to acquire. Recent work shows that multiple MRI modalities are required to provide researchers and clinicians with the most accurate view of a patient's physiological state [81, 21, 23]. For example, anatomical MRI (aMRI [2]) data is useful for identifying specific brain regions, but the Euclidean distance between regions does not well-approximate the functional or structural connectivity between them. Diffusion-weighted MRI (dMRI) measures the flow of water through the brain in order to track the tube-like connections between regions (i.e., tracking *nerve fiber bundles* a.k.a. *tracts* via white matter *tractography*; see Appendix A below for more information), and functional MRI (fMRI) measures changes in blood oxygenation throughout the brain over time to approximate which regions of the brain function together. As such, it is useful to analyze a combination of these modalities to gain insights from multiple measures of brain physiology. Processing and analyzing multimodal data together is both poorly defined and extremely challenging, requiring combined expertise from neuroscience and data analytics.

MRI data is inherently noisy data and requires extensive preprocessing before analysis can be performed. This is often left to the researcher to carry out; many techniques exist, and the technical implementation decisions made along the way can affect the outcome of later analysis. This is a major barrier to reproducibility and prevents data analysts from applying their skills in this domain. More work is needed to automate the procedure and provide better documentation for steps requiring case-specific input. To that end, we discuss our findings and methods below, and our code is available on GitHub [3].

Following preprocessing, we address the issue of analyzing multimodal MRI data together. Previous work has shown that graph-based signal processing techniques allow multimodal analysis in a common data space [82, 83, 84]. It has been shown that graph-based signal processing classifiers can be incorporated in neural network-like architectures and

[2] In this paper we use anatomical MRI to refer to standard *T1-weighted* (T1w) MR imaging. T1 weighted refers to the specific sequence and timing of magnetic pulses and radio frequencies used during imaging. T1w MRI is a common MR imaging procedure; the important thing to note is that T1 weighting yields high-resolution images which show contrast between different tissue types, allowing for segmentation of different anatomical regions.

[3] `https://github.com/xtianmcd/GCNeuro`

applied to neuroimage data. Similar to convolutional neural networks, Graph Convolutional Networks (GCNs) learn *filters* over a graph so as to identify patterns in the graph structure, and ultimately perform classification on the nodes of the graph. In this paper, following the discussion of our preprocessing pipeline, we propose a novel GCN architecture which uses graph attention network (GAT) layers to perform whole-graph classification on graphs formed from multimodal neuroimage data.

On data downloaded from the Parkinson's Progression Markers Initiative (PPMI), we compare the performance of the novel GCN architecture to that of baseline models. We find that our GCN model outperforms baseline models on our data. The weights from GAT layers provide a means for direct interpretation of the results, indicating which brain regions contributed the most to the distinction between patients with PD and healthy controls.

Related Works

While genetic and molecular biomarkers have exhibited some efficacy in developing a PD blueprint [20, 85, 86], many research efforts have turned to neuroimaging due to its non-invasive nature and alignment with existing knowledge of the disease. Namely, PD affects a major dopamine-producing pathway (i.e., the nigrostriatal dopaminergic pathway) of the brain [26], and results in various structural and functional brain abnormalities that can be captured by existing imaging modalities [87, 85, 88, 89, 90, 28]. Subsequent whole-brain neuroimage analysis has identified PD-related regions of interest (ROIs) throughout the brain, from cortical and limbic regions to the brainstem and cerebellum [91, 89, 28].

As neuroimage data has accumulated, researchers have worked to develop sound analytical techniques for the complex images. Powerful machine learning techniques have been employed for analyzing neuroimage data [85, 89, 91, 90], but algorithmic differences can result in vastly different results [88, 92, 87]. [27] and [28] found that implementation choices made during the processing pipeline can affect analysis results as much as anatomical differences themselves (e.g., when performing white matter tractography on

diffusion-weighted MRI (dMRI) data and in group analysis of resting-state functional MRI (rfMRI) data, respectively). To overcome the effect of assumptions made by a given analysis algorithm, many researchers have turned to applications of deep machine learning (DL) for neuroimage data analysis. Considered universal function approximators [93], DL algorithms are highly flexible and therefore have low bias in their modeling behavior. Examples of DL applications to neuroimage analysis are widespread. [94] proposes a 3D convolutional neural network (CNN) for skull stripping 3D brain images, [95] proposes a novel recurrent neural network plus independent component analysis (RNN-ICA) model for fMRI analysis, and [96] demonstrate the efficacy of the restricted Boltzmann machine (RBM) for network identification. [4] offer a comprehensive review of deep learning-based methods for medical image computing.

Multi-modal neuroimage analysis is increasing in prevalence [97, 81, 21, 22, 23] due to limitations of single modalities, resulting in larger and increasingly complex data sets. Recently, researchers have utilized advances in graph convolutional networks to address these concerns. We discuss the mathematical background of graph convolutional networks (GCNs) and graph attention networks (GATs, a variant of GCNs with added attention mechanisms) in the Methods Section below and Appendix B. Principally, our model is based on advancements made by [98] and [99] on GCNs and GATs, respectively.

This work follows from previous efforts applying GCNs to similar classification tasks. [100] — in addition to providing in-depth intuition behind spectral graph processing (i.e., processing a signal defined on a graph structure) — demonstrate spectral graph processing on diffusion signals defined on a graph of connected brain regions. Their paper preceded but laid the groundwork for incorporating spectral graph processing into convolutional neural network architectures. To classify image objects based on multiple views or angles, [101, 102] developed siamese and multi-view neural networks. These architectures share weights across parallel neural networks to incorporate each view of the data. They group

examples into pairs, aiming to classify the pairs as being from the same class or different classes.

Efforts to utilize GCNs for multimodal neuroimage data have used similar pairwise grouping as a way to increase the size of their data set. Ktena, *et. al.* 2017 and Ktena, *et. al.* 2018 train GCN models to learn similarity metrics between subjects with Autism Spectrum Disorder (ASD) and healthy controls (HC), using fMRI data from the Autism Brain Imaging Data Exchange (ABIDE) database [82, 83]. Zhang, *et. al.* 2018 apply a similar architecture to learn similarity metrics between subjects with PD and HC, using dMRI data from the PPMI data set [84]. Their work inspired our paper; to our knowledge, our study is the first that uses GCNs to predict the class of neuroimage data directly, instead of making predictions on pairwise examples.

Discussion of the Processing Pipeline

This section walks through our pipeline, which handles the formatting and preprocessing of multimodal neuroimage data and readies it for analysis via our GCN architecture. We reference the specific python files that handle each task, and we provide some background information. More information can be found in the Appendices below.

Data Formatting MRI data requires extensive artifact correction and removal before it can be used. MRI signals are acquired through the application of precisely coordinated magnetic fields and radiofrequency (RF) pulses. Each image is reconstructed from a series of recordings averaged over many individual signals. This inherently results in noisy measurements, magnetic-based artifacts, and artifacts from human error such as motion artifacts [103, 104]. As such, extensive preprocessing must be performed to clean the data before analysis. Appendix A provides more details on the main MRI modalities.

Our pipeline assumes that a "multi-zip" download is used to get data from the PPMI database [4]. The file `neuro_format.py` combines the data from multiple download folders into a single folder, consolidating the multiple zip files and recombining data from the same subject.

Next, before preprocessing, images should be converted to the Neuroimaging Informatics Technology Initiative (NIfTI) [5] file format. Whereas many MRI data are initially in the Digital Information and Communications in Medicine (DICOM) [6] format for standardized transfer of medical data and metadata, the NIfTI format is structured for ease of use when conducting computational analysis and processing on these files. The size, orientation, and location in space of the voxel data is dependent on settings used during image acquisition and requires an *affine matrix* to relate two images in a standard coordinate space. The NIfTI file format automatically associates each image with an affine matrix as well as a *header file*, which contains other helpful metadata. The software `dcm2niix` [7] is helpful for converting the data from DICOM format to NIfTI format.

Next, it is common practice to convert the data file structure to the Brain Imaging Data Structure (BIDS) [8] format. Converting data to the BIDS format is required by certain softwares, and ensures a standardized and intuitive file structure. There exist some readily available programs for doing this, but we wrote our own function specifically for PPMI data in `make_bids.py`, as the PPMI data structure is quite nuanced. This file also calls `dcm2niix` to convert the image files to NIfTI format.

Data Preprocessing. This subsection discusses the various softwares and commands used to preprocess the multimodal MRI data. The bash script `setup` should help with getting the necessary dependencies installed [9]. The script was written for setting up a

[4]When using the "Advanced Download" option on the PPMI database, the data is split into multiple zip files, often splitting up the data of a single subject.

[5]`https://nifti.nimh.nih.gov`

[6]`https://www.dicomlibrary.com`

[7]`https://github.com/rordenlab/dcm2niix`

[8]`https://bids.neuroimaging.io`

[9]We install the softwares to the home (‘ ‘) directory due to permission issues when connect to Google cloud virtual machines via the ‘ssh‘ command. Freesurfer's setup does not automatically adapt to installation

Google cloud virtual machine, and assumes the data and pipeline files are already stored in a Google cloud bucket.

The standard software for preprocessing anatomical MRI (aMRI [10]) data is Freesurfer [11]. Freesurfer is an actively developed software with responsive technical support and rich forums. The software is dense and the documentation is lacking in some areas, so training may still be helpful, although not available in our case. The `recon-all` command performs all the steps needed for standard aMRI preprocessing, including motion correction, registration to a common coordinate space using the Talairach atlas by default, intensity correction and thresholding, skull-stripping, region segmentation, surface tessellation and reconstruction, statistical compilation, etc.

The entire process takes around 15 or more hours per image. Support for GPU-enabled processing was stopped years ago, and the `-openmp<num_cores>` command, which allows parallel processing across the designated number of cores, may only reduce the processing time to around 8-10 hours per image [12]. We found that running parallel single-core CPU processes worked the best, especially when many processing cores are available. For this we employed a Google Cloud Platform virtual machine and utilized the python module `joblib.Parallel` to run many single-core processes in parallel. For segmentation, the Deskian/Killiany atlas is used, resulting in around 115 volume segmentations per image, to be used as the nodes for the graph.

in the home directory, so several of its environment variables need to be hard coded. See the 'setup' bash script provided for details.

[10] In this paper we use anatomical MRI to refer to standard *T1-weighted* (T1w) MR imaging. T1 weighted refers to the specific sequence and timing of magnetic pulses and radio frequencies used during imaging. T1w MRI is a common MR imaging procedure; the important thing to note is that T1 weighting yields high-resolution images which show contrast between different tissue types, allowing for segmentation of different anatomical regions.

[11] `https://surfer.nmr.mgh.harvard.edu`

[12] In the release notes, it is recommended for multi-subject pipelines to use a single core per image and process subjects in parallel, and in the forums it is discussed that multiprocessing may only reduce the processing time to around 10 hours. It is also mentioned that the time required to transfer data on and off GPU cores may diminish the speedup provided by GPU processing. GPU support has not been provided by Freesurfer for quite some time, and we were unable to compile Freesurfer to use newer versions of CUDA. We tested multiple CPU multiprocessing approaches and found that running images in parallel with a single core per process was the fastest method.

The Functional Magnetic Resonance Imaging of the Brain (FMRIB) Software Library (FSL) [13] is often used to preprocess diffusion data (dMRI). The *b0* volume is taken at the beginning of dMRI acquisition and is used to align dMRI images to aMRI images of the same subject. This volume is isolated (`fslroi`) and merged with *b0*'s of other clinic visits (CVs) [14] for the same subject (`fslmerge`). `fslmerge` requires that all dMRI acquisitions for a given subject have the same number of coordinates (e.g., `(116,116,78,65)` vs. the standard `(116,116,72,65)`). Since some acquisitions had excess coordinates, we manually examined these images and, if possible, removed empty space above or below the brain. Otherwise, these acquisitions were discarded. Next, the brain is isolated from the skull (*skull stripped*, `bet` with the help of `fslmaths-Tmean`), magnetic susceptibility correction is performed *for specific cases* (see below) using `topup`, and eddy correction is performed using `eddy_openmp`. Magnetic susceptibility and eddy correction refer to specific noise artifacts that significantly affect dMRI data.

The `topup` tool requires two or more dMRI acquisitions for a given subject, where the image acquisition parameters `TotalReadoutTime` and/or `PhaseEncodingDirection` (found in the image's header file) differ from one another. Since the multiple acquisitions for a given subject typically span different visits to the clinic, the same parameters are often used and `topup` cannot be utilized. We found another software, BrainSuite [15], which can perform magnetic susceptibility correction using a single acquisition. Although we still include FSL in our pipeline since it is the standard software used in many other papers, we employ the BrainSuite software's Brain Diffusion Pipeline to perform magnetic susceptibility correction and to align the corrected dMRI data to the aMRI data for a given subject (i.e., *coregistration*).

[13] `https://fsl.fmrib.ox.ac.uk/fsl/fslwiki`

[14] Each subject has anatomical and diffusion MRI data for varying numbers of visits to the clinic. We use clinic visit or CV to refer to the MRI acquisitions (anatomical and diffusion) obtained during a single visit to the clinic.

[15] `http://brainsuite.org`

First, a BrainSuite compatible brain mask is obtained using `bse`. Next, `bfc` is used for bias field (magnetic susceptibility) correction, and finally `bdp` performs co-registration of the diffusion data to the aMRI image of the same subject. The calls to the Freesurfer, FSL, and BrainSuite software libraries are included in `automate_preproc.py`.

Once the data has been cleaned, additional processing is performed on the diffusion (dMRI) data. As discussed in the Introduction section, dMRI data measures the diffusion of water throughout the brain. The flow of water is constricted along the tube-like pathways (tracts) that connect regions of the brain, and the direction of diffusion can be traced from voxel to voxel to approximate the paths of tracts between brain regions. There are many algorithms and softwares that perform tractography, and the choice of algorithm can greatly affect the analysis results. We use the Diffusion Toolkit (DTK) [16] to perform multiple tractography algorithms on each diffusion image. In `dtk.py` we employ four different diffusion tensor imaging (DTI)-based deterministic tractography algorithms: Fiber Assignment by Continuous Tracking (FACT; [105]), the second-order RungeKutta method (RK2; [106]), the tensorline method (TL; [107]), and the interpolated streamline method (SL, [108]). [109] provide more information on each method. `dti_recon` first transforms the output file from Brainsuite into a usable format for DTK, and then the function `dti_tracker` is called for each of the tractography algorithms. Finally, the `spline_filter` function is used to smooth the generated tracts, denoising the outputs. Now that the images are processed, they can be efficiently loaded using python libraries `nibabel` and `dipy`, and subsequently operated on using standard data analysis packages such as `numpy` and `scipy`.

Defining Graph Nodes and Features. Neuroimage data is readily applied to graph processing techniques and is often used as a benchmark application for new developments in graph processing [100]. Intuitively, the objective is to characterize the structural and functional relationships between brain regions, since correlations between PD and abnor-

[16] `http://trackvis.org/dtk/`

mal brain structure and function have been shown. As such, the first step is to define a graph structure for our data. This step alone has intuitive benefits. Even after preprocessing, individual voxels of MRI data contain significant noise that can affect analysis [28]. Brain region sizes vary greatly across individuals and change over one individual's lifetime (e.g., due to natural aging [110]). Representing regions as vertices on a graph meaningfully groups individual voxels and mitigates these potential red herrings from analysis.

We use an undirected weighted graph $\mathcal{G} = \mathcal{V}, \mathcal{E}, \mathbf{W}$ with a set of vertices $\mathcal{V}$ with $|\mathcal{V}| =$ the number of brain regions N, a set of edges $\mathcal{E}$, and a weighted adjacency matrix $\mathbf{W}$, to represent our aMRI data. $\mathcal{G}$ is shared across the entire data set to represent general population-wide brain structure. Each vertex $v_i \in \mathcal{V}$ represents a brain region. Together, $\mathcal{V}, \mathcal{E}$, and $\mathbf{W}$ form a *k-Nearest Neighbor adjacency matrix*, in which each vertex is connected to its *k* nearest neighbors (including itself) by an *edge*, and edges are weighted according to the average Euclidean distance between two vertices. The weight values are normalized by dividing each distance by the maximum distance from a given vertex to all of its neighbors, $d_{ij} \in [0, 1]$. (Refer to Appendix B for details.)

`gen_nodes.py` first defines the vertices of the graph using the anatomical MRI data, which has been cleaned and *segmented* into brain regions by Freesurfer. The center voxel for each segmentation volume in each image is calculated. Next, `adj_mtx.py` calculates the mean center coordinate across all aMRI images for every brain region. The average center coordinate for each region i is a vertex $v_i \in \mathcal{V}$ of the graph $\mathcal{G}$. See Figure 3 for a depiction of the process.

Using these vertices, we wish to incorporate information from other modalities to characterize the relationships between the vertices. We define a *signal* on the vertices as a function $f : \mathcal{V} \to \mathbb{R}$, returning a vector $\mathbf{f} \in \mathbb{R}^N$. These vectors can be analyzed as signals on each vertex, where the change in signal across vertices is used to define patterns throughout the overall graph structure. In our case, the vector signal defined on a vertex v_i represents that vertex's weighted connectivity to all other vertices [100]. The weights

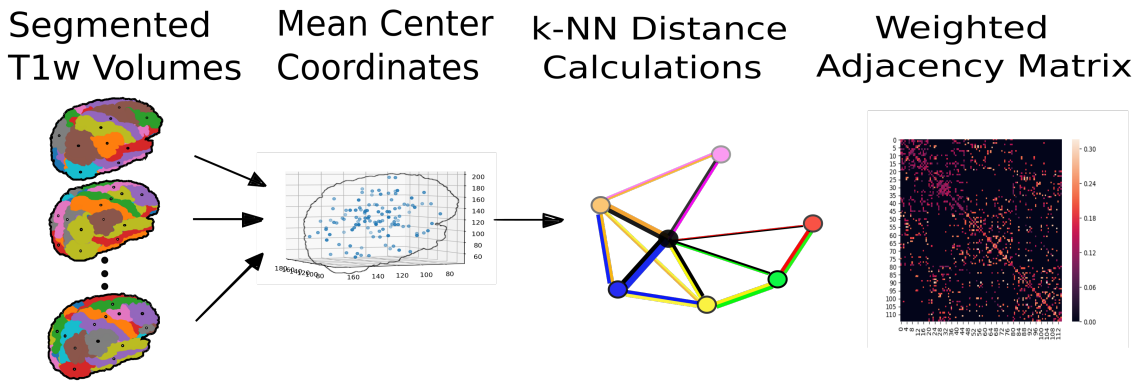

Figure 3.1: A depiction of the steps involved in forming the adjacency matrix. First, anatomical images are segmented into regions of interest (ROIs), which represent the vertices of the graph. The center voxel for each ROI is then calculated. An edge is placed between each node *i* and its *k*-nearest neighbors, calculated using the center coordinates. Lastly, each edge is weighted by the normalized distance between each node *i* and its connected neighbor *j*.

correspond to the strength of connectivity between v_i and some other vertex v_j, as calculated by a given tractography algorithm. As such, each signal is a vertex of size N and there are N signals defined on each graph (one for each vertex), forming an NxN *weighted connectivity matrix*. Each dMRI image has one NxN set of signals for each tractography algorithm. In this way, the dimensionality of the data is drastically reduced, and information from multiple modalities and processing algorithms may be analyzed in a common data space.

`gen_features.py` approximates the strength of connectivity between each pair of vertices. For this, the number of tracts (output by each tractography algorithm) connecting each pair of brain regions must be counted. Recall that each image carries with it an affine matrix that translates the voxel data to a coordinate space. Each preprocessing software uses a different coordinate space, so a new affine matrix must be calculated to align the segmented anatomical images and the diffusion tracts (i.e., *coregistration*). Freesurfer's `mri_convert`, FSL's `flirt`, and DTK's `track_transform` are used to put the two modalities in the same coordinate space so that voxel-to-voxel comparisons can be made. Next, `nibabel`'s i/o functionality is used to generate a mask file for

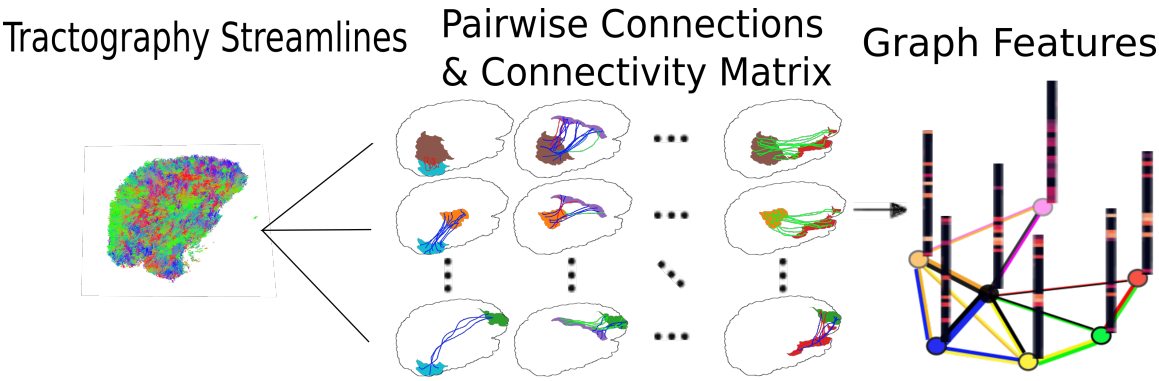

Figure 3.2: The process of generating the features from a single tractography algorithm is shown. Tractography streamlines are aligned to a corresponding anatomical image. The number of streamlines connecting each pair of brain regions is calculated to represent the strength of connection. Using each brain region as a vertex on the graph, the connection strengths between a given vertex to all other vertices are compiled to form the signal vector for that vertex.

each brain region, `nibabel.streamlines` is used to read in the tractography data and `dipy.tracking.utils.target` is used to identify which tracts travel through each volume mask. The tracts are encoded using a unique hashing function to save space and allow later identification.

To generate the signals for each vertex, `utils.py` uses the encoded tract IDs assigned to each volume to count the number of tracts connecting each volume pair. The number of connections between pairs of brain regions approximate the connection strength, and these values are normalized similar to the normalization scheme mentioned above for the k-nearest neighbor weights. Figure 3.2 offers a visualization.

Graph Convolutional Networks. Common to many areas of data analysis, *spectral graph processing* techniques (i.e., processing a signal defined on a graph structure) have capitalized on the highly flexible and complex modeling capacity of so-called deep learning neural network architectures. The layered construction of nonlinear calculations loosens rigid parameterizations of other classical methods. This is desirable, as changes in parameterizations have been shown to affect results in both neuroimage analysis (e.g., independent

component analysis (ICA) [27]) and in graph processing (e.g., the explicit parameterization used in Chebyshev approximation [98]; further discussed in Appendix B).

In this paper, we utilize the Graph Convolutional Network (GCN) to compute signal processing on graphs. GCNs were originally used to classify the vertices of a single graph using a single set of signals defined on its vertices. Instead, our task is to learn signal patterns that generalize over many subjects' data. To this end, we designed a novel GCN architecture, which combines information from anatomical and diffusion MRI (dMRI) data, processes data from multiple diffusion MRI tractography algorithms for each dMRI image, and consolidates this information into a single vector so as to compare many subjects' data side-by-side. A single complete forward pass of our model consists of multiple parallel Graph Convolutional Networks (one for each tractography algorithm), max pooling, and graph classification via Graph Attention Network layers. We will briefly explain each part in this subsection; see Appendix B for a deeper discussion.

The convolution operation measures the amount of change enacted on a function f_1 by combining it with another function f_2. We can define f_2 such that its convolution with instances of f_1 from one class (e.g., PD) produce large changes while its convolution with instances of f_1 from another class (e.g., HC) produce small changes; this provides a way to discriminate instances of 'f_1 into classes without explicitly knowing the class values. Recall that we have defined a function f over the vertices of our graph using dMRI data (i.e., the *signals*). We seek to learn functions, termed *filters*, that, when convolved with the input graph signals, transform the inputs into distinguishable groups according to class value (e.g., PD vs. healthy control). This is similar to the local filters used in convolutional neural networks, except that the filters of GCNs use the connections of the graph (i.e., the edges) to establish locality.

Our specific implementation is based off the `GCN` class from [98]'s PyTorch implementation [17], which has several computational improvements over the original graph convolution formula. In short, the graph convolutional operation is based off the graph Laplacian

$$Ł = I - D^{\frac{-1}{2}}\mathbf{W}D^{\frac{-1}{2}}, \tag{3.1}$$

where I is the identity matrix with 1's along the diagonal and 0's everywhere else, W is the weighted adjacency matrix defined earlier w.r.t. $\mathcal{G}$, and D is a weighted degree matrix such that $D_{ii} = \sum_j \mathbf{W}_{ij}$. We define the graph convolutional operation as

$$Z = \tilde{D}^{\frac{-1}{2}}\tilde{W}\tilde{D}^{\frac{-1}{2}}X\Theta. \tag{3.2}$$

A so-called *renormalization trick* has been applied to Ł wherein identity matrix I_N has been added; i.e., self-loops have been added to the adjacency matrix. $I_N + D^{\frac{-1}{2}}WD^{\frac{-1}{2}}$ becomes '$\tilde{D}^{\frac{-1}{2}}\tilde{W}\tilde{D}^{\frac{-1}{2}}$, where $\tilde{W} = W + I_N$ and $\tilde{D}_{ii} = \sum_j \tilde{W}_{ij}$. $\Theta \in \mathbb{R}^{CxF}$ is a matrix of trainable coefficients, where $C = N$ is the length of the input signals at each node, and $F = N$ is the number of C-dimensional filters to be learned. X is the NxN matrix of input signals for all vertices (i.e., the signals from a single tractography output of a single dMRI image). $Z \in \mathbb{R}^{NxF}$ is the output matrix of convolved signals. We will call the output signals *features* going forward.

Generalizing Θ to the weight matrix $\mathbf{W}(l)$ at a layer l and $X = H(l)$ as the inputs to layer l, where $H(0)$ is the original data, we can calculate a hidden layer of our GCN as

$$Z = f(X, A) = softmax(\hat{A}ReLU(\hat{A}X\mathbf{W}(0))\mathbf{W}(1)), \tag{3.3}$$

where $\hat{A} = \tilde{D}^{\frac{-1}{2}}\tilde{A}\tilde{D}^{\frac{-1}{2}}$.

Multi-View Pooling. For each dMRI acquisition, *d* different tractography algorithms are used to compute multiple views of the diffusion data. To account for the variability in the

[17] `https://github.com/tkipf/pygcn`

outputs produced by each algorithm, we wish to compile the information from each before classifying the whole graph. As such, *d* GCNs are trained side-by-side, such that the GCNs share their weights [101, 82]. This results in *d* output graphs, i.e. *d* output vectors for each vertex. The vectors corresponding to the same vertex are pooled using max pooling, which has been shown to outperform mean pooling [84].

Graph Attention Networks. Recent development of attention-based mechanisms allows for a weighting of each vertex based on its individual contribution during learning, thus facilitating whole-graph classifications. In order to convert the task from classifying each node to classifying the whole graph, the features on each vertex must be pooled to generate a single feature vector for each input. The *self-attention* mechanism, widely used to compute a concise representation of a signal sequence, has been used to effectively compute the importance of graph vertices in a neighborhood [99]. This allows for a weighted sum of the vertices' features during pooling.

Velickovic, *et. al.* 2018 use a single-layer feedforward neural network as an attention mechanism a to compute *attention coefficients e* across pairs of vertices in a graph. For a given vertex v_i, the attention mechanism attends over its first-order neighbors v_j:

$$e_{ij} = a(\mathbf{W}_a h_i, \mathbf{W}_a h_j), \tag{3.4}$$

where h_i and h_j are the features on vertices v_i and v_j, and $\mathbf{W}_a$ is a shared weight matrix applied to each vertex's features. e_{ij} is normalized via the softmax function to compute a_{ij}: $a_{ij} = softmax(e_{ij}) = exp(e_{ij}) / \sum_{k \in \mathcal{N}_i} exp(e_{ik})$, where $\mathcal{N}_i$ is the neighborhood of vertex v_i. The new features at v_i are obtained via linear combination of the original features and the normalized attention coefficients, wrapped in a nonlinearity σ : $h_i\prime = \sigma(\sum_{j \in \mathcal{N}_i} a_{ij} \mathbf{W}_a h_j)$ [99].

Multi-head attention can be used, yielding K independent attention mechanisms that are concatenated (or averaged for the final layer). This helps to stabilize the self-attention learning process.

$$h_i = ||_{k=1}^{K} \sigma(\sum_{j \in \mathcal{N}_i} a_{ij}^k \mathbf{W}_a^k h_j), \tag{3.5}$$

or

$$h_{final} = \sigma(\frac{1}{K} \sum_{k=1}^{K} \sum_{j \in \mathcal{N}_i} a_{jk}^k \mathbf{W}_a^k h_j). \tag{3.6}$$

We employ a PyTorch implementation [18] of [99]'s `GAT` class to implement a graph attention network, learning attention coefficients as

$$a_{ij} = \frac{exp(LeakyReLU(a^T[\mathbf{W}_a h_i || \mathbf{W}_a h_j]))}{\sum_{k \in \mathcal{N}_i} exp(LeakyReLU(a^T[\mathbf{W}_a h_i || \mathbf{W}_a h_k]))}, \tag{3.7}$$

where $||$ is concatenation.

Multi-Subject Training. The model is trained using `train.py`. First, several helper functions in `utils.py` are called to load the graph, input signals, and their labels, and prepare them for training. The model is built and run using the `GCNetwork` class in `GCN.py`. During training, the model reads in the signals for one dMRI acquisition at a time, where the signals from each tractography algorithm are processed in parallel, pooled into one graph, and then pooled into a single feature vector via the graph attention network. Using this final feature vector, a class prediction is made. Once a class prediction is made for every input dMRI instance, the error is computed and the weights of the model are updated through backpropagation. This is repeated over many epochs to iteratively fit the weights to the classification task. Figure 3 shows an outline of the network architecture.

[18] `https://github.com/Diego999/pyGAT`

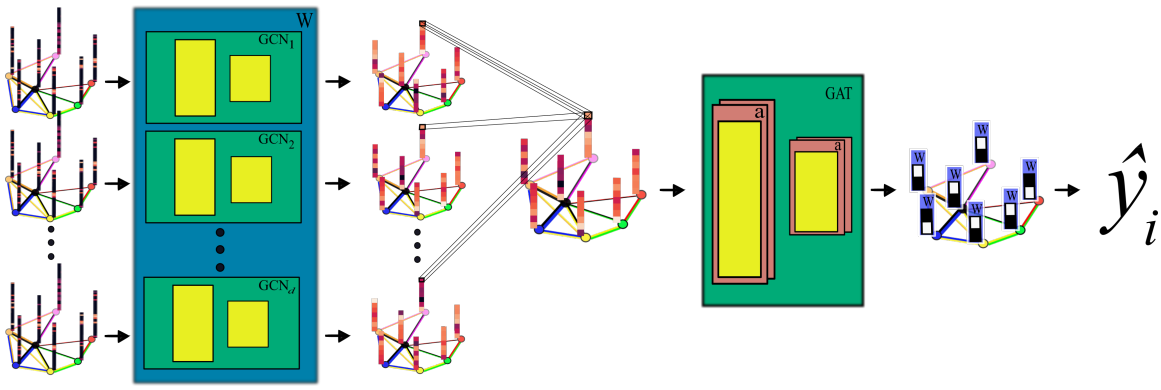

Figure 3.3: A depiction of the novel GCN architecture is shown. First, a GCN is trained for each view of the data, corresponding to a specific tractography algorithm. The GCNs share weights, and the resulting features are pooled for each vertex. This composite graph is then used to train a multi-head graph attention network, which assigns a weight (i.e., attention) to the feature computed at each vertex. The weight assigned to each vertex is used to compute a weighted sum of the features, yielding a single feature vector for graph classification.

Our data is downloaded from the Parkinson's Progression Markers Initiative (PPMI) [19] database. We download 243 images, consisting of 96 aMRI images and 140 diffusion images. The images are from 20 individuals (each subject had multiple visits to the clinic and data from multiple image modalities). Among the images, 117 are from the Parkinson's Disease (PD) group and 30 are from healthy controls (HC). We preprocessed our data using the pipeline described above. We ran this preprocessing using a Google cloud virtual machine with 96 CPU cores over the course of several days.

Following preprocessing, we constructed the shared adjacency matrix and trained the model on the dMRI signals, which totaled to 588 (147 dMRI acquisitions x 4 tractography algorithms) NxN connectivity matrices. We calculated the adjacency matrix using each node's 20 nearest neighbors. To account for the class imbalance between PD and HC images, we use a bagging method. On each of five iterations, all the images from the HC group were combined with an equally-sized subset from the PD group. All of the images were used at least once during training, and the overall performance measures were averaged across training folds.

[19] `https://www.ppmi-info.org`

Using caution to prevent any forms of data leakage, we used a roughly 80/20 train-test split, wherein we ensured all data from the same subject was used as only training or testing data. To assess the performance of our GCN model, we first trained a number of baseline models on the features constructed from the diffusion data. These models include k-nearest neighbor, logistic regression, ridge regression, random forest, and support vector machine (SVM, with both linear and polynomial kernels) from `scikit-learn`; we also trained a fully-connected neural network (fcNN) and a 4-channel convolutional neural network (CNN) using `PyTorch`. Finally, we compare our model to the siamese multi-view GCN (sMVGCN) used in [84]. This network utilizes diffusion and anatomical MRI data and trains on pairs of image data to predict whether the pairs are from the same or different classes. The data is also from the PPMI data set and uses the PD and HC classes during classification. This was the closest model to ours that we found in the literature.

Except for the multi-channel CNN, we trained each model on the features from each tractography algorithm individually, and averaged the results. We calculated the overall accuracy, F1 score, and area under the ROC curve (AUC) as our performance measures. The default parameters were used for the `scikit-learn` models. The fcNN was a three-layer network with two hidden layers. The first layer had 128 ReLU units; the second had 64. For the CNN, a single convolutional layer was used, containing 18 filters of size 3; stride of 1 was used. Max pooling with a kernel size of 2 and stride of 2 was used to feed the features through two fully-connected layers before the final output. The first fully-connected layer reduced the 18x57x57-dimension input — where 57 is the original input width and height of 115 halved via max pooling — to 64 ReLU hidden units. For both neural networks, softmax activation was applied to the outputs and negative log likelihood was used as the loss function (i.e., cross entropy). Again for both models, learning rate was set to 0.01 and dropout of 0.5 was used between fully-connected hidden layers. These parameters coincide with the default parameters of the graph convolutional network class

we used [20], and are commonly used in the literature. We used a validation set to find the optimal number of epochs to train each network for. We tested 40, 80, 100, 140, 200, and 400 epochs for each model and found that 140 worked best for the fcNN, and 100 for the CNN.

We trained the graph convolutional network (GCN) on the same bagged subsets of data for comparison purposes. The only difference is that the features are mapped to the vertices of the adjacency matrix before training. We used a validation set to tune the model parameters. We tested with or without dropout (set to 0.5 when used), with or without weight decay (set to 5e-4 when used), the number of hidden units for the first GCN layer (8,16,32), the number of "heads" or individual attention weights (2,4,6,8), and the number of epochs (same options as for the fcNN and GCN). We found that dropout of 0.5, weight decay of 5e-4, 8 hidden units, 8 attention heads, and 80 epochs worked best for our model. The results from training the GCN are also included in Table 3.1.

Results

The results from training the diffusion data on baseline models, and the combined diffusion and anatomical data on the GCN are included in Table 3.1. We report accuracy, F1-score, and AUC for each model; these numbers are averaged across five training iterations using subsets of the data to account for class imbalance. Subsequently, we analyze the attention weights from the GCN model. Each node of the adjacency matrix was assigned an attention weighting corresponding to that nodes importance in determining the overall class of the graph. Since each node of the adjacency matrix corresponds to an anatomical brain region, we could interpret the magnitude of each nodes attention weight as the relative importance of a brain region for distinguishing the PD vs. HC classes. We compiled the attention weights from each training iteration and determined the 16 brain regions with the highest

[20] https://github.com/tkipf/pygcn

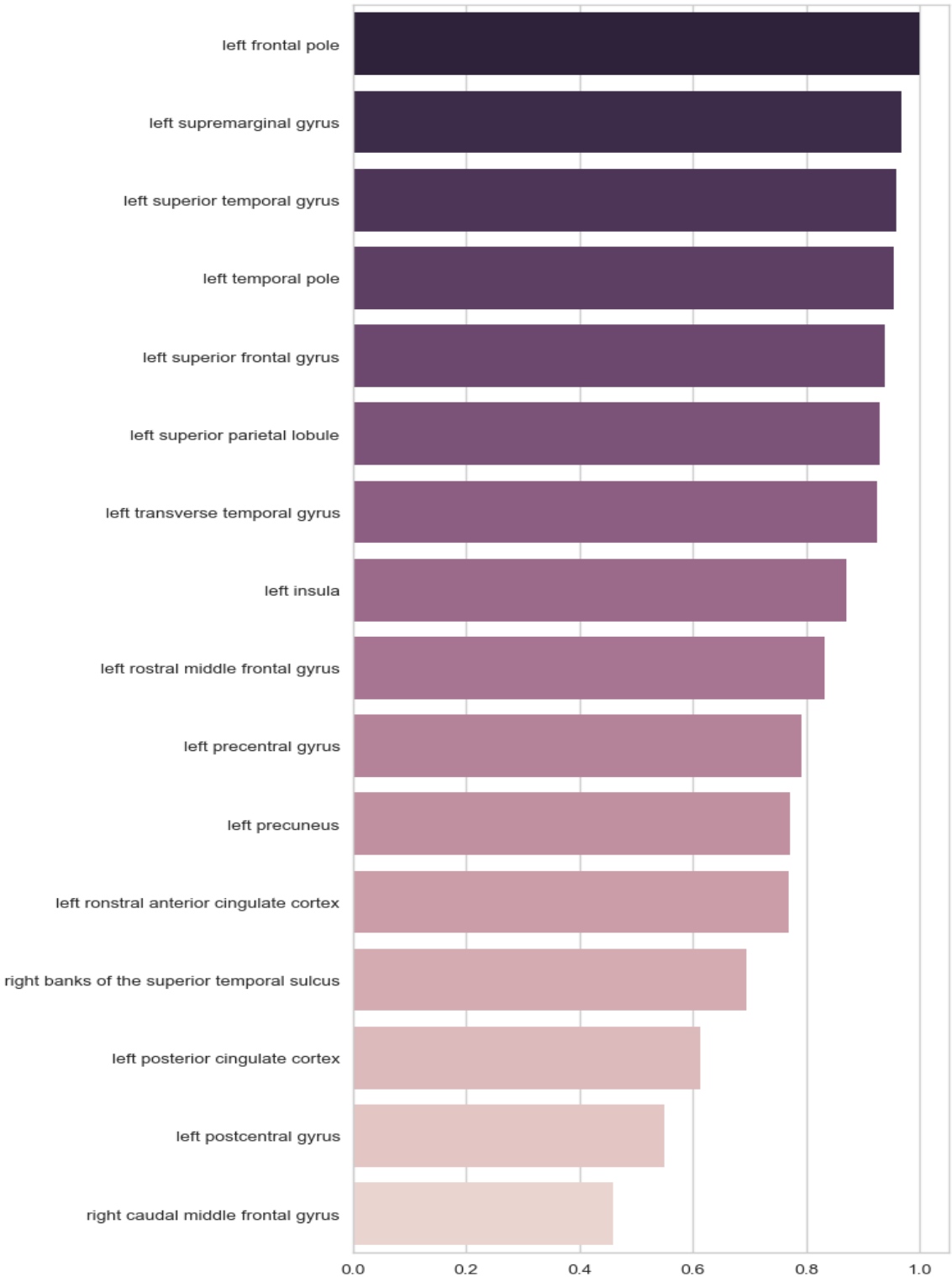

Figure 3.4: The 16 regions with highest attention weighting across all training iterations.

Table 3.1: The results from our testing of the baseline algorithms on the features constructed from the diffusion data alone, and our graph convolutional network (GCN) which additionally incorporates anatomical information. The results are averaged across five training iterations, which use subsamples of the data to ensure class balance.

Model	Accuracy	F1-Score	AUC
k-Nearest Neighbor	63.66%	0.636	0.646
Logistic Regression	75.72%	0.749	0.839
Ridge Regression	85.54%	0.883	0.500
Random Forest	77.77%	0.765	0.782
SVM (linear kernel)	87.66%	0.873	0.894
SVM (Polynomial kernel)	87.02%	0.899	0.887
Fully-Connected NN	83.98%	0.854	0.881
Convolutional NN	85.33%	0.900	0.908
Graph Convolutional NN	**92.14%**	**0.953**	**0.943**

weights. The names and relative importance assigned to these regions are shown in Figure 3.

Discussions and Conclusions

From the results on the baseline models, we can see that the features generated from the diffusion MRI data are suitable for distinguishing the PD vs. HC classes. Furthermore, we see from the improved performance of the GCN model that the incorporation of anatomical data improves the capacity for the data to be modeled. Of the 16 highest-weighted regions according to the GAT attentions layers, 9 coincide with lateral or contralateral regions identified by [84] as significantly contributing to the distinction between PD and HC classes. All but two of the regions listed in Figure 3 were from the left hemisphere, whereas the majority of regions in [84] were from the right hemisphere. We arent sure why this may be, but the stronger identification of left hemispheric regions aligns with asymmetries found by [111], wherein the left hemisphere is more significantly affected in early-stage PD.

Due to the time required to construct the pipeline, and the substantial time and compute resources required for each additional image, we used a relatively small data set. The models showed signs of overfitting during training, due to increasing performance on the

training data after improvement with the testing data had stopped. We feel that reproduction with a larger dataset may mitigate this issue and improve the robustness of our initial results.

We would also like to see future studies incorporate both diffusion and functional MRI data. We investigated the use of the C-PAC preprocessing software to generate features from functional MRI (fMRI) data, and we believe these features could be incorporated into our model. Additional anatomical information such as the volume of each region could also be incorporated, and even metadata such as age or genetic information could be added to each node of an image to encourage class separation. These points reflect our use of graph convolutional networks for multimodal neuroimage analysis, as the format allows for the combination of multiple forms of data in an efficient and intuitive manner. All of these points were beyond the scope of the current experiment, and are possibilities for future research.

We have made the code for our pipeline available on GitHub [21]. Included in the repository are the parameters we used to download our data from PPMI, so that researchers with access to the database might download similar data for reproduction. Processing this data is very technical; there are multiple ways of doing so and our pipeline is surely capable of being improved upon. For example, we utilized all 115 brain regions returned by Freesurfers segmentaion. Instead, [84] selectively utilize only 84 of the regions. By confining the number of regions, e.g., to only those with clinical significance to PD, we may see improvements in performance and interpretability.

We have presented here a complete pipeline for preprocessing multi-modal neuroimage data, applied to real-world data aimed at developing image biomarkers for Parkinson's disease research. We propose a novel graph-based deep learning model for analysing the data in an interpretable format. Our focus in this paper was to explicitly delineate the steps we took and implement sound data analysis techniques To this end, we hope to help bridge the gap between neuroscience research and advanced data analysis.

[21] `https://github.com/xtianmcd/GCNeuro`

Acknowledgements

Data used in the preparation of this article were obtained from the Parkinson's Progression Markers Initiative (PPMI) database (www.ppmi-info.org/data). For up-to-date information on the study, visit www.ppmi-info.org. PPMI - a public-private partnership - is funded by the Michael J. Fox Foundation for Parkinson's Research and funding partners, including Abbvie, Allergan, Avid, Biogen, BioLegend, Bristol-Mayers Squibb, Colgene, Denali, GE Healthcare, Genentech, GlaxoSmithKline, Lilly, Lundbeck, Merck, Meso Scale Discovery, Pfizer, Piramal, Prevail, Roche, Sanofi Genzyme, Servier, Takeda, TEVA, UCB, Verily, Voyager, and Golub Capital.

Appendix A: MRI Modalities

The modality which serves as the basis for the nodes of the graphs is anatomical T1-weighted MRI (aMRI [22]) data. This modality provides high resolution images which are quite useful for distinguishing different tissue types and region boundaries. The speed and relative simplicity of aMRI imaging results in fewer and less severe artifacts. For a given subject, images from the other modalities are often aligned to aMRI images, and this modality is often used to obtain brain masks (via skull stripping) and perform volumetric segmentation. Typical preprocessing includes motion-correction, intensity normalization, magnetic susceptibility correction, skull stripping, registration to a common brain atlas, and segmentation [23] [103, 104].

Diffusion-weighted MR imaging (dMRI) introduces additional noise sources. dMRI measures the diffusion of water molecules in the brain by applying pulsed magnetic field gradients in numerous directions, resulting in multiple 3D volumes for a single image. Typically, a higher resolution image (resembling anatomical images) is taken as the first volume, and is termed the *b0* volume. During processing, all other volumes are aligned to this volume. dMRI data is usually obtained using an MRI variant known as spin-echo echo planar imaging (EPI), which results in artifacts such as eddy currents and magnetic susceptibility artifacts. Typical preprocessing includes correcting these artifacts and co-registering the diffusion data to aMRI images of the same acquisition, for comparison to the aMRI data during analysis [103, 104].

Once dMRI data is cleaned, the information can be processed to trace the directionality of water diffusion across voxels, forming connected paths between them. This process, called *tractography* estimates white matter (WM) tracts, which are bundles of nerve fibers,

[22] In this paper we use anatomical MRI to refer to standard *T1-weighted* (T1w) MR imaging. T1 weighted refers to the specific sequence and timing of magnetic pulses and radio frequencies used during imaging. T1w MRI is a common MR imaging procedure; the important thing to note is that T1 weighting yields high-resolution images which show contrast between different tissue types, allowing for segmentation of different anatomical regions.

[23] `https://surfer.nmr.mgh.harvard.edu`

or axons, that connect regions of the brain. The specific tractography algorithm can significantly affect the analysis results, so we incorporate the output from four different tractography algorithms in our model.

Appendix B: Graph Convolutional Networks

Given an undirected weighted graph $\mathcal{G} = \mathcal{V}, \mathcal{E}, \mathbf{W}$ with a set of vertices ‘$\mathcal{V}$ with $|\mathcal{V}| = N$, a set of edges $\mathcal{E}$, and a weighted adjacency matrix $\mathbf{W}$, we define a signal on the vertices as a function ‘$\{ : \mathcal{V} \rightarrow \mathbb{R}$, returning a vector $\mathbf{f} \in \mathbb{R}^N$ for each vertex. The vector *signal* defined on each vertex represents that vertex's weighted connectivity to all other vertices [100].

We seek to learn filters g over the graph, similar to the local filters used in convolutional neural networks. The discrete Fourier transform (FT) matrix of the normalized graph Laplacian Ł provides a means for doing this. Ł is a real symmetric matrix represented as

$$Ł = I - D^{\frac{-1}{2}} \mathbf{W} D^{\frac{-1}{2}} \tag{3.8}$$

and with eigendecomposition $Ł = U \Lambda U^T$, where D is a diagonal matrix with entries $D_{ii} = \sum_j \mathbf{W}_{ij} = \mathbf{W} \cdot \mathbf{1} U$, $U = (u_1, ..., u_N)$ is a complete set of orthonormal eigenvectors, and Λ are the associated real, non-negative eigenvalues.

The graph FT $\hat{\mathbf{f}}$ of any function $f \in \mathbb{R}^N$ on the vertices of $\mathcal{G}$ gives the expansion of f in terms of the eigenvectors of Ł [100]. This allows us to define functions f and g, which are both functions on the vertices of $\mathcal{G}$, in terms of the eigendecomposition of the graph Laplacian of $\mathcal{G}$.

The Convolution Theorem [112] defines a linear operator that diagonalizes in the Fourier domain as a convolution operator in the vector domain. Commuting Ł with the translation operator produces such an operator [113] and can be used to convolve functions f and g.

We can now define a graph convolution of input signals x with filters g on $\mathcal{G}$ by

$$x * g_\theta = U g U^T x, \tag{3.9}$$

Where x is a specific instance of f (a single connectivity matrix of graph signals), U is the matrix of eigenvectors of Ł given by the graph FT, and θ are the parameters we wish to learn. We consider g_θ as a function of the eigenvalues Λ, $g_\theta(\Lambda) = diag(\theta)$; thus the parameters θ are the Fourier coefficients from the graph FT on Ł [98].

Finding these parameters are computationally expensive as multiplication with U is $O(N^2)$, and Ł itself may be quite expensive to calculate. So, an approximation is made in terms of Chebyshev polynomials $T_k(x)$ up to the K^{th} order [114]. Chebyshev polynomials are recursively defined $T_k(x) = 2xT_{k-1}(x) - T_{k-2}(x)$, with $T_0(x) = 1$ and $T_1(x) = x$. Now, $g_{\theta\prime}(\Lambda) \approx \sum_{k=0}^{K} \theta_{k\prime} T_k(\tilde{\Lambda})$, where rescaled $\tilde{\Lambda} = \frac{2}{l_{max}}\Lambda - I_N$ and l_{max} is the largest eigenvalue of Λ. Defining $\tilde{Ł} = \frac{2}{l_{max}}Ł - I_N$, we have

$$g_{\theta\prime} * x \approx \sum_{k=0}^{K} \theta_{k\prime} T_k(\tilde{Ł}) x \tag{3.10}$$

[98].

The expression is K-localized, relying only on nodes that are K-steps away from a given node (its K^{th}-order neighborhood). Evaluating such a function is $O(\mathcal{E})$. By limiting $K = 1$ we have a linear function with respect to Ł as the preactivation $\hat{H}$ of our convolutional layer. Wrapping $\hat{H}$ in a nonlinear activation function and stacking multiple layers gives us our graph convolutional network architecture. This so-called deep learning architecture removes the rigid parameterization enforced by Chebyshev polynomials [98].

[98] further approximate $l_{max} \approx 2$ and simplify the equation for $\hat{H}$ to $g_{\theta\prime} * x \approx \theta_{0\prime}(x) + \theta_{1\prime}(Ł - I_N)x = \theta_{0\prime}(x) - \theta_{1\prime} D^{\frac{-1}{2}} A D^{\frac{-1}{2}} x$, reducing the task to learning two free parameters

which can be shared over the whole graph. If $\theta_0\prime$ is set equal to $-\theta_1\prime$, then the equation can be expressed with a single parameter '$\theta = \theta_0\prime$:

$$g_{theta} * x \approx \theta(I_N + D^{\frac{-1}{2}} A D^{\frac{-1}{2}})x. \tag{3.11}$$

k successive applications of this operator effectively convolve the k^{th}-order neighborhood of a given node, but may also lead to numerical instabilities and the exploding/vanishing gradient problem, since $I_N + D^{\frac{-1}{2}} A D^{\frac{-1}{2}}$ now has eigenvalues in [0,2]. [98] solve this issue via a *renormalization trick* such that $I_N + D^{\frac{-1}{2}} A D^{\frac{-1}{2}}$ becomes $\tilde{D}^{\frac{-1}{2}} \tilde{A} \tilde{D}^{\frac{-1}{2}}$, where $\tilde{A} = A + I_N$ and $\tilde{D}_{ii} = \sum_j \tilde{A}_{ij}$. I.e., self-loops have been added to the adjacency matrix. The weights given to these connections should bear similar importance to the other connections, e.g., using the mean edge weight.

Finally, the equation is generalized to a signal $X \in \mathbb{R}^{NxC}$ with C-dimensional feature vectors at every node (each *element* will learn a single parameter) and F filters:

$$Z = \tilde{D}^{\frac{-1}{2}} \tilde{A} \tilde{D}^{\frac{-1}{2}} X \Theta, \tag{3.12}$$

where $\Theta \in \mathbb{R}^{CxF}$ are the parameters and $Z \in \mathbb{R}^{NxF}$ is the convolved signal matrix. This equation is of complexity $O(|\mathcal{E}|FC)$. Generalizing $X = H(l)$ as the inputs to a layer, where $H(0)$ is the original data and 'Θ to the weight matrix $\mathbf{W}(l)$ at a layer l, we can calculate a hidden layer as

$$H(l+1) = \sigma(\tilde{D}^{\frac{-1}{2}} \tilde{A} \tilde{D}^{\frac{-1}{2}} H(l) \mathbf{W}(l)). \tag{3.13}$$

The time complexity of computing a single attention mechanism is $O(|\mathcal{V}|FF\prime + |\mathcal{E}|F\prime)$, where F is the number of input features and $F\prime$ is the number of output features.

CHAPTER 4
CONCLUSION

Reproducibility and the Data Science Pipeline

In this thesis we have demonstrated the potential impact of deep machine learning algorithms in biomedical research, and the importance of ensuring reproducibility in studies related thereto. We have echoed the consensus among experts in the field that reproducibility needs increased attention, and have outlined the utility in the data science pipeline as a means to facilitate reproducible work.

Chapters 2 and 3 of this thesis demonstrate two domain specific experiments in which we formulated and employed data science pipelines to perform deep learning-based analysis on complex biomedical data. We used the guidelines compiled by [14], represented by the acronym PRIMAD and discussed in Chapter 1 of this thesis, to guide the construction of our pipelines. We also included tips and strategies we learned along the way, and provided in-depth background information on the algorithms used. In doing so we aimed to highlight the rich corpus of works that came before ours, and hope to have made productive contributions to the domains in which we focused.

As mentioned by [14] in their report on the 2016 Dagstuhl Seminar, an international convergence of computer science and domain experts, ensuring reproducibility requires significant time and consideration by researchers. We encountered this challenge in our own experiments, as our focus on designing thorough pipelines, implementation tips, and rich background information took away from the breadth of analysis we could perform on our data. We would also like to see our pipelines undergo further development with testing, increased documentation, and APIs for user modularity, as these improvements were outside the scope of our research but would further improve reproducibility.

As mentioned in Chapter 1 of this thesis, our focus on developing pipelines for our experiments resulted from difficulties when trying to reproduce other experiments. During both of our studies, we encountered works that lacked sufficient detail to reproduce. For example, our use of Graph Convolutional Networks (GCNs) in Chapter 3 was largely inspired by [84]. Although they provide the code they used to train their model and perform analysis, portions of the experimental methods are lacking, and their data is confidential and cannot be shared. Without example data to experiment with, we could not decipher portions of their code or reproduce some of their methods. Especially in cases where data cannot be shared (as with the data we used in Chapter 3), it is important to include detailed reports of the experimental process. We include as part of the code repository the settings we used during download from the PPMI database to help others download the same or a similar data set, once they acquire access from PPMI.

On the other hand, we also benefited from reproducible code during our experiment in Chapter 3. The novel GCN architecture we devised borrows from several publications, namely [98] and [99]. Both of these authors provide very well-documented code, and their background sections were instrumental in our understanding of GCNs. Our ability to build upon these works is a great reproducibility success.

Deep Machine Learning in Biomedical Research

The potential for deep learning (DL) algorithms to find new relationships and structures among the growing corpus of biomedical data has caught the attention of many researchers. We have conducted the work in this thesis in accordance with that potential, to encourage practices which enhance the robustness and utility of DL-based findings.

As with many emerging technologies, DL has limitations that should be considered. DL algorithms are particularly sensitive to the data they are trained on. They are highly capable of overfitting the training data, and may model any sampling or selection bias therein [6]. As discussed in Chapters 2 and 3, there are regulation schemes which can help

alleviate overfitting, and the use of a testing set improves the generalizability of results. The application of data-centered deep learning algorithms poses a particular challenge when dealing with real-world data. For example, models trained to recognize the Parkinsonian tremor may not be able to distinguish this PD motor symptom from a subject wearing the sensor while mowing the lawn [6]. Another PD motor symptom, bradykinesia, is marked by slowed movement, which is also typical during bouts of fatigue or when navigating a risky environment [24]. Algorithms trained in the lab or on limited data sets may not be equipped to handle such real-world cases.

While DL algorithms are quite useful for making predictions on new examples, forming explanations for how or why they generated those predictions can be more tricky. Analyzing intermediate outputs and visualizing the training process are helpful methods for increasing explanatory power. Similar to producing reproducible code, performing this type of *post-hoc* analysis may exceed the resources available for a single study. However, when experiments are reproducible, follow-up studies can be performed which dig deeper into the models and methods of the original work.

The Application to Parkinson's Disease

Parkinson's Disease (PD) is an intriguing example for examining DL applications to and reproducibility in biomedical research. Its diagnosis has traditionally relied on clinical assessments with some degree of subjectivity [20]. There is significant interest in establishing more quantitatively rigorous tools for PD diagnosis and research [24], and the nature of the disease lends itself to multiple modes of quantitative analysis.

Technological advances have yielded so-called technology-based objective measures (TOMs) capable of capturing biometric information for PD research [24]. There is concern over these technologies forming "competing islands of expertise" [24], particularly since these emerging technologies are not included in traditional clinical and research training.

Reproducible work alleviates this concern by allowing researchers to easily implement new technologies and methods, enabling comparison and integration of various TOM domains.

We focused on two principle areas of TOM-based PD research. Both experiments involved a great deal of background research. We focused on developing pipeline-style reports to enable others to pick up where we left off, as this was impossible at times when trying to implement the work of others. In the experiment in Chapter 2, we focus on developing a pipeline that unites implementational disparities we found in the literature. We focus on optimization and model simplicity to strengthen the legitimacy of DL-based methods for accelerometer research. We also focused on sound data science techniques to prevent issues such as data leakage and ensure robust results.

Although we were not able to use our pipeline in Chapter 2 to process the PD-related accelerometer data directly (due to remaining confidientiality of the data), we hope that our work may aid future experiments seeking to use LSTMs for PD- or other HAR-related accelerometer research. The PD accelerometer data from the challenge will be released following the publication of a manuscript, which contains findings from *post-hoc* analysis conducted by many of the participating teams, including ours.

In the pipeline described in Chapter 3, we again seek to unite methods and fill gaps we found in the literature related to neuroimage preprocessing. The processing required of neuroimage data is extensive and technical, each modality requiring specific steps. We were surprised by the general lack of information available. Unlike other areas of biomedical analysis, where equipment is proprietary and includes instructions with the purchase, open source software may have very little documentation and may not be actively maintained despite ongoing use.

Many papers do name the software and algorithms they use for preprocessing. We wanted to go a step further and provide the specific calls made to each software and discuss some of the challenges we faced, as it was very difficult to compile this information.

By generating our pipelines for real-world research problems we were able to obtain tangible analysis results, both to contribute to the field and verify the efficacy of our pipelines. That being said, we recognize that these are single examples and there is room for improvement in each of them. As much as we encourage use of our code and methods, we equally encourage a discussion of our methods and constructive revisions and refutations to decisions we made.

Closing Remarks

We believe that open source science has been and will continue to be essential to the modern scientific process. Open source technologies allow more people than ever before to enhance and apply their skills to a multitude of domains. We also find that deep learning-based algorithms are well-suited for the challenges of biomedical research. As such, we have conducted this thesis as a reflection of the emergence of open science in healthcare and a proposal for how to improve it going forward. We feel that the ultimate goal of biomedical research should be the application of its findings to curing diseases and solving healthcare-related problems in the real world. We advocate the framing of computational biomedical experiments around the creation of reusable code and models as beneficial to that mission.

REFERENCES

[1] J. Soni, U. Ansari, D. Sharma, and S. Soni, "Predictive data mining for medical diagnosis: An overview of heart disease prediction," *International Journal of Computer Applications*, vol. 17, pp. 43–48, Mar. 2011.

[2] J. Roski, G. W. Bo-Linn, and T. A. Andrews, "Creating value in health care through big data: Opportunities and policy implications," *Health Affairs*, vol. 33, no. 7, pp. 1115–1122, 2014.

[3] T. B. Murdoch and A. S. Detsky, "The Inevitable Application of Big Data to Health CareThe Inevitable Application of Big Data to Health Care," *JAMA*, vol. 309, no. 13, pp. 1351–1352, 2013.

[4] L. Lu, Y. Zheng, G. Carneiro, and L. Yang, Eds., *Deep learning and convolutional neural networks for medical image computing*, 2017.

[5] T. Hastie, R. Tibshirani, and J. Friedman, *The elements of statistical learning*. Springer, 2017.

[6] K. J. Kubota, J. A. Chen, and M. A. Little, "Machine learning for large-scale wearable sensor data in parkinson's disease: Concepts, promises, pitfalls, and futures," *Movement Disorders*, vol. 31, no. 9, pp. 1314–1326, 2016.

[7] D. W. Bates, S. Saria, L. Ohno-Machado, A. Shah, and G. Escobar, "Big data in health care: Using analytics to identify and manage high-risk and high-cost patients," *Health Affairs*, vol. 33, no. 7, pp. 1123–1131, 2014.

[8] R. Peng, "The reproducibility crisis in science: A statistical counterattack," *Significance*, vol. 12, no. 3, pp. 30–32, 2015.

[9] R. D. Peng, "Reproducible research in computational science," *Science*, vol. 334, no. 6060, pp. 1226–1227, 2011.

[10] L. A. Barba, *Reproducibility pi manifesto*, 2012.

[11] M. McNutt, "Reproducibility," *Science*, vol. 343, no. 6168, pp. 229–229, 2014.

[12] S. C. Landis, S. G. Amara, K. Asadullah, C. P. Austin, R. Blumenstein, E. W. Bradley, R. G. Crystal, R. B. Darnell, R. J. Ferrante, H. Fillit, R. Finkelstein, M. Fisher, H. E. Gendelman, R. M. Golub, J. L. Goudreau, R. A. Gross, A. K. Gubitz, S. E. Hesterlee, D. W. Howells, J. Huguenard, K. Kelner, W. Koroshetz, D. Krainc,

S. E. Lazic, M. S. Levine, M. R. Macleod, J. M. McCall, R. T. Moxley III, K. Narasimhan, L. J. Noble, S. Perrin, J. D. Porter, O. Steward, E. Unger, U. Utz, and S. D. Silberberg, "A call for transparent reporting to optimize the predictive value of preclinical research," *Nature*, vol. 490, 2012.

[13] J. Ioannidis, "Why most published research findings are false," *PLoS Med*, vol. 2, no. 8, 2005.

[14] J. Freire, N. Fuhr, and A. Rauber, "Reproducibility of Data-Oriented Experiments in e-Science (Dagstuhl Seminar 16041)," *Dagstuhl Reports*, vol. 6, no. 1, J. Freire, N. Fuhr, and A. Rauber, Eds., pp. 108–159, 2016.

[15] A. Karpathy, J. Johnson, and L. Fei-Fei, "Visualizing and understanding recurrent networks," *ArXiv*, 2015.

[16] G. E. Hinton, *A practical guide to training restricted boltzmann machines (version 1)*, Aug. 2010.

[17] X.-N. Zuo, J. S. Anderson, P. Bellec, R. M. Birn, B. B. Biswal, J. Blautzik, J. C. Breitner, R. L. Buckner, V. D. Calhoun, F. X. Castellanos, A. Chen, B. Chen, J. Chen, X. Chen, S. J. Colcombe, W. Courtney, R. C. Craddock, A. Di Martino, H.-M. Dong, X. Fu, Q. Gong, K. J. Gorgolewski, Y. Han, Y. He, Y. He, E. Ho, A. Holmes, X.-H. Hou, J. Huckins, Y. Jiang Tianziv Jiang, W. Kelley, C. Kelly, M. King, S. M. LaConte, J. E. Lainhart, X. Lei, H.-J. Li, K. Li, K. Li, Q. Lin, D. Liu, J. Liu, X. Liu, Y. Liu, G. Lu, J. Lu, B. Luna, J. Luo, D. Lurie, Y. Mao, D. S. Margulies, A. R. Mayer, T. Meindl, M. E. Meyerand, W. Nan, J. A. Nielsen, D. OAOConnor, D. Paulsen, V. Prabhakaran, Z. Qi, J. Qiu, C. Shao, Z. Shehzad, A. Tang Weijun and, H. Wang, K. Wang, D. Wei, G.-X. Wei, X.-C. Weng, X. Wu, T. Xu, N. Yang, Z. Yang, Y.-F. Zang, L. Zhang, Q. Zhang, Z. Zhang, Z. Zhang, K. Zhao, Z. Zhen, Y. Zhou, X.-T. Zhu, and M. P. Milham, "An open science resource for establishing reliability and reproducibility in functional connectomics," *Scientific Data*, vol. 1, 2014.

[18] S. Fernandes-Taylor, J. K. Hyun, R. N. Reeder, and A. H. Harris, "Common statistical and research design problems in manuscripts submitted to high-impact medical journals," vol. 4, p. 304, 2011.

[19] A. Reeve, E. Simcox, and D. Turnbull, "Ageing and parkinson's disease: Why is advancing age the biggest risk factor?" *Elsevier Sponsored Documents Ageing Research Reviews*, vol. 14, no. 100, pp. 19–30, 2014.

[20] K Gmitterova, J Gawinecka, F Llorens, D Varges, P Valkovic, and I Zerr, "Cerebrospinal fluid markers analysis in the differential diagnosis of dementia with lewy bodies and parkinsons disease dementia," *European Archives of Psychiatry and Clinical Neuroscience*, 2018.

[21] F DuBois Bowman, D. F. Drake, and D. E. Huddleston, "Multimodal imaging signatures of parkinson's disease," *Frontiers Neuroscience*, vol. 10, 2016.

[22] J. H. Lanskey, P. McColgan, A. E. Schrag, J. Acosta-Cabronero, G. Rees, H. R. Morris, and R. S. Weil, "Can neuroimaging predict dementia in parkinsons disease?" *Brain*, vol. 141, no. 9, 2018.

[23] D. Long, J. Wang, M. Xuan, Q. Gu, X. Xu, D. Kong, and M. Zhang, "Automatic classification of early parkinson's disease with multi-modal mr imaging," *PLOS ONE*, vol. 7, no. 11, 2012.

[24] A. J. Espay, P. Bonato, F. B. Nahab, W. Maetzler, J. M. Dean, J. Klucken, B. M. Eskofier, A. Merola, F. Horak, A. E. Lang, R. Reilmann, J. Giuffrida, A. Nieuwboer, M. Horne, M. A. Little, I. Litvan, T. Simuni, E. R. Dorsey, M. A. Burack, K. Kubota, A. Kamondi, C. Godinho, J.-F. Daneault, G. Mitsi, L. Krinke, J. M. Hausdorff, B. R. Bloem, and S. Papapetropoulos, "Technology in parkinson's disease: Challenges and opportunities," *Movement Disorders, the official Journal of the International Parkinson and Movement Disorder Society*, vol. 31, no. 9, pp. 1272–1282, 2016.

[25] J. L. Reyes-Ortiz, A. Ghio, D. Anguita, X. Parra, J. Cabestany, and A. Catala, "Human activity and motion disorder recognition - towards smarter interactive cognitive environments," *21th European Symposium on Artificial Neural Networks, Computational Intelligence and Machine Learning, ESANN 2013*, 2013.

[26] P. Brodal, *The central nervous system, fifth edition*. Oxford University Press, 2016.

[27] M Cousineau, P. Jodoin, F. Morency, V Rozanski, M GrandMaison, B. Bedell, and M Descoteaux, "A test-retest study on parkinsons ppmi dataset yields statistically significant white matter fascicles," *NeuroImage: Clinical*, vol. 16, 2017.

[28] L Griffanti, M Rolinski, K Szewczyk-Krolikowski, R. Menke, N Filippini, G Zamboni, M Jenkinson, M. Hu, and C. Mackay, "Challenges in the reproducibility of clinical studies with resting state fmri: An example in early parkinson's disease," *Neuroimage*, vol. 124(Pt A), pp. 704–703, 2016.

[29] E. Kim, S. Helal, and D. Cook, "Human activity recognition and pattern discovery," *IEEE Persuasive Computing*, vol. 9, no. 1, 2010.

[30] N. Ravi, N. Dandekar, P. Mysore, and M. L. Littman, "Activity recognition from accelerometer data," *IAAI 2005 Proceedings of the 17th conference on Innovative applications of artificial intelligence*, vol. 3, pp. 1541–1546, 2005.

[31] L. Bao and S. S. Intille, "Activity recognition from user-annotated acceleration data," *Springer-Verlag*, 2004.

[32] J. Ordonez and D. Roggen, "Deep convolutional and lstm recurrent neural networks for multimodal wearable activity recognition," *Sensors*, 2016.

[33] W. Gao, C. Ruan, and R. Xu, "Sensor-based semantic-level human activity recognition using temporal classification," Stanford University, Tech. Rep., 2016.

[34] A. Mannini and A. M. Sabatini, "Machine learning methods for classifying human physical activity from on-body accelerometers," *Sensors*, 2010.

[35] H. Gjoreski, J. Bizjak, M. Gjoreski, and M. Gams, "Comparing deep and classical machine learning methods for human activity recognition using wrist accelerometer," Jozef Stefan Institute Department of Intelligent Systems, Tech. Rep., 2016.

[36] H. Sjostrum and C. N. Sanchez, "Deepconvlstm on single accelerometer locomotion recognition," UMEA Universitet, Tech. Rep., 2017.

[37] A. Rassem, M. El-Beltagy, and M. Saleh, "Cross-country skiing gears classification using deep learning," *ArXiv*, 2017.

[38] M. Fiterau, J. Fries, E. Halilaj, N. Siranart, S. Bhooshan, and C. Re, "Similarity-based lstms for time series representation learning in the presence of structured covariates," *29th Conference on Neural Information Processing Systems - NIPS 2016*, 2016.

[39] W Seok, Y Kim, and C Park, "Pattern recognition of human arm movement using deep reinforcement learning," *2018 International Conference on Information Networking - ICOIN*, 2018.

[40] T. Zebin, P. J. Scully, and K. B. Ozanyan, "Human activity recognition with inertial sensors using a deep learning approach," *Sensors, 2016 IEEE*, 2017.

[41] R. J. Williams and D. Zipser, "A learning algorithm for continually running fully recurrent neural networks," *Neural Computation*, 1989.

[42] S. Hochreiter and J. Schmidhuber, "Long short-term memory," *Neural Computation*, 1997.

[43] F. A. Gers, N. N. Schraudolph, and J. Schmidhuber, "Learning precise timing with lstm recurrent networks," *Journal of Machine Learning Research*, 2002.

[44] N. Reimers and I. Gurevych, "Optimal hyperparameters for deep lstm-networks for sequence labeling tasks," *ArXiv*, 2017.

[45] O. Press and L. Wolf, "Using the output embedding to improve language models," *ArXiv*, 2017.

[46] S. Merity, N. S. Keskar, and R. Socher, "Regularizing and optimizing lstm language models," *ArXiv*, 2017.

[47] W. McCulloch and W Pitts, "A logical calculus of the ideas immanent in neurons activity," *Bulletin of Mathematical Biology*, vol. 5, no. 4, pp. 115–133, 1943.

[48] A. Geron, *Hands-on machine learning with scikit-learn and tensorflow*. Sebastopol, CA: OReilly Media, Inc., 2017, pp. 253–413.

[49] F. Rosenblatt, "The perceptron - a perceiving and recognizing automation," Cornell Aeronautical Laboratory, Tech. Rep., 1957.

[50] J. A. Miller, "Introduction to data science using scalation," University of Georgia Department of Computing Science, Tech. Rep., 2018.

[51] N. Y. Hammerla, S. Holloran, and T. Ploetz, "Deep, convolutional, and recurrent models for human activity recognition using wearables," *ArXiv*, 2016.

[52] R. Jozefowicz, W. Zaremba, and I. Sutskever, "An empirical exploration of recurrent network architectures," *32nd International Conference on Machine Learning*, 2015.

[53] R. Pascanu, T. Mikolov, and Y. Bengio, "On the difficulty of training recurrent neural networks," *ArXiv*, 2013.

[54] P. Rivera, E. Valerezo, M.-T. Choi, and T.-S. Kim, "Recognition of human hand activities based on a single wrist imu using recurrent neural networks," *International Journal of Pharma Medicine and Biological Sciences*, 2017.

[55] Y. Zhao, R. Yang, G. Chevalier, and M. Gong, "Deep residual bidir-lstm for human activity recognition using wearable sensors," *ArXiv*, 2017.

[56] S. Broome, "Objectively recognizing human activity in body-worn sensor data with more or less deep neural networks," Master's thesis, KTH Royal Institute of Technology School of Computer Science and Communication, 2017.

[57] Y. Guan and T. Plotz, "Ensembles of deep lstm learners for activity recognition using wearables," *Proceedings of the ACM on Interactive, Mobile, Wearable and Ubiquitous Technologies*, 2017.

[58] C. U, "A machine learning based activity recognition for ambient assisted living," *International Journal on Future Revolution in Computer Science and Communication Engineering*, vol. 4, no. 3, 2018.

[59] D. Setterquist, "Using a smartphone to detect the standing-to-kneeling and kneeling-to-standing postural transitions," Master's thesis, KTH Royal Institute of Technology School of Electrical Engineering and Computer Science, 2018.

[60] X. Zhang, L. Yao, C. Huang, S. Wang, M. Tan, G. Long, and C. Wang, "Multi-modality sensor data classification with selective attention," *2018 International Joint Conference on Artificial Intelligence At Stockholm, Sweden*, 2018.

[61] Y. Chen, K. Zhong, J. Zhang, and X. Zhao, "Lstm networks for mobile human activity recognition," *2016 International Conference on Artificial Intelligence*, 2016.

[62] M. Musci, D. D. Martini, N. Blago, T. Facchinetti, and M. Piastra, "Online fall detection using recurrent neural networks," *ArXiv*, 2018.

[63] V. Bergelin, "Human activity recognition and behavioral prediction using wearable sensors and deep learning," Master's thesis, Linkopings Universitet Matematiska Institutionen, 2017.

[64] K. Kyritsis, C. Diou, and A. Delopoulos, "Food intake detection from inertial sensors using lstm networks," *New Trends in Image Analysis and Processing - ICIAP*, 2017.

[65] A. Moreau, P. Anderer, M. Ross, A. Cerny, T. Almazan, and B. Peterson, "Detection of nocturnal scratching movements in patients with atopic dermatitis using accelerometers and recurrent neural networks," *IEEE Journal of Biomedical and Health Informatics*, 2016.

[66] E. Carvalho, B. V. Ferreira, J. Ferreira, C. de Souza, H. V. Carvalho, Y. Suhara, A. S. Pentland, and G. Pessin, "Exploiting the use of recurrent neural networks for driver behavior profiling," *2017 International Joint Conference on Neural Networks - IJCNN*, 2017.

[67] G. Lefebvre, S. Berlemont, F. Mamalet, and C. Garcia, *Inertial gesture recognition with blstm-rnn*. Switzerland: Springer Series in Bio-/Neuroinformatics 4 - Artificial Neural Networks, 2015, pp. 393–410.

[68] S. Shin and W. Sung, "Dynamic hand gesture recognition for wearable devices with low complexity recurrent neural networks," *2016 IEEE International Symposium on Circuits and Systems - ISCAS*, 2016.

[69] E. Wu and P. Adu, "What am i doing - robust human activity detection with smartphones athletics and sensing devices," Stanford University, Tech. Rep., 2017.

[70] B Hu, P. Dixon, J. Jacobs, and J. Schiffman, "Machine learning algorithms based on signals from a single wearable inertial sensor can detect surface- and age-related differences in walking," *Journal of Biomechanics*, 2018.

[71] S. Wiesler, A. Richard, R. Schluter, and H. Ney, "Mean-normalized stochastic gradient for large-scale deep learning," *IEEE International Conference on Acoustics, Speech and Signal Processing*, 2014.

[72] R. Zhao, J. Wang, R. Yan, and K. Mao, "Machine health monitoring with lstm networks," *2016 10th International Conference on Sensing Technology - ICST*, 2016.

[73] A. Murad and J.-Y. Pyun, "Deep recurrent neural networks for human activity recognition," *Sensors*, vol. 17, no. 11, 2017.

[74] F. Karim, S. Majumdar, H. Darabi, and S. Harford, "Multivariate lstm-fcns for time series classification," *ArXiv*, 2018.

[75] M.-C. Lee and S.-B. Cho, "Mobile gesture recognition using hierarchical recurrent neural network with bidirectional long short-term memory," *6th International Conference on Mobile Ubiquitous Computing, Systems, Services and Technologies*, 2012.

[76] G. James, D. Witten, T. Hastie, and R. Tibshirani, *An introduction to statistical learning*. New York: Springer, 2017.

[77] D. Anguita, A. Ghio, L. Oneto, X. Parra, and J. L. Reyes-Ortiz, "A public domain dataset for human activity recognition using smartphones," *21th European Symposium on Artificial Neural Networks, Computational Intelligence and Machine Learning, ESANN 2013*, 2013.

[78] J. Bergstra, R. Bardenet, Y. Bengio, and B. Kegl, "Algorithms for hyper-parameter optimization," *NIPS*, 2011.

[79] J. Bussmann, W. Martens, J. Tulen, F. Shasfoort, H. van den Berg-Emons, and H. Stam, "Measuring daily behavior using ambulatory accelerometry - the activity monitor," *Behavior Research Methods, Instruments, and Computers*, 2001.

[80] J. Bussmann, J. Tulen, E. van Herel, and H. Stam, "Quantification of physical activities by means of ambulatory accelerometry - a validation study," *Psychophysiology*, 1998.

[81] S. Liu, W. Cai, S. Liu, F. Zhang, M. Fulham, D. Feng, S. Pujol, and R. Kikinis, "Multimodal neuroimaging computing: A review of the applications in neuropsychiatric disorders," *Brain Inform*, vol. 2, no. 3, pp. 167–180, 2015.

[82] M Descoteaux, L Maier-Hein, A Franz, P Jannin, D Collins, and S Duchesne, Eds., *Distance metric learning using graph convolutional networks: Application to functional brain networks*, vol. 10433, ser. Lecture Notes in Computer Science, Medical Image Computing and Computer Assisted Intervention (MICCAI 2017), Springer, Cham, 2017.

[83] S. I. Ktena, S. Parisot, E. Ferrante, M. Rajchl, M. Lee, B. Glocker, and D. Rueckert, "Metric learning with spectral graph convolutions on brain connectivity networks," *NeuroImage*, vol. 169, 431442, 2018.

[84] *Multi-view graph convolutional network and its applications on neuroimage analysis for parkinsons disease*, AMIA Annual Symposium Proceedings Archive, 2018.

[85] D. Matthews, H Lerman, A Lukic, R. Andres, A Mirelman, M. Wenick, N Giladi, S. Strother, K. Evans, J. Cedarbaum, and E Even-Sapir, "Fdg pet parkinsons disease-related pattern as a biomarker for clinical trials in early stage disease," *NeuroImage: Clinical*, 2018.

[86] J Blesa and S Przedborski, "Parkinson's disease: Animal models and dopaminergic cell vulnerability," *Frontiers Neuroanatomy*, vol. 8, 2014.

[87] M Zhong, W Yang, B Huang, W Jiang, X Zhang, X Liu, L Wang, J Wang, L Zhao, Y Zhang, Y Liu, J Lin, and R Huang, "Effects of levodopa therapy on voxel-based degree centrality in parkinsons disease," *Brain Imaging and Behavior*, 2018.

[88] R Geevarghese, D. Lumsden, N Hulse, M Samuel, and K Ashkan, "Subcortical structure volumes and correlation to clinical variables in parkinsons disease," *Journal of Neuroimaging*, vol. 25, no. 2, 2014.

[89] M Tahmasian, L. Bettray, T van Eimeren, A Drzezga, L Timmermann, C. Eickhoff, S. Eickhoff, and C Eggers, "A systematic review on the applications of resting-state fmri in parkinson's disease: Does dopamine replacement therapy play a role?" *Cortex*, vol. 73, pp. 80–105, 2015.

[90] C Luo, W Song, Q Chen, Z Zheng, K Chen, B Cao, J Yang, J Li, X Huang, Q Gong, and H. Shang, "Reduced functional connectivity in early-stage drug-naive parkinson's disease: A resting-state fmri study," *Neurobiol Aging*, vol. 35, no. 2, pp. 431–441, 2014.

[91] S Baudrexel, T Witte, C Seifried, F von Wegner, F Beissner, J. Klein, H Steinmetz, R Deichmann, J Roeper, and R Hilker, "Resting state fmri reveals increased subthalamic nucleus-motor cortex connectivity in parkinson's disease," *Neuroimage*, vol. 55, no. 4, pp. 1728–1738, 2011.

[92] K Kumar, "Data driven methods for characterizing individual differences in brain mri," PhD thesis, Ecole De Technologie Superieure Universite Du Quebec, 2018.

[93] E. J. Hartman, J. D. Keeler, and J. M. Kowalski, "Layered neural networks with gaussian hidden units as universal approximations," *Neural Computation*, vol. 2, no. 2, pp. 210–215, 1990.

[94] J Kleesiek, G Urban, A Hubert, D Schwarz, K Maier-Hein, M Bendszus, and A Biller, "Deep mri brain extraction: A 3d convolutional neural network for skull stripping," *NeuroImage*, vol. 129, 2016.

[95] R. Hjelm, E Damaraju, K Cho, H Laufs, S Plis, and V Calhoun, "Spatio-temporal dynamics of intrinsic networks in functional magnetic imaging data using recurrent neural networks," *Frontiers in Neuroscience*, vol. 12, no. 600, 2018.

[96] R. Hjelm, V Calhoun, R Salakhutdinov, E Allen, T Adali, and S Plis, "Restricted boltzmann machines for neuroimaging: An application in identifying intrinsic networks," *NeuroImage*, vol. 96, 2014.

[97] R. Burciu, R. Seidler, P Shukla, M. Nalls, A. Singleton, M. Okun, and D. Vaillancourt, "Multimodal neuroimaging and behavioral assessment of a-synuclein polymorphism rs356219 in older adults," *Neurobiol Aging*, vol. 66, pp. 32–39, 2018.

[98] *Semi-supervised classification with graph convolutional networks*, ICLR, 2017.

[99] *Graph attention networks*, ICLR, 2018.

[100] D. I. Shuman, S. K. Narang, P Frossard, A Ortega, and P Vandergheynst, "The emerging field of signal processing on graphs: Extending high-dimensional data analysis to networks and other irregular domains," *IEEE Signal Processing Magazine*, vol. 30, no. 3, pp. 83–98, 2013.

[101] *Siamese neural networks for one-shot image recognition*, vol. 37, Lille, France: JMLR: W&CP, 2015.

[102] V. Kumar BG, G. Carneiro, and I. Reid, "Learning local image descriptors with deep siamese and triplet convolutional networks by minimizing global loss functions," *IEEE CVPR*, 53855394, 2016.

[103] Y. Wang, *Principles of magnetic resonance imaging: Physics concepts, pulse sequences and biomedical applications*. 2015.

[104] R. H. Hashemi, W. G. Bradley Jr., and C. J. Lisanti, *Mri: The basics, third edition*. Lippincott Williams & Wilkins, 2010.

[105] S Mori, B. Crain, V. Chacko, and P. van Zijl, "Three-dimensional tracking of axonal projections in the brain by magnetic resonance imaging," *Ann. Neurol.*, vol. 45, pp. 265–269, 1999.

[106] P. Basser, S Pajevic, C Pierpaoli, J Duda, and A Aldroubi, "In vivo fiber tractography using dt-mri data," *Magn. Reson. Med.*, vol. 44, pp. 625–632, 2000.

[107] M Lazar, D. Weinstein, J. Tsuruda, K. Hasan, K Arfanakis, M. Meyerand, B Badie, H. Rowley, V Haughton, A Field, and A. Alexander, "White matter tractography using diffusion tensor deflection," *Hum. Brain Mapp.*, vol. 18, pp. 306–321, 2003.

[108] *Tracking neuronal fiber pathways in the living human brain*, vol. 96, 18, Proceedings of the National Academy of Sciences of the United States of America, 1999.

[109] L. Zhan, J. Zhou, Y. Wang, Y. Jin, N. Jahanshad, G. Prasad, T. M. Nir, C. D. Leonardo, J. Ye, and P. M. Thompson, "Comparison of nine tractography algorithms for detecting abnormal structural brain networks in alzheimers disease," *Frontiers in Aging Neuroscience*, vol. 7, no. 48, 2015.

[110] R Peters, "Ageing and the brain," *Postgraduate Medical Journal*, vol. 82, no. 964, pp. 84–88, 2006.

[111] D. O. Claassen, K. E. McDonell, R. Donahue, N. Wylie, H. Kang, P. Hedera, D. Zald, B. Landman, B. Dawant, and S. Rane, "Cortical asymmetry in parkinson's disease: Early susceptibility of the left hemisphere," vol. 6, no. 12, 2016.

[112] S. Mallat, *A wavelet tour of signal processing: The sparse way*. Academic Press, 2009.

[113] M. Henaff, J. Bruna, and Y. LeCun, "Deep convolutional networks on graph-structured data," *ArXiv*, 2015.

[114] D. K. Hammond, P. Vandergheynst, and R. mi Gribonval, "Wavelets on graphs via spectral graph theory," *Applied and Computational Harmonic Analysis*, vol. 30, no. 2, 129150, 2011.