# LEARNING WITH LESS: LOW-RANK DYNAMICS, COMMUNICATION, AND INTROSPECTION IN NEUROIMAGING

A Dissertation
Presented to
The Academic Faculty

By

Bradley Baker

In Partial Fulfillment
of the Requirements for the Degree
Doctor of Philosophy in the
College of Computing
Department of Computational Science and Engineering

Georgia Institute of Technology

December 2023

**LEARNING WITH LESS: LOW-RANK DYNAMICS, COMMUNICATION, AND INTROSPECTION IN NEUROIMAGING**

Thesis committee:


Dr. Vince Calhoun
Electrical and Chemical Engineering
*Georgia Institute of Technology*

Dr. Anqi Wu
Department of Computational Science and Engineering
*Georgia Institute of Technology*

Dr. Sergey Plis
Tri-Institutional Rsearch Center for Neuroimaging and Data Science
*Georgia State University*

Dr. Chao Zhang
Department of Computational Science and Engineering
*Georgia Institute of Technology*

Dr. Robyn Miller
Tri-Institutional Rsearch Center for Neuroimaging and Data Science
*Georgia State University*

Dr. Xiuwei Zhang
Department of Computational Science and Engineering
*Georgia Institute of Technology*



Date approved: March 23, 2023

"The multiple must be made, not by always adding a higher dimension,

but rather in the simplest of ways, by dint of sobriety,

with the number of dimensions one already has available—

always $n - 1$ (the only way the one belongs to the multiple: always subtracted).

Subtract the unique from the multiplicity to be constituted; write at $n - 1$ dimensions."

*Gilles Deleuze - Mille Plateaux*

For my parents and sister, who have always supported me,

and my cat Asiago, who is utterly, impossibly cute.

## ACKNOWLEDGMENTS

I would like to thank Dr. Vince Calhoun and Dr. Sergey Plis, with whom I have been working for nearly a decade. Through work with you, I have discovered a way to apply my talents in a scientific field that is endlessly rich and complex. You have helped shape my interests, and provided mentorship through difficult projects. My being able to study the brain is just the beginning of our work together, but what a beautiful beginning to have. I am looking forward to the future!

I would like to thank the faculty of the New College of Florida, where I completed my undergraduate work in 2016. To Dr. Patrick McDonald, Dr. David Gillman, Dr. April Flakne, Dr. Aron Edidin, Dr. Glenn Cuomo and others: you inspired me to achieve academic independence, rigor, and curiosity which has continued to characterize my graduate work. I began my work with Vince's lab through the opportunities New College created, and I owe all of my success to that beautiful and strange institution. May the weird heart of new college never change, despite those who would have it otherwise.

I would like to thank H. Jeremy Bockholt for his help learning all of the diverse ways we can apply our analysis techniques in real and helpful settings, and for allowing me to help with projects run through the Advanced Biomedical Imaging Research Group. I could not have continued to work in this field without the inspiration you provided.

I would like to individually thank the committee members for reviewing this proposal, and for agreeing to provide insights which will improve the work leading up to the full dissertation. Dr. Chao Zhang, Dr. Xiuwei Zhang, Dr. Anqi Wu, Dr. Robyn Miller, Dr. Sergey Plis, and Dr. Vince Calhoun: I am excited to receive your feedback!

To the students and staff of TReNDsCenter: thank you for your discussions, collaborations, jokes, and kind words. You all will accomplish great things!

## TABLE OF CONTENTS

## LIST OF TABLES

## LIST OF FIGURES

## SUMMARY

The enclosed research is a focused empirical and theoretical analysis of the optimization methods in machine learning, and the underlying role that the matrix rank of utilized learning statistics plays in these algorithms. We show that this new perspective on machine learning optimization provides benefits in terms of communication-efficient federated learning algorithms, as well as novel insights in terms of model introspection and theory of learning dynamics. In applications to the complex domain of Neuroimaging data analysis, we aim to show that this rank-focused frame of reference allows for unique insights into how models perform on particular populations.

Chapter 1 provides a survey of literature with a focus on low-rank models in machine learning and neuroimaging, as well as perspectives that rank have provided into the dynamics of machine learning. Our unique perspective on low-rank learning from the standpoint of PCA, ICA and subsequently Auto-Differentiation stand out from the literature in two primary ways: first, our methods for federated learning take an approach which emphasizes communicating only the "most important" statistics (in the sense of maximally dominant in the spectrum), while also starting from principled structure such as the low-rank assumptions of ICA or the outer-product structure of AD; second, because our methods our founded in low-rank structures at work in ubiquitous optimization techniques, we are given a unique and intuitive theoretical perspective for analyzing the dynamics of these models which has otherwise only come with otherwise limiting assumptions.

In chapters 2 and 3, we have shown that federated learning algorithms for Independent Component Analysis (ICA) can be achieved in two ways. First, by making the assumption that sources across sites share a common low-rank mixing matrix (allowing us to compute a decentralized Joint ICA), we show that sites in a federated learning setup can use iterative principal component analysis (PCA) to reduce decentralized data to the desired rank while still preserving privacy of individual subjects. We have shown in subsequent work that this same iterative PCA reduction allows for computation of a decentralized group ICA if each

site performs a local Infomax ICA optimization after the distributed PCA procedure. We apply both of these algorithms as distributed variants of standard algorithms for analyzing neuroimaging data, demonstrating that the complex domain benefits from the low-rank structure of the optimization.

Having explored two algorithms which emerge from low-rank assumptions on input data, we encounter an intriguing question if such a structure may not only be applied to data, but to the dynamics of complex machine learning models, such as Artificial Neural Networks. In chapters 4, 5 and 6, we present work investigating the outer-product structure of the gradient in backpropagation optimization for artificial neural networks. Chapter 4 contains the text and empirical results on rank-efficient distributed auto-differentiation (rank-dAD). In chapter 5 (submitted to Neurips 2023), we have found a number of theoretical bounds on the rank of the gradient during training, and we have found that the effective rank of the gradient changes dynamically during training, providing some fascinating insights into the relationship between the learning of dominant modes in data sets and models overfitting to noise. We show how these theoretical constraints lead to intuitive algorithms for communication-efficient distributed auto-differentiation. Additionally we show theoretically how bounds on gradient rank can be derived for any nonlinear activation which is piece-wise linear. Finally, in chapter 6 we show how our investigations into gradient rank allow for a novel kind of model introspection by way of the singular value decomposition of gradients computed dynamically during auto-differentiation. The unique perspective we take from auto-differentiation allows us to uniquely compare group-specific dynamics in the gradient rank and individual singular values. We show in this chapter how the method can be applied in practice to both numerical and neuroimaging data sets.

Finally, chapter 7 provides a discussion of the overall body of the dissertation, and focuses on a number of promising directions for future work. Beyond expansion of the methods presented here to new and interesting data sets and architectures, we provide an initial empirical demonstration of how general nonlinear functions can affect the spectra of

low-rank matrices, providing empirical groundwork for a theoretical extension of the work in chapter 5. Furthermore, we discuss how these insights may lead to significant runtime improvements for the computationally-intensive introspection method from chapter 6.

# CHAPTER 1
# LITERATURE SURVEY

In this section, we provide a brief review of the literature relevant to the work in this dissertation. First, we provided a brief review of rank-based methods in Neuroimaging, with the initial work of this dissertation standing apart from these works by exploiting their low-rank structure to invent intuitive federated learning versions of these algorithms. Subsequently, we provide a brief review of federated deep learning with a focus on one extant methods which exploits rank to produce communication-efficient algorithms. Our method for distributed auto-differentiation stands out from this work by exploiting an inherently low-rank structure in deep learning optimization, rather than simply starting from random projections. Finally, we provide a brief survey of theoretical work on deep learning optimization. Our approach from distributed auto-differentiation fits well into state-of-the-art research on deep learning dynamics, and it provides us with an intuitive theoretical ground which does not require any assumptions on the structure of the feature space.

## 1.1 Rank-Based Methods in Neuroimaging

In scientific domains which deal with highly complex data, low-rank machine-learning methods can play an important role in addressing data complexity, or providing results which are more easily interpretable. In neuroimaging in particular, regardless of the modality acquired the number of voxels typically exceends the number of samples acquired for a particular study, and thus suffers from the well-known problem of *curse of dimensionality* [1]. The rationale behind complexity reduction in neuroimaging then is to select for features which are *most salient* to a given problem setting, and can be achieved in a number of different ways.

### 1.1.1 Partial Least Squares

A number of supervised learning techniques have been used for reducing feature complexity; however, the majority of these methods, such as elastic-net regularization or pearson-correlation filtering, do not utilize a low-rank structure in their optimization (for a survey of these methods see [2]). The exception to this are Partial Least Squares methods [3, 4], such as Partial Least Squares Correlation (PLSC) [5] and Partial Least Squares Regression (PLSR) [6], where the product of a set of predictor variables $X$ and target variables $Y$ is computed as

$$M = Y^\top X$$

. The singular value decomposition of $M$ is then computed as

$$M = U\Sigma V^\top$$

, and the top $k$ left and right eigenvectors in $U$ and $V$ are used to reduce select the most salient input features (brain images) and target features (diagnosis or behavioral design e.g.). PLSC and PLSR have been applied succesfully to a number of predictive tasks in neuroimaging such as age classification, prediction of cognitive scores, and multimodal feature-reduction.

### 1.1.2 Principal and Independent Component Analysis

Unsupervised feature reduction methods are also prevalent in neuroimaging, allowing researchers to explore imaging features without being constrained to a particular behavioral task, demographic variable, or other target. For example, resting state functional Magnetic Resonance Imagining (fMRI) can especially benefit from unsupervised data-driven reduction, as it inherently involves no underlying design.

Principal component analysis (PCA) [7] is a popular method for unsupervised dimen-

sion reduction, which computes the SVD of the covariance matrix of the input features, and uses the top $k$ eigenvectors (with $k$ chosen by the user) to obtain a linear combination of features capturing the most variance in the data. PCA has been utilized to extract relevant principal components in neuroimaging studies in Schizophrenia [8, 9, 10], Alzheimers [11], Major Depressive Disorder [12], and face recognition in MRI [13].

Independent Component Analysis (ICA) is a popular blind source separation (BSS) method which attempts to decompose mixed signals into independent components (ICs), or sources, without prior knowledge of the structure of those sources. Empirically, ICA applied to brain imaging data produces robust features which are physiologically interpretable and markedly reproducible across studies [14, 15, 16, 17]. Indeed, while justification for successful ICA of fMRI results had been previously attributed to sparsity alone [18], it has been shown that statistical independence between the underlying sources is in fact a key driving mechanism of ICA algorithms [19], with additional benefits possible by trading off between the two [20].

In linear ICA, we model a data matrix $X \in \mathbb{R}^{M \times N}$ as a product $X \approx WS$, where $S \in \mathbb{R}^{k \times N}$ is composed of $N$ observations from $k$ *statistically independent* components, each representing an underlying signal source. Thus, we can interpret ICA in terms of this generative model, with independent sources $S$ submitted to a linear mixing process described by a mixing matrix $A \in \mathbb{R}^{M \times k}$, forming the observed data $X$. Most ICA algorithms seek to recover the "unmixing matrix" $\hat{W} = A^{-1}$ (or in the case where $A$ is not square, the pseudo-inverse, $A^{+}$), by maximizing independence between rows of the product $WX$, assuming the matrix $A$ is invertible.

Maximal information transfer (Infomax) [21] is a popular heuristic for estimating $W$ by maximizing an entropy functional related to $WX$. Another class of algorithms includes the famous family of fixed-point methods such as Fast ICA [22, 23, 24]. These locally optimize a "contrast" function such as kurtosis or negentropy.

ICA, along with other methods for BSS, has found wide application. In particular,

functional magnetic resonance imaging (fMRI) and other biomedical imaging data use ICA models to interpret subject imaging data [16]. For fMRI, many models assume that functionally connected regions in the brain are systematically nonoverlapping. ICA has been used in applications ranging from interpreting physiology to analyzing task-related signals in both the spatial and temporal domains.

Additionally, a number of extensions of ICA exist for the purpose of jointly analyzing multiple data sets to perform a simultaneous decomposition across a large number number of subjects and different modalities [25, 26, 27]. Group spatial ICA (GICA) stands out as the leading approach for multi-subject analysis of task- and resting-state fMRI data [28], building on the assumption that the spatial map components ($S$) are common (or at least similar) across subjects. Another approach, called joint ICA (jICA) [29], is popular in the field of multimodal data fusion and assumes instead that the mixing process ($\mathbf{S}$) over a group of subjects is common between a pair of data modalities.

### 1.1.3 Nonlinear Methods

PCA, ICA, and NMF are all ultimately linear methods, and any nonlinear interactions that are modelled within data are thus not accounted for. Methods such as Nonlinear ICA [30, 31, 32] utilize nonlinear similarity metrics such as Mutual Information [33], or inherently nonlinear models such as Neural Networks to estimate nonlinear interactions between statistically indepdendent components. More recently, variational auto-encoders [34], which learn a set of latent variables to generate a probability distribution which contains input data, have become especially popular for nonlinear representation learning of neuroimaging data [35, 36, 37, 38], and multimodal fusion [39].

## 1.2 Federated Learning

Federated learning is a relatively new concept in the machine learning field, emerging from an increased concern for the privacy of data in involved in training machine learning mod-

els, as well as from a demand for increased statistical power in models from larger and more complex data, which can be difficult to store in centralized locations. Although the term was coined in 2016 papers from Google Research [40, 41], the practice existed in under the name "decentralized" learning or privacy-sensitive learning in fields such as Neuroimaging as early as 2014 [42, 43, 44], where the problem of data-sharing was especially relevant.

Although federated learning methods vary widely, the common feature between the majority is that individually identifiable samples are kept at local data-collection sites, with only intermediate statistics shared during training [44]. Additional privacy assurances can then be applied to shared statistics [44, 45] such as differential privacy [46].

A number of federated learning methods have been invented for neuroimaging in particular [44, 47], such as distributed joint and group ICA [43, 48], which are presented as part of this dissertation, distributed dynamic functional network connectivity (ddFNC) [49], distributed independent vector analysis [50], distributed multi-layer perceptrons [51], and distributed stochastic neighbor embeddings [52, 53].

### 1.2.1 Federated Deep Learning

Deep Learning models are notoriously data-hungry, easily succumbing to the curse of dimensionality and other issues when not enough samples are available. Federated learning has thus exploded in interest with deep learning models in particular [54, 55, 56], including a number of methods applied to healthcare [57].

Most relevant work in distributed deep learning focuses on gradients as the primary shared statistic, and applies techniques such as dropping unnecessary values via sparsification [58, 59, 60, 61, 62, 63, 64], mapping values into bins via quantization [65, 66, 67, 68, 69], or otherwise compressing gradients [70, 71, 72]. Although these methods focus on gradients, the majority of them are applicable to any shared statistic which can be used for learning. Since we focus on the more fundamental question of what statistics are being shared, these methods are potentially synergistic with ours.

A second class of distributed deep learning method reduces bandwidth by following an update schedule, so gradients are not shared for every batch or epoch. These methods, such as bursty aggregation [73], lazy aggregation [74], periodic averaging [75], and other scheduling strategies [60, 76, 77], are again agnostic to the actual statistic shared—as long as the statistic can be used for learning. With the exception of methods which average statistics during training, the methods proposed here should be entirely compatible with any update schedule desired, since the auto-differentiation statistics we share can be used to reconstruct gradients at any point during training.

Recently, methods exploiting optimization algorithms such as auto-differentiation aim to provide more general federated learning algorithms. We released a preprint in 2021 [78] along with three accepted workshop submissions which present our method for distributed auto-differentiation, exploiting the inherent low-rank outer-product structure. A new preprint released by google [79] also claims to do federated auto-differentiation; however, their federation is relegated only to the sum operations at work in auto-differentiation.

### 1.2.2 Rank-Based Federated Deep Learning

Another class of federated deep learning follows approaches from rank-based decomposition. Methods like PowerSGD [80], uses a power iterations of local gradients to estimate two low-rank matrices which can be used to reconstruct a low-rank approximation of the gradient. Although PowerSGD is a gradient compression method on its face, the sharing of low-rank Q and R matrices does represent a shift away from sharing raw gradients. Indeed, for a chosen low rank $r$, PowerSGD is able to achieve a bandwidth per layer of $\Theta(r(h_i + h_{i+1}))$ for hidden layer sizes $h_i$ and $h_{i+1}$. PowerSGD, which achieves a bandwidth reduction via the path of QR decomposition rather than auto-differentiation, thus represents a conceptually closest alternative to our method. It will be our goal in this work to illustrate the benefits we receive by taking the path of auto-differentiation both in terms of mathematical intuition, model performance, and bandwidth reduction.

As mentioned above, our method for distributed auto-differentiation [78] exploits the low-rank structure of the gradient computed in deep learning models to allow for an efficient federated learning algorithm.

### 1.3 The Role of Deep Learning Rank

In the preliminary work for this dissertation, we took an approach to federated learning which exploited the rank of certain matrices involved in optimization in order to create efficient algorithms. In methods like ICA, the rank of the input is assumed by the researcher, but in neural networks, the low rank of the gradient emerges naturally as an artifact of the learning process. The theoretical contribution of our work provides a perspective into the role of rank in training dynamics of artificial neural networks.

While Neural Networks have been widely and successfully applied to a number of problem settings, no comprehensive theory exists for describing training dynamics, explanations of model inference, and other important insights. Andrew Saxe's solutions to the differential equations which describe training and semantic learning in deep neural networks [81, 82] provides an intriguing first picture of how deep these networks learn. Indeed, the results of Saxe's work indicate that rank may play a pivotal role in these dynamics, as his work shows dominant modes of input-output covariance learned in order during training. In a separate body of work, Tishby et al. [83, 84] have described an information-theoretic interpretation of deep learning (which has gained some criticism from Saxe [85]), in which deep neural networks enter into the information-bottleneck paradigm, trading off the amount of input and output information represented in the latent space. Although Tishby's work is also not explicitly related to rank, there may be connections via the information bottleneck theory itself [86] even if some of Tishby's findings are problematic.

A more directly relevant work analyzing the role of rank in deep learning training comes from work on the so-called "Neural Collapse" phenomenon [87] in classification. According to the original 2020 work, several distinct characteristics of terminal training in deep

neural networks were observed empirically: 1) loss of cross-example within class variable, 2) collapse of class means to a simplex, 3) last layer classifiers converge to a simplex, 4) classifier decisions collapse to whichever class has the closest training mean. So far in the literature, this phenomenon has primarily been analyzed theoretically via so-called "unconstrained features" [88, 89, 90, 91, 92], where the features of a deep neural network are treated as free optimization variables, allowing the underlying problem to be viewed as a matrix factorization problem. More recent work has shown that Neural Collapse does not need to make the assumption of unconstrained features, with neural collapse emerging naturally in the dynamics of SGD with mean squared error loss [93]. Our theoretical work is in the same class of this most recent work; however, our auto-differentiation first approach, along with our analysis of linear networks, provides an intuitive picture of how rank plays a unique role in neural collapse.

[obeyspaces]url

# CHAPTER 2

# DECENTRALIZED TEMPORAL INDEPENDENT COMPONENT ANALYSIS: LEVERAGING FMRI DATA IN COLLABORATIVE SETTINGS

## 2.1 Introduction

The benefits of collaborative analysis on fMRI data are deep and far-reaching. Research groups studying complex phenomena (such as mental disorders) often gather data with the intent of performing specific kinds of analyses. However, researchers can often leverage the data gathered to investigate questions beyond the scope of the original study. For example, a study focusing on the role of functional connectivity in mental health patients may collect a brain scan using magnetic resonance imaging (MRI) from all enrolled subjects, but may only examine one particular aspect of the data. The scans gathered for the study, however, are often saved to form a data set associated with that study—they therefore remain available for use in future research. This phenomenon often results in the accumulation of vast amounts of data, distributed in a decentralized fashion across many research sites. In addition, since technological advances have dramatically increased the complexity of data per measurement while lowering their cost, researchers hope to leverage data across multiple research groups to achieve sufficiently large sample sizes that may uncover important, relevant, and interpretable features that characterize the underlying complex phenomenon.

The standard industry solution to data sharing involves each group uploading data to a shared-use data center, such as a cloud-based service like the OpenfMRI data repository [94] or the more-recently proposed OpenNeuro service [95]. Despite the prevalence of such frameworks, centralized solutions may not be feasible for many research applications. For example, since neuroimaging uses data taken from human subjects, data sharing may be limited or prohibited due to issues such as (i) local administrative rules, (ii) lo-

cal desire to retain control over the data until a specific project has reached completion, (iii) a desire to pool together a large external dataset with a local dataset without the computational and storage cost of downloading all the data, or (iv) ethical concerns of data re-identification. The last point is particularly acute in scenarios involving genetic information, patient groups with rare diseases, and other identity-sensitive applications. Even if steps are taken to assure patient privacy in centralized repositories, the repository maintainers are often forced to deal with monumental tasks of centralized management and standardization. This can require many hours of additional processing, occasionally reducing the richness of some of the contributed data [96].

In lieu of centralized sharing techniques, a number of practical decentralization approaches have recently been proposed by researchers looking to perform privatized analyses. For example, the "enhancing neuroimaging genetics through meta analysis" (ENIGMA) consortium [97] allows groups to share local summary statistics rather than gathering all the original imaging data at a single site for a centralized analysis. This method has proven very successful when using both mega- and meta-analysis approaches [97, 98, 99, 100]. Particularly, the meta-analysis at work in ENIGMA has been used for large-scale genetic association studies, with each site performing the same analysis, the same brain measure extraction, or the same regressions, and then aggregating local results globally. Meta-analyses can summarize findings from tens of thousands of individuals, so the summaries of aggregated local data need not be subject to institutional firewalls or even require additional consent from subjects [100, 101]. This approach represents one proven, widely used method for enabling analyses on otherwise inaccessible data.

Although ENIGMA has spurred innovation through massive international collaborations, there are some challenges which complicate the approach. Firstly, the meta-analyses at work in ENIGMA are effectively executed manually: a very time-consuming process. For each experiment, researchers have to write analysis scripts, coordinate with personnel at all participating sites to make sure these scripts are implemented there, adapt and debug

scripts at each site, and then gather the results through the use of proprietary software. In addition, an analysis using the ENIGMA approach described above is typically "single-shot," i.e., it does not iterate among sites to compute results holistically, as informed by the global data. From a statistical and machine learning perspective, single-shot model averaging has asymptotic performance with respect to the number of subjects for some types of analysis [102, 103]. However, simple model averaging does not account for variability between sites driven by small sample sizes and cannot leverage multivariate dependence structures that might exist across sites. Furthermore, the ability to iterate over local site computations allows not only continuous refinement of the solution at the global level but also greater algorithmic complexity, enabling multivariate approaches like group ICA [104] and support vector machines [105], and increased efficiency due to parallelism, facilitating the processing of images containing thousands of voxels.

These, together with the significant amount of manual labor required for single-shot approaches to decentralization, motivates decentralized analyses which favor more frequent communication. For example, sites running a global optimization algorithm can communicate following each iteration or after a number of iterations. In this paper, we further previous work in this direction [43] to develop iterative algorithms for collaborative, decentralized feature learning. Namely, we implement a real-data application of a successful algorithm for decentralized independent component analysis (ICA), a widely-used method in neuroimaging applications. Specifically, we show that our decentralized implementation can help further advance the as-of-yet mostly unexplored domain of temporal ICA of functional magnetic resonance imaging (fMRI) data. The resulting method is a ready fit for decentralized collaboration frameworks, such as the COINSTAC neuro-imaging analysis platform [105], which promises innovation in privacy-sensitive decentralized analysis.

Decentralized approaches such as ENIGMA allow research sites to maintain control over data access, thus providing plausible privacy protection at the cost of additional labor in implementing and updating a distributed architecture. For many applications, keeping

data stored on sites without transfer of entire data samples may provide substantial privacy. These decentralized methods, however, are amenable to quantifiable measures of privacy, such as differential privacy [46]. In this work, we leave the addition of differential privacy aside, and focus on the presentation of djICA as a separate algorithm first, with plausible privacy; however, we have pursued the addition of differential privacy to djICA elsewhere [106].

One widespread analysis which stands to benefit from decentralization is temporal independent component analysis (tICA). In resting-state fMRI studies, we can assume that the overall spatial networks remain stable across subjects and experiment duration, while the activation of certain neurological regions varies over time and across subjects. Temporal ICA, first utilized for fMRI by Biswal et al. [17], locates temporally independent components corresponding to independent activations of a subjects' intrinsic common spatial networks [107]. Both spatial and temporal ICA evidently provide reliable estimates of these intrinsic networks from fMRI data [108, 109, 110, 111], but, unlike its spatial counterpart, temporal ICA allows spatial correlation between them (i.e. overlaps in the spatial maps) [112]. Spatial and temporal ICA can result in similar estimated networks [113, 108, 114, 109], while temporal ICA provides estimates not otherwise available to spatial ICA [111, 115], specifically for task-related data. Temporal ICA has also proven particularly useful for extracting information from high-resolution fMRI scans with overlapping spatial activations, a feature not available to spatial ICA [116]. Beyond estimation of novel temporal components, temporal ICA can also aid in isolating and removing noise from fMRI signals [117, 118].

While useful, the existing literature for temporal ICA is limited. This can be partially attributed to computational complexity and dependence on statistical sample size, since temporal ICA requires more data points in the time dimension than the typical fMRI time series can offer [108, 109]. Specifically, the ratio of the spatial to the temporal dimension often requires the temporal dimension to be at least similar to the voxel dimension. This

often motivates the temporal aggregation of datasets composed of many temporally concatenated subjects. This temporal aggregation is also a key feature of the well-established group spatial ICA in the fMRI literature [119, 120, 14]. Beyond accumulation of subjects, other studies implementing temporal ICA for fMRI utilize higher-resolution scans to perform temporal ICA with fewer subjects [116]. Further methods reduce the spatial dimension to make a temporal ICA tractable: Seifritz et al. [121] use an initial spatial ICA to reduce spatial dimensional by locating a region of interest on which to perform temporal ICA, and Van et al. restrict the temporal analysis to a predetermined region of voxels deemed relevant to their particular problem of speech pattern monitoring [122].

Although temporal ICA would benefit tremendously from increasing the temporal frequency of scanners, or analyzing a large number of subjects at a central location, as mentioned above, this is not always feasible. To overcome the challenges of centralized temporal ICA, we present a novel method, decentralized joint Independent Component Analysis (djICA), which allows for the computation of aggregate spatial maps and local independent time courses across decentralized data stored at different servers belonging to independent labs. Our approach combines individual computations performed locally with global processes to obtain both local and global results. The resulting method for temporal ICA produces results with similar performance to the pooled-data case and provides estimated components in line with previous literature, demonstrating the effectiveness of decentralized collaborative algorithms for this difficult task.

In sum, the contributions of this paper are as follows:

- In Section 2.2, we present decentralized joint independent component analysis (algorithm 1, Section 2.2.2), which is closely related to Infomax ICA (Section 2.2.1) with decentralized PCA preprocessing (Section 2.2.3).

- In Section 2.3 we include experiments and evaluation of djICA over different subject and site distributions for simulated data sets, including simulated fMRI data, thus providing a baseline result and proper motivation for real-data experiments.

- In Section 2.4, we perform experiments which evaluate djICA on a real set of fMRI data in a simulated decentralized environment, using a novel pseudo-ground-truth evaluation scheme to compare our results with the pooled case.

- Finally, in Section 2.5, we discuss the performance of djICA as a novel method for performing temporal ICA in decentralized settings, comparing our results with previously estimated results from the pooled temporal ICA literature.

## 2.2 Materials and Methods

In this section, we provide the details of our method for decentralized joint independent component analysis and provide a basis for its evaluation. We first review Independent Component Analysis for the pooled case (where all samples are located on a single site) in Section 2.2.1, which provides basis for our presentation of the djICA algorithm in section 2.2.2. In section 2.2.3 we discuss performing PCA preprocessing in a decentralized setting, and finally, in section 2.2.4, we discuss our methods for evaluating the djICA algorithm. The code used for evaluation is available on GitHub[1], and its inclusion in the COINSTAC decentralized analysis framework is currently ongoing.

### 2.2.1 Independent Component Analysis

ICA is a popular blind source separation (BSS) method which attempts to decompose mixed signals into independent components (ICs), or sources, without prior knowledge of the structure of those sources. Empirically, ICA applied to brain imaging data produces robust features which are physiologically interpretable and markedly reproducible across studies [14, 15, 16, 17]. Indeed, while justification for successful ICA of fMRI results had been previously attributed to sparsity alone [18], it has been shown that statistical independence between the underlying sources is in fact a key driving mechanism of ICA algorithms [19], with additional benefits possible by trading off between the two [20].

[1] https://github.com/MRN-Code/djica_paper_code_release

In linear ICA, we model a data matrix $\mathbf{X} \in \mathbb{R}^{d \times N}$ as a product $\mathbf{X} \approx \mathbf{SA}$, where $\mathbf{A} \in \mathbb{R}^{r \times N}$ is composed of $N$ observations from $r$ *statistically independent* components, each representing an underlying signal source. Thus, we can interpret ICA in terms of this generative model, with independent sources $\mathbf{A}$ submitted to a linear mixing process described by a mixing matrix $\mathbf{S} \in \mathbb{R}^{d \times r}$, forming the observed data $\mathbf{X}$. Most ICA algorithms seek to recover the "unmixing matrix" $\mathbf{W} = \mathbf{S}^{-1}$ (or in the case where $\mathbf{S}$ is not square, the pseudo-inverse, $\mathbf{S}^{+}$), by maximizing independence between rows of the product $\mathbf{WX}$, assuming the matrix $\mathbf{S}$ is invertible.

Maximal information transfer (Infomax) [21] is a popular heuristic for estimating $\mathbf{W}$ by maximizing an entropy functional related to $\mathbf{WX}$. This can alternatively be interpreted as a Bayesian estimator with a super-Gaussian prior on the density of the sources. More precisely, with some abuse of notation, let

$$g(z) = \frac{1}{1 + e^{-z}} \tag{2.1}$$

be the sigmoid function with $g(\mathbf{Z})$ being the result of element-wise application of $g(\cdot)$ on the entries of a matrix or vector $\mathbf{Z}$. The differential entropy of a random vector $Z$ with joint density $p$ is

$$h(Z) = -\int p(Z) \log p(Z) dZ. \tag{2.2}$$

The objective of Infomax ICA then becomes

$$\widehat{\mathbf{W}} = \underset{\mathbf{W}}{\operatorname{argmax}}\, h(g(\mathbf{WX})). \tag{2.3}$$

Another class of algorithms includes the famous family of fixed-point methods such as Fast ICA [22, 23, 24]. These locally optimize a "contrast" function such as kurtosis or negentropy.

ICA, along with other methods for BSS, has found wide application. In particular, functional magnetic resonance imaging (fMRI) and other biomedical imaging data use ICA models to interpret subject imaging data [16]. For fMRI, many models assume that functionally connected regions in the brain are systematically nonoverlapping. ICA has been used in applications ranging from interpreting physiology to analyzing task-related signals in both the spatial and temporal domains.

Additionally, a number of extensions of ICA exist for the purpose of jointly analyzing multiple data sets to perform a simultaneous decomposition across a large number number of subjects and different modalities [25, 26, 27]. Group spatial ICA (GICA) stands out as the leading approach for multi-subject analysis of task- and resting-state fMRI data [28], building on the assumption that the spatial map components ($\mathbf{A}$) are common (or at least similar) across subjects. Another approach, called joint ICA (jICA) [29], is popular in the field of multimodal data fusion and assumes instead that the mixing process ($\mathbf{S}$) over a group of subjects is common between a pair of data modalities.

A largely unexplored area of fMRI research is group temporal ICA, which, like spatial ICA, assumes common spatial maps but with statistically independent timecourses. Group temporal ICA has been most commonly applied to EEG data [123] but less frequently to fMRI data. Consequently, like jICA, in the fMRI case, the common spatial maps from temporal ICA describe a common mixing process ($\mathbf{S}$) among subjects. However, temporal ICA of fMRI is not typically investigated because the small number of time points in each data set can lead to unreliable estimates. Our decentralized jICA (djICA) approach overcomes that limitation by leveraging information from data sets distributed over multiple sites. This is an important extension of single-subject temporal ICA and a further example of methods which can benefit from leveraging data in collaborative settings.

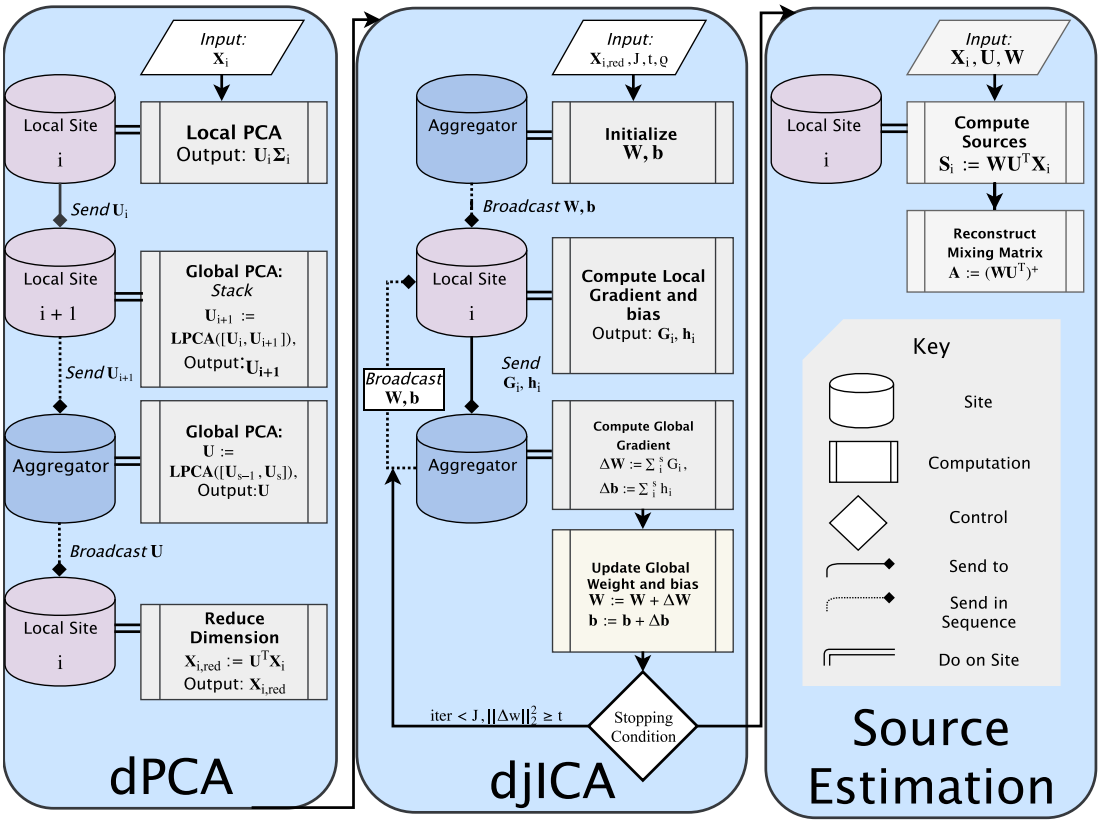



Figure 2.1: Figure reprinted with permission. An overview of the djICA pipeline. Each panel in the flowchart represents one stage in the pipeline and provides an overview of the processes done on local sites and on the aggregator site, as well as communication between nodes. The dPCA panel corresponds to Algorithms 2 and 3, the djICA panel corresponds to Algorithm 1, and the Source Estimation panel corresponds to the procedure for computing local sources given in equation (2.5). On each panel, local site $i$ is an arbitrary site in the decentralized network, and local site $i+1$ represents the next site in a given ordering over the decentralized network. *Broadcast* communication sends data to all sites, and *Send* communication sends data to one site. Lines with an arrowhead indicate procedural flow. Lines with diamond endpoints indicate communication flow. Dotted lines with diamond endpoints indicate that the sending process occurs iteratively to neighbors until the aggregator is reached, or in the case of broadcasting, indicates that all nodes receive the latest update. Double-lines indicate site-specific computations.

### 2.2.2 Decentralized Joint ICA

Our goal in this paper is to show that the decentralized joint ICA algorithm can be applied to decentralized fMRI data and produce meaningful results for temporal ICA. We present djICA in detail here and provide notation in table.

For an integer $n$ let $[n] = \{1, 2, \ldots, n\}$. Suppose that we have $s$ total sites indexed by

Table 2.1: Figure reprinted with permission. A summary of important notation used throughout this paper, especially in Algorithms 1, 2, and 3.

| $\mathbf{X}$ | $\mathbf{A}$ | $\mathbf{S}$ | $\mathbf{U\Sigma V}$ |
|---|---|---|---|
| Data Matrix | Source Matrix | Mixing Matrix | SVD results |
| $\mathbf{X}_i$ | $\mathbf{X}_{i,red}$ | $U_i$ | $\mathbf{G}_i(j)$ |
| Data Site $i$ | Reduced Data Site $i$ | Eigenvectors Site $i$ | Gradient Site $i$ |
| $\Delta_{\mathbf{W}}(j)$ | $\mathbf{W}(j)$ | $\mathbf{b}(j)$ | $(j)$ |
| Weight Update | Weight Matrix | Bias | iter $j$ |
| $s$ | $r$ | $d$ | $N$ |
| # sites | rank (# ICs) | # rows | # cols |
| $\rho$ | $w_{max}$ | $\theta_{max}$ | $\alpha$ |
| learning rate | max weight | max angle | anneal rate |

$[s]$; each site $i \in [s]$ has a data matrix $\mathbf{X}_i \in \mathbb{R}^{d \times N_i}$ consisting of a total time course of length $N_i$ time points over $d$ voxels. Let $N = \sum_{i=1}^{s} N_i$ be the total length. We model the data at each site as coming from a common (global) mixing matrix $\mathbf{S} \in \mathbb{R}^{d \times r}$ applied to local data sources $\mathbf{A}_i \in \mathbb{R}^{r \times N_i}$. Thus, the total model can be written as

$$\mathbf{X} = [\mathbf{S}\mathbf{A}_1 \; \mathbf{S}\mathbf{A}_2 \; \cdots \; \mathbf{S}\mathbf{A}_s] \in \mathbb{R}^{d \times N.} \tag{2.4}$$

Our algorithm, decentralized joint ICA (djICA), uses locally computed gradients to estimate a common, global unmixing matrix $\mathbf{W} \in \mathbb{R}^{r \times d}$ corresponding to the Moore-Penrose pseudo-inverse of $\mathbf{S}$ in (2.4), denoted $\mathbf{S}^+$.

Figure 2.1 summarizes the overall algorithm in the context of temporal ICA for fMRI data. Each site $i$ has data matrices $\mathbf{X}_{i,m} \in \mathbb{R}^{d \times n_i}$ corresponding to subjects $m \in [M_i]$ with $d$ voxels and $n_i$ time samples. Sites concatenate their local data matrices temporally to form a $d \times n_i M_i$ data matrix $\mathbf{X}_i$, so the total time course length at site $i$ is $N_i = n_i M_i$, and the total number of subjects is $M = \sum_{i=1}^{s} M_i$. Each site performs local PCA (Algorithm 2) using the singular value decomposition (SVD), with matrices $\mathbf{U}_i \in \mathbb{R}^{d \times k}$ and $\mathbf{\Sigma}_i \in \mathbb{R}^{k \times k}$ corresponding to the top $k$ singular vectors and values, respectively. Then, in a decentralized principal component analysis (dPCA) framework, the sites approximate a

global PCA (Algorithm 3) to form a common $r$-dimensional projection matrix $\mathbf{U} \in \mathbb{R}^{d\times r}$. This approach is an adaptation of the sub-sampled time PCA (STP) method [124] to the case of decentralized data, offering an accurate bandwidth-efficient alternative to other dPCA algorithms [125] which can compute the global $\mathbf{U}$ directly (without local PCA) but at the expense of communicating a large $d \times d$ matrix between sites. Finally, all sites project their data onto the subspace corresponding to $\mathbf{U}$ to obtain reduced local datasets $\mathbf{X}_{i,\text{red}} \in \mathbb{R}^{r\times N_i}$.

The projected data is the input to the iterative djICA algorithm that estimates the unmixing matrix $\mathbf{W} \in \mathbb{R}^{r\times r}$, as described in Algorithm 1. The full mixing matrix for the global data is modeled as $\mathbf{S} \approx (\mathbf{W}\mathbf{U}^\top)^+ \in \mathbb{R}^{d\times r}$. After initializing $\mathbf{W}$ (for example, as the identity matrix), the djICA algorithm iteratively updates $\mathbf{W}$ using a distributed natural gradient descent procedure [126]. At each iteration $j$ the sites update locally. In lines 5 and 6, the sites adjust the local source estimates $\mathbf{Z}_i = \mathbf{W}\mathbf{X}_{i,\text{red}}$ by their bias estimates $\mathbf{b}(j-1)\mathbf{1}^\top \in \mathbb{R}^{r\times N_i}$, followed by the sigmoid transformation $g(\cdot)$; then, local gradients are computed with respect to $\mathbf{W}_i$ and $\mathbf{b}_i$ in lines 7 and 8. Here, $\mathbf{y}_{l,i}(j)$ is the $l$-th column of $\mathbf{Y}_i(j)$. The sites then send their local gradient estimates $\mathbf{G}_i(j)$ and $\mathbf{h}_i(j)$ to an aggregator site, which aggregates them according to lines 11-13. After updating $\mathbf{W}(j)$ and $\mathbf{b}(j)$, the aggregator checks if any values in $\mathbf{W}(j)$ increased above an upper bound of $w_{\max} = 10^9$ in absolute value. If so, the aggregator resets the global unmixing matrix, sets the current iteration to $j = 0$, and anneals the learning rate by $\rho = 0.9\rho$. Otherwise, before continuing, if the angle between $\Delta_{\mathbf{W}}(j)$ and $\Delta_{\mathbf{W}}(j-1)$ is above $\theta_{\max} = 60°$, the aggregator anneals the learning rate by $\rho = 0.9\rho$, preventing $\mathbf{W}$ from changing too quickly without learning the structure of data. The aggregator sends the updated $\mathbf{W}(j)$ and $\mathbf{b}(j)$ back to the sites. Finally, the algorithm stops when $\|\Delta_{\mathbf{W}}(j)\|_2^2 < t$, and each site recovers the statistically independent source estimates $\mathbf{A}_i$ by

$$\mathbf{A}_i \approx \mathbf{W}\mathbf{X}_{i,\text{red}}. \tag{2.5}$$

**Algorithm 1** Figure reprinted with permission. decentralized joint ICA (djICA)

**Require:** data $\{\mathbf{X}_{i,\text{red}} \in \mathbb{R}^{r \times N_i : i \in [s]}\}$, where $r$ is the same across sites, tolerance level $t = 10^{-6}$, $j = 0$, maximum iterations $J = 1024$, initial learning rate $\rho = 0.015/\ln(r)$, maximum weight entry $w_{\max} = 10^9$ , maximum angle $\theta_{\max} = 60°$, annealing rate $\alpha = 0.9$

Initialize $\mathbf{W}(0) \in \mathbb{R}^{r \times r}$ ▷ for example, $\mathbf{W}(0) = \mathbf{I}$
**for** $j < J$ **and** $\|\Delta_{\mathbf{W}}(j)\|_2^2 \geq t$ **do**
  $j = j + 1$
  **for all** sites $i = 1, 2, \ldots, s$ **do**
    $\mathbf{Z}_i(j) = \mathbf{W}(j-1)\mathbf{X}_{i,\text{red}} + \mathbf{b}(j-1)\mathbf{1}^\top$
    $\mathbf{Y}_i(j) = g(\mathbf{Z}_i(j))$
    $\mathbf{G}_i(j) = \rho\left(\mathbf{I} + (\mathbf{1} - 2\mathbf{Y}_i(j))\mathbf{Z}_i(j)^\top\right)\mathbf{W}(j-1)$
    $\mathbf{h}_i(j) = \rho \sum_{l=1}^{N_i}(\mathbf{1} - 2\mathbf{y}_{l,i}(j))$
    Send $\mathbf{G}_i(j)$ and $\mathbf{h}_i(j)$ to the aggregator site.
  **end for**
  At the aggregator site, update global variables
  $\Delta_{\mathbf{W}}(j) = \sum_{i=1}^{s} \mathbf{G}_i(j)$
  $\mathbf{W}(j) = \mathbf{W}(j-1) + \Delta_{\mathbf{W}}(j)$
  $\mathbf{b}(j) = \mathbf{b}(j-1) + \sum_{i=1}^{s} \mathbf{h}_i(j)$
  Check Upper-Bound Conditions
  **if** $w_{i,j} \in \mathbf{W}, |w_{i,j}| > w_{\max}$ **then** ▷ $\mathbf{W}$ has blown-up
    Re-Initialize $\mathbf{W}(0)$, $j = 0$, $\rho = \alpha\rho$
  **else if** $\angle(\Delta_{\mathbf{W}}(j), \Delta_{\mathbf{W}}(j-1)) > \theta_{\max}$ **then**
    $\rho = \alpha\rho$ ▷ Prevent $\mathbf{W}$ from changing too quickly
  **end if**
  Broadcast global $\mathbf{W}(j)$ and $\mathbf{b}(j)$ to all sites.
**end for**

For the pooled-data case, Amari et al. [127] demonstrate theoretically that Infomax ICA meets with the conditions that guarantee convergence of $\mathbf{W}$ to an asymptotically stable solution as long as $A^{-1}$ is also asymptotically stable. In other words, the natural gradient provides convergence to an equilibrium point corresponding to a local minimum; however, in the general case for Infomax ICA, it is unfortunately not possible to assure convergence to a global minimum, i.e. complete separation of the source signals.

In the decentralized-data case, djICA converges to the solution of the pooled-data case: the assumption of a common mixing matrix across subjects assures that the global gradient sum is identical to the pooled-data gradient on average, likewise moving the global weight matrix towards convergence.

Table 2.2: Figure reprinted with permission. A summary of the hyper-parameters used in all experiments, for both simulations, and real-data scenarios.

| param. | $t$ | $J$ | $\rho$ | $w_{max}$ | $\theta_{max}$ | $\alpha$ |
|---|---|---|---|---|---|---|
| value | $10^{-6}$ | 1024 | 0.015/ln$(r)$ | $10^9$ | 60° | 0.9 |

Indeed, since the global iterates of djICA are taken as the average of the individually computed, on-site gradients, djICA run on a full-batch case (where each site has access to the full batch of data) is equivalent to the pooled version of infomax ICA. We show this empirically in section 1 of the supplementary material included with this work.

For our purposes, we chose the hyper-parameter values as specified in the "Required" parameters for Algorithm 1, and we utilized the stochastic version of the algorithm with block size $b = \left\lfloor \sqrt{\frac{\min(N_i)}{20}} \right\rfloor$, where $\min(N_i)$ is the minimum number of concatenated time-points across all sites. We summarize these parameters in Table 2.2.

### 2.2.3 PCA preprocessing

Here, we describe the decentralized principal component analysis (dPCA) algorithms used for dimension reduction and whitening in the djICA pipeline. The dPCA algorithm is a pre-processing step that standardizes the data prior to djICA and should also be decentralized so that the benefits of using a decentralized joint ICA are not made moot by dependence on a previous pooled step. There are many approaches to approximating the global PCA with a distributed algorithm [128].

We first chose to examine dPCA from Bai et al. [125]. Their proposed dPCA algorithm bypasses local data reduction, and thus works directly with the full data, which motivates its choice for some of our simulated experiments. One major downside of their approach, however, is that it requires the transfer of a large orthogonal matrix between all sites, thus increasing bandwidth usage significantly. As an alternative to the approach presented by Bai et al., a two-step dPCA approach was considered based on the STP approach [129] recently developed for large PCA of multi-subject fMRI data. One advantage of this approach is that only a small matrix $\mathbf{P} \in \mathbb{R}^{d \times k}$ is transmitted from one site to another, a

significant decrease compared to the large $d \times d$ matrix [125]. The downside is that there are no bounds on the accuracy of the final $\mathbf{U}$ and results can vary slightly with the order in which sites and subjects are processed. Nonetheless, our results suggest that the two-step dPCA approach, described in Algorithms 2 and 3, yields a fairly good estimate of $\mathbf{U}$. In principle, any suitable decentralized PCA algorithm could replace the two methods tested here. Thus, we leave room for future improvements of our framework to find the most effective dPCA approach for the djICA pipeline.

**Algorithm 2** Figure reprinted with permission. Local PCA algorithm (LocalPCA)

**Require:** data $\mathbf{X} \in \mathbb{R}^{d \times N}$ and intended rank $k$

1: Compute the SVD $\mathbf{X} = \mathbf{U}\mathbf{\Sigma}\mathbf{V}$.
2: Let $\mathbf{\Sigma}^{(k)} \in \mathbb{R}^{k \times k}$ contain the largest $k$ singular values and $\mathbf{U}^{(k)} \in \mathbb{R}^{d \times k}$ the corresponding singular vectors.
3: Save $\mathbf{U}^{(k)}$ and $\mathbf{\Sigma}^{(k)}$ locally and return $\mathbf{P} = \mathbf{U}^{(k)}\mathbf{\Sigma}^{(k)}$.

**Algorithm 3** Figure reprinted with permission. Global PCA algorithm (GlobalPCA)

**Require:** $s$ sites with data $\{\mathbf{X}_i \in \mathbb{R}^{d \times N_i : i=1,2,\ldots,s}\}$, intended final rank $r$, local rank $k \geq r$.

1: Choose a random order $\pi$ for the sites.
2: $\mathbf{P}(1) = \text{LocalPCA}(\mathbf{X}_{\pi(1)}, \min\{k, \text{rank}(\mathbf{X}_{\pi(1)})\})$
3: **for all** $j = 2, 3, \ldots, s$ **do**
4: Set site index $i = \pi(j)$
5: Send $\mathbf{P}(j-1)$ from site $\pi(j-1)$ to site $\pi(j)$
6: $k' = \min\{k, \text{rank}(\mathbf{X}_i)\}$
7: $\mathbf{P}' = \text{LocalPCA}(\mathbf{X}_i, k')$
8: $k' = \max\{k', \text{rank}(\mathbf{P}(j-1))\}$
9: $\mathbf{P}(j) = \text{LocalPCA}([\mathbf{P}' \; \mathbf{P}(j-1)], k')$
10: **end for**
11: $r' = \min\{r, \text{rank}(\mathbf{P}(s))\}$
12: $\mathbf{U} = \text{NormalizeTopColumns}(\mathbf{P}(s), r')$ ▷ At last site
13: Send $\mathbf{U}$ to sites $\pi(1), \ldots, \pi(s-1)$.
14: **for all** sites $i = 1, 2, \ldots, s$ **do**
15: $\mathbf{X}_{i,\text{red}} = \mathbf{U}^\top \mathbf{X}_i$ ▷ The locally reduced data
16: **end for**

Algorithm 3 uses a peer-to-peer scheme to iteratively refine $\mathbf{P}(j)$, with the last site broadcasting the final $\mathbf{U}$ to all sites. $\mathbf{U}$ is the matrix containing the top $r'$ columns of $\mathbf{P}(s)$ with largest $L_2$-norm, but normalized to unit $L_2$-norm instead. Following the recommen-

dation in Calhoun et al. [129], we set $r = 20$ and $k = 5 \cdot r$ for our simulations.

### 2.2.4 Evaluation Strategy

All of our experiments were run using the MATLAB 2007b parallel computation toolbox, on a Linux Server running Ubuntu 12.04 LTS, with a 9.6GHz processor (four Intel Xeon E7-4870 @ 2.40GHz each), a 120MB L3 cache (30MB L3 cache per processor), and 512GB of RAM. For any one experiment, we only used a maximum of 8 cores, due to a need to share the server with other researchers.

As a performance metric for our experiments we choose the Moreau-Amari [126] inter-symbol interference (ISI):

$$ISI(Q) = \frac{1}{2r(r-1)}\left[\sum_{i=1}^{r}\left(\sum_{j=1}^{r}\frac{|Q_{ij}|}{max_k|Q_{ik}|} - 1\right) + \sum_{j=1}^{r}\left(\sum_{i=1}^{r}\frac{|Q_{ij}|}{max_k|Q_{kj}|} - 1\right)\right]. \tag{2.6}$$

This is a function of the square matrix $\mathbf{Q} = \hat{\mathbf{W}}\mathbf{S}$, where $\hat{\mathbf{W}} = \mathbf{W}\mathbf{U}^{\top}$, $\mathbf{W}$ is the estimated unmixing matrix from Algorithm 1, $\mathbf{U}$ is the orthonormal projection matrix retrieved from dPCA, and $r = \mathrm{rank}(\mathbf{Q})$, i.e. the number of sources. In particular, a lower ISI measure indicates a better estimation of a set of ground-truth components.

## 2.3 Experiments with Simulated Data

First, we test djICA in a simulated environment where we can manufacture a known ground-truth and use djICA to reconstruct this ground-truth under different mixing and site configurations. For this simulated case, we explicitly construct the signal matrices, $\mathbf{A}$, and the mixing matrix $\mathbf{S}$ (using the methods described in section 2.3), such that the source matrices are statistically independent and provide, thus providing the assurance that a solution to underyling BSS problem exists. If djICA performs well in this simulated case, where a

solution is given, we can thus justify further experiments with real data, where a solution to the underlying BSS problem is not readiily available. To this end, we evaluated 5 different scenarios for synthesis and analysis of synthetic data, as summarized in Table 2.3. Based on what we have learned from these various scenarios, which include different PCA preprocessing strategies, we can construct a promising pipeline for djICAwhich can then be translated to the real data case.

Two kinds of mixing matrices $\mathbf{S}$ were used for experimentation:

1. Lower dimensional square mixing matrices were generated using MATLAB's `randn` function [130], which generates matrices whose elements are selected from an i.i.d. Gaussian distribution.

2. Higher dimensional mixing matrices were generated using the MIALab's fMRI simulation toolbox (simTB) [131]. The simTB spatial maps are intended to simulate spatial components of the brain which contribute to the generation of the simulated time course. Higher dimensional mixtures were masked using a simple circular mask which drops empty voxels outside of the generated spatial map.

For the first two scenarios indicated in Table 2.3, we generated i.i.d. Gaussian mixing matrices $\mathbf{S} \in \mathbb{R}^{r \times r}$. For the higher-dimensional problems (scenarios 3-5), we used the simTB spatial maps [131] to generate different $\mathbf{S} \in \mathbb{R}^{d \times r}$ mixing matrices.

The independent signals $\mathbf{A}_m$ were simulated using a generalized autoregressive (AR) conditional heteroscedastic (GARCH) model [132, 133], which has been shown to be useful in models of causal source separation [134] and time-series analyses of data from neuroscience experiments [134, 135], especially resting-state fMRI time courses [136, 137]. We simulated fMRI time courses using a GARCH model by generating an AR process (no moving average terms) randomly such that the AR series converges. We chose a random order between 1 and 10 and random AR coefficients $\{\alpha[\ell]\}$ such that $\alpha[0] \in [0.55, 0.8]$ and $\alpha[\ell] \in [-0.35, 0.35]$ for $\ell > 0$. For the error terms $\delta_t = \sigma_t \epsilon_t$, we used an ARMA model

driven by $\epsilon_t$ from a generalized normal distribution with shape parameter 100 (so it was approximately uniform on $[-1, 1]$) and $\sigma_t^2 = 0.1 + 0.1y[t-1]^2 + 0.75\sigma[t-1]^2$. For each of 1024 simulated subjects, we generated 20 time courses with 250 time points, each after a "burn-in" period of 20000 samples, checking that all pair-wise correlations between the 20 time courses stayed below 0.35. We generated a total of $1024$ mixed datasets for each experiment by computing $\mathbf{X}_m = \mathbf{S}\mathbf{A}_m$.

In summary, we considered the following combinations of algorithm, preprocessing, and mixing matrix: 1) pooled (centralized) temporal ICA with no preprocessing (no data reduction) and a square i.i.d Gaussian mixing-matrix, 2) djICA with no preprocessing and a square i.i.d. Gaussian mixing-matrix, 3) pooled temporal ICA with LocalPCA preprocessing (Algorithm 2) and a simTB mixing matrix, 4) djICA with dPCA from Bai et al. [125] and a simTB mixing matrix, and 5) djICA with GlobalPCA (Algorithm 3) and a simTB mixing matrix.

Table 2.3: Figure reprinted with permission. Five scenarios considered for synthesis and analysis of simulated data experiments.

| **scenario** | **algorithm** | **preprocessing** | **mixing matrix S** |
|---|---|---|---|
| 1 | ICA (pooled) | none | i.i.d. Gaussian |
| 2 | djICA | none | i.i.d. Gaussian |
| 3 | ICA (pooled) | LocalPCA | simTB map |
| 4 | djICA | One-Step dPCA [125] | simTB map |
| 5 | djICA | GlobalPCA | simTB map |

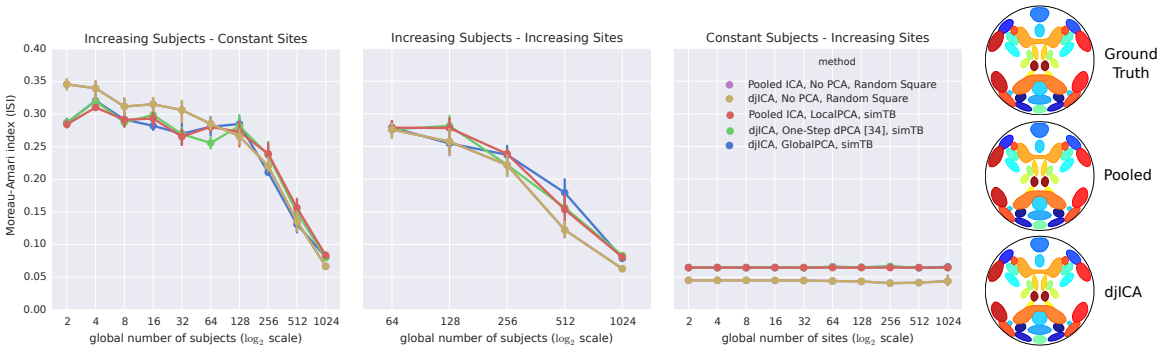
### 2.3.1 Simulation Results

In this section, the results for simulated experiments are presented. We are particularly interested in understanding how the proposed algorithm performs with different kinds of preprocessing, and how the results improve as a function of the global number of subjects, the global number of sites, or how the subjects are distributed over sites.

To test how the algorithms compare as we increase the data at a fixed number of sites, we fixed $s = 2$ sites and evaluated all five scenarios in Table 2.3, splitting the data evenly per site in the non-pooled cases. Figure 2.2a shows ISI versus the total data set size. As

the data set increases all algorithms improve and, more importantly, the distributed versions perform nearly as well as the pooled-data counterparts. Results are averaged over 10 randomly generated mixing matrices.

To test how the algorithms compare as we increase the number of sites $s$, we fix $M_i = 32$ subjects per site. Figure 2.2b demonstrates the convergence of the ISI curve with an increasing amount of combined data, with results averaged over 10 randomly generated mixing matrices. Again, we see that the performance of djICA is very close to the centralized pooled performance, even for such a small number of subjects per site.




(a) Figure reprinted with permission.

(b) Figure reprinted with permission.

(c) Figure reprinted with permission.

(d) Figure reprinted with permission.

Figure 2.2: Figure reprinted with permission. The ISI for pooled and decentralized algorithms for different distributions of subjects over sites under the five simulated scenarios indicated in Table 2.3. Panel 2.2a illustrates an increasing number of subjects over two, fixed sites. Panel 2.2b illustrates an increasing number of sites, with the number of subjects per site staying constant at 32 subjects per site, with the number of sites starting at 2 and increasing by a factor of two. Panel 2.2c illustrates 1024 total subjects distributed over an increasing number of sites. Panel 2.2d shows the 20 Ground-Truth spatial-maps, along with the estimated spatial-maps from Pooled ICA and djICA with 1024 subjects on 2 sites. In the cases with no PCA (panels 2.2a-2.2c), the pooled and decentralized algorithms perform identically.

To test how splitting the data sets across more sites affects performance, we fixed the total of $1024$ subjects and investigated the effect of splitting them over a growing number of sites $s$. Thus, the concentration of data per site $M_i$ decreased with increasing number of sites such that for small $s$ each site had more data sets and for large $s$ each site had fewer data sets. Figure 2.2c shows that the performance of djICA is very close to that of

the pooled-data ICA, even with more and more sites holding fewer and fewer data points. This implies that we can support largely decentralized data with little loss in performance.

## 2.4 Experiments with Real Data

The simulated experiments illustrate the clear benefit djICA provides by enabling the joint analysis of large decentralized data sets. In this section, we describe the methods utilized for real-data experiments with resting-state fMRI datasets. These experiments are intended to illustrate the effectiveness of djICA (Algorithm 1) in the particular domain of exploratory analysis of fMRI data. As mentioned earlier, the benefits of using this algorithm for fMRI analysis are numerous, and the experiments here aim to both highlight those benefits and illustrate the robustness of the algorithm when compared to pooled analyses.

### 2.4.1 Data Description

In this section, we describe the data sets used for real data analysis. The purpose here is to describe the preprocessing steps specific to the data utilized here. Experiments used data gathered on-site, according to the protocol in [28]. The data were collected using a 3-Tesla Siemens Trio scanner with a 12-channel radio frequency coil. T2*-weighted functional images were acquired using a gradient-echo EPI sequence with TE = 29 ms, TR = 2 s, flip angle = 75°, slice thickness = 3.5 mm, slice gap = 1.05 mm, field of view 240 mm, matrix size = 64×64, voxel size = 3.75 mm × 3.75 mm × 4.55 mm. In terms of duration, resting-state scans were a minimum of 2 min 8 s (64 volumes) long, on average 5 min 16 s (158 volumes) long, and at maximum 10 min 2 s (301 volumnes) long (see Table 2.4). In contrast to [28], subjects with greater number of time-points were retained in order to illustrate the general robustness of djICA to variation in the time-course length.

In terms of preprocessing, the data underwent rigid body alignment for head motion, slice-timing correction, spatial normalization to MNI space (using SPM5), regression of 6 motion parameters and their derivatives in addition to any trends (up to cubic or quintic),

and spatial smoothing using a 10 $mm^3$ full-width at half-maximum (FWHM) Gaussian kernel.

Table 2.4: Figure reprinted with permission. Statistics on the number of timepoints in the data set.

| min | mean | mode | median | max | range | std |
|---|---|---|---|---|---|---|
| 64 | 158 | 158 | 158 | 301 | 237 | 9 |

We also used the minimum description length (MDL) criterion [138] to estimate the number of independent components for each individual subject with the algorithm available in the MIALab's Group ICA of fMRI toolbox (GIFT) [139, 140, 104]. experiment. The median number of components over 2038 subjects was 50, and the mean was 49.4636. In all experiments, we thus elected to estimate $r = 50$ real components from the data.

### 2.4.2 Real Data with "Real" Ground-Truth

Our ultimate goal is to show that djICA can provide reasonable decentralized estimates for real fMRI components which are comparable to the pooled case. Thus, we first perform a pooled analysis in order to establish a "pseudo" ground-truth that can be used to evaluate djICA's performance on real component estimation. We estimated $r = 50$ real independent components from $M = 2038$ subjects by running a pooled instance of temporal ICA on a single site. The performance of djICA was assessed by matching the estimated decentralized components to the pooled components via the Hungarian algorithm [141] and then computing the ISI between the two sets of components. For PCA preprocessing, to avoid high communication costs, we elect to test only the GlobalPCA method given in algorithm 3.

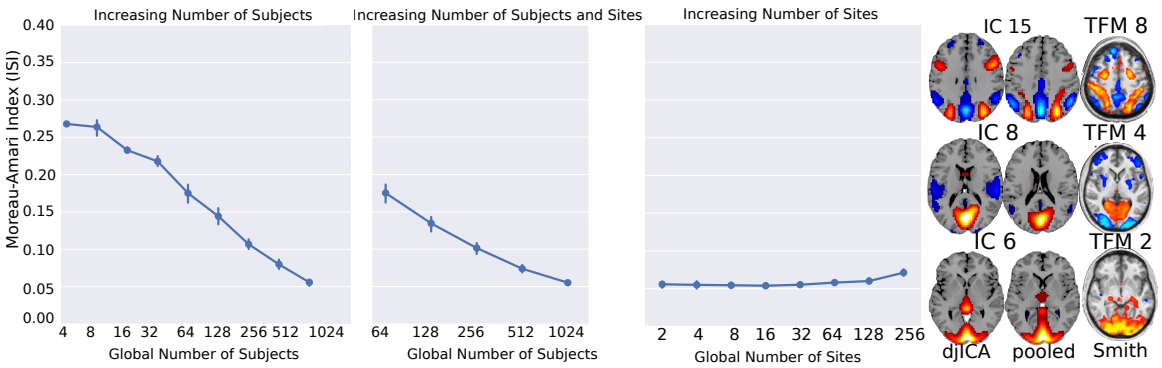
Using the pooled estimations as our basis for comparison, we then tested djICA in four distinct scenarios, varying the distribution of subjects across a network as follows:

1. when the global number of subjects in the network increases, but the number of sites in the network stays constant,

2. when the number of subjects per site stays constant, and the number of sites in the network increases,

3. when the global number of subjects in the network stays constant, but the number of sites in the network increases (subjects distributed evenly across sites), and

4. when the global number of subjects in the network stays constant, and subjects are randomly distributed across sites.




(a) Figure reprinted with permission.

(b) Figure reprinted with permission.

(c) Figure reprinted with permission.

(d) Figure reprinted with permission.

Figure 2.3: Figure reprinted with permission. The estimated ISI for real-data djICA over different distributions of subjects over sites. Panel 2.3a illustrates an increasing number of subjects over two fixed sites. Panel 2.3b illustrates an increasing number of sites, with the number of subjects per site staying constant. Panel 2.3c illustrates 1024 global subjects distributed over an increasing number of sites. Panel 2.3d shows three of the spatial maps from djICA with over 2016 subjects evenly split over 16 sites, the pooled temporal ICA "pseudo ground-truth" with 2038 subjects, and the corresponding temporal fluctuation modes (TFM) from Smith et al. [111].

For the first three scenarios, we cap the maximum number of subjects in the network at 1024 so that we can achieve an even distribution of subjects in terms of powers of 2, and so that the figures compare more directly with the simulated experiments. Furthermore, in order to get a more detailed picture of the effects of small numbers of subjects per site, we also evaluate djICA in the third scenario with a higher global number of subjects ($M = 2000$), and closely examine the results in Figure 2.4. For each of these scenarios,

we performed 10 repeated estimations of djICA components, where each run randomly assigned subjects to different sites (without duplication of subjects).

For the fourth and last scenario, 2000 subjects were randomly distributed across sites. Firstly, we selected a parametric probability distribution $P(\Theta)$ with parameters $\Theta$. We then sampled 100 different values from $P$, where each value corresponds with a potential number of subjects on the $i$-th site ($M_i$). We discarded any values below 4, so that each site has a minimum of 4 subjects per site, and took the ceiling of each real value so that site distributions are given as natural numbers. We then selected the first $s-1$ values, with $s$ being the number of sites, such that $\sum_{i=1}^{s-1} M_i < M$, where $M$ is the global number of subjects in the network. For the final site, we set $M_s = M - \sum_{i=1}^{s-1} M_i$ so that the total number of subjects in the network will remain constant at $M$. This process results in a varying number of sites between successive samplings, which we found more appealing for testing as opposed to a randomization method that would distribute a fixed number of subjects across a fixed number of sites. We also considered the effect of different distribution parameter values ($\Theta$) to assess the performance of djICA.

### 2.4.3 "Real" Ground-Truth Results

In this section, we present the results of djICA on the four experiments described above. In all cases, djICA is compared with a pooled case involving $M = 2038$ subjects, comparing across conditions using the Moreau-Amari ISI index as we did in the simulated experiments, now treating the pooled case as a our real-data "ground-truth".

*How do the estimated components compare as we increase the data, with a fixed number of sites?*

In 2.3a, we evaluate the ISI index for djICA using real-data in a scenario where the global number of subjects increases, but the number of sites in the network is fixed. This figure illustrates that as the number of subjects increases, the estimated djICA components

converge towards the components computed in the pooled case.

*How do the estimated components compare as we increase the number of sites, with a fixed amount of data sets per site?*

In 2.3b, we evaluate the ISI index for djICA using real-data in a scenario where the number of subjects on each site is held constant, while the number of sites in the network increases. This figure further illustrates that as the global number of subjects increases, the estimated djICA components converge towards the components computed in the pooled case. Indeed, 1024 global subjects was sufficient for good performance across the smaller network.

*How does spreading the data sets across more sites affect performance?*

In 2.3c, we evaluate the ISI index for djICA using real data in a scenario where the global number of subjects across the entire network is held constant, while the number of sites in the network increases. This figure illustrates that it is the global number of subjects included in the analysis, rather than the number of subjects per site, that mostly affects the performance of djICA. The concentration of subjects per site only begins to affect the performance of djICA when it is very low. At four subjects per site (256 sites in panel 2.3c), the performance is slightly worse than in previous runs. Thus, in 2.4, we provide more detailed results for the particular scenario using 2000 subjects in order to illustrate the effects of low number of subjects per site.

The three cases A, B, and C in 2.4 illustrate the performance of the algorithm for the minimum, median and maximum inter-symbol interference (ISI) scenarios respectively on an increasing number of sites, from 10 repeated runs for each subject-site distribution. Each run randomly placed different subjects on different sites. For all three cases, the plot to the left illustrates the correlations of each pooled ICA component with their corresponding match in each decomposition. It is evident from these plots that the correlation values for the majority of the 50 components clustered tightly above the mean correlation

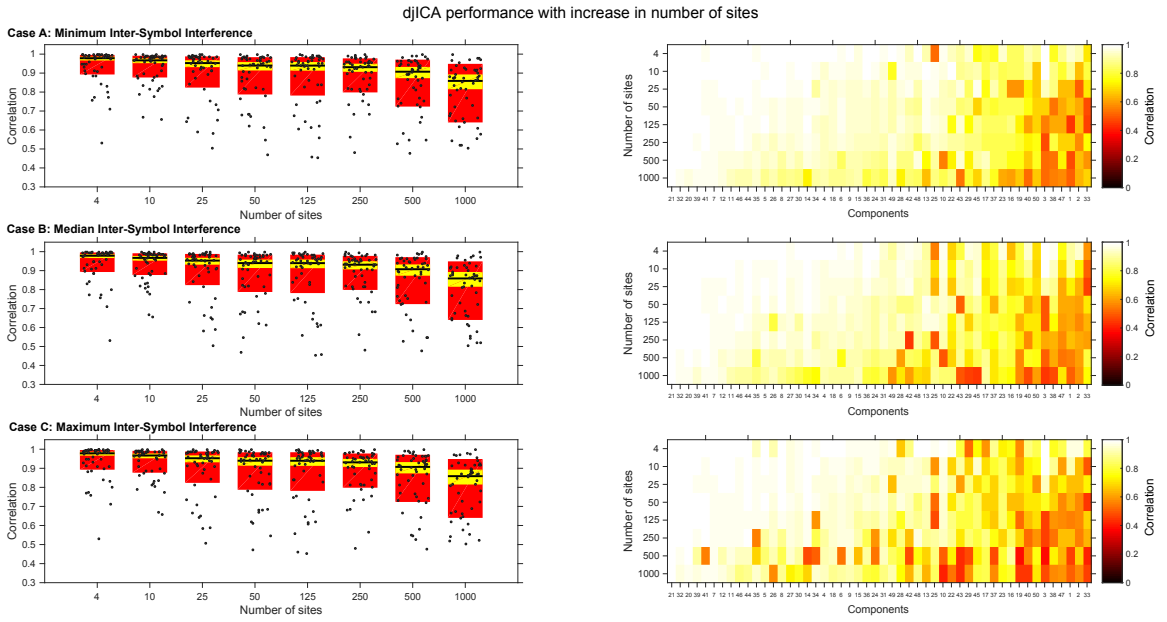



Figure 2.4: Figure reprinted with permission. Keeping the number of subjects fixed at 2000 and increasing the number of sites, we examine the correlations of the estimated components from djICA with the corresponding best match component from the pooled ICA case. The plots to the left illustrate the correlations between pooled ICA components and their best matched djICA components, from runs with the minimum (Case A, best), median (Case B), and maximum (Case C, worst) ISI selected out of 10 total runs. On the box-plots, the black horizontal bar represents the mean value of the Fisher-transformed correlations (z-space) for a specific decomposition transformed back to correlation space (r-space), the yellow shaded areas give the 95% confidence intervals of the Fisher-transformed correlations (z-space) transformed back to r-space, and the red box boundaries show the sample standard deviation over the Fisher-transformed correlations (z-space) transformed back to r-space. The panels to the right are a component-specific depiction of the similarity between the estimated djICA components and their corresponding pooled ICA component. Lighter colors indicate that the estimated component highly resembled the pooled ICA component estimated from 2038 subjects. The components (columns) are arranged in descending order of correlations for the minimum ISI case, and this sorting order was retained for the median and maximum ISI cases.

values (black horizontal bar) for the entire range of number of sites, thus suggesting that the mean correlation values for all cases were driven to a lower value by a few outliers (poorly replicated components). In fact, the lowest mean for the worst ISI case was 0.83 ($s$ = 1000). However, in general, the mean value of the component correlations for any given case decreased with an increase in number of sites.

A one-way analysis of variance (ANOVA) and multiple comparison of means tests

were used to determine the specific cases in which the mean Fisher-transformed correlation estimates (z-space) were significantly different (at a corrected significance level of 5% using Tukey's honest significant difference (HSD) method). For the minimum ISI case, significant differences were observed in the first five decompositions ($s$ = 4, 10, 25, 50, and 125) as compared to the last decomposition ($s$ = 1000), the first two decompositions ($s$ = 4 and 10) as compared to the second-to-last ($s$ = 500), and the first decomposition ($s$ = 4) as compared to the fourth, fifth and sixth ($s$ = 50, 125 and 250). The median ISI case showed an almost identical set of differences, except for one less significant difference between the first *four* decompositions ($s$ = 4, 10, 25 and 50) as compared to the last decomposition ($s$ = 1000). Finally, for the maximum ISI case, the first four decompositions ($s$ = 4, 10, 25 and 50) had mean correlation estimates significantly different than the last two ($s$ = 500 and 1000), and the first decomposition ($s$ = 4) showed additional significant differences as compared to the fifth and sixth ($s$ = 125 and 250). This overall pattern clearly indicates: (1) deterioration of performance with lower number of subjects per site; (2) significantly lower mean correlations for very low number of subjects per site (8 at $s$ = 250, 4 at s = 500, and 2 at $s$ = 1000).

For all cases A, B, and C, the component-specific performance of the algorithm can be traced in the correlation intensity images to the right. It is evident that all three cases feature high correlations for most of the components, especially in decompositions with lower number of sites. However, there are a few components that exhibit poor (or outlying) performance. For example, correlations for the last few (rightmost) components degrade significantly in decompositions with higher number of sites, while in the minimum ISI case the correlation for component 25 is unexpectedly low for the first decomposition ($s$ = 4) and higher for the remaining decompositions. Finally, it can be concluded from these correlation images that although the algorithm performance degrades with increasing number sites, the same set of components tends to be consistently well replicated.

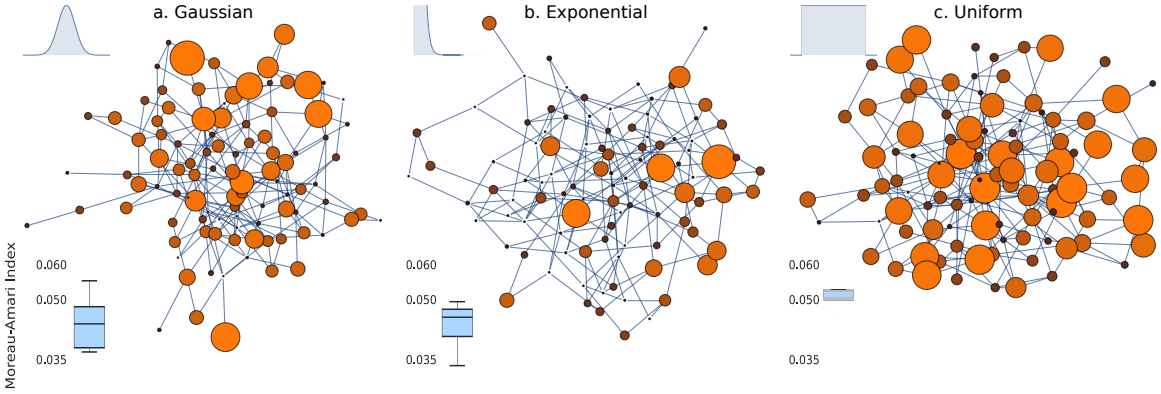



Figure 2.5: Figure reprinted with permission. An illustration of the effect of randomly distributing subjects across a decentralized data network. Each panel contains an example graph of connected nodes in the network, where each node represents a site in the network. The size of each node in the network corresponds to the number of subjects located on that site. The estimated ISI is computed after running djICA over 5 repeated runs, where each distinct run utilized a different network sampled from the same distribution. Panel (a) illustrates a network where the number of subjects on each site was sampled from a gaussian distribution, panel (b) illustrates a network of subjects where the number of subjects on each site was sampled from an exponential distribution, and panel (c) illustrates a network where the number of subjects on each site was sampled from a uniform distribution. In the bottom-left the corner of each panel, we plot the ISI after performing djICA for 5 different runs, where each run resampled the number of subjects on each site from the given distribution.

*How does randomly splitting the data sets across more sites affect performance?*

In real-world scenarios, fMRI data is not evenly distributed across research sites, thus motivating an investigation into the effect of randomly distributing the number of subjects per site on the effectiveness of djICA . In 2.5, we compare the estimated ISI of djICA using real-data in a scenario where 2000 subjects are randomly assigned to sites by sampling the number of subjects on each site from a given distribution $P(\Theta)$. We tested three different distributions for site assignment: normal, exponential, and uniform, setting the mean and standard deviation both at 128. The sampling process generates nodes (research sites) with different dataset sizes, and we then run djICA and compute the ISI as given above. We ran djICA five times for each distribution, resampling the number of subjects per site each time, and then plotted the ISI for each run. Each panel in 2.5 also illustrates a graph of a

network where each site is represented as a node. The size of each node in the network corresponds to the number of subjects on a given site, which is sampled from the given distribution.

As the figure shows, uniformly distributing subjects across the network reduces variance in computations when compared to normally distributed subjects; however, all of the given runs do not vary more than 0.02 with respect to the ISI, and all fall below 0.1 ISI, indicating favorable performance.

*How do the estimated maps compare with previous results?*

In 2.3d we provide the spatial maps from three of the highest-correlated components estimated using djICA and pooled ICA for comparison to their corresponding temporal fluctuation modes (TFM) from Smith et al. [111], which also investigated temporal ICA of fMRI. We discuss this comparison in the following section.

## 2.5 Discussion

In contrast to systems optimized for processing large amounts of data by making computation more efficient (Apache Spark, H2O and others), we focus on a different setting common in research collaborations: data are expensive to collect, are spread across multiple sites, and possibly not shareable directly. To that end, we proposed a *distributed data* joint ICA algorithm that, in synthetic experiments, finds underlying sources in decentralized data nearly as accurately as its centralized counterpart. This shows that algorithms like djICA may enable collaborative processing of decentralized data by combining local computation and communication of local summaries. djICA represents an important iteration towards toolboxes for computing on data distributed across private sites with an emphasis on collaboration. While other distributed methods for decentralized fMRI analysis have been recently proposed [50, 51], djICA in particular is able to benefit from the unique opportunity of globally accumulated multi-subject data.

To further validate our method we have evaluated it in experiments on real fMRI data. Our use of djICA to perform temporal ICA of fMRI produces results which compare well to the pooled version of the algorithm and produces estimated components which compare well with other work on temporal fMRI analysis [111] that uses much more elaborate multi-step analyses techniques. Additionally, djICA is robust to random allocation of subjects to sites, generally performing well with a high number of globally accumulated subjects, and insensitive to how these subjects are distributed across the sites. We have discovered one edge-case for real-data djICA in which having less than four subjects per site across all sites in the network leads to a slight decrease in global performance. While further investigation using a robust hyper-parameter search (which we did not pursue in this paper) may mitigate this performance reduction, the scenario where all or many sites in a collaborative analysis would each have fewer than four subjects is highly unlikely. Other decentralized approaches to fMRI analysis, such as approaches which use the ENIGMA consortium [97], do not explore this edge-case. Indeed, the lowest number of subjects on a site within the ENIGMA consortium was 36, with the majority of other sites in the consortium possessing over 200 subjects [100].

Our decentralized djICA algorithm is a good fit for decentralized collaborative frameworks, such as the COINSTAC collaboration platform, and is amenable to the privacy guarantees including in those platforms. The inclusion of djICA in a system like COINSTAC would allow for shared analysis between members of pre-arranged consortia without the exchange of raw data. This alone provides a level of plausible privacy to djICA which is not available to centralized ICA approaches. As we have explored elsewhere, djICA can be easily extended to include quantifiable notions of privacy, such as differential privacy [106]. Further investigation is required, however, to investigate the robustness of both plausible and differential privacy to scenarios involving malicious participants in the consortium. For example, it has been shown that malicious participants in a collaborative classification task using a decentralized Deep Neural Network can reconstruct data samples by utilizing Gen-

erative Adversarial Networks to leverage shared gradient information [142]. It is currently unclear whether or not methods such as djICA suffer from this information leakage issue; however, the issue demands future attention.

Privacy aside, real-world networks can suffer from a number of additional implementation issues: individual sites may have different computing hardware and messages may be dropped due to network latency or slow processing. While it is likely that issues such as hardware variance will not significantly influence the analysis in the decentralized case, other practical considerations should be handled by the overall software framework in which djICA would be included. The djICA algorithm can easily be made more robust to by including features such as timeouts, automated resets in response to errors or dropout, thresholds for minimum sufficient participation from each site, and so on.

Additionally, a number of decentralization-friendly heuristic choices can be made to improve runtime or performance beyond that of the default settings in djICA. For example, a stochastic gradient for weight updates can be computed over blocks (or mini-batches) of data in order to improve runtime. Thus, the block size $b$ can be chosen as a heuristic or evaluated as a hyper-parameter in order to examine the tradeoff between algorithm runtime and performance. Other hyper-parameters worth investigating are the tolerance level $t$, initial learning rate $\rho$, maximum iterations $J$, and the number of components chosen for local PCA.

In a pooled environment with a known ground-truth, it makes sense to find optimal values for these hyper-parameters using a grid search, or other hyper parameter selection method. In many real-life collaborative environments, however, a thorough hyper parameter search across sites may be impractical, and as far as we have found, no established method exists for hyper parameter optimization across decentralized sites. Finally, real-data problems often lack a reliable ground-truth, which makes it even more difficult and time-consuming to verify the effectiveness of multiple hyper-parameters. Nonetheless, in situations where a reliable ground-truth *is* available, such as in realistic simulations, one

simple solution would be to aggregate locally searched hyper-parameters; however, this method is unlikely to yield good performance if the number of subjects varies widely between sites, or if many of the sites contain only a small amount of data. Another potential solution would be to have each site participate in a global search using a randomly sampled subset of the local data. This may prove effective provided that enough data can be made available from each site, but would come at the expense of additional computation, and additional release of information from each site. In ICA for fMRI, certain auxiliary measures, such as the cross-correlation between components or the kurtosis of estimated independent components, could be used to assess performance empirically, starting with an initial heuristic choice of parameters and making adjustments if the auxiliary measures (or other indirect validation surrogates) indicate it would be helpful to do so.

Due to the lack of sufficient data problem that our method solves, temporal ICA networks from resting state neuroimaging data are rarely reported in the literature. A straightforward comparison of our observed networks with typical ones is not possible. However, our maps should be comparable, to some extent, to temporal fluctuation modes (TFMs) reported in Smith et al. [111], which performed temporal ICA on denoised spatial ICA component time courses. A qualitative comparison of the observed ground-truth maps in our work to the TFM maps reported in their work suggests similarities in certain spatial map activation patterns between the two. Component 15 resembles TFM 8 with task positive regions (dorsal visual regions and frontal eye fields) anti-correlated to the default mode (posterior cingulate, angular gyri, and medial prefrontal cortex). Component 8, with anti-correlated foveal and high-eccentricity visual areas corresponding to surround suppression observed in task studies, shows a good resemblance to TFM 4 in that work. As observed in TFM 2, component 6 shows coactivation patterns of lateral visual areas and parts of thalamus. Component 17 from our work, shows a good correspondence to TFM 13 in that work, with DMN regions anti-correlated with bilateral supramarginal gyri and language regions, albeit without strong lateralization reported in that work. TFM 1 and component 14 in this

work, demonstrate anti-correlated somatosensory regions to DMN regions of the brain. A couple other TFMs, 12 and 15, show moderate correspondence to components 11 and 9, respectively.

The differences between the networks we observed and the TFMs reported in Smith et al. [111] may stem from methodological differences and choice of number of independent components. In that work, instead of performing direct temporal ICA on preprocessed data to identify fluctuation modes, Smith et al. [111] use a two-step approach: firstly, performing a high model order spatial ICA, identifying artifactual components, and regressing out their variance from the time courses of seemingly non-artifactual components, and secondly, performing a temporal ICA on these denoised time courses. In contrast, we perform direct temporal ICA, leveraging the large number of samples available in large collaborative studies and directly getting to dynamics of fMRI. Therefore, the amount of variance captured during the PCA step in both methods differs. We identify 19 non-artifactual spatial modes, out of our 50 estimated components; all with spatial map activation patterns localized to gray matter regions and corresponding power spectra of independent time courses showing higher low frequency amplitude, as observed for intrinsic connectivity networks from spatial ICA analyses. These maps are included in 2.6. Finally, the data utilized in that work was from 36 ten minute-runs from 5 subjects, roughly sampled at TR=0.8s, which yielded 24000 concatenated timepoints, in contrast to roughly 300000 concatenated timepoints from 2000 subjects in this study, which is arguably a more general result.

## 2.6 Conclusions & Future Work

We have presented djICA, a novel method for decentralized temporal Independent Component Analysis, which represents a step toward facilitating large, collaborative analyses of data in a decentralized fashion. We evaluated djICA on simulated and real fMRI data, with both experiments illustrating the benefits of djICA, namely the increased availability of a larger, otherwise inaccessible, subject pool shared across multiple sites. Additionally, since

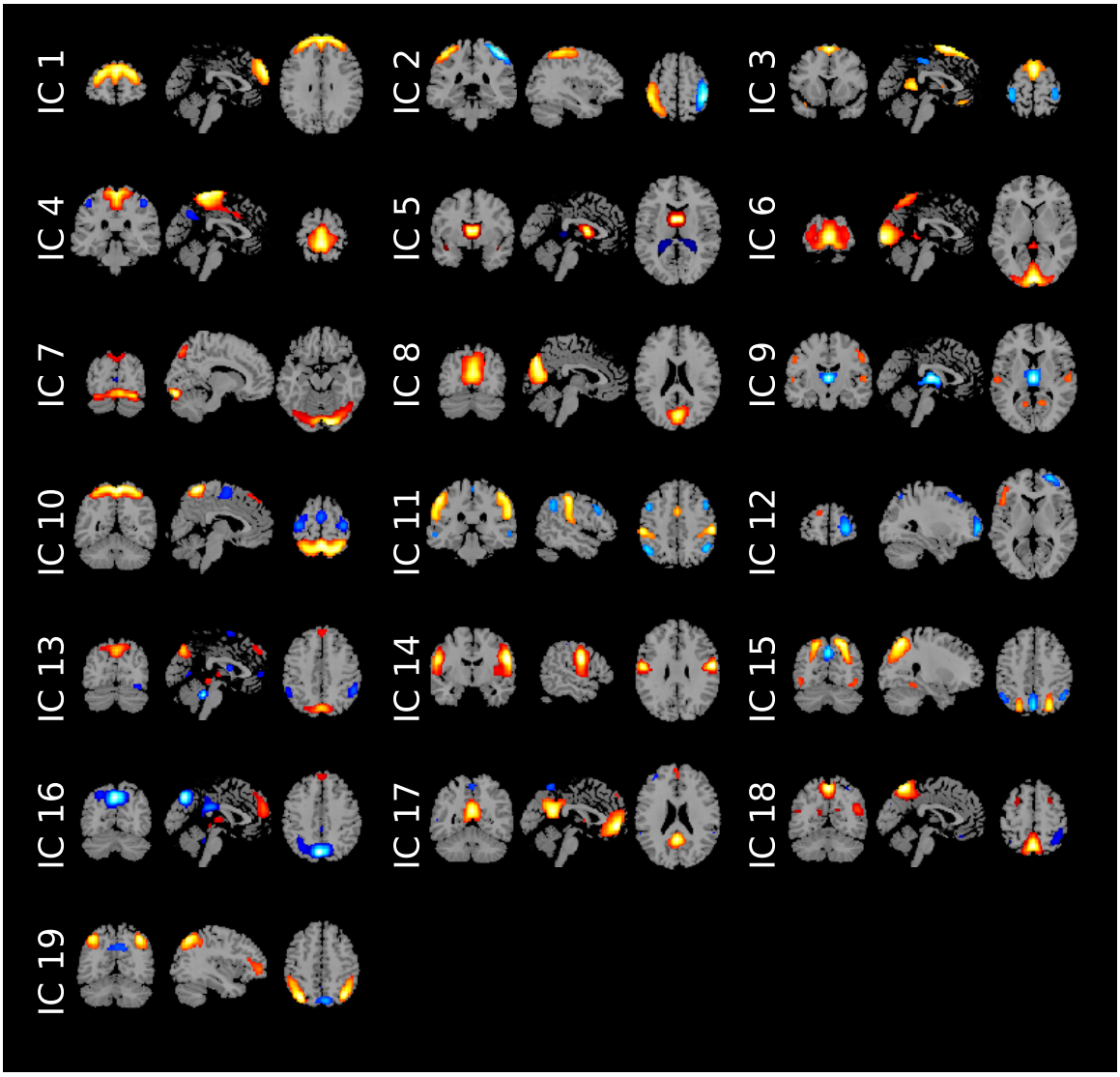



Figure 2.6: Figure reprinted with permission. The 19 identified non-artifactual spatial modes, with spatial map activation patterns localized to gray matter regions. Component 15 resembles TFM 8 from Smith et al.. [111], with task positive regions (dorsal visual regions and frontal eye fields) anti-correlated to the default mode (posterior cingulate, angular gyri, and medial prefrontal cortex). Component 8 shows anti-correlated foveal and high-eccentricity visual areas corresponding to surround suppression observed in task studies, and resembles to TFM 4 in Smith et al.. Component 6 shows coactivation patterns of lateral visual areas and parts of thalamus. Component 17 shows a good correspondence to TFM 13 in Smith et al.. with DMN regions anti-correlated with bilateral supramarginal gyri and language regions, but without strong lateralization reported in that Smith et al.. Component 14 demonstrates anti-correlated somatosensory regions to DMN regions of the brain. A couple other TFMs from Smith. et al.. 12 and 15, show moderate correspondence to components 11 and 9 from our estimation.

djICA does not communicate subject data across sites but only gradients, it is amenable to privatization via approaches like differential privacy [46], thus further opening the potential for collaboration between sites where direct sharing of data is not possible. Indeed, the increased availability of data provided by decentralized methods like djICA enables data-intensive, and thus underutilized, analyses like temporal Independent Component Analysis. Our comparison to the results from Smith et al. [111] confirms that djICA produces comparable temporal components. Finally, djICA and other methods like it foster further research on previously unexplored temporal dynamics in fMRI, such as the effects on temporal ICA of common confounds often found in datasets consisting of multi-site data.

Additional extensions to the methods provided here include reducing the bandwidth of the method and designing privacy-preserving variants, possibly, with differential privacy guarantees, which we have previously investigated for simulated cases [106]. In such cases, reducing the iteration complexity will help guarantee more privacy and hence incentivize larger research collaborations. If we were to return to the simulated data case, additional explorations into robust hyper-parameter searches and the deliberate corruption by noise may prove interesting for discovering and ameliorating further edge-cases for djICA. Beyond temporal ICA, decentralized spatial ICA is also worth investigation, and could be paired with decentralized clustering to evaluate decentralized dynamic functional network connectivity. Finally, Infomax ICA represents only one optimization approach to perform ICA, and while it is amenable to decentralization, other algorithms for ICA, such as fastICA [23] or the flexible entropy bound minimization (EBM) [143] approach, may provide other benefits beyond ease of decentralization.

# CHAPTER 3
# DECENTRALIZED DYNAMIC FUNCTIONAL NETWORK CONNECTIVITY: STATE ANALYSIS IN COLLABORATIVE SETTINGS

## 3.1 Introduction

The prospects of sharing data across studies provide researchers with clear and exciting prospects for collaborative analysis. Although the possible advantages to data-sharing are clear - increasing sample size and diversity, for example - directly transferring samples between sites is not always feasible, or desirable. Lack of post hoc sharing provisions, tedious negotiations of data usage agreements (DUAs), and limitations on local storage and bandwidth may all impede efforts for direct sharing. Additionally, in privacy-sensitive settings, direct sharing of data comes at the risk of re-identification, which becomes especially important in cases where samples belong to particularly rare groups, such as rare patient populations. Although steps toward anonymization in direct sharing scenarios can be taken, this anonymity often comes at the expense of data richness, or in the best cases, at the expense of significant effort by the collaborators involved.

Direct sharing of data is most often favored by centralized analysis frameworks, which pool data in one location. Though centralized sharing efforts can be powerful, to overcome the limitations outlined above, the research community requires a new family of decentralized approaches, where the analysis is performed without any direct data transfer, and data remains stored on disparate sites. One such decentralized alternative utilized by the ENIGMA framework performs meta-analyses utilizing summary statistics and references to existing literature to perform analysis [97, 99]. Though the approach ingeniously skirts issues endemic to centralized approaches, heterogeneity among studies and reliance on summary statistics tend to negatively impact the effectiveness of meta-analysis approaches.

The answer to the shortcomings of meta-analysis frameworks are iterative decentralized methods, where numerical optimization methods and other analysis techniques are split across multiple sites. Aggregation of shared iterates between sites allow these decentralized analysis frameworks to converge to solutions which are equivalent to the pooled case. The developers of the COINSTAC decentralized analysis framework [44] have successfully amassed a number of decentralized algorithms vital to neuroimaging analysis, including but not limited to Independent Vector Analysis [50], Deep Neural Networks [51], and Voxel-Based morphometry [144]. In this work, we further one particular iterative pipeline, decentralized dynamic functional network connectivity (ddFNC), which combines a number of distinct and useful algorithms used primarily in neuroimaging analysis. We build on preliminary work introduced elsewhere [144], extending the presentation of ddFNC to include more thorough analysis of the individual algorithms contained within it.

### 3.1.1 Dynamic Functional Network Connectivity

Functional connectivity (FC) [145] is one popular method for neuroimaging analysis which evaluates the connectivity between functional networks extracted from functional magnetic resonance images (fMRI). In particular, the assessment of functional connectivity from resting-state data has revealed new findings surrounding the high-level spatio-temporal organization of the brain. In this section, we present a framework for performing decentralized dynamic functional network connectivity (ddFNC) analysis (where FNC refers to the evaluation of FC between brain networks or components rather than isolated seeds). The resulting multi-step framework includes decentralized versions for each step of the standard dynamic functional network connectivity (dFNC) pipeline, including novel algorithms for decentralized principal component analysis (GlobalPCA ) and decentralized group independent component analysis (dgICA), as well as an application of decentralized K-Means clustering to completely reproduce the full dFNC pipeline.

The standard, data-driven approach to assess FNC dynamics, utilizes 1) spatial indepen-

dent component analysis (ICA), 2) sliding window temporal correlation, and 3) k-means clustering of windowed correlation matrices in order to evaluate connectivity between distinct functional networks. The approach, described by Allen et al. [146] utilizes group ICA (GICA) [119] to decompose resting-state data from multiple subjects into statistically independent functional regions. To evaluate temporal dynamics in FNC, the correlation between component time courses are then computed using a series of sliding windows [147]. Finally, k-means clustering is used to identify FNC patterns that reoccur in time and across subjects. These resulting clusters are called "FNC states", describing short periods during which FNC topography remains relatively stable in the functional domain. In particular, these states and their shift over time can be used to evaluate group differences between patients suffering from various kinds of mental illness and healthy controls [148, 149].

### 3.1.2 Federated Learning for Neuroimaging

Although no other methods for decentralized dFNC exist in the literature, a number of other approaches for federated learning on neuroimaging data exist in the literature. First, meta-analysis frameworks such as ENIGMA [97, 99], perform analysis on local data, where meta-statistics of the analyses are then aggregated in a decentralized fashion to produce global results. For example, Silva et al. implement the ENIGMA framework to provide structural analysis of subcortical brain-data between multi-site neuroimaging studies [150].

As mentioned above, meta-analyses can introduce artifacts to standard machine-learning algorithms due to heterogeneity between studies. As such a number of approaches for iterative federated training of machine learning algorithms have been proposed in the literature. In general machine-learning applications much focus has been given to federated deep learning [151, 40, 41, 152, 153, 154], since training of deep learning models requires large amounts of data which may be decentralized across a data network.

In neuroimaging applications, a more diverse array of algorithms have recently appeared for federated learning. On the deep learning side, Lewis et al. propose apply a

decentralized approach for deep learning to aid in the classification of neuroimaging addiction data [51]. Similarly, Remedios et al. provide a decentralized application of deep learning for neuroimage segmentation [155]. Decentralized Joint Indepdenent Component Analysis [156], Independent Vector Analysis [50], decentralized Stochastic Neighbor Embeddings [53], and Voxel-Based Morphometry [144] have also been applied to the analysis of decentralized neuroimaging data. In general, many of these frameworks proceed by iteratively computing the statistics used for optimization of a particular algorithm in a decentralized way. Though the statistics used for optimization are different and present novel challenges, our algorithm for ddFNCwill proceed much in the same way.

## 3.2 Materials and Methods

In this section, we present the data and experimental methodology utilized to evaluate decentralized group ICA, along with decentralized PCA (parallel and otherwise), decentralized clustering as well as the complete decentralized dFNC pipeline. First, section 3.2.1 presents our novel method for performing group ICA in a decentralized setting. Second, section 3.2.1 presents a novel method for performing decentralized PCA in parallel, improving the runtime of our previous decentralized PCA method.

Section 3.2.4 describes the functional MRI data used for evaluation of all novel methods. Then, section 3.2.5 provides outlines of all the experiments performed for each method.

### 3.2.1 Decentralized Group ICA

The first step in the dFNC pipeline for fMRI is group independent component analysis (gICA) [119]. Suppose that sites collect data $\mathbf{X} \in \mathbb{R}^{d \times N}$, where $d$ is the size of the voxel dimension, and $N$ is the total number of time-points across all subjects on all sites. In linear spatial ICA, we model each individual subject as a mixture of $r$ statistically independent spatial components, $\mathbf{A} \in \mathbb{R}^{d \times r}$, and their time-courses, $\mathbf{S}_i \in \mathbb{R}^{r \times N_i}$, where $N_i$ is the

length of the time-course belonging to site $i$. Although there are multiple approaches to aggregating subjects for the group analysis [157], we can model the global (i.e., cross-site) data set $\mathbf{X}$ as the column-wise concatenation of $s$ sites in the temporal dimension:

$$\mathbf{X} = [\mathbf{AS}_1 \cdots \mathbf{AS}_i \ \cdots \ \mathbf{AS}_s] \in \mathbb{R}^{d \times N},$$

where $[\cdots]$ represents column-wise concatenation, $s$ is the total number of sites in the consortium, and each site is modeled as a set of subjects concatenated in the temporal dimension as $\mathbf{AS}_i = \mathbf{X}_i = [\mathbf{X}_{i1} \cdots \mathbf{X}_{im} \cdots \mathbf{X}_{iM}]$, i.e., the collection of all $M$ subjects in site $i$. The advantage of the temporal concatenation approach is that it only requires the computation of one ICA, yielding unique time courses for each subject while assuming common group spatial maps. Thereafter, subject-specific maps can be easily estimated via local back-reconstruction. Spatial concatenation for group analysis is also possible, allowing for direct estimation of unique spatial maps while assuming common time courses instead. Although the two approaches to concatenation amount to different ways of organizing the data, temporal concatenation appears to perform better for fMRI data [158].

In this work, the goal is to learn a cross-site global unmixing matrix, $\mathbf{B} \in \mathbb{R}^{N \times r}$, such that $\hat{\mathbf{A}} = \mathbf{XB} \approx \mathbf{A}$, where $\hat{\mathbf{A}} \in \mathbb{R}^{d \times r}$ is the set of unmixed maximally spatially independent components. To this end, we perform a decentralized group independent component analysis (dgICA), and use least squares to estimate the $m$-th subject's temporal components in the $i$-th site by computing $\hat{\mathbf{S}}_{im} = \mathbf{A}^{-}\mathbf{X}_{im}$, where $\mathbf{A}^{-}$ is the pseudo-inverse of the estimated sources.

Prior to ICA, we perform principal component analysis (PCA), as is typically done to reduce computational complexity and/or memory usage. In order to prevent disparate sites from obtaining full data samples, we resort to decentralized PCA [156]. Firstly, however, a (local) subject-wise preprocessing step recommended prior to spatial GICA [157] is performed, thus constituting a minor variation of the two-stage decentralized PCA procedure

utilized in [156]. Effectively, all sites preprocess each subject by removing local means in the voxel dimension, followed by reducing and whitening their temporal dimension to a common (and large) $k_1$ components. Then, decentralized PCA of the preprocessed data takes place in the usual two stages. First, each site performs a LocalPCA dimension-reduction (without whitening) of all preprocessed concatenated local subject data to a common $k_2$ principal components in the temporal dimension. A decentralized second stage (GlobalPCA) then produces a global set of $r$ spatial eigenvectors, $\mathbf{U} \in \mathbb{R}^{d \times r}$. As outlined in [156], this second stage asks sites to pass locally-reduced eigenvectors to other sites in a round-robin scheme where, upon receiving a set of eigenvectors, a site then stacks them in the column dimension along with its local preprocessed (but not $k_2$ reduced) data, and performs a further reduction of the stacked matrix. The resulting (locally updated) set of $k_2$ eigenvalues is then passed to the next peer in the network. This process iterates once through each site until the global eigenvectors reach some aggregator, or otherwise terminal site in the network. The algorithms for the LocalPCA and GlobalPCA steps are given in Algorithms 6 and 4, respectively. Following the recommendation for choices of $k_1$ and $k_2$, we follow the recommendations in [157] and [159], choosing $k_1 = 120$ and $k_2 = 5 \cdot r$.

**Algorithm 4** Figure reprinted with permission. Global PCA algorithm (GlobalPCA)

**Require:** $s$ sites with *preprocessed* data $\{\mathbf{X}_i \in \mathbb{R}^{d \times k_1 : i=1,2,\ldots,s}\}$, intended final rank $r$, local rank $k \geq r$.

Choose a random order $\pi$ for the sites.
$\mathbf{P}(1) = \text{LocalPCA}(\mathbf{X}_{\pi(1)}, \min\{k, \text{rank}(\mathbf{X}_{\pi(1)})\})$ ▷ Assume
**for** $j = 2, 3, \ldots, s$ **do** ▷ Round-robin scheme
    $i = \pi(j)$ ▷ Set site index.
    Send $\mathbf{P}(j-1)$ from site $\pi(j-1)$ to site $\pi(j)$
    $k' = \min\{k, \text{rank}(\mathbf{X}_i)\}$
    $\mathbf{P}' = \text{LocalPCA}(\mathbf{X}_i, k')$
    $k' = \max\{k', \text{rank}(\mathbf{P}(j-1))\}$
    $\mathbf{P}(j) = \text{LocalPCA}([\mathbf{P}' \; \mathbf{P}(j-1)], k')$
**end for**
$r' = \min\{r, \text{rank}(\mathbf{P}(s))\}$
$\mathbf{U} = \text{NormalizeTopColumns}(\mathbf{P}(s), r')$ ▷ At last site

After performing decentralized PCA either via GlobalPCA or some other decentral-

**Algorithm 5** Figure reprinted with permission. NormalizeTopColumns

**Require:** data $\mathbf{P}$ and number of columns to reduce $r'$

1: $U = [P_1/||P_1||, P_2/||P_2||, \cdots, P_{r'}/||P_{r'}||]$ ▷ first $r'$ columns of $\mathbf{P}$

2: **return** U

**Algorithm 6** Figure reprinted with permission. Local PCA algorithm (LocalPCA)

**Require:** data $\mathbf{X} \in \mathbb{R}^{d \times N}$ and intended rank $k$

1: Compute the SVD $\mathbf{X} = \mathbf{U}\mathbf{\Sigma}\mathbf{V}$.

2: Let $\mathbf{\Sigma}^{(k)} \in \mathbb{R}^{k \times k}$ contain the largest $k$ singular values and $\mathbf{U}^{(k)} \in \mathbb{R}^{d \times k}$ the corresponding singular vectors.

3: Save $\mathbf{U}^{(k)}$ and $\mathbf{\Sigma}^{(k)}$ locally and return $\mathbf{P} = \mathbf{U}^{(k)}\mathbf{\Sigma}^{(k)}$.

ized algorithm, the aggregator site then performs whitening on these resulting global eigenvectors and runs a local ICA algorithm, such as infomax ICA [21], or fastICA [23] to produce the spatial unmixing matrix, $\mathbf{W} \in \mathbb{R}^{r \times r}$. The global eigenvectors, $\mathbf{U}$, are then unmixed to produce $\hat{\mathbf{A}}$ by computing $\hat{\mathbf{A}} = \mathbf{U}\mathbf{W}$, which is shared across the decentralized network (Algorithm 7). Each site $i$ then uses this unmixing matrix to produce individual time-courses for each $m$-th subject by computing $\hat{\mathbf{S}}_{im} = \mathbf{A}^{-}\mathbf{X}_{im}$. Each site can then perform back-reconstruction or spatio-temporal regression (STR) approaches locally [119, 159] to produce subject-specific spatial maps, such as $\hat{\mathbf{A}}_{im} = \mathbf{X}_{im}\mathbf{S}_{im}^{-}$ in GICA1 back-reconstruction, where $\mathbf{S}_{im}^{-}$ is the pseudo-inverse of $\hat{\mathbf{S}}_{im}$.

**Algorithm 7** Figure reprinted with permission. Decentralized group ICA algorithm (dgICA)

**Require:** $s$ sites with data $\{\mathbf{X}_i \in \mathbb{R}^{d \times N_i : i=1,2,\ldots,s}\}$, intended final rank $r$, local site rank $k_2 \geq 5 \cdot r$, local subject rank $k_1 \leq N_{im}$.

1: **for all** sites $i = 1, 2, \ldots, s$ **do**

2: **for all** subjects $m = 1, 2, \ldots, M$ **do**

3: $\mathbf{X}_{i,m}^{\text{pre}} = \mathbf{X}_{i,m} - \mu(\mathbf{X}_{i,m})$ ▷ Remove **column** means

4: $\mathbf{X}_{i,m}^{\text{pre}} = \text{NORMALIZETOPCOLUMNS}(\text{LocalPCA}(\mathbf{X}_{i,m}, k_1), k_1)$

5: **end for**

6: **end for**

7: $\mathbf{U} = \text{GlobalPCA}(\{\mathbf{X}_i^{\text{pre}} \in \mathbb{R}^{d \times k_1 : i=1,2,\ldots,s}\}, r, k_2)$

8: $\mathbf{W} = \text{ICA}(\mathbf{U})$ ▷ At aggregator site $i = \pi(s)$.

9: Send $\hat{\mathbf{A}} = \mathbf{U}\mathbf{W}$ to sites $\pi(1), \ldots, \pi(s-1)$.

**Algorithm 8** Figure reprinted with permission. Back-Reconstruction

**Require:** Global unmixing matrix $\hat{\mathbf{A}} = \mathbf{UW}$, local data $\{X_{i,1}, \ldots, X_{i,M}\}$ on site $i$

1: **for all** subjects $m = 1, 2, \ldots, M$ **do**
2: $\quad \hat{\mathbf{S}}_{i,m} = \hat{\mathbf{A}}^{-}\mathbf{X}_{i,m}$ ▷ Retrieve subject time-courses
3: $\quad \hat{\mathbf{A}}_{i,m} = \mathbf{X}_{i,m}\mathbf{S}^{-}_{i,m}$ ▷ Use Spatio-Temporal regression or other back-reconstruction [119, 159] to retrieve subject-specific spatial maps
4: **end for**

*Parallel Global PCA*

The global PCA algorithm given above in algorithm 4, taken from [156], can be extended from the serial version so that it runs in parallel, thus taking advantage of the decentralized nature of the computation to also increase computation speed. The parallel strategy involves breaking up the consortium into sub-clusters, where GlobalPCAis computed in parallel within the sub-clusters until the final eigenvectors $U$ arrive at the aggregator. A diagram of the process for a consortium of 8 sites is given in 3.1, and the general algorithm is given in Algorithm 9.

**Algorithm 9** Figure reprinted with permission. Parallel Global PCA algorithm (pGlobalPCA)

**Require:** $s$ sites with data $\{\mathbf{X}_i \in \mathbb{R}^{d \times N_i : i=1,2,\ldots,s}\}$, intended final rank $r$, local rank $k \geq r$, cluster size = $C$, base cluster size = $B$.

1: $K = \lfloor s/C \rfloor$ ▷ Number of Clusters
2: **if** $K > B$ **then** ▷ At an "aggregator" site
3: $\quad$ **for all** $c = 1, 2, \ldots, K$ **do**
4: $\quad\quad a = (c-1)C + 1$
5: $\quad\quad b = \min(c \cdot C, s)$
6: $\quad\quad \mathbf{U}_c = \text{pGLOBALPCA}(b - a, \{\mathbf{X}_a, \ldots, \mathbf{X}_b\}, k, r)$
7: $\quad$ **end for**
8: $\quad \mathbf{U} = \text{pGLOBALPCA}(K, \{\mathbf{P}_1, \ldots, \mathbf{P}_K\}, k, r)$
9: **else** ▷ at a non-"aggregator" site
10: $\quad \mathbf{U} = \text{GLOBALPCA}(s, \{X_1, \ldots, X_s\}, r, k)$
11: $\quad$ **if** At final aggregator **then**
12: $\quad\quad \mathbf{U} = \text{NORMALIZETOPCOLUMNS}(\mathbf{U})$
13: $\quad$ **end if**
14: **end if**
15: **return** U

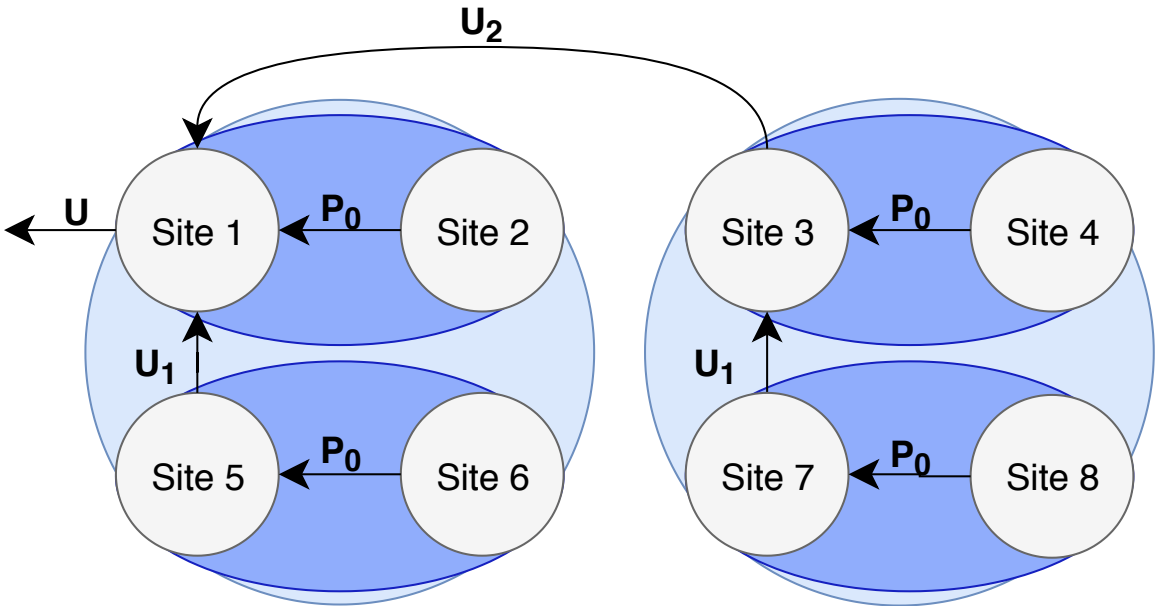



Figure 3.1: Figure reprinted with permission. Diagram of the pGlobalPCA algorithm for a consortium of $s = 8$ sites, with cluster size $C = 2$. First, the recursion of the algorithm breaks the full consortium into clusters of decreasing size until the number of sites in each cluster is equal to $C$. Then, each cluster performs the standard GlobalPCA. As the recursion steps back from this base-case, the result from GlobalPCA is passed between sub-clusters, and GlobalPCA performed again until the recursion ends.

### 3.2.2 Decentralized clustering

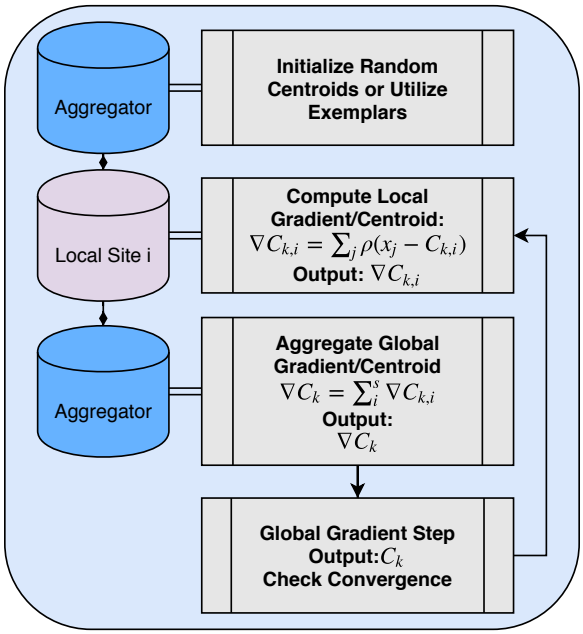
In order to perform dFNC in a decentralized setting, we first require a notion of decentralized clustering, used to cluster windowed patient timecourses into one of several connectivity states. Although other kinds of clustering are possible, previous work in dFNC has focused on the use of K-Means clustering, and thus, we focus first on decentralized K-Means clustering. A number of decentralized approaches to K-Means exist: first, Dhillon et al. [160] implement an exact version of lloyd's algorithm for K-Means over distributed memory multiprocessors, where each processor broadcasts an updated set of local centroids, according to locally stored data, and global centroids are aggregated by taking the average of these local centroid updates. Jagannathan et al. implement a similar version of this approach, but add additional privacy guarantees via encrypted message passing, and random sharing of centroids, rather than sharing at each iteration [161]. Jagannathan et al. also provide a general version of privacy-preserving clustering (including K-Means), where rather than sharing centroid locations each iteration, local clusters are computed to convergence, and then merged at some aggregator site [162]. Finally, a number of modern methods improve over standard methods with additional features often attractive in real-network scenarios. For example, Datta et al. provide an approximative, peer to peer methods for distributed K-Means [163, 164], and Di et al. provide a fault-tolerant version of dK-Means, well-suited to large, asynchronous networks [165].

Our aim in this paper is to provide a novel, end-to-end pipeline for decentralized dFNC, which includes clustering. Thus, which exact choice of algorithm is made for the decentralized K-Means step is an implementation choice, rather than an essential part of our pipeline. For our purposes, we test four different version of simple decentralized K-Means algorithms, focusing primarily on differences in centroid computation and updates, rather than details such as asynchronous updates, or peer to peer schema. First, we implement the algorithm from Dhillon et al. [160], and we also implement a version of the same iterative algorithm using a decentralized gradient update, rather than exact centroid computation.

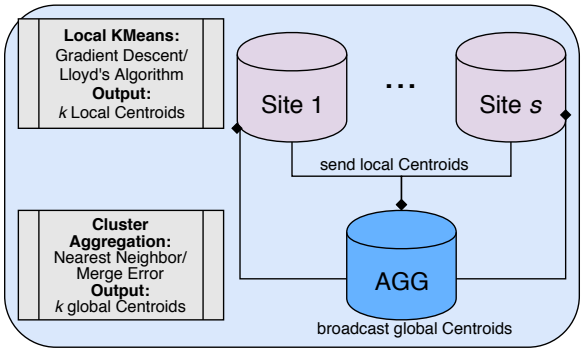
For this latter strategy, we implement the gradient descent algorithm described in [166], where at each iteration, locally computed gradients are averaged on the aggregator node in place of locally computed centroids. Finally, we implement version of these algorithms using the cluster aggregation strategy described in Jagannathan et al. [162]; however, we omit the additional privatization strategies for simplicity's sake. The two former strategies we call "multi-shot", because they involve decentralization at each iteration of the algorithm, and the two later strategies we call "single-shot" because they involve aggregation of the results of locally converged optimization strategies.

To perform clustering for distributed dFNC, we first have each site compute sliding-window time-course correlations for each subject, where the window length is fixed across the decentralized network. Additionally, initial clustering is performed on a subset of windows from each subject, corresponding to windows of maximal variability in correlation across component pairs. To obtain these exemplars, we follow the approach from [148], and have each site compute variance of dynamic connectivity across all pairs of components at each window. We then select windows corresponding to local maxima in this variance time course. This results in an average of 8 exemplar windows per subject. We then perform decentralized K-Means on the exemplars to obtain a set of centroids, which are shared across the decentralized network, which we feed into a second stage of K-Means clustering.

For the second stage of decentralized clustering, at each iteration, each site computes updated centroids according to [160], which corresponds to a local K-Means update. These local centroids are then sent to the aggregator node, which computes the weighted average of these updated centroids, and re-broadcasts the updated global centroids until convergence. A summary of the complete steps in the dFNC pipeline is given in 3.3.




(a) Figure reprinted with permission. Multi-Shot dK-Means with Gradient Descent and Lloyd's algorithm optimization




(b) Figure reprinted with permission. Single-Shot dK-Means algorithms

Figure 3.2: Figure reprinted with permission. Diagram of the multi-shot and single-shot dK-Means algorithms. Panel 3.2a outlines the multi-shot schema using gradient descent or lloyd's algorithm. First, randomized centroids are picked by the aggregator, and broadcast out to the sites. Each site then computes cluster membership, and perform their dK-Means updates, either by computing a gradient, or by updating the centroid according to lloyd's algorithm. These are then broadcast back to the aggregator, and aggregated into new centroids or gradients. New centroids are then rebroadcast, and the algorithm continues until convergence. In panel 3.2b, a diagram of the single-shot schema is given. In this approach, each site performs a separate, local K-Means optimization, and the final centroids are broadcast to the aggregator, which then merges clusters either by merging nearest centroids, or by querying sites to compute a merging error, as is done in [162].

**Algorithm 10** Figure reprinted with permission. Decentralized dFNC algorithm (ddFNC)

**Require:** $s$ sites with data $\{\mathbf{X}_i \in \mathbb{R}^{d \times N_i : i=1,2,\ldots,s}\}$, win-size $t$, number of clusters $k$.

dgICA $\rightarrow \mathbf{W}$, global unmixing matrix. Compute $\hat{\mathbf{A}} = \mathbf{UW}$, and broadcast to sites.

**for all** sites = $i = 1, 2, \ldots, s$ **do**

  $\hat{\mathbf{S}}_{i,m} = \hat{\mathbf{A}}^{-}\mathbf{X}_{i,m}$ ▷ Back-reconstruct subject TCs

  **for all** windows $w = 1, 2, \ldots, N_i - t$ **do**

    $\hat{\mathbf{S}}_{i,m,w} = [\hat{S}_{i,m,w} \ldots \hat{S}_{i,m,(w+t)}]$ ▷ Sliding window of size $t$ over time

    $\mathbf{V}_{i,m,w} = corr(\hat{\mathbf{S}}_{i,m,w})$

  **end for**

  Obtain local exemplar correlation matrices $\mathbf{V}_{i,ex}$ using the process from [148].

**end for**

Run DK-MEANS($\mathbf{V}_{ex}$) [160] to obtain $k$ initial centroids, $\mathbf{C}_0$.

Run DK-MEANS($\mathbf{V}$) [160] with initial clusters $\mathbf{C}_0$ to obtain $k$ centroids $\mathbf{C}$, and clustering assignment for each instance, $L$.

Return $\mathbf{C}, L$

### 3.2.3 Computational Complexity

Because ddFNC is a pipeline containing multiple distinct algorithmic components, the overall computational complexity of the pipeline will depend greatly on implementation details for each pipeline stage. The choice of ICA algorithm, or whether or not an iterative method is used to computed SVD for example, will greatly influence the actual complexity of the entire pipeline. That said, we provide an initial analysis of the GlobalPCA component of our pipeline as presented here, with the caveat that further changes can still be made within each of these depending on implementation preferences and availability of computational resources. We omit an analysis of complexity for Independent Component Analysis, since in principle any ICA algorithm could be used, and complexity varies with the choice of algorithm.

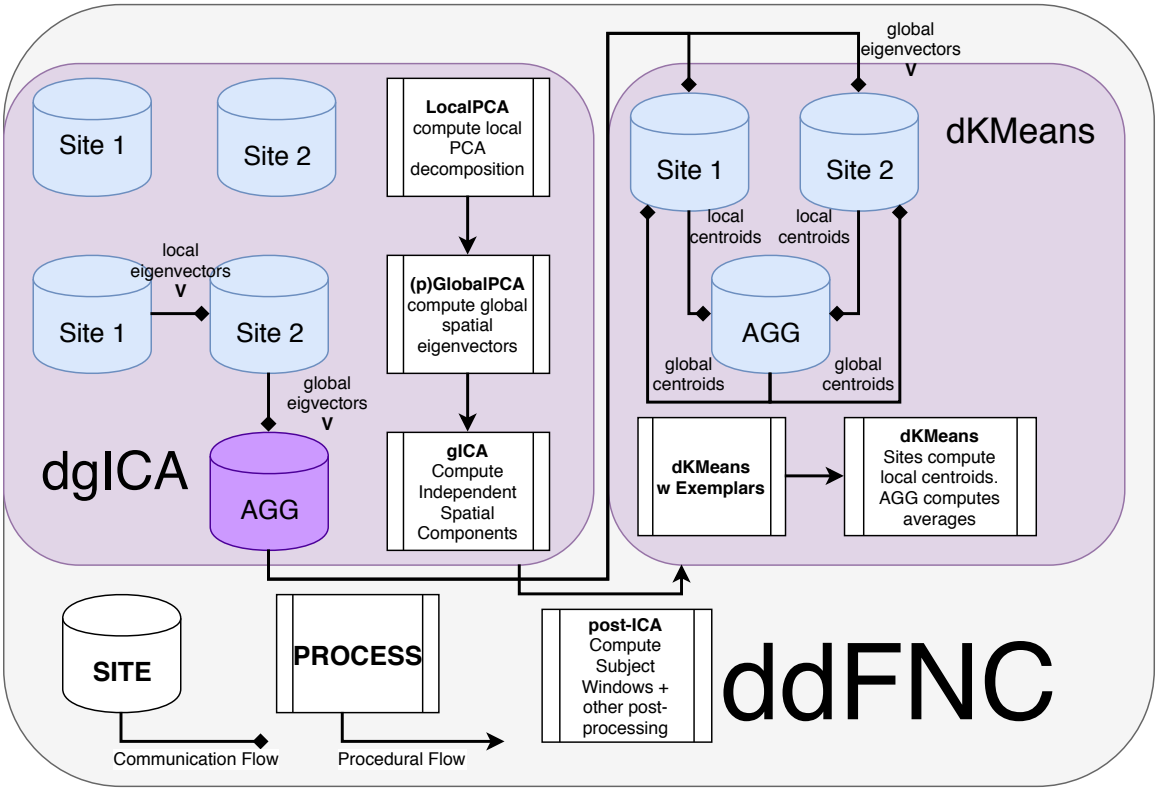



Figure 3.3: Figure reprinted with permission. Flowchart of the ddFNC procedure e.g. with 2 sites, using multi-shot lloyd's algorithm for K-Means clustering. To perform dgICA , sites first locally compute subject-specific LocalPCA to reduce the temporal dimension, and then use the GlobalPCA procedure from [156] to compute global spatial eigenvectors, which are then sent to the aggregator. The aggregator then performs ICA on the global spatial eigenvectors, using InfoMax ICA [21] for example, and passes the resulting spatial components back to local sites. The dK-Means procedure then iteratively computes global centroids using the procedure outlined in [160], first computing centroids from subject exemplar dFNC windows, and then using these centroids to initialize clustering over all subject windows.

The overall computational complexity of ddFNC is best analyzed in terms of the complexity on individual sites, since the decentralization of the algorithm reduces overall complexity into a sum of individual computational demands at each site. Suppose at an individual site, we begin with the matrix of temporally concatenated subjects $\mathbf{X}_i \in \mathbb{R}^{d \times N_i}$.

The complexity of Global PCA can be analyzed in terms of the complexity for the two Singular Value Decompositions performed first on the covariance matrix $X_i \in \mathbb{R}^{N_i \times N_i}$, and second on the $k_1 \times k_1$ matrix computed from stacking eigenvectors. Standard algorithms for computing SVD by Jacobi rotations have a complexity of $O(n^3)$ when computed on an $n \times n$ covariance matrix [167]. Thus, if GlobalPCA uses standard SVD, the overall complexity will be $O(N_i^3) + O(k_1^3)$, with complexity generally increasing as the number of subjects on local sites increases, or as the desired number of independent components increases.

Prior to decentralized K-Means, each site computes correlation matrices on the windowed time-courses oflength $N_{i,j} - w$ for each subject $j$. If $m_i$ subjects are located at a given site, then the local complexity for computing these matrices is $O(m_i(N_i - w)k_1^3)$, so again local computational cost increases with the number of subjects, the number of timepoints at each subject, and the desired number of independent components $k_1$.

For an analysis of decentralized K-Means, we refer the reader to the discussion in Dhillon et al. [160]. Let $\mathcal{J}$ is the number of K-Means iterations required for the centroid stability for K-Means with $C$ centroids, and let $S_i = m_i(N_i - w)$ be the number of correlation matrices computed at each site. The analysis provided in Dhillon et al. gives the site-wise computational complexity for dK-Means as $O((3Ck_1^2 + S_iC + S_ik_1^2 + Ck_1^2) \cdot \mathcal{J})$. For our pipeline, the choice of decentralized K-Means algorithm is modular, and local site complexity may be reduced in a number of different ways. For example, implementing a decentralized version of K-Means++ initialization [168, 169, 170] to may lower the number of iterations required for stability, thus reducing site-wise complexity as well as overall complexity. Because further analysis requires digging into the particulars of K-Means and

Table 3.1: Figure reprinted with permission. Distribution of subjects over original 7 sites.

| Site | HC | SZ | Total |
|---|---|---|---|
| 1 | 28 | 24 | 52 |
| 2 | 9 | 9 | 18 |
| 3 | 28 | 26 | 54 |
| 4 | 28 | 23 | 51 |
| 5 | 14 | 13 | 27 |
| 6 | 28 | 29 | 57 |
| 7 | 28 | 27 | 55 |

decentralized K-Means which is outside the scope of this paper, we leave further analysis of complexity for K-Means as future work.

### 3.2.4 Functional MRI data for dFNC

To evaluate ddFNC, we utilize imaging data from [148] collected from 163 healthy controls (117 males, 46 females; mean age 36.9) and 151 age - and gender matched patients with schizophrenia (114 males, 37 females; mean age 37.8), for a total of 314 subjects.

The scans were collected during an eyes closed resting fMRI protocol at 7 different sites across United States (see table 3.1) and pass data quality control. Informed and written consent was obtained from each participant prior to scanning in accordance with the Internal Review Boards of corresponding institutions [171]. A total of 162 brain-volumes of echo planar imaging BOLD fMRI data were collected with a temporal resolution of 2 seconds on 3-Tesla scanners.

Imaging data for six of the seven sites was collected on a 3T Siemens Tim Trio System and on a 3T General Electric Discovery MR750 scanner at one site. Resting state fMRI scans were acquired using a standard gradient-echo echo planar imaging paradigm: FOV of 220 $\times$ 220 mm (64 $\times$ 64 matrix), TR = 2 s, TE = 30 ms, FA = 770, 162 volumes, 32 sequential ascending axial slices of 4 mm thickness and 1 mm skip. Subjects had their eyes closed during the resting state scan. Data pre-processing for dgICA was performed according to the preprocessing steps in [148].

### 3.2.5 Experiments

In this section, we describe each of the experiments performed to step the various parts of our ddFNC pipeline. Since the ultimate goal is to provide ddFNC, we concentrate the bulk of our quality analysis on that final output; however, at each stage, we perform a number of small evaluations to make sure each piece works individually using either simulated data. We also measure the runtime of each stage separately, and compare runtimes and quality measures for different implementations of each algorithm.

*decentralized group ICA*

In this section, we present the experimental methodology used to evaluate decentralized group ICA, which includes decentralized PCA.

*Do pGlobalPCA and GlobalPCA produce equivalent components?*

Although it is clear mathematically that pGlobalPCA and GlobalPCA are equivalent, we perform a brief initial experiment to provide empirical evidence of the equivalence. First, to evaluate our novel method for parallel decentralized PCA, we generate a synthetic data set using the MATLAB randn function. We generate a single $100 \times 100$ data set, and use pooled PCA, GlobalPCA , and pGlobalPCA to reduce the column dimension to 10 principal components. For GlobalPCA and pGlobalPCA , we first split the data set onto 10 simulated "sites", where each site contains ten rows of the original matrix. If pGlobalPCA and PCA are functionally equivalent, we expect the correlation matrices to be nearly completely diagonal. We repeat this experiment 1000 times for each algorithm, and plot the results in 3.4.

After completing the synthetic experiments, we perform the same experiment using the real-data described above. We utilize the site-distribution used above, and again compute the correlation of the estimated PCs, and plot the results in 3.5. For real data, we repeat each experiment 100 times, with each repetition shuffling subjects between the sites.

*How does pGlobalPCAimprove runtime?*

The parallelization in pGlobalPCA (algorithm 9) should provide a performance improvement over vanilla GlobalPCA (algorithm 4) , especially in consortia with a large number of sites, allowing for many computations to be performed in parallel. In order to test this hypothesis, we perform two experiments designed to evaluate the runtime of GlobalPCA and pGlobalPCA , and how pGlobalPCA offers an improvement over GlobalPCA for certain distributions of subjects over the network.

First, we perform an experiment with synthetic data, using the same data-generation process as above. In order to evaluate how the runtime improvement for pGlobalPCA varies depending on the subject/site distribution, we vary both the size of the global data set and the number of sites in the consortium in order to evaluate how the distribution of data affects the runtime of both algorithms.

Again, we repeat a similar experiment utilizing the real data-set, evaluating how the distribution of subjects over the network affects the runtime of GlobalPCA and pGlobalPCA . We begin with 2 subjects, and increase by powers of 2 until we are dividing the 314 subjects over 64 sites.

*How does the choice of ICA method affect performance?*

ddFNC is a highly modular algorithm, thus allowing for the aggregator node in a given consortium to choose from any kind of Group ICA algorithm made available. Thus, we perform a brief analysis which compares multiple ICA algorithms in terms of component estimation quality and runtime. To measure the quality of components, we match the estimated components from the given ICA algorithm with the components estimated in [148], selecting the top components which best match with that ground-truth. Then, we compute the Moreau-Amari Inter-Symbol Interference index [126] between the estimated components and the components from [148], and plot the results for the given choice of algorithm. We note that in [148], the authors utilize infomax ICA, and so a decentralized infomax will have a comparative edge over other methods.

*decentralized clustering*

We perform dK-Means [160] on the computed correlation matrices from the sliding windows described above. We first cluster the "exemplar" temporal windows computed for each subject according to the strategy utilized in [148], and then utilize these centroids to cluster the entire set of computed windows. This provides a set of a k=5 resulting centroids as well as clustering assignments for each subject's window.

*decentralized dFNC*

In this section, we present the experimental methodology used to evaluate the final results of the decentralized dynamic dFNC pipeline.

We verify that ddFNC can generate sensible dFNC clusters by replicating the centroids produced in [148]. We closely follow the experimental procedure in [148], with some of the additional post-processing omitted for simplicity. To evaluate the success of our pipeline, we run a simple experiment where we implement the ddFNC pipeline end-to-end on the data, simulating 314 subjects being evenly shared over 2 decentralized sites.

We use a window-length of 22 time-points (44 s), for a total of 140 windows per subject. For dgICA, we first estimate 120 subject-specific principal components locally, and reduce each subject to 120 points in the temporal dimension. Subjects are then concatenated temporally on each site, and we use the parallel GlobalPCA algorithm to estimate 100 spatial components, and perform whitening. We then use local infomax ICA [21] on the aggregator to estimate the unmixing matrix $\mathbf{W}$, and estimate 100 spatially independent components, $\hat{\mathbf{A}}$. We then broadcast $\hat{\mathbf{A}}$ back to the local sites, and each site computes subject-specific time-courses.

After spatial ICA, we have each site perform a set of additional post-processing steps prior to decentralized dFNC. First, we select 47 components from the initial 100, by computing components which are most highly correlated with the components from [148]. We then have each site drop the first 2 points from each subject, regress subject head move-

ment parameters with 6 rigid body estimates, their derivatives and squares (total of 24 parameters). Additionally, any spikes identified are interpolated using 3rd order spline fits to good neighboring data, where spikes are defined as any points exceeding mean (FD) + 2.5 *std(FD) , where FD is framewise displacement (interpolating 0 to 9 points (mean,sd: 3, 1.76)).

For clustering in general, elbow-criterion estimation can be used to determine an optimal number of clusters. For comparison's sake, however, we use the optimal number of clusters from [148], setting $k$=5. For the exemplar stage of clustering, we evaluate 200 runs where we initialize centroids uniformly randomly from local data, and then run dK-Means using the cluster averaging strategy in [160]. For our distance measure, we use scikit-learn [172] to compute the correlation distance between covariance matrices following the methods in [148]. To keep our implementation simple, unlike [148], we do not utilize graphical LASSO to estimate the covariance matrix, and thus do not optimize for any regularization parameters. Additionally, we do not perform additional Fisher-Z transformations or perform additional regularization using a previously computed static dFNC result. Future implementations may also utilize a decentralized sFNC algorithm as preprocessing, as is done for the pooled case in [148]. Finally, for the second stage of dK-Means, we initialize using the centroids from the run with the highest silhouette score, computed using the scikit-learn python toolbox [172], again running dK-Means to convergence. After computing the centroids, we use the correlation distance and the Hungarian matching algorithm [173] to match both plotted spatial components from dgICA and the resulting centroids from dK-Means.

Finally, to make a more direct comparison between our analysis and the pooled case, we compare the resulting centroids with centroids estimated using pooled K-Means, measuring the correlation between the resulting centroids over multiple runs.

We also separate out the centroids for each group, and visualize them according to the procedures in [148]. Following the procedures in [148], we first calculated the element-

wise subject medians for each state according to the final clustering assignments from dK-Means. We then use the subject medians for each state and evaluated the differences between patient and healthy-control groups using a two sample t-test.

## 3.3 Results & Discussion

### 3.3.1 GlobalPCA vs pGlobalPCA

In 3.5 we plot the correlation of the components estimated from GlobalPCA and pGlobalPCA, averaged over 10 repeated runs, where each run created a new simulated matrix to be reduced. Clearly, the results indicate near-equivalence of the two algorithms, with minor differences likely due to noise from the serial GlobalPCAutilizing a different, random ordering of sites, or from the stochastic nature of infomax ICA.

In 3.4, we plot the average runtime for GlobalPCA and pGlobalPCA across three different scenarios of changing the subject and site distributions across a consortium. In panel a, we increase the number of subjects in a global consortium with 2 fixed sites. In panel b, we increase the number of sites in a global consortium, keeping the number of subjects fixed at 1024. In panel c, we increase the number of sites and subjects simultaneously.

The runtime comparison for the fixed number of sites in panel a illustrate the equivalent runtime for each algorithm in a scenario where the total number of sites is equal to the number of allowed cluster groups in pGlobalPCA. In such cases, where the parameter $b$ is set to equal the number of sites in the consortium, pGlobalPCA is equivalent to GlobalPCA, and the algorithms perform comparatively.

The runtime comparison in figure b, however, illustrates the benefits of parallel, decentralized PCA. In a highly distributed setting, where the number of sites is much larger than the parameter $b$, pGlobalPCA decreases runtime over standard GlobalPCA by executing certain steps in parallel, leveraging the decentralized design to improve runtime.

Panel C illustrates both a failure case for pGlobalPCA, where increased bandwidth between many small sites with small data invokes a small hit in runtime; however, it also

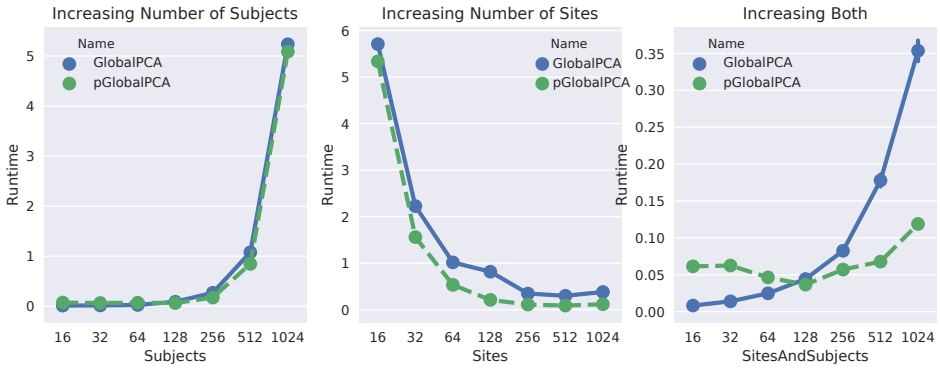



Figure 3.4: Figure reprinted with permission. Runtime comparison of GlobalPCA (algorithm 4) and Parallel Global PCA (pGlobalPCA, algorithm 9) for three different scenarios. In panel a, we increase the number of subjects in a global consortium with 2 fixed sites. In panel b, we increase the number of sites in a global consortium, keeping the number of subjects fixed at 1024. In panel c, we increase the number of sites and subjects simultaneously. The blue curve represents the mean runtime over 10 repeated runs for the GlobalPCA algorithm, and the green curve represents the mean runtime over 10 repeated runs for the pGlobalPCA algorithm.

illustrates that pGlobalPCA does not suffer as significant of a hit as more sites are added into the consortium, whereas the serial design of GlobalPCA suffers significantly.

### 3.3.2 dgICA results

3.6 plots the Moreau-Amari index for several different ICA algorithms performed at the aggregator node. In 3.9 we plot some of the estimated components from dgICA with infomax ICA, and compare with the matched components from pooled ICA. We also provide the correlation of estimated components in 3.9c and 3.9d. Indeed, dgICA with Infomax ICA provides components which are a good estimate of the pooled case, with the ISI between pooled and decentralized cases measured below 0.1, and the component-wise correlation of components providing a near exact estimate of the pooled components. This indicates that our decentralization strategy works, and does not incur a significant penalty to quality of the estimations via decentralization alone. This assurance of quality in decentralization may change when privacy measures, such as differential privacy, are taken; however, our analyses here is sufficient to show that decentralization alone does not significantly affect

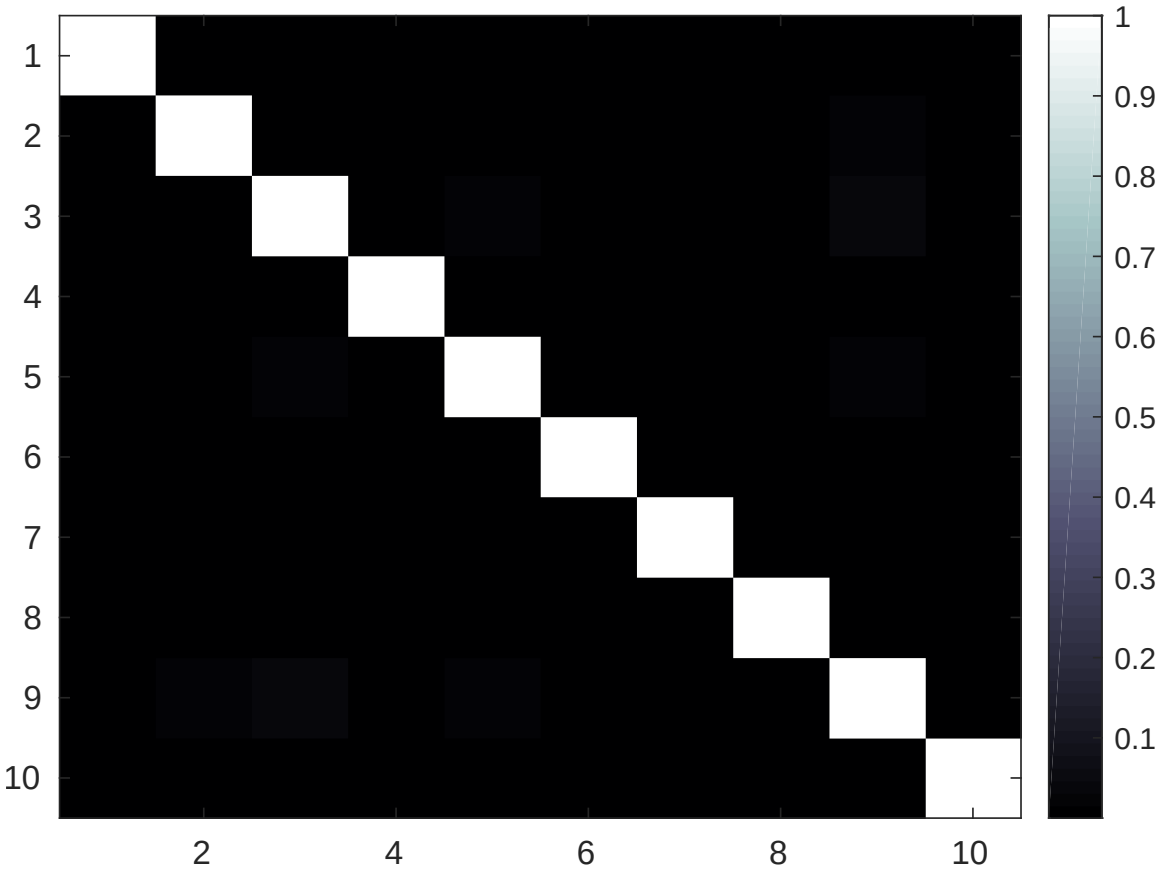



Figure 3.5: Figure reprinted with permission. Correlation of components estimated from GlobalPCA and Parallel GlobalPCA , averaged over 10 separate runs.

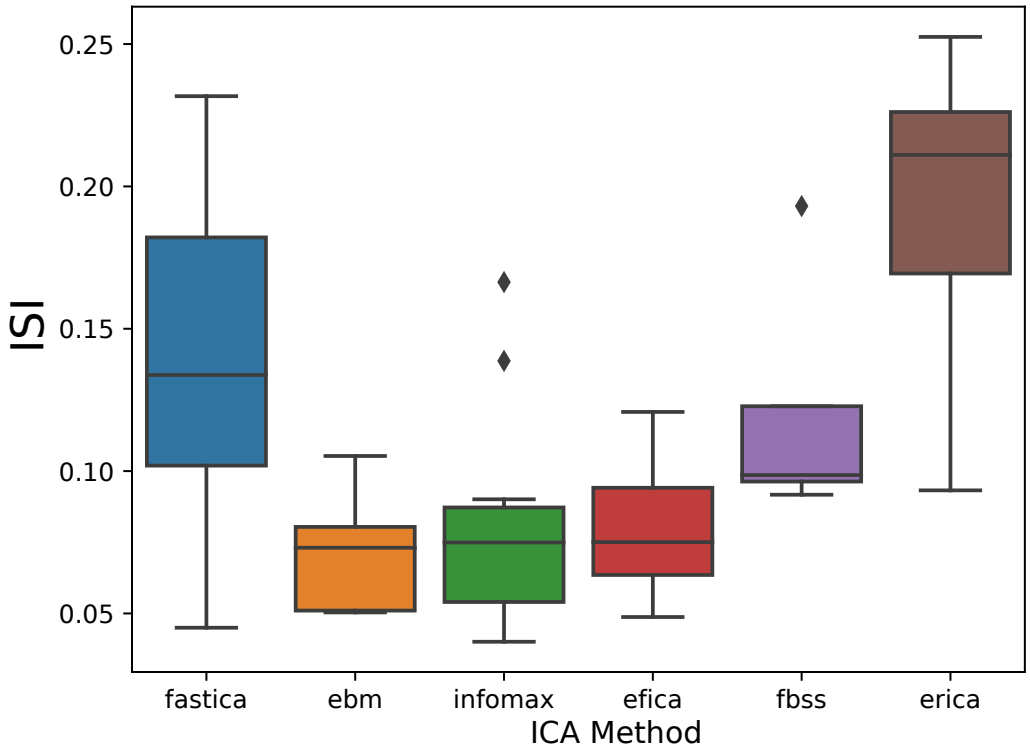
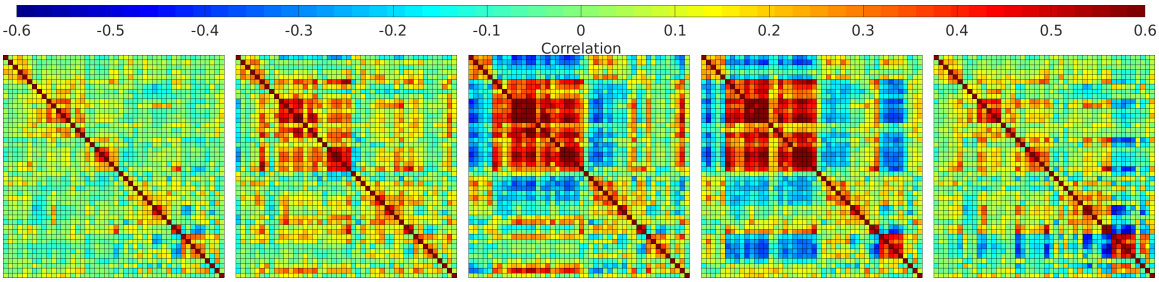



Figure 3.6: Figure reprinted with permission. The Moreau-Amari Index (y-axis) computed for our algorithm, compared over multiple ICA algorithms (x-axis). Choices of ICA algorithm were evaluated 10 times over the same set of principal components, and then compared with the ground truth set of estimated components.

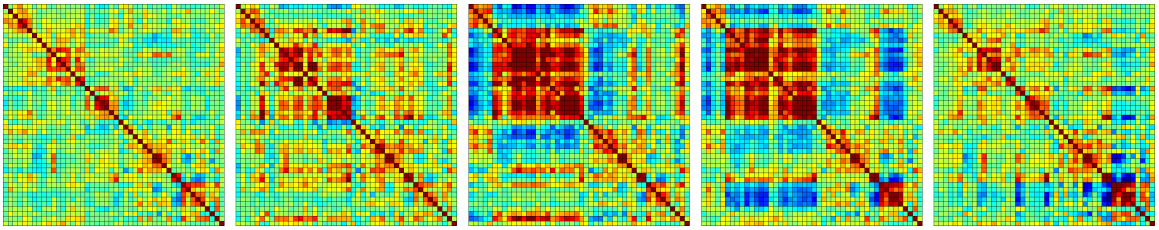
the quality of estimation, and we leave the further problem of assuring estimation along with quality for future work.

### 3.3.3 ddFNC results

In 3.9 we plot some examples of the components estimated from decentralized spatial ICA in comparison with the spatial components from [148], after performing Hungarian matching between the estimated spatial maps. We also plot the correlation of the components from our ICA implementation in comparison to the components from [148]. Indeed, the estimated components are highly correlated with the results from [148], for all 100 estimated components, as well for the 47 selected neurological components from [148], indicating




(a) Figure reprinted with permission. Median Centroids over all groups, estimated with Pooled dFNC [148]



(b) Figure reprinted with permission. Median Centroids over all groups, estimated with ddFNC

Figure 3.7: Figure reprinted with permission. The $k = 5$ median centroids over all groups for pooled dFNC from [148] (panel 3.7a), and the hungarian-matched centroids from ddFNC (panel 3.7b).

that dgICA is able to produce results comparable to the pooled case. We include additional spatial maps for all 47 estimated spatial components in the supplementary material.

First, in 3.8, we plot the correlation between the centroids estimated with our method, and those estimated with a pooled gICA and pooled K-Means. Decentralized centroids estimated with decentralized LLoyd's algorithm match better to the pooled case, with each centroid correlating above 98% with the pooled estimation. The gradient-descent implementation does not converge to the pooled solution as well, though the results are still greatly similar, correlating above 85% with the pooled case.

The improved performance of decentralized Lloyd's algorithm can be explained in part by the lack of thorough hyper-parameter searching for the Gradient-Descent based algorithm, which would likely improve the results. For the purpose of this work, since Lloyd's algorithm provides near perfect estimation of the pooled centroids, we leave the task of decentralized hyper-parameter searching for future work.

In 3.7 we plot the centroids from [148] (panel 3.7a), as well as the centroids estimated

using decentralized dFNC (panel 3.7b). Additionally, we plot the correlation between the centroids estimated with our method, and those estimated in the pooled case, given in 3.8.

In 3.10 through 3.12, we plot the group centroids for healthy controls (3.10), patients with schizophrenia (3.11), and the differences between each group (3.12). Although our results show slight differences compared to the analysis in [148], states 2 and 4 from our estimation closely resemble states 2 and 3 in [148], with the high anticorrealtion within the sensory and motor regions. Our estimation of state 4 best fits with state 3 from [148], showing greater sensory-motor anticorrelations than our state 2, as well as higher activation in the default mode. Our states 1 and 5 bear striking similarity to one another, and best compare with states 4 and 5 from [148], while our state 3 compared best with state 1 from [148].

### 3.3.4 Privacy

One of the advantages of decentralized analysis pipelines is that only intermediary statistics are passed between sites, and full patient records never are released across the network. These kinds of decentralized algorithms are "plausibly private" [47], due to the lack of directly identifiable records in the global data network. Our pipeline for ddFNCis clearly plausibly private, since no full data instances are explicitely passed between sites during analysis.

The limitation of plausibly private algorithms is that the actual ensured privacy is not quantifiable, with risk of identification never clearly assured. Measures such as Cynthia Dwork's differential privacy [174] have been proposed to alleviate the concerns of plausible privacy, with concrete mechanisms available to ensure privacy up to a given level with some loss of model utility accrued in exchange for privacy assurances [175].

The addition of differential privacy introduces further problems to a pipeline which often involve new variables in the pipeline such as optimal privacy mechanism, choice of privacy budget, and how the privatized algorithm compares in terms of utility with non-

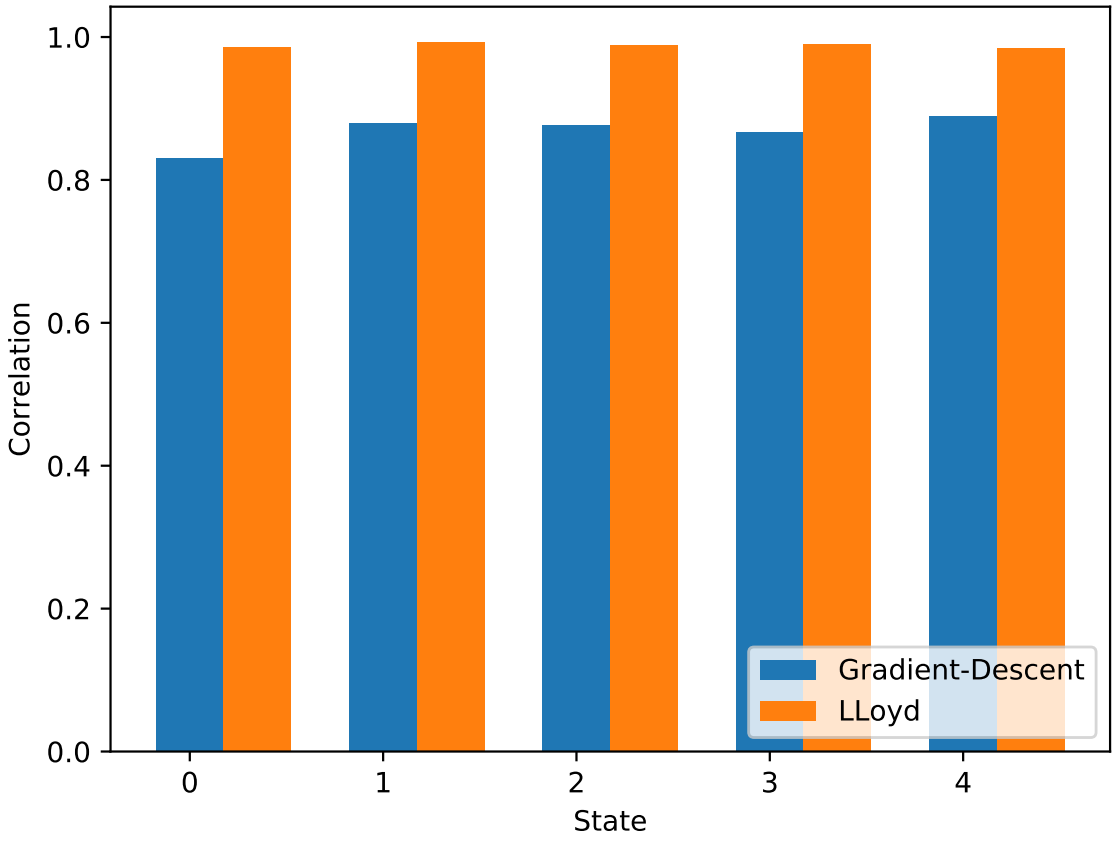


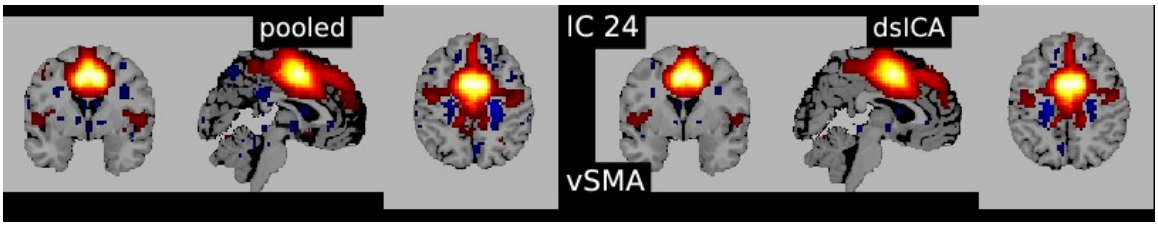

Figure 3.8: Figure reprinted with permission. Correlation between pooled centroids and decentralized centroids estimated using decentralized LLoyd's algorithm and decentralized Gradient-Descent. The centroids from LLoyd's algorithm are much closer to the pooled case.

privatized models. Thus, our pipeline represents an important first step towards fully differentially private ddFNC, providing a clear direction for future work.

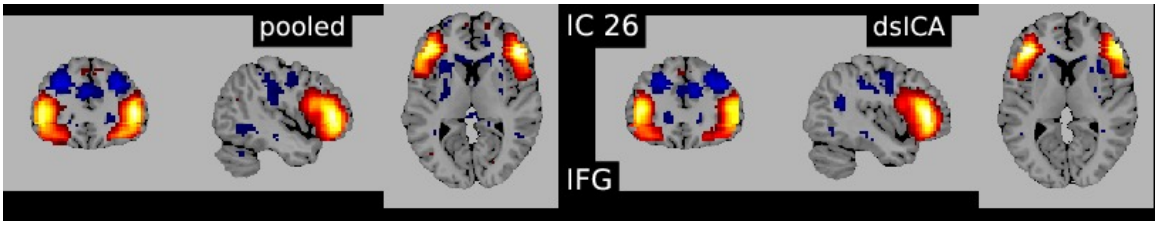
### 3.4 Conclusion

In this paper, we presented a simple case study of how functional network connectivity analysis can be performed on multi-site data without the need for pooling data at a central site. The study shows that both the decentralized regression as well as the decentralized dynamic functional network connectivity yield results that are comparable to its pooled counterparts guaranteeing a virtual pooled analysis effect by a chain of computation and communication process. Other advantages of such a decentralized platform include data privacy and support for large data. Further extensions to the decentralized regression algorithm presented here include- adding a regularization term (ridge, lasso and elastic-net) to the objective function, standardized development of gradient descent schemes to perform optimization in a more iterative fashion and developing a differential privacy version for each algorithm. In conclusion, the results presented here strongly encourage the use of decentralized algorithms in large neuroimaging studies over systems that are optimized for large-scale centralized data processing.

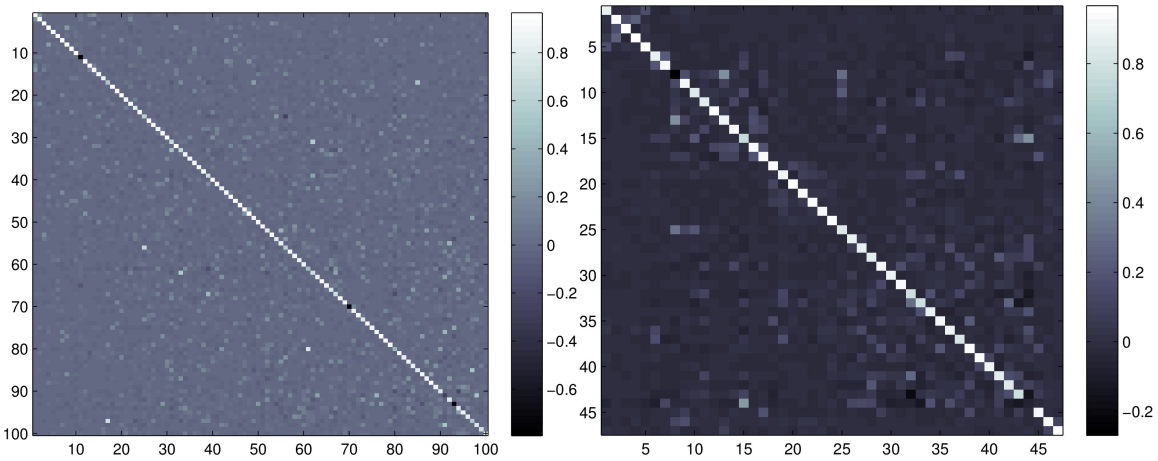
citations




(a) Figure reprinted with permission. Activation in the ventricular Supplmentary Motor Area (vSMA)



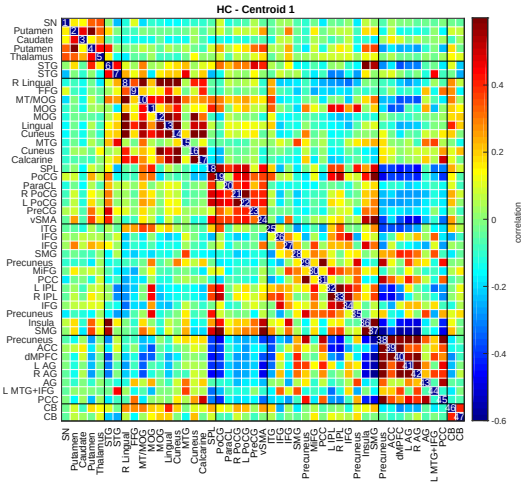

(b) Figure reprinted with permission. Activation in the Inferior Frontal Gyrus (IFG)



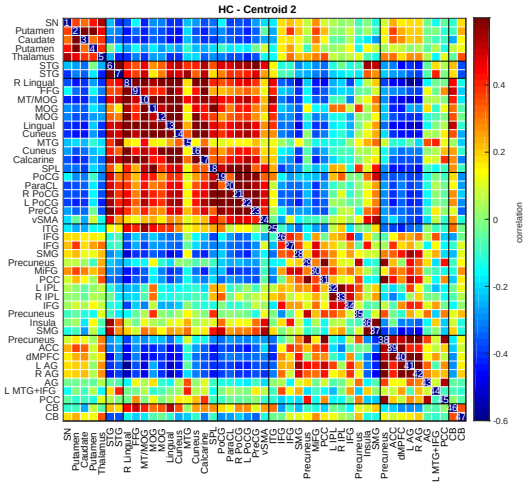

(c) Figure reprinted with permission.

(d) Figure reprinted with permission.

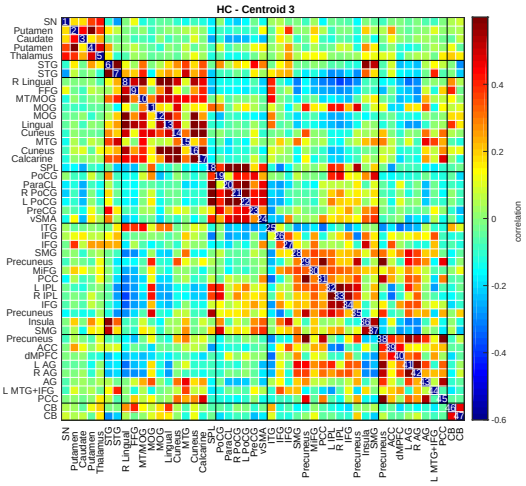
Figure 3.9: Figure reprinted with permission. Panels 3.9a-3.9b illustrate examples of matched spatial maps from dgICA and pooled ICA. Panels 3.9c and 3.9d show the correlation of the components between pooled spatial ICA and dgICA after hungarian matching. Panel 3.9c shows correlation between all 100 components, and panel 3.9d shows correlation between the 47 neurological components selected in [148].



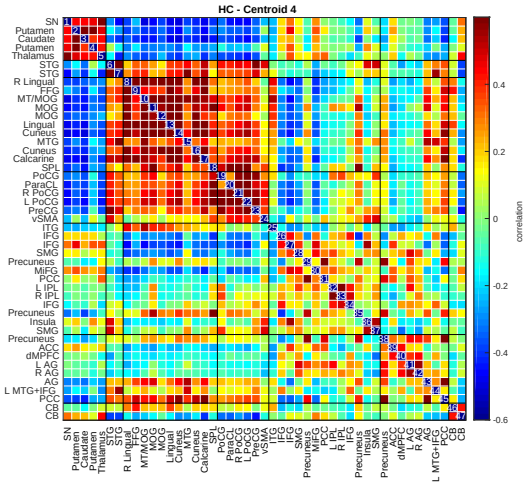

(a) Figure reprinted with permission. Healthy Control State 1



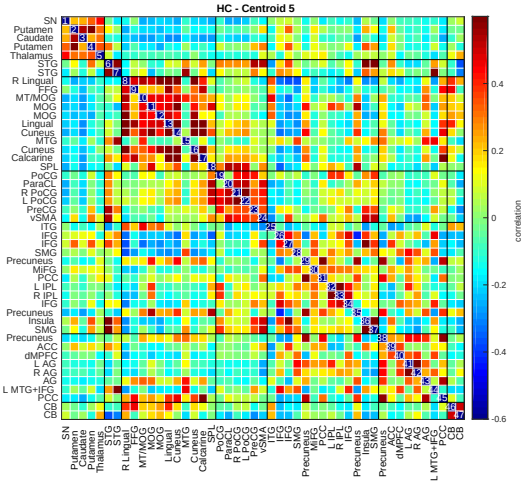

(b) Figure reprinted with permission. Healthy Control State 2




(c) Figure reprinted with permission. Healthy Control State 3

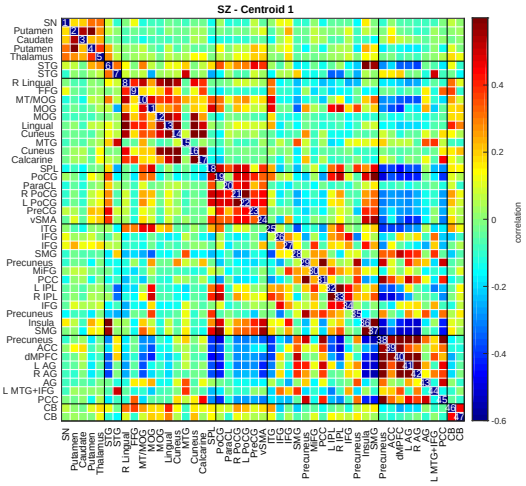



(d) Figure reprinted with permission. Healthy Control State 4s

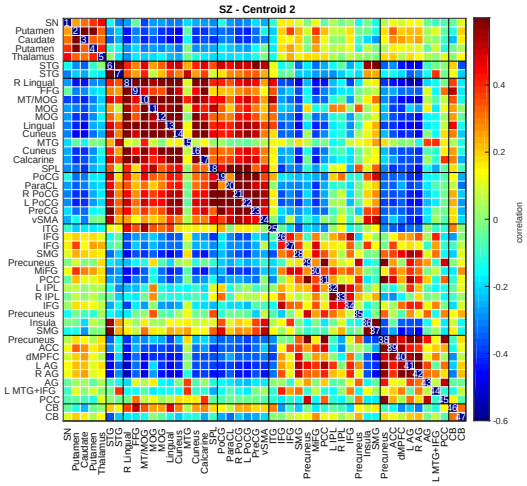



(e) Figure reprinted with permission. Healthy Control State 5

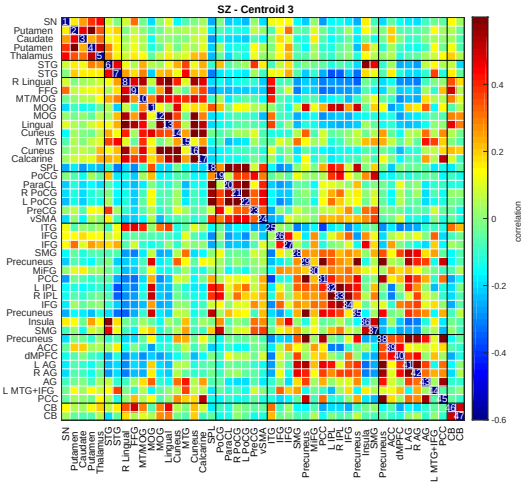
Figure 3.10: Figure reprinted with permission. Estimated median states for 163 healthy controls, computed using decentralized dFNC with $k = 5$, using the original site configuration from the Fbirn data set described in section 3.2.4.



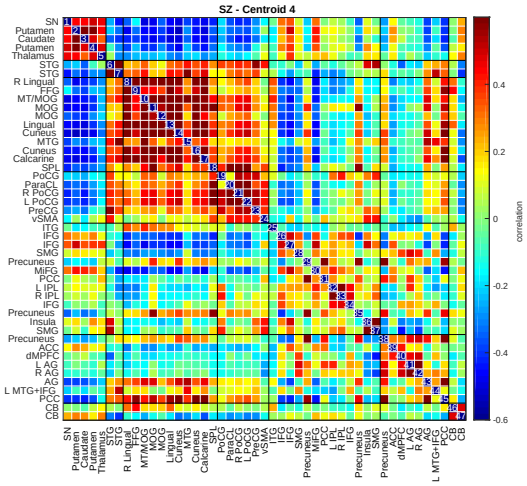

(a) Figure reprinted with permission. Patient State 1



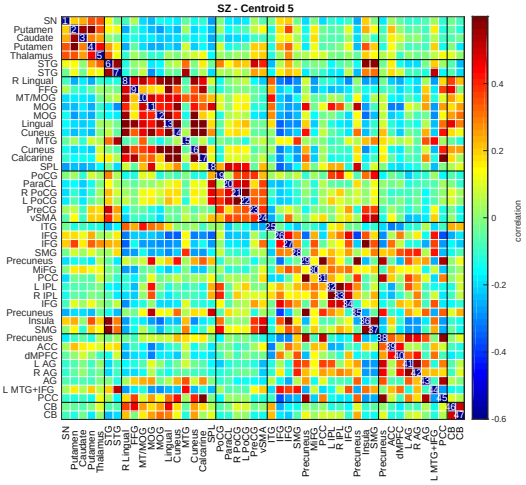

(b) Figure reprinted with permission. Patient State 2




(c) Figure reprinted with permission. Patient State 3




(d) Figure reprinted with permission. Patient State 4




(e) Figure reprinted with permission. Patient State 5

Figure 3.11: Figure reprinted with permission. Estimated median states for 151 patients, computed using decentralized dFNC with $k = 5$, using the original site configuration from the Fbirn data set described in section 3.2.4.

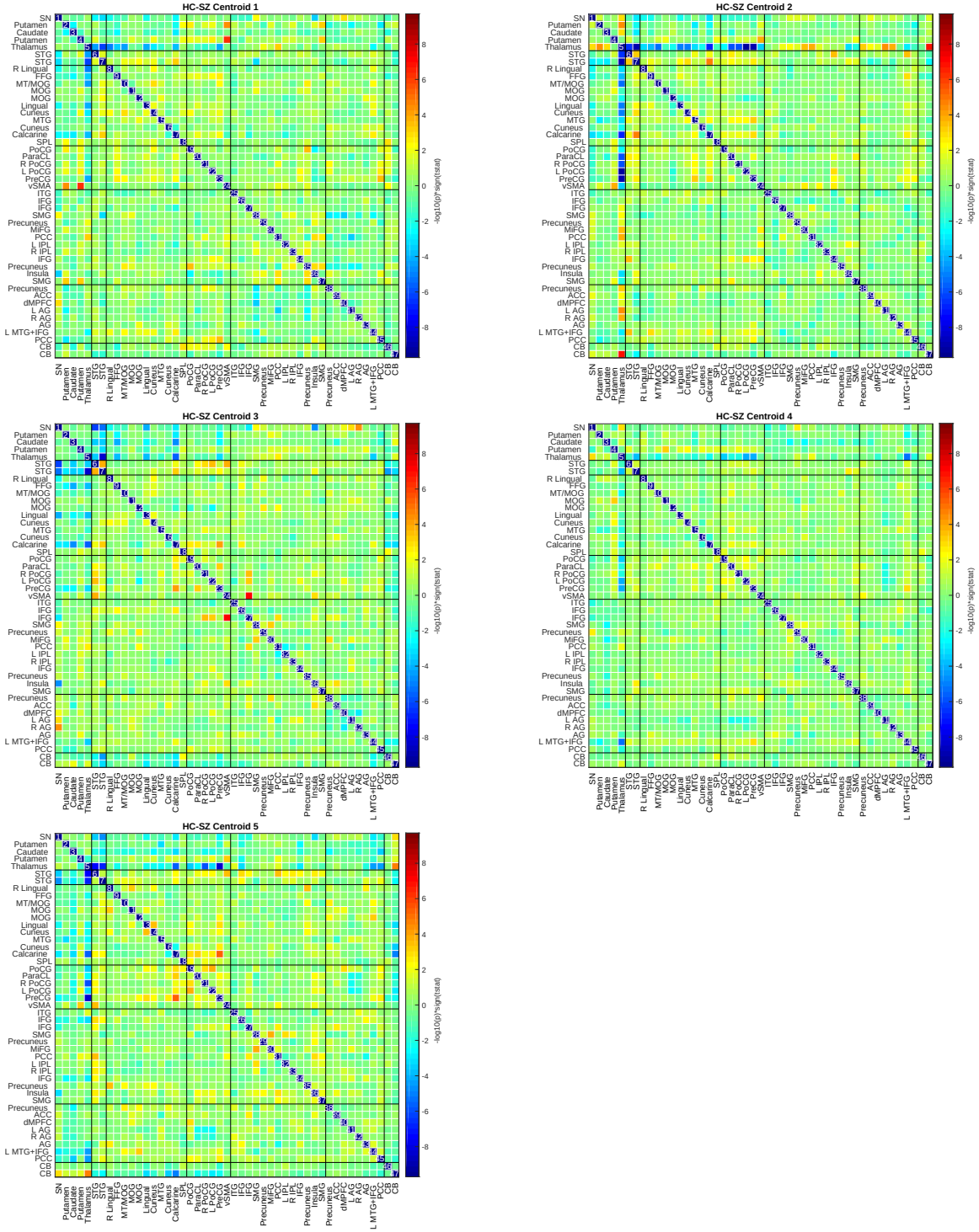



Figure 3.12: Figure reprinted with permission. Estimated median group differences for a two-tailed t-test between the 151 patients and 163 healthy controls, computed after decentralized dFNC with $k = 5$, using the original site configuration from the Fbirn data set described in section 3.2.4.

# CHAPTER 4

# PEERING BEYOND THE GRADIENT VEIL WITH DISTRIBUTED AUTO DIFFERENTIATION

## 4.1 Introduction

The distributed deep learning community has long gravitated towards methods which share gradients during training [176, 177]. Owing in part to the linearity of the gradient, methods like distributed stochastic gradient descent (dSGD) have served as the backbone for large-scale frameworks such as horovod [178], PyTorch [179], and others. When viewed through the lens of auto-differentiation (AD) [180], however, we can easily observe that the gradient is computed as the outer-product of two smaller matrices, which are accumulated during the forward and backward passes through the network. In this work, we develop this simple fact into an elegant new framework for distributed deep learning. We show that methods grounded in AD naturally provide a bandwidth reduction over standard dSGD and other state of the art methods like PowerSGD, along with competitive performance. We aim to show that much can be gained by turning the focus of distributed learning away from gradient-centrism and toward auto-differentiation.

The many parameters at work in deep neural networks (DNNs) require significant amounts of data to train, with over-fitting becoming a real possibility if not enough data are provided, or the network is not otherwise regularized. The need for training on large amounts of data in reasonable time has led the deep learning community to focus on data-parallel training, where models on different (GPU) processors are synchronously trained on their respective subsets of data [181] maintaining the same gradient. Distributed deep learning can also be motivated by a desire to keep local training samples hidden. For example, the application of deep learning to medical problems which utilize highly personal

data such as medical imaging scans or DNA sequences can require models to be trained on samples which cannot be transferred from one data gathering site to another due to legal or ethical considerations. These issues motivate privacy-sensitive toolboxes for distributed learning [44].

One of the main obstacles to the scalability of distributed deep learning is the bottleneck introduced by the large amount of information transmitted over the network during training. Zhang et al. [182] showed that in $< 100$ Gb/s networks, training runtime is significantly worse without compression, preventing linear scaling with workers or model size. Svyatkovskiy et al. [183] also showed that runtime increased with network size when training distributed RNNs. In extreme cases, [184] where the number of parameters is much larger than network bandwidth, communication time dominates training time. Finally, efficient communication of statistics during training has been a preoccupation of algorithm designers since the advent of the field (see [185]&[178] for specific examples, and [186] for survey; also methods mentioned in related works). Even if network architectures are ideally constructed, communication will always limit the overall runtime of distributed algorithms.

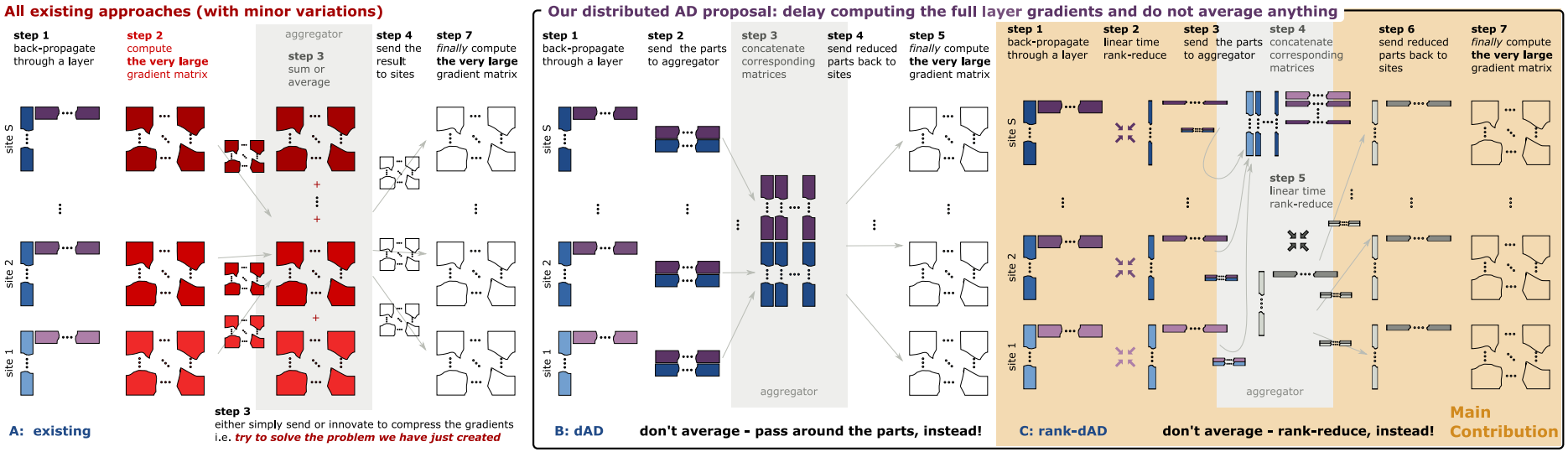



Figure 4.1: Figure reprinted with permission. A high level depiction of the state of the art approaches to distributed SGD (A), demonstration of our observation that working with AD process directly provides bandwidth reduction (B), and a linear reduction algorithm that significantly reduces the bandwidth (C).

If we take a step back from gradient-centric distributed methods, however, we notice a simple but startlingly profound observation regarding the gradient's structure. In reverse-mode auto-differentiation, the gradient of a layer is computed as an outer-product of the input activations to that layer and the partial derivative with respect to that layer's output. These two component matrices are themselves often smaller in size than the full gradient, and perhaps more importantly, they represent an explicit structure at work behind the gradient computation which deserves consideration on its own ground. In this work, we will aim to show that distributed auto-differentiation algorithms exploiting this inherent outer-product structure exhibit myriad benefits, such as significant bandwidth reduction and performance improvements, when compared to standard dSGD algorithms.

Our contributions in this work can be summarized as follows: First, we highlight a key observation that the outer product structure of the gradient inspires a class of inherently communication efficient distributed learning algorithms. Next, we present a method which elegantly and naturally arises from the gradient's outer-product structure: rank-distributed Auto Differentiation (rank-dAD). Rank-dAD exploits the gradient's outer-product structure so that rank-reduced estimates of the component matrices can be efficiently communicated to reconstruct an estimate for the true global gradient. Through the use of a new, linear-time algorithm for structured power iterations (SPI), rank-dAD drastically reduces bandwidth while still maintaining model performance. The differences between the existing approaches and our proposed step away from the gradient compression are highlighted in Figure 4.1. We illustrate our methods on benchmark deep learning problems such as digit recognition with MNIST, continuous time-series classification with a GRU-RNN on the UEA datasets [187], and large-scale image recognition with a vision-transformer [188] on the cifar-10 dataset.

## 4.2 Methods

Let $\mathcal{N}(W)$ be a deep neural network with $L$ hidden layers. Let $\mathbf{X} \in \mathbb{R}^{N \times M}$ be the input batch of data with $N$ samples and $M$ dimensions. Let $\mathbf{Y} \in \mathbb{R}^{N \times C}$ be the target variable of dimension with $N$ samples and $C$ dimensions. let $h_i$ be the size of the $i$th hidden layer. For a feed-forward network, the weight matrices are thus $\mathbf{W}_i \in \mathbb{R}^{h_i \times h_{i+1}}$, with $h_0 = M$ and $h_{L+1} = C$. Let $\varphi_i(x)$ be the activation used at layer $i$ in the network, and let $\mathcal{L}$ be a given cost-function.

For a feed-forward network, the activations at layer $i$ are thus computed as

$$\mathbf{Z}_i = \mathbf{A}_{i-1}\mathbf{W}_i + \mathbf{B}_i \qquad \mathbf{A}_i = \varphi_i(\mathbf{Z}_i) \tag{4.1}$$

### 4.2.1 Reverse-Mode Auto-Differentiation

AD is a class of methods through which derivatives of functions may be calculated during the execution of a code which evaluates that function. The backpropagation algorithm for training deep neural networks is specific case of *reverse-mode* AD, in which derivatives are propagated *backwards* along the data flow graph. This is a two stage process: first a *forward pass* through the function is computed, during which intermediate outputs from expression saved and relationships between variables are recorded. After the forward pass, a *backward pass* evaluates the contribution of each intermediate variable to the derivative of the output, retracing and combining intermediate variables and expressions until returning to the input. [For a survey, see 189].

Following reverse mode AD, we use the chain rule to compute the derivative at each layer. At the output layer, we first compute the gradient of the loss w.r.t. the output activations $\nabla_{\mathbf{A}_L}\mathcal{L}$, and take the Hadamard product with the derivative at the output activation

$\varphi'_L(\mathbf{Z}_L)$ to compute the gradient w.r.t. the output, i.e., $\mathbf{\Delta}_L$

$$\mathbf{\Delta}_L = \nabla_{\mathbf{A}_L}\mathcal{L} \odot \varphi'_L(\mathbf{Z}_L) \tag{4.2}$$

At higher layers (where $i < L$), we can continue to compute these errors as

$$\mathbf{\Delta}_i = (\mathbf{\Delta}_{i+1}\mathbf{W}_i \odot \varphi'_i(\mathbf{Z}_i)) \tag{4.3}$$

At layer $i$, the gradient of the weights $\nabla_{W_i}\mathcal{L}$ at layer $i$ can thus be computed exactly as

$$\nabla_{\mathbf{W}_i}\mathcal{L} = \mathbf{A}_{i-1}^\top \mathbf{\Delta}_i \tag{4.4}$$

The key insight offered to us by the reverse-mode AD perspective into deep learning is that in many cases the dimensionality of the intermediate variables accumulated during the forward and backwards passes of AD will be less than that of the gradient. In other words. the gradient is a low rank matrix in most practically relevant cases, and this rank is limited from above by the batch size. For example, consider a matrix-vector product $y = Wx$. If Alice is transmitting the gradient $\nabla_W\mathcal{L}$ to Bob, where only Alice knows $x$, we can note that $\nabla_W\mathcal{L}$ is the outer product of $x$ and $\nabla_y\mathcal{L}$. If $W \in \mathbb{R}^{m\times n}$, then these two vectors together have dimensionality $m + n$, and we are often in the regime where $m + n \ll m \times n$.

This observation suggests a novel algorithm for backpropagation via distributed auto-differentiation (dAD). We present this algorithm and some more communication-efficient variants in the following sections.

### 4.2.2 Exploiting Outer-Product Structure for Distributed Learning

The core of our work relies on a rather simple observation: standard auto-differentiation computes the gradient via an outer-product of two smaller matrices (see equation 4.4). This elementary fact encourages a new way of looking at distributed learning in which the

gradient's component matrices are shared instead of the full gradient itself, as is standard practice. Indeed, in cases where the batch size is significantly less than the hidden dimension ( i.e. $N << h_i$ ) if we were to simply transfer the two component matrices of the gradient, we would already significantly reduce communication overhead. After communicating component matrices, sites can compute *exact* gradients by merely concatenating in the batch dimension, and computing the product as normal.

Although the naïve circulation of component matrices seems an attractive approach on its own, certain difficulties emerge motivating further work to create a more practically useful distributed learning algorithm. One problem, for example, with fully communicating component matrices is that bandwidth usage is directly tied to the chosen batch size, which can be undesirable in applications where large batch-size is desired, or where we accumulate results over an additional sequence dimension, such as when utilizing recurrent architectures. Our fully-realized method achieves efficient communication by capitalizing on the outer-product structure to perform a structured rank-reduction of the component matrices in linear time. In the next sections, we present our complete algorithm for rank distributed auto-differentiation (rank-dAD), the first in our newly-revealed class of efficient algorithms for distributed auto-differentiation.

**Algorithm 11** Figure reprinted with permission. rank distributed auto-differentiation (rank-dAD)

**Require:** $\{\mathcal{N}_s\}_s^S, \{\mathbf{X}_s\}_s^S, \{\mathbf{Y}_s\}_s^S$

**for all** site $s$ **do**
  $\{\mathbf{A}_i^{(s)}\}_{i=0}^L = \text{forward}(\mathcal{N}_s, \mathbf{X}_s)$
**end for**
**for all** hidden layer $i = L, 0 < i \leq L$ **do**
  **for all** site $s$ **do**
    **if** i == L **then**
      $\boldsymbol{\Delta}_L^{(s)} = \nabla_{\mathbf{A}_L} \mathcal{L} \odot \varphi'(\mathbf{Z}_L^{(s)})$
    **else**
      $\boldsymbol{\Delta}_i^{(s)} = \boldsymbol{\Delta}_{i+1}^{(s)} \mathbf{W}_i^{(s)} \odot \varphi_i'(\mathbf{Z}_i^{(s)})$
    **end if**
    $\mathbf{Q}_i^{(s)}, \mathbf{G}_i^{(s)} = \text{SPI}(\boldsymbol{\Delta}_i^{(s)}, \mathbf{A}_{i-1}^{(s)})$
  **end for**
  $\mathbf{Q}_i = \text{vertcat}(\{\mathbf{Q}_i^{(s)}\}_s^S)$ ▷ At Aggregator
  $\mathbf{G}_i = \text{vertcat}(\{\mathbf{G}_i^{(s)}\}_s^S)$ ▷ At Aggregator
  $\hat{\mathbf{Q}}_i, \hat{\mathbf{G}}_i = \text{SPI}(\mathbf{Q}_i, \mathbf{G}_i)$ ▷ see Algorithm 12
  broadcastToSites($\hat{\mathbf{Q}}_i, \hat{\mathbf{G}}_i$)
  **for** site $s$ **do**
    $\nabla_{\mathbf{W}_i}^{(s)} = \hat{\mathbf{P}}_i^\top \hat{\mathbf{Q}}_i$
  **end for**
**end for**

### 4.2.3 Rank distributed Auto Differentiation

The outer-product structure of the gradient invites myriad improvements for distributed learning. In this section, we elaborate on how iterative rank-reduction methods can capitalize off of the outer-product structure, fostering an algorithm in which rank-reduced compo-

nent matrices can be computed efficiently and communicated instead of the full gradient. These component matrices provide a method for estimating the true global gradient while reducing the overall bandwidth usage from quadratic to linear with respect to the layer size. We call this method rank distributed auto-differentiation (rank-dAD), as it combines iterative rank-reduction with our core insight of distributed auto-differentiation into a single, efficient algorithm.

First, we assume that all sites coordinate the initialization of local copies of the chosen architecture - for example, they can share a choice of random seed and probability distribution when generating initial weights. Each site will maintain a local copy of the model weights in memory, and these models will ultimately have equal weights. For simplicity, each site also shares a set of hyper-parameters, such as learning rate, momentum, number of epochs, etc. In principal, hyper-parameters and even certain architectural elements could be allowed to vary between sites based on local needs; however, such circumstances will not affect the overall performance of the model in terms of its communication and computational benefits compared to standard distributed algorithms, and so we leave these as future work.

For a given batch, rank-dAD first performs the typical forward and backward passes from reverse-mode auto-differentiation, accumulating local activations and partial derivatives as would normally be used for local gradient computation. Next, these component matrices are rank-reduced along the batch dimension. Formally, we begin with $\mathbf{A}_{i-1} \in \mathbb{R}^{N \times h_{i-1}}$ and $\mathbf{\Delta}_i \in \mathbb{R}^{N \times h_i}$, and reduce these to matrices $\mathbf{G} \in \mathbb{R}^{k \times h_{i-1}}$ and $\mathbf{Q} \in \mathbb{R}^{r \times h_i}$ where $r$ is a chosen natural number designating our maximum target rank. Following rank reduction, we transmit the rank-reduced matrices to a single aggregator, or set of aggregators which will perform the next further reduction step. At aggregator nodes, we then concatenate the received $\mathbf{Q}$ and $\mathbf{G}$ matrices, and perform a final rank-reduction to obtain $\hat{\mathbf{Q}}$ and $\hat{\mathbf{G}}$. Once the reduction has obtained data from all sites in the network, we broadcast the final reduced component matrices, and can locally estimate the gradient from these

components. For simplicity, we will only designate one node in the network as the aggregator, as this models the architecture supported by COINSTAC [44], our target platform; however, in principal, multiple aggregation stages could be used, as in ring-reduce, or hierarchical communication frameworks for example. We leave the detailed investigation of these alternate communication setups as future work.

**Algorithm 12** Figure reprinted with permission. Structured Power Iterations (SPI)

**Require:** $\mathbf{\Delta}_i \in \mathbb{R}^{N \times h_i}$, $\mathbf{A}_{i-1} \in \mathbb{R}^{N \times h_{i-1}}$, $r \in \mathbb{N}_+$, $n, \theta = 10^{-3}$

$\mathbf{Q} = [\,], \mathbf{G} = [\,]$

$\mathbf{C} = \mathbf{A}_{i-1}\mathbf{A}_{i-1}^\top$

$\mathbf{B} = \mathbf{\Delta}_i^\top \mathbf{C}$

**for** $j = 0, j \leq r$ **do**

    $\mathbf{g}_0^j \sim \mathcal{N}(0, 1)$

    **for** $k = 1, k < n$ **do**

        $\mathbf{g}_{k+1}^j = \mathbf{B}\Delta_i \mathbf{g}_k^j - \mathbf{Q}(\mathbf{G}^\top \mathbf{g}_k^j)$

    **end for**

    $\mathbf{v} = \mathbf{\Delta}_i \mathbf{g}_k^j$

    $\sigma^j = \sqrt{\mathbf{v}^\top \mathbf{C} \mathbf{v}}$ ▷ May be avoided[1]

    $\mathbf{q}^j = \mathbf{A}_{i-1}^\top \mathbf{v} / \sigma^j$

    **if** $||\mathbf{g}^{j-1} - \mathbf{g}^j||_2 / ||\mathbf{g}^{j-1}||_2 \leq \theta$ **then**

        break

    **end if**

    $\mathbf{Q}$ =concat$(\mathbf{Q}, \mathbf{q}^j)$

    $\mathbf{G}$ =concat$(\mathbf{G}, \mathbf{g}^j)$

**end for**

return$(\mathbf{Q}, \mathbf{G})$

*Structured Power Iterations*

Observe that we can compute the singular vector corresponding to the dominant singular value of $\nabla_{W_i}\mathcal{L}$ by iterating the following recurrence:

$$\mathbf{g}_{k+1}^1 = (\nabla_{\mathbf{W}_i}\mathcal{L})^\top(\nabla_{\mathbf{W}_i}\mathcal{L})\mathbf{g}_k^1 \tag{4.5}$$

Relying on (4.4), and pre-computing $\mathbf{C} = \mathbf{A}_{i-1}\mathbf{A}_{i-1}^\top$, $\mathbf{B} = \boldsymbol{\Delta}_i^\top\mathbf{C}$, we can instead iterate:

$$\mathbf{g}_{k+1}^1 = \mathbf{B}\left(\boldsymbol{\Delta}_i\mathbf{g}_k^1\right) \tag{4.6}$$

Compared to the $O(h^2)$ complexity of the iteration (4.5), the complexity of the structured power iteration is just $O(h \times N)$ and since $N \ll h$ for all practical models, it is linear in $h$. The corresponding singular value $\sigma^1 = \sqrt{\mathbf{v}^T\mathbf{C}\mathbf{v}}$ (where $\mathbf{v} = \boldsymbol{\Delta}_i\mathbf{g}$) is computed in $O(h \times N)$ once per singular vector, while computation of the left singular vector $\mathbf{q}^1$ is just $O(h \times N)$ as $\mathbf{q}^1 = \mathbf{A}_{i-1}^\top\mathbf{v}/\sigma^1$.

We successively collect $(\sigma^j\mathbf{g}^j, \mathbf{q}^j)$, absorbing singular values into one of the vectors[1] constructing respective $\mathbf{G}_j$ and $\mathbf{Q}_j$ by concatenating $\mathbf{g}$s and $\mathbf{q}$s as columns, and proceed to computing the next singular vectors set by peeling the previously computed least-squares optimal low-rank representation:

$$\begin{aligned}\mathbf{g}_{k+1}^j &= \left((\nabla_{\mathbf{W}_i}\mathcal{L})^\top(\nabla_{\mathbf{W}_i}\mathcal{L}) - \mathbf{Q}_{j-1}\mathbf{G}_{j-1}^\top\right)\mathbf{g}_k^j \\ &= (\nabla_{\mathbf{W}_i}\mathcal{L})^\top(\nabla_{\mathbf{W}_i}\mathcal{L})\mathbf{g}_k^1 - \mathbf{Q}_{j-1}(\mathbf{G}_{j-1}^\top\mathbf{g}_k^j)\end{aligned} \tag{4.7}$$

We have already shown that complexity of the first component of (4.7) is linear in $h$, but the second component is clearly linear in $h$ as well. Thus, thanks to the outer-product structure of AD gradients, we can compute low rank approximation in time linear in the layer width

[1] In practice we bypass computing $\sigma^j$ and gain additional speed up because it cancels out in the outer product since $\mathbf{q}^j$ contains $1/\sigma^j$ factor.

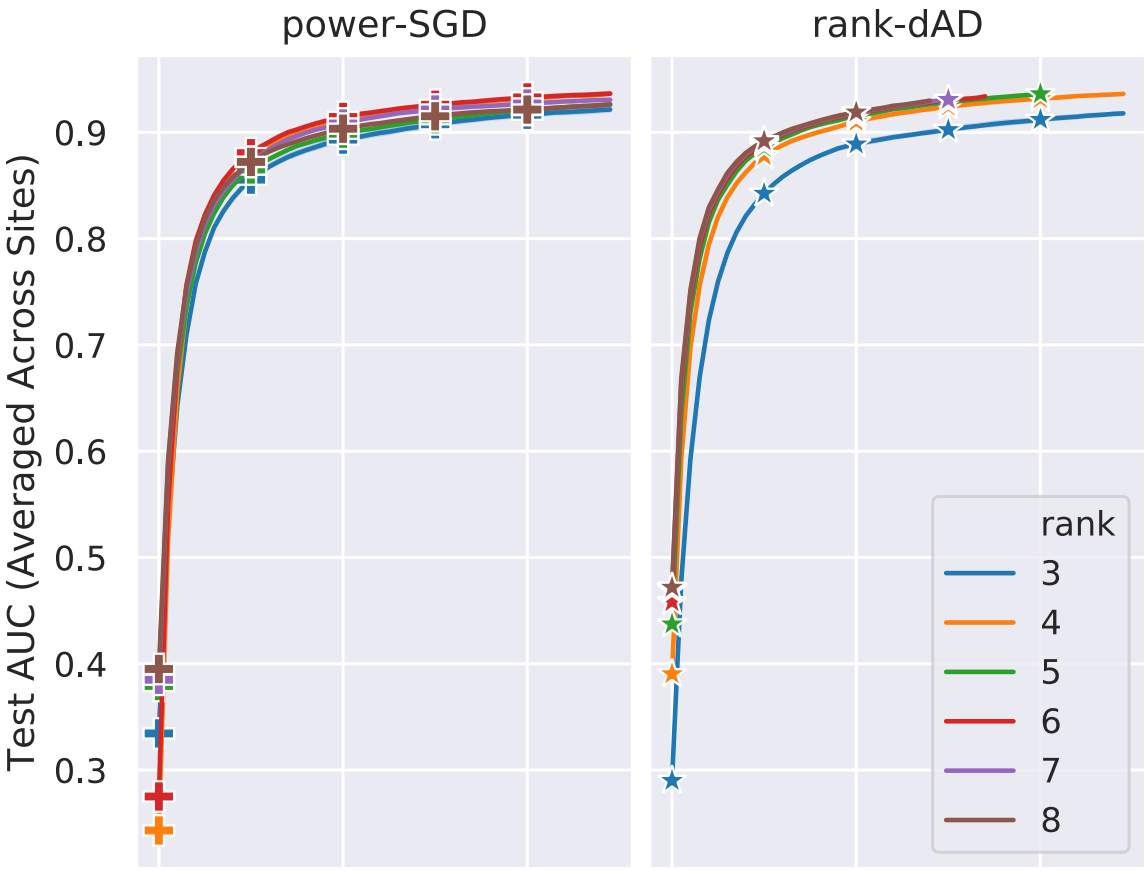



Figure 4.2: Figure reprinted with permission. The average test AUC across sites for power-SGD and rank-dAD with increasing rank on the MNIST data set.

$h$.[2]

We have additionally observed, that during training the true rank of $\nabla_{\mathbf{W}_i}\mathcal{L}$ fluctuates and although always below $N$ may take significantly lower values than the desired $r$ we pick for the structured power iterations. To skip computing noisy columns for our $\mathbf{Q}$ and $\mathbf{G}$ matrices, we stop the process when $\|\mathbf{g}^j - \mathbf{g}^{j+1}\|_2 / \|\mathbf{g}^j\|_2 < \theta$, where we set the threshold $\theta$ to $10^{-3}$.

### 4.3 Results

This section presents the results of several experiments which illustrate the communication and computational benefits of rank-dAD. We begin with small-scale experiments using a

[2]To declutter notation we have dropped the layer index on $h$, as each $h \gg N$.

feed-forward architecture for digit classification of the MNIST data set, and then show a similar small example with a GRU-based recurrent architecture used for multivariate time-series classification on several datasets from the UEA repository. Finally, we show how rank-dAD can be used for tasks using modern transformer architectures for both sentiment analysis of the IMBD dataset, and a larger scale experiment using Vision Transformers [188] for image recognition on Cifar-10. Table 4.1 provides information on the three architectures which were used for experiments.

| Architecture | Problem Type | Hidden Layer Sizes | Sequence Length | Effective Depth | Datasets Tested |
|---|---|---|---|---|---|
| Feed Forward | Digit Recognition | 1024,1024 | - | 2 | MNIST |
| Feed Forward | Disorder Prediction from sMRI | 1024,512,256,128,64,32 | - | 6 | FreeSurfer Volumes Estimated from sMRI |
| GRU | Multivariate Time-Series Classification | 512,256 | 256 | $2 \times 256$ | Spoken Arabic Digits, PENS-SF, NATOPS, Pen-Digits |
| Vision Transformer | Object Recognition | 128 | 50 (patch size 32) | $12 \times 50$ | CIFAR-10 |

Table 4.1: Figure reprinted with permission. Data sets and architectures tested as part of the experiments for this paper.

All experiments were run on a SLURM cluster which submits jobs to one of the 26 machines on the same network. Specs for these machines are provided in Table 4.2. For all experiments, all networks were implemented in PyTorch with Python 3.8 using an Adam optimizer with a fixed learning rate of $10^{-4}$ and batch size of $64$ per site. We performed $k = 5$-fold cross-validation for all experiments, and plot the average results with error bars across these folds. We use the gloo distributed backend for communication between nodes, with all distributed communication methods implemented in native PyTorch.

In Figure 4.2, as a performance sanity-check we compare the Area-Under the Curve (AUC) for digit recognition on MNIST between power-SGD and our method. Table 4.3

Table 4.2: Figure reprinted with permission. Specs for the SLURM cluster used to run the experiments described in §4.3.

| Manufacturer | Cores | Memory | GPUs |
|---|---|---|---|
| AMD | 64 | 512 GB | 1×Nvidia 2080 |
| Nvidia DGX-1 | 40 | 512 GB | 8×Nvidia V100 |
| Dell | 40 | 192 GB | 4×Nvidia V100 |

Table 4.3: Figure reprinted with permission. For a feed-forward network trained on the MNIST data set, average per-batch runtime of rank-dAD as a ratio of the per-batch runtime of dSGD. Even with only 4 sites, rank-dAD sees an over 15 times runtime decrease, and with 18 sites, over 25 times.

| Mode/Sites | 4 | 6 | 8 | 10 | 12 | 14 | 16 | 18 |
|---|---|---|---|---|---|---|---|---|
| dSGD | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |
| rank-dAD | 0.063 | 0.055 | 0.049 | 0.046 | 0.039 | 0.040 | 0.037 | 0.038 |

shows the average per-batch runtime for rank-dAD as a ratio of the average dSGD runtime. Rank-dAD provides comparable performance to dSGD, regardless of the choice of rank, and provides a speedup between 15 and 25 times over vanilla dSGD.

Figure 4.3 plots the effective rank of the gradient as computed by rank-dAD during training. We notice that as the model trains, the rank needed for reliable estimation, and thus overall communication, decreases in all layers of the model.

Figure 4.5 compares the performance of rank-dAD with power-SGD for a GRU-RNN used to classify three data sets from the UEA [187] multivariate time-series repository. Each curve on each axis plots the mean AUC over 5-fold cross-validation for different choices of maximum effective rank. Again rank-dAD performs comparably to power-SGD during training, regardless of the choice of rank.

To test our method applied to a simple neuroimaging problem, we run a simple MLP architecture for the diagnosis of schizophrenia using simulated volumes generated from the Freesurfer package. A total of 72 subjects were simulated with 66 brain regions each, using the formula $y = \beta_0 + \beta_1 \times \text{AGE} + \beta_2 \times \text{isCONTROL} + \beta_3 \times e$, where $e \sim \mathcal{N}(0, 1)$. Each region in the volume has a fixed intercept $\beta_0$ depending on the region (e.g. 48446.3 for Right-Cerebellum-Cortex). For each subject, the effect of age on the model, $\beta_1$, is selected from a uniformly random range [-300, -100], and the effect of diagnosis, $\beta_2$ is selected from a uniformly random range [500, 1000]. Standard unit gaussian noise is multiplied by a random index $\beta_3$ chosen from the range [1800, 2200]. More information on this data set can be found in [190]. To demonstrate the importance of communication bottlenecks in

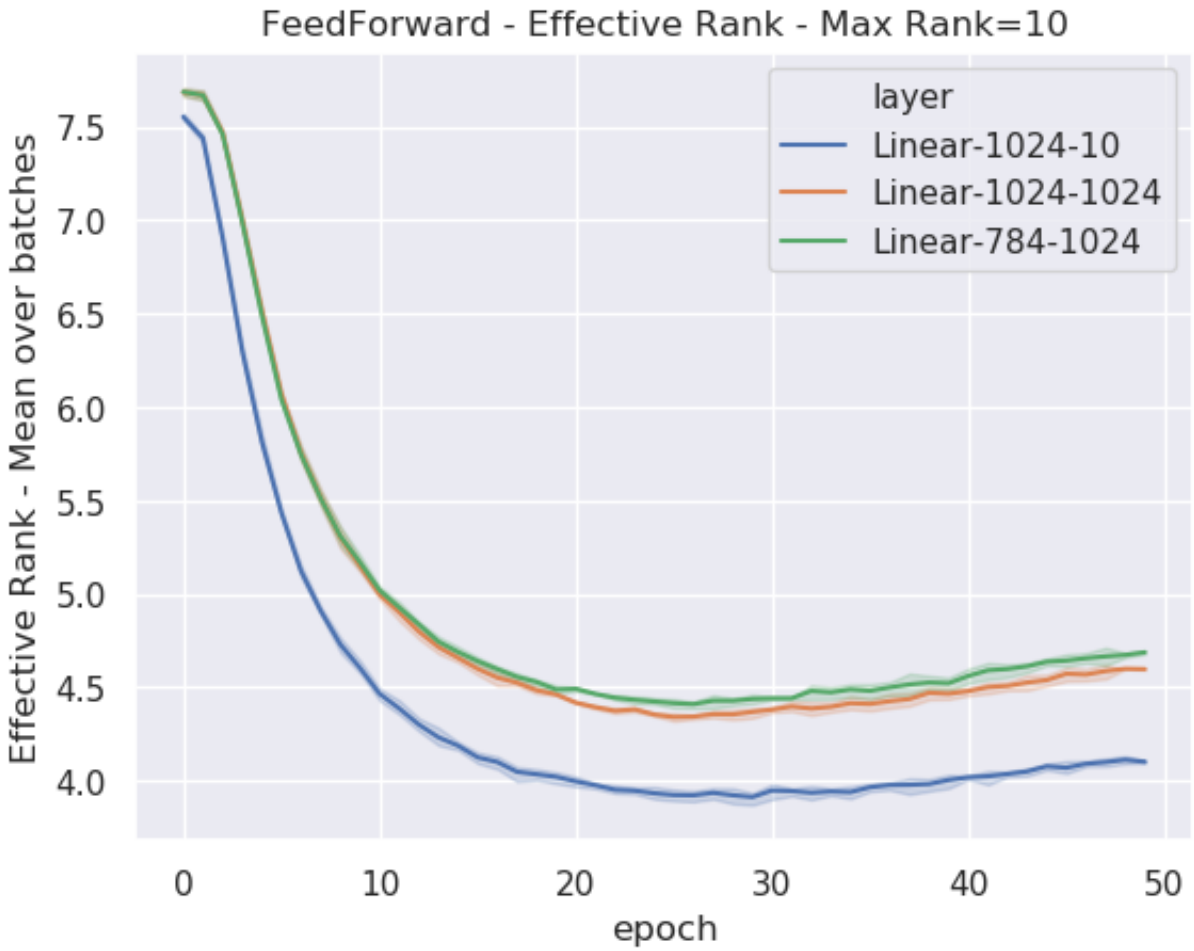



Figure 4.3: Figure reprinted with permission. Effective rank for the MNIST dataset across training time, where the initial maximum rank was set to the number of classes (10).

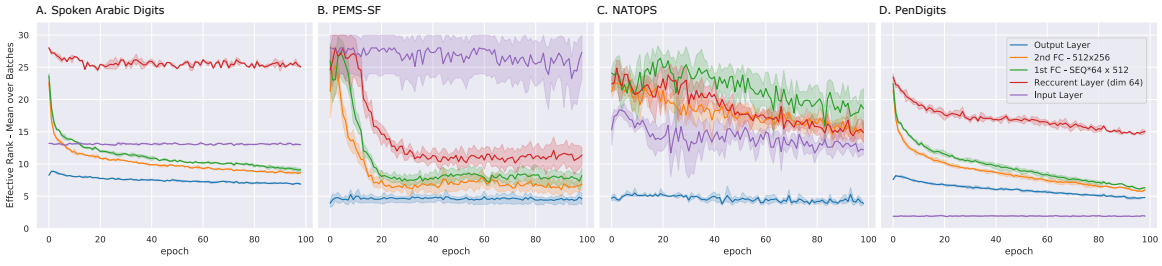



Figure 4.4: Figure reprinted with permission. Effective rank for the four time-series data sets across training time, where the initial maximum rank was set to the batch size of 32.

Table 4.4: Figure reprinted with permission. For a simple Neuroimaging diagnosis task using an MLP, runtime taken to achieve 0.86 AUC for each distributed method, and the speedup compared to dSGD. Our method shows a 2.55 times speedup compared to dSGD, and powerSGD does not converge to the target AUC in the time it takes vanilla dSGD to do so (thus showing a slowdown of 0.704).

| Method | Duration to 0.86 AUC | Speedup vs dSGD | Epochs | Runtime per Epoch |
|---|---|---|---|---|
| **rank-dAD** | **57.723 (s)** | **2.55**× | 20 | 2.886 (s) |
| powerSGD | 184.773 (s) | 0.704× | 23 | 8.033 (s) |
| dSGD | 130.163 (s) | 1× | 10 | 13.016 (s) |

this setting, we tested a connection between nodes in two different regions in AWS (Tokyo and London), and ran vanilla dSGD, powerSGD, and rank-dAD to their best AUC over 100 epochs. We report the wall-clock duration taken by each model to achieve the best validation AUC from dSGD (0.86 AUC). Table 4.4 shows the duration for each distributed method to achieve the target. Rank-dAD demonstrates a **2.55** times speedup over vanilla dSGD, and powerSGD actually takes longer to converge to the target AUC, showing a 0.705 times slowdown compared to vanilla dSGD.

Next, in Figure 4.6, we were interested in examining how in large-scale settings with modern architectures, rank-dAD and dSGD compare with a more rudimentary baseline. As a baseline, we send only the top 3 columns from the activation and delta matrices to the aggregator and use these to compute the gradient. This amounts to effectively reducing the batch-size to 3, and discarding the majority of each batch. We illustrate in this figure how quickly it takes rank-dAD and dSGD to train to AUC comparable to this baseline.

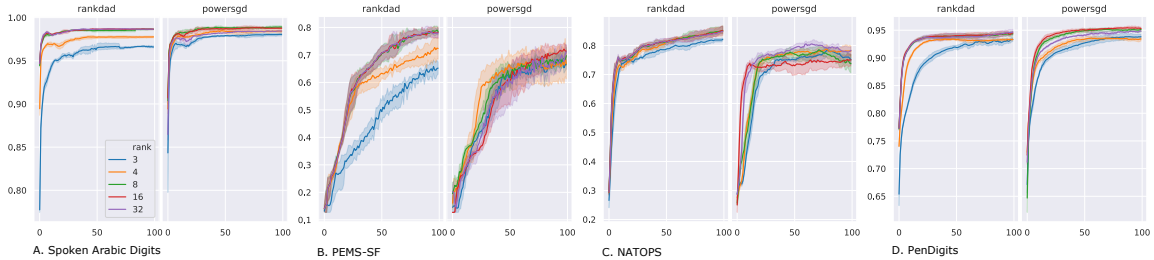



Figure 4.5: Figure reprinted with permission. The test AUC for a GRU trained with rank-dAD compared with the same architecture trained with PowerSGD. Each curve in the two plots provides the AUC over training for a different maximum rank.

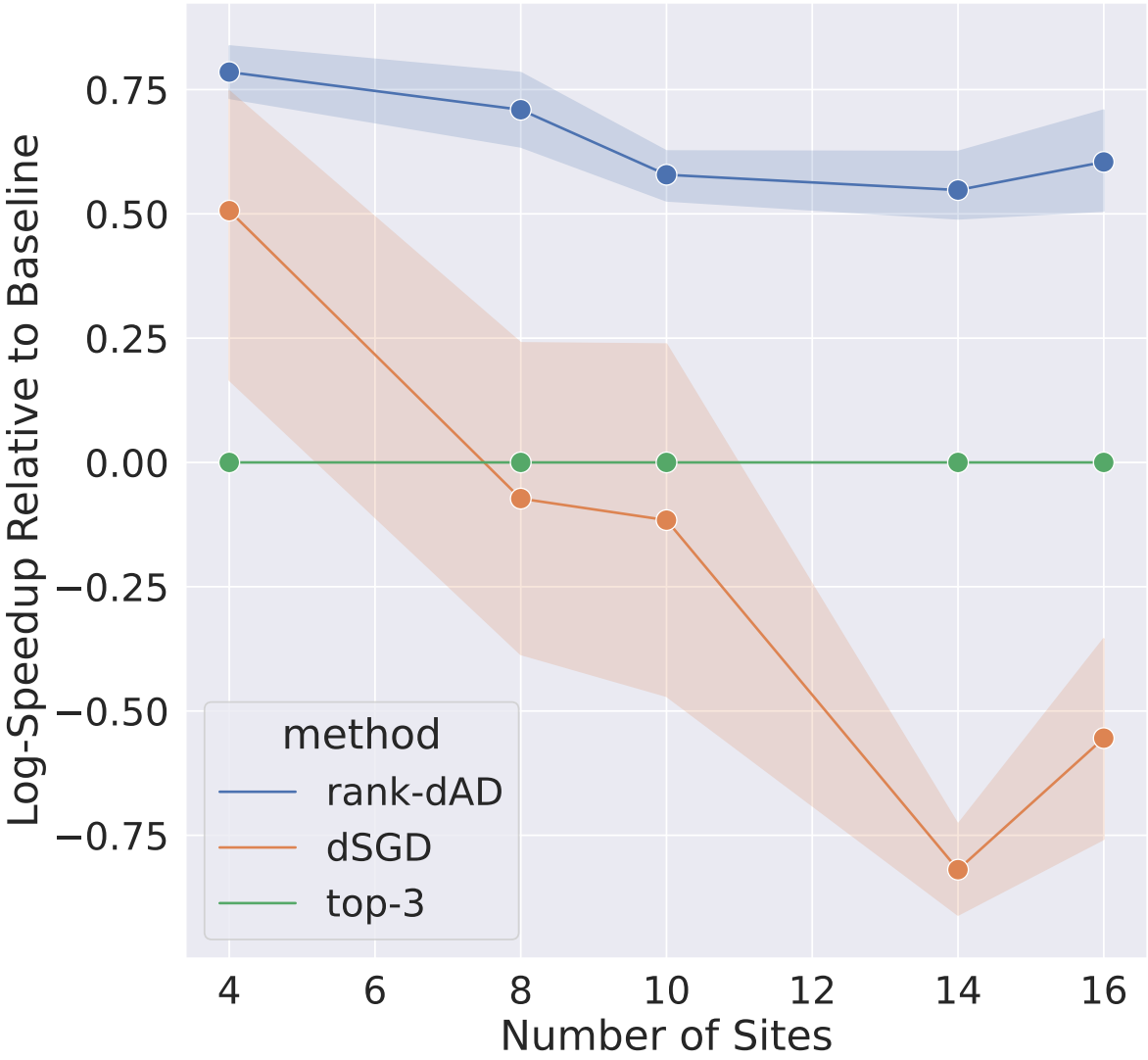



Figure 4.6: Figure reprinted with permission. Using a Vision Transformer, the log speedup for achieving matching baseline validation **AUC (0.91)** on CIFAR-10 trained with rank-dAD and dSGD compared to the baseline of sharing the top 3 columns from the activation and delta matrices. In this plot, positive values represent a faster convergence rate in terms of wall-clock runtime, and negative values represent a slowdown.

Using a Vision-Transformer as our base architecture for image recognition on CIFAR-10, our results clearly illustrate that rank-dAD provides a clear speedup over dSGD, achieving comparable AUC in a much faster runtime. Since our baseline represents a lower bound on the communication allowed between sites, we also illustrate here that rank-dAD, despite having the same communication as the baseline, is able to achieve comparble AUC faster, illustrating the clear benefit of using the rank-reduction method to leverage information from all samples.

## 4.4 Discussion

This section presents an analysis of the theoretical and empirical results provided above, taking note of how each result contributes to support our claims.

### 4.4.1 Performance

Because dAD involves the transmission of full activations and deltas to all sites, the gradients computed by this method exactly matches those which would be computed in the pooled case, or in distributed SGD. Thus, dAD is well-suited to applications where the exact gradients are required, and its bandwidth improvements over vanilla dSGD make it the favorable choice in such cases.

In many applications, however, low-rank approximations may be sufficient, and methods like rank-dAD may be applied. For different initial ranks, rank-dAD performs on par with PowerSGD on MNIST, and often better on the UEA datasets (see figure 4.2). We attribute our improved performance over PowerSGD to a stronger robustness of our low-rank approximation of the gradient in the $L_2$ sense. For the application to the transformer, rank-dAD sees a small performance hit when compared to the pooled case, which we believe is attribute to layer norms being computed locally for the distributed transformers.

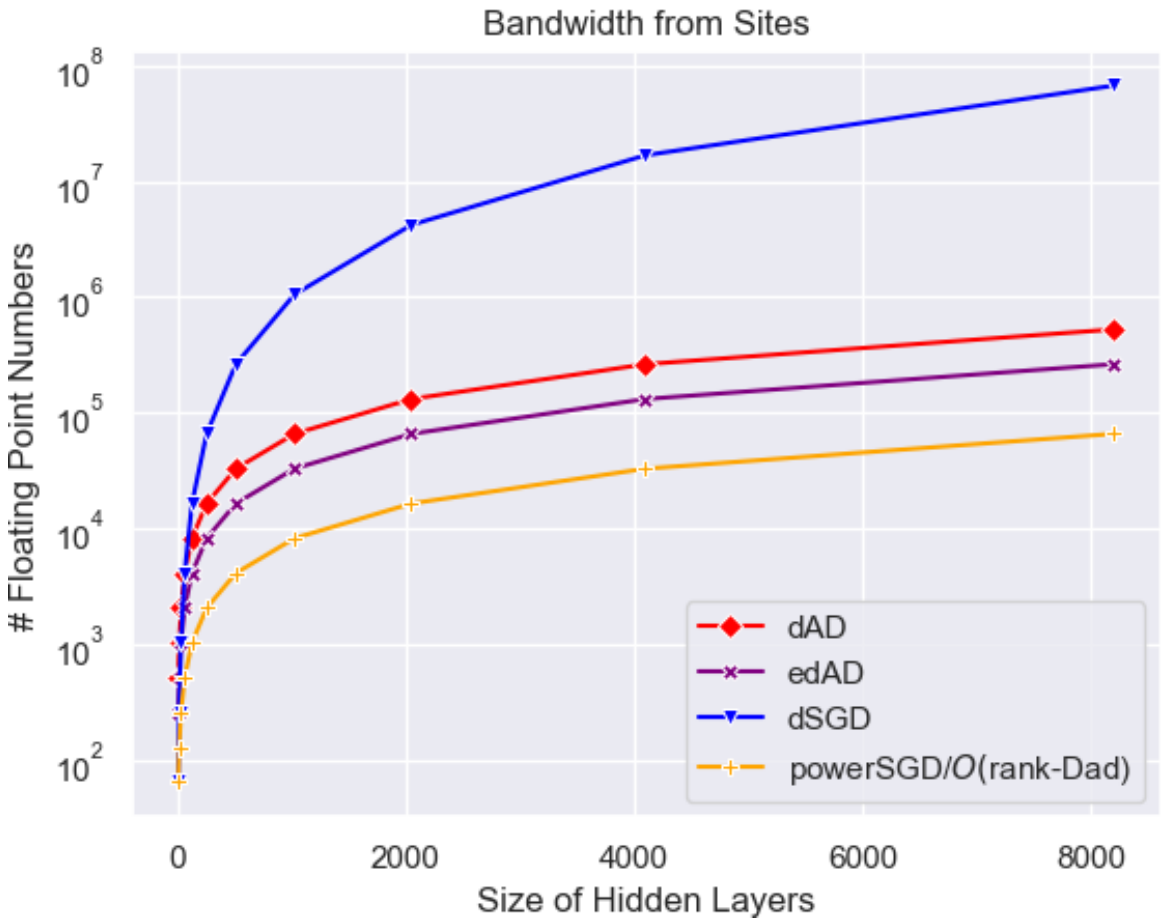



Figure 4.7: Figure reprinted with permission. Assuming a fixed batch-size of 32, and number of sites as 2, we show the bandwidth benefits of dAD, edAD, and rank-dAD. The blue curve shows the bandwidth sent during dSGD of $\Theta(h_i \times h_{i+1})$, which grows roughly quadratically with the size of the parameters. The red and purple curves show the bandwidth of dAD ($\Theta(N(h_i + h_{i+1}))$) and edAD ($\Theta(N(h_i))$) respectively. Finally the yellow curve shows the bandwidth passed during power-sgd, which serves as an upper-limit on the bandwidth of rank-dad

### 4.4.2 Complexity

*Distributed Auto-Differentiation*

Standard Distributed Auto-Differentiation is quite naïve in terms of its aggregation strategy; however, the sending of auto-differentiation statistics rather than full gradients does see a reduction in bandwidth over dSGD for most classes of networks.

Standard Distributed Auto-Differentiation is very similar in terms of computational complexity to local auto-differentiation, since the exact statistics computed during a normal forward or backwards pass are used for our distributed algorithm. The only difference is that at the local sites, we are required to compute the gradient in equation 4.4 with matrices of shared dimension $SN$, rather than $N$. So in the worst case, the complexity of dAD will be $O(h_i SN h_{i+1})$, rather than just $O(h_i N h_{i+1})$. In sum, with standard dAD, each site will be required to do slightly more work when more sites which are involved in training.

In terms of communication, we do see that for most classes of networks this naïve approach to aggregation does still give us an immediate reduction in communication compared to standard distributed SGD. Indeed, as discussed above, the communication at a given layer for dSGD is quadratic in the layer sizes with total communication $O(h_i \times h_{i+1})$. As long as the effective batch size across the network (i.e., $SN$) is significantly less than the individual number of neurons in a given layer, the communication we have in the worst case of $SN(h_i + h_{i+1})$ is an improvement. Again, however, this improvement is decreased when more sites are added to the distributed network, or when the batch size is increased significantly.

*Rank Distributed Auto-Differentiation*

Rank-dAD implements structured power-iterations as described in algorithm 12 in order to reduce local deltas and activations into rank-$r$ $B$ and $C$ matrices, which can be multiplied together to retrieve the gradient. The power-iterations themselves require a complexity of

$O(N(h_i \times h_{i+1}))$, supposing that the number of iterations is much less than the size of the hidden layers.

Rank-dAD thus represents a trade-off of local computational complexity in exchange for reduced bandwidth. Since the rank of $B$ and $C$ will have a rank of *at most* $r$, the bandwidth at each batch at each layer will be $O(r(h_i + h_{i+1}))$. This is an improvement over the complexity of PowerSGD which has complexity of $\Theta(r(h_i + h_{i+1}))$. We can empirically support this improvement in bandwidth by pointing to the effective rank results in 4.4 and 4.3. In general, we observe that the effective rank at all layers is lower than the initially chosen maximum rank, and during training, that effective rank tends to decrease, further decreasing bandwidth during training.

### 4.4.3 Privacy Considerations

In its current form, both dAD involves the sharing of input activations across the distributed network with their batch dimension preserved. If all layers in the network are being shared, this may create privacy issues because individual activations could be linked with particular samples. The exactness of the gradient computation and reduced bandwidth in dAD and edAD thus comes at a privacy risk without the addition of further measures such as differential privacy,

Rank-dAD addresses these concerns, however, since the batch dimension is reduced as part of the local power iterations, and so inofmration about individual samples is not preserved when the statistics are sent. Rank-dAD by itself is thus already plausibly private.

### 4.4.4 Limitations and Future Work

The key insight of our work is that auto-differentiation provides a unique and useful perspective into distributed deep learning, which can be utilized both to improve the bandwidth of distributed algorithms and to examine the dynamics involved in learning. 4.7 puts potential bandwidths saving in perspective.

**Extension to General-Purpose AD** AD has applications to machine learning beyond deep learning, such as generic gradient-based optimization with the Hessian [191, 192], or Bayesian posterior inference in MCMC [193]. Further work is required to look into auto-differentiation as a potentially distributable process in and of itself; however, such work would open up a much wider domain of machine learning to the insights provided here.

**The Problem with Convolutions** In their current form, dAD and rank-dAD share input activations for the given layers in the network. For feed-forward and recurrent networks (as well as transformers), the bandwidth improvements provided over dSGD are obvious. Convolutional layers, however, present a bandwidth problem for these methods, because the size of the resulting output activations from a convolutional layer tend to be much larger in size than the number of parameters within that layer. Thus, further work is needed in examining AD applied to convolutional layers to see if bandwidth reduction is available without the addition of heuristics.

One potential solution to this problem for CNN-based-classifiers would apply PowerSGD to the convolutional layer, and our method to all fully-connected layers found elsewhere in the network. Thus, we would still see a bandwidth reduction in the fully-connected layers without taking a bandwidth hit on the convolutions.

For networks involving only convolutions, however, the problem remains. Since our method in its current form does provide significant benefits for Feed Forward and Recurrent networks, we leave the study of distributed-auto-differentiation for convolutional layers as future work.

**Backflow to Private Encoders** Because our method focuses on the underlying statistics used for computing gradients in AD,

**Effective Gradient Rank for Introspection and DNN Dynamics** Although our initial approach was to use the unique structure provided to us by AD to compute an accurate

low-rank approximation, the apparent dynamics this approach reveals beg for further empirical and theoretical analysis. The Singular Value Decomposition has been used to study the dynamics of training in networks with linear activations [81], and it is possible that distributed models may be provided with such an analysis for free when using our method. By opening the black box of AD, we may get a peek into the black box of deep learning for free.

## 4.5 Conclusions

In this work, we took a step back from standard gradient-based methods for distributed deep learning, and presented a novel algorithm for distributed auto-differentiation (dAD). The insight that the intermediate outer product factors gathered by AD can be transferred instead of the full gradient provides a significant bandwidth reduction over full-gradient methods like dSGD without a loss in performance. Furthermore, we are able to show that the structure of AD can be further capitalized on to reduce bandwidth again by half, since for standard backpropagation, the global delta values can be back-propagated through the network as long as the activations are still shared. Finally, we push dAD even further by exploiting the explicit outer-product structure to obtain low rank approximations for the gradient in terms of low-rank versions of the intermediate statistics. With this, rank-dAD provides an intriguing method for adaptively reducing bandwidth where the chosen rank is an *upper*-limit on communication. We are also able to analyze how the effective rank of the gradient changes during training, and thus obtain introspective information about the learning dynamics. It has been our goal to illustrate that auto-differentiation provides a rich landscape for further exploration into distributed deep learning. The reduction in bandwidth, intuitive algorithms, and competitive performance we have demonstrated here represent the first of potentially many benefits available to distributed machine learning practice and theory.

# CHAPTER 5
# LOW-RANK LEARNING BY DESIGN: THE ROLE OF NETWORK ARCHITECTURE AND ACTIVATION LINEARITY IN GRADIENT RANK COLLAPSE

## 5.1 Introduction

Deep Neural Networks (DNNs) continue to achieve state-of-the-art performance on a number of complex data sets for a diverse array of tasks; however, modern DNN architectures are notoriously complex, with millions of parameters, nonlinear interactions, and dozens of architectural choices and hyper-parameters which can all significantly affect model performance. Internal complexity and a lack of thorough theoretical groundwork has given DNNs a reputation as "black box" models, where architectures may excel or fail on a given problem with relatively little indication how their structure facilitated that performance. Engineering a neural network that works well on a particular problem can often take the form of fairly arbitrary and exhaustive model tweaking, and even in cases where researchers systematically perturb particular settings, the primary explanations of performance come down to observing minor changes in performance evaluation metrics such as loss, accuracy/precision, dice-scores or other related metrics. In this work, we examine a particular emergent phenomenon in deep neural networks—the collapse of gradient rank during training; however, we take a theory-first approach, examining how bounds on gradient rank collapse appear naturally and deterministically as a result of particular architectural choices such as bottleneck layers, level of nonlinearity in hidden activations, and parameter tying.

This work is part of a growing body of theoretical research studying the dynamics of learning in deep neural networks. Beginning from first principles, Andrew Saxe provided a foundational work on exact solutions to nonlinear dynamics which emerge in fully-

connected networks with linear activations [81], which has inspired a body of related work on simple networks with various architectural or learning setups such as parent-teacher interactions [194], online learning and overparametrization [195], and gated networks [196]. This work on theory and dynamics has also been extended to studying high-dimensional dynamics of generalization error [197], emergence of semantic representations [82], and other phenomemon which can be first characterized mathematically and observed empirically. Our work in this paper follows this tradition in the literature of beginning with mathematical principles which affect learning dynamics in simple networks, and demonstrating how these principles emerge in practical scenarios.

An additional related body of work studies the phenomemon of Neural Collapse [87, 198], in which deep classifier neural networks converge to a set of rigid geometrical constraints during the terminal phase of training, with the geometry of the output space determined exactly by the number of classes (i.e., the rank of the output space). This neural collapse phenomenon has been thoroughly studied as emerging in constrained [199] and unconstrained feature settings [88, 89, 200], with various loss functions [90, 91, 201], under transfer learning regimes [202], class imbalance scenarios [203], and exemplifying affects on the loss-landscapes [92]. This growing body of work suggests that geometric constraints during learning influence a number of desirable and undesirable deep learning behaviors in practice.

Like the works on neural collapse, we are interested in geometric constraints to learning; however, we follow the example of Saxe et al. and begin our theoretical examination with simple linear networks, showing how we can expand on simple constraints of batch size (which has been discussed previously elsewhere [204]) to constraints dependent on a number architectural features such as bottlenecked layers, parameter-tying, and level of linearity in the activation. Our work invites the study of network-wide geometric constraints, and while we do not dive into training dynamics in this work, we are able to set a stage which bounds dynamics, hopefully clearing the way for further dynamical analysis in the

style of Saxe, Tishby and others.

Finally, our work studying the affect of a particular nonlinear activation and its level of linearity stands out from the existing work on purely linear networks and neural collapse in linear classifiers. Although nonlinear activations introduce some complexity to analysis, we can draw from some previous work on analyzing the spectrum of ReLU activations [205], extending the analysis in that work to its natural consequences with Leaky-ReLU activations and even deriving explicit bounds for numerical estimation of rank which require only singular values of the underlying linear transformations.

Our derivation of upper bounds on gradient dynamics during training also has implications for distributed models which utilized low-rank decompositions for efficient communication. For example, PowerSGD [80] and distributed Auto-Differentiation [78] compute a low-rank decomposition of the gradient prior to transfer between distributed data-collection sites. Our theoretical results here can help to provide insights into how high of a rank may be needed to preserve model performance between the pooled and distributed cases, or how much information may be lost when a lower-rank decomposition is used for communication efficiency.

Our primary results in this work are theoretical; however, we perform a number of empirical verifications and demonstrations which verify our theoretical results and study the implications of our derived bounds on gradient rank for various archictural design choices. In sum, the contributions here include:

1. an upper bound on the rank of the gradient in linear networks;

2. upper bounds on the rank of the gradient in linear networks with shared parameters, such as RNNs and CNNs;

3. extension of previous work on ReLU Singular Values to study the effect of Leaky-ReLUs and the level of linearity on the upper bound of rank;

4. empirical results on numerical data which verify our bounds and implicate particular

architectural choices;

5. empirical demonstration of theoretical implications on large-scale networks for Computer Vision and Natural Language Processing;

6. natural extensions in the future work to rank dynamics during training, explicit connections to neural collapse, and implications for rank effects of other architectural phenomena such as dropout layers, batch norms, and more.

## 5.2 Theoretical Methods

### 5.2.1 Reverse-Mode Auto-Differentiation

We will define a simple neural network with depth $L$ as the operator $\Phi(\{\mathbf{W}_i\}_{i=0}^L, \{\mathbf{b}_i\}_{i=0}^L, \{\phi_i\}_{i=0}^L) : \mathbb{R}^m \to \mathbb{R}^n$. This is given in as a set of weights $\{\mathbf{W}_1, \ldots, \mathbf{W}_L\}$, bias variables $\{\mathbf{b}_1, \ldots, \mathbf{b}_L\}$, and activation functions $\{\phi_1, \ldots, \phi_L\}$, where each function $\phi_i : \mathbb{R}^n \to \mathbb{R}^n$ is an element-wise operator on a vector space.

Let $\mathbf{x} \in \mathbb{R}^m$ be the input into the network and let $\mathbf{y} \in \mathbb{R}^n$ be a set of target variables. Then we define $z_i$ as the *internal activations* and $a_i$ as the external activations at layer $i$ given with the recursive relation:

$$\mathbf{z}_0 = \mathbf{x}$$

$$\mathbf{a}_0 = \mathbf{z}_0$$

$$\mathbf{z}_i = \mathbf{W}_i \mathbf{a}_{i-1} + \mathbf{b}_i$$

$$\mathbf{a}_i = \phi_i(\mathbf{z}_i)$$

To define the sizes of the hidden layers, we have $\mathbf{W}_i \in \mathbb{R}^{h_{i-1} \times h_i}$, with $h_0 = m$ and $h_L = n$. We note then that $\mathbf{z}_i, \mathbf{a}_i$ are column vectors in $\mathbb{R}^{h_i}$.

Let $\mathcal{L}(\mathbf{y}, \mathbf{a}_L) : \mathbb{R}^n \to \mathbb{R}$ be a loss function which measures the *error* of the estimate of $\mathbf{y}$ at $\mathbf{a}_L$. The gradient update for this loss, computed for the set of weights $\mathbf{W}_i$ can be

written as the outer-product

$$\nabla_{\mathbf{W}_i} = \mathbf{a}_{i-1}\delta_i^\top$$

where $\delta_i$ is the partial derivative of the output at layer $i$ w.r.t its input. At the output layer $L$, $\delta_L$ is computed as

$$\delta_L = \frac{\partial \mathcal{L}}{\partial \mathbf{a}_L} \odot \frac{\partial \phi_L}{\partial \mathbf{z}_L}$$

and subsequent $\delta_i$ are computed as

$$\delta_i = \delta_{i+1}\mathbf{W}_{i+1} \odot \frac{\partial \phi_i}{\partial \mathbf{z}_i}.$$

These definitions are all given for $x$ as a column vector in $\mathbb{R}^m$; however, for standard batch SGD we compute these quantities over a batch of $N$ samples. If we rewrite the variables $\mathbf{x}$, $\mathbf{y}$, $\mathbf{z}_i$, $\mathbf{a}_i$ and $\delta_i$ as matrices $\mathbf{X} \in \mathbb{R}^{N\times m}$, $\mathbf{Y} \in \mathbb{R}^{N\times n}$, $\mathbf{Z}_i, \mathbf{A}_i, \mathbf{\Delta}_i \in \mathbb{R}^{N\times h_i}$. The gradient can then be computed as the matrix-product

$$\sum_k^N \mathbf{a}_{n,i-1}\delta_{n,i}^\top = \mathbf{A}_{i-1}^\top \mathbf{\Delta}_i.$$

## 5.3 Theoretical Results

### 5.3.1 Bounds on Gradients in Linear Networks

First consider the set of neural networks where $\phi_i(\mathbf{x}) = \mathbf{x}$ is the identity operator $\forall i$. These networks are called "linear networks" or equivalently "multi-layer perceptrons" (MLP), and $\mathbf{Z}_i = \mathbf{A}_i, \forall i$. For these networks, the rank of the gradients has an exact bound. Trivially, for a given gradient $\nabla_{\mathbf{W}_i}$, the rank is

$$\text{rank}(\nabla_{\mathbf{W}_i}) = \text{rank}(\mathbf{Z}_{i-1}^\top \mathbf{\Delta}_i) \leq \min\{\text{rank}(\mathbf{Z_{i-1}}), \text{rank}(\mathbf{\Delta}_i)\}.$$

Since in linear networks we can easily compute $\mathbf{Z}_i = \mathbf{X}\prod_{j=1}^i \mathbf{W}_j$, we use a similar

rule as for the gradient to compute the bound

$$\text{rank}(\mathbf{Z}_i) \leq \min\{\text{rank}(\mathbf{X}), \text{rank}(\mathbf{W}_1), \ldots, \text{rank}(\mathbf{W}_i)\}.$$

For the adjoint variable $\Delta_i$ we can use the fact that $\frac{\partial \phi}{\partial \mathbf{Z}_i} = \mathbf{1}$ where $\mathbf{1}$ is a matrix of ones in $\mathbb{R}^{N \times h_i}$ to derive a bound on the adjoint as

$$\text{rank}(\boldsymbol{\Delta}_i) \leq \min\{\text{rank}(\mathbf{W}_i), \text{rank}(\mathbf{W}_{i+1}), \ldots, \text{rank}(\mathbf{W})_L, \text{rank}(\frac{\partial \mathcal{L}}{\partial \mathbf{Z}_L})\}.$$

Therefore, the bound for the gradient rank is

$$\begin{aligned} \text{rank}(\nabla_{\mathbf{W}_i}) &\leq \min\{\text{rank}(\mathbf{Z_{i-1}}), \text{rank}(\boldsymbol{\Delta}_i)\} \\ &\leq \min\{\text{rank}(\mathbf{X}), \text{rank}(\mathbf{W}_i), \text{rank}(\mathbf{W}_{i+1}), \ldots, \text{rank}(\mathbf{W})_L, \text{rank}(\frac{\partial \mathcal{L}}{\partial \mathbf{Z}_L})\} \end{aligned} \tag{5.1}$$

### 5.3.2 Bounds on Gradients in Linear Networks with Parameter Tying

We will begin our analysis with recurrent neural networks and Back-Propagation through Time (BPTT) [206].

*Recurrent Layers*

Let $\mathcal{X} \in \mathbb{R}^{N \times n \times T}$ be the $N$ samples of an $n$-dimensional variable over a sequence of length $T$ (over time, for example). We will set an initial hidden state for this layer as $\mathbf{H}_{i,0} \in \mathbb{R}^{N \times h_i}$.

Let $f_i : \mathbb{R}^{N \times h_{i-1} \times T} \to \mathbb{R}^{N \times h_i \times T}$ be the function given by a linear layer with a set of input weights $\mathbf{U} \in \mathbb{R}^{h_{i-1} \times h_i}$ and a set of hidden weights $\mathbf{V} \in \mathbb{R}^{h_i \times h_i}$. The output of this layer is computed as the tensor

$$f_i(\mathcal{X}) = \{\mathbf{H}_{t,i}\}_{t=1}^T = \{\phi_i(\mathbf{Z}_{t,i})\}_{t=1}^T = \{\phi_i(\mathbf{X}_t \mathbf{U}_i + \mathbf{H}_{t-1,i} \mathbf{V}_i)\}_{t=1}^T.$$

Supposing the error $i$ feeds into another recurrent layer, the error on the output (which is used for the gradients of both $\mathbf{U}$ and $\mathbf{V}$) is thus computed as the tensor

$$\mathcal{D}_i = \{\mathbf{\Delta}_{t,i}\}_{t=1}^{T} = \left\{ (\Delta_{t,i+1}\mathbf{U}_{i+1} + \mathbf{\Delta}_{t+1,i}\mathbf{V}_i) \odot \frac{\partial \phi_i}{\partial \mathbf{Z}_{t,i}} \right\}_{t=1}^{T}$$

where we have $\mathbf{\Delta}_{T+1,i} = \mathbf{0}$ for convenience of notation. In the case where the next layer in the network is not recurrent (for example if it is a linear layer receiving flattened output from the RNN), we can set $\mathbf{\Delta}_{t,i+1}$ to be the elements of $\mathbf{\Delta}_{i+1}$ which correspond to each timepoint $t$.

The gradients for $\mathbf{U}_i$ and $\mathbf{V}_i$ are then computed as the sum over the products of each element in the sequence

$$\nabla_{\mathbf{U}_i} = \sum_{t=1}^{T} \mathbf{X}_{t,i}^{\top} \mathbf{\Delta}_{t,i} \qquad\qquad \nabla_{\mathbf{V}_i} = \sum_{t=0}^{T-1} \mathbf{H}_{t,i}^{\top} \mathbf{\Delta}_{t,i}$$

Fully linear RNNs are not typically implemented in practice; however, for the purpose of demonstrating how parameter-tying can improve with parameter-tying, the analysis may still prove helpful. The first thing to notice is that even for as small as $T = 2$, we reach the potential for full-rank gradients quite quickly. Even in the degenerate case when the batch size is $N = 1$, $\nabla_{\mathbf{U}_i}$ and $\nabla_{\mathbf{V}_i}$ may become rank $T$. Thus, the analysis of rank no longer depends much on the architecture beyond the number of timepoints chosen, and parameter-tying can affect rank-collapse that emerges from low-rank product bottlenecks in other layers. Rather, it will become of interest to look at correspondences between input such as temporal correlation in order to provide a clearer picture. We will leave this for future work.

*Convolutional Layers*

Our derivation of bounds on the gradient rank of convolutional layers will follow much of what was derived for RNNs. The primary difference will appear in the number of steps

over which gradients are accumulated, and their relationship to image size, stride, kernel size, padding, etc.

Suppose we are working on a convolution of dimension $m$. If we let $N$ denote the size of a given batch, and let $C_{in}$, $C_{out}$ be the input/output channels, and let $\mathbf{w}_{in} \in \mathbb{N}_+^m$ be a list of image dimensions (such as Height and Width for a 2d image). Then the input image to the convolution is denoted as a tensor $\mathcal{X} \in \mathbb{R}^{N \times C_{in} \times w_{in,1} \times w_{in,2} \times \cdots \times w_{in,m}}$.

Suppose we are performing a convolution with a kernel sizes $\mathbf{k} \in \mathbb{N}_+^m$, dilations $\mathbf{d} \in \mathbb{N}_+^m$, padding $\mathbf{p} \in \mathbb{N}_+^m$, and stride $\mathbf{s} \in \mathbb{N}_+^m$. The weight tensor for this convolution is denoted as $\mathcal{W} \in \mathbb{R}^{C_{out} \times C_{in} \times \mathbf{k}_1 \times \mathbf{k}_2 \times \cdots \times \mathbf{k}_m}$. The output of this convolution is thus computed as the tensor $\mathcal{Y} \in \mathbb{R}^{N \times C_{out} \times w_{out,1} \times w_{out,2} \times \cdots \times w_{out,m}}$:

$$\mathcal{Y}_{i,j,\ldots} = \sum_{k=1}^{C_{in}} \mathcal{W}_{i,k,\ldots} \star \mathcal{X}_{i,k,\ldots}$$

where $\star$ is the $m$-dimensional cross-correlation operator.

Following reverse-mode auto-differentiation for $m$-dimensional convolutions, the gradient of $\mathcal{W}$ is computed as a convolution between the input $\mathcal{X}$ and the adjoint from the backward pass $\mathbf{\Delta}$:

$$\nabla_{\mathcal{W}} = \sum_{i=1}^{N} \{\mathcal{X} \star \mathbf{\Delta}\}_i$$

It is clear that with linear activations, even if $\mathcal{X}$ and $\mathbf{\Delta}$ are low-rank, the rank of this gradient is accumulated over the cross-correlation operator $\star$ (as well as over $N$, in a similar way to for linear layers). It becomes apparent that like for RNNs, we are accumulating over the sequence of $\prod_{i=1}^{d} w_{out,i}$ many patches, where $w_{out,i}$ is the size of the output in the $i$th dimension.

For convolutional layers, this can be explicitly computed as

$$w_{out,i} = \left\lfloor \frac{w_{in,i} + 2p_i - d_i \times (k_i - 1) - 1}{s_i} + 1 \right\rfloor$$

if we let $\mathcal{B}$ be the linear bound computed in section 2.3, then the bound on the rank of the gradient is thus

$$\mathsf{rank}(\nabla_{\mathcal{W}_i}) \leq \mathcal{B} \prod_{i=1}^{m} \left\lfloor \frac{w_{in,i} + 2p_i - d_i \times (k_i - 1) - 1}{s_i} + 1 \right\rfloor \tag{5.2}$$

Intuitively, this means the bound will shrink as the input image size and padding shrinks, and will shrink as the stride, dilation and kernel size increase.

### 5.3.3 Bounds on Gradients in Leaky-ReLU Networks

The bounds we provide on gradient rank in networks with purely linear activations builds off intuitive principles from linear algebra; however, the introduction of nonlinear activations confuses these notions. For a general nonlinear operator $\phi$, the notion of singular values which we obtain from linear algebra does not hold. Even though we can compute the SVD of the matrix $\phi_\alpha(\mathbf{Z})$ for a given layer with internal activations $\mathbf{Z}$, little can be initially said about the relationship of the this decomposition to the linear transformations which generated $\mathbf{Z}$. Thus, although we can still empirically compute the rank of the resulting matrix from the nonlinear transformation, it is not initially clear how rank-deficiency will propagate through a network as it will in the fully linear case. In this section, we show how Leaky-ReLU activations with different levels of nonlinearity affect numerical estimates of rank. In supplementary material, we also explore theoretical connections to previous work on so-called "ReLU Singular Values", which also follow our derived bounds.

*Numerical effect of Leaky-ReLUs on Rank*

In this section, we analyze the numerical effect of Leaky-ReLU nonlinearities on the singular values of the internal activations $\mathbf{Z}_i = \mathbf{A}_{i-1}\mathbf{W}_i$. Our tactic will be to observe how the slope $\alpha$ can push the singular values of $\phi_\alpha(\mathbf{Z}_i)$ above a numerical threshold used for estimating rank. As a choice of threshold, we use $\epsilon \cdot \max_i \sigma_i$, where $\epsilon$ is the machine-epsilon for floating-point calculations on a computer. This threshold is utilized in most modern libraries which can perform rank estimation, including PyTorch [179] and Tensorflow [207]. We then say that a singular value $\sigma_k$ does not contribute to our estimation of rank if

$$\sigma_k < \epsilon \cdot \max_i \sigma_i \tag{5.3}$$

Let $\mathbf{D}_\alpha(\mathbf{Z}) \in \mathbb{R}^{h \times h}$ be the matrix with entries corresponding to the linear coefficients from the Leaky-ReLU activation of $\mathbf{Z}_i = \mathbf{X}\mathbf{W}_i$ for $\mathbf{W} \in \mathbb{R}^{h-1 \times h}$. Leaky-ReLU activations can be written as the Hadamard product

$$\phi_\alpha(\mathbf{Z}_i) = \mathbf{D}_\alpha \odot \mathbf{Z}_i \tag{5.4}$$

From Zhan [208], we have the inequality for the singular values of the Hadamard product:

$$\sum_{i=1}^{k} \sigma_i(\mathbf{D}_\alpha \odot \mathbf{Z}_i) < \sum_{i=1}^{k} \min\{c_i(\mathbf{D}_\alpha), r_i(\mathbf{D}_\alpha)\}\sigma_i(\mathbf{Z}_i) \tag{5.5}$$

where $c_1(\mathbf{D}_\alpha) \geq c_2(\mathbf{D}_\alpha) \geq \cdots \geq c_h(\mathbf{D}_\alpha)$ are the 2-norm of the columns sorted in decreasing order, and $r_i(\mathbf{D}_\alpha)$ are the 2-norm also in decreasing order.

We can say that $\mathbf{D}_\alpha \odot \mathbf{Z}$ will remain numerically rank-deficient up to rank $k$ when

$$\min\{c_k(\mathbf{D}_\alpha), r_k(\mathbf{D}_\alpha)\}\sigma_k(\mathbf{Z}) \leq \epsilon\sigma_1(\mathbf{D}_\alpha \odot \mathbf{Z}) \tag{5.6}$$

We then have two cases: 1) $\sigma_1(\mathbf{D}_\alpha \odot \mathbf{Z}) \leq \sigma_1(\mathbf{Z})$ or 2) $\sigma_1(\mathbf{D}_\alpha \odot \mathbf{Z}) > \sigma_1(\mathbf{Z})$. If case 1) holds, the Hadamard product will remain rank deficient if the euclidean length of the corresponding column or row of $\mathbf{D}_\alpha$ is less than or equal to $\epsilon\sigma_1(\mathbf{Z})/\sigma_k(\mathbf{Z})$, and will increase otherwise.

Case 2 is best analyzed by a choice of $\mathbf{Z}$ which saturates $\sigma_1$ at the bound in (5.7), since it cannot grow anywhere beyond that bound. If we have $\sigma_1(\mathbf{D}_\alpha \odot \mathbf{Z}) = \min\{c_1(\mathbf{D}_\alpha), r_1(\mathbf{D}_\alpha)\}\sigma_1(\mathbf{Z})$, $\sigma_k$ will always contribute numerically to the rank when the corresponding column/row norm at $k$ equals

$$\epsilon \min\{c_1(\mathbf{D}_\alpha), r_1(\mathbf{D}_\alpha)\}\sigma_1(\mathbf{Z})/\sigma_k(\mathbf{Z}). \tag{5.7}$$

Because we are dealing with Leaky-ReLU activations in particular, the 2-norm of $c_i(\mathbf{D}_\alpha)$ and $r_i(\mathbf{D}_\alpha)$ take on a particular closed form. Indeed, we have

$$c_i(\mathbf{D}_\alpha) = \sqrt{N_-\alpha^2 + N_+} \qquad\qquad r_i(\mathbf{D}_\alpha) = \sqrt{M_-\alpha^2 + M_+}$$

where $N_-$ is the number of rows in column $i$ which fall into the negative domain of the Leaky-ReLU activation, and $N_+$ is the number of rows which fall into the positive domain. Similarly, $M_-$ and $M_+$ count the number of columns in the negative and positive domains in row $i$. In other words, we can exactly determine the threshold at which a singular value will fail to contribute to numerical estimation of rank if we know the non-negative coefficient and can count the number of samples which reside in particular regions of the activation function's domain.

We can use a similar kind of analysis as in section 3.3.1 to derive a bound on the rank for the partial derivative of the Leaky-ReLU activation on the output. As we mentioned in that section, the analysis can extend trivially to any piecewise linear function, and indeed the derivative of a Leaky-ReLU is piecewise linear.

Let $\mathcal{D}_\alpha(\mathbf{Z}) \in \mathbb{R}^{h\times h}$ be the matrix with entries corresponding to the linear coefficients from the Leaky-Relu activation applied to internal activation $\mathbf{Z}_i = \mathbf{X}\mathbf{W}_i$. The partial

derivative w.r.t to the output can be written as the hadamard product

$$\phi'_\alpha(\mathbf{Z}_i) = \mathbf{D}_\alpha \odot \mathbf{1}^{h\times h} \tag{5.8}$$

where $\mathbf{1}^{h\times h}$ is an $h \times h$ matrix of ones.

From our derivation in section 3.3.1, we can say that $\mathbf{D}_\alpha \odot \mathbf{1}^{h\times h}$ will remain numerically rank-deficient according to the bound

$$\min\{c_k(\mathbf{D}_\alpha), r_k(\mathbf{D}_\alpha)\} \leq \epsilon \min\{c_1(\mathbf{D}_\alpha), r_1(\mathbf{D}_\alpha)\}\sigma_1(\mathbf{Z}_i)/\sigma_k(\mathbf{Z}_i) \tag{5.9}$$

We note that $\mathbf{1}^{h\times h}$ is rank-1 with $\sigma_1 = h$ and $\sigma_k \leq \epsilon h$. If we suppose this bound is saturated, the bound depends primarily on the quantity $\min\{c_1(\mathbf{D}_\alpha), r_1(\mathbf{D}_\alpha)\}$, and will loosen somewhat as the precision of $\sigma_k$ shrinks it toward 0.

*Connection to ReLU-Singular Values*

One initial theoretical answer to analyse particular nonlinearities involves extending the notion of singular values to certain nonlinear functions which are locally linear, such as ReLU or Leaky-ReLU activations [205]. For the nonlinear operator $\phi_\alpha(\mathbf{Z}) = \mathrm{LeakyReLU}_\alpha(\mathbf{Z})$, where $\alpha \in [0, 1]$ controls the slope of the activation when $\mathbf{Z} < 0$. We note that when $\alpha = 0$, this is equivalent to a ReLU activation and when $\alpha = 1$ it is equivalent to a Linear activation.

Following the work in [205], for a matrix $\mathbf{Z} \in \mathbb{R}^{m\times n}$ the $k$th "Leaky-ReLU Singular Value" is defined for the operator $\phi_\alpha(\mathbf{Z})$ as

$$s_k(\phi_\alpha(\mathbf{Z})) = \min_{\mathrm{rank}\,\mathbf{L}\leq k} \max_{x\in\mathcal{B}} \|\mathrm{LeakyReLU}_\alpha(\mathbf{Z}x) - \mathrm{LeakyReLU}_\alpha(\mathbf{L}x)\|. \tag{5.10}$$

In [205], the extension of the notion of ReLU singular values to Leaky-ReLUs carries

naturally; however, for completeness, we have included Leaky-ReLU-specific versions of each of the proofs from that work in the supplement. Among these results, we have the following lemma:

*Lemma:* Let $\phi_\alpha(\mathbf{Z}) = \text{LeakyReLU}_\alpha(\mathbf{Z})$ for $\mathbf{Z} \in \mathbb{R}^{n \times m}$, then

$$s_k(\phi_\alpha(\mathbf{Z})) \leq \sigma_k(\mathbf{Z}) \tag{5.11}$$

In other words, the Leaky-ReLU singular values will be bounded above by the singular values of the underlying linear transformation $\mathbf{Z}$. It follows then that as we increase $\alpha$ along the interval [0, 1], the analogous notion of " rank (the number of non-zero values of $s_k$) will converge converge upward to the linear rank of $\mathbf{Z}$, and our boundaries will still hold.

### 5.4 Empirical Methods

We perform two broad classes of experiment: simple verification, and large-scale demonstration. In the first class of experiments, we show how the bounds derived in our theoretical results appear in simple, numerical experiments, where it is easy to verify how particular architectural choices and level of nonlinearity affect the bound of the gradient rank. In the second class of experiments, we perform larger-scale experiments and demonstrate how our derived bounds can also affect these models in practice. These latter class of experiments utilize models which include modules such as Drop-Out, BatchNorms, and LayerNorms. We do not explore these additional modules theoretically in this work, even though they may have an influence on gradient rank [209]; however, we believe this allows for a productive direction for future theoretical and empirical work.

In both styles of experiment, the primary architectural elements we demonstrate when possible are: 1) bottleneck layers, i.e., layers within a network which have a much smaller number of neurons than their input and output spaces, 2) length of sequences in parameter

Table 5.1: The Data Sets evaluated as part of our empirical verification. For space, only the Gaussian and Sinusoid data sets are included in the main body of text, and the remaining data sets are included in the supplementary material.

| **Dataset** | **# Samples** | **Subspace** | **Type** |
|---|---|---|---|
| Gaussian | 16384 | $\mathbb{R}^{N \times m}$ | Numeric |
| Sinusoids | 16384 | $\mathbb{R}^{N \times m \times T}$ | Numeric |
| MNIST | $6 \times 10^4$ | $\mathbb{R}^{N \times H \times W}$ | Image |
| CIFAR10 | $6 \times 10^4$ | $\mathbb{R}^{N \times H \times W}$ | Image |
| TinyImageNet | $10^5$ | $\mathbb{R}^{N \times 3 \times H \times W}$ | Image |
| WikiText | $> 10^8$ | $\mathbb{R}^{N \times \lvert V \rvert \times T}$ | Text |
| Multi30k | $> 3 \times 10^5$ | $\mathbb{R}^{N \times \lvert V \rvert \times T}$ | Text |

Table 5.2: The models evaluated as part of our empirical verification. For space and demonstrating key features, we include results from the Fully Connected Network, Elman RNN, and ResNet16 in the main text. Additional model architectures are included in the supplement.

| **Model** | **Datasets** |
|---|---|
| MLP | Gaussian |
| Elman RNN | Sinusoids |
| BERT | WikiText |
| XLM | Multi30k |
| ResNet16 | MNIST, CIFAR10, ImageNet |
| VGG11 | MNIST, CIFAR10, ImageNet |

tying, 3) low-rank input/output spaces, 4) level of non-linearity in hidden activations.

For the sake of numerical verification, we implement auto-encoding networks on two numerical datasets. For our first data set, we generate an $m$-dimensional gaussian $X \in \mathbb{R}^{N \times m}$ as $\mathbf{x} \sim \mathcal{N}(\mu_i, \Sigma_i)$ We then use a fully-connected network as an auto-encoder to reconstruct the random samples.

Our second kind of numerical data is generated by sine waves of the form $\mathbf{x}_{i,j} = a_{i,j} \sin(b_{i,j} t) + c_{i,j}$, where we sample the parameters $a_{i,j}, b_{i,j}, c_{i,j}$ for a particular sample $i$ and dimension $j$ independently from their own univariate Gaussian distributions $a_{i,j} \sim \mathcal{N}(\mu_a, \sigma_a), b_{i,j} \sim \mathcal{N}(\mu_b, \sigma_b), c_{i,j} \sim \mathcal{N}(\mu_c, \sigma_c)$. We choose $t$ to be $T$-many points in the interval $[-2\pi, 2\pi]$. We then used an RNN with Elman Cells, GRUs and LSTMs to reconstruct the input sequence, and demonstrate how four architectural principles affect our derived bound gradient rank.

For our larger-scale experiments, we choose two popular data sets from computer vision and natural language processing. We choose Cifar10 [210] and a Tiny-ImageNet (a Subset of ImageNet [211]) for computer vision, and we choose WikiText [212] for natural language processing. Because our empirical analysis requires repeated singular value decompositions and multiple architectural tweaks, we avoid overly long experiment runtimes by using relatively smaller-sized versions of popular network architectures which can fit alongside batches on single GPUs. In terms of computer-vision architectures, we utilize ResNet16 [213] and VGG11 [214], and for natural language processing, we utilize the BERT [215] and XLM [216] transformers for Language Modeling and Machine Translation respectively. Again, since we are interested primarily in rank as initially bounded by architecture, we do not use pretrained weights, and we only train models for a maximum of 100 epochs.

For both numerical verification and large-scale experiments, all models were trained using $k = 5$-fold cross validation, with 4 uniquely initialized models on each fold for a total of 20 models per result. The rank metrics in the following section are thus accumulated


Figure 5.1: For a 3-layer Linear FC network, we plot the mean rank of gradients, activation, and deltas change with respect to the size of a neuron bottleneck in the middle layer. The axis axis provides the name of the module, with depth increasing from right to left. In each panel, green, blue and orange bars represent the estimated rank of gradients, activations and deltas respectively. Black vertical lines on a bar indicate the standard error in the mean estimated rank across folds and model seeds.

over these 20 runs.

## 5.5 Empirical Results

### 5.5.1 Numerical Verification

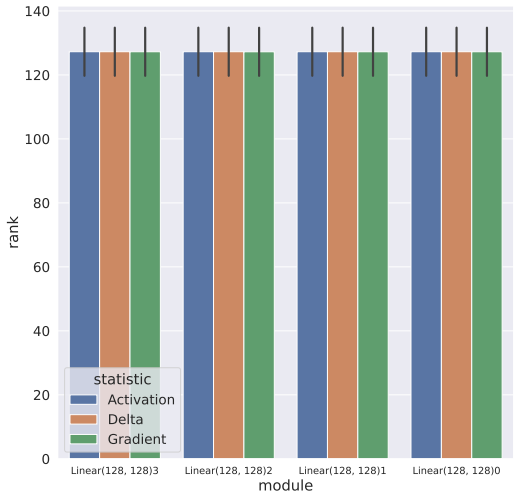
***Hypothesis 1:*** *Bottleneck layers reduce gradient rank throughout linear networks.* The bound computed in (5.1) suggests that reducing the size of a particular layer in a fully-connected network will reduce gradients throughout that network, regardless of the size of the input activations or adjoints computed at those layers. In Figure 5.1, we provide a demonstration of this phenomenon in a simple fully-connected network used for reconstructing gaussian mixture variables. In Figure 5.1 (left), we provide the numerical estimate of the gradient rank at each in a 3-layer network with each layer having the same dimension as the input ($d = 128$). In Figure 5.1 (right), we provide the same rank estimates withe the middle layer being adjusted to contain only 16 neurons. We clearly see that our bound in (5.1) holds, and we can see that the two layers preceding the bottleneck have their rank bounded by the adjoint, and the following layers are bounded by activation rank.

***Hypothesis S1***: *Low-rank input/output spaces systematically bound gradient rank.* The remaining terms in the bound in (5.2) show that gradient rank is bound by the rank of

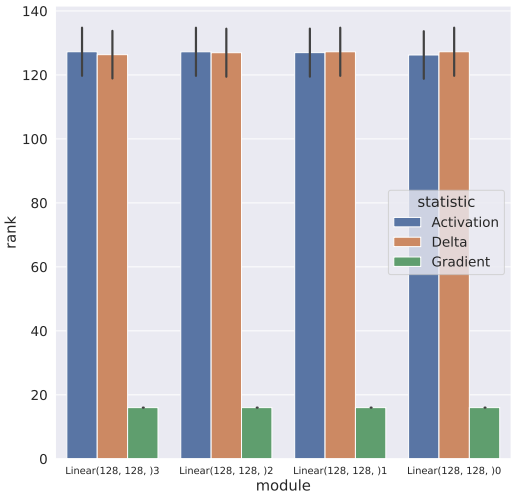
the inputs and the rank partial derivative of the loss function (i.e., the rank of the output space). In Figure 5.2, we show the computed gradient rank on a fully-connected network with three layers of 128 neurons each, trained to reconstruct a full-rank (128-dim) gaussian input (panel 5.2a) in contrast to a model trained to reconstruct a low-rank (16-dim) gaussian embedded in a higher-dimensional (128-dim) space (panel 5.2b). Indeed, in panel 5.2c we see that any linear model which receives the low-rank embedded input has gradients which are bounded entirely by the rank of the input space.

***Hypothesis 2:*** *Parameter-Sharing such as in BPTT restores gradient rank according to the number of points in the accumulated sequence* In §5.3.2 we discussed how parameter-tying restores gradient rank in models which accumulate gradients over sequence, such as RNNs using BPTT (§5.3.2) or CNNs (§5.3.2) accumulating over an image. Indeed, the number of points over which back-propagation is performed will affect how much of the gradient rank is restored in the presence of a bottleneck. In Figure 5.3, we demonstrate the gradient rank in an 3-layer Elman-Cell RNN [217] trained to reconstruct high-dimensional, sinusoidal signals. We introduce a severe bottleneck in the second RNN, constraining its hidden units to 2, with the other RNNs having 128 hidden units each. We demonstrate how the introduced gradient bottleneck is ameliorated in the adjacent layers according to the number of timepoints included in truncated BPTT over the sequence. With a maximum of 50 timepoints, the bottleneck can at most be restored to a rank of 100.

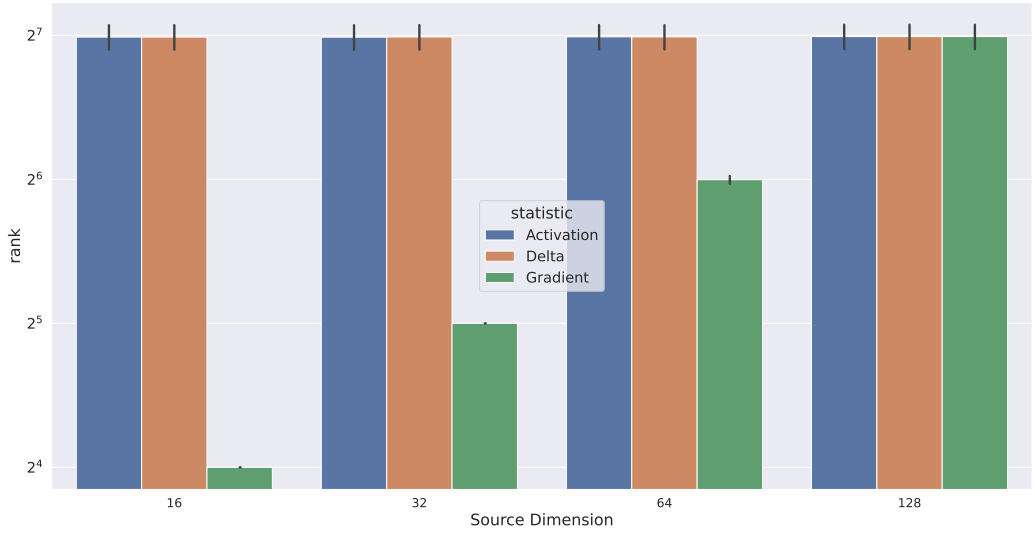
***Hypothesis 3:****Using the derivation in §5.3.3, we can compute the bound over which an estimated singular value will contribute to the rank, without computing the eigenvalues themselves* One of the primary empirical upshots of the theoretical work in §5.3.3 is that using only the singular values of the underlying linearity and a count of how samples contribute to particular areas of the domain of our nonlinearity, we can compute the bound over which singular values will cease to contribute to the estimated rank. In Figure 5.4 we construct a low-rank random variable by computing the matrix product of two low-rank gaussian variables. We then compute the estimated eigenvalues after applying a Leaky-




(a) Full-dimensional input




(b) Input of dimension 16 embedded




(c) Gradients, Deltas, Activations for different dimension input

Figure 5.2: Low-Dimensional Input

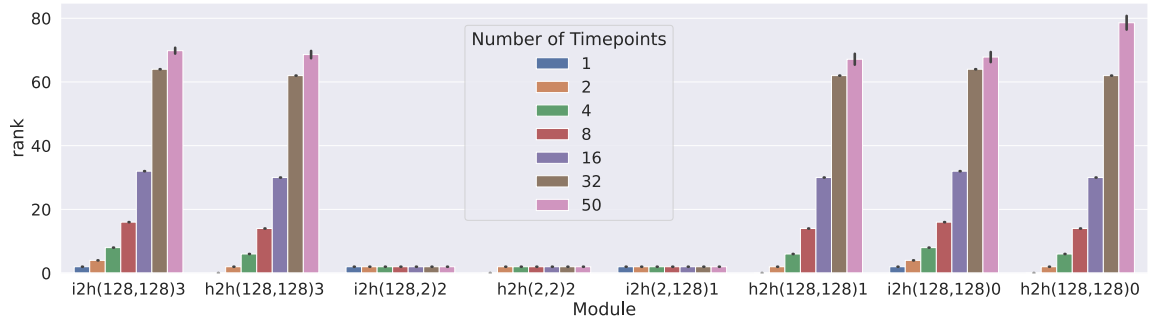



Figure 5.3: For a 3-layer Elman-Cell RNN, we show how mean rank of gradients, activation, and deltas change with respect to the number of timepoints used in truncated BPTT. The x axis groups particular modules, with depth increasing from right to left. Each colored bar shows the mean estimated rank over multiple seeds and folds using a different sequence length for truncated BPTT.

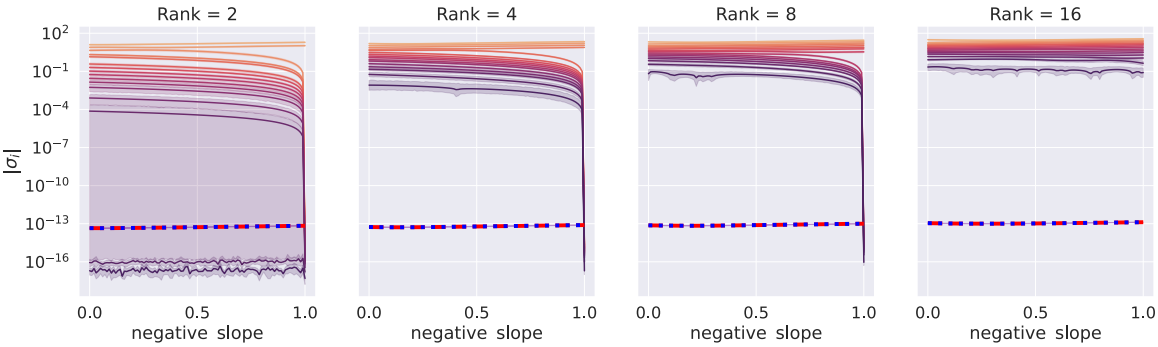



Figure 5.4: A numerical verification of the derived boundary over which a given eigenvalue computed on a Leaky-ReLU activation $\sigma_k$ will cease to contribute to the rank. In each panel, we plot how the change in estimated singular values as solid curves, with color corresponding to order of initial magnitude. We plot the rank boundary as a function of estimated largest eigenvalue as a red dotted line, and the rank boundary using (5.6) with a blue dotted line.

ReLU nonlinearity. We then estimate the bound to rank contribution first by computing it using the maximum estimated singular value, and then using the boundary we derived in (5.6) which does not require singular values. This empirical demonstration indicates that this derived bound is equivalent to numerical boundary on singular value contribution to rank computed from the output of the leaky-ReLU. This demonstrates that our theoretical result allows for exact prediction of when singular values will no longer contribute to rank, using only the singular values of the input to the activation function.

***Hypothesis 4:****The negative slope of the Leaky-ReLU activation is related to the amount*

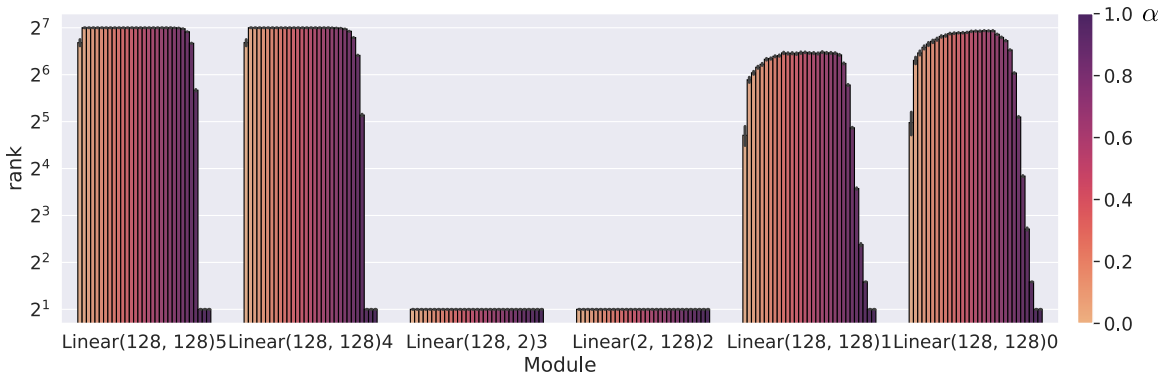



Figure 5.5: For a 5-layer (6 weight) FC network with Leaky-ReLU activations, we show how mean rank of gradients, activation, and deltas change with respect to the negative slope $\alpha$ of the noninearity. Layer sizes are plotted on the x axis with the depth increasing from left to right. We enforce a bottleneck of 2 neurons in the central layer. For each module, we estimate the rank and provide a colorbar corresponding to the level of nonlinearity increasing in the range of [0,1].

*of gradient rank restored*. Although our theoretical analysis in §5.3.3 was given primarily in terms of how Leaky-ReLU contributes to the rank of the input activations, it stands to reason that the resulting product of activations and adjoint variables would be affected by the negative slope as well. In Figure 5.5, we implement a 3-layer fully-connected network for reconstruction of Gaussian variables, and we compute the rank of the gradients changing as a function of the slope of the Leaky-ReLU activation. We use a network with 128 neurons at the input and output layers and a 2-neuron bottleneck layer. Our results indicate that pure ReLU activations do not fully restore gradient rank in either direction of back-propagation; however, the negative slope does increase estimated rank close to full-linearity. As the activations approach linearity, the rank returns to the gradient bottleneck.

### 5.5.2 Large-scale demonstration

In this section, we include results on larger-scale networks and problem settings. Our goal is to illustrate how architectural choices can affect rank not only in small models, but also in modern architectures. These results highlight the relevance of our theoretical result to the deep learning community at large.

***ImageNet with ResNet18:*** In Figure 5.6, we illustrate the effect of increasing the input

image size on the rank of the gradient in the ResNet18 architecture. In each panel the image size increases from top to bottom, and as expected smaller image sizes produce smaller ranks.

Additionally, we estimated the effect of artificially introduced Leaky-ReLU activations with different levels of $\alpha$; however, we observed little noticeable effect as long as the input image was large.

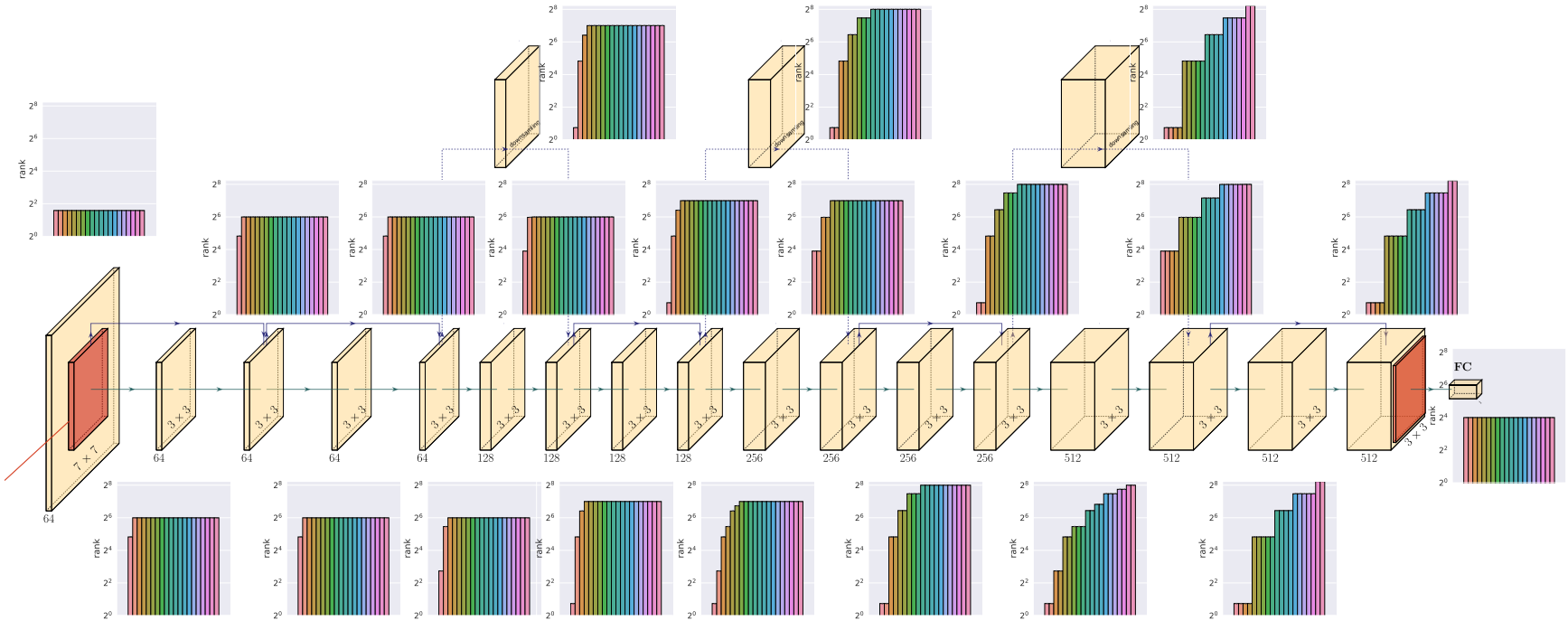



Figure 5.6: Illustration of the rank of the gradient at each layer in the ResNet18 architecture used to classify Tiny-Imagenet. Each panel shows the effect of increasing image size (from top to bottom) on the rank, illustrating that larger image spaces provide more accumulation of the gradient.

***ImageNet with VGG11:*** In Figure 5.7 we illustrate the effect of increasing the input image size on the rank of the gradient in the ResNet18 architecture. In each panel the image size increases from top to bottom, and as expected smaller image sizes produce smaller ranks.

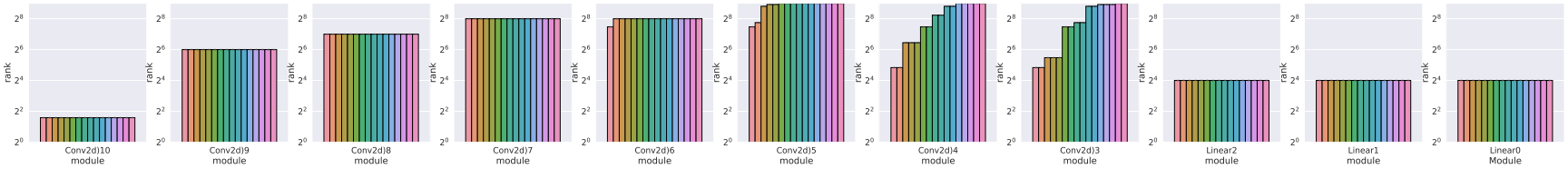



Figure 5.7: Illustration of the rank of the gradient at each layer in the VGG11 architecture used to classify Tiny-Imagenet. Each panel shows the effect of increasing image size (input is from top to bottom) on the rank, illustrating that larger image spaces provide more accumulation of the gradient.

***WikiText/M130k with BERT/XLM:*** We have included a demonstration of decreasing the sequence length in two large-scale NLP datasets (WikiText and Multi30K) for the BERT architecture with standard pretraining and XLM pretraining. In Figure 5.8 we include the estimated rank of the first two and last two linear layers for BERT applied to the WikiText data set. The rank for all other layers and for the4 XLM/Multi30k application are included in separate PDFs along with this supplement. In general we do not see much variation between linear layers in each architecture, except for near the input (layers 73,72), in which length 1 sequences collapse the rank of the gradients down to 1.

### 5.5.3 Phenomena Study of Other Nonlinear Activations

## 5.6 Discussion

Our theoretical bound on gradient rank in linear networks provides a number of startling insights. As we empirically demonstrate in Figure 5.1, linear networks with bottleneck layers are constrained to low-rank learning dynamics, where the rank cannot exceed that of the smallest number of neurons in the network. Beyond the somewhat artificial introduction of bottlenecked layers, these constraints also emerge when dealing with low-dimensional input spaces (even when they are embedded in higher-dimensional spaces like in Figure 1 in supplement). Perhaps more startling is the realization that in supervised learning tasks with relatively few output variables (such as only 10 neurons for 10 classes in Cifar10/MNIST, which can be seen in the rank of the linear classifier in Figures 2+3), the gradient rank throughout the entire network will not exceed that number of variables. When using linear activations, this finding suggests that the ramifications of neural collapse to input layers beyond the terminal linear classifier, with Neural Collapse playing a role at *all* phases of training, not just during the terminal phase. Since our analysis in this work is on architectural constraints which will affect dynamics during *all* training, further work studying the actual gradient dynamics and their effect on the weights during training is needed to fully flesh-out the connection to neural collapse.

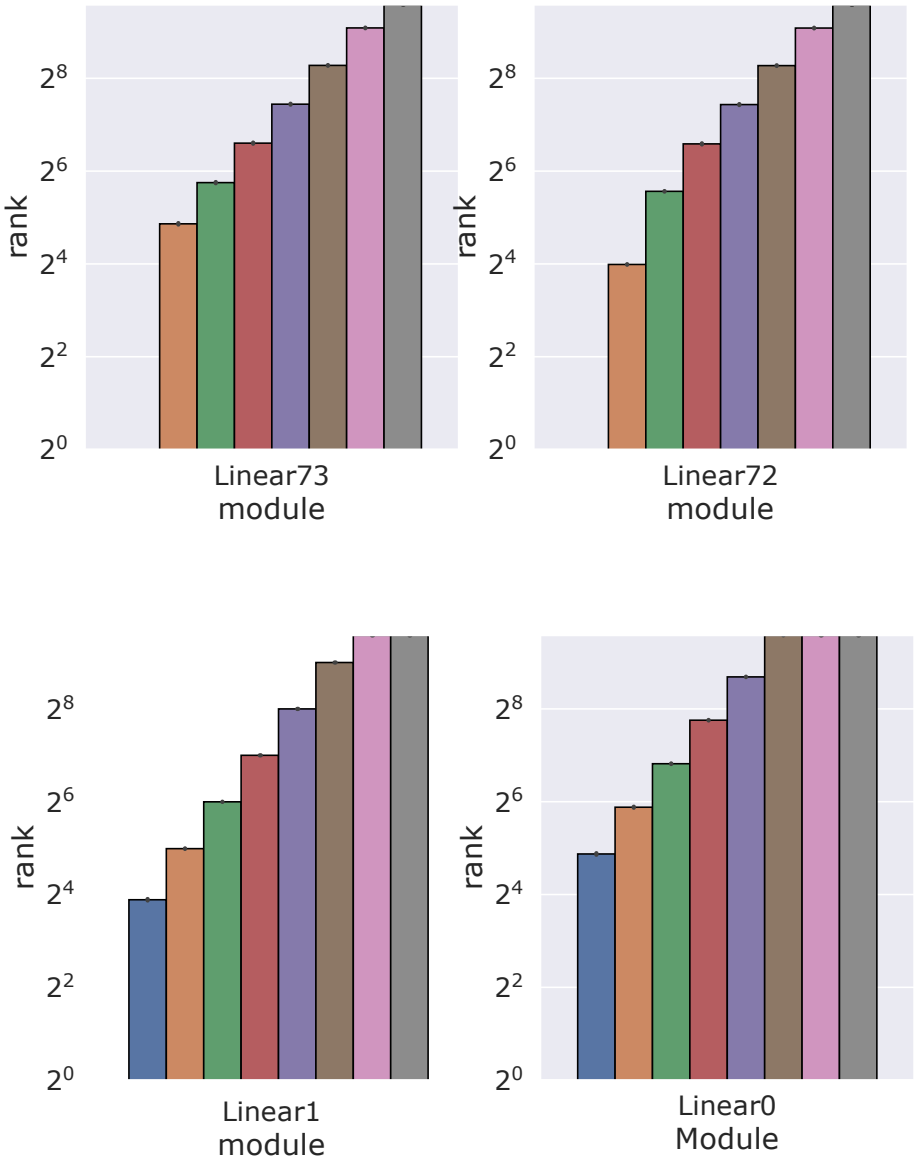



Figure 5.8: Illustration of the rank of the gradient at each layer in the BERT architecture used for language modeling on the WikiText2 data set. Each panel shows the effect of increasing sequence length (increasing from right to left) on the rank, illustrating that larger sequence lengths. provide more accumulation of the gradient.

Through our analysis of linear networks, we also provide an explanation for how parameter-tying mitigates the tight constraints on rank which may emerge due to bottlenecks. In Figures 5.4 and 5.5 our empirical verification demonstrates that when low-rank gradients computed at each point in the sequence, rank may be partially restored; however, the level to which rank is restored depends on the length of the sequence. The implication for networks with parameter tying is that aggressively truncated sequences in parameter-tied networks will still be affected by low-rank gradient dynamics.

Our theoretical result identifying how to compute the numerical boundary on rank for Leaky-ReLU networks provides a novel theoretical extension to how nonlinear activations can affect learning dynamics. Additionally, the ability to control the negative slope of the Leaky-ReLU activations allows us to demonstrate how numerical precision can affect the bounds. At the end of our analysis; however, we are left with a boundary that is highly data-dependent, and as we show in Figure 5.5 even fully nonlinear ReLU activations may lower the numerical estimation of rank. This remaining data-dependence in our theory suggests that there is only so much insight to be gained from architectural choice alone, and future work will require analysis of how particular input and output spaces may impose particular boundaries or dynamics on gradient rank. This input/output analysis may also provide deeper insights and tighter bounds the affects of nonlinear activations and parameter tying in particular, with highly correlated or sparse input and output spaces potentially affecting bounds and dynamics.

One limitation of our analysis in this work is that we have purposely avoided relating the emergence of low-rank phenomenon to model performance on particular tasks. Our reason for shying away from this discussion is that model performance is closely related to dynamics throughout the entire training phase, and our theoretical results apply to networks at *any* phase of training, and as such are agnostic to whether a model performs well or not. Our work here provides ample groundwork for analyzing the dynamics within our derived boundaries, and so we leave the connection of gradient rank collapse and performance as

future work which can focus on correspondences between collapse and dynamics.

An additional constraint of our analysis here is our restriction to analyzing linear and Leaky-ReLU networks. Our primary reason for doing this is that with minimal theoretical pretext, Linear and Leaky-ReLU networks call be discussed within the machinery of linear algebra. Other nonlinear activations such as the hyperbolic tangent, logistic sigmoid, and swish require significant theoretical pretext before the linear-algebraic notions of singular values and rank can be applied. Future work may be able to draw on mathematical fields of functional analysis (in the style of [205]), algebraic topology [218], or analysis of linearized activations (as is done in [219]) to provide theoretical frameworks for studying low-rank dynamics. We leave this as a substantial opening for future work.

## 5.7 Conclusion

In this work, we have presented a theoretical analysis of gradient rank in linear and Leaky-ReLU networks. Specifically, we have shown that intuitive bounds on gradient rank emerge as direct consequences of a number of architectural choices such as bottleneck layers, parameter tying, and level of linearity in hidden activations. Our empirical verification and demonstration illustrate the computed bounds in action on numerical and real data sets. The bounds we provide connect simple principles of architectural design with gradient dynamics, carving out the possible space in which those dynamics may emerge. Our work thus serves as a groundwork for continued study of the dynamics of neural spectrum collapse and gradient dynamics, not only in linear classifiers, but in many classes of network architecture.

# CHAPTER 6

# AUTOSPEC: SPECTRAL STATISTICS IN AUTO DIFFERENTIATION FOR INTROSPECTION AND IDENTIFICATION OF GROUP DIFFERENCES IN DEEP NEURAL NETWORKS LEARNING DYNAMICS

## 6.1 Introduction

Neural networks, which have had a profound effect on how researchers study complex phenomena, do so through a complex, nonlinear mathematical structure which can be difficult to interpret or understand. This obstacle can be especially salient when researchers want to better understand the emergence of particular model behaviors such as bias, overfitting, overparametrization, and more. In certain contexts such as Neuroimaging, the understanding of how such phenomena emerge is fundamental to preventing and informing users of the potential risks involved in practice.

In recent years, a broad library of "introspection" methods have emerged to mitigate the difficulty of straightforward interpretation [220, 221, 222, 223, 224, 225]. In particular, most state of the art research on DNN interpretation has utilized post-hoc analysis of model gradients. For example, the popular integrated gradients measure [220], saliency maps [225], and other gradient-based methods [222, 223] have proved popular in applications to convolutional neural networks as they provide simple spatial visualizations which make for easy building of intuition. While these methods are extremely popular, the resulting visualizations tend to be noisy and inconsistent [226, 227, 228, 229], and recent research in the field has attempted to mitigate these limitations, for example by using refining maps based on region attribution density [230, 231], providing an adaptive interpolation path [232], or imposing geometric constraints on produced maps [233, 234].

While post-hoc gradient analysis techniques can be useful for interpretation on pre-

trained models, these methods are not useful for evaluating training dynamics; they can explain where a model currently is, but now how it arrived there. While theoretical analyses of dynamics such as the one we developed in chapter 5 and related analyses [81, 82] provide some guidance to describing model dynamics, new empirical metrics and visualizations are also important for building intuition about model behavior while also accumulating evidence for further theoretical interpretation.

One existing method which empirically describes model dynamics is the information-plane method [83, 84, 235], in which the mutual information between layer weights and the input and output spaces is computed and visualized during training. While this method provides fascinating visualizations of dynamic behavior, the general interpretation of the dynamics adhering to the information bottleneck principle [236] is not immediately clear [85] and can be sensitive to architectural choices or the empirical method used to estimate mutual information.

In this chapter, we present a novel empirical method for analyzing model learning dynamics, which builds off of our theoretical work studying gradient rank in chapter 5. We call our method AutoSpec, as it utilizes unique opportunities within auto-differnetiation to analyze model learning dynamics. While we previously showed that model architecture can impose theoretical limits on gradient rank, there is more that can be observed at work within auto-differentiation. Namely, we show that the singular values of the gradient and of the component matrices which are used to compute it can be studied they dynamically evolve during model training. Furthermore, because we can analyze gradients on-the-fly within the auto-differentiation mechanism, we have the unique opportunity to analyze these dynamics as they adhere to individual samples from the training data set. As long as these samples have some kind of common group labelling, we can thus do statistical comparisons of gradient trajectories between groups without breaking normal training behavior. This further allows our method to stand out from post-hoc methods which not only occur outside of training, but can only be evaluated for between different classes in disjoint

contexts.

## 6.2 Methods

In this section, we provide an overview of the methods at work in AutoSpec. First, we provide a brier review of how the gradient of the weights is computed within auto-differentiation, recall how the spectrum is bounded by particular architectural decisions, and show how the spectrum of the gradient relates to the spectrum of the input activations and adjoint variables accumulated within auto-differentiation. We then describe how auto-differentiation uniquely allows us to analyze dynamics between particular groups of samples.

### 6.2.1 Gradient Spectra via Auto-Differentiation

Recall from chapters 4 and 5 that during reverse-mode auto-differentiation, the gradient of the weights at a given layer $i$ is computed as a product of the input activations $\mathbb{A}$ and the adjoint variable $\gtrless$ which is the partial derivative computed on the output neurons during back-propagation. Formally, if we have weights $\mathbb{W}_i \in \mathbb{R}^{h_{i-1} \times h_i}$ where $h_{i-1}$ and $h_i$ are the number of input and output neurons respectively. Formally, we write this as:

$$\nabla_{\mathbf{W}_i} = \mathbf{A}_{i-1}^{\top} \boldsymbol{\Delta}_i \tag{6.1}$$

where $\mathbb{A} \in \mathbb{R}^{N \times h_{i-1}}$ and $\gtrless \in \mathbb{R}^{N \times h_i}$ are the input activations and adjoint variables with batch size $N$. See chapters 5 and 6 for a further review of how these variables are all computed. The Singular Value Decompositions of $\nabla_{\mathbf{W}_i}$, $\mathbf{A}_{i-1}$ and $\boldsymbol{\Delta}_i$ can be written as:

$$\nabla_{\mathbf{W}_i} = \mathbf{U}_{\nabla_i} \boldsymbol{\Sigma}_{\nabla_i} \mathbf{V}_{\nabla_i}^{\top}, \;\; \mathbf{A}_{i-1} = \mathbf{U}_{A_{i-1}} \boldsymbol{\Sigma}_{A_{i-1}} \mathbf{V}_{A_{i-1}}^{\top}, \;\; \boldsymbol{\Delta}_i = \mathbf{U}_{\Delta_i} \boldsymbol{\Sigma}_{\Delta_i} \mathbf{V}_{\Delta_i}^{\top} \tag{6.2}$$

We can then write $\nabla_{\mathbf{W_i}}$ as a product of the SVDs of $\mathbf{A}_{i-1}$ and $\boldsymbol{\Delta}_i$, and use the fact that the $\mathbf{U}$ matrices are orthogonal to get:

$$\nabla_{\mathbf{W}_i} = \mathbf{V}_{A_{i-1}} \mathbf{\Sigma}_{A_{i-1}} \mathbf{U}_{A_{i-1}} \mathbf{U}_{\Delta_i}^\top \mathbf{\Sigma}_{\Delta_i} \mathbf{V}_{\Delta_i}^\top \tag{6.3}$$

$$= \mathbf{V}_{A_{i-1}} \mathbf{\Sigma}_{A_{i-1}} \mathbf{\Sigma}_{\Delta_i} \mathbf{V}_{\Delta_i}^\top \tag{6.4}$$

Thus, we can see that the singular values of $\nabla_{\mathbf{W}_i}$ are just the singular values of the first $\min(h_{i-1}, h)$ singular values from the input activations and adjoint variable.

For the sake of analysis, we can compute the SVD of all three statistics-of-interest just by computing the SVD and the input activations and adjoint matrices; however, because the batch size $N$ might be large, if the gradient is the only statistic of interest, it would often be more efficient to compute the SVD of $\nabla_{\mathbf{W}_i}$ directly.

For networks which utilize parameter tying, such as Convolutional or Recurrent Neural Networks, the gradients are often accumulated over time and space. If desired, our unique perspective from within Auto-Differentiation allows us to peek further into these dimensions, characterizing the spectra not only of the gradient of the weights, but of the gradient of the weights over the dimension of tying.

### 6.2.2 Identifying Group Differences

One of the advantages of computing our introspection statistics within auto-differentiation is that we have access to the individual gradients for each input sample to the model. Thus, we can evaluate how particular groups of samples individually contribute to the aggregated gradients prior to the aggregated gradient. Formally, if we have $C$ distinct groups of samples in our training set, we can compute the set of $C$ gradients and their SVD as

$$\{\nabla_c = \mathbf{U}_c \mathbf{\Sigma}_c \mathbf{V}_c^\top\}_{c=1}^{C} \tag{6.5}$$

we can then perform statistical testing by accumulating these group-specific statistics

during training, and evaluating the differences between groups. For example, if we perform a two-tailed T-test, we can obtain a measure of which training steps were significant between groups if we treat the number of singular values as features. Additionally, we can obtain a measure of per-singular-value significance by taking the T-tests between the transpose.

### 6.2.3 Data sets and Experimental Design

To demonstrate the kinds of dynamics which AutoSpec can reveal, we have organized a battery of experiments across different data modalities and architecture types.

First, we use two numerical data sets to show how AutoSpec allows for dynanic introspection on Multi-Layer Perceptrons, Elman Cell RNNs, and 2-D CNNs. Our choice of numerical data sets are the MNIST and Sinusoid data sets which have presented previously in chapter 5.

We then move from numerical data to an application in Neuroimaging analysis. We first apply a Multi-Layer perceptron on the FreeSurfer volumes we utilized in chapter 4, and then move to analyzing functional MRI from the COBRE data set [237], which is a well-studied data set especially for applications of deep learning [238, 239, 240, 241]. For our demonstration, we perform Spatially Constrained Independent Component Analysis using the NeuroMark template [242], which provides us with 53 neurologically relevant spatially independent maps and associated time-series. Using the time-series data, we demonstrate how AutoSpec can reveal group-specific gradient dynamics in 1D and 2D CNNs, LSTMs and the BERT transformer [243]. We then utiliez the spatial maps (aggregated over the number of components by taking the maximum over the voxel dimension) to demonstrate group-specific gradient dynamics in 3D-CNNs.

For all architectures and all data sets, we evaluate a few different scenarios demonstrating the diversity of dynamics available in gradient spectra. For all models and datasets, we perform two tasks: classification and auto-encoding; however, we only demonstrate one

model instance and group differences for the auto-encoding task for clarity. Additionally, we evaluate "wide/shallow" (1 layer with 128 neurons) and "deep/thin" (3 layers with 8 neurons) variants of each model. We finally compare how a different choice of activation function can affect dynamics by evaluating each model with Sigmoid and Tanh activations in constrast to ReLU activations which we use elsewhere. For all analyses, we perform two-tailed T-Tests between each pair of classes in a given data set to demonstrate where significant group differences emerge within a particular scenario. Group comparisons for all architectural variants are included in supplementary material.

A detailed outline of our experimental is included in 6.1 and 6.2. When not otherwise specified, we use ReLU activations in all models, a learning rate of $1 \times 10^{-3}$, and 1000 epochs of training. Where possible, we perform full-batch training rather than SGD, as the batch size can artificially restrict the rank of the gradient as we explored in chapter 5. For the sake of this demonstration, we set all models to use the same seed, and we only evaluate the models in a one-shot training scenario.

### 6.3 Results

In this section, we present the empirical results demonstrating how AutoSpec can be used to reveal group gradient dynamics in deep neural networks. All of our figures follow the same format as follows: panels A and B compare the dynamics between a model trained for sample reconstruction (panel A) and for classification (panel B); panels C and D compare dynamics between tanh (panel C) and relu (panel D) activations, panels E and F compare dynamics between "thin" (panel E) and "wide" (panel F) variants of the base network with 8 and 64 neurons respectively. For a review of how each experiment is organized into panels see table 6.1.

The experiments on the MNIST data set are included in 6.1 and 6.2 for the MLP and 2DCNN architectures respectively. The experiments for the sinusoid data set evaluated with an RNN can be found in 6.3. The MLP applied to FSL data can be found in 6.4.

Table 6.1

| Panel | Task | Model Dims | Activation |
|---|---|---|---|
| A | Auto-Encoding | [32] | ReLU |
| B | Classification | [32] | ReLU |
| C | Auto-Encoding | [32] | Tanh |
| D | Auto-Encoding | [32] | Sigmoid |
| E | Auto-Encoding | [8] | ReLU |
| F | Auto-Encoding | [64] | ReLU |
| G | Auto-Encoding (Group Differences) | [32] | ReLU |
| H | Classification (Group Differences) | [32] | ReLU |

Table 6.2

| Dataset | Modality | Model | Figure |
|---|---|---|---|
| MNIST | Image | MLP | 6.1 |
| MNIST | Image | 2DCNN | 6.2 |
| SINUSOID | Time-Series | RNN | 6.3 |
| FreeSurfer | Tabular | MLP | 6.4 |
| COBRE | ICA Time-Series | LSTM | 6.5 |
| COBRE | ICA Time-Series | BERT | 6.6 |
| COBRE | ICA Time-Series | 1D-CNN | 6.7 |
| COBRE | ICA Spatial Mapps | 3D-CNN | 6.8 |

The experiments on COBRE ICA time-series can be found in figures 6.5, 6.6 and 6.7 for the LSTM, BERT, and 1D-CNN architectures respectively. Finally, the experiments on COBRE ICA spatial maps can be found in figure 6.8. See table 6.2 for a review of which data set and architecture combinations can be found in particular figures.

## 6.4 Discussion

In this chapter, we have introduced a new method for model introspection called AutoSpec, in which we utilize the singular value decomposition to study the dynamic evolution of the spectrum of the gradient and its component matrices. Our method reveals fascinating dynamics at work in a number of model architectures, and also allows us to identify unique dynamics belonging to particular groups within a data set. We demonstrated our model on numerical datasets for sequence and image reconstruction and classification, and also

demonstrated the identification of group differences on a real neuroimaging data set.

We will provide a brief discussion of some of the observed differences in dynamics; however, we would like to stipulate that any general conclusions regarding these dynamics will require further experimentation testing specific architecture choices systematically over many repetitions, seeds, and data sets. First of all we notice that for all data sets and architectures, the dynamics we find with AutoSpec show very different trajectories between and AutoEncoding and Classification task (see 6.1a and 6.1b for the MLP applied to MNIST for example). Particularly the input layers across all cases are affected by the change in the output structure, and corresponding singular values in output layers are also different between the two tasks. The choice of activation function can affect the gradient spectrum as well - we notice for example when we compare the Sigmoid and Tanh activated LSTM (6.5) and 1D CNN (6.7), we can see slight differences in the trajectory toward the start of training, but the overall evolution stays the same. In the 2D (6.2) and 3D (6.8) CNNs, however, we notice that Sigmoid and Tanh activations affect the spectrum quite differently, with the effects in the 3D CNN particularly noticeable early in the training period. The dynamics in the layers pulled from the BERT architecture (6.6) are difficult to interpret as singular values tend to jump drastically between individual epochs, perhaps indicated; however, even in this noisy scenario, the BERT architecture reflects a general trend of shrinking singular values which occurs in other architures during training.

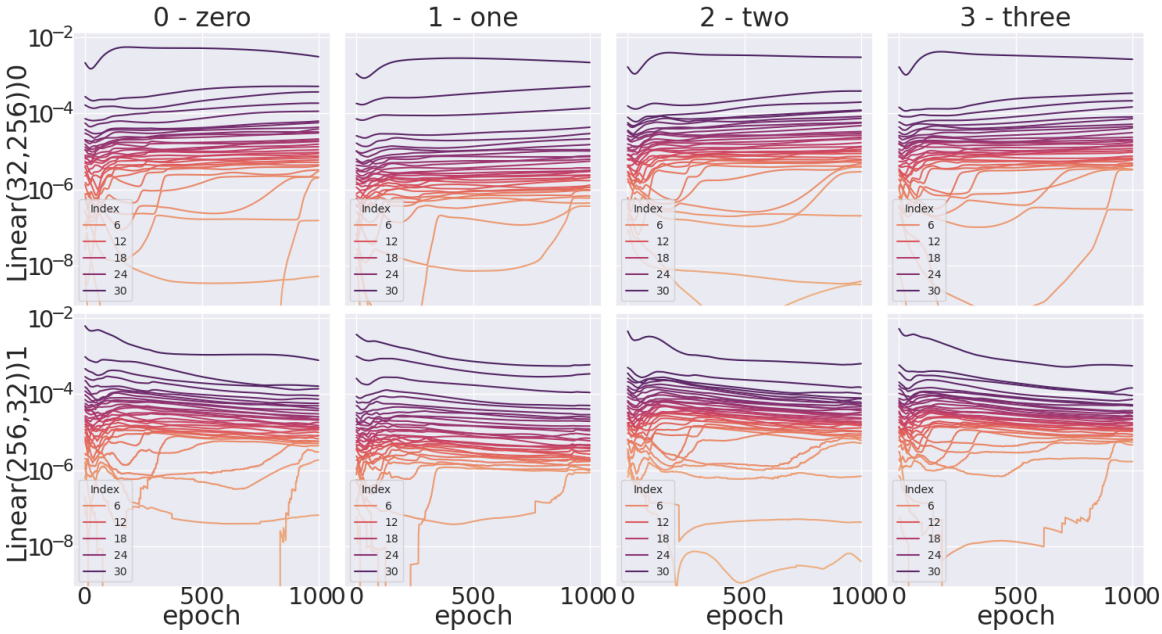
In each of our analyses, we also compute group differences between the observed dynamics. In general, smaller observed singular values tend to change more drastically; however, because the values are so small and near the threshold for machine epsilon of floating point numbers, it is unclear if any observed differences are merely the result of noise. We do see larger singular values show significant differences during training across a few different architectures however, and interestingly, these differences are often contained with a few layers. For example, the MLP autoencoder trained on FSL data shows significant differences between SZ and HC groups in the middle singular values of the output layer

(see 6.4g), while the corresponding classifier shows more group differences in the input layer (see 6.4h). The LSTM, BERT and 1D CNN models all show significant differences between male SZ and HC groups, with the LSTM showing differences mostly the Hidden-to-Hidden gradients (see 6.5g and 6.5h), the 1D CNN showing significant differences at the output layer (see 6.7g and 6.7h), and BERT showing differences across the entire model for the autoencoder task (see 6.6g) with almost no significant differences in the classifier ((see 6.6h). While more work is needed to investigate what these differences mean for how the model treats different sample groups differently, our finding of significant effects across multiple tasks and architectures demonstrates that these kinds of dynamics may be useful for further, targeted investigation.

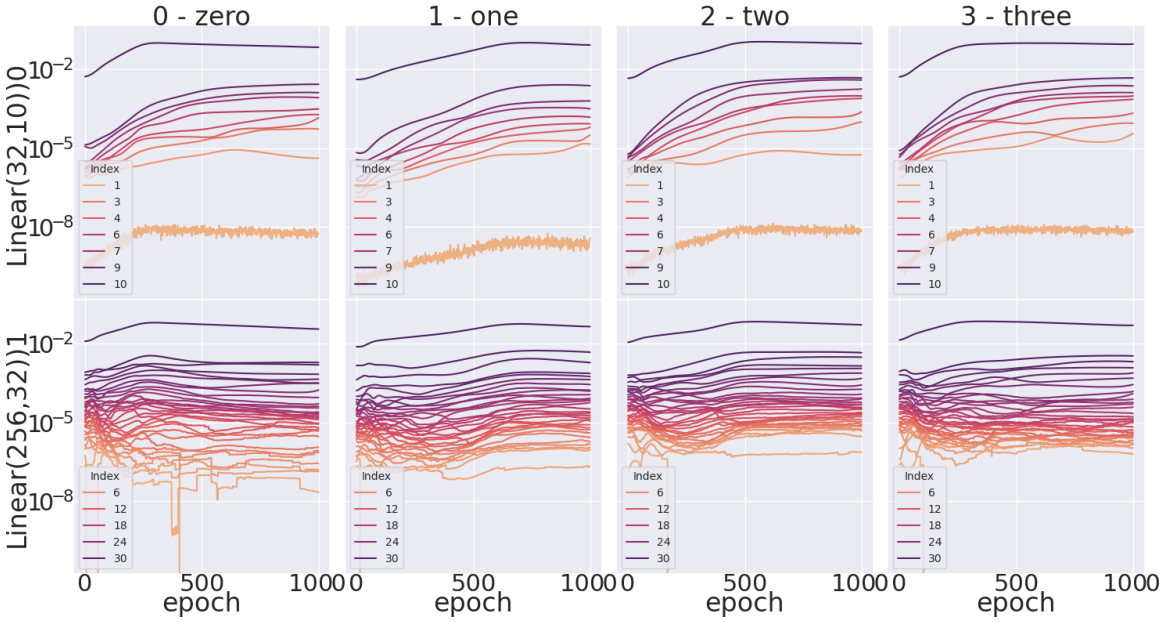
The major limitation of our AutoSpec framework is the computational overhead required for computing the singular values on-the-fly during training. In general for a matrix $A \in \mathbb{R}^{m \times n}$, the complexity required for computing the SVD is $O(\min(mn^2, m^2n))$. If we want to analyze $L$ different layers at $T$ different training periods, the complexity of our method further increases to $O(LT \min(mn^2, m^2n))$. Even more overhead is accumulated if we perform group-specific analyses. As such, AutoSpec in practical usage will require long runtimes during training to perform a comprehensive analysis; however, limiting the analysis to specific singular values, layers, periods during training, or groups would reduce this overhead. One potential direction of future work could derive an analytic update to the gradient spectrum based on the initial SVD of the weights and input data, and thus avoid recomputing the SVD entirely for each update.

The computational overhead has limited our experiments in this work to smaller architectures, or variants of large architectures with smaller dimension sizes. In future work, we would like to expand the analysis to one or two larger scale architectures to provide principled insights into the dynamics of these models; however, for the sake of surveying many model types in this work we have kept the scope smaller for scalability.

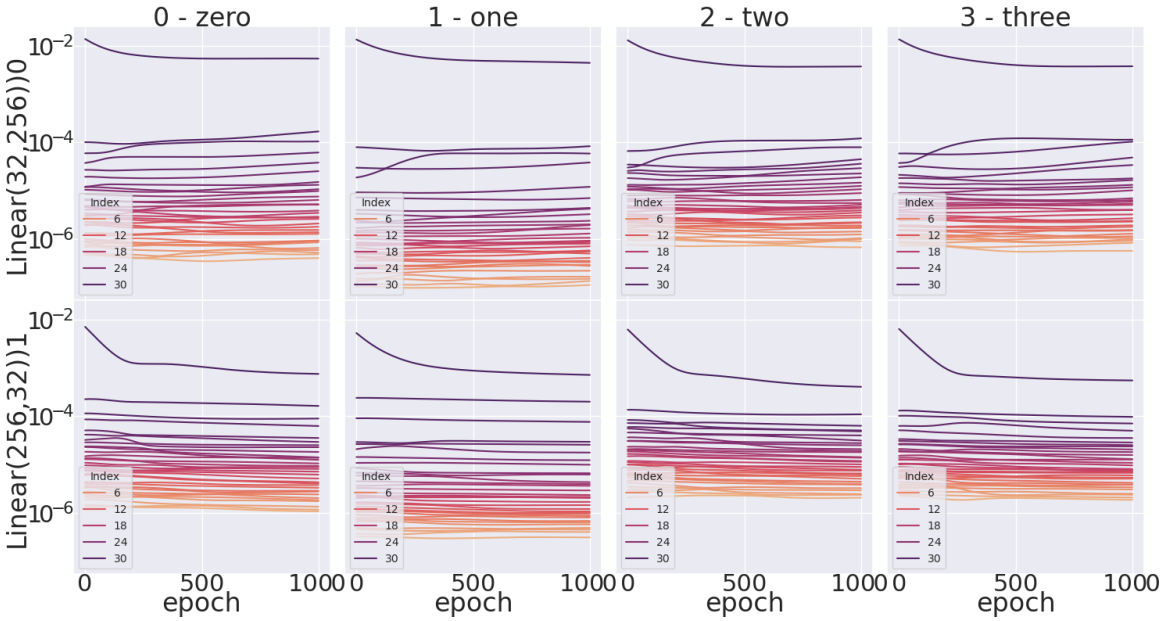
Figure 6.1: Differences in Auto-Differentiation Spectra Dynamics for the first 4 classes in the MNIST data set, trained with various architectures and tasks with a Multi-Layer Perceptron.



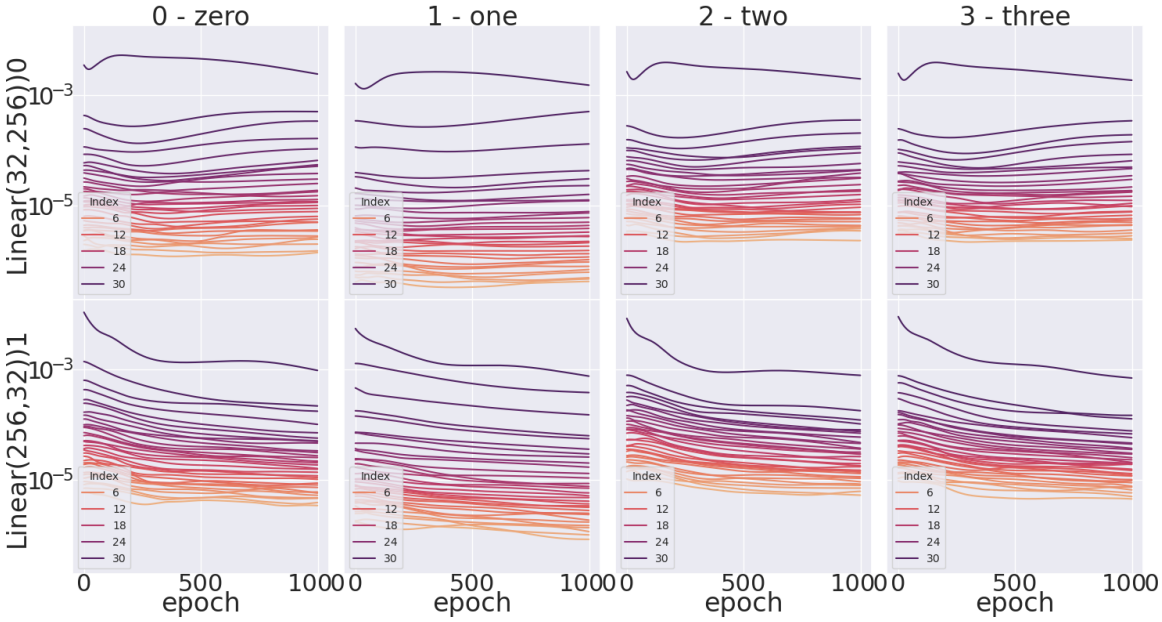

(a) MLP Autoencoder with hidden dim 32 and ReLU activations



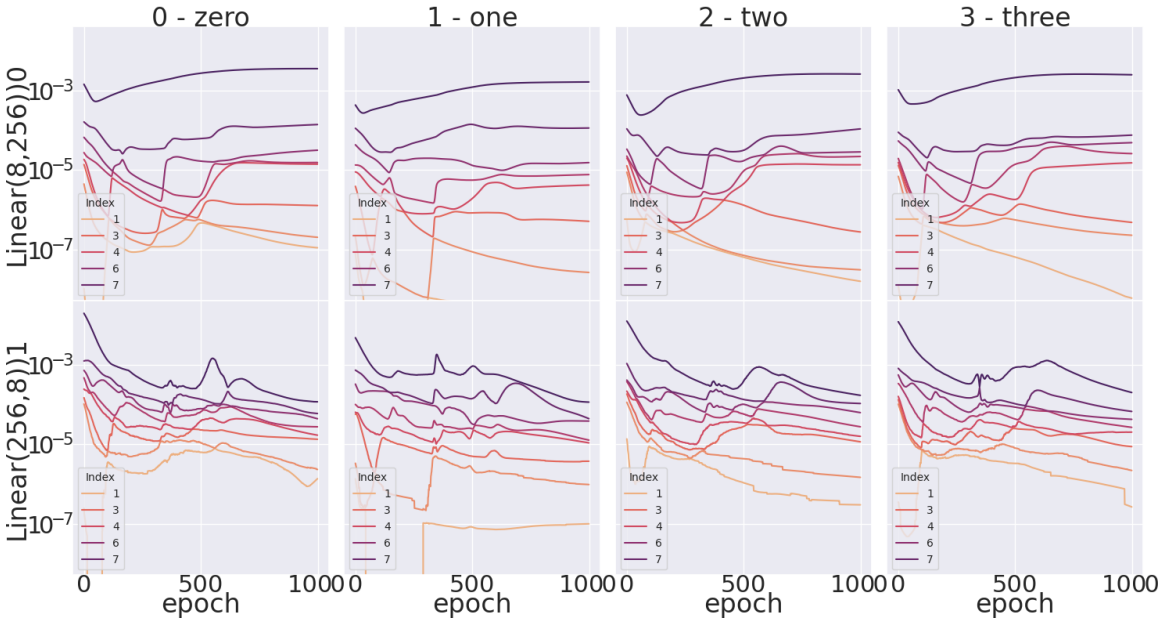

(b) MLP Classifier with hidden dim 32 and ReLU activations



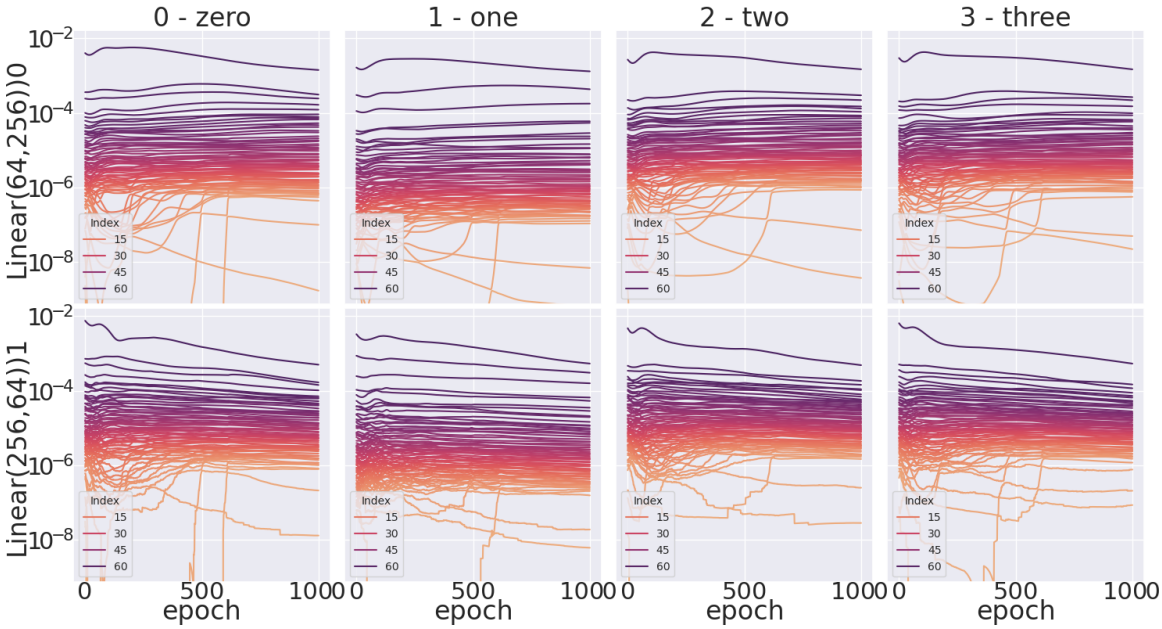

(c) MLP Autoencoder with hidden dim 32 and Sigmoid activations



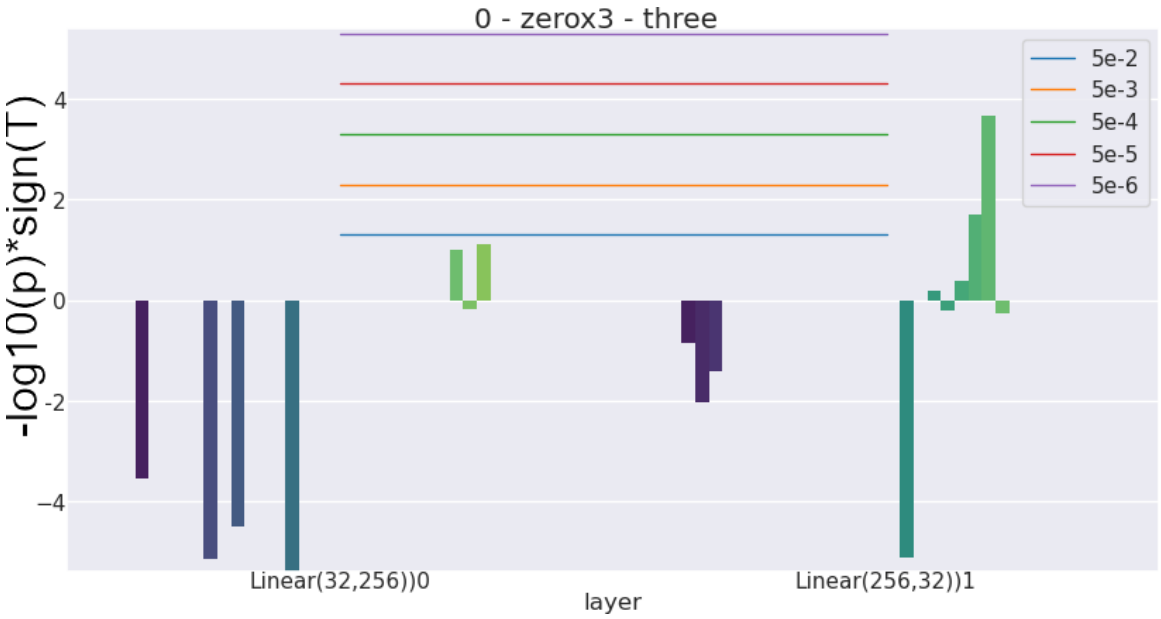

(d) MLP Autoencoder with hidden dim 32 and Tanh activations



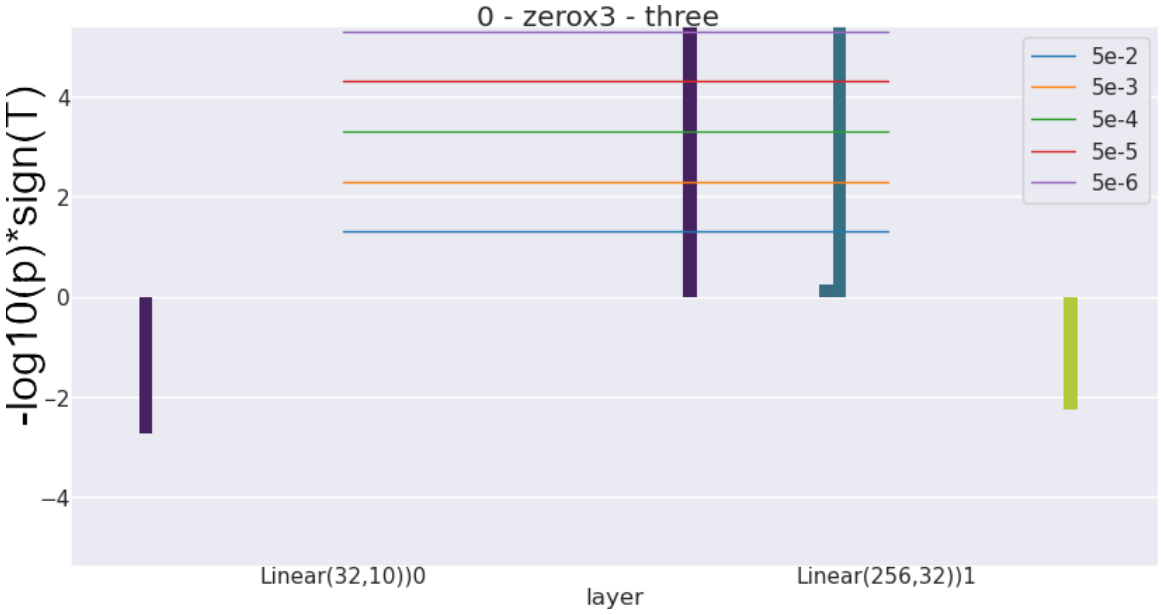

(e) MLP Autoencoder with hidden dim 8 and ReLU activations




(f) MLP Autoencoder with hidden dim 64 and ReLU activations



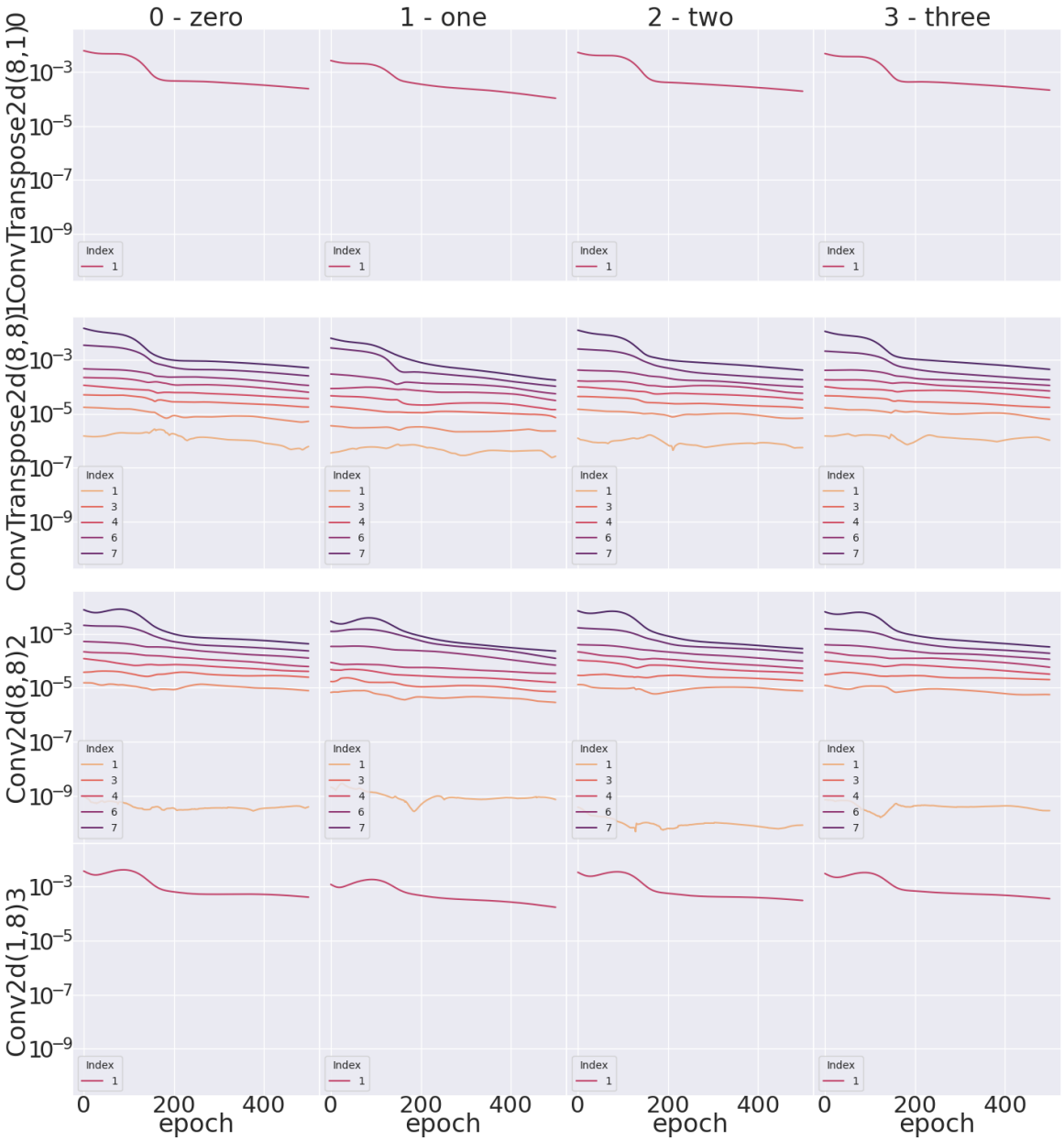

(g) Significant spectral differences between groups 0 and 3: MLP Autoencoder with hidden dim 32 and ReLU activations



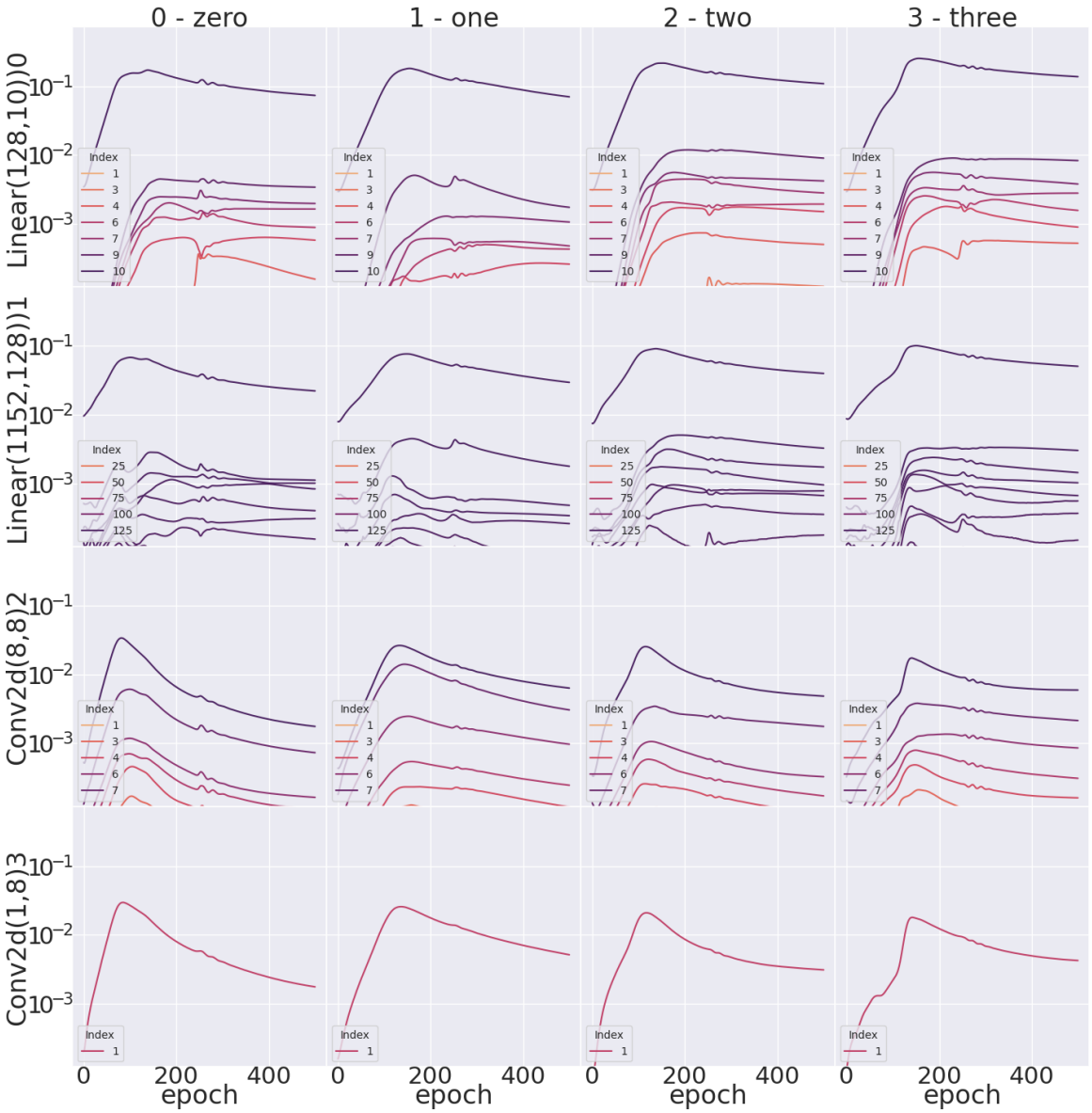

(h) Significant spectral differences between groups 0 and 3: MLP Classifier with hidden dim 32 and ReLU activations

Figure 6.1: Differences in Auto-Differentiation Spectra Dynamics for the first 4 classes in the MNIST data set, trained with various architectures and tasks with a Multi-Layer Perceptron.

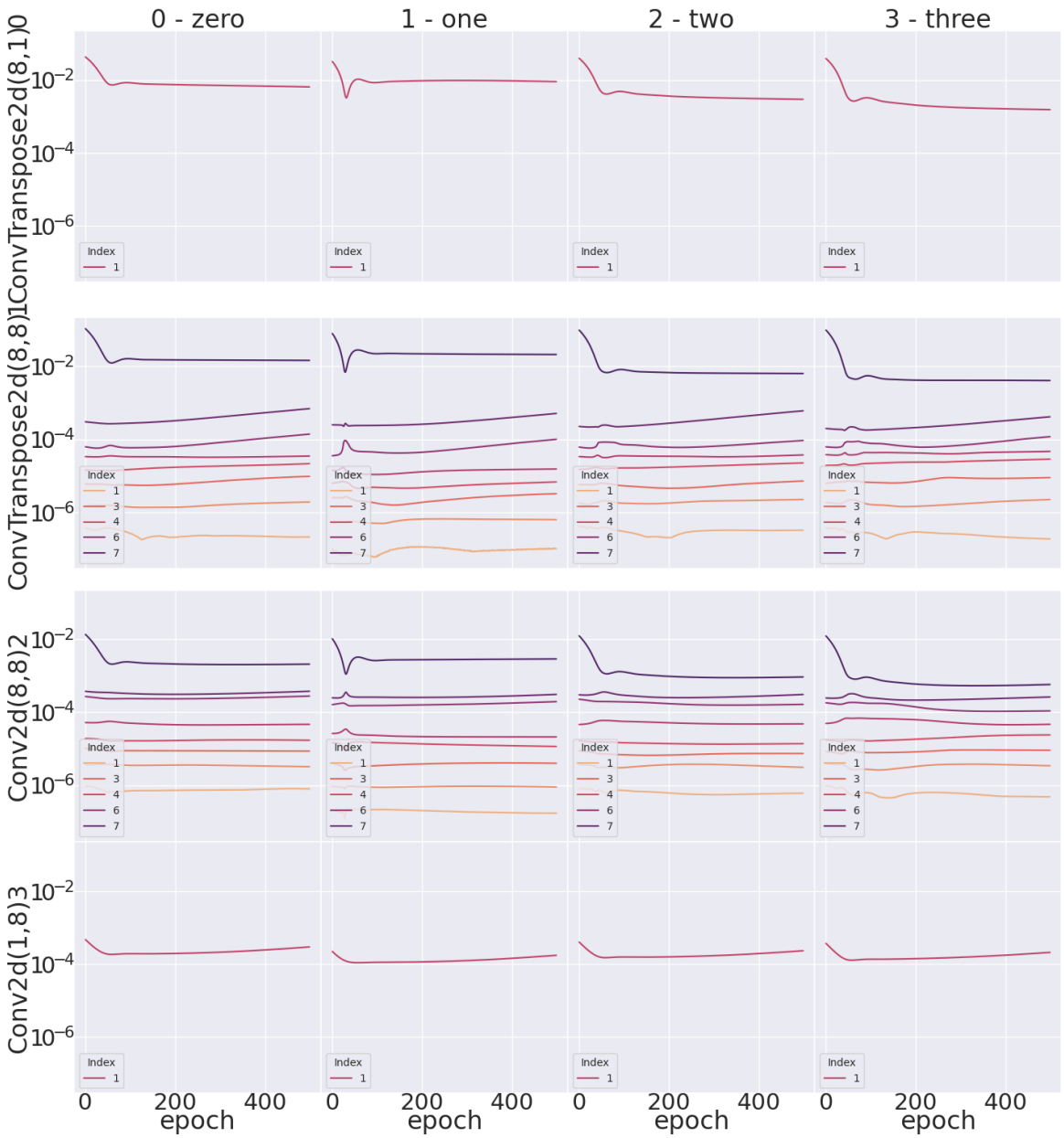
Figure 6.2: Differences in Auto-Differentiation Spectra Dynamics for the first 4 classes in the MNIST data set, trained with various architectures and tasks with a 2D CNN.



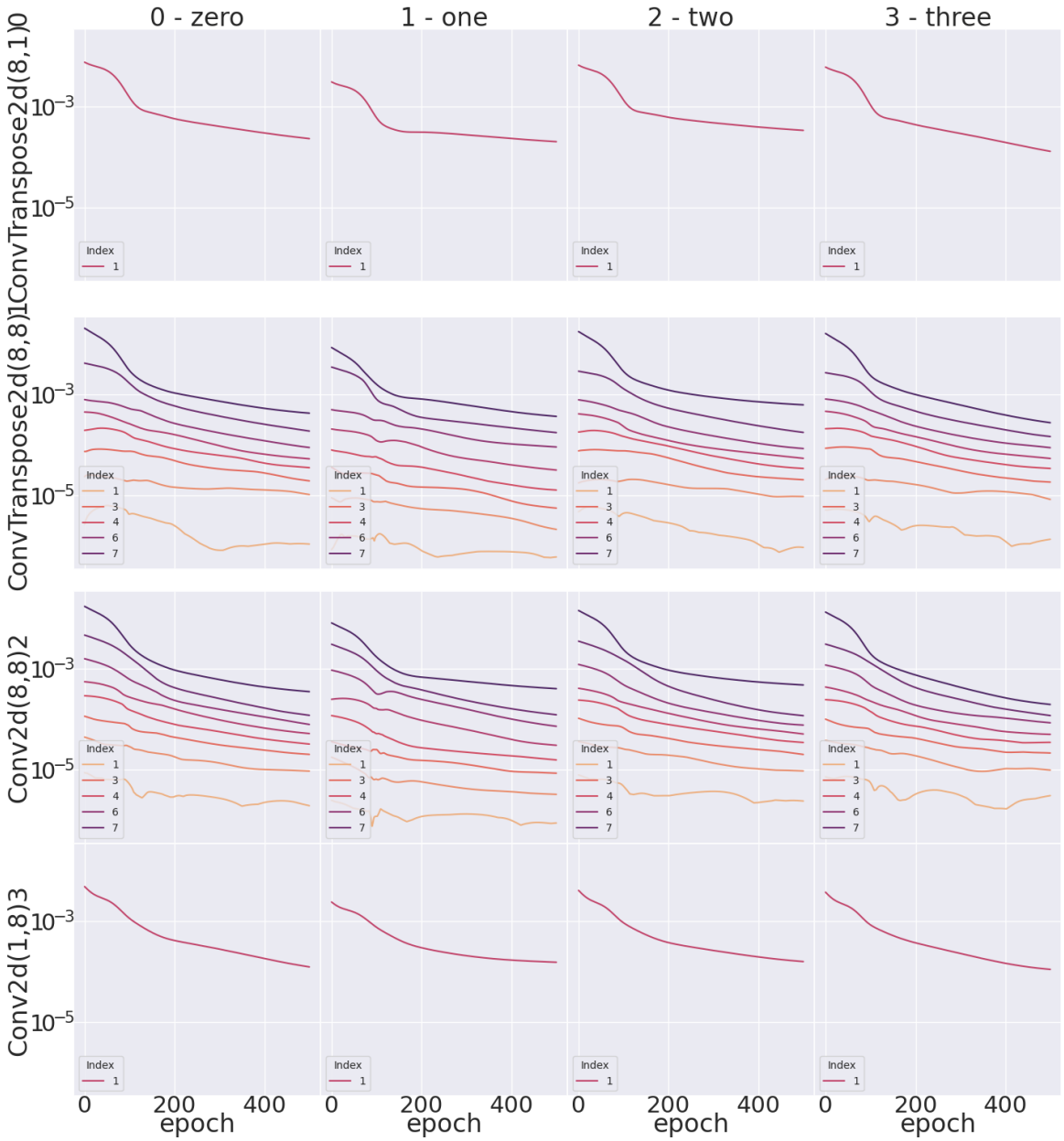

(a) CNN2D Autoencoder with hidden dim 8 and ReLU activations



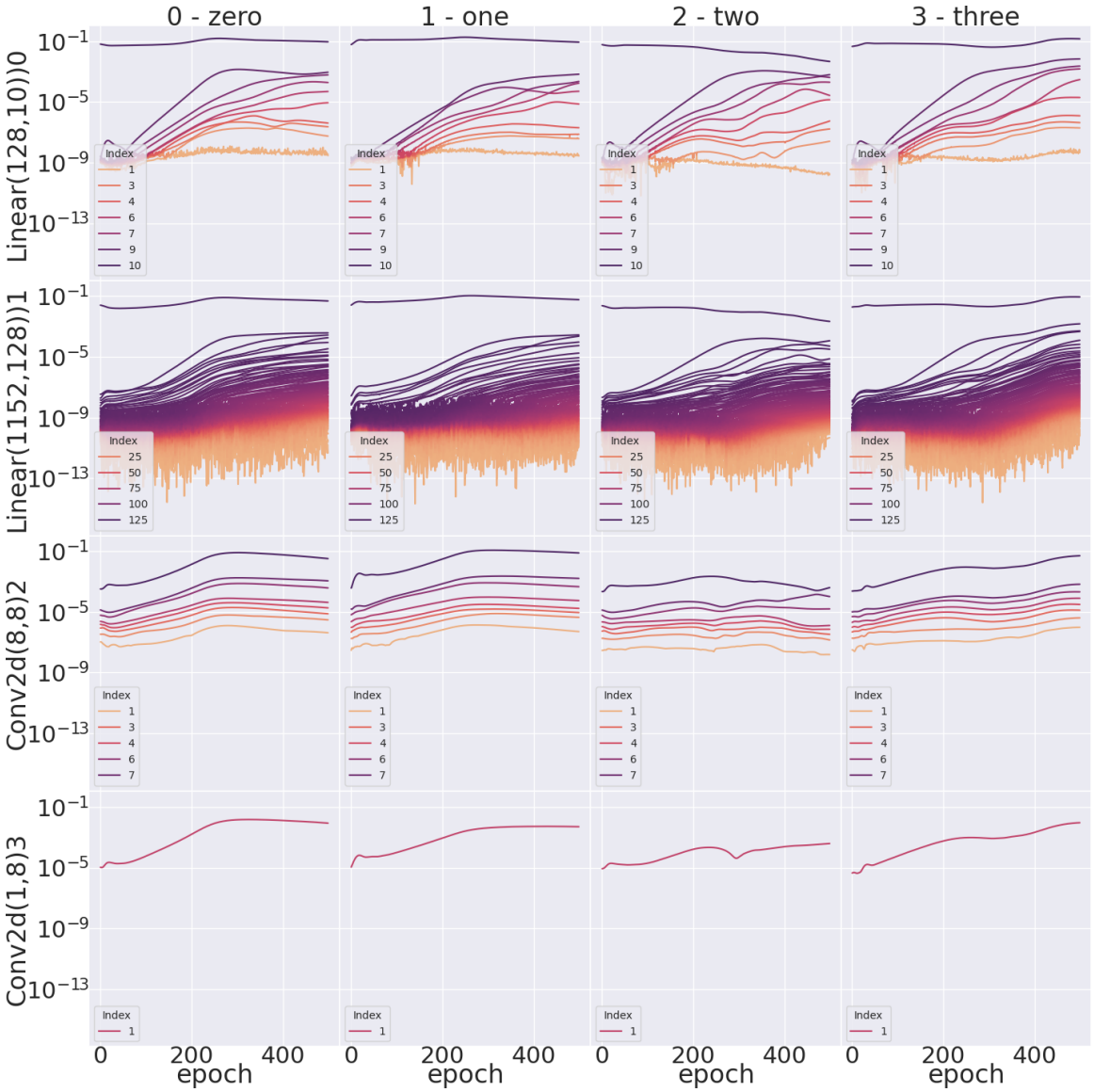

(b) CNN2D Classifier with hidden dim 8 and ReLU activations



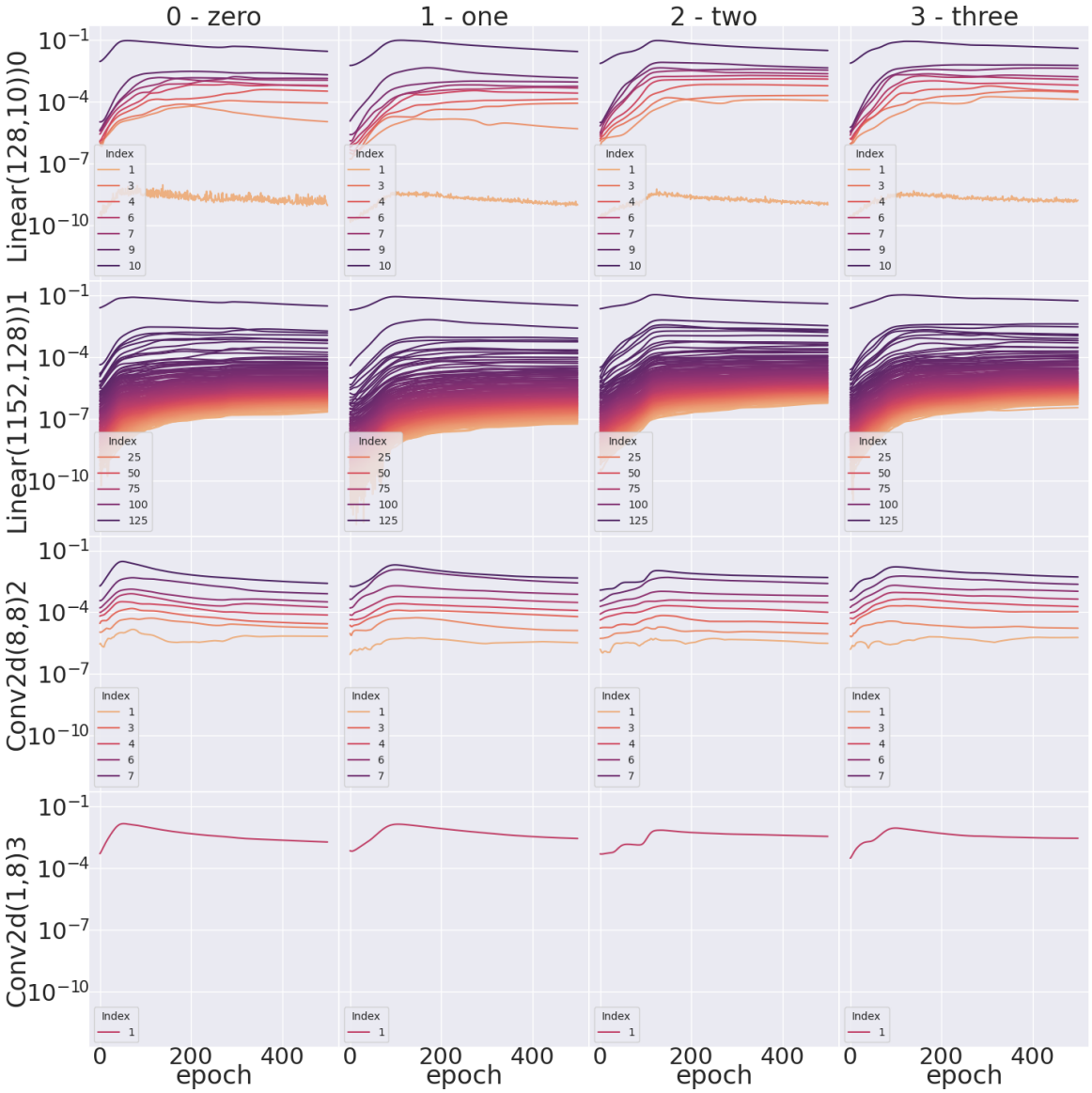

(c) CNN2D Autoencoder with hidden dim 8 and Sigmoid activations



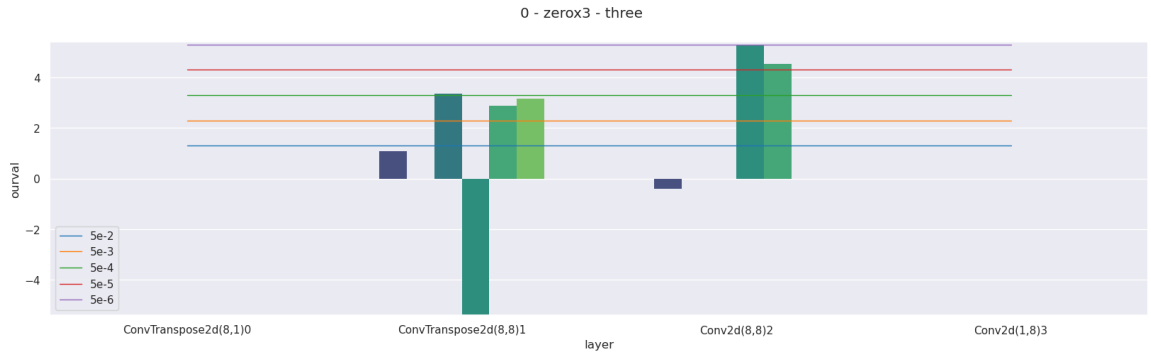

(d) CNN2D Autoencoder with hidden dim 8 and Tanh activations



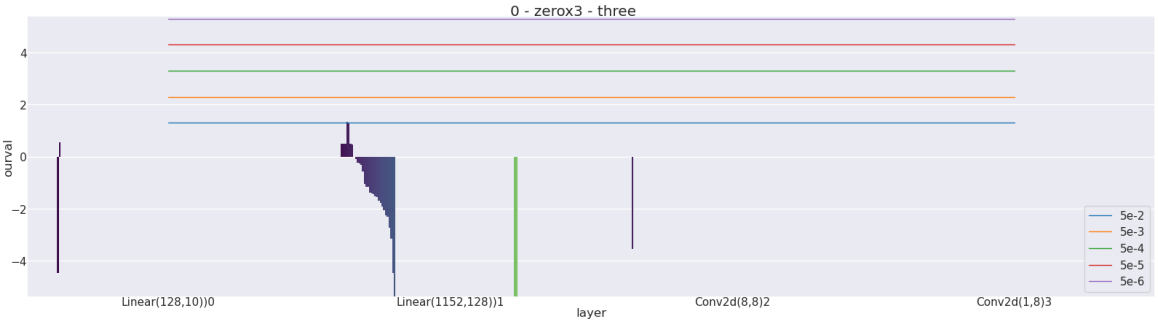

(e) CNN2D Classifier with hidden dim 8 and Sigmoid activations




(f) CNN2D Classifier with hidden dim 8 and Tanh activations



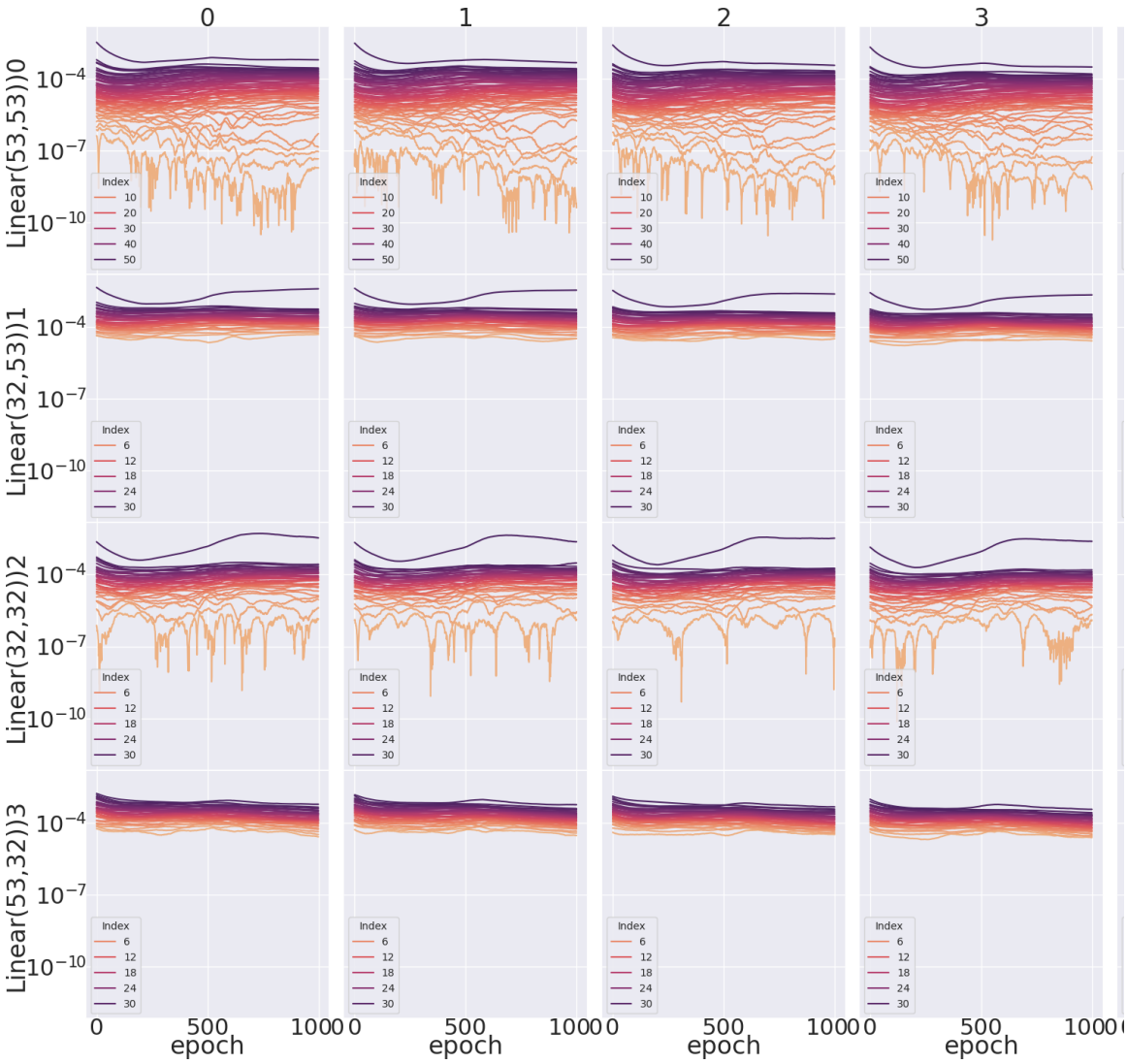

(g) Significant spectral differences between groups 0 and 3: CNN2D Autoencoder with hidden dim 8 and ReLU activations



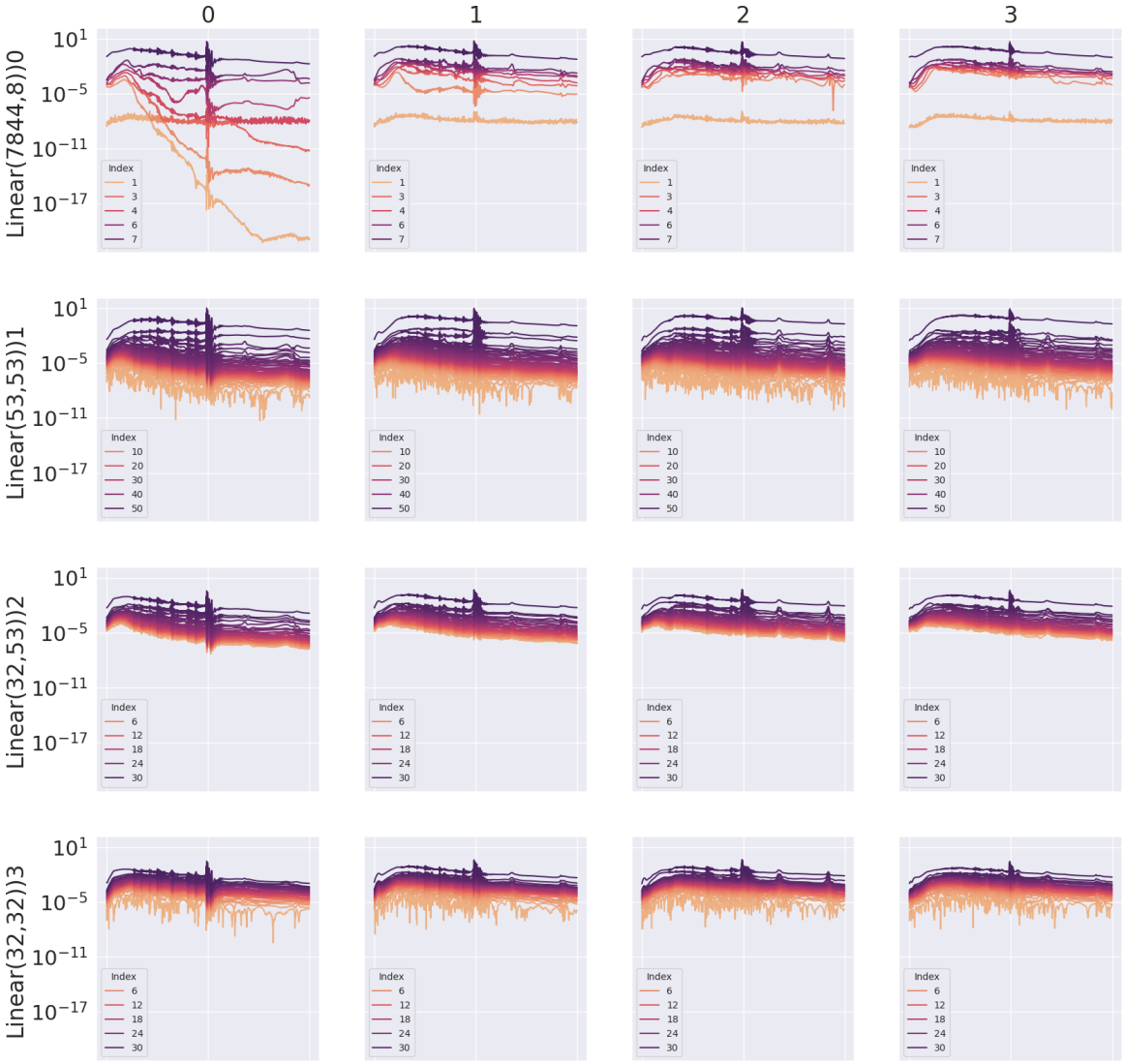

(h) Significant spectral differences between groups 0 and 3: CNN2D Classifier with hidden dim 8 and ReLU activations

Figure 6.2: Differences in Auto-Differentiation Spectra Dynamics for the first 4 classes in the MNIST data set, trained with various architectures and tasks with a 2D CNN.

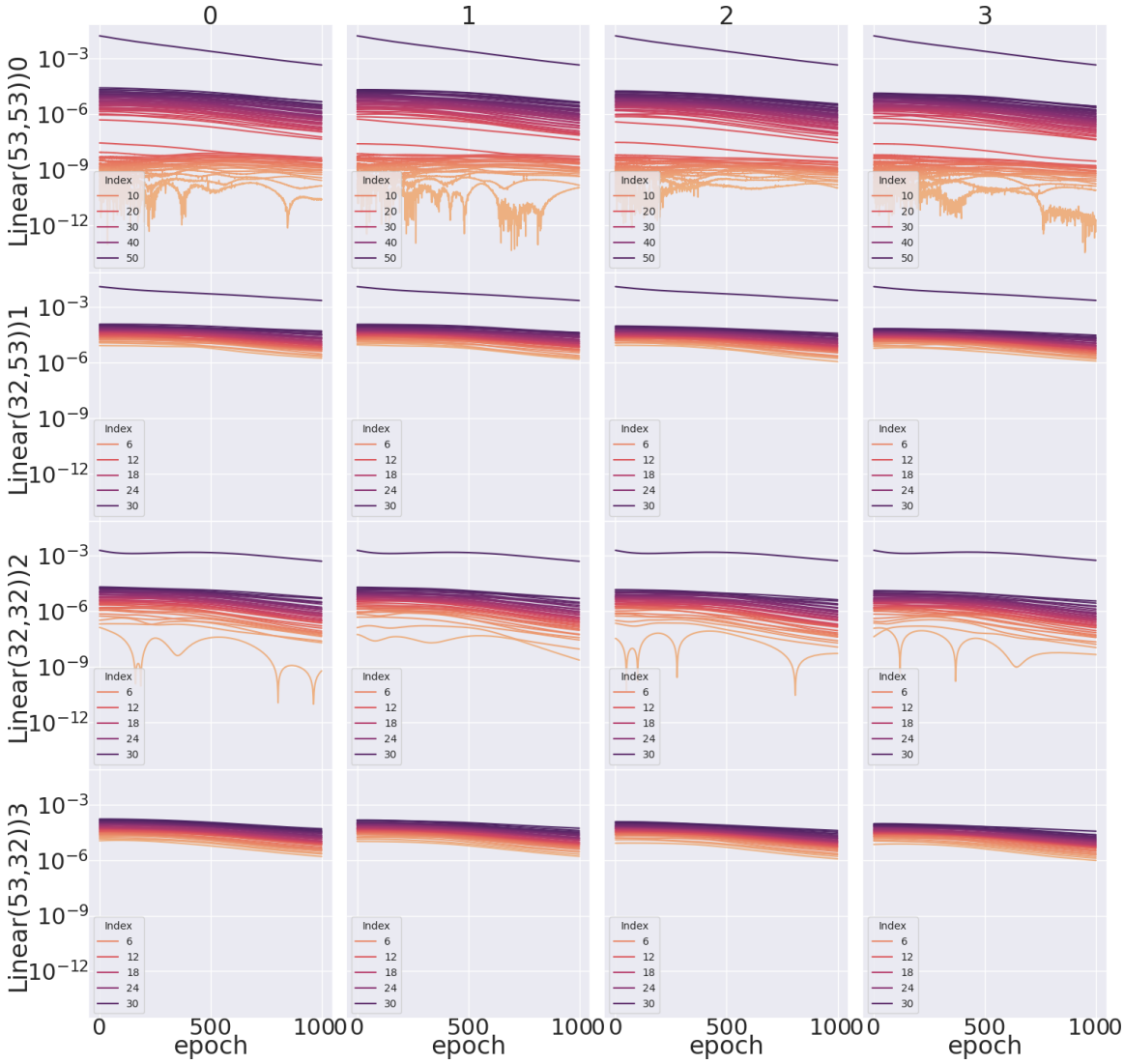
Figure 6.3: Differences in Auto-Differentiation Spectra Dynamics for the first 4 classes in the Sinusoid data set, trained with various architectures and tasks with an RNN



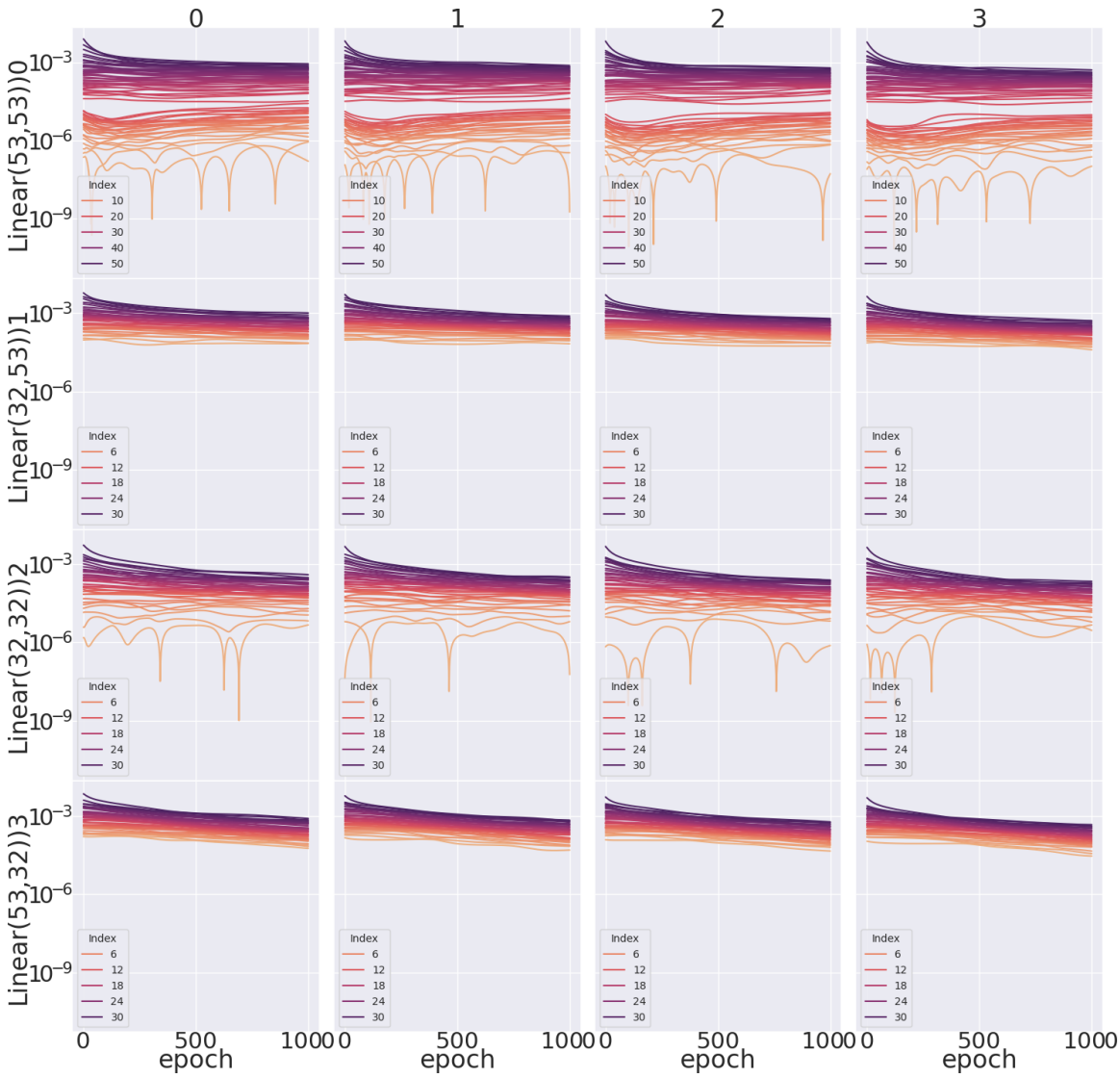

(a) RNN Autoencoder with hidden dim 32 and ReLU activations



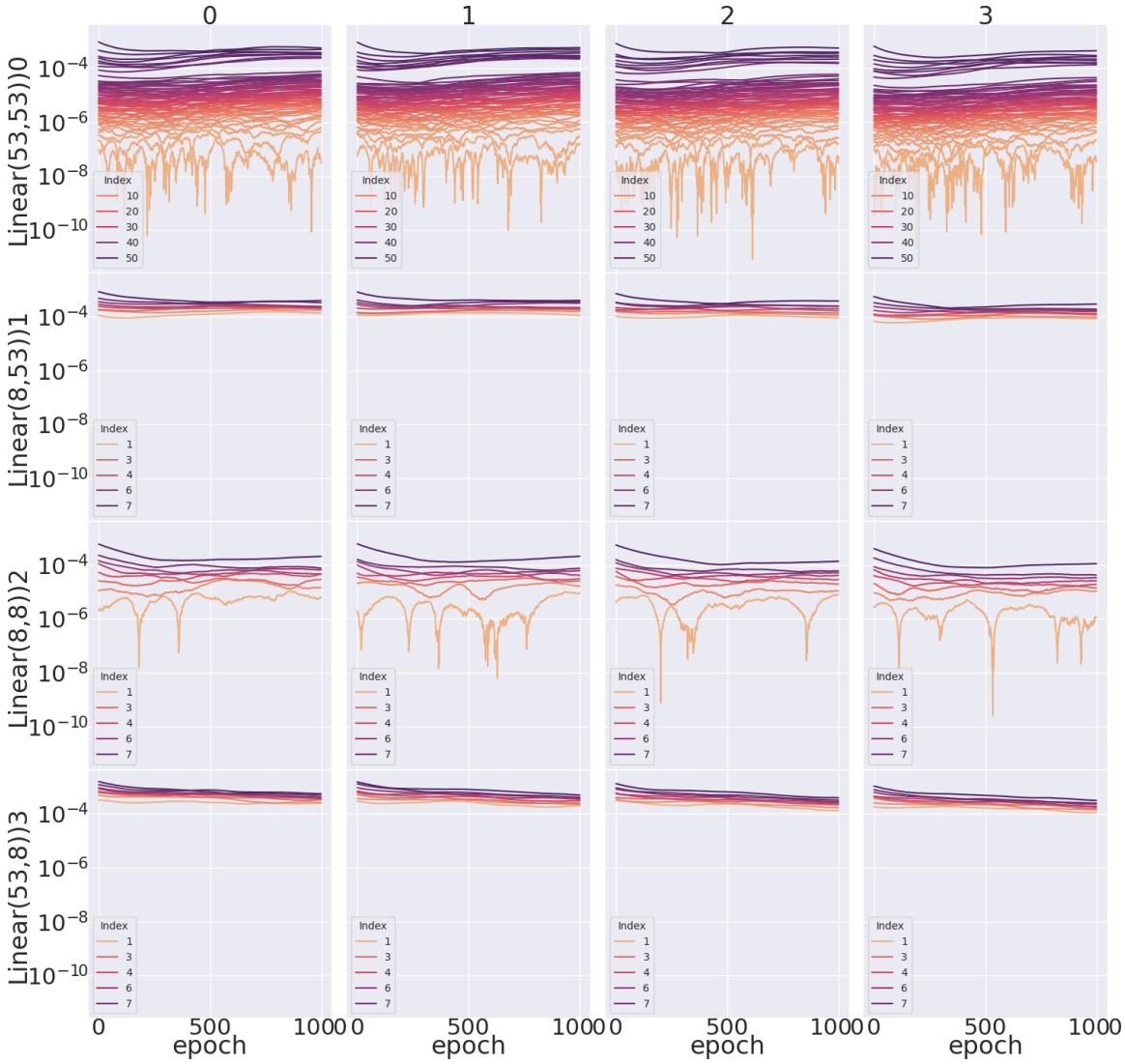

(b) RNN Classifier with hidden dim 32 and ReLU activations



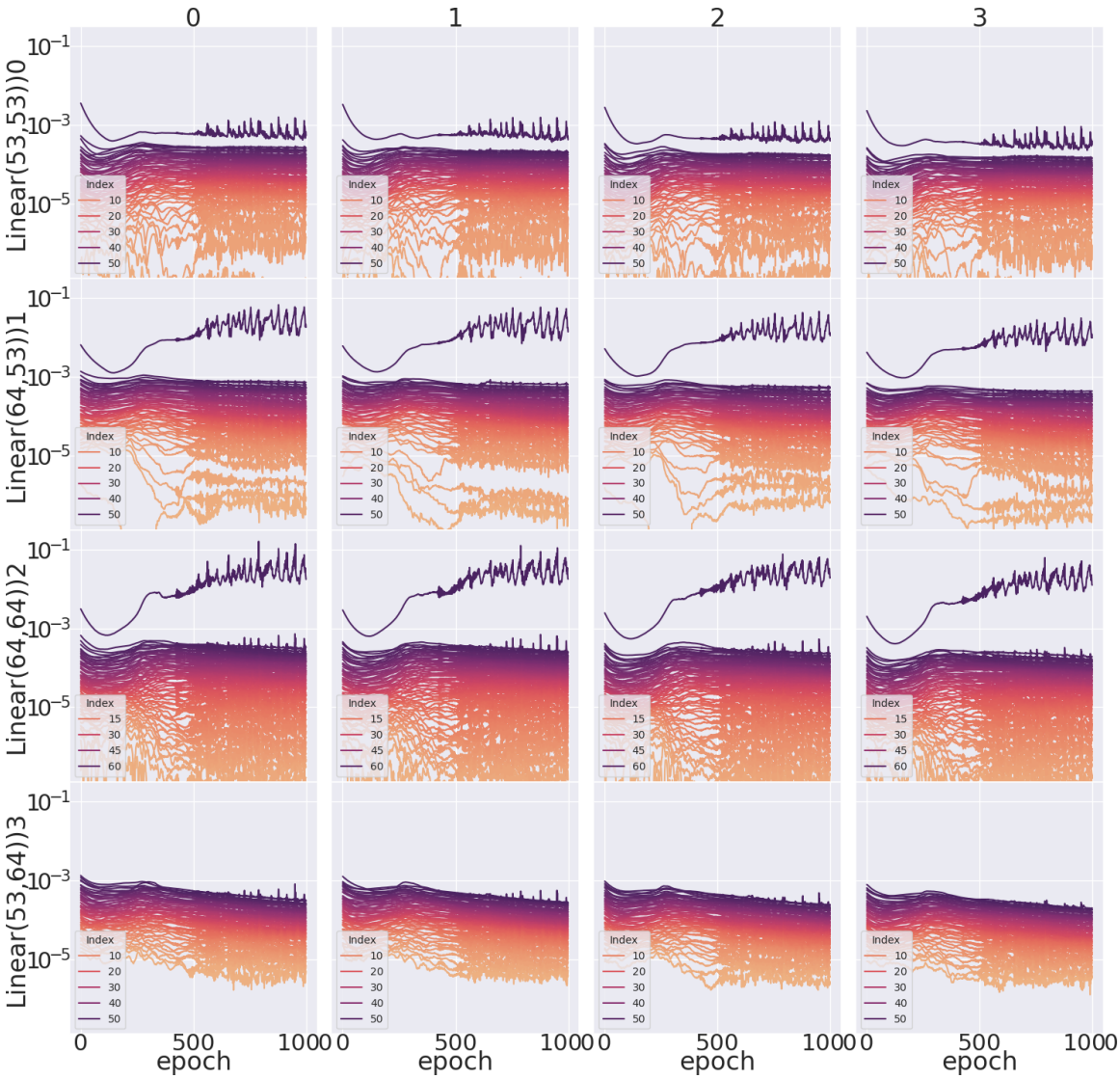

(c) RNN Autoencoder with hidden dim 32 and Sigmoid activations



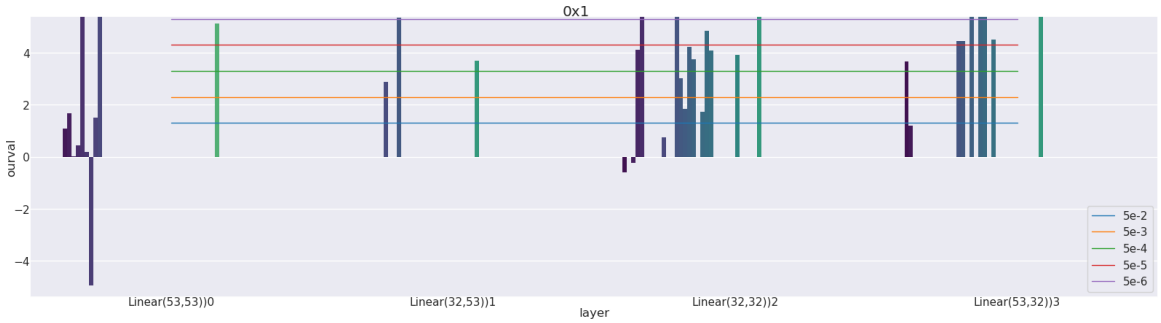

(d) RNN Autoencoder with hidden dim 32 and Tanh activations



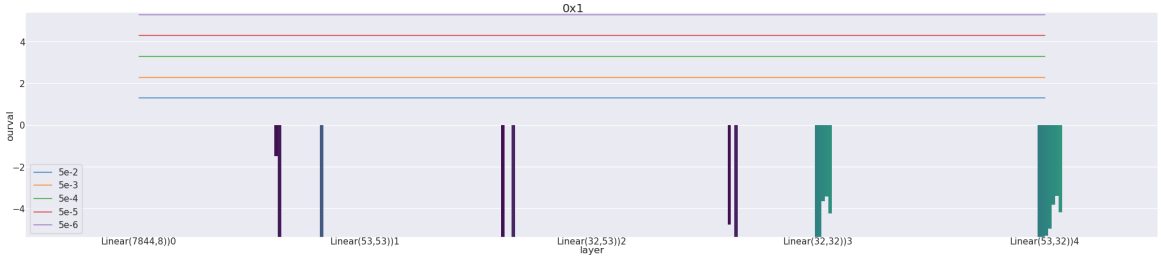

(e) RNN Autoencoder with hidden dim 8 and ReLU activations




(f) RNN Autoencoder with hidden dim 64 and ReLU activations



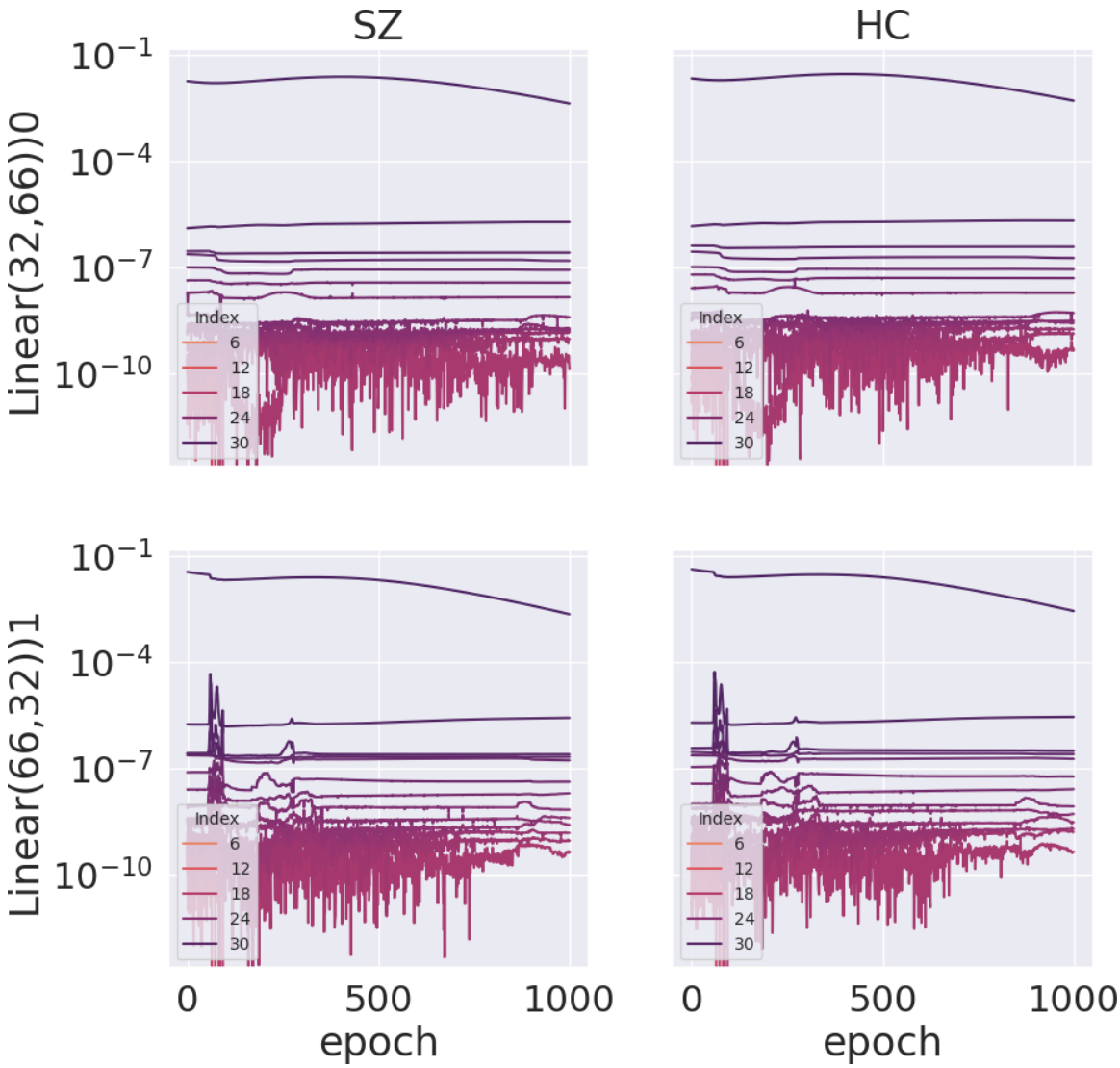

(g) Significant spectral differences between groups 0 and 3: RNN Autoencoder with hidden dim 32 and ReLU activations



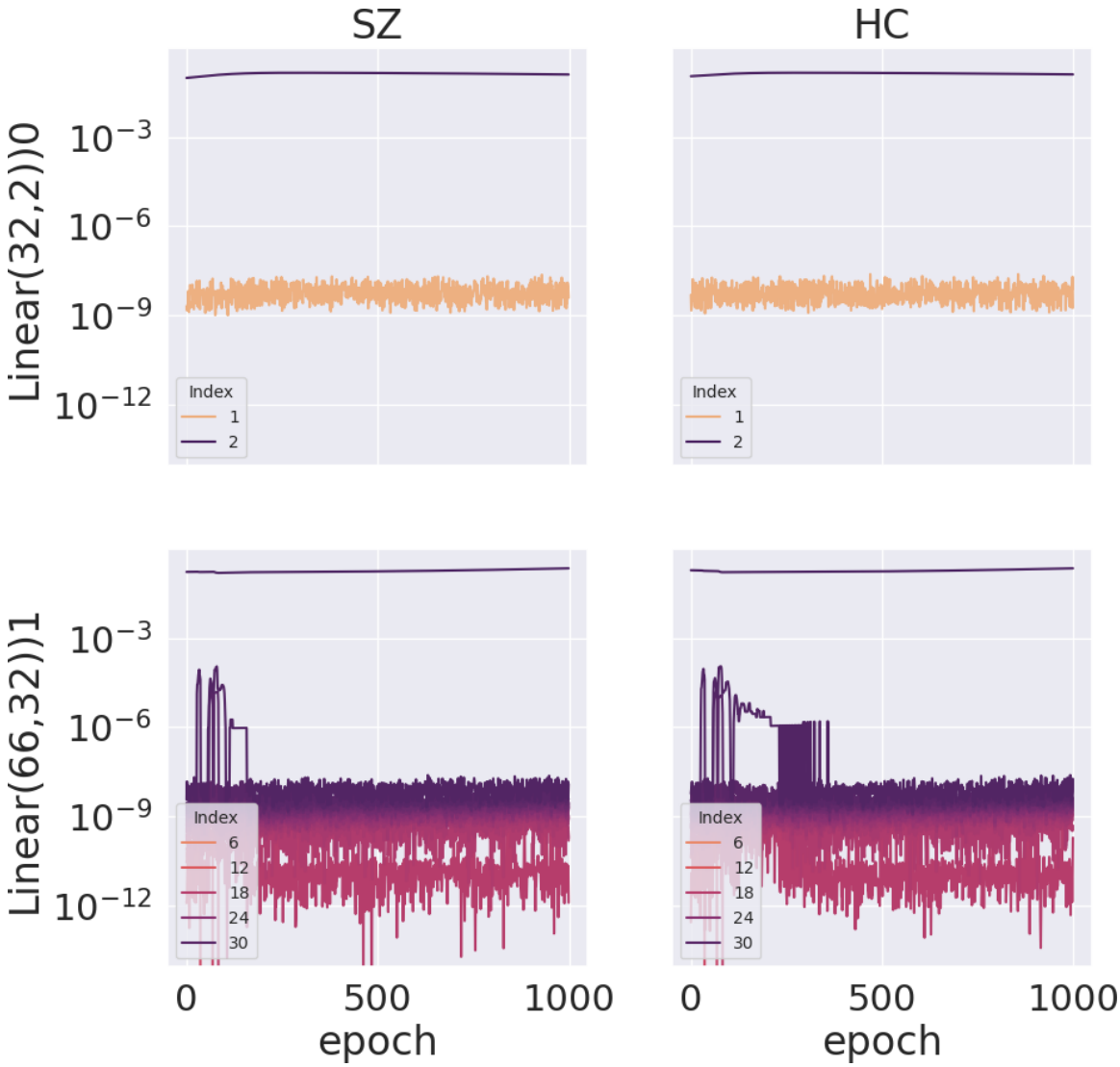

(h) Significant spectral differences between groups 0 and 3: RNN Classifier with hidden dim 32 and ReLU activations

Figure 6.3: Differences in Auto-Differentiation Spectra Dynamics for the first 4 classes in the Sinusoid data set, trained with various architectures and tasks with an RNN

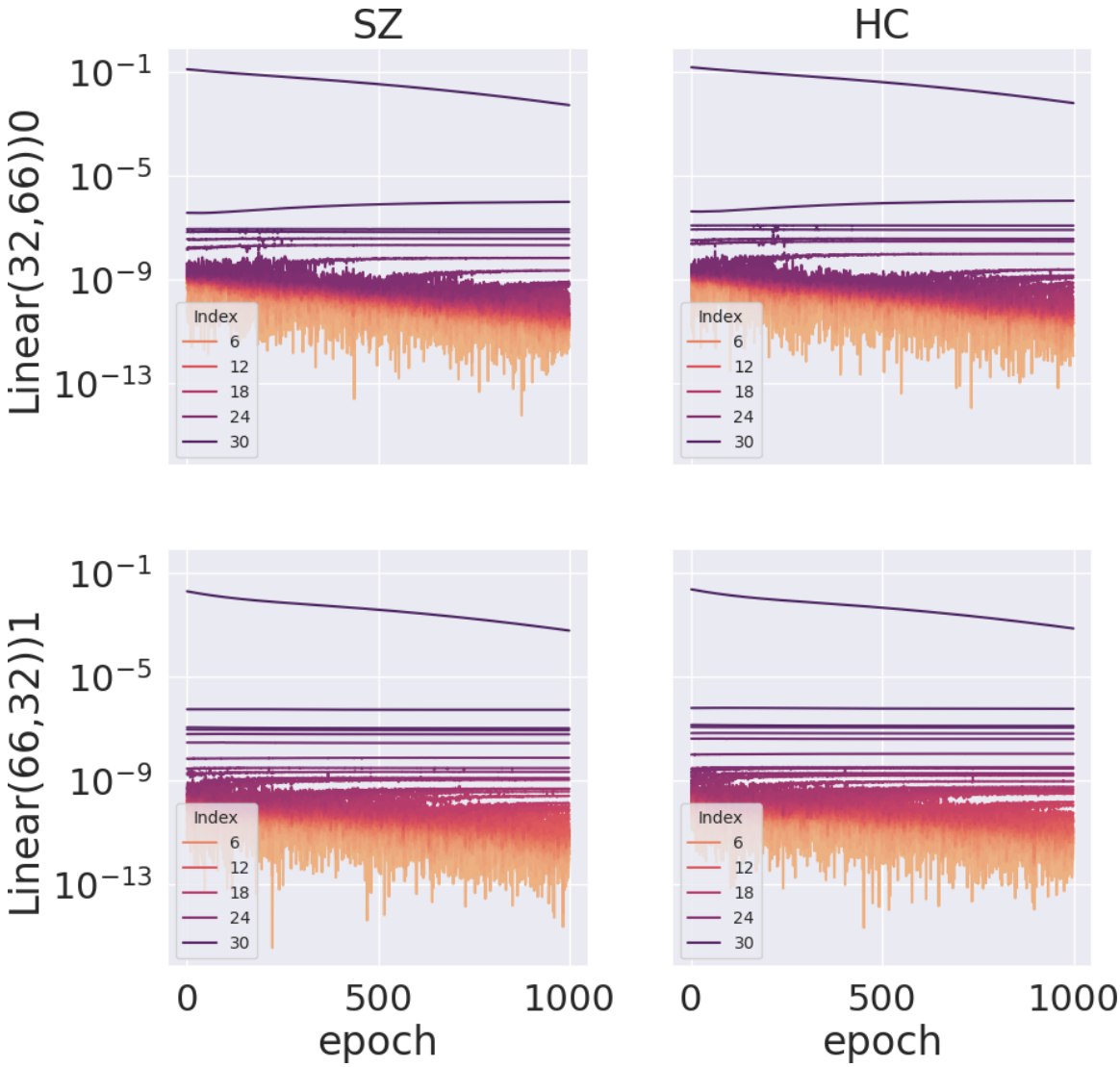
Figure 6.4: Differences in Auto-Differentiation Spectra Dynamics on the FSL data set, trained with various architectures and tasks with a Multi-Layer Perceptron.



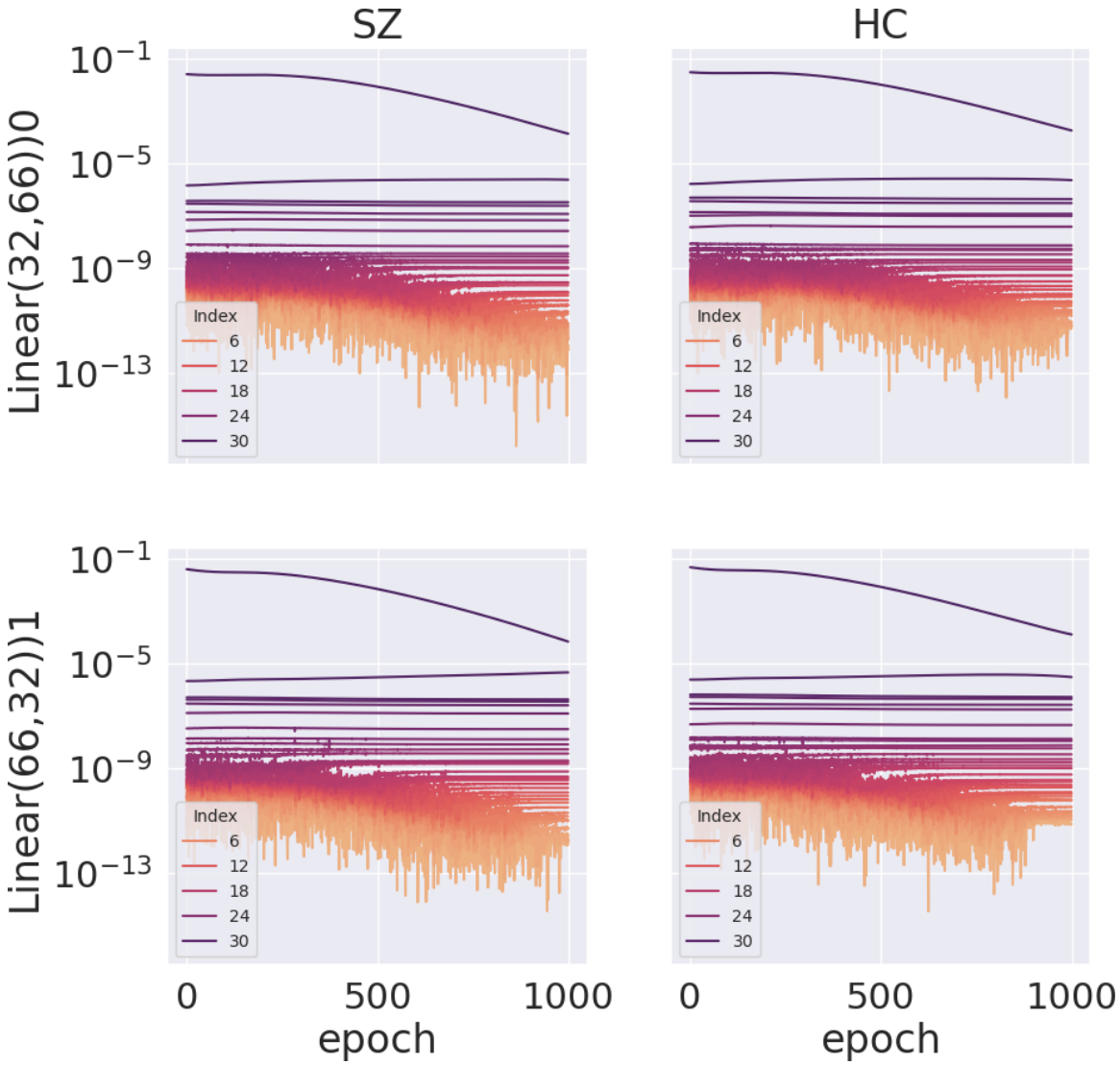

(a) MLP Autoencoder with hidden dim 32 and ReLU activations



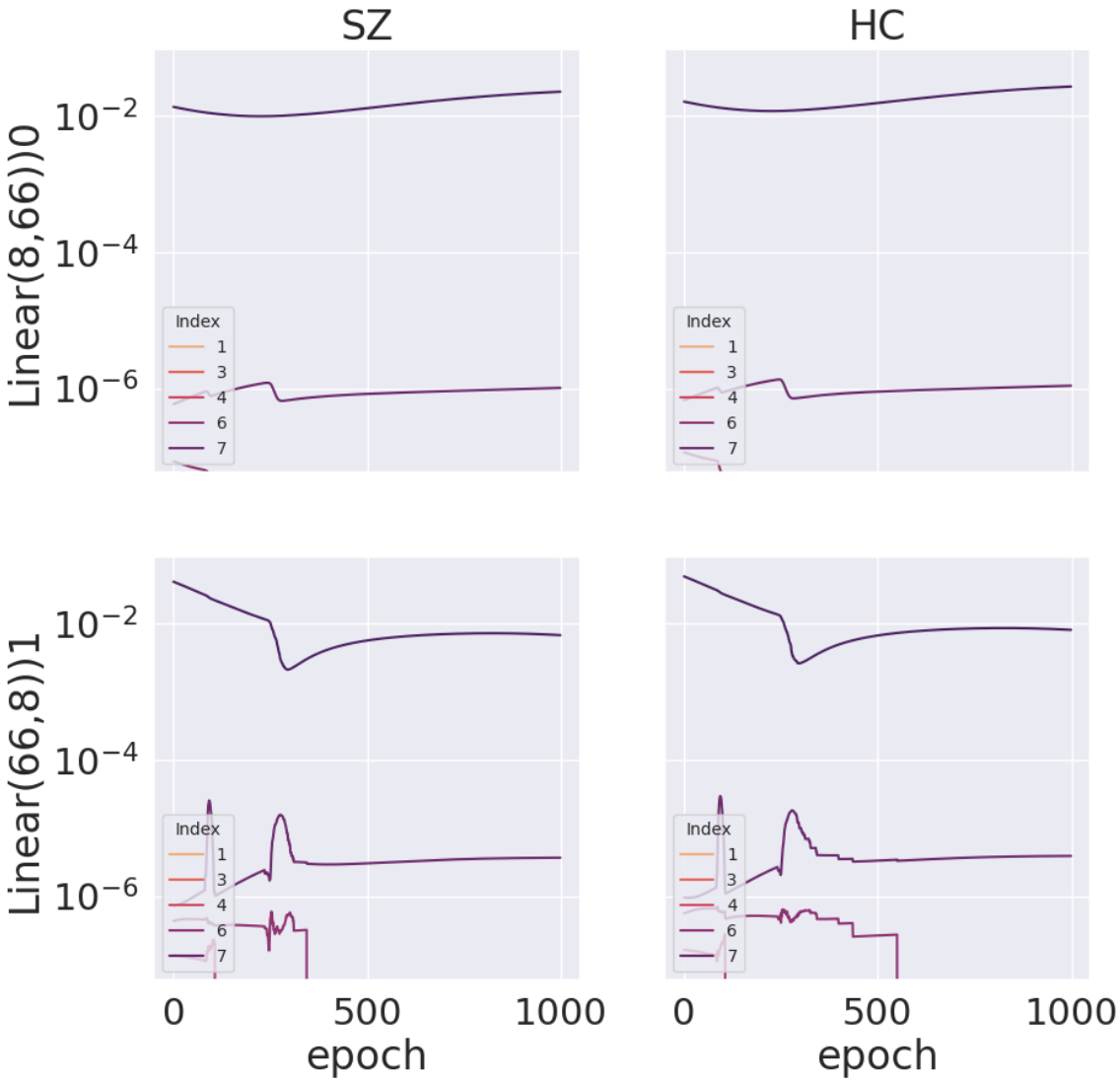

(b) MLP Classifier with hidden dim 32 and ReLU activations



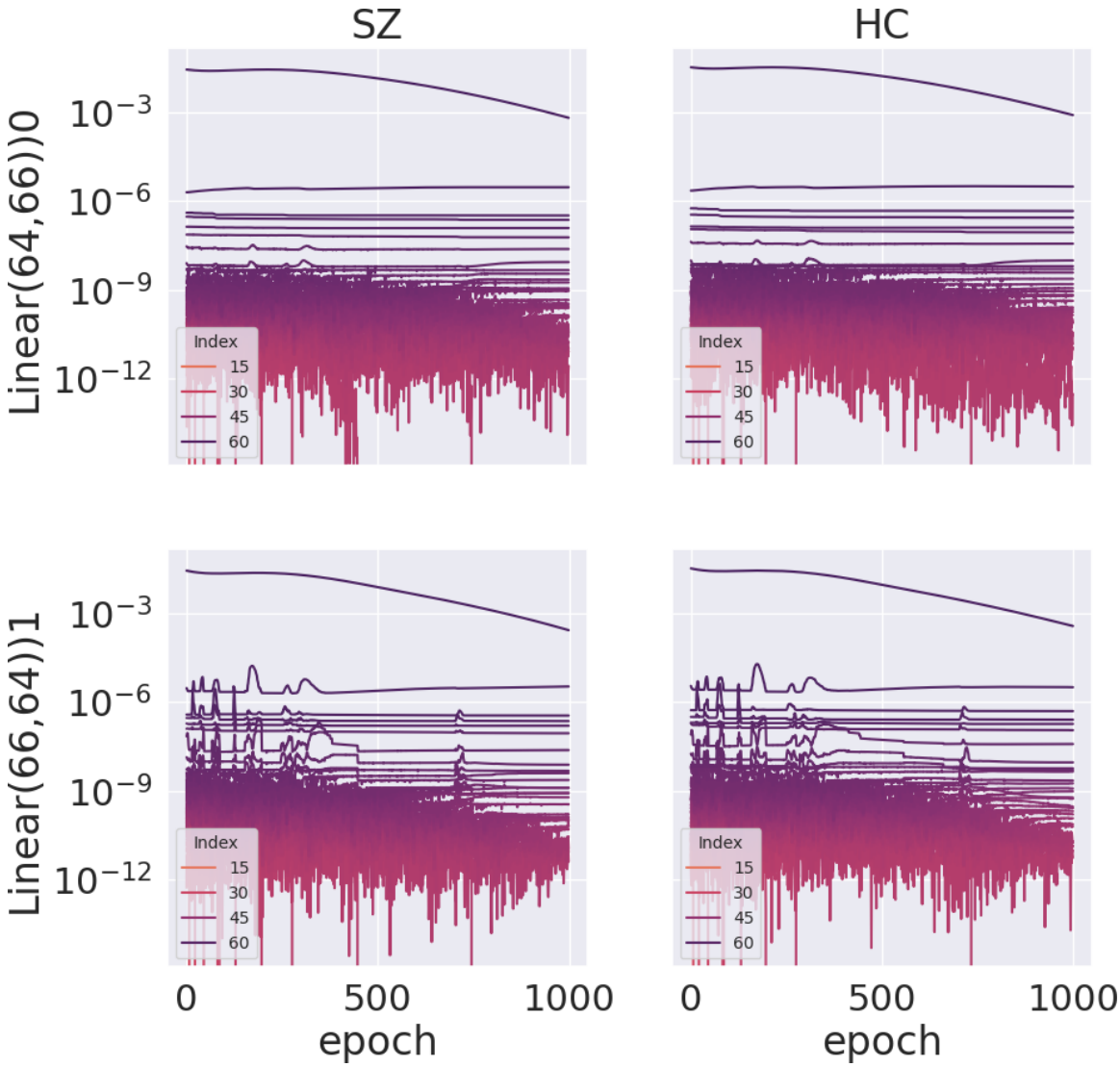

(c) MLP Autoencoder with hidden dim 32 and Sigmoid activations



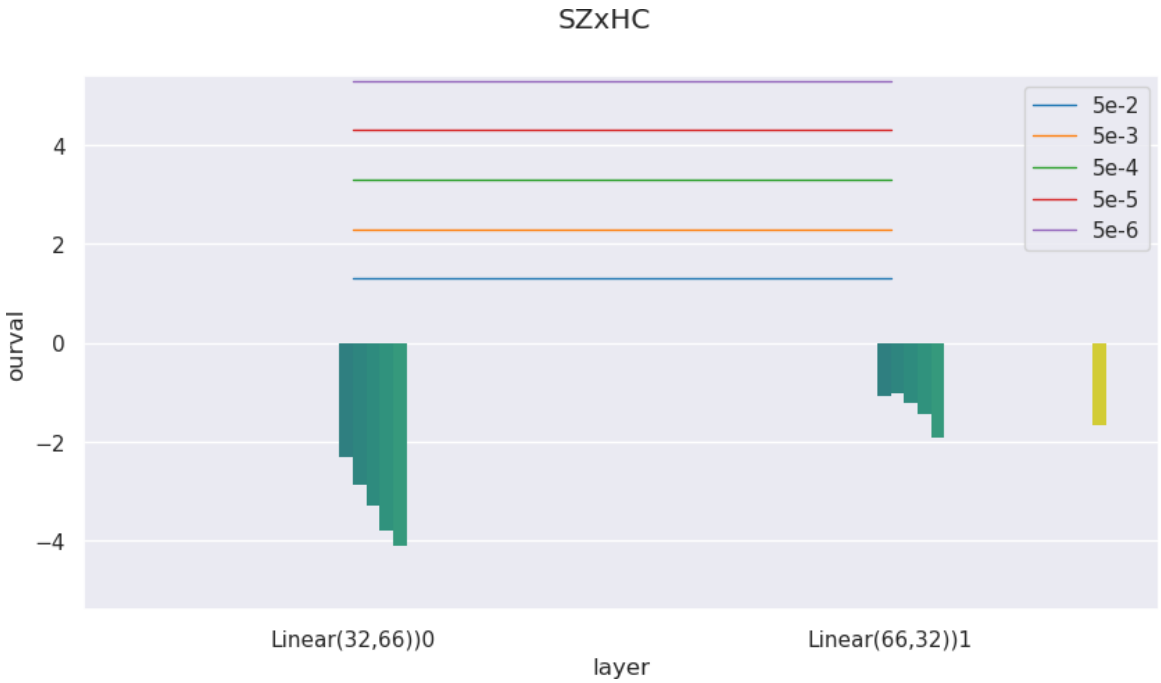

(d) MLP Autoencoder with hidden dim 32 and Tanh activations



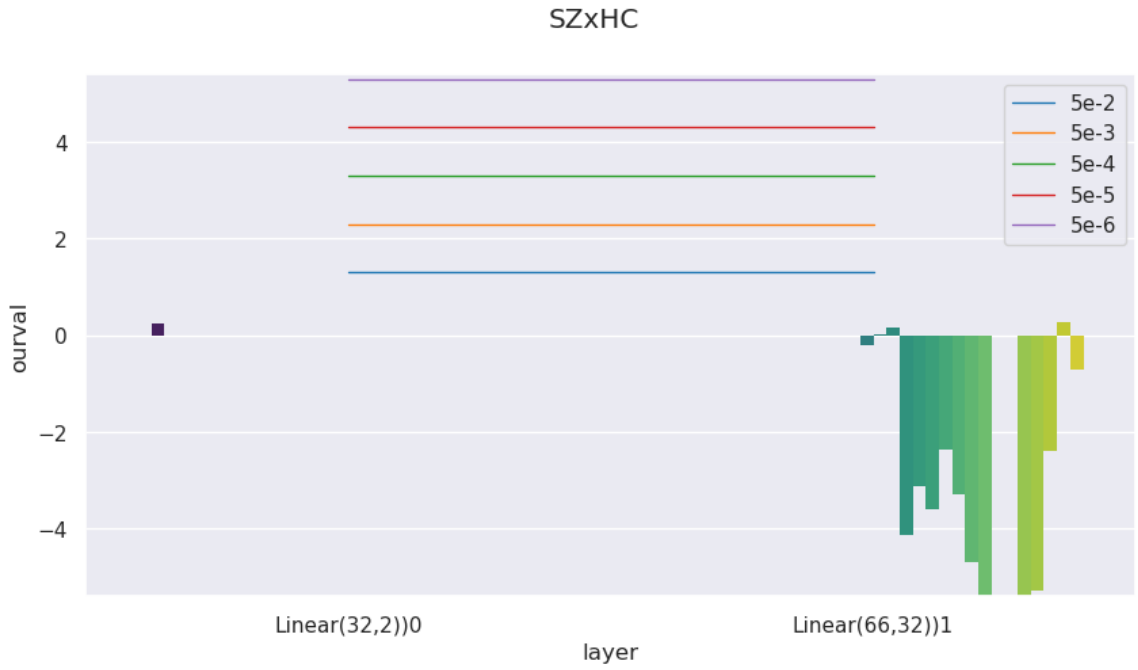

(e) MLP Autoencoder with hidden dim 8 and ReLU activations




(f) MLP Autoencoder with hidden dim 64 and ReLU activations



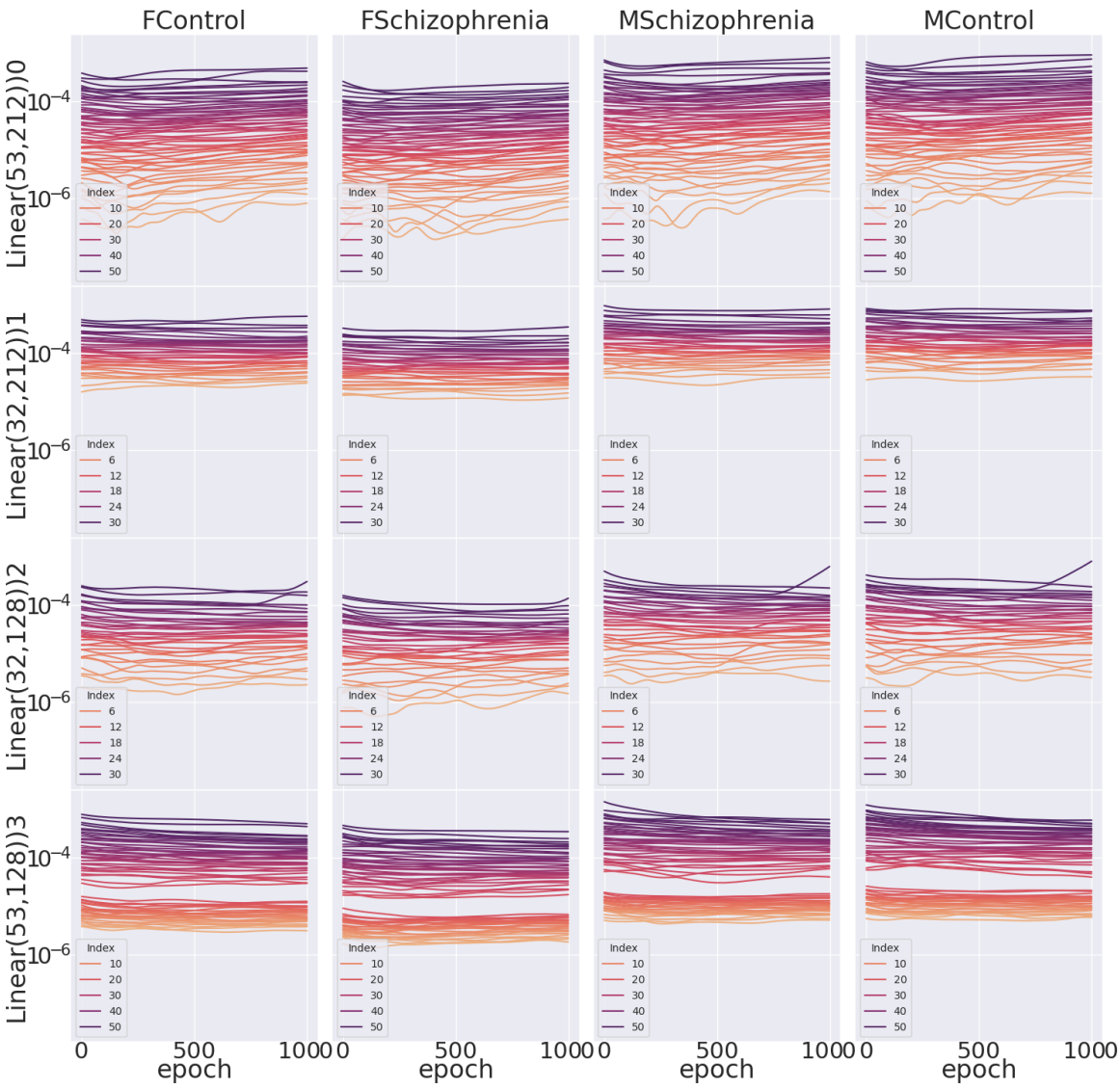

(g) Significant spectral differences between groups SZ and HC: MLP Autoencoder with hidden dim 32 and ReLU activations



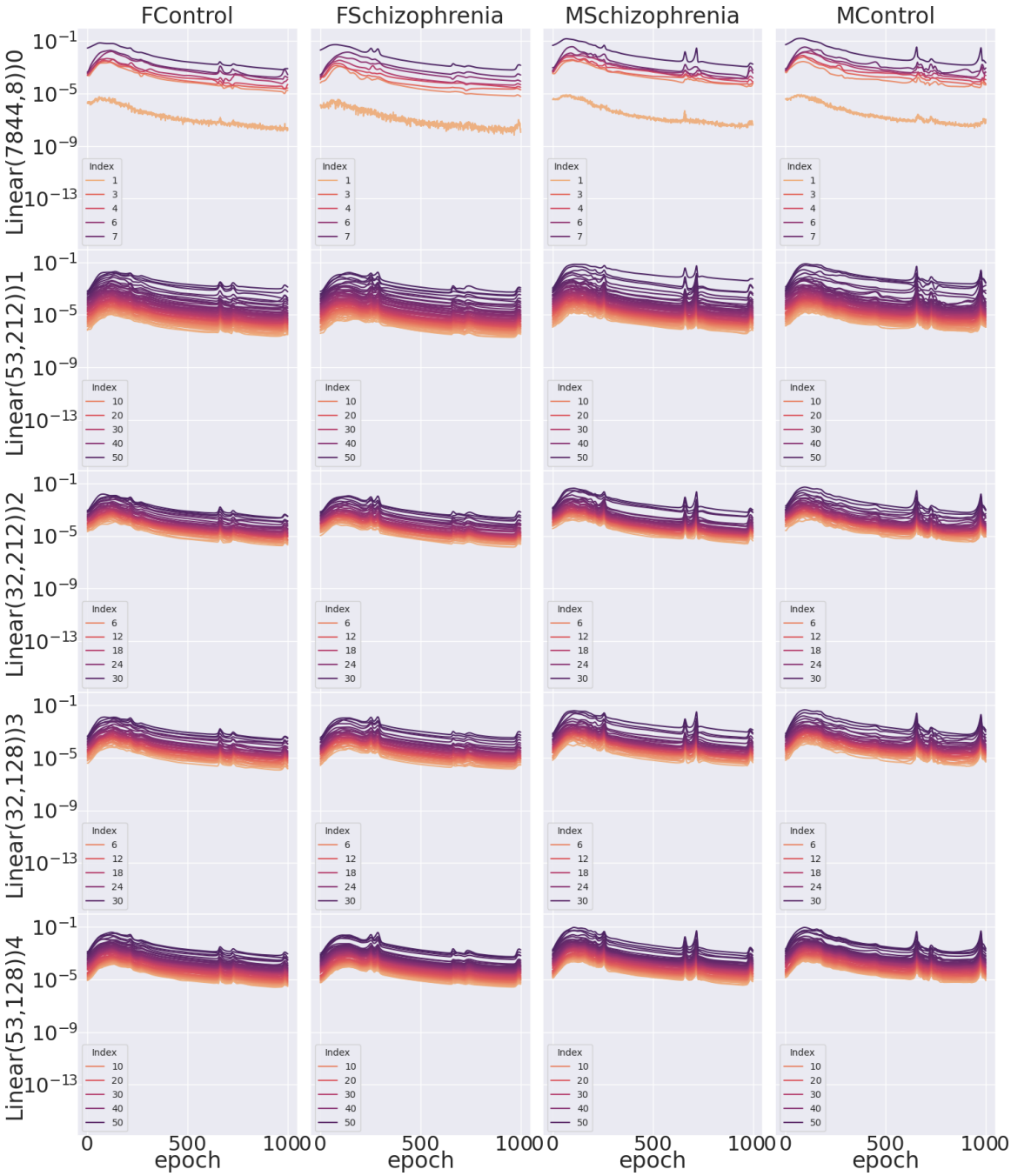

(h) Significant spectral differences between groups SZ and HC: MLP Classifier with hidden dim 32 and ReLU activations

Figure 6.4: Differences in Auto-Differentiation Spectra Dynamics on the FSL data set, trained with various architectures and tasks with a Multi-Layer Perceptron.

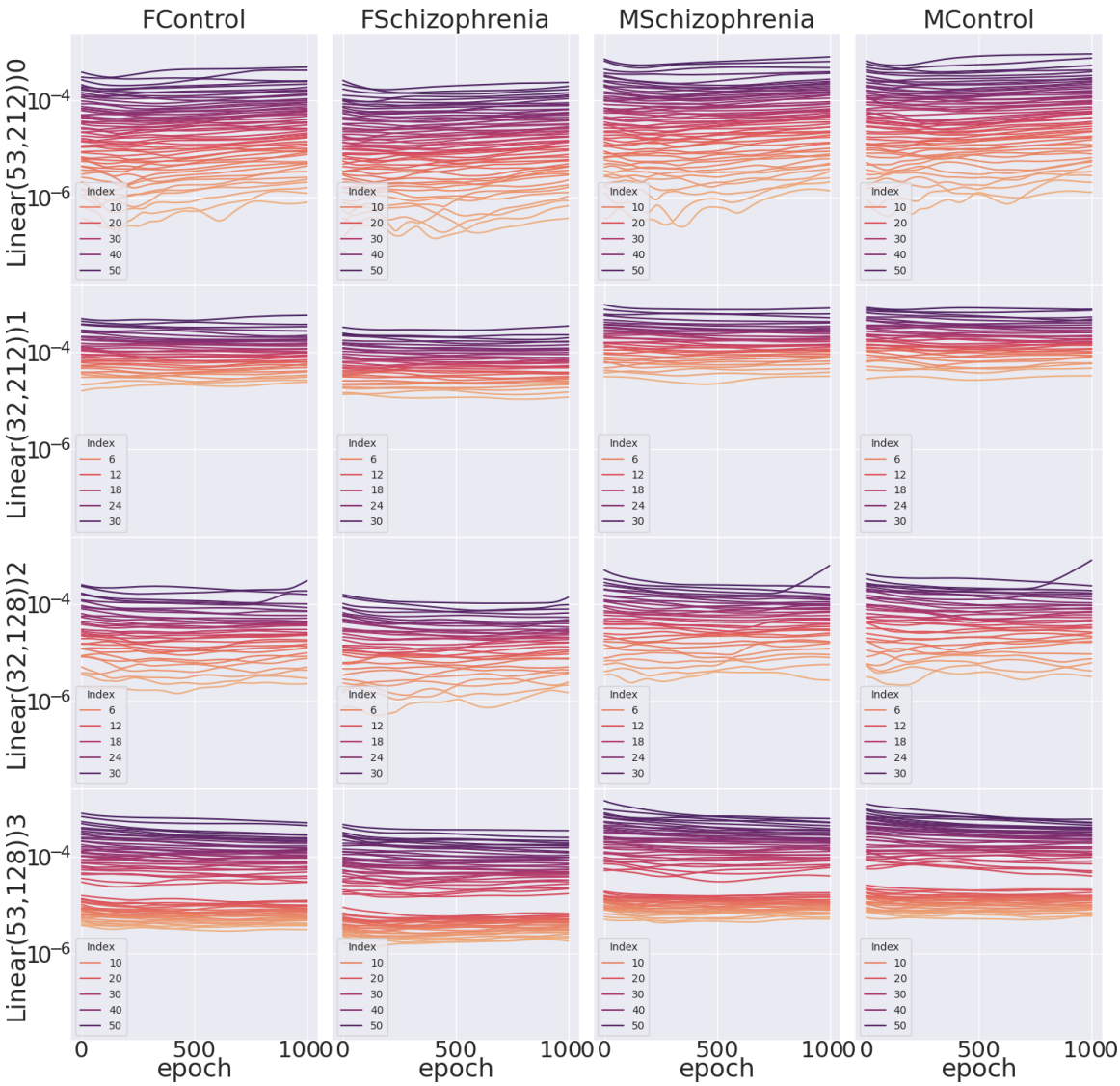
Figure 6.5: Differences in Auto-Differentiation Spectra Dynamics on the COBRE data set, trained with various architectures and tasks with an LSTM.



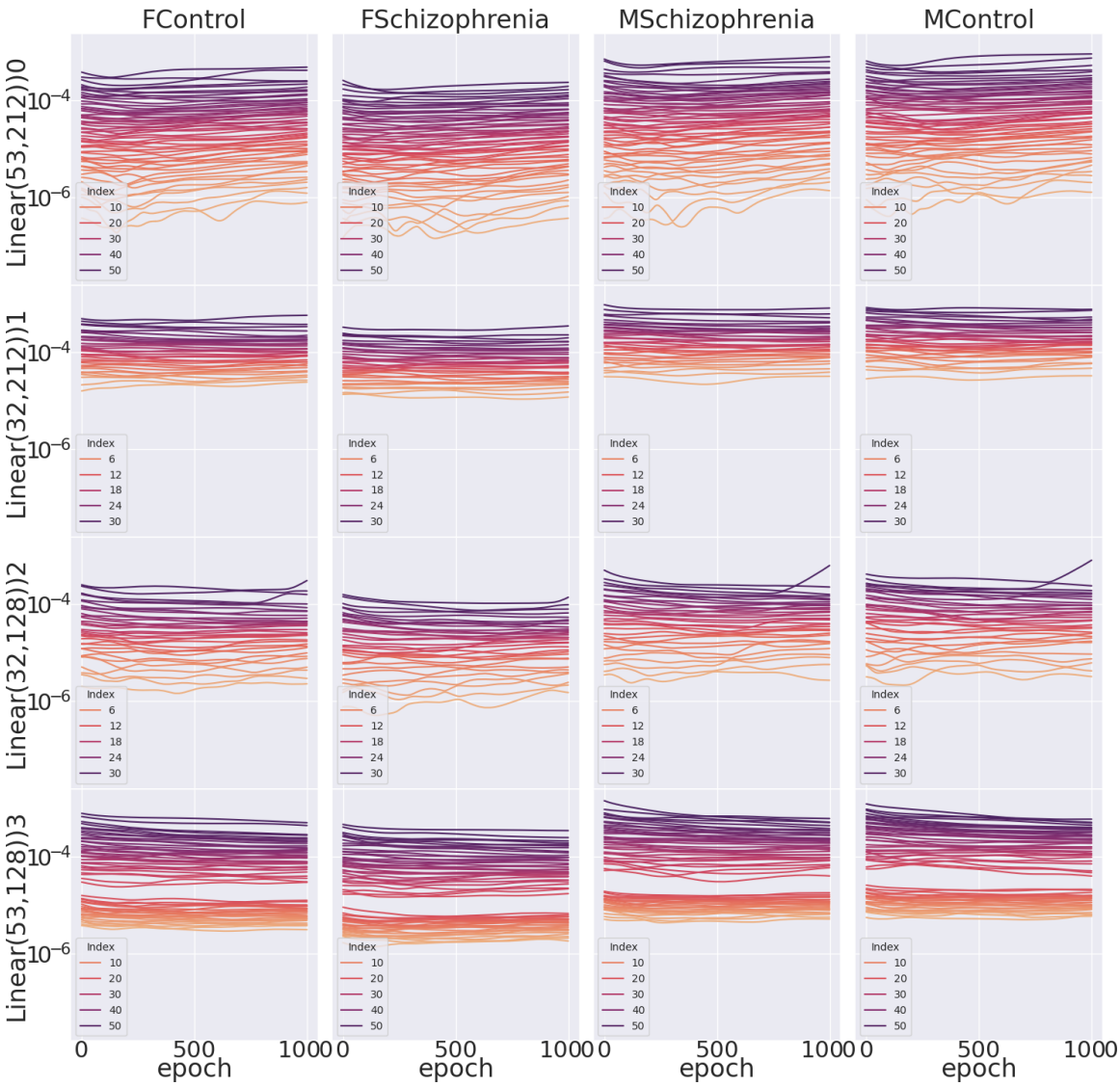

(a) LSTM Autoencoder with hidden dim 32 and ReLU activations



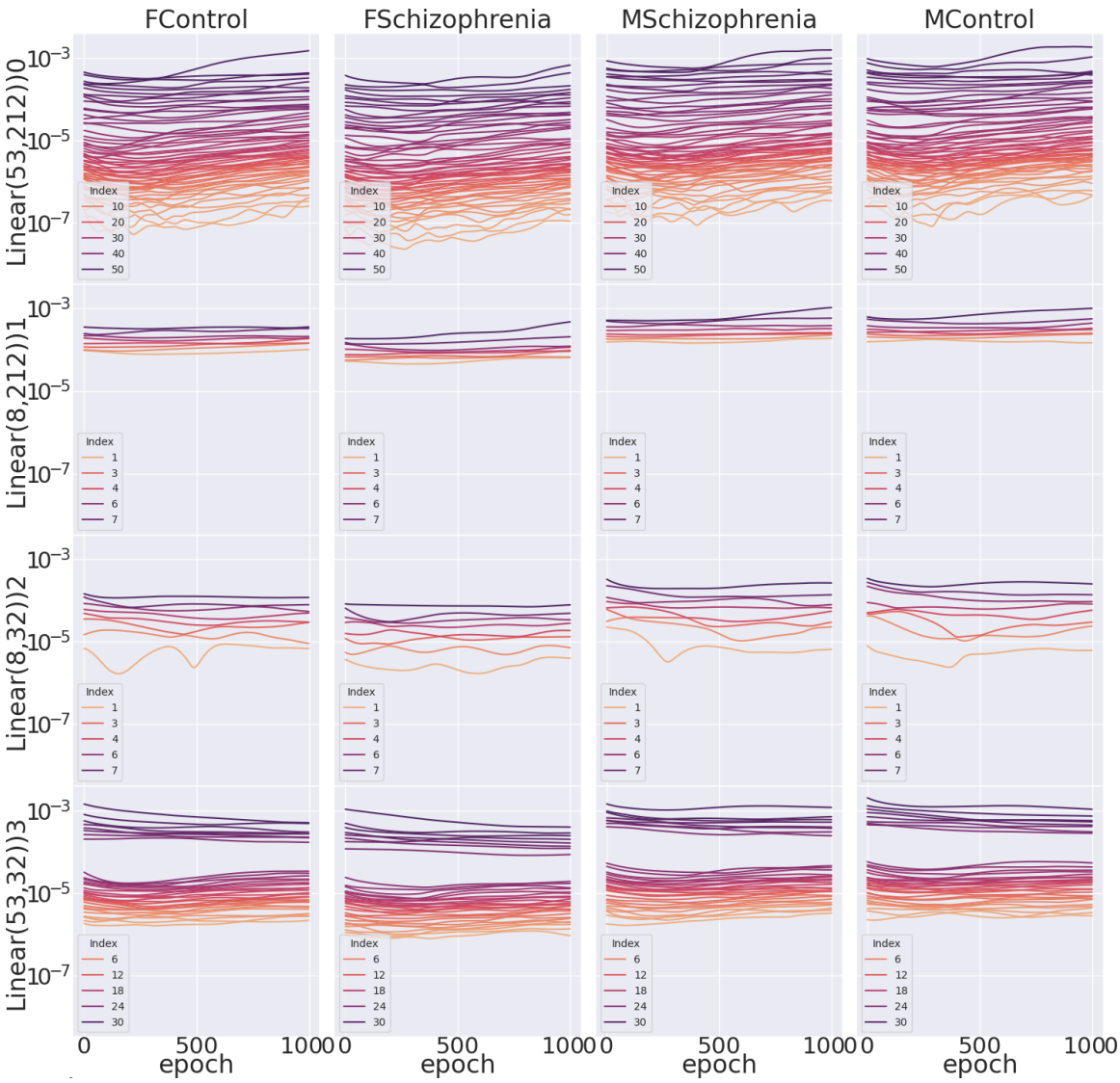

(b) LSTM Classifier with hidden dim 32 and ReLU activations



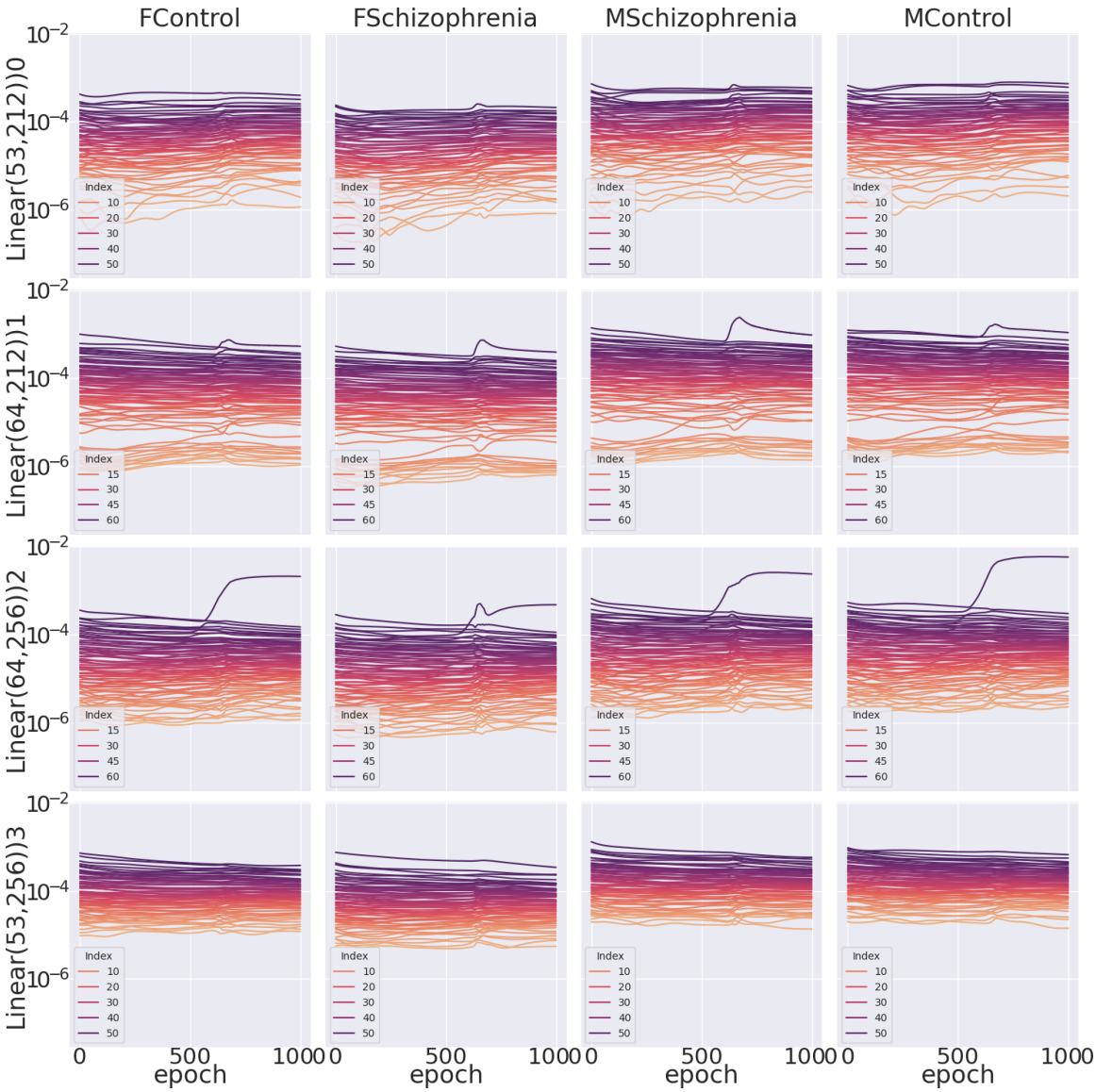

(c) LSTM Autoencoder with hidden dim 32 and Sigmoid activations



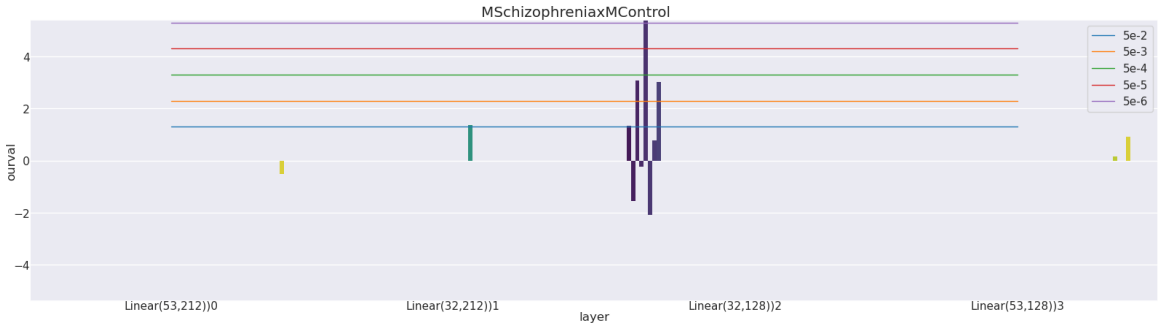

(d) LSTM Autoencoder with hidden dim 32 and Tanh activations



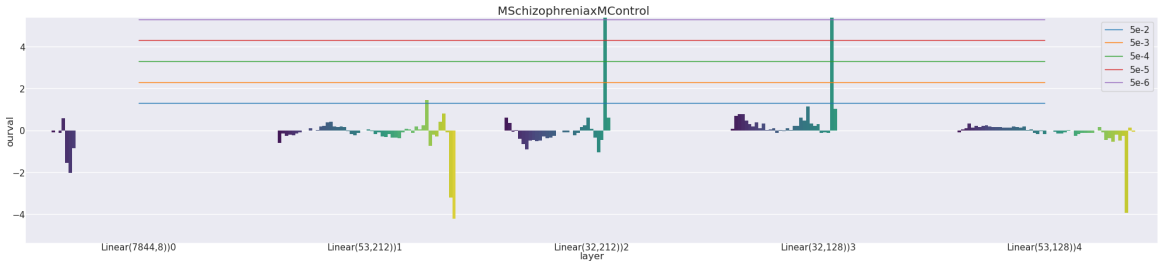

(e) LSTM Autoencoder with hidden dim 8 and ReLU activations




(f) LSTM Autoencoder with hidden dim 64 and ReLU activations



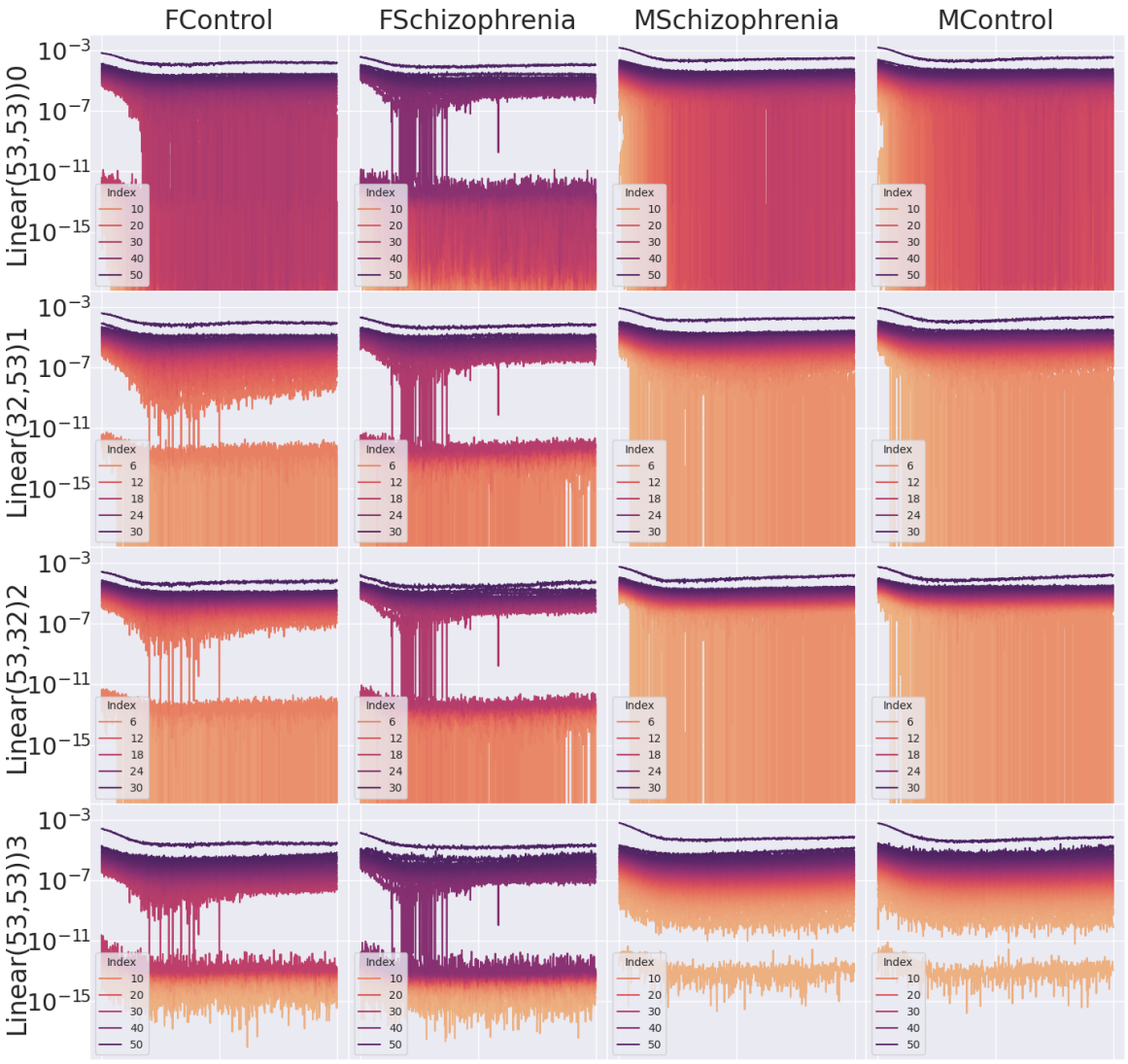

(g) Significant spectral differences between HC, SZ and Sex: LSTM Autoencoder with hidden dim 32 and ReLU activations



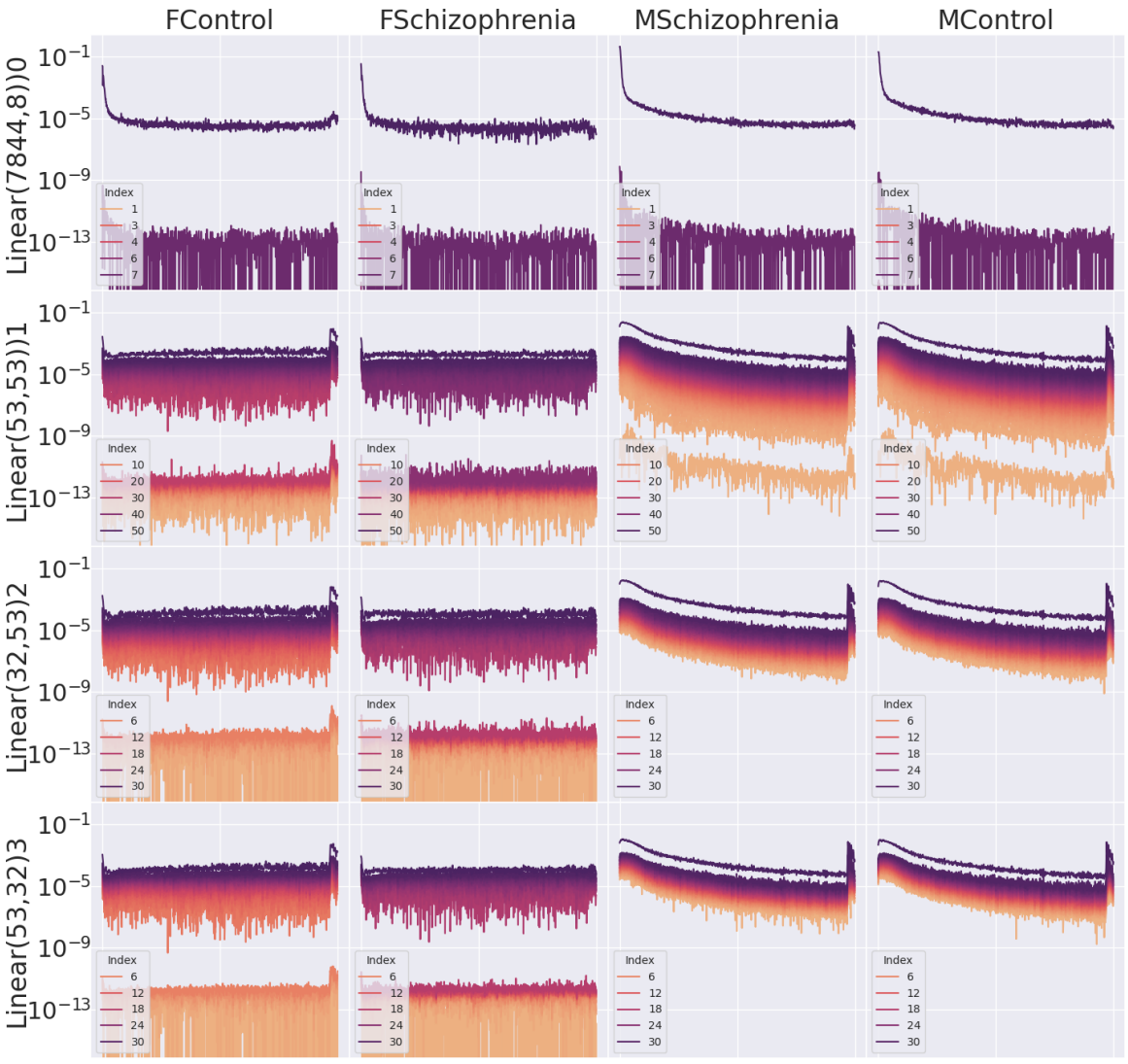

(h) Significant spectral differences between HC, SZ and Sex: LSTM Classifier with hidden dim 32 and ReLU activations

Figure 6.5: Differences in Auto-Differentiation Spectra Dynamics on the COBRE data set, trained with various architectures and tasks with an LSTM.

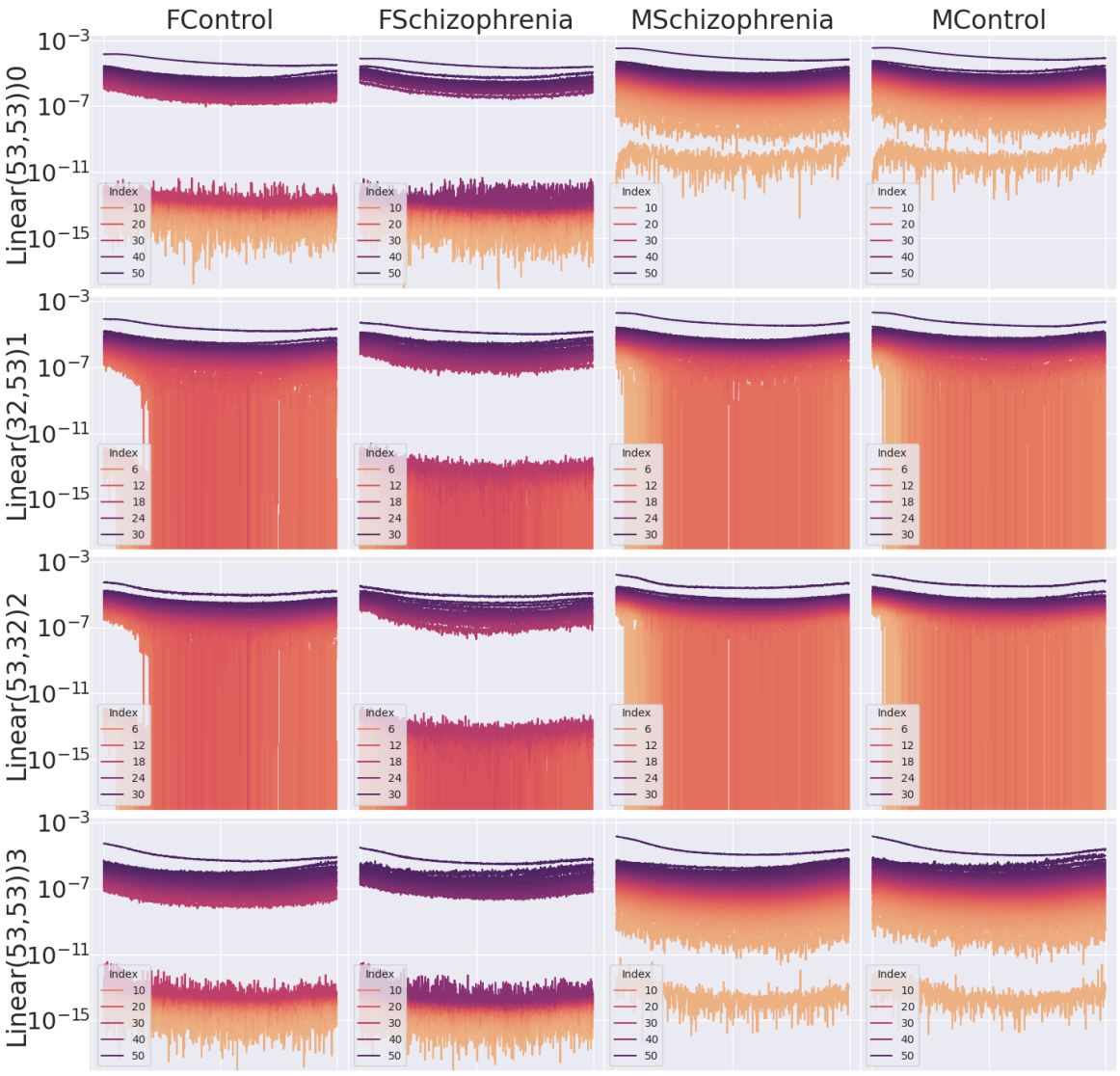
Figure 6.6: Differences in Auto-Differentiation Spectra Dynamics on the COBRE data set, trained with various architectures and tasks with a BERT Transformer.



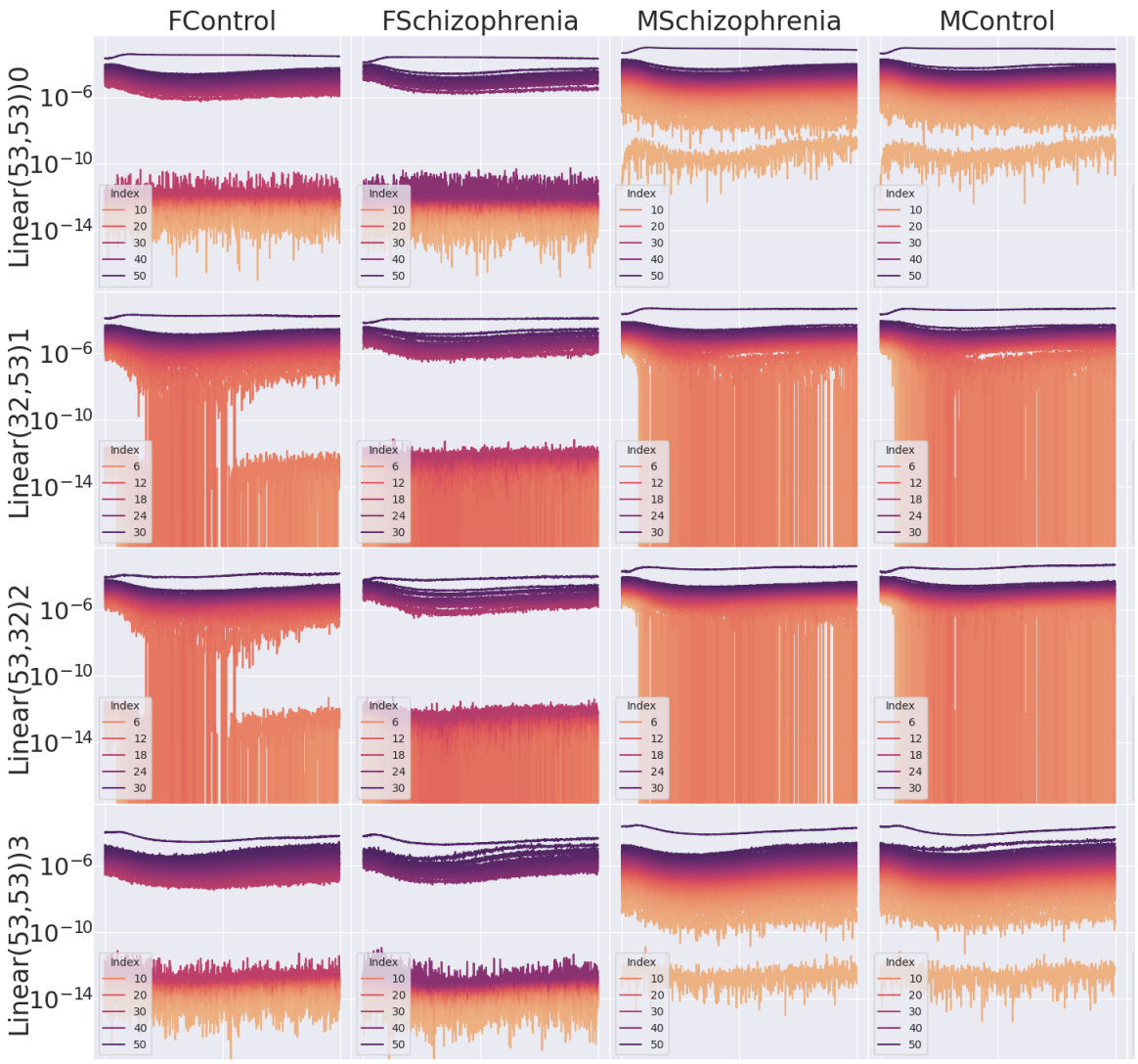

(a) BERT Autoencoder with hidden dim 32 and ReLU activations



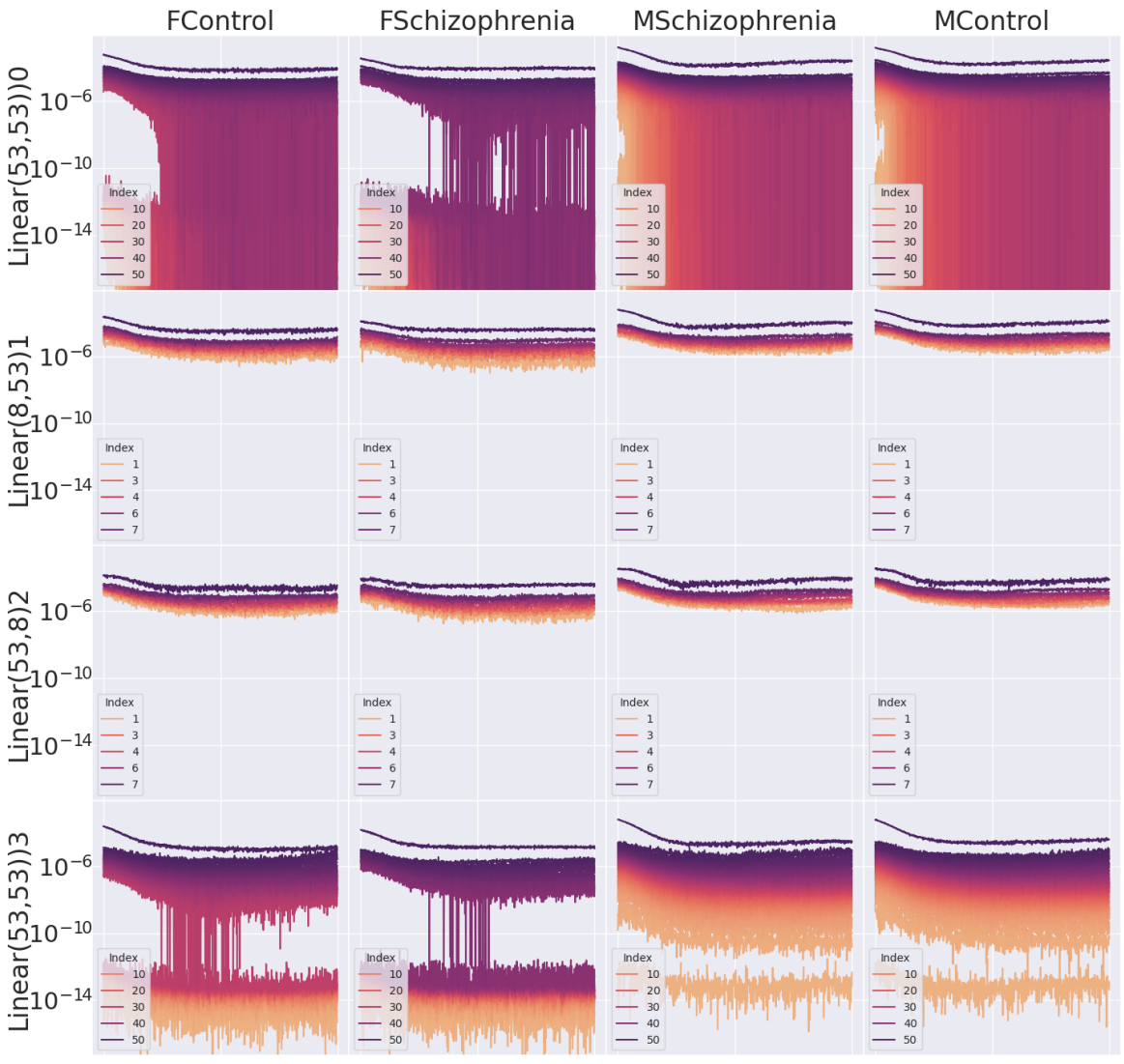

(b) BERT Classifier with hidden dim 32 and ReLU activations



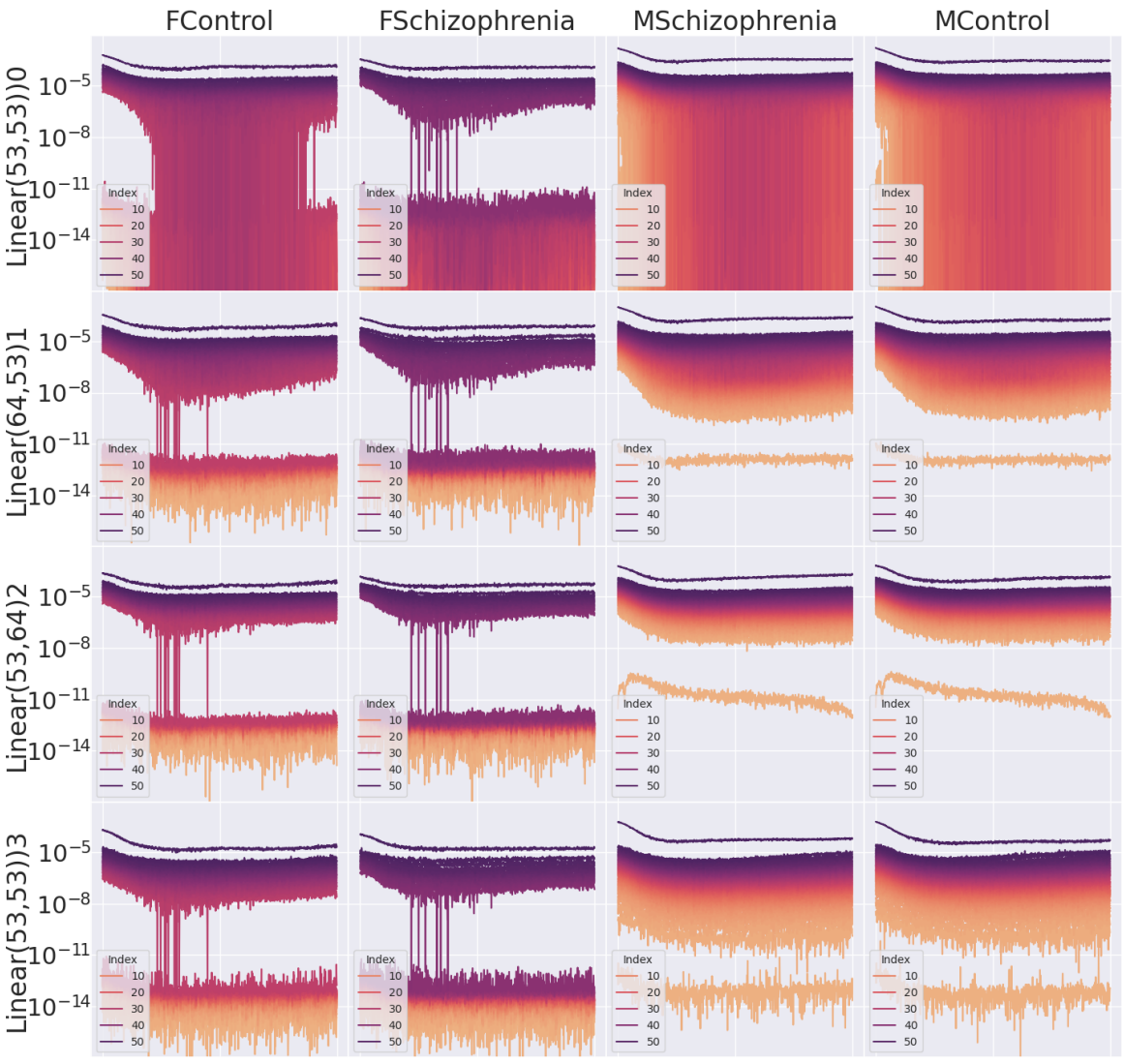

(c) BERT Autoencoder with hidden dim 32 and Sigmoid activations



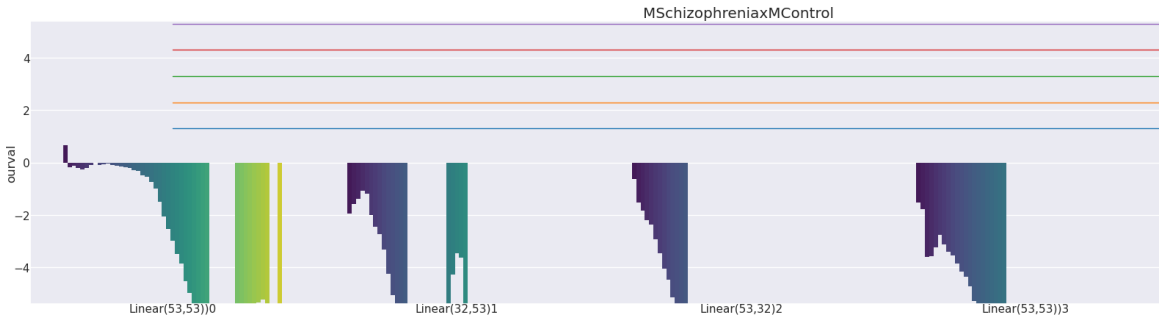

(d) BERT Autoencoder with hidden dim 32 and Tanh activations



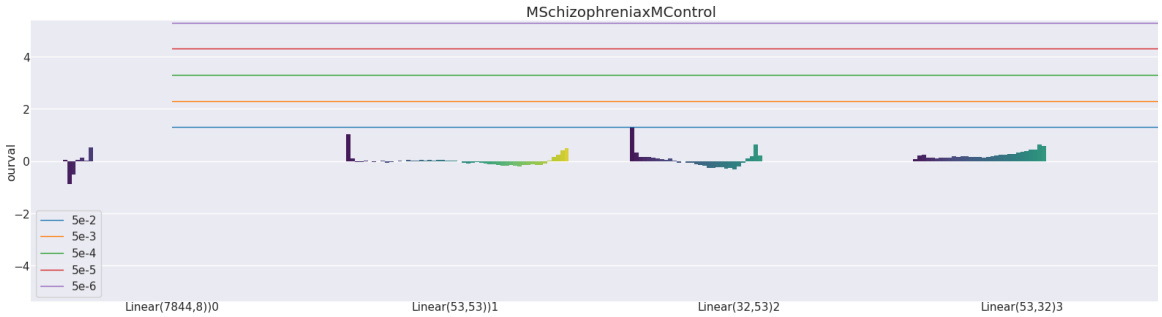

(e) BERT Autoencoder with hidden dim 8 and ReLU activations




(f) BERT Autoencoder with hidden dim 64 and ReLU activations



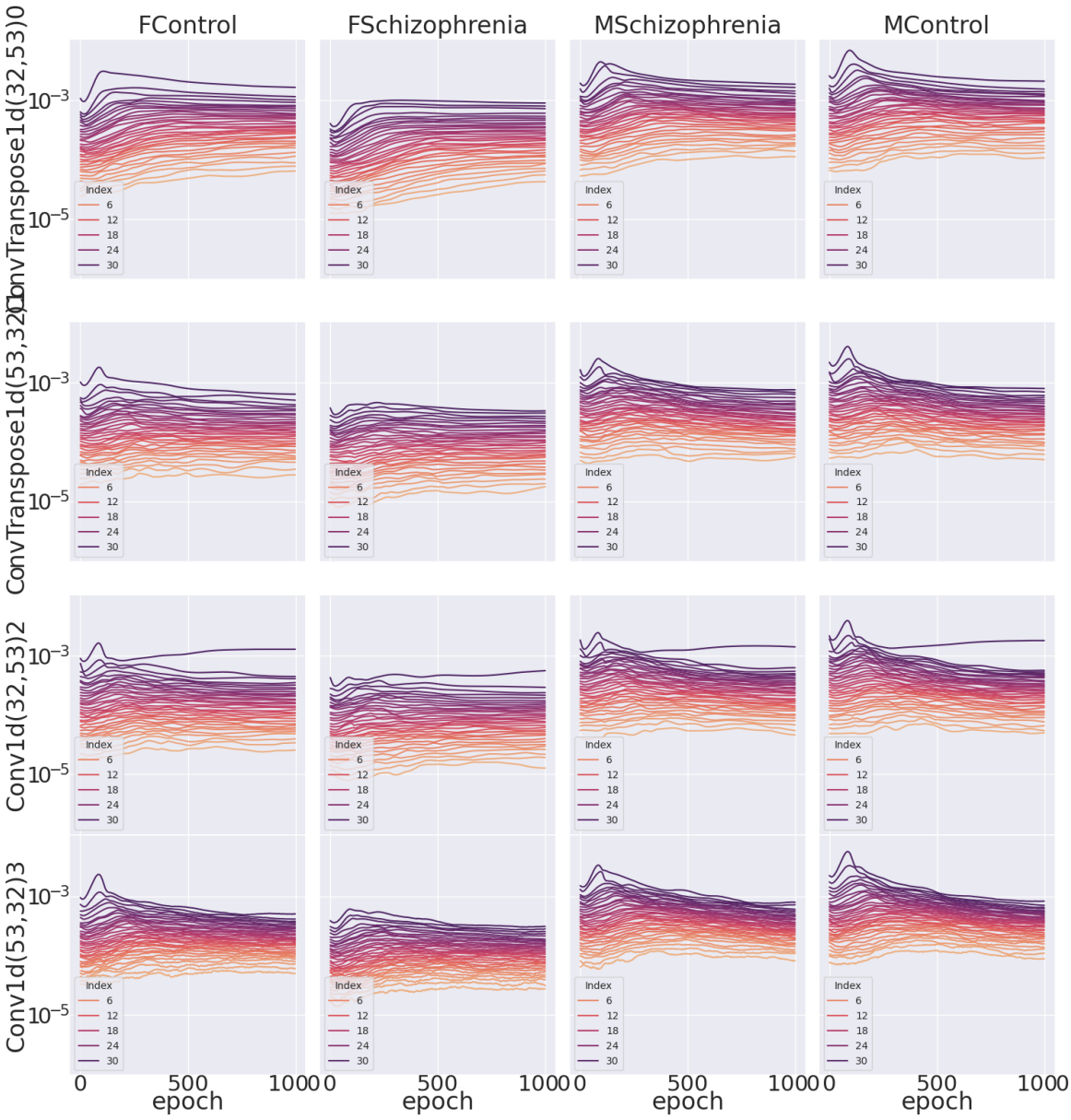

(g) Significant spectral differences between HC, SZ and Sex: BERT Autoencoder with hidden dim 32 and ReLU activations



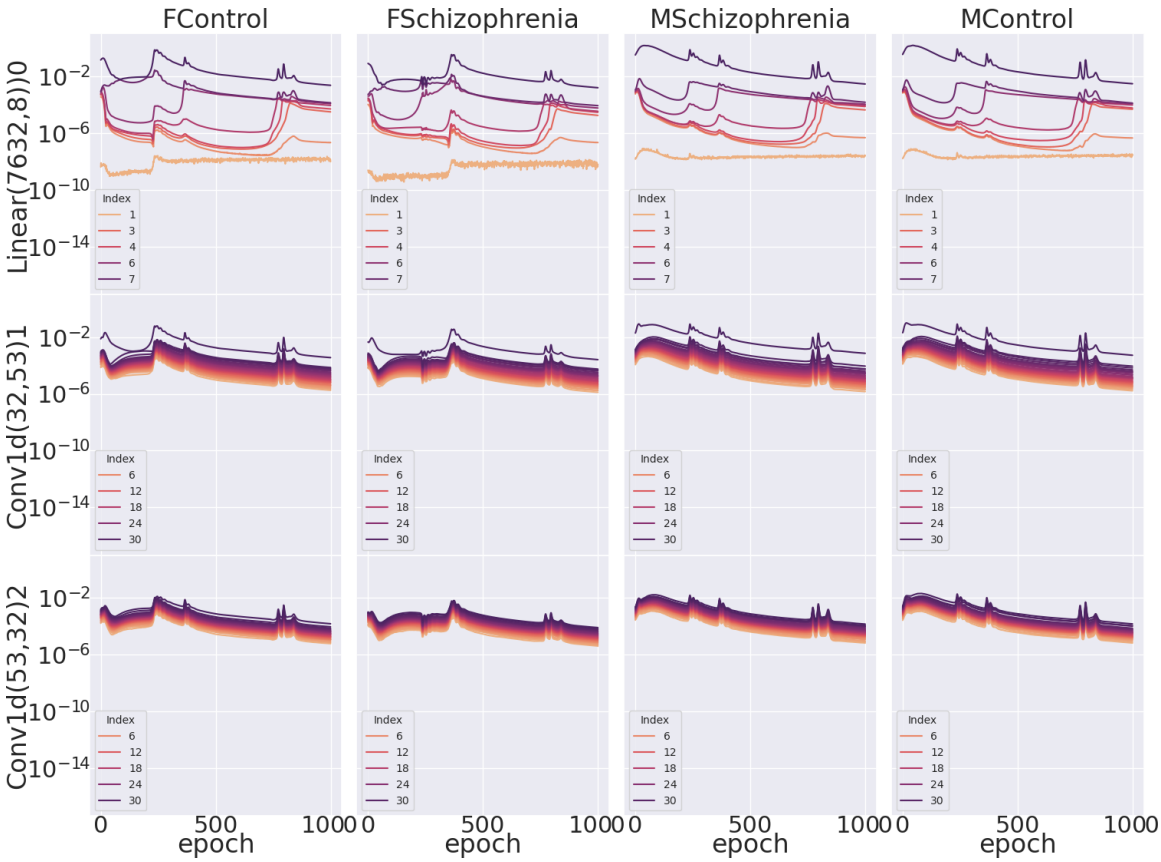

(h) Significant spectral differences between HC, SZ and Sex: BERT Classifier with hidden dim 32 and ReLU activations

Figure 6.6: Differences in Auto-Differentiation Spectra Dynamics on the COBRE data set, trained with various architectures and tasks with a BERT Transformer.

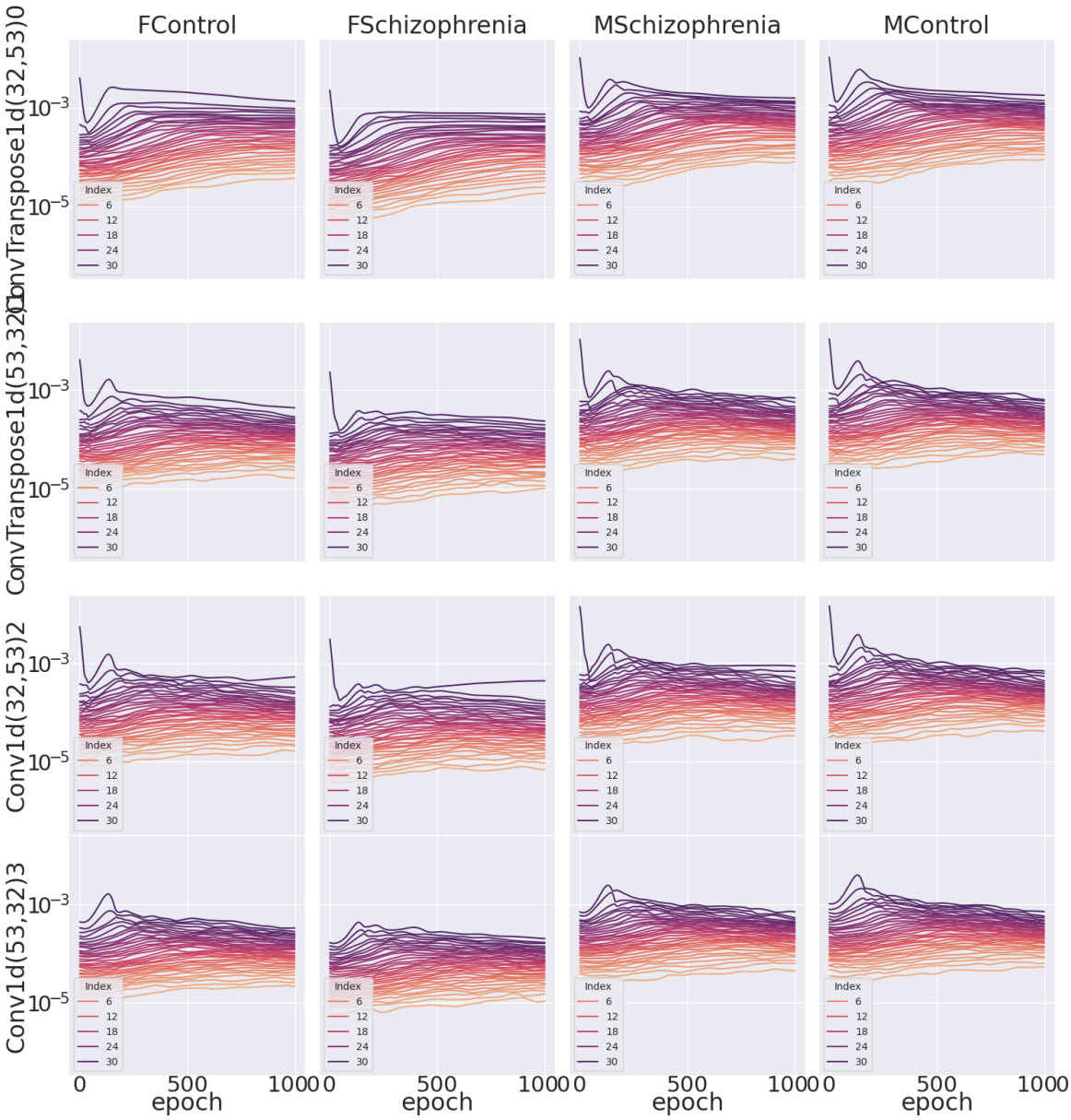
Figure 6.7: Differences in Auto-Differentiation Spectra Dynamics on the COBRE data set, trained with various architectures and tasks with a 1D CNN.



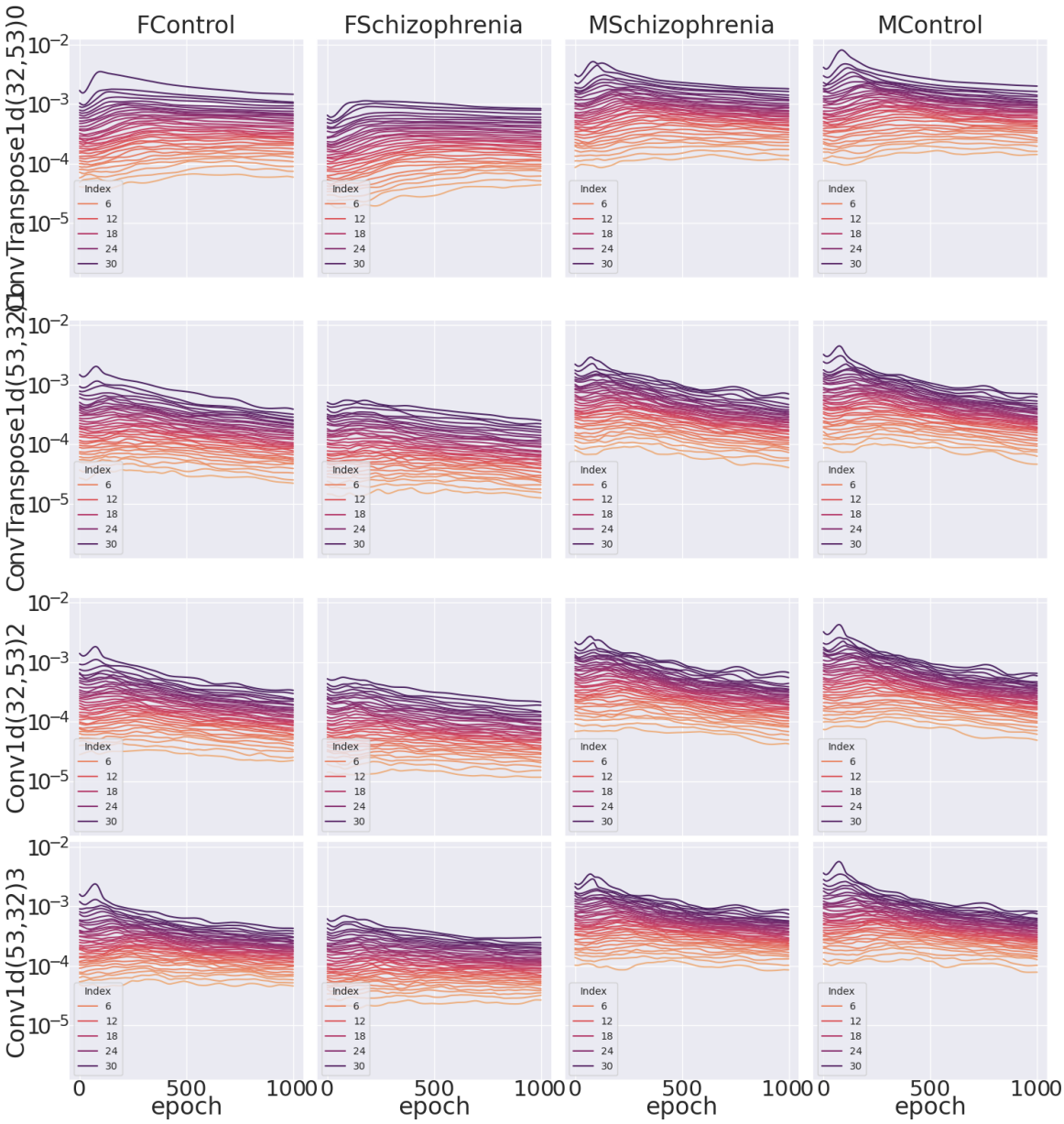

(a) CNN1D Autoencoder with hidden dim 32 and ReLU activations



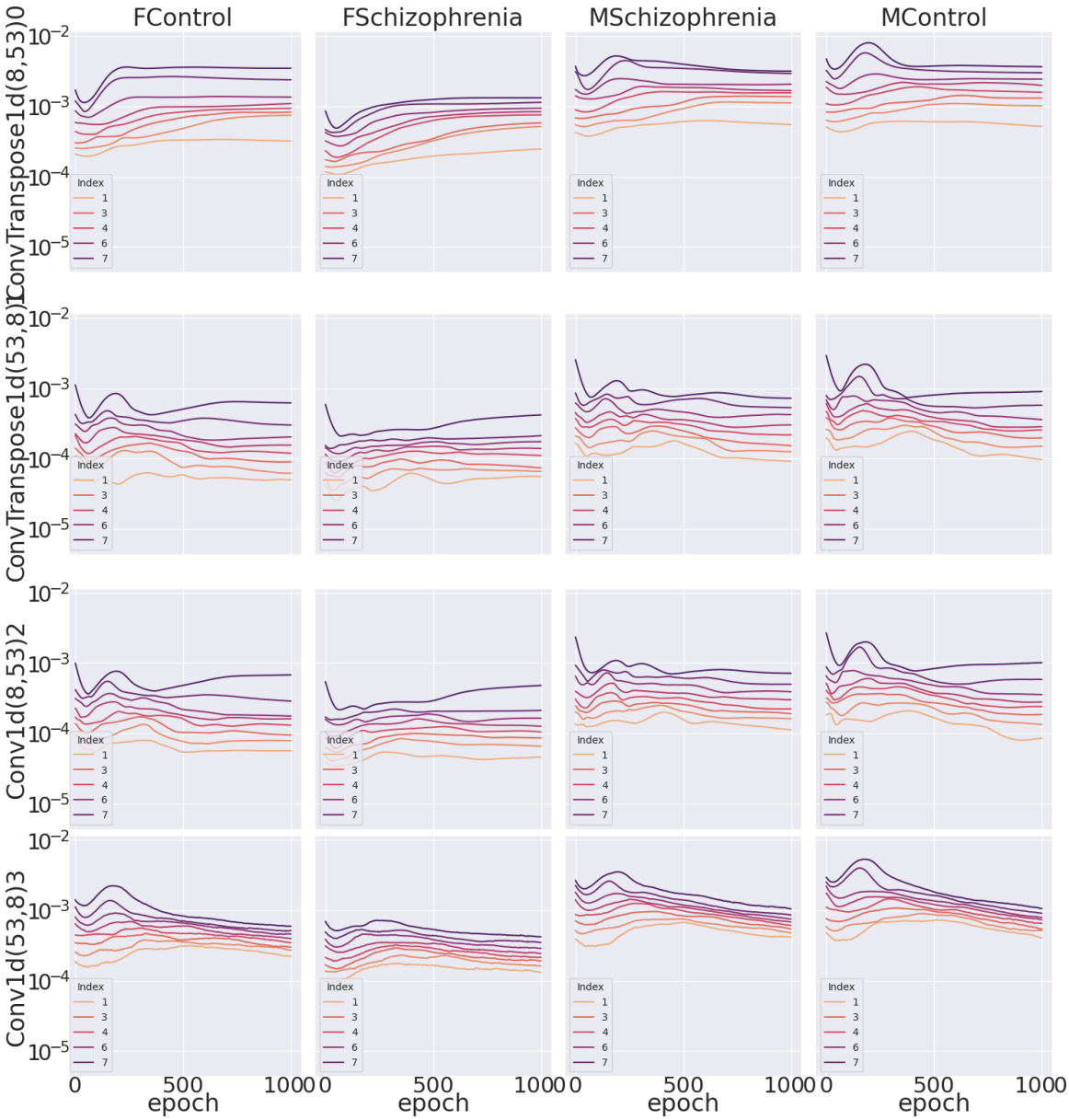

(b) CNN1D Classifier with hidden dim 32 and ReLU activations



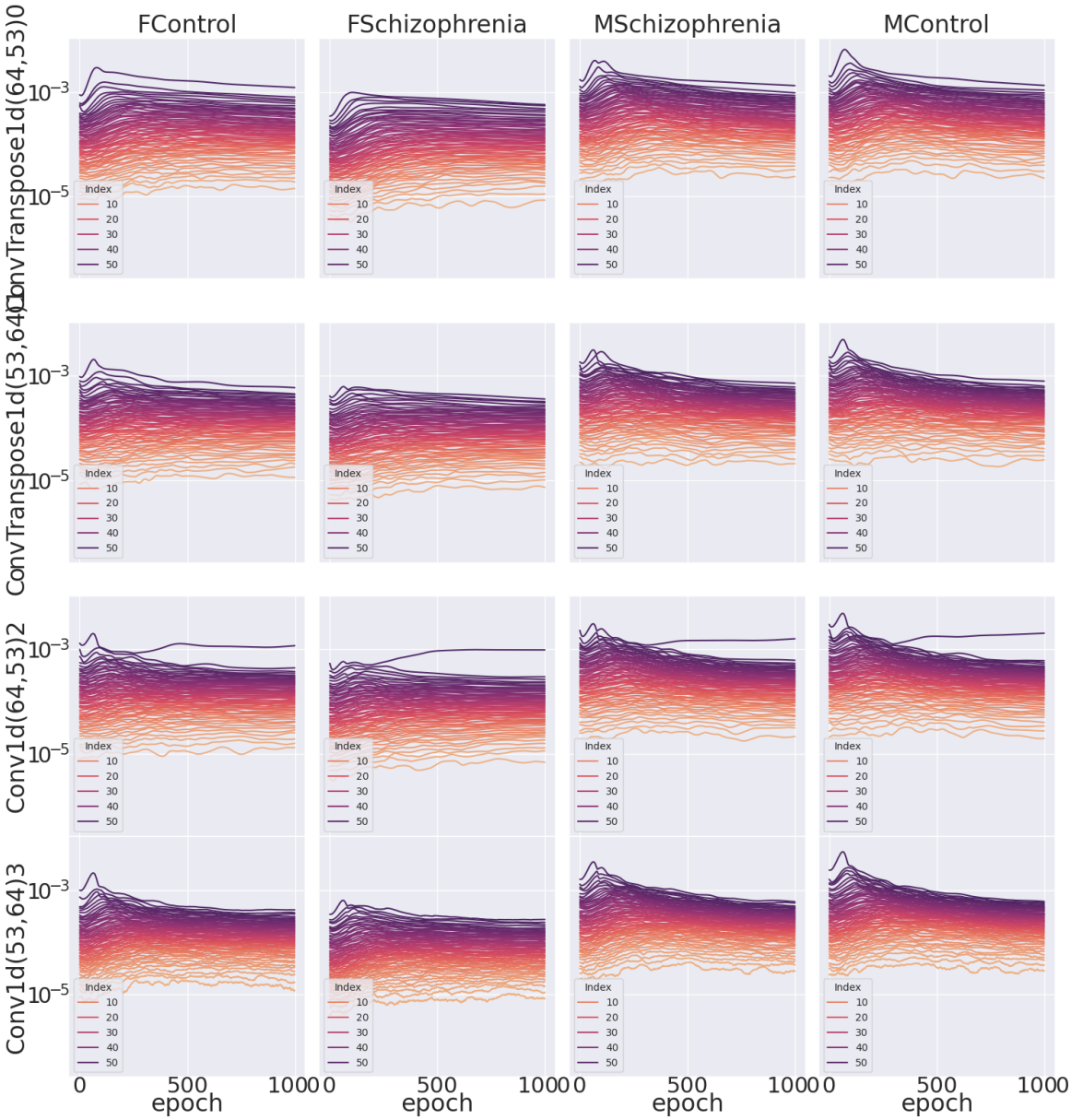

(c) CNN1D Autoencoder with hidden dim 32 and Sigmoid activations



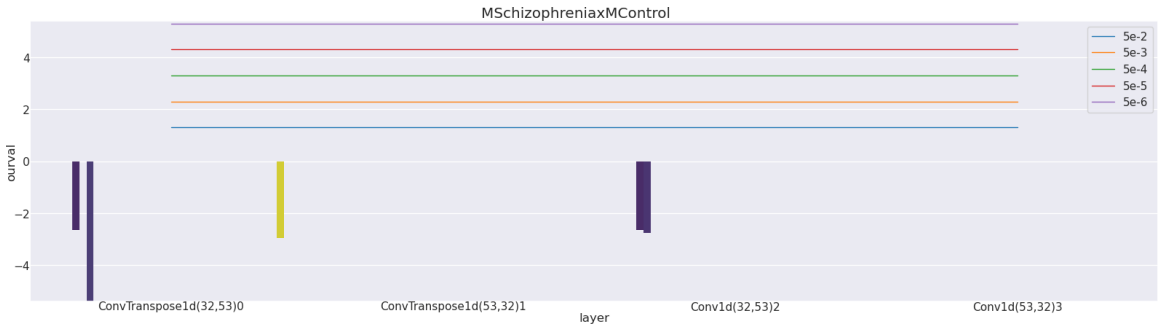

(d) CNN1D Autoencoder with hidden dim 32 and Tanh activations



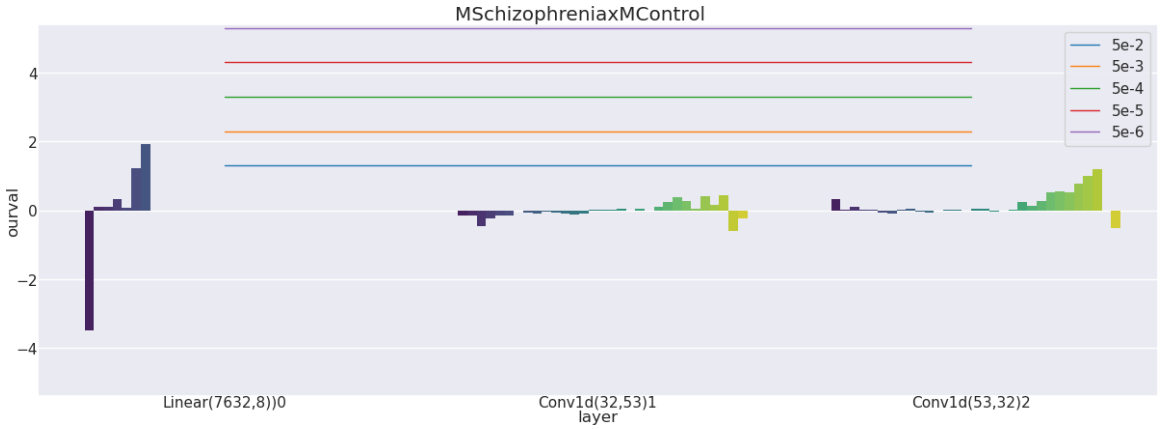

(e) CNN1D Autoencoder with hidden dim 8 and ReLU activations




(f) CNN1D Autoencoder with hidden dim 64 and ReLU activations



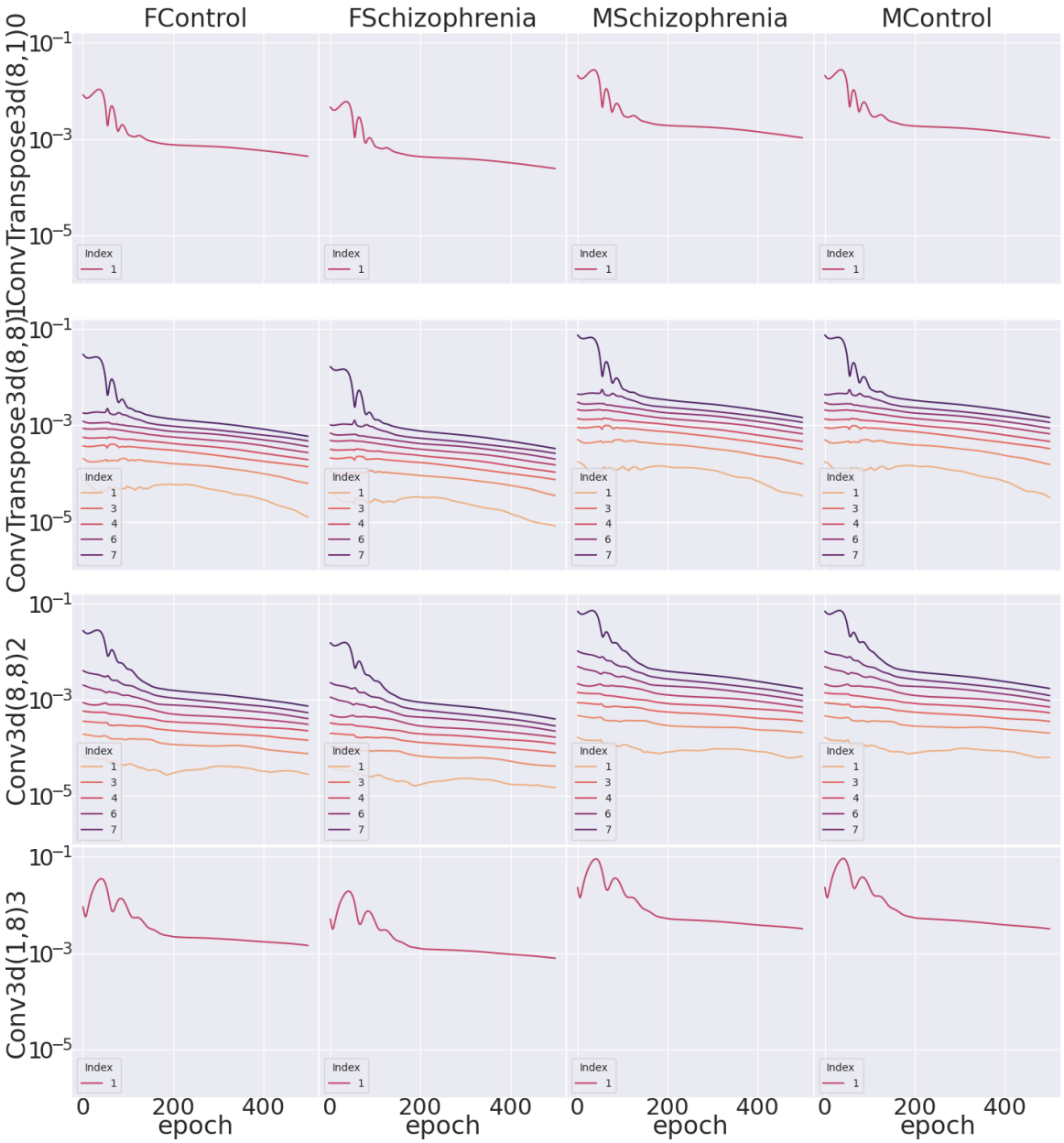

(g) Significant spectral differences between HC, SZ and Sex: CNN1D Autoencoder with hidden dim 32 and ReLU activations



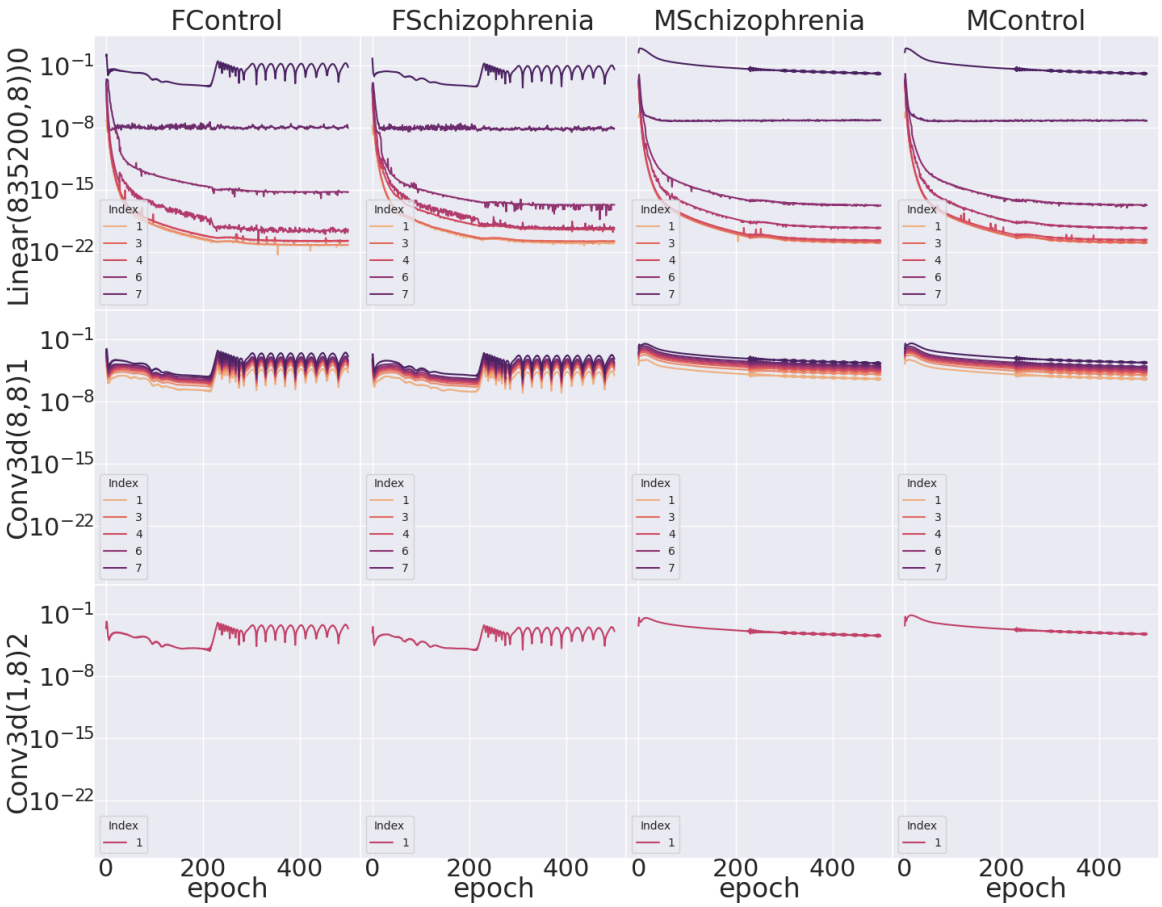

(h) Significant spectral differences between HC, SZ and Sex: CNN1D Classifier with hidden dim 32 and ReLU activations

Figure 6.7: Differences in Auto-Differentiation Spectra Dynamics on the COBRE data set, trained with various architectures and tasks with a 1D CNN.

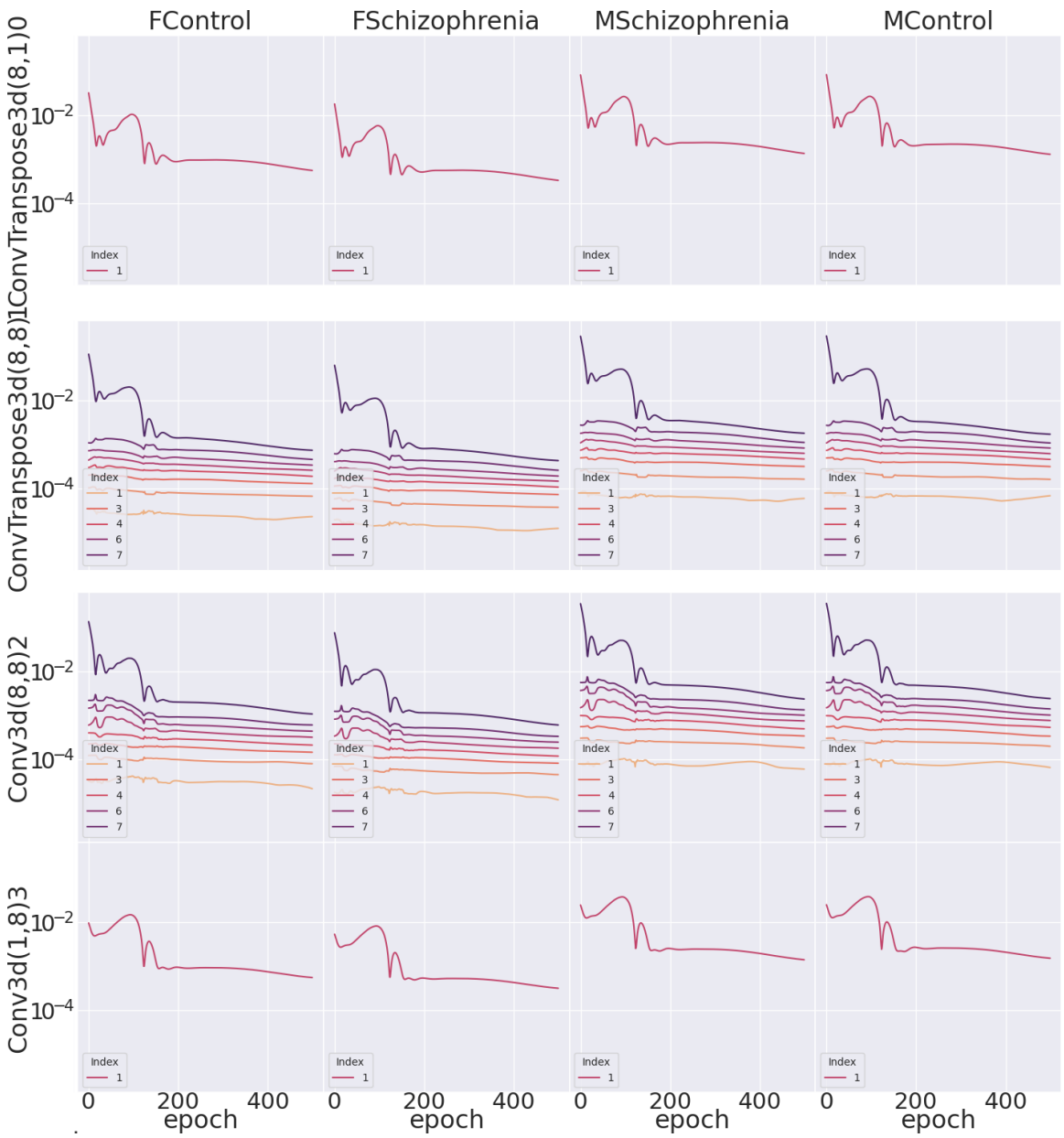
Figure 6.8: Differences in Auto-Differentiation Spectra Dynamics on the COBRE data set, trained with various architectures and tasks with a 3D CNN.



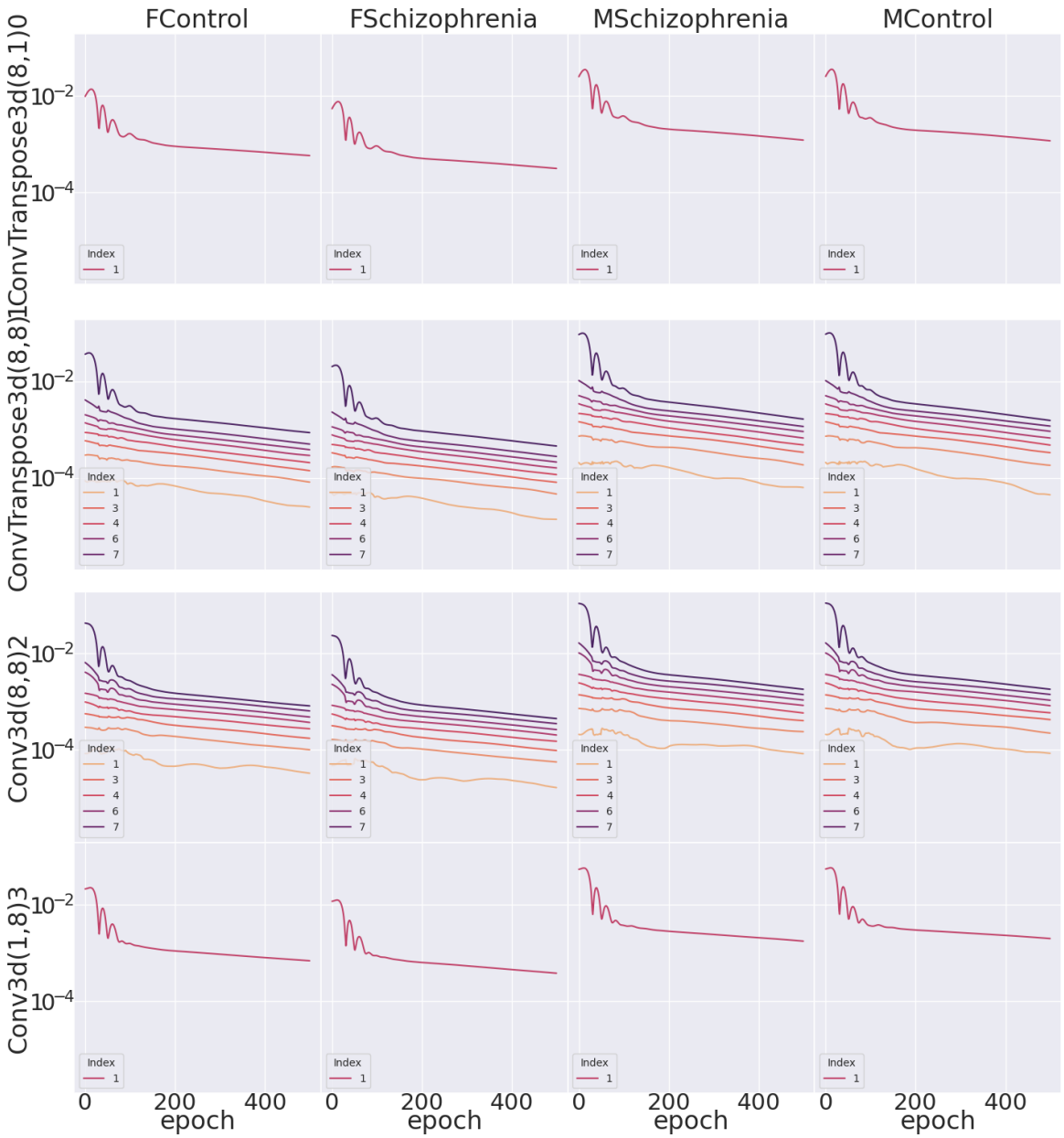

(a) CNN3D Autoencoder with hidden dim 32 and ReLU activations



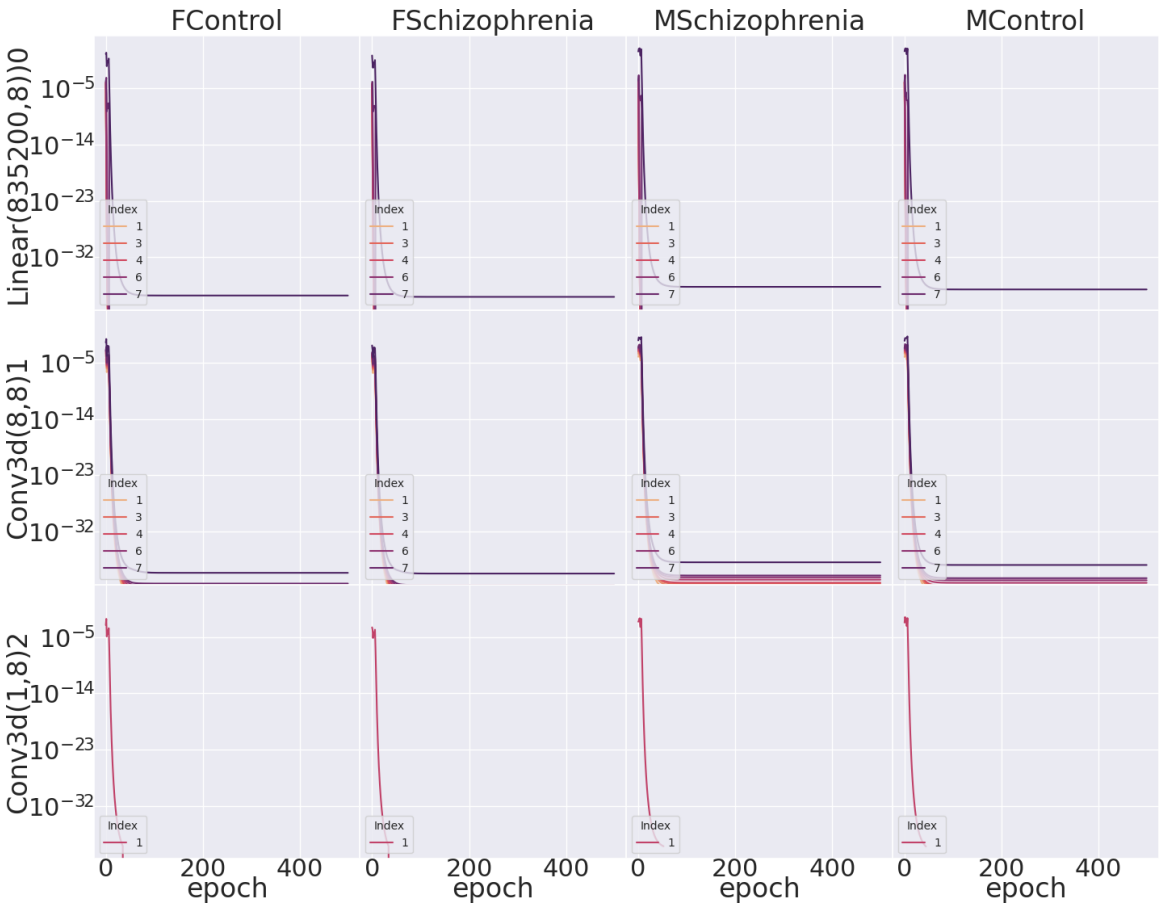

(b) CNN3D Classifier with hidden dim 32 and ReLU activations



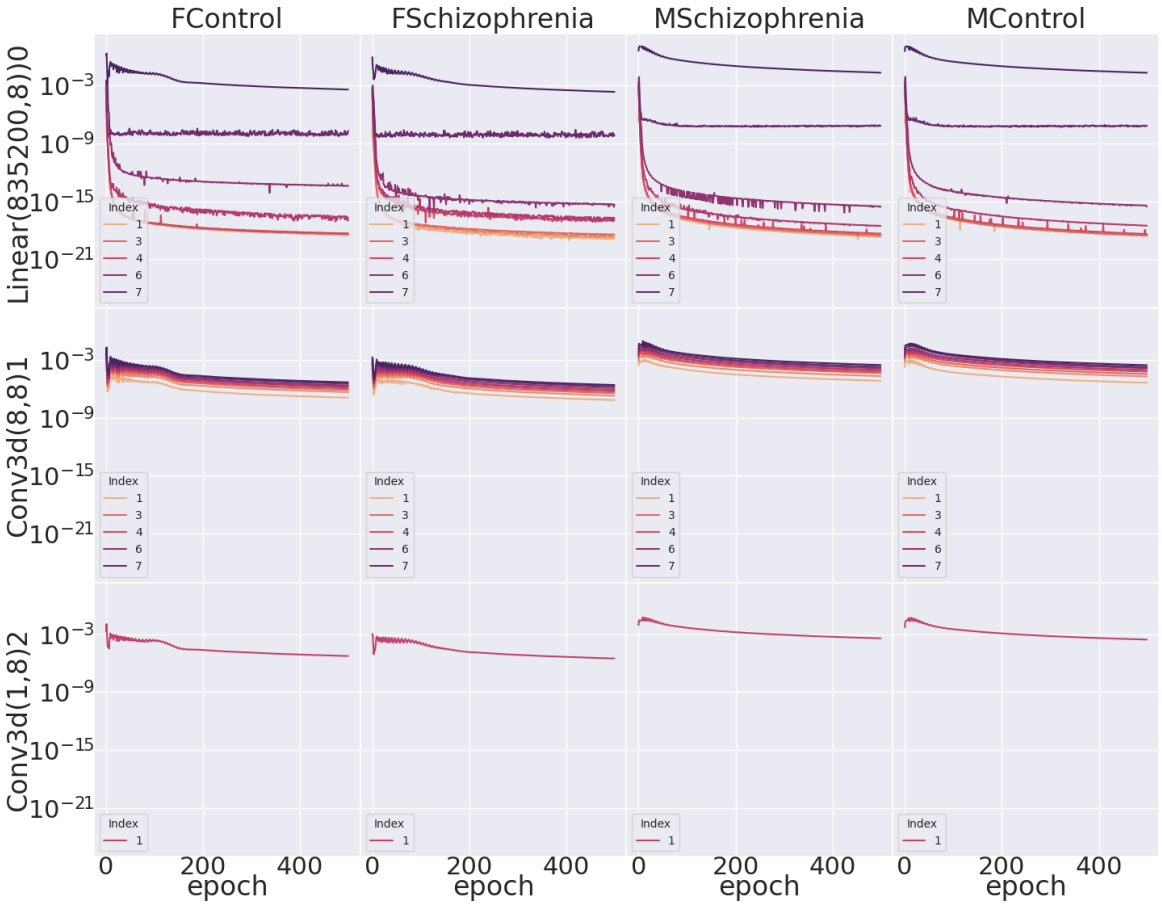

(c) CNN3D Autoencoder with hidden dim 32 and Sigmoid activations



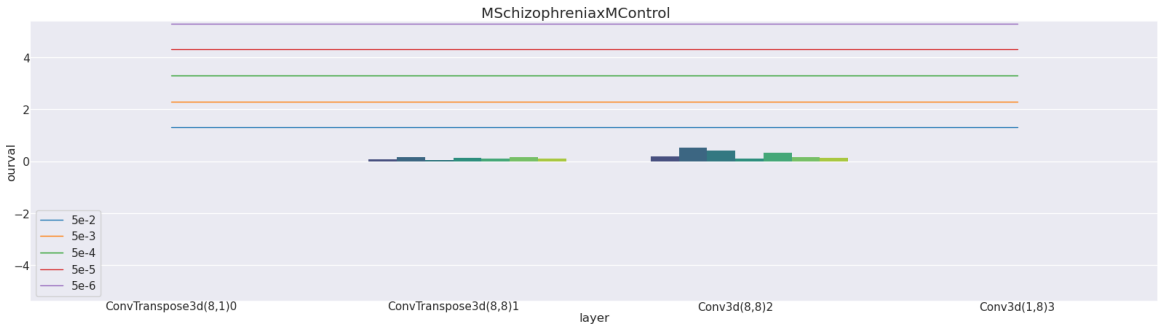

(d) CNN3D Autoencoder with hidden dim 32 and Tanh activations



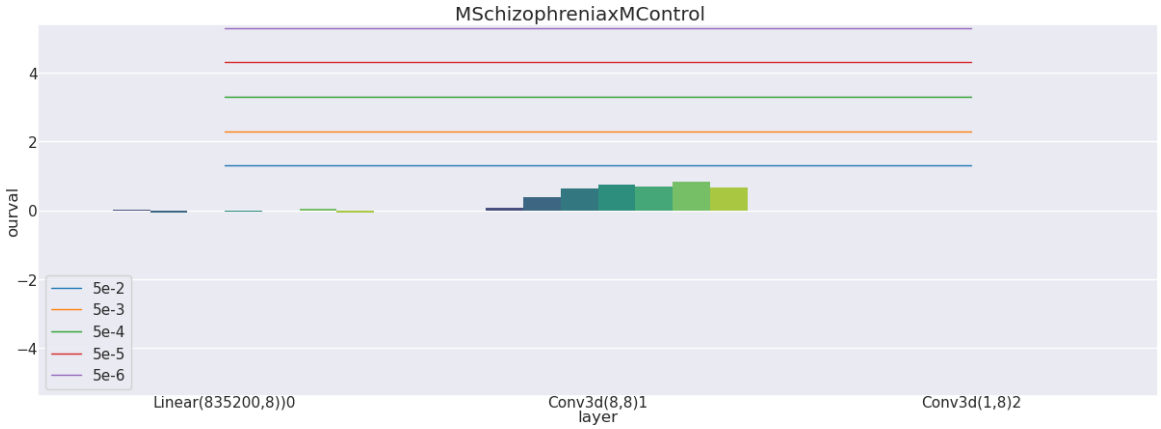

(e) CNN3D Autoencoder with hidden dim 8 and ReLU activations




(f) CNN3D Autoencoder with hidden dim 64 and ReLU activations




(g) Significant spectral differences between HC, SZ and Sex: CNN3D Autoencoder with hidden dim 32 and ReLU activations




(h) Significant spectral differences between HC, SZ and Sex: CNN3D Classifier with hidden dim 32 and ReLU activations

Figure 6.8: Differences in Auto-Differentiation Spectra Dynamics on the COBRE data set, trained with various architectures and tasks with a 3D CNN.

# CHAPTER 7
# CONCLUSION

In this dissertation, we have followed the concept of low-rank decompositions from the creation of efficient and intuitive distributed learning algorithms to the formulation of novel insights regarding the nature of how models learn. The distributed learning algorithms in chapters 2, 3 and 4 demonstrate how low-rank decompositions can provide novel, data-driven insights in settings where communication efficienc and privacy are paramount. Chapters 5 and 6 then build on the notion of low-rank learning to peer into how models learn themselves, allowing us to provide constraints on learning dynamics and provide new tools for showing how different model architectures behave differently are affected by samples belonging to different groups. The insights in chapters 5 and 6 feed back into the work on distributed learning by providing guidelines for how far the communication and privacy benefits of low-rank learning can go; however, there are implications for general learning theory beyond the distributed setting as well.

There are many, promising future directions for the work contained here, and many of them have been discussed already in individual chapters. One particularly interesting future direction for the theoretical work in chapter 5 aims to extend the analysis of nonlinear activations beyond piecewise linear functions to general nonlinear functions. While the machinery of linear algebra and the singular value decomposition in particular break down when introducing such functions, initial empirical findings indicate a rich phenomenon which will either require leveraging theory from the analysis of nonlinear functions in branches pure mathematics, or otherwise inventing new analysis techniques which can account for the observed behavior.

Formally, if we have a low-rank matrix $\mathbf{A} \in \mathbb{R}^{m \times n}$ which has rank $k < \min(m, n)$, we compute its singular value decomposition as $\mathbf{A} = \mathbf{U}_A \mathbf{\Sigma}_A \mathbf{V}_A^\top$. If we apply a nonlinear

activation $\varphi(A)$ and then compute the SVD of the resulting matrix as $\varphi(A) = \mathbf{U}_{\varphi}\mathbf{\Sigma}_{\varphi}\mathbf{V}_{\varphi}^{\top}$, we want to describe a connection between the singular values (or an analogous metric) in $\Sigma_A$ and $\Sigma_{\varphi}$.

In 7.1 we demonstrate how the application of different nonlinear activation functions can affect the magnitude of the singular values following nonlinear activation. To generate this plot, we first constructed a matrix $A \in \mathbb{R}^{4\times4}$, computed the SVD, and then removed the two smallest singular values to make it rank two. We then control the magnitude of the remaining singular values of $A$, and show how they affect the magnitude of the 4 singular values of the resulting nonlinear activation $\varphi(A)$. Although we see a consistent increase in the magnitude of nonlinear-activation singular values as the underlying linear singular values increase, the behavior is not a simple linear increase. For example, sigmoid and relu activations seem to actually supress the magnitude of the dominant singular value. Tanh activations seem to nearly reproduce the effect of linear activations on the top two dominant singular values; however, they also introduce increases in the magnitude of the lower singular values, especially when the original linear values are large. We can see from the observations in figure 7.1 that there is a rich underlying phenomenon at work here, and it very much seems to depend on the choice of nonlinear activation function.

If we are able to extend the analysis in chapter 5 to general nonlinear functions, there is another promising line of work in which we can analytically determine the updates to the singular values of the gradient (and perhaps the weights) without computing the full SVD at each training iteration. This would in and of itself be a fascinating theoretical result; however, it would also allow for the AutoSpec method to be performed more efficiently during runtime, instead of requiring the overhead of a full SVD computation for each computed gradient.

All of the methods included in this work either already are or are in the process of being made available in public software for neuroimaging and general machine learning applications. The methods in chapters 2, 3, and 4 are already integrated into the COINSTAC [44]

distributed neuroimaging toolbox, and the theoretical results in chapter 5 are being used to improve the communication efficiency of distributed deep learning algorithms in COINSTAC as well. The AutoSpec method is planned to be integrated as part of a comprehensive deep learning introspection toolbox for deep learning, and also perhaps as a contribution to the captum introspection library in pytorch.

There is still much to be garnered in machine learning from the study of low-rank algorithms and analysis. This work has presented innovations in distributed learning which led to studies in the very underlying learning itself. As the world of machine learning grows ever more complex, there are still many useful algorithms and profound theoretical insights laying in low in lower-dimensional spaces.

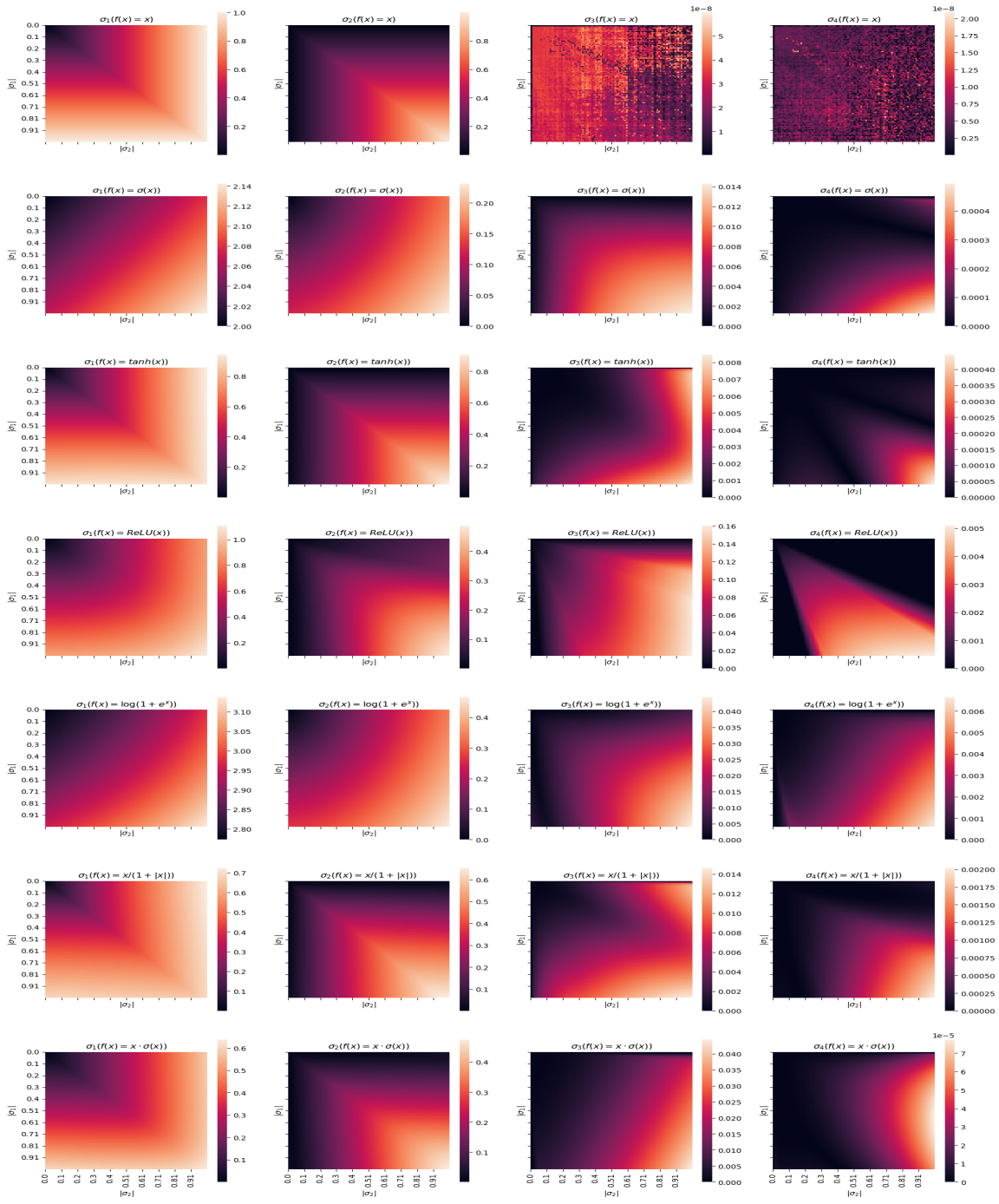



Figure 7.1: The effect of different nonlinear activation functions on the top 4 singular values of a rank 2 matrix, as a function of the top two singular values of the underlying matrix. In each plot, the vertical axis is the intensity of the largest singular value, and the horizontal axis is the intensity of the second largest singular value.

# Appendices

# APPENDIX A

# LIST OF AUTHOR PUBLICATIONS

This list contains publications listing the author for reference. Publications which were partially utilized for dissertation chapters are included in bold.

1. Baker, Bradley T., et al. “Large scale collaboration with autonomy: Decentralized data ICA.” 2015 IEEE 25th international workshop on machine learning for signal processing (MLSP). IEEE, 2015.

2. Imtiaz, Hafiz, et al. ”Privacy-preserving source separation for distributed data using independent component analysis.” 2016 Annual Conference on Information Science and Systems (CISS). IEEE, 2016.

3. Gazula, Harshvardhan, et al. ”Decentralized analysis of brain imaging data: Voxel-based morphometry and dynamic functional network connectivity.” Frontiers in neuroinformatics (2018): 55.

4. Imtiaz, Hafiz, et al. ”Improved differentially private decentralized source separation for fMRI data.” arXiv preprint arXiv:1910.12913 (2019).

5. **Baker, Bradley T., et al. ”Decentralized temporal independent component analysis: leveraging fMRI data in collaborative settings.” NeuroImage 186 (2019): 557-569.**

6. Gazula, Harshvardhan, et al. ”Coinstac: Collaborative informatics and neuroimaging suite toolkit for anonymous computation.” Journal of Open Source Software 5.54 (2020): 2166.

7. **Baker, Bradley T., et al. ”Decentralized dynamic functional network connectivity: State analysis in collaborative settings.” Human brain mapping 41.11 (2020): 2909-2925.**

8. Imtiaz, Hafiz, et al. ”A Correlated Noise-Assisted Decentralized Differentially Private Estimation Protocol, and its Application to fMRI Source Separation.” IEEE Transactions on Signal Processing 69 (2021): 6355-6370.

9. Saha, Debbrata Kumar, et al. ”dcSBM: A federated constrained source-based morphometry approach for multivariate brain structure mapping.” bioRxiv (2022): 2022-12.

10. Verner, Eric, et al. ”Brainforge: An online data analysis platform for integrative neuroimaging acquisition, analysis, and sharing.” Concurrency and Computation: Practice and Experience (2022): e6855.

11. Saha, Debbrata K., et al. ”Decentralized spatially constrained source-based morphometry.” 2022 IEEE 19th International Symposium on Biomedical Imaging (ISBI). IEEE, 2022.

12. Baker, Bradley Thomas, et al. ”Information Bottleneck for Multi-Task LSTMs.” NeurIPS 2022 Workshop on Information-Theoretic Principles in Cognitive Systems.

13. **Baker, Bradley T., et al. ”Peering Beyond the Gradient Veil with Distributed Auto Differentiation.” arXiv preprint arXiv:2102.09631 (2022).**

## REFERENCES


[1] E. Wright, "Adaptive control processes: A guided tour. by richard bellman. 1961. 42s. pp. xvi+ 255.(princeton university press)," *The Mathematical Gazette*, vol. 46, no. 356, pp. 160–161, 1962.

[2] B. Mwangi, T. S. Tian, and J. C. Soares, "A review of feature reduction techniques in neuroimaging," *Neuroinformatics*, vol. 12, pp. 229–244, 2014.

[3] A. McIntosh, F. Bookstein, J. V. Haxby, and C. Grady, "Spatial pattern analysis of functional brain images using partial least squares," *Neuroimage*, vol. 3, no. 3, pp. 143–157, 1996.

[4] A. R. McIntosh and N. J. Lobaugh, "Partial least squares analysis of neuroimaging data: Applications and advances," *Neuroimage*, vol. 23, S250–S263, 2004.

[5] A. Krishnan, L. J. Williams, A. R. McIntosh, and H. Abdi, "Partial least squares (pls) methods for neuroimaging: A tutorial and review," *Neuroimage*, vol. 56, no. 2, pp. 455–475, 2011.

[6] S. Wold, M. Sjöström, and L. Eriksson, "Pls-regression: A basic tool of chemometrics," *Chemometrics and intelligent laboratory systems*, vol. 58, no. 2, pp. 109–130, 2001.

[7] I. T. Jolliffe, *Principal component analysis for special types of data*. Springer, 2002.

[8] A. Caprihan, G. D. Pearlson, and V. D. Calhoun, "Application of principal component analysis to distinguish patients with schizophrenia from healthy controls based on fractional anisotropy measurements," *Neuroimage*, vol. 42, no. 2, pp. 675–682, 2008.

[9] A. R. Rădulescu and L. R. Mujica-Parodi, "A principal component network analysis of prefrontal-limbic functional magnetic resonance imaging time series in schizophrenia patients and healthy controls," *Psychiatry Research: Neuroimaging*, vol. 174, no. 3, pp. 184–194, 2009.

[10] U. Yoon *et al.*, "Pattern classification using principal components of cortical thickness and its discriminative pattern in schizophrenia," *Neuroimage*, vol. 34, no. 4, pp. 1405–1415, 2007.

[11] M. López *et al.*, "Principal component analysis-based techniques and supervised classification schemes for the early detection of alzheimer's disease," *Neurocomputing*, vol. 74, no. 8, pp. 1260–1271, 2011.

[12] C. H. Fu *et al.*, "Pattern classification of sad facial processing: Toward the development of neurobiological markers in depression," *Biological psychiatry*, vol. 63, no. 7, pp. 656–662, 2008.

[13] J. Mourao-Miranda, A. L. Bokde, C. Born, H. Hampel, and M. Stetter, "Classifying brain states and determining the discriminating activation patterns: Support vector machine on functional mri data," *Neuroimage*, vol. 28, no. 4, pp. 980–995, 2005.

[14] V. D. Calhoun, J. Liu, and T. Adalı, "A review of group ICA for fMRI data and ICA for joint inference of imaging, genetic, and ERP data," *NeuroImage*, vol. 45, no. 1, S163–S172, Mar. 2009.

[15] M. Svensén, F. Kruggel, and H. Benali, "ICA of fMRI group study data," *NeuroImage*, vol. 16, no. 3, pp. 551–563, Jul. 2002.

[16] V. D. Calhoun and T. Adalı, "Unmixing fMRI with independent component analysis," *IEEE Engineering in Medicine and Biology Magazine*, vol. 25, no. 2, pp. 79–90, Mar. 2006.

[17] B. B. Biswal and J. L. Ulmer, "Blind source separation of multiple signal sources of fMRI data sets using independent component analysis," *Journal of Computer Assisted Tomography*, vol. 23, no. 2, pp. 265–271, 1999.

[18] I. Daubechies *et al.*, "Independent component analysis for brain fMRI does not select for independence," *Proceedings of the National Academy of Sciences*, vol. 106, no. 26, pp. 10 415–10 422, Jun. 2009.

[19] V. Calhoun *et al.*, "Independent component analysis for brain fMRI does indeed select for maximal independence," *PloS One*, vol. 8, no. 8, e73309, Aug. 2013.

[20] Z. Boukouvalas, Y. Levin-Schwartz, V. D. Calhoun, and T. Adalı, "Sparsity and independence: Balancing two objectives in optimization for source separation with application to fmri analysis," *Journal of the Franklin Institute*, vol. 355, no. 4, pp. 1873–1887, Mar. 2017.

[21] A. J. Bell and T. J. Sejnowski, "An information-maximization approach to blind separation and blind deconvolution," *Neural Computation*, vol. 7, no. 6, pp. 1129–1159, Apr. 1995.

[22] A. Hyvärinen, "A family of fixed-point algorithms for independent component analysis," in *Proceedings of the 1997 IEEE International Conference on Acoustics, Speech, and Signal Processing*, vol. 5, Apr. 1997, pp. 3917–3920.

[23] A. Hyvärinen, "Fast and robust fixed-point algorithms for independent component analysis," *IEEE Transactions on Neural Networks*, vol. 10, no. 3, pp. 626–634, May 1999.

[24] A. Hyvärinen and E. Oja, "Independent component analysis: Algorithms and applications," *Neural Networks*, vol. 13, no. 4–5, pp. 411–430, 2000.

[25] R. Silva, S. Plis, J. Sui, M. Pattichis, T. Adalı, and V. Calhoun, "Blind source separation for unimodal and multimodal brain networks:a unifying framework for subspace modeling," *EEE Journal of Selected Topics in Signal Processing*, vol. 10, no. 7, pp. 1134–1149, 2016.

[26] J. Sui, T. A. G. Pearlson, and V. Calhoun, "An ICA-based method for the identification of optimal fMRI features and components using combined group-discriminative techniques," *NeuroImage*, vol. 46, no. 1, pp. 73–86, May 2009.

[27] J. Liu and V. Calhoun, "Parallel independent component analysis for multimodal analysis: Application to fMRI and EEG data," in *Proceedings of the 4th IEEE International Symposium on Biomedical Imaging: From Nano to Macro (ISBI 2007)*, Apr. 2007, pp. 1028–1031.

[28] E. Allen *et al.*, "A baseline for the multivariate comparison of resting state networks," *Frontiers in Systems Neuroscience*, vol. 5, no. 2, 2011.

[29] V. Calhoun, T. Adalı, N. Giuliani, J. Pekar, K. Kiehl, and G. Pearlson, "Method for multimodal analysis of independent source differences in schizophrenia: Combining gray matter structural and auditory oddball functional data," *Human Brain Mapping*, vol. 27, no. 1, pp. 47–62, 2006.

[30] L. B. Almeida, "Misep—linear and nonlinear ica based on mutual information," *The journal of Machine Learning Research*, vol. 4, pp. 1297–1318, 2003.

[31] A. Hyvarinen and H. Morioka, "Unsupervised feature extraction by time-contrastive learning and nonlinear ica," *Advances in neural information processing systems*, vol. 29, 2016.

[32] A. Hyvarinen and H. Morioka, "Nonlinear ica of temporally dependent stationary sources," in *Artificial Intelligence and Statistics*, PMLR, 2017, pp. 460–469.

[33] C. E. Shannon, "A mathematical theory of communication," *The Bell system technical journal*, vol. 27, no. 3, pp. 379–423, 1948.

[34] D. P. Kingma and M. Welling, "Auto-encoding variational bayes," *arXiv preprint arXiv:1312.6114*, 2013.

[35] K. Han *et al.*, "Variational autoencoder: An unsupervised model for encoding and decoding fmri activity in visual cortex," *NeuroImage*, vol. 198, pp. 125–136, 2019.

[36] N. Qiang, Q. Dong, Y. Sun, B. Ge, and T. Liu, "Deep variational autoencoder for modeling functional brain networks and adhd identification," in *2020 IEEE 17th International Symposium on Biomedical Imaging (ISBI)*, IEEE, 2020, pp. 554–557.

[37] N. Qiang *et al.*, "Deep variational autoencoder for mapping functional brain networks," *IEEE Transactions on Cognitive and Developmental Systems*, vol. 13, no. 4, pp. 841–852, 2020.

[38] X. Zhang, E. A. Maltbie, and S. D. Keilholz, "Spatiotemporal trajectories in resting-state fmri revealed by convolutional variational autoencoder," *NeuroImage*, vol. 244, p. 118 588, 2021.

[39] E. Geenjaar, N. Lewis, Z. Fu, R. Venkatdas, S. Plis, and V. Calhoun, "Fusing multimodal neuroimaging data with a variational autoencoder," in *2021 43rd Annual International Conference of the IEEE Engineering in Medicine & Biology Society (EMBC)*, IEEE, 2021, pp. 3630–3633.

[40] J. Konečnỳ, H. B. McMahan, F. X. Yu, P. Richtárik, A. T. Suresh, and D. Bacon, "Federated learning: Strategies for improving communication efficiency," *arXiv preprint arXiv:1610.05492*, 2016.

[41] J. Konečnỳ, H. B. McMahan, D. Ramage, and P. Richtárik, "Federated optimization: Distributed machine learning for on-device intelligence," *arXiv preprint arXiv:1610.02527*, 2016.

[42] S. Plis, A. Sarwate, J. Turner, M. Arbabshirani, and V. Calhoun, "From private sites to big data without compromising privacy: A case of neuroimaging data classification," *Value in Health*, vol. 17, no. 3, A190, 2014.

[43] B. T. Baker, R. F. Silva, V. D. Calhoun, A. D. Sarwate, and S. M. Plis, "Large scale collaboration with autonomy: Decentralized data ICA," in *Proceedings of the IEEE 25th International Workshop on Machine Learning for Signal Processing (MLSP)*, Boston, MA, USA, Sep. 2015.

[44] S. M. Plis *et al.*, "Coinstac: A privacy enabled model and prototype for leveraging and processing decentralized brain imaging data," *Frontiers in neuroscience*, vol. 10, p. 365, 2016.

[45] K. Bonawitz *et al.*, "Practical secure aggregation for privacy-preserving machine learning," in *proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*, 2017, pp. 1175–1191.

[46] C. Dwork, F. McSherry, K. Nissim, and A. Smith, "Calibrating noise to sensitivity in private data analysis," in *Theory of Cryptography*, S. Halevi and T. Rabin, Eds., ser. Lecture Notes in Computer Science, vol. 3876, Berlin, Heidelberg: Springer, Mar. 2006, pp. 265–284.

[47] A. D. Sarwate, S. M. Plis, J. A. Turner, M. R. Arbabshirani, and V. D. Calhoun, "Sharing privacy-sensitive access to neuroimaging and genetics data: A review and preliminary validation," *Frontiers in neuroinformatics*, vol. 8, p. 35, 2014.

[48] B. T. Baker *et al.*, "Decentralized temporal independent component analysis: Leveraging fmri data in collaborative settings," *NeuroImage*, vol. 186, pp. 557–569, 2019.

[49] B. T. Baker, E. Damaraju, R. F. Silva, S. M. Plis, and V. D. Calhoun, "Decentralized dynamic functional network connectivity: State analysis in collaborative settings," *Human brain mapping*, vol. 41, no. 11, pp. 2909–2925, 2020.

[50] N. P. Wojtalewicz, R. F. Silva, V. D. Calhoun, A. D. Sarwate, and S. M. Plis, "Decentralized independent vector analysis," in *2017 IEEE international conference on acoustics, speech and signal processing (ICASSP)*, IEEE, 2017, pp. 826–830.

[51] N. Lewis, S. Plis, and V. Calhoun, "Cooperative learning: Decentralized data neural network," in *2017 international joint conference on neural networks (IJCNN)*, IEEE, 2017, pp. 324–331.

[52] D. K. Saha, V. D. Calhoun, S. R. Panta, and S. M. Plis, "See without looking: Joint visualization of sensitive multi-site datasets.," in *IJCAI*, 2017, pp. 2672–2678.

[53] D. K. Saha, V. D. Calhoun, D. Yuhui, F. Zening, S. R. Panta, and S. M. Plis, "Dsne: A visualization approach for use with decentralized data," *BioRxiv*, p. 826 974, 2019.

[54] S. AbdulRahman, H. Tout, H. Ould-Slimane, A. Mourad, C. Talhi, and M. Guizani, "A survey on federated learning: The journey from centralized to distributed on-site learning and beyond," *IEEE Internet of Things Journal*, vol. 8, no. 7, pp. 5476–5497, 2020.

[55] C. Zhang, Y. Xie, H. Bai, B. Yu, W. Li, and Y. Gao, "A survey on federated learning," *Knowledge-Based Systems*, vol. 216, p. 106 775, 2021.

[56] H. Zhu, H. Zhang, and Y. Jin, "From federated learning to federated neural architecture search: A survey," *Complex & Intelligent Systems*, vol. 7, pp. 639–657, 2021.

[57] C.-R. Shyu *et al.*, "A systematic review of federated learning in the healthcare area: From the perspective of data properties and applications," *Applied Sciences*, vol. 11, no. 23, p. 11 191, 2021.

[58] A. F. Aji and K. Heafield, "Sparse communication for distributed gradient descent," *arXiv preprint arXiv:1704.05021*, 2017.

[59] H. Wang, S. Sievert, S. Liu, Z. Charles, D. Papailiopoulos, and S. Wright, "Atomo: Communication-efficient learning via atomic sparsification," in *Advances in Neural Information Processing Systems*, 2018, pp. 9850–9861.

[60] S. U. Stich, "Local SGD converges fast and communicates little," *arXiv preprint arXiv:1805.09767*, 2018.

[61] W. Wen *et al.*, "Terngrad: Ternary gradients to reduce communication in distributed deep learning," in *Advances in neural information processing systems*, 2017, pp. 1509–1519.

[62] F. Sattler, S. Wiedemann, K.-R. Müller, and W. Samek, "Sparse binary compression: Towards distributed deep learning with minimal communication," in *2019 International Joint Conference on Neural Networks (IJCNN)*, IEEE, 2019, pp. 1–8.

[63] F. Sattler, S. Wiedemann, K.-R. Müller, and W. Samek, "Robust and communication-efficient federated learning from non-iid data," *IEEE Transactions on Neural Networks and Learning Systems*, 2019.

[64] S. Shi *et al.*, "A distributed synchronous SGD algorithm with global Top-$k$ sparsification for low bandwidth networks," in *2019 IEEE 39th International Conference on Distributed Computing Systems (ICDCS)*, IEEE, 2019, pp. 2238–2247.

[65] D. Alistarh, D. Grubic, J. Li, R. Tomioka, and M. Vojnovic, "QSGD: Communication-efficient SGD via gradient quantization and encoding," *arXiv preprint arXiv:1610.02132*, 2016.

[66] N. Agarwal, A. T. Suresh, F. X. X. Yu, S. Kumar, and B. McMahan, "CPSGD: Communication-efficient and differentially-private distributed SGD," in *Advances in Neural Information Processing Systems*, 2018, pp. 7564–7575.

[67] J. Bernstein, Y.-X. Wang, K. Azizzadenesheli, and A. Anandkumar, "signSGD: Compressed optimisation for non-convex problems," *arXiv preprint arXiv:1802.04434*, 2018.

[68] S. Horváth, D. Kovalev, K. Mishchenko, S. Stich, and P. Richtárik, "Stochastic distributed learning with gradient quantization and variance reduction," *arXiv preprint arXiv:1904.05115*, 2019.

[69] M. Yu *et al.*, "Gradiveq: Vector quantization for bandwidth-efficient gradient aggregation in distributed cnn training," in *Advances in Neural Information Processing Systems*, 2018, pp. 5123–5133.

[70] Y. Lin, S. Han, H. Mao, Y. Wang, and W. J. Dally, "Deep gradient compression: Reducing the communication bandwidth for distributed training," *arXiv preprint arXiv:1712.01887*, 2017.

[71] A. Koloskova, T. Lin, S. U. Stich, and M. Jaggi, "Decentralized deep learning with arbitrary communication compression," *arXiv preprint arXiv:1907.09356*, 2019.

[72] K. Mishchenko, E. Gorbunov, M. Takáč, and P. Richtárik, "Distributed learning with compressed gradient differences," *arXiv preprint arXiv:1901.09269*, 2019.

[73] H. Zhang *et al.*, "Poseidon: An efficient communication architecture for distributed deep learning on GPU clusters," in *2017 USENIX Annual Technical Conference (USENIX ATC 17)*, 2017, pp. 181–193.

[74] T. Chen, G. Giannakis, T. Sun, and W. Yin, "LAG: Lazily aggregated gradient for communication-efficient distributed learning," in *Advances in Neural Information Processing Systems*, 2018, pp. 5050–5060.

[75] F. Haddadpour, M. M. Kamani, M. Mahdavi, and V. Cadambe, "Local SGD with periodic averaging: Tighter analysis and adaptive synchronization," in *Advances in Neural Information Processing Systems*, 2019, pp. 11 082–11 094.

[76] H. Yu, S. Yang, and S. Zhu, "Parallel restarted SGD with faster convergence and less communication: Demystifying why model averaging works for deep learning," in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 33, 2019, pp. 5693–5700.

[77] M. Assran, N. Loizou, N. Ballas, and M. Rabbat, "Stochastic gradient push for distributed deep learning," in *International Conference on Machine Learning*, PMLR, 2019, pp. 344–353.

[78] B. T. Baker, A. Khanal, V. D. Calhoun, B. A. Pearlmutter, and S. M. Plis, "Peering beyond the gradient veil with distributed auto differentiation," *arXiv preprint arXiv:2102.09631*, 2021.

[79] K. Rush, Z. Charles, and Z. Garrett, "Federated automatic differentiation," *arXiv preprint arXiv:2301.07806*, 2023.

[80] T. Vogels, S. P. Karimireddy, and M. Jaggi, "PowerSGD: Practical low-rank gradient compression for distributed optimization," *Advances in Neural Information Processing Systems*, vol. 32, 2019.

[81] A. M. Saxe, J. L. McClelland, and S. Ganguli, "Exact solutions to the nonlinear dynamics of learning in deep linear neural networks," *arXiv preprint arXiv:1312.6120*, 2013.

[82] A. M. Saxe, J. L. McClelland, and S. Ganguli, "A mathematical theory of semantic development in deep neural networks," *Proceedings of the National Academy of Sciences*, vol. 116, no. 23, pp. 11 537–11 546, 2019.

[83] N. Tishby and N. Zaslavsky, "Deep learning and the information bottleneck principle," in *2015 ieee information theory workshop (itw)*, IEEE, 2015, pp. 1–5.

[84] R. Shwartz-Ziv and N. Tishby, "Opening the black box of deep neural networks via information," *arXiv preprint arXiv:1703.00810*, 2017.

[85] A. M. Saxe *et al.*, "On the information bottleneck theory of deep learning," *Journal of Statistical Mechanics: Theory and Experiment*, vol. 2019, no. 12, p. 124 020, 2019.

[86] G. Chechik, A. Globerson, N. Tishby, and Y. Weiss, "Information bottleneck for gaussian variables," *Advances in Neural Information Processing Systems*, vol. 16, 2003.

[87] V. Papyan, X. Y. Han, and D. L. Donoho, "Prevalence of neural collapse during the terminal phase of deep learning training," *Proceedings of the National Academy of Sciences*, vol. 117, no. 40, pp. 24 652–24 663, 2020.

[88] D. G. Mixon, H. Parshall, and J. Pi, "Neural collapse with unconstrained features," *arXiv preprint arXiv:2011.11619*, 2020.

[89] Z. Zhu *et al.*, "A geometric analysis of neural collapse with unconstrained features," *Advances in Neural Information Processing Systems*, vol. 34, pp. 29 820–29 834, 2021.

[90] J. Lu and S. Steinerberger, "Neural collapse with cross-entropy loss," *arXiv preprint arXiv:2012.08465*, 2020.

[91] X. Y. Han, V. Papyan, and D. L. Donoho, "Neural collapse under MSE loss: Proximity to and dynamics on the central path," *arXiv preprint arXiv:2106.02073*, 2021.

[92] J. Zhou, X. Li, T. Ding, C. You, Q. Qu, and Z. Zhu, "On the optimization landscape of neural collapse under mse loss: Global optimality with unconstrained features," in *International Conference on Machine Learning*, PMLR, 2022, pp. 27 179–27 202.

[93] M. Xu, A. Rangamani, Q. Liao, T. Galanti, and T. Poggio, "Dynamics in deep classifiers trained with the square loss: Normalization, low rank, neural collapse, and generalization bounds," *Research*, vol. 6, p. 0024, 2023.

[94] R. A. Poldrack *et al.*, "Toward open sharing of task-based fMRI data: The OpenfMRI project," *Frontiers in Neuroinformatics*, vol. 7, 2013.

[95] K. Gorgolewski, O. Esteban, G. Schaefer, B. Wandell, and R. Poldrack, "Openneuro—a free online platform for sharing and analysis of neuroimaging data," in *Organization for Human Brain Mapping (OHBM)*, Vancouver, Canada, 2017, p. 1677.

[96] R. A. Poldrack and K. J. Gorgolewski, "Making big data open: Data sharing in neuroimaging," *Nature Neuroscience*, vol. 17, no. 11, pp. 1510–1517, 2014.

[97] P. M. Thompson *et al.*, "The ENIGMA consortium: Large-scale collaborative analyses of neuroimaging and genetic data," *Brain Imaging and Behavior*, vol. 8, no. 2, pp. 153–182, 2014.

[98] C. R. Jack *et al.*, "The Alzheimer's disease neuroimaging initiative (ADNI): MRI methods," *Journal of Magnetic Resonance Imaging*, vol. 27, no. 4, pp. 685–691, 2008.

[99] P. M. Thompson *et al.*, "ENIGMA and the individual: Predicting factors that affect the brain in 35 countries worldwide," *NeuroImage*, vol. 145, pp. 389–408, Jan. 2017.

[100] T. G. van Erp *et al.*, "Subcortical brain volume abnormalities in 2028 individuals with schizophrenia and 2540 healthy controls via the ENIGMA consortium," *Molecular Psychiatry*, vol. 21, no. 4, p. 547, Jun. 2016.

[101] D. P. Hibar *et al.*, "Common genetic variants influence human subcortical brain structures," *Nature*, vol. 520, no. 7546, p. 224, Jan. 2015.

[102] R. Mcdonald, M. Mohri, N. Silberman, D. Walker, and G. S. Mann, "Efficient large-scale distributed training of conditional maximum entropy models," in *Advances in Neural Information Processing Systems 22 (NIPS 2009)*, Y. Bengio, D. Schuurmans, J. D. Lafferty, C. K. I. Williams, and A. Culotta, Eds., Curran Associates, Inc., 2009, pp. 1231–1239.

[103] M. Zinkevich, M. Weimer, L. Li, and A. J. Smola, "Parallelized stochastic gradient descent," in *Advances in Neural Information Processing Systems 23 (NIPS 2010)*, J. D. Lafferty, C. K. I. Williams, J. Shawe-Taylor, R. S. Zemel, and A. Culotta, Eds., Curran Associates, Inc., 2010, pp. 2595–2603.

[104] V. D. Calhoun and T. Adalı, "Multisubject independent component analysis of fMRI: A decade of intrinsic networks, default mode, and neurodiagnostic discovery," *IEEE Reviews in Biomedical Engineering*, vol. 5, pp. 60–73, 2012.

[105] S. Plis *et al.*, "COINSTAC: A privacy enabled model and prototype for leveraging and processing decentralized brain imaging data," *Frontiers in Neuroscience*, vol. 10, no. 365, Aug. 2016.

[106] H. Imtiaz, R. Silva, B. Baker, S. M. Plis, A. D. Sarwate, and V. D. Calhoun, "Privacy-preserving source separation for distributed data using independent component analysis," in *Proceedings of the 2016 Annual Conference on Information Science and Systems (CISS)*, Princeton, NJ, USA, Mar. 2016, pp. 123–127.

[107] J. Stone, J. Porrill, C. Buchel, and K. Friston, "Spatial, temporal, and spatiotemporal independent component analysis of fMRI data," in *Spatio-temporal Modelling and its applications*, Citeseer, Department of Statistics, University of Leeds, Jul. 1999, pp. 7–9.

[108] V. Calhoun, T. Adalı, G. Pearlson, and J. Pekar, "Spatial and temporal independent component analysis of functional MRI data containing a pair of task-related waveforms," *Human Brain Mapping*, vol. 13, no. 1, pp. 43–53, Mar. 2001.

[109] X. Gao, T. Zhang, and J. Xiong, "Comparison between spatial and temporal independent component analysis for blind source separation in fmri data," in *Biomedical Engineering and Informatics (BMEI), 2011 4th International Conference on*, IEEE, vol. 2, 2011, pp. 690–692.

[110] M. J. McKeown, L. K. Hansen, and T. J. Sejnowsk, "Independent component analysis of functional mri: What is signal and what is noise?" *Current opinion in neurobiology*, vol. 13, no. 5, pp. 620–629, 2003.

[111] S. M. Smith *et al.*, "Temporally-independent functional modes of spontaneous brain activity," *Proceedings of the National Academy of Sciences*, vol. 109, no. 8, pp. 3131–3136, 2012.

[112] K. J. Friston, "Modes or models: A critique on independent component analysis for fmri," *Trends in cognitive sciences*, vol. 2, no. 10, pp. 373–375, 1998.

[113] S. Dodel, J. M. Herrmann, and T. Geisel, "Comparison of temporal and spatial ica in fmri data analysis," in *Proceedings of ICA2000, the Second International Conference on Independent Component Analysis and Signal Separation*, 2000, pp. 543–547.

[114] K. Petersen, L. K. Hansen, and T. Kolenda, "On the independent components of functional neuroimages."

[115] V. D. Calhoun, T. Adali, V. McGinty, J. J. Pekar, T. Watson, and G. Pearlson, "Fmri activation in a visual-perception task: Network of areas detected using the general linear model and independent components analysis," *NeuroImage*, vol. 14, no. 5, pp. 1080–1088, 2001.

[116] R. N. Boubela, K. Kalcher, W. Huf, C. Kronnerwetter, P. Filzmoser, and E. Moser, "Beyond noise: Using temporal ICA to extract meaningful information from high-frequency fMRI signal fluctuations during rest," *Frontiers in Human Neuroscience*, vol. 7, p. 168, May 2013.

[117] M. F. Glasser *et al.*, "Using temporal ica to selectively remove global noise while preserving global signal in functional mri data," *bioRxiv*, p. 193 862, 2017.

[118] E. B. Beall and M. J. Lowe, "Isolating physiologic noise sources with independently determined spatial measures," *Neuroimage*, vol. 37, no. 4, pp. 1286–1300, 2007.

[119] V. D. Calhoun, T. Adalı, G. D. Pearlson, and J. Pekar, "A method for making group inferences from functional MRI data using independent component analysis," *Human Brain Mapping*, vol. 14, no. 3, pp. 140–151, Aug. 2001.

[120] N. Correa, T. Adalı, and V. D. Calhoun, "Performance of blind source separation algorithms for fMRI analysis using a group ICA method," *Magnetic Resonance Imaging*, vol. 25, no. 5, pp. 684–694, Jun. 2007.

[121] E. Seifritz *et al.*, "Spatiotemporal pattern of neural processing in the human auditory cortex," *Science*, vol. 297, no. 5587, pp. 1706–1708, Sep. 2002.

[122] V. van de Ven, F. Esposito, and I. K. Christoffels, "Neural network of speech monitoring overlaps with overt speech production and comprehension networks: A sequential spatial and temporal ica study," *Neuroimage*, vol. 47, no. 4, pp. 1982–1991, 2009.

[123] T. Eichele, S. Rachakonda, B. Brakedal, R. Eikeland, and V. D. Calhoun, "EEGIFT: A toolbox for group temporal ICA event-related EEG," *Computational Intelligence and Neuroscience*, vol. 2011, Article ID 129365, 2011.

[124] S. Rachakonda, R. F. Silva, J. Liu, and V. D. Calhoun, "Memory efficient PCA methods for large group ICA," *Frontiers in Neuroscience*, vol. 10, p. 17, 2016.

[125] Z.-J. Bai, R. Chan, and F. Luk, "Principal component analysis for distributed data sets with updating," in *APPT 2005: Advanced Parallel Processing Technologies*, ser. Lecture Notes in Computer Science, vol. 3756, Hong Kong, China: Springer, 2005, pp. 471–483.

[126] S.-i. Amari, A. Cichocki, and H. H. Yang, "A new learning algorithm for blind signal separation," *Advances in NIPS*, pp. 757–763, 1996.

[127] S.-I. Amari, T.-P. Chen, and A. Cichocki, "Stability analysis of learning algorithms for blind source separation," *Neural Networks*, vol. 10, pp. 1345–1351, 1997.

[128] H. Imtiaz and A. D. Sarwate, "Differentially private distributed principal component analysis," in *Proceedings of the 43rd IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP 2018)*, Calgary, Canada, Apr. 2018.

[129] V. D. Calhoun, R. F. Silva, T. Adalı, and S. Rachakonda, "Comparison of PCA approaches for very large group ICA," *NeuroImage*, vol. 118, pp. 662–666, 2015.

[130] MATLAB, *Rand:uniformly distributed random numbers*, mathworks.

[131] E. Erhardt, E. Allen, Y. Wei, T. Eichele, and V. Calhoun, "SimTB, a simulation toolbox for fMRI data under a model of spatiotemporal separability," *NeuroImage*, vol. 59, no. 4, pp. 4160–4167, 2012.

[132] R. Engle, "Autoregressive conditional heteroscedasticity with estimates of the variance of United Kingdom inflation," *Econometrica*, vol. 50, no. 4, pp. 987–1007, 1982.

[133] T. Bollerslev, "Generalized autoregressive conditional heteroskedasticity," *Journal of Econometrics*, vol. 31, no. 3, pp. 307–327, Apr. 1986.

[134] K. Zhang and A. Hyvärinen, "Source separation and higher-order causal analysis of MEG and EEG," in *26th Conference on Uncertainty in Artificial Intelligence (UAI 2010)*, Catalina Island, California: AUAI Press, 2012, pp. 709–716.

[135] T. Ozaki, *Time Series Modeling of Neuroscience Data*, 1st. Boca Raton, FL, USA: CRC Press, 2012.

[136] Q. Luo, G. Tian, F. Grabenhorst, J. Feng, and E. Rolls, "Attention-dependent modulation of cortical taste circuits revealed by Granger causality with signal-dependent noise," *PLoS Comp. Bio.*, vol. 9, no. 10, e1003265, Oct. 2013.

[137] M. Lindquist, Y. Xu, M. Nebel, and B. Caffo, "Evaluating dynamic bivariate correlations in resting-state fMRI: A comparison study and a new approach," *NeuroImage*, vol. 101, pp. 531–546, 2014.

[138] R. Balan, "Estimator for number of sources using minimum description length criterion for blind sparse source mixtures," in *ICA 2007: Independent Component*

*Analysis and Signal Separation*, ser. Lecture Notes in Computer Science, D. M. E., J. C. J., A. S. A., and P. M. D., Eds., vol. 4666, Springer, 2007, pp. 333–340.

[139] E. Egolf, K. A. Kiehl, and V. D. Calhoun, "Group ICA of fMRI toolbox (GIFT)," in *Proceedings of Human Brain Mapping*, Budapest, Hungary, 2004.

[140] M. R. N. Medical Image Analysis Lab, *Group ICA Of fMRI Toolbox (GIFT)*.

[141] H. W. Kuhn, "The Hungarian method for the assignment problem," in *50 Years of Integer Programming 1958-2008*, M. Jünger *et al.*, Eds., Berlin, Heidelberg: Springer, 2010, pp. 29–47.

[142] B. Hitaj, G. Ateniese, and F. Perez-Cruz, "Deep models under the gan: Information leakage from collaborative deep learning," in *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*, ACM, 2017, pp. 603–618.

[143] X.-L. Li and T. Adali, "Complex independent component analysis by entropy bound minimization," *IEEE Transactions on Circuits and Systems I: Regular Papers*, vol. 57, no. 7, pp. 1417–1430, Jul. 2010.

[144] H. Gazula *et al.*, "Decentralized analysis of brain imaging data: Voxel-based morphometry and dynamic functional network connectivity," *Frontiers in neuroinformatics*, vol. 12, p. 55, 2018.

[145] M. P. Van Den Heuvel and H. E. H. Pol, "Exploring the brain network: A review on resting-state fmri functional connectivity," *European neuropsychopharmacology*, vol. 20, no. 8, pp. 519–534, 2010.

[146] E. A. Allen, E. Damaraju, S. M. Plis, E. B. Erhardt, T. Eichele, and V. D. Calhoun, "Tracking whole-brain connectivity dynamics in the resting state," *Cerebral cortex*, vol. 24, no. 3, pp. 663–676, 2014.

[147] Ü. Sakoğlu, G. D. Pearlson, K. A. Kiehl, Y. M. Wang, A. M. Michael, and V. D. Calhoun, "A method for evaluating dynamic functional network connectivity and task-modulation: Application to schizophrenia," *Magnetic Resonance Materials in Physics, Biology and Medicine*, vol. 23, no. 5-6, pp. 351–366, 2010.

[148] E. Damaraju *et al.*, "Dynamic functional connectivity analysis reveals transient states of dysconnectivity in schizophrenia," *NeuroImage: Clinical*, vol. 5, pp. 298–308, 2014.

[149] B. Rashid, E. Damaraju, G. D. Pearlson, and V. D. Calhoun, "Dynamic connectivity states estimated from resting fmri identify differences among schizophrenia,

bipolar disorder, and healthy control subjects," *Frontiers in human neuroscience*, vol. 8, p. 897, 2014.

[150] S. Silva, B. A. Gutman, E. Romero, P. M. Thompson, A. Altmann, and M. Lorenzi, "Federated learning in distributed medical databases: Meta-analysis of large-scale subcortical brain data," in *2019 IEEE 16th International Symposium on Biomedical Imaging (ISBI 2019)*, IEEE, 2019, pp. 270–274.

[151] V. Smith, C.-K. Chiang, M. Sanjabi, and A. S. Talwalkar, "Federated multi-task learning," in *Advances in Neural Information Processing Systems*, 2017, pp. 4424–4434.

[152] R. C. Geyer, T. Klein, and M. Nabi, "Differentially private federated learning: A client level perspective," *arXiv preprint arXiv:1712.07557*, 2017.

[153] F. Sattler, S. Wiedemann, K.-R. Müller, and W. Samek, "Robust and communication-efficient federated learning from non-iid data," *IEEE transactions on neural networks and learning systems*, 2019.

[154] K. Bonawitz *et al.*, "Towards federated learning at scale: System design," *arXiv preprint arXiv:1902.01046*, 2019.

[155] S. W. Remedios *et al.*, "Distributed deep learning across multisite datasets for generalized ct hemorrhage segmentation," *Medical physics*, vol. 47, no. 1, pp. 89–98, 2020.

[156] B. T. Baker, R. F. Silva, V. D. Calhoun, A. D. Sarwate, and S. M. Plis, "Large scale collaboration with autonomy: Decentralized data ica," in *2015 IEEE 25th International Workshop on Machine Learning for Signal Processing (MLSP)*, IEEE, 2015, pp. 1–6.

[157] S. Rachakonda, R. F. Silva, J. Liu, and V. D. Calhoun, "Memory efficient pca methods for large group ica," *Frontiers in neuroscience*, vol. 10, p. 17, 2016.

[158] V. J. Schmithorst and S. K. Holland, "Comparison of three methods for generating group statistical inferences from independent component analysis of functional magnetic resonance imaging data," *Journal of Magnetic Resonance Imaging: An Official Journal of the International Society for Magnetic Resonance in Medicine*, vol. 19, no. 3, pp. 365–368, 2004.

[159] E. B. Erhardt, S. Rachakonda, E. J. Bedrick, E. A. Allen, T. Adali, and V. D. Calhoun, "Comparison of multi-subject ica methods for analysis of fmri data," *Human brain mapping*, vol. 32, no. 12, pp. 2075–2095, 2011.

[160] I. S. Dhillon and D. S. Modha, "A data-clustering algorithm on distributed memory multiprocessors," in *Large-Scale Parallel Data Mining*, Springer, 2002, pp. 245–260.

[161] G. Jagannathan and R. N. Wright, "Privacy-preserving distributed k-means clustering over arbitrarily partitioned data," in *Proceedings of the eleventh ACM SIGKDD international conference on Knowledge discovery in data mining*, ACM, 2005, pp. 593–599.

[162] G. Jagannathan, K. Pillaipakkamnatt, and R. N. Wright, "A new privacy-preserving distributed k-clustering algorithm," in *Proceedings of the 2006 SIAM International Conference on Data Mining*, SIAM, 2006, pp. 494–498.

[163] S. Datta, C. Giannella, and H. Kargupta, "K-means clustering over a large, dynamic network," in *Proceedings of the 2006 SIAM International Conference on Data Mining*, SIAM, 2006, pp. 153–164.

[164] S. Datta, C. Giannella, and H. Kargupta, "Approximate distributed k-means clustering over a peer-to-peer network," *IEEE Transactions on Knowledge and Data Engineering*, vol. 21, no. 10, pp. 1372–1388, 2009.

[165] G. Di Fatta, F. Blasa, S. Cafiero, and G. Fortino, "Fault tolerant decentralised k-means clustering for asynchronous large-scale networks," *Journal of Parallel and Distributed Computing*, vol. 73, no. 3, pp. 317–329, 2013.

[166] L. Bottou and Y. Bengio, "Convergence properties of the k-means algorithms," in *Advances in neural information processing systems*, 1995, pp. 585–592.

[167] A. K. Cline and I. S. Dhillon, "Computation of the singular value decomposition," 2006.

[168] D. Arthur and S. Vassilvitskii, "K-means++: The advantages of careful seeding," Stanford, Tech. Rep., 2006.

[169] B. Bahmani, B. Moseley, A. Vattani, R. Kumar, and S. Vassilvitskii, "Scalable k-means++," *arXiv preprint arXiv:1203.6402*, 2012.

[170] O. Bachem, M. Lucic, S. H. Hassani, and A. Krause, "Approximate k-means++ in sublinear time," in *Thirtieth AAAI Conference on Artificial Intelligence*, 2016.

[171] S. G. Potkin and J. M. Ford, "Widespread cortical dysfunction in schizophrenia: The fbirn imaging consortium," *Schizophrenia Bulletin*, vol. 35, no. 1, pp. 15–18, 2008.

[172] F. Pedregosa *et al.*, "Scikit-learn: Machine learning in python," *Journal of machine learning research*, vol. 12, no. Oct, pp. 2825–2830, 2011.

[173] H. W. Kuhn, "On the origin of the hungarian method," *History of mathematical programming*, pp. 77–81, 1991.

[174] C. Dwork, "Differential privacy: A survey of results," in *International conference on theory and applications of models of computation*, Springer, 2008, pp. 1–19.

[175] C. Dwork, A. Roth, *et al.*, "The algorithmic foundations of differential privacy," *Foundations and Trends® in Theoretical Computer Science*, vol. 9, no. 3–4, pp. 211–407, 2014.

[176] L. Bottou, "Large-scale machine learning with stochastic gradient descent," in *Proceedings of COMPSTAT'2010*, Springer, 2010, pp. 177–186.

[177] J. Verbraeken, M. Wolting, J. Katzy, J. Kloppenburg, T. Verbelen, and J. S. Rellermeyer, "A survey on distributed machine learning," *ACM Computing Surveys (CSUR)*, vol. 53, no. 2, pp. 1–33, 2020.

[178] A. Sergeev and M. Del Balso, "Horovod: Fast and easy distributed deep learning in TensorFlow," *arXiv preprint arXiv:1802.05799*, 2018.

[179] A. Paszke *et al.*, "Pytorch: An imperative style, high-performance deep learning library," *Advances in neural information processing systems*, vol. 32, 2019.

[180] B. Speelpenning, "Compiling fast partial derivatives of functions given by algorithms," AAI8017989, Ph.D. dissertation, Department of Computer Science, University of Illinois at Urbana-Champaign, USA, 1980.

[181] C. J. Shallue, J. Lee, J. Antognini, J. Sohl-Dickstein, R. Frostig, and G. E. Dahl, "Measuring the effects of data parallelism on neural network training," *Journal of Machine Learning Research*, vol. 20, pp. 1–49, 2019, Also arXiv:1811.03600v3.

[182] Z. Zhang, C. Chang, H. Lin, Y. Wang, R. Arora, and X. Jin, "Is network the bottleneck of distributed training?" In *Proceedings of the Workshop on Network Meets AI & ML*, 2020, pp. 8–13.

[183] A. Svyatkovskiy, J. Kates-Harbeck, and W. Tang, "Training distributed deep recurrent neural networks with mixed precision on gpu clusters," in *Proceedings of the Machine Learning on HPC Environments*, 2017, pp. 1–8.

[184] Y. Li *et al.*, "A network-centric hardware/algorithm co-design to accelerate distributed training of deep neural networks," in *2018 51st Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*, IEEE, 2018, pp. 175–188.

[185] J. Dean *et al.*, "Large scale distributed deep networks," in *Advances in neural information processing systems*, 2012, pp. 1223–1231.

[186] Z. Tang, S. Shi, X. Chu, W. Wang, and B. Li, "Communication-efficient distributed deep learning: A comprehensive survey," *arXiv preprint arXiv:2003.06307*, 2020.

[187] A. Bagnall *et al.*, "The UEA multivariate time series classification archive, 2018," *arXiv preprint arXiv:1811.00075*, 2018.

[188] A. Dosovitskiy *et al.*, "An image is worth 16x16 words: Transformers for image recognition at scale," *arXiv preprint arXiv:2010.11929*, 2020.

[189] A. G. Baydin, B. A. Pearlmutter, A. A. Radul, and J. M. Siskind, "Automatic differentiation in machine learning: A survey," *Journal of Machine Learning Research*, vol. 18, 2018.

[190] J. Ming *et al.*, "Coinstac: Decentralizing the future of brain imaging analysis," *F1000Research*, vol. 6, 2017.

[191] B. A. Pearlmutter, "Fast exact multiplication by the Hessian," *Neural Computation*, vol. 6, no. 1, pp. 147–160, 1994.

[192] N. Agarwal, B. Bullins, and E. Hazan, "Second-order stochastic optimization in linear time," *Journal of Machine Learning Research*, vol. 18, no. 116, pp. 1–40, 2017, Also arXiv:1602.03943.

[193] R. Meyer, D. A. Fournier, and A. Berg, "Stochastic volatility: Bayesian computation using automatic differentiation and the extended Kalman filter," *The Econometrics Journal*, vol. 6, no. 2, pp. 408–420, 2003.

[194] S. Goldt, M. Advani, A. M. Saxe, F. Krzakala, and L. Zdeborová, "Dynamics of stochastic gradient descent for two-layer neural networks in the teacher-student setup," *Advances in neural information processing systems*, vol. 32, 2019.

[195] S. Goldt, M. S. Advani, A. M. Saxe, F. Krzakala, and L. Zdeborová, "Generalisation dynamics of online learning in over-parameterised neural networks," *arXiv preprint arXiv:1901.09085*, 2019.

[196] A. Saxe, S. Sodhani, and S. J. Lewallen, "The neural race reduction: Dynamics of abstraction in gated networks," in *International Conference on Machine Learning*, PMLR, 2022, pp. 19 287–19 309.

[197] M. S. Advani, A. M. Saxe, and H. Sompolinsky, "High-dimensional dynamics of generalization error in neural networks," *Neural Networks*, vol. 132, pp. 428–446, 2020.

[198] V. Kothapalli, "Neural collapse: A review on modelling principles and generalization," *arXiv preprint arXiv:2206.04041*, 2023.

[199] C. Yaras, P. Wang, Z. Zhu, L. Balzano, and Q. Qu, "Neural collapse with normalized features: A geometric analysis over the riemannian manifold," *arXiv preprint arXiv:2209.09211*, 2022.

[200] T. Tirer and J. Bruna, "Extended unconstrained features model for exploring deep neural collapse," in *International Conference on Machine Learning*, PMLR, 2022, pp. 21 478–21 505.

[201] J. Zhou *et al.*, "Are all losses created equal: A neural collapse perspective," *arXiv preprint arXiv:2210.02192*, 2022.

[202] T. Galanti, A. György, and M. Hutter, "On the role of neural collapse in transfer learning," *arXiv preprint arXiv:2112.15121*, 2021.

[203] C. Thrampoulidis, G. R. Kini, V. Vakilian, and T. Behnia, "Imbalance trouble: Revisiting neural-collapse geometry," *Advances in Neural Information Processing Systems*, vol. 35, pp. 27 225–27 238, 2022.

[204] V. Papyan, "The full spectrum of deepnet Hessians at scale: Dynamics with SGD training and sample size," *arXiv preprint arXiv:1811.07062*, 2018.

[205] S. Dittmer, E. J. King, and P. Maass, "Singular values for ReLU layers," *IEEE Transactions on Neural Networks and Learning Systems*, vol. 31, no. 9, pp. 3594–3605, 2019.

[206] M. C. Mozer, "A focused backpropagation algorithm for temporal pattern recognition. backpropagation: Theory, architectures, and applications. researchgate," *ResearchGate. Hillsdale, NJ: Lawrence Erlbaum Associates*, pp. 137–169, 1995.

[207] M. Abadi *et al.*, "Tensorflow: A system for large-scale machine learning.," in *Osdi*, Savannah, GA, USA, vol. 16, 2016, pp. 265–283.

[208] X. Zhan, "Inequalities for the singular values of Hadamard products," *SIAM Journal on Matrix Analysis and Applications*, vol. 18, no. 4, pp. 1093–1095, 1997.

[209] H. Daneshmand, J. Kohler, F. Bach, T. Hofmann, and A. Lucchi, "Batch normalization provably avoids ranks collapse for randomly initialised deep networks," *Advances in Neural Information Processing Systems*, vol. 33, pp. 18 387–18 398, 2020.

[210] A. Krizhevsky, G. E. Hinton, *et al.*, "Learning multiple layers of features from tiny images," 2009.

[211] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei, "Imagenet: A large-scale hierarchical image database," in *2009 IEEE conference on computer vision and pattern recognition*, Ieee, 2009, pp. 248–255.

[212] S. Merity, "The wikitext long term dependency language modeling dataset," *Salesforce Metamind*, vol. 9, 2016.

[213] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2016, pp. 770–778.

[214] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," *arXiv preprint arXiv:1409.1556*, 2014.

[215] J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova, "Bert: Pre-training of deep bidirectional transformers for language understanding," *arXiv preprint arXiv:1810.04805*, 2018.

[216] G. Lample and A. Conneau, "Cross-lingual language model pretraining," *arXiv preprint arXiv:1901.07291*, 2019.

[217] J. L. Elman, "Finding structure in time," *Cognitive science*, vol. 14, no. 2, pp. 179–211, 1990.

[218] D. A. Cox, J. Little, and D. O'shea, *Using algebraic geometry*. Springer Science & Business Media, 2005, vol. 185.

[219] V. S. Bawa and V. Kumar, "Linearized sigmoidal activation: A novel activation function with tractable non-linear characteristics to boost representation capability," *Expert Systems with Applications*, vol. 120, pp. 346–356, 2019.

[220] M. Sundararajan, A. Taly, and Q. Yan, "Axiomatic attribution for deep networks," in *International conference on machine learning*, PMLR, 2017, pp. 3319–3328.

[221] R. R. Selvaraju, M. Cogswell, A. Das, R. Vedantam, D. Parikh, and D. Batra, "Grad-cam: Visual explanations from deep networks via gradient-based localization," in *Proceedings of the IEEE international conference on computer vision*, 2017, pp. 618–626.

[222] D. Baehrens, T. Schroeter, S. Harmeling, M. Kawanabe, K. Hansen, and K.-R. Müller, "How to explain individual classification decisions," *The Journal of Machine Learning Research*, vol. 11, pp. 1803–1831, 2010.

[223] K. Simonyan, A. Vedaldi, and A. Zisserman, "Deep inside convolutional networks: Visualising image classification models and saliency maps," *arXiv preprint arXiv:1312.6034*, 2013.

[224] S. Bach, A. Binder, G. Montavon, F. Klauschen, K.-R. Müller, and W. Samek, "On pixel-wise explanations for non-linear classifier decisions by layer-wise relevance propagation," *PloS one*, vol. 10, no. 7, e0130140, 2015.

[225] A. Shrikumar, P. Greenside, and A. Kundaje, "Learning important features through propagating activation differences," in *International conference on machine learning*, PMLR, 2017, pp. 3145–3153.

[226] D. Smilkov, N. Thorat, B. Kim, F. Viégas, and M. Wattenberg, "Smoothgrad: Removing noise by adding noise," *arXiv preprint arXiv:1706.03825*, 2017.

[227] G. Montavon, S. Lapuschkin, A. Binder, W. Samek, and K.-R. Müller, "Explaining nonlinear classification decisions with deep taylor decomposition," *Pattern recognition*, vol. 65, pp. 211–222, 2017.

[228] W. Samek, A. Binder, G. Montavon, S. Lapuschkin, and K.-R. Müller, "Evaluating the visualization of what a deep neural network has learned," *IEEE transactions on neural networks and learning systems*, vol. 28, no. 11, pp. 2660–2673, 2016.

[229] P. Sturmfels, S. Lundberg, and S.-I. Lee, "Visualizing the impact of feature attribution baselines," *Distill*, vol. 5, no. 1, e22, 2020.

[230] A. Kapishnikov, T. Bolukbasi, F. Viégas, and M. Terry, "Xrai: Better attributions through regions," in *Proceedings of the IEEE/CVF International Conference on Computer Vision*, 2019, pp. 4948–4957.

[231] S. Xu, S. Venugopalan, and M. Sundararajan, "Attribution in scale and space," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2020, pp. 9680–9689.

[232] A. Kapishnikov, S. Venugopalan, B. Avci, B. Wedin, M. Terry, and T. Bolukbasi, "Guided integrated gradients: An adaptive path method for removing noise," in *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, 2021, pp. 5050–5058.

[233] M. M. Rahman, N. Lewis, and S. Plis, "Geometrically guided saliency maps," in *ICLR 2022 Workshop on PAIR $\{\backslash$textasciicircum$\}$ 2Struct: Privacy, Accountability, Interpretability, Robustness, Reasoning on Structured Data*, 2022.

[234] M. M. Rahman, N. Lewis, and S. Plis, "Geometrically guided integrated gradients," *arXiv preprint arXiv:2206.05903*, 2022.

[235] B. T. Baker, N. Lewis, D. Saha, M. A. Rahaman, S. Plis, and V. Calhoun, "Information bottleneck for multi-task lstms," in *NeurIPS 2022 Workshop on Information-Theoretic Principles in Cognitive Systems*, 2022.

[236] N. Tishby, F. C. Pereira, and W. Bialek, "The information bottleneck method," *arXiv preprint physics/0004057*, 2000.

[237] A. R. Mayer *et al.*, "Functional imaging of the hemodynamic sensory gating response in schizophrenia," *Human brain mapping*, vol. 34, no. 9, pp. 2302–2312, 2013.

[238] P. Patel, P. Aggarwal, and A. Gupta, "Classification of schizophrenia versus normal subjects using deep learning," in *Proceedings of the Tenth Indian Conference on Computer Vision, Graphics and Image Processing*, 2016, pp. 1–6.

[239] J. Oh, B.-L. Oh, K.-U. Lee, J.-H. Chae, and K. Yun, "Identifying schizophrenia using structural mri with a deep learning algorithm," *Frontiers in psychiatry*, vol. 11, p. 16, 2020.

[240] Y. Zhu, S. Fu, S. Yang, P. Liang, and Y. Tan, "Weighted deep forest for schizophrenia data classification," *IEEE Access*, vol. 8, pp. 62 698–62 705, 2020.

[241] U. Mahmood *et al.*, "Whole milc: Generalizing learned dynamics across tasks, datasets, and populations," in *Medical Image Computing and Computer Assisted Intervention–MICCAI 2020: 23rd International Conference, Lima, Peru, October 4–8, 2020, Proceedings, Part VII 23*, Springer, 2020, pp. 407–417.

[242] Y. Du *et al.*, "Neuromark: An automated and adaptive ica based pipeline to identify reproducible fmri markers of brain disorders," *NeuroImage: Clinical*, vol. 28, p. 102 375, 2020.

[243] A. Vaswani *et al.*, "Attention is all you need," *arXiv preprint arXiv:1706.03762*, 2017.