Sparse Learning in Brain Networks and Brain-like Artificial Intelligence

A Dissertation

Submitted to the Faculty

of

WORCESTER POLYTECHNIC INSTITUTE

In partial fulfillment of the requirements for the

Degree of Doctor of Philosophy

in

Data Science

by

Hang Yin

April 10, 2022

APPROVED:

Professor Xiangnan Kong Worcester Polytechnic Institute Advisor	Professor Randy C. Paffenroth Worcester Polytechnic Institute Committee Member
Professor Ziming Zhang Worcester Polytechnic Institute Committee Member	Professor Tian Guo Worcester Polytechnic Institute Committee Member
Professor Sihong Xie Lehigh University External Committee Member	

Abstract

Sparsity is one of the concepts that can improve effectiveness of machine learning methods. A sparse machine learning model is one in which only a relatively small number of parameters or predictors play an important role. The main benefits of sparsity in machine learning include: simplifying the model for improving interpretability; reducing runtime and generalization error. Our works in this dissertation attempt to study methods that exploit sparsity to help recover the underlying information in many real-world applications. In particular, our works involve two different domains: sparse graph discovery of brain network, and spike trains classification on resource-constrained devices.

1. Brain Network Discovery. The network discovery of brain is useful in many ways such as aiding brain disease diagnosis and understanding cognitive behaviors. The goal of brain network discovery is to discover both brain regions (nodes) and functional connections (edges) between these regions from fMRI scan of human brain. In this research, we focus on extending current sparse Gaussian graphical models by discovering the nodes and edges simultaneously through the model

with mathematical consistency. Because sparse graphic models can simplify the network representation by only discovering meaningful edges of brain network. Moreover, the node discovery and edge detection facilitate each other in a synergistic fashion, which improves the discovered network. Besides, we further reform the problem of multi-state brain network and propose two approaches to solve this problem.

2. Spike Train Classification. Spike train classification is an important problem in many areas such as healthcare and mobile sensing, where each spike train is a high-dimensional time series of binary values. Conventional research on spike train classification mainly focus on developing Spiking Neural Networks (SNNs) under resource-sufficient settings (e.g., on GPU servers). However, in many real-world applications, we often need to deploy the SNN models on resource-constrained platforms (e.g., mobile devices) to analyze high-dimensional spike train data. We propose two extensions of SNN to improve the performance of accuracy and computational cost on spike trains related tasks. One of them is based on the re-parameterization of parameters in an SNN and the application of sparsity regularization during optimization. The other one is proposed to quickly find sparse and useful signals from long and noisy spike trains.

Acknowledgements

I would like to express my deepest gratitude to those who have helped me get to this point.

Firstly, I want to express my sincere appreciate to Prof. Xiangnan Kong. Thank you for being a very hands-on advisor, showing me firsthand the value of hard work and tenacity. My works would have never been possible without your noble guidance, patience, and insightful comments. Those have definitely helped me improve the way I do research.

Second, I also want to thank Prof. Randy Paffenroth, Prof. Ziming Zhang, Prof. Tian Guo, and Prof. Sihong Xie for agreeing to serve on my dissertation committee and showing insightful comments and suggestions during presentations of my dissertation.

Last but not least, as a PhD student, I have had the opportunity to interact with many wonderful individuals. More often than not I would walk away from a conversation with a deeper understanding of a problem. In no particular order I would like to thank Prof. Liping Liu, Prof. Yanhua Li, Dr. Xin Dai, Dr. Xinyue Liu, Dr. John Boaz Lee and Dr. Thomas Hartvigsen for their contributions in my research projects. My

thank you also goes to Dr. Chong Zhou, Jianjun Luo, Yao Su for their support and friendship. Last but not the least, I would like to thank my parents for the unconditional love.

Publications

• [KDD'21] **Hang Yin**, John Lee, Xiangnan Kong, Thomas Hartvigsen, Sihong Xie. *Energy-Efficient Models for High-Dimensional Spike Train Classification using Sparse Spiking Neural Networks.* ACM SIGKDD Conference on Knowledge Discovery and Data Mining (KDD), 2021.

• [BIGDATA'20] **Hang Yin**, Xinyue Liu, Xiangnan Kong. *Gaussian Mixture Graphical Lasso with Application to Edge Detection in Brain Networks.* IEEE International Conference on Big Data (BIGDATA), 2020.

• [ICDM'18] **Hang Yin**, Xinyue Liu, Xiangnan Kong. *Coherent Graphical Lasso for Brain Network Discovery.* IEEE International Conference on Data Mining (ICDM), 2018.

Contents

List of Figures

List of Tables

1

Introduction

1.1 Motivation

Nowadays, large quantities of data are collected and mined in nearly every area of science, entertainment, business, and industry. There is a crucial need to sort through this mass of information, and pare it down to its bare essentials. To this end, we hope that the world is not as complex as it might be. For example, not all of the genes in the human body are directly involved in the process that leads to the development of cancer. Or that seeing a part of the picture maybe enough for us to recognize the objects in it. This points to an underlying assumption of simplicity. One form of simplicity in machine learning area is called sparsity. A sparse machine learning model is one in which only a relatively small number of parameters or predictors play an important role. The main benefits of sparsity can be summarized into three aspects: first, simplify complex model for improving interpretability; second, reduce the mounds of matrix multiplication, shortening the

time for inference; third, reduce generalization error leading into more powerful model. Our works in this dissertation attempt to develop methods that exploit sparsity to help recover the underlying information in many real-world applications. In particular, our works involve two different domain: sparse graph discovery of brain network, and spike trains classification on resource-constrained devices.

1.2 Brain network inference

Network discovery has become an important and active research topic in the last decade, where the goal is to discover both nodes and edges from the spatio-temporal signals generated by a networked system. For example, in brain network discovery, researchers are interested in discovering brain regions (nodes) and functional connections (edges) between these regions from fMRI scan of human brain. Sparse gaussian graphic models are a very useful for discovering a meaningful connectivity of brain network based on large-scale dataset by using sparse inverse covariance estimation. In this research, we focus on extending current sparse graphical models by discovering the nodes and edges simultaneously through the model with mathematical consistency. Because sparse modeling for brain network can improve the interpretability of the model, meanwhile, the node discovery and edge detection facilitate each other in a synergistic fashion, which improves the discovered network. Besides, we further reform the problem of multi-state brain network and propose two approaches to solve this problem. We present our efforts on this subject in Chapter 2, all materials in this chapter are from our papers

"Coherent Graphical Lassofor Brain Network Discovery" [1](in Proceedings of the 2018 IEEE International Conference on Data Mining, 2018, IEEE), "Gaussian Mixture Graphical Lasso with Application to Edge Detection in Brain Networks" [2](in Proceedings of the 2020 IEEE International Conference on Big Data, 2020, IEEE), "Multi-State Brain Network Discovery".

1.3 Spike trains inference

Spike trains are sequences of binary events where the spiked time steps can be encoded as 1's, and the remaining time steps can be encoded as 0's. Such data are common to a variety of domains and are classically analogous to electrochemical signals in the human brain. Therefore, spike train classification is an important problem in many areas such as healthcare and mobile sensing. Conventional research on spike train classification mainly focus on developing Spiking Neural Networks (SNNs) under resource-sufficient settings (e.g., on GPU servers). However, in many real-world applications, we often need to deploy the SNN models on resource-constrained platforms (e.g., mobile devices) to analyze high-dimensional spike train data. We propose two extensions of SNN to improve the performance of accuracy and computational cost on spike trains related tasks: (1) The first task is to study the problem of energy-efficient SNNs with sparsely-connected neurons. We propose an SNN model with sparse spatio-temporal coding. Our solution is based on the re-parameterization of weights in an SNN and the application of sparsity regularization during optimization. Chapter 3 presents this model that improves the state-of-art performance on related problem, and all materials in this chapter

are from our previous paper "Energy-Efficient Models for High-Dimensional Spike Train Classification using Sparse Spiking Neural Networks" [3](in Proceedings of the 27th ACM SIGKDD Conference on Knowledge Discovery and Data Mining (KDD), 2021). (2) The second task is proposed to quickly find sparse and useful signals from long and noisy spike trains. The proposed framework consider to extend existing SNN models by learning to mask out noise by skipping membrane potential updates and shortening the effective size of the computational graph. Chapter 4 summarize this model and experimental results on related problem, all materials in this chapter are from our paper "SkipSNN: Efficient Classification of Sparse and Noisy Spike Trains".

2

Brain Network Discovery

2.1 Task 1: Coherent Modeling in Brain Network Discovery

2.1.1 Motivation

Network discovery [4, 5] is to discover both the nodes and edges from a spatio-temporal signals produced by a networked system. It is involved in many areas, especially the study of human brain, which is one of the most sophisticated systems that have attracted a large number of researchers. In a brain network, a node refers to a collection of brain tissues which are coherent in function. An edge between two nodes measures the functional connectivity between pairs of nodes. Constructing a brain network usually consists of two sub-task: one is grouping a set of pixels in a such a way that pixels in the same node are similar to each other in function, which is called *nodes discovery*; The other is finding the correlations

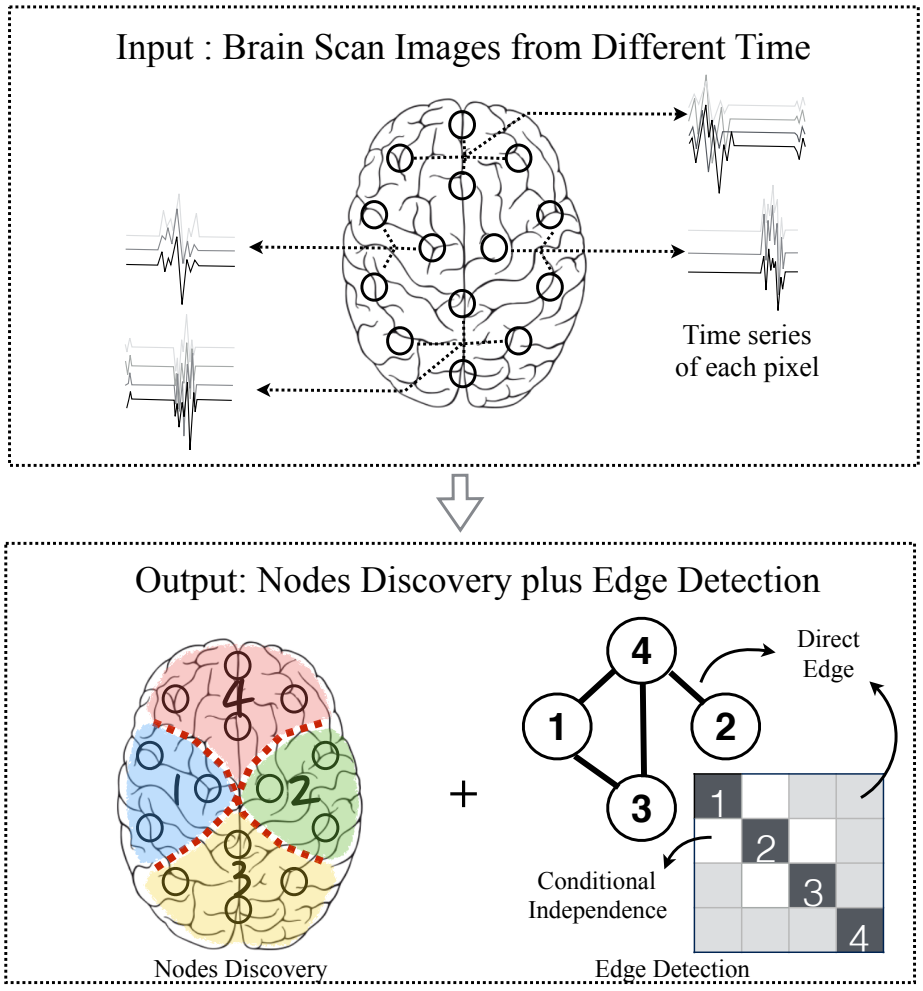


Figure 2.1: *The problem of brain network discovery.*

(edges) between each node, which is called *edge detection*. In this task, we consider to discover the brain network from a collection of fMRI scans, where each scan corresponds to a 4D brain image (a sequence of 3D images) of a subject. These scans are usually transferred into time series of voxels in the 3D images space, and the goal is to discover nodes and edge of the brain network from the time series in an unsupervised fashion. We illustrate this problem in Figure 2.1.

Most of previous studies consider two sub-task of brain network discovery sep-

arately. Some works focus on the task of edge detection only, where one assumes that the nodes of the network are given, either by the anatomical atlas from neuroscience studies or by applying other node discovery methods (such as k-means clustering or spectral clustering). Sparse gaussian graphic models [6, 7, 8, 9, 10] are a great candidate for edge detection of brain network based on large-scale dataset by using sparse inverse covariance estimation. The main ideas of these methods are that they can distinguish direct edges from indirect edges due to their solid probabilistic foundation. Therefore, the results derived from such models have better interpretability performance based on the sparse structure of meaningful edges. However, the brain network derived based upon these assumptions are not optimal and may exhibit inconsistencies due to following reasons. First, the atlas provided by neuroscientist is mainly derived from anatomical studies, and some groups (nodes) in the atlas may contain sub-regions of brain that are characterized by different brain functions. Second, applying separate node discovery method to obtain the clustering (nodes) of brain may preclude one further refines the clustering based upon the discovered connectivity patterns. Therefore, although nodes discovery and edge detection are two different tasks for brain network discovery, the derivations of them are highly affected by each other during the process. Accordingly, a cohesive method for collectively discover nodes and edges of a brain network is more desired than independent discovery methods or pipeline discovery methods.

Toward this end, some coherent methods for brain network discovery that can handle both of the two aspects of the problem have been proposed recently [5, 11] . However, they all have their own limitations. Although the method proposed in

[11] considers the brain network discover problem as a coherent one, they apply two separate objective functions for two sub-task respectively, and update each other alternatively in the same framework. Besides, they employ spectral clustering for the nodes discovery task, which loses interpretation in the terms of nature properties of brain parcellation. The method proposed in [5] aims at discovering nodes with spatial continuity constraint for the sake of interpretability, but it fails to distinguish the direct connections and indirect connections among nodes in the network.

To address above issues of the existing methods, we propose a novel model that could perform node discovery and edge detection at the same time without any prior knowledge of the brain. Our main challenges are as follows:

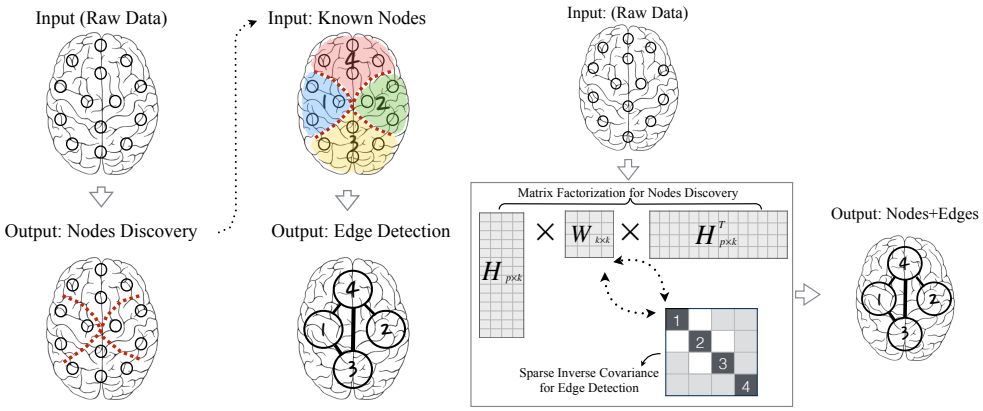
- **Edge detection without given nodes**: The first challenge we faced is how to find connectivity between different nodes without background knowledge of these nodes. Most of previous methods dealing with inter-node edge detection, require given nodes as input, which is contradictory with real cases. In reality, the given nodes maybe inferred anatomically and contain sub-regions that are each characterized by different functional connectivity patterns. It may limit the quality and utility of the inferred network. In addition, we hope to distinguish partial interactions among all edges, which is more instructive in further study of the inferred brain network. So when we are designing a coherent discovery model for brain network, how to infer a sparse connectivity network without the requirement of using given nodes as input is the first challenge we need to conquer.

- **Nodes discovery without given edges**: In the consideration of nodes discov-

ery, we have some principles to follow: First, the discovered nodes should be non-overlapped regions of pixels, so we need to make sure the interpretation of the inferred networks; Second, these sub-regions should be parcellated by same functional connectivity patterns. In real cases, the nodes of the brain we have obtained by anatomical atlases are likely to be based on different functional and structural patterns. So the accuracy of edge detection is also important for the results of nodes discovery. However, it is difficult for us to deal with it without given edges as input. Consequently, it is the second challenge for us to solve nodes discovery via a coherent model, due to input of only raw time series transferred from brain scan image instead of any prior information of the connectivity of brain network.

- **Designing a coherent objective function instead of combining two independent functions in a framework**: In the consideration of this coherent model itself, we need to design a model different from the existing model in [11]. If following the idea introduced in [11], we can just find two models for nodes discovery and edge detection, respectively, and then combining the two objective functions of them in one framework, updating each other alternatively. However, it is still an independent method in fact. Due to the independence of optimization functions for nodes discovery and edge detection, this method has the risk of estimation inconsistency. The third challenge we faced is how to finding a unified optimization function to express the problem of collective cognitive brain network discovery.

To tackle above challenges, we propose a new model, namely CGLasso, to discovery nodes and edges of a brain network coherently. In Figure 2.2, we compare



(a) *Independent Inference* **(b)** *Coherent Inference with Proposed Model*

Figure 2.2: *Comparison of different brain network inference methods. (a) inference with known groups; (b) the proposed collective discovery of brain network. Here* $\mathbf{H}$ *is a* $p \times k$ *indicator matrix for nodes discovery, where* p *is the number of pixels from the brain image,* k *is the number of inferred nodes.* W *is the inter-node covariance matrix.*

the differences among the previous models and our proposed model. Our model is based on Glasso (graphical lasso) and ONMtF (orthogonal non-negative matrix tri-factorization) and combine the ideas of them in a new mathematical model, meanwhile, using an alternative update rule to optimize each other. Our model only needs original brain data (time series of all pixels from brain scan images) as input without any prior knowledge related to nodes or connectivity between them, and then output nodes and edges at the same time. In addition, due to the matrix factorization for nodes discovery and sparse inverse covariance for edge detection, we follow the properties of Glasso and ONMtF in a single model.

2.1.2 Related Works

In this chapter, we discuss some existing methods used in network discovery, especially in brain network analysis. Due to the fact that currently there is no coherent

method that can solve nodes discovery and edge detection in a single model, we will discuss the two sub-problems separately.

In the consideration of edge detection, this issue has two major branches: effective connectivity estimation and functional connectivity estimation. For the first branch, scholars pay more attention on obtaining a directed network from fMRI data through structure learning method for Bayesian networks [12]. In contrast, the second branch focuses on some approaches such as hierarchical clustering, pairwise correlations and independent component analysis, which can be found in [13] for more details. In this chapter of the thesis, we focus on sparse gaussian graphic models [6, 7, 8, 9, 10], which are a very useful for discovering connectivity of brain network based on large-scale dataset by using sparse inverse covariance estimation. The main ideas of these methods are that they can distinguish direct links from indirect connections due to their solid probabilistic foundation. However, in the task of edge detection, these methods assume the nodes of brain network are given, which is opposed to the fact. In the [11], Liu proposes a new method by combining Glasso and spectral clustering under a single framework. This method, to some extent, achieves the idea of completing nodes discovery and edge detection at the same time. The problem is, in this framework, because each task corresponds to a different objective function, the two tasks are still worked separately in fact.

In the task of inferring the nodes of brain network, early works focus on anatomical atlases. However, there is no unified functional or structural connectivity that can be used to construct and explain them, the nodes of the brain we have obtained by anatomical atlases are likely to be based on different functional and

structural patterns. From the perspective of computer science, more and more studies have begun to focus on data-driven methods. [14] has addressed the issue of distinguishing brain nodes that are highly related to Alzheimer's disease from neuroimaging data. Researchers have also begun to parcellated the brain by regarding it as a combination of several functionally homogeneous nodes. For example, [15] used projection methods such as principal component analysis to discover the nodes of brain network that are co-active. However, it can only provide a coarse output of the brain network, due to the parcellation of "in network" and "out of network". [4] handled network discovery via constrained tensor analysis method, but this method is a supervised method, which is less feasible in real cases. [5] proposed to apply matrix tri-factorization with nonnegative and orthogonal constraints (ONMtF) to fMRI data. ONMtF is an extended model of NMF. Here Non-negative constraints lead to a parts-based representation because they allow only additive, not subtractive and combination, meanwhile, orthogonality makes sure the interpretability of the results. [16, 17, 18, 19, 20, 21] have shown it to be useful for pattern recognition and later studies have continued to show that most applications make use of the clustering aspect of NMF. Then [22, 23] show the equivalence between NMF and K-means. [24] extend NMF algorithm by adding a spatial continuity penalty, which can make sure the interpretability of the parcellated regions. This method can also output the result of nodes discovery and edge detection at the same time, however, it has discovered the edges based on the correlation matrix instead of inferring direct links as Glasso does.

Based on the above discussion, we believe that we still have much space for progressing on the network discovery. In Chapter 2.1.4, we propose a new method,

which blends the ideas of Glasso and NMF. In order to introduce the proposed method more smoothly, we will provide more details about two baseline methods at first.

2.1.3 Problem Definition

Now we discuss our proposed model at the functional level and provide more details. First, we introduce two basic methods to deal with network discovery, and then discuss our proposed model which combines the ideas of them. Here we still analyze the network discovery from two perspectives: nodes discovery and edge detection.

Problem1 : Nodes Discovery

Let $\mathbf{X} = (\mathbf{x}_1, ..., \mathbf{x}_n) \in \mathbb{R}^{p \times n}$ be the observations of p-variate Gaussian distribution, where p denotes the number of variables and n denotes the number of observations. Then we set $\mathbf{\Sigma}$ as the covariance matrix of n samples. The NMF factorizes $\mathbf{\Sigma}$ into two non-negative matrices:

$$\mathbf{\Sigma} \approx \mathbf{F}\mathbf{G}^T \tag{2.1}$$

where $\mathbf{F} = (\mathbf{f}_1, ..., \mathbf{f}_k) \in \mathbb{R}^{p \times k}$ and $\mathbf{G} = (\mathbf{g}_1, ..., \mathbf{g}_k) \in \mathbb{R}^{p \times k}$, k is a pre-specified parameter. In the case of the network discovery, our target is an absolute covariance matrix $\mathbf{\Sigma}$. So $\mathbf{\Sigma}$ us a systematic analysis matrix and $\mathbf{F}$ is equal to $\mathbf{G}$, which we denote them as matrix $\mathbf{H}$ in the rest, meanwhile, k can be treated as the number of nodes to discover. According to [24], we can extend NMF model to weighted

orthogonal non-negative factorization, which is called ONMtF, then the objective function is:

$$\min_{\mathbf{H}\geqslant 0, \mathbf{H}\mathbf{H}^T=\mathbf{I}} ||\mathbf{\Sigma} - \mathbf{H}\mathbf{S}\mathbf{H}^T||^2 \tag{2.2}$$

The [23] shows that, by adding non-negativity and orthogonality constraints, the model is equivalent to k-means clustering and the Laplacian-based spectral clustering. Furthermore, the $k \times k$ matrix $\mathbf{S}$ has a special meaning. In the next part, we will introduce how we use this idea in our proposed model to implement collective discovery of network.

Problem2: Edge Detection

In the problem1, we define the systematic covariance matrix $\mathbf{\Sigma}$. In graphic lasso, we assume the precision matrix $\mathbf{\Theta} = \mathbf{\Sigma}^{-1}$. Then according to the [25], the problem of estimate $\mathbf{\Theta}$ can be cast as follows:

$$\min_{\mathbf{\Theta}\succ 0} -\log\det\mathbf{\Theta} + \text{tr}(\mathbf{S}\mathbf{\Theta}) + \lambda||\mathbf{\Theta}||_1 \tag{2.3}$$

where $\mathbf{S} = \frac{1}{n}\mathbf{X}\mathbf{X}^T$ is the empirical covariance matrix, ℓ_1 regularization is used to force sparsity, and $\breve{}$ is the parameter to control the sparseness of $\mathbf{\Theta}$. The edge e_{ij} between $\mathbf{x}_i$ and $\mathbf{x}_j$ exist if and only if $\theta_{ij} \neq 0$, where θ_{ij} is the (i, j)-element of $\mathbf{\Theta}$.

2.1.4 Coherent Graphical Lasso

Problem Formulation: CGLasso

Now we will formulate the network discovery as a graphic lasso with orthogonal non-negative matrix factorization as shown in Equation (2.4):

$$\begin{aligned} \min_{\mathbf{H},\mathbf{\Theta}^\star} & -\log\det\mathbf{\Theta}^\star + \text{tr}(\mathbf{H}^T\mathbf{S}\mathbf{H}\mathbf{\Theta}^\star) + \lambda||\mathbf{\Theta}^\star||_1 \\ \text{s.t.} \quad & \mathbf{\Theta}^\star \succ 0, \mathbf{H} \geqslant 0, \mathbf{H}\mathbf{H}^T = \mathbf{I} \end{aligned} \tag{2.4}$$

where $\mathbf{\Theta}^\star$ is the inverse of the $k \times k$ absolute inter-node covariance matrix, and $\mathbf{H}$ is a $p \times k$ cluster indicator matrix. . We call this model CGLasso. The equation can output the results of nodes discovery and edge detection at the same time. Specifically, because $\mathbf{H}$ is a cluster indicator matrix, the original variable space $\mathbf{X}$ can be mapped to a new feature space similarly on the covariance matrix $\mathbf{\Sigma}$:

$$\begin{aligned} \mathbf{X} \leftarrow \mathbf{Y} &= \mathbf{H}^T\mathbf{X} \\ \mathbf{\Sigma} \leftarrow \mathbf{W} &= \mathbf{H}^T\mathbf{\Sigma}\mathbf{H} \end{aligned} \tag{2.5}$$

where $\mathbf{Y}$ denotes the new k dimensional feature space where each feature represents node, and $\mathbf{W}$ represents the inter-node covariance matrix. According to the weighted NMF model, $\mathbf{W}$ has a special meaning here. When mapping matrix $\mathbf{\Sigma}$ by following Equation (2.27), we can obtain:

$$\mathbf{W}_{lk} = \mathbf{h}_l^T\mathbf{\Sigma}\mathbf{h}_k = \frac{\sum_{i\in\mathbf{C}_l}\sum_{j\in\mathbf{C}_k}\Sigma_{ij}}{\sqrt{n_l n_k}} \tag{2.6}$$

where

$$\mathbf{h}_k = (0, ..., 0, \overbrace{1, ..., 1}^{n_k}, 0, ..., 0)^T / n_k^{1/2} \tag{2.7}$$

In the Equation (2.6), $\mathbf{C}_l$ and $\mathbf{C}_k$ denote the node l and the node k, respectively, meanwhile, n_l and n_k denote the number of variables in these nodes. So we can consider $\mathbf{W}$ as a measure of association between the nodes. Then we rewrite the log-likelihood function of graphic lasso as:

$$\begin{aligned} \text{logL} &= ln\left[\frac{1}{(2\pi)^{\frac{kn}{2}}|\mathbf{W}|^{\frac{1}{2}}}\text{exp}\{-\frac{1}{2}\text{tr}(\mathbf{X}^T\mathbf{H}\mathbf{W}^{-1}\mathbf{H}^T\mathbf{X})\}\right] \\ &= -\frac{kn}{2}ln(2\pi) + \frac{1}{2}ln|\mathbf{W}^{-1}| - \frac{1}{2}\text{tr}(\mathbf{X}^T\mathbf{H}\mathbf{W}^{-1}\mathbf{H}^T\mathbf{X}) \\ &= -\frac{kn}{2}ln(2\pi) + \frac{1}{2}ln|\mathbf{W}^{-1}| - \frac{1}{2}\text{tr}(\mathbf{H}\mathbf{W}^{-1}\mathbf{H}^T\mathbf{S}) \end{aligned} \tag{2.8}$$

In this way, the original output of Glasso (we denotes as $\hat{\mathbf{\Theta}}$) can be transfer into a new result ($\hat{\mathbf{\Theta}}^\star$). We think it makes sense for the reason that, in some real cases, we are more interested in the connectivity among inter-nodes. Meanwhile, it can also provide the estimator of matrix $\mathbf{H}$, thus finishing the nodes discovery and edge detection simultaneously.

Algorithm For Solving CGLasso

We consider the algorithm for our proposed model in this chapter. The specific steps are presented in Algorithm 1. In this solver, as the objective function is non-convex caused by the coupling between $\mathbf{\Theta}^\star$ and $\mathbf{H}$, we choose to alternatively update them in each iteration. Specifically, in each iteration, we hold one parameter and then update the other one. Since these two parameters are involved in the same objective function, each updated $\hat{\mathbf{\Theta}}^\star$ is reflected in the update function of $\hat{\mathbf{H}}$. The same is true on the update of $\hat{\mathbf{\Theta}}^\star$.

Group Inference: To solve the group discovery, we use a multiplicative update

Algorithm 1 Algorithm for `CGLasso`

Require: i: $\mathbf{S}$: the $p \times p$ empirical covariance matrix
ii: k: the number of groups discovered
iii: λ: the parameter to control the sparseness of $\mathbf{\Theta}^\star$
iv: iter$_{max}$: the maximum number of iteration
Output: $\hat{\mathbf{H}}$, $\hat{\mathbf{\Theta}}^\star$

1: Initialization: Randomly initialize $\hat{\mathbf{H}}^{(0)}$
2: Solve Glasso and get initial estimation $\hat{\mathbf{\Theta}}^{\star(0)}$
3: **repeat**
4: Update $\hat{\mathbf{H}}^{(t)}$ with Equation*
5: Update $\hat{\mathbf{\Theta}}^{\star(t)}$ with the algorithm of Glasso
6: **until** $iter = iter_{max}$ or convergence
7: Set $\hat{\mathbf{H}}$ for group discovery
8: Rescale $\hat{\mathbf{\Theta}}^\star$ and set it for edge detection
Return: $\hat{\mathbf{\Theta}}^\star$, $\hat{\mathbf{H}}$

rule, which is the same as NMF. The basic idea is using KKT complementary slackness conditions to enforce the non-negativity and orthogonality constraints. Base on this, we can directly obtain the Lagrangian function of Equation (2.4):

$$\begin{aligned} \mathrm{L}(\mathbf{H}, \mathbf{\Theta}^\star, \lambda_1, \lambda_2) &= -\log\det\mathbf{\Theta}^\star + \mathrm{tr}(\mathbf{H}^T\mathbf{S}\mathbf{H}\mathbf{\Theta}^\star) \\ &+ \lambda||\mathbf{\Theta}^\star||_1 + \mathrm{tr}(\lambda_1(\mathbf{H}^T\mathbf{H} - \mathbf{I})) - \mathrm{tr}(\lambda_2\mathbf{H}^T) \end{aligned} \tag{2.9}$$

Here λ_1, λ_2 are $k \times k$ and $k \times p$ matrices following KKT conditions:

$$\begin{aligned} \lambda_2 &\geqslant 0 \\ \lambda_2 \circ \mathbf{H} &= 0 \end{aligned} \tag{2.10}$$

where $\circ$ denotes the Hadamard product.Then we can get the deviation of matrix $\mathbf{H}$:

$$\frac{\partial \mathbf{L}}{\partial \mathbf{H}_{ij}} = (2\mathbf{S}\mathbf{H}\mathbf{\Theta}^{\star} + 2\mathbf{H}\lambda_1)_{ij} - (\lambda_2)_{ij} \tag{2.11}$$

Combining the above equation with KKT conditions, we have:

$$(\mathbf{S}\hat{\mathbf{H}}\hat{\mathbf{\Theta}}^{\star} + \hat{\mathbf{H}}\lambda_1)_{ij}\hat{\mathbf{H}}_{ij} = 0 \tag{2.12}$$

According to the multiplicative update rule, the successive update of

$$\hat{\mathbf{H}}_{ij}^{t+1} \leftarrow \hat{\mathbf{H}}_{ij}^{t}(\frac{\mathbf{S}\hat{\mathbf{H}}^t\hat{\mathbf{\Theta}}^{\star -} + \hat{\mathbf{H}}^t\lambda_1^-}{\mathbf{S}\hat{\mathbf{H}}^t\hat{\mathbf{\Theta}}^{\star +} + \hat{\mathbf{H}}^t\lambda_1^+})_{ij} \tag{2.13}$$

will converge to a local minima of the objective problem. Compared to additive update rule, [26] prove that this multiplicative update rule is a good compromise between speed and ease of implementation for solving our objective problem. Here in order to make sure the sign of numerator and denominator are all positive, we divide the λ_1 and $\hat{\mathbf{\Theta}}^{\star}$ into two parts, respectively, to make sure each part is non-negative:

$$\begin{aligned} \lambda_1 &= \lambda_1^+ - \lambda_1^- \\ \lambda_1^+ &= \frac{(|\lambda_1| + \lambda_1)}{2} \\ \lambda_1^- &= \frac{(|\lambda_1| - \lambda_1)}{2} \end{aligned} \tag{2.14}$$

The same is true on the $\hat{\mathbf{\Theta}}^{\star}$.

Now we need to determine the Lagrangian multiplier λ_1 to make sure the signs of numerate and denominator of Equation (2.39) are positive. First, it is obvious that summing over index i, we have $(\mathbf{S}\hat{\mathbf{H}}\hat{\mathbf{\Theta}}^{\star} + \hat{\mathbf{H}}\lambda_1)_{ii} = 0$. Thus we obtain the

diagonal elements of the Lagrangian multipliers:

$$(\lambda_1)_{ii} = -(\hat{\mathbf{H}}^T \mathbf{S} \hat{\mathbf{H}} \hat{\mathbf{\Theta}}^\star)_{ii} \tag{2.15}$$

Then for the off-diagonal elements of the Lagrangian multipliers, we ignore the non-negativity constraint on $\mathbf{H}$. By setting $\frac{\partial \mathbf{L}}{\partial \mathbf{H}_{ij}} = 0$, we obtain:

$$(\lambda_1)_{ij} = -(\hat{\mathbf{H}}^T \mathbf{S} \hat{\mathbf{H}} \hat{\mathbf{\Theta}}^\star)_{ij} \tag{2.16}$$

So we have a compact solution for the Lagrangian multipliers

$$\lambda_1 = -\hat{\mathbf{H}}^T \mathbf{S} \hat{\mathbf{H}} \hat{\mathbf{\Theta}}^\star \tag{2.17}$$

According to the expression of update function, we can notice that if $\hat{\mathbf{H}}_{ij}^{(t)}$ in one iteration, it will never jump out from this local sholution. In the process of experiment, we initialize $\hat{\mathbf{H}}^{(0)}$ by k-means clustering, then setting $\hat{\mathbf{H}}^{(0)} \leftarrow \hat{\mathbf{H}}^{(0)} + 0.2$.

Edge Detection: For collaborative filtering, we can update $\hat{\mathbf{H}}$ and $\hat{\mathbf{\Theta}}^\star$ alternatively. When we update $\hat{\mathbf{H}}^{(t)}$ by the above algorithm, the update of $\hat{\mathbf{\Theta}}^{\star(t)}$ will combine it in the objective problem. So the solve of $\hat{\mathbf{\Theta}}^{\star(t)}$ follows the Equation (2.18):

$$\begin{aligned} \hat{\mathbf{\Theta}}^{\star(t)} = \text{argmin} &- \log\det \mathbf{\Theta}^\star \\ &+ \text{tr}(\hat{\mathbf{H}}^{(t)} \mathbf{\Theta}^\star \hat{\mathbf{H}}^{(t)T} \mathbf{S}) + \lambda ||\mathbf{\Theta}^\star||_1 \end{aligned} \tag{2.18}$$

Here the lasso penalty proposed by Tibshirani [27] aims at achieving sparsity in the regression setting. This ℓ_1 penalty function was applied in Glasso by Friedman et al. [25]. He proposed a coordinate descent procedure for solving Equation (2.18), which is remarkaably fast. We will follow this algorithm for updating $\hat{\mathbf{\Theta}}^\star$.

Furthermore, it allows a "warm" start, which allows us to use the estimate for one value of the tuning parameter as the start point for the next value. We can also use the SCAD penalty and the adaptive penalty proposing by Fan [28] and Zou [29], respectively. These penalties can achieve three desirable properties (Fan and Li [30]): producing sparse solutions, to ensure consistency of model selection, and to result in unbiased estimates for large coefficients.

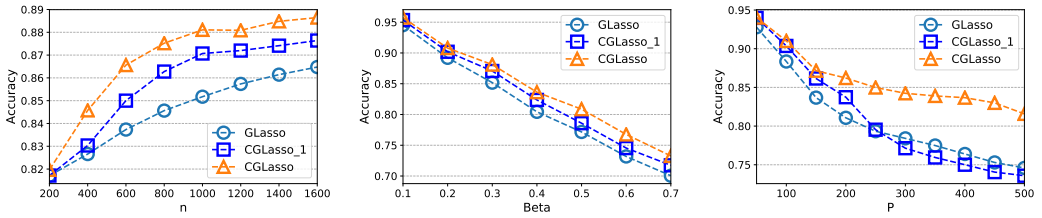
In addition, in the consideration of the connectivity among all variables, we can transfer $\hat{\mathbf{\Theta}}^{\star}$ into $\hat{\mathbf{\Theta}}$ by the following function:

$$\hat{\mathbf{\Theta}} = \hat{\mathbf{H}}\hat{\mathbf{\Theta}}^{\star}\hat{\mathbf{H}}^{T} \tag{2.19}$$

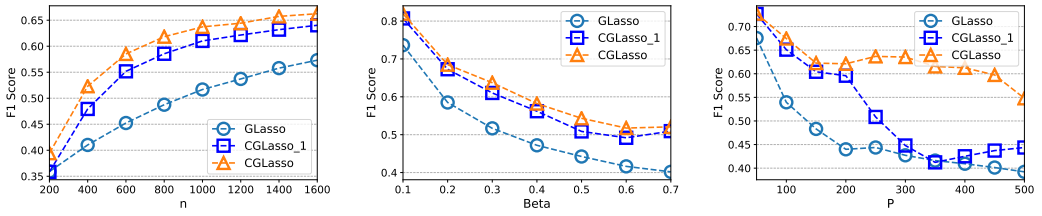
In the aspect of scaling matrix, we set a threshold on the entries of $\hat{\mathbf{\Theta}}^{\star}$ as:

$$\mathbf{E}_{ij} = \begin{cases} 1, \text{ if } |(\hat{\mathbf{\Theta}}^{\star})_{ij}| > \delta. \\ 0, \text{ otherwise} \end{cases} \tag{2.20}$$

to obtain the graph edge among all variables. Here δ is a very small amount, which is pre-specific parameter. We also use the same scheme on $\hat{\mathbf{\Theta}}$ to obtain the graph edge among each group, which is more intuitive for analyzing the network study in real case.



(a) *Accuracy of Scenario1* **(b)** *Accuracy of Scenario2* **(c)** *Accuracy of Scenario3*



(d) *F1-score of Scenario1* **(e)** *F1-score of Scenario2* **(f)** *F1-score of Scenario3*

Figure 2.3: *Comparison of CGLasso, CGLasso-1 and GLasso on edge detection. In the figure, n denotes sample size, beta denotes density of inter-connection among different groups, p denotes the number of variables. The first row shows the results of accuracy, and the second shows the results of F1-score. The blue line indicates the metric resulting from GLasso. The orange trace indicates the metric resulting from CGLasso. The green trace indicates the metric resulting from CGLasso-1. The subfigures (a) and (d) consider different sample size n from 200 to 1600, meanwhile fix the others, including $\beta = 0.3$ and $p = 110$; The subfigures (b) and (e) consider different density of inter-connection beta from 0.1 to 0.7, meanwhile hold on $n = 1000$ and $p = 110$; The subfigures (c) and (f) consider different variables number p from 50 to 500 and hold on $n = 1000$ and $\beta = 0.3$;*

2.1.5 Experimental Evaluations

Synthetic Data with Ground-Truth

We evaluate the performance of our model on synthetic data, where ground-truth is given. Firstly, we generate a block-diagonal matrix $\mathbf{\Theta}$, which has p features and L diagonal blocks (groups on the real-case). Meanwhile we give random sparity structures for each block $\mathbf{\Theta}_{G_i,G_i}$. The size of these blocks is $p/L \times p/L$. So each block has the same scale. Secondly, to generate the inter-connectivity between each diagobal block, we add off-diagonal blocks to $\mathbf{\Theta}$. Generally, we select $\beta L(L-1)/2$

pairs of groups randomly, where β means the density of inter-connectivity between off-diagonal blocks. In order to simulate the connectivity of variables among diagonal and off-diagonal blocks, we control the connectivity of each variables on diagonal block with a high density, which denotes as α_{inner}, then giving a density α_{inter} with a low value to each off-diagonal block. Following the above steps, we can obtain the precision matrix $\mathbf{\Theta}$. Then we select n samples randomly from the Gaussian distribution to obtain the empirical covariance matrix. In the process, when we generate the co-variance matrix of all variables, we sample 10 times for all experiments. So all the results shown as follow are the average of each experiment. It allows us to evaluate the precision and stability of our model.

We simulate three scenarios by controlling one parameter and holding on the others:

- **Scenario 1**: We fix $\beta = 0.3$ (the density of inter-connection among off-diagonal blocks) and variables number $p = 110$, and then control sample size n from 200 to 1600.
- **Scenorio 2**: We hold on $n = 1000$ and $p = 110$, and then control β from 0.1 to 0.7.
- **Scenorio 3**: We hold on $n = 1000$ and $\beta = 0.3$, and then p from 50 to 500.

Here we only focus on these three primary parameters and fix all other parameters in all scenarios, including $L = 10$ (groups), $\alpha_{inner} = 0.9$ (density of the connectivity in each variables on diagonal block) and $\alpha_{inter} = 0.3$ (density of the connectivity in each variables on off-diagonal block).

Because the nature of collective discovery in our model, we look at clustering and edge detection separately. On clustering, we compare CGLaso with three methods: k-means, ONMtF and CGLasso-1. Among these three models, k-means and ONMtF are widely used baselines in the literature [31]. 1 Specifically, we only compare ONMtF here, not ONMtF-SCR of [5], for the reason that: our experiment is a common network without the constraint of spatial continuity and we pay more attention on the terms of accurateness and robustness of models, so we don't need to consider the spatial continuity regularization in ONMtF. However, in the real word ADHD-200 data, we apply ONMtF-SCR in the comparison for supplement, because a nicely interpretation is more important in this case.

On edge detection, we compare our model with GLasso and CGLasso-1. It makes sense to involve CGLasso-1 in the comparison, for the reason that CGLasso-1 can be treated as a pipeline method. It is necessary for us to check if iteration alternates between $\hat{\mathbf{\Theta}}^{\star}$ and $\hat{\mathbf{H}}$ is helpful for the performance of our model. To evaluate the quality of edge detection. We follow [32] to define the accuracy and F1-score of edge detection as

$$Accuracy = \frac{n_d}{n_g}, \tag{2.21}$$

$$F1 = \frac{2n_d^2}{n_a n_d + n_g n_d} \tag{2.22}$$

where n_d is the number of true edges detected by the algorithm, n_g is the number true edges and n_a is the total number of edges detected. Higher accuracy score or higher F1 score indicates better quality of edge detection.

Figure 2.3 shows the comparison between CGLasso, GLasso and CGLasso-1

in terms of edge detection. The first row shows the result of accuracy score, and the second line shows F1-score. We denote all the methods with different marks and colors, orange triangle for CGLasso, dark blue square for CGLasso-1, light blue circle for GLasso, respectively. Meanwhile, each column denotes different scenario. According to the results of first column, we can observe that as sample size rises, all the models can be improved on different levels (CGLassp $>$ CGLasso-1 $>$ GLasso). In the second column, as the density of inter-connectivity among off-diagonal blocks rises, the performance of all methods become worse on different levels. It makes sense, for the reason that the network with a high β is more complex than that with a low β. Meanwhile, in this case, our proposed model still performs better than the others. In the third column, we can note that as the number of variables rises, which is common in the real data, the performance will fall down and fluctuate on different levels. Among these three method, the decline and fluctuation of GLasso is most obvious, and our proposed model performs stable relatively. Consequently, compared to the GLasso, the CGLasso-1 model already generates a more accurate results on synthetic data, meanwhile, our proposed model CGLasso can continue to improve the performance of CGLasso-1. In short, CGLasso outperforms the others in terms of detecting true edges in the global precision matrix.

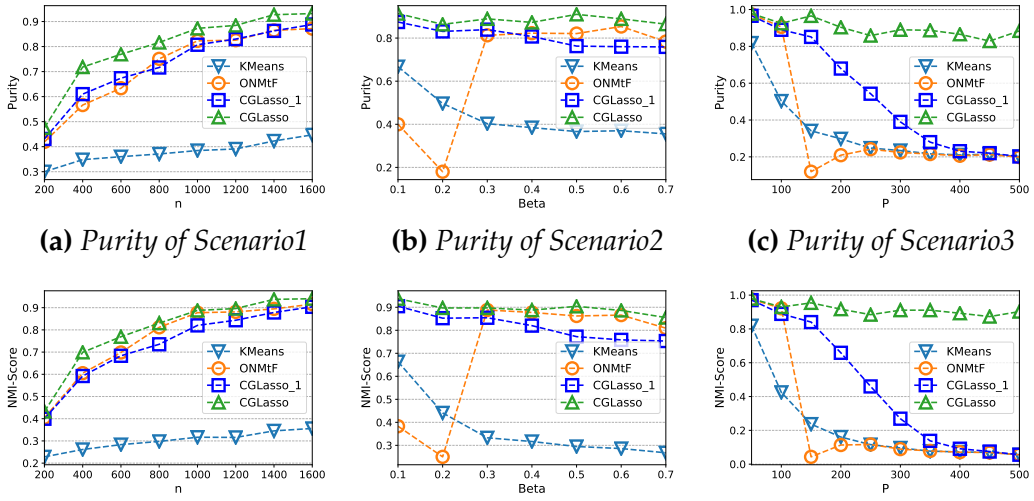
Figure 2.4 shows the comparison of CGLasso, CGLassp-1,k-means and ONMtF in terms of purity and NMI-score, in the consideration of group clustering. Similar to the comparison of edge detection, we denote orange triangle for CGLasso, dark blue square for CGLasso-1, light blue inverted triangle for K-Means and green circle for ONMtF, respectively. According to the first and second columns, we

Table 2.1: *Variance of edge detection in each scenario*

		Accuracy $\times 10^2$			F1-score $\times 10^2$		
Scenario	Dataset	GLasso	CGLasso	CGLasso-1	GLasso	CGLasso	CGLasso-1
n	200	0.05	0.25	0.20	0.88	2.46	2.39
	600	0.20	0.74	1.12	1.03	2.45	3.01
	1000	0.14	0.70	0.76	0.57	1.48	1.91
	1400	0.18	0.67	0.92	0.94	1.32	2.37
β	0.1	0.17	0.44	0.36	0.87	1.37	1.18
	0.3	0.14	0.70	0.76	0.57	1.48	1.91
	0.6	0.19	0.61	0.80	0.68	1.16	1.19
p	50	0.36	0.70	0.71	2.40	2.90	0.30
	150	0.14	0.50	1.30	0.67	0.87	0.21
	300	0.07	0.41	0.55	0.37	1.36	0.24
	500	0.03	0.48	0.00	0.14	1.67	0.00

note that CGLasso, CGLasso-1 and ONMtF can provide a significant better performance than k-means, no matter in the consideration of purity or NMI-score, and CGLasso is slightly better than CGLasso-1 and ONMtF. This advantage is extremely extended in scenario 3, which controls the number of variables and fixes sample size and the density of inter-connection among off-diagonal blocks. We can observe from the third column that as p rises, all models drop rapidly, except CGLasso. So when a network is very simple with a low $beta$ and a small size of variable space, all the methods can work as long as we have enough dataset. However, when the variable space becomes large and the $beta$ increases, CGLasso can show its advantages compared to k-means and ONMtF. In fact, a complex network with a large variable space is relatively common in real-world dataset such as brain networks, especially when we transfer an image to several pixel points.

Combining the above two comparison, we can conclude that, in synthetic data with ground-truth, our proposed model CGLasso can improve the performance of



(a) *Purity of Scenario1* **(b)** *Purity of Scenario2* **(c)** *Purity of Scenario3*

(d) *NMI Score of Scenario1* **(e)** *NMI Score of Scenario2* **(f)** *NMI Score of Scenario3*

Figure 2.4: *Comparison of CGLasso, CGLasso-1, k-means and ONMtF on clustering. n denotes sample size, $beta$ denotes density of inter-connection among different groups, p denotes the number of variables. The first row shows the purity of each model in three scenarios, the second row shows the NMI score of each model in three scenarios. In each subfigure, the blue line indicates the metric resulting from k-means, the orange line indicates the metric resulting from CGLasso, the green line indicates the metric resulting from CGLasso-1 and the red one indicates the metric resulting from ONMtF. The subfigures (a) and (d) consider different sample size n from 200 to 1600 and hold on $\beta = 0.3$ and $p = 110$; The subfigures (b) and (e) consider different density of inter-connection $beta$ from 0.1 to 0.7 and hold on $n = 1000$ and $p = 110$; The subfigures (c) and (f) consider different variables number p from 50 to 500 and hold on $n = 1000$ and $\beta = 0.3$;*

edge detection and clustering at the same time. Specifically, our proposed model improve the power of Glasso on the edge detection, meanwhile, having more robustness results of nodes discovery than that of ONMtF. At addition, to showing the stability of our methods, we enclose two tables of variance among each model. In the next part of this chapter, we demonstrate the interpretation of cognitive network inferred by our model by using ADHD-200 dataset.

Table 2.2: *Variance of group clustering in each scenario*

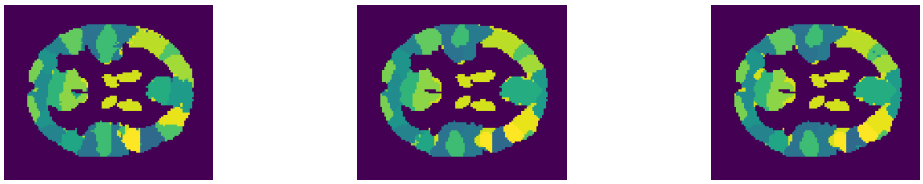
		Purity $\times 10^2$				NMI-Score $\times 10^2$			
Scenario	Dataset	KMeans	CGLasso	CGLasso-1	ONMtF	KMeans	CGLasso	CGLasso-1	ONMtF
n	200	2.23	4.34	2.83	2.68	2.67	3.72	2.18	3.49
	600	5.05	7.21	5.86	7.01	3.70	6.93	4.48	6.75
	1000	3.30	6.26	5.39	8.05	3.58	4.09	5.39	6.09
	1400	4.19	5.17	6.57	5.46	3.94	3.23	5.15	4.42
β	0.1	2.58	5.56	3.24	0.00	1.76	3.78	2.78	1.93
	0.3	2.57	4.22	5.55	12.00	2.08	3.04	5.77	7.27
	0.6	1.99	5.20	5.23	6.13	2.04	4.66	5.82	4.56
p	50	4.24	3.52	4.65	4.01	3.27	3.00	3.59	3.17
	150	2.54	3.45	5.63	3.20	2.92	3.48	6.07	6.71
	300	1.40	5.56	4.09	2.37	0.97	3.77	3.54	2.01
	500	1.08	8.53	1.07	1.32	0.86	7.34	0.86	0.88

ADHD-200 Dataset

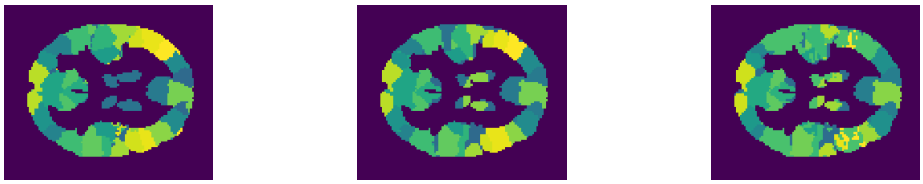
In this section, we evaluate our proposed method by using fMRI dataset from ADHD-200 project[1]. ADHD (Attention Deficit Hyperactivity Disorder) is a chronic condition, and has been happened on 5% - 10% of school-age children. Up to now, the annal costs on ADHD have exceeded 36 billion in the United States. Our real world dataset is distributed by nilearn[2]. Specifically, there are 40 subjects in total. Among them, 20 of which are labeled as ADHD, and the others are labeled as TDC. The rsfMRI scan of each subject in the dataset is a series of snapshots of 3D brain images of size $61 \times 76 \times 61$ over $\sim$176 time steps.

By using this real world datasets, our experiments aims at discovering the cognitive networks of human brain of ADHD and TDC groups, which can provide some guidelines in terms of identification and diagnosis from a view of computer science. In the consideration of extracting the voxels that can be considered as parts of the brain, we use the AAL brain-shaped mask which is provided by neurology

[1]http://fcon_1000.projects.nitrc.org/indi/adhd200
[2]http://nilearn.github.io/



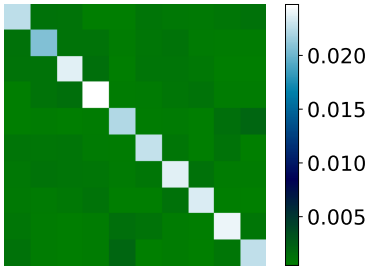
(a) *CGLasso* **(b)** *ONMtF with continuity* **(c)** *ONMtF*



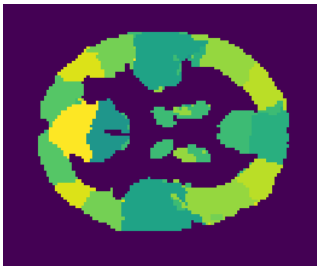
(d) *CGLasso* **(e)** *ONMtF with continuity* **(f)** *ONMtF*

Figure 2.5: *Comparison among CGLasso, ONMtF and ONMtF with spatial continuity on ADHD-200 Brain Dataset. Among them, (a), (b) and (c) are worked on ADHD subjects; (d), (e) and (f) are worked on TDC subjects. All the models set the number of groups to 20.*

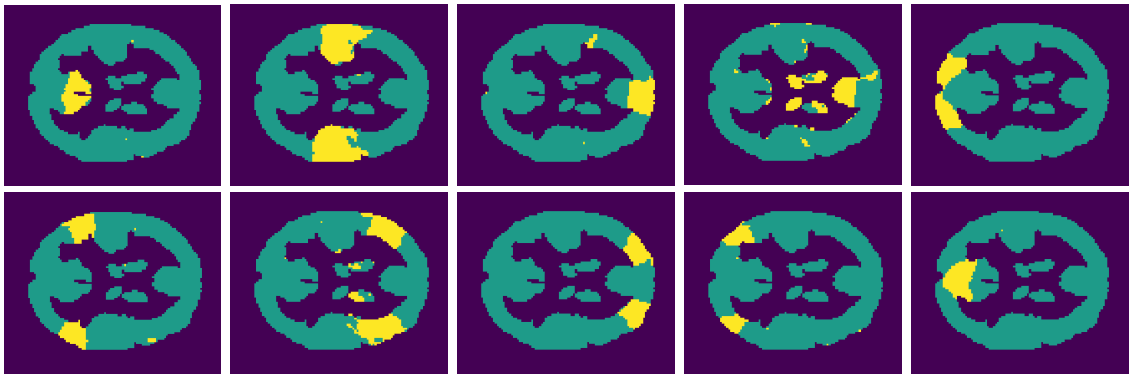
professionals. We follow [33], using a middle slice of these scan for the ease of presentation, consequently, each of the brain scans can be represented by about 3281 voxels. In this real case, in the consideration of nicely interpretation of the result, we compare our proposed model to ONMtF with spatial continuity penalty especially, which we denote as ONMtF-SCR. In Figure 2.5, we present the comparison of clustering results between CGLasso, ONMtF and ONMtF-SCR. Here we assume the number of groups $k = 20$. We can notice that in the case of TDC subjects, the groups inferred by ONMtF is very scattered, which means a bad interpretation of the result. However, when we adjust ONMtF by the method ONMtF-SCR, the result provides a better spatial continuity than original method. According to the first colume of Figure 2.5, our proposed model CGLasso can also provide a result with a good spatial continuity. Meanwhile, the groups inferred by CGLasso shows a better nature of symmetry than ONMtF and ONMtF-SCR, which makes sense for



(a) *Discovered edges*



(b) *Discovered groups*



(c) *Discovered Groups (1-10)*

Figure 2.6: *Discovered network of TDC subjects ($k = 10$). (a) is based on the precision matrix of each group. The degrees of color in off-diagonal blocks indicates edges or connectivity between each group, where dark green means connection and light green means independence. (b) shows the inferred groups by CGLasso, which are shown below individually.*

the symmetrical structure of the right and left brain. Consequently, the proposed CGLasso presents a much interpretable results compared to the ones of ONMtF and ONMtF-SCR. So here we demonstrate that our proposed method can produce the same good result as the method in [5], which aims at spatial continuity in the brain network. Combining the results of synthetic experiments without spatial continuity constraint, we believe that our proposed model can provide more effectiveness and robustness results than ONMtF-SCR in brain network discovery.

Figure 2.17 shows the result from CGLasso ($k = 10$) on real TDC subjects, which can demonstrate the capability of our method both on the terms of edge detection and clustering. In the first row of it, the left one shows the estimate precision matrix of $\Theta^{\star}$, which indicates the result of edge detection, and the right one shows the result of group clustering. In the terms of precision matrix, the degree of color in the off-diagonal elements indicates whether there is an edge between the corresponding groups. In the second row, the nature of spatial continuity and structure symmetry is shown in a more intuitive way.

In summary, based on the comparative results of synthetic and real-world cases, the results show that the proposed method performs better than the compared baselines in terms of four quantitative metrics. Meanwhile, our method produces more meaningful networks comparing with other baseline methods.

2.2 Task 2: Mixture Model in Brain Edge Detection

2.2.1 Motivation

In this work, we explore a noval setting on edge detection problem of brain network. We already introduced the problem setting of edge detection in the last task. Since a well-constructed connectivity network servers as the prerequisite for many graph mining algorithms on brain disorder diagnosis and brain functionality analysis [34], it is significant to design a more effective and accurate edge detection method. Existing edge detection methods usually rely on the assumption that all nodes' activities obey an unified multivariate Gaussian distribution, and GLasso imposes sparseness on the estimation of inverse covariance matrix (*a.k.a.* precision

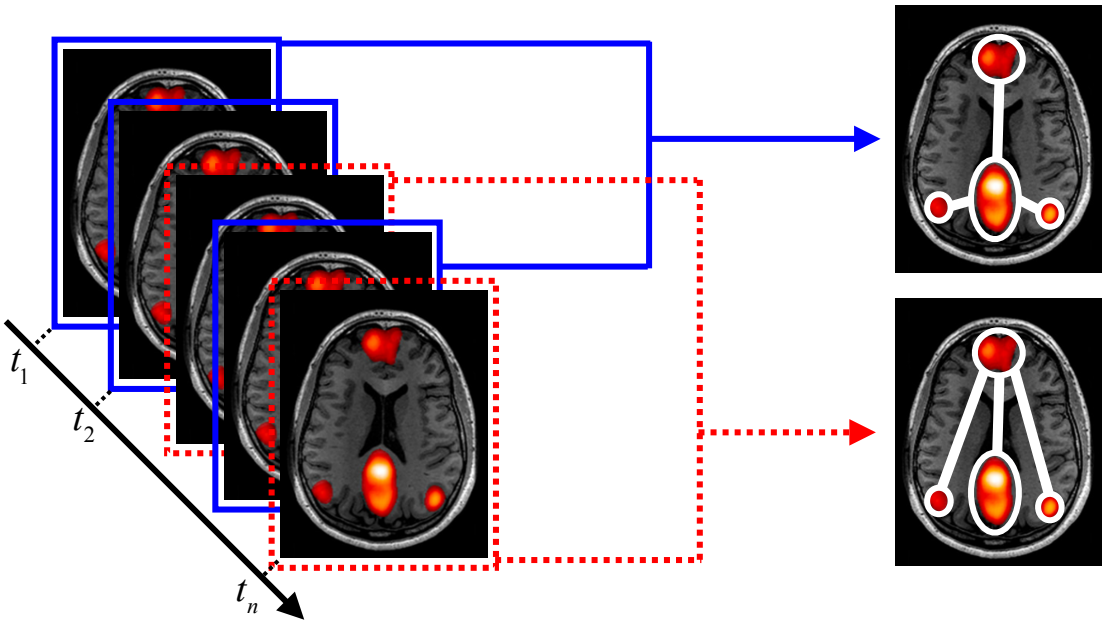


Figure 2.7: *The problem of Gaussian mixture sparse inverse covariance estimation. The brain activities over time may originate from the mixture of multiple latent cognitive brain modes* (i.e. *different connectivity structures among nodes). Without knowing the mode proportions and assignments in the observed brain images, our goal is to discover these underlying sub-networks for different modes.*

matrix) to predict connectivity structure. However, in many neurology studies such as [35], human brains usually exhibit dramatically different activity modes when they perform different tasks. Based on these studies, we believe that the cognitive structure of the human mind can be paralleled into several sub-graphs based on different activity modes.

Applying GLasso without considering different latent cognitive modes is equivalent to deriving an "average" network representation. Since the behavior of different brain modes varies significantly, the derived "average" network may lose crucial information. Under such context, as illustrated in Figure 2.7, it is natural to investigate whether and how one could extend the edge detection methods

applied in brain network to capture the connectivity structures of multiple underlying cognitive brain modes.

To incorporate the concept of multiple connectivity structure into edge detection, we follow the idea of Probabilistic Latent Semantic Analysis (PLSA) [36] to adopt Gaussian mixture model on this problem. PLSA views a document as a mixture of various topics, and it assumes that the generation of a document follows some topic-word distributions which can be found by sampling. Similarly, we could view brain scans as mixtures of latent modes, where each mode is characterized by a Gaussian distribution with different covariance.

In this work, our goal is to reveal these structure of underlying sub-network from the observed activities simultaneously. To tackle this problem, we propose a new model, namely MGL, to discover such mixture connectivity structures of the brain network. Similar to GMM, MGL learns the proportions and assignments of each latent cognitive mode iteratively via an EM framework, with the emphases on inferring the inverse covariance matrix of each latent distribution. A novel regularization approach called Mutual Exclusivity Regularization (MER) is also proposed to differ each inverse covariance matrix, implying that sub-network of different brain regions are activated under different cognitive modes.

2.2.2 Related Works

In the edge detection of brain network, it has two major branches: effective connectivity estimation and functional connectivity estimation. For the first branch, scholars pay more attention on obtaining a directed network from fMRI data through structure learning method for Bayesian networks [12]. In contrast, the second

branch focuses on some approaches such as hierarchical clustering, pairwise correlations and independent component analysis, which can be found in [13] for more details. [6] proposed sparse gaussian graphic models , which are a very useful for discovering directed links of brain network based on large-scale dataset by using sparse inverse covariance estimation. However, in the task of edge detection, these methods focus on unimodal distributions, where it is usually assumed that the observed samples are drawn from a single Gaussian distribution, which is opposed to some recent studies [37]. The Joint Graphical Model with fused lasso, which is proposed in [38], is in the framework of multivariate Gaussian mixture modeling. However, this method has shown to be sensitive to the noise and small size of the data sample.

2.2.3 Mixture Graphical Lasso

Gaussian Mixture Graphical Lasso

Given the number of base distributions K and the number of node N, we assume the observed sample of each node is a mixture of the K distributions. Thus, the joint probability of all observations $\mathbf{X} = (\boldsymbol{x}_1^\top, \cdots, \boldsymbol{x}_N^\top) \in \mathbb{R}^{N \times D}$ is given by

$$p(\mathbf{X}|\boldsymbol{\Theta}_k, \boldsymbol{\mu}_k, \phi_k) = \prod_{i=1}^{N} \sum_{k=1}^{K} \phi_k \mathcal{N}(\boldsymbol{x}_i|\boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k)$$

We could assume $\boldsymbol{\mu}_k = \mathbf{0}$ without losing generality, so the negative log likelihood (NLL) in terms of $\{\boldsymbol{\Theta}_k\}$ is given by,

$$\text{NLL}(\boldsymbol{\theta}) = -\sum_{i=1}^{N} \log\Big(\sum_{k=1}^{K} \phi_k \mathcal{N}(\boldsymbol{x}_i|\mathbf{0}, \boldsymbol{\Theta}_k^{-1})\Big) \tag{2.23}$$

where $\boldsymbol{\theta} = \{\phi_1, \cdots, \phi_k, \boldsymbol{\Theta}_1, \cdots \boldsymbol{\Theta}_k\}$ is the model parameters.

The Mutual Exclusivity Regularization

Similar to the Adaptive Lasso in [29], we also need to impose regularization on our mixture model to obtain interpretable results, which means non overlapping edges exist among all estimators of precision matrices. However, be different with adaptive lasso or fused lasso, the intuitions are two folds: (1) we want each $\boldsymbol{\Theta}_k$ to be sparse; (2) we want each $\boldsymbol{\Theta}_k$ to be fairly different from other $\boldsymbol{\Theta}_{k'}$. Towards this end, we propose to the mutual exclusivity regularization as follows,

$$\ell_{\lambda_1,\lambda_2}(\{\boldsymbol{\Theta}_k\}) = \lambda_1 \sum_{k=1}^{K} \|\boldsymbol{\Theta}_k\|_1 + \lambda_2 \sum_{i \neq j} \text{tr}(\bar{\boldsymbol{\Theta}}_i \bar{\boldsymbol{\Theta}}_j) \tag{2.24}$$

where $\bar{\boldsymbol{\Theta}}$ is the non-negative copy of $\boldsymbol{\Theta}$ removed all diagonal elements. The first term is identical to graphical lasso, which imposes sparsity controlled by $\lambda_1 > 0$ on each $\boldsymbol{\Theta}_k$. The second term is the summation of the approximate divergence measure between each pair $(\boldsymbol{\Theta}_i, \boldsymbol{\Theta}_j)$. It is easy to see when there is no overlapping non-zero entities between each $\boldsymbol{\Theta}_k$, this term reaches its minimal value 0. $\lambda_2 > 0$ is employed to tune the strength of the second regularization. So it makes sense that we can use this term to force each estimation of $\boldsymbol{\Theta}_k$ in the result to have as few

over-lapping elements as possible.

Hence, we formally present the objective of our MGL as follows,

$$\min_{\{\boldsymbol{\Theta}_k \succ 0\}} \text{NLL}(\{\boldsymbol{\Theta}_k\}) + \ell_{\lambda_1,\lambda_2}(\{\boldsymbol{\Theta}_k\}) \tag{2.25}$$

The Latent States

Since there are K separate latent distributions, so each data sample $\boldsymbol{x}_i$ could come from one of the K distributions, we denote the corresponding state as $z_i \in \{1, \cdots, K\}$. Thus, the NLL function could be rewritten as follows,

$$\begin{aligned} \text{NLL}(\boldsymbol{\theta}) &= -\sum_{i=1}^{N} \log \sum_{k=1}^{K} \Big(\frac{\mathbf{Q}(z_{ik}) p\big(\boldsymbol{x}_i | \boldsymbol{\Theta}_k \, \phi_k\big)}{\mathbf{Q}(z_{ik})} \Big) \\ &= -\sum_{i=1}^{N} \log \sum_{k=1}^{K} \Big(\frac{p\big(\boldsymbol{x}_i, z_{ik} | \boldsymbol{\Theta}_k \, \phi_k\big)}{\mathbf{Q}(z_{ik})} \Big) \end{aligned} \tag{2.26}$$

Here $\mathbf{Q}(z_{ik})$ is the latent variable and $\sum_{k=1}^{K} \mathbf{Q}(z_{ik}) = 1$.

According to the expression in the Equation (2.29), it can not be directly computed because the expression in log is a sum term. So we can use the Expectation Maximization (EM) algorithm to optimize the above NLL *w.r.t.* $\{\boldsymbol{\Theta}_k\}$. We summarized the MGL algorithm in Algorithm (3).

Initialization

As we know from the Algorithm (3), we need to give starting values of each estimators. In the process of comparative experiments, we found that the initialization of the parameters will largely affects the performance of our model. The following scheme we found empirically works well in our experiments. For each observation

Algorithm 2 Algorithm for `MGL`

Require: i: $\mathbf{X}$: The observations of D-variate Gaussian distribution
ii: k: the number of Gaussian distributions
iii: λ_1: the Lagrangian multiplier of sparsity constraint
iv: λ_2: The Lagrangian multiplier of mutual exclusivity constraint
v: iter$_{max}$: the maximum number of iteration
Output: $\hat{\mathbf{\Theta}}_\mathbf{k}$, $\hat{\mathbf{\Phi}}_\mathbf{k}$

1: Initialization: initialize $\mathbf{\Phi}_k^{(0)}$, $\mathbf{\Theta}_k^{(0)}$ and $r_{ik}^{(0)}$
2: **repeat**
3: E step: Update the latent variable $r_{ik}^{(t)}$ with given $\mathbf{\Phi}_k^{(t-1)}$ and $\mathbf{\Theta}_k^{(t-1)}$
4: M step: Update $\mathbf{\Phi}_k^{(t)}$, $\mathbf{\Theta}_k^{(t)}$ with $r_{ik}^{(t-1)}$
5: **until** $iter = iter_{max}$ or convergence

$i = 1, \ldots, N$, we distribute it randomly a class $k \in \{1, \ldots, K\}$. Then we assign a weight $\hat{r}_{ik} = 0.9$ for this observation i and distribution k and $\hat{r}_{ij} = \frac{0.1}{K-1}$ for all other distributions. In the M-step, we update $\mathbf{\Theta}_k$ from the initial values $\hat{\mathbf{\Theta}}_k^{(0)}$ computed by GLasso based on the whole samples. and ϕ_k from the initial values $\hat{\phi}_k = \frac{1}{K}$.

2.2.4 Experimental Evaluations

In this part, we demonstrate the performance of MGL through extensive comparative experiments. We evaluate MGL in synthetic datasets at first. To comprehensively evaluate proposed model, we conduct experiment to answer the following research questions:

- **RQ 1**: How does MGL perform compared with state-of-the-art models in the consideration of the effect of sample size?
- **RQ 2**: Does our model still show robustness under noise? If the MER regularization term has positive influence on the performance under noise?

- **RQ 3**: How do hyper-parameters in comparative experiments impact each model performance?
- **RQ 4**: Is there a problem with mixture brain network structure in real ADHD-200 datasets?

Compared Baselines

To demonstrate the effectiveness of our proposed method, we test against several variations of the state-of-art method Graphical Lasso:

- **GLasso + Spectral Clustering**: GLasso algorithm that assumes all data samples are drawn from the same Gaussian distribution, then using Spectral Clustering divide the whole network into several sub-graph.
- ***k*-means + GLasso**: This is a pipeline method that first employs k-means to assign each $\boldsymbol{x}_i$ to different groups, then using GLasso for each group to obtain the final $\boldsymbol{\Theta}_k$.
- **JGL [38]**: This is the Joint Graphical Model with fused lasso, which is proposed in [38]. It is equivalent to our proposed model without MER term. So it can work as the comparative method for assessing the performance of MER.

Data Set

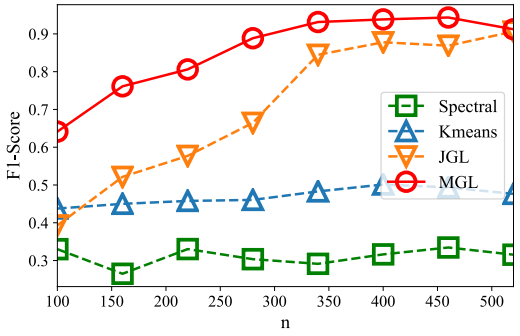
Due to the lack of ground truth in many real-world data, we first compare our proposed method against other competitors on several carefully designed synthetic data sets.

At first, we design some synthetic data sets purposefully. Firstly, we generate k diagonal matrices (k is the number of distribution, which is given in advance), then divide it into several equal-scale blocks. It makes sense for two reasons: we need to control each sub-graphs $\mathbf{\Theta}_k$ with non-overlapping edges on off-diagonal areas; by making edges of each sub-graph more concentrated, it is helpful for making results conductive to visualization. Secondly, we choose different off-diagonal blocks on each $\mathbf{\Theta}_k$, giving connectivities for these chosen blocks with a high density. Following the above steps, we generate each $\mathbf{\Theta}_k$ without overlapping edges on off-diagonal areas. Based on $\mathbf{\Theta}_k$, we compute each $\mathbf{\Sigma}_k$, then select N_k samples ($\sum_{k=1}^{K} N_k = N$) randomly from each Gaussian distribution. In the next section, in order to evaluate the stability of our model, we also add noise into the samples. To exclusive the system randomness, we sample 10 times for all experiments, calculate the average of each experiments. So we can evaluate the precision and stability of our model at the same time.

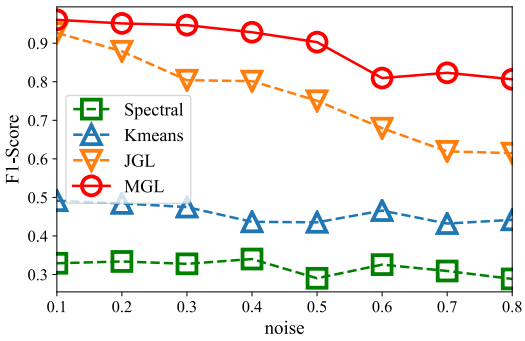
Experimental Settings

We simulate four scenarios by controlling one parameter and holding on the others. In these situations, we select sample size N and the standard error of noise œ as the controlled parameters.

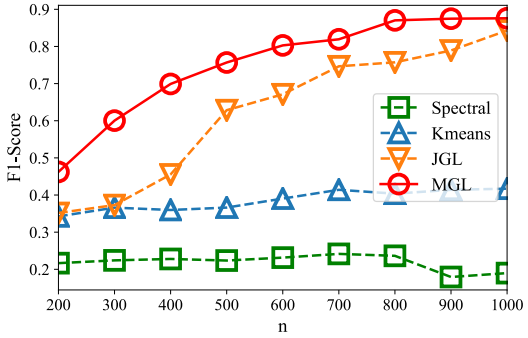
- **Scenario 1**: We fix $p = 8$ (the number of variables), $k = 2$ (the number of Gaussian distributions), and œ $= 0$ (the standard error of noise), and then control sample size N from 100 to 520.
- **Scenario 2**: We fix $p = 8$, $k = 2$, and $N = 500$, and then control noise œ from 0.1 to 0.8.



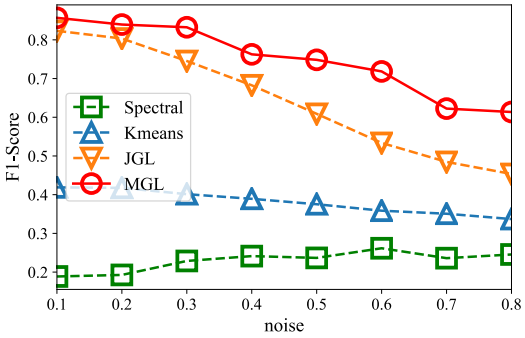
(a) *Low Dimension - Sample Size (Scenario 1)*



(b) *Low Dimension - Noise (Scenario 2)*



(c) *High Dimension - Sample Size (Scenario 3)*



(d) *High Dimension - Noise (Scenario 4)*

Figure 2.8: *Comparison of each model on edge detection. Each figure shows the results of F1-score. The dark blue line indicates GLasso + Spectral; the light blue indicates k-means + GLasso; the orange one shows the result of MGL without Mutual Exclusivity Regularization and the green one shows the result of MGL.*

- **Scenario 3**: We fix $p = 20$, $k = 2$, and œ $= 0$, and then control sample size N from 200 to 1000.

- **Scenario 3**: We fix $p = 20$, $k = 2$, and $N = 1000$, and then control noise œ from 0.1 to 0.8.

Evaluation

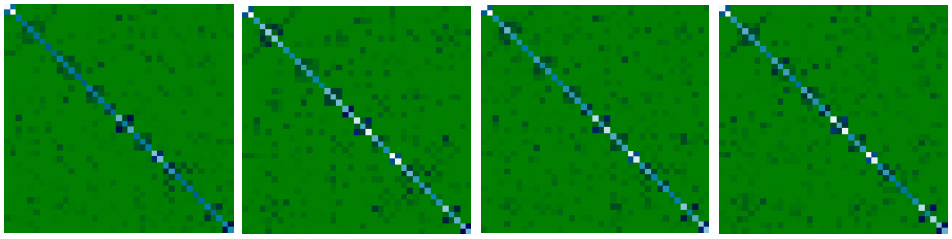
Figure 2.8 shows the comparison between MGL and other baseline models. The results in the figure answer the first three **RQ** mentioned before. The first column shows the results when we control sample size N and hold on the others, which corresponds to **RQ1**. It is obvious that k-means and Spectral models are useless when the ground truth data sets are drawn from mixture Gaussian distribution. Meanwhile, when the sample size is not large enough, the precision of JGL is lower than that with MGL. The second column shows the results when we control noise, which corresponds to **RQ2**. We fix the sample size N on 500, so when œ $= 0$, JGL is as good as MGL. According to the results, The louder the noise, the worse JGL performs, which means sensitive to the noise.So the result demonstrates that MER regularization can improve the performance of our proposed model. Compared to the others, MGL shows robustness in this scenario. To answer **RQ3**, we can figure out the answer from both column in this figure. Since our experiments are setting in low-dimensional and high-dimensional space separately, we can see from all comparison results that the issue of hyper-parameters does not affect the performance of MGL. In contrast, the performance of JGL in high-dimensional space isn't as well as that in the low-dimensional space, no matter in the scenario of sample size or noise. In summary, in the comparative experiment of synthetic datasets with ground-truth, our proposed method MGL shows better accuracy and robustness than that of other comparison methods.

Real fMRI Data

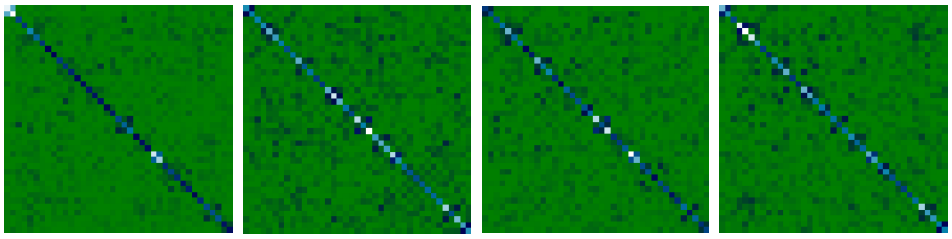
We also evaluate our proposed method on fMRI dataset from ADHD-200 project. In our experiment, we only choose the subjects which are labeled as ADHD. We focus on the multiple connectivity structures among the same subjects, in order to provide evidence on feature selection between different subjects in further study. Rather than discover the brain network on the level of voxels, we extracts the signal on regions defined via a probabilistic atlas, to construct the data sets. So it is more conventional for visualization of the results. The data sets is a 1899×39 data sets and we consider that they are drawn from a mixture Gaussian distribution. However, the number k of it is unknown, which need to be given in advance. Through repeated experimental observations, we found that $k = 4$ can provide the most reasonable results on the data sets.

Because real fMRI data lacks ground-truth as a reference to measure the accuracy and robustness of the model. We are more concerned with the interpretability and rationality of the results. Specific to our proposed model, we are more concerned about whether our model can mine different connectivity structures among nodes from the fMRI datasets.

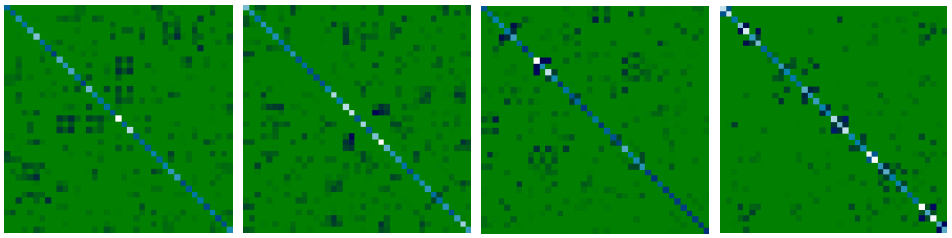
According to the Figure 2.9, we can find that there are almost no differences among four sub-graphs discovered by k-means plus GLasso. It indicates that this method is useless for mining sub-graphs in ADHD data sets. JGL shows four different sub-graphs, however, so many overlapped areas among them. These results seem not to be sparse matrices, which indicates that the corresponding connectivity structure is not very clear through this method. Compared to it, sub-graphs discovered by MGL is clearer and the number of overlapped areas is less. There-



(a) *Sub-graphs discovered by k-means*



(b) *Sub-graphs discovered by JGL*



(c) *Sub-graphs discovered by MGL*

Figure 2.9: *Comparison of k-means + GLasso, JGL and MGL on ADHD dataset. The results show how to estimate a mixture connectivity structure on a group of subjects using different group sparse inverse covariance estimation models from real fMRI data set. The closer the color of elements in off-diagonal is to blue, the bigger probability the directed edges between corresponding nodes.*

fore, although lacking the ground truth in ADHD data, we can still believe that the inferred results of MGL is consistent with the defined problem in this chapter, especially in the consideration of mixture Gaussian distribution with non-overlapping areas among their precision matrices. The Figure 2.10 shows the corresponding connectivity structure of the results discovered by MGL. Here we only choose the axial direction of the cuts to show. The closer the color is to red, the stronger the

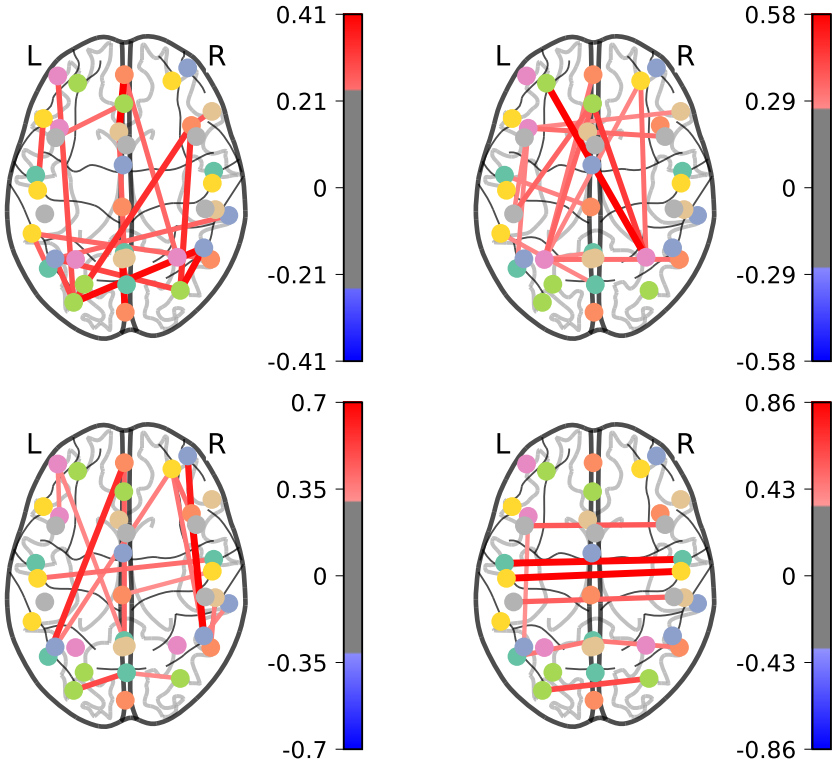


Figure 2.10: *We turn the results of Fig. 2.9 into connectome for visualization. Each precision matrix is displayed on glass brain on extracted coordinates. These graphs of precision matrices discovered by MGL in ADHD dataset. The closer the color of edge is to red, the stronger the directed relationship between corresponding nodes.*

directed relationship between the corresponding nodes. We highlight the stronger edges by adjusting the threshold of colorbar. According to the visualization of results, we can see that different sub-graphs highlight different relationships among all nodes. Different sub-graphs emphasize the relationships of different nodes, which means that subjects present different network structures on the time-line. This phenomenon is more obvious between the nodes related to DMN (default mode network), which includes the Parietal, Occipital Lobes, the Cingulum Region Posterior and the Frontal Cortex. Although the hypothesis about non-overlapped

areas among each connectivity structure may not exist in real ADHD subjects, we believe that MGL with MER regularization can more prominently show the difference between each connectivity structures discovered, so that we can have a better understanding of the association between cognitive network and human activities.

According to the analysis above, despite the lack of ground-truth, we believe that the existing results are still consistent with the problem defined in this chapter. So the result shows that there is a mixture connectivity structure among nodes in the fMRI datasets, and our proposed model MGL can effectively mine this mixture connectivity structure.

2.3 Task 3: Multi-State Brain Network Discovery

2.3.1 Motivation

. In this work, we explore task2 further by studying brain network at the pixel level of fMRI brain images. In particular, based on the recent studies [35, 37, 39], different activity states of the brain can not only have different connectivity structure, but also different brain parcellations. A successful brain network discovery method must therefore be adaptive to the dynamic changes of brain parcellations and connectivity structure. Such *state-based* adjustment of brain parcellation and connectivity over time is crucial for learning human brain networks. Ultimately, characterizing this mixture functional structure of brain parcellation and functional connectivity, as shown in Figure 2.11, leads to a better understanding of brain function and human behavior.

Formally, the brain network discovery problem, as shown in Figure 2.11, corre-

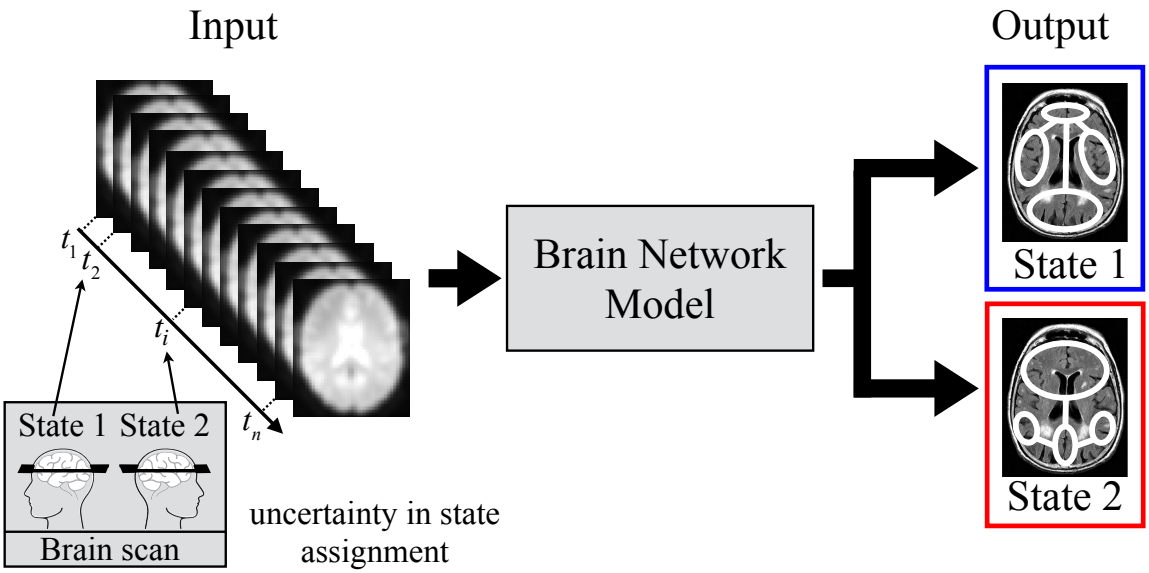
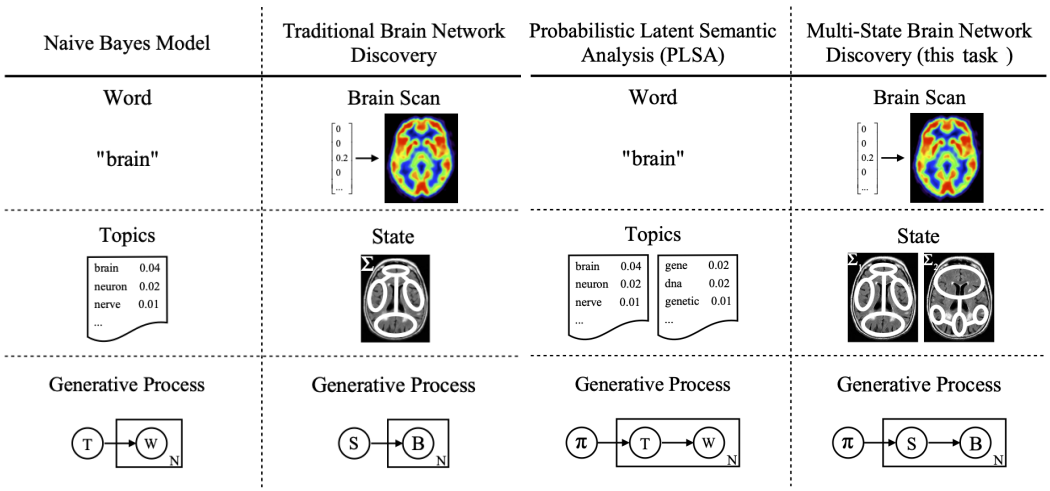


Figure 2.11: *Multi-State Brain Network Discovery.*

sponds to inferring a set of functionally homogeneous brain regions as the network nodes along with the connectivity between these nodes as the network edges from a series of brain scans over time. While there are some recent solutions [1, 5, 11], they often ignore the flexibility of functional network configurations when they are being learned. Instead, they assume instead that the brain is always in a single activity state, implying that signals extracted from different regions of the brain at different times are members of the *same* network, even though this is now known to be false [39]. As illustrated in Figure 2.11, it is natural to investigate whether and how one could extend the traditional methods applied in brain network discovery to capture *multiple* underlying brain network states, allowing for differing brain parcellation and connectivity. However, the ground truth brain activity states are rarely known.

In this work, We still refer the idea of Probabilistic Latent Semantic Analysis (PLSA) [36], which is originally proposed to adopt a mixture model for natural



(a) *Naive Bayes [40] and Brain Network Discovery Model [5, 11]*

(b) *PLSA[36] and Multi-State Brain Network Discovery Model*

Figure 2.12: *Two pairs of comparison: (a) naive bayes model and traditional brain network discovery model, (b) PLSA and our model in this chapter. In each generative process, the boxes are "plates" representing replicates. The outer plate represents document in naive bayes and PLSA, or observation subject in brain network study, while the inner plate represents the generative process of word (W) in a given document or brain scan (B) in a given subject, each of which word or brain scan is associated with a choice of topic (T) or state (S). π is the topic or state distribution. N denotes the number of words or scans.*

language. Specifically, PLSA assumes a given document is a mixture of topics, and that the document was generated according to a probabilistic model with latent topics. Get to the problem setting of multi-state brain network, it makes sense to follow the idea of PLSA and Gaussian Mixture Model to build a probabilistic model for representing the presence of sub-networks without requiring that an observed dataset should identify the sub-network to which an individual scan belongs.

In this work, we propose a variation of CGL, named MNGL , for multi-state brain network discovery, which jointly achieves brain parcellation and edge detection. First, we view brain scans as mixtures of latent states, where each state S is

characterized by a Gaussian distribution with its own covariance matrix Σ_{S}. Each Σ_{S} corresponds to a specific brain parcellation and connectivity between nodes. Therefore, in the generation of each brain scan, our model chooses a state S based on the mode distribution π (similar to how PLSA chooses a topic), and then generates a brain scan $B_i \sim \text{Multinomial}(\mathbf{0}, \Sigma_{\mathrm{S}})$ (as PLSA generates a word based on the topic chosen). MNGL follows the basic idea of PLSA. By contrast, traditional brain network discovery models [1, 5] are analogous to naive bayes model[40]. Figure 2.12 illustrates these two pairs of comparison. To model this multi-state network, we combine CGL with GMM in a unified objective function to deal with multi-state networks.

2.3.2 Related Works

Existing works can be divided into two categories. Firstly, for coherent brain network discovery, ONMtF [16] is a useful pattern recognition method. [5] extend ONMtF by adding a spatial continuity penalty, which can increase the interpretability of the parcellated regions. This method is a coherent model which can output the result of nodes discovery and edge detection simultaneously. However, it has discovered the edges based on the correlation matrix instead of inferring direct links between each node. Instead of using a correlation matrix, [6, 7] focus on sparse inverse covariance estimation for discovering connectivity of brain network based on large-scale datasets. These kinds of methods can distinguish direct links from indirect connections due to their solid probabilistic foundation. [1] propose a model called CGL to achieve the coherent brain network discovery, including edge detection and node discovery. However, this method ignores the problem of

multi-state problem we mentioned in this study.

For the multi-state problem, we consider the Gaussian Mixture Model (GMM) [41]. GMMs model the distribution of data observations as a weighted sum of parameterized Gaussian distributions. However, a prominent issue related to GMM is estimating the parameters given observations [42]. Through many extensions, the EM algorithm has proven to be a powerful algorithm for the maximum-likelihood estimation of GMMs [43]. Additionally, [44, 45] consider the issue of the number of mixture components in the model, which can lead to over-fitting in practice. GMMs have been widely used in many areas, especially for network discovery [46, 47, 48]. Most existing studies for mixture modeling focus on regularizing only the mean parameters with diagonal covariance matrices [49, 50], though some works [51, 52, 53] have started considering regularization of the covariance parameters. However, these works do not touch on the key issue of identifying the varying sparse structures of the precision matrices across the components of a mixture model in brain network discovery. [38] proposes a joint graphical model (JGL) to deal with cluster-specific networks. [54] aims to edge detection task by combing graphical lasso with GMM. However, these models need brain parcellation to be given first. Thus, existing models related to GMM are thus not suitable for the special problem defined in this study.

2.3.3 Mixture Coherent Graphical Lasso

Multi-State Network Graphical Lasso. We propose the first method for Multi-State Brain Network Discovery, which we refer to as the **M**ulti-State **N**etwork **G**raphical **L**asso, or MNGL. Firstly, following the idea of task 1, we map the origi-

nal variable space $\mathbf{X}$ into a new feature space $\mathbf{Y}$ similarly on the covariance matrix $\mathbf{\Sigma}$:

$$\begin{aligned} \mathbf{X} \leftarrow \mathbf{Y} &= \mathbf{H}^\top \mathbf{X}, \\ \mathbf{\Sigma} \leftarrow \mathbf{\Sigma}^\star &= \mathbf{H}^\top \mathbf{\Sigma} \mathbf{H}, \end{aligned} \tag{2.27}$$

where $\mathbf{Y}$ denotes the new k-dimensional feature space where each feature represents node, $\mathbf{\Sigma}^\star$ represents the inter-node covariance matrix, and $\mathbf{H}$ represents a cluster indicator matrix. $\mathbf{\Sigma}^\star$ thus measures the association between each node $\mathbf{y}_i$ [1].

For the rest of this chapter, we describe our proposed model in terms of $\mathbf{y}_i$ instead of $\mathbf{x}_i$. k is the index of nodes, j is the index of distributions, i is the index of samples. $\boldsymbol{\mu}_j^\star$ and $\boldsymbol{\Sigma}_j^\star$ represent the parameters of mean vector and covariance matrix corresponding to the j-th mixture gaussian distribution of $\mathbf{Y}$, respectively. Then, $\boldsymbol{\Theta}_j^\star$ represents the inverse matrix of covariance matrix $\boldsymbol{\Sigma}_j^\star$.

According to the notation above, given the number of base distributions m and the number of node k, we assume the observed sample of target feature space can be mapped into a new feature space (nodes), which also follows a mixture of the k gaussian distributions. The sample size is given as n. Thus, the joint probability of these nodes $\mathbf{Y} = (\boldsymbol{y}_1^\top, \cdots, \boldsymbol{y}_n^\top) \in \mathbb{R}^{n \times k}$ is given by

$$p(\mathbf{Y} | \{\mathbf{\Theta}_j^\star\}, \{\boldsymbol{\mu}_j^\star\}, \{\phi_j\}) = \prod_{i=1}^{n} \sum_{j=1}^{m} \phi_j \mathcal{N}(\boldsymbol{y}_i | \boldsymbol{\mu}_j^\star, \mathbf{\Sigma}_j^\star).$$

By assuming $\boldsymbol{\mu}_j^\star = \mathbf{0}$ without losing generality, the negative log likelihood (NLL)

in terms of $\{\boldsymbol{\Theta}_k^\star\}$ is given by,

$$\text{NLL}(\boldsymbol{\theta}) = -\sum_{i=1}^{n} \log\Big(\sum_{j=1}^{m} \phi_j \mathcal{N}(\boldsymbol{y}_i|\mathbf{0}, \boldsymbol{\Theta}^{\star -1}_j)\Big), \tag{2.28}$$

where $\boldsymbol{\theta} = \{\phi_1, \cdots, \phi_m, \boldsymbol{\Theta}_1^\star, \cdots, \boldsymbol{\Theta}_m^\star\}$ is the model parameters.

Latent States. In order to solve the Equation 2.28, we follow the idea of Janson inequality and build a latent variable in the sum term of each expression in $\log$. Since there are m separate latent distributions, each data sample of the corresponding node $\boldsymbol{y}_i$ could come from one of the K distributions. We therefore construct a latent variable $\mathbf{Q}(z_{ij})$ which we constrain such that $\sum_{j=1}^{m} \mathbf{Q}(z_{ij}) = 1$. Then, the NLL function can be rewritten as follows:

$$\text{NLL}(\boldsymbol{\theta}) = -\sum_{i=1}^{n} \log \sum_{j=1}^{m} \Big(\frac{\mathbf{Q}(z_{ij}) p\big(\boldsymbol{y}_i|\boldsymbol{\Theta}_j^\star, \phi_j\big)}{\mathbf{Q}(z_{ij})}\Big) \tag{2.29}$$

$$= -\sum_{i=1}^{n} \log \sum_{j=1}^{m} \Big(\frac{p\big(\boldsymbol{y}_i, z_{ij}|\boldsymbol{\Theta}_k^\star, \phi_j\big)}{\mathbf{Q}(z_{ij})}\Big). \tag{2.30}$$

We next prove that this can be treated as the posterior probability of the i-th observation generated by the j-th distribution.

According to the Janson inequality, the expression in the Equation 2.29 can be rewritten for the EM algorithm to optimize the function, which can be split into expectation and maximization steps, respectively.

Expectation. First, according to the Jensen inequality, we know that when the optimal function is convex,

$$f(E(x)) \leqslant E(f(x)). \tag{2.31}$$

Because NLL is convex, and $\sum_{j=1}^{m}\left(\frac{p\left(\boldsymbol{y}_i, z_{ij}|\boldsymbol{\Theta}_j^{\star}, \phi_j\right)}{\mathbf{Q}(z_{ij})}\right)$ can be treated as the expectation of $p\left(\boldsymbol{y}_i, z_{ij}|\boldsymbol{\Theta}_j^{\star}, \phi_j\right)$. So we apply Jensen inequality here to find a lower bound:

$$\text{NLL}(\boldsymbol{\theta}) \leqslant -\sum_{i=1}^{n}\sum_{j=1}^{m}\mathbf{Q}(z_{ij})\log(p\left(\boldsymbol{y}_i, z_{ij}|\boldsymbol{\Theta}_j^{\star}, \phi_j\right)). \tag{2.32}$$

These terms are only equal when

$$\frac{p(\boldsymbol{y}_i, z_{ij})}{\mathbf{Q}(z_{ij})} = C, \tag{2.33}$$

where C is a constant. So, we simply have:

$$\sum_{j=1}^{m} p(\boldsymbol{y}_i, z_{ij}) = C\sum_{j=1}^{m}\mathbf{Q}(z_{ij}) = C, \tag{2.34}$$

$$\mathbf{Q}(z_{ij}) = \frac{p(\boldsymbol{y}_i, z_{ij})}{\sum_{j=1}^{m} p(\boldsymbol{y}_i, z_{ij})} = r_{ij}. \tag{2.35}$$

The equation of $\text{NLL}(\boldsymbol{\theta})$ is correct only when the constraint of $\mathbf{Q}(z_{ij})$ is true. Thus we can conclude that the latent variable is the posterior probability of the i-th observation generated by the j-th distribution. Therefore, we can compute each r_{ij} based on the initialization or update results of $\boldsymbol{\Theta}_j^{\star}$ and Œ_j.

Maximization. Given the $\boldsymbol{r}_{ij}^{(t)}$ from the Expectation step, we update $\hat{\phi}_j$, $\hat{\mathbf{H}}_j$ and $\hat{\boldsymbol{\Theta}}_j^{\star}$, respectively. First, we update $\hat{\phi}_j$ based as follows:

$$\hat{\phi}_j^{(t)} = \frac{1}{n}\sum_{i=1}^{n} r_{ij}^{(t)}. \tag{2.36}$$

The remaining problem is to find the optimal estimations of $\mathbf{H}_j$ and $\boldsymbol{\Theta}_j^{\star}$ that maxi-

mizes the expectation we obtain in the E step. Through a simple proof, it is equivalent to minimize the following function:

$$\begin{aligned} \min \sum_{i=1}^{n}\sum_{j=1}^{m} -r_{ij}^{(t)}\big(\log|\boldsymbol{\Theta}_j^\star| - \boldsymbol{x}_i^\top \mathbf{H}_j \boldsymbol{\Theta}_j^\star \mathbf{H}_j^\top \boldsymbol{x}_i\big), \\ \text{s.t.} \quad \boldsymbol{\Theta}^\star \succ 0, \mathbf{H} \geqslant 0, \mathbf{H}\mathbf{H}^\top = \mathbf{I}. \end{aligned} \tag{2.37}$$

Intuitively, the problem above is equivalent to m separate conventional graphical lasso sub-problems weighted by $r_{ij}^{(t)}$ where each sub-problem has the form of

$$\begin{aligned} \min -\log|\boldsymbol{\Theta}_j^\star| + \text{tr}(\tilde{\mathbf{X}}_j^\top \mathbf{H}_j \boldsymbol{\Theta}_j^\star \mathbf{H}_j^\top \tilde{\mathbf{X}}_j), \\ \text{s.t.} \quad \boldsymbol{\Theta}_j^\star \succ 0, \mathbf{H}_j \geqslant 0, \mathbf{H}_j\mathbf{H}_j^\top = \mathbf{I}, \end{aligned} \tag{2.38}$$

where $\tilde{\mathbf{X}}_j = (\sqrt{r_{1j}/s_j}\boldsymbol{x}_1^\top, \cdots, \sqrt{r_{nj}/s_j}\boldsymbol{x}_n^\top)$, $\boldsymbol{r}_{ij} = (r_{1j}, \cdots, r_{nj})^\top$ and $s_j = \sum_{i=1}^n r_{ij}$. Then we bring in the ℓ_1 regularization $\lambda||\boldsymbol{\Theta}_j^\star||_1$ to obtain the final objective function for the Maximization step.

This problem is not convex *w.r.t.* $\{\boldsymbol{\Theta}_j^\star\}$, but we could solve it alternatively for each $\boldsymbol{\Theta}_j^\star$ by regarding other $\boldsymbol{\Theta}_{j'\neq j}^\star$ fixed. Each sub-problem of $\boldsymbol{\Theta}_j^\star$ is exactly in the form of Equation 2.38 plus the ℓ_1 regularization terms. Thus the estimation of $\boldsymbol{\Theta}_j^\star$ could be solved by any existing method for solving Graphical Lasso without significant modifications. To estimate $\mathbf{H}_j$ we follow the algorithm similar to NMF, using Karush–Kuhn–Tucker (KKT) complementary slackness conditions to enforce the non-negativity and orthogonality constraints, then solving the estimation of $\mathbf{H}_j$ by the multiplicative update rule. Thus, we have:

$$(\hat{\mathbf{H}}_j^{(t+1)})_{\text{ls}} = \left(\hat{\mathbf{H}}_j^{(t)}\right)_{\text{ls}} \left(\frac{\tilde{\mathbf{X}}_j\tilde{\mathbf{X}}_j^\top\hat{\mathbf{H}}_j^{(t)}\hat{\boldsymbol{\Theta}}_j^{\star-} + \hat{\mathbf{H}}_j^{(t)}\lambda_1^-}{\tilde{\mathbf{X}}_j\tilde{\mathbf{X}}_j^\top\hat{\mathbf{H}}_j^{(t)}\hat{\boldsymbol{\Theta}}_j^{\star+} + \hat{\mathbf{H}}_j^{(t)}\lambda_1^+}\right)_{\text{ls}}. \tag{2.39}$$

Here λ_1 is $k \times k$ Lagrangian multip matrices following the non-negativity constraint and its compact expression follows as below:

$$\lambda_1 = -\hat{\mathbf{H}}_\mathbf{j}^\top \tilde{\mathbf{X}}_j \tilde{\mathbf{X}}_j^\top \hat{\mathbf{H}}_\mathbf{j} \hat{\mathbf{\Theta}}_j^\star. \tag{2.40}$$

To make sure each part is non-negative, We divide the λ_1 and $\hat{\mathbf{\Theta}}^\star$ into two parts, respectively:

$$\begin{aligned} \lambda_1 &= \lambda_1^+ - \lambda_1^-, \\ \lambda_1^+ &= \frac{(|\lambda_1| + \lambda_1)}{2}, \\ \lambda_1^- &= \frac{(|\lambda_1| - \lambda_1)}{2}. \end{aligned} \tag{2.41}$$

The same is true on the $\hat{\mathbf{\Theta}}_j^\star$. Thus we can make sure the sign of numerator and denominator are all positive, abiding by the non-negative constraint of $\mathbf{H}_j$.

In each iteration of the Maximization step, the alternating optimization repeats until all estimated $\hat{\mathbf{\Theta}}_j^\star$, $\hat{\mathbf{H}}_j$ and $\hat{\mathbf{\Phi}}_j$ become stable or reaches the maximal number of iterations. The final solutions to Equation 2.38 and the updated $\{\hat{\mathbf{\Phi}}_j\}$ are obtained using Equation 2.36 are used in the upcoming iteration of Expectation step to update the responsibility weights $\{r_{ij}\}$. This looping of Expecation and Maximization repeats until the loss function converges. The MNGL algorithm is also summarized in Algorithm 3.

Initialization. As shown in Algorithm 3, we need to provide starting values for each estimator. The following scheme we found empirically works well in our experiments. For each observation $i = 1, \ldots, n$, we distribute the observation randomly a class $j \in \{1, \ldots, m\}$. Then we assign a weight $\hat{r}_{ij} = 0.9$ for this observation i and distribution k and $\hat{r}_{ij} = \frac{0.1}{m-1}$ for all other distributions. In the Maximization

Algorithm 3 Algorithm for `MNGL`

Require: i: $\mathbf{X}$: The observations of D-variate Gaussian distribution
ii: m: the number of Gaussian distributions
iii: k: the number of nodes (groups)
iv: λ_1: the Lagrangian multiplier of the ℓ_1 regularization in graphical lasso
v: iter_{max}: the maximum number of iteration
Output: $\hat{\mathbf{\Theta}}_j^{\star}$, $\hat{\mathbf{H}}_\mathbf{j}$ and $\hat{\mathbf{C}}_\mathbf{j}$

1: Initialization: initialize $\hat{\mathbf{C}}_j^{(0)}$, $\hat{\mathbf{\Theta}}_j^{\star\,(0)}$,$\hat{\mathbf{H}}_j^{(0)}$ and $r_{ij}^{(0)}$
2: **repeat**
3: E step: Update the latent variable $r_{ij}^{(t)}$ with given $\hat{\mathbf{C}}_j^{(t-1)}$, $\hat{\mathbf{\Theta}}_j^{\star\,(t-1)}$ and $\hat{\mathbf{H}}_j^{(t-1)}$
4: M step: Update $\hat{\mathbf{C}}_j^{(t)}$, $\hat{\mathbf{\Theta}}_j^{\star\,(t)}$ and $\hat{\mathbf{H}}_j^{(t)}$ with $r_{ij}^{(t)}$
5: **until** $iter = iter_{\text{max}}$ or convergence

step, we update $\hat{\mathbf{\Theta}}_j^{\star}$ from the initial values $\hat{\mathbf{\Theta}}_j^{\star\,(0)}$ computed by CGL based on the whole samples and $\hat{\phi}_j$ from the initial values $\hat{\phi}_k = \frac{1}{m}$. Then for $\hat{\mathbf{H}}_j$, according to the Equation 2.39, we note that if $(\hat{\mathbf{H}}_j^{(t+1)})_{ls} = 0$ in one iteration, it will never jump out from this local solution. Thus, our experiments we initialize $\hat{\mathbf{H}}_j^{(0)}$ by performing k-means clustering then setting $\hat{\mathbf{H}}_j^{(0)} \leftarrow \hat{\mathbf{H}}_j^{(0)} + 0.2$.

2.3.4 Experimental Evaluations

To comprehensively evaluate the proposed model, we conduct experiments to answer the following research questions:

- *RQ 1*: How does sample size affect MNGL's performance relative to state-of-the-art alternatives?
- *RQ 2*: How robust is MNGL to the presence of noise compared to other recent models?

- *RQ 3*: How do hyper-parameters in comparative experiments impact each model's performance?
- *RQ 4*: How does the number of nodes affect each compared model?

Experiment Setup

Synthetic Data with Ground-Truth: We evaluate the performance of our model on synthetic data, where the ground-truth is known. The first step of generating these synthetic data is to build a mixture Gaussian distribution of network structure. By following the approach of task 2 in generating a single network, we generate m different block-diagonal matrix $\mathbf{\Theta}_j$ and $\mathbf{H}_j$ firstly. We refer to each diagonal block as the node in real-case. For each $\mathbf{\Theta}_j$, we give random sparity structures for each block $\mathbf{\Theta}_{G_i,G_j}$. In this chapter, we design each diagonal block $\mathbf{\Theta}_{G_i,G_i}$ in one $\mathbf{\Theta}_j$ with different scale. Thus by adjusting scale of diagonal blocks in different matrix $\mathbf{\Theta}_j$, we can make different network have different node parcelation. To simulate the connectivity of variables among diagonal and off-diagonal blocks, we control the connectivity of each variables on diagonal block with a high density, then giving a low density to each off-diagonal block. Following the above steps, we generate several different $\mathbf{\Theta}_j$ and $\mathbf{H}_j$. Then each $\mathbf{\Theta}_j^\star$ can be derived from $\mathbf{H}_j^\top \mathbf{\Theta}_j \mathbf{H}_j$.

Given $\mathbf{\Theta}_j$, we can thus obtain $\mathbf{\Sigma}_j$, which is the inverse of $\mathbf{\Theta}_j$. Due to the assumption of the independence of each Gaussian distribution, we obtain the covariance matrix $\mathbf{\Sigma}$ of the mixture Gaussian distribution. Then we generate n samples randomly from the mixture Gaussian distribution.

Compared methods: To demonstrate the effectiveness of our proposed method, we test against several state-of-art methods coherent brain network discovery meth-

ods:

- *CGL* [1]: CGL aims to achieve node discovery and directed edge detection at the same time. Meanwhile, it can distinguish direct links from indirect connections due to its solid probabilistic formulation.

- *ONMtF* [5]: ONMtF also aims to complete node discovery and edge detection at the same time. However, it focuses on explaining the spatial continuity of results. We only apply it on the task of nodes discovery, due to its inability of directed edge discovery.

- k-*means* + *CGL*: This pipeline method is more appropriate than CGL for the problem defined. We first employ *k*-means to assign each $\boldsymbol{x}_i$ to different nodes, then using CGLasso for each group to obtain the final $\hat{\boldsymbol{\Theta}}_j^\star$ and $\hat{\mathbf{H}}_j$.

- k-*means* + *OMNtF*: This is also a pipeline method that first splits the whole sample of $\boldsymbol{x}_i$ into different nodes by using *k*-means, then using ONMtF on each node to obtain each $\hat{\boldsymbol{\Theta}}_j^\star$ and $\hat{\mathbf{H}}_j$.

- k-*means* + *JGL* [38]: A Joint Graphical Model is proposed in [38], which aims to discover a mixture Gaussian distribution. However, it applies to the level of nodes. We therefore employ *k*-means to map $\boldsymbol{x}_i$ into the node space of $\boldsymbol{y}_i$ first.

Experiment Setting: We simulate four scenarios by changing one parameter and keeping the others fixed. Each scenario aims to study one of aforementioned research questions (*RQ*). In these situations, we select sample size n, the standard

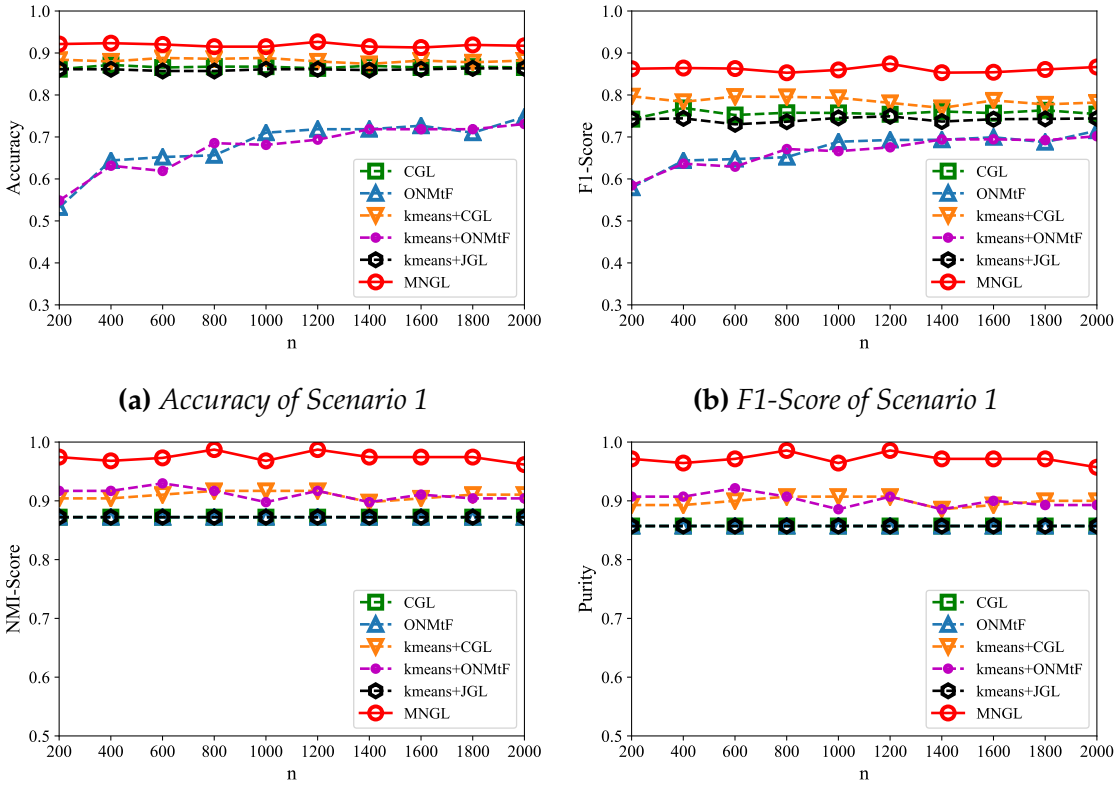
error of noise œ, the variables number p of $\boldsymbol{x}_i$, and the group number k as the controlled parameters.

- *Scenario 1*: We fix $p = 70$ (the number of variables), œ $= 0$ (the standard error of noise), $k = 5$ (the number of nodes) and then control sample size n from 200 to 2000.
- *Scenario 2*: We fix $n = 2000$, $p = 70$ and $k = 5$, meanwhile control œ from 2 to 5.
- *Scenario 3*: We fix $n = 2000$, œ $= 0$ and $k = 5$, and then control p from 70 to 350.
- *Scenario 4*: We fix $n = 2000$, œ $= 0$, and then control k from 3 to 11.

To generalize the results of comparative experiments, we sample 10 times for all experiments and average their results to evaluate the precision and stability of our model.

Comparative Results

To study the effect of sample size on the performance of MNGL, we design comparative experiments based on *Scenario 1*. Figure 2.13 shows the comparative results. We compare our proposed model with five baseline methods. The first row shows the results of the comparison on edge detection; the second row shows the results for node discovery. In the results of all scenarios, we use the same symbol, which is illustrated in the caption below Figure 2.13. From the results in the first row of Figure 2.13, we observe that the sample size n indeed affects some methods,



(a) *Accuracy of Scenario 1*

(b) *F1-Score of Scenario 1*

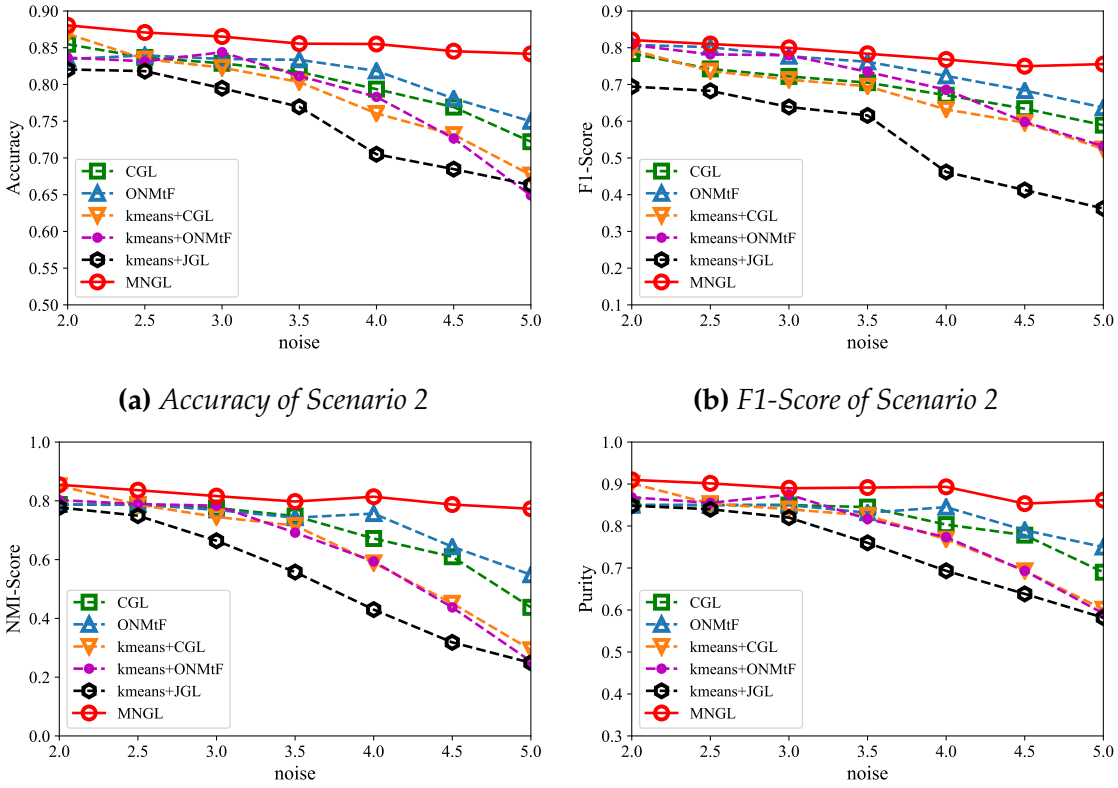
(c) *NMI of Scenario 1*

(d) *Purity of Scenario 1*

Figure 2.13: *Comparison of each method on edge detection and node discovery. The first row shows the results of edge detection, and the second shows the results of node discovery. The four sub-figures above consider different sample size n from 200 to 2000. The other parameters are left fixed.*

especially ONMtF and its derivations. As the sample size increases, the accuracy of these two methods become much higher. Encouragingly, this factor has no significant effect on our model. Overall, we can clearly see that our method MNGL is more accurate and robust than other methods as the sample size n changes. Meanwhile, n does not have a significant impact on the performance of MNGL, which means that our model performs well even with a small training set.

To study the effect of noise on the performance of MNGL, we use the experimental setup *Scenario 2*. Our results are shown in Figure 2.14 where the horizontal



(a) *Accuracy of Scenario 2*

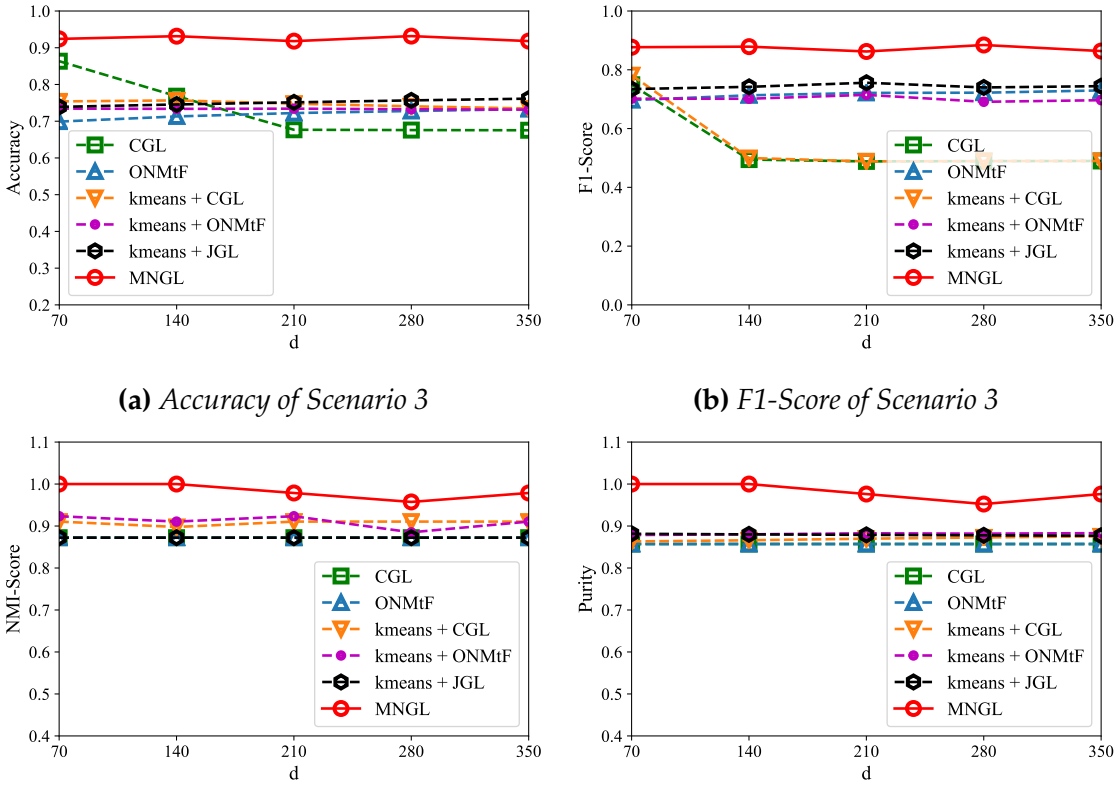
(b) *F1-Score of Scenario 2*

(c) *NMI of Scenario 2*

(d) *Purity of Scenario 2*

Figure 2.14: *The four sub-figures above consider different* œ *(the standard error of noise) from 2 to 5, meanwhile fix the other parameters, which correspond to scenario2;*

axis in the figure represents the standard error of noise œ. The larger the œ, the stronger the noise. It leads to smaller signal-to-noise ratio, which means it is more difficult to mine the network structure from the available samples. As seen in the four sub-graphs, we find that noise affects all compared methods. In particular, while JGL suffers the most influence, ONMtF and derivatives of it are more robust than CGL and its derivatives in this scenario. Meanwhile, our method, MNGL, is better than all other comparison methods in this experiment for both edge detection and node discovery. Furthermore, œ does not significantly decay the performance of MNGL.



(a) *Accuracy of Scenario 3*

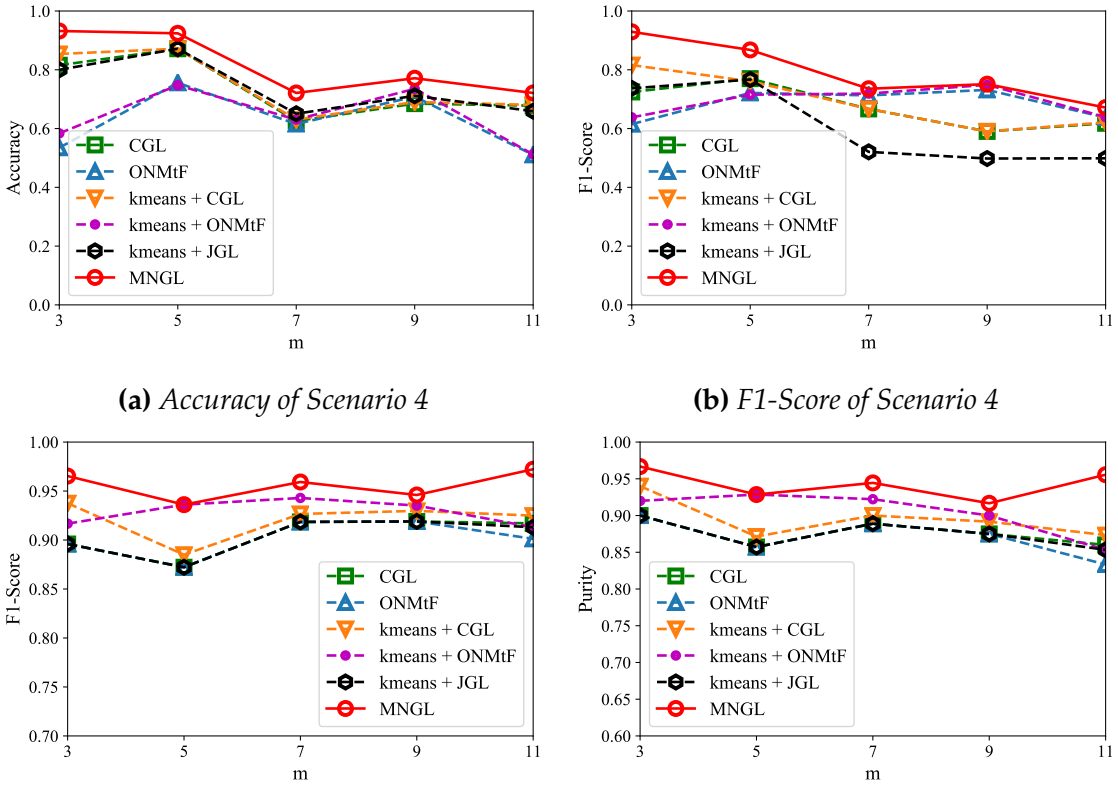
(b) *F1-Score of Scenario 3*

(c) *NMI of Scenario 3*

(d) *Purity of Scenario 3*

Figure 2.15: *The four sub-figures above consider different p (the number of variables $\mathbf{x}_i$) from 70 to 350, meanwhile fix the other parameters, which correspond to scenario3;*

To study the effect of the number of variables on the performance of MNGL, we next use *Scenario 3*, the results for which are shown in Figure 2.15. For the node discovery task, we see that the dimension of feature space has no impact on the performance of any methods. However, for edge detection, when the dimension is low, the performance of CGL and its derivatives outperforms the other baseline methods. In particular, as the dimension increases, the accuracy of CGL and its derivatives shows a significant downward trend compared to the others. Again, as expected, in this scenario the performance of MNGL is more accurate and robust than the baseline methods.



(a) *Accuracy of Scenario 4*

(b) *F1-Score of Scenario 4*

(c) *NMI of Scenario 4*

(d) *Purity of Scenario 4*

Figure 2.16: *The four sub-figures above consider different k (the number of nodes $\mathbf{y}_i$) from 3 to 11, meanwhile fix the other parameters, which correspond to scenario4;*

To study the effect of the number of states on the performance of MNGL, we turn to *Scenario 4* and report our findings in Figure 2.16. Across all sub-figures, as the number of nodes increases, the robustness of all compared methods shows a downward trend. Specifically, for the edge detection task, the accuracy of CGL and its derivatives perform slightly better than ONMtF and its derivatives. However, when considering the F1-Score, ONMtF outperforms the CGL methods. For node discovery, ONMtF and its derivatives show more robustness than other baseline methods. Overall, MNGL still significantly outperforms the compared methods in these settings, though there is a small degree of fluctuation.

Combining the four *RQ*s raised above and the results of all these comparative experiments, we can draw the following four conclusions: First, ONMtF and its derivatives are not as good as other methods in the case of insufficient samples. Second, CGL and its derivatives are more restrictive in high-dimensional space. Third, Both the accuracy and robustness of all comparative methods will decrease under the impact of noise and the number of nodes. Fourth, compared to the alternative methods, our proposed method MNGL exhibits greater accuracy and robustness in each scenario, indicating that neither sample size n, the dimension of feature space p, noise œ nor the number of nodes k significantly degrades the performance of our method.

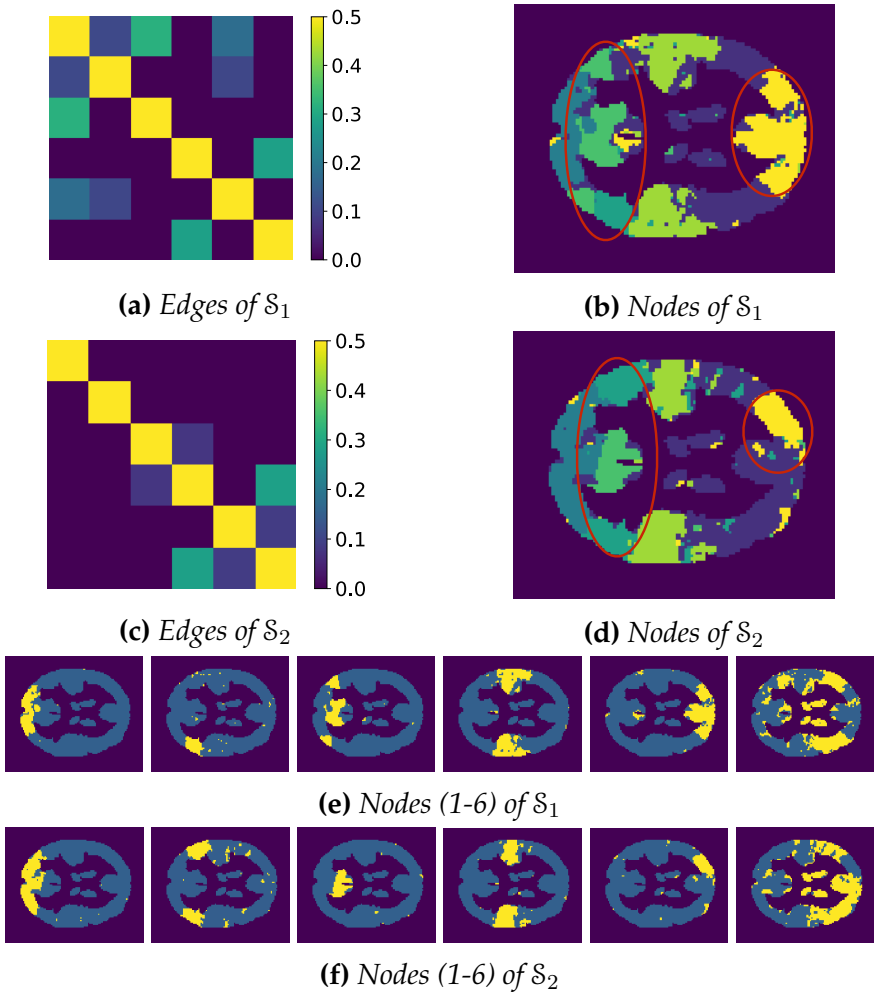
Real-World Datasets

We also evaluate our proposed method on the fMRI dataset from the ADHD-200 project . Again, this part of the experiment lacks ground-truth as a reference to measure the accuracy and robustness of the model. Therefore we must consider the interpretability and rationality of the results. Specific to our proposed model, we are primarily concerned with whether or not our model can mine different cognitive networks from the fMRI datasets (various node assignments and functional connectivity). Therefore for this section, we focus solely on applying our proposed method, MNGL, to this challenging task.

In our experimental setting, we focus on the multi-state brain network discovery among the same subjects, and report the results of both nodes discovery and edge detection. To assign the voxels that can be considered as parts of the brain, we use anatomical automatic labeling (AAL) brain-shaped mask, which is provided

by neurology professionals. We follow [33] and use a middle slice of these scan for the ease of presentation. Consequently, each of the brain scans can be represented by about 3281 voxels. So it is more conventional for the visualization of the results. The datasets is a 3281 (variables) $\times$ 2992 (time steps) datasets and reasonably assume that they are drawn from a mixture Gaussian distribution. However, the number of Gaussian distributions m and the number of nodes k are both unknown and need to be selected in advance. Through repeated experimental observations, we find that $m = 2$ and $k = 6$ can provide the most reasonable results on the data sets.

Figure 2.17 shows the multi-state network discovered by MNGL on the fMRI datasets. The results of edge detection and node discovery are shown on the first and second line, which corresponds to the functional network of state $\mathcal{S}_1$ and $\mathcal{S}_2$ respectively. Meanwhile, each inferred node is displayed on the third and fourth line individually. First, we can see the difference between the two networks from the discovered edges and nodes. We deliberately mark the differences between the nodes of two networks with red circles. More specifically, in the first line of Figure 2.17, we find a strong and complete default mode network (DMN) for ADHD subjects, corresponding to group 3 and 5 in the third line. A DMN is a network of interacting brain regions known to have activity highly correlated with each other while being distinct from other networks in the brain, including the Parietal and Occipital Lobes, the Cingulum Region Posterior, and the Frontal Cortex. However, in the second line of Figure 2.17, this mode is not intact. In particular, the Frontal Cortex is missing in the network, while the rest of the connections are different from the functional network in the first line. The specific relationship between



(a) *Edges of* $\mathcal{S}_1$ **(b)** *Nodes of* $\mathcal{S}_1$

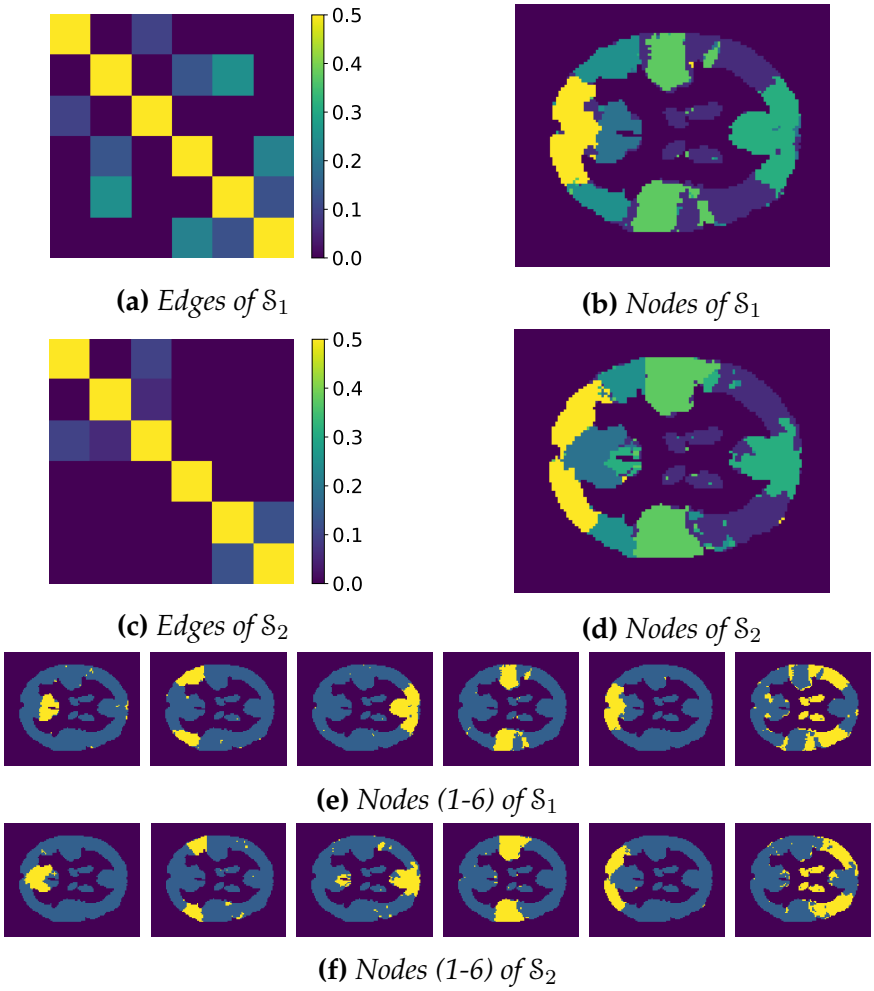
(c) *Edges of* $\mathcal{S}_2$ **(d)** *Nodes of* $\mathcal{S}_2$

(e) *Nodes (1-6) of* $\mathcal{S}_1$

(f) *Nodes (1-6) of* $\mathcal{S}_2$

Figure 2.17: *Discovered results of multi-state brain network in ADHD subjects* ($k = 6$).
each node can be found in the sub-figures of discovered edges.

For comparison, we apply MNGL again on the subjects of TDC, which represent the group of typically-developing children. We can observe from Figure 2.18 that, although there are differences in the node assignments and the connectivity structure among each node, there is no deletion of DMN in each network. At the same time, the network of state $\mathcal{S}_1$ from TDC and that from ADHD have a certain degree of similarity from the nodes result to the connectivity structure, which allows us to have more reference when analyzing the differences and connections be-



(a) *Edges of* $\mathcal{S}_1$ **(b)** *Nodes of* $\mathcal{S}_1$

(c) *Edges of* $\mathcal{S}_2$ **(d)** *Nodes of* $\mathcal{S}_2$

(e) *Nodes (1-6) of* $\mathcal{S}_1$

(f) *Nodes (1-6) of* $\mathcal{S}_2$

Figure 2.18: *Discovered results of multi-state brain network in TDC subjects* ($k = 6$).

tween the two subjects. These results give us reason to believe that the brain scans of these subjects have some similar functional structure correspondences similar to on-task and off-task states.

Despite the lack of ground-truth, we believe that the current results are still consistent with the problem defined in this chapter: We find strong evidence that there is a multi-state brain cognitive network in the fMRI datasets, and our proposed model MNGL can effectively mine this mixture network structure.

3

Spike Trains Inference

3.1 Task 4: High Dimension Spike Trains Classification

3.1.1 Motivation

Our brains have about a hundred billion neurons that fire signals to communicate with each other all the time. Each signal is electrochemical in nature and is referred to as a spike, or an action potential. The most popular way to think of spike *trains* is as a digital sequence of events: 1 for a spike, and 0 for no spike. Such spike trains arise during physical sensory stimuli such as vision and motion, or abstract stimuli such as memory. Recently, spike train classification has attracted much attention in the field of data mining [55, 56, 57, 58]. Unlike tradition classification, classifying spike trains is a task with sequences of spikes as both inputs and outputs. By assuming that all spikes are discrete characteristic events, the processing

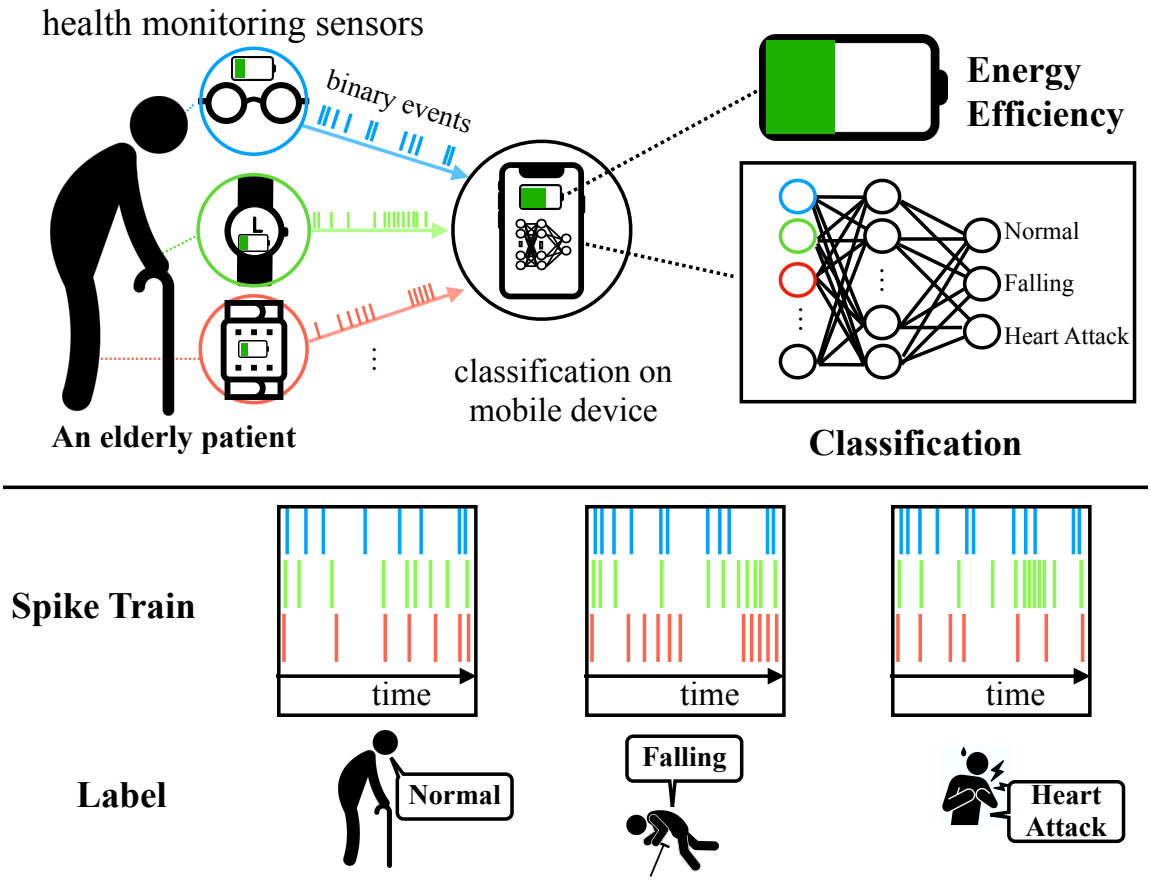


Figure 3.1: *An example of energy-efficient spike train classification problem.*

of information is reduced to the timing and counting of said spikes. Designing machine learning algorithms for spike train classification is very important in many high-impact fields such as sensor systems for disease diagnosis and human activity monitoring.

Spiking Neural Network (SNN) show great potential for dealing with spike train classification [57, 58, 59, 60, 61]. Originally proposed to imitate biological information processing [62], the neurons transfer information between one another via spike trains. Unlike Recurrent Neural Networks (RNN), which use continuous value as inputs and outputs, SNNs take sparse spike trains as inputs and outputs,

building large-scale neural networks with far less energy and memory on neuromorphic hardware systems, which operate on principles that are fundamentally different from standard digital computers. Thus, SNNs are clear candidates for spike train classification.

However, opportunities are always accompanied by challenges. Due to significant advances in miniaturization of sensor systems, more and more smart devices such as wearable sensors and smart phones for elderly care and aerial robots appear around us, which can produce high-dimensional data in the form of spike trains. These devices require high quality pattern recognition to meet their design requirements. At the same time, they are often limited by available energy and thus low computational cost is required during inference. As illustrated in Figure 3.1, wearable devices incorporated with varieties of motion sensors are used to monitor different physical conditions of seniors for identifying their body conditions. They generate data that cover massive measurements, including heart rate (HR), blood pressure (BP), and oxygen saturation (SpO2), among others, and are expected to be collected by smart devices subject to limited power.To run SNNs efficiently on such high-dimensional data, we need to ensure both high classification accuracy *and* low computational cost during inference.

Regarding computational cost, however, modern SNNs [57, 58] perform many unnecessary computations due to their dense network architectures. These unnecessary computations are caused by *weak* connections between neurons. Weak connections play a limited role in model performance during inference, as shown in the example in Figure 3.1. On one hand, inferring *Falling* needs activity-related signals given by a person's intertial measurement units (IMU), HR, and BP. On the

other hand, inferring *Heart Disease* doesn't consider signals from IMUs, but needs BP, SpO2 and could use more measurements from their photoplethysmograms. As a result, current SNNs are still not suitable for spike train classification, especially when the data is high dimensional and comes from devices with limited power.

In this work, we propose an energy-efficient method for high-dimensional spike train classification. To solve this problem, there are two main challenges:

- *Sparse SNN vs Sparse RNN*: Sparsification techniques have been employed in RNNs [63] to reduce computational costs. However, RNNs model sequences via continuous values, and are outmatched by SNNs for spike train classification. By sparsifying SNNs, we can avoid unnecessary computations caused by weak connections between neurons. A sparsified model has far fewer non-zero parameters and so performs fewer computations during inference, making it more power efficient. Thus, instead of using sparse RNNs, we must find a way to sparsify the network structure of SNNs.

- *Sparsifying Inputs vs Networks*: SNNs have been accelerated by either sparsifying the *inputs* [64] or using stochastic computing [65]. However, by only focusing on spike *rate*, as opposed to spike *timing*, they disregard a major component of the problem. A successful method must consider both rate and timing together to successfully perform high-dimensional spike train classification.

Inspired by the success of Artificial Neural Networks (ANN) with a sparse structure [66, 67], we propose an SNN model with sparse spatio-temporal coding. We reparameterize the connection between each neuron in an SNN by multiplying

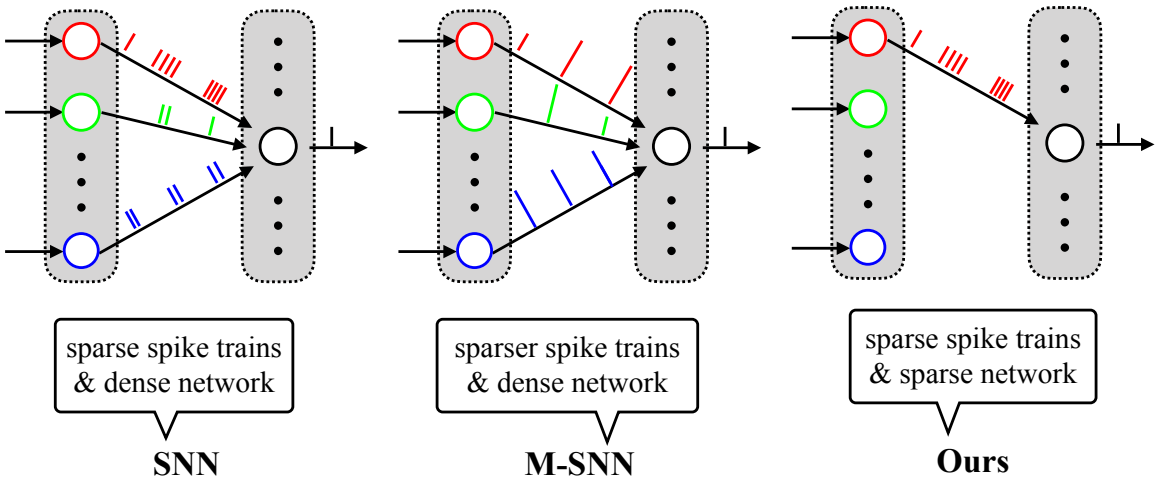


Figure 3.2: *Comparison of the key differences between SNNs [57, 60], M-SNNs [64], and proposed sparse SNN.*

each original weight by a binary "gate". Each gate is considered to be a Bernoulli random variable. As a result, our proposed approach allows each neuron in the SNN to consider the necessity of coming into contact with each neuron in the next layer. Therefore, it allows us to penalize the possibility of each gate for being different nonzero with no further restrictions, thereby pruning weak links. This reduces the overall computational cost and adds the benefit of regularization, reducing overfitting. We show through empirical evaluation on multiple real-world datasets that, compared to baselines, the sparse SNN we propose greatly speeds up computation while incurring only a negligible deterioration in classification performance. Meanwhile, we also show improved generalizability by varying the size of the training set.

3.1.2 Related Works

In spike train classification, “indirect" learning methods, such as ANN-to-SNN conversion [68, 69, 70, 71], have been proposed to high dimensional inputs. These

are indirect learning methods because a regular non-spiking ANN (e.g., a multi-layer perceptron) is initially used during the training phase. At inference-time, the trained model is then converted to an SNN. However, there are several disadvantages associated with such indirect training. First, it doesn't align well with how an SNN operates. In ANNs, it does not matter if activations are negative, but firing rates in SNNs are always positive. Furthermore, many limiting constraints are typically added while training the ANN models. These include not using bias terms, only supporting average pooling, and only using ReLU activation functions.

In response, methods for directly training an SNN have recently been proposed [58, 60, 72]. These approaches are mainly based on conventional gradient descent. Most notably, different from previous techniques based only on spatial back-propagation [72, 73], SNNs trained directly using back-propagation in both the spatial as well as the temporal domains [58, 60] have achieved state-of-the-art accuracy on the MNIST and N-MNIST datasets. However, although these methods perform better than the others described above on many real-world datasets, from the perspective of computational efficiency, they are still far from power-efficient in solving high-dimensional spike trains classification. Therefore there have been some recently-proposed power-efficient SNNs [64, 65]. [64] aims to enforce more neurons silence by making input spikes of each neuron sparser. [65] introduces a stochastic SNN by exploiting the benefits of stochastic computing to generate input spike trains and reduce the connection complexity. However, both of them are only applicable to standard datasets, but not to neuromorphic datasets.

3.1.3 Preliminary: Spiking Neural Network

Typically, we use SNN for spike train classification problem. To describe the SNN models with a sparse structure, we first introduce the baseline framework for SNNs, as proposed by [58]. We begin by describing the simplest possible SNN, one which comprises a single neuron with one input entry. This neuron is a recurrent unit that is affected by the current input, and the previous input and output. For each timestep t, it combines the current input with the previous input and output to compute a new value. This value can be referred to as the *membrane* potential in biological neural network. If the membrane is greater than a threshold, the neuron fires and outputs 1 to indicate a spike, otherwise, it outputs 0 to indicate silence. Therefore, for each timestep t, the membrane and output are expressed as follows:

$$u_t = \tau u_{t-1}(1 - z_{t-1}) + wx_t + b, \tag{3.1}$$

$$z_t = \Theta(u_t - \vartheta), \tag{3.2}$$

where we write u_t, x_t, and z_t to denote the membrane potential, input, and output of the neuron on timestep t, respectively. $\tau \in [0, 1]$ is the time decay constant hyperparameter, and w and b are the connection and bias between input and this neuron, respectively. $\Theta(\cdot)$ is the step function, which satisfies $\Theta(x) = 0$ when $x < 0$, otherwise $\Theta(x) = 1$.

The SNN expressed in Equations 3.1-3.2 mimics natural neural networks more closely than a traditional ANN. In this way, we represent a neuron as the parallel combination of a "leaky" resistor and a capacitor. The second term of the r.h.s. of Equation 3.1 is used as external current input to charge up the capacitor to update

the potential u_t. If the neuron emits a spike $z_t = 1$ at timestep t, the capacitor discharges to a resting potential (which we fix at zero throughout this chapter) by using the first term in Equation 3.1.

An SNN is built by hooking together many of these simple "neurons", so that the output of a neuron can be the input of another. We let $u_i^{t,n}$ and $z_i^{t,n}$ denote the membrane and output of neuron i in layer n at timestep t. The network has parameters $\mathbf{W} = \{\mathbf{W}^1, \ldots, \mathbf{W}^{N-1}\}$, where $\mathbf{W}_{ij}^n$ denote the parameter associated with the connection between neuron j in layer n, and neuron i in layer $n+1$. We also let $l(n)$ denote the number of neurons in layer n and let N be the number of layers in our network. Therefore, for layer $n \in \{2, \ldots, N\}$, we write $\mathbf{u}^{t,n} = (u_1^{t,n}, \ldots, u_{l(n)}^{t,n})^\top$ and $\mathbf{z}^{t,n} = (z_1^{t,n}, \ldots, z_{l(n)}^{t,n})^\top$ to denote the membrane and output vector of neurons in layer n at timestep t. For $n = 1$, we will use $\mathbf{z}^{t,1} = \mathbf{x}^t$ to denote the input vector. Thus, the expression of an SNN is given by:

$$\mathbf{u}^{t,n} = \tau \mathbf{u}^{t-1,n} \odot (1 - \mathbf{z}^{t-1,n}) + \mathbf{W}^{n-1}\mathbf{z}^{t,n-1}, \tag{3.3}$$

$$\mathbf{z}^{t,n} = \Theta(\mathbf{u}^{t,n} - V_{\text{th}}). \tag{3.4}$$

From Equations 3.3-3.4, the spike signals not only propagate through the layer-by-layer spatial domain, but also affect the neuronal states through the temporal domain. Therefore, it considers both the spatial and temporal directions during the error backpropagation, *i.e.*, spatio-temporal backpropagation (STBP) [57, 58], which significantly improves the network accuracy. During backpropagation, because the activity function $\Theta(\cdot)$ is non-differentiable, it is common to use the rectangular function to approximate the corresponding derivative.

Given the expressions above, we can easily solve a standard SNN classification problem by training a classifier f: $\mathbb{R}^{P\times T} \mapsto \{1,\ldots,N\}$ on a given dataset $\{(\mathbf{x}^{(1)},y^{(1)}),\ldots,(\mathbf{x}^{(K)},y^{(K)})\}$ that contains K training samples, of which each instance $\mathbf{x}^{(i)} \in \mathbb{R}^{P\times T}$ has an observed label $y^{(i)} \in \{1,\ldots,l(N)\}$. P is the number of input entries and T denotes the length of spike train. To train the SNN, we define the following loss function L for a single training example $(\mathbf{x},y)$:

$$L = \left(y - \frac{1}{T}\sum_{t}^{T} \mathbf{M}\mathbf{z}^{t,N}\right)^2 \tag{3.5}$$

where $\mathbf{z}^{t,N}$ denotes the voting vector of the last layer N at time step t, $\mathbf{M}$ denotes a constant voting vector connecting neurons in the output layer to a specific class. Thus, we can use STBP to propagate the gradients $\frac{\partial L}{\partial o_i^{t,n+1}}$ from the $(n+1)$-th layer and $\frac{\partial L}{\partial o_i^{t+1,n}}$ from time step $t+1$ as follows:

$$\frac{\partial L}{\partial o_i^{t,n}} = \sum_{j=1}^{l(n+1)} \frac{\partial L}{\partial o_j^{t,n+1}}\frac{\partial o_j^{t,n+1}}{\partial o_i^{t,n}} + \frac{\partial L}{\partial o_i^{t+1,n}}\frac{\partial o_i^{t+1,n}}{\partial o_i^{t,n}} \tag{3.6}$$

$$\frac{\partial L}{\partial u_i^{t,n}} = \frac{\partial L}{\partial o_i^{t,n}}\frac{\partial o_i^{t,n}}{\partial u_i^{t,n}} + \frac{\partial L}{\partial o_i^{t+1,n}}\frac{\partial o_i^{t+1,n}}{\partial u_i^{t,n}} \tag{3.7}$$

Sparsity regularization and optimization

In this work, we propose a sparsification procedure for deep SNNs that accelerates both training and inference while improving the their generalization capabilities through regularization. To build a sparse structure, we consider a re-

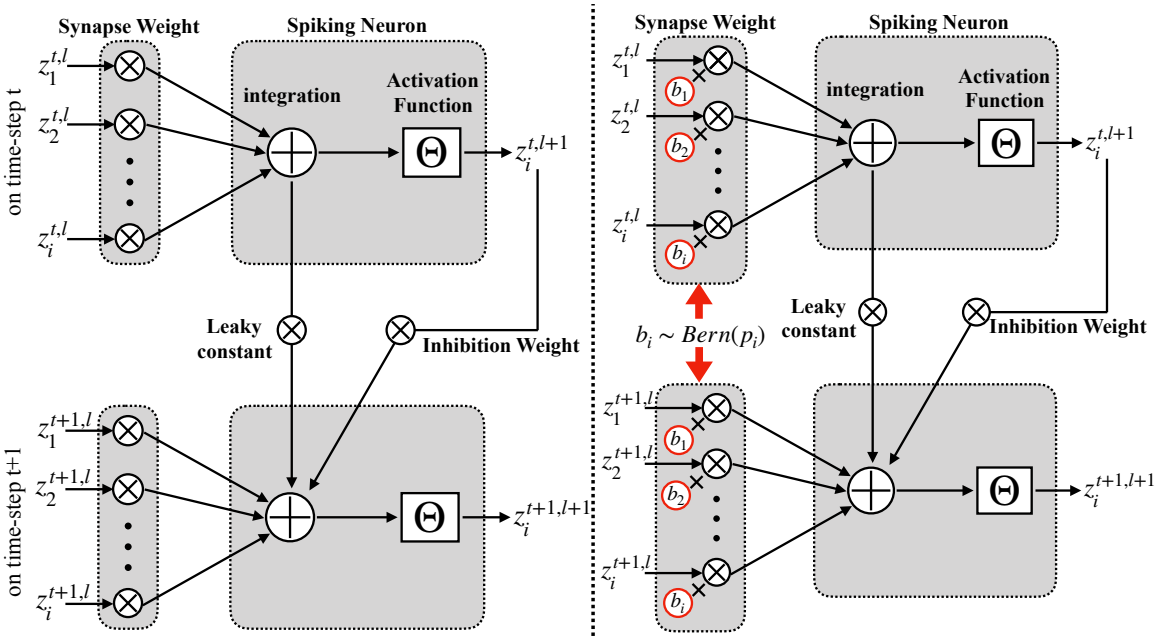


Figure 3.3: *Difference between traditional and sparse SNNs.*

parametrization of $\mathbf{W}_{ij}^n$, inspired by [67]:

$$\mathbf{W}^n = \tilde{\mathbf{W}}^n \odot \mathbf{b}^n, \quad \mathbf{b}_{ij}^n \in \{0, 1\}, \quad \tilde{\mathbf{W}}_{ij}^n \neq 0 \tag{3.8}$$

where the $\mathbf{b}_{ij}^n$ correspond to binary "gates" that denote whether the corresponding parameter $\tilde{\mathbf{W}}_{ij}^n$ is utilized or not utilized. $\tilde{\mathbf{W}}^n$ and $\mathbf{b}^n$ is also independent of time t. To simplify the later derivations, we reformulate the minimization of Equation 3.5 as $L = f(y, \mathbf{x}; \tilde{\mathbf{W}}, \mathbf{b})$.

By letting $p(\mathbf{b}_{ij}^n | \mathbf{\Pi}_{ij}^n) = \text{Bern}(\mathbf{\Pi}_{ij}^n)$ be a Bernoulli distribution over each gate $\mathbf{b}_{ij}^n$, we reconsider a sparse network structure as a regularized minimization procedure with a regularization on the number of parameters being used, on average, as fol-

lows:

$$L = L_E + L_C, \tag{3.9}$$

$$L_E = \mathbb{E}_{p(\mathbf{b}|\mathbf{\Pi})}\left[f(y, \mathbf{x}; \tilde{\mathbf{W}}, \mathbf{b})\right], \tag{3.10}$$

$$L_C = \lambda \sum_{n=1}^{N} \|\mathbf{\Pi}^n\|_1 \tag{3.11}$$

where L_E denotes the expectation of loss with respect to the Bernoulli distribution of $\mathbf{b}$. Meanwhile, L_C corresponds to the complexity loss that measures the sparsity of the model. Due to the positive nature of each $\mathbf{\Pi}_{ij}^n$, this term also corresponds to the expectation of the amount of gates being "on." Based on [74], the objective described in Equation 3.9 is a close surrogate to a variational bound involving a spike and slab distribution over the parameters and a fixed coding cost for the parameters when the gates are active. However, the first term in Equation 3.9 is problematic for $\mathbf{\Pi}$ due to the discrete nature of $\mathbf{b}$, which does not allow for efficient gradient-based optimization. The unbiased gradient estimator in [75] could be employed, however, it suffers from high variance. The straight-through estimator in [76] can also be used in this problem, but it provides biased gradients as it ignores the Heaviside function during gradient evaluation.

In this chapter, inspired by [67], we find a simple alternative way to smooth the objective function such that we allow for efficient gradient-based optimization of Equation 3.9. Let $\mathbf{s}_{ij}^n$ be a continuous random variable with a distribution $q(\mathbf{s}_{ij}^n)$ that has parameters $\mathbf{\Phi}_{ij}^n$. We can now let each gate be given by a hard-sigmoid

rectifiation of $\mathbf{s}_{ij}^n$ as follows:

$$\mathbf{s}_{ij}^n \sim q\left(\mathbf{s}_{ij}^n | \mathbf{\Phi}_{ij}^n\right), \quad \mathbf{b}_{ij}^n = \min\left(1, \max(0, \mathbf{s}_{ij}^n)\right) \tag{3.12}$$

This allows $\mathbf{b}_{ij}^n$ to be exactly zero. Due to the i.i.d assumption of each $\mathbf{s}_{ij}^n$, we can thus smooth the binary Bernoulli gates by replacing each $\mathbf{b}_{ij}^n$ appearing in the first term of Equation 3.9 with $\mathbf{s}_{ij}^n$ and the second term with the probability of the variable $\mathbf{s}_{ij}^n$ being positive:

$$L_E = \mathbb{E}_{q(\mathbf{s}|\mathbf{\Phi})}\left[f(y, \mathbf{x}; \tilde{\mathbf{W}}, \mathbf{s})\right], \tag{3.13}$$

$$L_C = \lambda \sum_{ijn} P(\mathbf{s}_{ij}^n > 0 | \mathbf{\Phi}_{ij}^n) \tag{3.14}$$

Here we similarly have a cost that explicitly penalizes the probability of a gate being different from zero, thus Equations 3.13-3.14 act as a close surrogate to the original loss function in Equation 3.10-3.11. By following the reparameterization trick [77], we can describe the expression in Equation 3.13 as an expectation over a parameter-free noise distribution $p(\epsilon)$ and a deterministic and differentiable transformation $g(\cdot)$ of the parameter $\mathbf{\Phi}$ and ϵ. This allows us to make the following Monte Carlo approximation to the intractable expectation over the noise distribution:

$$L_E = \frac{1}{M} \sum_{m=1}^{M} \left[f(y, \mathbf{x}; \tilde{\mathbf{W}}, \mathbf{s}^{(m)})\right], \tag{3.15}$$

$$\mathbf{s}^{(m)} = \min\left(1, \max(0, g(\mathbf{\Phi}, \epsilon^{(m)}))\right), \epsilon^{(m)} \sim p(\epsilon) \tag{3.16}$$

Next we provide more details about $g(\cdot)$ in Equations 3.16.

The hard concrete distribution

The framework above enables us to employ efficient stochastic gradient-based optimization, while still allowing for exact zeros of the parameters. For the differentiable transformation $g(\cdot)$, we follow [78]: assume that we have a binary concrete random variable s distributed in the interval (0, 1). The parameters of this distribution include $\log \alpha$ and β, where $\log \alpha$ denotes the location and β is referred to as the temperature.

Temperature β controls the degree of approximation. With $\beta = 0$, we recover the original Bernoulli distribution, whereas with $0 < \beta < 1$ we obtain a probability density that concentrates its mass near 0 and 1. Therefore the hard concrete distribution can inherit statistical properties very similar to that of the Bernoulli distribution. We then stretch s to the interval (γ, ς), with $\gamma < 0$ and $\varsigma > 1$. Following [78], we fix $\gamma = -0.1, \varsigma = 1.1,$ and all $\beta = \frac{2}{3}$ throughout this chapter. Then we sample b based on the expressions as follows:

$$s = \sigma\left(\left(\log u - \log(1-u) + \log \alpha\right)/\beta\right), \tag{3.17}$$

$$\bar{s} = s(\varsigma - \gamma) + \gamma, \quad u \sim U(0,1), \tag{3.18}$$

$$b = \min(1, \max(0, \bar{s})). \tag{3.19}$$

Thus, the complexity loss L_C of the objective function in Equation 3.14 under

the hard concrete distribution can be calculated as:

$$L_C = \sum_{ijn} \sigma \left(\log \alpha_{ij}^n - \beta \log \frac{-\gamma}{\varsigma} \right). \tag{3.20}$$

Given these derivations, we can easily obtain the corresponding iterative state update equations and gradients for sparse deep SNNs.

$$\mathbf{u}_i^{t+1,n+1} = k_\tau \mathbf{u}_i^{t,n+1} \left(1 - \mathbf{o}_i^{t,n+1}\right) + \sum_j^{l(n)} \tilde{\mathbf{W}}_{ij}^n \mathbf{b}_{ij}^n \mathbf{o}_j^{t+1,n} \tag{3.21}$$

$$\mathbf{o}_i^{t+1,n+1} = \Theta\left(\mathbf{u}_i^{t+1,n+1} - V_{\text{th}}\right) \tag{3.22}$$

$$\mathbf{b}_{ij}^n = \min\left(1, \max(0, \bar{\mathbf{s}}_{ij}^n)\right), \tag{3.23}$$

$$\bar{\mathbf{s}}_{ij}^n = \mathbf{s}_{ij}^n(\varsigma - \gamma) + \gamma, \tag{3.24}$$

$$\mathbf{s}_{ij}^n = \sigma\left(\left(\log u - \log(1-u) + \log \alpha_{ij}^n\right)/\beta\right) \tag{3.25}$$

We also summarize the overall training process of our proposed sparse SNNs as pseudo-code in Algorithm 4.

3.1.4 Experimental Evaluations

To comprehensively validate the effectiveness of our proposed method, we conduct experiments to answer two questions: First, we are interested in computational improvement with very negligible degradation in accuracy. Our work in this chapter thus aims to improve the state-of-the-art SNN in this regard. We choose the Spiking CNN (SCNN) [58] as the basic model to which we apply our proposed sparsification procedure on this model and name it sparse SCNN. We then com-

Algorithm 4 Training code for sparse SNN

Require: : i: Network inputs $\{X^t\}_t^T$; ii: class label Y; iii: parameters and states of convolutional layers ($\{\mathbf{W}^n, \mathbf{b}^l, \mathbf{u}^{0,n}, \mathbf{o}^{0,n}\}_{n=1}^{N_1-1}$); iv: full-connected layers ($\{\mathbf{W}^n, \mathbf{b}^n, \mathbf{u}^{0,n}, \mathbf{o}^{0,n}\}_{n=1}^{N_2-1}$); v: simulation window T; vi: the parameters of the hard-concrete distribution ($\log\alpha^n, \beta, \gamma, \varsigma$); vii: the parameters of iterative LIF ($T, k_\tau, \delta, V_{th}$)

Ensure: : Update network parameters

Forward (inference):

for all $t = 1$ to T **do**
  $\mathbf{b}^n \leftarrow$ Generate($\log\alpha^n, \beta, \gamma, \varsigma$) //Eq. (3.17)
  $\mathbf{o}^{t,1} \leftarrow$ EncodingLayer(X^t)
  for all $l = 2$ to $N_1 - 1$ **do**
    $(\mathbf{u}^{t,n}, \mathbf{o}^{t,n}) \leftarrow$ StateUpdate($\mathbf{W}^{n-1}, \mathbf{b}^{b-1}, \mathbf{u}^{t-1,n}, \mathbf{o}^{t-1,n}, \mathbf{o}^{t,n-1}, \mathbf{x}^{t,n-1}$)//Eq. (3.21,3.22)
  end for
end for

Loss:

$L \leftarrow$ ComputeLoss($\mathbf{Y}, \mathbf{o}^{t,N_2}, \log\alpha$)//Eq. (3.9)

Backward:

Gradient Initialization: $\frac{\partial L}{\partial \mathbf{o}^{t+1,*}} = 0$
for all $t = T$ to 1 **do**
  $\frac{\partial L}{\partial \mathbf{o}^{t,N_2}} \leftarrow$ LossGradient($L, \frac{\partial L}{\partial \mathbf{o}^{t+1,N_2}}$)//Eq. (3.6,3.7,3.9)
  for all $l = N_2 - 1$ to 1 **do**
    $(\frac{\partial L}{\partial \mathbf{o}^{t,n}}, \frac{\partial L}{\partial \mathbf{u}^{t,n}}, \frac{\partial L}{\partial \mathbf{W}^n}, \frac{\partial L}{\partial \alpha^n}) \leftarrow$ BackwardGradient ($\frac{\partial L}{\partial \mathbf{o}^{t,n+1}}, \frac{\partial L}{\partial \mathbf{o}^{t+1,n}}, \mathbf{W}^n, \log\alpha^n$)//Eq. (3.6,3.7,3.9)
  end for
  for all $l = N_1$ to 2 **do**
    $(\frac{\partial L}{\partial \mathbf{o}^{t,n}}, \frac{\partial L}{\partial \mathbf{u}^{t,n}}, \frac{\partial L}{\partial \mathbf{W}^{n-1}}, \frac{\partial L}{\partial \alpha^{n-1}}) \leftarrow$ BackwardGradient ($\frac{\partial L}{\partial \mathbf{o}^{t,n+1}}, \frac{\partial L}{\partial \mathbf{o}^{t+1,n}}, \mathbf{W}^{n-1}, \log\alpha^{n-1}$)//Eq. (3.6,3.7,3.9)
  end for
end for

pare the efficiency and accuracy of our Sparse SCNN with SCNN, M-SNN [64], and stochastic SNN [65] on various classification tasks. To better compare our work with them, we follow the same experimental setting as in [58], including the same experimental datasets and the same network structure. Second, we want

Table 3.1: *Fixed parameter values for the various experiments.*

Parameter	Description	Chosen Value (MNIST/CIFAR10/N-MNIST/DVS-Gesture)
T	Time window	30ms,12ms,300ms,1450ms
k_τ	Decay factor	0.1ms,0.3ms,0.2ms,0.2ms
δ	Derivative approximation parameter	1.0,0.5,0.5,0.5
V_{th}	Threshold	0.5
β	Temperature of hard-concrete distribution	2/3
γ, ς	Other parameters of hard-concrete distribution	-0.1, 1.1
λ	The weight factor of sparse regularization	0.001

to explore the generalizability of our proposed model, especially for high dimensional data with very few training samples. We thus test on small training subsets of MNIST and N-MNIST. We validate our sparse deep SNN framework by using the state-of-the-art fully connected and convolutional architectures for deep SNNs [58] on these datasets. To combat randomness in the experiment system, we run all experiments 10 times and report the average results, except when otherwise stated.

Datasets

We evaluate our sparse SNN models and baselines on various datasets. Using the same datasets as in [58], we test on both static (non-spiking) as well as dynamic (neuromorphic) data.

Static Datasets: MNIST is a popular dataset comprised of a training set with $60,000$ samples and a testing set with $10,000$ samples of hand-written digits $0-9$.

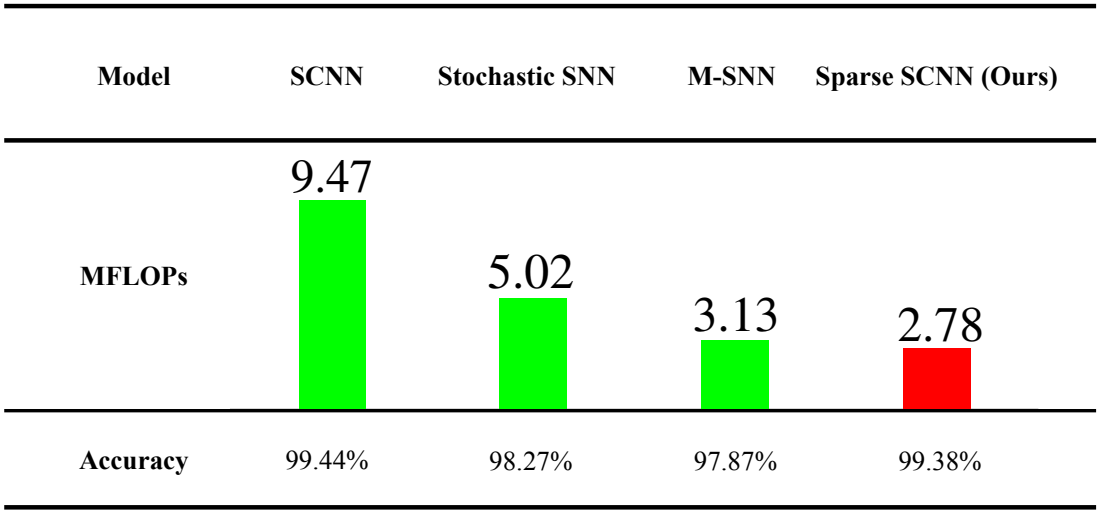


Figure 3.4: *Comparison with SCNN[57], M-SNN[64] and stochastic SNN[65] on MNIST.*

CIFAR-10 is an established computer-vision dataset used for object recognition. It consists of $60,000$ 32×32 color images containing one of 10 object classes, with $6,000$ images per class. Since our method and baselines are spike based learning algorithm, the static images should be converted to spike trains. To this end, we use the Bernoulli sampling conversion from original pixel intensity to the spike trains in this chapter. Each normalized pixel is converted to a spike event "1") or no spike event "0") at each time step by using an independent and identically distributed Bernoulli sampling. The probability of generating a spike event is proportional to the normalized value of the entry. Thus, given a certain time window T, the spike events form a spike train. During training, we set T to 12 and 30ms in MNIST and CIFAR-10, respectively.

Dynamic Datasets: Compared to the static datasets, dynamic datasets contain richer temporal features and are therefore more suitable for evaluating SNNs since SNNs can take advantage of the added information. We use the N-MNIST[1] and

[1]https://www.garrickorchard.com/datasets/n-mnist

DVS-Gesture[1] datasets to evaluate the capability of our method on dynamic datasets. The N-MNIST dataset [79] consists of MNIST images converted into a spiking dataset using a Dynamic Vision Sensor (DVS) moving on a pan-tilt unit. Each dataset sample is 300ms long, with a shape of 34×34 pixels, containing both "on" and "off" spikes. The dataset is split into training and test sets following the original split in MNIST of $60,000$ training samples and $10,000$ testing samples.

The DVS-Gesture dataset [80] contains $1,342$ instances of a set of 11 hand and arm gestures, grouped into 122 trials and collected from 29 subjects under 3 different lighting conditions. During each trial, one subject stood against a stationary background and performed all 11 gestures sequentially under the same lighting conditions. These gestures are recorded using a DVS128 camera, which is a 28×28-pixel Dynamic Vision Sensor. The problem is to identify the correct action label associated with each action sequence video.

Network structure

Throughout this chapter, we use the following notations to describe the deep SNN architecture. Layers are separated by "$-$" and spatial dimensions are separated by "$\times$". A convolution layer is represented by "C" and a pooling layer is represented by "P". For example, "$28 \times 28 - 15C5 - P2 - 10$" represents a 4-layer spiking CNN with 28×28 input, followed by 15 convolution filters that are (5×5), followed by 2×2 pooling layer and finally a dense layer connected to 10 output neurons. Table 3.2 provides the network structures for experiments. We use the exact same network architecture for our model and baselines for a fair comparison.

[1] https://ibm.ent.box.com/s/3hiq58ww1pbbjrinh367ykfdf60xsfm8

Table 3.2: *Network structures used for experiments.*

Static Dataset	
MNIST	28×28-15C5-P2-40C5-P2-300-10
CIFAR10	34×34×2-32C3-P2-64C3-P2-256-10
Dynamic Dataset	
N-MNIST	34×34×2-16C5-P2-32C3-P2-64C3-10
DVS-Gesture	128×128×2-P4-16C5-P2-32C3-P2-512-11

Initialization

In our proposed model, some parameters, such as the model weights and the locations of the hard-concrete distribution, need to be learned while others need to be fixed throughout the optimization. We now discuss our choice for initializing these parameters, which includes the weights, the thresholds and the decay factor for each neuron, the weighting factor for the sparse regularization, and the parameters of the hard-concrete distribution. We divide these parameters into two sets to consider.

First, to better mimic the neural dynamics, we need to control the relative magnitude between the weights and thresholds to avoid too much spiking, which reduces neuronal selectivity. In practice, and as a simplification, we fix the threshold value as a constant for each neuron and only adjust the weights that is responsible for controlling/balancing activity. We initialize all the weight parameters by sampling from the standard uniform distribution followed by normalization.

Second, while sparsifying the network, we follow [78] and set $\gamma = -0.1, \varsigma = 1.1, \beta = \frac{2}{3}$ for the concrete distributions. Meanwhile, we initialize the locations $\log\alpha$ by sampling from a normal distribution with a standard deviation of 0.01 and

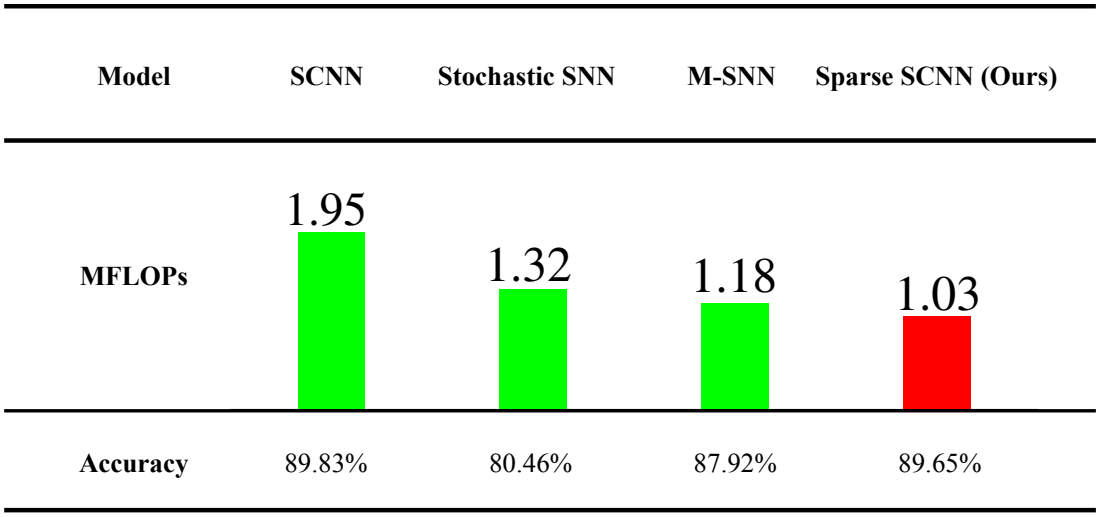


Figure 3.5: *Comparison with SCNN[57], M-SNN[64] and stochastic SNN[65] on CIFAR-10.*

a mean of 1. In practice, we use a single sample of the gate z for each mini-batch of the dataset during the training, even though this can lead to larger variance in the gradients. This way, we show that we can obtain the speedups in optimization with a practical implementation without incurring a significant loss in classification accuracy. A summary of the values of the fixed parameters used is shown in Table 3.1.

Evaluation metrics

To evaluate classification performance, we use the standard Accuracy metric. To evaluate the computational efficiency, we count the floating point operations (FLOPs) to measure the potential speedup. FLOPs are computed by assuming one flop for multiplication and one flop for addition.

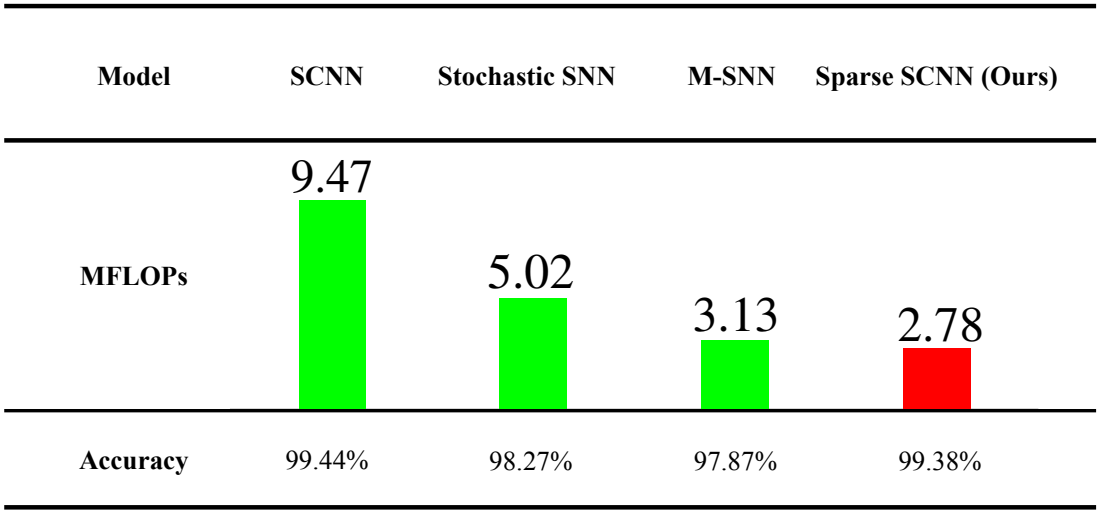


Figure 3.6: *Comparison with SCNN[57], M-SNN[64] and stochastic SNN[65] on N-MNIST.*

Experiment results

In this chapter, we discuss the experimental results pertaining to each of the two previously-raised research questions separately.

Potential speedup: The tables shown in Figures 3.6 and 3.5 compare our proposed sparse deep SNNs with a traditional SNN, M-SNN, and stochastic SNN on the static MNIST and CIFAR-10 datasets, respectively. Even without a complex architecture, the proposed deep SNNs and their competitors still perform well on these datasets. We find that there is only a slight difference in accuracy between our sparse deep SNNs and their competitors (*i.e.*, only between 0.05% and 0.1%). This is a negligible difference. However, as we can observe, there is a significant improvement in the FLOP count between our sparse deep SNNs and the competitors. On CIFAR-10, our sparse network and M-SNN incurs only half the computational cost compared to traditional SNN. On MNIST, this ratio is further reduced to less than 25%, which allows for a potentially significant speedup in inference

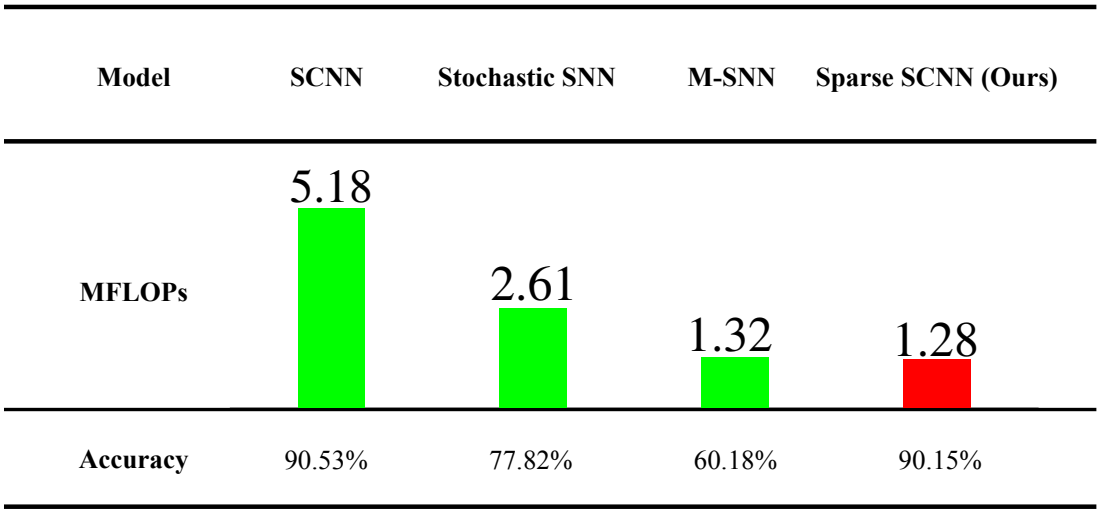


Figure 3.7: *Comparison with SCNN[57], M-SNN[64] and stochastic SNN[65] on DVS-Gesture.*

Table 3.3: *Comparison on small datasets. Best accuracy highlighted.*

Dataset	Method	Accuracy
MNIST	SCNN	69%
	Sparse SCNN (Ours)	**92%**
N-MNIST	SCNN	95%
	Sparse SCNN (Ours)	**97%**

phase.

The results for the neuromorphic datasets are shown in Figures 3.4 and 3.7. We find that both M-SNN and stochastic SNN, which perform well on static datasets, have a significant degradation in accuracy. This demonstrates that the works proposed in [64, 65] are not as suitable for neuromorphic datasets. Meanwhile, by using a sparse network structure, our proposed model incur a slight degradation in accuracy (*i.e.*, decrease between 0.05% and 0.4%) but sparsity can provide a significant speedup – nearly $5x$ times. Our experimental results show that sparsifying deep SNNs using our proposed framework can greatly speed up training and in-

ference while only incurring a minimal and negligible loss in classification performance. In summary, our proposed model achieves better computational efficiency than previous works when tested on both neuromorphic as well as static datasets and achieves very negligible degradation in accuracy.

Better generalization: To evaluate the ability of the model to generalize, we first compare the performance of our proposed method to SCNN on MNIST as the size of the training set is varied We continuously reduce the training set of MNIST and test on a test set of the same size.

As shown in Figure 3.8, although both methods achieve very competitive accuracy when the whole training set is used, our sparse deep SNN demonstrates much higher robustness when training set size is decreased. In particular, we observe that the performance of the non-sparse model drops sharply when the training set size is reduced to below $3,000$ while the sparse deep SNN's performance remains fairly steady. We conclude that when there are not enough training samples, deep SNNs will easily overfit and even memorize random patterns in the training set. This overfitting can lead to poor generalization. In contrast, by using sparse architecture in deep SNNs, the model shows better generalization even when training samples are limited.

We summarize the results of these experiments, run on MNIST and N-MNIST, in Table 3.3. During training, we limit the percentage of available training samples to only 1.67% (*i.e.*, only $1,000$ samples). As can be observed from the results, all the sparse deep SNNs demonstrate higher accuracy than that of their competitors. This demonstrates that by inducing model sparsity in the architecture, deep SNNs can achieve better generalization in practice.

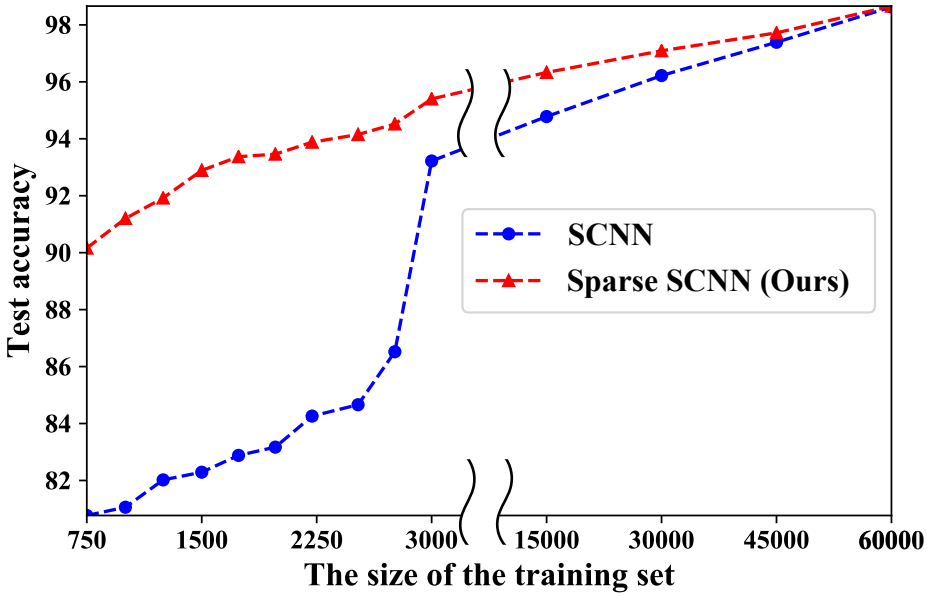


Figure 3.8: *Generalization test on MNIST dataset. The Y-axis denotes the accuracy of each model on the test set* ($10,000$ *samples). The X-axis denotes the size of the training set.*

Impact of the weight factor of sparse regularization

In this thesis, we achieve sparsity in the network structure of deep SNNs via sparsity regularization, which balances the accuracy with the percent of non-zero weights. Now we quantitatively analyze the impact of this sparsity regularization. We implement a sparse spiking CNN on MNIST, keeping the model configurations the same as our previous experiment on MNIST (28×28-15C5-P2-40C5-P2-300-10).

In Figure 3.9, the left side shows the level of sparsity at each layer when $\lambda = 0.001$. We use subgraphs of different widths to correspond to the number of weights of each layer in the network, while using the height of blue shaded area to correspond to the sparsity of each layer. The right side shows the overall level of sparsity under different values of λ. The X-axis denotes the value of the sparsity regularization term while the Y-axis denotes the percent of non-zero weights.

The results in Figure 3.9 show that sparse regularization has a different influ-

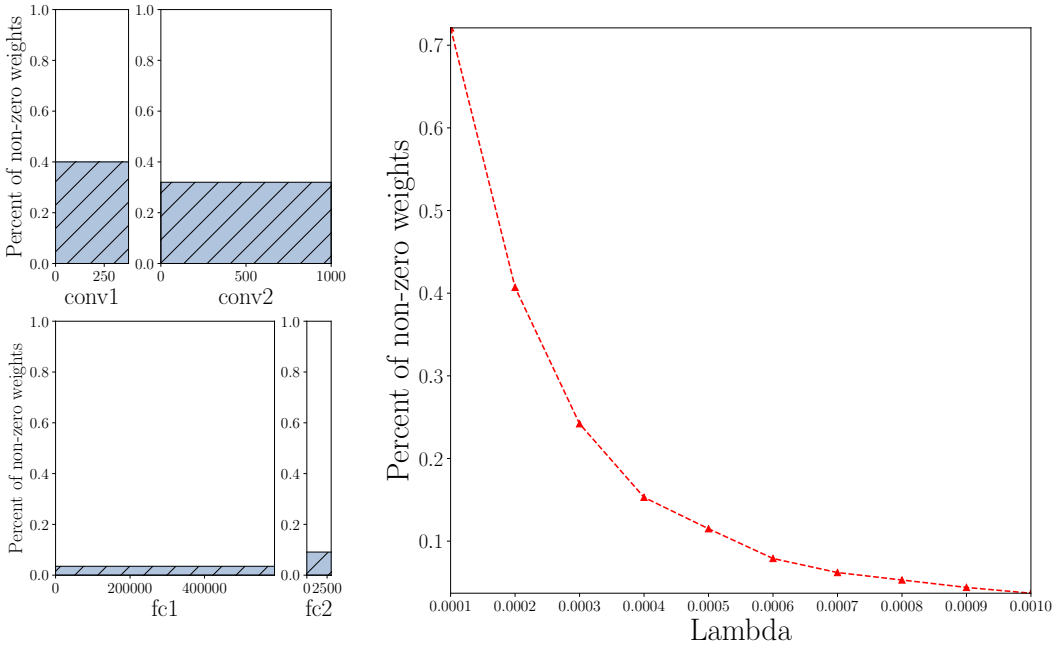


Figure 3.9: *(Left) Level of sparsity with fixed λ=0.001; (Right) Overall sparsity when λ is varied.*

ence on convolutional layers and fully connected layers. One reason why the the fully connected layers are sparser than the convolutional layers may be due to the difference in nature of the two types of layers. Convolutional layers apply the same set of weights repeatedly at different positions of the input. On the other hand, each weight in a fully connected layer will only be used once. The parameters in convolutional layers therefore learn general features at possibly multiple locations while each parameter in fully connected layers computes a single feature. As a result, the effect of sparse regularization is more significant in fully connected layers than in convolutional layers.

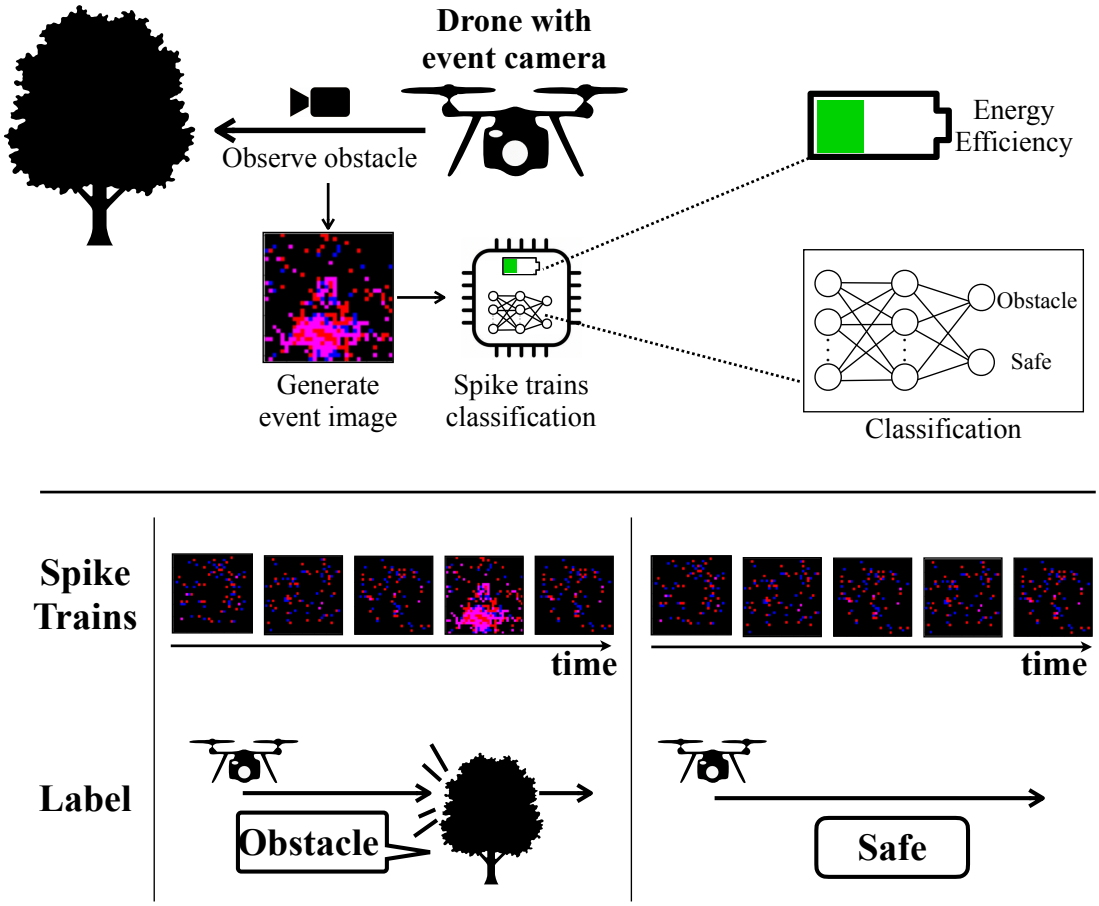


Figure 3.10: *The problem definition of efficient classification of spike trains. The spike trains are generated by an event camera, which is an imaging sensor that responds to local changes in brightness. Each pixel inside an event camera operates independently and asynchronously, reporting changes in brightness as they occur, and staying silent otherwise. Therefore, each image can be considered as binary event image.*

3.2 Task 5: Sparse Spike Trains Classification with Noise

3.2.1 Motivation.

Spike trains are sequences of binary signals where 1s are spikes and 0s are not spikes. Such data are common to a variety of domains and are classically analogous to electrochemical signals in the human brain. Spike train datasets are gen-

erated from event cameras, which resemble the human eye. Event cameras, also called neuromorphic cameras, require little energy and are designed to capture objects at high speed. Thus, spike train datasets naturally arise during the development of dynamic vision devices [55, 56, 81, 82, 83].

Recently, spike train classification has attracted much attention in the machine learning community [3, 57, 58, 60, 84, 85, 86]. Compared with the traditional sequence classification tasks, spike train classification is unique in the following two aspects [87, 88, 89, 90]: 1) *Temporal-sparsity* of signals of interest. The label of a spike train is only related with certain objects that may only appear in a very small portion of the whole time window. 2) *Temporal-noise* problem. The signals at the majority of time steps are generated from background activities that are not related to objects of interest. These two properties of spike train classification impede the application of the widely used deep learning models, e.g. recurrent neural networks (RNNs), because of their high computational costs, unnecessarily spent on the whole time window. Tasks on spike train data are instead usually processed on energy-sensitive platforms such as wireless monitors and drones, so more computationally efficient models are required. To this end, we propose a new *design principle* to guide developement of machine learning models for spike train classification: *perform intensive computation only when signals of interest appear*.

Spiking Neural Networks (SNNs) are potential candidates for spike train classification [57, 58, 60], since they are attempt to meet the aforementioned design principle by considering *the temporal-sparsity* of spike trains. They take spike trains as inputs and outputs, using biologically inspired, event-driven computation and communication in their design. An SNN neuron's core function is to react only

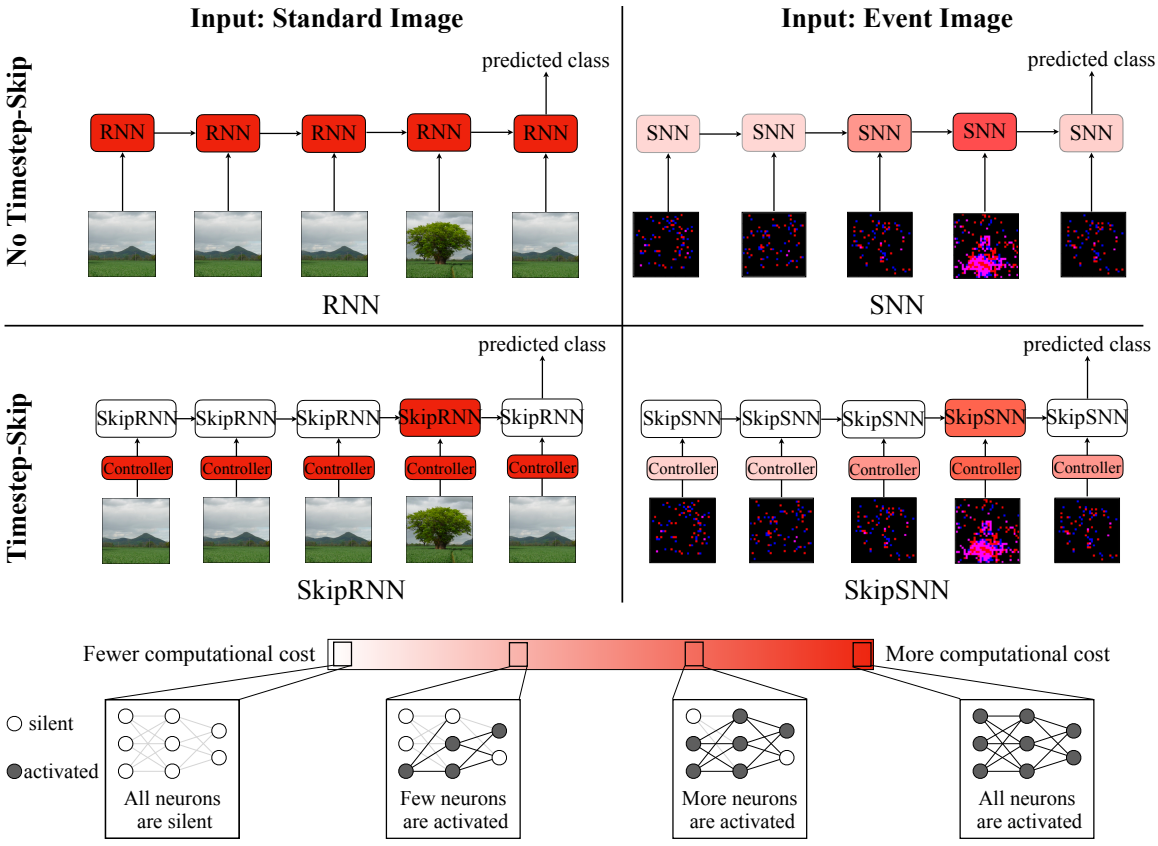


Figure 3.11: *Differences among Recurrent Neural Network (RNN) , SkipRNN [91], Spiking Neural Network (SNN) [57, 60], SkipRNN and SkipSNN (our model). Compared with other models, our model can achieve better computational efficiency and higher classification accuracy based on an event-attention mechanism of filtering noise.*

when its cummulative membrane potential exceeds a fixed value. As a result, the neuron has a chance to be activated only when it currently has an event signal, which is passed in as a binary spike. Thus, compared to traditional deep learning models, SNNs can build large-scale neural networks with far less energy and memory for spike train classification.

However, SNNs only consider the *temporal-sparsity* of spike trains, not the *temporal-noise* issue. As illustrated in Figure 3.10, suppose a drone uses an event camera to detect obstacles and then a model decides if the drone needs to change route. For

the most of the timeline, the camera collects signals that are irrelevant to the task. But SNNs will react to any signals, even if they are only noise. As a result, SNNs often can't meet the principle that the model should only work when the signals of interest appears, and we argue that is a primary reason which may lead to the poor generalization ability and lower the energy efficiency of SNNs in the real world applications.

To achieve high classification accuracy with low computational cost for real-world spike trains, we need to follow the aforementioned design principle by considering both temporal-sparsity of useful signals and temporal-noise issue. An intuitive approach, which we pioneer in this study, is for the model to stop processing data when the relevant object is out of its field of view. This behavior is analogous to how we open and close our eyes to filter out the information we see.

We propose a novel method for allowing SNNs to efficiently classify spike trains. Solving this problem is challenging for two main reasons:

- *Neuron Consistency*: The promise of SNNs comes from their likeness to real neural circuits in the human brain. Maintaining this similarity is essential to successful SNNs. However, in the standard SNN when a neuron enters a hibernation state, it is hard to wake it up again if there is no new signal input to the network. This means that if the model ignores the input at a timestep, the neurons in the network will lack new input signals. This keeps the neurons silent, making the model likely to ignore potentially useful signals in the future. Thus, when extending SNNs to our problem setting, it is challenging to train successful spiking neurons to skip updates.

- *Non-differentiability*: SNNs are notoriously difficult to train due to non-differentiable

nature of spike activity. Most related works use the rectangular function or sigmoid to approximate the corresponding derivative. However, in practice we find that when our optimization objective considers both accuracy and efficiency, this approximation leads to decayed performance. Additionally, optimization largely depends on the initial values of the parameters of the model. Even though some parameter initialization methods such as Glorot [92] work for traditional artificial neural networks, they lack theoretical basis in SNNs. Designing an efficient optimization algorithm is the second challenge we face.

To solve our problem, we introduce an event attention mechanism that enables SNNs to dynamically highlight useful signals in the input spike train. We extend existing SNNs to have two different states: *awake* and *hibernating*, inspired by how people's eyes open and close, turning on and off data intake. If our SNN enters its awake state at time step t, it will consider the input at t. Otherwise, if it hibernates at time step t, it will ignore the input at t. To this end, we design a controller that switches the model between these two states. Since this is not differentiable, we also introduce a new loss function with a penalty that trades off accuracy and computational cost. In this way, our extended SNN learns to mask out noise by skipping updates and shorten the effective size of the computational graph without requiring any additional supervision signal. We refer to our model as SkipSNN and illustrate the difference between it and traditional SNN in Figure 3.11.

Our contributions are summarized as follows:

- We define the problem and modeling principle of general spike train classification, which is important for smart dynamic sensor systems with limited

energy.

- We propose SkipSNN, which solves this problem and can be used on energy-limited dynamic sensor devices.
- We develop an efficient optimization technique to train our SkipSNN model.
- We demonstrate that our model outperforms recent state-of-the-art alternatives by achieving higher accuracy and lower computational cost when tested on both the neuromorphic MNIST and DVS-Gesture datasets.

The rest of this chapter is organized as follows. First, we review related work, then introduce details of the background methods. Next, in Chapter 3.2.2, we present our proposed method. We then describe our experimental setup and discuss our results in Chapter 3.2.3. Finally, we conclude the chapter of the thesis with key take-aways and give some directions for future work.

3.2.2 Related Works

In spike train classification, methods for training SNNs can be divided into two categories: “Indirect" learning and “Direct" learning. Indirect learning mainly focuses on ANN-to-SNN conversion [68, 69, 70, 71]. These methods are indirect in that a regular non-spiking Artificial Neural Network (ANN), such as a multi-layer perceptron, is initially used during the training phase. At inference-time, the trained model is then converted to an SNN. However, such indirect training doesn’t align well with how an SNN operates. For example, in ANNs, it does not matter if activations are negative, while firing rates in SNNs are always positive. As for Direct learning, many methods have recently been proposed [3, 58, 60].

These approaches train SNNs directly using back-propagation in both the spatial and temporal domains. [58, 60] have achieved state-of-the-art accuracy on the MNIST and N-MNIST datasets. [3] is trains SNNs with a spatial sparsification technique that allows SNNs to perform inference with lower computational cost. While these methods perform better than the indirect methods on many neuromorphic datasets, they are still not suitable for efficient classification of real-world spike trains, especially in terms of temporal-noise problem.

Although there is no model for dealing with our defined problem for SNNs, some methods have been proposed for dealing with similar problems for RNNs [91, 93, 94, 95, 96, 97, 98, 99]. For example, SkipRNN [91] extends classic RNN models by learning an additional controller network that learns to skip state updates. The input of this neuron is the state value of other neurons. So it can generate a binary value based on the sigmoid function value of its state. This model can significantly reduce the computational cost. However, while both RNNs and SNNs are used to deal with sequence analysis, they remain completely different. The main difference is that neurons in RNNs are mostly non-linear, continuous function approximators that operate on a common clock cycle, whereas the neurons in SNNs use asynchronous spikes that signal the occurrence of some characteristic event and temporally precise membrane potentials. When dealing with spiking training tasks, many neurons of an SNN may stay silent based on its mechanism, whereas all neurons of an RNN will be activated. Consequently, in terms of inference efficiency, RNNs are outmatched by SNNs for spike train classification. We illustrate this key difference in Figure 3.11.

3.2.3 Skip Spiking Neural Network

Model Definition

We propose a novel modification for existing SNN architectures that allow them to mask noise and skip membrane potential updates without requiring any additional supervision signal. To build a new architecture, which we call SkipSNN, we view SNN neurons as having two different states: awake and hibernating. Then, we design a new neuron to control the model to switch freely between these two states.

Inspired by [91], we augment the network with a binary gate, but use a novel neuron to control it. This neuron doesn't have any special settings, but follows the basic mechanism of SNNs, exactly like other neurons in the network. In the rest of this study, in order to distinguish it from other SNN neurons, we call this neuron the *controller*.

Let v_t denote the membrane potential of the controller at time step t. This controller is connected with all the neurons in the first layer. It is therefore affected by the outputs of the first layer and will generate a binary output a_t according to the SNN mechanism described in Equations 3.1 and 3.2. Then, this value decides whether to consider the input of the next time step by treating a_t as a multiplier of the next time step. If it emits a spike, this means that the network enters the awake state at next time step. Otherwise, the network hibernates at next time step. Based on the SNN mechanism, the controller will reset its membrane potential to zero after emit a spike. Therefore, when it decides to enter the awake state at the next time step, it is hard for the controller to spike again due to the lack of new spike

inputs from the first layer.

In order not to miss the potential useful signals in the future, a temporal form of bias is needed to adjust the membrane potential of the controller. In this chapter, we introduce synchronisation pulses that act as additional inputs to the controller, in order to provide the information of time. These can be thought of as similar to internally-generated rhythmic activity in biological networks, such as alpha waves in the visual cortex [100] or theta and gamma waves in the hippocampus [101].

In our proposed SkipSNN, a set of pulses are fully connected to the controller in the network. Each pulse spikes at a different frequency. For example, some spike once every time step, some spike once every 10 time steps, and some spike once every 100 time steps. Subsequently, these pulses can modify the membrane potential of the controller when it stays hibernating state, thereby being activated again. At every time step t, affected by the output from the neurons in the first hidden layer and pulses, the controller emits a binary signal, which is multiplied by the model input at $t+1$. The resulting architecture—depicted in Figure 3.12—can thus be described as follows:

$$\mathbf{u}_t^{(2)} = \tau \mathbf{u}_{t-1}^{(2)} \odot (1 - \mathbf{z}_{t-1}^{(2)}) + a_{t-1}\mathbf{W}^{(1)}\mathbf{x}_t, \tag{3.26}$$

$$v_t = \tau v_{t-1}(1 - a_{t-1}) + \mathbf{W}_{\mathrm{z}}\mathbf{z}_t^{(2)} + \mathbf{W}_{\mathrm{o}}\mathbf{o}_t, \tag{3.27}$$

$$a_t = \Theta(v_t - V_{\mathrm{th}}), \tag{3.28}$$

where $\mathbf{W}_{\mathrm{z}}$ and $\mathbf{W}_{\mathrm{o}}$ are the weights vectors between the controller and neurons in the first hidden layer, and the controller and pulses, respectively. $\mathbf{o}_t = (o_{1,t}, ..., o_{p,t})$ is the pulse vector that contains the signals from p different pulses at t. v_t and a_t

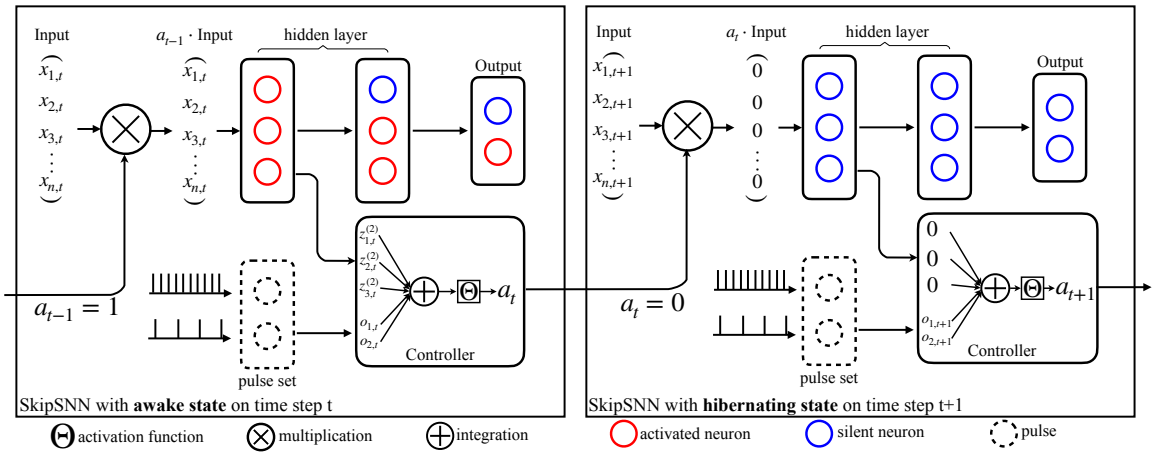


Figure 3.12: *Model architecture of the proposed SkipSNN.*

are the membrane potential and the output of the controller at t, respectively.

According to the model formulation from Equations 3.26-3.28, SkipSNN can switch between awake and hibernating state based on the value of a_t. If $a_t = 1$, SkipSNN will be updated based on the input $\mathbf{x}_{t+1}$ at $t+1$, otherwise, the proposed model will skip $\mathbf{x}_{t+1}$ due to $a_t \mathbf{x}_{t+1} = 0$.

In particular, the controller can be connected to any layer of SNN. Meanwhile, the controller itself can also be a multilayer SNN. However, in practice, we found that the controller works better when connected to the first hidden layer. In addition, using a multi-layer network structure as the controller does not improve the performance but will increase the computational cost. Thereby, we keep using the network structure depicted in Figure 3.12.

Limiting Computation

The proposed SkipSNN is able to learn when to enter awake or hibernating state without requiring any additional supervision signal. The longer the model stays

hibernating state, the lower computational cost required during inference phase. However, there is a trade-off between classification accuracy and computational power. If the model sacrifices some important time steps that contain useful information, it can reduce the computational cost, but it will also sacrifice the accuracy of the model. To balance between accuracy and efficiency, we add an additional penalty term

$$L_{\text{penalty}} = \lambda \frac{\sum_{t=1}^{T} a_t}{T}, \tag{3.29}$$

where L_{penalty} is the cost associated to one single sample, λ is the cost per sample, and T is the spike train length.

Optimizing SkipSNN

SNN is hard to optimize because the derivative of its activation function is a δ function, whose value is zero everywhere except at threshold. As is done in prior works [57, 58], we could use the rectangular function to approximate the corresponding derivative. However, the switch between awake and hibernating state is directly determined by the output of the controller. When the corresponding gradient of its membrane potential is zero, we will not be able to continue to optimize the relevant parameters of the controller through gradient descent. Therefore, we propose simulated annealing for SkipSNN; the training process for SkipSNN is divided into two stages.

In the first stage, we set $\lambda = 0$, which will cause the model to stay in the awake state at all times. This means that the controller will always emit $a_t = 1$ during

Algorithm 5 Algorithm for `SkipSNN`

Require: i: Network inputs $\{X^t\}_t^T$;
ii: class label Y;
iii: parameters and states of main network $(\{\mathbf{W}^{(\ell)}, \mathbf{u}_0^{(\ell)}, \mathbf{z}_0^{(\ell)}\}_{\ell=1}^{N_1-1})$;
iv: parameters and states of controller $(\mathbf{W}_z, \mathbf{W}_o, \mathbf{v}_t, \mathbf{a}_0)$;
v: the parameters of iterative LIF and penalty $(T, \epsilon, \Delta, V_{th}, \lambda)$;
iv: iter$_{max}$: the maximum number of iteration
Ensure: : Update network parameters

1: **Stage 1**:
1: Set the approximation of activation with Eq.3.30
2: Set $\lambda = 0$
3: **repeat**
4: Update the parameters of main network $(\{\mathbf{W}^{(\ell)}\}_{\ell=1}^{N_1-1})$
5: **until** $iter = iter_{max}$ or convergence

6: **Stage 2**:
1: freeze $(\{\mathbf{W}^{(\ell)}\}_{\ell=1}^{N_1-1})$
2: Set the approximation of activation with Eq.3.31
3: Set $\lambda > 0$
4: **repeat**
5: Update the parameters of controller $(\mathbf{W}_z, \mathbf{W}_o)$
6: **until** $iter = iter_{max}$ or convergence

this stag, because its membrane potential v_t is always larger than the its threshold $V_{]\text{th}}$. Therefore, our goal at this stage is to make SkipSNN solve the classification task as accurately as possible. In this stage, the derivative of each activation is approximated by the rectangular function denoted by $h(u)$:

$$h(u) = \frac{1}{a}\text{sign}(|u - V_{\text{th}}| < \frac{a}{2}), \tag{3.30}$$

where a determines the peak width.

In the second stage, we freeze parameters that are not related to the controller, and start optimizing the controller by elevating the multiplier λ of the penalty loss.

Because v_t at that point is always larger than V_{th}, we deploy a sigmoid function instead of rectangular function to approximate its derivative:

$$h(u) = \frac{1}{1 + e^{\frac{1}{T}(u - V_{\text{th}})}}. \tag{3.31}$$

The parameter T in Equation 3.31 controls the steepness of the sigmoid function, which is considered to be the reciprocal pseudo-temperature. As $T \to 0$, the sigmoid becomes a step function and the stochastic unit becomes deterministic. As T increases, this sharp threshold is "softened", thus making the range of the sigmoid wider. Therefore in this stage we initialize T with a high value to make sure the controller has a gradient when the membrane potential is high. Then, we decrease T periodically during the optimization process of the parameters related to the controller.

3.2.4 Experimental Results

To comprehensively validate the effectiveness of our proposed method, we conduct experiments to answer two research questions: First, we are interested in accuracy improvement on classification problem of spike trains; Second, we want to demonstrate that our model can significantly reduce computational cost with very negligible degradation in accuracy. As ours is the first SNN proposed to deal with our defined problem, we compare our model with Fixed-skip SNN and Random-skip SNN. We also use a modified SNN converted from SkipRNN [91] as a baseline, because SkipRNN is a cutting-edge model for skipping frames on time series, which is similar to our defined problem. To better compare our work with them,

we test on various neuromorphic datasets, including N-MNIST and DVS-Gesture. Both are widely used to evaluate SNN models in related work. To combat randomness in the experiment system, we run all experiments 10 times and report the average results, except when otherwise stated.

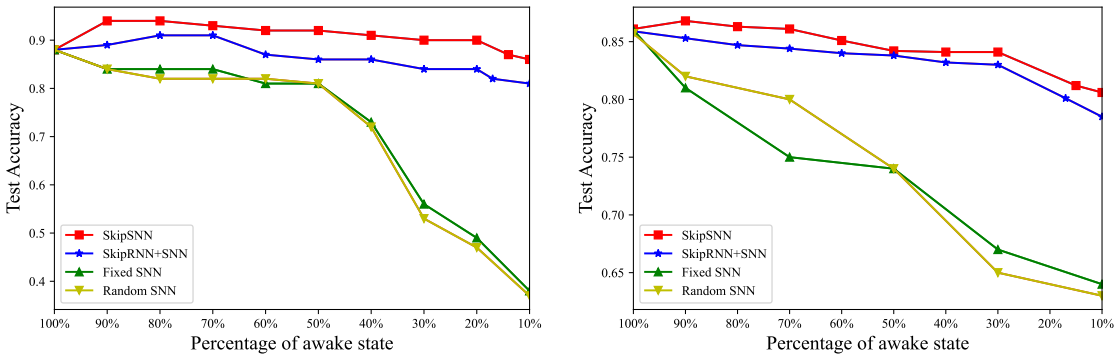
Datasets

We evaluate our proposed model and baselines on various datasets. To simulate the properties of general spike trains in the real world, we modified these neuromorphic datasets (N-MNIST[1] and DVS-Gesture[2]) by making signals of interest temporal-sparse and adding noise into them.

N-MNIST The N-MNIST dataset [79] consists of MNIST images converted into a spiking dataset using a Dynamic Vision Sensor (DVS) moving on a pan-tilt unit. In our experiments, each dataset sample is 50 ms long, with a shape of 34×34 pixels, containing two channels to preserve "on" and "off" spikes, respectively. This dataset is harder than MNIST because one has to deal with saccadic motion. For SkipSNN experiments, we generate a 300 ms blank time sequence firstly, meanwhile increasing the noise by adding one signal to a random pixel of the image in each millisecond (one time step). Then we put each N-MNIST sample into a random period of each time sequence. So in each final sequence, only 16.7% of the time steps have useful signals related to original N-MNIST dataset. The dataset is split into $60,000$ training samples and $10,000$ testing samples.

DVS-Gesture The DVS-Gesture dataset [80] contains $1,342$ instances of a set of 11 hand and arm gestures, grouped into 122 trials and collected from 29 subjects

[1] https://www.garrickorchard.com/datasets/n-mnist
[2] https://ibm.ent.box.com/s/3hiq58ww1pbbjrinh367ykfdf60xsfm8



(a) *Accuracy on N-MNIST* **(b)** *Accuracy on DVS-Gesture*

Figure 3.13: *Observing the performance of different models with different percentage of updated time-steps.*

Table 3.4: *Comparative results on N-MNIST dataset.*

Model	Percentage of awake state	Accuracy	Inference MFLOPs
SNN	100% ± 0.0%	88.92% ± 0.76%	1.15
Fixed-skip SNN	13.33% ± 0.0%	38.43% ± 4.13%	0.51
Random-skip SNN	10.00% ± 0.0%	37.16% ± 3.41%	0.47
SkipRNN+SNN, $\lambda = 10^{-3}$	91.72% ± 0.37%	89.86% ± 0.16%	0.71
SkipRNN+SNN, $\lambda = 10^{-1}$	11.13% ± 1.24%	81.24% ± 1.02%	0.52
SkipSNN, $\lambda = 10^{-3}$	92.15% ± 0.08%	94.47% ± 0.12%	0.75
SkipSNN, $\lambda = 10^{-1}$	11.03% ± 0.58%	86.65% ± 0.27%	0.46

under 3 different lighting conditions. During each trial, one subject stood against a stationary background and performed all 11 gestures sequentially under the same lighting conditions. The problem is to identify the correct action label associated with each action sequence video. In our experiments, each DVS-Gesture sample is 400 ms long, and 32×32 pixels big, containing two channels to preserve "on" and "off" spikes. To allow us to test SkipSNN on spike trains, we modify DVS-Gesture dataset by using the method applied to modify N-MNIST. So each spike train in the final dataset has 1000 time steps (ms), and only 400 consecutive time steps in each spike train have useful signals related to original DVS-Gesture dataset.

Table 3.5: *Comparative results on DVS-Gesture dataset.*

Model	Percentage of awake state	Accuracy	Inference MFLOPs
SNN	$100\% \pm 0.0\%$	$86.12\% \pm 0.25\%$	1232.5
Fixed-skip SNN	$13.33\% \pm 0.0\%$	$64.13\% \pm 2.05\%$	147.4
Random-skip SNN	$10.00\% \pm 0.0\%$	$63.41\% \pm 2.37\%$	112.5
SkipRNN+SNN, $\lambda = 10^{-5}$	$92.41\% \pm 0.42\%$	$85.32\% \pm 0.43\%$	1013.1
SkipRNN+SNN, $\lambda = 10^{-4}$	$10.02\% \pm 3.35\%$	$78.50\% \pm 0.43\%$	117.3
SkipSNN, $\lambda = 10^{-6}$	$89.63\% \pm 0.23\%$	$86.82\% \pm 0.13\%$	973.7
SkipSNN, $\lambda = 10^{-4}$	$9.04\% \pm 1.44\%$	$80.24\% \pm 0.18\%$	72.6

Compared Methods

To demonstrate the effectiveness of SkipSNN, we test against several baselines:

- *Fixed-skip SNN*:Control the percentage of awake state according to the fixed time-step size. For example, 90% means skip 1 time step after every 9 time step updates; 50% means skip one time step after every one time step.

- *Random-skip SNN*:Determine awake and hibernating state through Bernoulli sampling. We control the percentage of awake states by changing the parameter of Bernoulli distribution.

- *SkipRNN + SNN*: SkipRNN [91] is proposed to extend existing RNN models by learning to skip state updates and reduce the computational cost. The input and output of SkipRNN are not spike trains. But due to the similarity of the defined problem, we convert SkipRNN to an SNN version as a competitor.

To better compare our work with them, we use the same network structure and optimization algorithm on each model.

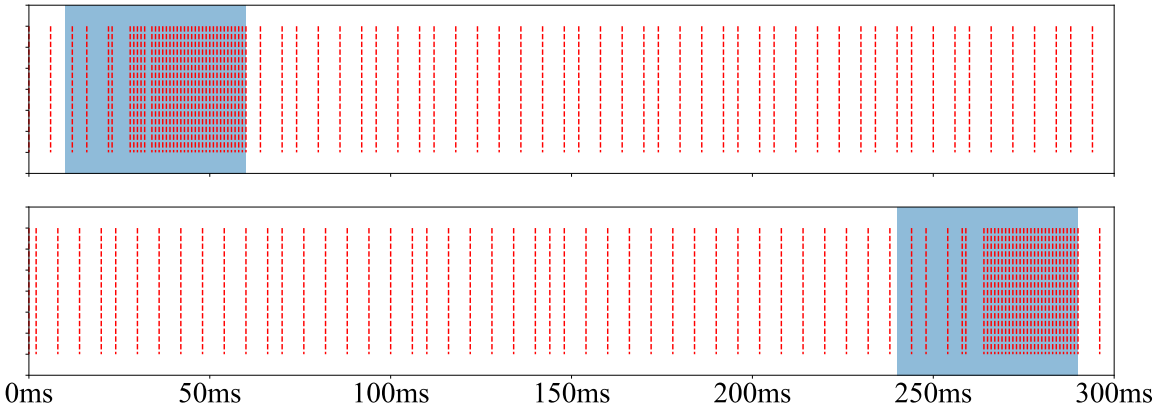
Evaluation metrics

To evaluate classification performance, we use the standard Accuracy metric. To evaluate the computational efficiency, we use the million floating point operations (MFLOPs) to measure the potential speedup. MFLOPs are computed by assuming one flop for multiplication and one flop for addition. In our experiments, we use MFLOPs of each model for one same sample in the inference phase as the evaluation metric.

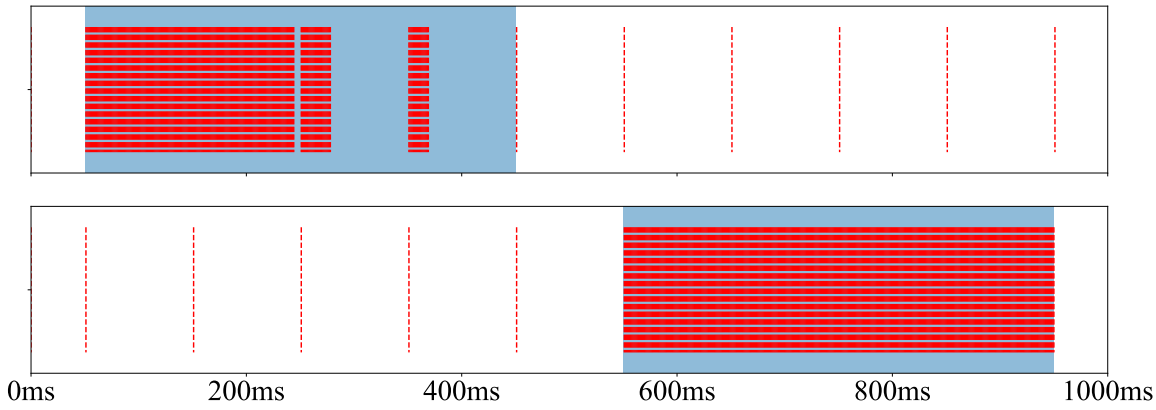
Experiment results

In this chapter, we discuss the experimental results pertaining to each of the two previously-raised research questions separately.

Better accuracy The results shown in Figure 3.13 compare the performance of each model with different percentage of updated time-steps on two different datasets. We control the percentage of awake state of SkipRNN+SNN and SkipSNN by changing the multiplier λ of the time-budget penalty. For random-skip SNN, we control it through the parameter of Bernoulli distribution. For fixed-skip SNN, we control it according to different time-step sizes. According to the results, we can observe that as the percentage of awake state decreases, fixed-skip and random-skip SNN drop rapidly. By contrast, SkipRNN+SNN and SkipSNN show their advantages in this scenario. Especially, when each model only considers less than 20% of awake state, SkipRNN+SNN and SkipSNN still maintain an accuracy of more than 80%. In contrast, the other two models are no longer valid. According to the comparison of SkipSNN+SNN and SkipRNN, our proposed model has a better performance of classification than the other one on the testing set of both



(a) *Examples of N-MNIST*

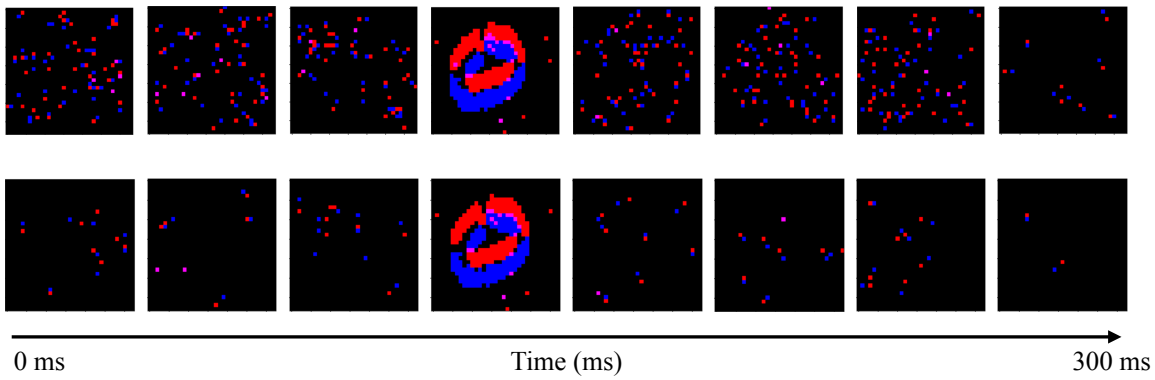


(b) *Examples of DVS-Gesture*

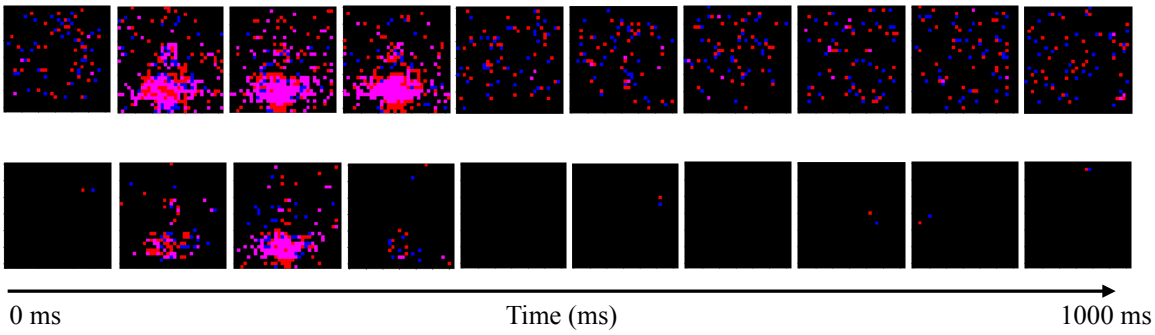
Figure 3.14: *Temporal raster plot of spikes in the controller neuron during the inference of the examples in N-MNIST and DVS-Gesture by SkipSNN. The spikes shown are generated from the controller, and used to control when to enter awake state. The blue box represents the period of useful signals. The dash red line represents the location of awake state.*

two datasets. This robustness in performance directly leads to a significant drop of the computational cost during inference phase. Moreover, the accuracy of SkipSNN with the percentage between 90% and 70% is even higher than that without skipping time steps. It means that compared with traditional SNNs, our proposed model is more accurate in dealing with the classification of spike trains.

Examples like those shown in Figure 3.14 show how SkipSNN learns to skip time steps that are mainly noise or not helpful for classification problem. Based on



(a) *Original and reconstructed input of N-MNIST*



(b) *Original and reconstructed input of DVS-Gesture*

Figure 3.15: *Visualization of original and reconstructed input of N-MNIST and DVS-Gesture. The inputs are, in row order, original and reconstructed. Reconstructed input means that we mask the signal in original input according to the time step locations that the model decides to skip.*

the dominance of the controller in SkipSNN over time-step skipping, we display the decision of SkipSNN on each time step by using a temporal raster plot of spikes in the controller during the inference of the examples in two datasets. The blue area in each plot indicates the period of useful signals. Meanwhile, the remaining white areas are mainly noise. According to Figure 3.14, we can observe that the spikes are mainly concentrated in the blue areas and sparse in the white areas. Therefore, it demonstrates that SkipSNN knows how to distinguish between signals of interest and noise and when to switch between awake and hibernating state.

Finally, the reason for the success of our proposed model is that SkipSNN filters out noise with the presence of controller architecture, reconstructing input that is much purer than the original input. We demonstrate it in Figure 3.15. We show the gif picture of original data and that of reconstructed data, which mask out most of the noise through SkipSNN.

Potential speedup: The tables shown in Table 3.4 and 3.5 compare SkipSNN with a traditional SNN, fixed-skip SNN, random-skip SNN, and SkipRNN+SNN on two neuromorphic datasets. For N-MNIST dataset, even without a complex architecture, the proposed SkipSNN and their competitors still perform well. We find that there is only a slight difference in accuracy between SkipSNN and SNN (*i.e.*, only about 2%). This is a negligible difference. However, as we can observe, there is a significant improvement in the MFLOP count between SkipSNN and SNN. Compared to SNN, SkipSNN can provide only half the computational cost of SNN. Meanwhile, when the computational cost of all models is similar, SkipSNN achieves the higher accuracy than fixed-skip SNN, random-skip SNN and SkipRNN+SNN.

On DVS-Gesture, our proposed model SkipSNN incurs a slight degradation in accuracy (*i.e.*, decrease about 6%), but provides a significant speedup – more than $10x$ times. The accuracy of SkipSNN at this level of MFLOP is still higher than other competitors. Our experimental results show that SkipSNN using our proposed network structure can greatly speed up inference while only incurring a minimal and negligible loss in classification performance. In summary, our proposed model achieves better computational efficiency than previous works when tested on neuromorphic datasets and achieves very negligible degradation in ac-

curacy. Moreover, according to Table 3.4 and 3.5, our model can also improve the accuracy performance when we consider the trade-off between accuracy and computational cost. When SkipSNN considers about 90% of updated time-step, it can slightly increase accuracy (*i.e.*, increase about 6% on N-MNIST and 0.7% on DVS-Gesture). This observation is consistent with the conclusion of the first research question.

4

Conclusion and Future Works

Sparse learning is a very active field of research which tackles a multitude of problems, from a variety of domains, including general supervised learning, object recognition, image processing, and graph building for large scale semi-supervised learning. Due to the advantages of sparse machine learning models, we have witnessed the meteoric rise in popularity of sparse learning in recent years. The goal of this dissertation is to develop methods that exploit sparsity to help recover the underlying information in many real-world applications. In particular, our works involve two different domain: sparse graph discovery of brain network, and spike trains classification on resource-constrained devices.

4.1 Conclusion

4.1.1 Brain Network Discovery

We first explored sparse graph representation techniques and proposed new models for different network discovery problems.

In Task 1, we posed the issue of brain network discovery involving the group clustering with cohesive activities and the edge detection between those groups. We discussed a series of methods that can be applied into the problem of brain network discovery, however, to our knowledge no method can solve group clustering and edge detection simultaneously as we expect. We compared two mainstream models, including graphic lasso and non-negative matrix factorization, and then proposed a new method which is a combination model them. We developed multiplicative update rules on our proposed model. Then through extensive controlled experiments, we demonstrated that our proposed model shows more effectiveness and robustness than comparison methods, even surpassing the model of pervious work in the synthetic cases without the requirement of spatial continuity. In the real fMRI brain scanning datasets from AHDH subjects, our proposed model also shows a good interpretation of results due to the structural symmetry and spatial continuity. These results are consistent with known earlier work which focuses on these requirements. Therefore, we believe that our method can also be applied in other domains when network structure is very complex, group discovery and connectivity analysis are both needed for researchers to solve.

In Task 2, we defined the problem of mixture connectivity substructures between nodes in brain network discovery. To address this problem, we proposed

embedding one of the current methods of estimating multiple Gaussian graphical models in the framework of Gaussian mixture modeling, then designed a new regularization term, called mutual exclusivity regularization, to make sub-graphs un-overlapped with each other. Through extensive controlled experiments, we demonstrated that our proposed model MGL shows more effectiveness than other baseline models, meanwhile, MGL shows more robustness than JGL, especially in the consideration of small samples or noisy data sets. In addition, this conclusion is also demonstrated in the experiment of real fMRI brain scanning datasets from ADHD subjects. So we have reason to believe that, our method can also be applied in other domains when network connectivity structure is very complex.

In Task 3, we defined the open problem of multi-state brain network discovery, which is to infer various brain parcellations and connectivities across different brain states. Previous works on brain network discovery derive an average brain network based on the assumption that only one single activity state of the brain generates the signals. However, according to recent studies in the area of brain network, assuming single-state networks ignores a crucial of cognitive brain networks. To better understand the temporally-changing functional network of the brain, we proposed a novel model called MNGL, which can discover *multiple* brain networks, including nodes and their connectivity based on on only unlabeled fMRI scans. Through controlled experiments, we demonstrated that our proposed model shows more effectiveness and robustness than other baseline models. MNGL also shows expected and meaningful results on the real ADHD-200 fMRI dataset. We thus have reason to believe that our method can be applied in multi-state brain network for a better understanding of brain function and behavioral

performance.

4.1.2 Spike Trains Classification

Next, we studied machine learning solutions for spike trains classification problem. In Task 4, we tried to design a novel algorithm for high-dimensional spike train classification to be performed on energy-limited smart devices. To this end, we proposed a novel sparse architecture for deep SNNs. Our sparsification is achieved by reparametrizing original weights among neurons in the network and then employing a sparsity regularization during optimization. In addition, we also proposed an algorithm that can directly train sparse deep SNNs via back-propagation. In empirical study, we choosed SCNN as the basic model and apply our proposed sparsification procedure on it. To validate the effectiveness of our proposed method, we compared our model with SCNN, M-SNN and stochastic SNN. Our experimental results on both non-spiking (MNIST and CIFAR-10) and neuromorphic datasets (N-MNIST and DVS-Gesture) show that we can achieve significant speedup with little or no loss in classification accuracy. Furthermore, compared with densely-connected SNNs, we also show through extensive experiments that sparsification can result in better generalizability of the trained model on small-size datasets.

To better deal with spike train classification, we need a model that follows the design principle of performing intensive computation only when signals of interest appear. However, current SNNs ignore the temporal-noise issue of spike trains, which makes them computationally expensive and thus high power consumption for spike trains classification. So in the last task, to overcome the limitation of tra-

ditional SNNs on spike train tasks, we introduced an event attention mechanism that enables SNNs to dynamically highlight useful signals of the original spike trains. To this end, we proposed a novel model called SkipSNN. SkipSNN can learn to mask out noise by skipping membrane potential updates, thereby decreasing computational cost. In our empirical study, we compared SkipSNN against fixed-skip SNNs and random-skip SNNs and an SNN version of SkipRNN. Using two key neuromorphic datasets, N-MNIST and DVS-Gesture, we found that SkipSNN achieves significantly better computational efficiency and classification accuracy.

4.2 Future Works

4.2.1 Brain Network Discovery

A common problem of the first three tasks is how to simplify the data to discover a single or mixture underlying network that consists of brain nodes and connectivity structure between those nodes. This network discovery problem naturally exists in multiple domains including climate data, astronomical data and the focus of our tasks, fMRI scans of human subjects.

We proposed CGLasso and its extension MNGL to deal with different complex brain network problems in our works. Though CGLasso proposed in this dissertation can be applied in brain network discovery, it is not directly applicable to spatiotemporal problems. There are both advantages and disadvantages to this model. On the bright side, CGLasso presents much interpretable results by splitting some symmetric but discontinuous regions into a node. It makes sense for

the symmetrical structure of the right and left brain. On the other side, however, based on the objective function of this model, it can't make sure to find out spatially continuous nodes. To be interpreted as a cognitive brain network, the spatial regions should be continuous as these allow explanations in terms of the anatomical atlas. Although our experimental results on ADHD datasets show good spatial continuity of brain nodes, our method cannot ensure that this continuity is an inevitable consequence of the model itself. Therefore, we believe that s for our proposed models such as CGLasso and NMGL which can yield many important breakthroughs.

4.2.2 Spike Train Classification

Spike train classification has recently become an important topic in the machine learning community, where each spike train is a binary event sequence with *temporal-sparsity of signals of interest*. In our last two tasks, we focus on designing deep learning model with low power consumption for analyzing spike trains on resource-constrained platforms.

In our tasks, we considered the spike trains that are converted from static/dynamic image datasets in our experiments. However, the proposed model should generalize well to different original data types including but not limited to image or video sequences. Therefore, before the spike train can be used in energy efficient models, it typically undergoes a series of pre-processing steps for signal conversion.

In the last work, we focused on simulating turning on or off behaviors of the network for efficient object detection. Our proposed model uses synchronisation pulses that act as additional inputs to the controller, in order to provide the infor-

mation of time. The final performance of our model is sensitive to the frequencies of pulses. Thus, it may happen to miss some important timesteps, consequently, resulting in an energy-efficient model with some loss of accuracy. There are two potential ways to improve our model: First, our current method can only quickly find signals of interest by skipping forthcoming timesteps and shortening the effective size of the computational graph. So a potential way to further improve the accuracy is to enable the model to recall previous timesteps while skipping forthcoming timesteps; Second, we can also consider training a separate but much simpler network to learn whether to skip forthcoming timesteps.

Bibliography

[1] Hang Yin, Xiangnan Kong, and Xinyue Liu. Coherent graphical lasso for brain network discovery. In *2018 IEEE International Conference on Data Mining*, pages 1392–1397. IEEE, 2018. 3, 45, 47, 49, 56

[2] Hang Yin, Xinyue Liu, and Xiangnan Kong. Gaussian mixture graphical lasso with application to edge detection in brain networks. In *2020 IEEE International Conference on Big Data (Big Data)*, pages 1430–1435. IEEE, 2020. 3

[3] Hang Yin, John Boaz Lee, Xiangnan Kong, Thomas Hartvigsen, and Sihong Xie. Energy-efficient models for high-dimensional spike train classification using sparse spiking neural networks. In *Proceedings of the 27th ACM SIGKDD Conference on Knowledge Discovery & Data Mining*, pages 2017–2025, 2021. 4, 92, 96, 97

[4] Ian Davidson, Sean Gilpin, Owen Carmichael, and Peter Walker. Network discovery via constrained tensor analysis of fmri data. In *Proceedings of the 19th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (SIGKDD '13)*, pages 194–202. ACM, 2013. 5, 12

[5] Zilong Bai, Peter Walker, Anna Tschiffely, Fei Wang, and Ian Davidson. Unsupervised network discovery for brain imaging data. In *Proceedings of the 23rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (SIGKDD '17)*, pages 55–64. ACM, 2017. 5, 7, 8, 12, 23, 29, 45, 46, 47, 56

[6] Jerome Friedman, Trevor Hastie, and Robert Tibshirani. Sparse inverse covariance estimation with the graphical lasso. *Biostatistics*, 9(3):432–441, 2008. 7, 11, 33, 47

[7] Onureena Banerjee, Laurent El Ghaoui, and Alexandre d'Aspremont. Model selection through sparse maximum likelihood estimation for multivariate

gaussian or binary data. *Journal of Machine Learning Research*, 9(Mar):485–516, 2008. 7, 11, 47

[8] Benjamin M Marlin and Kevin P Murphy. Sparse gaussian graphical models with unknown block structure. In *Proceedings of the 26th International Conference on Machine Learning (ICML '09)*, pages 705–712, 2009. 7, 11

[9] Rahul Mazumder and Trevor Hastie. Exact covariance thresholding into connected components for large-scale graphical lasso. *Journal of Machine Learning Research*, 13(Mar):781–794, 2012. 7, 11

[10] Liang Sun, Rinkal Patel, Jun Liu, Kewei Chen, Teresa Wu, Jing Li, Eric Reiman, and Jieping Ye. Mining brain region connectivity for alzheimer's disease study via sparse inverse covariance estimation. In *Proceedings of the 15th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (SIGKDD '09)*, pages 1335–1344. ACM, 2009. 7, 11

[11] Xinyue Liu, Xiangnan Kong, and S Yu Philip. Collective discovery of brain networks with unknown groups. In *Proceedings of the 30th International Joint Conference on Neural Networks (IJCNN '17)*, pages 3569–3576. IEEE, 2017. 7, 8, 9, 11, 45, 46

[12] Shuai Huang, Jing Li, Jieping Ye, Adam Fleisher, Kewei Chen, Teresa Wu, and Eric Reiman. Brain effective connectivity modeling for alzheimer's disease by sparse gaussian bayesian network. In *Proceedings of the 17th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (SIGKDD '11)*, pages 931–939. ACM, 2011. 11, 32

[13] Karl J Friston. Functional and effective connectivity: a review. *Brain Connectivity*, 1(1):13–36, 2011. 11, 33

[14] Shuai Huang, Jing Li, Jieping Ye, Teresa Wu, Kewei Chen, Adam Fleisher, and Eric Reiman. Identifying alzheimer's disease-related brain regions from multi-modality neuroimaging data using sparse composite linear discrimination analysis. In *Proceedings of the 24th Advances in Neural Information Processing Systems (NIPS '11)*, pages 1431–1439, 2011. 12

[15] Dan A Alcantara, Owen Carmichael, Will Harcourt-Smith, Kirstin Sterner, Stephen R Frost, Rebecca Dutton, Paul Thompson, Eric Delson, and Nina Amenta. Exploration of shape variation using localized components analysis. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 31(8):1510–1516, 2009. 12

[16] Jean-Philippe Brunet, Pablo Tamayo, Todd R Golub, and Jill P Mesirov. Metagenes and molecular pattern discovery using matrix factorization. *Proceedings of the National Academy of Sciences*, 101(12):4164–4169, 2004. 12, 47

[17] Matthew Cooper and Jonathan Foote. Summarizing video using non-negative similarity matrix factorization. In *Proceedings of IEEE Workshop on Multimedia Signal Processing*, pages 25–28. IEEE, 2002. 12

[18] Patrik O Hoyer. Non-negative matrix factorization with sparseness constraints. *Journal of Machine Learning Research*, 5(Nov):1457–1469, 2004. 12

[19] Stan Z Li, Xin Wen Hou, Hong Jiang Zhang, and Qian Sheng Cheng. Learning spatially localized, parts-based representation. In *Proceedings of the 17th IEEE Conference on Computer Vision and Pattern Recognition (CVPR '01)*, volume 1, pages 207–212. IEEE, 2001. 12

[20] Pentti Paatero and Unto Tapper. Positive matrix factorization: A non-negative factor model with optimal utilization of error estimates of data values. *Environmetrics*, 5(2):111–126, 1994. 12

[21] Wei Xu, Xin Liu, and Yihong Gong. Document clustering based on non-negative matrix factorization. In *Proceedings of the 26th International SIGIR Conference on Research and Development in Informaion Retrieval (SIGIR '03)*, pages 267–273. ACM, 2003. 12

[22] Daniel D Lee and H Sebastian Seung. Learning the parts of objects by non-negative matrix factorization. *Nature*, 401(6755):788, 1999. 12

[23] Chris Ding, Xiaofeng He, and Horst D Simon. On the equivalence of non-negative matrix factorization and spectral clustering. In *Proceedings of the 5th SIAM International Conference on Data Mining (SDM '05)*, pages 606–610. SIAM, 2005. 12, 14

[24] Chris Ding, Tao Li, Wei Peng, and Haesun Park. Orthogonal nonnegative matrix t-factorizations for clustering. In *Proceedings of the 12th ACM SIGKDD International Conference on Knowledge discovery and Data Mining (SIGKDD '06)*, pages 126–135. ACM, 2006. 12, 13

[25] Jerome Friedman, Trevor Hastie, and Robert Tibshirani. Sparse inverse covariance estimation with the graphical lasso. *Biostatistics*, 9(3):432–441, 2008. 14, 19

[26] Daniel D Lee and H Sebastian Seung. Algorithms for non-negative matrix factorization. In *Proceedings of the 14th Advances in Neural Information Processing Systems (NIPS '01)*, pages 556–562, 2001. 18

[27] Robert Tibshirani. Regression shrinkage and selection via the lasso. *Journal of the Royal Statistical Society. Series B (Methodological)*, pages 267–288, 1996. 19

[28] Jianqing Fan. Comments on wavelets in statistics: A review by a. antoniadis. *Journal of the Italian Statistical Society*, 6(2):131, 1997. 20

[29] Hui Zou. The adaptive lasso and its oracle properties. *Journal of the American Statistical Association*, 101(476):1418–1429, 2006. 20, 34

[30] Jianqing Fan and Runze Li. Variable selection via nonconcave penalized likelihood and its oracle properties. *Journal of the American Statistical Association*, 96(456):1348–1360, 2001. 20

[31] R Cameron Craddock, G Andrew James, Paul E Holtzheimer, Xiaoping P Hu, and Helen S Mayberg. A whole brain fmri atlas generated via spatially constrained spectral clustering. *Human Brain Mapping*, 33(8):1914–1928, 2012. 23

[32] Sen Yang, Zhaosong Lu, Xiaotong Shen, Peter Wonka, and Jieping Ye. Fused multiple graphical lasso. *SIAM Journal on Optimization*, 25(2):916–943, 2015. 23

[33] Chia-Tung Kuo, Xiang Wang, Peter Walker, Owen Carmichael, Jieping Ye, and Ian Davidson. Unified and contrasting cuts in multiple graphs: Application to medical imaging segmentation. In *Proceedings of the 21st ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (SIGKDD '15)*, pages 617–626. ACM, 2015. 28, 63

[34] Mehran Ahmadlou, Hojjat Adeli, and Amir Adeli. Graph theoretical analysis of organization of functional brain networks in adhd. *Clinical EEG and Neuroscience*, 43(1):5–13, 2012. 30

[35] Ibai Diez and Jorge Sepulcre. Neurogenetic profiles delineate large-scale connectivity dynamics of the human brain. *Nature Communications*, 9(1):1–10, 2018. 31, 44

[36] Thomas Hofmann. Probabilistic latent semantic analysis. *arXiv preprint arXiv:1301.6705*, 2013. 32, 45, 46

[37] Alana J Anderson and Sammy Perone. Developmental change in the resting state electroencephalogram: Insights into cognition and the brain. *Brain and cognition*, 126:40–52, 2018. 33, 44

[38] Chen Gao, Yunzhang Zhu, Xiaotong Shen, and Wei Pan. Estimation of multiple networks in gaussian mixture models. *Electronic Journal of Statistics*, 10:1133, 2016. 33, 37, 48, 56

[39] Pan Lin, Yong Yang, Junfeng Gao, Nicola De Pisapia, Sheng Ge, Xiang Wang, Chun S Zuo, James Jonathan Levitt, and Chen Niu. Dynamic default mode network across different brain states. *Scientific reports*, 7(1):1–13, 2017. 44, 45

[40] Andrew McCallum, Kamal Nigam, et al. A comparison of event models for naive bayes text classification. In *AAAI-98 workshop on learning for text categorization*, volume 752, pages 41–48. Citeseer, 1998. 46, 47

[41] Karl Pearson. Contributions to the mathematical theory of evolution. *Philosophical Transactions of the Royal Society of London. A*, 185:71–110, 1894. 48

[42] Kerry Back and David P Brown. Implied probabilities in gmm estimators. *Econometrica: Journal of the Econometric Society*, pages 971–975, 1993. 48

[43] Arthur P Dempster, Nan M Laird, and Donald B Rubin. Maximum likelihood from incomplete data via the em algorithm. *Journal of the Royal Statistical Society. Series B (methodological)*, pages 1–38, 1977. 48

[44] Gideon Schwarz et al. Estimating the dimension of a model. *The Annals of Statistics*, 6(2):461–464, 1978. 48

[45] Hirotugu Akaike. A new look at the statistical model identification. *IEEE Transactions on Automatic Control*, 19(6):716–723, 1974. 48

[46] Mark EJ Newman and Elizabeth A Leicht. Mixture models and exploratory analysis in networks. *Proceedings of the National Academy of Sciences*, 104(23):9564–9569, 2007. 48

[47] Yunzhang Zhu, Xiaotong Shen, and Wei Pan. Structural pursuit over multiple undirected graphs. *Journal of the American Statistical Association*, 109(508):1683–1696, 2014. 48

[48] Geoffrey McLachlan and David Peel. Finite mixture models, willey series in probability and statistics, 2000. 48

[49] Sijian Wang and Ji Zhu. Variable selection for model-based high-dimensional clustering and its application to microarray data. *Biometrics*, 64(2):440–448, 2008. 48

[50] Benhuai Xie, Wei Pan, and Xiaotong Shen. Penalized model-based clustering with cluster-specific diagonal covariance matrices and grouped variables. *Electronic Journal of Statistics*, 2:168, 2008. 48

[51] Hui Zhou, Wei Pan, and Xiaotong Shen. Penalized model-based clustering with unconstrained covariance matrices. *Electronic Journal of Statistics*, 3:1473, 2009. 48

[52] Steven M Hill and Sach Mukherjee. Network-based clustering with mixtures of l1-penalized gaussian graphical models: an empirical investigation. *arXiv preprint arXiv:1301.2194*, 2013. 48

[53] Meng Yun Wu, Dao Qing Dai, Xiao Fei Zhang, and Yuan Zhu. Cancer subtype discovery and biomarker identification via a new robust network clustering algorithm. *PloS One*, 8(6), 2013. 48

[54] Hang Yin, Xinyue Liu, and Xiangnan Kong. Gaussian mixture graphical lasso with application to edge detection in brain networks, 2021. 48

[55] Michael Pfeiffer and Thomas Pfeil. Deep learning with spiking neurons: opportunities and challenges. *Frontiers in neuroscience*, 12:774, 2018. 66, 92

[56] Srinjoy Mitra, Stefano Fusi, and Giacomo Indiveri. Real-time classification of complex patterns using spike-based learning in neuromorphic vlsi. *IEEE transactions on biomedical circuits and systems*, 3(1):32–42, 2008. 66, 92

[57] Yujie Wu, Lei Deng, Guoqi Li, Jun Zhu, and Luping Shi. Spatio-temporal backpropagation for training high-performance spiking neural networks. *Frontiers in neuroscience*, 12:331, 2018. 66, 67, 68, 70, 73, 82, 85, 86, 87, 92, 93, 101

[58] Yujie Wu, Lei Deng, Guoqi Li, Jun Zhu, Yuan Xie, and Luping Shi. Direct training for spiking neural networks: Faster, larger, better. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 33, pages 1311–1318, 2019. 66, 67, 68, 71, 72, 73, 79, 80, 81, 92, 96, 97, 101

[59] Evangelos Stromatias, Miguel Soto, Teresa Serrano-Gotarredona, and Bernabé Linares-Barranco. An event-driven classifier for spiking neural networks fed with synthetic or dynamic vision sensor data. *Frontiers in neuroscience*, 11:350, 2017. 67

[60] Sumit Bam Shrestha and Garrick Orchard. Slayer: Spike layer error reassignment in time. In *Advances in neural information processing systems*, pages 1412–1421, 2018. 67, 70, 71, 92, 93, 96, 97

[61] Amirhossein Tavanaei, Masoud Ghodrati, Saeed Reza Kheradpisheh, Timothée Masquelier, and Anthony Maida. Deep learning in spiking neural networks. *Neural Networks*, 111:47–63, 2019. 67

[62] Giacomo Indiveri and Shih-Chii Liu. Memory and information processing in neuromorphic systems. *Proceedings of the IEEE*, 103(8):1379–1397, 2015. 67

[63] Sharan Narang, Eric Undersander, and Gregory Diamos. Block-sparse recurrent neural networks. *arXiv preprint arXiv:1711.02782*, 2017. 69

[64] Ruizhi Chen, Hong Ma, Shaolin Xie, Peng Guo, Pin Li, and Donglin Wang. Fast and efficient deep sparse multi-strength spiking neural networks with dynamic pruning. In *2018 International Joint Conference on Neural Networks (IJCNN)*, pages 1–8. IEEE, 2018. 69, 70, 71, 80, 82, 85, 86, 87

[65] Mohammed Alawad, Hong-Jun Yoon, and Georgia Tourassi. Energy efficient stochastic-based deep spiking neural networks for sparse datasets. In *2017 IEEE International Conference on Big Data (Big Data)*, pages 311–318. IEEE, 2017. 69, 71, 80, 82, 85, 86, 87

[66] Yarin Gal, Jiri Hron, and Alex Kendall. Concrete dropout. In *Advances in neural information processing systems*, pages 3581–3590, 2017. 69

[67] Christos Louizos, Max Welling, and Diederik P Kingma. Learning sparse neural networks through l_0 regularization. *arXiv preprint arXiv:1712.01312*, 2017. 69, 75, 76

[68] Peter U Diehl, Daniel Neil, Jonathan Binas, Matthew Cook, Shih-Chii Liu, and Michael Pfeiffer. Fast-classifying, high-accuracy spiking deep networks through weight and threshold balancing. In *2015 International Joint Conference on Neural Networks*, pages 1–8. ieee, 2015. 70, 96

[69] Yongqiang Cao, Yang Chen, and Deepak Khosla. Spiking deep convolutional neural networks for energy-efficient object recognition. *International Journal of Computer Vision*, 113(1):54–66, 2015. 70, 96

[70] Yangfan Hu, Huajin Tang, Yueming Wang, and Gang Pan. Spiking deep residual network. *arXiv preprint arXiv:1805.01352*, 2018. 70, 96

[71] Rivu Midya, Zhongrui Wang, Shiva Asapu, Saumil Joshi, Yunning Li, Ye Zhuo, Wenhao Song, Hao Jiang, Navnidhi Upadhay, Mingyi Rao, et al. Artificial neural network (ann) to spiking neural network (snn) converters based on diffusive memristors. *Advanced Electronic Materials*, 5(9):1900060, 2019. 70, 96

[72] Jun Haeng Lee, Tobi Delbruck, and Michael Pfeiffer. Training deep spiking neural networks using backpropagation. *Frontiers in neuroscience*, 10:508, 2016. 71

[73] Yingyezhe Jin, Wenrui Zhang, and Peng Li. Hybrid macro/micro level backpropagation for training deep spiking neural networks. In *Advances in neural information processing systems*, pages 7005–7015, 2018. 71

[74] Toby J Mitchell and John J Beauchamp. Bayesian variable selection in linear regression. *Journal of the american statistical association*, 83(404):1023–1032, 1988. 76

[75] Ronald J Williams. Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Machine learning*, 8(3-4):229–256, 1992. 76

[76] Yoshua Bengio, Nicholas Léonard, and Aaron Courville. Estimating or propagating gradients through stochastic neurons for conditional computation. *arXiv preprint arXiv:1308.3432*, 2013. 76

[77] Durk P Kingma, Tim Salimans, and Max Welling. Variational dropout and the local reparameterization trick. In *Advances in neural information processing systems*, pages 2575–2583, 2015. 77

[78] Chris J Maddison, Andriy Mnih, and Yee Whye Teh. The concrete distribution: A continuous relaxation of discrete random variables. *arXiv preprint arXiv:1611.00712*, 2016. 78, 84

[79] Garrick Orchard, Ajinkya Jayawant, Gregory K Cohen, and Nitish Thakor. Converting static image datasets to spiking neuromorphic datasets using saccades. *Frontiers in neuroscience*, 9:437, 2015. 83, 104

[80] Arnon Amir, Brian Taba, David Berg, Timothy Melano, Jeffrey McKinstry, Carmelo Di Nolfo, Tapan Nayak, Alexander Andreopoulos, Guillaume Garreau, Marcela Mendoza, et al. A low power, fully event-based gesture recognition system. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 7243–7252, 2017. 83, 104

[81] David Drazen, Patrick Lichtsteiner, Philipp Häfliger, Tobi Delbrück, and Atle Jensen. Toward real-time particle tracking using an event-based dynamic vision sensor. *Experiments in Fluids*, 51(5):1465–1469, 2011. 92

[82] Elias Mueggler, Henri Rebecq, Guillermo Gallego, Tobi Delbruck, and Davide Scaramuzza. The event-camera dataset and simulator: Event-based data for pose estimation, visual odometry, and slam. *The International Journal of Robotics Research*, 36(2):142–149, 2017. 92

[83] Henri Rebecq, René Ranftl, Vladlen Koltun, and Davide Scaramuzza. High speed and high dynamic range video with an event camera. *IEEE transactions on pattern analysis and machine intelligence*, 43(6):1964–1980, 2019. 92

[84] Jianguo Xin and Mark J Embrechts. Supervised learning with spiking neural networks. In *IJCNN'01. International Joint Conference on Neural Networks. Proceedings (Cat. No. 01CH37222)*, volume 3, pages 1772–1777. IEEE, 2001. 92

[85] Johannes Schemmel, Andreas Grubl, Karlheinz Meier, and Eilif Mueller. Implementing synaptic plasticity in a vlsi spiking neural network model. In *The 2006 ieee international joint conference on neural network proceedings*, pages 1–6. IEEE, 2006. 92

[86] John J Wade, Liam J McDaid, Jose A Santos, and Heather M Sayers. Swat: a spiking neural network training algorithm for classification problems. *IEEE transactions on neural networks*, 21(11):1817–1830, 2010. 92

[87] Guillermo Gallego, Tobi Delbruck, Garrick Orchard, Chiara Bartolozzi, Brian Taba, Andrea Censi, Stefan Leutenegger, Andrew Davison, Jörg Conradt, Kostas Daniilidis, et al. Event-based vision: A survey. *arXiv preprint arXiv:1904.08405*, 2019. 92

[88] Diederik Paul Moeys, Federico Corradi, Chenghan Li, Simeon A Bamford, Luca Longinotti, Fabian F Voigt, Stewart Berry, Gemma Taverni, Fritjof Helmchen, and Tobi Delbruck. A sensitive dynamic and active pixel vision sensor for color or neural imaging applications. *IEEE transactions on biomedical circuits and systems*, 12(1):123–136, 2017. 92

[89] Yuji Nozaki and Tobi Delbruck. Temperature and parasitic photocurrent effects in dynamic vision sensors. *IEEE Transactions on Electron Devices*, 64(8):3239–3245, 2017. 92

[90] Emery N Brown, Robert E Kass, and Partha P Mitra. Multiple neural spike train data analysis: state-of-the-art and future challenges. *Nature neuroscience*, 7(5):456–461, 2004. 92

[91] Víctor Campos, Brendan Jou, Xavier Giró-i Nieto, Jordi Torres, and Shih-Fu Chang. Skip rnn: Learning to skip state updates in recurrent neural networks. *arXiv preprint arXiv:1708.06834*, 2017. 93, 97, 98, 103, 106

[92] Xavier Glorot and Yoshua Bengio. Understanding the difficulty of training deep feedforward neural networks. In *Proceedings of the thirteenth international conference on artificial intelligence and statistics*, pages 249–256. JMLR Workshop and Conference Proceedings, 2010. 95

[93] Varsha S Lalapura, J Amudha, and Hariramn Selvamuruga Satheesh. Recurrent neural networks for edge intelligence: A survey. *ACM Computing Surveys (CSUR)*, 54(4):1–38, 2021. 97

[94] Tao Lei. When attention meets fast recurrence: Training language models with reduced compute. In *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing*, pages 7633–7648, 2021. 97

[95] Romero Morais, Vuong Le, Svetha Venkatesh, and Truyen Tran. Learning asynchronous and sparse human-object interaction in videos. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 16041–16050, 2021. 97

[96] Wendong Zheng and Gang Chen. An accurate gru-based power time-series prediction approach with selective state updating and stochastic optimization. *IEEE Transactions on Cybernetics*, 2021. 97

[97] Thomas Hartvigsen, Cansu Sen, Xiangnan Kong, and Elke Rundensteiner. Learning to selectively update state neurons in recurrent networks. In *Proceedings of the 29th ACM International Conference on Information & Knowledge Management*, pages 485–494, 2020. 97

[98] Yacine Jernite, Edouard Grave, Armand Joulin, and Tomas Mikolov. Variable computation in recurrent neural networks. *arXiv preprint arXiv:1611.06188*, 2016. 97

[99] Yikang Shen, Shawn Tan, Alessandro Sordoni, and Aaron Courville. Ordered neurons: Integrating tree structures into recurrent neural networks. *arXiv preprint arXiv:1810.09536*, 2018. 97

[100] Wolfgang Klimesch. Alpha-band oscillations, attention, and controlled access to stored information. *Trends in cognitive sciences*, 16(12):606–617, 2012. 99

[101] Gyorgy Buzsaki. *Rhythms of the Brain*. Oxford university press, 2006. 99