Representation Learning on Large-Scale Neural and Healthcare Data: A Practitioner's Perspective

A DISSERTATION

SUBMITTED TO THE FACULTY OF THE GRADUATE SCHOOL

OF THE UNIVERSITY OF MINNESOTA

BY

Tong Wu

IN PARTIAL FULFILLMENT OF THE REQUIREMENTS

FOR THE DEGREE OF

DOCTOR OF PHILOSOPHY

Dr. Zhi Yang

August, 2020

Acknowledgements

I would like to sincerely thank my advisor, Dr. Zhi Yang for the comprehensive training, caring, and help that I received from him in the past decade. I learned not only knowledge and research skills from Dr. Zhi Yang, but also the most important traits and mentalities to be an independent and mature researcher: being courageous, confident, self-disciplined, self-motivated, and compassionate. The years that I have spent with him have truly changed my life. I thank the committee members (also my prelim commitee members), Dr. Hubert Lim, Dr. Catherine Qi Zhao, Dr. Alexander Opitz, and Dr. Tay Netoff for participating in my thesis, and providing insightful comments and helpful feedback. I would like to acknowledge all the help I received from researchers I collaborated with over the past years: Dr. Edward Keefer, Dr. Chun-Huat Heng, Dr. László Grand, Dr. Yunlong Wang, Dr. Gao Wang, Dr. Cao Xiao, Dr. Emily Zhao, Dr. Yilian Yuan, Dr. Azam Khalili, and Dr. Amir Rastegarnia. I thank my colleagues and school mates, not just for research collaboration but also for supporting each other and having fun together: Dr. Jian Xu, Dr. Wenfeng Zhao, Dr. Yin Zhou, Dr. Wing-Kin Tam, Dr. Hongsun Guo, Yan Luo, Dr. Ming Jiang, Dr. Kok-Hin Teng, Tianqi Li, Anh Tuan Nguyen, Edward Bello, Diu Khue Luu, Markus Drealan, Fan Zhang, and Yue Wang. My last thanks go to my family, especially my wife Dr. Zheng Wang who have supported me profoundly. I would never come this far without their support and love.

Abstract

The rapid growth of data volume has promoted development of advanced computational methods, particularly machine learning (ML) and deep learning (DL) algorithms, to address challenges emerged in various fields. A key step to solve ML/DL tasks is to find a good data representation by mapping input data into a feature space where representations can entangle or disentangle the different explanatory factors of variation behind the data, and predictive modeling can be more accurate and reliable. In this thesis, we focus on the study of representation learning methods in neuroscience and healthcare, and propose algorithms based on recent ML/DL developments to address both critical and practical challenges in neural signal processing and healthcare predictive modeling.

For neural signal processing, we first strive to bridge the gap of the fast-increasing scale of data acquisition in neural recording and the limited bandwidth of data links for transmission. We propose two unsupervised compression algorithms to reduce the bandwidth of neural signals without sacrificing their utilities in downstream tasks. This is mainly achieved by leveraging the morphological consistency of neural signals across geometrically adjacent recording sites to capture common data variations. Next, we propose a semi-automatic spike sorting algorithm to decompose multi-unit recordings into single-unit activities based on adversarial representation learning that can sort spikes from a small number of labeled examples, thereby mitigating the data-hungry limitation of DL-based classification models.

For healthcare predictive modeling, we propose to represent the hierarchical and relational structures of medical entities (patients, doctors, and medical services) in patients electronic health records (EHR) using a collection of graph-based network embedding algorithms. The proposed framework can bring a number of advantages such as enhanced clinical outcome prediction accuracies and more interpretable modeling of patient medical profiles and treatment history, which suggest the potential of being used as a comprehensive and general-purpose solution for representation learning of EHR data.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

1.1 Big Data in Neuroscience and Healthcare

We are entering an era of big data thanks to the fast advancements of information technologies in computation and storage. This has impacted many research areas. In this dissertation, we focus on neuroscience and healthcare that are undergoing unprecedented transformations in terms of scales and machine-enabled autonomy [8, 9]. In the rest of the section, we take a closer look at two aspects from neuroscience and healthcare, respectively: the massive scale of neural data recorded from human nervous systems, and the extensive adoption of electronic health record in healthcare systems.

1.1.1 Electrophysiological Recording

In electrophysiological studies of brains, the fast advancements in manufacturing and fabrication technologies of microelectrodes and electronics have allowed the development of data acquisition systems that can record neural activities of animal or human subjects from hundreds to thousands of channels simultaneously [10]. For instance, one notable development of high-density, large-scale microelectrodes recently is *Neuropixels*, which is 10-mm long, 70×20-μm cross-section per shank, and integrates 384 recording channels within one silicon probe [11]. More follow-up works have appeared in recent years and pushed the limits of channel count and density of recording sites even further [12, 13, 14].

For *in vivo* recording, current microelectrode technologies typically capture $\sim$0.5 neuron per channel on average [15]. We can fit a curve using the data reported in publications on neural recordings over the years, and derive a trajectory of the number of simultaneously recorded neurons, which is approximately doubling every 6.3 years, similar to Moore's law [16]. By 2020, we can already record the activities of over one thousand neurons at the same time, which is also in line with the recent progress [17, 18]. However, this is nowhere near the amount of neurons that we aim to record from concurrently with single-cell resolution to understand how the brain represents, transforms, and communicates information [19]. For example, it is estimated that a brain-machine interface (BMI) aimed at restoring dexterous limb movements would require 5,000–10,000 neurons recorded simultaneously from the motor cortex [20]. As another example, the Neural Engineering System Design (NESD) program announced by DARPA aimed to design neural interfaces that "can read 10^6 neurons, write to 10^5 neurons, and interact with 10^3 neurons full-duplex"[1] to "help deepen our understanding of that organ's underlying biology, complexity, and function".[2]

Reading 10^6 neurons simultaneously would generate 700–800 gigabyte bits per second (Gbps) with a sampling rate of 32 kHz and a 12-bit analog-to-digital precision. The demand for recording large populations of neurons presents challenges not only for electrode microfabrication technologies that must increase channel count dramatically while maintaining a reasonable device size, but also for data processing techniques that must scale well with the data volume and uncover hidden knowledge essential to downstream tasks.

1.1.2 Electronic Health Record

Healthcare, according to Wikipedia, has been defined as "the maintenance or improvement of health via the prevention, diagnosis, treatment, recovery, or cure of disease, illness, injury, and other physical and mental impairments in people. Healthcare is delivered by health professionals in allied health fields. Physicians and physician associates are a part of these health professionals. Dentistry, pharmacy, midwifery, nursing, medicine, optometry, audiology, psychology, occupational therapy, physical therapy and other health professions are all part of healthcare. It includes work done in providing primary care, secondary care, and tertiary care, as well as in public health.".[3]

A plethora of health data are generated, collected, and used when patients or subjects interact with healthcare systems, which come from a wide and diverse range of sources,

[1] https://www.darpa.mil/program/neural-engineering-system-design
[2] https://www.darpa.mil/news-events/2017-07-10
[3] https://en.wikipedia.org/wiki/Health_care

including clinics/physician offices, pharmacies, payers/insurance companies, hospitals, and laboratories, as well as participation in clinical trials, health agency surveys, medical devices, and genomic testing. Reports said that in 2011, there were about 150 exabytes of health data generated in the United States [21]. The volume of health data is expected to increase dramatically, to the level of about 2,314 exabytes in 2020.[4]

Electronic health record (EHR) is the systematized collection of patient and population electronically-stored health information in a digital format [22]. EHR data comprise various data types, including patient demographics, medical history, medication and allergies, immunization status, laboratory test results, radiology images, vital signs, and billing information. Thanks to initiates like the US$19 billion HITEC act [23] in the United States and the €2 billion public-private partnership Innovative Medicine Initiative (IMI) [24] in the European Union, the adoption of EHR in healthcare systems has skyrocketed over the past 10 years. According to the report from the Office of the National Coordinator for Health Information Technology (ONC) in 2016, nearly 84% of US hospitals have adopted at least a basic EHR system, a 9-fold increase since 2008 [25, 26].

1.2 Machine Learning in Neuroscience and Healthcare

The rapid growth of data volume has necessitated and promoted development of advanced computational methods, particularly machine learning (ML) algorithms, to address various challenges emerged in neuroscience and healthcare. In this section, we first discuss some basics of representation learning and its contrast to traditional feature engineering, then survey several representative applications of ML algorithms in neuroscience and healthcare with a special emphasis on how representation learning (or feature extraction in general) contributes to information extraction and outcomes prediction in each task.

1.2.1 Representation Learning

In general, successful application of ML algorithms is heavily dependent upon finding a good representation of input data through data preprocessing and transformations, based on which predictive modeling can be more effective and reliable than using the raw inputs. In the past, such feature engineering has been mostly done manually to leverage human

[4] https://www.visualcapitalist.com/big-data-healthcare/

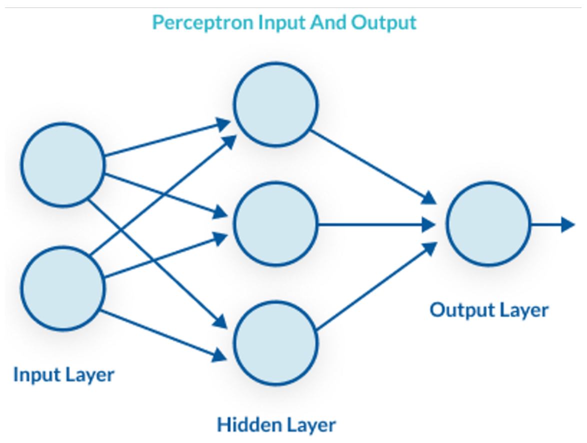


Figure 1.1: Schematic diagram of an multilayer perceptron. Figure courtesy of [5].

understanding and prior knowledge, hence is labor-intensive and also subject to human biases. In order to overcome these limitations and expand the scope of applicability of ML algorithms to tasks where manual feature engineering is difficult, data-driven representation learning approaches have been developed and become an indispensable and critical component in modern ML algorithms, in particular, deep learning (DL) [27].

What is data-driven representation learning? Take *multilayer perceptron* (MLP), a basic feed-forward neural network, as an example. Figure 1.1 shows the schematic diagram of an MLP that comprises three layers: input layer, hidden layer, and output layer. Each layer consists of a varied number of perceptrons (or neurons), each of which implements a linear weighted summation of outputs of neurons in the previous layer, followed by a nonlinear activation function. More formally, the operation of one neuron $f(\cdot)$ can be described as

$$f(\mathbf{x}) = \sigma(\mathbf{w}^\top \mathbf{x} + b), \tag{1.1}$$

where $\{\mathbf{x}, \mathbf{w}\} \in \mathbb{R}^n$ are the n-dimensional input vector and weight vector of the neuron, respectively, b is the bias coefficient, and $\sigma(\cdot)$ represents a nonlinear activation function, such as logistic function. To determine $\mathbf{w}$ and b, we acquire a sufficient amount of ground truth data $\{(\mathbf{x}_i, y_i)\}_{i=1}^N$ (note: $\mathbf{x}_i$ here refers to the MLP inputs) to train the MLP function $f_{MLP}(\cdot)$ via backpropagation [28] and stochastic gradient descent (SGD) [29], an optimization process that computes the loss function $\mathcal{L}(\cdot)$, i.e., a distance measure between each pair $(y_i, f_{MLP}(\mathbf{x}_i))$, and updates the MLP's parameters by the amount proportional to the gradient of the loss function with respect to the weights until the total loss $\sum_{i=1}^N \mathcal{L}(y_i, f_{MLP}(\mathbf{x}_i))$ is minimized. This type of training is also termed *supervised learning*, since ground truth

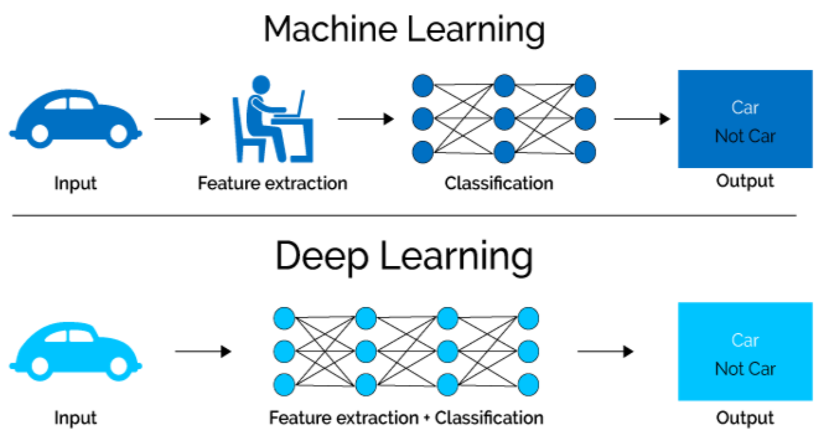


Figure 1.2: Feature engineering versus representation learning. Figure courtesy of [6].

data are required to compute and propagate the loss backward through the network. In contrast, *unsupervised learning* and *semi-supervised learning* (SSL) require none and partial ground truth data, respectively.

The idea of representation learning and its distinction from feature engineering are rooted in the comparison between ML and DL, which is illustrated in Figure 1.2 that shows two paradigms for the same task, a binary classification to recognize if the input image is a car or not. Although both the paradigms employ an MLP-like model, DL does not include a separate feature engineering or extraction prior to classification; instead, it employs a deeper MLP with extra hidden layers to "shoot two birds with one stone". In contrast to feature engineering that uses predefined, fixed mathematical rules, in representation learning (or more broadly, DL), we identify a problem space by specifying a neural network architecture (e.g., MLP) as well as an objective (e.g., reducing the misclassification rate of recognizing car images), and use computational resources to search this problem space via backpropagation and SGD for a working solution defined by the model structure and all the network parameters (weights, activation functions, etc.) [30].

The most predominant advantage of representation learning over feature engineering is *computational homogeneity*. For many real-world applications of ML, it is significantly easier to collect large amounts of data than to write explicit rules for prediction. For example, in the car image recognition task shown in Figure 1.2, defining features from images that can reliably differentiate cars from anything else is a nontrivial job, and would take many rounds of trial and error. Now what if one is asked again to recognize faces of Asian males from photos covering different gender and racial groups? Obviously the features used for car recognition do not apply for the new task, and new rounds of feature engineering are required. Hence, feature engineering is task-specific, heterogeneous, and labor-intensive. Meanwhile for data-driven representation, as long as we are dealing with the same type of tasks (e.g., image recognition, no matter for cars or faces), we could stick to the same model structure and objective, and train the model using corresponding data that can automatically find the features that are most effective for the task at hand.

Data-driven representation learning also has its own disadvantages. One apparent shortcoming is the requirement of large amounts of ground truth data for training. In many cases, this is not critical since it is fairly easy to collect and label large amounts of data at low or acceptable cost. However, there are scenarios where data collection and labeling poses great difficulties that can limit the application of data-driven representation learning at scale. Another disadvantage of representation learning that has probably received the most criticism is the lack of *interpretability* due to the black-box nature of DL models that underpin representation learning [31]. Interpretability is especially valued and demanded for ML applications in high-stakes scenarios such as healthcare [32], where healthcare providers want to gain insights on why a certain prediction was made for a patient, instead of just accurate predictions [33]. According to the universal approximation theorem [34], a simple DL model such an MLP with at least one hidden layer containing a finite number of neurons can approximate any continuous functions on $\mathbb{R}^n$, under mild assumptions on the activation function. Given the supreme ability of approximation, a DL model would strive to learn any arbitrary forms of representations from data that can lead to the most accurate predictions, regardless of whether the learned representations are interpretable or not.

For data-driven representation learning, interpretability and prediction accuracy are not always a conflicting pair of metrics for trade-off. For example, it has been found by several studies that deep convolutional neural networks (CNN) trained on object classification task using large-scale image databases (e.g., ImageNet [35], CIFAR-10 [36]) can achieve super-human performance and at the same time show a good agreement between hidden-layer features learned inside the model and neural activities in biological visual networks [37, 38, 39]. As CNN was initially inspired by biological processes of the primate visual cortex [40, 41, 42], there lies great hope in bio-inspired deep neural networks (DNN) trained on high-level abstraction tasks (e.g., object recognition) that can reveal biologically plausible and interpretable features without compromising performance.

1.2.2 Neural Signal Processing

Recent years have seen a surge of ML/DL applications in neural signal processing, thanks to both the substantial theoretical progress and the availability of high-performance and massively-parallelable computational resources (e.g., graphics processing units), that jointly provide the capability to handle large volume of neural data in both offline and online modes with minimal or even no human supervisions. In the rest of this subsection, we discuss neural decoding for prostheses control, a major category of neural signal processing.

Neural decoding for prostheses control The development of neural interface has been an effective channel through which patterns of neural activities are interpreted to interact with off-body assistive devices. Here *neural decoding* refers to the process of understanding and translating recorded neural signals into commands to actuate prosthetic devices. We limit the discussion of prostheses to those for restoring upper extremity functions for subjects with hand or trans-radial amputations. There are two major sources of signal acquisition, or *modalities*, for upper limb prostheses, which are electromyography (EMG) signals that are recorded from muscles with surface or intramuscular electrodes [43], and signals directly recorded from the peripheral nerve system (PNS) with epineural (e.g., cuff [44] and FINE [45]) or intraneural (e.g., LIFE [46, 47] and TIME [48]) electrodes.

EMG-based decoding EMG-based prostheses are the most commonly used type of prostheses in clinical settings because of the noninvasiveness and low risks to human bodies. The general flow of EMG-based decoding is similar to a typical ML prediction task, in which features are first extracted from recorded EMG data (usually also filtered to attenuate noise and artifacts), and then used to train a classification or regression model that predicts the category or trajectory of the intended movements. One major limitation of EMG-based decoding is the low resolution of EMG signals (the potential to isolate independent signal sources coming from different muscles), which is mainly caused by the inadequate remaining musculature, signal cross-talk contamination, and attenuation of deep muscle signals at the skin level. As a result, amputees can only utilize the EMG-based interfaces to control gross motions including forearm rotation, wrist flexion and extension, and digit flexion and extension, all of which have a degree-of-freedom (DoF) rarely greater than 2 [49, 50].

Table 1.1: Types of EMG features and examples. Table modified from Table 2 in [1].

Feature Types	Examples
Time	Linear envelope [56], mean absolute value, root mean square [57], zero crossings, slope sign changes, waveform length [58], wave complexity [59], Willison amplitude [60], log-detector [61], histogram [62]
Frequency	Power spectral moments [63], power spectral density [64], spectral magnitude averages [65], short time Fourier transform, median frequency [66], cepstrum [67], short time Thompson transform [65]
Time-frequency	Wavelet packet transform [68, 69, 70], discrete wavelet transform [52, 71]
Synergy	NMF [72], PCA [66], ICA, fuzzy clustering [73], LDA, orthogonal fuzzy neighborhood discriminant analysis [74], self organizing feature maps, Common spatio-spectral pattern [75], multiresolution muscle synergy analysis [76]

To address this challenge, tremendous research efforts have been devoted in feature engineering for EMG-based decoding that leverages a wide range of mathematical transformations to extract useful information content that correlates accurately and robustly with movements. Features can be roughly categorized into four groups: time domain, frequency domain, time-frequency domain, and synergy features. Time-domain features are based on signal amplitudes, which are believed to reflect the number and rate of motor unit activations [51]; frequency-domain features mainly concern the information of rate and shape of motor unit action potentials (MUAPs); features from the time-frequency domain represent transient as well as steady-state patterns from dynamic muscle contractions [52, 53]; synergy features extract information from multiple EMG recoding sites simultaneously to depict time-invariant synergies representing underlying muscle coordination while performing various tasks [54, 55]. Table 1.1 gives examples for each of the categories that have been explored in previous works, which is far from being exhaustive.

There have been several attempts of EMG-based decoding that leverages DL techniques in recent years, which either perform data-driven feature learning or still do conventional feature engineering and simply use DL models for enhanced prediction [77, 78, 79]. This line of research is drastically outnumbered by EMG-based decoding based on feature engineering. The heavy reliance on feature engineering is attributed to the difficulty of obtaining ground truth data, i.e., the kinematics of hand movement, which are essential for the success of data-driven representation learning. To obtain hand kinematics, subjects are often asked to do contralateral mirror movements, where the actual trajectory information are measured from the intact hand, since the kinematics of the hand and wrist cannot be recorded from the missing limb. This process is time-consuming, and also presents both mental and physical burdens to the subjects if lasting for a prolonged period of time to collect sufficient ground truth data. As a result, data-driven representation learning methods that normally require a substantial amount of training data are not suitable here.

PNS-based decoding In comparison, neural decoding from PNS signals for prostheses control is much more challenging, and also rare, than EMG-based decoding. It requires sensing, understanding, and manipulation of neural signals at fascicular, or even sub-fascicular precision for functionally "connecting" the nerves system and the robotic hand. It poses multiple challenges, including (1) to develop high-fidelity neural recording capability, and (2) to design dedicated, effective ML algorithms for analyzing and decoding nerve data.

The first challenge is perhaps the most difficult and critical step in designing PNS-based interfaces: A human peripheral nerve is an enclosed, cable-like bundle of myelinated nerve fibers and fascicles, structured and isolated by sheaths. Spike trains originated from primary motor cortex (M1) and encoding movement intents are propagated along axons of motor neurons that are further wrapped deep inside nerve fibers. Due to these multiple layers of lamination around an axon, the amplitude of a peripheral nerve signal is usually very small, can be around 5–20 μV [80]. PNS signals can also be contaminated by noises generated from muscle contractions and motion artifacts. Hence it is extremely difficult to record high-fidelity PNS signals with high signal-to-noise ratios (SNRs).

Regarding the second challenge, the development of PNS-based decoding for prostheses control has been closely following the decoding procedures commonly used for central nerves system (CNS) based prostheses [81, 82, 83, 84]. Intracortically recorded signals with penetrating electrodes or microelectrode arrays (MEA) are usually *multi-unit*, meaning that the recorded signals from one electrode are the mixture of the activities of several nearby neurons plus background noise. However, it is oftentimes desired to obtain *single-unit* electrical activities of individual neurons, which permits the analysis of human cognition and cortical mapping, and can also be applied to brain-machine interfaces (BMI) for brain control of prosthetic devices [85]. The process of obtaining single-unit activities of individual neurons from multi-unit recordings is called *spike sorting*, which typically consists of the following steps: filtering, spike detection and alignment, dimensionality reduction, feature extraction, and classification [86]. With spike sorting, we can estimate the spike train of each individual neuron, which is further used to compute *firing rate*, i.e., the frequency of a neuron "firing" spikes, that has been commonly used in BMI systems for patients with tetraplegia to predict kinematics of prosthetic devices via Kalman filter or multivariate linear regression [87, 88, 89]. From the perspective of ML, the firing rates of sorted neurons are treated as features engineered from the original PNS recordings, which are essentially the same as the EMG features listed in Table 1.1.

Despite the made progress, it is questionable whether following the same decoding procedures as in CNS-based decoding is the best practice for feature extraction in PNS-based decoding. One primary concern stems from the observation that the characteristics of the CNS recording and PNS recording differ drastically from each other in terms of noise level, electrode configuration, spatial resolution, recording selectivity, etc., all of which can impact the quality and morphology of recorded signals, and hence the choice of feature extraction approaches. For example, the amplitude of spikes from CNS recordings can be as large as several hundred microvolts to a few millivolts, whereas the amplitude of recorded PNS signals is much weaker. Furthermore, unlike CNS recordings where successful capture of action potentials fired by individual neurons is not unusual, it is extremely difficult to record single fiber action potentials in PNS recording with epineural or even intraneural electrodes. Signals recorded from PNS are either compound action potentials, a synchronized response generated by a group of electrically activated nerve fibers [90], or simply the superposition of many extracellular potential fields generated by single fiber action potentials [91]. Either way, the composition of "spikes" in PNS recording differs significantly from that in CNS recording, which could render the classic decoding procedure in CNS based on spike sorting and single-unit firing rate estimation not only ineffective but fundamentally invalid.

Another major concern is about the neural code for motor control, i.e., how neural activities (sequence of spikes) control muscles. It has been found in several neurophysiological and computational studies that the motor cortex exhibits neuronal redundancy, which refers to the facts that the number of motor cortex neurons far exceeds the number of muscles, and that many different combinations of neural activities can generate identical muscle movements [92, 93, 94]. In practice, this provides relaxation for CNS-based decoding over the choice of electrode placement and the amount of motor cortex neurons to be monitored. Furthermore, as firing rate is essentially a type of rate coding, it is not sensitive to occasional missing or false-alarmed spike detections. In comparison, each skeletal muscle fiber is innervated by a single motor axon and the same motor axon can also innervate other muscle fibers, leaving no space for redundancy on motor neurons innervating muscles. In addition, recent research studies have shown that the precise timing of spikes on the resolution of millisecond (i.e., timing code), rather than just the rate, plays a crucial role in predicting and causally controlling motion behaviors [95, 96]. These findings indicate that compared to CNS-based decoding, PNS-based decoding has more stringent requirements on the quality of recording as well as the accuracy of spike detection, if we rely on spike trains for feature extraction. However, developing recording electronics capable of high-fidelity nerve signal acquisition is extremely difficult due to the tight constraints on the power consumption and size of the electronics, the weak signal amplitude on the surface of nerves (intraneural electrodes are invasive thus not preferred), the heterogeneous types of noises that can be picked up by the electrodes and the electronics, including bioelectric noise, chemoelectric noise, motion artefacts, electromagnetic pickup of radio-frequency sources, and electrical power lines, etc. On the one hand, alternative feature extraction approaches that are less dependent on spike train analysis are in great demand to push forward the effectiveness and practicality of PNS-based decoding for prostheses control. It would be sensible and worthwhile to explore data-driven representation learning to extract features directly from recorded nerve signals and completely avoid spike detection and timing or rate estimation, albeit still facing the difficulty of obtaining ground truth data. On the other hand, more in-depth theoretical investigation and experimental exploration of the mechanism of the neural code for motor control can significantly contribute to the design of feature extraction frameworks for PNS-based decoding by introducing more biologically plausible inductive biases to relax the demand of representation learning for large amounts of training data and improve its generalization capability.

1.2.3 Predictive Analytics in Healthcare

EHRs have provided a rich source for researchers from various backgrounds (statistics, computer science, health informatics, etc.) to study phenotypes and treatment of patients and reveal unnoticed disease correlations, which can ultimately improve the quality of patient care, enable cost-effectiveness, and reduce readmission and mortality rates [97, 98]. In clinical studies, major efforts have been made to leverage vast amounts of EHR data as real-world evidence (RWE) to complement the knowledge gained from traditional randomized controlled trials (RCTs) and improve regulatory decision making [99, 100, 101, 102]. A number of research topics based on EHR have been developed and under active studies in ML communities, including information extraction [103, 104, 105, 106], representation learning [107, 108, 109, 110, 111, 112], outcome prediction [113, 114, 115, 116, 117], computational phenotyping [118, 119, 120, 121, 122], and more. A search result on Google Scholar for studies that include topics of EHR and ML indicates that the annual number of publications relating to ML in EHR has increased by 8-fold from 2012 to 2017 [26]. In fact, mining useful medical or clinical insights from related literatures using automated data mining/ML approaches has become an active research direction, thanks to the booming publications in EHR related studies [123, 124, 125]. In this section, we discuss key aspects about ML in EHRs that are closely related to representation learning.

Word embedding In the 10th version of the International Statistical Classification of Disease and Related Health Problems (ICD-10), there are more than 70,000 codes for diseases, signs and symptoms, abnormal findings, complaints, social circumstances, and external causes of injury or diseases [126]. Now consider the following situation: We want to predict the diagnostic result of a certain disease based on a patient's medical history that typically consists of tens to hundreds of different ICD codes. If we simply use multi-hot encoding, the feature vector that indicates the patient's medical history would be very high-dimensional (over 70,000) and extremely sparse (over 99% of the dimensions are zero). This would result in severely inefficient computations as well as the inability to capture the semantic relations between any medically related codes.

Here the technique of *word embedding* comes to the rescue by transforming sparse, high-dimensional features into dense and low-dimensional representations. Word embedding refers to the collection of a set of language modeling and feature learning in *natural language processing* (NLP), a major subfield of computer science and and artificial intelligence focusing on how to program computers to process and analyze large amounts of natural language data. The motivation of word embedding is illustrated in Figure 1.3, which shows that after embedding, the geometric distance between words should be consistent with their semantic relations. This further suggests that we can perform arithmetic computation on

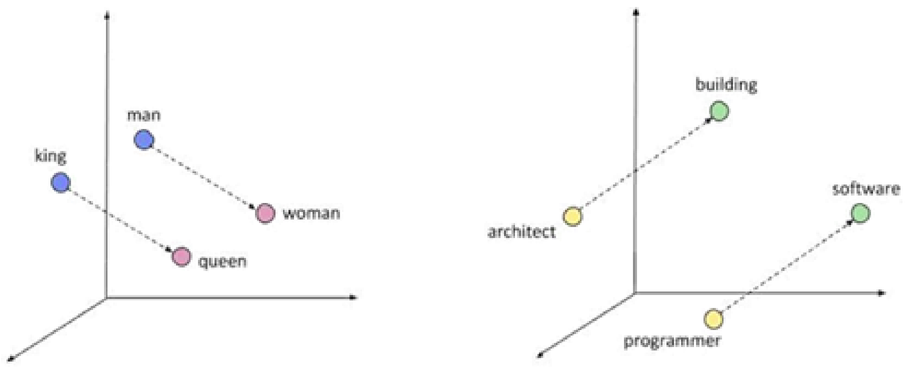


Figure 1.3: Illustration of the motivation of word embedding. Figure courtesy of [7]

the embedding vectors of words that are also semantically plausible. For example, as indicated in Figure 1.3, $Embed(king) - Embed(queen) \approx Embed(man) - Embed(woman)$, and $Embed(architect) - Embed(building) \approx Embed(programmer) - Embed(software)$.

In its original form (`Word2Vec` [127, 128]), the word embedding algorithm first randomly initializes the embedding vectors of all the words in the text inputs, then optimizes the embedding of each word by maximizing the accuracy of predicting the neighbors of the word in the text inputs, where the probabilistic distributions on the predicted neighbors are computed using the estimated embedding vectors (usually through a `softmax` function). The underlying idea is straightforward: If two words frequently appear together, then they should also be close to each other in the embedding space. Though effective, `Word2Vec` suffers from a shortcoming that it only considers the local information (i.e. neighbors) of each word, and completely ignores the global information of each word relative to all other words in the dictionary, which could lead to issues such as misinterpretation and ambiguity. To resolve this issue, `GloVe` leverages both local and global statistics of a text corpus through global matrix factorization and local context windowing, and can produce a vector space with meaningful substructure and improved performance on a word analogy task [129]. Another work that was based on and improved upon `Word2Vec` is `FastText`, which proposed a different scoring function than `Word2Vec` when computing the distance between two embedding vectors by treating each word as a bag of character n-gram, i.e. a subword model [130]. In doing so, `FastText` can learn embedding vectors for words that are rare or absent in the training data.

Generalized language model So far all the introduced word embeddings are context-independent – they are learned from word concurrency but not actual context, and each word has a fixed embedding vector. This can be problematic as the same word can have drastically different interpretations depending on its context. For example, in the two sentences “I have an Apple phone” and “my phone is placed next to an apple ”, the word “apple” clearly refers to two different things but would share the same embedding vector. To address this limitation, generalized language models are proposed in recent years that allow for *contextualized embedding* of words by passing sentences either uni-directionally or bidirectionally through a sequential model that normally consists of either multilayer long short-term memories (LSTMs) [131] or Transformers [132]. Recent developments of generalized language models include `CoVe` [133], `ELMo` [134], `ULMFiT` [135], `GPT` [136], `GPT-2` [137], `BERT` [138], and more.

One new era in NLP opened up by the recent generalized language models is the unsupervised pre-training of large-volume models (containing millions to even billions of parameters) on text corpora. Such pre-trained language models have shown unprecedented performance in many NLP tasks, such as question answering, commonsense reasoning, natural language inference, named entity recognition, sentiment analysis, sentence similarity, and many more. For example, `BERT` was pre-trained using the BookCorpus (800M words) [139] and the English Wikipedia (2,500M words), and contains 340M parameters. At the time of release, `BERT` broke several records of NLP benchmarks. The success of `BERT` is primarily attributed to the following two features: First, `BERT` is built completely upon the Transformer architecture, which solely relies on the self-attention mechanism instead of more conventional sequential modeling (e.g. LSTMs) to learn contextual relations between words. In this sense, it is more accurate to say that `BERT` is non-directional rather than bidirectional. Second, `BERT` is trained in a “fill-in-the-blank” way by randomly masking words in sentences and trying to predict them. In this way, `BERT` can take both the previous and next words surrounding the masked words into account *simultaneously*, as opposed to other bidirectional language models (e.g. `ELMo`) that can only do either left-to-right or right-to-left modeling, but not both at the same time.

As a result of the superior capabilities of `BERT` in extracting contextual relations between words (or phrases) form large text corpus, researchers from biomedical and healthcare informatics have been inspired to pre-train `BERT` on large-scale biomedical corpora and use it as a domain-specific language presentation model, and have achieved state-of-the-art performance on a number of biomedical text mining tasks such as biomedical named entity recognition, biomedical relation extraction, and biomedical question answering [140, 141, 142, 143, 144].

Graph embedding In discrete mathematics, graph is a structure that uses vertices (or nodes) and edges (or links) to represent objects and their relations, respectively. Real-world data often exhibit non-Euclidean, irregular organizational structures that are suitable for graph-based representation. For example, a molecule can be represented as a small, sparse, and static graph, whereas a social network could be represented by a large, dense, and dynamic graph. These real-world graphs represent important and rich information which cannot be fully captured by individual entities alone. Therefore, representation learning of nodes in graphs that aims to extract high-level features from a node as well as its neighbors has proved extremely valuable for many tasks, such as node classification and link prediction that can find a wide range of real-world applications [145].

A major branch of graph embedding algorithms is based on random walks, including `DeepWalk` [146], `node2vec` [147], and others. To obtain embedding vectors of nodes, random walks are performed for each node in a graph to explore its neighborhood structure and generate node sequences which are analogous to sentences that consist of words. Then word embedding techniques can be applied to node sequences to derive node embeddings that encode their neighborhood information. There are different strategies to guide random walks on graphs: some strike to balance the depth and width of neighborhood search (e.g., `node2vec`), and some focus on visiting nodes with different properties in pre-defined orders to encode node type heterogeneity (e.g., `metapath2vec` [148]). Similar to the random-walk based methods, there are other graph embedding algorithms that strive to preserve the first (one-hop) and second (two-hop) order neighborhood structures of nodes in the embedded space, such as `LINE` [149] and `PTE` [150]. Though they are not based on random walks, the optimization process to derive embedding vectors are essentially the same as `DeepWalk` and `node2vec` by using network training tricks from `Word2Vec` such as negative sampling [127].

One common issue of the above graph embedding methods is that they are all inherently *transductive* and do not naturally generalize to unseen nodes. They focus on embedding nodes from a fixed graph; however, many real-world applications require generating embeddings quickly for newly added nodes or new subgraphs, which is essential for high-throughput, production machine learning systems. Hence, an *inductive* learning paradigm is highly desired for graph embedding to generate embeddings for new nodes without an exhaustive optimization of all nodes from scratch. The first work that addresses this issue by leveraging the advanced generalization capability of neural network is `GraphSAGE` [151], which learns a set of aggregator functions to aggregate feature information from a node's local neighborhood. At test or inference phase, we can use the trained aggregator functions to generate embeddings for entirely unseen nodes based on the features of the new nodes' neighbors in the existing graph. `GraphSAGE` allows several types of aggregation, including mean, LSTM, and pooling through a fully connected neural network. In a similar fashion, Graph Attention Network (`GAT`) learns a set of aggregator functions based on the self-attention mechanism that allows for variable sized inputs, as opposed to `GraphSAGE` that only searches for a fixed number of neighbors of a node.

Patients, doctors, and medical code embedding Let us revisit the scenario discussed at the beginning of this subsection: We are about to predict the diagnostic results for patients from their history medical records that consist of hundreds to thousands of medical codes that represent different diagnoses, prescriptions, and procedures. With the help of the embedding techniques introduced above, we could represent each original medical code (e.g., from ICD-10) as a low-dimensional dense vector, upon which we could do not only diagnosis prediction, but also many other important medical data mining tasks such as hospital readmission prediction, physician targeting, disease phenotype extraction, and many more. It is then clear to us that as the first step of many healthcare informatics tasks, representation learning is critical that turns heterogeneous medical records into structured and actionable information, therefore has received increasing attentions in recent years from researchers in the area of medical data mining.

Medical records data are organized in a hierarchical, multilevel structure, which can be leveraged to improve learning efficiency, interpretability, and prediction accuracy. The hierarchical structure starts from patients, followed by visits, then diagnosis codes within visits, which are further linked to treatment codes including prescriptions and procedures. For a patient, visits happen in a sequential order; each level of the hierarchy is also linked to a doctor with a specific primary specialty. All the doctors can be grouped as a set, which is linked to the set of patients as a bipartite graph, where each edge indicates interactions between the corresponding patient and the doctor.

Given such a complex topology and the vast quantity of patient medical records, there is a rich amount of information and knowledge that can be uncovered and leveraged for various applications. For medical code embedding, Choi *et al.* proposed `MiME` [152], a multilevel medical embedding scheme of EHR that leveraged the *visits-diagnoses-treatments* hierarchy as auxiliary prediction tasks to enhance performance of predictive healthcare applications such as heart failure prediction and sequential disease prediction. In a follow-up work [153], Choi *et al.* proposed Graph Convolutional Transformer (`GCT`) to address the situation when the complete hierarchical structure information is not or partially available by utilizing the implicit structure of EHR. Patient embeddings have also found important applications in prediction of medical events and next visit time from their medical codes [154, 155, 156, 157, 158]. Doctor embeddings, in comparison, are less studied than medical code and patient embeddings, and have started to attract research attentions recently because of its importance in clinical trial recruitment to identify the right doctors to help conduct the trials based on trial description and patient medical records [159].

1.3 Summary

In this chapter, we first discussed the impact of the growth of big data in neuroscience and healthcare studies. To properly handle the complexity of big data and the challenge it brings, we discussed the solution of adopting ML approaches and their wide applications in neuroscience and healthcare researches, with a special focus on representation learning regarding its core position in ML-based data processing pipelines as well as its significant impact on the overall performance in various tasks. In greater details, we analyzed the encountered challenges and existing solutions for two specific areas: neural signal processing and predictive healthcare analytics.

For neural signal processing, we focused on the discussion of decoding from recorded neural signals for prostheses control. We compared the pros and cons of CNS-based decoding and PNS-based decoding, and discussed their technical difficulties from the perspective of representation learning, such as spike sorting and the estimation of neural codes, and how techniques from representation learning in ML can be leveraged to gain more accurate and interpretable insights of the neural decoding process and also improve the performance of prostheses control.

For predictive healthcare analytics, we started from word embedding, the most widely adopted representation learning methods in NLP, by explaining its mechanism and demonstrating its critical importance in various biomedical data mining tasks. We then moved to the discussion of contextualized embedding using generalized language models pre-trained on large text corpora that overcome the "fixed embedding regardless of contexts" issue of word embedding techniques. We further introduced graph embedding, an effective graph-based representation learning framework, and emphasized on how graph embedding techniques can model the multilevel, hierarchical structure of patient medical records.

As the main contributions of this dissertation, we (me and other collaborators in each work) present the design of several representation learning algorithms and their applications in three carefully chosen topics: neural data compression, spike sorting, and medical entity embedding in EHR. In each topic, we show that how the proposed representation learning algorithm can effectively address the "pain point" unique to the specific application without losing the trait of being generalizable and applicable to similar problems, which echoes with the phrase in the title "a practitioner's perspective". It is therefore our utmost aspiration that researchers especially those in the areas of neural signal processing and healthcare informatics can benefit from our works by using our proposed algorithms as tools in their own studies, or developing more advanced and effective methods based on our inventions.

The rest of the dissertation is organized as follows:

- In Chapter 2 we present two unsupervised neural data compression algorithms that leverage representation learning methods from ML to address the challenge of real-time compression of large-scale neural data for wireless recording.

- In Chapter 3, we present a semi-supervised, or "few-shot" spike sorting algorithm to obtain single-unit activities from both synthetic dataset (with ground truth information) and real neural data from *in-vitro* recordings.

- In Chapter 4, we present a medical entity embedding algorithm (`ME2Vec`) to encode all major components in EHR data: patients, doctors, and medical services. `ME2Vec` is intended to serve as a general-purpose representation learning solution for EHR data to facilitate both accurate and interpretable predictions.

- Chapter 5 concludes the dissertation.

Chapter 2

Unsupervised Representation Learning for Neural Data Compression

2.1 Background

Understanding the coordinated activity underlying brain computations requires large-scale, simultaneous electrophysiological recordings from distributed neuronal structures at a cellular-level resolution. There is a recent trend to develop high-density neural interfaces that include tens of thousands and even hundreds of thousands channels [160]. For example, multiple studies have been proposed that developed high-channel-count, high-precision neural recorders [161, 162] and high-density microelectrode arrays [11, 163]. Given the successful development of high-density arrays, it requires streaming the data to a remote computer for processing, which can be challenging: large-scale recording experiments can produce data at tens of hundreds of Gbps [10], which would require hundreds to thousands of I/O pads on the recorder chip and power consumption on the order of Watts for data streaming alone. To solve the problem, it requires to compress the neural data in the recorder chip before transmission, and reconstruct the data or directly utilize the compressed signals on the remote computer.

Extracellularly recorded neural data are mainly comprised of two types of signals – action potentials (or "spikes") and local field potentials (LFPs). The functional role of spikes in neural coding of information representation and propagation in multiple brain regions has been studied for decades. For example, in [164], spiking activities recorded from human motor cortex are used to develop assistive devices to improve lost functions in patients with paralysis. LFPs are also widely used to study cortical network mechanisms involved in sensory processing, motor planning, higher cognitive processing, etc. [165, 166, 167]. Compared with scalp electroencephalography (EEG) signals, LFPs have better signal-to-noise ratios and cover a wider frequency bandwidth; compared with spikes, LFPs have better chronic stability, but do not report single-unit activities. As a result, both spikes and LFPs are popular choices in neuroscience experiments and clinical treatments.

In this chapter, we present two neural data compression algorithms distinct features and thus different emphases on functionalities and applications. The first algorithm is based on streaming PCA, and can compress both spikes and LFPs simultaneously for real-time multichannel recording and data transmission [2, 168]. The second algorithm is based on autoencoder and neural network, and is dedicated for high-quality spike compression that pushes the rate-distortion trade-off to extreme [3, 169]. Both of the algorithms are unsupervised, meaning that users need nothing more than the neural data to be compressed to run the algorithms. In addition to the algorithmic details, we discuss the design (or feasibility) of on-chip hardware implementation of the algorithms, which is essential for real-time and large-scale neural recording and signal processing.

2.2 Related Works

2.2.1 Spike Compression

In the literatures of neural signal processing, spikes and LFPs are often separately processed. Spike compression is closely related to spike sorting, which is an important procedure to isolate single-unit activity from the large amount of raw data [170]. Spikes are obtained through detection, a crucial step in spike sorting that can offer significant data rate reduction [171, 172, 173, 174, 175, 176]. After spike sorting, neural activities are coded with a few low-rate binary streams that can be easily transferred to computing devices for further processing. Hence it is advantageous to implement part of spike sorting into the recording chip for data rate reduction [177, 178, 179]. An important issue in data compression is the trade-off between data compression ratio (CR) and data distortion. Highest CR is achieved by treating spikes as binary trains while waveforms are not transmitted. However, in applications where informations of spike shapes are useful, it is crucial to compress spikes with minimum signal distortion. Feature extraction can be used for this purpose [180, 181, 182]. Principal component analysis (PCA) is a popularly used feature extraction algorithm to compress spikes by facilitating dimensionality reduction with low signal distortion [183, 184, 185, 186, 187, 188, 189]. In addition to PCA, compressive sensing and wavelet methods are also widely used in compressing spikes [190, 191, 192, 193, 194, 195, 196, 197, 198].

Another type of approaches that can potentially boost CR is learning-based compression, such as vector quantization (VQ), where a signal-dependent codebook is learned from data and only the indexes of individual codeword in the codebook are transmitted [199]. This is different from conventional compression techniques that exploit efficient coding of the bit patterns of data after quantization; rather, it seeks "distilled" representations of the information content of signals, which could be more advantageous to achieving higher CR for data subject to certain statistical distributions. However, distributions of real-world data are usually in high-dimensional space, and are difficult or even intractable to estimate analytically. Furthermore, learned codebooks often "overfit" training data and do not generalize well, which leads to the requirement of frequent re-training and transmitting the entire codebook or uncompressed data that may interrupt data transmission and deteriorate CR. To make the learning-based compression approach effective and practical in large-scale neural recording, we need to address the following issues: (i) The size of the codebook cannot grow arbitrarily large to maintain good signal reconstruction accuracy in situations of low SNRs and/or diverse spike waveforms; (ii) The codebook must represent inherent spike features, such that the compression algorithm can be robust to non-stationarity of neural activities, e.g., waveform variations. These requirements entail the search of the "optimal codebook" that best characterizes the statistics of spikes by sampling a reasonable amount of spikes in a low-dimensional feature space.

2.2.2 LFP Compression

In comparison, LFP compression is not as extensively studied as spike compression. This is due to the low bandwidth requirement of LFPs for recording systems with relatively low channel counts. However, with the fast advancement of high-density microelectrodes and large-scale recording systems, the bandwidth required for transmitting LFPs is linearly scaling with channel count and can easily reach several Mbps for systems with thousands of recording channels. Current low-power wireless technology can only transmit raw data on a small amount of LFP channels, which is insufficient to support data transmission from a large number of channels and simultaneous recording from multiple brain regions and structures. Given the importance of LFPs in neuroscience studies and clinical applications, we need a method that can effectively compress multichannel LFPs and is also suitable for hardware implementation.

2.3 Algorithm 1: Streaming PCA

2.3.1 Overview

We aim to develop a data compression method that can compress both spikes and LFPs with low signal distortions, support simultaneous processing of multichannel data, and consume as few hardware resources as possible. This will allow for an unified and scalable hardware architecture for neural data compression, which is beneficial to efficient chip implementation. However, spikes are transient and localized [200], while LFPs are widespread in extracellular medium, for example, 600$\sim$1000μm parallel [201] and centimeter scales ventral to the cortical surface [202]. The spatial propagation and variation of LFPs recorded using a 32-channel microelectrode are demonstrated in Figure 2.1. The distinctions between spikes and LFPs in both temporal and spatial scales make it challenging to compress them using the same computational structure.

Given the strong correlations of LFP signals across multiple channels, a straightforward way for LFP compression/information extraction is selecting a few number of channels without going for any compression schemes. Although attractive for its simplicity, this practice suffers two main concerns: First, it is questionable if a small portions of channels can fully represent the high dynamics of LFP activities; second, it bears the risk of loss of useful information when the selected channels are corrupted by artifacts/interferences due to electrode drift or other environmental changes. Furthermore, it is difficult to ensure that the selected channels are optimal for feature extractions in decoding experiments.

To solve the challenges, we propose a streaming PCA algorithm as well as its microchip implementation to compress both multichannel spikes and LFPs. We extend the original PCA algorithm by exploiting the commonality between temporal correlation of spikes and

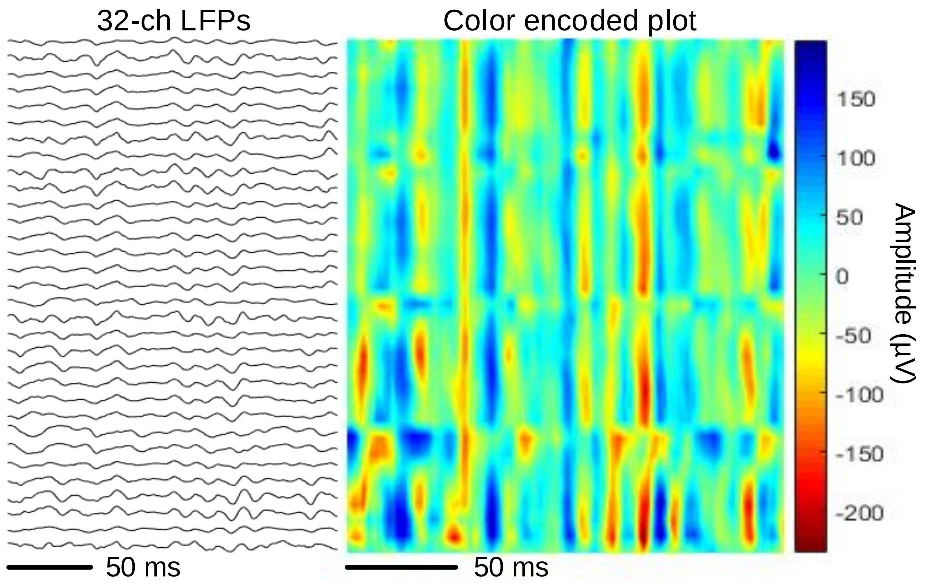


Figure 2.1: Left: a segment of waveforms of 32-channel LFPs from an in-vivo preparation. Right: color encoded magnitude map that indicates a repetitive spatial pattern in every 8 adjacent channels. Reproduced from [2], ©2017 IEEE.

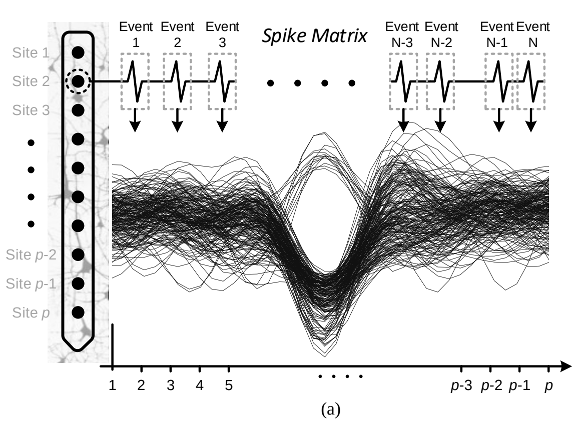
spatial correlation of LFPs. PCA is a batch method that can only adapt to new data by re-computing data covariance matrix through eigenvector decomposition. We refer to the original PCA as batch PCA hereafter. We circumvent this limitation by introducing stochastic approximation into batch PCA to incrementally update principal components without training an expensive covariance matrix. After the modification, the streaming PCA algorithm can process neural data in real-time at low computational cost. We implement the proposed algorithm into a microchip designed in a 65 nm CMOS technology, supporting up to 100 recording channels and occupying a silicon area of 240×260 μm^2.

2.3.2 Methods

Denote as $\mathbf{x}(t) = [x_{t,1}, x_{t,2}, ..., x_{t,p}]^T$ a vector of neural data sampled at time t, where p is the dimension of the data vector. With N data vectors sampled at time $t_1, t_2, ..., t_N$, we have a $p \times N$ data matrix $\mathbf{X} = [\mathbf{x}_1, \mathbf{x}_2, ..., \mathbf{x}_N]$. Our goal is to find k feature vectors for $\mathbf{X}$ ($k \ll p$), and by projecting $\mathbf{X}$ onto the k directions, the data dimension can be significantly reduced while data precision can be much preserved.

Schemes of PCA-based feature extraction for multichannel LFPs and spikes

Figure 2.2(a) shows the formulation of the $p \times N$ spike data matrix for PCA-based feature



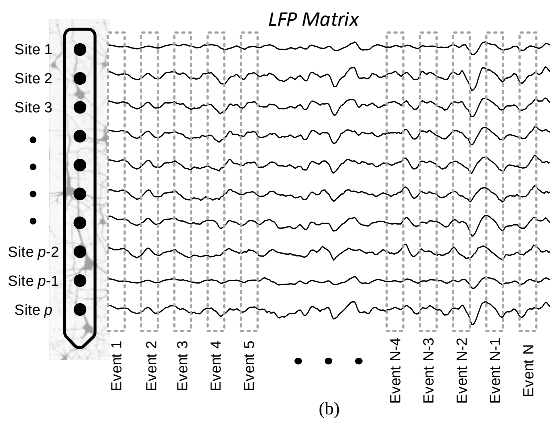


Figure 2.2: Schemes of PCA-based feature extraction for multichannel LFPs and spikes. (a) In spike compression, p is the number of samples per spike, N is the spike count. (b) In LFP compression, p is the number of recording electrodes, N is the number of cross-channel LFP vectors. Reproduced from [2], ©2017 IEEE.

extraction, where p is the number of samples per spike and N is the spike count. Spikes are collected from one channel. It has been shown that single neurons can be observed up to 50 μm away from the cell body [203]. Hence it is difficult for spike waveforms to propagate across multiple channels. One exception is for spikes recorded with closely spaced electrodes, such as tetrode, with a typical separation of 25–50 μm. For tetrodes, the cross-channel correlation is strong, yet limited to the four channels in a tetrode. In short, single-unit activity shows poor spatial correlations (if not considering network properties, e.g., neuronal ensemble synchrony [204]). On the other hand, as each neuron exhibits a distinct shape due to the differences in current path between the neuron and the electrode [205], single-unit activity has strong temporal correlations, which is beneficial to PCA-based feature extraction that seeks an ordered set of feature vectors capturing directions of largest variations in the data set.

The characteristic spatiotemporal profile of LFP data can be utilized in a similar fashion to facilitate PCA-based feature extraction. As shown in Figure 2.2(b), each column of the $p \times N$ data matrix is composed of LFP data sampled from all the electrodes in a probe at the same time. Grouping data samples from many electrodes into a single LFP vector is inspired by the idea of vector quantization, which is a high-dimensional generalization of scalar quantization and can be viewed as a form of pattern recognition to reveal efficient coding structure of data chunks [206]. In this way, the spatial correlation of LFPs as suggested in Figure 2.1 is inherently encapsulated in each cross-channel LFP vector, which is similar to single-unit activities demonstrating temporal correlations and hence suitable for PCA-based feature extraction. Another difficulty of applying PCA-based feature extraction to compress multichannel LFPs is how to determine the dimensions of the feature space, i.e., the choice of the values of k and p. Unlike in spike compression where it is more a trade-off between accuracy and compression ratio, in LFP compression it is also closely related to electrode geometries, which significantly affect the validity of LFP spatial correlations.

Mathematical formulation With p and N defined, we are looking for a linear transformation matrix $\mathbf{W}$ to achieve dimensionality reduction

$$\mathbf{Y} = \mathbf{W^T X}, \tag{2.1}$$

where $\mathbf{W}$ is a $p \times k$ linear transformation matrix with $k \ll p$, and $\mathbf{Y} = [\mathbf{y}_1, \mathbf{y}_2, ..., \mathbf{y}_N]$ represents k-dimensional neural data after compression. To maintain signal quality, the transformation matrix $\mathbf{W}$ should minimize the reconstruction error defined as the normalized squared distance

$$\arg\min_{\mathbf{W}} \left\{ \sum_{n=1}^{N} ||\mathbf{x}_n - \mathbf{W}\mathbf{W}^T\mathbf{x}_n||_2 / \sum_{n=1}^{N} ||\mathbf{x}_n|| \right\}. \tag{2.2}$$

It has been shown that the reconstruction error is minimized when the column variables in $\mathbf{W}$ are the first k eigenvectors of the covariance matrix of neural data. It is a formulation under batch PCA. The compression ratio is close to p/k.

Streaming PCA Batch PCA is unable to process streaming neural data. In this paper, we apply the expectation-maximization (EM)-PCA proposed in [207] to compress high-dimensional neural data. However, the EM algorithm presented in [207] requires multiple iterations over the same data set, which is difficult to process streaming data. Inspired from the modifications introduced in [208] on the general EM algorithm, we introduce a stepwise EM-PCA (SEM-PCA) to incrementally update parameters in the EM-PCA and make the method fully incremental and adaptive.

In EM-PCA, the linear transformation matrix that maps a latent variable to observed data is defined as

$$\mathbf{x} = \hat{\mathbf{W}}\mathbf{y} + \boldsymbol{\epsilon}, \tag{2.3}$$

where the k-dimensional compressed data $\mathbf{y}$ is a latent variable to the observed neural signal $\mathbf{x}$, and $\boldsymbol{\epsilon}$ is assumed an additive noise. In a Gaussian latent variable model, both $\mathbf{y}$ and $\boldsymbol{\epsilon}$

are assumed Gaussian, i.e., $\mathbf{y} \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$ and $\boldsymbol{\epsilon} \sim \mathcal{N}(\mathbf{0}, \sigma^2\mathbf{I})$. The conditional distribution of $\mathbf{x}$ over the latent variable $\mathbf{y}$ is given by

$$p(\mathbf{x}|\mathbf{y}) = \mathcal{N}(\mathbf{x}|\hat{\mathbf{W}}\mathbf{y}, \sigma^2\mathbf{I}). \tag{2.4}$$

The EM algorithm is applied to obtain the maximum likelihood estimation of $\hat{\mathbf{W}}$. To do that, the complete-data log-likelihood function is defined as

$$\ln p(\mathbf{X}, \mathbf{Y}|\hat{\mathbf{W}}, \sigma^2) = \sum_{n=1}^{N} \{\ln p(\mathbf{x}_n|\mathbf{y}_n) + \ln p(\mathbf{y}_n)\}. \tag{2.5}$$

Substitute equation (2.4) into (2.5), and take the expectation with respect to the posterior distribution over the latent variables

$$\begin{aligned}\mathbb{E}[\ln p(\mathbf{X}, \mathbf{Y}|\hat{\mathbf{W}}, \sigma^2)] = -\sum_{n=1}^{N}\Big\{&\frac{p}{2}\ln(2\pi\sigma^2) + \frac{1}{2}\mathbf{Tr}(\mathbb{E}[\mathbf{y}_n\mathbf{y}_n^T]) + \frac{1}{2\sigma^2}\,||\mathbf{x}_n||^2 \\ &- \frac{1}{\sigma^2}\,\mathbb{E}[\mathbf{y}_n]^T\hat{\mathbf{W}}^T\mathbf{x}_n + \frac{1}{2\sigma^2}\mathbf{Tr}(\mathbb{E}[\mathbf{y}_n\mathbf{y}_n^T]\hat{\mathbf{W}}^T\hat{\mathbf{W}})\Big\},\end{aligned} \tag{2.6}$$

where $\mathbb{E}[\mathbf{y}_n]$ is the only sufficient statistics. $\mathbb{E}[\mathbf{y}_n]$ can be derived from the conditional distribution of $\mathbf{y}_n$ over $\mathbf{x}_n$

$$p(\mathbf{y}_n|\mathbf{x}_n) = \mathcal{N}(\mathbf{y}_n|\hat{\mathbf{M}}^{-1}\mathbf{W}^T\mathbf{x}_n, \sigma^2\mathbf{M}^{-1}), \tag{2.7}$$

where

$$\mathbf{M} = \hat{\mathbf{W}}\hat{\mathbf{W}}^T + \sigma^2\mathbf{I}. \tag{2.8}$$

To approximate batch PCA, we let $\sigma^2 \to 0$, then the E-step in the EM algorithm becomes

$$\mathbb{E}[\mathbf{y}_n] = (\hat{\mathbf{W}}\hat{\mathbf{W}}^T)^{-1}\hat{\mathbf{W}}^T\mathbf{x}_n. \tag{2.9}$$

In the M-step we maximize equation (2.6) with respect to $\hat{\mathbf{W}}$

$$\hat{\mathbf{W}}_{new} = \left[\sum_{n=1}^{N}\mathbf{x}_n\,\mathbb{E}[\mathbf{y}_n]^T\right]\left[\sum_{n=1}^{N}\mathbb{E}[\mathbf{y}_n\mathbf{y}_n^T]\right]^{-1}. \tag{2.10}$$

In batch EM, the expectations of sufficient statistics are computed over the entire data set

$$\mathbf{S}_l = \frac{1}{N}\sum_{n=1}^{N}\mathrm{E}_{\theta_{l-1}}\left[s(\mathbf{x}_n, \mathbf{y}_n)|\mathbf{y}_n\right], \tag{2.11}$$

where $s(\cdot)$ is a sufficient statistics, $\mathbf{S}_l$ is the current collection of sufficient statistics, and θ_{l-1} represents the guess of unknown parameters from last iteration.

Algorithm 1 Streaming PCA

```
Require: Data dimension $p$, number of principal components $k$, current step $n$, input data
    sample $\mathbf{x}_n \in \mathbb{R}^p$, $\mathbf{S}_{n,1} \in \mathbb{R}^k$, $\mathbf{S}_{n,2} \in \mathbb{R}^p$, transformation matrix $\hat{\mathbf{W}} \in \mathbb{R}^{p\times k}$
Initialize: $\hat{\mathbf{W}} \leftarrow \texttt{rand}$, $\mathbf{S}_{n,1} \leftarrow \mathbf{0}$, $\mathbf{S}_{n,2} \leftarrow \mathbf{0}$
Ensure: Compressed data sample $\mathbf{y}_n \in \mathbb{R}^k$
1: $\eta_n \leftarrow n^{-\alpha}$
2: $\mathbf{y}_n \leftarrow \hat{\mathbf{W}}_{n-1}^T \mathbf{x}_n$
3: $\mathrm{E}_{\theta_{n-1}} \leftarrow (\hat{\mathbf{W}}_{n-1}^T \hat{\mathbf{W}}_{n-1})^{-1} \mathbf{y}_n$
4: $\mathbf{S}_{n,1} \leftarrow (1-\eta_n)\mathbf{S}_{n-1,1} + \eta_n \mathrm{E}_{\theta_{n-1}}[s(\mathbf{x}_n, \mathbf{y}_n)|\mathbf{y}_n]$
5: $\mathbf{S}_{n,2} \leftarrow (1-\eta_n)\mathbf{S}_{n-1,2} + \eta_n \mathbf{x}_n$
6: $\hat{\mathbf{W}}_n \leftarrow \mathbf{S}_{n,2}\mathbf{S}_{n,1}^T(\mathbf{S}_{n,1}\mathbf{S}_{n,1}^T)^{-1}$
7: return $\mathbf{y}_n$
```

In SEM-PCA, instead of calculating the expectation across all examples, only one data sample is used. To alleviate the poor approximation caused by using one sample, we interpolate between the current sufficient statistics and the inferred expectations by introducing a learning factor η, which can also accelerate parameter convergence. After including η, the parameter update becomes

$$\mathbf{S}_{l,n} = (1-\eta_n)\mathbf{S}_{l,n-1} + \eta_n \mathrm{E}_{\theta_{l,n-1}}[s(\mathbf{x}_n, \mathbf{y}_n)|\mathbf{y}_n]. \tag{2.12}$$

In this formation, we assume that data streams are infinitely long and the iteration index l thus stays unchanged and can be removed from (2.12).

Applying the stepwise EM to EM-PCA, we have SEM-PCA as stated in Algorithm 1. The learning factor should satisfy $\sum_n \eta_n = \infty$, $\sum_n \eta_n^2 < \infty$ to ensure the convergence of parameter estimation. Parameter α can be chosen arbitrarily within 0.6 to 0.9 [208]. The M-step is a natural expansion of (2.10), which is the same as in batch EM since the required computation is trivial in low dimensions ($k \ll p$). One variant of SEM-PCA is mini-batch, which updates on multiple observations in one time to improve computational stability at the expense of more storage requirement. Compared with EM-PCA, the proposed method is fully adaptive, does not require excessive on-chip memories, and can on-the-fly process data streams.

2.3.3 Low-Power Chip Implementation

We have implemented the streaming PCA algorithm into a microchip fabricated in a 65 nm CMOS technology. This design is part of a system technology for closed-loop neural

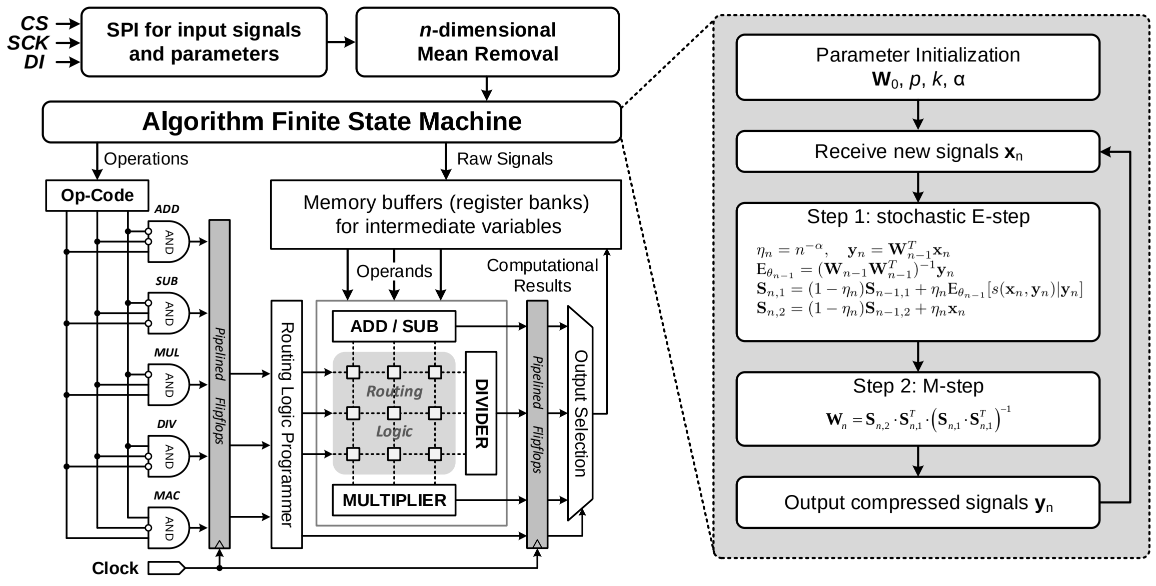

CS
SCK
DI
SPI for input signals and parameters
n-dimensional Mean Removal
Algorithm Finite State Machine
Operations
Raw Signals
Op-Code
ADD
SUB
MUL
DIV
MAC
AND
Pipelined Flipflops
Memory buffers (register banks) for intermediate variables
Operands
Computational Results
Routing Logic Programmer
ADD / SUB
Routing Logic
DIVIDER
MULTIPLIER
Output Selection
Clock
Parameter Initialization $\mathbf{W}_0, p, k, \alpha$
Receive new signals $\mathbf{x}_n$
Step 1: stochastic E-step
$\eta_n = n^{-\alpha}, \quad \mathbf{y}_n = \mathbf{W}_{n-1}^T \mathbf{x}_n$
$\mathrm{E}_{\theta_{n-1}} = (\mathbf{W}_{n-1}\mathbf{W}_{n-1}^T)^{-1}\mathbf{y}_n$
$\mathbf{S}_{n,1} = (1-\eta_n)\mathbf{S}_{n-1,1} + \eta_n \mathrm{E}_{\theta_{n-1}}[s(\mathbf{x}_n, \mathbf{y}_n)|\mathbf{y}_n]$
$\mathbf{S}_{n,2} = (1-\eta_n)\mathbf{S}_{n-1,2} + \eta_n \mathbf{x}_n$
Step 2: M-step
$\mathbf{W}_n = \mathbf{S}_{n,2} \cdot \mathbf{S}_{n,1}^T \cdot (\mathbf{S}_{n,1} \cdot \mathbf{S}_{n,1}^T)^{-1}$
Output compressed signals $\mathbf{y}_n$

Figure 2.3: System schematic of the streaming PCA chip. Reproduced from [2], ©2017 IEEE.

recording and electrical microstimulation [162]. The chip is able to process up to 100-channel LFPs simultaneously or multiple channels of spikes in real-time.

System architecture The system architecture of the streaming PCA chip is shown in Figure 2.3. The mapping of the SEM-PCA algorithm into a hardware architecture is accomplished by re-structuring the algorithm into a scheduling finite state machine (FSM), a computational unit that provides frequently used arithmetic operations, and memory buffers to store intermediate computational results. In addition, a serial peripheral interface (SPI) module is included for data communication with off-chip peripherals as well as parameter programming.

The scheduling FSM translates each step of SEM-PCA into commands of operations and operands. The operation commands are decoded into arithmetic computations through a decoding interface before execution. The arithmetic computations are implemented in floating-point representations to provide sufficient dynamic ranges and avoid overflow, and are interconnected through a routing logic that receives instructions from the decoding interface to assemble into requested computations. During the computations, operands are fetched from the memory buffers under the control of the scheduling FSM. Both the outputs of the decoding interface and the routing logic are pipelined to support a variety of

Sign	***6-Bit Exponent***	***12-Bit Normalized Fraction***
[18]	**[17:12]**	**[11:0]**

$$X = (-1)^{Sign} \cdot 2^{Exponent\text{-}bias} \cdot (1+Fraction),\ \ bias = 2^{6-1}-1 = 31$$

Figure 2.4: Format of the customized 19-bit floating-point representation. Reproduced from [2], ©2017 IEEE.

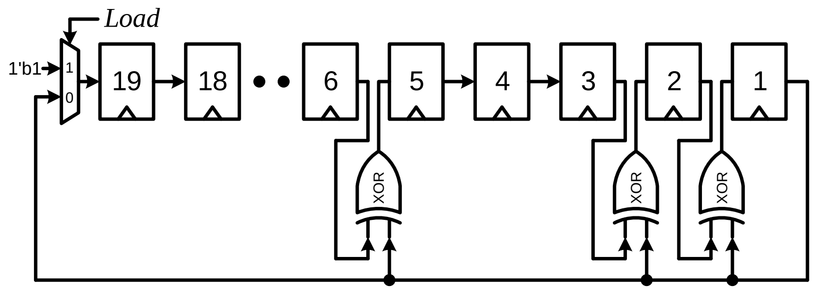
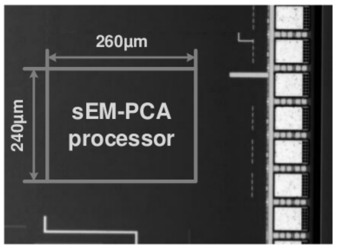


Figure 2.5: Structure of the 19-bit LFSR. Reproduced from [2], ©2017 IEEE.

arithmetic operations efficiently at a scalable speed, which also simplifies the synchronization with the memory buffers.

The memory buffers are realized in register banks which are flexible and easy to access at the cost of relatively large circuit area and high power dissipation. Major memory space is consumed by the $p \times k$ elements of the transformation matrix $\hat{\mathbf{W}}$, each of which is a customized 19-bit floating-point variable. The purposes of the 19-bit floating point structure are to ensure computational accuracy and avoid data overflow. We verify that each step of the computation on hardware is sufficiently close to the result obtained from a MATLAB version. When testing the algorithm implementation with various inputs, we found that floating-point structures less than 18-bit would occasionally result in data overflow or non-trivial deviations from software versions. This might be caused by the insufficient accuracy of data representation when performing matrix inversion. To fix this problem and leave some margins, we empirically determined the 19-bit floating-point structure. The detailed format of the floating-point representation is given in Figure 2.4. Dedicated memory technologies, such as SRAM, are to be explored in future developments to achieve higher area density and power efficiency of the memory buffers.

In this implementation, parameters k, p, and α are programmable through the SPI module. The initial randomization of $\hat{\mathbf{W}}$ is achieved by implementing a 19-bit linear-feedback shift register (LFSR). The structure of the LFSR is shown in Figure 2.5, which



(a)

Process	65nm CMOS
Chip Area	240x260µm^2
Voltage	0.5V
Clock Frequency	LFP: 1MHz Spike: 100kHz
Power Consumption	LFP: 144nW/ch Spike: 3.05µW/ch

(b)

	FSM	Computational Unit	Memory Buffer	MISC.
Power	6.4%	18.8%	59.1%	15.7%
Area	3.0%	12.7%	78.5%	5.8%

(c)

Figure 2.6: (a) Die photo of the streaming PCA microchip. (b) Chip specifications. (c) Power & area breakdown. Misc. includes clock trees, SPI module, reset circuits, and etc. Reproduced from [2], ©2017 IEEE.

is cost-effective and can provide needed randomness. Due to limited on-chip resources, the values of k and p are upper bounded at 4 and 100, respectively. It has been argued that the first two or three principal components ($k = 2$ or 3) is sufficient to accurately describe neural data [209]. The available choices of p and k can accommodate a wide range of experiment setups and recording configurations. These parameters are to balance storage consumptions, reconstruction accuracies, and compression ratios.

System prototyping As shown in Figure 2.6(a), the streaming PCA microchip occupies a silicon area of 240×260 μm^2 in a 65 nm CMOS technology. The nominal voltage of the process is 1.2 V. In order to achieve low-power operations favored by implantable biomedical electronics [162], we have re-characterized the digital synthesis library with a reduced supply voltage at 0.5 V to provide more accurate timing and power analysis [210].

The minimum working clock frequency of the system is determined as follows: In the worst case when k and p are 4 and 100, our system requires slightly less than 1000 clock cycles to process one spike or LFP vector. To process 100-channel LFPs with a base sampling rate of 1 kHz per channel, a 1 MHz system clock is needed. The measured peak power consumption in this case is 144 nW/channel. To process spikes at a firing rate of 100 Hz with 100 samples/spike, a 100 kHz clock frequency is needed, and the measured peak power consumption is 3.05 μW/channel. The significant difference on power consumption per channel of the chip processing LFPs and spikes results from the comparison that a LFP

Table 2.1: Comparisons between the proposed streaming PCA chip with previous works. Reproduced from [2], ©2017 IEEE.

Ref.	**Process**	**Signal type**	**Signal Dimension**	**No. of Channels**	**Area**	**Power**	**Methods**
[189]	90nm	Spike	64	1	0.255mm^2/ch	521μW/ch	PCA
[186]	350nm	Spike	16	N/A	1.77mm^2/ch	256.9μW/ch	PCA
[211]-1	90nm	Spike	64	32	0.083mm^2/ch	78.7μW/ch	GHA
[211]-2	90nm	Spike	64	64	0.08mm^2/ch	85.8μW/ch	GHA
[212]	90nm	Spike	64	32	0.021mm^2/ch	20μW/ch	Peak Search
[185]	130nm	Spike	64	1	0.268mm^2/ch	8.6μW/ch	SPIRIT
[176]	130nm	Spike	N/A	1	0.16mm^2/ch	200nW/ch	PWL
[181]	65nm	Spike	N/A	1	0.03mm^2/ch	8μW/ch	Pulse-based
Ours	65nm	Spike; LFP	Up to 100	LFP: 100 Spike: 1	LFP: 0.0006mm^2/ch Spike: 0.06mm^2/ch	LFP: 144nW/ch Spike: 3.05μW/ch	SEM-PCA

vector is constructed from multiple channels, while a spike comes from only one channel. Lowering the supply voltage from 1.2 V to 0.5 V reduces the overall power consumption by over 85% at an area overhead of merely 0.02 mm^2. The increased area is due to the resizing of digital gates in low-voltage synthesis to compensate for timing degenerations.

Figure 2.6 shows the chip die photo with the streaming PCA module highlighted. The memory buffers consume the major power and chip area, which calls for more advanced technologies to improve memory circuits. Comparison of the designed chip with previous works is given in Table 2.1. We can see that our design outperforms other feature extraction hardwares in many aspects. To our best knowledge, this work is so far the first hardware that applies PCA-based feature extraction to compress both LFPs and spikes with high area-power efficiency.

2.3.4 Experiments

We present the testing results of the streaming PCA chip on two data sets recorded from different in-vivo preparations. In one data set, neural data were recorded using a 4-shank, 32-channel NeuroNexus probe. The other data set was recorded using a 100-channel MicroProbes microelectrode array (MEA) organized in 10 lines. We also discuss a strategy to determine the dimensions of feature spaces when compressing multichannel LFPs, and how this strategy will lead to a better rate-distortion trade-off.

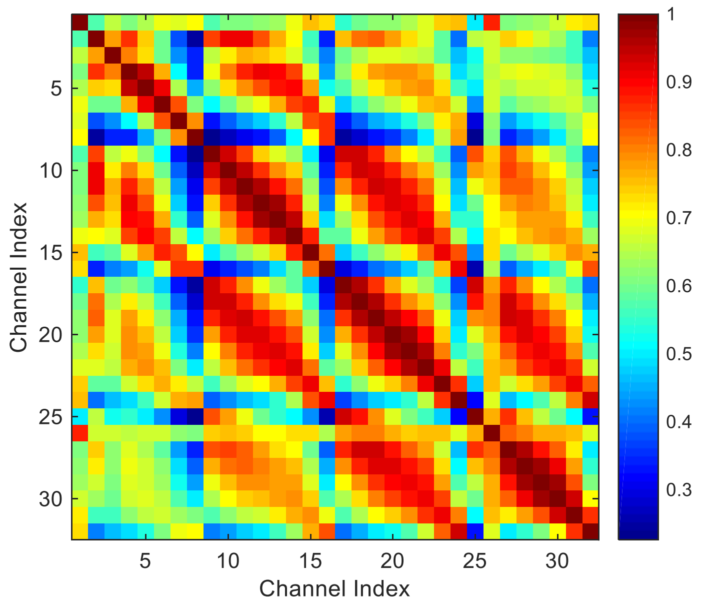


Figure 2.7: Inter-channel correlation map of 32-channel LFPs. Higher correlations are observed for channels in the same shank or adjacent shanks. Channels from two adjacent shanks correlate more if they are at the same vertical depth. Reproduced from [2], ©2017 IEEE.

Compression of LFP data recorded with a 4-shank, 32-channel NeuroNexus probe The recording experiments were conducted with a 4-shank, 32-site NeuroNexus probe with pitch size of 200 μm and shank-to-shank distance of 400 μm. The raw data were recorded with 24 kHz sampling rate and digitized with 16-bit precision. The recorded data were low-pass filtered and downsampled by 32$\times$ to remove high-frequency activities, leading to a sampling rate of 750 Hz per channel. There were 14 sessions of recordings with each session including 100 trials, resulting in 1400 trials in total.

We first run a correlation analysis of LFP amplitude between every electrode pairs from 32 channels and averaged over 1400 trials. This is to study the similarities among different channels and shanks, which helps to determine the number of latent variables (the value of k) in SEM-PCA. As shown in Figure 2.7, higher correlations appear if electrodes are situated on the same or adjacent shanks. Channels from two adjacent shanks correlate more if they are at the same vertical depth. As expected, correlation decreases with increasing distances between electrodes.

The number of latent variables can be updated in real-time by constraining how closely the energy of reconstructed signals tracks that of original signals. Nevertheless, we need to upper bound k due to limited on-chip hardware resources as the spatial complexity for $\hat{\mathbf{W}}$

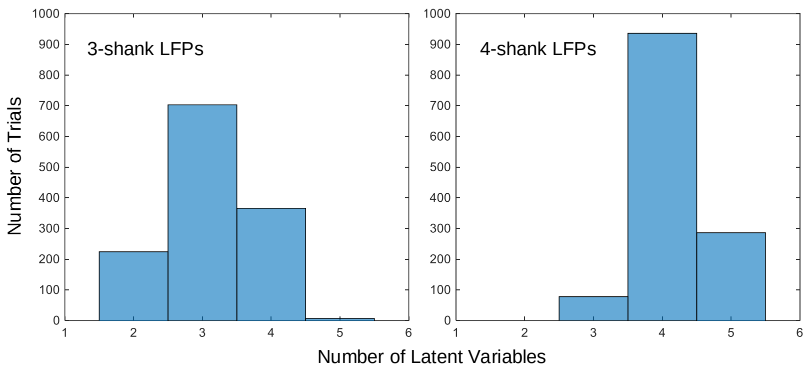


Figure 2.8: Number of trials corresponding to the needed number of latent variables to achieve $>$ 90% reconstruction accuracy. Left panel: using 3-shank LFPs; Right panel: using 4-shank LFPs. Reproduced from [2], ©2017 IEEE.

is $\mathcal{O}(kp)$. It has been shown that the maximum likelihood estimation of $\hat{\mathbf{W}}$ converges to eigenvectors in batch PCA multiplied with an arbitrary linear transformation [213]. As a result, the column vectors of $\hat{\mathbf{W}}$ can be non-orthogonal and not in any orders. From the experiment data, however, we find they have more close correspondence with the number of shanks and their locations. For example, we have tested the streaming PCA chip on LFPs recorded from 3 adjacent shanks compared with those from all 4 shanks. Figure 2.8 shows the distributions of k needed to achieve >90% reconstruction accuracies over all 1400 trials. For 3-shank and 4-shank recordings, k peaks at 3 and 4, respectively. We will discuss this empirical approximation of latent variables using the number of shanks in more detail. With $k = 4$ and $p = 32$, a nearly $8\times$ compression ratio of LFPs can be achieved for this 4-shank, 32-channel probe.

Next we examine the reconstruction accuracy of LFPs using the streaming PCA chip in both temporal and frequency domains. As shown in the top panel of Figure 2.9, the reconstruction errors of all 32 channels are less than 8%. The reconstruction in temporal domain is illustrated in the bottom panel of Figure 2.9. In frequency domain, we calculate the power spectrum density (PSD) of LFP sequences from all 1400 trials as well as their reconstructed versions, and plot their averages and spans into the top panel of Figure 2.10. The result suggests that the streaming PCA chip can enable accurate LFP reconstruction. The discrepancies are mostly situated in higher frequencies (>150 Hz), which also appear in the spectrograms in the bottom panel of Figure 2.10. The discrepancies are mainly caused by the nature of PCA-based method that tries to capture the largest variations of data using a few leading principal components, thus unable to represent accurately the relatively small

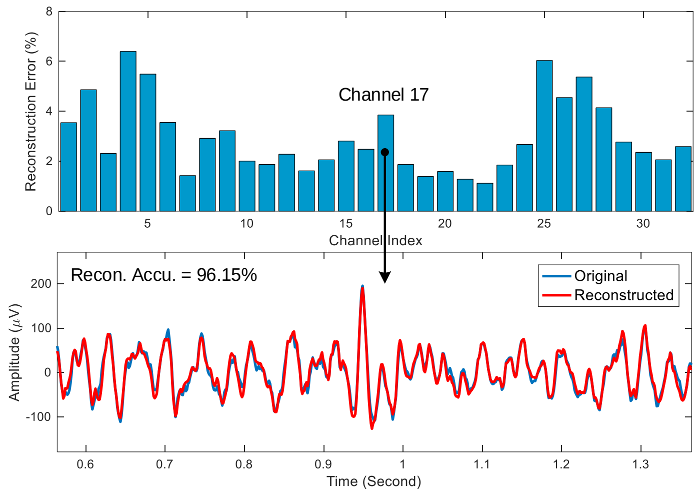


Figure 2.9: LFP reconstruction in time domain. The original and reconstructed LFP waveforms of Channel 17 are shown as an example. Reproduced from [2], ©2017 IEEE.

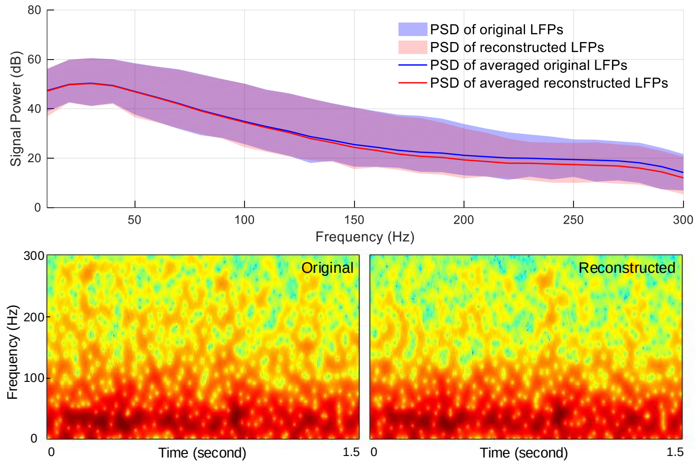


Figure 2.10: LFP reconstruction in frequency domain. Reproduced from [2], ©2017 IEEE.

fluctuations at the same time. One workaround is high-pass filtering of LFPs to attenuate the variations contributed by low-frequency contents, if high-frequency contents of LFPs are of more interest.

Compression of LFP Data Recorded With A 100-channel MicroProbes MEA

The 100 channels of the MircoProbes MEA are organized in 10 lines. The distance between two adjacent contacts in one line is 200μm. Intracortical neural signals were sampled at 12.5 kHz. There were in total 11 recording sessions. In each session, neural data were low-pass filtered at 300 Hz and downsampled by 16×. The sampling rate of LFPs was 780 Hz. Again we construct the correlation map of the 100-channel LFP data as shown in Figure

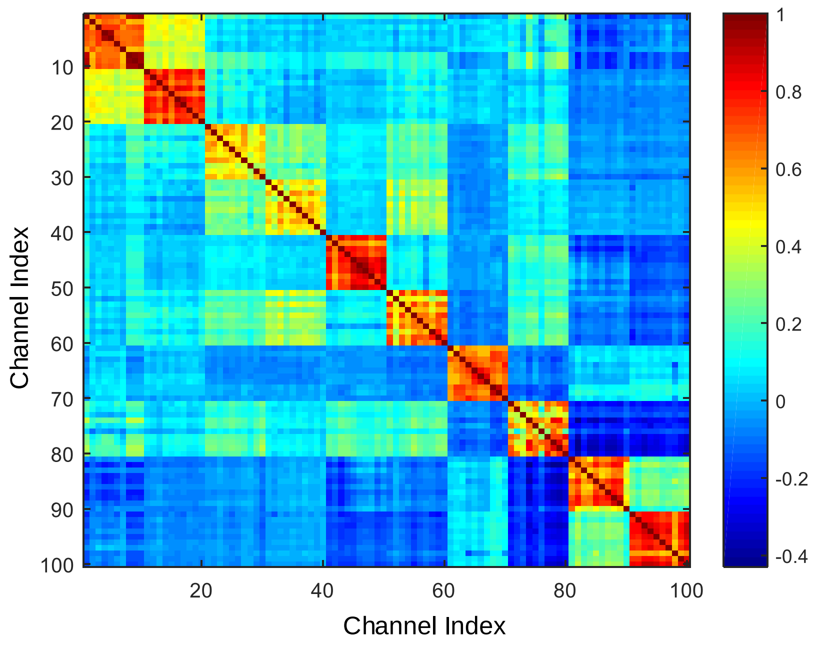


Figure 2.11: Inter-channel correlation map of 100-channel LFPs. The results are averaged over all 11 sessions. Channels from the same line correlate closely with each other; channels from different lines correlate poorly, or even reversely. Reproduced from [2], ©2017 IEEE.

2.11. It shows clearly that only channels from the same line correlate closely with each other. Channels from different lines correlate poorly, or even reversely.

Following the empirical approximation from last experiment that the number of latent variables should equal the number of shanks in a probe to ensure reconstruction accuracy, 10 latent variables are needed for a 10-line MEA. However, due to the limited on-chip resources, k is no more than 4 in the streaming PCA chip. With $k = 4$, p is limited to be no more than 40 for such a MEA with 10 contacts per line. Therefore, to simultaneously compress all the 100-channel LFP data from this MEA, 3 copies of the streaming PCA chip are needed as a multi-chip solution.

Next we validate the empirical approximation on this 100-channel MEA. To simplify the validation, we simulate the streaming PCA chip in Verilog HDL with equivalent functions except that k can be as large as 10. A RTL simulation also allows us to probe and monitor internal variables with much higher flexibility. Throughout the experiment, p is kept 10 times of k. For each recording session, we select neural data recorded from k out of 10 lines and compress the data using the simulated streaming PCA chip, with k varied from 1 to 10. The data selection in each session is repeated by $C_{10}^{k} = \frac{10!}{(10-k)!k!}$ times to allow different combinations of k lines of data. The data reconstruction accuracies corresponding to $k =$

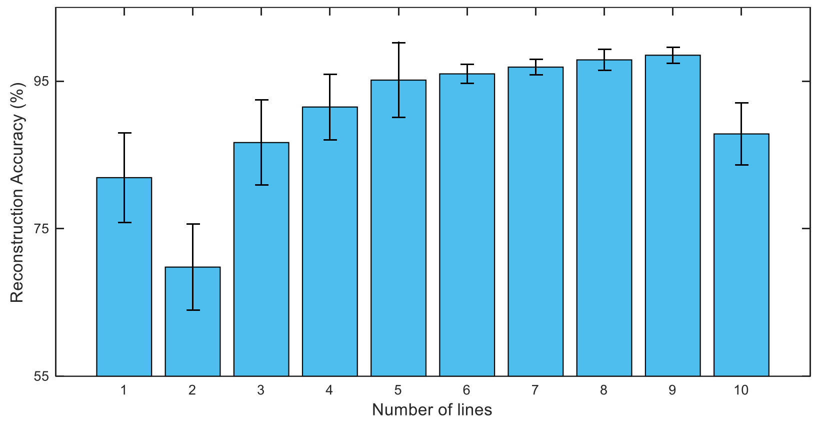


Figure 2.12: Reconstruction accuracies of LFPs with k varied from 1 to 10. X-axis denotes the number of lines (k) in the 100-channel MEA used in the simulation. Error bars denote the variations from C_{10}^{k} combinations of lines and 11 recording sessions. Reproduced from [2], ©2017 IEEE.

1 to 10 are shown in Figure 2.12. We can see that the empirical approximation of k works generally well, achieving $\sim$90% reconstruction accuracies except for $k = 2$. An interesting phenomenon in Figure 2.12 is the monotonically increasing accuracy from $k = 2$ to $k = 9$, suggesting an optimal range where the approximation applies. In general, the streaming PCA chip performs slightly worse on the 100-channel MEA than the 32-channel probe. This is because that the pitch size of the MEA (200 μm) is much larger than that of the probe (50 μm), resulting in poorer correlations of LFPs among adjacent electrodes. Again, this experiment illustrates the importance of taking electrode geometries into consideration for LFP compression. As p is always kept as 10 times of k regardless of the value of k, a nearly 10$\times$ compression ratio of LFPs can be achieved for the 10-line, 100-channel MicroProbes MEA.

Spike compression Unlike LFPs, spike compression is much less susceptible to electrode geometries. In addition to compression ratio and reconstruction accuracy, we are particularly interested in two aspects regarding spike compression. First, how fast the streaming PCA chip can converge in estimating $\hat{\mathbf{W}}$ when processing spikes from one channel. Convergence speed is measured in spike count. Second, how fast the streaming PCA chip can adapt to nonstationary spiking activities by updating $\hat{\mathbf{W}}$, for example, when switching the chip among multiple channels. By studying these problems, we can estimate the processing delay of the chip, which is an important parameter for applications that require fast responses. For example, in closed-loop neuromodulation, accurate control of individual neurons requires the information of single-unit activity, and the time delay between recording and data-dependent stimulation should be kept as small as possible. Therefore, it is helpful to evaluate the processing delay of the chip performing feature extraction in an effort to reduce the overall system time delay.

The convergence of the streaming PCA algorithm is closely related to the qualities of neural recording and spike detection. With a high threshold in spike detection, ambiguous spikes which are confused with background noise are much excluded. The remaining spikes tend to have large amplitudes and more regular shapes, possibly making it faster for the streaming PCA algorithm to converge in parameter training. However, high detection thresholds may not reduce the overall processing delay of both detection and feature extraction, because it takes longer time to detect large spikes that exceed high thresholds.

First we investigate the relation between detection threshold and convergence speed of the streaming PCA chip. As mentioned in Section III, this design is part of a system technology [162], which implements an exponent component–polynomial component (EC-PC) algorithm for spike detection [214, 215, 216, 217]. The EC-PC method features a probabilistic threshold which quantifies the likelihood of recorded signals being real spikes [218, 219]. Using the EC-PC algorithm, we detect spikes from all 11 sessions with a range of probabilistic thresholds, including 60%, 70%, 80%, 90%, 95%, and 100%. We process each group of spikes detected with the same threshold using the streaming PCA chip and study how many spikes are needed for the transformation matrix $\hat{\mathbf{W}}$ to converge. We define that convergence is met when the mean squared error of estimated principal components relative to batch PCA values stabilizes (for at least 3 consecutive instances). The spike counts needed for convergence for each detection threshold are averaged over 1100 trials (100 channels $\times$ 11 sessions). Figure 2.13 shows that the streaming PCA chip can converge within 200$\sim$250 spikes on average and with similar variations for a wide range of detection thresholds. Higher thresholds indeed require less spikes for convergence, but not much. For example, the streaming PCA chip converges faster for 100% threshold than 60% by less than 50 spikes. The small difference indicates that in processing spikes, convergence of feature extraction parameters is less sensitive to detection thresholds. It normally takes much longer time to detect 50 large spikes than to process the same amount of spikes by the

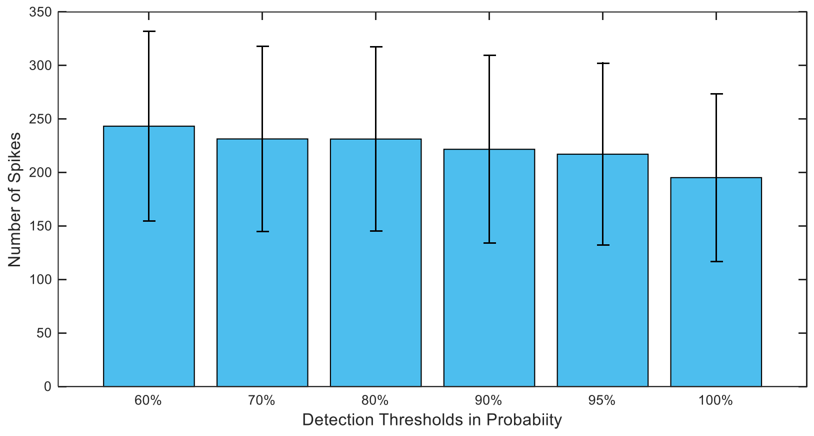
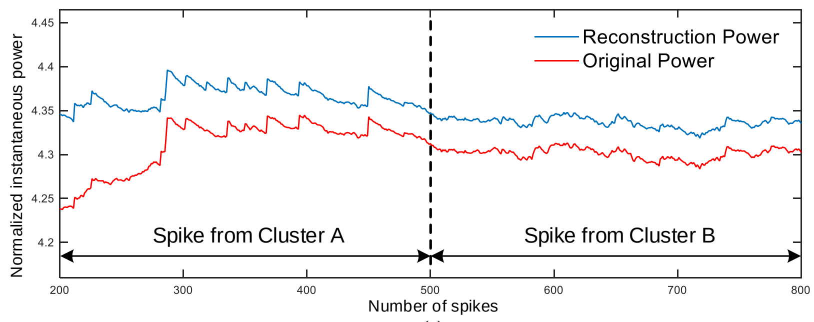


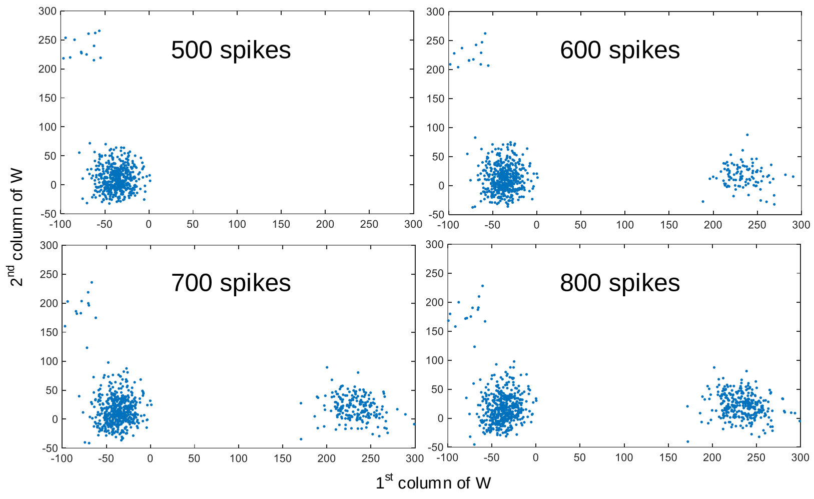
Figure 2.13: How many spikes are needed for the streaming PCA chip to converge in calculating $\hat{\mathbf{W}}$. X-axis denotes the probabilistic thresholds. Y-axis denotes the spike counts. Spike count for each threshold is averaged over 1100 trials. Higher thresholds require slightly less spikes for convergence. Reproduced from [2], ©2017 IEEE.

streaming PCA chip. This suggests that spike detection is more critical in reducing processing delays compared with feature extraction. Therefore in closed-loop operations such as neuromodulation or brain-machine interface where processing delay is critical, lowering detection thresholds could be an effective measure.

Next we examine how the streaming PCA chip adapts to nonstationary spiking activities. We prepare two distinct clusters of spikes, A and B, extracted from one session of the second data. Each cluster contains 500 spikes. We feed the spikes into the streaming PCA chip in the following sequence: The first 500 spikes are all from Cluster A to ensure that $\hat{\mathbf{W}}$ is fully converged and trained for Cluster A. The next 500 spikes are from Cluster B, during which we monitor the normalized instantaneous power of both original spikes and reconstructed spikes. As shown in Figure 2.14(a), the streaming PCA chip converges after ∼300 spikes. During the transition from Cluster A to B, the instantaneous power of reconstructed spikes tracks that of original spikes stably, indicating a smooth and swift adaptation of the chip between two distinct spike clusters. In processing the most recent ∼700 spikes after convergence, the streaming PCA chip achieves a robust reconstruction accuracy over 92% with $k = 4$ and $p = 100$. The compression ratio of spikes is therefore roughly 25×. The chip adaptation is further validated in Figure 2.14(b). By projecting compressed spikes onto a 2-D plane defined by the first two columns of $\hat{\mathbf{W}}$ (after orthogonalization) calculated in real-time, we find that Cluster B gradually emerges while Cluster A stays almost unchanged. The chip's ability in adapting to nonstationary spike data both smoothly and



(a)



(b)

Figure 2.14: Adaptation of the streaming PCA chip to changing spiking activities. First 500 spikes are from Cluster A, and the next 500 spikes are from Cluster B. (a) Tracking of the normalized instantaneous spike power. Red curves denote power of original spikes. Blue curves denote power of reconstructed spikes. Y-axis is the instantaneous power of spikes normalized to current spike count. (b) Tracking of 2-D representation of compressed spikes. X-axis and Y-axis denote the first two columns of $\hat{\mathbf{W}}$. The axis vectors are orthogonalized for visualization purpose. Reproduced from [2], ©2017 IEEE.

swiftly shows great potentials to process multichannel spikes for feature extraction and data compression.

2.4 Algorithm 2: Deep Compressive Autoencoder

2.4.1 Overview

We propose to construct high-quality codebooks using deep neural network (DNN) to facilitate effective learning-based compression. DNN-based feature extraction relies heavily on carefully designed network architectures and well tuned hyperparameters that are tailored for specific types of data. Hence it is crucial to design DNN structures that are suitable to extract representative features from multichannel spikes. Another challenge is the integration of a DNN model, normally with millions of parameters, into a neural recorder chip with limited hardware resources and power budgets. This requires hardware-aware design optimizations to obtain an extremely efficient and compact DNN model that is feasible for on-chip implementation without compromising performance.

In this work, we tackle these challenges by proposing a lightweight DNN model – compressive autoencoder (CAE) – that can compress thousands of spikes simultaneously by 20–500× with signal quality comparable to or better than that of existing approaches. Our main contributions include: (i) Instead of hand-crafted features or signal-agnostic transformations, we use convolutional neural network (CNN) along with vector quantization to extract and sample hierarchical features, which exhibit strong representational capability and generalize well to unseen spikes; (ii) We show that CAE is capable of leveraging geometrical information of spikes from multichannel recording, which is useful to expose localized features and improve qualities of reconstructed signals; (iii) We demonstrate that CAE can allow for high compression ratios without noticeably compromising spike sorting accuracy; (iv) CAE features an asymmetric model structure for signal encoding and decoding, where the encoding part (along with quantization) requires fewer than 20K parameters, which is over 40× smaller than the decoding and suitable for efficient on-chip implementation into large-scale neural recording systems.

2.4.2 Methods

Compressive autoencoder for neural data compression At the core of the proposed model is autoencoder [220], a neural network structure widely used to learn compact data representations by forcing outputs to be identical as inputs and imposing constraints in the latent space. Mathematically, the general operation of an autoencoder can be described as $\hat{\mathbf{x}} = g_s(g_a(\mathbf{x};\phi);\theta)$, where $\mathbf{x}$ and $\hat{\mathbf{x}}$ are input and output data; g_a and g_s denote *analysis* and *synthesis*, respectively, or are commonly referred to as encoder and decoder (parameterized by ϕ and θ).

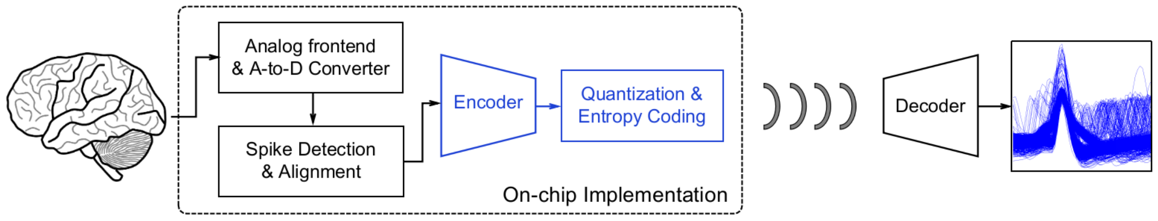


Figure 2.15: Conceptual diagram of deploying CAE in wireless neural recording system. For simplicity, microelectrodes are omitted from the figure. Spike snapshots are from synthetic datasets `Wave_Clus` [4]. Reproduced from [3].

In the context of lossy data compression, the operation of CAE becomes:

$$\phi, \theta = \arg\min_{\phi,\theta} \|\mathbf{x} - \hat{\mathbf{x}}\|_2^2,$$

$$\text{subject to} \begin{cases} \mathbf{y} &= g_a(\mathbf{x}; \phi) \\ \hat{\mathbf{y}} &= \text{quantize}(\mathbf{y}) \\ \hat{\mathbf{x}} &= g_s(\hat{\mathbf{y}}; \theta) \end{cases} \tag{2.13}$$

where the `quantize` function discretizes encoder outputs and introduces quantization error. Conventionally, the loss function of a CAE that optimizes both bit rates and distortion is:

$$\mathcal{L}_{CAE} = \underbrace{-\log_2 Q(\hat{\mathbf{y}})}_{\text{Number of bits}} + \alpha \cdot \underbrace{d(\mathbf{x}, \hat{\mathbf{x}})}_{\text{Distortion}}, \tag{2.14}$$

where $Q(\cdot)$ is the operation to estimate the discrete probability distribution of discretized data, and α is used to adjust the rate-distortion trade-off.

The conceptual diagram of deploying CAE into wireless neural recording systems is illustrated in Figure 2.15. After recorded from analog front-end circuitry and digitized, spikes are extracted from raw recording data and aligned. For CAE, only the encoder and the quantization block need to be on-chip implemented; the decoder can run on a remote computer. θ and ϕ are programmable to allow flexible choices of rate-distortion trade-offs. The indexes of VQ codebook corresponding to encoder outputs are coded and transmitted, leading to significant data rate reduction.

In practice, direct optimization of CAE using equation (2.14) proves difficult, because (i) $Q(\cdot)$ and `quantize` are typically non-differentiable thus cannot be updated via back-propagation, and (ii) joint optimization of both rate and distortion requires complex computations and carefully designed training schedules. For example, in [221] an extra Gaussian scale mixture model is used to model distribution of coefficients and estimate bit rates, as well as the requirement of fine-tuning a pre-trained autoencoder for different bit rates. On top of that, [221] needs an incremental training that gradually "releases" coefficients for updates and takes up to 10^6 iterations to achieve good performance. The first difficulty can be solved by directly copying the gradient of decoder inputs to the encoder outputs during training, bypassing the quantization block [222, 221, 223]. To address the second difficulty, we remove the rate penalty from equation (2.14) and leave the size of VQ codebook programmable by the users. The advantage of this modification is two-fold: (i) We found that penalizing only distortions can lead to fast convergence of training (typically $\sim$200 epochs) with good performance. This is beneficial to fast and simplified deployment of the model onto mobile hardware platform for real-time spike compression; (ii) It allows straightforward optimization towards lower distortion. To update VQ codebook, we add the Euclidean distance between encoder outputs and corresponding VQ codewords into the overall loss function. After the modification, the loss function of the proposed CAE is:

$$\mathcal{L}_{CAE} = d(\mathbf{x}, \hat{\mathbf{x}}) + d(\mathbf{y}, \hat{\mathbf{y}}). \tag{2.15}$$

Compared with Equation (2.14), the rate penalty is removed and another Euclidean loss to optimize VQ codewords is added. Thus the VQ codewords can be updated in the same way as other parameters via back-propagation, which simplifies the network training.

Encoder and decoder networks Fusing spatial and temporal information by stacking a set of convolutional filters interleaved with non-linearity and pooling is essential to enhance the representational power of DNN [224]. The extraction and fusion of spatial features is realized within the computation of each CNN layer. For a CNN layer with C_{in} input channels and C_{out} output channels, the value of the j_{th} output channel can be described as

$$\text{out}(N_i, C_{out_j}) = \sum_{k=0}^{C_{in}-1} \text{weight}(C_{out_j}, k) \star \text{input}(N_i, k), \tag{2.16}$$

where N is the batch size, `weight` is the coefficients of a CNN filter, $\star$ denotes cross-convolution, i.e., $I \star K(i,j) = \sum_m \sum_n I(i+m, j+n)K(m,n)$. Thus it is clear that to compute one channel output of a CNN layer requires all C_{in} input channels, which essentially realizes fusion of spatial features from previous layers and propagation of the features to subsequent layers.

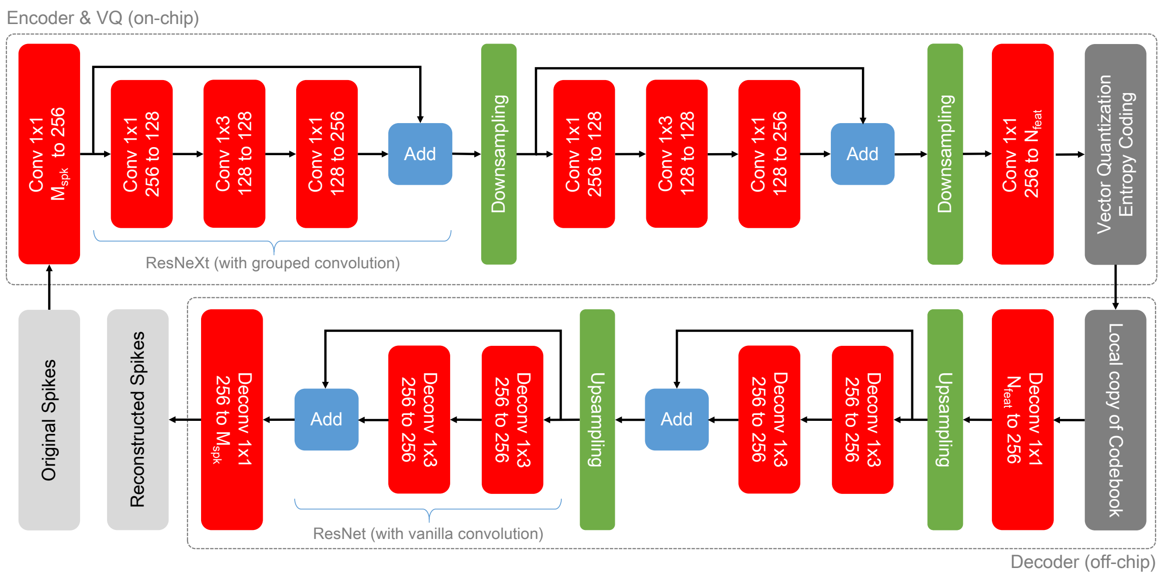


Figure 2.16: Structural diagram of the proposed spike compression model. Activation layers and normalization layers are skipped from the diagram for simplicity. Reproduced from [3].

In general, deep networks are representationally superior to shallow ones to allow for sufficient information fusion and propagation [225]. However, there are two major constraints with deeper networks in our application. First, with more layers, the amount of network parameters (weights of convolutional filters, etc.) increases drastically, making it more difficult for on-chip implementation in the neural recorder chip. Second, as in large-scale recording the probes can span a relatively large brain area or a long range of vertical structures, the underlying signal characteristics of neural data from adjacent recording sites in one local region will be different from that of other regions due to the differences in current path between the neurons and the electrodes. Therefore, features learned by the convolutional filters from all channels may not be optimally representative for each distributed recording regions, especially when one cannot afford sufficiently deep networks on-chip.

To circumvent the first limitation, the main structures of the encoder and decoder are based on `ResNet`, i.e., residual network with identity-mapping shortcut connections [226]. It has the capability of representing a richer set of complex features compared with other network structures with the same or even larger model size, presumably thanks to its behavior like ensembles of relatively shallow networks by introducing the shortcut connection [227]. To resolve the second constraint, more effective ways of organizing convolutional filters are explored. One promising configuration is grouped convolution [228]. In vanilla convolution, a total number of $C_{in} \times C_{out}$ filters are required for a convolutional layer as evidenced in equation (2.16). With grouped convolution, the input and output channels are evenly split into C groups, thus the number of filters is reduced C-fold. By restricting the "receptive fields" of convolutional filters, local features can be preserved within each group and are more representative of neural spikes recorded from corresponding physical channels. Moreover, the amount of parameters can be significantly reduced thus more hardware efficient. In this work, 32 groups are used in all `ResNeXt` modules (following the naming in [229], i.e., `ResNet` with grouped convolution).

The detailed diagram of the proposed spike compression network is shown in Figure 2.16. In the encoder network, the input convolutional layer with kernel size 1×1 maps detected spikes organized in M_{spk} channels to a 256-channel feature space. Following the first channel-expansion layer, we cascade two stages of `ResNeXt` to enhance the feature extraction capability. The main pathway of each `ResNeXt` is configured in bottleneck connection [226], consisting of a stack of 3 layers with 1×1, 1×3, and 1×1 convolutional kernels, respectively, where the 1×1 layers are responsible for reducing and restoring dimensions, and the 1×3 layer extracts features with halved input/output dimensions. Each `ResNeXt` is followed by a 2× downsampling along the temporal dimension. The last stage of the encoder network is a vanilla 1×1 convolutional layer that aggregates the features learned from previous stages and reduces the channel dimension from 256 to N_{feat}. The decoder network is a reverse implementation of its encoder counterpart, where convolution and downsampling are replaced by transposed convolution (deconvolution) and upsampling. As the decoder network is implemented on a remote computer with abundant computational resources and is primarily used to reconstruct the inputs, we stack two 1×3 deconvolutional layers in the main pathway of each `ResNet` instead of using grouped convolution to enhance the reconstruction capability.

Dimensionality reduction, vector quantization, and entropy coding For M_{spk} input spikes in D-dimension, the encoder outputs N_{feat} feature vectors in d-dimension. The operation of `quantize` can be described as $R^d \rightarrow C$ that maps a feature vector in d-dimensional space into a codebook C containing K codewords $\{C_i; i = 1, 2, ..., K\}$. Each codeword requires $\log_2 K$ bits to represent in unsigned binary representation. Therefore, the overall data rate reduction without entropy coding is

$$\mathrm{CR} = \frac{M_{spk} \cdot D \cdot W}{N_{feat} \cdot \log_2 K}, \tag{2.17}$$

where W is the original bit-length of spike sample, typically 10–16 bits; D is the original spike dimension, typically 40–80. d is not involved in the denominator of CR as only the codeword indexes need to be transmitted. Note that in the rest of paper, the reported CRs are calculated using the entropy of codewords. The actual CR will be slightly decreased due to the overhead of using a practical code (e.g., Huffman coding).

In this work, we adopt a Voronoi vector quantizer that partitions the CAE latent space into K cells, of which the centroids are the codewords. Each cell consists of all points $\mathbf{y}$ which have less distortion when reproduced with codeword $\hat{\mathbf{y}}$ than with any other codeword. All codewords are initialized from a uniform distribution and updated according to the distance between the current values of codewords and the feature vectors output by the encoder.

Finally we discuss the distortion introduced by quantization in the CAE latent space. Given a d-dimensional quantizer with a distortion measure $\|x - y\|^r$ ($r \geq 1$), we have a high rate lower bound of the quantization distortion as ([230]):

$$\begin{aligned} Dist. &\geq \frac{d}{d+r}(V_d)^{-r/d} e^{-\frac{r}{d}(H(Q(X)) - h(X))} \\ &\approx \frac{d}{d+r}(V_d)^{-r/d} e^{-\frac{r}{d}(\log_2 K - h(X))} \\ &= \frac{d}{d+r}(V_d \cdot K)^{-r/d} e^{\frac{r}{d} \cdot h(X)}, \end{aligned} \tag{2.18}$$

where $Q(X)$ is the entropy of quantizer output, $h(X)$ is the differential entropy of quantizer input X, V_d is the volume of unit sphere in d-dimensional space. In practice, the probability density function of VQ codewords is approximately uniform, thus $H(Q(X))$ is very close to $\log_2 K$. Hence for a CAE model with d, r, and K fixed, the best performance (lowest distortion) depends primarily on the complexity of input, which is approximated as $h(X)$.

Parameter configuration and model training As suggested in equation (2.17), parameters M_{spk}, N_{feat}, and K jointly determine the achievable CR (D and W are determined by the recording specification thus excluded from discussion).

The choices of these parameters require careful trade-offs between CR, signal quality, and hardware cost. In general, a larger codebook is needed for the compression of noisier spikes or spikes recorded from many channels and with more diverse waveforms to ensure the reconstruction accuracy, as evidenced in equation (6). The ratio M_{spk}/N_{feat} affects the trade-off between CR and reconstruction accuracy, and their actual values have little impact on the performance. However, larger M_{spk} and N_{feat} would lead to increased hardware cost. M_{spk} is empirically set as 4 in all the experiments (except for Section 3.3.4). For recordings from more than 4 channels, spikes from adjacent channels are grouped together and sent into one of the M_{spk} ports.

The design and testing environment is Intel i7-6800K@3.40GHz, NVIDIA GeForce Titan Xp 12GB, 32GB memory, 256GB SSD, and Ubuntu 16.04 LTS. The proposed CAE model is implemented using the deep learning framework PyTorch 0.4.1 (with CUDA 9.0) [231]. We used the ADAM optimizer [232] with learning rate 1e-3 and evaluated the model performance after 500 epochs with batch-size 48 in all the experiments.

Datasets preparation The synthetic dataset we have chosen is `Wave_Clus` from University of Leicester [4], which has been widely used in the evaluation of spike sorting algorithms. The dataset is generated by adding several spike waveform templates to background noise of various levels, thus realizing different SNRs. We used four datasets `C_Easy1_noise01`, `C_Easy2_noise01`, `C_Difficult1_noise01`, `C_Difficult2_noise01`, each of which was generated using different spike templates. We used the ground truth spike times included in the datasets to extract spikes from the continuous data. Spikes from the four datasets were grouped together, presenting more challenge for compression due to combined spike templates. All spikes were aligned to their maximum peaks with 64 samples per spike.

The first *in vivo* dataset we used is the data recorded from the rat CA1 hippocampal region using tetrodes that are publicly available as `HC1` [233, 203]. The tetrodes consist of four 13μm nichrome wires bound together by twisting them and melting their insulation [234]. The dataset `d53301` was used for evaluation, which contains four extracellular channels from tetrodes and one intracellular channel from a micropipette. Extracellular signals were high-pass filtered at 300 Hz. Spikes were extracted from the four extracellular channels with the spike times determined by the occurrences of intracellular action potentials on the micropipette. All extracted spikes were aligned to their maximum peaks with 48 samples per spike.

To test the performance of the proposed method in compressing neural signals from more recent large-scale, high-density recording setups, we used the in-vivo data recorded from an awake, head-fixed mouse using the `Neuropixels` probe [235]. `Neuropixels` has 384 recording sites with 70×20 μm^2 per site. The neural data are band-pass filtered at 300–5000 Hz. Spikes were detected from each channel by amplitude thresholding. The threshold was set at

$$\text{Threshold} = 5 \times \text{median}\{\frac{|X|}{0.6745}\} \tag{2.19}$$

where X is the band-pass filtered signal [4]. All detected spikes were aligned to their maximum peaks with 48 samples per spike. All channels used the same parameter setting for detection and no further fine-tuning was performed. Therefore the detected spikes contain a large number of false alarms contributed by various noise sources. The existence of many non-spike activities would significantly increase the difficulty of spike compression due to the diverse signal and noise characteristics. A successful compressor should reduce the bandwidth of both spikes and non-spike activities at the same time and shift the burden of carefully differentiating spikes from noise to a remote computer.

We used the mean squared error (MSE) to optimize the neural network. To measure the signal reconstruction accuracy and also allow for comparison with other works, we reported accuracy in average signal to noise and distortion ratio (SNDR) defined as:

$$\text{SNDR} = 20 \cdot \log_{10} \frac{\|X\|_2}{\|X - \hat{X}\|_2}. \tag{2.20}$$

The bit-length of all spike data is assumed 16-bit, which is a common setting adopted in commercial neural recording devices.

Methods for comparison We have chosen three transformation-based methods for comparison, including PCA, DCT, and discrete wavelet transform (DWT). We also compared with a recent work based on compressive sensing, group weighted analysis l_1-minimization (GWALM) [236], that showed better performance compared with other compressive sensing-based approaches.

- *Proposed CAE* For each dataset, spike data are randomly split into two parts for training (50%) and testing (50%). The random training/testing partition was repeated five times on each dataset and for each method we took the average performance as the final results. We train the network using the training data and evaluate the model performance on the testing data. The number of partitions K in VQ is assumed powers of two. The CR is computed using equation (2.17) with the $\log_2 K$ replaced by the entropy of the codeword indexes on the testing data.

- *PCA* We apply PCA on the training spikes, and keep the leading m eigenvectors as the transformation matrix. For compression, we multiply the transformation matrix with the testing spikes to obtain principal components as the compressed signals. The number of eigenvectors m is set as 2, 4, 6, 8, 10 for `Neuropixels`; 1, 2, 4, 6, 8 for `HC1` and `Wave_Clus`. Each principal component is represented using the same bit-length as that of spikes. The CR can be computed as CR $= D/m$.

- *DWT* Spikes are first transformed into wavelet representation and the m most significant coefficients are kept (others are zeroed). The number of wavelet coefficient m is set from 2 to 12 with an increment of 2 for all datasets. The *Symmlet*4 wavelet basis is used as it is advantageous over other wavelet basis families for processing neural signals [196]. Each wavelet coefficient is represented using the same bit-length as that of spikes. The CR can be computed as CR $= D \cdot W/(W \cdot m + D)$, where the D bits in the denominator are used to denote the positions of the m non-zero coefficients.

- *DCT* We keep the m leading coefficients of each spike after transformed by DCT. m is set from 8 to 16 with an increment of 2 for `Wave_Clus`; 6, 8, 10, 11, 12 for `Neuropixels`; from 6 to 10 with an increment of 1 for `HC1`. Each coefficient is represented using the same bit-length as that of spikes. The CR can be computed as CR $= D/m$.

- *GWALM* First, an analysis model is adopted to enforce sparsity of spikes; second, a multi-fractional-order difference matrix is constructed as the analysis operator;third, by exploiting the statistical properties of the analysis coefficients, a group weighting approach is developed to enhance the performance of analysis l_1-minimization. Each spike was compressed to a vector of length m. m is set from 8 to 16 with an increment of 2 for `Neuropixels`; from 8 to 12 with an increment of 1 for `HC1`; from 12 to 17 with an increment of 1 for `Wave_Clus`. The CR can be computed as CR $= D/m$. More details of the algorithm can be found in [236].

2.4.3 Experiments

Compression of synthetic and *in vivo* spikes We run CAE and other approaches on each of the synthetic and in-vivo datasets. The rate-quality curves of all methods on each dataset are plotted in Figure 2.17. Both horizontal and vertical axes are in logarithmic scale

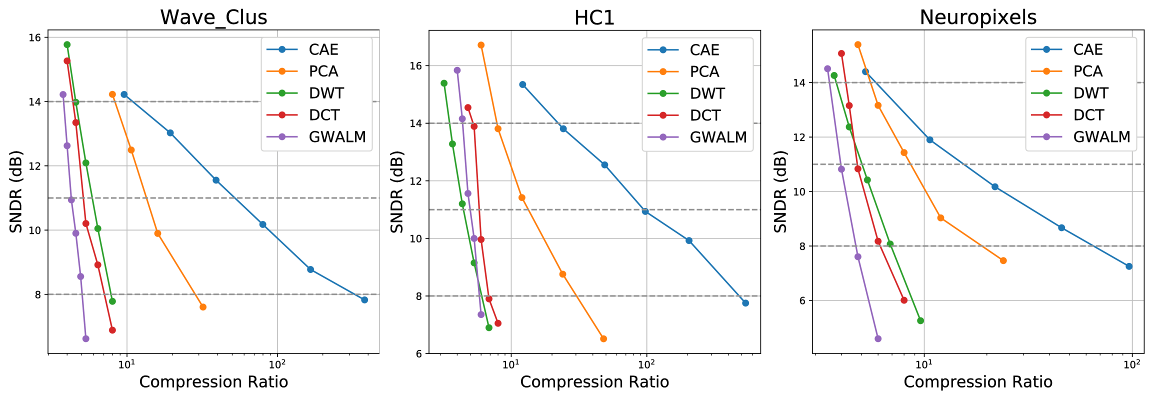


Figure 2.17: Rate-quality curves of all methods on `Wave_Clus`, `HC1`, and `Neuropixels` datasets. Vertical axis is signal reconstruction accuracy measured in SNDR. Horizontal axis is compression ratio (also in logarithmic scale). Configuration of CAE: $M_{spk}/N_{feat} = 1/4$, $K = 128, 32, 512$ for `Wave_Clus`, `HC1`, and `Neuropixels` datasets, respectively. SNDRs at 8dB, 11dB, and 14dB are highlighted in gray lines. Reproduced from [3].

to clearly distinguish curves corresponding to different methods. As shown in the figure, CAE outperforms all other methods, primarily by extending CR into the range of 20–500×. We also highlight three levels of reconstructed signal qualities measured in SNDR: 8 dB, 11 dB, and 14 dB. It is clear that CAE achieves much higher CRs on both synthetic and in-vivo datasets than other methods especially at SNDR of 8 dB and 11 dB, e.g., up to 500× CR on `HC1`, which is 15× better than PCA and over 70× better than DWT, DCT, and GWALM. The performance gap on `Neuropixels` at SNDR of 14 dB is small due to the more complex signal characteristics of spikes from hundreds of recording channels. The qualities of the reconstructed spikes using CAE are illustrated in Figure 2.18 and Figure 2.19, each of which shows 24 reconstructed spikes with various shapes randomly chosen from each dataset.

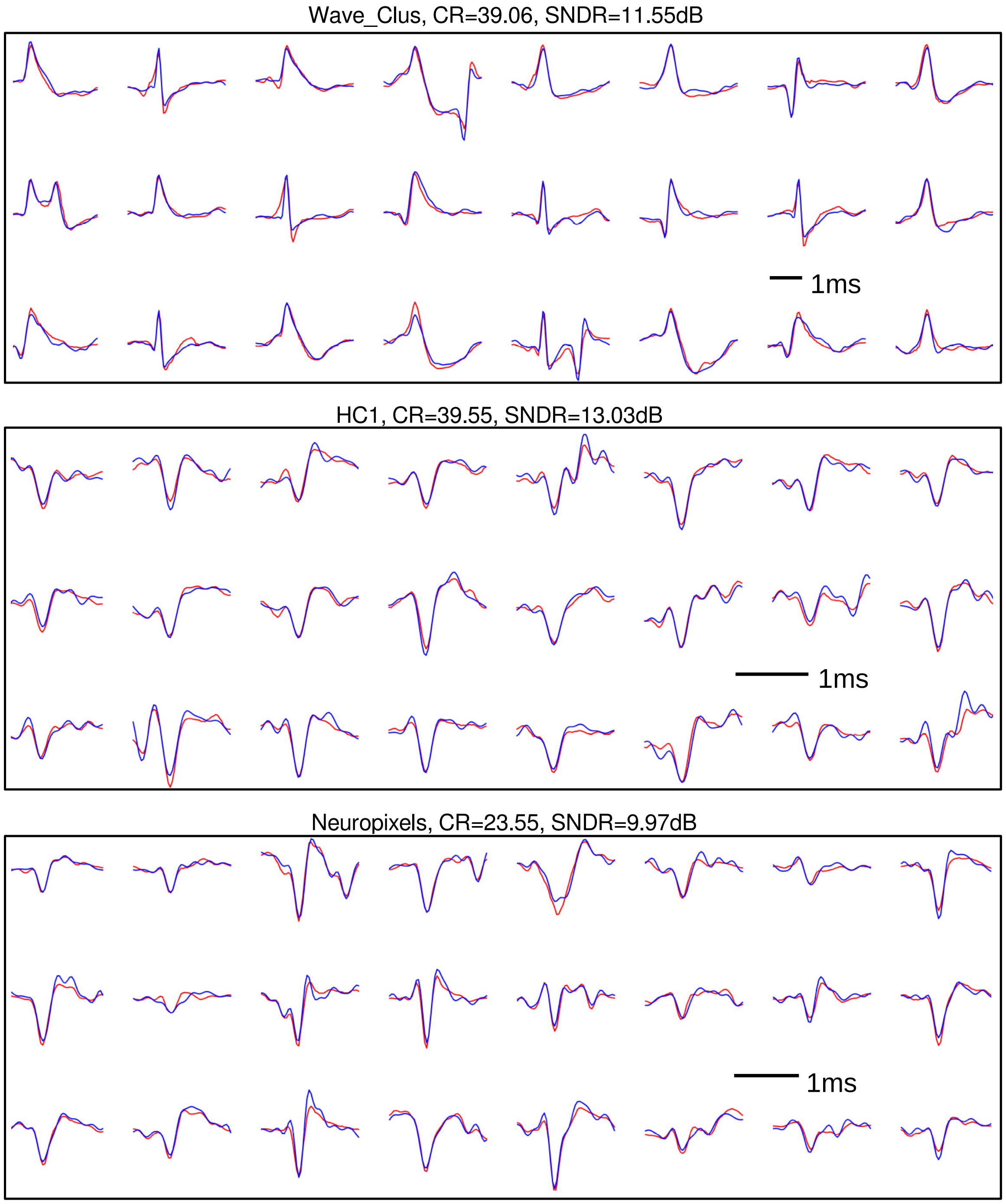


Figure 2.18: Reconstructed spikes using CAE. 24 spikes with various shapes are chosen and shown for each dataset. Original and reconstructed spikes are drawn in blue and red colors, respectively. Configuration of CAE: $M_{spk}/N_{feat} = 1/4$ for all datasets; $K = 128, 32, 512$ for `Wave_Clus`, `HC1`, and `Neuropixels` datasets, respectively. The reported CR and SNDR on top of each sub-figure are calculated over the entire testing part of corresponding dataset. Reproduced from [3].

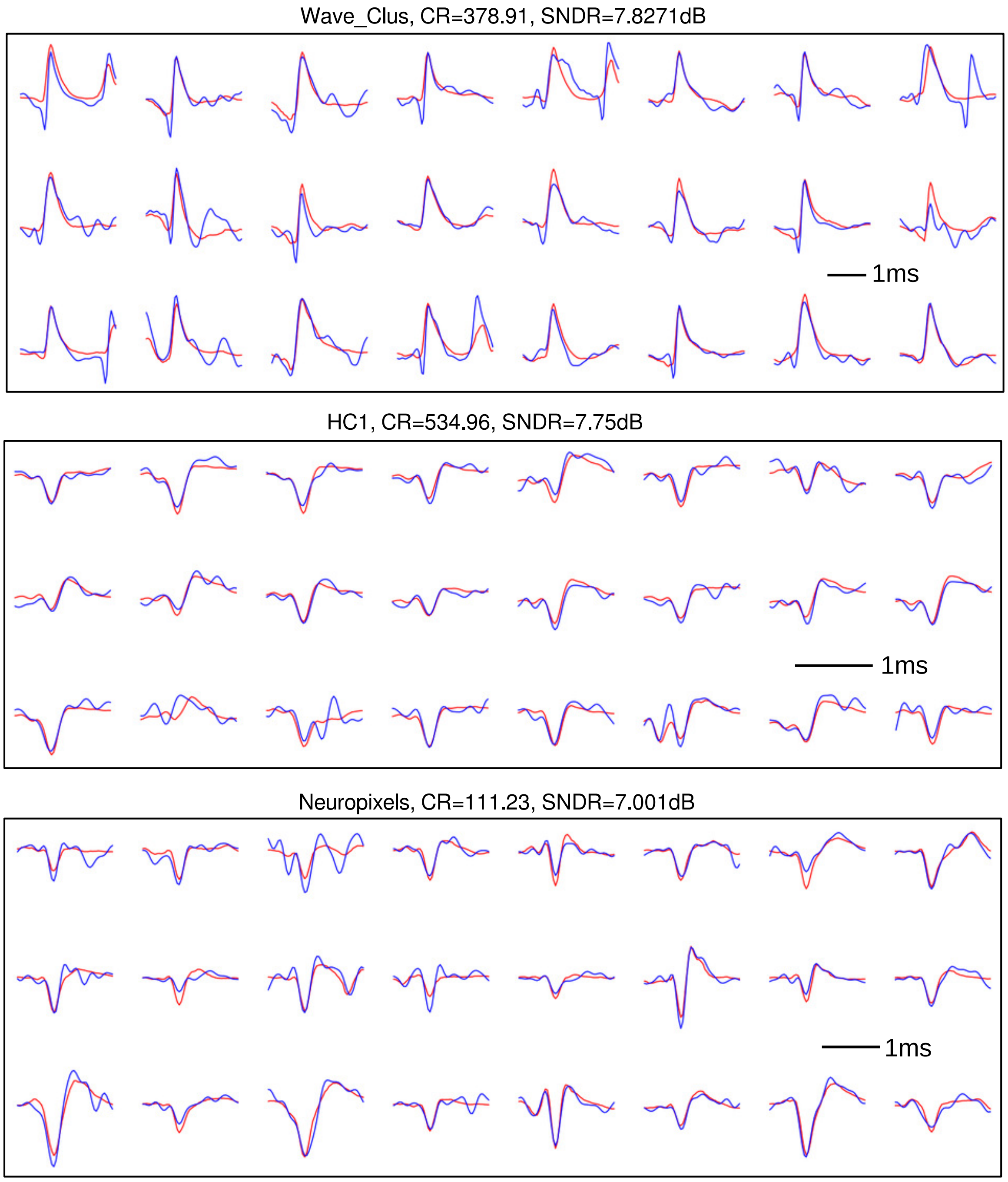
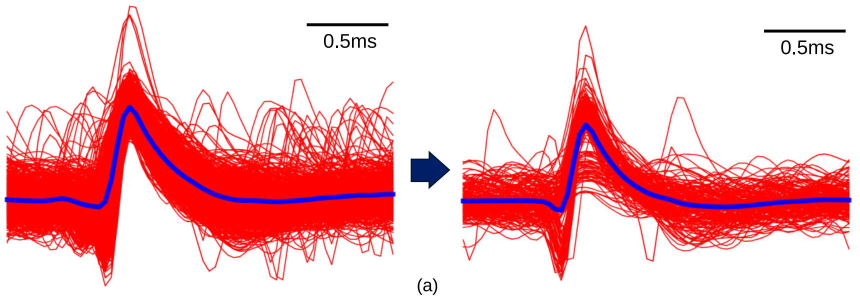


Figure 2.19: Reconstructed spikes using CAE. 24 spikes with various shapes are chosen and shown for each dataset. Original and reconstructed spikes are drawn in blue and red colors, respectively. Configuration of CAE: $M_{spk}/N_{feat} = 2$ for all datasets; $K = 128, 16, 256$ for `Wave_Clus`, `HC1`, and `Neuropixels` datasets, respectively. The reported CR and SNDR on top of each sub-figure are calculated over the entire testing part of corresponding dataset. Reproduced from [3].

DWT, DCT, and GWALM have similar performances: their signal qualities decrease radically to smaller than 5 dB when CR approaches 10×. PCA achieves better results than other conventional methods, possibly due to that in all linear projections, PCA can achieve the minimum reconstruction error given fixed input/output dimensions, which is consistent with our previous research [2]. It is worth noting that PCA can be considered as a type of linearized autoencoder optimized over MSE-based loss functions, which is similar to CAE; however, PCA cannot take advantage of nonlinear features that are representative for many high-dimensional data, thus its representation capability is inherently inferior to CAE. Another limiting factor of PCA for compression is that its CR cannot exceed the original spike dimension as at least one principal component is required to represent one spike. The performance of GWALM is not as good as others.

Evaluation of generalization capability of CAE One common issue of learning-based compression methods is the generalization capability of codebooks learned from training data. The compression performance is largely dependent on the similarity of the statistics of testing data relative to that of training data. Thus it is crucial to extract representative features from training data that can capture the underlying statistical distributions as accurately as possible. Such capability is of critical importance for a compression method to stay robust against various non-stationarity of neural activities. For example, individual spikes in a burst can have more than 50% amplitude variation according to simultaneous intracellular and extracellular recordings [203]. Electrode drift is another common source that causes systematic changes in the shape and amplitude of recorded spike waveforms [237].

To demonstrate that CAE indeed learns representative features instead of simply "memorizing" instances of training data, we used a synthetic dataset `C_Drift_Easy2_noise015` from `Wave_Clus` that simulates the effect of electrode drift and caused waveform variation. The sequence contains 3444 spikes, and the shapes of spikes gradually change along the temporal axis. We trained a CAE model using the first 500 spikes and tested the model on the last 200 spikes in the sequence. Figure 2.20(a) shows clear differences between the training and testing spikes, primarily including (i) decrease of average spike amplitude, (ii) reduced noise in the non-polarization parts of spikes, and (iii) a newly emerged spike cluster with much smaller amplitude, which jointly mimic the effects of waveform variation and electrode drift. The results given in Figure 2.20(b) show that CAE can compress and



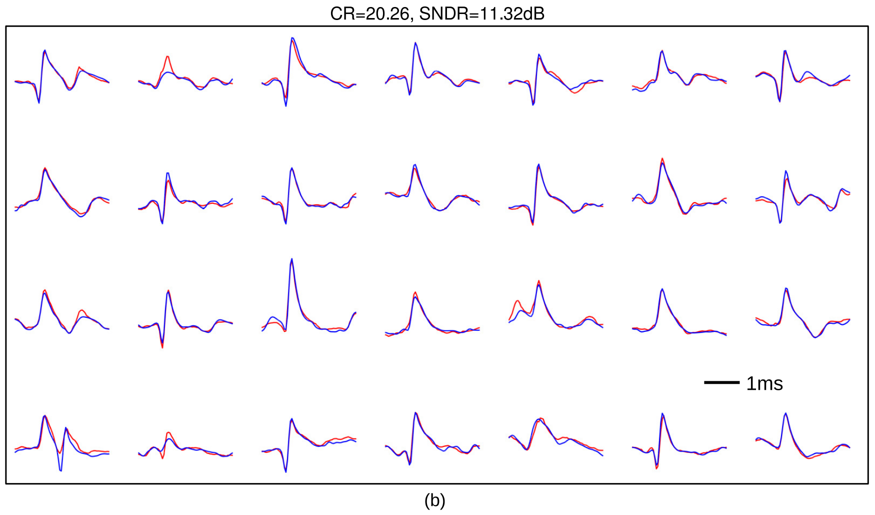


Figure 2.20: (a) Left: the first 500 spikes in `C_Drift_Easy2_noise015`; right: the last 200 spikes in the same sequence. Blue curves in each sub-figure represent the mean of all spikes. (b) Reconstruction of spikes in `C_Drift_Easy2_noise015` using CAE. 28 spikes are randomly chosen from the last 200 spikes. Blue and red waveforms are original and reconstructed spikes, respectively. Configuration of CAE: $M_{spk}/N_{feat} = 1/8$, $K = 128$. Performance on testing dataset: CR=20.26, SNDR=11.32dB. Reproduced from [3].

reconstruct not only spikes similar to the training data with high fidelity, but also unseen spikes exhibiting significant changes on amplitude and shape.

We have done another experiment to evaluate the generalization capability of CAE by using different sequences for training and testing, including `C_Difficult1` with noise levels 0.2 and 0.05, and `C_Difficult2` with noise levels 0.2 and 0.05, all of which are from `Wave_Clus` dataset. `C_Difficult1` and `C_Difficult2` are synthesized using different spike templates. For each sequence, the first 50% of spikes are used for training and the rest 50% for testing.

The results given in Table 2.2 shows that CAE can generalize well against different spike templates and varied noise levels. Specifically, (i) when spike templates are the same in the training and testing data, closer noise levels can lead to higher compression accuracies; (ii)

Table 2.2: Evaluation of CAE performance by using different sequences for training and testing. Reproduced from [3].

Training \ Testing	C_Difficult1 Noise 0.05	C_Difficult1 Noise 0.2	C_Difficult2 Noise 0.05	C_Difficult2 Noise 0.2
C_Difficult1 Noise 0.05	13.8223	8.271	12.2762	8.0851
C_Difficult1 Noise 0.2	13.0852	10.2059	14.4029	10.4476
C_Difficult2 Noise 0.05	11.6319	8.4129	15.8518	9.1695
C_Difficult2 Noise 0.2	12.3602	9.9329	15.2567	10.9564

Configuration of CAE: $K = 128$, $M_{spk}/N_{feat} = 1/4$. Numbers are SNDR (dB).

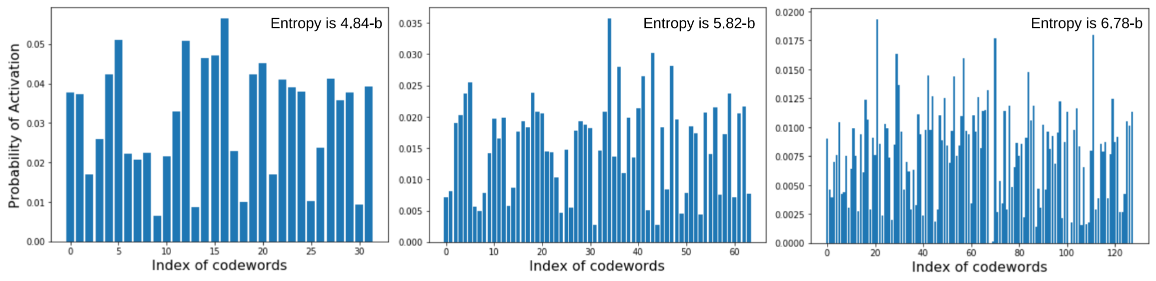


Figure 2.21: Activation patterns of the VQ codewords in CAE trained on `Wave_Clus` dataset. From left to right, the numbers of codewords are 32, 64, and 128. The entropies of VQ codewords are 4.84-bit, 5.82-bit, and 6.78-bit. Reproduced from [3].

when spike templates are different, higher noise levels can lead to better performance. In the first case, VQ codewords learned from training data are closer to the spike components in testing data than noise due to the same templates. In the second case, in the absence of common spike templates, CAE can learn more diverse features from noisier waveforms that better represent testing data in the latent space. Performance in the first case is consistently better than in the second case. The results that CAE can generalize well over different spike templates and background noises suggest the potential application of CAE in chronic wireless recording experiments to reliably compress neural signals without frequent re-training or parameter tuning.

Effects of different numbers of VQ codewords Another appealing feature of CAE is that it can uncover a low-dimensional space from spikes where features naturally conform to uniform distribution, which facilitates efficient and accurate vector quantization. To understand this feature, we examine the activation patterns of VQ codewords. As shown in Figure 2.21, after trained on the `Wave_Clus` dataset, the VQ codewords tend to be uniformly activated regardless of the number of codewords. In other words, the entropy of

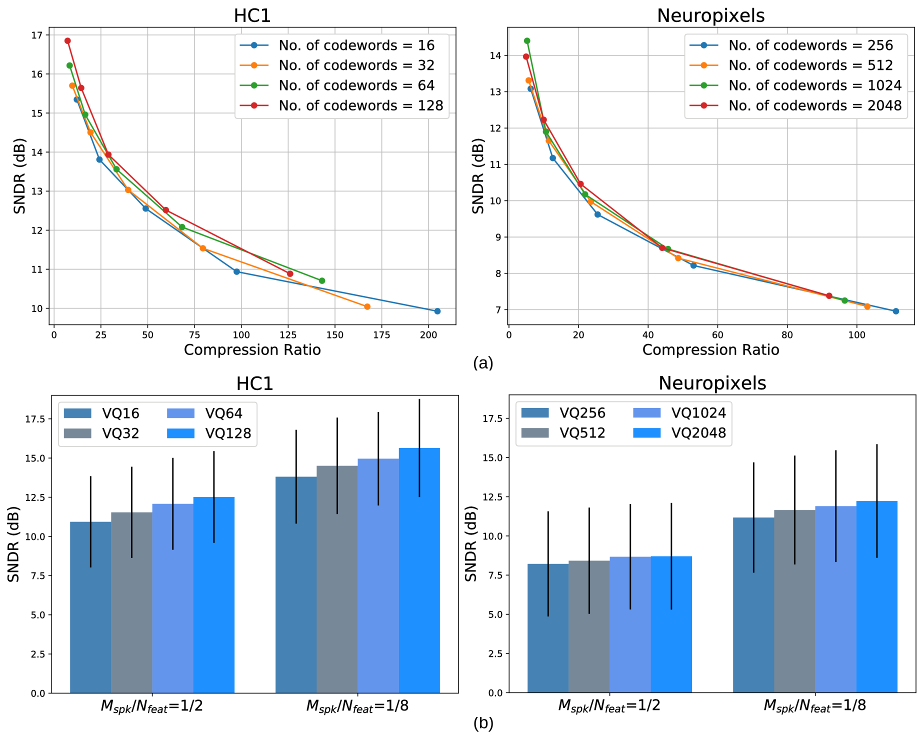


Figure 2.22: Effects of different M_{spk}/N_{feat} ratios and numbers of codewords on reconstruction accuracy. K is varied from 16 to 128 for `HC1` and from 256 to 2048 for `Neuropixels`. (a) Rate-quality curves for both in-vivo dataset. (b) Comparison of accuracies at low and high M_{spk}/N_{feat} ratios. Error bars representing standard variations of SNDR are labeled. Reproduced from [3].

VQ codewords is always close to $\log_2 K$. The situation is similar on other datasets. It should be noted that the uniformity is attained in the absence of any entropy regularization term in the loss function of CAE, which is normally required to enforce certain output distributions [238]. We hypothesize that CAE transforms spikes into a group of relatively invariant and uniformly distributed features inherent to spikes in the low-dimensional latent space, and VQ codewords converge to the grid-like spike features via nearest neighbor search. Due to the uniform distribution of features, the convergence of VQ codewords can be fast, robust, and accurate.

Under this hypothesis, achieving higher accuracy is bottlenecked by the amount and quality of spike features output by the encoder, not VQ codewords. To demonstrate this, we run CAE on the two in-vivo datasets (`HC1` and `Neuropixels`) with different numbers of codewords and plot their rate-quality curves. As shown in Figure 2.22(a), for each dataset the number of codewords is varied by up to 8× while the reconstruction accuracy changes by only ∼1dB. In comparison, increasing N_{feat} by 4× can lead to 3–4dB improvement on SNDR, as shown in Figure 2.22(b). This feature is useful for hardware implementation, since the indexing logic that searches the nearest VQ codewords of encoder outputs can be simplified, thus reducing the processing delay and improving the throughput.

Figure 2.22(b) further shows that when spike morphologies are more complicated, the accuracy is more insensitive to the number of codewords. Under this condition, it requires a larger N_{feat} to allow for fine-grained sampling from the spike feature space; however, increasing N_{feat} would decrease CR at the same time. To maintain decent CRs, the inherent resolution of spike features is limited by the upper-bounded N_{feat} and hence more codewords in this case will not be effective in improving accuracy.

Effects of preserving spatial proximity of spikes CAE is capable of extracting localized features from spikes recorded from channels that are geometrically closed to each other. We have designed the following experiment to verify that CAE can leverage the geometric information of spikes to achieve higher accuracies at no cost: simply preserving their spatial proximity at the input to the network.

We picked 15 channels from the `Neuropixels` dataset along the longitudinal dimension of the probe with a spacing of 400 μm. This is to ensure that spikes detected from different channels are generated by different neurons and thus with independent waveform characteristics. Two CAE models with the same configurations were created, where $K = 512$, $M_{spk} = 15$, and M_{spk}/N_{feat} is set as 1, 1/2, 1/4, and 1/8. The training and testing spikes for the two models are the same, except that **Model 1** was trained with spikes randomly shuffled along the channel dimension; **Model 2** was trained with spikes preserving their spatial proximity, e.g., spikes detected from probe channel 1 are fed into CAE input port 1.

The testing performances of the two CAE models are given in Figure 2.23. With training spikes shuffled along the channel dimension, the spatial proximity was disrupted. The compression accuracy of **Model 1** is consistently poorer than that of **Model 2** by 1–2 dB across different M_{spk}/N_{feat} ratios; meanwhile, their CRs are comparable (not shown in Figure 2.23). Thus, preserving spatial proximity is important for CAE to extract localized features from multichannel spikes and achieve higher accuracies.

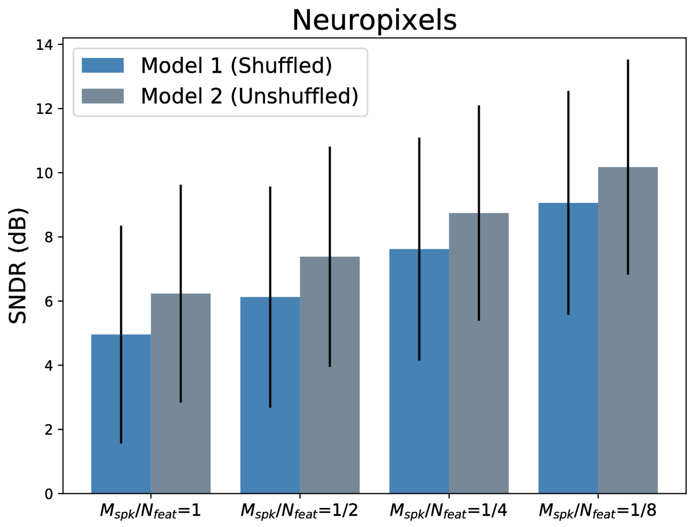


Figure 2.23: Effects of shuffling spikes along channel dimension on compression accuracies. For both CAE models, $K = 512$, $M_{spk} = 15$, and M_{spk}/N_{feat} is set as 1, 1/2, 1/4, and 1/8. Error bars represent standard deviations of SNDR. Reproduced from [3].

Evaluation of clustering performance before and after compression An important analysis in neural signal processing is to obtain single-unit activity from raw recordings, a process commonly known as spike sorting that classifies spikes to their originating neurons [170]. Hence it is necessary to evaluate the distortions on spike sorting accuracy introduced by compression. We used the following datasets from `Wave_Clus`: `C_Easy1`, `C_Easy2`, `C_Difficult1`, and `C_Difficult2`. From each dataset, we picked two sequences with the lowest and highest background noise levels, respectively. Spikes were identified according to the ground truth timestamps. In each sequence, the first 50% of spikes were used to train the CAE model, and the rest were used for testing. For each spike, the first 3 principal components are extracted as features using PCA. We run K-Means 500 times on the principal components of spikes from each sequence with randomized centroid initializations to ensure the best classification result. We repeated the spike sorting pipeline on each sequence compressed by CAE with different compression ratios, and compared the results with the ideal classification accuracies obtained from the uncompressed spikes.

In Figure 2.24, we visualize the testing spikes in the 2-dimensional PCA feature space at different CRs. It shows that with smaller CRs (higher SNDR), compressed spikes tend to be more "scattered" and resemble the distribution patterns of uncompressed spikes. The spike sorting results are given in Figure 2.25. In each sub-figure, the two gray dashed

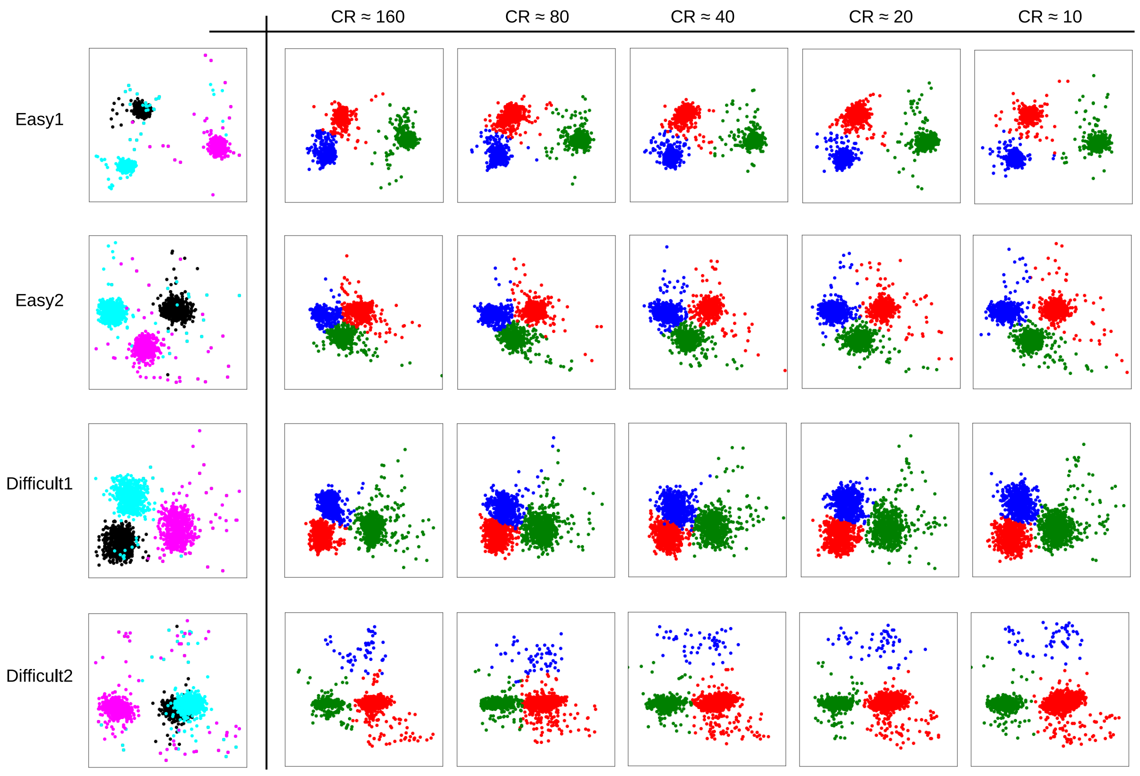


Figure 2.24: Visualization of spikes in 2-dimensional PCA space. From top to bottom, the four sequences are `C_Easy1_noise005`, `C_Easy2_noise005`, `C_Difficult1_noise005`, and `C_Difficult2_noise005`. Sequences with higher noise levels are not plotted because of poor separation in PCA feature space. Plots in the leftest column are the uncompressed spikes with ground truth labels. Plots in the second to the sixth columns are compressed spikes at different compression ratios classified with PCA + K-Means. In each row, compressed spikes are plotted in the same feature directions as the uncompressed spikes. Reproduced from [3].

lines represent the sorting accuracies using uncompressed spikes with low and high noise levels, respectively. With low noise levels, the drop of classification accuracy caused by CAE compression is less than 4% for up to 178× CR; with high noise levels, the drop of classification accuracy is slightly larger than with low noise levels, mostly less than 5% except for `C_Easy2` where the performance drops by 9% at 161× CR. In addition, on all sequences, the sorting accuracy stays almost unchanged with respect to exponentially increased CR (until CR is over 64×). The results suggest that CAE can allow for high CRs without noticeably compromising spike sorting performance.

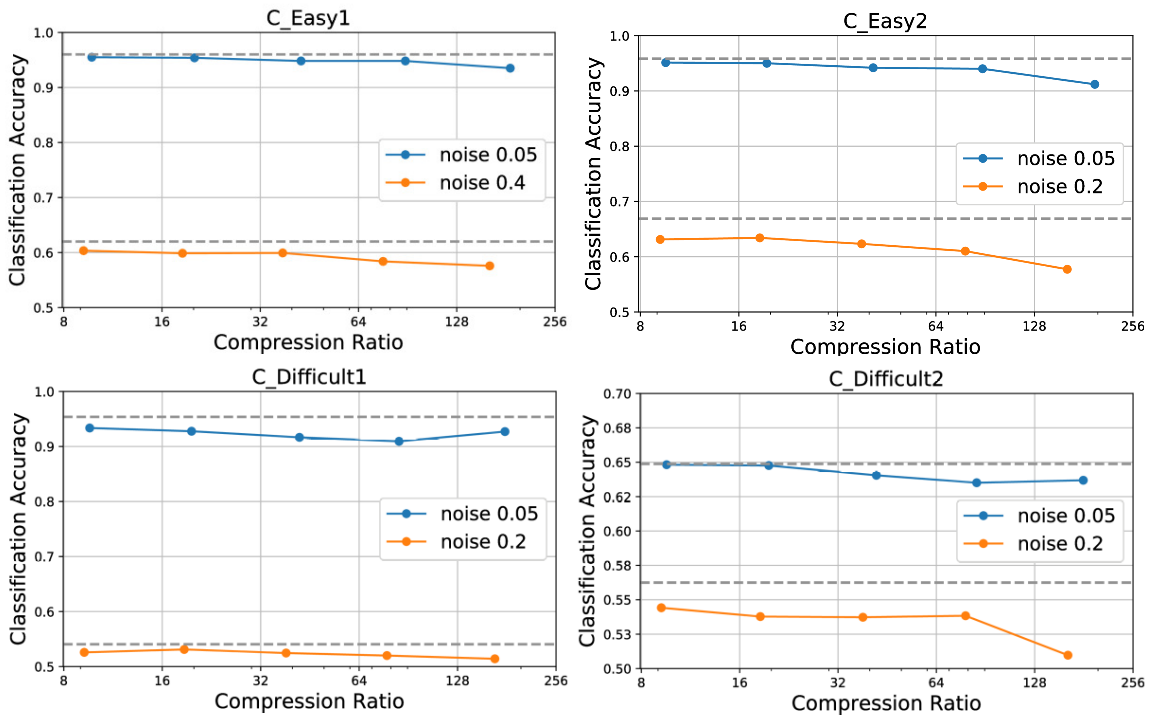


Figure 2.25: Evaluation of clustering (PCA + K-Means) performance before and after compression using CAE. For all datasets, $K = 128$, M_{spk}/N_{feat} is varied from 1/1 to 1/16. Feature dimension after PCA is 3. K-Means is run 500 times on each sequence after PCA with randomized centroid initializations. In each sub-figure, the two gray dashed lines represent the sorting accuracies before compression and correspond to the low and hight noise levels, respectively. Reproduced from [3].

Evaluation of CAE performance in the presence of spike misalignment and overlapping For the purpose of reliable and accurate feature extraction, it is often required that spikes are aligned to the peaks or maximum slopes. However, accurate spike alignment is difficult in low-SNR recordings due to sampling jitter combined with noise effects [170], and results in misaligned spikes.

One potential solution is to perform careful spike detection, which discerns "clean" spike shapes that can be well aligned from noisy backgrounds. However, such operation is often supervised and time-consuming, and also computationally unrealistic for on-chip and real-time implementation. To overcome this difficulty, spike compressor is expected to perform robustly against misalignment. To examine this capability of CAE, we chose a low-SNR sequence from `Wave_Clus` (`C_Difficult2_noise02`). We added a small temporal jitter to each ground truth timestamp and extracted spikes. The jitters were randomly sampled from

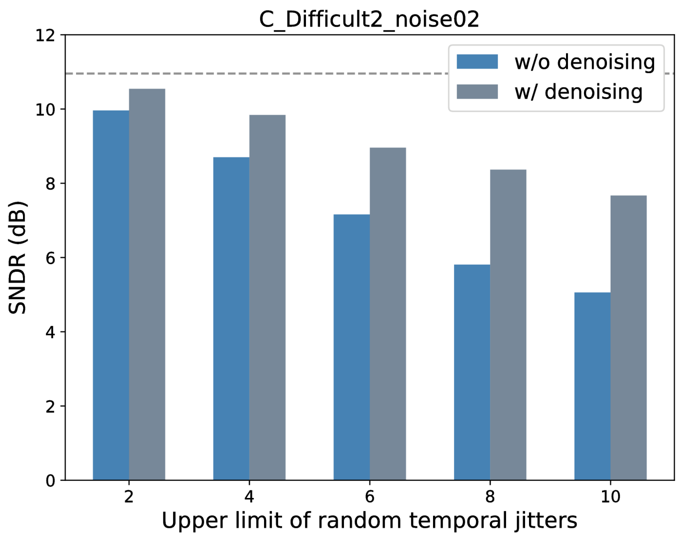


Figure 2.26: Performance of CAE against spike misalignment. Each spike is temporally jittered by up to 2, 4, 6, 8, 10 points. Compression accuracy is 10.95dB without misalignment (gray dashed line). Compression accuracies w/o and w/ denoising CAE are drawn in blue and gray bars, respectively. Reproduced from [3].

a centered uniform distribution spanning a width from 2 to 10 points with an increment of 2. CAE was trained using the jittered spikes and evaluated by attempting to reconstruct clean spikes from jittered spikes.

As shown in Figure 2.26, the compression accuracies decrease by 1–6 dB at different extents of misalignment (blue bars). Here we present a technique to enhance the performance by configuring CAE as a denoising autoencoder without modifying its structure. Referring to equation (3), instead of using the jittered spikes $\mathbf{x}$, we use the clean spikes $\mathbf{x}_{clean}$ that correspond to the jittered spikes to compute the loss as:

$$\mathcal{L}_{CAE} = d(\mathbf{x}_{clean}, \hat{\mathbf{x}}) + d(\mathbf{y}, \hat{\mathbf{y}}), \tag{2.21}$$

where $\mathbf{y}$ and $\hat{\mathbf{y}}$ are still calculated from the jittered spikes $\mathbf{x}$. The access to $\mathbf{x}_{clean}$ is feasible since CAE needs to be trained off-line, where training spikes can always be accurately aligned. Optimized with the new loss, CAE is encouraged to learn reconstructing clean spikes from misaligned spikes – an essentially denoising process, thus can perform substantially better on unseen spikes with similar misalignment (gray bars in Figure 2.26).

Another issue that will hurt compression performance is spike overlapping, which can be frequent in high-density recordings, especially with high-rate spike activities. Resolving overlapped spikes is a challenging task. In the recent spike sorting pipelines (e.g. `KiloSort`

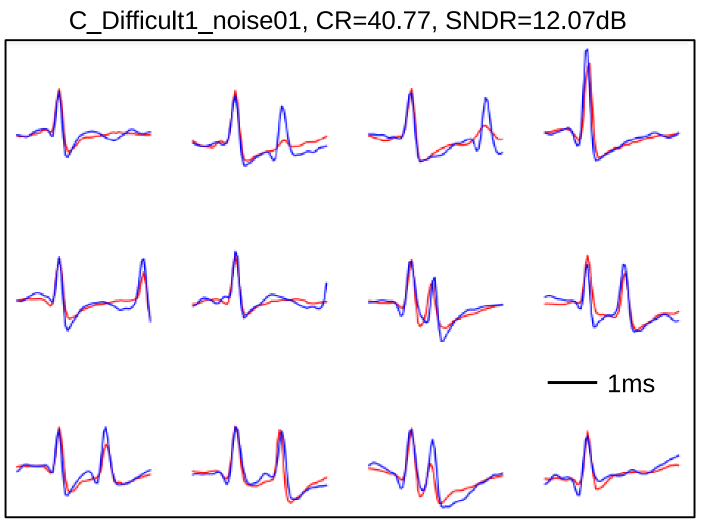


Figure 2.27: Performance of CAE against spike overlapping. Original and reconstructed spikes are drawn in blue and red, respectively. Configuration of CAE: $K = 128$ and $M_{spk}/N_{feat} = 1/4$. Reproduced from [3].

[239]), it is approached by comparing overlapped spikes with an exhaustive search of linear combinations of clean spike templates (typically two spikes with varied amplitudes and phases), and requires iterative processing that can only be afforded off-line. In this work, our design goal is to make CAE capable of compressing both clean and overlapped spikes as accurately as possible, and leave the computationally expensive resolving overlapped spikes to off-line processing. As shown in Figure 2.27, CAE shows a promising performance in representing both clean and overlapped spikes at a reasonably good CR.

2.4.4 Feasibility of On-Chip Implementation

Deployment of deep learning models onto hardware platforms with limited resources and constraints of power/heat dissipation is challenging due to the excessive amount of model parameters and incurred computations. For example, `ResNet-152`, the first deep learning model that won the ImageNet classification challenge by surpassing human-level accuracy, contains 60 million weights and requires 11.3G multiply-accumulates (MAC) to process one image [240]. Consequently, the model size of `ResNet-152` is over 200MByte. It is impractical to implement compression models with similar sizes as application-specific integrated circuit (ASIC) chips. Hence it is crucial to take into serious consideration the complexity of compression models for on-chip integration with analog front-end recording circuitry.

Table 2.3: Model parameters of encoder and VQ. Reproduced from [3].

Layers	Output Size	Parameters
Convolution	N[a], 256, 48	1024
Normalization	N, 256, 48	512
`ResNeXt` (3×conv.)	N, 256, 48	4608
Downsampling	N, 256, 24	0
`ResNeXt` (3×conv.)	N, 256, 24	4608
Downsampling	N, 256, 12	0
Convolution	N, 16, 12	4096
Normalization	N, 16, 12	32
Vector Quantization (256 codewords)	N×16, 12	3072
Total	**17952**	

[a] N denotes batch size in Table 2.3 and 2.4.

Table 2.4: Model parameters of decoder. Reproduced from [3].

Layers	Output Size	Parameters
Deconvolution	N, 256, 12	4096
Normalization	N, 256, 12	512
Upsampling	N, 256, 24	0
`ResNet` (2×deconv.)	N, 256, 24	394240
Upsampling	N, 256, 48	0
`ResNet` (2×deconv.)	N, 256, 48	394240
Deconvolution	N, 4, 48	1028
Total	**794116**	

We first examine the model size of CAE. CAE contains 8 convolutional layers in the encoder network interleaved with pooling and normalization layers which require none or trivial amount of parameters. Table 2.3 and 2.4 give detailed information of a CAE model trained on `Neuropixels` dataset for the on-chip and off-chip parts, respectively. The output size of each layer follows the format of {`batch size`, `channel dimension`, `feature dimension`}. The parameters of the encoder and VQ are counted together as they are to be on-chip implemented. The total amount of parameters is 812 K, which is a minor fraction of that of `ResNet-152`; furthermore, the encoder (including VQ) is over 44× smaller than the decoder, thanks to the grouped convolution technique employed for the `ResNeXt` module, resulting in fewer than 18 K parameters. Assuming an 8-bit weight precision (which has been used successfully in several commercial products such as Tensor Processing Unit), it would require 18KByte memory to store the CAE on-chip. Taking `Eyeriss` [241] as a reference design (one of the state-of-the-art deep learning chips), it has 181.5 KByte on-chip SRAM and 108 KByte global buffers, both of which are sufficient to load the on-chip part of CAE.

Next we examine the computational complexity and power efficiency of the on-chip part of the CAE model reported in Table 2.3. On average, it takes 79.25 K MACs to process one spike for on-chip computation. To estimate the power efficiency, again we refer to `Eyeriss` as a reference design, which has an energy efficiency of 83.1 GMACs per Watt [241]. Therefore, the on-chip part of CAE would consume 0.95 μW to process one spike if implemented on `Eyeriss`. Assuming an average firing rate of 20 Hz per channel, the power consumption of spike compression using CAE would be 19 μW/channel, which is comparable to that of analog recording circuitry (10–50 μW/channel [242]).

Regarding the processing speed of the CAE model, `Eyeriss` has a throughput of at least 16.8 GMACs. Referring to the requirement of 79.25 KMACs/spike derived earlier, the on-chip CAE model has a theoretical peak throughput of 0.22 M spikes. However, this astonishing processing capability cannot be achieved, because the power density of invasive neural implants that conduct brain signal sensing, processing, and transmission must adhere to rigid regulations, that is smaller than 400 μW/mm^2, to prevent from damaging brain tissues caused by increased temperatures as a result of heat dissipation [243]. The power density of `Eyeriss` is 22.67 mW/mm^2 at a throughput of 23.1 GMACs. Constraining the power density to 400 μW/mm^2, the highest throughput is 0.4 GMACs, which translates to processing around 5000 spikes simultaneously. As neuronal firing is in general sparse and concurrent firing of multiple nearby neurons is infrequent, the 5000 spikes throughput should be able to support thousands of recording channels simultaneously.

2.5 Summary

In this chapter, we discussed two unsupervised neural data compression algorithms about their mathematical details, hardware implementation feasibility, and applications to different types of neural data recorded with various models of electrodes.

Streaming PCA is originally motivated from the observation that in the full spectrum of neural data processing, LFP compression has long been a missing piece in literatures.

We found that by properly arranging the recording data matrix, it is feasible to compress multichannel LFPs using the same computational structure of PCA-based feature extraction as that of spikes. Streaming PCA is fully adaptive, autonomous, and suitable to process streaming data without training an expensive data covariance matrix. We developed a streaming PCA chip, that is fabricated in a 65nm CMOS technology, and features an ultra-low-power design, consuming only 144nW/channel for LFPs and 3.05μW/channel for spikes. In compressing LFPs recorded with a 4-shank, 32-channel probe, the chip is configured as $k = 4$ and $p = 32$, achieving an 8$\times$ compression ratio and 1$\sim$7% reconstruction errors; in compressing LFPs recorded with a 10-line, 100-channel MEA, the chip is configured as $k = 4$ and $p = 40$, achieving a 10$\times$ compression ratio and $\sim$10% reconstruction errors; in compressing spikes, the chip is configured as $k = 4$ and $p = 100$, achieving a 25$\times$ compression ratio and $\sim$8% reconstruction errors. The chip can also adapt to nonstationary spike data both smoothly and swiftly without compromising reconstruction accuracy.

CAE, on the other hand, is a DNN-based spike compression model to significantly reduce the data rate of spikes in large-scale neural recording experiments. Compared with existing methods, CAE can raise the CR to 20–500$\times$ while provides comparable or better signal qualities. There are several advantageous features of CAE:

- CAE can extract representative features from spikes, which are robust to non-stationarity of neural activities (waveform variation and electrode drift) and recording imperfections (spike misalignment and overlapping).
- CAE is capable of leveraging the spatial proximity of spikes from multiple channels to improve compression performance.
- CAE allows for high compression ratios while retaining spike sorting accuracy.
- CAE features an asymmetric model structure, in which the encoder can be designed in a way that requires much less hardware resources than the decoder without undermining feature extraction capability, thus making CAE very suitable for hardware-efficient deployment into implantable neural recording systems.

We also provided quantitative evaluation of implementing CAE on a recent state-of-the-art deep learning acceleration chip `Eyeriss`, and demonstrated the potential to support thousands of recording channels simultaneously for spike compression.

Essentially, streaming PCA and CAE each represents a type of methods operating in a distinctive feature space. Streaming PCA converts signals into an orthogonal space with the directions of axes capturing the data variance in the descending order (**orthogonal domain**). In comparison, the hierarchical features extracted by CAE in the latent space are trained to optimize the reconstruction quality thus accuracy is guaranteed; meanwhile, compression is gained from the dimensionality reduction achieved by the encoder and the bottleneck structure of the network (low-dimensional latent space) as well as the entropy coding based on VQ results. Furthermore, CAE features can be trained end-to-end, which avoids any heuristic constructions that may limit performance.

Chapter 3

Semi-Supervised Representation Learning for Spike Sorting

3.1 Overview

Understanding the coordinated activity underlying brain computations requires large-scale, simultaneous electrophysiological recordings from distributed neuronal structures at a cellular-level resolution. A key step in interpreting the multi-unit neural activities is spike sorting, the process of detecting spiking events from continuously sampled extracellular voltages and assigning the events to putative neurons [170, 244]. Spike sorting consists of three main steps: detection, feature extraction, and clustering. Spike sorting is important for many downstream investigations, for instance, individual neurons' tuning properties, firing characteristics, as well as the lateral inhibition and excitatory-inhibitory competition between nearby neurons [86].

Recent works of spike sorting are mainly based on three fundamental types of algorithms (oftentimes a mixture of them): template matching [239, 245, 237], density-based clustering [245, 246, 247], and model-based clustering [248, 237]. Template matching assumes that extracellularly recorded signals can be decomposed into a weighted sum of spike templates plus noise. Identifying and clustering spikes usually requires solving a customized optimization problem through iterative computations to infer spike times and waveforms. In density-based clustering, spikes are grouped into regions of higher densities than the rest in a feature space, requiring little or no prior knowledge about the data. In practice, the number of clusters is often sensitive to the parameter that determines the radius of neighborhood to estimate densities. Furthermore, calculating the distance between every pair of data points in each iteration could drastically slow down the processing. Model-based clustering classifies spikes by fitting assumed model structures (e.g., Gaussian mixture model, t-distribution) to the empirical data distribution. It also allows incorporating prior information or assumptions into modeling. However, the learning and inference of such probabilistic models often incur high computational costs.

One important commonality of the recent spike sorting methods is the request for human knowledge to improve sorting performance. Human supervision is necessitated due to the lack of ground truth information (e.g., the number and the identities of neurons) and that even well designed spike sorting algorithms cannot exhaustively cover all possible situations. For example, in [237], an interactive clustering is designed to allow users to merge or split clusters for model re-fitting. There are attempts to design fully unsupervised, automated spike sorting algorithms [249, 250, 247, 251]. Despite these efforts, in practice semi-automatic spike sorting is still widely adopted to ensure sorting performance. In general, human supervision is used primarily as a step of post-processing in existing spike sorting algorithms for correcting the erroneous or suboptimal decisions made by the automated clustering routines. There are shortfalls of this "cluster-refine-cluster" arrangement: 1) It requires a suitable feature space for visualizing and manipulating spikes, of which the design is difficult and mostly heuristic; 2) Human supervision can only happen after the collection and automated clustering of a vast quantity of spikes, making it less suitable for online decoding experiments that require minimum processing delay.

In recent years deep learning has made rapid advancements in various applications such as computer vision and natural language processing [252, 138, 253]. On some specific tasks, e.g. image classification [226], the performance of deep learning models has already surpassed human-level accuracy. As sorting spikes is similar to classifying images by nature, it is tempting to leverage deep learning for spike sorting and achieve human-like performance or even beyond. There has been a few works along this direction with some preliminary results being made [248, 254, 255]. However, there are several important issues about applying deep learning for spike sorting that remain unaddressed. First, the deep learning models presented in these works target only spike detection, which is the first step of spike sorting, leaving the rest steps, especially clustering, still handled by conventional approaches that suffer the aforementioned limitations. Second, these works did not propose a principled approach to mitigate the data-hungry limitation of deep learning that is particularly critical in spike sorting. Deep learning models normally requires a large number of labeled training examples to learn an accurate mathematical mapping from inputs to targets, whereas humans can learn abstract relationships in a few trials, a capability commonly referred to as *few-shot learning*. Meanwhile, it is impractical and labor-intensive to ask domain experts in neurophysiology to manually label sufficient amount of spikes (e.g., hundreds to thousands) for training deep learning models. These limitations, especially the second one, severely impede effective utilization of deep learning models in spike sorting, and, in a broader sense, neural signal processing.

In this work, we propose a **F**ew-**S**hot **S**pike **S**orting (`FSSS`) deep learning model for semi-automatic spike clustering. The proposed work is not intended to be a complete solution of spike sorting, instead it focuses on feature extraction and clustering, and can be used together with recent works such as `SpikeDeeptector` [255] to provide a complete flow of spike sorting. `FSSS` has a number of desirable properties. First, it can learn how to sort spikes from a small number of examples labeled by human operators (manually or semi-manually), thereby mitigating the data-hungry limitation of deep learning models. This few-shot learning capability is achieved primarily through an adversarial representation learning process, which is inspired by the meta-learning theory from machine learning community [256, 257]. Second, thanks to the end-to-end training of the deep learning model, it avoids the handcrafted design of a suitable feature space for visualizing and manipulating spikes, and thus can better imitate the way human operators sort spikes and avoid problematic clustering decisions that might be made by automated routines. Third, thanks to the layer-wise structures with no loops, it can be significantly accelerated by dedicated hardwares such as graphics processing units (GPUs) and has the potential to process spikes from hundreds of thousands of recording channels in real-time.

In addition to `FSSS`, we propose a lightweight clustering routine termed `DidacticSort` to aid users in labeling spikes semi-manually. `DidacticSort` processes spikes in a rudimentary yet robust way: Spikes are progressively assigned to the closest clusters based on the Euclidean distances between the discriminative part of spikes and cluster templates. A new cluster is created if a spike is too distant from all exiting clusters. Despite the simplicity, it performs robustly and can generate high-quality candidate clusters from typically a few

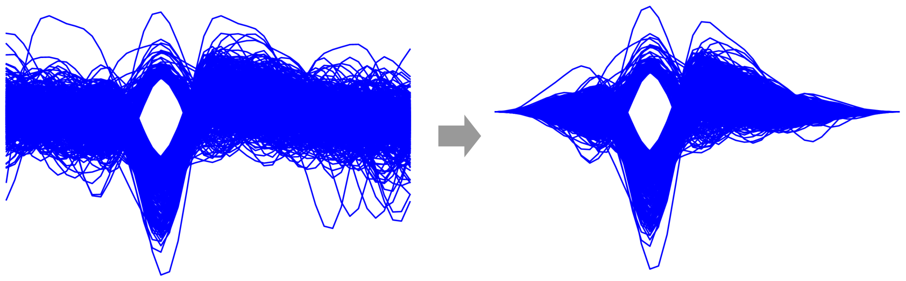


Figure 3.1: Spikes are from an MEA recording dataset. Original spikes versus spikes filtered by a Hanning window. Each spikes contains 48 samples. The figure shows that the noisy parts on the two sides of spike peaks are significantly attenuated, while the central parts of spikes are left almost intact.

hundred spikes.

3.2 Methods

3.2.1 `DidacticSort`

We designed a simple and effective spike sorting routine, `DidacticSort`, to aid users in labeling spikes semi-manually. Detected spikes are first aligned to their absolute peaks. Next, spikes are windowed by a customized Hanning filter *Spk_Hann* to attenuate the spike waveforms on the two sides of the peak, with stronger attenuations if farther away from the peak. *Spk_Hann* is created by concatenating the head and the tail parts of two conventional Hanning windows, as detailed by (3.1):

$$\begin{aligned} w_{head}(n) &= \frac{1}{2}(1 - \cos(\frac{2\pi n}{2 \times N_{head} - 1})), \\ w_{tail}(n) &= \frac{1}{2}(1 - \cos(\frac{2\pi n}{2 \times N_{tail} - 1})), \\ Spk\_Hann &= \{w_{head}[1 : N_{head}], w_{tail}[-N_{tail} : end]\}. \end{aligned} \tag{3.1}$$

Windowing spikes is motivated by the heuristic that spikes belonging to different clusters can be mainly differentiated by the shapes of depolarization and repolarization; the rest are more random and susceptible to noise, which should be considered with less importance. The effect of windowing spikes is illustrated in Figure 3.1, showing that the noisy parts of spikes on the two ends are significantly attenuated while the central parts are almost intact.

`DidacticSort` works as follows: It processes spikes sequentially. At the beginning, a spike template is initialized as the first spike. A spike is first compared to all templates

Algorithm 2 `DidacticSort`

```
Require: Spike events $\mathbf{X} \in \mathbb{R}^{N \times p}$, distance threshold $dist\_thr$
Ensure: Number of clusters $k$, spike IDs $\mathbf{I} \in \mathbb{R}^{N}$, spike templates $\mathbf{S} \in \mathbb{R}^{k \times p}$
 1: $\tilde{\mathbf{X}} \leftarrow Spk\_Hann \circ \mathbf{X}$
 2: repeat
 3:     $k \leftarrow 1$, $\mathbf{I} \leftarrow \mathbf{1}$, $\mathbf{S} \leftarrow \tilde{\mathbf{X}}_{(1,:)}$
 4:     Set $dist\_thr$ manually
 5:     for $i \leftarrow 2$ to $N$ do
 6:         $\mathbf{x} \leftarrow \tilde{\mathbf{X}}_{(i,:)}$
 7:         $d_{1:k} \leftarrow L_2\_dist(\mathbf{x}, \mathbf{S})$
 8:         if $min(d_{1:k}) > dist\_thr$ then
 9:             $k \leftarrow k+1$, $\mathbf{I}_{(i)} \leftarrow k$
10:             $\mathbf{S} \leftarrow concatenate(\mathbf{S}, \mathbf{x})$
11:         else
12:             $min\_loc \leftarrow argmin(d_{1:k})$
13:             $\mathbf{I}_{(i)} \leftarrow min\_loc$
14:             $\mathbf{S}_{(min\_loc,:)} \leftarrow mean(\tilde{\mathbf{X}}_{(\mathbf{I}==min\_loc,:)})$
15: until $k_{min} \leq k \leq k_{max}$
16: return $k, \mathbf{I}_{1:N}, \mathbf{S}$
```

(only one at the beginning) to find the smallest Euclidean distance. If the distance is greater than a pre-defined threshold *dist_thr*, this spike is considered as the first member of a new cluster and also initializes the corresponding template; otherwise, the spike is assigned to the closest cluster, and the template of that cluster is updated by taking the average of all assigned spikes. After processing all spikes, it is up to the users to evaluate the quality of clustering, and decide which clusters to keep or start a new round of clustering with a different *dist_thr*. It should be noted that *dist_thr* is the only tunable parameter to achieve satisfying results, similar to the bandwidth parameter in Mean Shift [258]. Intuitively, a large *dist_thr* would lead to a small number of "loosened" clusters, whereas a small *dist_thr* would lead to many "condensed" clusters. Experimental results show that the number of spike clusters is much less sensitive to *dist_thr* than the bandwidth parameter. The details of `DidacticSort` are given in Algorithm 2, where "$\circ$" denotes element-wise multiplication.

3.2.2 Few-Shot Spike Sorting (`FSSS`)

At the core of the proposed model is an autoencoder [220], a neural network structure widely used to learn compact data representations by forcing outputs to be identical to inputs and imposing constraints over the latent space. More formally, the operations of a vanilla autoencoder can be described as $\hat{\mathbf{x}} = g_s(g_a(\mathbf{x}; \boldsymbol{\phi}); \boldsymbol{\theta})$, where $\mathbf{x}$ and $\hat{\mathbf{x}}$ are input and output data; g_a and g_s denote *analysis* and *synthesis*, respectively, or are more commonly referred to as encoder and decoder (parameterized by $\boldsymbol{\phi}$ and $\boldsymbol{\theta}$).

A deficiency of vanilla autoencoder if used as a generative model is that it can only map a fixed input $\mathbf{x}$ to a fixed low-dimensional vector $\mathbf{z}$. In other words, the only source of stochasticity of the latent variables $\mathbf{z}$ comes from the input $\mathbf{x}$, which restricts the generative capability of the encoder network. We therefore propose to use variational autoencoder (VAE) [259] that formulates the encoder as a generative function $p_{\boldsymbol{\phi}}(\mathbf{z}|\mathbf{x})$ and the decoder as a likelihood function $p_{\boldsymbol{\theta}}(\mathbf{x}|\mathbf{z})$. As direct computations of the true posterior $p_{\boldsymbol{\phi}}(\mathbf{z}|\mathbf{x}) = p_{\boldsymbol{\phi}}(\mathbf{x}|\mathbf{z})p_{\boldsymbol{\phi}}(\mathbf{z})/p_{\boldsymbol{\phi}}(\mathbf{x})$ is intractable, we approximate it with an isotropic Gaussian $q_{\boldsymbol{\phi}}(\mathbf{z}|\mathbf{x}) = \mathcal{N}(\mathbf{z};\boldsymbol{\mu},\boldsymbol{\sigma}^2\mathbf{I})$. To facilitate the representation of $q_{\boldsymbol{\phi}}(\mathbf{z}|\mathbf{x})$ through the encoder network g_a and the training of g_a with back-propagation, the latent variable $\mathbf{z}$ can be reparameterized as $\tilde{\mathbf{z}} = \boldsymbol{\mu} + \boldsymbol{\sigma}\boldsymbol{\epsilon}$, where the noise variable $\boldsymbol{\epsilon} \sim \mathcal{N}(\mathbf{0},\mathbf{I})$, and both $\boldsymbol{\mu}$ and $\boldsymbol{\sigma}$ can be derived by the encoder function g_a from input $\mathbf{x}$. In so doing, the stochasticity of the latent variable $\mathbf{z}$ comes not only from $\mathbf{x}$, but also from the random Gaussian noise $\boldsymbol{\epsilon}$, thereby overcoming the limitation of fixed mapping. The VAE parameters $\{\boldsymbol{\phi},\boldsymbol{\theta}\}$ can be updated by minimizing the following loss function:

$$\mathcal{L}_{VAE} = D_{KL}(q_{\boldsymbol{\phi}}(\mathbf{z}|\mathbf{x})||p_{\boldsymbol{\theta}}(\mathbf{z})) - \mathbb{E}_{q_{\boldsymbol{\phi}}(\mathbf{z}|\mathbf{x})}(\log p_{\boldsymbol{\theta}}(\mathbf{x}|\mathbf{z})), \tag{3.2}$$

where $p_{\boldsymbol{\theta}}(\mathbf{z})$ is the prior of $\mathbf{z}$, conveniently assumed Gaussian $\mathbf{z} \sim \mathcal{N}(\mathbf{0},\mathbf{I})$; the KL-divergence can be computed in closed-form as both $q_{\boldsymbol{\phi}}(\mathbf{z}|\mathbf{x})$ and $p_{\boldsymbol{\theta}}(\mathbf{z})$ are Gaussian distributions; the second term is equivalent to the Euclidean reconstruction loss. In this work, we adopt a smoothed L_1 loss defined as follows:

$$\begin{aligned}\text{Loss}(\mathbf{x},\mathbf{x}') &= \frac{1}{N}\sum_i z_i,\\ z_i &= \begin{cases}\frac{1}{2}(x_i - x_i')^2, & \text{if } |x_i - x_i'| < 1.\\ |x_i - x_i'| - \frac{1}{2}, & \text{otherwise.}\end{cases}\end{aligned} \tag{3.3}$$

as a replacement of the Euclidean distance, which can provide more appropriate amount of gradient during back-propagation and is more robust to outliers.

In this work, one central task of the VAE model is to infer the label information $\mathbf{y}$ from spikes along with $\boldsymbol{\mu}$ and $\boldsymbol{\sigma}$. As shown in Figure 3.2, the VAE encoder $q_{\boldsymbol{\phi}}(\mathbf{y},\mathbf{z}|\mathbf{x})$ takes spikes $\mathbf{x}$ as inputs and outputs 1) a one-hot vector $\mathbf{y}$ through a softmax layer that predicts cluster labels, and 2) $\boldsymbol{\mu}$ and $\boldsymbol{\sigma}$ that specify the Gaussian distribution of $\mathbf{z}$. The decoder

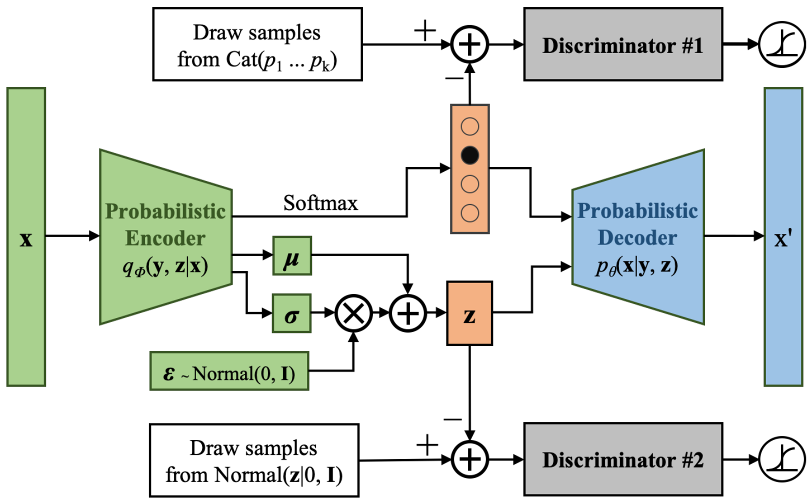

Draw samples from Cat(p_1 ... p_k)
+
−
Discriminator #1
x
Probabilistic Encoder q_Φ(y, z|x)
Softmax
μ
σ
ε ~ Normal(0, I)
z
Probabilistic Decoder p_θ(x|y, z)
x'
−
+
Draw samples from Normal(z|0, I)
Discriminator #2

Figure 3.2: Diagram of the proposed `FSSS` model based on adversarial autoencoder. The two discriminators are implemented as feed-forward neural networks, both of which output a probability through a sigmoid unit.

$p_{\boldsymbol{\theta}}(\mathbf{x}|\mathbf{y},\mathbf{z})$ takes both $\mathbf{y}$ and $\mathbf{z}$ as inputs and is practically a replication of the encoder in the reverse order to reconstruct the original spikes $\mathbf{x}$. Our objective is to disentangle $\mathbf{y}$ from $\mathbf{z}$ such that we could use the limited label information to regularize and derive the posterior distribution $q_\phi(\mathbf{y}|\mathbf{x})$.

To do that, we resort to the generative adversarial networks (GAN), a framework that establishes a min-max adversarial game between a generative model G and a discriminative model D [260]. G generates data from random samples $\mathbf{z}$ subject to a prior $p(\mathbf{z})$, and D estimates the probability that a sample comes from the actual data distribution $\mathbf{x} \sim p_{data}$ instead of G. The purpose is to train G to maximize the probability of D making mistakes:

$$\min_G \max_D \mathrm{E}_{\mathbf{x}\sim p_{data}}[\log D(\mathbf{x})] + \mathrm{E}_{\mathbf{z}\sim p(\mathbf{z})}[\log(1 - D(G(\mathbf{z})))]. \tag{3.4}$$

In the context of spike sorting, the encoder $q_{\boldsymbol{\phi}}(\mathbf{y},\mathbf{z}|\mathbf{x})$ is the generator G that maps spikes $\mathbf{x}$ to the labels $\mathbf{y}$, as well as the mean $\boldsymbol{\mu}$ and the variance $\boldsymbol{\sigma}$. Two discriminative networks, $D_{\mathbf{y}}$ and $D_{\mathbf{z}}$, are optimized to differentiate $\mathbf{y}$ and $\mathbf{z}$ from the true samplings $\mathbf{y}_D$ drawn from a categorical distribution, and $\mathbf{z}_D$ from a d-dimensional Gaussian distribution:

$$\mathbf{y}_D \sim \mathrm{Cat}(p_1, p_2, \ldots, p_k) \text{ and } \mathbf{z}_D \sim \mathcal{N}(\mathbf{0}, \mathbf{I}), \tag{3.5}$$

where k is the number of clusters obtained from performing `DidacticSort`, and $p_1, p_2, \ldots, p_k$ denote the event probabilities of the categorical distribution, representing the proportions of

Algorithm 3 FSSS

```
Require: Labeled spikes $\hat{\mathbf{X}} \in \mathbb{R}^{N_1 \times p}$ and IDs $\hat{\mathbf{I}} \in \mathbb{R}^{N_1}$, unlabeled spikes $\mathbf{X} \in \mathbb{R}^{N_2 \times p}$
Ensure: IDs of unlabeled spikes $\mathbf{I} \in \mathbb{R}^{N_2}$
 1: repeat
 2:     // Reconstruction
 3:     Sample $\mathbf{X}_{n_2 \times p}$ randomly from $\mathbf{X}_{N_2 \times p}$
 4:     $\mathbf{X}'_{n_2 \times p} \leftarrow g_s(g_a(\mathbf{X}_{n_2 \times p}; \phi); \theta)$
 5:     $\mathcal{L}_{recon} \leftarrow Smooth\_L_1\_dist(\mathbf{X}'_{n_2 \times p}, \mathbf{X}_{n_2 \times p})$
 6:     Update $g_a$ and $g_s$ with $\nabla_\phi \mathcal{L}_{recon}$ and $\nabla_\theta \mathcal{L}_{recon}$
 7:     // Regularization
 8:     Draw $\mathbf{Z}_{n_2 \times d} \sim \mathcal{N}(0, \mathbf{I})$, and $\mathbf{Y}_{1:n_2} \sim \mathrm{Cat}(\mathbf{y})$
 9:     $\mathbf{Z}'_{n_2 \times d}$, $\mathbf{Y}'_{1:n_2} \leftarrow g_a(\mathbf{X}_{n_2 \times p}; \phi)$
10:     $\mathcal{L}_{gauss} \leftarrow -mean(\log(D_{\mathbf{z}}(\mathbf{Z}_{n_2 \times d})) + \log(1 - D_{\mathbf{z}}(\mathbf{Z}'_{n_2 \times d})))$
11:     $\mathcal{L}_{cat} \leftarrow -mean(\log(D_{\mathbf{y}}(\mathbf{Y}_{1:n_2})) + \log(1 - D_{\mathbf{y}}(\mathbf{Y}'_{1:n_2})))$
12:     Update $D_{\mathbf{z}}$ and $D_{\mathbf{y}}$ with $\nabla \mathcal{L}_{gauss}$ and $\nabla \mathcal{L}_{cat}$
13:     $\mathcal{L}_{reg} \leftarrow -mean(\log(D_{\mathbf{z}}(\mathbf{Z}'_{n_2 \times d}))) - mean(\log(D_{\mathbf{y}}(\mathbf{Y}'_{1:n_2})))$
14:     Update $g_a$ with $\nabla_\phi \mathcal{L}_{reg}$
15:     // $Semi - supervision$
16:     Sample $\hat{\mathbf{X}}_{n_1 \times p}$ randomly from $\hat{\mathbf{X}}_{N_1 \times p}$
17:     $\hat{\mathbf{Z}}'_{n_1 \times d}$, $\hat{\mathbf{Y}}'_{1:n_1} \leftarrow g_a(\hat{\mathbf{X}}_{n_1 \times p}; \phi)$
18:     Update $g_a$ with $\nabla_\phi Cross\_Entropy(\hat{\mathbf{Y}}'_{1:n_1}, \hat{\mathbf{I}}_{1:n_1})$
19: until $\hat{\mathbf{Y}}'_{1:N_1} \approx \hat{\mathbf{I}}_{1:N_1}$
20: $\mathbf{Z}_{N_2 \times d}, \mathbf{I}_{1:N_2} \leftarrow g_a(\mathbf{X}_{N_2 \times p}; \phi)$
21: return $\mathbf{I}_{1:N_2}$
```

each cluster among the total labeled spikes. We should also note that samples $\mathbf{z}_D \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$ are independent to $\boldsymbol{\epsilon} \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$.

Therefore, the key to successful spike sorting using the model structure given in Figure 3.2 is to first train the encoder $q_{\boldsymbol{\phi}}(\mathbf{y}, \mathbf{z}|\mathbf{x})$ (also the decoder $p_{\boldsymbol{\theta}}$) with unlabeled spikes such that it can generate $\mathbf{y}$ and $\mathbf{z}$ that are sufficiently close to $\mathbf{y}_D$ and $\mathbf{z}_D$, respectively, followed by using a small amount of spikes labeled by `DidacticSort` to guide the parameter updating of the encoder $q_{\boldsymbol{\phi}}(\mathbf{y}, \mathbf{z}|\mathbf{x})$ hence it can correctly predict labels of input spikes. The algorithmic procedure of `FSSS` can be split into three phases:

- *Reconstruction:* The autoencoder is trained with an unlabeled mini-batch to update the encoder and the decoder by minimizing the reconstruction error measured in smoothed L_1 distance.

- *Regularization:* $D_{\mathbf{y}}$ and $D_{\mathbf{z}}$ are updated by differentiating true samples generated by the categorical and Gaussian distributions from the fake samples generated by

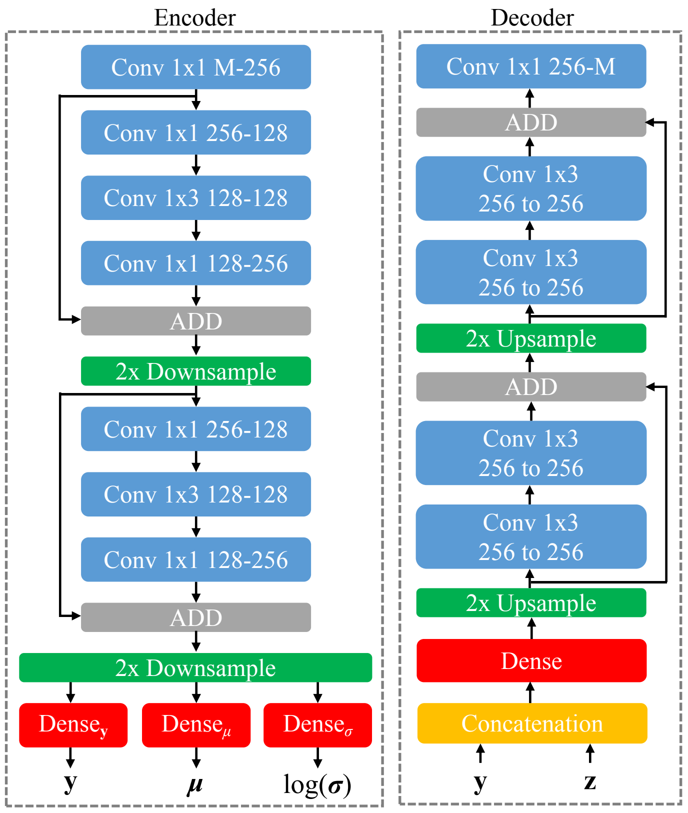


Figure 3.3: Structural diagram of the encoder and the decoder networks. Activation layers and normalization layers are skipped from the diagram for simplicity. M is the dimension of input channels, which is by default 1.

the encoder. This is followed by updating the encoder to confuse the discriminative networks.

- *Semi-supervision:* The encoder $q_\phi(\mathbf{y}, \mathbf{z}|\mathbf{x})$ is updated by minimizing the cross-entropy cost on the mini-batch of spikes labeled by `DidacticSort`.

The detailed mathematical description of `FSSS` is given in Algorithm 3.

3.2.3 Encoder and Decoder Networks

The detailed diagram of the proposed autoencoder is shown in Figure 3.3. The design of the encoder and decoder is almost identical to that for CAE discussed in Chapter 2.4. The last stage of the encoder network contains three separate dense layers that maps the current latent feature to the labels $\mathbf{y}$, the mean $\boldsymbol{\mu}$, and the variance $\boldsymbol{\sigma}$. The decoder network is a reverse implementation of its encoder counterpart, where convolution and downsampling are replaced by transposed convolution (deconvolution) and upsampling.

3.3 Experiments

3.3.1 Baselines

We prepared two baseline methods to compare with the proposed spike sorting algorithm. The first is the classic, unsupervised, wavelet-based clustering algorithm `Wave_Clus` from [4]; the second is a DNN-based model, which is essentially the encoding network of the model structure of `FSSS` and termed `Base_NN`. When using the encoder as a stand-alone classifier, we ignore the encoder outputs $\boldsymbol{\mu}$ and $\boldsymbol{\sigma}$, and take $\mathbf{y}$ as the predicted labels. Training of the classifier is the same as the semi-supervision step as shown in Algorithm 3. The motivation for comparing with `Base_NN` is to demonstrate the advantage of the adversarial learning built upon the VAE structure that enables high-quality few-shot learning.

We run all experiments on a system equipped with Intel i7-6800K@3.40 GHz, NVIDIA GeForce Titan Xp 12GB, 32GB memory, and 256GB SSD. Both `FSSS` and `Base_NN` are implemented using the deep learning framework PyTorch 1.0.1 (with CUDA 10.0) [231]. We used the ADAM optimizer [232] with a learning rate 1e-4, and evaluated the model performance after 150 epochs with a batch-size 64.

3.3.2 Sorting Synthetic Neural Data

The synthetic dataset we used is `Wave_Clus` [4]. This collection of datasets is generated by inserting multiple instances of several spike templates to continuous background noise of various levels, thus realizing different signal-to-noise ratios (SNRs). `Wave_Clus` consists of four groups of datasets, with each group synthesized using a unique set of spike templates and a number of different background noise levels. We chose datasets with relatively high background noise levels from each of the four groups to more clearly differentiate the performance of tested methods, resulting in seven datasets: `C_easy1` (0.3, 0.35, 0.4), `C_easy2` (0.15, 0.2), `C_difficult1` (0.15, 0.2), and `C_difficult2` (0.15, 0.2). All the selected datasets contain overlapping spikes, which makes spike sorting more challenging.

Spikes were extracted using the ground truth timestamps included in the datasets, with 64 samples per spike. As the spike labels and number of clusters are known *a priori*, we skipped the step of `DidacticSort` on the selected datasets. For each dataset, we first excluded overlapping spikes, and compared the spike sorting performance of `Wave_Clus`, `Base_NN`, and `FSSS`; we then re-run the comparison between `Base_NN` and `FSSS` on the

Table 3.1: Comparison of spike sorting performance between `Wave_Clus`, a baseline neural network, and the proposed `FSSS` on the `Wave_Clus` dataset (w/ and w/o overlapping spikes). The performance of `Wave_Clus` is from [4]. Both `Base_NN` and `FSSS` used 100 labeled spikes that are randomly chosen.

	noise level	No. of spikes (w/o o.l.)	`Wave_Clus`	`Base_NN`	`FSSS`	No. of spikes (w/ o.l.)	`Base_NN`	`FSSS`
C_easy1	0.3	2629	89.50%	97.23%	**99.96%**	3475	94.13%	**99.29%**
	0.35	2702	82.12%	95.42%	**99.44%**	3534	95.22%	**98.84%**
	0.4	2645	71.98%	92.43%	**98.07%**	3386	91.51%	**98.78%**
C_easy2	0.15	2648	98.30%	96.42%	**99.77%**	3411	95.77%	**98.40%**
	0.2	2715	88.73%	96.13%	**99.57%**	3526	88.24%	**98.98%**
C_difficult1	0.15	2660	96.95%	83.78%	**97.33%**	3472	82.03%	**93.21%**
	0.2	2624	75.19%	71.60%	**91.55%**	3414	66.45%	**90.59%**
C_difficult2	0.15	2631	83.16%	94.58%	**99.61%**	3440	94.40%	**99.13%**
	0.2	2716	46.17%	85.03%	**99.81%**	3493	85.76%	**99.17%**

complete dataset that includes overlapping spikes. We used the same 100 labeled spikes that were randomly chosen from the dataset to supervise the training of `Base_NN` and `FSSS`.

The results are given in Table 3.1. When processing non-overlapping spikes, `FSSS` outperformed both `Wave_Clus` and `Base_NN` by large margins especially on more challenging datasets. For example, `FSSS` was over 50% better than `Wave_Clus` on `C_difficult2` (0.2), and nearly 20% better than `Base_NN` on `C_difficult1` (0.2). `Base_NN` achieved better performance than `Wave_Clus` in general, except for `C_easy2` (0.15) and `C_difficult1` (0.15, 0.2). When processing a mixture of both overlapping and non-overlapping spikes, `FSSS` achieved consistently better results than `Base_NN` by 2.6∼24.1%.

3.3.3 Improvements of `FSSS` over `Base_NN`

Here we are interested in quantifying the improvement of `FSSS` over `Base_NN` for sorting spikes in a different way than that used in Table 3.1. Specifically, we investigate how many more labeled spikes, in addition to the 100 common spikes, `Base_NN` needs to achieve sorting performance on par with `FSSS`. This measurement can give a more intuitive estimation of the effort saved by `FSSS` on collecting and labeling training data.

We chose the dataset with the highest background noise level from each of the four groups. For each dataset, we started from 100 labeled spikes for training `Base_NN` and recorded the classification accuracy; we then repeated the following step, by expanding the training dataset with another 100 randomly selected labeled spikes (different from existing training spikes) and re-training `Base_NN`, until the testing performance of `Base_NN` exceeded that of `FSSS` obtained on the very first 100 labeled spikes.

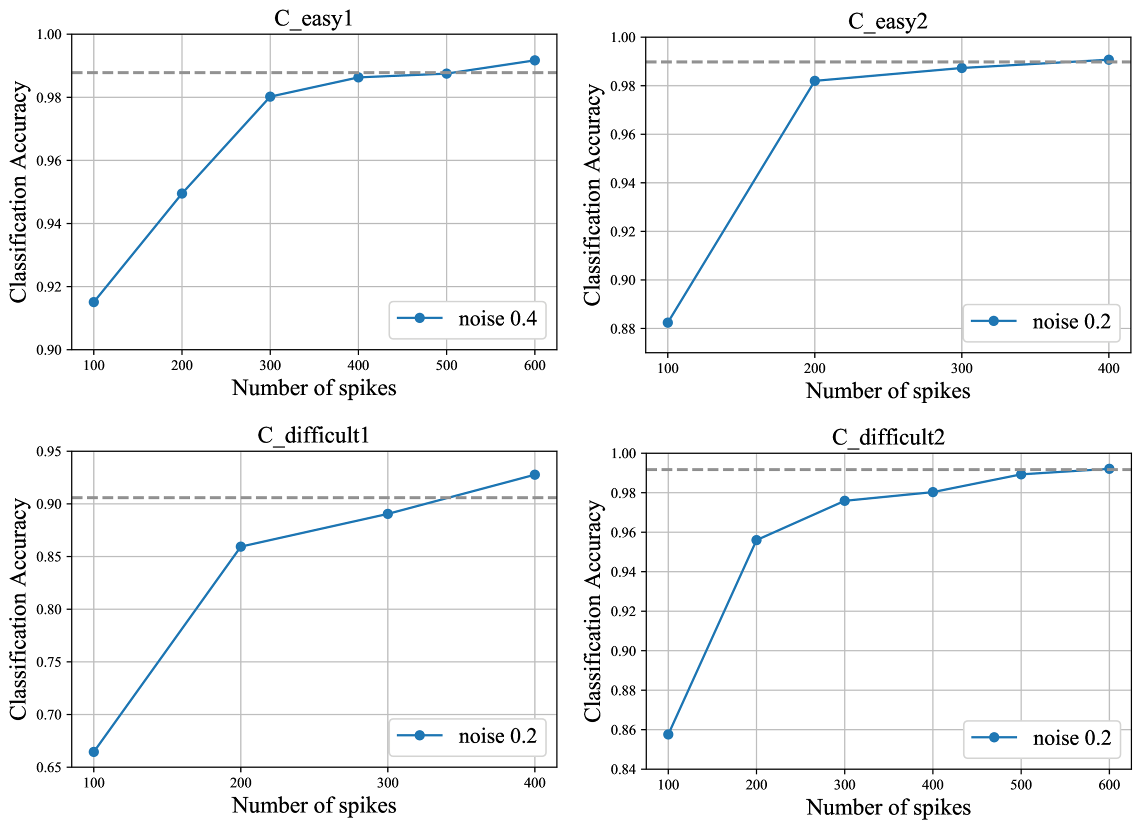


Figure 3.4: Number of additional labeled spikes required by `Base_NN` to catch up with `FSSS` on classification accuracy. For each dataset, the volume of training spikes increments by 100 spikes each time.

The results are given in Figure 3.4. In general, `FSSS` needs much less labeled spikes to achieve the same level of classification accuracy. Specifically, `Base_NN` took additional over 400 spikes to catch up with `FSSS` on `C_easy1` and `C_difficult2`, and around 200 spikes on `C_easy2` and `C_difficult1`. On all datasets, the first additional 100 spikes contributed the most significant raise of classification accuracy. It should be noted that the actual amount of additional labeled spikes required by `Base_NN` to catch up with `FSSS` could be more. A recent study [261] suggests that there exists *forgettable events* in the training dataset, defined as those whose predicted labels fluctuate during training, that determine the classification margin in a way similar to the role of support vectors in support vector machine. For a relatively simple, small-scale dataset like `Wave_Clus`, its forgettable events may constitute a small proportion of the entire dataset, and thus might be insufficient to gauge the difference between `FSSS` and `Base_NN` on this regard.

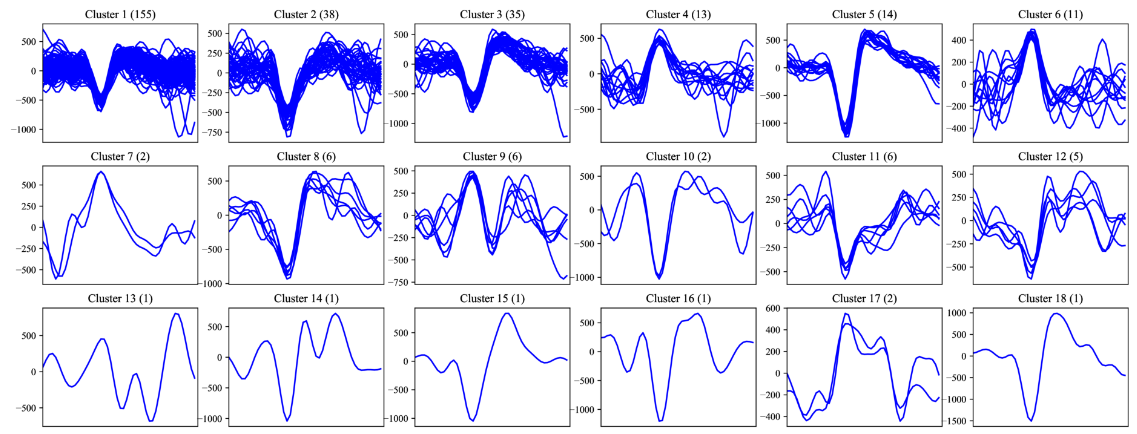


Figure 3.5: `DidacticSort` on 300 spikes from MEA data (*dist_thr* = 4). The numbers in parentheses are the spike count of each cluster. Each spike contains 48 samples. The vertical axis of each sub-figure is in μV. Ten clusters (1, 2, 3, 4, 5, 6, 8, 9, 10, 11, 12) are kept.

3.3.4 Sorting Neural Data Recorded with MEA

We also tested the proposed algorithm on *in vitro* data recorded with a multielectrode array (MEA). Mouse hippocampal neurons were cultivated on transparent commercial MEA arrays. Neurophysiological data recordings were carried out on DIV11 and DIV12 with 20 kHz sampling rate. The MEA arrays were mounted on the recording hardware located outside of the incubator. The recording was maximized to be 10 minutes to prevent pH changes.

We selected one channel data with active spiking activities from a 10-minute recording for demonstration. 3064 spikes were detected from this data sequence using a median-based spike detection method [262], with 48 samples per spike. Figure 3.5 shows the 18 clusters found by `DidacticSort` from 300 randomly chosen spikes with *dist_thr* set as 4. We chose the 10 clusters (292 spikes in total) that contain 5 spikes or more, and kept these labeled spikes for next-step processing. The criterion of choosing clusters is at the user's choice. The proportions of the spike counts of these clusters are used to initialize the event probability of the categorical distribution in `FSSS`, which can in practice accelerate convergence of model training compared with a uniform initialization.

Next we trained the deep learning model of `FSSS`. We found the resulting model quite robust to the latent dimension of the autoencoder, which can be arbitrary from 3 to 10.

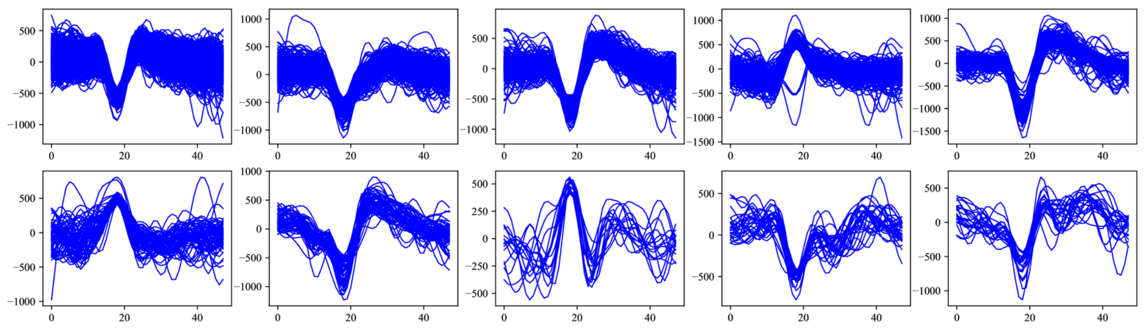


Figure 3.6: Ten sorted clusters on all the unlabeled spikes processed by `FSSS`. The vertical axis of each sub-figure is in μV. A few clusters contain several outliers that are not labeled by `DidacticSort`.

With ~300 labeled spikes, the training requires around 2 minutes on a server with one Intel i7-6800K, one GeForce Titan Xp 12 GB, and 32 GB memory. Deployed in inference mode, the model can sort spikes at a speed of over 200,000 spikes/second on one GPU, which could support hundreds of thousands of channels at the same time, and facilitate large-scale neural signal processing.

Figure 3.6 shows the ten clusters found by `FSSS` from all the unlabeled spikes, which correspond to the sample clusters in Figure 3.5 identified by `DidacticSort`. Some of the sample clusters contain very few spikes, yet `FSSS` can reliably recognize and classify similar unseen spikes by exploiting the limited supervised information. It should be noted that this is fundamentally different from template matching in which a new spike is exhaustively compared with every template to find the nearest cluster; instead, `FSSS` learns a parametric function that characterizes the statistical distributions of clusters, thereby encoding the label information of spikes analytically. It should also be noted that there are a few outliers in some of the clusters, which mostly came from the left-out spikes by `DidacticSort`. Given the rare occurrences of the outliers, their impact on the downstream tasks such as decoding is minimal. Figure 3.7 shows the spike templates of the 10 identified clusters.

3.4 Summary

In this chapter, we propose a new spike sorting paradigm that consists of two algorithms, `DidacticSort` and `FSSS`. `DidacticSort` is a simple classification routine with only one tunable parameter that can quickly generate candidate clusters using a small number of

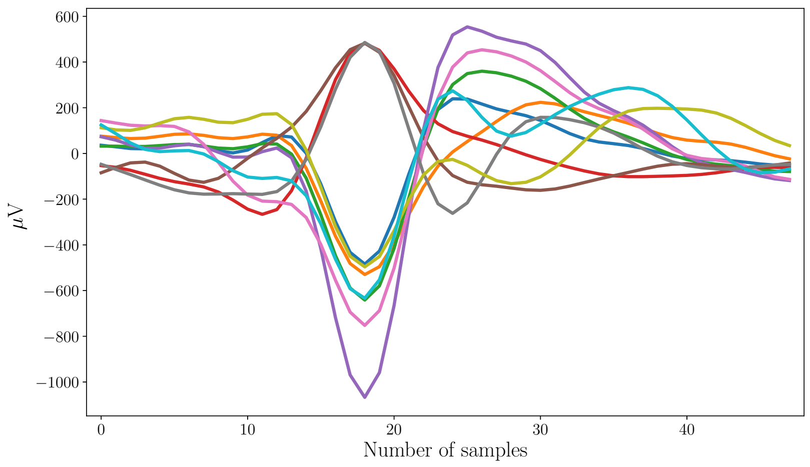


Figure 3.7: Spike templates of the ten clusters.

spike samples. It keeps a minimum level of pre-processing of spikes and so as the influence of users' decision making. `FSSS` features the learning capability from a small amount of labeled samples and generalizing the learned knowledge to many unseen events for unsupervised clustering. Combined together, `FSSS` can imitate the way `DidacticSort` clusters spikes that encompasses human's decision making. The proposed paradigm brings several useful features to the development of spike sorting: 1) Human knowledge can be better utilized as *guiding* (prior instructive information) instead of merely *intervention* (post-processing) to achieve more reasonable spike sorting results; 2) `FSSS` can learn a parametric function that encodes the categorical distribution of spike clusters analytically, thus can avoid iterative computations and easily be accelerated by GPUs to facilitate online, large-scale neural signal processing in real time; 3) The paradigm only requires a small number of spikes for labeling & model training, and can perform robustly on large amounts of unseen data.

Chapter 4

Graph-Based Hierarchical Representation Learning of EHR

4.1 Overview

Recent years have seen an explosion in the growth of electronic health record (EHR) data, which has motivated extensive use of machine learning methods, in particular deep learning, in tasks such as diagnosis prediction [263, 155], risk prediction [113, 264], and patient subtyping [122, 265]. Under the hood, all these tasks involve some form of neural networks that first learn features or patterns from data, and then make predictions.

One major challenge of representation learning in EHR comes from the heterogeneity of the various medical entities that compose EHR data, including diagnoses, prescriptions, lab test results, medical procedures, doctor profiles, and patient demographics, etc., that are a mixture of tabular values, text notes, and medical codes. Furthermore, the relational and longitudinal structure of organizing medical entities in patient medical records (or *patient journeys*) makes it more challenging to design effective and scalable representation learning algorithms: as illustrated in Table 4.1, a patient may visit one or more clinical sites multiple times with irregular time intervals, with each visit generating a varying number of medical service records (diagnoses, prescriptions, or procedures) from possibly different doctors. In addition, hospitals or clinical institutes may use different or in-house medical coding systems. As a result, one medical service could have multiple codes, which challenges the transferability of representations learned from individual systems.

Table 4.1: Snippet of a patient journey. 'dx' and 'px' represent diagnosis and medical procedure, respectively.

Day	Doctor	Medical Service
1	A	dx-*Iron Deficiency Anemias*
15	A	px-*Chemotherapy*
	A	px-*Infusions*
	B	px-*Injectable Chemotherapy Drugs*
	B	dx-*Antineoplastic Chemotherapy*
	A	dx-*Antineoplastic Chemotherapy*
55	A	px-*Chemotherapy*
	A	px-*Infusions*
	A	px-*Injectable Chemotherapy Drugs*
	A	dx-*Antineoplastic Chemotherapy*
	A	dx-*Neutropenia*

To address the above challenges, several related works have been proposed. Choi et al. leveraged the multilevel structure of EHR data where diagnosis codes categorize treatment codes within each visit and learned a multilevel medical embedding for predictive healthcare [152, 153]. Though being effective, their approaches do not consider the temporal characteristics unique to individual medical services, hence cannot address the irregular time intervals of visits that are pervasive in patient journeys. To incorporate temporal features into EHR-based representation learning, some recent works treated medical services in patient journeys as words in documents [157, 266], and since similar words (medical services) tend to share similar contexts, word embedding techniques such as `Word2Vec` [127] can be adopted to train the embedding vectors of medical services. In this approach, a key design choice is the length of context window, or temporal scope, which should preferably vary for different medical services (for example, the influence of acute diseases has shorter time spans compared with that of chronic conditions). As manually specifying the temporal scope for each service is infeasible, an attention mechanism is proposed in [266] to derive a "soft" temporal scope for each service, where the attention coefficients can be trained jointly with the parameters in `Word2Vec`. A caveat of this approach is that the context window has to be sufficiently large for medical services with long time spans of influence, which would significantly elevate the computational overheads for all services.

In this work, we propose a unified and hierarchical medical entity embedding framework

ME2Vec that can simultaneously address all the aforementioned challenges. At the service level, we make two improvements over the recent works. First, we use the actual timestamps in patient journeys to choose a single temporal scope (no need to be excessively large) for all medical services, which drastically reduces computational loads without loss of specificity for individual services (thanks to the second improvement); Second, instead of learning medical service embeddings directly from patient journeys, we create a graph from patient journeys with each vertex being a service and each edge weight indicating the co-occurrence frequency of the two services. This allows us to estimate temporal distances between any pair of medical services in a probabilistic manner even if they are remote in patient journeys. The time-aware medical service embedding also improves transferability: while the specific code of a service might change in different systems, its temporal position relative to other services is primarily governed by disease progression and treatment decisions, which stays more robust to cross-system migration.

The other unique features of ME2Vec are the capabilities of embedding doctors and patients, motivated by the need to investigate patient similarity that is believed to be an enabling technique for various healthcare applications such as cohort analysis and personalized medicine [157, 267, 268]. A fundamental principle that we adhere to in designing the doctor and patient embedding is "*It's what you do that defines you*", which empowers the interpretability of embeddings. For example, the embedding vector of a doctor is solely calculated from the doctor's conducted medical services; similarly, the embedding vector of a patient is jointly determined by the embeddings of both visited doctors and received medical services. To preserve the network proximities of patient vertices with both doctor and service vertices, we develop a method called *duplication & annotation* that can convert an attributed multigraph to a simple graph without loss of structural information, on which efficient and scalable graph embedding techniques can be applied at ease.

Our contributions in this work include: (i) We proposed a hierarchical medical entity embedding framework as a comprehensive and general-purpose solution for representation learning of EHR data; (ii) We designed a time-aware service embedding that turns the irregular time intervals of medical services into their temporal characteristics through random-walk-based graph embedding; (iii) We proposed a *duplication & annotation* method to convert a multigraph with attributed edges to a simple graph, which facilitates learning patient embeddings from doctor and service embeddings.

4.2 Methods

4.2.1 Service Embedding

Insights We evaluate the design criteria of service embedding from the perspective of patient similarity. Intuitively, patients who have received the same medical services should be more similar. However, it must be used with caution. For example, two patients are not necessarily similar simply because they all have the diagnosis of *hypertension*, as their hypertension could be possibly caused by different and more severe medical conditions. In other words, routine services (e.g., *hypertension* or *blood counts measurement*) should be considered with less importance for evaluating patient similarity compared with more complicated services which are often infrequent in patient journeys.

In `Word2Vec`, the distance between two word embeddings reflects their co-occurrence frequency derived from a text corpus. Similarly, we can count the co-occurrence frequency of every pair of medical services by using a fixed-size context window from patient journeys. In analogy to words in a document following a semantic and grammatical order, the sequence of medical services in a patient journey is jointly determined by the patient's disease progression and doctors' treatment decisions. By applying `Word2Vec` or other contextual learning approaches, we can derive service embeddings that preserve the inter-service distances, wherein a small co-occurrence frequency corresponds to long distance, and *vice versa*. Therefore, we posit that in the embedding space, *complicated services should be distant from routine services, and also from each other.*

Medical service graph We first create the graph of medical services $\mathcal{G}_{svc} = (\mathcal{S}, \mathcal{E}_{svc})$, where $\mathcal{S} = \{s_1, s_2, \ldots, s_{|\mathcal{S}|}\}$ is the set of medical services, and $\mathcal{E}_{svc}$ is the set of edges connecting medical services. The weight of e_{ij} denotes the co-occurrence frequency of services s_i and s_j. The co-occurrence frequencies of every pair of services are exactly the elements of the adjacency matrix $\mathbf{A}_{svc} \in \mathbb{R}^{|\mathcal{S}|\times|\mathcal{S}|}$ of the service graph. To obtain $\mathbf{A}_{svc}$, we use a T-day context window to traverse all patient journeys with no overlap. At each location, we update $\mathbf{A}_{svc}$ with the count of the occurrence of each unique pair of medical services appeared within the T days of the current window by adding the count to the corresponding element of $\mathbf{A}_{svc}$.

Note that (i) the co-occurrence frequencies of services from different patients are summed together, thus reflecting a generalized knowledge of the time intervals between medical services, which can enhance the transferability of the learned service embedding; (ii) the choice of T serves as prior knowledge of scheduling various types of medical services, suggesting that we could choose different values of T for different diseases in need of learning disease-specific service embeddings; (iii) the choice of T also serves as a proxy to control the sparseness of $\mathbf{A}_{svc}$: a smaller T will lead to a sparser $\mathbf{A}_{svc}$, and *vice versa.*

Algorithmic details As we are interested in preserving temporal distances between medical services, a biased-random-walk-based embedding scheme such as `node2vec` [147] is a better choice than `Word2Vec`. In a biased random walk, the probability of jumping from node s_i to s_j is proportional to the edge weight between the two nodes. Formally, we have

$$P(s_j|s_i) = \frac{e_{s_i,s_j}}{\sum_{k\in\mathcal{N}_i} e_{s_i,s_k}}, \tag{4.1}$$

if node s_i and s_j are connected. A biased-random-walk-based embedding can allow for more accurate estimation of a node's location in a graph through biased random walks by generating "pseudo sequences" wherein service nodes of higher degree appear more frequently. In this work, we adopted `node2vec` in service embedding as it can provide extra tunable parameters to adjust redundant node sampling and also balance breadth-first and depth-first search.

The details of learning medical service embedding vectors are given in Algorithm 4. The `combinations` function lists all pairs of medical services within the segment $\mathcal{J}_{seg}^{(i)}$. To embed medical services, we first obtain the adjacency matrix $\mathbf{A}_{svc}$ from patient journeys and use it to generate biased random walks, then optimize the embeddings of medical services by maximizing the probability of each service "seeing" its neighbors in the walks via stochastic gradient descent (SGD).

4.2.2 Doctor Embedding

Insights We observe that medical services conducted by a doctor exhibit patterns that are consistent with the doctor's primary specialty. For example, medications and/or medical procedures administered by an *obstetrician* (or *gynecologist*) are in general different from those of an *oncologist.* This suggests that a doctor embedding should not only reflect the specific medical services the doctor has conducted, but also inform the doctor's primary specialty.

Algorithm 4 Medical service embedding

```
Require: Patient journeys $\{\mathcal{J}^{(i)}\}_{i=1}^{P}$, context window length $T$, dimension $p$, walks per
    node $r$, walk length $l$, context size $k$
Ensure: Service embedding $\mathbf{S} \in \mathbb{R}^{|\mathcal{S}| \times p}$
 1: $\mathbf{A}_{svc} \leftarrow \mathbf{0} \in \mathbb{R}^{|\mathcal{S}| \times |\mathcal{S}|}$
 2: for $i = 1$ to $P$ do
 3:     for $j = 1$ to $\lceil \frac{|\mathcal{J}^{(i)}|}{T} \rceil$ do
 4:         $\mathcal{J}_{seg}^{(i)} \leftarrow \mathcal{J}^{(i)}[(j-1)T : jT]$
 5:         for $s_x, s_y$ in combinations($\mathcal{J}_{seg}^{(i)}$, 2) do
 6:             $\mathbf{A}_{svc}[s_x, s_y]$ += 1
 7:             $\mathbf{A}_{svc}[s_y, s_x]$ += 1
 8: $\mathcal{E}_{svc} \leftarrow \mathbf{A}_{svc}$, $\mathcal{G}_{svc} \leftarrow \{\mathcal{S}, \mathcal{E}_{svc}\}$
 9: $walks \leftarrow \{\}$
10: for $iter = 1$ to $r$ do
11:     for all nodes $s \in \mathcal{S}$ do
12:         $walk \leftarrow$ BiasedRandomWalk($\mathcal{G}_{svc}, s, l$)
13:         Append $walk$ to $walks$
14: $\mathbf{S} \leftarrow$ SGD($k, p, walks$)
15: return $\mathbf{S}$
```

Instead of training doctor embedding in an unsupervised fashion as is the case for service embedding, we propose to train the embedding of a doctor in an auxiliary task by predicting the doctor's primary specialty from his or her conducted medical services. It is worth noting that this auxiliary task is a type of supervised learning as we can leverage the available knowledge of doctor primary specialty that is normally included in patient journeys. Another practical benefit of the supervised learning formulation is that we can reuse the learned mapping functions to predict missing doctor specialties (which is pervasive in many medical databases) according to their conducted medical services.

To account for that doctor embedding should reflect the specific medical services of each doctor, we initialize the embedding of a doctor as the weighted average of the embedding vectors of the medical services conducted by the doctor, such that the trained doctor embedding can be close to its associated medical services in the embedding space.

Graph attention network As the amount and type of unique medical services vary significantly for different doctors, we propose to use Graph Attention Network [269] to predict doctor specialties from services, as the attention mechanism naturally supports the mapping from a varying number of inputs to the output.

For a doctor d_j whose conducted medical services are $\{s_i\}^{(d_j)}$, the attention coefficient

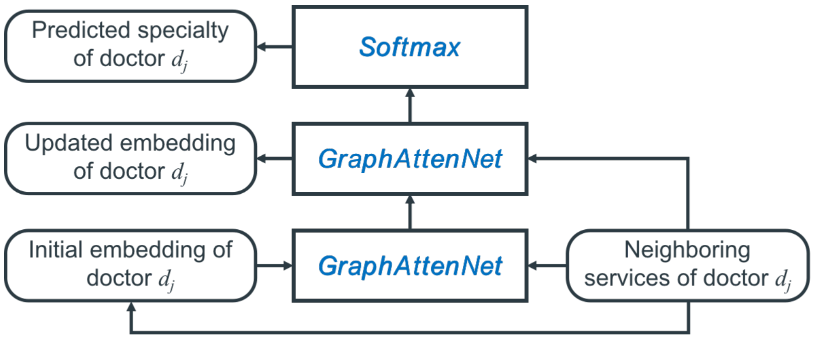

Predicted specialty of doctor d_j
Softmax
Updated embedding of doctor d_j
GraphAttenNet
Initial embedding of doctor d_j
GraphAttenNet
Neighboring services of doctor d_j

Figure 4.1: Structural diagram of a 2-level attention-based doctor embedding model. The embedding vectors of the medical services conducted by doctor d_j are pre-trained in the step of service embedding.

e_{ij} between the doctor embedding $\mathbf{d}_j$ and each of the service embeddings $\{\mathbf{s}_i\}^{(d_j)}$ conducted by doctor d_j is

$$e_{ij} = \texttt{LeakyReLU}\left(\mathbf{a}^T[\mathbf{W}\mathbf{d}_j||\mathbf{W}\mathbf{s}_i]\right), \tag{4.2}$$

where $\{\mathbf{d}, \mathbf{s}\} \in \mathbb{R}^p$, $\mathbf{a} \in \mathbb{R}^{2p'}$, $\mathbf{W} \in \mathbb{R}^{p' \times p}$, $\texttt{LeakyReLU}$ is the Leaky Rectified Linear Unit with a negative input slope of 0.2 [270], $\cdot^T$ represents transposition, and $||$ is the concatenation operation. $\{\mathbf{W}, \mathbf{a}\}$ are parameters of the aggregation functions that "aggregate" the information of neighboring service vertices into the targeted doctor vertex. After normalizing the attention coefficient through a $\texttt{softmax}$ layer, we obtain the final expression:

$$\alpha_{ij} = \frac{\exp\left(\texttt{LeakyReLU}(\mathbf{a}^T[\mathbf{W}\mathbf{d}_j||\mathbf{W}\mathbf{s}_i])\right)}{\sum_{s_k \in \mathcal{N}_{d_j}} \exp\left(\texttt{LeakyReLU}(\mathbf{a}^T[\mathbf{W}\mathbf{d}_j||\mathbf{W}\mathbf{s}_k])\right)}. \tag{4.3}$$

The updated embedding vector of doctor d_j can then be obtained as a linear combination of the associated service embeddings weighted by corresponding attention coefficients. To stabilize the learning process, we adopt a multi-head attention comprising K heads, such that the output dimension of the attention layer is Kp' instead of p'. The operation of the multi-head attention layer can be described as

$$\mathbf{d}'_j = \Big\Vert_{k=1}^{K} \sigma\left(\sum_{s_i \in \mathcal{N}_{d_j}} \alpha_{ij}^k \mathbf{W}^k \mathbf{s}_i\right), \tag{4.4}$$

where $||$ denotes concatenation. Note that we have already obtained $\mathbf{s}_i$, thus making the doctor embedding a simper task than ordinary graph embedding wherein the embeddings of all nodes are unknown and to be learned.

The structure of the two-level attention-based doctor embedding model is shown in Figure 4.1. As the proposed auxiliary task is a supervised classification, we configure the

Algorithm 5 Doctor embedding

```
Require: Patient journeys $\{\mathcal{J}^{(i)}\}_{i=1}^{P}$, service embedding $\mathbf{S}$, number of attention heads $K$,
    learning rate $\eta$
Ensure: Doctor embedding $\mathbf{D} \in \mathbb{R}^{|\mathcal{D}| \times p'}$, aggregation functions $\{\mathbf{W}^k, \mathbf{a}^k\}_{k=1}^{K}$
 1: $L_{gt} \leftarrow \mathcal{D} \leftarrow \mathcal{G}_{doc} \leftarrow \{\mathcal{J}^{(i)}\}_{i=1}^{P}$
 2: for all node $d \in \mathcal{D}$ do
 3:     $\mathbf{d}_{init} \leftarrow \frac{1}{|\mathcal{N}_d|} \sum_{i=1}^{|\mathcal{N}_d|} \mathbf{s}_i \ (s_i \in \mathcal{N}_d)$
 4: $\mathbf{D}' \leftarrow$ GraphAttenNet-2L$(\mathbf{D}_{init}, \mathbf{S})$
 5: $L_{pred} \leftarrow$ softmax$(\mathbf{D}')$
 6: while CrossEnt$(L_{pred}, L_{gt})$ is large do
 7:     $\mathbf{D}' \leftarrow$ GraphAttenNet-2L$(\mathbf{D}_{init}, \mathbf{S})$
 8:     $L_{pred} \leftarrow$ softmax$(\mathbf{D}')$
 9:     $\{\mathbf{W}^k, \mathbf{a}^k\} \leftarrow \{\mathbf{W}^k, \mathbf{a}^k\} - \eta\nabla($CrossEnt$(L_{pred}, L_{gt}))$
10: $\mathbf{D} \leftarrow$ GraphAttenNet$(\mathbf{D}_{init}, \mathbf{S})$
11: return $\mathbf{D}$, $\{\mathbf{W}^k, \mathbf{a}^k\}_{k=1}^{K}$
```

final layer of the attention-based doctor embedding model as a `softmax`. In addition, following the practice proposed in [269], the output embedding from each of the K attention heads of the second `GraphAttenNet` are averaged instead of concatenated, followed by the final nonlinearity transformation, as shown in Eq. 4.5

$$\mathbf{d}'_j = \sigma\left(\frac{1}{K}\sum_{k=1}^{K}\sum_{s_i \in \mathcal{N}_{d_j}} \alpha_{ij}^k \mathbf{W}^k \mathbf{s}_i\right). \tag{4.5}$$

Algorithmic details We first create a bipartite graph $\mathcal{G}_{doc}$ consisting of two sets of vertices, doctors $\mathcal{D} = \{d_1, d_2, \ldots, d_{|\mathcal{D}|}\}$ and $\mathcal{S} = \{s_1, s_2, \ldots, s_{|\mathcal{S}|}\}$, from patient journeys. $\mathcal{E}_{doc}$ is the set of edges connecting the two sets of vertices, where the weight of each edge represents the number of times that doctor d_j has conducted service s_i.

Next, we initialize the embedding vector of each doctor vertex as the weighted average of the embedding vectors of its connected service vertices. After that, we update the doctor embeddings using Eqs. (4.3)-(4.5) by predicting the primary specialty of each doctor. Finally, we obtain the aggregation functions parameterized by $\{\mathbf{W}^k, \mathbf{a}^k\}_{k=1}^{K}$ that can be used to derive embeddings of not only doctor vertices already in the patient journeys, but also new doctor vertices that might be added in the future. We summarize the steps for doctor embedding in Algorithm 5, where `GraphAttenNet-2L` denotes the operations of two `GraphAttenNet` stacked together; `CrossEnt` denotes cross-entropy loss; L_{gt} and L_{pred} represent the ground-truth and the predicted doctor specialties, respectively.

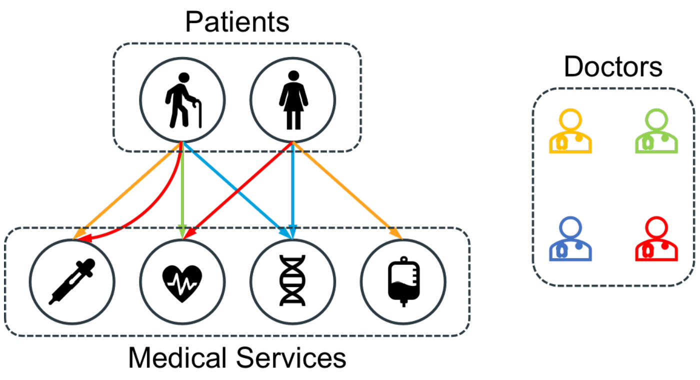


Figure 4.2: A toy example showcasing the complex similarity relations of patients with both doctors and medical services. An edge denotes the patient has received the service the edge connects to. The color of an edge denotes the particular doctor who has conducted that service.

4.2.3 Patient Embedding

Insights The similarity between patients can be defined from the perspectives of shared doctors and/or shared services. In general, we expect the patient embedding can facilitate that *patients are more similar to each other if they receive the same medical services from the same doctors.*

Following this guideline, patient similarity can be categorized into: (i) different patients receive the same services from the same doctor; (ii) different patients receive the same services from different doctors; (iii) different patients receive different services from the same doctor; (iv) one patient receives the same service multiple times from different doctors. We illustrate these scenarios in Figure 4.2, where an arrow indicates a patient (starting node) has received a service (ending node) from a doctor (the color of the arrow).

The versatile forms of patient similarity can be formalized as a *bipartite multigraph* $\mathcal{G}_{pat}$, where the two disjoint sets of vertices ($\mathcal{P}$ and $\mathcal{S}$) represent the patients and services, respectively. A multigraph allows multiple edges connecting a node pair, which precisely models the scenario that a patient may have received the same service multiple times from different doctors. An edge connecting patient p_k and service s_i carries two attributes: the doctor d_j who treated p_k with s_i, and the weight $w_{p_k \to d_j \to s_i}$ denoting the count of the service. So far, there are no known methods that explicitly address the node embedding of attributed multigraph.

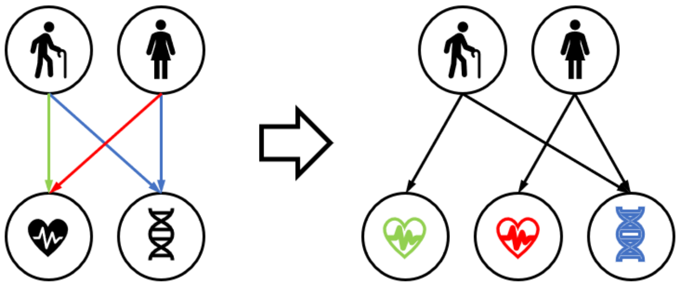


Figure 4.3: A toy example showcasing the *duplication & annotation* in patient embedding. After *duplication*, one new ECG service node is generated; after *annotation*, all service nodes are annotated with their edge attributes, and edges have no doctor attributes but weights.

Algorithmic details The challenge of embedding patient vertices in a bipartite multigraph comes mainly from the attributed edges. In an attributed network, the node or edge attributes are often heterogeneous with respect to the network structure, thus creating difficulty in joint information extraction. A common practice of recent efforts to address this challenge is to generate heterogeneous "meta-paths" that consist of both entity nodes and their attribute nodes by random walks, followed by complex deep learning models to learn the node embeddings [271, 272]. Though being effective, it raises a concern about the efficiency of a stochastic random walker exploring the network structure, especially in a multigraph where multiple edges with different attributes connect a node pair that demands more extensive localized searches of a node's neighborhood. Furthermore, random-walk-based embeddings generalize node connections beyond existing network topologies, which would potentially result in more false alarm predictions and hence must be used with extra cautions on the patient level.

In this work, we propose a simple and scalable node embedding algorithm tailored for attributed multigraph. Our algorithm is an extension of the network embedding algorithm `LINE` [149]. First, we develop a simple approach called *duplication & annotation* to convert $\mathcal{G}_{pat}$ into a simple graph with no attributes:

- *Duplication:* We duplicate each of the service nodes by the number of unique attributes of edges linked to the node. A service node will not be duplicated if all its edges are of the same attribute. For example, as shown in Figure 4.3, the electrocardiography (ECG) service has two edges with two different doctor attributes, thus was duplicated into two new ECG nodes, whereas the gene service has two edges with the same doctor attribute, and was not duplicated. After *duplication*, a service node must connect to either multiple edges with the same attribute or a single attributed edge.

- *Annotation:* We annotate each service node with the doctor attribute of its edges into a "hybrid node", and remove the doctor attribute from its edges, thereby converting a multigraph into a simple graph with no attributed edges.

Annotation can be implemented as a linear transformation of the concatenation of the doctor and service embedding vectors, as at this point we have already obtained the doctor and service embedding vectors:

$$\mathbf{h}_{s_i,d_j} = \mathbf{W}_a[\mathbf{s}_i||\mathbf{d}_j] + \mathbf{b}_a, \tag{4.6}$$

where $\mathbf{W}_a \in \mathbb{R}^{p'' \times (p+p')}$, $\mathbf{b}_a \in \mathbb{R}^{p''}$, and $\mathbf{h}_{s_i,d_j} \in \mathbb{R}^{p''}$ is the embedding of the hybrid node created from s_i and d_j.

In `LINE`, node embeddings are optimized by preserving nodes' first-order and second-order proximities defined in the network structure. As in patient embedding, we are dealing with a bipartite graph, and that the embedding vectors of the hybrid nodes are already known (except for the transformation parameters), we can skip the first-order part and focus on optimizing the second-order proximities of patient nodes only. For a patient p_k, its second-order proximity relative to other patients is defined over the "context" probability of seeing a hybrid node h_{s_i,d_j}:

$$p_2(h_{s_i,d_j}|p_k) = \frac{\exp(\mathbf{h}_{s_i,d_j} \cdot \mathbf{p}_k)}{\sum_{l \in \{h\}} \exp(\mathbf{h}_l \cdot \mathbf{p}_k)}, \tag{4.7}$$

where $\mathbf{p}_k \in \mathbb{R}^{p''}$ and $\{h\}$ is the collection of all hybrid nodes. Meanwhile, each context probability p_2 corresponds to an empirical distribution defined by the edge weights:

$$\hat{p}_2(h_{s_i,d_j}|p_k) = \frac{w_{p_k \to h_{s_i,d_j}}}{\sum_{l \in \mathcal{N}_{p_k}} w_{p_k \to h_l}}, \tag{4.8}$$

where $\mathcal{N}_{p_k}$ represents the collection of all hybrid node neighbors of patient p_k.

Then we can optimize $\{\mathbf{p}_k\}_{k=1}^P$, $\mathbf{W}_a$, and $\mathbf{b}_a$ by minimizing the following loss function

$$\mathcal{L}_{pat} = \sum_{k=1}^{P} d\left(\hat{p}_2(\cdot|p_k), p_2(\cdot|p_k)\right), \tag{4.9}$$

where d is the *Kullback–Leibler* (*KL*) distance. Plugging Eq. (4.8) into (4.9) and expanding the *KL* distance, we have

$$\mathcal{L}_{pat} = -\sum_{(i,j,k) \in \mathcal{E}_{pat}} \frac{w_{p_k \to h_{s_i,d_j}}}{\sum_{l \in \mathcal{N}_{p_k}} w_{p_k \to h_l}} \log(p_2(h_{s_i,d_j}|p_k)), \tag{4.10}$$

Algorithm 6 Patient embedding

```
Require: Patient journeys $\{\mathcal{J}^{(i)}\}_{i=1}^{P}$, service embedding $\mathbf{S}$, doctor embedding $\mathbf{D}$, learning
    rate $\eta$
Ensure: Patient embedding $\mathbf{P} \in \mathbb{R}^{P \times p''}$, linear transformation parameters $\{\mathbf{W}_a, \mathbf{b}_a\}$
 1: $\mathcal{G}_{pat} \leftarrow \{\mathcal{J}^{(i)}\}_{i=1}^{P}$
 2: $\mathcal{G}_{pat} \leftarrow$ Duplication_Annotation($\mathcal{G}_{pat}, \mathbf{S}, \mathbf{D}$)
 3: for all node $p \in \mathcal{P}$ do
 4:     $p_2(\cdot|p) \leftarrow$ Eq. (4.7)
 5:     $\hat{p}_2(\cdot|p) \leftarrow$ Eq. (4.8)
 6: while KLDist($\hat{p}_2, p_2$) is large do
 7:     (Algorithmic steps in rows 3–5)
 8:     $\{\mathbf{P}, \mathbf{W}_a, \mathbf{b}_a\} \leftarrow \{\mathbf{P}, \mathbf{W}_a, \mathbf{b}_a\} - \eta\nabla(\text{KLDist}(\hat{p}_2, p_2))$
 9: return $\mathbf{P}, \{\mathbf{W}_a, \mathbf{b}_a\}$
```

Table 4.2: Statistics of dataset.

Number of total patients	8,942
Number of CLL patients	1,241
Number of non-CLL patients	7,701
Number of doctors	8,170
Number of unique doctor primary specialty	114
Number of unique medical services	394
Average number of services per patient	111
Maximum number of different doctors for a patient	15
Average number of different doctors per patient	1.54

where $\mathcal{E}_{pat}$ is the set of all edges of the patient-service bipartite graph after *duplication & annotation*. We summarize the steps for patient embedding in Algorithm 6.

4.3 Experiments

4.3.1 Setup

Data preparation We test the proposed method on a proprietary clinical dataset from IQVIA Inc. that consists of medical records for patients who are either diagnosed as chronic lymphocytic leukemia (CLL) or undiagnosed as CLL but with related risk factors and/or symptoms. The CLL-related risk factors and symptoms are pre-specified by a medical expert. For CLL patients, we pulled their one-year medical records backward from six months before the date of diagnosis. Some vital statistics of the dataset are listed in Table 4.2. It should be noted that there is no restriction on the diseases for which `ME2Vec` is

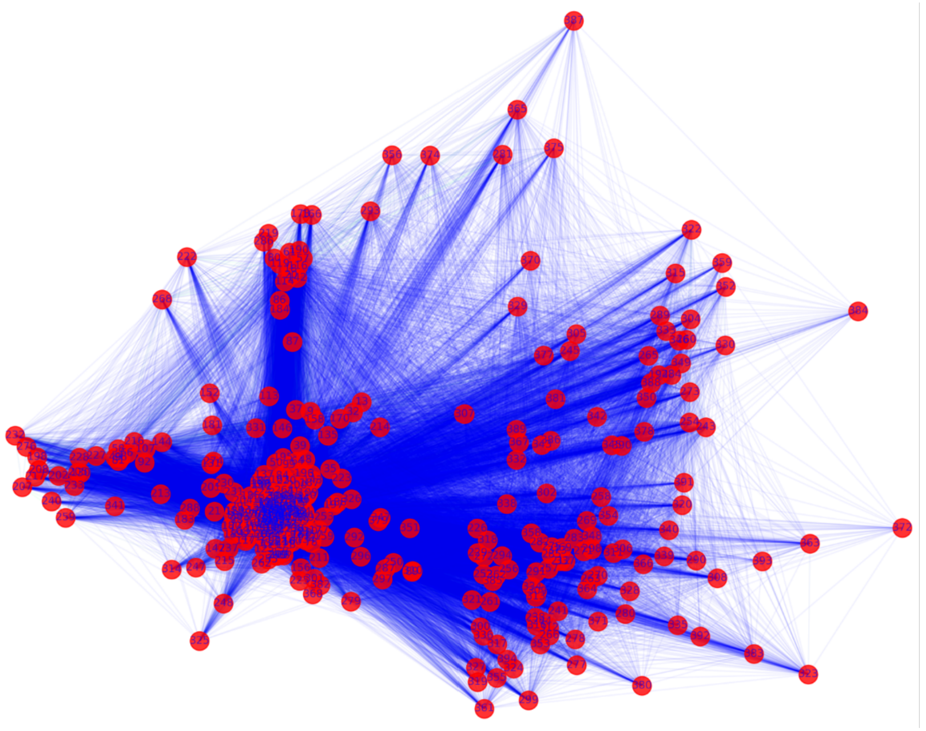


Figure 4.4: 2-dimensional visualization of service embeddings from `ME2Vec` after PCA. Each red dot represents a medical service with its ID labeled. Each blue line connecting two dots indicates that the two services co-occur as least once.

applicable. We choose CLL-related patient journeys simply for demonstration purposes.

Baselines We compare `ME2Vec` with the following baselines for medical entity embedding: `node2vec` [147], `LINE` [149], spectral clustering (SC) [273], and non-negative matrix factorization (NMF) [274].

In all experiments, we use the Adam optimizer [232] to update the parameters of `ME2Vec` and `LINE` for 150 epochs with a batch-size of 512 and learning rate of 2.5e–3. Also, the amount of negative samples in training the graph embedding based methods is set as 10. For `ME2Vec`, the context window length T is set as 8 days, and the number of attention heads K is 4. The dimensions of embeddings for all entities are set as 128. The remaining parameter settings for all baselines are as default.

4.3.2 Embedding Visualization

We examine some intermediate steps of `ME2Vec` to showcase the learning process happened along the hierarchy.

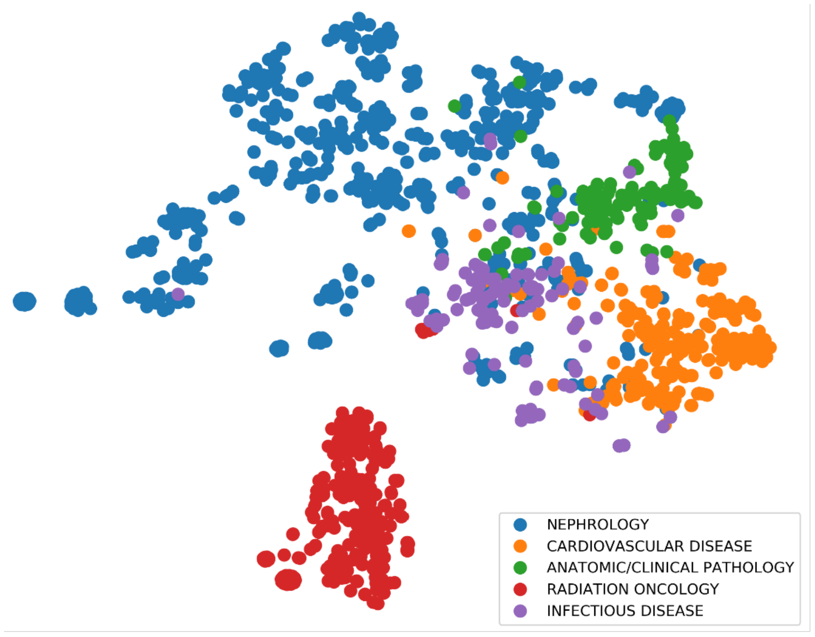


Figure 4.5: 2-dimensional visualization of a portion of doctor embeddings from `ME2Vec` after t-SNE. Each dot represents a doctor, with its color indicating the doctor's primary specialty. Doctors with five different primary specialties are displayed for illustration.

Service embedding We visualize the trained embedding vectors of 394 medical services in Figure 4.4. The 128-dimensional vectors are projected to a 2-dimensional space via principal component analysis (PCA). Figure 4.4 shows clearly that infrequent services (with larger IDs) spread out in the embedding space, whereas routine services (with smaller IDs) aggregate themselves closely in the centering area, which is consistent with our posit.

Doctor embedding We also visualize the trained embedding vectors of some of the doctors in Figure 4.5, where we can see a clear separation of doctors with different primary specialties. For example, *nephrology* doctors are far away from *cardiovascular disease* doctors, while *radiation oncology* doctors are even further away from the rest.

Together with the service embedding, these results from `ME2Vec` offer great potentials for enhancing the interpretability of models that leverage `ME2Vec` as the first step of vectorizing medical entities for further analysis.

4.3.3 Node Classification

In node classification, we first train `ME2Vec` and the baselines on the entire dataset to obtain patient embeddings for each of the methods. Unlike `ME2Vec`, the baseline methods

Table 4.3: Performance of node classification in micro-F1 and macro-F1.

Algorithms	Micro-F1				Macro-F1			
	20%	40%	60%	80%	20%	40%	60%	80%
ME2Vec	**0.869**	**0.877**	**0.878**	**0.879**	**0.664**	**0.679**	**0.682**	**0.676**
node2vec (service)	0.865	0.875	0.876	0.878	0.613	0.630	0.632	0.640
node2vec (doctor)	0.850	0.862	0.860	0.861	0.474	0.466	0.462	0.463
LINE (service)	0.855	0.864	0.866	0.866	0.587	0.592	0.592	0.586
LINE (doctor)	0.854	0.863	0.860	0.861	0.470	0.465	0.462	0.463
SC (service)	0.862	0.861	0.861	0.868	0.463	0.463	0.463	0.465
SC (doctor)	0.862	0.861	0.861	0.868	0.463	0.463	0.463	0.465
NMF (service)	0.868	0.870	0.869	**0.879**	0.584	0.586	0.589	0.600
NMF (doctor)	0.861	0.860	0.860	0.867	0.469	0.472	0.470	0.469

cannot integrate information from both doctors and services at the same time. To address this, we create two bipartite graphs from the dataset that model the patient-doctor and patient-service relations, respectively. Therefore each baseline has two versions of patient embeddings, with one derived from the patient-service graph, and the other derived from the patient-doctor graph.

Next, we use the patient embeddings in the training set as well as their diagnostic labels to train a logistic regression (LR) classifier with L2 regularization. After that, we predict the diagnostic labels of patients in the testing set from their embeddings using the trained LR classifier. We vary the training ratio from 20% to 80%, and under each training ratio we repeat the experiment for 10 times with randomized train/test split and report the average micro-F1 and macro-F1 in Table 4.3. The results show that `ME2Vec` outperforms all baselines. It is worth noting that all baselines achieve consistently poorer performance from the patient-doctor graph than from the patient-service graph, suggesting their common weakness of extracting useful information from the patient-doctor relation. Additionally, for each baseline, we tried a simple integration by concatenating the two versions of patient embeddings, which, however, did not lead to consistent performance improvement over both of the two versions, and thus was not reported.

4.3.4 Link Prediction

In link prediction, we predict if a patient should visit a doctor or receive a medical service. This task has a direct real-world significance that we can leverage the trained medical entity embeddings for personalized medical service recommendation or physician targeting.

Table 4.4: Performance of link prediction in AUC.

	Patient-Service	Patient-Doctor
ME2Vec	0.894	**0.736**
node2vec	0.918	0.608
LINE	**0.919**	0.552
SC	0.914	0.508
NMF	0.913	0.521

We first randomly remove 10% of the edges from both the patient-doctor graph and the patient-service graph as the positive edges, while ensuring that the residual graphs are connected. For the negative edges, we randomly sample an equal amount of node pairs from the original patient-service and patient-doctor graphs, respectively, which have no edges connecting them. We then obtain the embeddings of patients, doctors, and services using `ME2Vec` and the baselines from the residual graphs, and train an LR classifier for each method to predict edge existence between a patient-service or a patient-doctor node pair. The input to the LR is simply the concatenation of two embeddings. We report the performance in area-under-the-curve (AUC) as shown in Table 4.4.

4.3.5 Using `ME2Vec` as Pretrained Embeddings for Recurrent Models

In this experiment, we evaluate the effectiveness of service embedding from `ME2Vec` in a sequential learning task that predicts the probabilities of patients diagnosed as CLL from their longitudinal EHR records.

Recurrent neural networks (e.g., GRU and LSTM) have been widely adopted to model the long-range dependencies and nonlinear dynamics of sequential data. It has been the *de facto* approach to embed the individual tokens in a sequence into low-dimensional dense vectors before feeding them into recurrent models for enhanced performance, as embedding can better capture the relationship between input tokens than one-hot or multi-hot encoding. The weights of the input embedding layer can be randomly initialized and optimized together with the recurrent model in an end-to-end training, or initialized using pretrained embedding vectors and fine-tuned along with the recurrent model.

Data preparation and model training. For each patient, we prepared a sequence that tracks the patient's received medical services in the temporal order to predict CLL

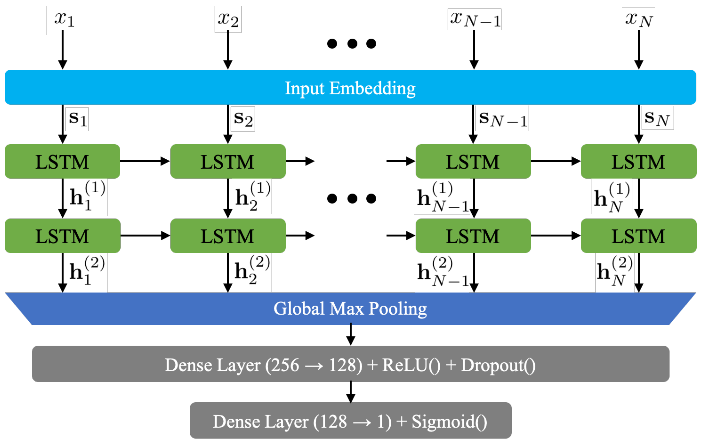


Figure 4.6: Structure of the recurrent model for predicting diagnosis results from patients' service sequences.

diagnosis. Each sequence is either truncated or padded to be 400 in length. Firstly, we randomly divided all the patients into three groups for training (80%), validating (10%), and testing (10%), respectively. The ratio of the positive (w/ CLL) versus negative (w/o CLL) is kept the same across the three groups through stratified split. We also ensured that the training dataset contains all the unique medical services in the validating and testing datasets. Secondly, we run `ME2Vec` and `Word2Vec` on the training dataset and obtained the service embeddings. Thirdly, for each patient, we trained three sequential models, with one whose input embedding layer was randomly initialized and the others initialized using the pretrained `ME2Vec` and `Word2Vec` embeddings, respectively. The models were tuned on the validating datasets. Finally, we evaluated the models with the best validating performance on the testing datasets. We repeated the above procedures for 10 times and reported the average prediction accuracy in precision-recall AUC (PR-AUC) since the cohort is highly imbalanced.

Recurrent model. As shown in Figure 4.6, the recurrent model we used for this experiment is a two-layer LSTM with 256-dimensional hidden units and a 128-dimensional input embedding layer. The hidden outputs of the LSTM enter a global max-pooling layer that for each of the 256 dimensions, keeps the maximum value from all the time steps. The outputs of the global max-pooling layer are further processed by an multilayer perceptron (MLP) ended with a sigmoid activation function to make the final prediction. In this experiment, the recurrent model is trained for 30 epochs using an Adam optimizer with a batch size of 64 and learning rate of 1e-4.

Table 4.5: Averaged performance of the recurrent model predicting CLL diagnoses from patients' service sequences.

	PR-AUC	Improvement
Input embedding initialized with pretrained `ME2Vec`	**0.823**	0%
Input embedding initialized with pretrained `Word2Vec`	0.759	+8.4%
Input embedding randomly initialized	0.753	+9.3%

Results. As shown in Table 4.5, pretrained service embeddings using `ME2Vec` can substantially improve the prediction accuracy than random initialization (9.3%) and `Word2Vec` (8.4%).

It is worth noting that in many NLP tasks, the performance improvement brought by using pretrained embeddings (or pretrained language models) is conditioned upon the access to large-scale, cheap, and unlabeled text corpora (e.g., Wikipedia or millions of web pages). However, such abundant data sources are usually not available in medical data analysis due to the legal and regulatory barriers to sharing patient-level data across different institutions. In this experiment, we show that the service embeddings given by `ME2Vec` can improve the performance of downstream tasks without requiring extra patient-level data. This advantage is primarily ascribed to that in the service embedding of `ME2Vec`, the original patient journeys are only used to construct the service graph and generate pseudo journeys via biased random walk instead of for the actual contextualized embedding learning process like in `Word2Vec`. Therefore, service embeddings from `ME2Vec` can generalize robustly to unseen patient journeys as long as the new patient journeys follow similar transition probabilities of medical services, which is a fundamental presumption in contextualized embedding and works well in practice.

4.4 Summary

In this chapter, we propose `ME2Vec`, a graph-based, hierarchical medical entity embedding framework. `ME2Vec` offers a comprehensive set of functionalities for embedding medical services, doctors, and patients. We design a time-aware service embedding that can leverage the temporal profiles of medical services to characterize their importance through random-walk based node embedding. We also adapt a recent state-of-the-art graph embedding algorithm, Graph Attention Network, to learning doctor embeddings in an auxiliary task that can reflect their administered services and primary specialties. Moreover, we develop an effective and scalable approach of node embedding for attributed multigraph that uniquely addressed the difficulty of patient embedding learning from both doctors and services.

Although overall medical entities in EHR are heterogeneous, we make the embedding learning process homogeneous in each hierarchy by carefully designing entity-specific training paradigms tailored to the structural properties and statistical characteristics of entities. An alternative solution is to learn their embeddings altogether in one graph. One advantage of this approach is that the learned embeddings of heterogeneous nodes are in the same space, therefore their distances or similarities can be more easily evaluated. However, this is at the cost of restricted flexibility of designing entity-specific training paradigms where `ME2Vec` prevails. For example, for medical services we employ random walk based contextualized embedding to characterize their temporal profiles, whereas for doctor embeddings, we are not interested in the temporal information of administered services but their relations to primary specialties. These two types of learning are distinct in nature (unsupervised versus supervised) and data structures (one-dimensional context window versus non-Euclidean neighborhood over graphs), and thus difficult to be replaced by one unified paradigm without performance degradation.

We conduct three experiments on a real-world clinical dataset, including node classification, link prediction, and pretraining input embeddings for sequential learning. The results show consistent performance improvements of `ME2Vec` compared with strong baselines on different tasks, suggesting the potentials of `ME2Vec` as a comprehensive and general-purpose solution for representation learning of EHR data.

Chapter 5

Conclusion and Future Work

In this dissertation, we proposed to develop deep learning based representation learning methods that can learn automatically effective features from input data to facilitate various downstream tasks. We motivated our works by describing the implication and challenge of handling big data in neuroscience and healthcare, and the necessity and benefits of employing advanced representation learning methods to properly tackle with the ever increasing data complexity by "understanding" the data.

Using neural signal processing as one of the main applications, we discussed two major learning paradigms (unsupervised and semi-supervised) with three novel algorithms (streaming PCA, CAE, and `FSSS`) based on two classic ML models (PCA and autoencoder). From a practitioner's perspective, each of the proposed algorithms specifically addressed one or more "pain points" encountered in reducing the required bandwidth for transmitting recoded neural data, or obtaining single-unit activities of neurons. In addition, we discussed the hardware implementation of the proposed algorithms into VLSI chips for the purpose of integrating these algorithms into neural recording and signal processing systems. For another important application, predictive healthcare analytics, we proposed a hierarchical medical entity embedding algorithm (`ME2Vec`) that offers a comprehensive solution to embed major components appeared in EHR data, including patients, doctors, and medical services. `ME2Vec` is intuitive and straightforward in terms of capturing the interactions between entities of different types, and, in the meantime, can lead to competitive performance when serving as a data pre-processing step for many predictive healthcare tasks such as diagnoses prediction or physician targeting. Essentially, it turns table-structured texts into low-dimensional dense vectors that preserve their original semantic relations and can serve as a meaningful basis for more complex predictive analysis. For future works, we propose the following directions that are important and not satisfactorily resolved.

Fully adaptive and autonomous neural data compression Neural data often exhibit non-stationarity, i.e., change of properties over time, caused by a number of reasons such as waveform variation and electrode drift. Although a learning based data compression algorithm (e.g., CAE) can extract versatile features from spikes that can generalize well to unseen spikes, the concern remains that if the trained features can stay robust in chronic recordings. Therefore it is desired to include additional mechanism into the existing model that explicitly addresses the non-stationarity of neural activities such that the model can adapt to changing properties of neural signals fully autonomously. One major difficulty is that, if we retrain the model periodically using most up-to-date spikes that are morphologically different from those used for initial training, the model may completely adapt to the new spikes and lose the capability of compressing spikes appeared in the earlier time, a phenomenon called "catastrophic forgetting" [275]. To address this issue, Bayesian neural network (BNN) can be used that allows proper treatment of the uncertainty of model prediction by assigning probabilistic distributions to the model parameters. By treating initially trained model as priors and newly arrived spikes as likelihood, BNN can learn model posteriors through Bayes' theorem, thereby accommodating new data distributions while preserving the capability to compress previously seen data.

Fully unsupervised spike sorting algorithms In `FSSS`, we still need `DidacticSort` to provide labels of a small number of randomly chosen spikes, which is unavoidably susceptible to human bias, and also creates troubles for chronic neural recording and decoding experiments that require online spike sorting in real-time. The key challenges include (i) determining the number of neurons, and (ii) assigning spikes to their originating neurons, both in a fully unsupervised fashion. As a subset of unsupervised learning, *self-supervised learning* has gained increasing attentions recently and shown promising results. In principle, self-supervised learning allows generating output labels intrinsically from data by exposing a relation between parts of the data object, or different views of the object. Self-supervised learning has been actively explored in processing natural language, images, and audio signals [276, 277, 278, 279]. Borrowing techniques from self-supervised learning, we can study the consistency on morphological properties of spikes from the same neurons, thereby providing a basis to estimate the number of neurons and the decision boundaries for clustering.

Interpretable medical embedding and clinical outcome prediction Interpretability of predictive models in healthcare related applications is critical in the sense that it determines whether or not a model can be deployed into clinical practice. For example, when deploying a model to assist with diagnoses, it is vital for doctors to understand why the model makes a prediction to build trust, instead of asking doctors to blindly trust the model simply because it achieved high accuracies on a test dataset. We could contribute to this goal from two aspects. Firstly, the embedding of input data (e.g., medical codes) must be interpretable. This requires that the embedding of a medical concept should reflect its relations with other medically relevant concepts defined by ontology that can be well understood by medical experts. Our work `ME2Vec` is such a work that well preserves the sequential relations of medical entities with respect to others extracted from large amounts of patient longitudinal EHRs. Secondly, models must reveal human-understandable explanations in support of their decisions. As DL models are notoriously famous for their black-box properties, interpreting predictions made by DL models in healthcare applications is usually done by approximating the predictions using much simpler models whose behaviors are much more explainable, such as linear regression or decision tree. The approximation is constrained to local instances (instead of globally) to ensure that simple models can mimic the predictions of DL models [280, 281]. Though sounds promising, what these approaches actually explain is the behavior of simple models, not the actual (also much more complex) model. This raises a serious concern about what we are really explaining: a high-performance yet complex model, or an ensemble of simple models at each instance. We aim to develop predictive models that are self-explanatory, without using other simple models for approximation.

References

[1] Mark Ison and Panagiotis Artemiadis. The role of muscle synergies in myoelectric control: trends and challenges for simultaneous multifunction control. *Journal of neural engineering*, 11(5):051001, 2014.

[2] Tong Wu, Wenfeng Zhao, Hongsun Guo, Hubert H Lim, and Zhi Yang. A streaming pca vlsi chip for neural data compression. *IEEE transactions on biomedical circuits and systems*, 11(6):1290–1302, 2017.

[3] Tong Wu, Wenfeng Zhao, Edward Keefer, and Zhi Yang. Deep compressive autoencoder for action potential compression in large-scale neural recording. *Journal of neural engineering*, 15(6):066019, 2018.

[4] R. Q. Quiroga, Z. Nadasdy, and Y. Ben-Shaul. Unsupervised spike detection and sorting with wavelets and superparamagnetic clustering. *Neural Comput.*, 16:1661–1687, 2004.

[5] Missinglink. Schematic diagram of an multilayer perceptron. Available at `https://missinglink.ai/guides/neural-network-concepts/perceptrons-and-multi-layer-perceptrons-the-artificial-neuron-at-the-core-of-deep-learning/`.

[6] Artem Oppermann. Artificial intelligence vs. machine learning vs. deep learning. Available at `https://www.deeplearning-academy.com/p/ai-wiki-machine-learning-vs-deep-learning`.

[7] SmartCat. Word2vec - the world of word vectors. Available at `https://www.smartcat.io/blog/2017/word2vec-the-world-of-word-vectors/`.

[8] Esther Landhuis. Neuroscience: Big brain, big data, 2017.

[9] Smadar Shilo, Hagai Rossman, and Eran Segal. Axes of a revolution: challenges and promises of big data in healthcare. *Nature Medicine*, 26(1):29–38, 2020.

[10] Ian H Stevenson and Konrad P Kording. How advances in neural recording affect data analysis. *Nature neuroscience*, 14(2):139, 2011.

[11] James J Jun, Nicholas A Steinmetz, Joshua H Siegle, Daniel J Denman, Marius Bauza, Brian Barbarits, Albert K Lee, Costas A Anastassiou, Alexandru Andrei, Çağatay Aydın, et al. Fully integrated silicon probes for high-density recording of neural activity. *Nature*, 551(7679):232, 2017.

[12] Jelena Dragas, Vijay Viswam, Amir Shadmani, Yihui Chen, Raziyeh Bounik, Alexander Stettler, Milos Radivojevic, Sydney Geissler, Marie Engelene J Obien, Jan Müller, et al. In vitromulti-functional microelectrode array featuring 59 760 electrodes, 2048 electrophysiology channels, stimulation, impedance measurement, and neurotransmitter detection channels. *IEEE journal of solid-state circuits*, 52(6):1576–1590, 2017.

[13] Carolina Mora Lopez, Ho Sung Chun, Laurent Berti, Shiwei Wang, Jan Putzeys, Carl Van Den Bulcke, Jan-Willem Weijers, Andrea Firrincieli, Veerle Reumers, Dries Braeken, et al. A 16384-electrode 1024-channel multimodal cmos mea for high-throughput intracellular action potential measurements and impedance spectroscopy in drug-screening applications. In *2018 IEEE International Solid-State Circuits Conference-(ISSCC)*, pages 464–466. IEEE, 2018.

[14] Bogdan C Raducanu, Refet F Yazicioglu, Carolina M Lopez, Marco Ballini, Jan Putzeys, Shiwei Wang, Alexandru Andrei, Veronique Rochus, Marleen Welkenhuysen, Nick van Helleputte, et al. Time multiplexed active neural probe with 1356 parallel recording sites. *Sensors*, 17(10):2388, 2017.

[15] David A Schwarz, Mikhail A Lebedev, Timothy L Hanson, Dragan F Dimitrov, Gary Lehew, Jim Meloy, Sankaranarayani Rajangam, Vivek Subramanian, Peter J Ifft, Zheng Li, et al. Chronic, wireless recordings of large-scale brain activity in freely moving rhesus monkeys. *Nature methods*, 11(6):670, 2014.

[16] Statistical Neuroscience Lab. Tracking advances in neural recording. Available at `https://stevenson.lab.uconn.edu/scaling/#`.

[17] Abdulmalik M Obaid, Mina-Elraheb S Hanna, Yu-Wei Wu, Mihaly Kollo, Romeo R Racz, Matthew R Angle, Jan Muller, Nora Brackbill, William Wray, Felix Franke, et al. Massively parallel microwire arrays integrated with cmos chips for neural recording. *bioRxiv*, page 573295, 2019.

[18] Gian Nicola Angotzi, Fabio Boi, Aziliz Lecomte, Ermanno Miele, Mario Malerba, Stefano Zucca, Antonino Casile, and Luca Berdondini. Sinaps: An implantable active pixel sensor cmos-probe for simultaneous large-scale neural recordings. *Biosensors and Bioelectronics*, 126:355–364, 2019.

[19] Christopher M Lewis, Conrado A Bosman, and Pascal Fries. Recording of brain activity across spatial scales. *Current opinion in neurobiology*, 32:68–77, 2015.

[20] Mikhail A Lebedev and Miguel AL Nicolelis. Toward a whole-body neuroprosthetic. In *Progress in brain research*, volume 194, pages 47–60. Elsevier, 2011.

[21] Wullianallur Raghupathi and Viju Raghupathi. Big data analytics in healthcare: promise and potential. *Health information science and systems*, 2(1):3, 2014.

[22] Tracy D Gunter and Nicolas P Terry. The emergence of national electronic health record architectures in the united states and australia: models, costs, and questions. *Journal of medical Internet research*, 7(1):e3, 2005.

[23] MPP David Blumenthal MD. Launching hitech. *The New England journal of medicine*, 362(5):382, 2010.

[24] A Jackie Hunter. The innovative medicines initiative: a pre-competitive initiative to enhance the biomedical science base of europe to expedite the development of new medicines for patients. *Drug discovery today*, 13(9-10):371, 2008.

[25] J Henry, Yuriy Pylypchuk, Talisha Searcy, and Vaishali Patel. Adoption of electronic health record systems among us non-federal acute care hospitals: 2008–2015. *ONC data brief*, 35:1–9, 2016.

[26] Benjamin Shickel, Patrick James Tighe, Azra Bihorac, and Parisa Rashidi. Deep ehr: a survey of recent advances in deep learning techniques for electronic health record (ehr) analysis. *IEEE journal of biomedical and health informatics*, 22(5):1589–1604, 2017.

[27] Yoshua Bengio, Aaron Courville, and Pascal Vincent. Representation learning: A review and new perspectives. *IEEE transactions on pattern analysis and machine intelligence*, 35(8):1798–1828, 2013.

[28] David E Rumelhart, Geoffrey E Hinton, and Ronald J Williams. Learning representations by back-propagating errors. *nature*, 323(6088):533–536, 1986.

[29] Jack Kiefer, Jacob Wolfowitz, et al. Stochastic estimation of the maximum of a regression function. *The Annals of Mathematical Statistics*, 23(3):462–466, 1952.

[30] Andrej Karpathy. Software 2.0. Available at `https://medium.com/@karpathy/software-2-0-a64152b37c35`.

[31] Zachary C Lipton. The mythos of model interpretability. *Queue*, 16(3):31–57, 2018.

[32] Kun-Hsing Yu, Andrew L Beam, and Isaac S Kohane. Artificial intelligence in healthcare. *Nature biomedical engineering*, 2(10):719–731, 2018.

[33] Scott M Lundberg, Bala Nair, Monica S Vavilala, Mayumi Horibe, Michael J Eisses, Trevor Adams, David E Liston, Daniel King-Wai Low, Shu-Fang Newman, Jerry Kim, et al. Explainable machine-learning predictions for the prevention of hypoxaemia during surgery. *Nature biomedical engineering*, 2(10):749–760, 2018.

[34] Balázs Csanád Csáji et al. Approximation with artificial neural networks. *Faculty of Sciences, Etvs Lornd University, Hungary*, 24(48):7, 2001.

[35] Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. Imagenet: A large-scale hierarchical image database. In *2009 IEEE conference on computer vision and pattern recognition*, pages 248–255. Ieee, 2009.

[36] Alex Krizhevsky, Geoffrey Hinton, et al. Learning multiple layers of features from tiny images. 2009.

[37] Daniel LK Yamins and James J DiCarlo. Using goal-driven deep learning models to understand sensory cortex. *Nature neuroscience*, 19(3):356, 2016.

[38] Daniel LK Yamins, Ha Hong, Charles F Cadieu, Ethan A Solomon, Darren Seibert, and James J DiCarlo. Performance-optimized hierarchical models predict neural responses in higher visual cortex. *Proceedings of the National Academy of Sciences*, 111(23):8619–8624, 2014.

[39] Jack Lindsey, Samuel A Ocko, Surya Ganguli, and Stephane Deny. A unified theory of early visual representations from retina to cortex through anatomically constrained deep cnns. *arXiv preprint arXiv:1901.00945*, 2019.

[40] Kunihiko Fukushima. Neocognitron: A hierarchical neural network capable of visual pattern recognition. *Neural networks*, 1(2):119–130, 1988.

[41] David H Hubel and Torsten N Wiesel. Receptive fields and functional architecture of monkey striate cortex. *The Journal of physiology*, 195(1):215–243, 1968.

[42] Masakazu Matsugu, Katsuhiko Mori, Yusuke Mitari, and Yuji Kaneda. Subject independent facial expression recognition with robust face detection using a convolutional neural network. *Neural Networks*, 16(5-6):555–559, 2003.

[43] Jacquelin Perry, Catherine Schmidt Easterday, and Daniel J Antonelli. Surface versus intramuscular electrodes for electromyography of superficial and deep muscles. *Physical therapy*, 61(1):7–15, 1981.

[44] GE Loeb and RA Peck. Cuff electrodes for chronic stimulation and recording of peripheral nerve activity. *Journal of neuroscience methods*, 64(1):95–103, 1996.

[45] Dustin J Tyler and Dominique M Durand. Functionally selective peripheral nerve stimulation with a flat interface nerve electrode. *IEEE Transactions on Neural Systems and Rehabilitation Engineering*, 10(4):294–303, 2002.

[46] Ken Yoshida and Ken Horch. Selective stimulation of peripheral nerve fibers using dual intrafascicular electrodes. *IEEE transactions on biomedical engineering*, 40(5):492–494, 1993.

[47] Stephen M Lawrence, Gurpreet S Dhillon, and Kenneth W Horch. Fabrication and characteristics of an implantable, polymer-based, intrafascicular electrode. *Journal of neuroscience methods*, 131(1-2):9–26, 2003.

[48] Tim Boretius, Jordi Badia, Aran Pascual-Font, Martin Schuettler, Xavier Navarro, Ken Yoshida, and Thomas Stieglitz. A transverse intrafascicular multichannel electrode (time) to interface with the peripheral nerve. *Biosensors and Bioelectronics*, 26(1):62–69, 2010.

[49] Aaron J Young, Lauren H Smith, Elliott J Rouse, and Levi J Hargrove. A comparison of the real-time controllability of pattern recognition to conventional myoelectric control for discrete and simultaneous movements. *Journal of neuroengineering and rehabilitation*, 11(1):5, 2014.

[50] Ning Jiang, Hubertus Rehbaum, Ivan Vujaklija, Bernhard Graimann, and Dario Farina. Intuitive, online, simultaneous, and proportional myoelectric control over two degrees-of-freedom in upper limb amputees. *IEEE transactions on neural systems and rehabilitation engineering*, 22(3):501–510, 2013.

[51] Mohammadreza Asghari Oskoei and Huosheng Hu. Myoelectric control systems—a survey. *Biomedical signal processing and control*, 2(4):275–294, 2007.

[52] Ugur Sahin and Ferat Sahin. Pattern recognition with surface emg signal based wavelet transformation. In *2012 IEEE international conference on systems, man, and cybernetics (SMC)*, pages 295–300. IEEE, 2012.

[53] Kevin Englehart, Bernard Hudgins, Philip A Parker, and Maryhelen Stevenson. Classification of the myoelectric signal using time-frequency based representations. *Medical engineering & physics*, 21(6-7):431–438, 1999.

[54] Lena H Ting and Stacie A Chvatal. Decomposing muscle activity in motor tasks. *Motor Control: Theories, Experiments, and Applications*, pages 102–138, 2010.

[55] Matthew C Tresch, Vincent CK Cheung, and Andrea d'Avella. Matrix factorization algorithms for the identification of muscle synergies: evaluation on simulated and experimental data sets. *Journal of neurophysiology*, 95(4):2199–2212, 2006.

[56] Felix E Zajac. Muscle and tendon: properties, models, scaling, and application to biomechanics and motor control. *Critical reviews in biomedical engineering*, 17(4):359–411, 1989.

[57] Edward A Clancy, Evelyn L Morin, and Roberto Merletti. Sampling, noise-reduction and amplitude estimation issues in surface electromyography. *Journal of electromyography and kinesiology*, 12(1):1–16, 2002.

[58] Bernard Hudgins, Philip Parker, and Robert N Scott. A new strategy for multifunction myoelectric control. *IEEE Transactions on Biomedical Engineering*, 40(1):82–94, 1993.

[59] Dennis Tkach, He Huang, and Todd A Kuiken. Study of stability of time-domain features for electromyographic pattern recognition. *Journal of neuroengineering and rehabilitation*, 7(1):21, 2010.

[60] RG Willison. A method of measuring motor unit activity in human muscle. In *JOURNAL OF PHYSIOLOGY-LONDON*, volume 168, page P35. CAMBRIDGE UNIV PRESS 40 WEST 20TH STREET, NEW YORK, NY 10011-4211, 1963.

[61] Necille Hogan and Robert W Mann. Myoelectric signal processing: Optimal estimation applied to electromyography-part i: Derivation of the optimal myoprocessor. *IEEE Transactions on Biomedical Engineering*, (7):382–395, 1980.

[62] Mahyar Zardoshti-Kermani, Bruce C Wheeler, Kambiz Badie, and Reza M Hashemi. Emg feature evaluation for movement control of upper extremity prostheses. *IEEE Transactions on Rehabilitation Engineering*, 3(4):324–333, 1995.

[63] Rami N Khushaba, Maen Takruri, Jaime Valls Miro, and Sarath Kodagoda. Towards limb position invariant myoelectric pattern recognition using time-dependent spectral features. *Neural Networks*, 55:42–58, 2014.

[64] Mustafa Sezer Erkilinc and Ferat Sahin. Camera control with emg signals using principal component analysis and support vector machines. In *2011 IEEE International Systems Conference*, pages 417–421. IEEE, 2011.

[65] Sijiang Du and Marko Vuskovic. Temporal vs. spectral approach to feature extraction from prehensile emg signals. In *Proceedings of the 2004 IEEE International Conference on Information Reuse and Integration, 2004. IRI 2004.*, pages 344–350. IEEE, 2004.

[66] Panagiotis K Artemiadis and Kostas J Kyriakopoulos. An emg-based robot control scheme robust to time-varying emg signal features. *IEEE Transactions on Information Technology in Biomedicine*, 14(3):582–588, 2010.

[67] Silvestro Micera, Angelo M Sabatini, Paolo Dario, and Bruno Rossi. A hybrid approach to emg pattern analysis for classification of arm movements using statistical and fuzzy techniques. *Medical engineering & physics*, 21(5):303–311, 1999.

[68] Kevin Englehart, B Hudgin, and Philip A Parker. A wavelet-based continuous classification scheme for multifunction myoelectric control. *IEEE Transactions on Biomedical Engineering*, 48(3):302–311, 2001.

[69] Xiao Hu, Qun Yu, Waixi Liu, and Jian Qin. Feature extraction of surface emg signal based on wavelet coefficient entropy. In *2008 2nd International Conference on Bioinformatics and Biomedical Engineering*, pages 1758–1760. IEEE, 2008.

[70] Lingling Chen, Peng Yang, Linan Zu, and Xiaoyun Xu. Electromyogram signal analysis and movement recognition based on wavelet packet transform. In *2009 International Conference on Information and Automation*, pages 1482–1487. IEEE, 2009.

[71] Marie-Françoise Lucas, Adrien Gaufriau, Sylvain Pascual, Christian Doncarli, and Dario Farina. Multi-channel surface emg classification using support vector machines and signal-based wavelet optimization. *Biomedical Signal Processing and Control*, 3(2):169–174, 2008.

[72] Silvia Muceli, Ning Jiang, and Dario Farina. Extracting signals robust to electrode number and shift for online simultaneous and proportional myoelectric control by factorization algorithms. *IEEE Transactions on Neural Systems and Rehabilitation Engineering*, 22(3):623–633, 2013.

[73] Abidemi Bolu Ajiboye and R Fff Weir. A heuristic fuzzy logic approach to emg pattern recognition for multifunctional prosthesis control. *IEEE Transactions on Neural Systems and Rehabilitation Engineering*, 13(3):280–291, 2005.

[74] Rami N Khushaba, Ahmed Al-Ani, and Adel Al-Jumaily. Orthogonal fuzzy neighborhood discriminant analysis for multifunction myoelectric hand control. *IEEE Transactions on Biomedical Engineering*, 57(6):1410–1419, 2010.

[75] Gan Huang, Zhiguo Zhang, Dingguo Zhang, and Xiangyang Zhu. Spatio-spectral filters for low-density surface electromyographic signal classification. *Medical & biological engineering & computing*, 51(5):547–555, 2013.

[76] Mark R Ison and Panagiotis Artemiadis. Beyond user-specificity for emg decoding using multiresolution muscle synergy analysis. In *ASME 2013 Dynamic Systems and Control Conference*. American Society of Mechanical Engineers Digital Collection, 2013.

[77] Peng Xia, Jie Hu, and Yinghong Peng. Emg-based estimation of limb movement using deep learning with recurrent convolutional neural networks. *Artificial organs*, 42(5):E67–E77, 2018.

[78] Ahmed Ben Said, Amr Mohamed, Tarek Elfouly, Khaled Harras, and Z Jane Wang. Multimodal deep learning approach for joint eeg-emg data compression and classification. In *2017 IEEE Wireless Communications and Networking Conference (WCNC)*, pages 1–6. IEEE, 2017.

[79] Muhammad Zia ur Rehman, Asim Waris, Syed Omer Gilani, Mads Jochumsen, Imran Khan Niazi, Mohsin Jamil, Dario Farina, and Ernest Nlandu Kamavuako. Multiday emg-based classification of hand motions with deep learning techniques. *Sensors*, 18(8):2497, 2018.

[80] Silvestro Micera, Jacopo Carpaneto, and Stanisa Raspopovic. Control of hand prostheses using peripheral information. *IEEE reviews in biomedical engineering*, 3:48–68, 2010.

[81] TS Davis, HAC Wark, DT Hutchinson, DJ Warren, K O'neill, T Scheinblum, GA Clark, RA Normann, and Bradley Greger. Restoring motor control and sensory feedback in people with upper extremity amputations using arrays of 96 microelectrodes implanted in the median and ulnar nerves. *Journal of neural engineering*, 13(3):036001, 2016.

[82] Silvestro Micera, Paolo M Rossini, Jacopo Rigosa, Luca Citi, Jacopo Carpaneto, Stanisa Raspopovic, Mario Tombini, Christian Cipriani, Giovanni Assenza, Maria C Carrozza, et al. Decoding of grasping information from neural signals recorded using

peripheral intrafascicular interfaces. *Journal of neuroengineering and rehabilitation*, 8(1):53, 2011.

[83] Paolo M Rossini, Silvestro Micera, Antonella Benvenuto, Jacopo Carpaneto, Giuseppe Cavallo, Luca Citi, Christian Cipriani, Luca Denaro, Vincenzo Denaro, Giovanni Di Pino, et al. Double nerve intraneural interface implant on a human amputee for robotic hand control. *Clinical neurophysiology*, 121(5):777–783, 2010.

[84] Stanisa Raspopovic, Marco Capogrosso, Francesco Maria Petrini, Marco Bonizzato, Jacopo Rigosa, Giovanni Di Pino, Jacopo Carpaneto, Marco Controzzi, Tim Boretius, Eduardo Fernandez, et al. Restoring natural sensory feedback in real-time bidirectional hand prostheses. *Science translational medicine*, 6(222):222ra19–222ra19, 2014.

[85] Roy Mukamel and Itzhak Fried. Human intracranial recordings and cognitive neuroscience. *Annual review of psychology*, 63:511–537, 2012.

[86] Hernan Gonzalo Rey, Carlos Pedreira, and Rodrigo Quian Quiroga. Past, present and future of spike sorting techniques. *Brain research bulletin*, 119:106–117, 2015.

[87] Leigh R Hochberg, Mijail D Serruya, Gerhard M Friehs, Jon A Mukand, Maryam Saleh, Abraham H Caplan, Almut Branner, David Chen, Richard D Penn, and John P Donoghue. Neuronal ensemble control of prosthetic devices by a human with tetraplegia. *Nature*, 442(7099):164–171, 2006.

[88] Chad E Bouton, Ammar Shaikhouni, Nicholas V Annetta, Marcia A Bockbrader, David A Friedenberg, Dylan M Nielson, Gaurav Sharma, Per B Sederberg, Bradley C Glenn, W Jerry Mysiw, et al. Restoring cortical control of functional movement in a human with quadriplegia. *Nature*, 533(7602):247–250, 2016.

[89] Jennifer L Collinger, Brian Wodlinger, John E Downey, Wei Wang, Elizabeth C Tyler-Kabara, Douglas J Weber, Angus JC McMorland, Meel Velliste, Michael L Boninger, and Andrew B Schwartz. High-performance neuroprosthetic control by an individual with tetraplegia. *The Lancet*, 381(9866):557–564, 2013.

[90] Lotte NS Andreasen Struijk, Metin Akay, and Johannes J Struijk. The single nerve fiber action potential and the filter bank—a modeling approach. *IEEE Transactions on Biomedical Engineering*, 55(1):372–375, 2007.

[91] Shuman He, Holly FB Teagle, and Craig A Buchman. The electrically evoked compound action potential: from laboratory to clinic. *Frontiers in neuroscience*, 11:339, 2017.

[92] Uri Rokni, Andrew G Richardson, Emilio Bizzi, and H Sebastian Seung. Motor learning with unstable neural representations. *Neuron*, 54(4):653–666, 2007.

[93] Nandakumar S Narayanan, Eyal Y Kimchi, and Mark Laubach. Redundancy and synergy of neuronal ensembles in motor cortex. *Journal of Neuroscience*, 25(17):4207–4216, 2005.

[94] Ken Takiyama and Masato Okada. Maximization of learning speed in the motor cortex due to neuronal redundancy. *PLoS computational biology*, 8(1), 2012.

[95] Kyle H Srivastava, Caroline M Holmes, Michiel Vellema, Andrea R Pack, Coen PH Elemans, Ilya Nemenman, and Samuel J Sober. Motor control by precisely timed spike patterns. *Proceedings of the National Academy of Sciences*, 114(5):1171–1176, 2017.

[96] Samuel J Sober, Simon Sponberg, Ilya Nemenman, and Lena H Ting. Millisecond spike timing codes for motor control. *Trends in neurosciences*, 41(10):644–648, 2018.

[97] Peter B Jensen, Lars J Jensen, and Søren Brunak. Mining electronic health records: towards better research applications and clinical care. *Nature Reviews Genetics*, 13(6):395–405, 2012.

[98] Chayakrit Krittanawong, HongJu Zhang, Zhen Wang, Mehmet Aydar, and Takeshi Kitai. Artificial intelligence in precision cardiovascular medicine. *Journal of the American College of Cardiology*, 69(21):2657–2664, 2017.

[99] Rachel E Sherman, Steven A Anderson, Gerald J Dal Pan, Gerry W Gray, Thomas Gross, Nina L Hunter, Lisa LaVange, Danica Marinac-Dabic, Peter W Marks, Melissa A Robb, et al. Real-world evidence—what is it and what can it tell us. *N Engl J Med*, 375(23):2293–2297, 2016.

[100] Jessica M Franklin and Sebastian Schneeweiss. When and how can real world data analyses substitute for randomized controlled trials? *Clinical Pharmacology & Therapeutics*, 102(6):924–933, 2017.

[101] Martin R Cowie, Juuso I Blomster, Lesley H Curtis, Sylvie Duclaux, Ian Ford, Fleur Fritz, Samantha Goldman, Salim Janmohamed, Jörg Kreuzer, Mark Leenay, et al. Electronic health records to facilitate clinical research. *Clinical Research in Cardiology*, 106(1):1–9, 2017.

[102] Roy H Perlis, Maurizio Fava, and TH McCoy. Can electronic health records revive central nervous system clinical trials? *Mol Psychiatry*, 24(8):1096–1098, 2019.

[103] Abhyuday N Jagannatha and Hong Yu. Structured prediction models for rnn based sequence labeling in clinical text. In *Proceedings of the conference on empirical methods in natural language processing. conference on empirical methods in natural language processing*, volume 2016, page 856. NIH Public Access, 2016.

[104] Jason Alan Fries. Brundlefly at semeval-2016 task 12: Recurrent neural networks vs. joint inference for clinical temporal information extraction. *arXiv preprint arXiv:1606.01433*, 2016.

[105] Xinbo Lv, Yi Guan, Jinfeng Yang, and Jiawei Wu. Clinical relation extraction with deep learning. *International Journal of Hybrid Information Technology*, 9(7):237–248, 2016.

[106] Yue Liu, Tao Ge, Kusum Mathews, Heng Ji, and Deborah McGuinness. Exploiting task-oriented resources to learn word embeddings for clinical abbreviation expansion. In *Proceedings of BioNLP 15*, pages 92–97, 2015.

[107] Truyen Tran, Tu Dinh Nguyen, Dinh Phung, and Svetha Venkatesh. Learning vector representation of medical objects via emr-driven nonnegative restricted boltzmann machines (enrbm). *Journal of biomedical informatics*, 54:96–105, 2015.

[108] Riccardo Miotto, Li Li, Brian A Kidd, and Joel T Dudley. Deep patient: an unsupervised representation to predict the future of patients from the electronic health records. *Scientific reports*, 6(1):1–10, 2016.

[109] Edward Choi, Andy Schuetz, Walter F Stewart, and Jimeng Sun. Medical concept representation learning from electronic health records and its application on heart failure prediction. *arXiv preprint arXiv:1602.03686*, 2016.

[110] Edward Choi, Andy Schuetz, Walter F Stewart, and Jimeng Sun. Using recurrent neural network models for early detection of heart failure onset. *Journal of the American Medical Informatics Association*, 24(2):361–370, 2017.

[111] Youngduck Choi, Chill Yi-I Chiu, and David Sontag. Learning low-dimensional representations of medical concepts. *AMIA Summits on Translational Science Proceedings*, 2016:41, 2016.

[112] Saaed Mehrabi, Sunghwan Sohn, Dingheng Li, Joshua J Pankratz, Terry Therneau, Jennifer L St Sauver, Hongfang Liu, and Mathew Palakal. Temporal pattern and association discovery of diagnosis codes using deep learning. In *2015 International Conference on Healthcare Informatics*, pages 408–416. IEEE, 2015.

[113] Stefan Ravizza, Tony Huschto, Anja Adamov, Lars Böhm, Alexander Büsser, Frederik F Flöther, Rolf Hinzmann, Helena König, Scott M McAhren, Daniel H Robertson, et al. Predicting the early risk of chronic kidney disease in patients with diabetes using real-world data. *Nature medicine*, 25(1):57–59, 2019.

[114] Samuele Fiorini, Farshid Hajati, Annalisa Barla, and Federico Girosi. Predicting diabetes second-line therapy initiation in the australian population via time span-guided neural attention network. *PloS one*, 14(10), 2019.

[115] Nitzan Shalom Artzi, Smadar Shilo, Eran Hadar, Hagai Rossman, Shiri Barbash-Hazan, Avi Ben-Haroush, Ran D Balicer, Becca Feldman, Arnon Wiznitzer, and Eran Segal. Prediction of gestational diabetes based on nationwide electronic health records. *Nature Medicine*, 26(1):71–76, 2020.

[116] Cao Xiao, Tengfei Ma, Adji B Dieng, David M Blei, and Fei Wang. Readmission prediction via deep contextual embedding of clinical concepts. *PloS one*, 13(4), 2018.

[117] Haishuai Wang, Zhicheng Cui, Yixin Chen, Michael Avidan, Arbi Ben Abdallah, and Alexander Kronzer. Predicting hospital readmission via cost-sensitive deep learning. *IEEE/ACM transactions on computational biology and bioinformatics*, 15(6):1968–1978, 2018.

[118] Juan Zhao, QiPing Feng, Patrick Wu, Jeremy L Warner, Joshua C Denny, and Wei-Qi Wei. Using topic modeling via non-negative matrix factorization to identify relationships between genetic variants and disease phenotypes: A case study of lipoprotein (a)(lpa). *PloS one*, 14(2), 2019.

[119] Victor A Rodriguez and Adler Perotte. Phenotype inference with semi-supervised mixed membership models. In *Machine Learning for Healthcare Conference*, pages 304–324, 2019.

[120] Ioakeim Perros, Evangelos E Papalexakis, Richard Vuduc, Elizabeth Searles, and Jimeng Sun. Temporal phenotyping of medically complex children via parafac2 tensor factorization. *Journal of biomedical informatics*, 93:103125, 2019.

[121] Andriy Mulyar, Elliot Schumacher, Masoud Rouhizadeh, and Mark Dredze. Phenotyping of clinical notes with improved document classification models using contextualized neural language models. *arXiv preprint arXiv:1910.13664*, 2019.

[122] Inci M Baytas, Cao Xiao, Xi Zhang, Fei Wang, Anil K Jain, and Jiayu Zhou. Patient subtyping via time-aware lstm networks. In *Proceedings of the 23rd ACM SIGKDD international conference on knowledge discovery and data mining*, pages 65–74, 2017.

[123] Eric Lehman, Jay DeYoung, Regina Barzilay, and Byron C Wallace. Inferring which medical treatments work from reports of clinical trials. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*, pages 3705–3717, 2019.

[124] Byron C Wallace. What does the evidence say? models to help make sense of the biomedical literature. In *Proceedings of the 28th International Joint Conference on Artificial Intelligence*, pages 6416–6420. AAAI Press, 2019.

[125] Shivashankar Subramanian, Ioana Baldini, Sushma Ravichandran, Dmitriy A Katz-Rogozhnikov, Karthikeyan Natesan Ramamurthy, Prasanna Sattigeri, Kush R Varshney, Annmarie Wang, Pradeep Mangalath, and Laura B Kleiman. A natural language processing system for extracting evidence of drug repurposing from scientific publications.

[126] The switch from icd-9 to icd-10: When and why. Available at `https://icd.codes/articles/icd9-to-icd10-explained`.

[127] Tomas Mikolov, Ilya Sutskever, Kai Chen, Greg S Corrado, and Jeff Dean. Distributed representations of words and phrases and their compositionality. In *Advances in neural information processing systems*, pages 3111–3119, 2013.

[128] Tomas Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean. Efficient estimation of word representations in vector space. *arXiv preprint arXiv:1301.3781*, 2013.

[129] Jeffrey Pennington, Richard Socher, and Christopher D Manning. Glove: Global vectors for word representation. In *Proceedings of the 2014 conference on empirical methods in natural language processing (EMNLP)*, pages 1532–1543, 2014.

[130] Piotr Bojanowski, Edouard Grave, Armand Joulin, and Tomas Mikolov. Enriching word vectors with subword information. *Transactions of the Association for Computational Linguistics*, 5:135–146, 2017.

[131] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural computation*, 9(8):1735–1780, 1997.

[132] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. In *Advances in neural information processing systems*, pages 5998–6008, 2017.

[133] Bryan McCann, James Bradbury, Caiming Xiong, and Richard Socher. Learned in translation: Contextualized word vectors. In *Advances in Neural Information Processing Systems*, pages 6294–6305, 2017.

[134] Matthew E Peters, Mark Neumann, Mohit Iyyer, Matt Gardner, Christopher Clark, Kenton Lee, and Luke Zettlemoyer. Deep contextualized word representations. *arXiv preprint arXiv:1802.05365*, 2018.

[135] Jeremy Howard and Sebastian Ruder. Universal language model fine-tuning for text classification. *arXiv preprint arXiv:1801.06146*, 2018.

[136] Alec Radford, Karthik Narasimhan, Tim Salimans, and Ilya Sutskever. Improving language understanding by generative pre-training.

[137] Alec Radford, Jeffrey Wu, Rewon Child, David Luan, Dario Amodei, and Ilya Sutskever. Language models are unsupervised multitask learners. *OpenAI Blog*, 1(8):9, 2019.

[138] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805*, 2018.

[139] Yukun Zhu, Ryan Kiros, Rich Zemel, Ruslan Salakhutdinov, Raquel Urtasun, Antonio Torralba, and Sanja Fidler. Aligning books and movies: Towards story-like visual explanations by watching movies and reading books. In *Proceedings of the IEEE international conference on computer vision*, pages 19–27, 2015.

[140] J Lee, W Yoon, S Kim, D Kim, CH So, and J Kang. Biobert: a pre-trained biomedical language representation model for biomedical text mining. *Bioinformatics (Oxford, England)*, 2019.

[141] Emily Alsentzer, John Murphy, William Boag, Wei-Hung Weng, Di Jindi, Tristan Naumann, and Matthew McDermott. Publicly available clinical bert embeddings. In *Proceedings of the 2nd Clinical Natural Language Processing Workshop*, pages 72–78, 2019.

[142] Yuqi Si, Jingqi Wang, Hua Xu, and Kirk Roberts. Enhancing clinical concept extraction with contextual embeddings. *Journal of the American Medical Informatics Association*, 26(11):1297–1304, 2019.

[143] Andrew L Beam, Benjamin Kompa, Allen Schmaltz, Inbar Fried, Griffin Weber, Nathan Palmer, Xu Shi, Tianxi Cai, and Isaac S Kohane. Clinical concept embeddings learned from massive sources of multimodal medical data. In *Pacific Symposium on Biocomputing*, volume 25, pages 295–306, 2020.

[144] Kexin Huang, Jaan Altosaar, and Rajesh Ranganath. Clinicalbert: Modeling clinical notes and predicting hospital readmission. *arXiv preprint arXiv:1904.05342*, 2019.

[145] Keyulu Xu, Chengtao Li, Yonglong Tian, Tomohiro Sonobe, Ken-ichi Kawarabayashi, and Stefanie Jegelka. Representation learning on graphs with jumping knowledge networks. In *International Conference on Machine Learning*, pages 5453–5462, 2018.

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

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

[148] Yuxiao Dong, Nitesh V Chawla, and Ananthram Swami. metapath2vec: Scalable representation learning for heterogeneous networks. In *Proceedings of the 23rd ACM SIGKDD international conference on knowledge discovery and data mining*, pages 135–144, 2017.

[149] Jian Tang, Meng Qu, Mingzhe Wang, Ming Zhang, Jun Yan, and Qiaozhu Mei. Line: Large-scale information network embedding. In *Proceedings of the 24th international conference on world wide web*, pages 1067–1077, 2015.

[150] Jian Tang, Meng Qu, and Qiaozhu Mei. Pte: Predictive text embedding through large-scale heterogeneous text networks. In *Proceedings of the 21th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 1165–1174, 2015.

[151] Will Hamilton, Zhitao Ying, and Jure Leskovec. Inductive representation learning on large graphs. In *Advances in neural information processing systems*, pages 1024–1034, 2017.

[152] Edward Choi, Cao Xiao, Walter Stewart, and Jimeng Sun. Mime: Multilevel medical embedding of electronic health records for predictive healthcare. In *Advances in neural information processing systems*, pages 4547–4557, 2018.

[153] Edward Choi, Zhen Xu, Yujia Li, Michael W Dusenberry, Gerardo Flores, Yuan Xue, and Andrew M Dai. Graph convolutional transformer: Learning the graphical structure of electronic health records. *arXiv preprint arXiv:1906.04716*, 2019.

[154] Edward Choi, Mohammad Taha Bahadori, Elizabeth Searles, Catherine Coffey, Michael Thompson, James Bost, Javier Tejedor-Sojo, and Jimeng Sun. Multi-layer representation learning for medical concepts. In *Proceedings of the 22nd ACM*

SIGKDD International Conference on Knowledge Discovery and Data Mining, pages 1495–1504, 2016.

[155] Edward Choi, Mohammad Taha Bahadori, Andy Schuetz, Walter F Stewart, and Jimeng Sun. Doctor ai: Predicting clinical events via recurrent neural networks. In *Machine Learning for Healthcare Conference*, pages 301–318, 2016.

[156] Jelena Stojanovic, Djordje Gligorijevic, Vladan Radosavljevic, Nemanja Djuric, Mihajlo Grbovic, and Zoran Obradovic. Modeling healthcare quality via compact representations of electronic health records. *IEEE/ACM transactions on computational biology and bioinformatics*, 14(3):545–554, 2016.

[157] Zihao Zhu, Changchang Yin, Buyue Qian, Yu Cheng, Jishang Wei, and Fei Wang. Measuring patient similarities via a deep architecture with medical concept embedding. In *2016 IEEE 16th International Conference on Data Mining (ICDM)*, pages 749–758. IEEE, 2016.

[158] Sebastien Dubois, Nathanael Romano, David C Kale, Nigam Shah, and Kenneth Jung. Learning effective representations from clinical notes. *stat*, 1050:15, 2017.

[159] Siddharth Biswal, Cao Xiao, Lucas M Glass, Elizabeth Milkovits, and Jimeng Sun. Doctor2vec: Dynamic doctor representation learning for clinical trial recruitment. *arXiv preprint arXiv:1911.10395*, 2019.

[160] Thomas R Insel, Story C Landis, and Francis S Collins. The nih brain initiative. *Science*, 340(6133):687–688, 2013.

[161] Antal Berényi, Zoltán Somogyvári, Anett J Nagy, Lisa Roux, John D Long, Shigeyoshi Fujisawa, Eran Stark, Anthony Leonardo, Timothy D Harris, and György Buzsáki. Large-scale, high-density (up to 512 channels) recording of local circuits in behaving animals. *Journal of neurophysiology*, 111(5):1132–1149, 2014.

[162] Zhi Yang, Jian Xu, Anh Tuan Nguyen, Tong Wu, Wenfeng Zhao, and Teris Tam. Neuronix enables continuous, simultaneous neural recording and electrical microstimulation. In *Engineering in Medicine and Biology Society (EMBC), 2016 38th Annual International Conference of the IEEE*, pages 1–4. IEEE, 2016.

[163] David Tsai, Esha John, Tarun Chari, Rafael Yuste, and Kenneth Shepard. High-channel-count, high-density microelectrode array for closed-loop investigation of neuronal networks. In *Engineering in Medicine and Biology Society (EMBC), 2015 37th Annual International Conference of the IEEE*, pages 7510–7513. IEEE, 2015.

[164] Sung-Phil Kim, John D Simeral, Leigh R Hochberg, John P Donoghue, and Michael J Black. Neural control of computer cursor velocity by decoding motor cortical spiking activity in humans with tetraplegia. *Journal of neural engineering*, 5(4):455, 2008.

[165] Karl J Friston, Andre M Bastos, Dimitris Pinotsis, and Vladimir Litvak. Lfp and oscillations—what do they tell us? *Current opinion in neurobiology*, 31:1–6, 2015.

[166] Eun Jung Hwang and Richard A Andersen. The utility of multichannel local field potentials for brain–machine interfaces. *Journal of neural engineering*, 10(4):046005, 2013.

[167] Yuwei Cui, Liu D Liu, James M McFarland, Christopher C Pack, and Daniel A Butts. Inferring cortical variability from local field potentials. *Journal of Neuroscience*, 36(14):4121–4135, 2016.

[168] Tong Wu, Wenfeng Zhao, Hongson Guo, Hubert Lim, and Zhi Yang. A streaming pca based vlsi chip for neural data compression. In *Biomedical Circuits and Systems Conference (BioCAS), 2016 IEEE*, pages 1–4. IEEE, 2016.

[169] Tong Wu, Wenfeng Zhao, Edward Keefer, and Zhi Yang. A lightweight deep compressive model for large-scale spike compression. In *2018 IEEE Biomedical Circuits and Systems Conference (BioCAS)*, pages 1–4. IEEE, 2018.

[170] S. Gibson, J. W. Judy, and D. Markoviċ. Spike Sorting: The First Step in Decoding the Brain. *IEEE Signal Processing Magazine*, 29(1):124–143, 2012.

[171] Lei Liu, Lei Yao, Xiaodan Zou, Wang Ling Goh, and Minkyu Je. Neural recording front-end ic using action potential detection and analog buffer with digital delay for data compression. In *Engineering in Medicine and Biology Society (EMBC), 2013 35th Annual International Conference of the IEEE*, pages 747–750. IEEE, 2013.

[172] Jeremy Holleman, Apurva Mishra, Chris Diorio, and Brian Otis. A micro-power neural spike detector and feature extractor in. 13μm cmos. In *Custom Integrated Circuits Conference, 2008. CICC 2008. IEEE*, pages 333–336. IEEE, 2008.

[173] Ermis Koutsos, Sivylla E Paraskevopoulou, and Timothy G Constandinou. A 1.5 μw neo-based spike detector with adaptive-threshold for calibration-free multichannel neural interfaces. In *Circuits and Systems (ISCAS), 2013 IEEE International Symposium on*, pages 1922–1925. IEEE, 2013.

[174] Yang-Guo Li, Qingyun Ma, Mohammad Rafiqul Haider, and Yehia Massoud. Ultra-low-power high sensitivity spike detectors based on modified nonlinear energy operator. In *Circuits and Systems (ISCAS), 2013 IEEE International Symposium on*, pages 137–140. IEEE, 2013.

[175] Enyi Yao, Yi Chen, and Arindam Basu. A 0.7 v, 40 nw compact, current-mode neural spike detector in 65 nm cmos. *IEEE transactions on biomedical circuits and systems*, 10(2):309–318, 2016.

[176] A. Rodriguez-Perez, J. Ruiz-Amaya, M. Delgado-Restituto, and A. Rodriguez-Vazquez. A Low-Power Programmable Neural Spike Detection Channel With Embedded Calibration and Data Compression. *IEEE Trans. Biomed. Circuits Syst.*, 6:87–100, 2012.

[177] V. Karkare, S. Gibson, and D. Marković. A 130-μW, 64-Channel Neural Spike-Sorting DSP Chip. *IEEE J. Solid-State Circuits*, 46:1214–1222, 2011.

[178] Vaibhav Karkare, Sarah Gibson, and Dejan Marković. A 75-μW, 16-Channel Neural Spike-Sorting Processor With Unsupervised Clustering. *IEEE J. Solid-State Circuits*, 48(9):2230–2238, Sep 2013.

[179] T-T Liu and Jan M Rabaey. A 0.25 v 460 nw asynchronous neural signal processor with inherent leakage suppression. *Solid-State Circuits, IEEE Journal of*, 48(4):897–906, 2013.

[180] Sivylla E Paraskevopoulou, Deren Y Barsakcioglu, Mohammed R Saberi, Amir Eftekhar, and Timothy G Constandinou. Feature extraction using first and second

derivative extrema (fsde) for real-time and hardware-efficient spike sorting. *Journal of neuroscience methods*, 215(1):29–37, 2013.

[181] Aritra Bhaduri, Enyi Yao, and Arindam Basu. Pulse-based feature extraction for hardware-efficient neural recording systems. In *Circuits and Systems (ISCAS), 2016 IEEE International Symposium on*, pages 1842–1845. IEEE, 2016.

[182] Awais M K and Andrew J M. Computationally Efficient Neural Feature Extraction for Spike Sorting in Implantable High-Density Recording Systems. *IEEE Trans. Neural Syst. Rehabil. Eng.*, 21(1):1–9, 2013.

[183] Stephen O'Driscoll, Krishna V Shenoy, and Teresa H Meng. Adaptive resolution adc array for an implantable neural sensor. *IEEE transactions on biomedical circuits and systems*, 5(2):120–130, 2011.

[184] Hyejung Kim, Sunyoung Kim, Nick Van Helleputte, Antonio Artes, Mario Konijnenburg, Jos Huisken, Chris Van Hoof, and Refet Firat Yazicioglu. A configurable and low-power mixed signal soc for portable ecg monitoring applications. *IEEE transactions on biomedical circuits and systems*, 8(2):257–267, 2014.

[185] Tong Wu and Zhi Yang. Power-efficient vlsi implementation of a feature extraction engine for spike sorting in neural recording and signal processing. In *Control Automation Robotics & Vision (ICARCV), 2014 13th International Conference on*, pages 7–12. IEEE, 2014.

[186] T. Chen, K. Chen, Z. Yang, K. Cockerham, and W. Liu. A Biomedical Multiprocessor SoC for Closed-Loop Neuroprosthetic Applications. *Dig. Tech. Papers IEEE Int. Solid-State Circuits Conf.*, pages 434–435, 2009.

[187] Bo Yu, Terrence Mak, Xiangyu Li, Fei Xia, Alexandre Yakovlev, Yihe Sun, and Chi-Sang Poon. A reconfigurable hebbian eigenfilter for neurophysiological spike train analysis. In *Field Programmable Logic and Applications (FPL), 2010 International Conference on*, pages 556–561. IEEE, 2010.

[188] Wen-Jyi Hwang, Wei-Hao Lee, Shiow-Jyu Lin, and Sheng-Ying Lai. Efficient architecture for spike sorting in reconfigurable hardware. *Sensors*, 13(11):14860–14887, 2013.

[189] Tung-Chien Chen, Wentai Liu, and Liang-Gee Chen. Vlsi architecture of leading eigenvector generation for on-chip principal component analysis spike sorting system. In *2008 30th Annual International Conference of the IEEE Engineering in Medicine and Biology Society*, pages 3192–3195. IEEE, 2008.

[190] Jie Zhang, Srinjoy Mitra, Yuanming Suo, Andrew Cheng, Tao Xiong, Frederic Michon, Marleen Welkenhuysen, Fabian Kloosterman, Peter S Chin, Steven Hsiao, et al. A closed-loop compressive-sensing-based neural recording system. *Journal of neural engineering*, 12(3):036005, 2015.

[191] Mehdi Aghagolzadeh and Karim Oweiss. Compressed and distributed sensing of neuronal activity for real time spike train decoding. *IEEE Transactions on Neural Systems and Rehabilitation Engineering*, 17(2):116–127, 2009.

[192] Yuanming Suo, Jie Zhang, Tao Xiong, Peter S Chin, Ralph Etienne-Cummings, and Trac D Tran. Energy-efficient multi-mode compressed sensing system for implantable neural recordings. *IEEE transactions on biomedical circuits and systems*, 8(5):648–659, 2014.

[193] Mohammad Ali Shaeri and Amir M Sodagar. A method for compression of intra-cortically-recorded neural signals dedicated to implantable brain–machine interfaces. *Neural Systems and Rehabilitation Engineering, IEEE Transactions on*, 23(3):485–497, 2015.

[194] Yuning Yang, C Sam Boling, Awais M Kamboh, and Andrew J Mason. Adaptive threshold neural spike detector using stationary wavelet transform in cmos. *Neural Systems and Rehabilitation Engineering, IEEE Transactions on*, 23(6):946–955, 2015.

[195] Hossein Hosseini-Nejad, Abumoslem Jannesari, and Amir M Sodagar. Data compression in brain-machine/computer interfaces based on the walsh–hadamard transform. *Biomedical Circuits and Systems, IEEE Transactions on*, 8(1):129–137, 2014.

[196] Karim G Oweiss, Andrew Mason, Yasir Suhail, Awais M Kamboh, and Kyle E Thomson. A scalable wavelet transform vlsi architecture for real-time signal processing in high-density intra-cortical implants. *Circuits and Systems I: Regular Papers, IEEE Transactions on*, 54(6):1266–1278, 2007.

[197] Awais M Kamboh, Matthew Raetz, Karim G Oweiss, and Andrew Mason. Area-power efficient vlsi implementation of multichannel dwt for data compression in implantable neuroprosthetics. *Biomedical Circuits and Systems, IEEE Transactions on*, 1(2):128–135, 2007.

[198] Letelier J and Wever P. Spike sorting based on discrete wavelet transform coefficients. *J. Neurosci. Meth.*, 101:93–106, 2000.

[199] Stefan Craciun, David Cheney, Karl Gugel, Justin C Sanchez, and Jose C Principe. Wireless transmission of neural signals using entropy and mutual information compression. *IEEE Transactions on Neural Systems and Rehabilitation Engineering*, 19(1):35–44, 2011.

[200] Buzsaki G. Large-scale recording of neuronal ensembles. *Nature Neurosci.*, 7:446–451, 2004.

[201] Philipp Berens, Georgios A Keliris, Alexander S Ecker, Nikos K Logothetis, and Andreas S Tolias. Feature selectivity of the gamma-band of the local field potential in primate primary visual cortex. *Frontiers in neuroscience*, 2:37, 2008.

[202] CE Schroeder, CE Tenke, and SJ Givre. Subcortical contributions to the surface-recorded flash-vep in the awake macaque. *Electroencephalography and Clinical Neurophysiology/Evoked Potentials Section*, 84(3):219–231, 1992.

[203] Henze D A, Borhegyi Z, Csicsvari J, Mamiya A, Harris K D, and Buzsaki G. Intracellular Features Predicted by Extracellular Recordings in the Hippocampus In Vivo. *J. Neurophysiol.*, 84(1):390–400, 2000.

[204] Adam Kohn, Ruben Coen-Cagli, Ingmar Kanitscheider, and Alexandre Pouget. Correlations and neuronal population information. *Annual review of neuroscience*, 39:237–256, 2016.

[205] Maneesh Sahani, John S Pezaris, and Richard A Andersen. On the separation of signals from neighboring cells in tetrode recordings. *Advances in neural information processing systems*, pages 222–228, 1998.

[206] Allen Gersho and Robert M Gray. *Vector Quantization and Signal Compression.* Springer US, 1992.

[207] S. Roweis. EM Algorithms for PCA and SPCA. *Adv. Neural Inf. Process. Syst.*, 10:626–632, 1998.

[208] Olivier Cappé and Eric Moulines. On-line expectation–maximization algorithm for latent data models. *Journal of the Royal Statistical Society: Series B (Statistical Methodology)*, 71(3):593–613, 2009.

[209] Dimitrios A Adamos, Efstratios K Kosmidis, and George Theophilidis. Performance evaluation of pca-based spike sorting algorithms. *Computer methods and programs in biomedicine*, 91(3):232–244, 2008.

[210] Wenfeng Zhao, Yajun Ha, and Massimo Alioto. Novel self-body-biasing and statistical design for near-threshold circuits with ultra energy-efficient aes as case study. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, 23(8):1390–1401, 2015.

[211] Ying-Lun Chen, Wen-Jyi Hwang, and Chi-En Ke. An efficient vlsi architecture for multi-channel spike sorting using a generalized hebbian algorithm. *Sensors*, 15(8):19830–19851, 2015.

[212] Yuan-Jyun Chang, Wen-Jyi Hwang, and Chih-Chang Chen. A low cost vlsi architecture for spike sorting based on feature extraction with peak search. *Sensors*, 16(12):2084, 2016.

[213] Michael E Tipping and Christopher M Bishop. Probabilistic principal component analysis. *Journal of the Royal Statistical Society: Series B (Statistical Methodology)*, 61(3):611–622, 1999.

[214] T. Wu, J. Xu, Y. Lian, A. Khalili, A. Rastegarnia, C. Guan, and Z. Yang. A 16-channel nonparametric spike detection asic based on ec-pc decomposition. *IEEE Transactions on Biomedical Circuits and Systems*, 10(1):3–17, Feb 2016.

[215] Tong Wu and Zhi Yang. An 8-channel neural spike processing ic with unsupervised closed-loop control based on spiking probability estimation. In *2014 36th Annual International Conference of the IEEE Engineering in Medicine and Biology Society*, pages 5260–5263. IEEE, 2014.

[216] Tong Wu. A Multichannel Integrated Circuit for Neural Spike Detection Based on EC-PC Threshold Estimation. *35th Ann. Int. Conf. of the IEEE Engineering in Medicine and Biology Society (EMBC)*, pages 779–782, Jul. 2013.

[217] Yin Zhou, Tong Wu, Amir Rastegarnia, Cuntai Guan, Edward Keefer, and Zhi Yang. On the robustness of EC–PC spike detection method for online neural recording. *Journal of neuroscience methods*, 235:316–330, 2014.

[218] Z. Yang, W. Liu, M. R. Keshtkaran, Y. Zhou, J. Xu, V. Pikov, C. Guan, and Y. Lian. A New EC-PC Threshold Estimation Method for in-*vivo* Neural Spike Detection. *J. Neural Eng.*, 9(4), 2012.

[219] Wing-Kin Tam, Rosa So, Cuntai Guan, and Zhi Yang. EC-PC spike detection for high performance brain-computer interface. In *Engineering in Medicine and Biology Society (EMBC), 2015 37th Annual International Conference of the IEEE*, pages 5142–5145. IEEE, 2015.

[220] Yoshua Bengio et al. Learning deep architectures for ai. *Foundations and trends® in Machine Learning*, 2(1):1–127, 2009.

[221] Lucas Theis, Wenzhe Shi, Andrew Cunningham, and Ferenc Huszár. Lossy image compression with compressive autoencoders. *arXiv preprint arXiv:1703.00395*, 2017.

[222] Johannes Ballé, Valero Laparra, and Eero P Simoncelli. End-to-end optimized image compression. *arXiv preprint arXiv:1611.01704*, 2016.

[223] Aaron van den Oord, Oriol Vinyals, and koray kavukcuoglu. Neural discrete representation learning. In I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, editors, *Advances in Neural Information Processing Systems 30*, pages 6306–6315. Curran Associates, Inc., 2017.

[224] Jie Hu, Li Shen, and Gang Sun. Squeeze-and-excitation networks. *arXiv preprint arXiv:1709.01507*, 2017.

[225] Nadav Cohen, Ronen Tamari, and Amnon Shashua. Boosting dilated convolutional networks with mixed tensor decompositions. In *International Conference on Learning Representations*, 2018.

[226] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016.

[227] Andreas Veit, Michael J Wilber, and Serge Belongie. Residual networks behave like ensembles of relatively shallow networks. In *Advances in Neural Information Processing Systems*, pages 550–558, 2016.

[228] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems*, pages 1097–1105, 2012.

[229] Saining Xie, Ross Girshick, Piotr Dollár, Zhuowen Tu, and Kaiming He. Aggregated residual transformations for deep neural networks. In *Computer Vision and Pattern Recognition (CVPR), 2017 IEEE Conference on*, pages 5987–5995. IEEE, 2017.

[230] Robert M Gray. *Source coding theory*, volume 83. Springer Science & Business Media, 2012.

[231] Adam Paszke, Sam Gross, Soumith Chintala, Gregory Chanan, Edward Yang, Zachary DeVito, Zeming Lin, Alban Desmaison, Luca Antiga, and Adam Lerer. Automatic differentiation in pytorch. In *NIPS-W*, 2017.

[232] Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

[233] Kenneth D Harris, Darrell A Henze, Jozsef Csicsvari, Hajime Hirase, and György Buzsáki. Accuracy of tetrode spike separation as determined by simultaneous intracellular and extracellular measurements. *Journal of neurophysiology*, 84(1):401–414, 2000.

[234] Charles M Gray, Pedro E Maldonado, Mathew Wilson, and Bruce McNaughton. Tetrodes markedly improve the reliability and yield of multiple single-unit isolation from multi-unit recordings in cat striate cortex. *Journal of neuroscience methods*, 63(1-2):43–54, 1995.

[235] Carolina Mora Lopez, Srinjoy Mitra, Jan Putzeys, Bogdan Raducanu, Marco Ballini, Alexandru Andrei, Simone Severi, Marleen Welkenhuysen, Chris Van Hoof, Silke

Musa, et al. 22.7 a 966-electrode neural probe with 384 configurable channels in 0.13 μm soi cmos. In *Solid-State Circuits Conference (ISSCC), 2016 IEEE International*, pages 392–393. IEEE, 2016.

[236] Biao Sun, Wenfeng Zhao, and Xinshan Zhu. Training-free compressed sensing for wireless neural recording using analysis model and group weighted-minimization. *Journal of neural engineering*, 14(3):036018, 2017.

[237] Kevin Q Shan, Evgueniy V Lubenov, and Athanassios G Siapas. Model-based spike sorting with a mixture of drifting t-distributions. *Journal of neuroscience methods*, 288:82–98, 2017.

[238] Piotr Bojanowski and Armand Joulin. Unsupervised learning by predicting noise. In Doina Precup and Yee Whye Teh, editors, *Proceedings of the 34th International Conference on Machine Learning*, volume 70 of *Proceedings of Machine Learning Research*, pages 517–526, International Convention Centre, Sydney, Australia, 06–11 Aug 2017. PMLR.

[239] Marius Pachitariu, Nicholas A Steinmetz, Shabnam N Kadir, Matteo Carandini, and Kenneth D Harris. Fast and accurate spike sorting of high-channel count probes with kilosort. In *Advances in Neural Information Processing Systems*, pages 4448–4456, 2016.

[240] Vivienne Sze, Yu-Hsin Chen, Tien-Ju Yang, and Joel S Emer. Efficient processing of deep neural networks: A tutorial and survey. *Proceedings of the IEEE*, 105(12):2295–2329, 2017.

[241] Yu-Hsin Chen, Tushar Krishna, Joel S Emer, and Vivienne Sze. Eyeriss: An energy-efficient reconfigurable accelerator for deep convolutional neural networks. *IEEE Journal of Solid-State Circuits*, 52(1):127–138, 2017.

[242] Jian Xu, Tong Wu, Wentai Liu, and Zhi Yang. A frequency shaping neural recorder with 3 pf input capacitance and 11 plus 4.5 bits dynamic range. *IEEE transactions on biomedical circuits and systems*, 8(4):510–527, 2014.

[243] Patrick D Wolf. Thermal considerations for the design of an implanted cortical brain–machine interface (bmi). *Indwelling Neural Implants: Strategies for Contending with the In Vivo Environment*, pages 33–38, 2008.

[244] Michael S Lewicki. A review of methods for spike sorting: the detection and classification of neural action potentials. *Network: Computation in Neural Systems*, 9(4):R53–R78, 1998.

[245] Pierre Yger, Giulia LB Spampinato, Elric Esposito, Baptiste Lefebvre, Stéphane Deny, Christophe Gardella, Marcel Stimberg, Florian Jetter, Guenther Zeck, Serge Picaud, et al. A spike sorting toolbox for up to thousands of electrodes validated with ground truth recordings in vitro and in vivo. *ELife*, 7:e34518, 2018.

[246] Gerrit Hilgen, Martino Sorbaro, Sahar Pirmoradian, Jens-Oliver Muthmann, Ibolya Edit Kepiro, Simona Ullo, Cesar Juarez Ramirez, Albert Puente Encinas, Alessandro Maccione, Luca Berdondini, et al. Unsupervised spike sorting for large-scale, high-density multielectrode arrays. *Cell reports*, 18(10):2521–2532, 2017.

[247] Jason E Chung, Jeremy F Magland, Alex H Barnett, Vanessa M Tolosa, Angela C Tooker, Kye Y Lee, Kedar G Shah, Sarah H Felix, Loren M Frank, and Leslie F Greengard. A fully automated approach to spike sorting. *Neuron*, 95(6):1381–1394, 2017.

[248] Jin Hyung Lee, David E Carlson, Hooshmand Shokri Razaghi, Weichi Yao, Georges A Goetz, Espen Hagen, Eleanor Batty, EJ Chichilnisky, Gaute T Einevoll, and Liam Paninski. Yass: Yet another spike sorter. In *Advances in Neural Information Processing Systems*, pages 4002–4012, 2017.

[249] Gopal Santhanam, Maneesh Sahani, Stephen I Ryu, and Krishna V Shenoy. An extensible infrastructure for fully automated spike sorting during online experiments. In *Engineering in Medicine and Biology Society, 2004. IEMBS'04. 26th Annual International Conference of the IEEE*, volume 2, pages 4380–4384. IEEE, 2004.

[250] Kyung Hwan Kim. A fully-automated neural spike sorting based on projection pursuit and gaussian mixture model. In *Conference Proceedings. 2nd International IEEE EMBS Conference on Neural Engineering, 2005.*, pages 151–154. IEEE, 2005.

[251] Fernando Chaure, Hernan Gonzalo Rey, and Rodrigo Quian Quiroga. A novel and fully automatic spike sorting implementation with variable number of features. *Journal of neurophysiology*, 2018.

[252] Yann LeCun, Yoshua Bengio, and Geoffrey Hinton. Deep learning. *nature*, 521(7553):436, 2015.

[253] Shaoqing Ren, Kaiming He, Ross Girshick, and Jian Sun. Faster r-cnn: Towards real-time object detection with region proposal networks. In *Advances in neural information processing systems*, pages 91–99, 2015.

[254] Kai Yang, Haifeng Wu, and Yu Zeng. A simple deep learning method for neuronal spike sorting. In *Journal of Physics: Conference Series*, volume 910, page 012062. IOP Publishing, 2017.

[255] Muhammad Saif-ur Rehman, Robin Lienkämper, Yaroslav Parpaley, Jörg Wellmer, Charles Liu, Brian Lee, Spencer Kellis, Richard A Andersen, Ioannis Iossifidis, Tobias Glasmachers, et al. Spikedeeptector: A deep-learning based method for detection of neural spiking activity. *Journal of neural engineering*, 2019.

[256] Alireza Makhzani, Jonathon Shlens, Navdeep Jaitly, and Ian Goodfellow. Adversarial autoencoders. In *International Conference on Learning Representations*, 2016.

[257] Marcin Andrychowicz, Misha Denil, Sergio Gomez, Matthew W Hoffman, David Pfau, Tom Schaul, Brendan Shillingford, and Nando De Freitas. Learning to learn by gradient descent by gradient descent. In *Advances in Neural Information Processing Systems*, pages 3981–3989, 2016.

[258] Yizong Cheng. Mean shift, mode seeking, and clustering. *IEEE transactions on pattern analysis and machine intelligence*, 17(8):790–799, 1995.

[259] Diederik P Kingma and Max Welling. Auto-encoding variational bayes. *arXiv preprint arXiv:1312.6114*, 2013.

[260] Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative adversarial nets. In *Advances in neural information processing systems*, pages 2672–2680, 2014.

[261] Mariya Toneva, Alessandro Sordoni, Remi Tachet des Combes, Adam Trischler, Yoshua Bengio, and Geoffrey J. Gordon. An empirical study of example forgetting during deep neural network learning. In *International Conference on Learning Representations*, 2019.

[262] Jost Tobias Springenberg. Unsupervised and semi-supervised learning with categorical generative adversarial networks. *arXiv preprint arXiv:1511.06390*, 2015.

[263] Fenglong Ma, Yaqing Wang, Houping Xiao, Ye Yuan, Radha Chitta, Jing Zhou, and Jing Gao. A general framework for diagnosis prediction via incorporating medical code descriptions. In *2018 IEEE International Conference on Bioinformatics and Biomedicine (BIBM)*, pages 1070–1075. IEEE, 2018.

[264] Nenad Tomašev, Xavier Glorot, Jack W Rae, Michal Zielinski, Harry Askham, Andre Saraiva, Anne Mottram, Clemens Meyer, Suman Ravuri, Ivan Protsyuk, et al. A clinically applicable approach to continuous prediction of future acute kidney injury. *Nature*, 572(7767):116, 2019.

[265] Xi Zhang, Jingyuan Chou, Jian Liang, Cao Xiao, Yize Zhao, Harini Sarva, Claire Henchcliffe, and Fei Wang. Data-driven subtyping of parkinson's disease using longitudinal clinical records: a cohort study. *Scientific reports*, 9(1):797, 2019.

[266] Xiangrui Cai, Jinyang Gao, Kee Yuan Ngiam, Beng Chin Ooi, Ying Zhang, and Xiaojie Yuan. Medical concept embedding with time-aware attention. In *Proceedings of the 27th International Joint Conference on Artificial Intelligence*, pages 3984–3990. AAAI Press, 2018.

[267] Anis Sharafoddini, Joel A Dubin, and Joon Lee. Patient similarity in prediction models based on health data: a scoping review. *JMIR medical informatics*, 5(1):e7, 2017.

[268] Qiuling Suo, Weida Zhong, Fenglong Ma, Yuan Ye, Mengdi Huai, and Aidong Zhang. Multi-task sparse metric learning for monitoring patient similarity progression. In *2018 IEEE International Conference on Data Mining (ICDM)*, pages 477–486. IEEE, 2018.

[269] Petar Veličković, Guillem Cucurull, Arantxa Casanova, Adriana Romero, Pietro Lio, and Yoshua Bengio. Graph attention networks. *arXiv preprint arXiv:1710.10903*, 2017.

[270] Andrew L Maas, Awni Y Hannun, and Andrew Y Ng. Rectifier nonlinearities improve neural network acoustic models. In *Proc. icml*, volume 30, page 3, 2013.

[271] Xiao Huang, Qingquan Song, Yuening Li, and Xia Hu. Graph recurrent networks with attributed random walks. In *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, KDD '19, pages 732–740. ACM, 2019.

[272] Xiao Wang, Houye Ji, Chuan Shi, Bai Wang, Yanfang Ye, Peng Cui, and Philip S Yu. Heterogeneous graph attention network. In *The World Wide Web Conference*, WWW '19, pages 2022–2032. ACM, 2019.

[273] Andrew Y Ng, Michael I Jordan, and Yair Weiss. On spectral clustering: Analysis and an algorithm. In *Advances in neural information processing systems*, pages 849–856, 2002.

[274] Daniel D Lee and H Sebastian Seung. Algorithms for non-negative matrix factorization. In *Advances in neural information processing systems*, pages 556–562, 2001.

[275] James Kirkpatrick, Razvan Pascanu, Neil Rabinowitz, Joel Veness, Guillaume Desjardins, Andrei A Rusu, Kieran Milan, John Quan, Tiago Ramalho, Agnieszka Grabska-Barwinska, et al. Overcoming catastrophic forgetting in neural networks. *Proceedings of the national academy of sciences*, 114(13):3521–3526, 2017.

[276] Aaron van den Oord, Yazhe Li, and Oriol Vinyals. Representation learning with contrastive predictive coding. *arXiv preprint arXiv:1807.03748*, 2018.

[277] R Devon Hjelm, Alex Fedorov, Samuel Lavoie-Marchildon, Karan Grewal, Phil Bachman, Adam Trischler, and Yoshua Bengio. Learning deep representations by mutual information estimation and maximization. *arXiv preprint arXiv:1808.06670*, 2018.

[278] Spyros Gidaris, Praveer Singh, and Nikos Komodakis. Unsupervised representation learning by predicting image rotations. In *ICLR 2018*, 2018.

[279] Carl Vondrick, Abhinav Shrivastava, Alireza Fathi, Sergio Guadarrama, and Kevin Murphy. Tracking emerges by colorizing videos. In *Proceedings of the European Conference on Computer Vision (ECCV)*, pages 391–408, 2018.

[280] Marco Tulio Ribeiro, Sameer Singh, and Carlos Guestrin. " why should i trust you?" explaining the predictions of any classifier. In *Proceedings of the 22nd ACM SIGKDD international conference on knowledge discovery and data mining*, pages 1135–1144, 2016.

[281] Cecilia Panigutti, Alan Perotti, and Dino Pedreschi. Doctor xai: an ontology-based approach to black-box sequential data classification explanations. In *Proceedings of the 2020 Conference on Fairness, Accountability, and Transparency*, pages 629–639, 2020.