INVARIANT INFORMATION SPIKE SORTING

by

Alexander Karavos

Submitted in partial fulfillment of the requirements
for the degree of Masters of Science

at

Dalhousie University
Halifax, Nova Scotia
March 2024

Table of Contents

List of Tables

List of Figures

Abstract

Spike sorting is the process of identifying and classifying voltage recordings from the brain or nervous system into discrete labelled waveform events. The core difficulty lies in unsupervised classification - one does not have definitive labels for signals, or how many unique labels there are per recording. Recent works within the field have converged on a prevalent architecture for approaching classification: feature extraction (FE) followed by a traditional clustering algorithm (CA). While there is unanimity in architecture, there is ambiguity as to what techniques to use for a given problem due to: inconsistency across datasets, arbitrary parameterization, abstract representation, and no standard dataset for sorting n independent waveforms. Given these issues, our goal was to challenge typical architecture with a deep learning based approach. It is worth noting that there have already been attempts at adding deep learning to spike sorting that act as extensions on prior methods. We propose to remove FE entirely by extending Invariant Information Clustering (IIC) - a method built for image classification - to spike sorting; thus creating Invariant Information Spike Sorting (IISS). IISS uses a physics inspired transform Φ, such as background noise addition, to create paired spike data $[x, \Phi(x)]$ where one is a plausible facsimile of the other. A neural network learns to predict identical δ distributions for paired spikes by maximizing mutual information between pair's predicted classes. Clusters emerge after learning core semantics, ensuring inter-cluster variance surpasses Φ induced differences. To test our model against classical clustering approaches we developed 4 simulation waveform datasets that mimic real data taken from the peripheral nervous system. To steel-man against IISS we compared 18 possible pairings of FE+CA solutions given optimal parameterization - a highly improbable event - while using a single parameter setting for IISS. We find IISS comparable or superior across datasets. The parameter choices for IISS are intuitive and stable. The final design can be seen as a first draft with substantial scope for enhancements. Consequently IISS demonstrated capacity to supersede traditional methods and paves the way for more intuitive, robust, physics-grounded spike sorting.

List of Abbreviations and Symbols Used

K_{max} Overclustering Output dimension used for heads. 44

Φ Transform operator used to create a noisy facsimile. 38

α Low frequency scaling bound. 39

σ Set of background noises. 40

m Number of heads used in IISS Architecture. 44

n Number of background noises added. 40

CA Clustering Algorithm. 9

CNN Convolutional Neural Networks. 12

DBSCAN Density Based Spatial Clustering of Applications With Noise. 28

DP Density Peaks. 30

FE Feature Extraction/Extractor. 9

GMM Gaussian Mixture Model. 26

HDBSCAN Hierarchical Density Based Spatial Clustering of Applications With Noise. 29

IIC Invariant Information Clustering. 35

IISS Invariant Information Spike Sorting. 45

j number of frequencies scaled. 39

MLP Multi-Layer Perceptron. 42

NMI Normalized Mutual Information. 25

PCA Principal Component Analysis. 5

UMAP Uniform Manifold Approximate Projection. 16

Acknowledgements

Thank you to Dr Guy Kember, who has adopted me as the longest two-year graduate student he has ever had.

Chapter 1

Introduction

1.1 Spike Sorting

Contemporary recording systems have gained capacity for continuous, high-frequency, extracellular voltage recordings of the nervous system and brain. As a result, the field of neuroscience is exploding with unprecedented amounts of data and potential experiments. Experimental findings crucially rely on processing recording data into neural behaviour for analysis. This pre-processing is known as *Spike Sorting*. Typically this is done in two independent phases: extraction and classification. Extraction of potential neural events aim to separate signal from noise, producing a set of neural activity timestamps from a voltage recording. Based on the timestamps, spiking waveforms makeup a dataset on which sorting can be done. The goal of sorting is to partition spikes into labelled groups where groups are homogeneous with respect to their own members and heterogeneous with respect to others [1]. An underlying principle is that unique neurons possess a signature template each time they fire and differences within a group are related to: noise, probe movements, or slight signature deviations. Accurate spike sorting allows detailed insights of neural behaviour at the single cell level, a key stepping stone in advancing the field.

In this work we will look into the later problem of sorting spikes. It is here we believe the majority of research has yet to be done and furthermore there is open debate on how it should be dealt with. From here on we assume we have an unclassified set of spike waveforms $x \in X, x = [v_0, v_1 ... v_n]$ and *sorting*[2] will refer to the creation of spike waveform classes and the assignment of individual spikes to a class.

[1] An analogy might be partitioning photos of pets. Dogs should all share like characteristics, and be sufficiently different than images of cats

[2] Through the remainder of this document spike sorting will be synonymous with spike: clustering, classifying, or labelling. Thus groups of the same spikes will be referred to as: clusters, label i, class i.

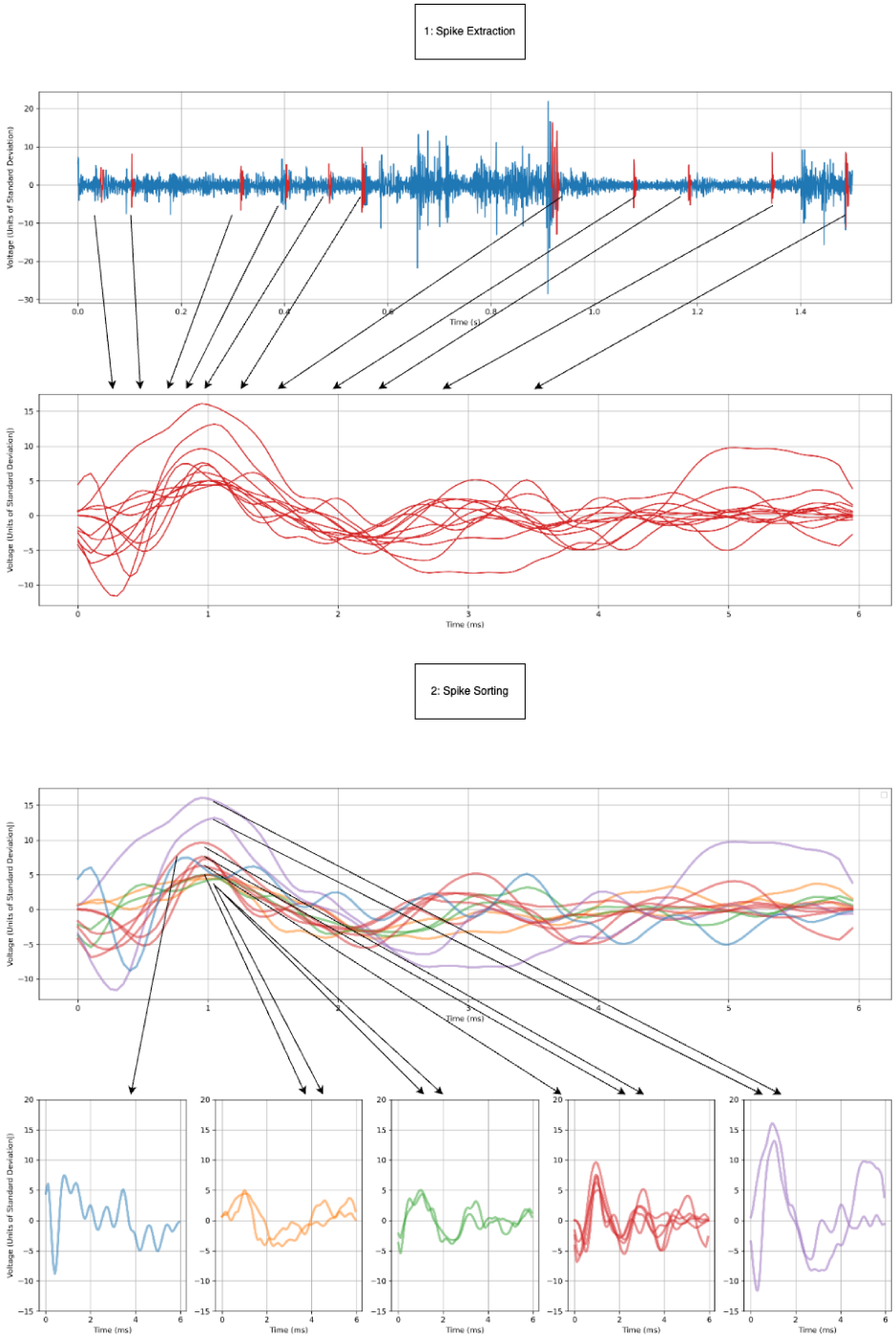


Figure 1.1: (A) Extraction: Spikes are identified within a long time-series sequence and grouped into a dataset for sorting. (B) Sorting: Spikes are sorted into unique events

The process of extraction and sorting a recording is visualized in Figure 1.1 where the set of spike waveforms X (red trace) is sorted into 5 classes. At a glance this process for 10 spikes appears trivial; however, sorting becomes difficult when we're dealing with 100,000 spikes. There is a stereotypical shape of a positive spike - a pronounced peak and trough (inverted for negative) followed by recovery to baseline; however aside from vast changes in height and width, there is no textbook margin we may draw to define unique behaviour. Even the height and width don't have specifically defined requirements. This margin problem is exacerbated by also not knowing how many unique events we are looking for. One may presume a low integer number (given the physics of the probe and surrounding somata) but not knowing a fixed number presents a considerable challenge since the margins change as a function of the set size!

This work is motivated by the task of sorting data from a series of 2019 experiments aimed at studying cardiopulmonary relationship with the peripheral nervous system. In this study the stellate ganglia[3] activity of pigs was put under inspection via incision of recording probes [28]. The neurons are recorded over long time periods of approximately 6 hours at 20kHz. Each experiment has an array-like probe that records 16 channels. Through spike extraction we find that each channel records on the order of 100,000 spike waveforms of length 6ms / 120-datapoints. The waveforms are extracted down to a signal-to-noise-ratio of approximately 3. Each channel is treated as its own dataset because the channels on the probes are spaced sufficiently far apart that voltages recorded at separate channels can be deemed independent [4]. Thus, we are tasked with finding a sorting method that can sort a large amount of independent waveform data into discovered classes. The findings of this thesis will be used in future works for potential physiological discovery in the peripheral nervous system. That said, the current goal is to find a contemporary and generic approach to waveform sorting.

[3]The stellate is a collection of sympathetic nerves found on the back of the neck[19]. For the non-physiologist reader - like myself - this is the extent of understanding you'll need for the remainder of this thesis.

[4]This is an uncommon property in modern spike sorting. Often probes are designed in a matrix-like shape and record the same signal from multiple angles or distances.

1.2 Overview of Current Spike Sorters

To gain insight of current approaches to spike sorting we will look at some of the prominent works from within the last decade. By no means is this an exhaustive literature review. The goal of this section is to get an idea of how the problem is tackled by the community. More details can found in the respective references.

WaveClust

WaveClust is a spike-sorting technique that employs wavelet transformations to spikes for analysis. This process is presumed to appropriate for non-stationary signals such as neural data[3]. The wavelet transform includes a feature extraction step where WaveClust is used to select the wavelet coefficients that are the least Gaussian. The principle of this step is that neural spikes tend to exhibit non-Gaussian distributions in the wavelet domain whereas noise tends to be Gaussian. After feature extraction, clustering is performed by Super Paramagnetic Clustering where the number of clusters is adjusted to meet a threshold set by a temperature parameter [5]. The resulting clusters are used to generate template spikes that can be matched to new or subsequent data.

Key Points	**Parameters**
Wavelet Transform: Uses wavelet transformation + non-Gaussian selection criterion as a feature extraction step to remove noisy aspects of the data and attack the essential features of spikes	Number of wavelet coefficients kept through the transform
Super Paramagnetic Clustering: Utilizing SPC for the clustering of the feature space is straightforward, although can be highly dependent on parameterization.	Temperature for SPC

[5] *Temperature* is a commonly used term for a parameter in Machine Learning. In this case it actually derives from statistical mechanics where at low temperatures there is less diverse behaviour amongst particles: In SPC this translates to 'temperature' controlling the clustering granularity, with low temperatures generally leading to fewer larger clusters. Unfortunately the algorithm lacks the physical context of what temperature should be; thus, selection of this parameter is arbitrary.

SpyKingCircus

SpyKingCircus [33] begins by applying Principal Component Analysis (PCA) to reduce the dimensionality of the spike data. PCA is a linear operation aimed at isolating the most significant features while retaining the essence of the original signals. Subsequently, a custom density-based algorithm inspired by popular algorithm Density Peaks Clustering [23] (DPClus) is applied to the PCA features to group spikes. Spike templates are generated through the mean and variances of the clustering results. The final step employs a cross-correlogram which checks for template dependencies on spike firing times.

Key Points	**Parameters**
PCA: Applies PCA to distill spike data into its most significant features.	Number of PCA components kept
Custom Density-Based Clustering: Employs a unique clustering algorithm that adapted from Density Peaks	Number of neighbours for local density estimates (Inherited from DPClus)
Template Generation and Matching: Generates templates from clusters then with a cross-correlogram step to ensure the accuracy and reliability of spike attribution.	Maximum number of clusters = 10
	Density threshold to be considered a cluster

IronClust

IronClust is a spike sorting algorithm which creates a self-supervised process for clustering [11]. The method is initializes using PCA for feature extraction followed by the DPClus clustering algorithm to generate a first set of labels. Once this step is completed IronClust introduces Linear Discriminant Analysis (LDA) - a supervised algorithm which given a feature space and labelled classes finds a new projection of the data that maximizes intra-class distance while simultaneously minimizing inter-class distance. LDA is applied to spikes given the initial labelling from the first pass of the PCA-DPClus and DPClus is run again to merge clusters. The LDA-DPClus-merging pairing is iteratively applied until a convergence criterion of the labelling is

met which ensures a stable labelling. [6]

Key Points	Parameters
PCA-DPClus: Applies PCA and Density Peaks Clustering as an initialization	Number of PCA components kept
LDA: Uses a supervised algorithm based on initialisation (making it self supervised) to find a feature space which is optimal for clustering the data.	Number of neighbours for local density estimates
Iterative clustering approach: Using the LDA-DPClus-merging iteratively, IronClust converges on a stable clustering.	Density threshold to be considered a cluster

JRClust

JRClust is a spike sorting pipeline aimed at optimizing the sorting process for high-density probes[4]. The main features are advances in preprocessing giving sufficient noise identification and removal using a custom channel-covariance strategy. Furthermore JRClust incorporates auto-detection of probe drift where signals are measured at multiple channels requiring waveform adjustments. The primary clustering mechanism is a PCA-DPClus pairing.

MountainSort

MountainSort is a nonparametric density-based spike sorting algorithm [5]. Feature extraction is first performed via PCA. The features are passed through the novel ISO-SPLIT algorithm. There are two main heuristics within ISO-SPLIT: firstly, spike templates form unimodal density distributions in the feature-space, add secondly these distributions are separated areas of relatively low density. By employing nonparametric tests for uni-modality ISO-SPLIT aims to be robust to cluster shape variation in a non-parametric way.

[6] This algorithm could be described as *iterative Contrastive Learning*, which although is out of scope for this thesis, there is more modern deep learning frameworks which are doing contrastive learning for spike sorting [31]

Key Points	Parameters
Preprocessing: The innovations in JRClus revolve around advances in noise detection across channels, along with capacity to detect movements in the probe.	Number of PCA components kept
Multi-Channel Dependence: The aforementioned preprocessing requires the space and time dependent channels.	Number of neighbours for local density estimates
PCA-DPClus: The clustering is performed by PCA-DPClus with no adaptations.	Density threshold to be considered a cluster

Key Points	Parameters
PCA: The paper suggests any feature extraction, but the authors used PCA.	Number of PCA components kept
ISO-SPLIT: A non-parametric approach to density-based clustering with suggested robustness.	

HerdingSpikes2

HerdingSpikes approaches spike sorting with a custom feature extraction method suited to matrix-probes [9]. Bary-center location estimates are obtained through spatio-temporal event maps of the waveform recorded from multiple angles across the matrix-probe . Location estimates are then used as a feature in tandem with PCA features for feature extraction method that combines spatial dependence of waveform morphology. These features are clustered by the Mean-Shift[7] Clustering algorithm.

Key Points	Parameters
PCA+Barycenter: The use of PCA and location estimation for feature extraction.	Number of components kept from PCA
MeanShift Clustering: Kilo-Sort employs a graph-based clustering which generates clusters through nearest neighbors.	MeanShift algorithm bandwidth parameter is used to control the density estimates.

KiloSort

Kilosort has been continually developed for the last ten years [20]. Kilosort splits spike extraction and clustering into 3 sections: pre-processing, template de-convolution and clustering. Pre-processing includes filtering and probe drift correction. The contemporary version of Kilosort uses a pre-trained feature extraction where clusters are discovered using a custom graph-based algorithm. The template de-convolution step is unique to Kilosort in that feature extraction evolves sequentially in time along with spike extraction. As spikes are found they're processed through the feature extraction clustering pipeline and create or add to clusters. The averages of clusters are used to form templates. These templates are then used for template de-convolution: overlapping templates are found within the signal. On conclusion of the signal the clustering is repeated on the final dataset of waveforms. The graph-based clustering approach connects nearest neighbours in the feature space and performs splits/mergers. The method concludes with learned templates that are aligned temporally and similar templates are merged through a cross-correlation inspection.

Key Points	**Parameters**
Pre-trained PCA feature extraction	Variable amounts depending on version of Kilosort used. See reference for details
Graph-Based Clustering	
Sequential Extraction and Clustering	
Cross Correlation Inspection on conclusion	

Summary

In conclusion, a consistent architectural theme is evident across all the spike sorting methods outlined: each of them employ a form of feature extraction followed by a clustering algorithm. Aside from the first mention of wavelets PCA is present in all methods. Each method is tailored to enhance unique signal characteristics with respect to noise. The subsequent clustering algorithms whether density-based, graph-based, or iterative in nature, are designed to group the unique features into meaningful clusters representing activity associated with individual neurons. Despite the apparent diversity in spike-sorting approaches their underlying architecture is the

same. An example of the clustering process similarity is shown in Figure 1.2. While Kilosort injects a degree of innovation into spike sorting with an iterative updating of clusters across time it retains the two-part clustering architecture.

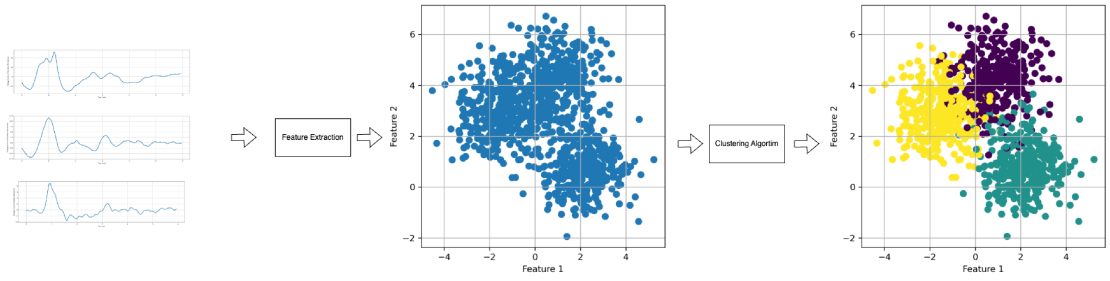


Figure 1.2: Common spike sorting can be broken into two common components. 1. Reducing the spatial dimensions explored by the problem by feature extraction. 2. Clustering the feature space.

We can break down all aforementioned methods into their respective two-part clustering solutions seen in Table 1.1. We will refer to the pairing of a feature extractor and a clustering algorithm as the *classical architecture* or the FE+CA pairing.

Sorter	**Feature Extraction**	**Clustering Algorithm**
WaveClus	Wavelet transforms	Superparamagnetic Clustering
SpyKingCircus	PCA	Custom Density Clustering + Spike Time Merging
IronClust	PCA	Density Peaks + LDA + Merging Loop
JRClust	Channel Covariances + PCA	Density Peaks
MountainSort	PCA	ISO-SPLIT
KiloSort(4)	Pre-trained PCA	Custom Graph Clustering
HerdingSpikes2	Barycenter estimates + PCA	Mean Shift

Table 1.1: 2-Part Clustering Solutions of Spike Sorters

1.3 Problem Set

During our attempts to apply existing spike-sorting methods to our measured data, we were led to a pivotal realization: the selection of the spike-sorter we should use is not possible since there is no ground truth. Hence, there is no independent assessment of sorter effectiveness as a function of input data difficulties - they are all proposed as the general solution to spike-sorting. We believe that the spike sorting problem is a subset of unsupervised learning: *'What is an approach to cluster high dimensional data?'*. Ignoring this aspect has led to the construction of data-specific methods sharing a common framework whose dependencies and limitations, with respect to the data being analyzed, are not considered. Consequently, how to sort neural data remains an open question, that continues to be considered piecemeal with no guiding framework. With this realization we set out to define a core set of challenges that must be overcomes if we are to build a more generic spike-sorting framework.

Ill-Posed

In a recent review paper [2] Buccino et al set out to test methods against each-other and compile modern works into a python package *SpikeInterface*. The comparison results were decidedly lackluster. The primary result showed none of the methods agree across datasets more than random chance would dictate. This led to the employment of two experts to manually label spike data so as to investigate which methods most resemble human clustering. Two results were from expert-labelled data: 1. No sorter agreed with human experts to high levels (above 75%) and 2. The human experts showed significant divergence among their data classification. This lack of human expert agreement demonstrates ambiguity in defining what constitutes a unique neural signal. The paper concludes by suggesting agglomeration of many sorting techniques may lead to the best approach as the spike-units agreed upon by many sorters often were correct. It is our opinion that this is a null-result. The practical implication is that clusters found by all sorters are obvious clusters which implies a failure to classify less obvious spikes. Note that 'less obvious' spikes are not less physiologically relevant because they may occur less frequently and thus be missed by a spike sorter. Lower activity in specific neurons can be associated with significant biological events

due to circuit architecture.

The findings from the SpikeInterface paper reinforce a basic feature of this problem: spike sorting is ill-posed as defined by the mathematician Jacques Hadamard. According to Hadamard's criteria for a well-posed problem, three conditions must be met: first, a solution must exist; second, there must be a unique solution; and third, the solution's behavior must remain stable with respect to parameterization. In the context of spike sorting, we can confidently assert that the first condition holds, as the physical problem has a discrete number of neurons that occupy unique locations. However, the other two criteria pose significant challenges. The absence of a reliable verification method has led to a lack of consensus among experts regarding what constitutes a definitive solution (condition two). Moreover, the performance of a single model across different datasets is highly variable due to the influence of numerous unpredictable parameters based on assumptions that may be inappropriate for a dataset - this renders the third condition unmet. This variability is evident in the wide-ranging outcomes observed across datasets, as detailed by [2, 15]. Consequently, the ill-posed nature of spike sorting underscores the complexity and ongoing challenges inherent in this field.

Generalized Methods

A problem tackled by SpikeInterface is that sorting algorithms are commonly packaged as a start-to-finish pipeline to neural data analysis that includes spike extraction, feature extraction, and clustering. While this is a step in the right direction there remains no capacity to swap pipeline components. For instance, what happens when using one pipeline's feature extractor with another piperline's clustering process? Inspection of components will lead to more thoughtful design and improved understanding of limitations.

Furthermore, the available narrow pipeline packaging of methods does not lend itself to extension to tangential problems - which includes our neural pig recordings. Our simple problem of sorting neural individual waveforms does not fit into any of the aforementioned packaged pipelines. This is because 1. we have significantly lower

signal-to-noise ratio and 2. there are no correlated channels to aid in signal filtering. Because our data does not fit into the structure of pipelines based on matrix-array/low-noise methods we must build a suitable pipeline based on components that have never been individually tested. This inability to swap and test pipeline components on differing datasets limits the scope of the available methods for research purposes.

Deep Learning Frameworks

At present research into deep-learning is dominated by the development of very large language and vision models. However the use of large neural networks is still relatively novel in the field of spike sorting with most works appearing in the past few years. The adoption of modern AI methods has led to enhancements of existing methods: in [22] it is found that embeddings and online spike classification can be improved through the use of Auto-Encoding neural networks. Contrastive learning [31] has been shown to increase performance of feature extractions by using labels produced by Kilosort to train a contrastive network. Convolutional neural networks (CNNs) [34] are useful to improve identification of overlapping signals. While these works are significant they are still using large neural networks to augment existing methods with a dependence on labelled data and/or classical FE+CA methodology. In other words, a layer of deep learning has been used to provide some improvement of existing classical methods.

Large neural networks are able to reduce the clustering pipeline to one step. That to say the neural networks should be used to provide the labels in an unsupervised fashion without relying on a historical clustering algorithm to produce an initial set of labels to turn it into a supervised problem. Revolutionary advancements in unsupervised vision recognition have solved datasets that were traditionally labelled and then used in supervised learning [10, 30, 32] . While image recognition has certain unique aspects there are parallels with the spike sorting problem: a large data corpus, a relatively low number of classes, and correlated data (time series vs RGB colouring). It is our belief that vision methods have not made the jump outside of vision due to rapid development and a lack of generalizability: the methods are quite specific to vision. Those working in vision are building vision networks and we are building

spike sorters. At heart though both are part of the broader class of problem where large amount of data are clustered. It is not possible to publish research based on PCA and a density algorithm to cluster images due to the overwhelming success of CNNs. Yet, the former remains state of the art in spike sorting which drives much of brain research. Development of truly novel and generic spike-sorting methods are lagging progress made in machine learning specifically related to vision.

Summary

The current state of spike-sorting, particularly in the context of noisy univariate recordings, presents several intertwined challenges. Spike-sorting is ill-posed,as evidenced by the lack of consensus among experts and algorithms on defining unique neural signals. Additionally, the specific challenges of single-channel, high-noise data highlight a gap in the capacity of existing methodologies to cluster independent waveforms. The process of spike-sorting, often treated as a monolithic solution through the construction of single pipelines, needs to be deconstructed into its component parts. The components for spike detection, feature extraction, and clustering should be interchangeable to enable independent optimization and a better understanding of the effectiveness of deep-learning / classical methods. Lastly, the limited application deep learning to spike-sorting represents untapped potential. Given the successes of machine learning in vision it makes sense to explore neural network-based approaches that reduce spike-sorting to a one-step unsupervised algorithm.

Over the next 4 chapters we execute a plan to tackle the three legs of our problem. The plan is as follows

Chapter 2 Create a standardized sorting problem that can be tested

Chapter 3 Solve the problem with the best classical methods

Chapter 4 Create a deep-learning solution to solve the problem

Chapter 5 Compare classical vs deep-learning solutions.

Chapter 2

Simulation

As previously mentioned, there is a lack of labelled spike datasets. Particularly, none were able to be found that have the following qualities: low signal-noise ratio, univariate waveforms, large sample size, more than 5 clusters. Due to this absence, there was a need to simulate waveforms with properties from the real data we have access to.

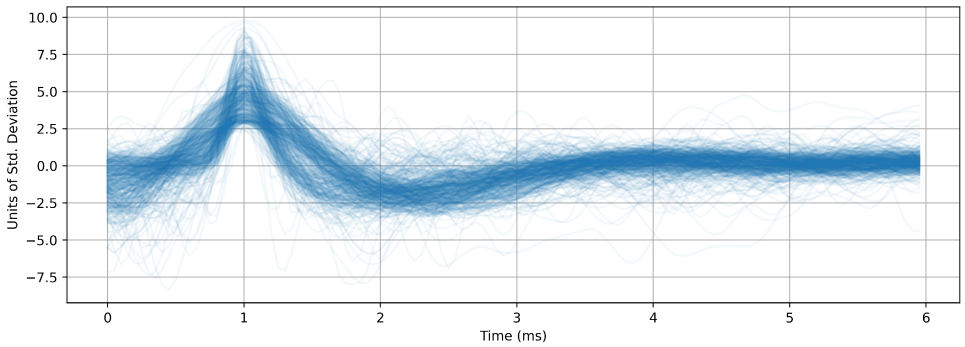


Figure 2.1: 500 randomly sampled real spikes superimposed

The goal of simulation will be to produce a similar dataset of waveforms with known labels. An idealistic simulation will replicate properties of real data. For our investigation we have taken a subset of 10,000 positive spikes. The spikes are visualized in Figure 2.1 and represent those with relatively less noise. The spikes have been processed though a Gaussian filter for whitening and those spikes with high ringing artifact (the subsequent voltages after the spike exceed 60% of the maximum) were removed. The spike waveforms are 6ms long composing of a 120pt array with the peak centred at 1ms - equivalently the 21st data point. The units on the Y-axis of Figure 2.1 are in std. of the recording. The subset does not include abnormally large signals (peak greater than10 standard deviations) as: 1. they are rare [28] , and 2.

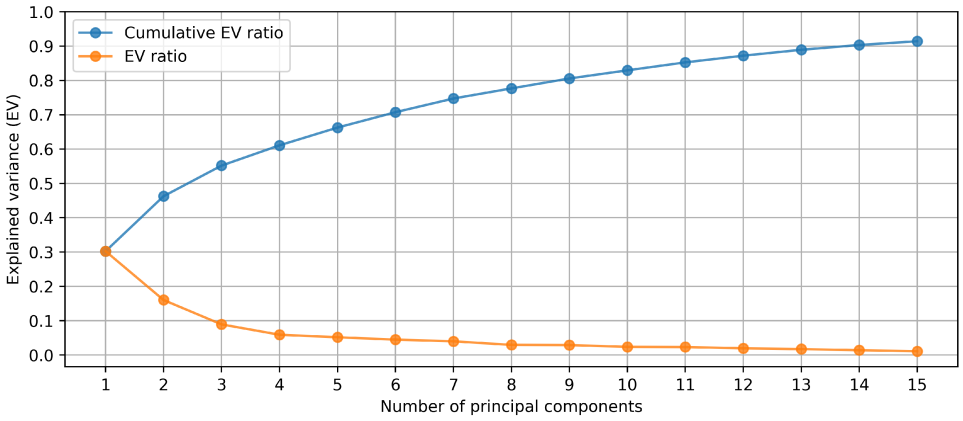


Figure 2.2: Explained Variance ratio of PCA components from Real Data

they form clusters that are easily found and represent a trivial problem.

2.1 PCA Difficulties

PCA aims to construct orthogonal vectors (PCs) that account for the majority of variance in the dataset in a least squares sense. The contribution of each component to the variance is known. As is shown in Figure 2.2 it is found that more than 10 components are needed to account for 90% of the variance in our subset of 10,000 spikes. This suggests PCA needs at least PCs to capture the essence of the data and this will cause trouble for the clustering algorithms referenced above that have been found to work best in fewer than 10 dimensions. Data embedded in low dimensions below 10 dimensions have stable distance calculations but less ability to sort spikes due to information lostin the embedding step. Alternatively, if we embed into higher dimensions exceeding and much greater than 10 dimensions distance calculations needed for clustering fail to separate spikes - in high dimensions points approximate a sphere.. These difficulties are experienced with PCA in the simulated data.

The difficulty PCA faces suggests that it cannot find a linear mapping to separate data. Upon this conclusion we look to the average of power spectrum's of the signals. In similar fashion we look at the ratio of the total power spectrum for each frequency. We find that the lowest frequencies make up the majority of the typical signal. Furthermore, we can compare the cumulative power ratio to the cumulative

explained variance ratios and find that they're near identical. We find a slight gap in the ratio lines which can be accounted for by the combination of the first two frequencies - shifting the orange line 1 unit left. Otherwise, the power of number of frequencies kept roughly approximates the variance explained by the same number of PCs. Overall this is concerning regarding the prevalence of PCA in spike sorting - we are not accessing significantly more information than using a standard Fourier Transform. Ideally our simulated data should not be separable by PCA and should have a similar power spectrum.

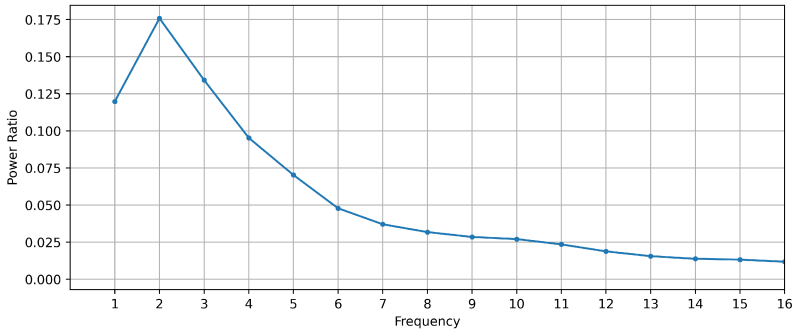


Figure 2.3: Average Power Ratio for Real Data

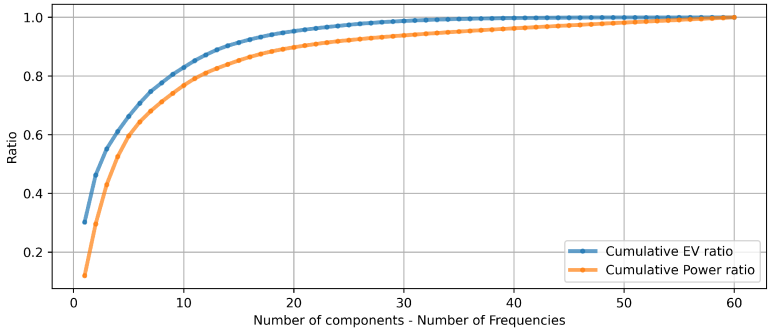


Figure 2.4: Cumulative Power and Cumulative EV ratios of real spike data

We find that the more modern dimensionality reduction technique [18] the Uniform Manifold Approximate Projection (UMAP) fairs a better at parsing spikes. This method doesn't have the interpretability of PCA but leads to improved performance in high-dimensional non-linear data. It's worth mentioning that t-distributed Stochastic Neighbor Embedding (t-SNE) [29] is a competitor for UMAP amongst modern

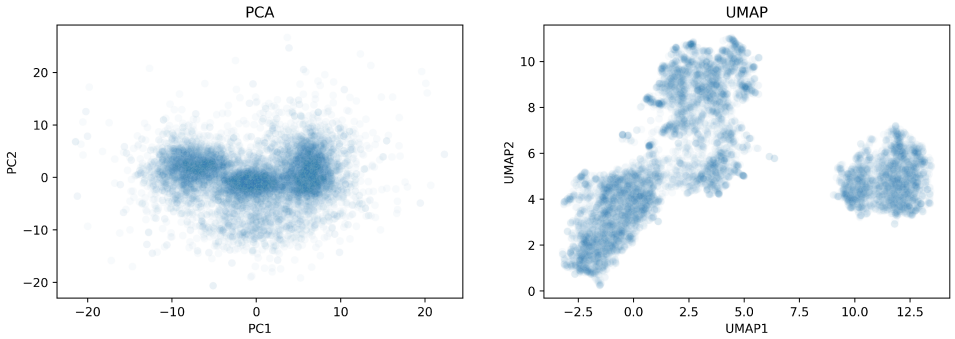


Figure 2.5: The embeddings of subset 10,000 spikes through PCA and UMAP into two dimensions. The separation between points in the UMAP embedding is much more definitive than PCA - in both cases three groupings but the PCA is arguably a single cluster.

dimensionality reduction techniques however it is more computationally expensive so we use UMAP. We will use UMAP to guide our simulation of data, and we will see in the bench marking chapter that the visual intuition of Figure 2.5, better parsing leads to much better clustering results for spike sorting.

2.2 Templates and Perturbations

To create labelled data we require there to be representative waveforms of each class. Rather than designing them we can simply sample K template waveforms from the real dataset. We insure that the template waveforms are sufficiently unique by running the K-Means algorithm on the UMAP embedding to parse data into K classes. Then from each class select a random subset of 5[1] signals and form their mean. In Figure 2.6 the sampling process is depicted. Note while $K = 8$ is chosen here it is a parameter used for the simulation and is unknown to any clustering model unless otherwise stated.

The next step is to introduce perturbations or transforms that create clusters of data around the templates. When analyzing UMAP embeddings of many real spike datasets a common structure is noted - there are banana/ellipse like densities forming in UMAP space. While UMAP dimensions aren't interpretable, these structures can

[1]Taking the mean of the entire class introduces a signal with zero high-frequncy elements, whereas a small subset offers a unique signal without the chance for a poor random sample. The specific number 5 could be any small integer.

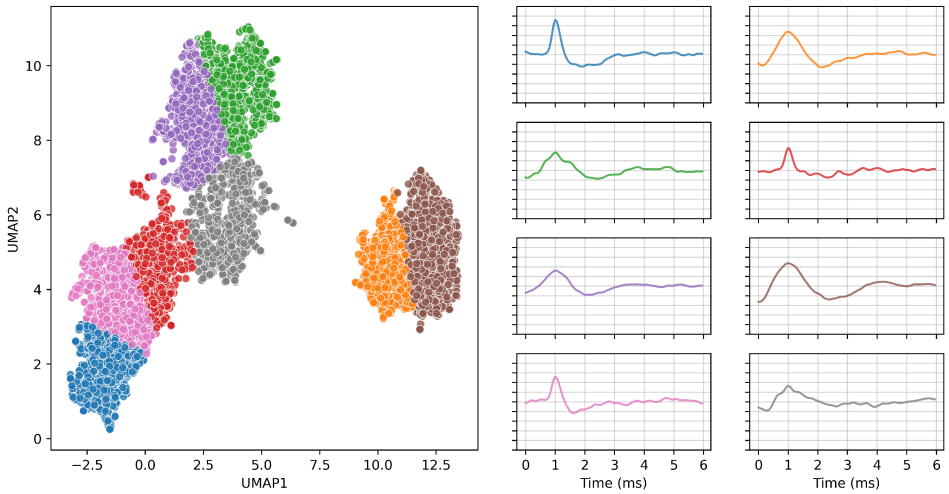


Figure 2.6: Left: K-means K=9 clustering on 2-dimensional UMAP embedding of 10,000 spike subset. Right: Representative waveforms generated from sampling the clustering

be plotted and shown to be low frequency amplifications of the same signal. These are in Figure 2.7 where the entire set of spikes is plotted (not just our clean subset used for template creation) [2]. In other words, the core morphology is being stretched (scaled). We find that if spikes along these banana structures are normalized to have a unit maximum, they all regard a similar morphology before the refractory period (the latter 60 data-points). In the ellipses shown in Figure 2.7 the maximum of the original spike only slightly correlated (¡0.2 correlation) with the amount of variance in the refractory period defined by the latter half of the signal. To us this suggests that the variance in refractory periods is generally caused by background noises.

The blurring of the ellipses tends to form from different background noises as shown in Figure 2.8. The fuzzy low-density areas around the ellipses are often signals of the same low-frequency components with different high-frequency behaviour. Thus if an envelope is drawn around the spikes using means and variances, the means between the two would be quite similar but the variance of the surroundings would be slightly higher. This leads us to suggest that the surrounding low densities are high levels of

[2]The embedding in 2.7 doesn't look like that in Figure 2.6 because they're separate UMAP calculations. The former is the entire dataset including many noisy, large, ringing spikes. The embedding is significantly blurring due to this.

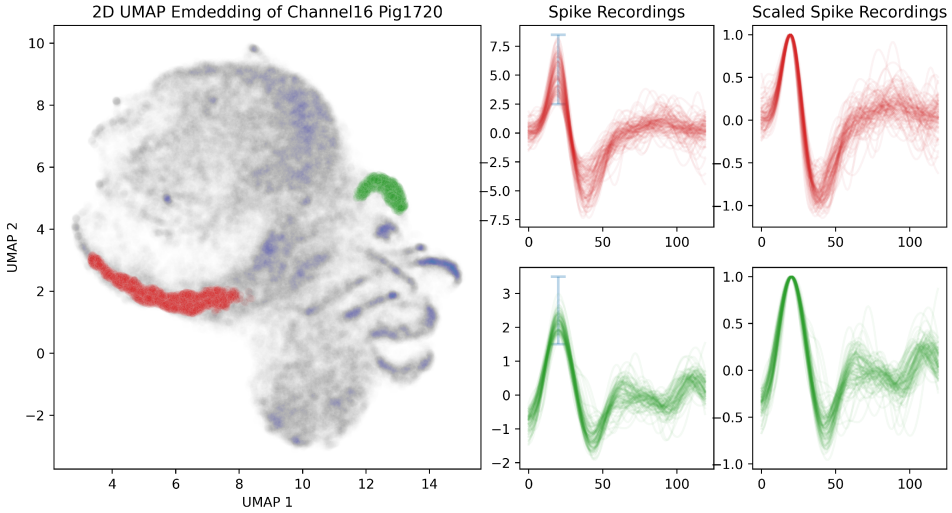


Figure 2.7: Ellipses Structures within UMAP Projections

background noise interfering with a template.

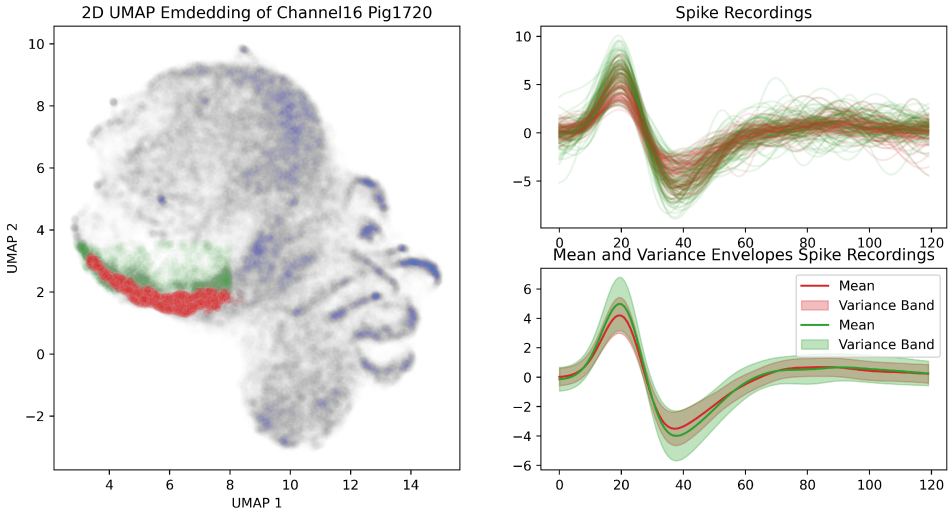


Figure 2.8: Left: UMAP embedding of entire channel dataset with an ellipse of high-density highlight in red, and surrounding low-density in green. Top Right: Sampled spikes from the respective colours on the left. Bottom Right: Mean and Variance of the red & green sets.

We found two processes that appear to be transforming our data: a low frequency scaling and background-noise. The correlated noise appearing within signals is difficult to simulate due to a non-stationary behaviour due to the incision of the probe

causing injury which leads to an injury response [25]. In our simulation we will not be inspecting a temporal aspect - that is to say each simulated spike does not have an associated timestamp. Therefore, we can use sampled noises from the real channel as one large set of noises we refer to as σ. Since the spike is a voltage recording the background noise be added with no loss of realism. For generating spikes via the templates we use two steps to generate simulated data. In the first step we take the template T and 'stretch' it by performing a convolution with a scaled[3] random walk. This process is performed using a weighted rolling average where the weights are taken from a Brownian motion process. Boundary effects are removed from the convolution by extending the template past the edges of the template 120 points after which the output waveform W is down-sampled to the original 120 samples. The second step sums n samples of background noise and adds it to W then applies a Gaussian filter [4] . This process is illustrated in Figure 2.9 where using UMAP embeddings. Pseudo-code implementation is outlined by the algorithm on the following page. Finally, all spikes where the peak has moved due to noise are removed for simplicity and to ensure uniformity across classes.

[3]When performing a convolution the two arrays are multiplied and summed leading to possibly very large changes in the order of magnitude between the input and output. The random walk is scaled such that the expected sum is 1 with all positive values.

[4]The slight filtering removes redundant white noise from summing multiple signals

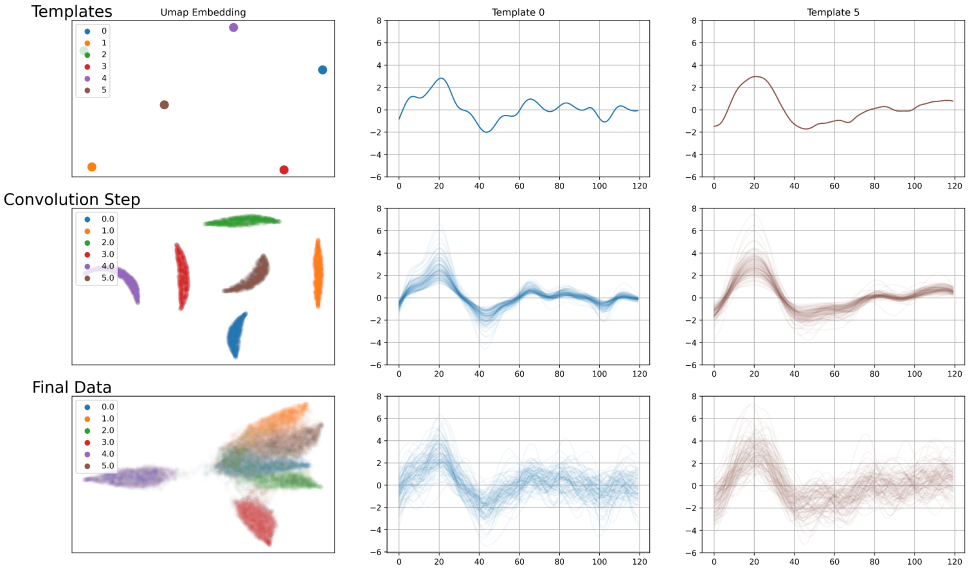


Figure 2.9: K=6 Simulation Example showing each step. Left: Umap embeddings. Middle and Right: Starting at with the templates showing the convolutional step then addition of noise.

Table 2.1: Parameter settings across 4 simulation runs

Parameter	**Effect**	**Data1**	**Data2**	**Data3**	**Data4**
K	Number of clusters	7	9	11	13
b_s	Variance of Random Walk	0.1	0.2	0.05	0.05
b_l	Length of Random walk	8	3	10	15
c_n	Noise prior to convol.	0.1	0.5	0.0	0.3
n_{noises}	Number of noises post convol.	3	3	3	2
	Number of Datapoints	16,761	29,012	22,417	21,776

We generate 4 datasets with varying simulation designs as outlined in Table 2.1. The UMAP visualization in Figure 2.11 shows some of the clusters overlap significantly - suggesting this will be a difficult clustering problem.

```
#given a list 'templates' size [K,120]
#given an extracted list of noises sigma [j,120], j = 100,000 noises
X = []
Y = []
num_spikes = 5000 #number of spikes per template
cn_scale = 0.1 #correlated noise temperature factor
bn_len = 8 #brown noise length
bn_scale = 0.1 #brown noise temperature factor
num_noises_added = 3

for i,template in enumerate(templates):
    #set of waveforms with label i
    x_i = []
    for _ in range(num_spikes):
        b_noise = np.cumsum(np.random.normal(0,1,bn_len)*bn_scale)
        #exponentiate and normalize
        exp_bn = np.exp(b_noise)/bn_length

        #add some noise prior to convolution step
        W = template.copy() + np.random.choice(sigma)*cn_scale

        #convolutional step

        #W goes from length 120 to 120+2*bn_len-1=135 here
        W = np.convolve(W,exp_bn,mode=full)
        peak_idx = np.argmax(W)
        W = W[peak-20:peak+100]
        if len(W)<120:
            #discard sample
            continue
        else:
            #add on noises and add W to x_i
            background = np.random.choice(sigma,num_noises_added)
            W += background.sum(axis=0)
            x_i.append(W)

    #add x_i and labels to X,Y
    X.append(x_i)
    Y.append(np.ones(len(x_i)*i))
```

Figure 2.10: Python Numpy Simulation Code for Generating Waveforms

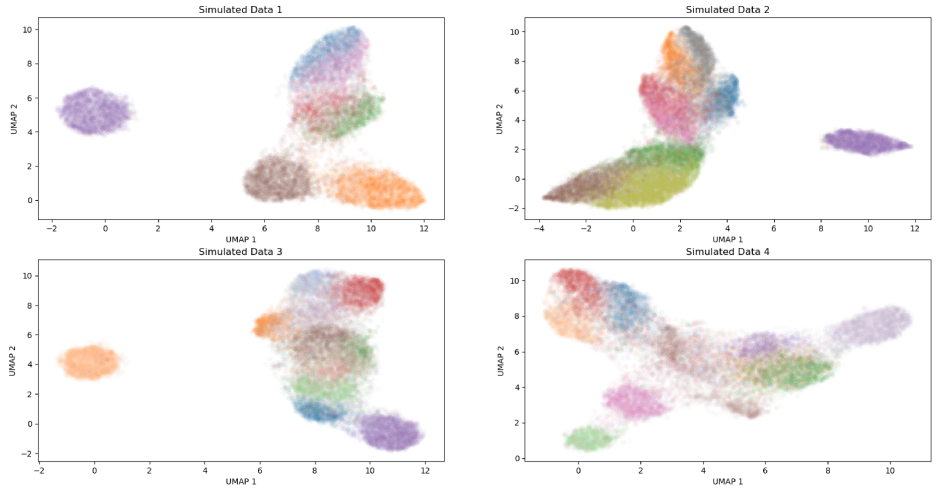


Figure 2.11: The 4 simulated datasets visualized in 2-dimensional UMAP

Chapter 3

Classical Models

As previously mentioned many spike sorters are not currently packaged to solve our problem of sorting individual spikes. We have seen that this problem is approached as a FE+CA pairing - these pairings numerous heuristics and associated parameter settings. Since the time required to reconfigure all known spike sorting methods would prove extensively long we will tested the general FE+CA pairings on our simulated data.

We will use feature extractors PCA and UMAP in 2-6 dimensions. UMAP has several parameters where the parameterization deals with the need to establish a balance between a local and global representation. In the unsupervised regime without context setting the these parameters is arbitrary but their typical values are known and these do not drastically effect results. Therefore for simplicity we will use the default UMAP parameters. For models, we inspect 3 levels: non-parametric, density, and centroid models. We expect performance to increase in the same order. The first models given their name, require no human input. The second category will be density / hierarchy based where the number of clusters has an unknown dependence on the parameterization[1]. Finally the last batch of models used will centroid models such as K-Means where the number of clusters is known beforehand. The number of clusters is an unknown piece of information and we expect this information will allow these methods to perform on par with the best parameterization of density models.

We provide a brief description of each clustering algorithm to understand the assumptions used in the construction of taken in the design of clustering techniques. Feature extraction and clustering models were deployed through the *scikit-learn* python library [26].

[1]It is due to this emergence that density models are commonly used in spike sorting

3.1 Normalized Mutual Information

$$\mathrm{NMI}(U,V) = \frac{2I(U;V)}{H(U)+H(V)} \tag{3.1}$$

$$I(U;V) = \sum_{u\in U}\sum_{v\in V} P_{U,V}(u,v)\log\left(\frac{P_{U,V}(u,v)}{P_U(u)P_V(v)}\right) \tag{3.2}$$

$$H(X) = -\sum_{i=1}^{n} P(x_i)\log P(x_i) \tag{3.3}$$

where U and V are clusterings of the same data. $I(U;V)$ is the mutual information between U and V, and H is the Shannon entropy. U and V are the respective prediction and true sets. Mutual information can be written as the entropy of one variable minus the conditional entropy with respect to the other variable. $I(U;V) = H(U) - H(U|V)$. It is helpful to think of NMI in terms of the conditional entropy $H(U|V)$ term. If knowledge of one set increases the chance of randomly guessing the other $H(U|V) < H(U)$ then there is mutual information. Consider the prediction of rain and whether you get your shoes wet on a given day. If someone predicts rain tomorrow you have gained information regarding the probability of getting your shoes wet tomorrow; thus, the two distributions for rain and wet shoes carry mutual information. The reverse may be said about the rain conditions and your birth date. When we use NMI for evaluation we are seeing how much information we gain about the truth from our answers.[2]

NMI values are bounded between 0 to 1 with 0 being random association, 1 being a perfect correlation. The purpose of using NMI as a metric is that it is robust to predicting the incorrect number of clusters whereas in most metrics there is a heavy penalty paid for incorrect number of clusters. Regular metrics such as recall or accuracy require a defined mapping criterion between the prediction set and test set. Since labels do not exist in our unsupervised problem there is no way to preform this mapping especially in the case when number of clusters predicted is different. NMI

[2]Alternatively, we can think of mutual information as a measurement of the dependence between variables. In our case we want our prediction to be very dependant on the answer.

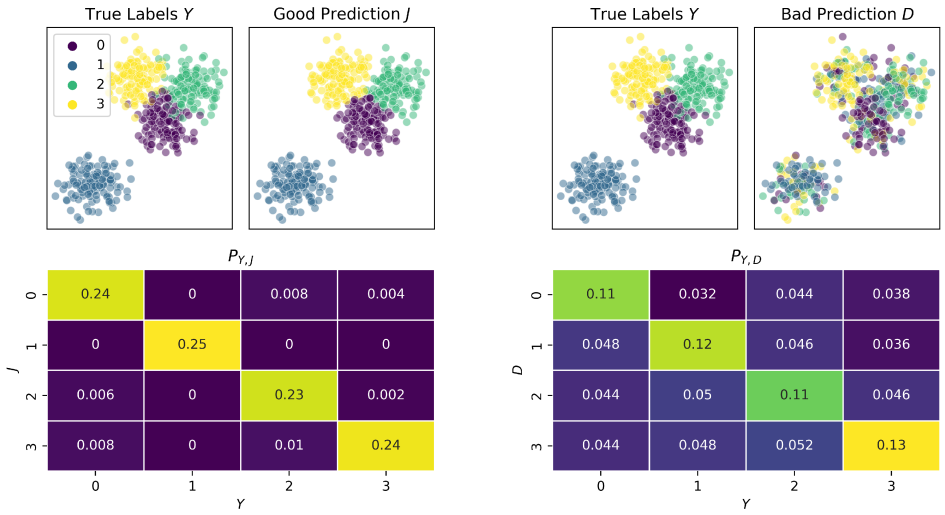


Figure 3.1: The left columns show a good clustering, on the right a semi-random one. The bottom heat maps visualize the joint distributions between the true and predicted sets. The left prediction scores NMI=0.885, the right NMI=0.087

also allows for the use-case of a noise class. Some clustering methods include a rejection criterion for data to belong to no class. This leads to an uncertain measurement of performance by standard metrics. In the event of NMI, the noise label is simply accounted for entropy and does not cause any issue. In the event we have a mapping between labels as in the example Figure 3.1 we find that the joint distribution for mutual sets centre around the diagonal given reasonable answers.

3.2 Benchmarks

3.2.1 Non-parametric Models

Often the term *non-parametric* can be attributed to models which don't specify the number of clusters, however this is a loose assumption as there may still be many parameters to be tuned. We impose a strict definition that the model must cluster the data without user specifications - that is to say, the clustering is only performed based on model heuristics. Two such models may be tested under this modelling restriction: Gaussian Mixture Models (GMM) paired with Bayesian Information Criterion (BIC) selection [1, 26], and the previously mentioned Isosplit [16, 5].

Gaussian Mixture Model + Bayesian Information Criterion

A GMM fits K multivariate Gaussian distributions to the data through likelihood maximization. Likelihood is defined as $\mathcal{L}(\Theta; X) = \prod_{i=1}^{n} \sum_{k=1}^{K} \pi_k P(x_i|\mu_k, \sigma_k)$ where n is the number of data points, K is the number of Gaussian's, π_k is the respective mixing coefficient, and $P(x_i|\mu_k, \sigma_k)$ is the probability density of the ith point under the kth distribution. The BIC is defined as $BIC = k \ln n - 2 \ln(\mathcal{L})$, where k is the number of parameters in the model. To remove the need to specify the number of clusters K we fit a GMM for all reasonable possibilities $K \epsilon [1..N]$ and select the model which maximizes the BIC. The process is computationally inexpensive, therefore, N can be selected to be large making the method essentially non-parametric.

ISOSPLIT

IsoSplit proposes to be a non-parametric method which searches for density peaks; furthermore it assumes that the peaks in density distributions are unimodal such that there can be a margin drawn between neighbouring peaks. By performing tests for uni-modality Isosplit removes need for distributional requirements - such as specifically a Gaussian - however also cannot account for non-convex distributions - such as the banana shaped distributions in UMAP we previously mentioned. Through finding uni-modal peaks, margins are drawn forming a clustered space. To implement we used the Python conversion found here [12].

Results

We paired these models with UMAP and PCA done in 2-5 dimensions. Results reported in the tables are the best found across all runs. All models fared better with UMAP than PCA - confirming our prior belief that PCA feature extraction is relatively less effective than UMAP. Across all datasets the BIC+GMM model outperforms ISOSPLIT.

Table 3.1: Non Parametric Model Performances Across Datsets

	Dataset1	Dataset2	Dataset3	Dataset4
GMM+BIC	0.585	0.548	0.595	0.585
ISOSPLIT	0.503	0.520	0.497	0.479

3.2.2 Density Models

In the previous test, the best average results came from using UMAP in 4 dimensions therefore that is what we use as feature extraction for the following clustering models.

Density Based Spatial Clustering of Applications with Noise (DBSCAN)

DBSCAN operates by linking all points that are within a certain radius together and if sufficient points are connected they form a cluster [6, 26]. For a dataset of length n consider a complete weighted graph G[3] where all edges have weights equal to the distance between the two vertices. DBSCAN performs an edge cut to G by removing all edges which have weights less than ϵ. G now has $1 \leq m \leq n$ components. Of these components, those larger than the minimum required cluster size are kept and define a cluster. Any remaining points are labelled as noise. As ϵ decreases components either split or diminish in size. The minimum cluster size is rather straightforward thus we set ours to min_cluster = 100. The main parameter to select is ϵ. An optimal ϵ assumes all clusters will have a minimum local density: ϵ too large and all points will be connected, too small and all points will be rejected to noise. The order of magnitude of ϵ can be inferred through average nearest neighbour distances. Varying ϵ across a reasonable range we find performance is unimodal. The sharp step-function-like changes are due to clusters being split or rejected. NMI treats the noise category as a randomized class. Results can slightly be improved by associating all points to their nearest cluster; however this gets away from the main theme of DBSCAN, and for our datasets won't change reported results much as less than 3% of data is labelled as noise in the optimal models.

Table 3.2: DBSCAN Performances

	Dataset1	Dataset2	Dataset3	Dataset4
Top 10 Mean	0.690	0.499	0.543	0.582
Best	0.704	0.543	0.582	0.563

[3]Complete Graph: All points are connected to all other points.
Weighted Graph: All connections between points have an associated weigth value.

Hierarchical DBSCAN

HDBSCAN is a modern update of DBSCAN [17]. In similar fashion HDBSCAN constructs a complete weighted graph G. DBSCAN allows for long chains of points to be connected to clusters even when their local density is very low. To ameliorate this HDBSCAN changes the weights of the graph G from distance to *mutual reachability distance*. For two points x_1, x_2 this is defined as $max\{d(x_1, x_2), c(x_1, k), c(x_2, k)\}$ where d is distance and c is *core distance* defined as the minimum distance to the k'th nearest point. What this does is max very close points but non-dense points appear far away on the graph. The k parameter is referred to as *min samples*. Now rather than performing an edge cut to G based on a parameter, HDBSCAN constructs a dendrogram[4] and finds the optimal weight to perform an edge cut through a criterion of which branches have the highest persistence. Finally like DBSCAN, the remaining components of G which meet a minimum cluster size form unique clusters and all other points are labelled as noise.

We perform tests on HDBSCAN by varying the *min samples* parameter which in effect controls spherical nature of the cluster forming. Similarly we set minimum cluster size to 100. We find large instability in performance across the first and fourth dataset - sharp changes in performance with single integer changes to *min samples* as can be seen in Figure 3.4. For datasets two and three there are individual clusters significantly far from the rest of the data that cause the selection criteria in HDBSCAN to miss all other clusters. Hopefully this could be amended by parsing the dataset, however even when we use offline cluster separation[5] HDBSCAN the results were still lackluster. The selection criterion for HDBSCAN makes it an infeasible choice for our problem and highlights a critical issue with hierarchical clustering: the variance across datasets and embeddings likely makes the criterion unstable.

[4]Dendrogram: A tree-like diagram displaying the hierarchy of the graph G with respect to density. At low density is the trunk and as density requirements are increase the branches split/shrink.

[5]Offline refers to a human looking at the dendrogram and splitting the dataset to remove obvious clusters.

Mean Shift

Mean Shift [7, 26] locates maxima of a density function to identify cluster centers without requiring a predefined number of clusters, distributions or distances. Initially a spherical window is drawn around all n datapoints creating n centriods. The centriods are shifted towards the mean of their window upon every iterate. If the mean is at the centre of the centriod it has converged. The algorithm executes until all centroids have followed their respective density gradients to convergence, upon which most centroids have overlapped. The remaining centroids define cluster centres. All data is labelled by proximity to cluster centriods. The crucial parameter required is the radius of the window, labelled *bandwidth*. Bandwidth influences cluster granularity. A smaller bandwidth leads to many small clusters, while a larger one may merge distinct clusters into larger ones. The choice of bandwidth is critical for balancing between over- and under-clustering but performance generally smooth as there's no rejection criterion.

We find this to be the most consistent density-based method for our datasets in low dimensions. There's a large range for bandwidth that performs consistently well across all datasets as can be seen in Figure 3.2. Once a certain bandwidth is reached, the model behaves consistently across datasets - a desirable quality in unsupervised problems. The Density Peaks (DP) clustering algorithm [23] discovers clusters

Table 3.3: Mean Shift Performances

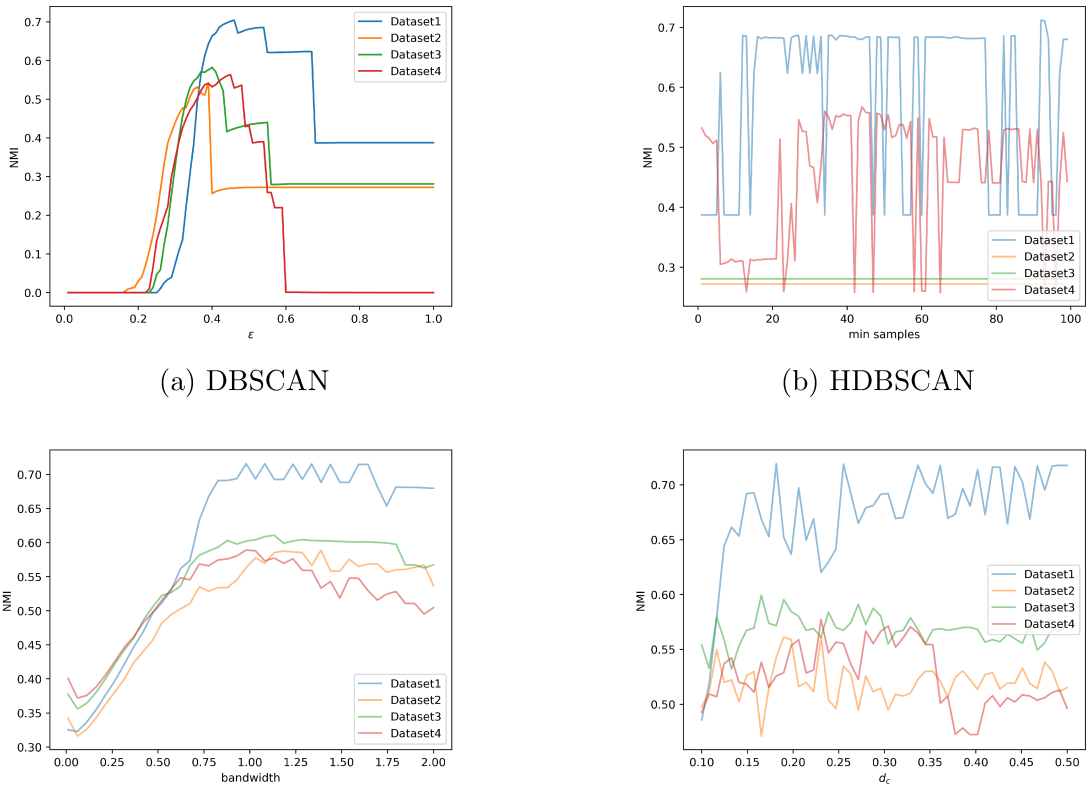
	Dataset1	Dataset2	Dataset3	Dataset4
Top 10 Mean	0.709	0.579	0.604	0.577
Best	0.716	0.589	0.610	0.589

through local density maxima and their separation from other maximums. DP begins by allocating an estimate of the local density ρ to each data point - an estimate drawn by counting the number of neighbouring data point within a radius d_c. Then all datapoints are given a distance measurement δ that is the distance to the closest point of greater ρ. For the point of greatest ρ, δ is set to the greatest distance between this point and any other; furthermore these two values are used to scale all others so that they all lie between $[0, 1]$. When plotting a graph of δ vs ρ the majority of data will sit along the ρ axis as there is very little distance to the next point of higher

density. Outliers will have higher δ and represent "peaks" in the density function. Given a threshold of ρ_{min} and δ_{min} we select these outliers as cluster centres. For our adaptation we'll assign remaining points to the nearest cluster centre. In our inspection we set $\rho_{min} = 0.5$, $\delta_{min} = 0.2$ and vary the d_c parameter for our inspection. The two choices for bounds we set through manual inspection of the δ vs ρ graph. We find that while peak performance is fairly good; however, the performance is not smooth with respect to d_c. Given our UMAP projections reasonable values for d_c lie within $[0.1, 0.5]$ and we see even a change of 0.01 in dc can change NMI results by up to 0.08. Although best performance of Density Peaks is higher than that of Mean Shift the notable gaps in performance are concerning given there isn't a self supervised method to detect the drop off in performance.

Table 3.4: Density Peak Performances

	Dataset1	Dataset2	Dataset3	Dataset4
Top 10 Mean	0.709	0.536	0.579	0.556
Best	0.719	0.562	0.599	0.577



(a) DBSCAN

(b) HDBSCAN

(c) Mean Shift

(d) Density Peaks

Figure 3.2: Density Models Performance vs Key Parameter

HDBSCAN's instability makes it unusable for spike sorting. DBSCAN performs decently well; however, the choice of ϵ varies across datasets. Density Peaks and Mean Shift have comparable best-case results, with the latter being smoother. Furthermore, Mean Shift contains a single parameter whereas we had to set 2 external parameters for Density Peaks. Given the simplicity of the Mean Shift algorithm, it's surprising to see that it's not more popular amongst spike sorters in low dimensions,and that working with Density Peaks has been more prevalent.

3.2.3 Specified-K Models

Turning away from density we look into models where the number of clusters is given. For all models we used UMAP and PCA for feature extraction in 2-6 dimensions and

selected the best results. We test 3 popular models: K-Means, GMM, Agglomorative Clustering (AggC). The first two construct clusters through the placement of centriods. K-means fits through an iterative movements of centriods by minimizing within-cluster sum of squares and the GMM as previously explained by fitting Gaussian centriods. AggC is a hierarchical approach. AggC constructs a graph through distances and forms clusters based on a linkage criterion; thereafter, a linkage criterion starts every point as cluster and constructs a dendrogram through merges. Once the dendrogram results in K remaining clusters the clustering is complete. GMM and AggC both contain four heuristic settings. For a GMM the covariance matrix used can be computed as: full, tied, diagonal, spherical. Full allows for elliptical clusters in any orientation by having a unique convariance matrix per component. Tied forces all clusters to share a common covariance matrix ensuring all clusters share a common shape. Diagonal creates clusters with dimension independent variances. Spherical enforces simplification to a single variance in all directions. For AggC the linkage criterion are: average, single, complete, Ward. Average linkage links clusters through the average intra-cluster distances and sits in the middle of single and complete which respectively use minimum and maximum intra-cluster distances. Ward's method minimizes variance within clusters by merging clusters which lead to the minimum increase in total intra-cluster variance. All 8 of theses models will be tested using all FE methods.

As expected we find that given K model perform about as well as properly parameterized density models. Performance is roughly consistent independent of model selection.

Table 3.5: Centroid Model Performances with best FE Across Datasets

Clustering Algorithm	**Feature Extraction**	**Heuristic Choice**	**NMI Score**
DataSet 1 K = 7			
K-Means	$UMAP_3$		0.671
GMM	$UMAP_4$	CovType = 'full'	0.691
AggC	$UMAP_4$	Linkage='average'	**0.693**
DataSet 2 K = 9			
K-Means	$UMAP_3$		0.562
GMM	$UMAP_5$	CovType = 'full'	**0.597**
AggC	$UMAP_6$	Linkage='average'	0.577
DataSet 3 K = 11			
K-Means	$UMAP_3$		0.611
GMM	$UMAP_5$	CovType = 'tied'	**0.623**
AggC	$UMAP_6$	Linkage = 'single'	0.610
DataSet 4 K = 13			
K-Means	$UMAP_2$		0.570
GMM	$UMAP_6$	CovType = 'tied'	**0.598**
AggC	$UMAP_6$	Linkage= 'average'	0.574

3.3 Summary

In this section we benchmarked clustering models across our datasets using NMI. We aimed to provide brief insight on how classical clustering is approached. One could affirm certain approaches are superior to others in our datasets, but that is a byproduct of these tests. The purpose of these tests was to find the best classical unsupervised model by exhausting methodologies and parameter spaces. Although the probability of selecting the optimal model and parameterization for each dataset is theoretically low we will use the best results as a goal-post for our deep-learning approach proposed in the next chapter.

Table 3.6: Benchmark Performance across Datasets and Clustering Types

	Dataset1	Dataset2	Dataset3	Dataset4
Non-Parametric	0.585	0.548	0.595	0.585
Density	0.719	0.589	0.610	0.589
Centriod	0.691	0.597	0.623	0.598

Chapter 4

Invariant Information Spike Sorting

In order to develop a deep learning clustering approach we must look at methods built for comparable problems outside of spike sorting due to the lack of development within our field. We assert that the popular MNIST dataset [14] of handwritten digits is a comparable problem to ours. The dataset contains 70,000 grey scale 28x28 images with 10 classes. The dimensions of this problem mimic ours for single waveform sorting. The current state of the art (SOTA) architecture for solving MNIST is Invariant Information Clustering IIC [10] thus, it became our adopted base solution. Before diving in to methodology it is worth noting that IIC is not considered a SOTA on more complex image datasets which include multiple colour channels and higher pixel count such as CIFAR10 [13]. Methods that outperform IIC on unsupervised CIFAR10 [30, 21] tackle learning representation separate from clustering whereas IIC performs the two jointly. [30] Specifically critiques IIC's stability w.r.t to colouring of images caused by not treating learning representation separate from clustering. While noted, IIC performs the best on single colour channel which is analogous to our single recording channel problem. Similar stability issues might arise if we extend our method to multi-probe recordings but that is out of the domain of this thesis; therefore, we select IIC to build from.

4.1 Invariant Information Clustering

IIC is a technique that was developed to perform unsupervised clustering using neural networks. The method requires a neural network F designed to produce a probability mass function $P = F(x)$ across K classes. IIC trains a neural network to maximize the mutual information between cluster assignments of pairs of data. That is to say, given a pair of like-data $[x, x']$ which belongs to the same class, the network outputs $F(x), F(x')$ will have maximal mutual information. Any differences between samples

are considered redundant information; thus, by maximizing mutual information between the pair's distributions the clustering becomes robust to redundancies. The training of F uses negative mutual information as the loss function by the following steps:

$$
\begin{aligned}
(x_1, x_2) &= \text{Paired data sample} \\
P_1 &= F(x_1) \\
P_2 &= F(x_2) \\
P_{1,2} &= \text{Joint distribution of } P_1, P_2 \\
Loss &= \sum_{i=1}^{k}\sum_{j=1}^{k} P_{1,2}(i,j)\log\left(\frac{P_{1,2}(i,j)}{P_1(i)P_2(j)}\right)
\end{aligned}
$$

Backpropagation adjusting the weights in F

We can expand the Loss to show it in forms of Shannon Entropy.

$$Loss = \sum_{i=1}^{k}\sum_{j=1}^{k} P_{1,2}(i,j)\log\left(\frac{P_{1,2}(i,j)}{P_1(i)P_2(j)}\right)$$

Expand Logarithms

$$= \sum_{i=1}^{k}\sum_{j=1}^{k} P_{1,2}(i,j)\log(P_{1,2}(i,j)) - \sum_{i=1}^{k}\sum_{j=1}^{k} P_{1,2}(i,j)\log(P_1(i)) - \sum_{i=1}^{k}\sum_{j=1}^{k} P_{1,2}(i,j)\log(P_2(j))$$

Remove redundant sums

$$= \sum_{i=1}^{k}\sum_{j=1}^{k} P_{1,2}(i,j)\log(P_{1,2}(i,j)) - \sum_{i=1}^{k} P_1(i)\log(P_1(i)) - \sum_{j=1}^{k} P_2(j)\log(P_2(j))$$

Rearrange

$$= -\sum_{i=1}^{k} P_1(i)\log(P_1(i)) + \left[\sum_{i=1}^{k}\sum_{j=1}^{k} P_{1,2}(i,j)\log(P_{1,2}(i,j)) - \sum_{j=1}^{k} P_2(j)\log(P_2(j))\right]$$

$$= -\sum_{i=1}^{k} P_1(i)\log(P_1(i)) + \sum_{i=1}^{k}\sum_{j=1}^{k} P_{1,2}(i,j)\log\left(\frac{P_{1,2}(i,j)}{P_2(j)}\right)$$

Display as Entropies

$$= H(P_1|P_2) - H(P_1)$$

Thus when we minimize the loss, conditional entropy drives to zero while entropy is maximized. Two properties that arise are:

1 : The distribution of predicted classes is encouraged to be uniform. That is to say the average of all P is driven toward uniformity across the classes.

2 : Individual distributions such as P_1, P_2 move toward one-hot vectors.

These two properties interplay to cluster data. The first property says single predictions have relatively high randomness, and the latter suggests that if one distribution of the pair is known, the other is relatively less random. Without the first property a degenerate solution arises where the second property gets met immediately - by labelling every piece of data identically. Without the second property, the network learns nothing and knowing an item of the pair gains no predictive power. For a network defined to predict K classes, the first property will drive to find K groups and the second will try to ensure homogeneity across similar samples.

As the problem is operating in an unsupervised regime, the process of obtaining a like-pair must be constructed. Tackling the task of like-pairs has often be done in vision by generating augmentations of a single sample [27]. To get a pair from a single data sample x_1 we simply pass through a transform aimed at creating a facsimile $x_2 = \Phi(x_1)$. The transform is intentionally designed to preserve the semantics of the original data. Operations on images such as: cropping, mirroring, rotating, blurring and color-filtering all can preserve semantics of images. For example, a cat upside-down is still a cat. These operations introduce redundant noise that the network should learn to ignore. The authors of IIC used these transforms paired with mutual information loss to get a neural network to learn semantics of hand writing and report a 99% accuracy in labelling MNIST.

While IIC has been shown to work on MNIST, the paper mentions that the method should generalize to any problem where paired data is available. However, what is not investigated in the original work is that there is general accord as to what transformations images may undergo without class change: background context, rotation, slight blurring/cropping etc. This requisite heuristic allows for materialization of paired data absent of labels. Moreover, definable transforms convey what the model attempts to deem acceptable intra-class variance. Extending the IIC method past the domain of vision therefore reduces to capacity to discover justifiable transforms.

It is a premise of this thesis that spike transforms may be defined through data & physics-based knowledge to obtained paired data. Through the ability to define what should be the same margins should be able to be drawn between spike clusters.

4.2 Spike Transformations Φ

The spike sorting problem conceptually relies on there being a core template spike that is representative of its class. It assumed that in an experiment this template is a repeatable, undeviating voltage signature. It reality this assumes all intra-class variations are acceptable transformations of the core template and of each other. If transformations are definable, they can be used to perform IIC on the dataset. In the simulation a process was defined to use core templates to generate noisy data. Here we plan to apply a transform which perturbs the already noisy data to obtain a paired data sample. The proposition is that what is shared amongst the two samples is the template. For example, say our functions in the simulation define operator S and we have now defined transform operator Φ. For a simulation template x_k the paired data sample would be $[S(x_k), \Phi(S(x_k))]$ which are both transforms of x_k. The difficulty is to have transforms which introduce enough redundancy that our network F can 'unlearn' them. We must unlearn redundancy while preserving enough information that the fundamental template waveform is still present. We aim to find F which given $S(x_k), \Phi(S(x_k))$ acts as an inverse to S.

As the experiments measure a physical process the deviations themselves must be arising from physical / physiological considerations. Two concepts arise to cause transformations: the background electromagnetic noise and probe drifts altering magnitude and shapes of the signal. These mirror the two steps used to generate the datasets; however its important to note that they will not be a copy of the simulation. We propose that if one can approximately describe the causes of noise in the dataset IIC should work to label. For example, handwriting was accurately identified using transforms that included cropping and mirroring - two transforms that are largely extremes compared to usual variance in writing. If our proposition is correct, it would mean if Φ even roughly approximates the simulation S we can label our datasets. In doing so the problem is reversed from "*What makes two spikes acceptably different*"

to "*What makes two spikes the same*". The former is an open question, whereas it is our ansatz that the latter is based in physics.

Probe Drift, Morphology Change

A location issue that plagues repeatable identification is signal drift. Across the lifetime of the experiment, the location of the probe in space may move relative to the desired neurons for measurement. This movement can cause a non-trivial change in the waveform appearance. Certain portions of the waveform may be amplified or diminished depending on the angle & distance changes. To combat this change, a transform editing magnitudes and frequencies is appropriate. This transform will take the approach of applying perturbations to the low-frequency power spectrum of the waveforms. The low frequency end of the power spectrum is responsible for the majority of characteristics as we may attribute the high frequency components of the waveform to be caused by the sum of background noises. Such perturbations to the low-frequency spectrum will cause stretching - as was seen in the simulation. We propose to randomly amplify/diminish the j first Fourier coefficients by α. We do so by random, scaled, exponentiation of α by a uniform random distribution between $[-1, 1]$. Note that the expected power spectrum of this process is identical to the input, however there will be large variations between samples. This process should introduce robustness to low frequency changes.

Fourier transform the waveform

$$C = \mathcal{F}(x)$$

scale low frequency components

$$C_2[1:j] = C[1:j] * \frac{\alpha^{U(-1,1)}}{E[\alpha^{U(-1,1)}]}$$

Inverse Fourier Transform

$$\Phi_1(x, j, \alpha) = \mathcal{F}^{-1}(C_2) \tag{4.1}$$

[1]

[1]The division of the expectation of the process defined in the second step is so that the expected frequencies return are centered around the original. Take the example $\alpha = 2$ so the product term is $2^{U(-1,1)}$. This has a median of 2 but a mean above 1. This can be seen by using the equilikely bounds of the uniform distribution $1/2+2 = 1.25$.

Background Noise

Similar to our simulation we consider the background noise of the recording. Given we have a library of sampled noise σ, adding samples to waveform x can create a paired sample. Therefore we define the addition of n samples noise σ as the second operation. We ensure the set of noises are mean 0 so this process also has an expected mean equal to the input. The idea here is to build robustness to random recorded signal perturbations. Recall in the simulation that noise was already added to the generated waveforms. We propose adding *more* to the same spike in hopes that the differences between the two uncover the underlying distribution. of the template.

$$\Phi_2(x, n) = x + \sum_{i}^{i=n} \sigma \tag{4.2}$$

In the real world application, local noise could be sampled to generate pairs to provide temporal context; however, for our simulation set we will randomly sample the entire set as the spikes were generated without respect to time.

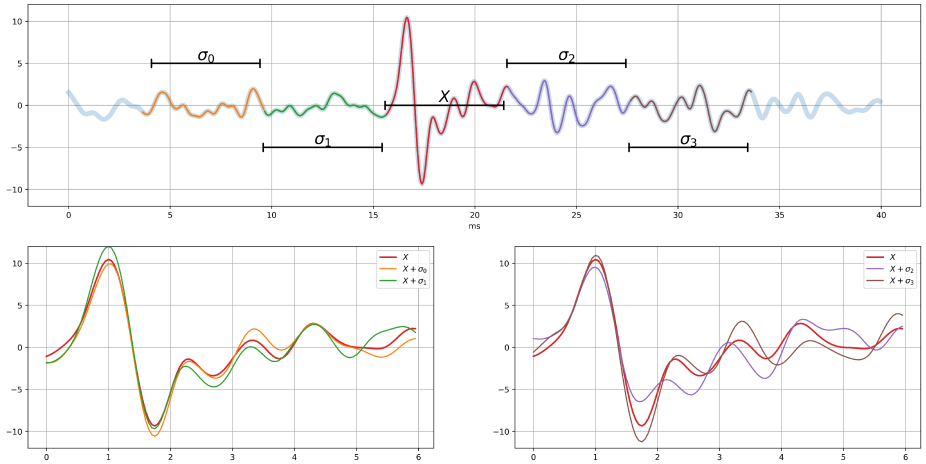


Figure 4.1: Creating paired data through addition of background noise. Top: 40 milliseconds of recording with one spike waveform present. The spike waveform along with 4 series of noise samples are outlined. Bottom Left: The waveform plus noises prior to the spike. Bottom Right: The waveform plus noises after the spike. All spike waveforms shown could be pairs w.r.t each other.

Summary

To apply IIC to spike-sorting problems theses transforms applied to the dataset will generate paired data. An outline of the training loop now looks like:

$$
\begin{aligned}
&x_1 = \text{Waveform} \\
&\Phi(j, \alpha, n) = \Phi_1(j, \alpha) + \Phi_2(n) \\
&x_2 = \Phi(x_1) \\
&F = \text{Neural Network} \\
&P_1 = F(x_1) \\
&P_2 = F(x_2) \\
&P_{1,2} = \text{Joint distribution of } P_1, P_2 \\
&Loss = -\sum_{i=1}^{k}\sum_{j=1}^{k} P(i,j) \log\left(\frac{P_{1,2}(i,j)}{P_1(i)P_2(j)}\right)
\end{aligned}
$$

4.3 Label Reconciliation

The length of softmax layer concluding clustering neural networks is a cardinal constraint of not only IIC but other contemporary deep clustering techniques as well [30, 24]. In these methods an optimal solution is achieved when all classes are realized. In the use of IIC, when a network introduces a new class - that is, using of a previously unoccupied index in the final layer - generally leads to a net negative step in the loss function. The reason being that introducing a new class greatly increases the total entropy of outputs while only marginally increasing conditional entropy. This can cause new classes to be introduced during training until each index is occupied. The length of the softmax becomes strong suggestion of number clusters possibly predicted. As this is an architectural choice made prior to any training, one can categorize IIC as belonging to the K-means style centroid clustering algorithms: where data is partitioned into K clusters. There have been works to amend deep clustering without known number of clusters [24, 32], however these novel techniques require further study, and have not shown stability or robustness w.r.t hyper-parameter selections.

A serious problem presents itself when applying IIC methodology to spike sorting: unknown K. Discovering how many unique spikes there are is as fundamental as properly labelling them. That said, there can be a known bound for what K cannot be. Depending on the probe and noise threshold, one can obtain an upper bound based on how many unique neurons could be within the area. For the our data application it's found that the upper limit can be reasonably set around 15 clusters [28]. Therefore in order to use IIC an extension must be made to find an approximation to K that is underneath some threshold K_{max}. IIC proposes an *over-clustering head* Figure 4.2, an alternative latter part of the neural network which concludes with a greater number of classes. In vision applications the network design consists of a CNN referred to as *the backbone* and two multi-layer perceptron (MLP) neural networks which conclude in softmax functions referred to as *heads*[2]. The main head had correct $K = 10$ for the MNIST digit problem whereas the secondary head had a much larger $K_{oc} = 50$ possible cluster assignments. During training the head used is periodically swapped per epoch. Both heads use the same backbone, thus the output of the backbone can be viewed as a high-dimensional feature space which MLPs attempt to cluster. The over-clustering head searches for more nuanced differences between data which get propagated through the backbone, adjusting the feature space to account for these differences. The lower dimensional main head learns within the same feature space tuning for the correct number of classes.

The over-clustering head was designed for the intent of *semi-supervised* learning: where data is partially labelled but there are swaths of data with unknown/noisy labels. With this in mind one can use the regular head for the portion of data that is labelled with logistic loss, while using the over clustering head uses IIC loss on the unlabelled data. The over-clustering head has capacity to parse many sub-categories within classes as it has access far greater than the true number of classes $K_{oc} > K$, potentially discovering separable intra-class categories. The over cluster thus allows the user to heavily influence the backbone through many discovered classes in unlabelled data while steering inter-network definitively in the right direction through the labelled data. When using the over clustering output for evaluation each true

[2]The term head will be used repeatedly from here on out. It is therefore useful to reinforce that this refers to a predictor. A head produces a prediction, multiple heads produce multiple predictions.

cluster may correspond to the union of multiple predicted clusters; therefore requiring a many-to-one mapping from $K_{oc} \rightarrow K$. In semi-supervised cases this mapping exists by using labelled data - simply pass the labelled data through the over clustering head and discover which classes should be merged. It was found that by designing a many-to-one map through the union of classes supervised level results can be achieved with a 75-90% reduction in labelled data .

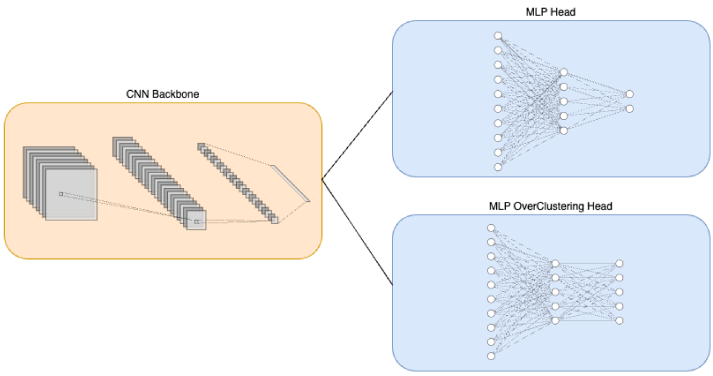


Figure 4.2: Original IIC Architecture composing of a convolutional backbone followed by a regular and overclustering head.

Another architectural design proposed to use with IIC addresses the randomness present in any neural network training due to random initialization of the weights and batching. It was found that using multiple heads simultaneously connected to the backbone bootstraps training and produces superior results. Given h heads the network produces h prediction distributions. Upon completion the head with the lowest loss is selected as optimal clustering - a heuristic that is found to be effective.

What really sparked our interest for IIC is that the semi-supervised results can be found *without* the supervised head - the network only is trained to over cluster, with no labels, and the mapping is found post-training using a subset of labels. This means the model weights have no context of labels but the output distribution is an informative representation that can be mapped to the true labels. In our application we want to discover this mapping without any labels. We propose to merge the idea

of over clustering and the ensemble of heads 4.3. Rather than selecting the individual head that achieves the lowest loss, we want to agglomerate the information learned by the heads, reconciling multiple labels to a single one. This reconciliation mapping will be found by identifying prevalent data labels that are ubiquitous across heads; that is, finding the largest groups of data that are in perfect agreement across heads. To visualize in space, this would mean finding the largest regions such that no head has drawn a margin through said region.

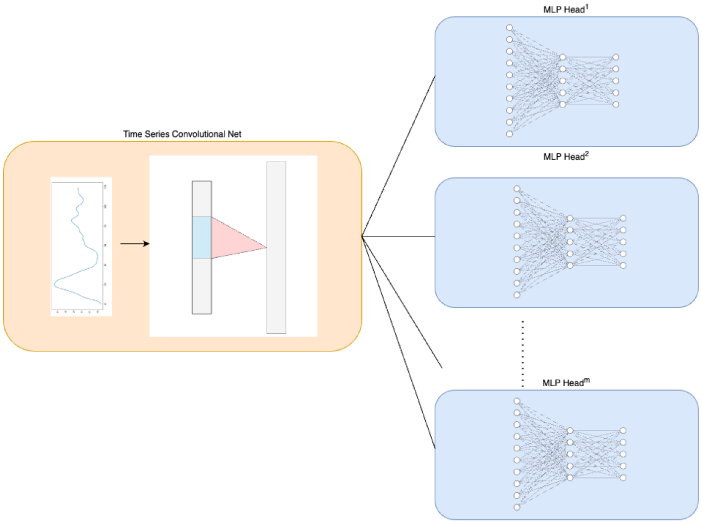


Figure 4.3: Proposed Neural Network Architecture to use for spike Sorting

To implement our proposal consider the same style of architecture with m heads all designed to over-cluster; furthermore all heads will have output length $K_{oc} = K_{max}$ equal to the previously mentioned appropriate upper bound. We train by computing m losses for m heads simultaneously. Upon completion of training there is a singular network which has m possible combinations of backbone-head that each predict data into across a maximum of K_{max} labels. Since mutual information loss strongly enforces singleton distributions taking the maximum location of each distribution preserves nearly all information; thus each head predicts an integer label. Consequently there is a array of labels $y = F(x)$ that is length m produced per input x_i seen in Figure 4.4. Its important to note that the labels from each head do not correspond - that is label *"1"* from one head has independent semantics from all other label 1s. To

reconcile the set of group labels $Y = F(x)\ \forall\ x\ \epsilon\ X$ we propose to find most common group label y_0 then remove all labels in the set which share any similarities with y_0 and repeat this process until the set is empty[3]. The label arrays found through this process will be defined as the cores c which form the set C. Corollaries of this process are: The number of cores is less than or equal to K_{max} and all cores share zero agreement. With the set of cores C we can relabel all data based on the label array percentage of each core. The final labelling Z will generate a labelling from Y based on the cores C. If less than K_{max} cores are found then we introduce a noise label to round out the probabilities. On the following pages we show code to perform the reconciliation, a description in set notation, then a matrix form calculation of the process.

We appropriately named the use of IIC with this reconciliation method Invariant Information Spike Sorting IISS .

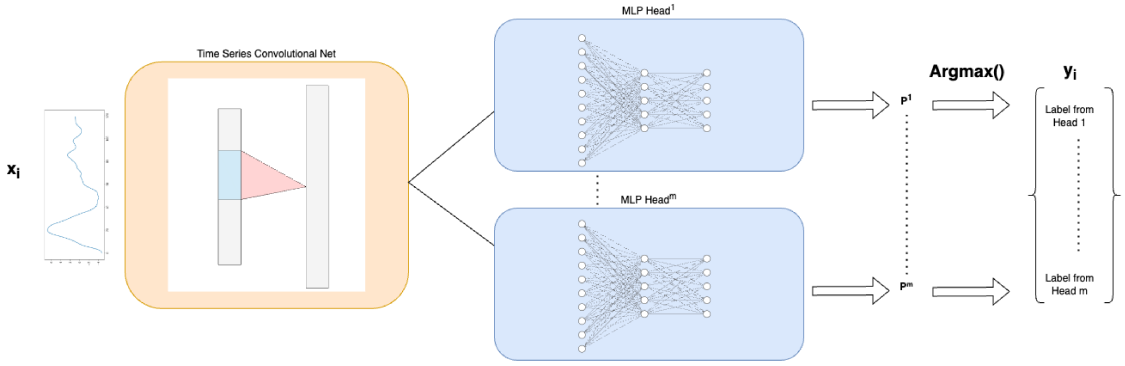


Figure 4.4: The forward pass of a spike x_i through the network creating a vector output of y

[3]We also impose a minimum cluster size parameter so that when the set is too small to start a new cluster we don't generate tiny new clusters.

```
#for each head get all predictions of the data.
#take the maximum of the softmax to be the label
Y = [F(X,head=i).argmax(dim=1) for i in range(m)]
Y = np.stack(Y).T # n x m matrix
mcs = 100 #minimum core size
Y_remain = Y.copy() #define a mutable copy of Y

while(len(Y_remain)!=0):
    #find all the unique labels and how many occurances of them

    unique_labels,counts = np.unique(Y_remain,return_counts=True):

    #find the most common one
    core = unique_labels(counts.argmax)

    #break the loop if it doesnt occur sufficiently
    if counts.max() < mcs:
        break
    else:
        cores.append(core)
        #remove all data that shares > s labels with this core
        Y_remain = Y_remain[np.sum(Y_remain==core,axis=1)==0]
cores = np.stack(cores)
```

$$F = \text{Neural Net},\ X = \text{The dataset}$$

$$Y = \{F(x)\ \forall\ x\ \epsilon\ X\}$$

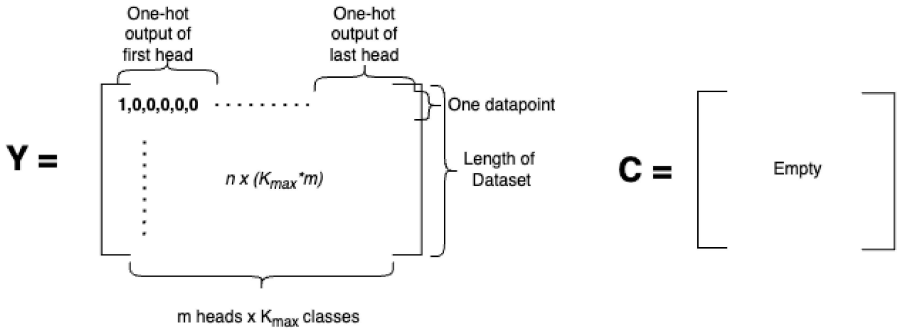
$$C = \text{Reconcile}(Y)$$

$$Z = \{\ \{\frac{\sum_{i=1}^{i=K_{max}}[y_i = c_i]}{||C||}\ \forall c\ \epsilon\ C\}\ \forall\ y\ \epsilon\ Y\}$$

Extend an extra category in Z to account for noise such that

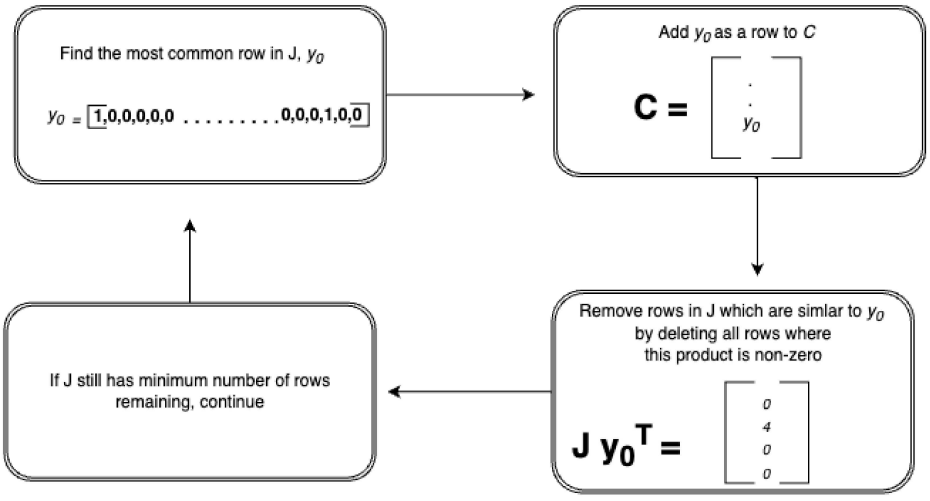
$$\forall\ z\ \epsilon Z, \sum z = 1$$

Y = F(X) The output of *m heads* from neural network F



J = Y.copy()

Core Finding Loop



$$C = \begin{bmatrix} 0,1,0,0,0,0 & \cdots & 0,0,0,1,0,0 \\ \vdots & & \\ & c \times (K_{max}{*}m) & \end{bmatrix}$$

c = number of cores found
c <= K_{max}

$$Z = Y\,C^{T} = \begin{bmatrix} 1,0,0,0,0,0 & \cdots & 0,0,0,1,0,0 \\ \vdots & & \\ & n \times (K_{max}{*}m) & \end{bmatrix} \begin{bmatrix} 0, & \cdots \\ 1, & \\ 0, & \\ 0, & \\ 0, & \\ \vdots & \\ & (K_{max}{*}m) \times c \\ 0, & \\ 0, & \\ 0, & \\ 1, & \\ 0, & \\ 0, & \end{bmatrix}$$

The i,j value
refers to the number of heads point i predicts
that are in agreement with core j

$$Z = \begin{bmatrix} 0, & 1, & \cdots & 1, & 2, \\ \vdots & & & & \\ & & n \times c & & \\ 5, & 0 & \cdots & 0, & 0 \end{bmatrix}$$

Since there are *m* heads the maximum value of any entry is *m*. Consequently the sum of the row is also less than or equal to m

Divide by *m*, in this example take m=5

$$Z = \begin{bmatrix} 0, & 0.2, & \cdots & 0.2, & 0.4, \\ \vdots & & & & \\ & & n \times c & & \\ 1, & 0 & \cdots & 0, & 0 \end{bmatrix}$$

Probability mass function across *c* labels

Append on an extra column for 'noise' so all rows sum to 1

$$Z = \begin{bmatrix} 0, & 0.2, & \cdots & 0.2, & 0.4, & 0.2 \\ \vdots & & & & & \vdots \\ & & n \times (c+1) & & & \\ 1, & 0 & \cdots & 0, & 0, & 0 \end{bmatrix}$$

\

4.4 Model Architecture & Training

Architecture

In the IIC work the authors used a modified ResNET, VGGnet, or custom CNN architecture. In all adaptations the backbone consists of numerous convolutional layers outputting a high-dimensional feature space which the MLP head(s) learn from. For an adaptation from vision to time-series the backbone convolutional layers must be exchanged from 2-D to 1-D. We suggest that the number of convolutional layers going from 2-D to 1-D also be greatly reduced. This reduction was found to greatly accelerate training with an increase in performance stability. This is likely due to there being less rejected information in the short time-series than there is in an image and that all our spikes are focused - the peak repeatedly being in the same index. The MLP heads will be two fully connected layers which include dropout. The dropout inclusion allows for adaptable training and smooth transitions when discovering new labels. For testing, we'll use a single convolutional layer followed by a MaxPool operation and a flattening as a backbone. The pass through this section creates the backbone representation which all m heads will learn from. The heads will be a 2 layer fully connected MLP that compresses the backbone representation to an output prediction distribution. Dimensions at each step can be seen in Figure 4.5.

Loss Calculation

The next deviation from the original method is to train the network using all heads at once. In the original method the clustering head was randomly selected per epoch to train. We propose that data should be passed through all heads of the network simultaneously for all batches. This is to say for paired batch data x_1, x_2 the network will produce m pairs of probability distributions P_1^i, P_2^i. The loss is then calculated as an ensemble average of losses from all the heads $\mathcal{L} = \sum_i^{i=m} I(P_1^i, P_2^i)/m$. We find that in training the ensemble we are able to train stably at faster learning rates versus when stochastically choosing individual heads to train per epoch or batch.

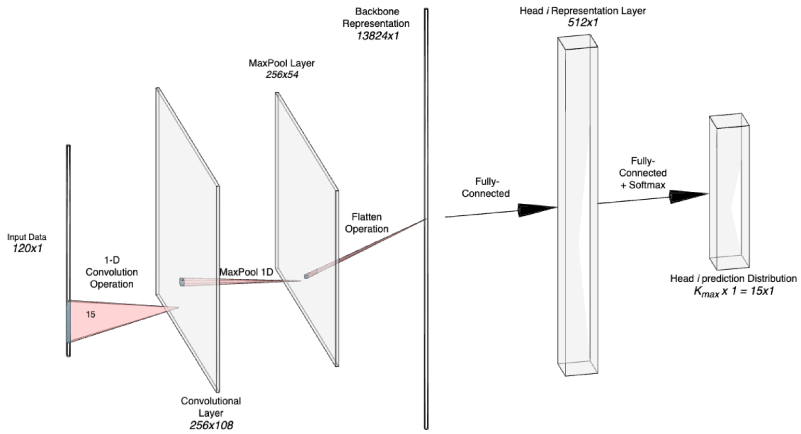


Figure 4.5: Neural Network Architecture used for IISS: A convolution of window length 15 is passed on the input data 256 times. Padding and Stride are set to 1, thus the output width is $\frac{120+2*1-15}{1} + 1 = 108$. The MaxPool layer down samples by a factor of 2. The backbone representation is presented to all m MLP heads for predictions.

4.5 Summary

We propose to cluster waveform data by creating an extension of IIC. For building invariant pairs we propose to add background noise and scale the low frequency aspects of the waveforms to generate facsimiles. We will use a two part $1+m$ (backbone + heads) component neural network to create m probability distributions over K_{max} labels. Further we will reconcile the m distributions through finding consensus groupings. The assumption the reconciliation approach makes is that what is commonly discovered across independent heads is a fundamental label. The final clustering will have $\leq K_{max}$ labels and include a noise category. Our hopes is that this approach will address the noise inherent in waveform spike sorting

The workflow to use the method is as follows

1 Define Transforms

2 Train network with m heads of output length K_{max}

3 Reconcile Labels

Chapter 5

Results

To test IISS are looking for minimal variance with respect to the transform definition. In other words, we want robustness with respect to the choice of Φ. We define four transform settings for testing using superscripts for clarity with respect to previous notation. Recall that simulations for each dataset have varying noise choices, scaling, and number of clusters. The first two parameter settings are a mix of low-frequency scaling and noise addition. The third only performs low-frequency scaling and the fourth only adds noise. There was no prior knowledge which of these Φ would perform best for any dataset. All transforms maintain the average power spectrum of the dataset.

Table 5.1: Transform Φ parameters for testing

	j	α	n
$\Phi^1(j, \alpha, n)$	10	2	1
$\Phi^2(j, \alpha, n)$	15	1.5	2
$\Phi^3(j, \alpha, n)$	10	2	0
$\Phi^4(j, \alpha, n)$	1	1	3

In testing we executed the same hyper-parameter settings 10 times for each dataset, for each Φ, for a total of 40 runs per dataset. The convolutional backbone as well as the MLP head(s) both consisted of a single layer making the forward pass a total of 2 layers leading to efficient training. Batch size of 1028 and learning rate of 0.001 were used with the ADAM optimizer. All models were complete after 100 epochs of training. For the tests which include reconciliation we set output length $K_{max} = 15$ across all with 5 MLP heads. For reconciliation minimum cluster size was set to 100 points. Including reconciliation calculation, each model took 1-5 minutes to train & compute labels with a Macbook Pro 64GB, M2Max 38-Core GPU. Different hyper parameter settings could be tuned for better performance - a larger network, learning

rate optimizers etc - however due to the unsupervised nature we wanted to test the same system across multiple conditions.

The IIC Method was designed for known K. At minimum we would hope it'll work given a case where the number of clusters is known. For this test we ignore reconciliation and use one backbone and one head with the correct softmax length given the dataset. This fundamental architecture should shine light on the methods upper-limit.

Table 5.2: IIC Performance using the Correct Number of Clusters

	Mean	Max	Min
DataSet 1 K = 7 Benchmark: 0.697			
Φ^1	0.720	0.750	0.702
Φ^2	0.738	0.764	0.723
Φ^3	0.665	0.68	0.626
Φ^4	0.712	0.722	0.706
DataSet 2 K = 9 Benchmark: 0.597			
Φ^1	0.791	0.886	0.740
Φ^2	0.817	0.851	0.751
Φ^3	0.765	0.810	0.721
Φ^4	0.798	0.831	0.775
DataSet 3 K = 11 Benchmark: 0.623			
Φ^1	0.636	0.66	0.615
Φ^2	0.683	0.694	0.673
Φ^3	0.476	0.488	0.460
Φ^4	0.738	0.767	0.722
DataSet 4 K = 13 Benchmark: 0.598			
Φ^1	0.592	0.613	0.572
Φ^2	0.621	0.643	0.59
Φ^3	0.458	0.498	0.431
Φ^4	0.653	0.657	0.648

The success and failure of Φ^4, Φ^3 respectively suggests that the low-frequency scaling is much less important than the background noises across all datasets. All models that included some factor of the background noises outperformed benchmarks the save for some using Φ^1, Φ^2 on the fourth dataset - we assume this would be an insufficient amount of background noises added (1 and 2 respectively). Overall the ordinary IIC

method shows great capacity for improvement over typical FE+CA architectures. This said, this method is presented more knowledge than FE+CA pairs - it's given the number of clusters and information on the noise. We next look to remove the knowledge of number of clusters and implement multiple heads and the reconciliation algorithm as the true test.

Once again we test four transforms across the datasets supplementing Φ^3 for another blend of scaling and background noise. This time we use the IISS method that includes the multiple heads, overclustering and label reconciliation.

Table 5.3: Updated Transform Φ parameters for testing

	j	α	n
$\Phi^1(j, \alpha, n)$	10	2	1
$\Phi^2(j, \alpha, n)$	15	1.5	2
$\Phi^3(j, \alpha, n)$	5	1.25	3
$\Phi^4(j, \alpha, n)$	1	1	3

We find that IISS clustering strongly outperforms the best FE+CA pairings for 3/4 datasets and consistently improves over density-based & non-parametric models which are used in spike sorting. This confirms our hypothesis given an idea of what causes intra-class variance, it can be removed for superior clustering. We also find that the reconciliation algorithm struggles to directly find K. This said, the GMM+BIC selected 11 clusters for each dataset suggesting that the overlaps in clusters is rather difficult.

Next we test for sensitivity w.r.t the upper bound of K_{max}. We use dataset 1 with transform 1 and test for $K_{max} = [8..17]$ with the same architecture as before. We find as K_{max} increases away from the true value there is a reduction in performance.

The final test is for sensitivity w.r.t the number of heads $m = [2...21]$. using $K_{max} = 15$. As expected increasing m causes a reduction in clusters produced from reconciliation. This increases performance if over-clustered and reduces in under-clustering. The jumps in solutions can be attributed to our relatively small networks and short training times - averaging over many runs we'd expect smooth curves. The error prone runs where performance dips are product of low-agreement across heads

Table 5.4: IISS NMI scores across datasets. Benchmarks listed in order of: non-parametric, density, centriod type models

	Mean	Max	Min	Std.	# Clusters
Dataset 1 K = 7 Benchmarks: 0.587, 0.704, 0.697					
Φ^1	0.669	0.678	0.654	0.018	9.8
Φ^2	0.693	0.716	0.673	0.013	9.5
Φ^3	0.696	0.709	0.675	0.011	9.7
Φ^4	0.660	0.680	0.638	0.013	10.8
Dataset 2 K = 9 Benchmarks: 0.531, 0.543 , 0.597					
Φ^1	0.842	0.866	0.808	0.016	11.4
Φ^2	0.849	0.896	0.815	0.022	11.4
Φ^3	0.859	0.896	0.838	0.015	11.4
Φ^4	0.836	0.880	0.787	0.027	11.0
Dataset 3 K = 11 Benchmark: 0.589, 0.582, 0.623					
Φ^1	0.650	0.664	0.636	0.009	11.1
Φ^2	0.717	0.755	0.688	0.017	11.1
Φ^3	0.726	0.734	0.718	0.005	11.0
Φ^4	0.784	0.801	0.769	0.009	11.2
Dataset 4 K = 13 Benchmark: 0.567, 0.563, 0.598					
Φ^1	0.627	0.639	0.618	0.006	11.9
Φ^2	0.637	0.648	0.614	0.010	11.5
Φ^3	0.663	0.675	0.644	0.009	11.8
Φ^4	0.655	0.662	0.634	0.008	11.9

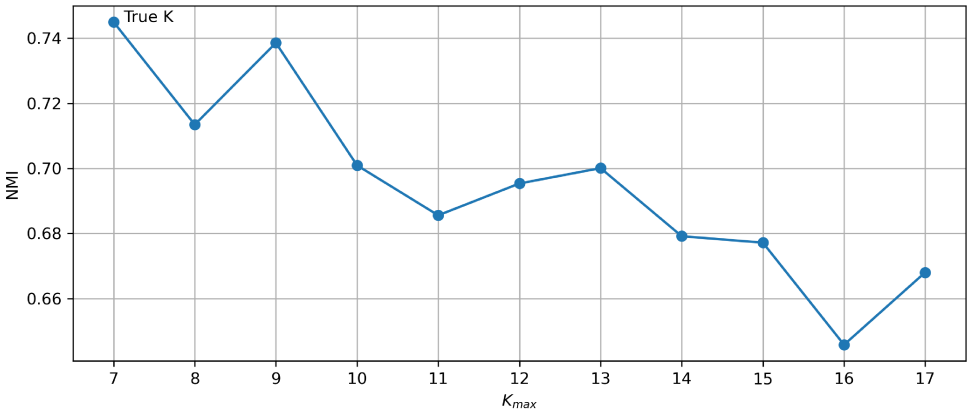
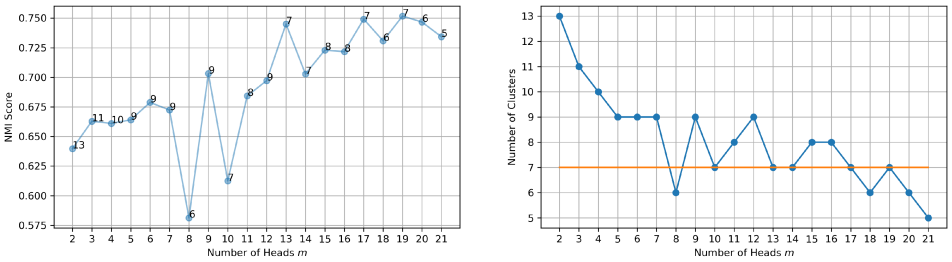
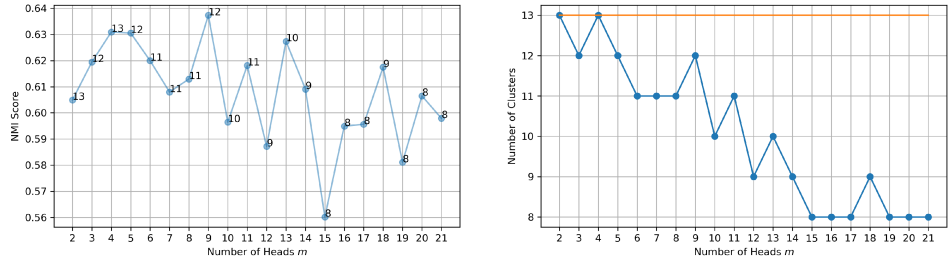


Figure 5.1: NMI of IISS vs K_{max} selection. Performance degrades as K_max shifts away from the true value.



Dataset 1 with Transform 1, annotations on left graph are the number of clusters found. Orange line on the right shows the true number of clusters.



Dataset 4 with Transform 1

Figure 5.2: Increasing the number of heads generally leads to a reduction in the number of clusters predicted. If that leads towards the true number of cluster there is a general increase in performance.

leading to missed clusters being labels as noise. Given longer training times agreement across heads stabilizes.

With arbitrary selection of K_{max} and m the method generates results comparable or better to that of FE+CA with known K or optimally tuned density models.

Chapter 6

Conclusion

6.1 Discussion

In our investigation of spike sorting we outlined three areas lacking developments: standardized datasets, clustering model inspections, and deep learning. Our simulation built tunable waveform datasets that can act as benchmark problems for sorting. While the simulation proposed is not hyper-realistic - no temporal effects or spatial modelling - the resulting clustering problems were difficult for published models to solve and give insight to how we can approach real data. We inspected popular clustering models and found that: 1. It's likely that PCA should be abandoned for UMAP in basic feature extraction and 2. The Mean Shift algorithm is the most likely the most consistent classical approach in low dimensions (2-5). Finally, the main novelty in this work is proposing IISS to address deep learning in spike sorting.

We introduced a method on this first inspection that surpasses the capacity to cluster spikes done across classical architectures. IISS shows that having knowledge of what causes intra-class variance is as powerful as knowing the number of clusters or knowing correct densities using classical methods. Given there is underpinning physics for every experiment, we propose that the approximation we make when assuming a transform is a more reasonable heuristic than presuming the number of clusters. When considering FE + density-based approaches the probability of selecting the optimal parameter setting is low and will likely be outperformed by any parameterization of IISS. This leads us to conclude that the classical architecture for spike sorting can and will be superseded by deep learning approaches.

6.2 Future Works

Extensions to Multivariate & Real Data

We have designed the IISS clustering method as an extension of IIC for unknown clusters and adapted the backbone for univariate time-series data. This method works to solve our problem of individual spike waveforms. The natural follow-up is multivariate time-series such as recordings from matrix array data where probes are spatially correlated. The method can be extended by using the same transforms and using the same convolutional backbones but with multiple channels. This extension is akin to using a 2-D convolutional neural net working on grey scale images then extending single channel to three channels for RGB images.

In our simulation and testing we referred to *real noise* as a set of noises σ sampled from a real recording. If this set of noises was inspected through a rolling window in time, we would find that the mean and variance go through large swings. This non-stationary distribution of background noise and neural activity relates to the probe drift problem and various other psychological responses. Thus, we incorrectly treated the entirety of σ as a stationary set however in application we could select for noises locally around the time of the spike. For example only using background noise within a minute of the action potential. In this case the background noise distribution should approximately be stationary. This gives our transform temporal context and we would expect a set of improved results.

Generative Φ

The notion of designing Φ to represent intra-class variance is a hurdle to overcome in order to use our method. Φ aims to model redundancy inherent in individual samples. The fact we're aiming for inherent redundancies suggests that the data itself has information of these. A complete method may propose generating Φ as a function of the dataset itself. We believe Φ might be able to be modeled as as a generative adversarial neural network (GANN) [8]. In this case a GANN Φ would attempt to generate paired data indiscriminate from reality. This idea requires further inspection however the end goal would be to produce transforms as a function of the real signal.

Improving Reconciliation

The reconciliation of heads needs further development. The method is a democratic process for finding areas of complete agreement amongst m voters which each bin data into K_{max} classes. A self-supervised metric which could point to the number of clusters would prove helpful as K_{max} could be edited during training to aim towards the correct number of clusters. Additionally, we consider the heads as training to find K classes; thus, varying K across heads may improve robustness. Given that the reconciliation sets an upper bound, heads may not use output lengths below a bound we deem acceptable. It may be helpful to use significantly lower K for auxiliary heads which aren't used for final labelling. This inverses the original over clustering idea proposed in IIC, a lower number of classes may direct the backbone to extract only core features.

Using IISS Architecture for Representation Learning

The output of the IISS backbone is a high dimensional (for our architecture approx 14000 dimensions) feature space which each MLP head attempts to learn a labelling. As a byproduct of clustering we've built a feature extraction method which sits halfway through the network. To explore the use-case as an encoding network we could add a secondary layer between the backbone and the heads which compresses to low dimensional space. Furthermore heads could vary output length $[2....n]$ which would compete for global vs local separation structures. In essence this would create an embedding space trained to separate data based on Φ.

Codes for this project will be posted on github @alexkaravos by April 2024.

Bibliography

[1] Christopher Bishop. Pattern recognition and machine learning. *Springer google schola*, 2:5–43, 2006.

[2] Alessio P Buccino, Cole L Hurwitz, Samuel Garcia, Jeremy Magland, Joshua H Siegle, Roger Hurwitz, and Matthias H Hennig. Spikeinterface, a unified framework for spike sorting. *Elife*, 9:e61834, 2020.

[3] Fernando J Chaure, Hernan G Rey, and Rodrigo Quian Quiroga. A novel and fully automatic spike-sorting implementation with variable number of features. *Journal of neurophysiology*, 120(4):1859–1871, 2018.

[4] 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.

[5] 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.

[6] Martin Ester, Hans-Peter Kriegel, Jörg Sander, Xiaowei Xu, et al. A density-based algorithm for discovering clusters in large spatial databases with noise. In *kdd*, volume 96, pages 226–231, 1996.

[7] Keinosuke Fukunaga and Larry Hostetler. The estimation of the gradient of a density function, with applications in pattern recognition. *IEEE Transactions on information theory*, 21(1):32–40, 1975.

[8] Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative adversarial networks. *Communications of the ACM*, 63(11):139–144, 2020.

[9] 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.

[10] Xu Ji, Joao F Henriques, and Andrea Vedaldi. Invariant information clustering for unsupervised image classification and segmentation. In *Proceedings of the IEEE/CVF international conference on computer vision*, pages 9865–9874, 2019.

[11] James J Jun, Catalin Mitelut, Chongxi Lai, Sergey L Gratiy, Costas A Anastassiou, and Timothy D Harris. Real-time spike sorting platform for high-density extracellular probes with ground-truth validation and drift correction. *BioRxiv*, page 101030, 2017.

[12] Guy Kember. isosplit. `https://github.com/gkember/isosplit`, 2021.

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

[14] Yann LeCun, Corinna Cortes, and Chris Burges. Mnist handwritten digit database. `http://yann.lecun.com/exdb/mnist/`, 2010.

[15] Jeremy Magland, James J Jun, Elizabeth Lovero, Alexander J Morley, Cole Lincoln Hurwitz, Alessio Paolo Buccino, Samuel Garcia, and Alex H Barnett. Spikeforest, reproducible web-facing ground-truth validation of automated neural spike sorters. *eLife*, 9:e55167, may 2020.

[16] Jeremy F Magland and Alex H Barnett. Unimodal clustering using isotonic regression: Iso-split. *arXiv preprint arXiv:1508.04841*, 2015.

[17] Leland McInnes, John Healy, Steve Astels, et al. hdbscan: Hierarchical density based clustering. *J. Open Source Softw.*, 2(11):205, 2017.

[18] Leland McInnes, John Healy, and James Melville. Umap: Uniform manifold approximation and projection for dimension reduction. *arXiv preprint arXiv:1802.03426*, 2018.

[19] M Mehrotra, V Reddy, and P Singh. Neuroanatomy, stellate ganglion. `https://www.ncbi.nlm.nih.gov/books/NBK539807/`, 2023. Updated 2023 Jul 24. In: StatPearls [Internet]. Treasure Island (FL): StatPearls Publishing; 2024 Jan-.

[20] Marius Pachitariu, Nicholas Steinmetz, Shabnam Kadir, Matteo Carandini, and Harris Kenneth D. Kilosort: realtime spike-sorting for extracellular electrophysiology with hundreds of channels. *BioRxiv*, page 061481, 2016.

[21] Qi Qian. Stable cluster discrimination for deep clustering. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 16645–16654, 2023.

[22] Mohammadreza Radmanesh, Ahmad Asgharian Rezaei, Mahdi Jalili, Alireza Hashemi, and Morteza Moazami Goudarzi. Online spike sorting via deep contractive autoencoder. *Neural Networks*, 155:39–49, 2022.

[23] Alex Rodriguez and Alessandro Laio. Clustering by fast search and find of density peaks. *science*, 344(6191):1492–1496, 2014.

[24] Meitar Ronen, Shahaf E Finder, and Oren Freifeld. Deepdpm: Deep clustering with an unknown number of clusters. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 9861–9870, 2022.

[25] Joseph W Salatino, Kip A Ludwig, Takashi DY Kozai, and Erin K Purcell. Glial responses to implanted electrodes in the brain. *Nature biomedical engineering*, 1(11):862–877, 2017.

[26] scikit-learn developers. Scikit-learn: Machine learning in python 2.3 clustering, 2023. Accessed: 2023-2024.

[27] Connor Shorten and Taghi M Khoshgoftaar. A survey on image data augmentation for deep learning. *Journal of big data*, 6(1):1–48, 2019.

[28] Koustubh B Sudarshan, Yuichi Hori, M Amer Swid, Alexander C Karavos, Christian Wooten, J Andrew Armour, Guy Kember, and Olujimi A Ajijola. A novel metric linking stellate ganglion neuronal population dynamics to cardiopulmonary physiology. *American Journal of Physiology-Heart and Circulatory Physiology*, 321(2):H369–H381, 2021.

[29] Laurens van der Maaten and Geoffrey Hinton. Visualizing data using t-sne. *Journal of Machine Learning Research*, 9(86):2579–2605, 2008.

[30] Wouter Van Gansbeke, Simon Vandenhende, Stamatios Georgoulis, Marc Proesmans, and Luc Van Gool. Scan: Learning to classify images without labels. In *European conference on computer vision*, pages 268–285. Springer, 2020.

[31] Ankit Vishnubhotla, Charlotte Loh, Liam Paninski, Akash Srivastava, and Cole Lincoln Hurwitz. Towards robust and generalizable representations of extracellular data using contrastive learning. *bioRxiv*, pages 2023–10, 2023.

[32] An Xiao, Hanting Chen, Tianyu Guo, Qinghua Zhang, and Yunhe Wang. Deep plug-and-play clustering with unknown number of clusters. *Transactions on Machine Learning Research*, 2022.

[33] 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.

[34] Liangyu Zhang, Dongrui Gao, and Manqing Wang. Sorting overlapping spikes based on log-mel spectrogram and convolutional neural networks. In *2023 6th International Conference on Artificial Intelligence and Big Data (ICAIBD)*, pages 482–485. IEEE, 2023.