Statistical Machine Learning & Deep Neural Networks Applied to Neural Data Analysis

Hooshmand Shokri Razaghi

Submitted in partial fulfillment of the
requirements for the degree of
Doctor of Philosophy
under the Executive Committee
of the Graduate School of Arts and Sciences

COLUMBIA UNIVERSITY

2020

Abstract

Statistical Machine Learning & Deep Neural Networks Applied to Neural Data Analysis

Hooshmand Shokri Razaghi

Computational neuroscience seeks to discover the underlying mechanisms by which neural activity is generated. With the recent advancement in neural data acquisition methods, the bottleneck of this pursuit is the analysis of ever-growing volume of neural data acquired in numerous labs from various experiments. These analyses can be broadly divided into two categories. First, extraction of high quality neuronal signals from noisy large scale recordings. Second, inference for statistical models aimed at explaining the neuronal signals and underlying processes that give rise to them. Conventionally, majority of the methodologies employed for this effort are based on statistics and signal processing. However, in recent years recruiting Artificial Neural Networks (ANN) for neural data analysis is gaining traction. This is due to their immense success in computer vision and natural language processing, and the stellar track record of ANN architectures generalizing to a wide variety of problems. In this work we investigate and improve upon statistical and ANN machine learning methods applied to multi-electrode array recordings and inference for dynamical systems that play critical roles in computational neuroscience.

In the first and second part of this thesis, we focus on spike sorting problem. The analysis of large-scale multi-neuronal spike train data is crucial for current and future of neuroscience research. However, this type of data is not available directly from recordings and require further processing to be converted into spike trains. Dense multi-electrode arrays (MEA) are standard methods for collecting such recordings. The processing needed to extract spike trains from these raw electrical signals is carried out by “spike sorting” algorithms. We introduce a robust and scalable MEA spike sorting pipeline `YASS` (Yet Another Spike Sorter) to address many challenges that are inherent to this task. We primarily pay attention to MEA data collected from the primate retina for important reasons such as the unique challenges and available side information that ultimately assist us in scoring different spike sorting pipelines. We also introduce a Neural Network architecture and an accompanying training scheme specifically devised to address the challenging task of deconvolution in MEA recordings.

In the last part, we shift our attention to inference for non-linear dynamics. Dynamical systems are the governing force behind many real world phenomena and temporally correlated data. Recently, a number of neural network architectures have been proposed to address inference for nonlinear dynamical systems. We introduce two different methods based on normalizing flows for posterior inference in latent non-linear dynamical systems. We also present gradient-based amortized posterior inference approaches using the auto-encoding variational Bayes framework that can be applied to a wide range of generative models with nonlinear dynamics. We call our method *Filtering Normalizing Flows* (FNF). FNF performs favorably against state-of-the-art inference methods in terms of accuracy of predictions and quality of uncovered codes and dynamics on synthetic data.

Table of Contents

List of Tables v

List of Figures vi

Chapter 1: Introduction and Background 1

1.1 Spike Sorting 1

1.1.1 Detection 4

1.1.2 Featurization and clustering 4

1.1.3 Deconvolution 5

1.1.4 Iterative approach 5

1.1.5 Drift Correction 6

1.1.6 Highlights of some of recent pipelines 6

1.2 Imitation Learning for Multi-Electrode Array Spike Deconvolution 8

1.3 Deep Generative Models and Neural Data 9

1.3.1 Variational Deep Networks for Dynamics 11

Chapter 2: YASS: Yet Another Spike Sorter 13

2.1 Methods 13

2.1.1 Overview 13

2.1.2 Preprocessing 16

2.1.3 Detection 16

2.1.4 Denoising 17

2.1.5 De-duplication of detected events 18

2.1.6 Alignment 19

2.1.7 Coarse-to-fine divide-and-conquer clustering with triaging and iterative re-featurization 19

2.1.8 Cluster post-processing 26

2.1.9 Deconvolution 28

2.1.10 Post-deconvolution merging and splitting 31

2.1.11 Drift handling 33

2.1.12 Soft assignment 34

2.1.13 Synthetic and semi-synthetic data 35

2.2 Results 43

2.2.1 Basic characterization of YASS output 43

2.2.2 Comparisons to manual sorts and other automated spike-sorters 47

Chapter 3: Imitation Learning for MEA Spike Deconvolution 55

3.1 Introduction 55

3.2 Enhancing Greedy Deconvolution with Imitation Learning 57

3.3 **PermSepConv**: **Perm**utation Equivariant **Sep**arable 1D Convolution Block 58

3.4 Training Spike Checker by Data Aggregation 59

3.5 Results 62

3.5.1 Synthetic MEA data 62

3.5.2 Neural Network Architecture 63

3.5.3 DAGGER Configuration 63

3.5.4 Comparisons with Greedy Deconvolution 64

Chapter 4: Filtering Normalizing Flows 70

4.1 Normalizing Flows 71

4.2 Filtering Normalizing Flows 72

4.2.1 Time Autoregressive 73

4.2.2 Time Inverse Autoregressive 74

4.3 Auto-Encoding Variational Bayes 75

4.4 Experimental Results 76

4.4.1 Latent Lorenz System 77

4.4.2 Rotating MNIST Images 78

References 87

Appendix A: Notation 88

Appendix B: Additional details on the detection algorithm 90

B.0.1 Neural network training data 90

B.0.2 Neural network structure 91

Appendix C: Additional details on the denoising algorithm 93

Appendix D: Additional details on the Mixture of Finite Mixture (MFM) model 94

D.0.1 Generative Model 94

D.0.2 Inference model and estimation 94

Appendix E: Additional details on spline interpolation within deconvolution 96

List of Tables

A.1 **Summary table of notation used within the appendix.** 89

List of Figures

1.1 **High per-electrode firing rates and the prevalence of collisions in primate retina data**. (Left) voltage signal on 7 neighboring channels during a randomly selected 10 ms interval; (right) spike shapes and times identified and sorted by YASS (each unit is assigned a random color). The gray shade (here and in the following figures) represents ± 1 estimated noise scale (estimated as the median absolute deviation of the raw data on each channel, divided by 0.67449). Note that some of the spikes visible on these electrodes are small but are larger on other electrodes (not shown); all units shown here have templates with peak-to-peak amplitude $> 4.5\times$ the estimated noise scale on at least one electrode. The total firing rate (summed over all visible units) on each channel here is in the hundreds of Hz range, leading to frequent collisions. 3

2.1 **Illustration of detection and de-duplication steps.** (**Left**) Raw recording on seven neighboring electrodes; (**Middle**) Neural Network (NN) output (red) overlaid on top of raw recording. (**Right**) detected and de-duplicated spikes. Red (blue) boxes denote spikes that are removed (retained) during the de-duplication step. 17

2.2 **Illustration of denoising step**. **(A)**: Four examples of raw detected spikes (black) and corresponding NN-denoised spikes (red) and PCA projections (blue). Note that the NN successfully suppresses both noise and contributions from other collided spikes, while the PCA projection is significantly more sensitive to collisions. **(B)**: (Top) raw detected spikes collected on a single electrode (left) vs. the same spikes after denoising (right). (Bottom): PCA projections corresponding to the raw (left) vs denoised spikes (right). Note that many of the outliers are no longer visible in the PCA projections, and the three clusters are much more easily distinguishable by eye. 18

2.3 **Divide-and-conquer clustering with triaging and iterative re-featurization**. (Top left): Outlier triaging. Top middle: mixture-of-finite-mixture model (MFM) clustering of datapoints remaining after triage step. Each color represents a single cluster returned by MFM run in a higher-dimensional feature space. Top right: grouping of clusters with substantial overlap using connected components. (Bottom row): Three examples illustrating the benefits of iterative re-featurization. Each box represents a single clustering iteration. Ellipses represent MFM clusters and colors represent connected components (CCs). While the actual featurization is 5 dimensional, for visualization here these 5d vectors are projected on the 2 dimensional feature space that best separates the clusters (via linear discriminant analysis). In each "generation" we perform a triage, then an MFM clustering, then a connected-components analysis (as in the top row); then we re-featurize the data within each CC and iterate in the next generation. Colored lines indicate the flow of data in this procedure: a CC in one generation might generate multiple CC in the following generation, if the new featurization is informative enough to split clusters that were not separable in the previous generation. Indeed, as shown in Examples 1 and 2, many CCs can be more clearly split in the next generation by adaptive re-featurization. Example 3 illustrates the necessity of features from distant electrodes. In the first two rows, clustering uses features from local electrodes only, and the last two rows utilize features from distant electrodes. Although MFM returns a single blue cluster in generation 1 of local clustering, re-featurizing using distant features shows two clearly separable red and green CCs in generation 0 using distant features. See also Figure 2.4 below for an illustration of some units that are only separable via distant features. 23

2.4 **Examples illustrating the need for distant-electrode featurization.** Two example pairs of units whose templates look locally similar but can be easily separated if information from all electrodes are used in the clustering process. In each example, the green dot represents the (shared) primary channel of the two templates indicated in blue and orange. Only "active" channels (where the template height is above a threshold) are shown. 24

2.5 **An example collision unit**. (**A**) The template of the collision unit. Blue dot indicates the primary electrode (with the largest template), and red dots indicate the top 3 electrodes with the highest variability. (**B**) Top: raw spikes (gray traces) and templates (color) at the primary electrode and the three most-variable electrodes. Bottom: the estimated variance of spikes (black) and the threshold (red) used to detect overly-variable units. Due to small differences in the timing of one collided spike relative to the other, the variability of collided waveforms tends to be significantly elevated. 27

2.6 **Temporally aligned low-rank template model.** (**Top**) An example unit, showing that the raw template is well-summarized by its low-rank SVD reconstruction following temporal alignment, i.e., the blue and black traces show a high degree of overlap. Template plot conventions as in Figure 2.5. (**Middle row**) Comparing the SVD residual with versus without alignment. First panel shows the raw template, normalized per channel; note that the peak time varies by about a millisecond across the different channels. Second panel shows that a low-rank SVD approximation leaves behind significant residual structure. Third panel shows aligned normalized template, and fourth panel shows that aligned SVD residual is significantly smaller than unaligned. (**Bottom**) Summary of residual errors across all templates in one dataset. For both rank 5 and 10 SVD approximations, aligning leads to significantly reduced residual error. 37

2.7 **Super-resolution interpolation and subtraction improves deconvolution.** (**Top row**) Three raw spikes from three distinct neurons are shown on the main channel of their respective neurons alongside neuron templates, their interpolations, and optimally time shifted and sub-sampled templates to achieve the smallest residual. Discrete signals are represented by scatter plot while interpolations are solid lines. (**Second row**) Residuals are shown when templates and optimal templates are subtracted from the raw spikes above. In all these cases the raw residuals have significant amplitude and can be mistaken as spikes from other neurons, while the optimally-aligned residuals are significantly smaller. (**Third row left & middle**) MAD of post-deconvolution residuals (in standardized voltage units) for spikes of two large neurons (“ptp" abbreviates the maximal peak-to-peak height of the spike template). Naive subtraction leaves significant residual variability behind (black traces); conversely, residuals are substantially smaller when super-resolution alignment and interpolation of the template prior are applied prior to subtraction (red trace). (**Third row right**) Residual variability using naive subtraction (black) scales with unit PTP, but with interpolation the residual error stays uniformly low (red; each symbol corresponds to a single cell). 38

2.8 **Comparing greedy vs iterative coordinate descent (ICD) deconvolution.** **(A)**: Examples of spikes recovered by ICD but missed by greedy deconvolution (and vice versa). All comparisons here performed on synthetic data with known ground truth spike times (see section 2.1.13 for a description of synthetic data generation). Blue indicates the true underlying (noiseless) spike template; black is an example of an observed noisy voltage trace containing the spike that greedy deconvolution missed but ICD caught; red indicates traces for which ICD missed the spike but greedy deconvolution accurately captured the spike (up to a maximum of 5 traces are plotted). Note that the miss rate is often much lower for ICD. **(B-D)**: ICD improves true positive rates on the order of a few % **(B)**, decreases false positive rates by up to 50% **(C)**, and modestly improves the timing precision of detected spikes, as measured by the standard deviation (SD) of the inferred spike times **(D)**. In each case, differences between greedy versus ICD are most significant for smaller units. 39

2.9 **Post-deconvolution spike waveform cleaning. (a)** A 5ms 7 channel raw voltage recording reveals the presence of multiple spikes. **(b)** Three spikes are identified by deconvolution and are depicted by inserting each template (colored traces) at the times identified by deconvolution. **(c, d, e)** 'Clean spikes' for each of the three spikes are obtained by superimposing the template shape with the residual voltages after all the other identified neuron spikes are removed from the raw record. **Middle row**: Raw, NN-denoised, post-deconvolution cleaned spikes, and cleaned then NN-denoised spikes from a single channel. **Bottom row**: PCA projections corresponding to the waveforms from the middle row. Note that post-deconvolution spike cleaning and NN-denoising both significantly improve the effective SNR of the data. 40

2.10 **Post-deconvolution merge using cleaned spike waveforms**. Each row illustrates the process for a different pair of units whose templates are close enough to be candidates for merging. **First column**: for each unit, we project the raw waveforms onto the difference between the two templates and plot the resulting histograms. **Second column**: same, but projecting the post-deconvolution cleaned waveforms onto the linear discriminant analysis (LDA) direction; note that this improves the separation. **Third column**: the templates of each pair of units. **Last column**: receptive fields (RFs) for all these neurons; note that these RFs are not used in the merge algorithm, but are shown here for verification purposes only. Using LDA applied to clean spikes, our algorithm suggests merging templates 1 and 2 while avoiding a merge of templates 3 and 4. This decision is supported by the corresponding RFs: RFs 1 and 2 are similar (though RF 1 is much noisier due to having fewer spikes than neuron 2), while RFs 3 and 4 are clearly distinct. 41

2.11 **Drift tracking via template scaling**. **A**. Drift model example: a template is scaled to match the drifting size of a neuron observed over a period of 2 hrs (magma colors; matching template from Kilosort is in blue and was computed using 5 minutes of data). Note the YASS template amplitude decreases by 15%-20% over the duration of the 2 hr (7200 sec) recording. **B**. Testing the stability of the template and rank 1 approximation: templates recomputed (and normalized) every 2 minutes for 2 hrs using deconvolved spikes (i.e. each template represents the mean of the spikes deconvolved in a 2 minute period). Note that the normalized template is nearly identical over time, suggesting scaling is an adequate model of drift. **C**. Template from the first 5 mins of recording (red) with deconvolved spikes (10) across 2hrs of recording. The change in individual spike size (black traces) compared with the starting template is observable at the single spike level. **D**. Drift model tracks changes: PTP of raw deconvolved spikes (black), PTP of spikes selected for the drift update (red) and PTP of the drift model (cyan). This shows that the drift model is tracking drift over time despite periodic updates (e.g. every 2 mins). **E**. PTP of spikes deconvolved for the same neurons by Kilosort. **F**. Additional examples of drift tracking; conventions as in panel D, but applied to different units. 42

2.12 **Basic statistics of units extracted from three retinal datasets**. **First panel**: number of neurons per electrode for three datasets. **Second panel**: Spatial extent of each neuron, measured by counting the number of electrodes on which the template PTP was greater than 2 (note, 2009 dataset contained a higher incidence of axons, leading to more units active on a large number of electrodes). **Third panel**: Estimated firing rates of all sorted neurons. **Last panel** The rates at which spikes collide with other distinct signals of PTP size 2, 4, and 10 within a 0.5ms time window for the three datasets. Concretely, for about half of the recovered units, about half of all spikes have a collision with a spike of PTP size ≥ 2 within 0.5ms; similarly, for about half of the recovered units, about one fifth of all spikes have a collision with a spike of PTP size ≥ 4 within 0.5ms. 43

2.13 **Spatio-temporal templates and receptive fields (RFs) from four major cell classes**. **Left panels**: Spatial (filled circles) and temporal (colors) representation of templates for neurons sorted using YASS. Each dot represents the PTP of the waveform at the location (only channels with PTP>2.0 are shown), warmer colors (orange) indicate earlier time (i.e. closer to spike time) and cooler colors (e.g. blue) represent channels active at later times. **Right panels**: Spatial RF of neurons corresponding to the templates shown in the left panels. Red contour shows Gaussian fits (c.f. Figure 2.15 below). Cell types were determined from RFs by standard clustering procedures, with borderline cases determined by hand using a simple custom GUI. 44

2.14 **Example diagnostic plots for a single recovered unit**. To examine the health of the extracted units we examine templates, RFs, and autocorrelations, along with corresponding plots for “nearby" units (where unit “neighbors" can be defined according to distance in RF space, template space, or other parameters such as the degree of correlation strength). Here we show an example of some of these analysis plots for an ON-parasol cell and its four nearest neighbors as defined by RF cosine distance. From left to right, the diagnostic plots show: template propagation profile across the 512 channel array (time since spike on primary channel in ms represented by color; firing rates and PTP in standardized units are provided in the panel titles); overlapping template plot of the main neuron on its primary and neighboring channel (black) and its neighboring cells (color); the (zoomed) spatial RF; autocorrelogram and cross-correlograms with the neighboring units (in Hz); and histograms of spikes-to-template distance for all spikes in each neuron against each template after the LDA projection described in section 2.1.10 (more distance between the histograms indicates that the cells are better separated). 45

2.15 **Receptive field contour analysis** Each ellipse indicates a Gaussian fit to the receptive field of an individual unit. (Top) RF contours of cells extracted by YASS and a hand sorted result on one 512-electrode dataset. Green indicates cells that were found by both YASS and the hand sort; red indicates cells that were absent in the hand sort; black indicates cells that were found in the hand sort but not by YASS. (Bottom) Contours of Yass and Kilosort2 on three separate 512-electrode datasets. YASS consistently recovers more cells with RFs, particularly midget cells. 46

2.16 **Pairwise per-unit comparisons of spike sorters**. **A**. Example of an apparent Kilosort de-duplication error. **(i)**: (*Left*) Venn diagram for YASS unit 1410 (red) and Kilosort unit 756 (blue), showing an overlap of 2237 spikes (beige), with YASS identifying 2 extra spikes. (*Middle*) RFs for the 2 units. (*Right*) Spikes randomly selected from the intersection set. **(ii)**: (*Left*) Kilosort unit 1631 also matches YASS unit 1410, indicating that Kilosort unit 1631 is a duplicate of Kilosort unit 756. (*Middle*) RFs for the units (note the slightly weaker Kilosort RF, likely due to slightly fewer spikes). (*Right*) Spikes taken from the intersection (beige) look similar to additional spikes from YASS (red) but not additional spikes from Kilosort (blue), indicating that Kilosort unit 1631 contains some incorrectly assigned spikes. Note that spikes from different groups are displaced horizontally for better visibility here. **B**. Example of a likely Kilosort oversplit error. **(i)**: (*Left*) Venn diagram for YASS unit 1477 (red) and Kilosort unit 1626 (blue) have an overlap of 784 spikes (beige), with YASS identifying 2186 spikes and Kilosort identifying 18 extra spikes. (*Middle*) RFs for the 2 units (note that Kilosort RF is relatively weak here). (*Right*) Spikes randomly selected from the intersection set (beige) and YASS (red) indicate that additional spikes identified by YASS are consistent with the spikes recovered in the (beige) intersection set. **(ii)**: (*Left*) Kilosort unit 1766 also matches YASS unit 1477. (*Middle*) RFs for the units (note the slightly weaker Kilosort RF). (*Right*) Spikes taken from the intersection (beige) look similar to additional spikes from YASS (red) but not additional spikes from Kilosort (blue), again indicating that Kilosort unit 1631 contains incorrectly assigned spikes. 48

2.17 **Summary analysis of oversplits, duplicates, and consensus across spike sorters**. **A**. Summary of Kilosort units matched to YASS units. Most KS units matched well with YASS units (black dots) but there were many examples of oversplits and duplicates (as described in Figure 2.16; duplicates were defined as KS units that matched a YASS unit after another KS unit was matched; oversplits were defined as duplicates with fewer than two thirds of the YASS unit spikes). The y-axis indicates how many KS units were matched to a single YASS unit (see the legend for details). **B**: Same analysis as in (A) for the same dataset sorted manually. **C**: Cosine similarity between templates computed by averaging extra spikes identified by YASS (red) or KS (blue, top panels) or manual sorting (green, bottom panels) versus the template computed by averaging the "consensus" spikes identified by both spike-sorters. YASS extra spikes tend to be much more similar to the consensus template, consistent with the examples shown in Figure 2.16. In the left panels we scatterplot these cosine similarities against the PTP size of the unit; in the right panels against the firing rate of the extra identified spikes. Note that in many cases YASS identifies a large number of extra spikes that closely match the consensus template. **D**. Same analysis as in (C) for another dataset. 50

2.18 **Comparing consensus and non-consensus units from YASS and Kilosort** Among the units with firing rates bigger than 0.1Hz, We found 444 unmatched YASS-only and 101 Kilosort-only units, compared to 491 (YASS) and 595 (Kilosort) matching consensus units (note that it is not a pairwise matching due to oversplits). The top panels show non-consensus units, and the bottom panels show consensus units. We scatterplot the firing rate of each unit against: the unit's PTP; the maximum absolute deviation (MAD) of the aligned cleaned spikes (overly large values indicate corruption of these spikes, due to e.g. deconvolution errors or over-merged units); the average soft-assignment probability (i.e., one minus the probability that a spike assigned to unit i should have been assigned to some other unit j, averaged over all spikes from unit i; small values indicate closely overlapping clusters that may correspond to over-splitting); the "outlier score" defined as the average norm of the whitened post-deconvolution residual around each spike (again, large values indicate corruption of these spikes); the maximal normalized cross-correlation (zero lag, one ms resolution) between unit i and all other units j (overly large values can indicate "fragmented" units, where for example axonal channels have been split from the somatic channels); and the minimal cross-correlation, defined similarly (a large "notch" in the cross-correlation function often indicates oversplitting, since if a cell is oversplit into two units i and j then i and j will tend not to spike at the same time). 51

2.19 **Residual analysis**. **A**. A 20 ms example of YASS and Kilosort residuals on 5 channels. For large units, Kilosort occasionally leaves large residuals behind (c.f. Figure 2.7). **B**. Another example on 8 channels reveals some spike mis-assignment by Kilosort. 52

2.20 **YASS results on hybrid semi-synthetic datasets containing low amplitude units. Top**. The true positives (solid colors) and false positives (hatched colors) for the best (blue) and second best (red) matches for neurons sorted by yass (for clarity, a perfect match is a blue bar that contains 100% of the spikes with no other bars). **Bottom**. Firing rate and PTP amplitude for neurons shown above. (Note: Kilosort did not detect any of the injected units using the default thresholds). 53

3.1 **Two input examples from synthetic MEA data and corresponding outputs of our spike checker classifier.** These two examples are the same except for the order in which channels are presented. In MEA recordings, we often subset electrodes. The order in which data from these electrodes are presented can vary arbitrarily. Therefore, it is essential for our model to have the same prediction regardless of how the electrodes are ordered. This constraint potentially can assist with regularizing our model and generalizing to unobserved datasets. 58

3.2 **Depiction of permutation equivariant separable 1D convolution block.** Input to the block is comprised of a set of C matrices each of them having $c \times t$ dimensions. Each one of these matrices corresponds to one of the channels (i.e. electrodes) of the raw electrical signal in the example. Therefore, this number remains constant throughout the PermSepConv blocks in our network. A block applies the same 1D convolution kernels to all matrices. Then it applies the same permutation equivariant map to all the elements of output matrices. This is followed by an average pooling across to reduce temporal dimension. 60

3.3 **Four input examples from synthetic MEA data and corresponding spike checker outputs.** Two top examples are cases where greedy deconvolution has proposed a false assignment, whereas the two bottom examples are correct proposals. In all these cases, our spike checker has accurately identified whether the proposed spikes are correct or wrong. In these examples $C = 7$ and raw voltages x_c are depicted by blue traces while templates y_c are visualized by red/green traces respectively for false/correct assignment. 61

3.4 **Performance of PermSepConv V.S. conventional 1D convolution on spike checker task.** This plot depicts the ROC for two different spike checkers. One is base on our proposed PermSepConv architecture while the other is based on conventional 1D convolution layers. Both models have been trained using DAGGER algorithm and result is reported for best models according to cross validation on hyper-parameters. It is worth noting that the PermSepConv network has smaller parameter size than the other. 65

3.5 **Performance comparison of greedy deconvolution and enhanced deconvolution on synthetic recording.** Top row depicts the true positive rates while middle row visualizes false assignment counts. Note that these counts are visualized on a logarithmic scale. The last row summarizes the PTP and firing rates for each neuron respectively. Observe that by increasing the threshold (to a more conservative setting) true positive rates are harmed while false assignment rates are drastically reduced, however this reduction is not as much as offered by our enhanced deconvolution. 67

3.6 **Performance comparison of greedy and enhanced deconvolution with missing dictionaries.** Top row depicts the true positive rates while middle row visualizes false assignment counts. The missing bars from true positive rates correspond to the missing templates. Note that the false assignment counts are visualized on a logarithmic scale. The last row summarizes the PTP and firing rates for each neuron respectively. Similar to the previous case where all the dictionaries were provided, false assignments are decreased by using enhanced deconvolution. However the improvement is less compared to the previous case. 68

3.7 **Performance comparison of greedy and enhanced deconvolution on hybrid recording.** Top row depicts the true positive rates while middle row visualizes false assignment counts. Note that these counts are visualized on a logarithmic scale. The improvements in false assignments by enhanced deconvolution are more evident than in the case of synthetic datasets. 69

4.1 **Visualization of and observation trajectories for Lorenz system and performances on them.** (**top**) Respectively, true, FNF, IFNF, DKF, and fLDS reconstructions of the validation set (blue) and stochastic extrapolations into the future time steps (orange) for the true values orange is the hold out. (**bottom right**) kMSE of extrapolations using recovered latent state only. This demonstrate how well FNF disentangles hidden dynamics from embeddings. (**bottom left**) Rate of convergence of models are on par with each other, however FNF attains higher Bounds that could potentially translate into higher evidence values. 79

4.2 **Visualization of reconstructions and forward extrapolations for rotating MNIST images.** (**top**) 10 sequences of digits of test set being reconstructed and forward extrapolated (after solid yellow line). This demostrates how dynamics of rotation and latent representation of digits are separated since each digit has different rotaion value. Top row is true values and following that is respectively FNF, IFNF, and DKF. (**bottom left**) Inferred latent trajectory of digits for each model each color shows a different digit. 0 is blue and 1 is orange. (**bottom right**) ELBO of different models. 80

B.1 **Illustration of Construction of NN Training Data**. (Column 1) One template is randomly chosen from a previous sort and randomly scaled. This template is centered temporally. (Column 2) Another template may be randomly chosen, randomly scaled, and randomly shifted. (Column 3) A Gaussian noise sample (both temporally and spatially correlated) is drawn. (Column 4) A single spike sample is created by superimposing template 1 and noise. (Column 5) A collision sample is constructed by adding template 1, template 2, and noise. No more than 2 templates are used for creating collision samples. (Column 6) A misaligned sample is created by adding noise to a randomly shifted template. When the convolutional neural network is applied to the recording, our goal is to output no duplicate detection of the same spike. Thus, during training, samples such as column 4 and 5 have label 1 and samples such as column 3 and 6 have label 0. 90

B.2 Visualization of the architecture of Neural Network detector. ReLUs are used as activation functions. 92

Chapter 1: Introduction and Background

Computational neuroscience seeks to discover the underlying mechanisms by which neural activity is generated. This is predicated on acquiring sufficient data from a wide variety of experiments. With the recent advancement in neural data acquisition methods the bottleneck of this pursuit is the analysis of the ever-growing volume of neural data collected in numerous labs and from various experiments. This analysis can be broadly divided into two categories. First, extraction of high quality neuronal signals from noisy large scale recordings. Second, inference for statistical models aimed at explaining the neuronal signals and underlying processes that give rise to them.

Conventionally, majority of the methodologies in this endeavor is based on statistics and signal processing. However, in recent years Artificial Neural Networks (ANN) are gaining traction in neuroscience research due to their immense success in computer vision and natural language processing, and their demonstrated track record of generalizing to a variety of other problems. In this work we investigate and improve upon statistical and ANN machine learning methods applied to multi-electrode array recordings and inference for dynamical systems which play critical roles in computational neuroscience.

1.1 Spike Sorting

The analysis of large-scale multineuronal spike train data is crucial for current and future neuroscience research. However, this type of data is not available directly from recordings that require further processing to be converted into spike trains. Therefore, reliable and reproducible methods that can scale to the large volume of ever-growing recordings is integral to spike train analyses. Dense multi-electrode arrays (MEA) are standard devices for collecting such recordings. The processing needed to extract spike trains from these raw electrical signals done by "spike sorting" algorithms.

Currently widely-employed MEAs enable recordings at a scale of $\sim 10^3$ electrodes (Jun et al., 2017a), but efforts are underway to increase this number to 10^6 electrodes[1]. At this scale manual processing of the obtained data is inconceivable. Therefore, interest in automatic spike sorting for dense MEAs has surged over the last few years (Franke et al., 2010; Carlson et al., 2014; Kadir et al., 2014; Ekanadham et al., 2014; Muthmann et al., 2015; Pachitariu et al., 2016; Yger et al., 2016; Hilgen et al., 2017; Jun et al., 2017b). Despite these efforts, spike sorting remains a critical computational bottleneck in the scientific pipeline when using dense MEAs, due both to the high computational cost of the algorithms and the human time spent on manual postprocessing.

We introduce a robust and scalable MEA spike sorting pipeline `YASS` (Yet Another Spike Sorter) to address many challenges that are inherent in this task. In developing this pipeline we are guided by several principles. First, robustness is critical since little hand-tuning and post-processing is possible at the scale of modern MEA data. Second, scalability is crucial. To feasibly process the oncoming data deluge, we use parallel, computationally efficient algorithms based on effective data summarizations wherever possible and focus computational power on the "hard cases," using cheap fast methods to handle easy cases. Finally, prior information is leveraged as much as possible; we share information across neurons, electrodes, and experiments. The last principle, complements the prior two, by reducing needed computation for extracting signals from previously unobserved MEA data streams while reducing sensitivity to small variations in data.

To evaluate the resulting pipeline, we focus on MEA data collected from the primate retina. This preparation is a useful spike sorting testbed for several important reasons. First, the two-dimensional MEA used here matches the approximately two-dimensional substrate of the retinal ganglion layer. Second, receptive fields of well-characterized retinal ganglion cell (RGC) types (e.g., ON parasols, OFF midgets, etc.) are known to approximately tile the visual field, providing useful side information for scoring different spike sorting pipelines. Third, many RGCs have moderately high firing rates and often have significant axonal projections that overlap with each other spatially

[1]DARPA Neural Engineering System Design program BAA-16-09

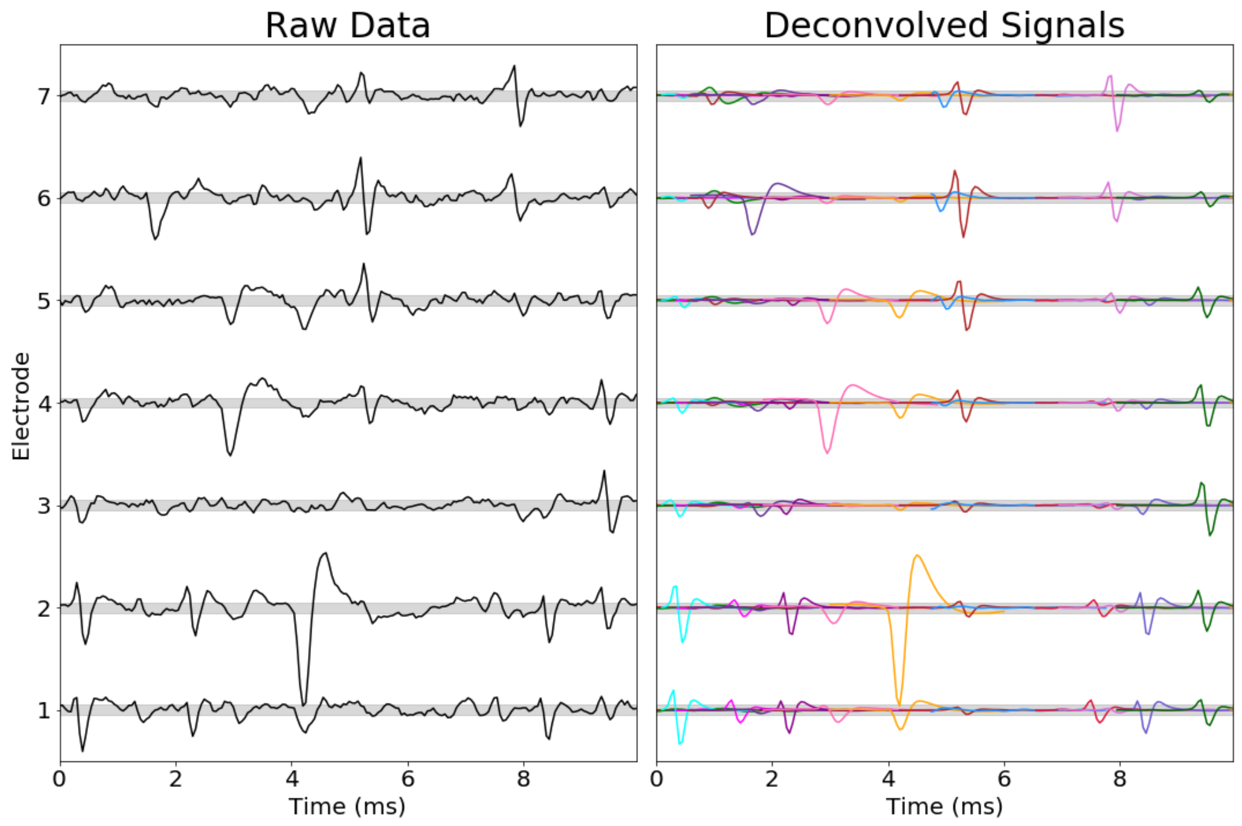


Figure 1.1: **High per-electrode firing rates and the prevalence of collisions in primate retina data**. (Left) voltage signal on 7 neighboring channels during a randomly selected 10 ms interval; (right) spike shapes and times identified and sorted by YASS (each unit is assigned a random color). The gray shade (here and in the following figures) represents ±1 estimated noise scale (estimated as the median absolute deviation of the raw data on each channel, divided by 0.67449). Note that some of the spikes visible on these electrodes are small but are larger on other electrodes (not shown); all units shown here have templates with peak-to-peak amplitude $> 4.5\times$ the estimated noise scale on at least one electrode. The total firing rate (summed over all visible units) on each channel here is in the hundreds of Hz range, leading to frequent collisions.

on the MEA, making it challenging to demix spikes that overlap spatially and temporally from different RGCs[2].

A typical spike sorting pipeline consists of three steps. The spike detection step extracts putative

[2]On the other hand, our focus on the primate retina means that in this work we will de-emphasize some issues that are critical in different parts of the nervous system: for example, firing rate dependent spike adaptation, or drifts and nonstationarities in the size and shape of spikes on the MEA (both of which are present in the retina but pose more serious challenges in in vivo recordings). Moreover, issues such as "glitches" due to behavior are absent in our in vitro retinal recordings. However, we believe that the pipeline developed here can be adapted to address these issues, and we plan to pursue these directions in future work.

spike events from noisy recordings. The clustering step groups similar spike waveforms into clusters, each representing a putative neuron. To resolve colliding waveforms, a deconvolution step is often performed. Spike clustering is at the core of the pipeline, as the clustering performance determines both the accuracy of the spike assignment and the quality of spike templates used for deconvolution. There are numerous methods proposed for these steps.

1.1.1 Detection

The most common detection algorithm selects events that cross a threshold, with alignment to the threshold crossing or a peak (Lewicki, 1998). Alternatively, `SpikeDetekt` (Kadir et al., 2014) used multiple thresholds and temporal and spatial adjacency to improve detection and aligned to the mean energy of a soft thresholded signal. `YASS` uses a neural network based method, which allows detecting low SNR spikes with small false positives. (Section 2.1.3)

1.1.2 Featurization and clustering

The most common approach for feature extraction is PCA (Abeles et al., 1977; Harris et al., 2000; Shoham et al., 2003). Wavelets are also used for feature extraction (Hulata et al., 2002; Quiroga et al., 2004). In Hilgen et al. (2017), Muthmann et al. (2015), and Jun et al. (2017b), spikes' locations and PCA based spike shape features are used for clustering. Hurwitz et al. (2019) further improves the quality of the location estimation using variational autoencoder.

The clustering stage of the spike sorting problem has been discussed in several reviews (Schmidt, 1984; Lewicki, 1998; Gibson et al., 2012; Rey et al., 2015). There are a number of approaches to the clustering step in the literature. Many of these approaches address the Gaussianity assumption on the shape of the clusters (Kadir et al., 2014; Carlson et al., 2014; Souza et al., 2019). Shoham et al. (2003) uses a mixture of t-distribution to account for its heavy tail. One of the most common alternative approaches to a probabilistic mixture model formulation for clustering is a density based approach. This has been done in the form of superparamagnetic clustering (Quiroga et al., 2004), consensus clustering (Fournier et al., 2016), uni-modal clustering (Magland et al., 2015; Chung et al., 2017), spectral clustering, finding density peaks (Rodriguez et al., 2014; Yger et al., 2018), and the mean shift algorithm (Hilgen et al., 2017). YASS uses PCA and Gaussian Mixture Models as a base. However, it recursively re-featurize and re-cluster to improve the clustering results (Section 2.1.7).

It is also worth mentioning the divide-and-conquer approach (Swindale et al., 2014). Instead of clustering all detected spikes together, they are first divided based on their primary channels (a channel with the largest amplitude) and then clustered independently. This is now the standard approach that most recent pipelines use.

1.1.3 Deconvolution

The deconvolution step begins with the assumption that voltage traces are decomposed into a superposition of spike waveforms (which are assigned to a single neuron) and background noise.Then, the deconvolution step tries to minimize the L_2 distance between the voltage traces and the estimated superposition of waveforms given the templates of neurons.The most common approach for this non-convex problem is a greedy method that deconvolves one spike at a time that reduces the L_2 distance the most. Binary Pursuit (Pillow et al., 2013) improves it further using a coordinate-descent approach. Given an initial estimate for a spike train, it improves the L_2 reconstruction error by exchanging spikes from one neuron to another. Continuous Basis Pursuit (Ekanadham et al., 2011) models each neuron's waveforms using a set of basis functions that can account for continuously shifted waveforms. This allows outputting continuous spike times. Both non-greedy and continuous-time spike deconvolution methods are adapted and improved upon in YASS (Section 2.1.9).

1.1.4 Iterative approach

There are also pipelines that implemented an iterative approach. Methods proposed in Pillow et al. (2013) and Ekanadham et al. (2014) initialize neuron waveforms using a simple detection and clustering method, and then iteratively run deconvolution and template updates. Whereas, Carlson et al. (2013) and Pachitariu et al. (2016) use an online method that updates templates as it deconvolves spikes. `YASS` also adapts the iterative approach. It iterates template estimation through clustering and deconvolution (by default, it only runs two iterations). Furthermore, `YASS` makes use of the post-deconvolution residuals to improve its re-clustering quality (Section 2.1.10).

1.1.5 Drift Correction

A number of methods address drift in electrode positions or change in the template waveforms over time. For instance, Dhawale et al. (2015) use a mixture of drifting t-distributions in combination with EM inference to deal with non-stationary templates. However, this method hardly scales to large, dense MEAs. This lack of scalability extends to other methods such as a mixture of Kalman filters (Calabrese et al., 2010) and first order autoregressive process on the mean of templates (Carlson et al., 2013). `JRClust` (Jun et al., 2017b) handles the drift in electrode positions by using spikes' locations and amplitudes as features. Then, the probe position is estimated every second and subtracted from spike locations to correct their positions. A recently developed Kilosort2 (Kilosort2 2019) also addresses drift correction. First, it allows time-varying templates. Secondly, it re-orders the recording in time such that the drift in electrode positions is as slow as possible. This idea works well with Neuropixel data (Jun et al., 2017a), where only one dimensional (vertical) probe drift is assumed. After partitioning data into two-second chunks, it computes a dissimilarity matrix of the chunks and re-orders them so that the diagonal of the matrix is the smallest. In retina MEA recording, where the electrodes are stay fixed, only a slow drift in waveform shapes is assumed. To handle this, `YASS` periodically updates templates every 5 minutes of chunks (Section 2.1.11).

1.1.6 Highlights of some of recent pipelines

Lastly, the key features of some of the recent pipelines are discussed. `Spyking Circus` (Yger et al., 2018) is an algorithm that aims to scale spike sorting to large MEA recordings by density estimation clustering (Rodriguez et al., 2014) of dimensionally reduced spikes by PCA, followed by template estimation and matching. However, PCA alone fails to featurize low SNR spikes properly and "undersplits" clusters. Collisions generally cause problems in PCA and clustering space (Ekanadham et al., 2014) and `Spyking Circus` does not address this problem. Furthermore, the assumption of spatial localization of spikes fails to "de-duplicate" spatially extended axonal units. In comparison, `YASS` recursively runs featurization and clustering, which helps to separate low SNR clusters. (Section 2.1.7) The spike denoising in `YASS` masks out collisions, improving the cluster quality (Section 2.1.4). By incorporating non-local features, `YASS` manages to separate locally indistinguishable units (Section 2.1.7). Additionally, by running a series of automatic post clustering processes, it removes duplicated or collided units (Section 2.1.8).

`JRClust` (Jun et al., 2017b) follows a similar method as `Spyking Circus` but it does not use deconvolution to infer collisions. Due to methodological similarity to `Spyking Circus`, it is expected that their performances are comparable. `JRClust` is further developed into `IronClust` (IronClust 2020) but no description is publicly available yet.

`Herding Spikes` (Hilgen et al., 2017; Muthmann et al., 2015) represents spikes using their spatial locations and the PCA based waveform shape features. However, this featurization is not robust enough to collisions, making it difficult to cluster low SNR spikes. Also, the type of featurization is clear for spatially extended axonal spikes.

`Kilosort` (Pachitariu et al., 2016) performs spike clustering and spike inference in an end-to-end fashion. Given an initial set of templates, it iteratively infers spikes using template matching and updates templates using discovered spikes. This method overcomes computational bottlenecks through several means. It applies greedy deconvolution while using a SVD based low-rank approximation for each template, therefore, significantly speeding up the template matching. In addition, majority of the computation is parallelized using Graphical Processing Units (GPUs). However, due to its lack of split-merge step, poor template initialization, and use of greedy deconvolution, `Kilosort` misses many neurons, especially the ones with low SNR. There are a few improvements on `Kilosort`'s deconvolution made by `YASS`. Instead of the greedy method, `YASS` runs coordinate-descent-like approach to avoid local optima. Also, `YASS` deconvolves spikes with continuous times while `Kilosort` rounds it to the recording's sampling rate. The continuous-time deconvolution significantly impacts the quality of deconvolution performance. For large spikes, even a small misalignment (less than the recording's sampling rate) can introduce a large residual error, which then often leads to many false positives (Section 2.1.9).

1.2 Imitation Learning for Multi-Electrode Array Spike Deconvolution

MEA spike deconvolution is a crucial part of any spike sorting pipeline. This step is necessary for recovering overlapping signals that are numerous. This problem arises under the assumption that voltage traces are a superimposition of neuron's spike waveforms and background electrical activity.

In Sparse Coding (SC), signals are reconstructed using a sparse linear combination of much smaller basis signals that are called dictionaries. Therefore, MEA spike deconvolution can be formulated as convolutional SC problem where waveforms take the role of dictionaries. In many spike sorting pipelines including YASS, however, MEA deconvolution does not seek to learn dictionaries. Instead, it merely aims to minimize the reconstruction error given a fixed set of neuron templates (i.e. dictionaries). Despite this simplification, the problem is a daunting one due to overlapping signals and its convolutional setting. The most computationally efficient models apply a greedy approach to this non-convex problem where templates are assigned to (and consequently subtracted from) the residual one at a time to obtain the highest reduction in local reconstruction error.

There are non-greedy methods based on heuristic search that generally improve performance with a downside. They come at a substantially higher computational cost due to searching the solution space more exhaustively. For instance, given an initial estimate of a candidate spike train, Binary Pursuit (Pillow et al., 2013) either subtracts spikes out or puts them back in the residual to achieve better local reconstruction errors. Continuous Basis Pursuit (Ekanadham et al., 2011), on the other hand, models each neuron's waveforms using a set of basis functions that can account for continuously shifted signals. This allows outputting continuous spike times. Song et al. (2018) develop convolutional versions of Orthogonal Matching Pursuit (OMP) and K-Singular

Vector Decomposition (KSVD) as more optimal dictionary learning methods. Despite its attempt at rendering them computationally efficient, these method do not scale to the current scale of MEA recordings.

In this work, we propose a solution to this MEA spike deconvolution that relies on the following observations. Greedy approach offers desirable computational efficiency while making errors along the way due to favoring short term gains over long term ones. Having an efficient yet non-trivial "spike checker" for greedy solutions leads to improved results. Also, because of stereotypical shape of neuron waveforms, virtually endless synthetic data can be created to train the spike checker. Therefore, we propose a 1D convolutional nerual network architecture as a spike checker during the iterations of greedy method. Inspired by imitation learning work of (Ross et al., 2010), we devise a methodology to obtain realistic training data for our spike checker.

1.3 Deep Generative Models and Neural Data

Since its introduction, Auto-Encoding Variational Bayes (AEVB) (Kingma et al., 2013) has become one of the pillars of deep generative models alongside Generative Adversarial Networks (GAN) (Goodfellow et al., 2016). In contrast to GANs, not only AEVB generates realistic data but it also provides insight into the mechanisms that give rise to data. This is why over the past few years, AEVB has been increasingly applied to a wide variety of data that had proved to be difficult for traditional unsupervised learning.

Dynamics and temporal data is no exception in this trend. Dynamical systems govern many real-world phenomena and temporal data that arise from them. In systems and computational neuroscience a major methodology of studying neuronal activity is from the temporal perspective. Single neuron voltage recordings are commonly modeled by a set of non-linear differential equations that are variants of the classical Hodgkin-Huxley model. On the other hand, high-dimensional neural population activity, that is assumed to be noisy and redundant observations of low-dimensional signals, is often modeled using state-space-models (Paninski et al., 2018). A considerable number of generative models alongside inference algorithms have been proposed for neural population modelling. Among these, are temporal structure models (Smith et al., 2003; Yu et al., 2009; Goris et al., 2014), switching dynamical structure models (Petreska et al., 2011; Linderman et al., 2017) and more recently ANN inspired generative models (Archer et al., 2016; Krishnan et al., 2016; Gao et al., 2016; Sussillo et al., 2016; Hernandez et al., 2018) that use AEVB (Kingma et al., 2013) framework to amortize the inference of posterior distributions of latent representations.

In this work we consider a family of state-space models expressed by the following generative process, where g, f are smooth differentiable functions, and $\Pi(\theta)$ is a noise distribution (e.g. Gaussian or Poisson respectively for continuous or discrete data) that is governed by parameters θ. The full joint distribution, denoted by $p(\mathbf{X}, \mathbf{Z})$, can be readily computed. This family subsumes well-know models such as *Linear Dynamical System* (LDS) and fLDS (Archer et al., 2016).

$$\begin{aligned} \mathbf{z}_1 &\sim \mathcal{N}(\mathbf{0}, Q_0) & & \\ \mathbf{z}_t|\mathbf{z}_{t-1} &\sim \mathcal{N}\left(g(\mathbf{z}_{t-1}), Q\right) & & t = 1, \ldots, T \\ \mathbf{x}_t|\mathbf{z}_t &\sim \Pi(\theta = f(\mathbf{z_t})) & & t = 1, \ldots, T \end{aligned}$$

The choice of approximation distribution family is a determining factor in the success of variational inference. Posterior distributions of dynamical systems are highly complex yet exhibit constrained correlation in temporal dimension. Normalizing flows(Rezende et al., 2015) are expressive deep density estimators well-suited as posterior approximation distributions. In this work we design two normalizing flows with meaningful temporal constraints. Also, we introduce an AEVB method to condition the parameters of the flows on data observations. This conditioning is done such that any latent state of the approximate posterior $\mathbf{z}_t$ depends on a sequence of observations $\mathbf{x}_{1..t+1}$ similar to particle filtering methods. Therefore, we name our method, *Filtering Normalizing Flows* (FNF).

1.3.1 Variational Deep Networks for Dynamics

Inference for state-space models that have underlying non-linearites is getting growing amounts of attention and a growing number of methods have been proposed for this task. Since the introduction of AEVB(Kingma et al., 2013) and its widespread success, many of these approaches have followed variational framework, and so does ours. The black box variational method that was introduced in (Archer et al., 2016) and extended to Poisson discrete observations in (Gao et al., 2016), known respectively as fLDS and PfLDS, consider a subset of the general class of latent dynamical systems with linear evolution in state space and nonlinear emissions. In other words, $\mathbf{z}_t|\mathbf{z}_{t-1} \sim \mathcal{N}(A\mathbf{z}_{t-1}, Q)$ governs latent state evolution. The variation approximation that fLDS uses is the result of mixing local independent Gaussian potentials $q_\phi(\mathbf{z_t}|\mathbf{x_t})$ with the prior of the model $p(\mathbf{z}_{1:T})$, the result of which is a Gaussian with a block-tri-diagonal precision matrix. Generalization of this method is offered by SVAE (Johnson et al., 2016), which leverages message passing algorithm for graphical models with conjugate priors along with AEVB inference framework to mix local neural network potentials with a structured prior distribution. The conjugacy requirement of PGM inference limits the type of generative models allowed by this method to LDS, switching LDS, etc.

fLDS method is expanded by VIND (Hernandez et al., 2018) to use time variant linear dynamics to approximate non-linear dynamics. To learn these locally linear parameters fixed point iteration method is used. However, this method uses a Gaussian approximation in non-linear dynamics regime, which can lead to underestimation of variance, subsequently affecting the fit of the model.

Deep Kalman Filter DKF and other structured inference networks (Krishnan et al., 2016) were proposed for inference for dynamical systems with non-linear transitions in the latent state space in the form of Gated Recurrent Units (GRU) with time variant noise. The variational posterior approximation is described by factorization $q_\phi(\mathbf{z}|\mathbf{x}) = \prod_{t=1}^{T} q(\mathbf{z}_t|\mathbf{z}_{t-1}, \mathbf{x}_{1:t})$ which is conditioned on the observations either using MLP or RNN functions to filter/smooth. Among limitation of this model is using time invariant diagonal Gaussian noise and the challenges of training RNN and how slowly they propagate information for smoothing/filtering purposes.

In a slightly different approach, (Le et al., 2018; Naesseth et al., 2018) optimize a tighter bound on evidence of the generative model. They achieve this by conditioning a Sequential Monte Carlo (SMC) sampler on observations to simultaneously estimate the partition function that gives a better bound on log-likelihood and amortizing the posterior inference. However, due to re-sampling step in SMC gradients cannot be reparameterized and biased gradient estimates are used for optimization that poses a variety of challenges. Also, this approach has the same problems as tighter lower bound approaches.

Finally, LFADS (Sussillo et al., 2016) proposes a sophisticated, bidirectional RNN architecture with neuroscience applications in mind, especially spike trains. The transition in this method functions are deterministic meaning that the underlying dynamics is not stochastic, instead, it is superimposed with noise that does not affect the evolution.

Chapter 2: YASS: Yet Another Spike Sorter

2.1 Methods

2.1.1 Overview

Our overall strategy can be considered a hybrid of a sparse deconvolution approach (specifically, an extension of the matching pursuit method used in (Pachitariu et al., 2016)) and a classical clustering approach, generalized and adapted to the large dense MEA setting. Our guiding philosophy is that it is essential to properly handle "collisions" between simultaneous spikes (Pillow et al., 2013; Ekanadham et al., 2014; Pachitariu et al., 2016), since collisions distort the extracted feature space and hinder clustering. (These collisions are quite prevalent in primate retinal data, as illustrated in Figure 1.1.) Sparse deconvolution approaches can potentially handle these collisions, but require good initializations of each neuron's "template" (average spike shape). Our approach therefore "triages" collided or noisy waveforms in the initial template estimation step, excluding these corrupted spikes from the feature extraction and clustering stages, and deferring recovery of such spikes to later deconvolution stages. This leads to significantly improved template estimates, and therefore more robust and accurate overall results.

Guided by this overall philosophy, our approach is a modular, multistage pipeline that includes the following main stages: (*i*) detection, denoising, and de-duplication of spike waveforms, (*ii*) division of the waveforms into distinct subsets for parallel processing, (*iii*) clustering the waveforms using adaptive featurization while triaging outliers and collided waveforms to obtain single neuron templates, and finally (*iv*) inferring missed and collided spikes via sparse deconvolution step. Pseudocode for the flow of the pipeline can be found in Algorithm TK. Below we provide a summary overview of each stage with the following subsections describing in detail each of the modules.

We start by band-pass filtering and standardizing the voltage signals using standard methods. For completeness, these steps are described in section 2.1.2.

The next stage is event or spike detection (section 2.1.3), implemented by a convolutional neural network (NN). The network is trained using simulated data that is designed to emulate the correlated noise and spike shapes observed in real data; no hand-labeling of individual spikes is needed.

Detected spike waveforms are then denoised using a network that is trained using a similar approach as the detection network (section 2.1.4). The denoising process also serves to suppress corruptions of spike shapes due to collisions[1].

The detection NN operates locally in space and time; spatially-extended axonal spikes may be detected in multiple locations. Therefore, we implement a spike de-duplication step before further processing (section 2.1.5).

Next we upsample and align spikes to the mean spike shape on each channel (section 2.1.6).

Next we perform a preliminary clustering of the denoised waveforms (section 2.1.7). Clustering this large and high-dimensional data is challenging; several main ideas are critical to make this clustering step tractable. First, we use a coarse-to-fine divide-and-conquer approach: instead of trying to run a single clustering algorithm on the full dataset, we iteratively split the data into smaller subsets (starting by examining each denoised waveform on its main channel — i.e., the electrode on which the spike is biggest — and then performing more splits on secondary channels) before clustering these smaller subsets of data. Second, we do not attempt to cluster the full high-dimensional waveforms; instead, we perform a featurization within each subset (adapting the chosen features to each subset) and cluster in the estimated low-dimensional feature space. Third, we triage outliers (caused mostly by collisions) that would otherwise lead to clustering errors. Finally, we apply a nonparametric Bayesian approach (similar to (Miller et al., 2018)) with stochastic variational split and merge methods to efficiently explore each clustering space (Hughes et al., 2013) and automatically estimate the number of clusters in each subset.

In practice, the divide-and-conquer approach described above can oversplit some units, and

[1]In a previous version of this work (published in NIPS 2017), we trained a network to triage collision events. We have found that it is more data-efficient to denoise instead of triage these collisions if possible.

sometimes forms small clusters that are dominated by collisions, instead of well-isolated single spikes. We developed some simple post-processing steps to handle both of these issues (section 2.1.8).

The cluster means or neuron templates yielded by the clustering stage are then used to infer the presence of all spikes (whether isolated or collided) across the entire dataset, using a deconvolution algorithm (section 2.1.9). We use a matching pursuit approach similar to that employed in Kilosort (Pachitariu et al., 2016), but with a few important modifications, including super-resolution temporal spike alignment and an iterative coordinate-descent (CD) step that helps correct errors made by the initial greedy matching pursuit pass.

Much of the variability in the shape of spikes in primate retinal recordings is due to collisions with other spikes. After the deconvolution step, for each spike, we can estimate the contributions of all the other inferred spikes, and regress these contributions away to effectively “clean" each observed spike. We then run additional split/merge steps on these post-deconvolution cleaned waveforms to update the templates (section 2.1.10).

To capture template scale changes during longer recordings, we implement a simple exponentially-weighted update algorithm, somewhat similar to the basic approach used in Kilosort2 (Pachitariu, 2019) (section 2.1.11).

Finally, low signal-to-noise-ratio (SNR) spikes from different cells can be very similar to each other; to properly handle uncertainty in the assignments of these spikes, we use a simple probabilistic model of the noise in the deconvolution step to compute soft assignments (section 2.1.12).

YASS is written in Python and CUDA; open source code can be found at https://github.com/paninski-lab/yass. The preprocessing, spike detection, denoising, and deconvolution steps are performed on temporal minibatches of data (parallelized over multiple GPUs, if available) in order to scale to large datasets; the other pipeline stages are parallelized over multiple CPU cores and operate on significantly reduced data representations to limit memory usage. The pipeline has been tested on PCs, multi-processor workstations, and on Amazon Web Services (AWS).

2.1.2 Preprocessing

The recording is first band-pass filtered using a Butterworth filter (300Hz-2KHz for the data analyzed here) then standardized by the Median Absolute Deviation (MAD).

2.1.3 Detection

Next we detect putative spike events. The optimal detection of spikes with unknown shapes and sizes, in the presence of collisions and spatiotemporally correlated noise, is a challenging statistical problem; simple thresholding is computationally straightforward but clearly statistically suboptimal. (Classical signal detection theory tells us that linear filtering followed by thresholding is the optimal approach for detecting a known signal shape in Gaussian noise — but here the signal shape is unknown and the “noise" is dominated by collisions and is highly non-Gaussian.) Typically an experimentalist will have very strong priors about what a “good spike" will look like, and a statistically-optimal approach will take advantage of this strong prior information — but it is not obvious how to encode these priors in a computationally-tractable detection algorithm.

Neural networks (NNs) have emerged as the current state-of-the-art method for translating large labeled datasets into highly-accurate classifiers or denoisers. However, we do not want to spend time hand-labeling many individual spikes here. Instead, we begin with a reasonable generative model that outputs short, small spatiotemporal snippets of voltage data, then sample at will from this generative model to provide as many training minibatches to the neural network as desired. When training converges, we have a trained NN that (locally) optimizes the classification accuracy under the generative model. Thus this approach is a method that translates the priors made explicit in a generative model into an optimal classifier that implicitly encodes these priors. See e.g. (Parthasarathy et al., 2017; Yoon et al., 2017; Weigert et al., 2018; Sun et al., 2018) for previous applications of similar ideas to image denoising and decoding problems.

Thus, given the availability of standard NN training routines, we have shifted most of the effort here onto the definition of the generative model. We use a straightforward convolutional model to generate the required spatiotemporal voltage snippets: we sample spike shapes from some library,

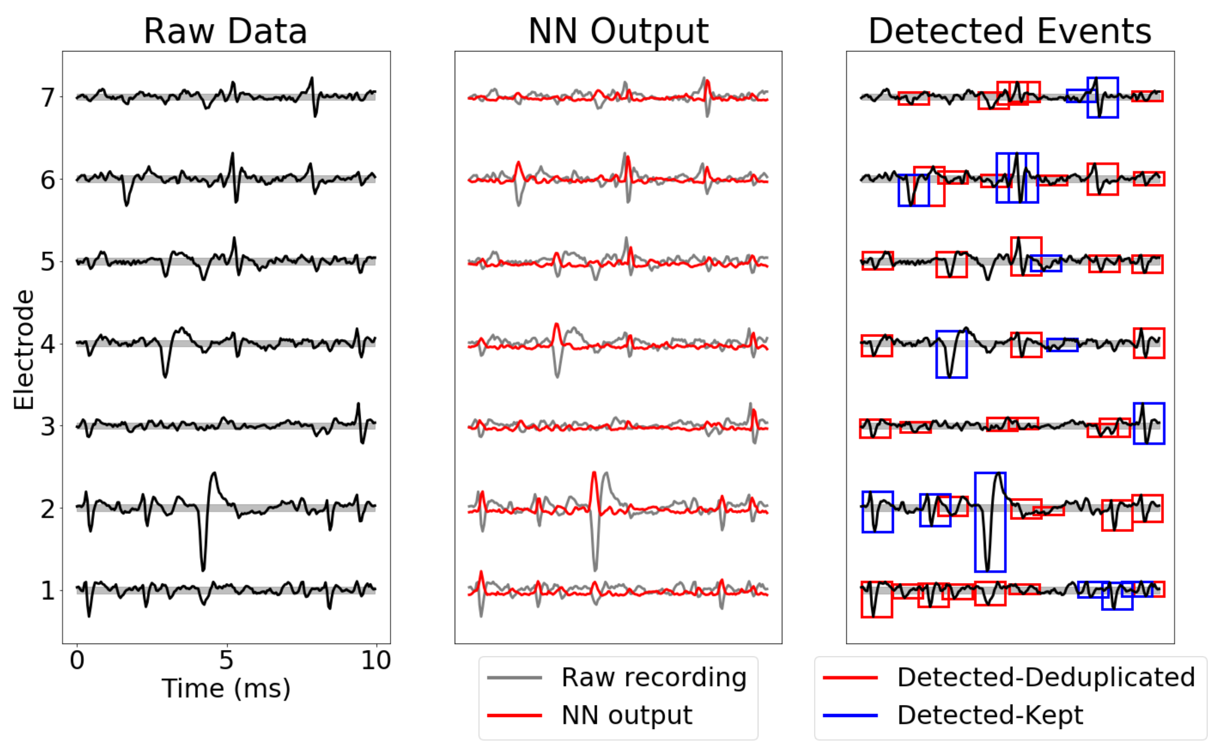


Figure 2.1: **Illustration of detection and de-duplication steps.** (**Left**) Raw recording on seven neighboring electrodes; (**Middle**) Neural Network (NN) output (red) overlaid on top of raw recording. (**Right**) detected and de-duplicated spikes. Red (blue) boxes denote spikes that are removed (retained) during the de-duplication step.

scale these shapes randomly, choose random times for the spikes, and linearly sum the result along with a sample from a stationary spatiotemporally-correlated noise process. We have found that a Gaussian process model for the noise suffices to obtain strong classification results; for the spike shape library, we simply collect templates from earlier successful sorts from the same preparation. Thus only very modest user input is required here, in the selection of “good" spike templates. See Figure 2.1 for an illustration, and the appendix for full details on the generative model and NN training.

2.1.4 Denoising

Single detected spikes often appear quite noisy; in the primate retinal preparation, much of this heavy-tailed “noise" is due to collisions with other spikes (recall Figure 1.1). It is useful to suppress this “noise" before further processing; the goal here is to input a noisy, potentially

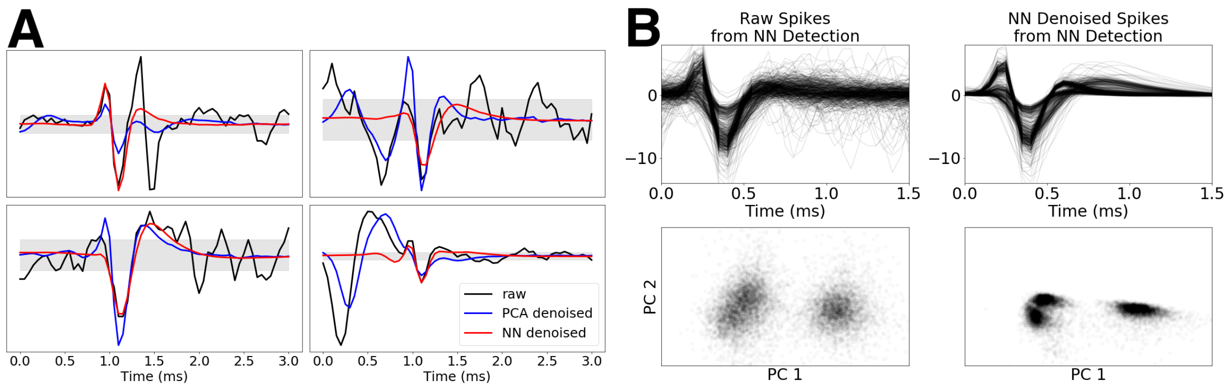


Figure 2.2: **Illustration of denoising step**. **(A)**: Four examples of raw detected spikes (black) and corresponding NN-denoised spikes (red) and PCA projections (blue). Note that the NN successfully suppresses both noise and contributions from other collided spikes, while the PCA projection is significantly more sensitive to collisions. **(B)**: (Top) raw detected spikes collected on a single electrode (left) vs. the same spikes after denoising (right). (Bottom): PCA projections corresponding to the raw (left) vs denoised spikes (right). Note that many of the outliers are no longer visible in the PCA projections, and the three clusters are much more easily distinguishable by eye.

collision-corrupted spike and output a “denoised" version of the spike. As in the detection context discussed above, simple linear denoising (e.g., projection onto a subspace inferred by principal or independent components analysis) is suboptimal here, due to the strongly non-Gaussian nature of collisions. To proceed, we use the same trick that we used for detection: we use the same generative model to create an arbitrarily large training set and then train a NN to solve this denoising task, i.e., take corrupted noisy spikes and output the original uncorrupted spike waveform. See Figure 2.2 for an illustration, and the appendix for full architecture and training details.

2.1.5 De-duplication of detected events

Our detection NN operates locally on just a few neighboring electrodes to detect spikes. In the datasets analyzed here, single spikes are often visible over hundreds of electrodes (see Figure 2.4 below for examples of some spatially-extended spike templates). Therefore single spikes may be detected simultaneously on many different electrodes, and before further processing we need to at least partially de-duplicate these multiple detected events.

Full de-duplication of single spikes, without knowledge of the full set of spike templates, is a statistically- and computationally-challenging problem that we do not attempt to solve. Instead, for each detected spike, we simply search locally in a spatiotemporal neighborhood for other smaller synchronous events, and drop any smaller events that we find. (Specifically, we use a window of $0.25ms$ and $150\mu m$ for this search, and compare the size of denoised events in terms of their peak-to-peak spike heights.) This de-duplication is fast but decidedly imperfect: it will kill duplicate detected events that come from the same cell but are smaller than the spike on the cell's “primary" channel (where the spike tends to be largest), as desired — but it will fail to de-duplicate very spatially extended spikes that extend outside of the spatial de-duplication window (false positives), and it will also kill spikes from smaller cells that happen to fire synchronously with the bigger cell (false negatives). We will correct the first issue with a template-deduplication step of the pipeline (section 2.1.8), and handle the second issue in the deconvolution step (section 2.1.9).

2.1.6 Alignment

Next we temporally align the de-duplicated waveforms. In practice, we find that super-resolution alignment (i.e., upsampling and then aligning waveforms) further helps reduce spike waveform variability, leading to better downstream clustering. Specifically, we use spline interpolation to upsample the raw waveform of each spike on its primary channel (to $5\times$ the acquisition rate), determine optimal shifts for each spike, and then use linear interpolation to apply this alignment to all secondary channels. The optimal shift is determined by maximizing the dot product of a shifted waveform with the average of all waveforms (separated by their primary channel).

2.1.7 Coarse-to-fine divide-and-conquer clustering with triaging and iterative re-featurization

At this point, for each electrode i, we have an array of detected spikes which are denoised, (partially) de-duplicated, aligned, and are largest on electrode i (i.e., we have grouped each spike by its primary electrode). Now, finally, we turn to clustering each of these datasets in parallel.

This clustering problem is challenging in several key respects. First, even after grouping detected spikes by their primary channel, the data remains very large: in the datasets considered here, we have to handle thousands of spikes per channel per minute, with tens of thousands of potential features per spike (since each spike can appear on any of the greater than > 500 channels, on ~ 60 timepoints per channel). Second, as emphasized above, the variability here is highly non-Gaussian, due largely to collisions; the NN denoiser suppresses many but not all collisions. Third, the number of clusters per channel is a priori unknown. Finally, while most cells can be separated using only local features from electrodes near the primary electrode, for some cells we do need to include features from more distant electrodes to achieve good cluster separation; see Figure 2.4 below.

Our approach combines several key ideas to address these challenges:

Divide-and-conquer. The first step is to try to divide the dataset into smaller subsets before applying any expensive clustering algorithms (Swindale et al., 2014). This “divide-and-conquer" approach is critical for both statistical and computational reasons. Naive clustering has a computational cost that scales superlinearly with both the number of datapoints and the dimensionality of the feature space. If we can split the data into subsets containing clearly different units, this would reduce both the number of datapoints per clustering call and the feature dimensionality needed to split the clusters; in turn, this dimensionality reduction can reduce overfitting given limited data. Finally, splitting the data improves parallelism; running more smaller jobs tends to be more scalable than running fewer bigger jobs.

Coarse-to-fine. Of course, all the above points are irrelevant if we can’t easily find ways to split the data. Luckily, for our problem, a coarse-to-fine strategy works quite well. We begin at the coarsest level with an extremely simple, one-dimensional featurization of the data: the size of the spike on the primary channel (as measured by the peak-to-peak height of the denoised spike), and perform a quick clustering using this feature, splitting the clearly distinct clusters into separate groups and leaving overlapping clusters in the same group. Next we incorporate “local" features (from neighbor electrodes to the primary electrode only) and perform further splits. Finally, at the “finest" level of the hierarchy, we incorporate features from more distant electrodes to perform a final set of splits.

Iterative re-featurization. The optimal featurization of the data might vary strongly from one subset of data to another; e.g., some clusters may be highly distinguishable based on the primary electrode, while for other clusters we might need to examine the secondary electrodes to achieve accurate splits. Therefore we do not start by reducing the dimensionality of the data (and then apply the same low-dimensional featurization to all the following splits); instead, we re-featurize after each split, i.e., compute a new dimensionality reduction of the data within each split subset. Figure 2.3 illustrates this process.

Outlier triaging. After dimensionality reduction, many outliers (largely due to collisions) are visible in the feature space. Instead of trying to assign these outliers to clusters (which leads in practice to unstable cluster shapes and over-splitting), we triage outliers — i.e., exclude them from the clustering step. (We will recover collision-corrupted spikes later in the pipeline during the deconvolution step described in section 2.1.9.) See the top-left panel of Figure 2.3 to see this triaging step at work.

Nonparametric Bayesian clustering. Finally, once we have chosen a good featurization and triaged outliers, we can choose a clustering algorithm. We suspect that there are many clustering algorithms that could work well at this stage; we have chosen a nonparametric Bayesian approach based on a “mixture of finite mixtures" (MFM) model Miller et al., 2018 that provides good results. Critically, the resulting algorithm outputs probabilistic estimates of cluster membership for each data point and is able to choose a reasonable number of clusters in a data-driven manner, without user input.

We discuss several of these issues in more detail in the following subsections; as usual, full mathematical and algorithmic details are provided in the appendix.

Algorithm 1 Pseudocode for the complete proposed pipeline.

```
1: Input: raw recording
2:
3: function PREPROCESS (SEC. 2.1.2)(raw recording)
4:     return standardized recording
5:
6: function SPIKE EXTRACTION(first batch of standardized recording)
7:     Detection (sec. 2.1.3)
8:     Denoising (sec. 2.1.4)
9:     De-duplication (sec. 2.1.5)
10:    return spike waveforms
11:
12: function INITIAL SORT(spike waveforms)
13:    Alignment (sec. 2.1.6)
14:    Clustering (sec. 2.1.7), Cluster Post-process (sec. 2.1.8)
15:    return templates
16:
17: function INITIAL DECONVOLUTION(templates, first batch of standardized recording)
18:    Deconvolution (sec. 2.1.9)
19:    return spike train
20:
21: function RE-SORT(spike train, templates, first batch of standardized recording)
22:    Alignment (sec. 2.1.6), Denoising (sec. 2.1.4)
23:    Post-deconvolution Reclustering (sec. 2.1.10), Cluster
       Post-process (sec. 2.1.8)
24:    Deconvolution (sec. 2.1.9), Post-deconvolution Merge (sec. 2.1.10)
25:    Soft Assignment (sec. 2.1.12)
26:    return templates, spike train
27:
28: function FINAL DECONVOLUTION(templates, remaining batches of standardized recording)
29:    Deconvolution with Drift Handling (sec. 2.1.11)
30:    Soft Assignment (sec. 2.1.12)
31:    return time-varying templates, spike train, soft assignment
32:
33: Output: time-varying templates, spike train, soft assignment
```

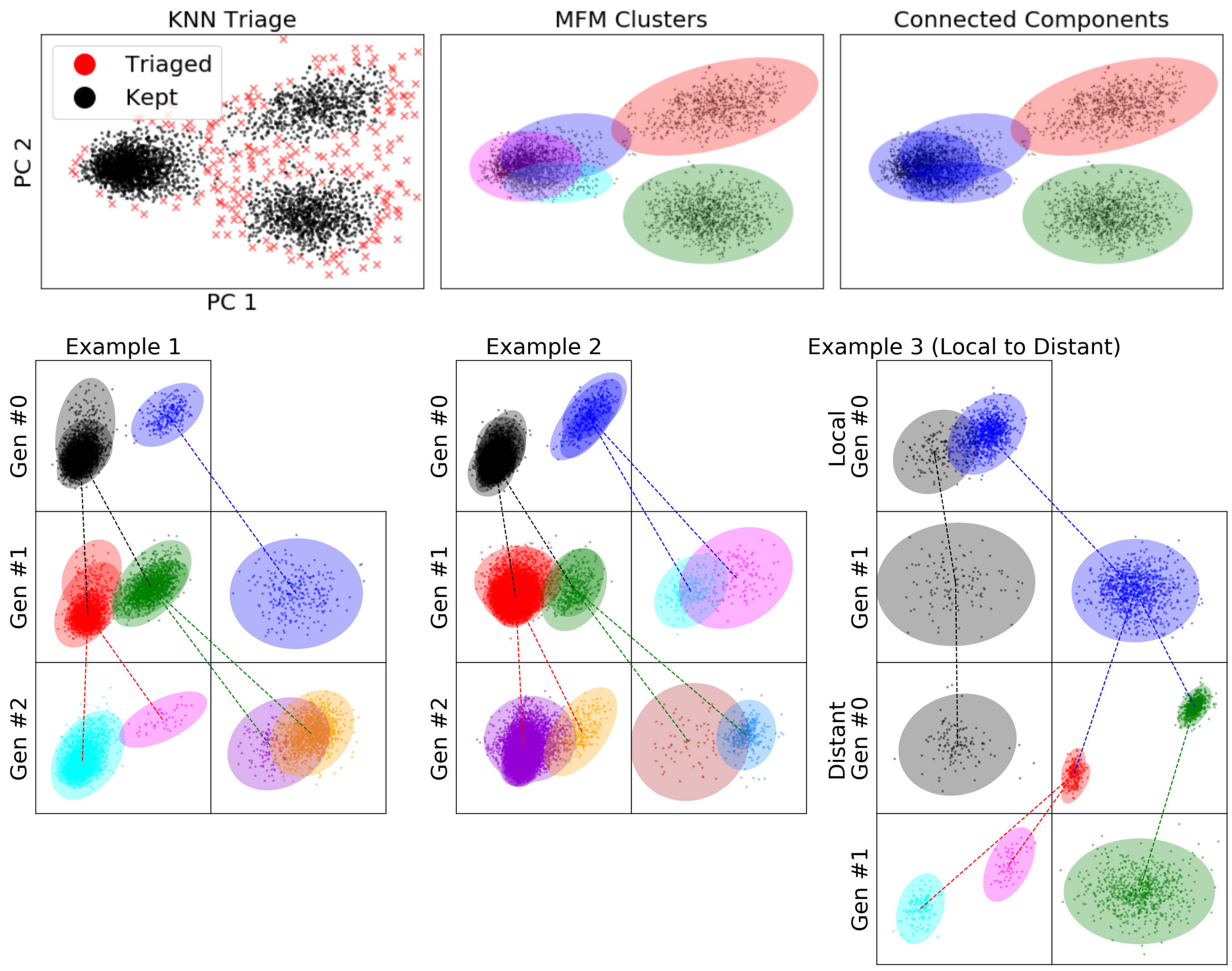


Figure 2.3: **Divide-and-conquer clustering with triaging and iterative re-featurization**. (Top left): Outlier triaging. Top middle: mixture-of-finite-mixture model (MFM) clustering of datapoints remaining after triage step. Each color represents a single cluster returned by MFM run in a higher-dimensional feature space. Top right: grouping of clusters with substantial overlap using connected components. (Bottom row): Three examples illustrating the benefits of iterative re-featurization. Each box represents a single clustering iteration. Ellipses represent MFM clusters and colors represent connected components (CCs). While the actual featurization is 5 dimensional, for visualization here these 5d vectors are projected on the 2 dimensional feature space that best separates the clusters (via linear discriminant analysis). In each "generation" we perform a triage, then an MFM clustering, then a connected-components analysis (as in the top row); then we re-featurize the data within each CC and iterate in the next generation. Colored lines indicate the flow of data in this procedure: a CC in one generation might generate multiple CC in the following generation, if the new featurization is informative enough to split clusters that were not separable in the previous generation. Indeed, as shown in Examples 1 and 2, many CCs can be more clearly split in the next generation by adaptive re-featurization. Example 3 illustrates the necessity of features from distant electrodes. In the first two rows, clustering uses features from local electrodes only, and the last two rows utilize features from distant electrodes. Although MFM returns a single blue cluster in generation 1 of local clustering, re-featurizing using distant features shows two clearly separable red and green CCs in generation 0 using distant features. See also Figure 2.4 below for an illustration of some units that are only separable via distant features.

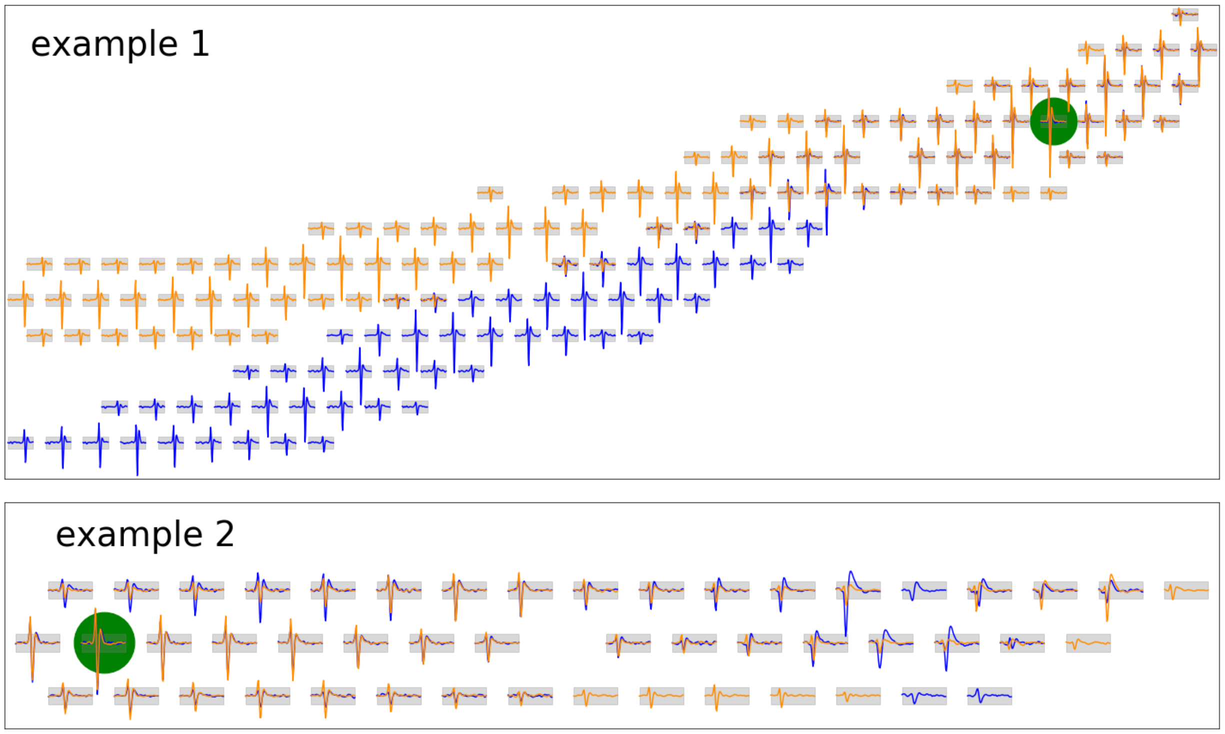


Figure 2.4: **Examples illustrating the need for distant-electrode featurization.** Two example pairs of units whose templates look locally similar but can be easily separated if information from all electrodes are used in the clustering process. In each example, the green dot represents the (shared) primary channel of the two templates indicated in blue and orange. Only “active" channels (where the template height is above a threshold) are shown.

Featurization for clustering using local or distant electrodes

During the “local" clustering phase of the coarse-to-fine approach, we start with denoised waveforms from the primary and nearest-neighbor electrodes. For hexagonal MEAs commonly used for retinal recordings, we considered 7 channels (i.e. the spike’s primary channel and the 6 neighboring channels), with 60 time points from each channel (3 ms for 20 kHz sampling rate), resulting in a vector with about 400 dimensions. At this point we randomly subsample at most 10,000 spikes on each channel, to limit the total memory consumption.

After splitting clusters using local features in the previous step, in the “distant" clustering phase we restrict attention to features from more distant electrodes. Here we need to restrict attention to electrodes which might provide useful information for additional splits — trying to include all time points from all electrodes in the feature vector would lead to poor results due to overfitting. First we drop any “inactive" electrodes for which the mean voltage for this group of spikes is below a threshold (about 1 standardized voltage unit, i.e., about the noise scale). Second, to find electrodes and time points where there is some significant variability that may be due to the presence of multiple clusters, we compute the MAD of the waveforms on these electrodes, select the voltages at any time points where these values are greater than a threshold (again on the order of one standardized voltage unit)[2], and append these voltages into the feature vector. The resulting feature dimensionality ranges from 5 to 80.

Finally, we take the feature vectors computed above and then apply PCA with 5 components to reduce the dimensionality.

Triaging

We use a simple, fast K-nearest-neighbors (K-NN) approach to detect outliers, following (Knox et al., 1998). When the feature dimensionality is small (recall here that our feature dimensionality is 5, following principal components projection), a kd-tree can find neighbors in $O(N \log N)$ average time. After computing the nearest neighbor graph we compute the average distance of each point to its five nearest neighbors (this can be thought of as a crude estimate of the local inverse density at each point), and triage out the top 1% datapoints with the largest average distance. This method effectively identifies outliers (Figure 2.3, top left panel).

Cluster stability and connected component grouping

After running the nonparametric Bayesian (MFM) clustering algorithm (described in full detail in the appendix) we want to form groups of similar clusters and then re-featurize and re-cluster within these groups.

We begin by computing the “stability" of each cluster i, defined as the average of the assignment probabilities p_{ij} that a point j is assigned to cluster i over all points j assigned with high probability

[2]A small technical note here; we do not denoise the distant channels, since the denoiser is trained on aligned spikes, and due to axonal propagation, spikes on distant channels may be poorly aligned. In the future, this issue could potentially be handled by training a denoising NN with more poorly-aligned training data, but we have not yet pursued this direction systematically.

to cluster i. Clusters that have a stability >0.90 are considered well isolated; we separate these clusters out and re-featurize and re-cluster their assigned spikes. This method is repeated recursively until MFM returns a single cluster; this is the termination point of the recursion.

In cases where none of the individual clusters returned by MFM has a stability >0.90, we group the pair of MFM-clusters that are closest (in Mahalanobis distance) and then recompute the stability metric on this group. We continue to do this until there is at least one group with >0.90 stability. See Figure 2.3, top, for an illustration of the resulting connected components.

2.1.8 Cluster post-processing

The clustering stage yields neuron templates, i.e. neuron shapes computed by averaging the aligned waveforms assigned to each cluster. Due to collisions, de-duplication errors, and oversplits due e.g. to spikes that are large on multiple channels (leading to variability in the assigned “primary channel" for these spikes), this first-pass estimate of templates typically requires some post-processing before we proceed with the deconvolution step. We outline two key post-processing steps below.

Removing collision units

Despite the denoising and triaging steps described above, some units extracted in the first clustering pass remain dominated by collisions. We can detect many of these units in two steps.

First, recall that all spikes are aligned on their primary channel prior to the clustering stage. Therefore, if a template on its primary channel is significantly off-centered compared other templates, then it is considered as a collision unit and removed.

Second, we could look directly for “collision templates" — i.e., templates that can be well-modeled as a weighted superposition of two or more other templates. However, we found that this strategy was ineffective, since small timing variability between the two colliding spikes can quickly lead to distortions in the shape of the mean of these collided spikes. Instead we found that it was more productive to examine the variability of the spikes in each unit, rather than just the mean:

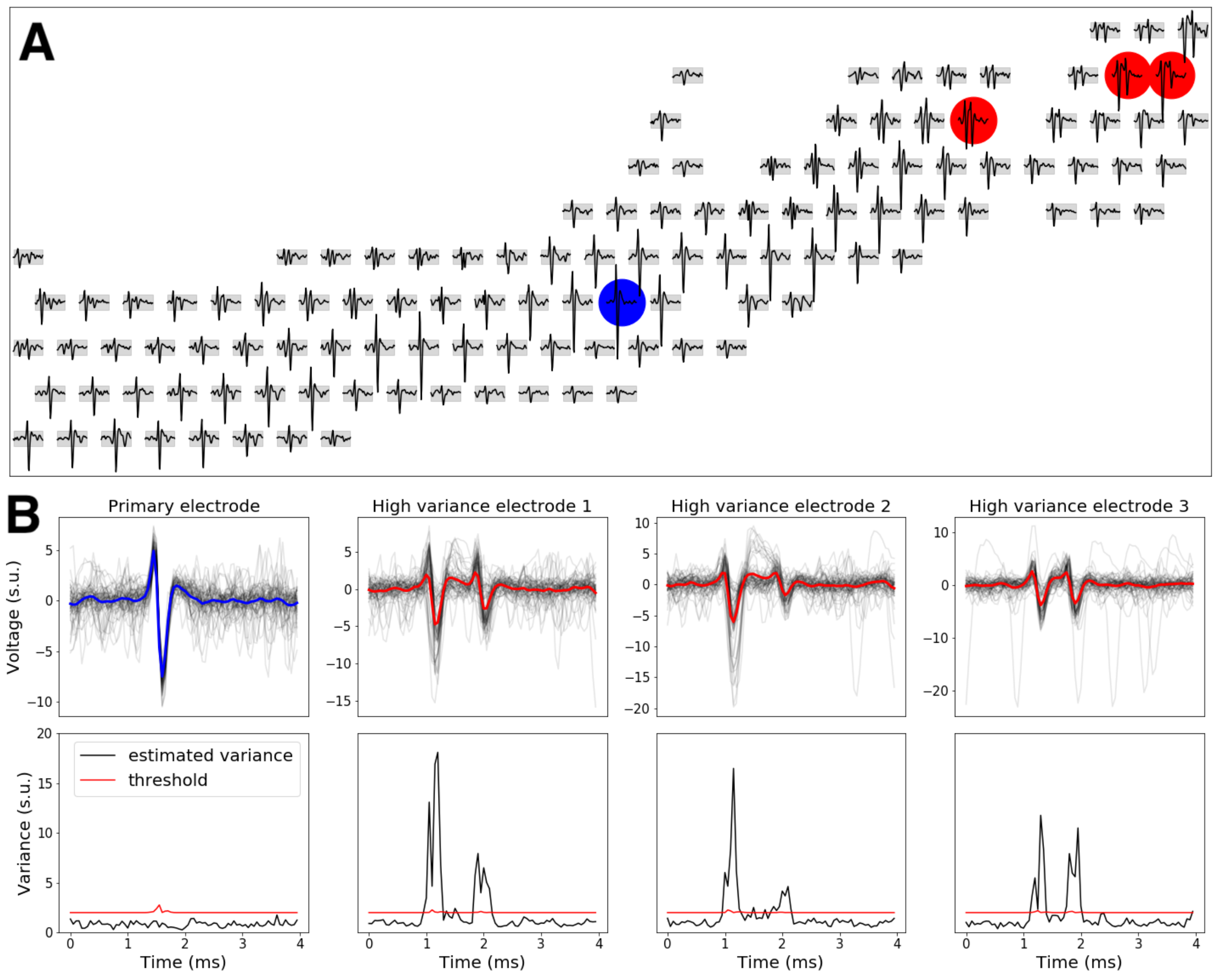


Figure 2.5: **An example collision unit**. (**A**) The template of the collision unit. Blue dot indicates the primary electrode (with the largest template), and red dots indicate the top 3 electrodes with the highest variability. (**B**) Top: raw spikes (gray traces) and templates (color) at the primary electrode and the three most-variable electrodes. Bottom: the estimated variance of spikes (black) and the threshold (red) used to detect overly-variable units. Due to small differences in the timing of one collided spike relative to the other, the variability of collided waveforms tends to be significantly elevated.

small shifts in the timing of one spike relative to the other in a collision can lead to significant differences between waveforms, leading to large variability that we can detect reliably. We model spikes as the superposition of a template and a background noise term; thus, the covariance structure of a temporally isolated[3] non-collided spike around its template is assumed to be the same as the

[3]We restrict attention to temporally isolated spikes here to avoid burst-dependent changes in spike height.

covariance of the background noise. In addition, small misalignments of the spike (at resolution below the voltage sampling frequency) can also increase the variance. To address the issue, the maximal additional variance due to misalignment is estimated and incorporated in the variance bounds. This extra variance due to misalignment is estimated numerically by uniformly jittering the template with $(-1, 1)$ time step shift. Thus, we can search for units with higher-than-expected variability; we find that many of these units are clear collision units and can be discarded. See Figure 2.5 for an illustration.

Template de-duplication

We find pairs of duplicate templates by computing the maximum absolute difference (over time points and channels) between the aligned templates. Both absolute and relative differences are used to measure the closeness of two templates; if the maximum absolute difference is less than about 1 (absolute scale) or 10% of the size of the bigger template, we conclude that the templates match. (For low-firing-rate units, we compute templates using denoised spikes, since with few spikes the raw template may be corrupted by outliers due to collisions.)

Once a list of pairs of duplicate units are determined, the unit with more spikes is kept and the other is discarded, since the unit with more spikes will typically have a cleaner template.

2.1.9 Deconvolution

Once stable neuron templates are obtained, we run a deconvolution step on the raw voltage data to infer the spike times corresponding to these templates. This step is able to recover spikes that were triaged (due to collisions) or killed in the de-duplication step described above. A number of deconvolution approaches have been described in the literature (Franke et al., 2010; Carlson et al., 2013; Pillow et al., 2013; Ekanadham et al., 2014). Our starting point is the matching pursuit approach used in (Pachitariu et al., 2016); this is a greedy method that essentially subtracts spike shapes from the raw data one at a time until a convergence criterion is reached. We improve this method in three key directions described below.

Low-rank-plus-shift model for templates

(Pachitariu, 2019) use a simple low rank model for templates (where the template is represented as a channel-by-time matrix). This leads to faster computations (since the deconvolution time is dominated by convolution calls, and the number of required convolutions can be reduced by exploiting this low-rank template model) and can also reduce some noise in the templates, if the templates can be well-approximated by a low rank matrix.

Unfortunately, we found that this simple low-rank approximation is highly inaccurate for units with long axons, for which the spike at the soma can be significantly shifted in time compared to the spikes observed on axonal channels. In addition, as discussed above, many raw waveforms are corrupted by collisions, and therefore simple averaging of these raw waveforms can lead to noisy templates, leading downstream to noisier residuals in the matching pursuit algorithm and worse deconvolution results overall.

Instead we found that performing an alignment step (temporally aligning the templates on each channel) before computing the low-rank factorization led to significantly improved results (Figure 2.6). The goal of the alignment is to increase co-linearity of template matrix rows, therefore reducing SVD residual errors. To this end, for each neuron we minimize ℓ_2 distances between the template on each channel or its negative (whichever leads to a smaller distance) and the primary channel template. In addition, we clean the waveforms (see section 2.1.10 below) before averaging to estimate templates, and then force the absolute value of the template to smoothly decrease monotonically to zero outside of a 2 millisecond window; both of these steps help reduce outliers prior to the align then SVD steps. Finally, we compute templates using isolated spikes (with interspike intervals greater than the width of the template), to avoid edge artifacts due to contributions from burst spikes.

Spline interpolation within deconvolution

In matching pursuit, first we identify putative spike times from a given cell and then we subtract away the corresponding template from the raw voltage trace. For large spikes, small timing differences between the spike and the corresponding template can lead to large residual errors if we naively subtract the template away; significant residual errors can accumulate even if these timing differences are below the acquisition sampling rate (20 KHz for the retinal data analyzed here).

This subsample-resolution subtraction issue has been previously addressed, e.g., in (Ekanadham et al., 2014), who introduced an iteratively-reweighted L1 minimization approach. We implemented a simpler, more computationally efficient B-spline method for finding the optimal spike times at subsample resolution, then interpolating the template accordingly, and subtracting out this interpolated template. These steps are performed on GPU, leading to negligible speed differences compared to naive subtraction.

Figure 2.7 shows examples of the post-deconvolution residuals from large amplitude spikes. The interpolation approach leads to a substantially lower residual compared to naive greedy subtraction, and can thus yield substantially lower residuals, reducing deconvolution errors in the following iterations.

Reducing greedy deconvolution error through Iterative Coordinate Descent (ICD)

Matching pursuit is a greedy method: it subtracts spikes away one at a time, and never “reverses course": e.g., it might become visually clear after subtracting more spikes away that an early spike was subtracted mistakenly, but simple greedy matching pursuit is unable to correct these errors. Thus it is natural to incorporate “reversal" steps into the algorithm — i.e., instead of always subtracting spikes away, we could also propose adding previously-removed spikes back in, and perhaps subtracting different spike times away in a second pass through the data. This combination of subtraction and addition steps is a well-known approach in the sparse regression literature (Zhang, 2011).

To make this idea slightly more concrete, matching pursuit can be cast as a greedy optimization of a squared-error objective function, where we are optimizing over the space of binary matrices indicating the presence or absence of a spike from each cell in each time bin. One simple way to improve on matching pursuit is to replace this greedy optimization with iterative coordinate descent

(ICD) on the elements of this binary spiking matrix, using the same squared-error objective function. To implement ICD here we simply augment matching pursuit with steps for changing spike times or adding spikes back in if they have been subtracted away previously. These steps can be easily incorporated into the existing matching pursuit GPU code; in practice we just end up taking an additional couple passes through the data in the deconvolution step.

While ICD does not correct all false negative errors, it decreases them in almost all cases relative to greedy deconvolution, in some case by an order of magnitude (Figure 2.8A). Empirically, we find that this ICD approach can increase true-positive rates by a few % for many units, while significantly decreasing false-negatives for many neurons.

2.1.10 Post-deconvolution merging and splitting

The deconvolution step can recover spikes that were dropped in the triage or de-duplication step. In addition, much of the “noise" we observe in each spike waveform is in fact due to collisions with other spikes; once we have determined these other spike times we can subtract out the corresponding spike shapes and therefore decrease the effective noise level. After this “spike waveform cleaning" step, in turn, some new clusters that were not visible before this “cleaning" step can become easily separable. Therefore we re-run the clustering step on the “cleaned" waveforms, re-splitting and merging as necessary to improve the overall cluster assignments. We describe each of these steps below.

Post-deconvolution cleaned spike waveforms

As a first step we need to clearly define these cleaned waveforms, and then describe efficient methods for computing these objects.

The deconvolution step returns spike times for each cell, along with the residual (i.e., the original raw voltages, minus the contribution of each spike that was subtracted away during deconvolution). (Ideally, if we are capturing all the spikes, this residual will resemble noise, in which no clearly discernible spikes remain.)

The i-th “cleaned" spike waveform is defined as the i-th spike waveform minus the contribution of all the other spikes with indices $j \neq i$. To compute this efficiently, we can simply rewrite this as the residual (which subtracts all of the spikes out, including i) plus the template for spike i added back in. Thus we can compute each cleaned waveform rapidly, in an embarrassingly parallel manner over all the spikes. See the top row of Figure 2.9 for an illustration.

Post-deconvolution reclustering

We do not expect the original clustering step described above to split all the cells perfectly: due to the inherent noisiness of the raw data, the high collision rates, and the stochasticity of our clustering algorithm (which includes some randomized split-merge moves), some neuron templates (albeit usually a small minority) can be missed, incorrectly merged, or split by our pipeline.

After deconvolution and post-deconvolution cleaning, we have more spikes than what was used in the original clustering step (because now we have recovered spikes that may have been triaged or killed during the first pass), and the cleaned spikes show significantly less variability than the raw data (Figure 2.9, lower panels). Therefore, a re-clustering step run on this post-deconvolution data has the potential to correct some errors that may have been made in the first pass.

The deconvolution step has already grouped spikes according to the unit each waveform was assigned to. We run our clustering pipeline individually on each of these groups. Because most of these spike assignments from the deconvolution step are already fairly accurate, this reclustering step tends to be significantly faster than the original per-channel clustering run; in most cases this re-clustering step merely reconfirms the original template identity, with occasional splits identifying additional neurons. In addition, because most outliers are due to collisions, and the deconvolution step has already resolved most collisions, we found that it is unnecessary to perform triaging during this re-cluster step.

Merging units using cleaned spikes

As in the first clustering pass, it is necessary to perform a final merge step following the re-cluster step to try to fix any oversplits that may have occurred. Here we use an approach based on linear discriminant analysis (LDA) applied to the cleaned waveforms.

We start by computing a candidate list of pairs of units to be considered for merging; any pair of units with aligned templates that are sufficiently close in ℓ_2 distance are added to this list. For a candidate pair of templates, we then select a subset of spikes (usually up to 2,000), compute cleaned spikes, and whiten the resulting waveforms. To avoid overfitting, we compute a separable (Kronecker) approximation to the whitening matrix (as in (Pillow et al., 2013; Ekanadham et al., 2014). Next, we project the whitened cleaned spikes onto the difference of the two templates (restricted to the union of the active channels of both units). (This projection is equivalent to LDA if the whitening matrix is a good approximation of the inverse square root of the covariance of both units.) If the marginal distribution of the resulting one-dimensional data is approximately unimodal (according to a Hartigan dip test (Hartigan et al., 1985)), we merge the units; see **mountain-sort** for a related approach but applied to raw voltage data. (We merge pairs greedily, merging pairs with smallest template distance first.) Figure 2.10 visualizes the procedure for two pairs of neurons and demonstrates why it is beneficial to use cleaned spikes as opposed to raw spikes in the analysis.

2.1.11 Drift handling

In the retinal recordings considered here we observe spike amplitude changes on the order of 1% (or less) of PTP height per minute (See Fig 2.11). We correct for this drift by implementing a model with two stages: (1) we obtain neuron templates from the beginning of a recording (first 5 minutes); and (2) update the templates periodically after every new batch of deconvolved data (in batches of size 5 minutes). (See Pachitariu, 2019 for a related method.)

To update the templates we exploit the low-rank model described in section 2.1.9: i.e., instead of updating the raw templates (which are specified by many parameters, leading to potential overfitting in units with low spike counts) we update the parameters of the low-rank-plus-shift model directly, using exponential weighted updates. Furthermore, in these recordings we found that templates change largely in amplitude rather than shape (See Fig 2.1.11, A-C). Accordingly, in each channel we compute the updates by scaling the starting templates based on the weighted average of the spikes in the previous deconvolved batch and the following batch. Due to the high collision rate present in raw spikes, we base the update solely on cleaned spikes falling within +/- 10-20% of the starting template (see red scatter plot in Fig 2.1.11 D,E). While this limits the rate of change in the drift model, we found that it accurately tracks the change observed in our MEA recordings.

To add templates of intermittently-active neurons that might not have been active in the the initial batches of data, a simple split step is performed in each new batch. For each unit, the PTPs of cleaned spikes are computed in each channel and the uni-modality (Hartigan et al., 1985) is tested on a one-dimensional PCA projection of these PTP values. If the diptest p-value is below a threshold, the unit is split using a two-cluster EM algorithm. The same method is applied recursively to ensure the uni-modality of all clusters. To reduce oversplits, the split units are accepted only if their firing rate is bigger than 1Hz or its ratio to the original unit is bigger than 0.15. Afterwards, the deconvolution step is re-run to update the obtained spikes, and any duplicate units and low firing (less than 0.2Hz) units are removed. The same split step is run on the initial batch also and all the split are accepted to make sure that there is no missing or merged unit to begin with. Also, the templates are recomputed using the cleaned waveforms so that the templates' PTPs align with their deconvolved spikes' PTP.

After completing a forward pass through all batches (adding in new templates as needed), we perform a final backwards deconvolution (to properly assign spikes to units that might have only been identified in the middle of the recording), updating the templates as we pass through batches in reverse order, but not performing any additional split steps.

2.1.12 Soft assignment

For lower-SNR units, there will invariably be some borderline spikes for which the correct assignment is unclear. In addition, even high-SNR units may contain some outliers (e.g., due to deconvolution or clustering errors). Therefore, as a final postprocessing step we use the cleaned waveforms to compute three soft assignment scores:

1. Soft noise assignment: for the smallest units (according to the ℓ_2 norm of the template, restricted to the unit's active channels), we run the detection neural network on the cleaned waveforms and record the output scores; these can be used downstream by the user to perform a soft-classification into noise versus signal waveforms.

2. Outlier scoring: for all units, we compute the ℓ_2 norm of the whitened cleaned waveforms (using the approximate Kronecker whitener described in section 2.1.10); this score can be used to soft-classify outliers for each unit.

3. Soft pairwise assignment: for the pairs of units that are closest (according to the ℓ_2 norm of the difference of the aligned templates, restricted to the union of the two units' active channels), we compute the LDA projection (again as in section 2.1.10), estimate the density of each unit in the resulting one-dimensional projected space, and compute the likelihood ratio. For each pair, if one unit has an average soft assignment score below a threshold, we consider the unit a duplicate and remove it.

2.1.13 Synthetic and semi-synthetic data

For testing purposes it is invaluable to run spike sorting pipelines on both fully-simulated and “hybrid" simulated+real datasets in which ground truth is either fully or partially available. For rapid testing purposes, we generated simulated datasets on 49 channels. We estimated templates from a held-out dataset, and then applied randomly sampled scale factors to these templates. For fully-simulated datasets, we selected about 150 templates and generated spatiotemporally correlated noise as the background signal (this noise was in turn generated by convolving many independent white noise traces with randomly chosen templates, then scaling and averaging the resulting spatiotemporal noise together). For “hybrid" datasets, following (Rossant et al., 2015; Pachitariu et al., 2016), we chose about 20 templates and added spikes with these template shapes to a separate real dataset which served as the “background" signal. Next we randomly sampled firing rates uniformly on $(1, 30)$ Hz, sampled Poisson spike trains with these rates, and added the selected templates (with sub-sample jitter and interpolation) at the selected times.

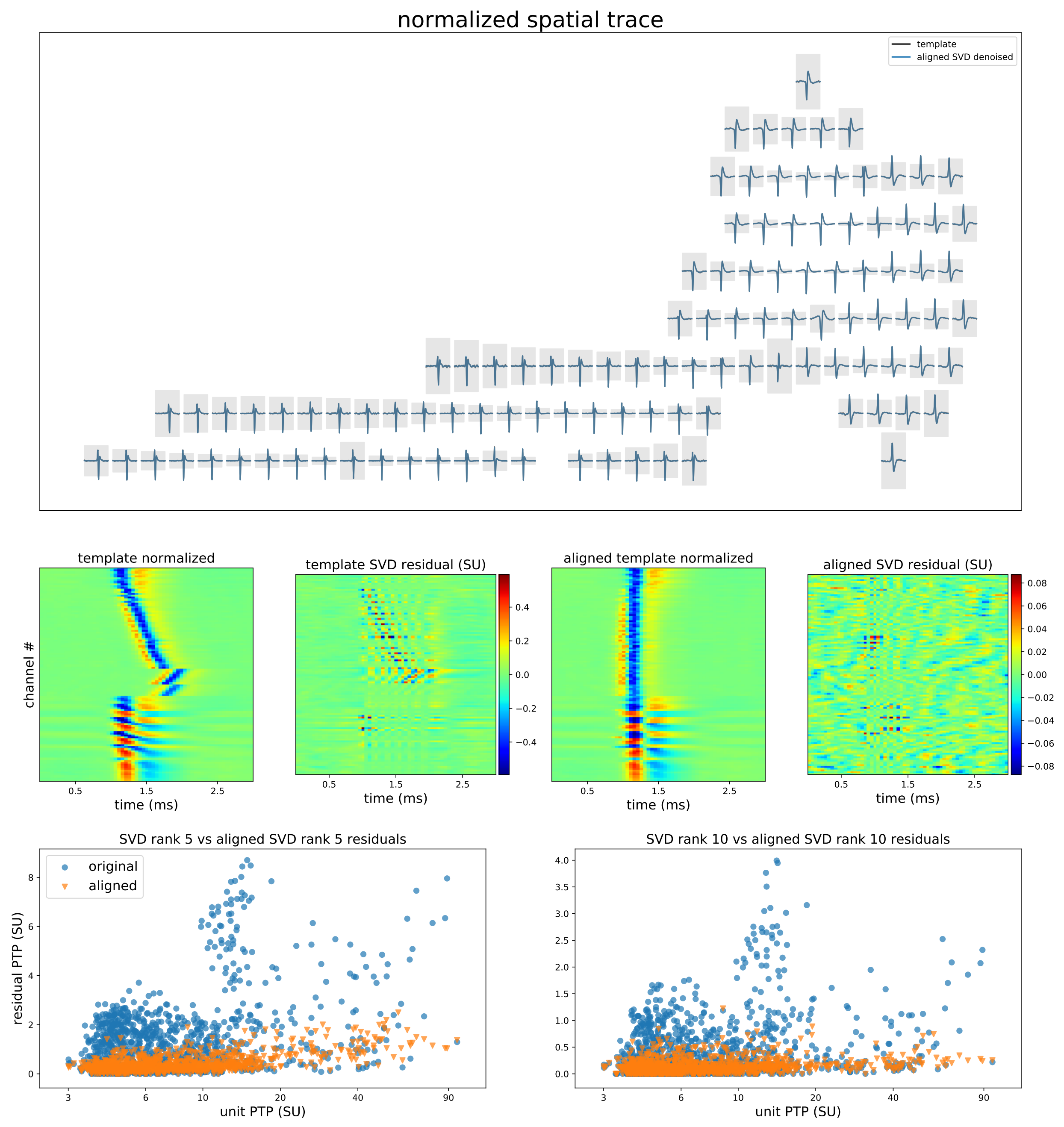


Figure 2.6: **Temporally aligned low-rank template model.** (**Top**) An example unit, showing that the raw template is well-summarized by its low-rank SVD reconstruction following temporal alignment, i.e., the blue and black traces show a high degree of overlap. Template plot conventions as in Figure 2.5. (**Middle row**) Comparing the SVD residual with versus without alignment. First panel shows the raw template, normalized per channel; note that the peak time varies by about a millisecond across the different channels. Second panel shows that a low-rank SVD approximation leaves behind significant residual structure. Third panel shows aligned normalized template, and fourth panel shows that aligned SVD residual is significantly smaller than unaligned. (**Bottom**) Summary of residual errors across all templates in one dataset. For both rank 5 and 10 SVD approximations, aligning leads to significantly reduced residual error.

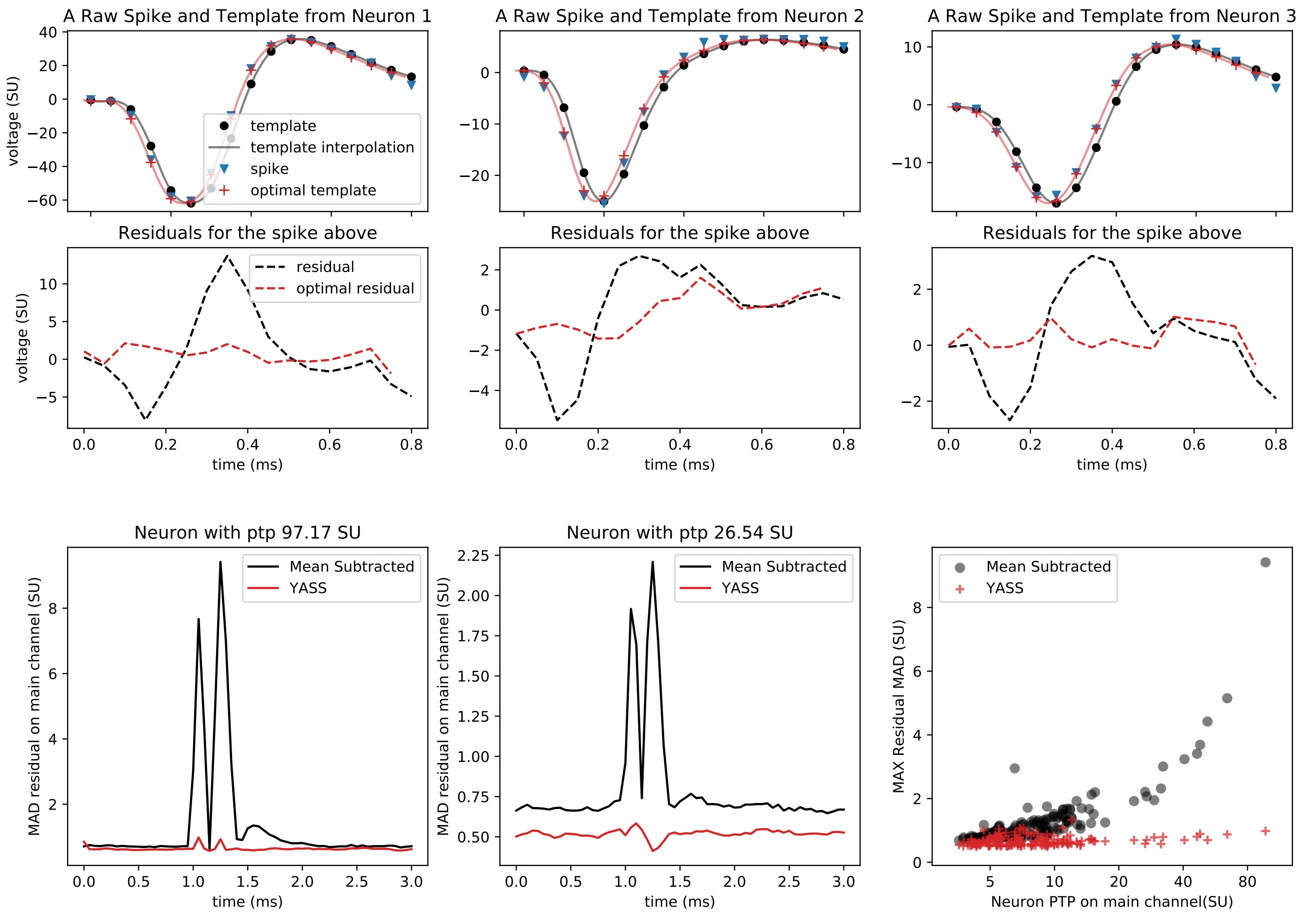


Figure 2.7: **Super-resolution interpolation and subtraction improves deconvolution.** (**Top row**) Three raw spikes from three distinct neurons are shown on the main channel of their respective neurons alongside neuron templates, their interpolations, and optimally time shifted and sub-sampled templates to achieve the smallest residual. Discrete signals are represented by scatter plot while interpolations are solid lines. (**Second row**) Residuals are shown when templates and optimal templates are subtracted from the raw spikes above. In all these cases the raw residuals have significant amplitude and can be mistaken as spikes from other neurons, while the optimally-aligned residuals are significantly smaller. (**Third row left & middle**) MAD of post-deconvolution residuals (in standardized voltage units) for spikes of two large neurons (“ptp" abbreviates the maximal peak-to-peak height of the spike template). Naive subtraction leaves significant residual variability behind (black traces); conversely, residuals are substantially smaller when super-resolution alignment and interpolation of the template prior are applied prior to subtraction (red trace). (**Third row right**) Residual variability using naive subtraction (black) scales with unit PTP, but with interpolation the residual error stays uniformly low (red; each symbol corresponds to a single cell).

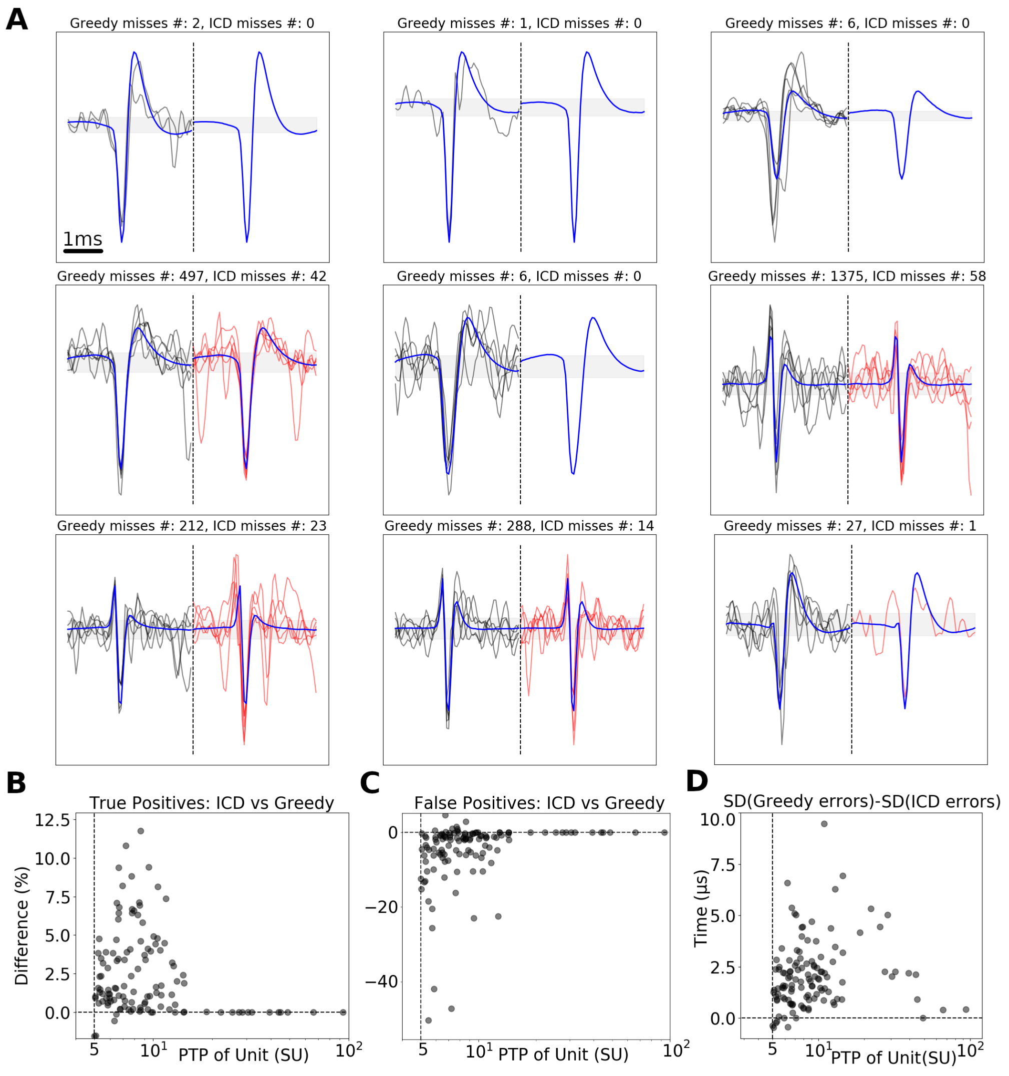


Figure 2.8: **Comparing greedy vs iterative coordinate descent (ICD) deconvolution.** **(A)**: Examples of spikes recovered by ICD but missed by greedy deconvolution (and vice versa). All comparisons here performed on synthetic data with known ground truth spike times (see section 2.1.13 for a description of synthetic data generation). Blue indicates the true underlying (noiseless) spike template; black is an example of an observed noisy voltage trace containing the spike that greedy deconvolution missed but ICD caught; red indicates traces for which ICD missed the spike but greedy deconvolution accurately captured the spike (up to a maximum of 5 traces are plotted). Note that the miss rate is often much lower for ICD. **(B-D)**: ICD improves true positive rates on the order of a few % **(B)**, decreases false positive rates by up to 50% **(C)**, and modestly improves the timing precision of detected spikes, as measured by the standard deviation (SD) of the inferred spike times **(D)**. In each case, differences between greedy versus ICD are most significant for smaller units.

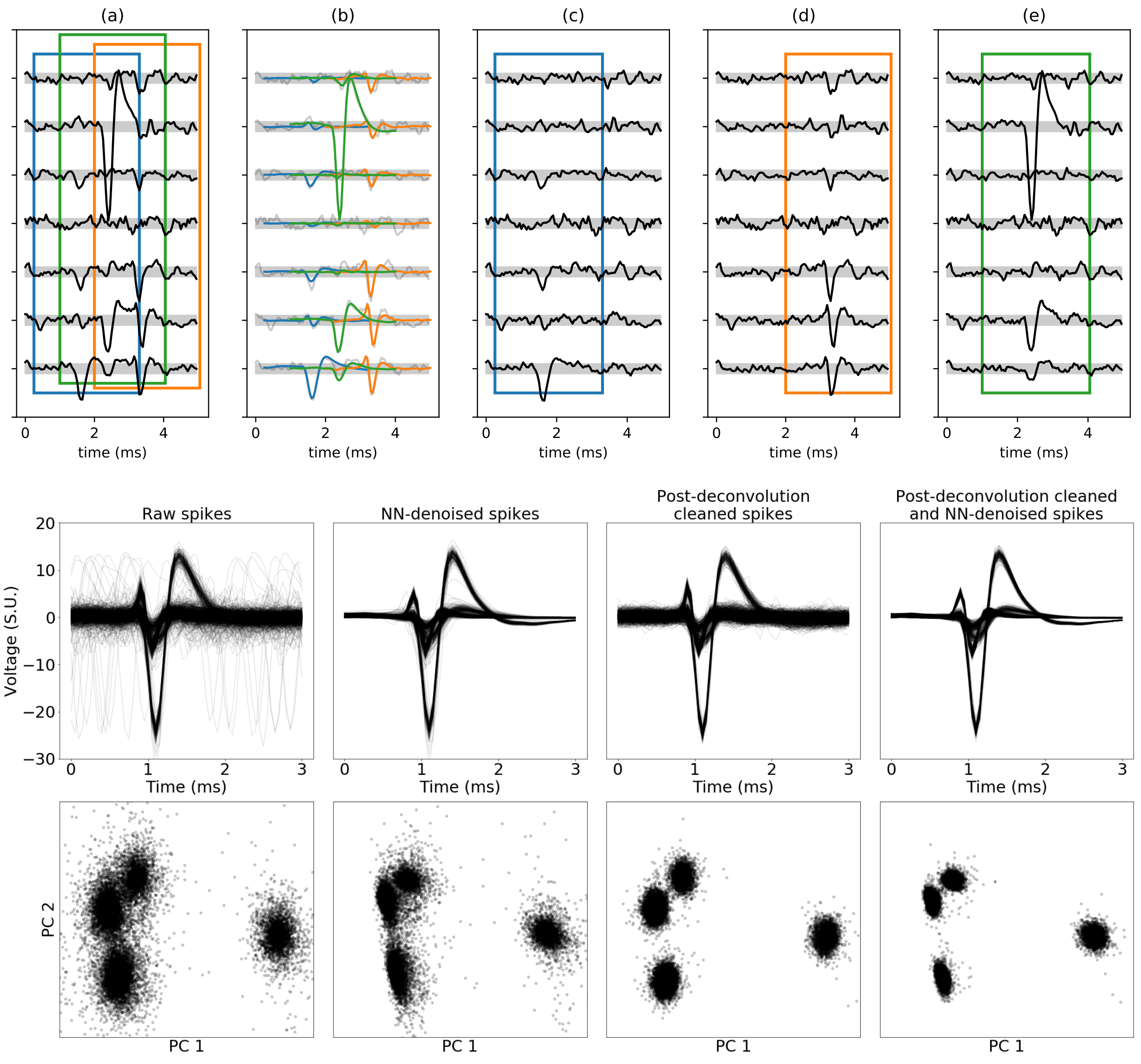


Figure 2.9: **Post-deconvolution spike waveform cleaning.** **(a)** A 5ms 7 channel raw voltage recording reveals the presence of multiple spikes. **(b)** Three spikes are identified by deconvolution and are depicted by inserting each template (colored traces) at the times identified by deconvolution. **(c, d, e)** 'Clean spikes' for each of the three spikes are obtained by superimposing the template shape with the residual voltages after all the other identified neuron spikes are removed from the raw record. **Middle row**: Raw, NN-denoised, post-deconvolution cleaned spikes, and cleaned then NN-denoised spikes from a single channel. **Bottom row**: PCA projections corresponding to the waveforms from the middle row. Note that post-deconvolution spike cleaning and NN-denoising both significantly improve the effective SNR of the data.

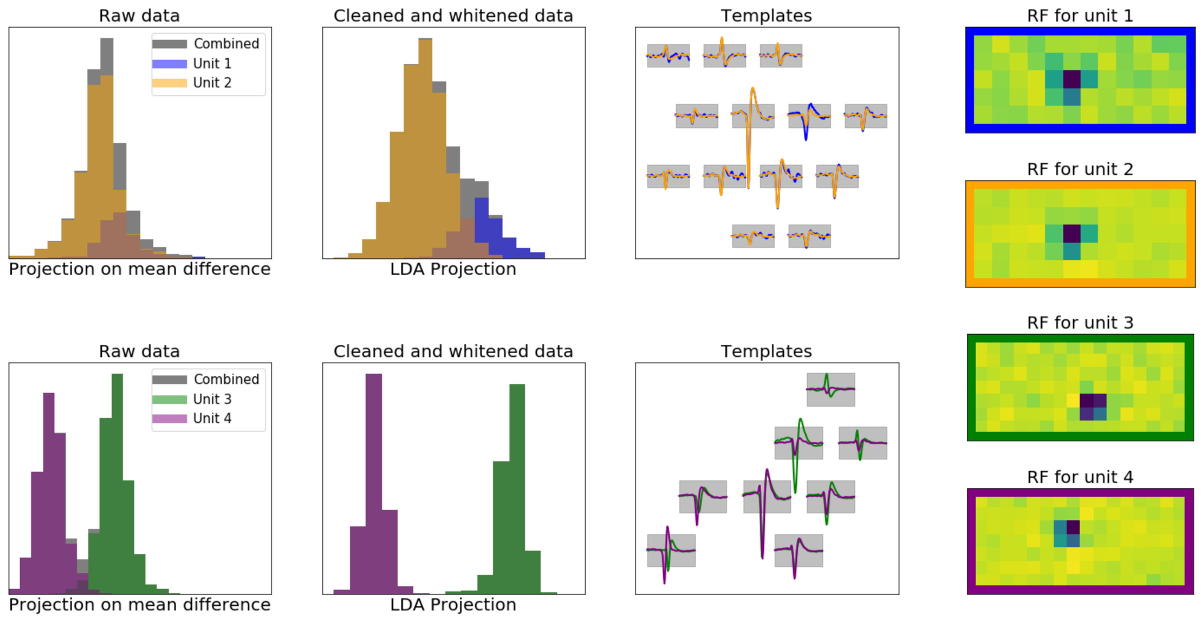


Figure 2.10: **Post-deconvolution merge using cleaned spike waveforms**. Each row illustrates the process for a different pair of units whose templates are close enough to be candidates for merging. **First column**: for each unit, we project the raw waveforms onto the difference between the two templates and plot the resulting histograms. **Second column**: same, but projecting the post-deconvolution cleaned waveforms onto the linear discriminant analysis (LDA) direction; note that this improves the separation. **Third column**: the templates of each pair of units. **Last column**: receptive fields (RFs) for all these neurons; note that these RFs are not used in the merge algorithm, but are shown here for verification purposes only. Using LDA applied to clean spikes, our algorithm suggests merging templates 1 and 2 while avoiding a merge of templates 3 and 4. This decision is supported by the corresponding RFs: RFs 1 and 2 are similar (though RF 1 is much noisier due to having fewer spikes than neuron 2), while RFs 3 and 4 are clearly distinct.

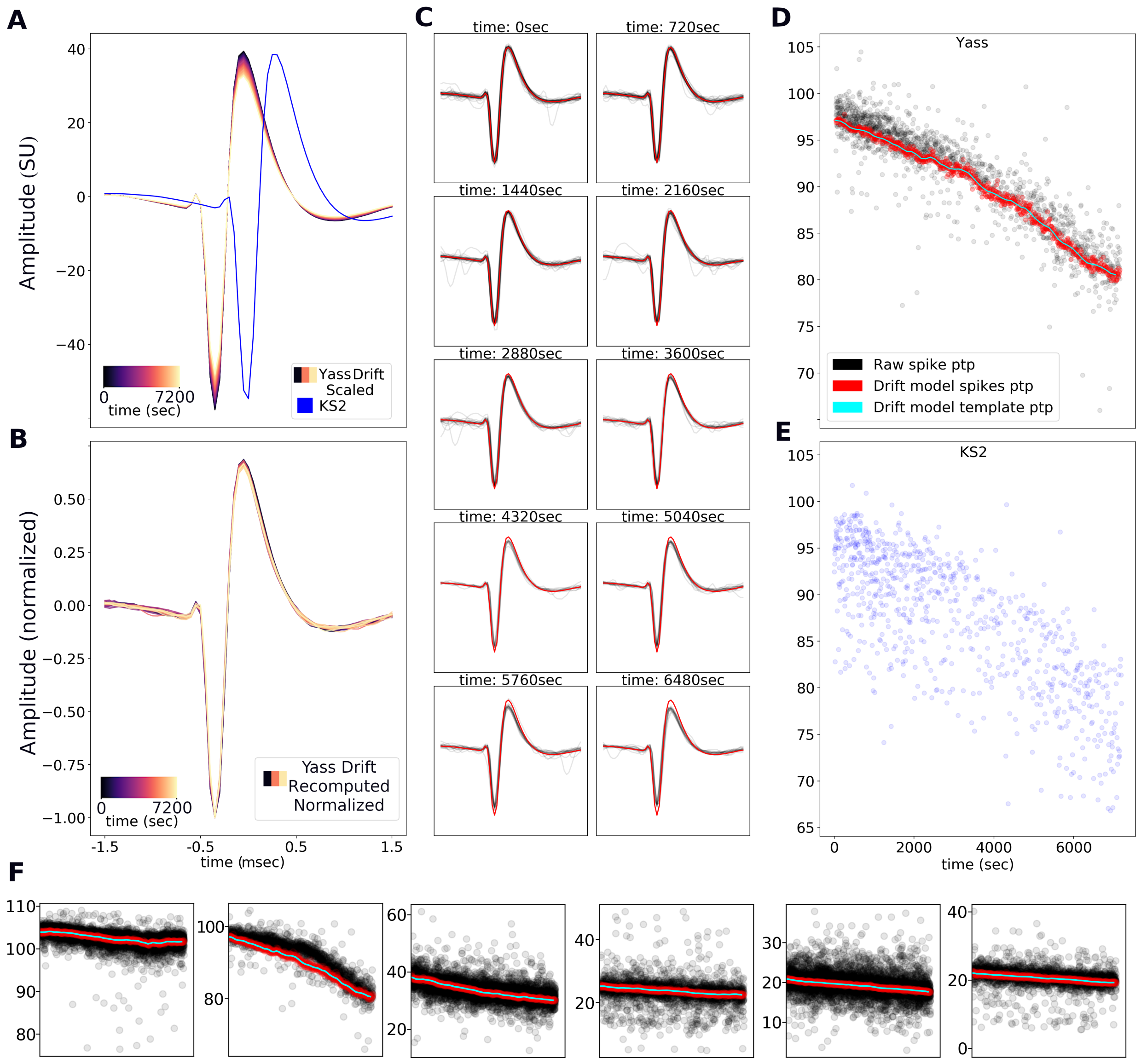


Figure 2.11: **Drift tracking via template scaling**. **A**. Drift model example: a template is scaled to match the drifting size of a neuron observed over a period of 2 hrs (magma colors; matching template from Kilosort is in blue and was computed using 5 minutes of data). Note the YASS template amplitude decreases by 15%-20% over the duration of the 2 hr (7200 sec) recording. **B**. Testing the stability of the template and rank 1 approximation: templates recomputed (and normalized) every 2 minutes for 2 hrs using deconvolved spikes (i.e. each template represents the mean of the spikes deconvolved in a 2 minute period). Note that the normalized template is nearly identical over time, suggesting scaling is an adequate model of drift. **C**. Template from the first 5 mins of recording (red) with deconvolved spikes (10) across 2hrs of recording. The change in individual spike size (black traces) compared with the starting template is observable at the single spike level. **D**. Drift model tracks changes: PTP of raw deconvolved spikes (black), PTP of spikes selected for the drift update (red) and PTP of the drift model (cyan). This shows that the drift model is tracking drift over time despite periodic updates (e.g. every 2 mins). **E**. PTP of spikes deconvolved for the same neurons by Kilosort. **F**. Additional examples of drift tracking; conventions as in panel D, but applied to different units.

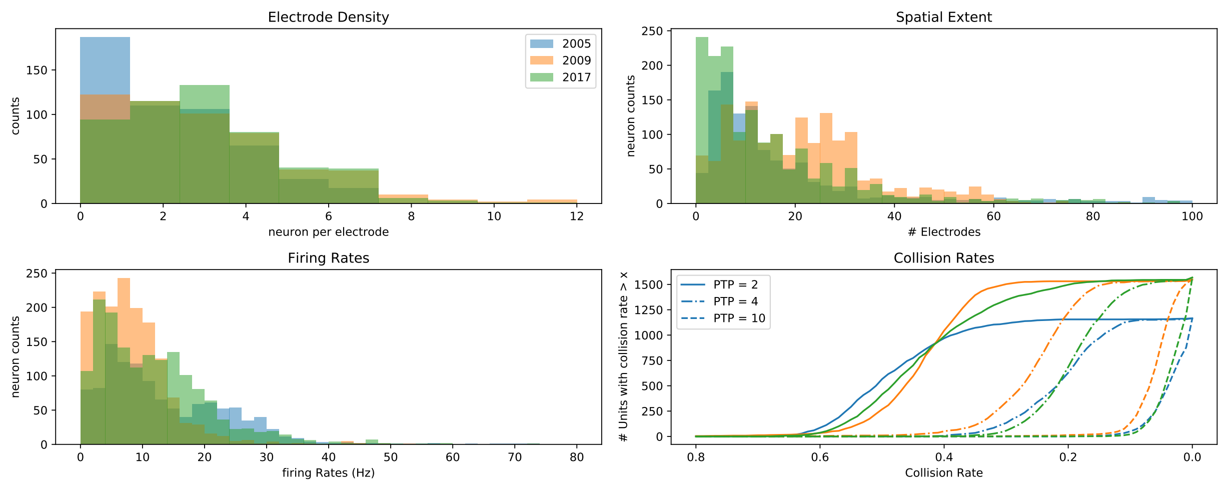


Figure 2.12: **Basic statistics of units extracted from three retinal datasets**. **First panel**: number of neurons per electrode for three datasets. **Second panel**: Spatial extent of each neuron, measured by counting the number of electrodes on which the template PTP was greater than 2 (note, 2009 dataset contained a higher incidence of axons, leading to more units active on a large number of electrodes). **Third panel**: Estimated firing rates of all sorted neurons. **Last panel** The rates at which spikes collide with other distinct signals of PTP size 2, 4, and 10 within a 0.5ms time window for the three datasets. Concretely, for about half of the recovered units, about half of all spikes have a collision with a spike of PTP size ≥ 2 within 0.5ms; similarly, for about half of the recovered units, about one fifth of all spikes have a collision with a spike of PTP size ≥ 4 within 0.5ms.

2.2 Results

2.2.1 Basic characterization of YASS output

Figure 2.12 provides some basic statistics characterizing the output of YASS applied to three retinal datasets (number of cells extracted, firing rate, etc.). As emphasized in Figure **??**, due to the large number of spikes visible on each electrode, the collision rate is high in these datasets: for about half of the recovered units, about 20% of all spikes have a collision with a spike of peak-to-peak size ≥ 4 within 0.5ms.

Figure 2.13 illustrates one of our basic diagnostic tools: examination of multiple templates and RFs, divided by cell type.

Figure 2.14 shows further diagnostic plots used to examine the health of a single sorted unit.

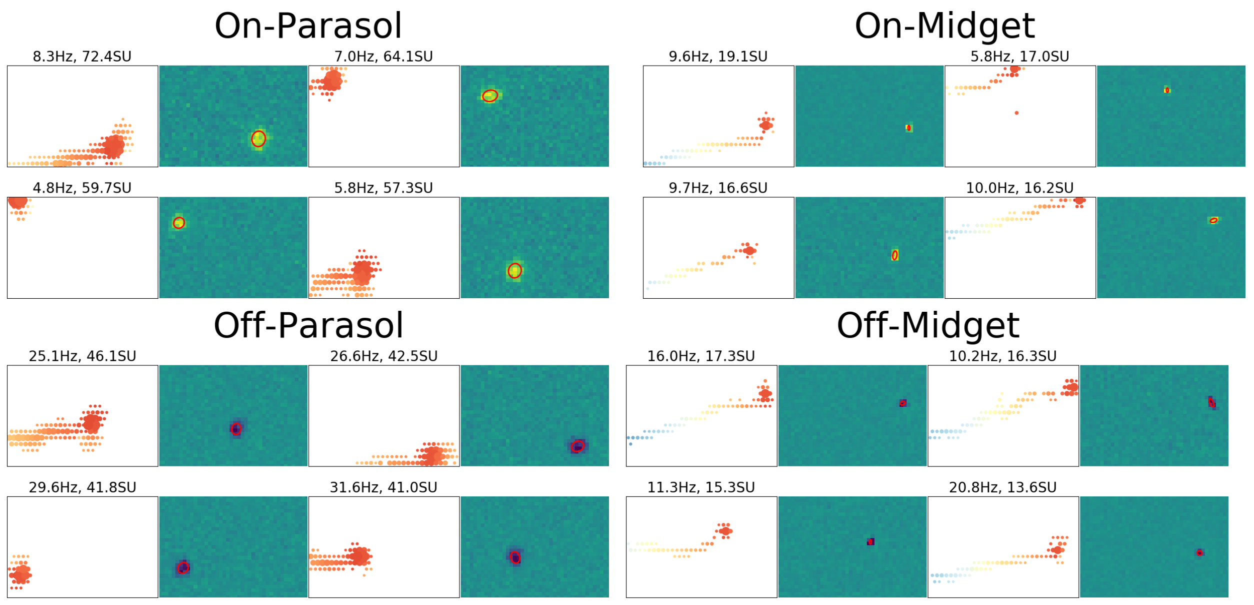


Figure 2.13: **Spatio-temporal templates and receptive fields (RFs) from four major cell classes**. **Left panels**: Spatial (filled circles) and temporal (colors) representation of templates for neurons sorted using YASS. Each dot represents the PTP of the waveform at the location (only channels with PTP>2.0 are shown), warmer colors (orange) indicate earlier time (i.e. closer to spike time) and cooler colors (e.g. blue) represent channels active at later times. **Right panels**: Spatial RF of neurons corresponding to the templates shown in the left panels. Red contour shows Gaussian fits (c.f. Figure 2.15 below). Cell types were determined from RFs by standard clustering procedures, with borderline cases determined by hand using a simple custom GUI.

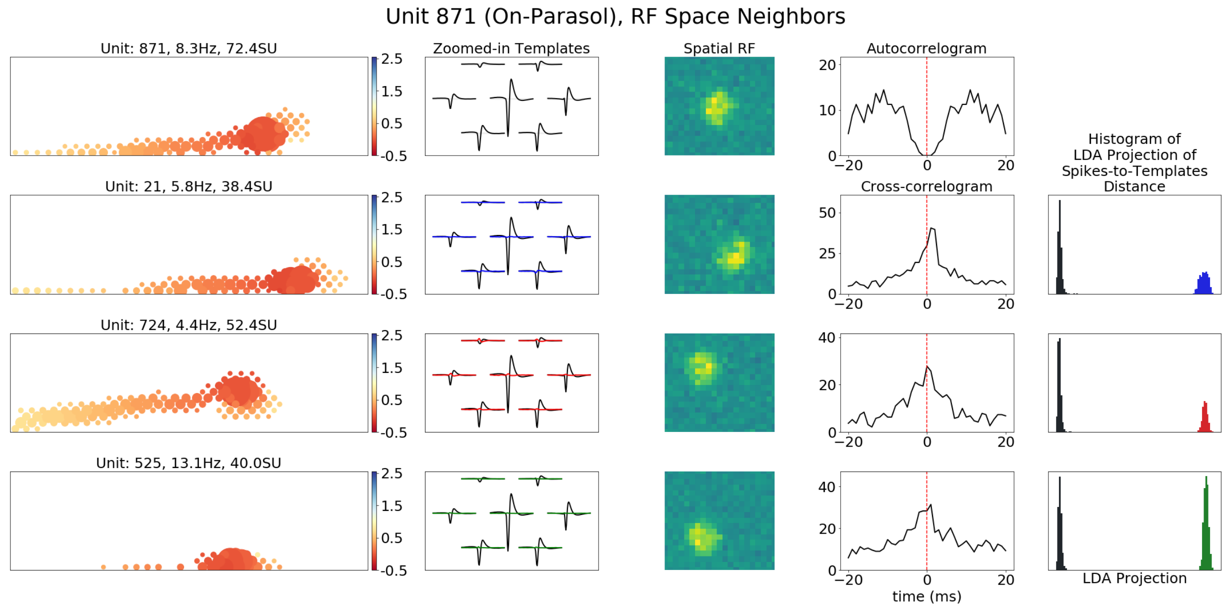


Figure 2.14: **Example diagnostic plots for a single recovered unit**. To examine the health of the extracted units we examine templates, RFs, and autocorrelations, along with corresponding plots for “nearby" units (where unit “neighbors" can be defined according to distance in RF space, template space, or other parameters such as the degree of correlation strength). Here we show an example of some of these analysis plots for an ON-parasol cell and its four nearest neighbors as defined by RF cosine distance. From left to right, the diagnostic plots show: template propagation profile across the 512 channel array (time since spike on primary channel in ms represented by color; firing rates and PTP in standardized units are provided in the panel titles); overlapping template plot of the main neuron on its primary and neighboring channel (black) and its neighboring cells (color); the (zoomed) spatial RF; autocorrelogram and cross-correlograms with the neighboring units (in Hz); and histograms of spikes-to-template distance for all spikes in each neuron against each template after the LDA projection described in section 2.1.10 (more distance between the histograms indicates that the cells are better separated).

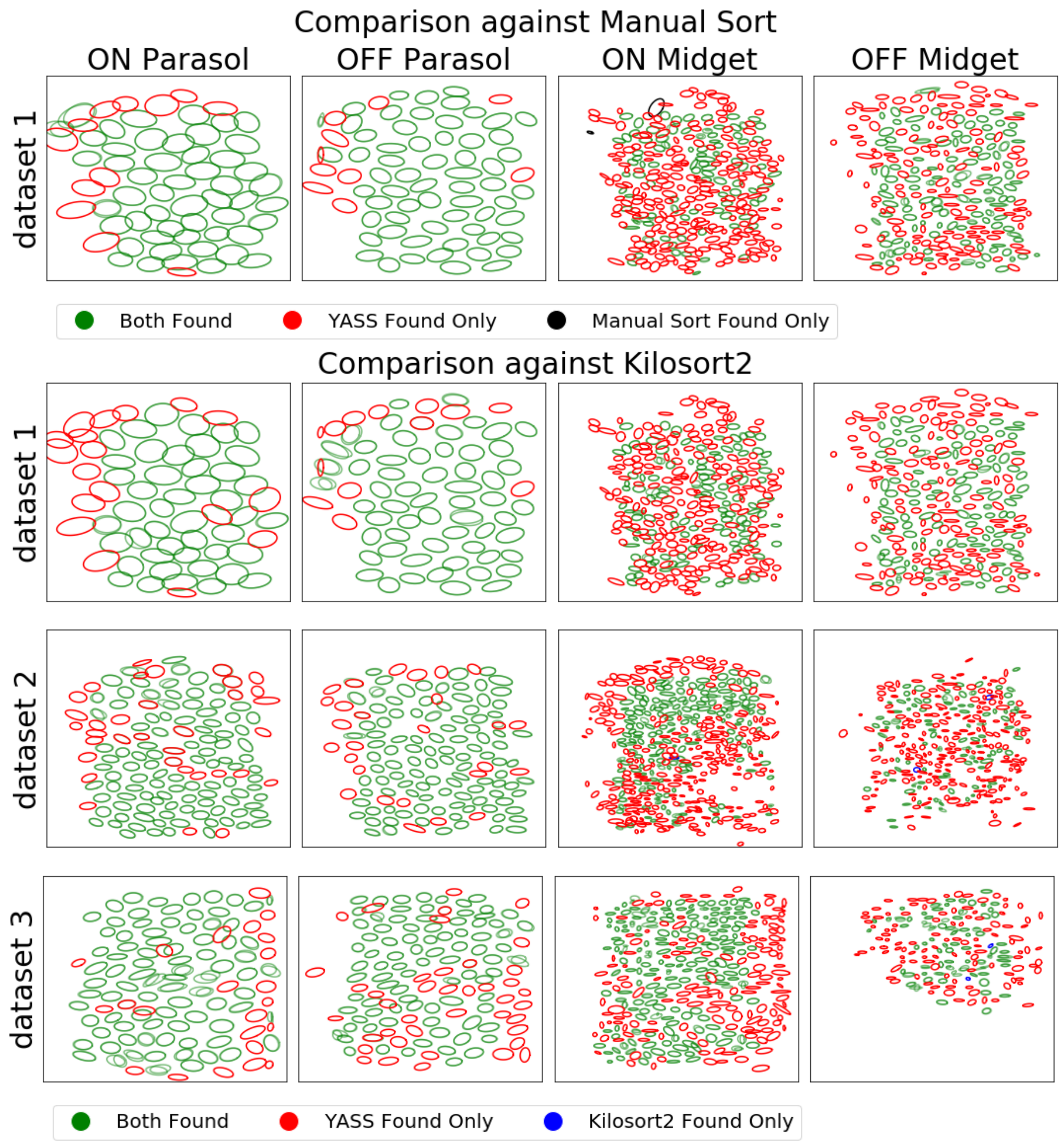


Figure 2.15: **Receptive field contour analysis** Each ellipse indicates a Gaussian fit to the receptive field of an individual unit. (Top) RF contours of cells extracted by YASS and a hand sorted result on one 512-electrode dataset. Green indicates cells that were found by both YASS and the hand sort; red indicates cells that were absent in the hand sort; black indicates cells that were found in the hand sort but not by YASS. (Bottom) Contours of Yass and Kilosort2 on three separate 512-electrode datasets. YASS consistently recovers more cells with RFs, particularly midget cells.

2.2.2 Comparisons to manual sorts and other automated spike-sorters

We have run head-to-head comparisons against Kilosort (1 and 2), Spyking Circus, Mountainsort, JRClust and Ironclust, and Herding Spikes. Of this group, in our hands Kilosort consistently led to the best results. Therefore in this section we will restrict attention to Kilosort for now, and provide full comparisons against the other packages in a future draft of this report.

Figure 2.15 is perhaps our main result: we see that YASS is able to recover much more of the underlying mosaic structure in the RFs of the four main retinal ganglion cell types, compared to the output of both a highly laborious manual sort and the best current semi-automated method (Kilosort).

Figure 2.16 illustrates a useful method for comparing between two spike sorters on real data, where no ground truth is available. For a given unit output by spikesorter A (e.g., YASS) we find the matching unit (or multiple units) output by spikesorter B (e.g., Kilosort). Then we run a spike-by-spike comparison to see which spikes were assigned to which unit; this comparison can very clearly show when one spikesorter or another is mis-assigning spikes, or is over-splitting units, or is missing units. The caption of Figure 2.16 provides further details. We summarize the results of this analysis in Figure 2.17: we find that the extra spikes found by YASS are typically similar to the “consensus" spikes found by both sorters; conversely, the extra spikes found by the non-YASS spike sorter often look significantly different from the “consensus" spikes (as measured by the cosine similarity of the average spike shape in each group).

Figure 2.18 analyzes the non-consensus units, i.e., units that are found by YASS but not KS2, or vice versa. Overall we find that KS2-only units tend to have few spikes (in some cases so few spikes that it was challenging to estimate reliable templates) that tend to be more noisy, with a higher overall rate of outliers. The YASS-only units, on the other hand, tend to have lower PTP values but lower outlier scores in general.

Figure 2.19 compares the residuals computed by YASS and Kilosort (i.e., the difference between the raw data and the sum of the templates subtracted away during deconvolution by each of these methods). We see that Kilosort fails to subtract away many clearly visible spikes, while there are

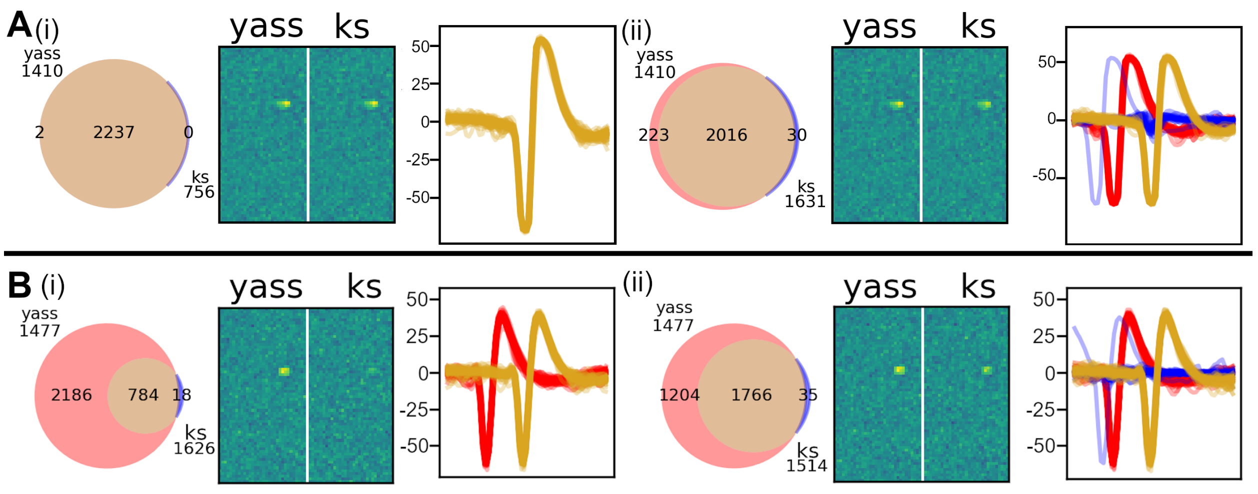


Figure 2.16: **Pairwise per-unit comparisons of spike sorters**. **A**. Example of an apparent Kilosort de-duplication error. **(i)**: (*Left*) Venn diagram for YASS unit 1410 (red) and Kilosort unit 756 (blue), showing an overlap of 2237 spikes (beige), with YASS identifying 2 extra spikes. (*Middle*) RFs for the 2 units. (*Right*) Spikes randomly selected from the intersection set. **(ii)**: (*Left*) Kilosort unit 1631 also matches YASS unit 1410, indicating that Kilosort unit 1631 is a duplicate of Kilosort unit 756. (*Middle*) RFs for the units (note the slightly weaker Kilosort RF, likely due to slightly fewer spikes). (*Right*) Spikes taken from the intersection (beige) look similar to additional spikes from YASS (red) but not additional spikes from Kilosort (blue), indicating that Kilosort unit 1631 contains some incorrectly assigned spikes. Note that spikes from different groups are displaced horizontally for better visibility here. **B**. Example of a likely Kilosort oversplit error. **(i)**: (*Left*) Venn diagram for YASS unit 1477 (red) and Kilosort unit 1626 (blue) have an overlap of 784 spikes (beige), with YASS identifying 2186 spikes and Kilosort identifying 18 extra spikes. (*Middle*) RFs for the 2 units (note that Kilosort RF is relatively weak here). (*Right*) Spikes randomly selected from the intersection set (beige) and YASS (red) indicate that additional spikes identified by YASS are consistent with the spikes recovered in the (beige) intersection set. **(ii)**: (*Left*) Kilosort unit 1766 also matches YASS unit 1477. (*Middle*) RFs for the units (note the slightly weaker Kilosort RF). (*Right*) Spikes taken from the intersection (beige) look similar to additional spikes from YASS (red) but not additional spikes from Kilosort (blue), again indicating that Kilosort unit 1631 contains incorrectly assigned spikes.

many fewer of these clear “missed spikes" visible in the YASS residual. Overall the YASS residual has a root-mean-square (RMS) of about 75% of that of the original “noise" level — i.e., YASS is able to explain a significant fraction of the “noise" by subtracting away many small contributions of distant spikes that are large and clearly identifiable on at least one channel. KS2, on the other hand, does not reduce the residual RMS significantly.

We additionally tested the full YASS pipeline on a number of “hybrid" datasets (following

Rossant et al., 2015), in which 20 neurons were added to an existing physiological recording containing 100-200 real neurons. YASS performed well on these datasets; here we restrict our discussion to a challenging dataset that contained 20 injected low-amplitude neurons (see Figure 2.20). On this dataset Kilosort did not detect any of the injected neurons at the default threshold). YASS is able to recover most spikes in most of the injected units, but for injected units with low firing rate and/or template size, the error rate increases sharply. We believe there may be room to continue to improve spike sorting performance in this challenging regime.

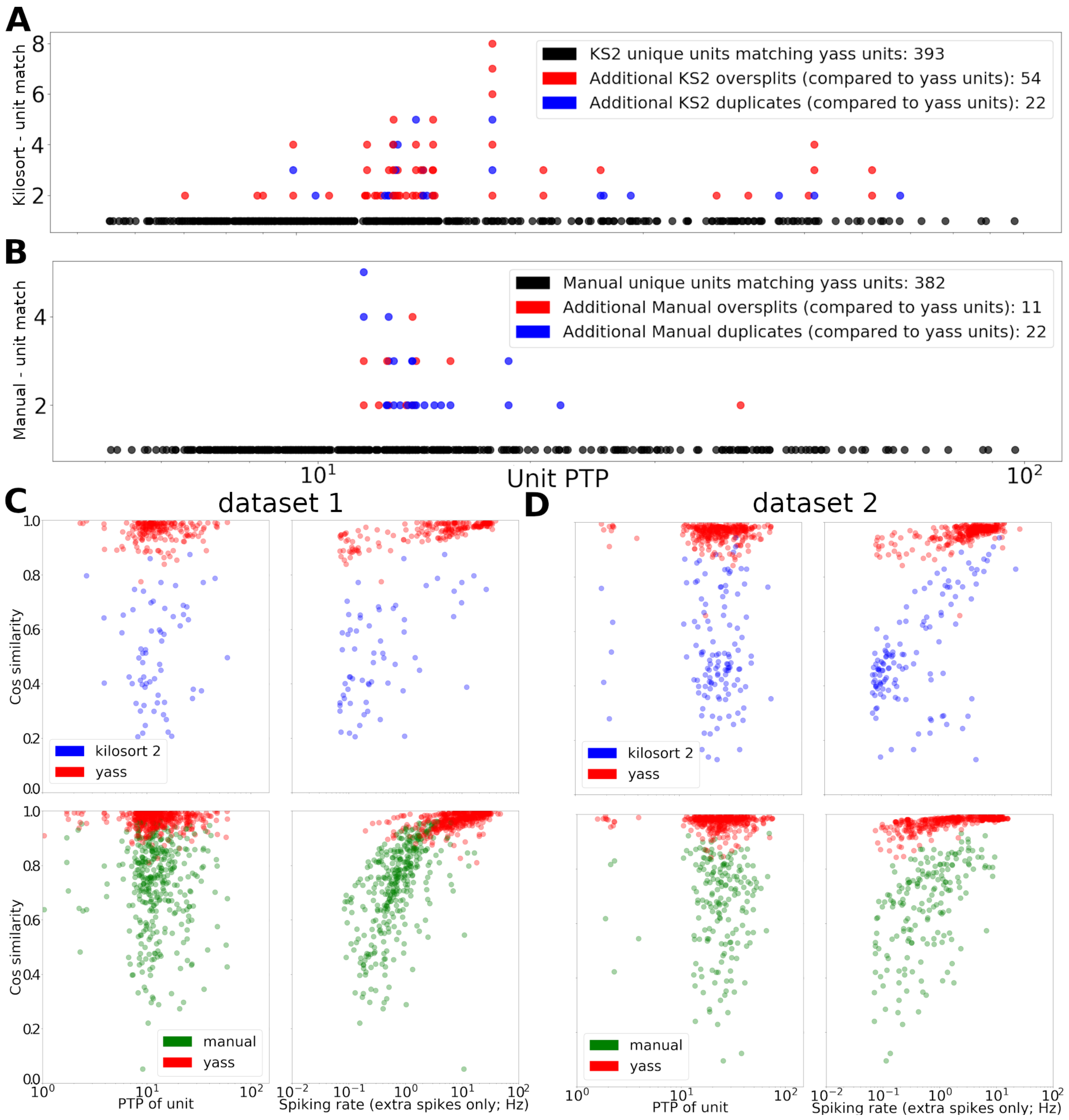


Figure 2.17: **Summary analysis of oversplits, duplicates, and consensus across spike sorters**. **A**. Summary of Kilosort units matched to YASS units. Most KS units matched well with YASS units (black dots) but there were many examples of oversplits and duplicates (as described in Figure 2.16; duplicates were defined as KS units that matched a YASS unit after another KS unit was matched; oversplits were defined as duplicates with fewer than two thirds of the YASS unit spikes). The y-axis indicates how many KS units were matched to a single YASS unit (see the legend for details). **B**: Same analysis as in (A) for the same dataset sorted manually. **C**: Cosine similarity between templates computed by averaging extra spikes identified by YASS (red) or KS (blue, top panels) or manual sorting (green, bottom panels) versus the template computed by averaging the "consensus" spikes identified by both spike-sorters. YASS extra spikes tend to be much more similar to the consensus template, consistent with the examples shown in Figure 2.16. In the left panels we scatterplot these cosine similarities against the PTP size of the unit; in the right panels against the firing rate of the extra identified spikes. Note that in many cases YASS identifies a large number of extra spikes that closely match the consensus template. **D**. Same analysis as in (C) for another dataset.

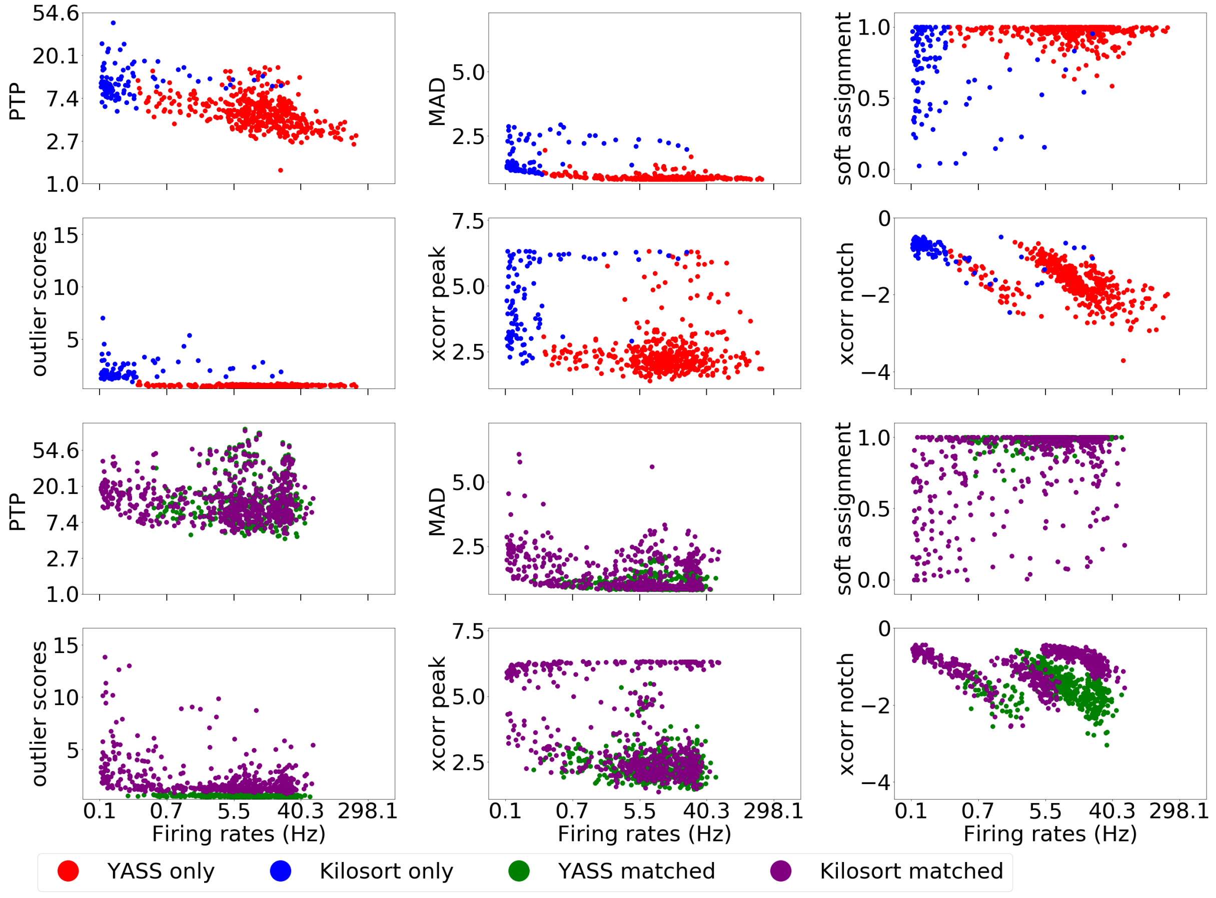
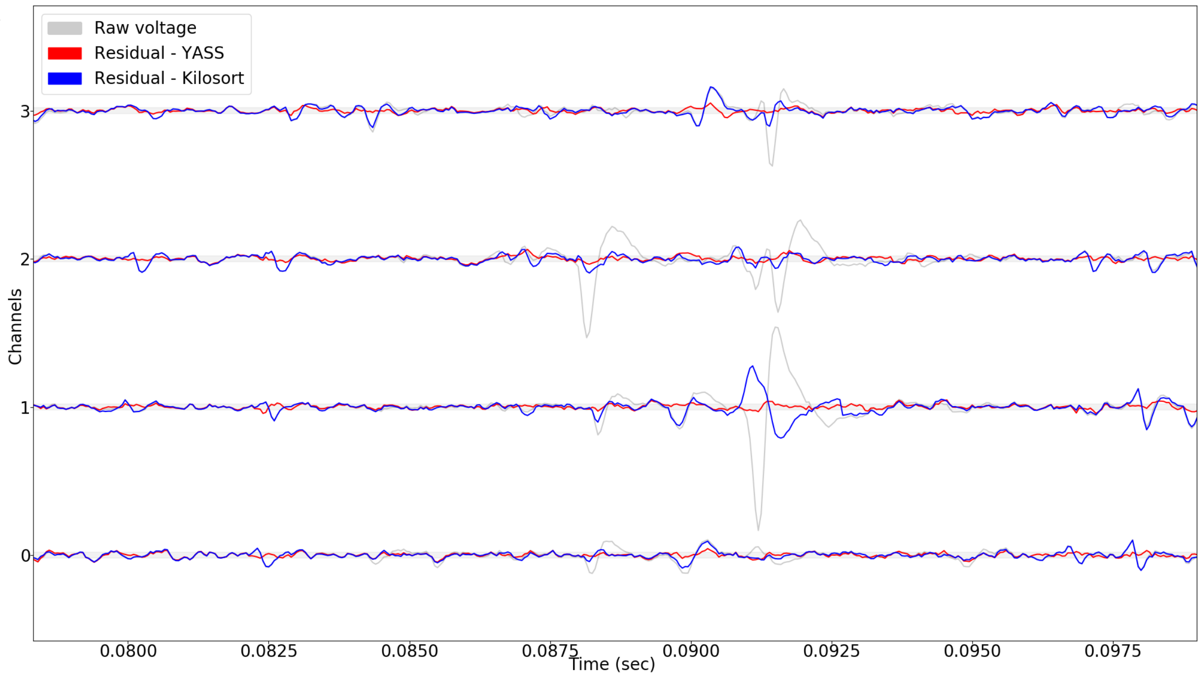


Figure 2.18: **Comparing consensus and non-consensus units from YASS and Kilosort** Among the units with firing rates bigger than 0.1Hz, We found 444 unmatched YASS-only and 101 Kilosort-only units, compared to 491 (YASS) and 595 (Kilosort) matching consensus units (note that it is not a pairwise matching due to oversplits). The top panels show non-consensus units, and the bottom panels show consensus units. We scatterplot the firing rate of each unit against: the unit's PTP; the maximum absolute deviation (MAD) of the aligned cleaned spikes (overly large values indicate corruption of these spikes, due to e.g. deconvolution errors or over-merged units); the average soft-assignment probability (i.e., one minus the probability that a spike assigned to unit i should have been assigned to some other unit j, averaged over all spikes from unit i; small values indicate closely overlapping clusters that may correspond to over-splitting); the "outlier score" defined as the average norm of the whitened post-deconvolution residual around each spike (again, large values indicate corruption of these spikes); the maximal normalized cross-correlation (zero lag, one ms resolution) between unit i and all other units j (overly large values can indicate "fragmented" units, where for example axonal channels have been split from the somatic channels); and the minimal cross-correlation, defined similarly (a large "notch" in the cross-correlation function often indicates oversplitting, since if a cell is oversplit into two units i and j then i and j will tend not to spike at the same time).

A



B

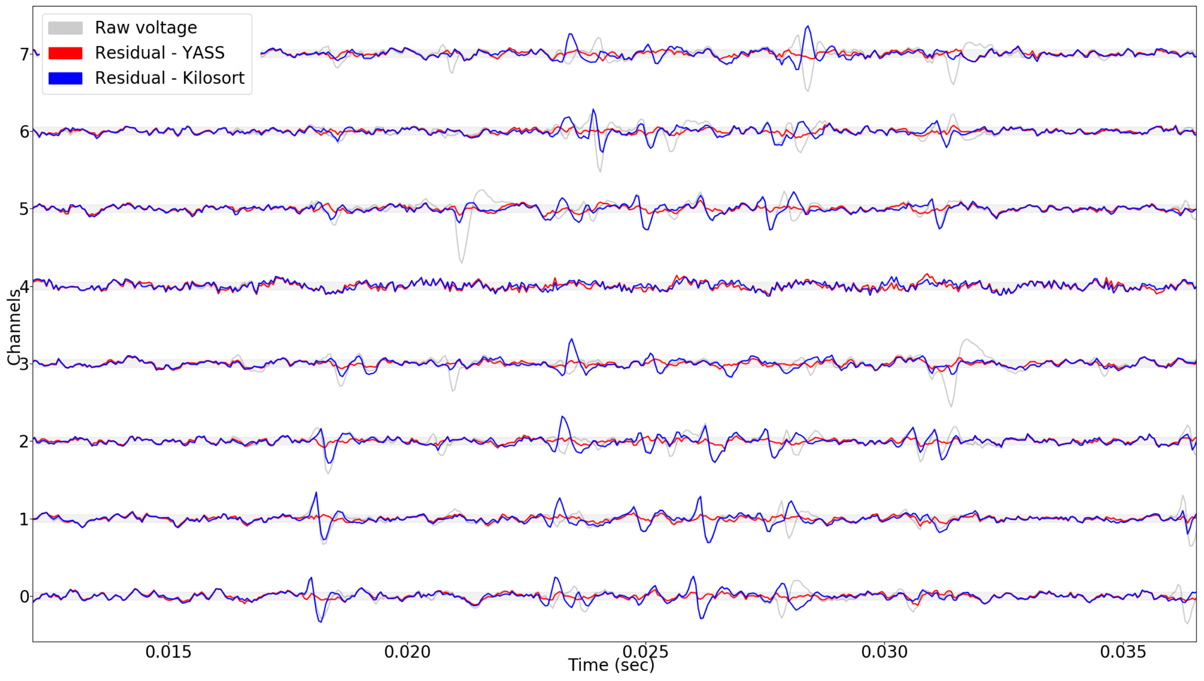


Figure 2.19: **Residual analysis**. **A**. A 20 ms example of YASS and Kilosort residuals on 5 channels. For large units, Kilosort occasionally leaves large residuals behind (c.f. Figure 2.7). **B**. Another example on 8 channels reveals some spike mis-assignment by Kilosort.

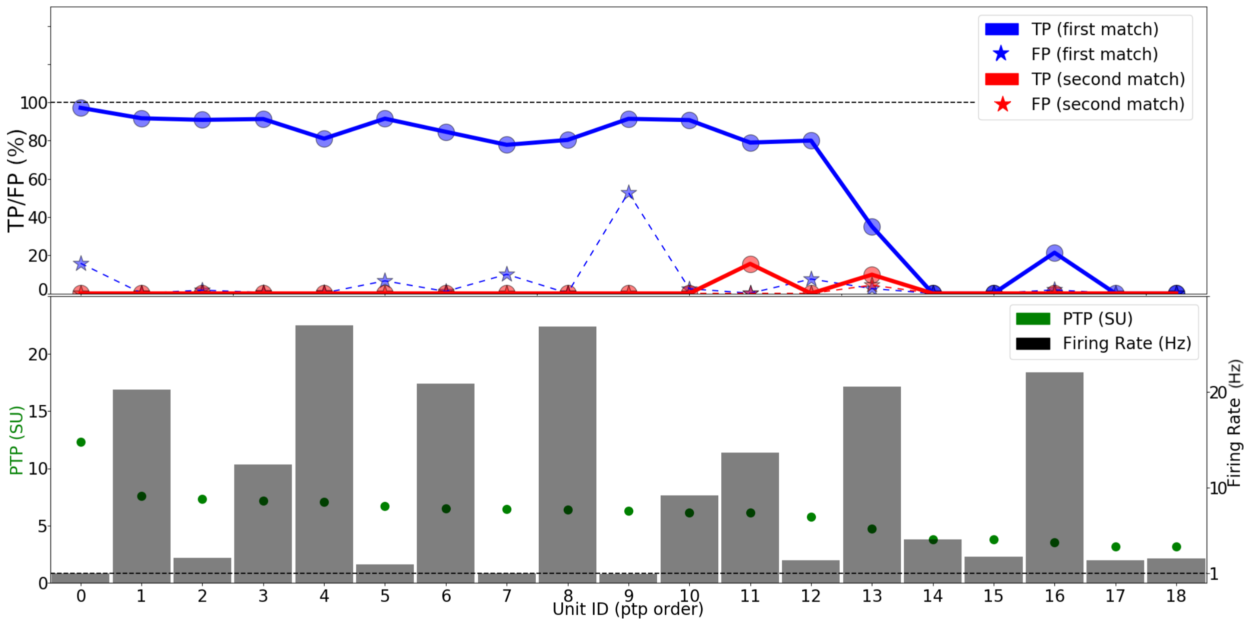


Figure 2.20: **YASS results on hybrid semi-synthetic datasets containing low amplitude units. Top**. The true positives (solid colors) and false positives (hatched colors) for the best (blue) and second best (red) matches for neurons sorted by yass (for clarity, a perfect match is a blue bar that contains 100% of the spikes with no other bars). **Bottom**. Firing rate and PTP amplitude for neurons shown above. (Note: Kilosort did not detect any of the injected units using the default thresholds).

Acknowledgements

We thank Nora Brackbill, Gaute Einevoll, Espen Hagen, Ari Pakman, Calvin Tong, Yueqi Wang, and Weichi Yao for many helpful conversations. We also thank the developers of Kilosort/Kilosort2, JRClust/Ironclust, Mountainsort, Spyking Circus, and Herding Spikes for generously sharing their code. This work was partially supported by the Simons Foundation, the ONR, and NSF grants IIS-1546296 and NSF IIS-1430239. This research was also developed with funding from the Defense Advanced Research Project Agency (DARPA), Contract No. N66001-17-C-4002. The views, opinions and/or findings expressed are those of the author and should not be interpreted as representing the official views or policy of the Department of Defense of the U. S. Government.

Chapter 3: Imitation Learning for MEA Spike Deconvolution

3.1 Introduction

Previously, in 2.1.9 we saw how MEA spike deconvolution is a crucial part of any spike sorting pipeline. Recall that this problem arises under the assumption that voltage traces are a superimposition of spike waveforms of individual neurons and background activity.

In Sparse Coding (SC), signals are reconstructed using a sparse linear combination of much smaller basis signals that are called dictionaries. Therefore, MEA spike deconvolution can be formulated as convolutional SC problem where waveforms take the role of dictionaries. In many spike sorting pipelines including YASS, however, MEA deconvolution does not seek to learn dictionaries. Instead, it merely aims to minimize the reconstruction error given a fixed set of neuron templates (i.e. dictionaries). Despite this simplification, the problem is a daunting one due to overlapping signals and its convolutional setting. The most computationally efficient models apply a greedy approach to this non-convex problem where templates are assigned to (and consequently subtracted from) the residual to obtain the highest reduction in local reconstruction error.

There are non-greedy methods based on heuristic search that generally improve performance with a downside. They come at a substantially higher computational cost due to searching the solution space more exhaustively. For instance, given an initial estimate of a candidate spike train, Binary Pursuit (Pillow et al., 2013) either subtracts spikes out or puts them back in the residual to achieve better local reconstruction errors. Continuous Basis Pursuit (Ekanadham et al., 2011), on the other hand, models each neuron's waveforms using a set of basis functions that can account for continuously shifted signals. This allows outputting continuous spike times. (Song et al., 2018) develops convolutional versions of Orthogonal Matching Pursuit (OMP) and K-Singular Vector Decomposition (KSVD) as more optimal dictionary learning methods. Despite its attempt at rendering them computationally efficient, these method do not scale to current scale of MEA data.

Deep networks are de facto state-of-the-art in computer vision and natural language processing due to their impressive performance. Deep-network-based approaches exist also for sparse coding. Two major examples include LISTA (Gregor et al., 2010) and Multi-Layer Convolutional SC (ML-CSC) (Sulam et al., 2017) that integrate deep learning into the frame work of SC. LISTA devises a deep network architecture that proposes approximate solutions to SC. Then, these are used as initial solutions to ISTA (Chambolle et al., 1998) that is an iterative algorithm with theoretical guarantees of producing optimal dictionaries and sparse codes. Thereby, it drastically reduces the number of iterations that is required by ISTA. However, LISTA does not address the convolutional setting of our problem. On the other hand, ML-CSC proposes an end-to-end deep learning framework for dictionary learning with some theoretical bounds on the performance. There are two major drawbacks to applying ML-SCS to MEA spike deconvolution. First, ML-SCS does not impose any constraint on the content of the dictionaries which represent neuron waveforms and occupy a small subspace of dictionary space. This can lead to fragmenting a spike into smaller meaningless partitions. Second, it is computationally infeasible for large streams of MEA data even during inference (i.e. forward pass).

In this work, we propose a solution to this MEA spike deconvolution that relies on the following observations. Greedy approach offers desirable computational efficiency while making errors along the way due to favoring short term gains over long term ones. Having an efficient yet non-trivial "spike checker" for greedy solutions leads to improved results. Second, because of stereotypical shape of neuron waveforms, virtually endless synthetic data can be created within a self-supervised framework for such a spike checker. Therefore, we propose a 1D convolutional neural network architecture as a spike checker during the iterations of greedy method. Inspired by imitation learning work of (Ross et al., 2010), we devise a methodology to obtain realistic training data for our spike checker.

3.2 Enhancing Greedy Deconvolution with Imitation Learning

Greedy deconvolution methods are desirable because of their low computational costs. However, these methods are prone to high rates of false assignment. These false assignments are either due to abundance of low SNR events or missing (undiscovered) neuron dictionaries. These challenges are amplified by the simplicity of the heuristics, i.e immediate reduction in L_2 reconstruction loss. We aim to circumvent this major obstacle by enhancing the greedy algorithm (2.1.9) by incorporating a spike checker classifier. The classifier inspects every spike proposed by the greedy methodd, allowing the correct assignments to be subtracted from the residuals while preventing false ones from being subtracted.

Our spike checker takes input examples in the form of $z^{(i)} = x_1^{(i)}, \dots x_C^{(i)}, y_1^{(i)}, \dots y_C^{(i)} \in \mathbb{R}^T$ that respectively denote raw signals on C channels and neuron template on the same C channels for spike i proposed by greedy method. The choice of channel subset is induced by neuron to which y belongs. For each neuron, we choose the C channels with highest peak to peak value for its template. Note that C and T are fixed across all neurons and are model parameters that are connected to our spike sorting pipeline. Therefore, our spike checker takes a pair of raw signal and template and outputs a probability $\in [0, 1]$ corresponding to the Bernoulli random variable of $x^{(i)}$ belonging to neuron template $y^{(i)}$.

There are two major challenges for devising such a classifier. To begin with, our examples that consist of raw electrical signals and templates, are multi-channel time series where the order of channels are not informative. In fact, the ideal classifier's prediction must be invariant with respect to the order in which signals from different channels are presented to it. This concept is further illuminated by Figure 3.1. Second, a synthetic dataset is required for training the classifier where examples do not follow the i.i.d assumption. This is due to the sequential nature of the predictions of such a model. The choices made in the earlier iterations of enhanced greedy method affect the distribution of examples that the spike checker faces in later iterations.

In section 3.3 we propose a new 1D convolutional architecture that satisfies that channel order agnostic behaviour that we expect from our ideal spike checker classifier. Subsequently, in section 3.4 we discuss the imitation learning approach that we take to obtain optimal and realistic synthetic data to train our spike checker.

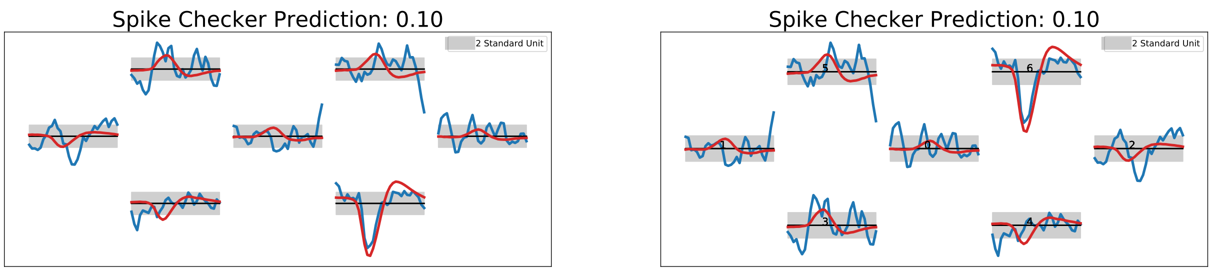


Figure 3.1: **Two input examples from synthetic MEA data and corresponding outputs of our spike checker classifier.** These two examples are the same except for the order in which channels are presented. In MEA recordings, we often subset electrodes. The order in which data from these electrodes are presented can vary arbitrarily. Therefore, it is essential for our model to have the same prediction regardless of how the electrodes are ordered. This constraint potentially can assist with regularizing our model and generalizing to unobserved datasets.

3.3 PermSepConv: Permutation Equivariant Separable 1D Convolution Block

One dimensional (1D) convolutional layers have successfully been applied to a variety of problems such as Real-Time Electrocardiogram (ECG) monitoring (Kiranyaz et al., 2015; Ince et al., 2009) and structural damage detection in civil engineering (Yu et al., 2019). This success is owed to the ability of these models to effectively incorporate temporal information while not introducing the the optimization challenges of Recurrent Neural Networks (RNN) e.g. exploding/vanishing gradients.

We choose 1D convolution layers as building blocks for our spike checker, since we successfully applied them to spike detection earlier in section 2.1.3. However, we face the issue of channel orders that we need to resolve. Note that a 1D conventional map's $f_{conv1D} : \mathbb{R}^{t\times c_1} \rightarrow \mathbb{R}^{t\times c_2}$ output can arbitrarily vary if we shuffle the order of input channels. Also, observe that our inputs consists of C two-channel time series. In other words, $z^{(i)} \in \mathbb{R}^{C\times T\times 2}$ where $z^{(i)}[c,t,1] = x^{(i)}[c,t]$ and $z^{(i)}[c,t,2] = y^{(i)}[c,t]$ respectively represent elements of raw signal and neuron template.

Permutation invariant maps (Zaheer et al., 2017) offer a trivial solution to the aforementioned problem. The sufficient condition for a map F to be channel permutation invariant is $F(z^{(i)}) = \sum_{c=1}^{C} f_{conv1D}(z^{(i)}[c])$ where $F : \mathbb{R}^{C \times t_1 \times c_1} \longrightarrow \mathbb{R}^{t_1 \times c_2}$. The simplicity of this additive approach comes with an extreme information bottleneck where information from all C channels linearly collapse onto only one channel. Instead we utilize permutation equivariant maps (Zaheer et al., 2017) that are described by Equation 3.1 as a way of symmetrically incorporate information from all C channels with more expressive power. This map, which we dub permutation equivariant separable 1D convolution or *PermSepConv*, is formulated by Equation 3.2 and visualized by Figure 3.2. Effectively, after applying the same 1D convolution kernels to all channels, we apply a permutation equivariant map, with the same parameters, to all elements of output channels. This is followed by an average pooling across all channels to reduce temporal dimension.

$$f_{\pi}(x; \lambda, q) = \left(\lambda \mathbf{I}_{n \times n} + q \mathbf{1}_n \mathbf{1}_n^{\intercal}\right) x \tag{3.1}$$

$$f_{PermSepConv}(z; \theta, \lambda, q) = \left[f_{\pi}\left(f_{conv1D}(z[1]; \theta), \cdots, f_{conv1D}(z[C]; \theta); \lambda, q\right)\right] \tag{3.2}$$

3.4 Training Spike Checker by Data Aggregation

To train our spike checker network we require a realistic dataset of raw electric signals and neuron templates that can arise during iterations of greedy deconvoltuion. If the actions of greedy deconvolution depend on predictions of a spike checker, then the examples within such a dataset violate the common i.i.d assumption. Additionally, the distribution of examples that the network encounters directly depends on the greedy denconvolution method and its parameters. Dataset Aggregation (DAGGER) (Ross et al., 2010) is a well suited approach for our sequential prediction task.

DAGGER is an iterative algorithm that trains a deterministic policy (i.e. spike checker) in a sequential prediction task that achieves optimal performance under the distribution of states (i.e.

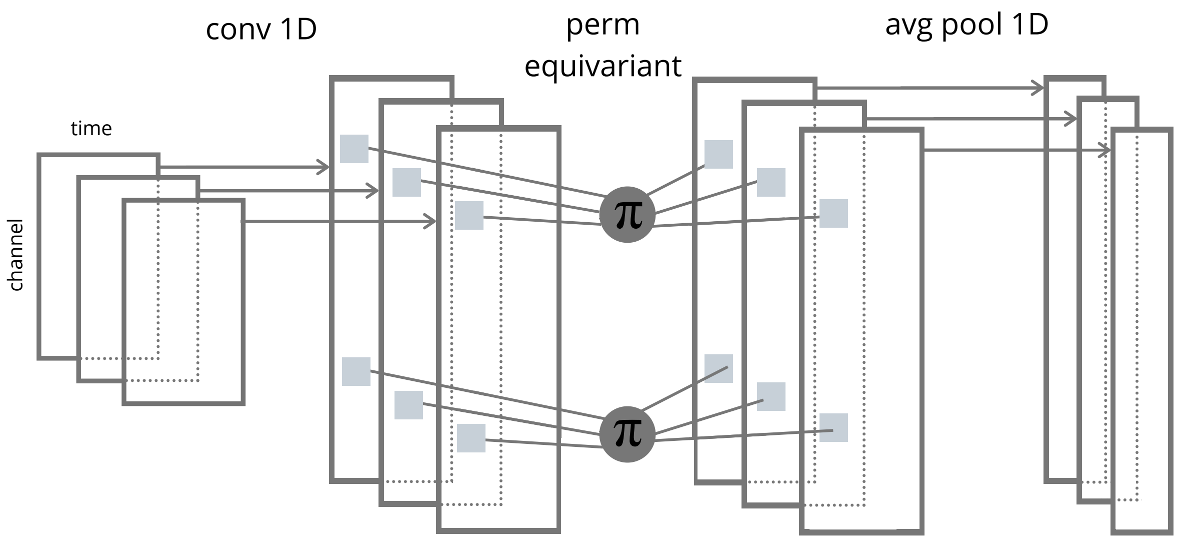


Figure 3.2: **Depiction of permutation equivariant separable 1D convolution block.** Input to the block is comprised of a set of C matrices each of them having $c \times t$ dimensions. Each one of these matrices corresponds to one of the channels (i.e. electrodes) of the raw electrical signal in the example. Therefore, this number remains constant throughout the PermSepConv blocks in our network. A block applies the same 1D convolution kernels to all matrices. Then it applies the same permutation equivariant map to all the elements of output matrices. This is followed by an average pooling across to reduce temporal dimension.

examples) that the policy induces. The algorithm starts by fully relying on the expert's policy $P^*(.)$ (i.e. ground truth labels of our synthetic data) to gather a dataset of trajectories. The trajectories are comprised of different examples that occur during the iterations of greedy deconvolution on a particular synthetically generated MEA recording. Then the spike checker $P(.)$ is trained on the collected examples. As DAGGER proceeds, two things happen. The algorithm encounters more trajectories on various synthetic MEA recordings and aggregates them into a single dataset. Additionally, it increasingly relies on the predictions of the spike checker rather than the expert. Each time before generating a new MEA recording, spike checker is trained on the growing aggregated dataset. Algorithm 2 describes a pseudo-code for our proposed algorithm that is adapted from DAGGER.

Algorithm 2 Pseudocode for training spike checker using DAGGER method.

```
1: Input: Deconvolution (sec. 2.1.9) function, manually curated templates $Y_k$
2: $\mathcal{D} \leftarrow \emptyset$
3: Initialize spike checker Neural Network $P(.)$
4: for i=1...N do                                          ▷ Generate a new synthetic MEA data
5:     $V_{noise} \leftarrow$ TemporallyCorrelatedNoise($\{Y_k\}$)
6:     $T_{synth} \leftarrow$ SyntheticSpikeTrain($K$)
7:     $\{Y_k^{synth}\} \leftarrow$ SyntheticTemplates($\{Y_k\}$)
8:     $V_{synth} \leftarrow$ SuperImposeTemplates($\{Y_k^{synth}\}, T_{synth}, V_{noise}$)
9:     for j=1...M do                   ▷ Runs a single iteration of greedy deconvolution (2.1.9)
10:        $\{z^{(j)}\} \leftarrow$ DeconvIteration($V_{synth}, \{Y_k^{synth}\}$)
11:        $\{z^{(j)}_{accepted}\} \leftarrow \{z^{(j)} | \beta_i P(z^{(j)}) + (1-\beta_i) P^*(z^{(j)}) > \theta\}$
12:        $V_{synth} \leftarrow$ Subtract($V_{synth}, \{z^{(j)}_{accepted}\}$)
13:        $\mathcal{D} \leftarrow \mathcal{D} \cup \{z^{(j)}\}, P^*(\{z^{(j)}\})$        ▷ Update data with current examples
14:    Train $P$ on $\mathcal{D}$ using SGD
15: Output: Spike checker Neural Network $P(.)$
```

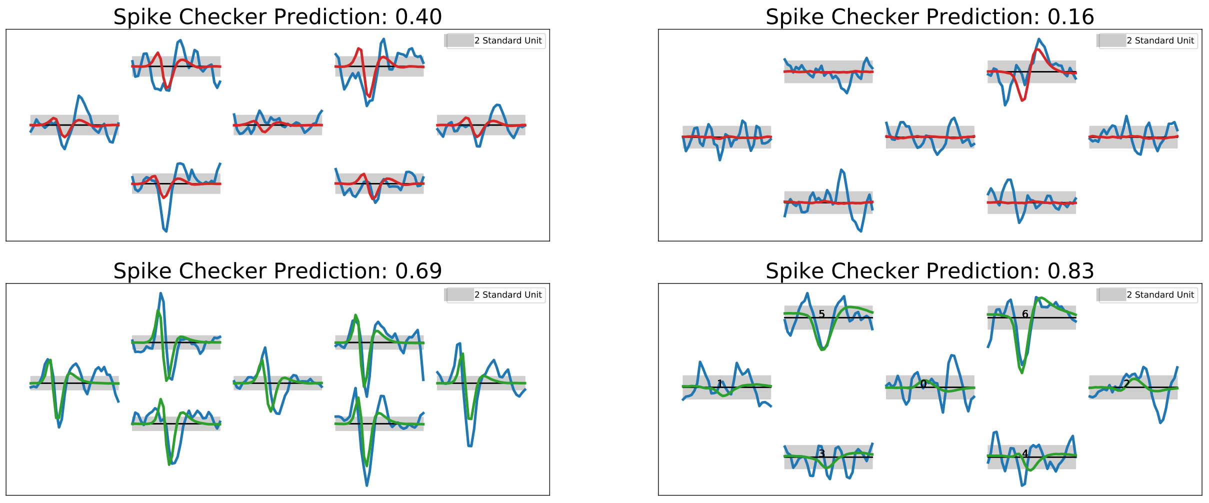


Figure 3.3: **Four input examples from synthetic MEA data and corresponding spike checker outputs.** Two top examples are cases where greedy deconvolution has proposed a false assignment, whereas the two bottom examples are correct proposals. In all these cases, our spike checker has accurately identified whether the proposed spikes are correct or wrong. In these examples $C = 7$ and raw voltages x_c are depicted by blue traces while templates y_c are visualized by red/green traces respectively for false/correct assignment.

3.5 Results

In this section we demonstrate through experiments with synthetic MEA data that our enhanced deconvolution method improves upon the greedy method especially when false assignments are considered. We start by discussing the architecture of our spike checker in more detail. We proceed to elaborating on the process of generating realistic synthetic dataset both for training spike checker and for testing the the enhanced deconvolution. Also, some choices that we considered in adapting DAGGER will be discussed. Finally, we present summaries of enhanced deconvolution on three distinct type of synthetic MEA recordings.

3.5.1 Synthetic MEA data

The first step of synthetic dataset generation involves creating artificial MEA recording. We follow the same recipe described in 2.1.3 where we obtain a sufficiently large set of neuron templates (from ≈ 120 neurons) that are manually curated on MEA recording from primate retina. This real recording is from 49 electrode array.

In order to generate realistic background electrical activity, per electrode we subset $K = 10$ neuron templates and generate K different white noise time series. This is followed by computing cross correlations of the K templates with their corresponding K white noise time series. The superimposition of the K results yields a single electrode correlated noise time series.

Synthetic spike trains are generated by sampling from a population firing rate Gamma distribution. Subsequently, spike times for each neuron are observations from a Poissson process with the neuron's corresponding firing rate. We also generate synthetic neuron templates given the curated templates by rotating the 2D grid of electrodes uniformly and scaling each channel by a different value generated from truncated Gaussians.

In our experiments, we have ≈ 100 synthetic neurons. We generated datasets of 49 electrodes and of time length 60 seconds with sampling frequency of $20kHz$. For fair evaluation, test time and training time datasets are generated from two distinct set of real MEA recordings with different neuron templates.

3.5.2 Neural Network Architecture

Our model is built on the foundation of PermSepConv blocks that we introduced in section 3.3. We use three consecutive PermSepConv blocks with 8, 16, 32 convolution kernels respectively and average pooling stride of 2. All kernels have length of 6 time sample and ReLU non-linearity is used following the 1D convolution of of each block. Our examples $\{z^{(i)}\}$ have 16 electrodes. To transform the convolutional features from 16 electrodes to binary prediction, two fully connected layers with 512 units and ReLU activation are applied to each 16 feature maps separately. This is followed by adding the 512 dimensional representation of each electrode together. This is the sufficient and necessary condition to obtain permutation invariance in predictions (Zaheer et al., 2017). Finally, a fully connected layer with sigmoid activation maps the features to probability of raw voltage correctly assigned to the neuron template.

The argument for choosing PermSepConv over conventional 1D convolution is substantiated by results on synthetic data for spike checker. As illustrated by Figure 3.4, PermSepConv improves upon the 1D convolution network that has the exact same parameter size. We attribute this improvement to the permutation equivariant constraints (regularization) that PermSepConv imposes on the network.

Spike length is set to 32 time samples which is approximately $1.5ms$. For computational efficiency, our spike checker considers only 16 channels out of possible 49. Therefore, our examples have the following dimensions $z \in \mathbb{R}^{16\times32\times2}$. We also use Adam (Kingma et al., 2015) for training our network. Learning rate is chosen by cross validation.

3.5.3 DAGGER Configuration

For training our spike checker classifier we set $N = 60$ in Algorithm 2. Each generated synthetic data has length $60s$ as mentioned earlier. β_i is scheduled to decrease from 1 to 0 by 0.02 decrements per DAGGER iteration. Therefore, our algorithm aggregates data for 10 out of 60 iterations without any input from the expert (ground truth). It is worth noting that we experimented with scheduling the acceptance threshold of spike checker θ as well. Whereby we set $theta = 0.9$ at iteration $j = 1$ and would steadily decrease it to 0.5 by $j = \frac{M}{2}$. Any spikes that is rejected with $\theta \leq 0.5$ will never be proposed again. However, we made the choice of having θ as a constant throughout as it lead to better outcome.

3.5.4 Comparisons with Greedy Deconvolution

Our motivation in introducing enhanced deconvolution method is primarily dealing with large numbers of false assignments that occur by using the greedy method. Recall that these false assignments are due to low SNR signals and/or missing neuron templates. With this in mind, we investigate the performance of greedy and enhanced deconvolution on three distinct synthetic datasets. We use greedy deconvolution with two different acceptance thresholds of 100 and 16 respectively for conservative and permissive settings. Permissive setting leads to better true positives at the cost of substantially increasing false assignments.

First, we focus on the case of low SNR templates. We generate an MEA recording with ≈ 160 neurons and we provide all the neuron templates with $PTP > 4SU$ as dictionaries to both greedy and enhanced deconvolution. Excluding templates with $PTP < 4SU$ generally provides robustness to any MEA deconvolution method as events with such low values of SNR are virtually indistinguishable from background activity. Figure 3.5 depicts the summary of true positive and false assignments for permissive, conservative and DAGGER enhanced greedy deconvolution on the aforementioned MEA recording. Observe that by increasing the threshold (being more conservative) true positive rates are harmed while false assignment rates are decreasing as well more evident in low PTP neuron templates. Note that false assignments are visualized in logarithmic scale. While our enhanced deconvolution obtains virtually the same true positive rates as the conservative greedy method, it greatly improves the False positive rates.

For investigating the case of missing dictionaries, we generate another synthetic dataset similar to the previous setting. However, this time at random we exclude 40 of the synthetic templates from the set that we give to the deconvolution methods. The summary of results are visualized in Figure

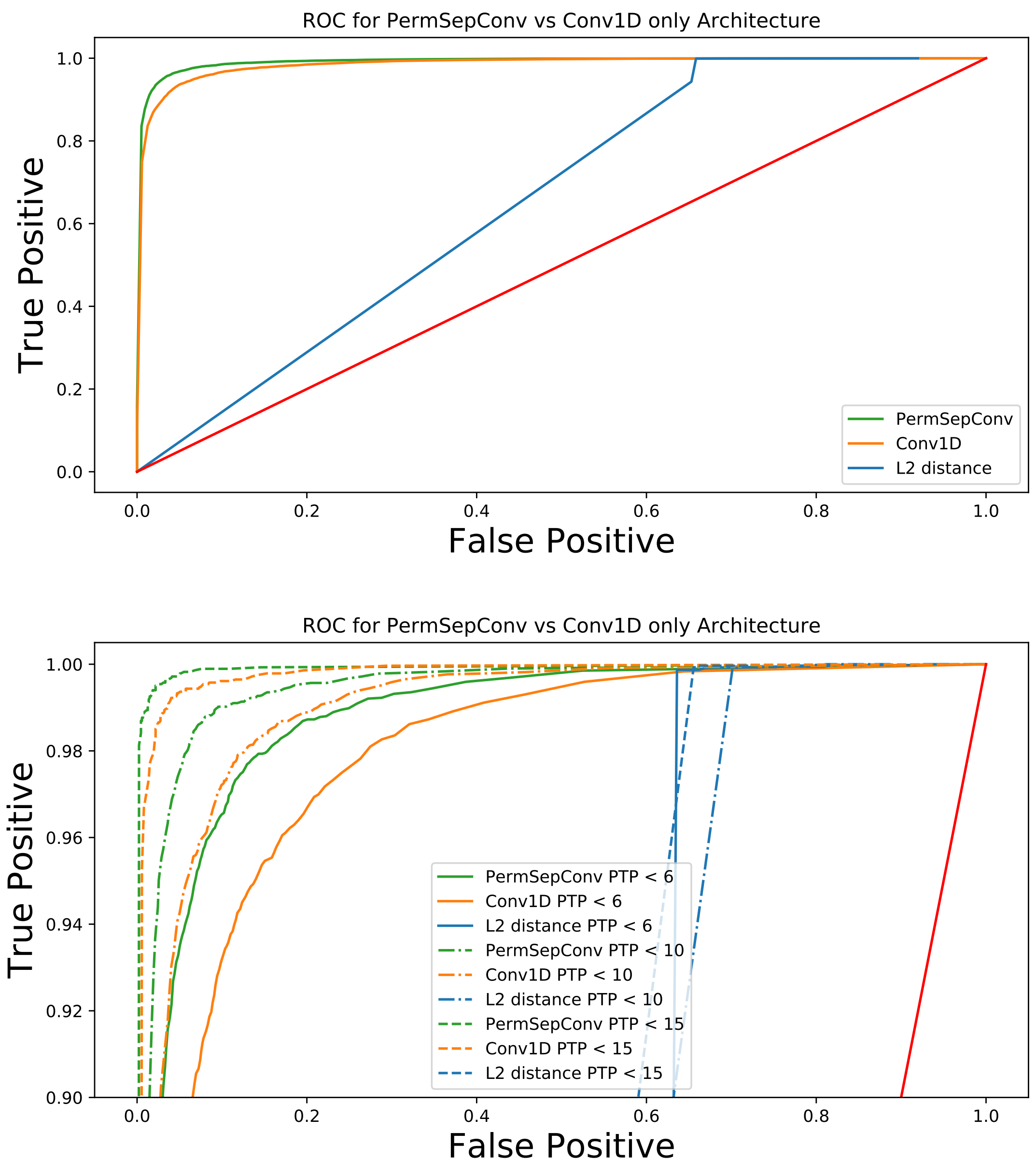


Figure 3.4: **Performance of PermSepConv V.S. conventional 1D convolution on spike checker task.** This plot depicts the ROC for two different spike checkers. One is base on our proposed PermSepConv architecture while the other is based on conventional 1D convolution layers. Both models have been trained using DAGGER algorithm and result is reported for best models according to cross validation on hyper-parameters. It is worth noting that the PermSepConv network has smaller parameter size than the other.

3.6. Although the patterns in true positive and false assignments are similar to our previous setup, the improvements offered by enhanced deconvolution is noticeably less in comparison.

We also generate a hybrid MEA recording by superimposing 40 synthetic neurons on a standardized real MEA recording from primate retina. This serves as another test for generalization of our method to unobserved data. We only provide the 40 templates to the deconvolution methods. Figure 3.7 summarizes the obtained results. The false assignments of enhanced deconvolution are drastically lower than those of permissive and conservative greedy deconvolution. Unlike the two previous recordings, this pattern is consistent across a wide range of template SNR.

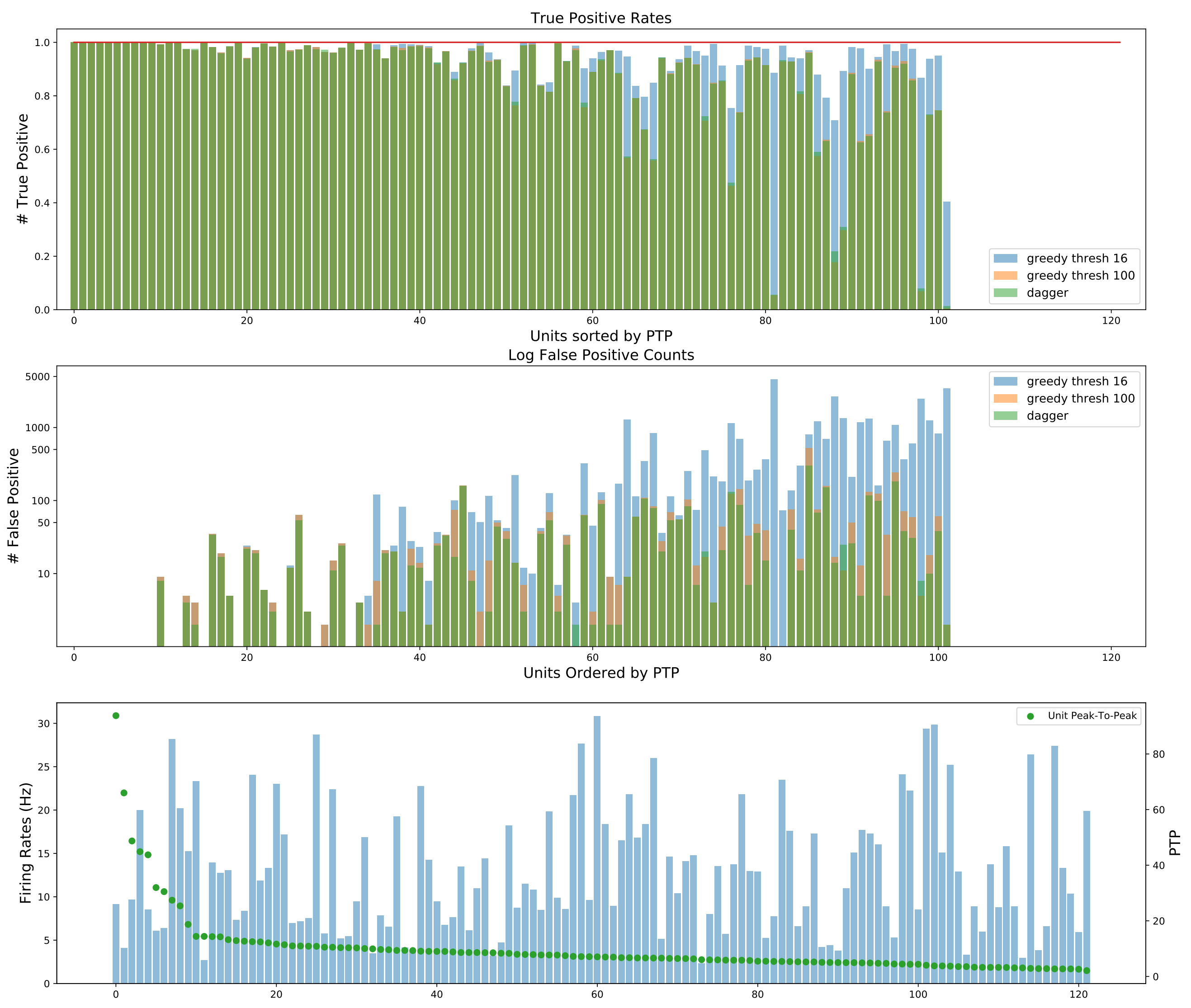


Figure 3.5: **Performance comparison of greedy deconvolution and enhanced deconvolution on synthetic recording.** Top row depicts the true positive rates while middle row visualizes false assignment counts. Note that these counts are visualized on a logarithmic scale. The last row summarizes the PTP and firing rates for each neuron respectively. Observe that by increasing the threshold (to a more conservative setting) true positive rates are harmed while false assignment rates are drastically reduced, however this reduction is not as much as offered by our enhanced deconvolution.

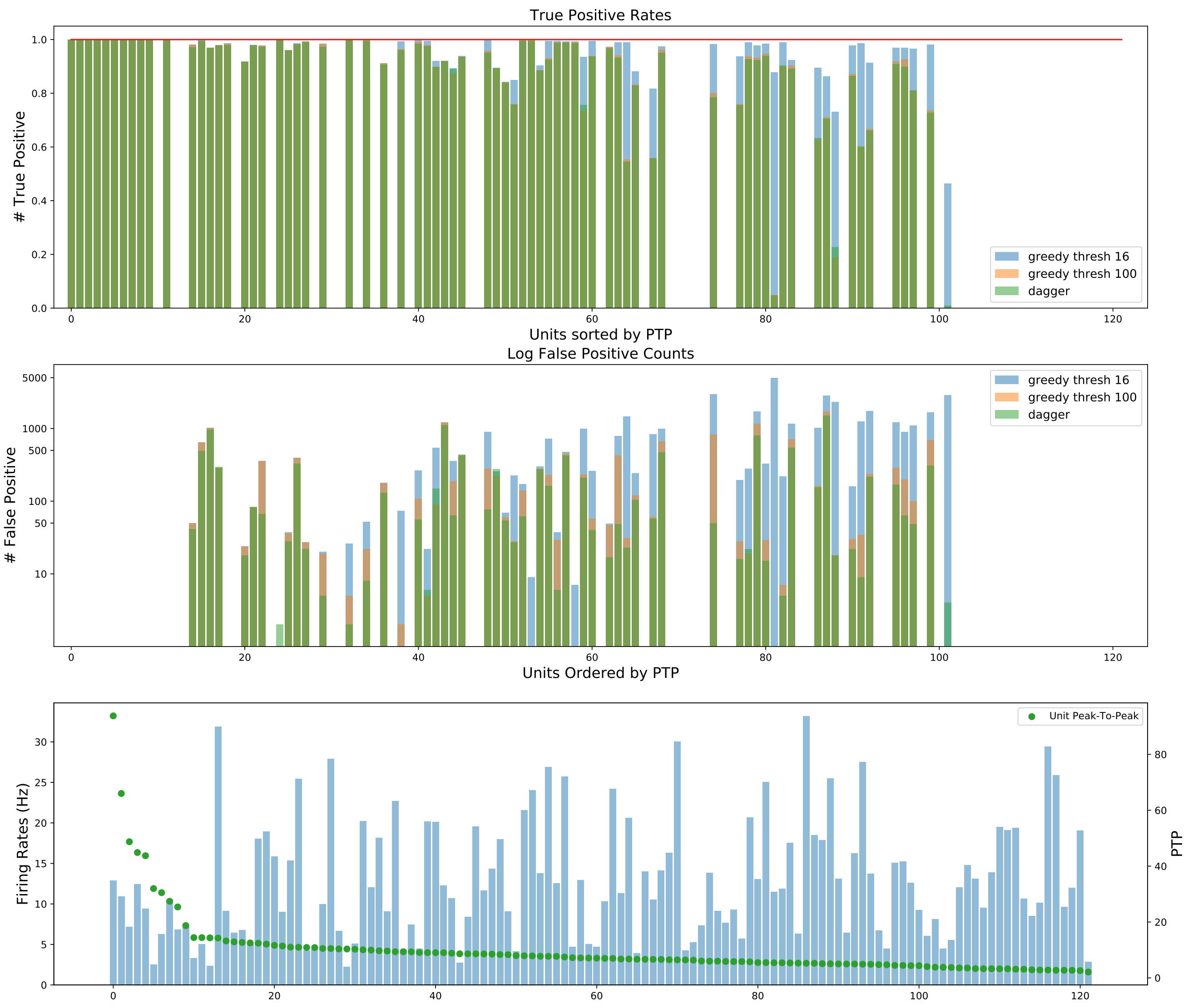


Figure 3.6: **Performance comparison of greedy and enhanced deconvolution with missing dictionaries.** Top row depicts the true positive rates while middle row visualizes false assignment counts. The missing bars from true positive rates correspond to the missing templates. Note that the false assignment counts are visualized on a logarithmic scale. The last row summarizes the PTP and firing rates for each neuron respectively. Similar to the previous case where all the dictionaries were provided, false assignments are decreased by using enhanced deconvolution. However the improvement is less compared to the previous case.

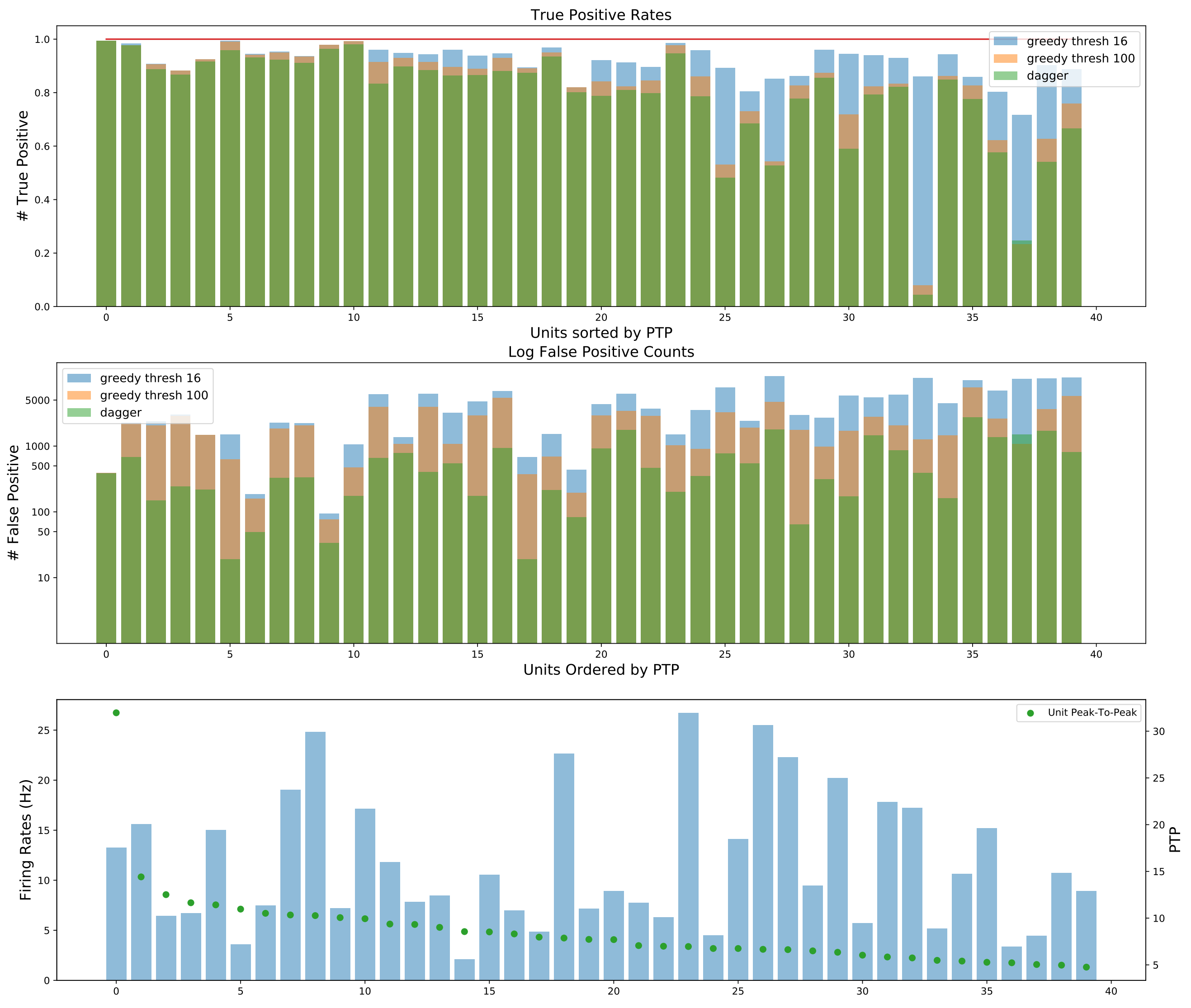


Figure 3.7: **Performance comparison of greedy and enhanced deconvolution on hybrid recording.** Top row depicts the true positive rates while middle row visualizes false assignment counts. Note that these counts are visualized on a logarithmic scale. The improvements in false assignments by enhanced deconvolution are more evident than in the case of synthetic datasets.

Chapter 4: Filtering Normalizing Flows

Dynamical systems are the governing force behind many real-world phenomena and temporal data. For instance, in neuroscience, single neuron voltage recordings are modeled by a set of non-linear differential equations that are variants of the classical Hodgkin-Huxley neuron model. On the other hand, high-dimensional neural population activity, which is assumed to be noisy, redundant observations of latent and low dimensional signals, is often modeled using state-space-models (Paninski et al., 2018). In recent years, a considerable number of generative models alongside inference algorithms have been proposed for neural population modelling. Among these, are temporal structure models (Smith et al., 2003; Yu et al., 2009; Goris et al., 2014), switching dynamical structure models (Petreska et al., 2011; Linderman et al., 2017) and more recently ANN inspired generative models (Archer et al., 2016; Krishnan et al., 2016; Gao et al., 2016; Sussillo et al., 2016; Hernandez et al., 2018) that use AEVB (Kingma et al., 2013) framework to amortize the inference of posterior distributions of latent representations.

In this work we consider a family of state-space models expressed by the following generative process. In the formulation, g, f are smooth differentiable functions, and $\Pi(\theta)$ is a noise distribution (e.g. Gaussian or Poisson respectively for continuous and discrete data) that is governed by parameters θ. The full joint distribution, denoted by $p(\mathbf{X}, \mathbf{Z})$, can be readily computed. This family subsumes well-know models such as *Linear Dynamical System* (LDS) and fLDS (Archer et al., 2016).

$$\begin{aligned}
\mathbf{z}_1 &\sim \mathcal{N}(\mathbf{0}, Q_0) & \\
\mathbf{z}_t|\mathbf{z}_{t-1} &\sim \mathcal{N}\left(g(\mathbf{z}_{t-1}), Q\right) & t = 1, \ldots, T \\
\mathbf{x}_t|\mathbf{z}_t &\sim \Pi(\theta = f(\mathbf{z_t})) & t = 1, \ldots, T
\end{aligned}$$

The choice of approximation distribution family is a determining factor in the success of variational inference. Posterior distributions of dynamical systems are highly complex yet exhibit constrained correlation in temporal dimension. Normalizing flows(Rezende et al., 2015) are expressive deep density estimators well-suited as posterior approximation distributions. In this work we design two normalizing flows with meaningful temporal constraints. Also, we introduce an AEVB method to condition the parameters of the flows on data observations. This conditioning is done such that any latent state of the approximate posterior $\mathbf{z}_t$ depends on a sequence of observations $\mathbf{x}_{1..t+1}$ similar to particle filtering methods. Therefore, we name our method, *Filtering Normalizing Flows* (FNF).

4.1 Normalizing Flows

Normalizing flows create complex distributions out of simpler distributions ,e.g. Gaussian, using bijective non-linear transformations. These transformations $f : \mathbb{R}^d \to \mathbb{R}^d$ map continuous random variables in $\mathbb{R}^d$ to another set of random variables in the same space. Through the change-of-variables theorem, given by equation 4.1, the distribution of the resulting random variables can be computed.

$$\log q(z') = \log p(f^{-1}(z')) + \log \left| \frac{\partial f^{-1}}{\partial z'} \right| = \log p(z) - \log \left| \frac{\partial f}{\partial z} \right| \tag{4.1}$$

A number of normalizing flows such as IAF(Kingma et al., 2016) have analytical inverses due to using simpler maps. These flows require numerous successive application and are computationally expensive. They apply the left hand side of equation 4.1 to model and optimize likelihoods directly. On the other hand, many normalizing flows do not admit analytical inverses. Despite this, they are often more expressive and can be used to construct variational approximation families $q_\phi(z)$ and estimate functions in the form of $\mathbb{E}_{q_\phi(z)}[L(z)]$ (e.g. ELBO) through Monte Carlo samples. In variational setting, application of these expressive densities is an attempt to decrease the gap between the variational objective and the log-likelihood, leading to more accurate and robust learning.

Despite the existence of a variety normalizing flows, the task of applying them to structured models such as latent dynamical systems can be daunting. First, true posterior distributions of such models do not exhibit arbitrary correlation among the latent state space variables. Additionally, conditioning the latent states on observations is not straight forward since $\mathbf{z}_t|\mathbf{z}_{-t}$ is dependent on the entire sequence of observations $\mathbf{x}_{1:T}$. In section 4.2 we introduce two different transformations that condition latent variables on observation such that $\mathbf{z}_t|\mathbf{z}_{-t}$ is dependent on $\mathbf{x}_{1:t}$. We call this property filtering on account of its similarity with particle filtering approaches. Furthermore, these models constrain the flow such that the correlation in time induces smoothness. Unlike DKF, this is achieved without factorizing the variational distribution the same way as the prior.

4.2 Filtering Normalizing Flows

Available normalizing flows, e.g. planar flows (Rezende et al., 2015), are capable of expressing arbitrary correlation in the space of $\mathbf{z} = \mathbf{z}_1\mathbf{z}_2\cdots\mathbf{z}_T \in \mathbb{R}^{d\times T}$. In other words, they operating on the entire space indiscriminately. While we can apply such a transformation on the entire state space, doing so is computationally expensive. Also, conditioning on observations for amortized inference is far from being trivial. Therefore, we seek to constrain the expressiveness of our distribution globally, while keeping it expressive enough locally. Our design choice is also motivated by allowing parameters to admit a trivial choice for conditioning on observations.

4.2.1 Time Autoregressive

Consider the function $f : \mathbb{R}^{2d} \to \mathbb{R}^{2d}$ with a set of parameters denoted by ϕ that transforms a consecutive a pair of latent states z_t, z_{t+1}, thereby, it correlates them with each other. We propose a flow that applies f sequentially to pairs $\mathbf{z}_{1:2}, \mathbf{z}_{2:3}, \ldots, \mathbf{z}_{T-1:T}$ with parameters $\phi_1, \phi_2 \ldots \phi_{T-1}$ respectively. By using this approach, we aim to correlate all the latent variables in a smooth and non-abrupt way with respect to time. Furthermore, as we see in section 4.3 this choice allows us to condition our latent trajectories on the observation in a straightforward way to achieve our filtering goal.

$$F_{\phi,t}(\mathbf{z}) = \begin{bmatrix} \mathbf{z}_{1:t-1} \\ f_\phi(\mathbf{z}_t, \mathbf{z}_{t+1}) \\ \mathbf{z}_{t+2:T} \end{bmatrix} \tag{4.2}$$

$$\left|\frac{\partial F_{\phi,t}}{\partial \mathbf{z}}\right| = \begin{vmatrix} I_{d(t-1)\times d(t-1)} & & \\ & \frac{\partial f}{\partial \mathbf{z_{t:t+1}}} & \\ & & I_{d(T-t-1)\times d(T-t-1)} \end{vmatrix} = \left|\frac{\partial f_\phi}{\partial \mathbf{z}_{t:t+1}}\right| \tag{4.3}$$

$$G(\mathbf{z}_{1:T}; \phi_1, \ldots, \phi_{T-1}) = F_{\phi_{T-1},T-1} \circ \cdots \circ F_{\phi_t,t} \circ \cdots \circ F_{\phi_1,1}(\mathbf{z}_{1:T}) \tag{4.4}$$

The composite flow G that is described by equation 4.4 subsumes the family of factorized models of form $q_1(\mathbf{z}) \prod_{t=2}^{T} q_t(\mathbf{z}_t|\mathbf{z}_{t-1})$ if $f(\mathbf{z}_t, \mathbf{z}_{t+1}) = \mathbf{z}_t, f(\mathbf{z}_{t+1})$ which contains the prior of the generative model. In this work we let $f(.)$ be multi-layer planar flows $f(\mathbf{z}_{t:t+1}) = \mathbf{z}_{t:t+1} + \mathbf{u}h(\mathbf{w}^T\mathbf{z}_{t:t+1} + b)$ which are appropriate choice for achieving expressiveness in low dimensionality of subspaces $\mathbf{z}_{t:t+1}$. For example, the choice of a single layer planar flow with time variant parameters $\{\mathbf{u}_t, \mathbf{w}_t, b_t\}$ gives us the recurrent solutions to the result of the transformation and the the probability that it induces on $\mathbf{z}_{1:T}$ in forms of equations 4.5 and 4.5.

$$\log q(\mathbf{z}') = \log p(\mathbf{z}) - \sum_{t=1}^{T-1} \log(1 + \mathbf{u}_t^\mathsf{T} \mathbf{w}_t h'(\mathbf{w}_t^\mathsf{T} \begin{bmatrix} \mathbf{z}_t'' \\ \mathbf{z}_{t+1} \end{bmatrix} + b_t)$$

Where $\mathbf{z}_t''$ is the intermediate transformation step from $\mathbf{z}_t$ to $\mathbf{z}_t'$ and is described by the equation below.

$$\mathbf{z}_t'' = \mathbf{z}_t + \mathbf{u}_{t-1} h'(\mathbf{w}_{t-1}^\mathsf{T} \begin{bmatrix} \mathbf{z}_{t-1}'' \\ \mathbf{z}_t \end{bmatrix} + b_{t-1}) \text{ for } t = 2, \ldots, T$$

4.2.2 Time Inverse Autoregressive

We also propose a second class of transformations described by equation 4.5 for achieving filtering property and constraining the resulting distribution for time-series data. This transformation is a combination of a fully connected layer and a skip layer where the weights of the fully connected layer form a lower/upper triangular matrix whose inverse is block-bi-diagonal. The rationale for this choice is threefold. First, the Jacobian is lower triangular, therefore, its determinant can be computed efficiently. Second, the inverse of $\mathbf{A}$ need not be computed for the forward transform since it can be solved efficiently and similarly it does not appear in log probability computation using determinant identity $|I + A^{-1}| = |A + I|/|A|$. Finally, this transformation imposes strong constraint on the correlation of the state space variables. This can be viewed as a nonlinear analogue of a Gaussian LDS's posterior samples being reparameterized by affine transformation of an inverse block-bi-diagonal matrix that is Cholesky factor of a block-tri-diagonal precision matrix of the true posterior.

$$\mathbf{z}_{1:T}' = f(\mathbf{z}; \mathbf{A}, \mathbf{b}, \mathbf{c}) = \mathbf{z}_{1:T} + \mathbf{c} \odot h(\mathbf{A}^{-1} \mathbf{z}_{1:T} + \mathbf{b}) \tag{4.5}$$

$$\mathbf{A} = \begin{bmatrix} \mathbf{D}_1 & & & \\ \mathbf{L}_1 & \mathbf{D}_2 & & \\ & \ddots & \ddots & \\ & & \mathbf{L}_{T-1} & \mathbf{D}`_T \end{bmatrix}, \mathbf{b} = \begin{bmatrix} \mathbf{b}_1 \\ \mathbf{b}_2 \\ \vdots \\ \mathbf{b}_T \end{bmatrix} \mathbf{c} = \begin{bmatrix} \mathbf{c}_1 \\ \mathbf{c}_2 \\ \vdots \\ \mathbf{c}_T \end{bmatrix} \tag{4.6}$$

$$\log \left| \frac{\partial f}{\partial \mathbf{z}} \right| = \sum_{t=1}^{T} \sum_{i=1}^{d} \log \left(1 + c_t(i) D_t(i,i) \psi_t(i) \right) - \sum_{t=1}^{T} \sum_{i=1}^{d} \log D_t(i,i) \tag{4.7}$$

In equation 4.7, $\boldsymbol{\psi} = h'(\mathbf{A}^{-1}\mathbf{z} + \mathbf{b})$, where $h(\cdot)$ is a smooth nonlinearity and $h'(\cdot)$ is its derivative. Unlike the previous method, we apply multiple layers of this transformation to construct expressive variational densities. We call this transformation an inverse filtering normalizing flow IFNF.

4.3 Auto-Encoding Variational Bayes

AEVB (Kingma et al., 2013) is a variational framework for training deep generative models of type $p_\theta(\mathbf{x}, \mathbf{z})$ where there is a latent component $\mathbf{z}$. This method has been applied to many different settings including dynamical systems. AEVB optimizes the log likelihood $\mathbb{E}_{\mathbf{x} \sim Data}[p_\theta(\mathbf{x})]$ of the model, which is an intractable marginal distribution, through optimizing a lower bound on it defined by equation 4.8 that requires introduction of $q_\phi(\mathbf{z}|\mathbf{x})$ a variational approximation to true posterior $p_\theta(\mathbf{z}|\mathbf{x})$.

$$\mathcal{L}(\theta, \phi; \mathbf{x}) = \mathbb{E}_{q_\phi(\mathbf{z}|\mathbf{x})}[\log p_\theta(\mathbf{x}, \mathbf{z})] + H[q_\phi(\mathbf{z}|\mathbf{x})] \tag{4.8}$$

Furthermore, AEVB computes unbiased estimates of gradient of this objective with respect to the variational and model parameters θ, ϕ through reparameterization trick described by equation 4.9. This is achieved if samples from the variational distribution can be expressed as reparameterization of some noise samples. In other words $\mathbf{z} = f(\epsilon; \phi) \sim q_\phi(\mathbf{z})$ where $\epsilon \sim p_\epsilon(\epsilon)$ is noise.

$$\nabla_{\theta\phi} \mathbb{E}_{q_\phi(\mathbf{z})}[L_\theta(\mathbf{z})] = \mathbb{E}_{p_\epsilon(\epsilon)}[\nabla_{\theta\phi} L_\theta(f(\epsilon; \phi))] \tag{4.9}$$

ELBO is not a tight bound on the log likelihood because the family of the variational distributions do not contain the true posterior of the models in many cases. Normalizing flows are introduced to increase expressiveness of $q_\phi(\mathbf{z}|\mathbf{x})$ and therefore hoping to tighten the gap and train better models (Rezende et al., 2015).

Our strategy in tackling the task of inference for dynamical systems is to rely on approximate evidence potentials coming from observations at each time steps denoted by $q_\phi^{(0)}(\mathbf{z}_t|\mathbf{x}_t)$. While these potentials can be described by normalizing flows themselves we simply use diagonal Gaussians $q_\phi^{(0)}(\mathbf{z}_t|\mathbf{x}_t) = \mathcal{N}(\mathbf{z}_t|\mu_\phi(\mathbf{x}_t), \sigma_\phi(\mathbf{x}_t))$. Our goal is to train our normalizing flows to mix local potentials with our prior and capture the true posterior better and ultimately learn a better model because of the expressiveness that our normalizing flows lend to our variational approximation.

While theoretically it is possible to mix local potentials without any further conditioning on the observations $\mathbf{x}_1 : T$ given the right invertible transformation, because of the limitations of the normalizing flows in general, we condition our normalizing flows on the observations as a proxy to gain more expressiveness. In the case of FNF time variant parameters $\phi_t(\mathbf{x}_t, \mathbf{x}_{t+1})$ are conditioned on pairs of consequitive observations through a neural network. In the case of IFNF, similarly we the parameters conditioned according to $\mathbf{D}_t(\mathbf{x}_t)$, $\mathbf{L}_t(\mathbf{x}_t, \mathbf{x}_{t+1})$, $\mathbf{b}_t(\mathbf{x}_t)$, $\mathbf{c}_t(\mathbf{x}_t)$. In our experiments we let the parameters of the flows to share networks with the local potential functions.

4.4 Experimental Results

In this section we compare the performance of our proposed method to that of fLDS and DKF which are two of the most widely used inference algorithms for latent dynamics models. We demonstrate through these experiments that FNF/IFNF perform favourably against the state-of-the-art in learning deep dynamical systems. We pay attention to disentagling the dynamics from the nonlinear embeddings, which is crucial for assessing the fit of any dynamics model. For all the results that we report, we use the term DKF to refer to the variational framework of Krishnan et al. (2016) that uses a bidirectional RNN for conditioning the factorized recognition model on the entire sequence of observations.

A quantitative way to asses the fit of state space models are forward extrapolations of the trained model given held out observations. In order to do so we use the trained $q_\phi(\mathbf{z}|\mathbf{x})$ to sample L inferred trajectories $\mathbf{z}_{1:T}^{(l)}$ given held out observations. This is followed by evolving $\mathbf{z}_T^{(l)}$, the last states, according to the trained dynamics model $p_\theta(\mathbf{x}, \mathbf{z})$ K steps forward to obtain L trajectories in the observation space denoted by $\hat{\mathbf{x}}_{T:T+K}^{(l)}$. The metric that we use for the divergence of the forward extrapolation samples from true observations is described by 4.10. Lower divergence metrics show better extrapolations and indicate that the low-dimensional embedding of the emission model is disentangled from the dynamics of the transition model. Also, qualitatively we inspect these extrapolations and compare them with the model samples.

$$\mathrm{MSE}_k = \frac{1}{L}\sum_{l=1}^{L} ||\mathbf{x}_{T+k} - \hat{\mathbf{x}}_{T+k}^{(l)}||_2 \tag{4.10}$$

We monitor and compare the rate of convergence of the evidence lower bound and its value to compare the fit of the models since higher ELBO potentially indicates higher log likelihoods for data.

4.4.1 Latent Lorenz System

The Lorenz system is a classical nonlinear differential equation in 3 independent variables.

$$\begin{aligned}
\frac{d}{dt}z_1 &= \sigma(z_1 - z_2) \\
\frac{d}{dt}z_2 &= z_1(\rho - z_3) - z_2 \\
\frac{d}{dt}z_3 &= z_1 z_2 - \beta z_3
\end{aligned}$$

This is a well studied system with chaotic solutions that serves to clearly demonstrate FNF's advantage for inferring nonlinear dynamics. We generated Euler discretized numerical solutions for a stochastic Lorenz system (Gaussian additive noise of $\mathcal{N}(0, 0.1)$ at each step) from randomly generated initial solutions with parameters $\sigma = 10, \beta = \frac{8}{3}, \rho = 28$. For the purpose of this experiment we generate 100 trials where $T = 60, \mathbf{z_t} \in \mathbb{R}^3, \mathbf{x}_t = f(\mathbf{z}_t) + \epsilon \in \mathbb{R}^{10}$ where f is a single Layer MLP with softplus non-linearity with a skip layer connection to create smooth nonlinear expansions into observation space where $\epsilon \sim \mathcal{N}(0, 0.2)$. We train our models on %80 of the trails and report result on %20 validation set. We train only on 40 time steps and hold out 20 time steps at the end for comparing to extrapolations. We emphasize that stochastic noise exists in both the latent state-space evolution and the observation space. Figure 4.1 demonstrates how FNF is capable of learning complex nonlinear dynamics as well as non-linear embeddings by comparing kMSE of all methods and the quality of their extrapolations. As is demonstrated by figure 4.1, fLDS is not equipped with learning nonlinear dynamics and the extrapolations quickly diverge from the true values. DKF and FNF on the other hand are capable approximating and generalizing the non-linear dynamics and the embeddings. While this is the case, FNF outperforms DKF at this task since the extrapolations are qualitatively and quantitatively closer to the true values.

4.4.2 Rotating MNIST Images

In this task we showcase the ability of our method on synthetic task with and implicit underlying dynamics. Our synthetic dataset consists of 3500 randomly sampled MNIST digits that are binarized and partitioned into 3000/500 for training/testing. Each image is rotated by $\alpha \sim \mathcal{N}(d \cdot \alpha, 2)$ degrees for 20 times where d is the value of the digit corresponding to the image and $\alpha = 18$ is the base rotation value. This way digit 0 rotates by noise and digit 9 rotates by 360 degrees and there digits in between proportional to their value. Beside the test set, we hold out 5 frames from each observation sequence to compare against forward extrapolations of the trained models. In this task we use a slightly different metric for the extrapolations which is the average likelihood of pixels under forecast samples given by equation 4.11.

$$\mathrm{L}_k = \frac{1}{N_{\text{pixels}}} \frac{1}{L} \sum_{N_{\text{pixels}}} \sum_{l=1}^{L} \text{Bernoulli}(\mathbf{x}_{T+k} | \hat{\mathbf{x}}_{T+k}^{(l)}) \tag{4.11}$$

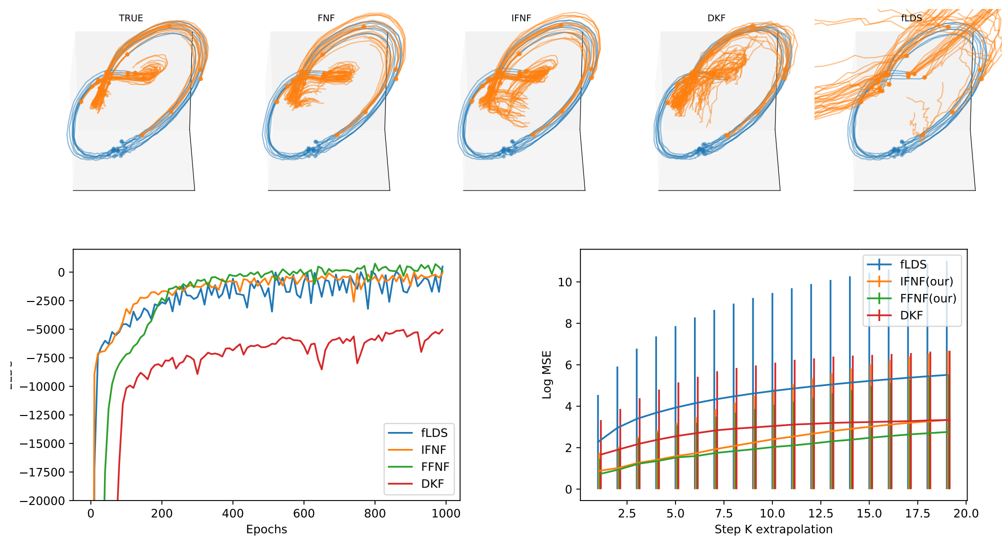


Figure 4.1: **Visualization of and observation trajectories for Lorenz system and performances on them.** (**top**) Respectively, true, FNF, IFNF, DKF, and fLDS reconstructions of the validation set (blue) and stochastic extrapolations into the future time steps (orange) for the true values orange is the hold out. (**bottom right**) kMSE of extrapolations using recovered latent state only. This demonstrate how well FNF disentangles hidden dynamics from embeddings. (**bottom left**) Rate of convergence of models are on par with each other, however FNF attains higher Bounds that could potentially translate into higher evidence values.

This dataset is constructed to demonstrated how our model/inference is capable of recovering meaningful dynamics that translate into meaningful forward samples from the observations. We use two layer MLP with ReLU activation and 20 hidden units with skip layer connection for our transition model. The emission model is fixed across models with ta Bernoulli with rates conditioned on the latent states through MLP with ReLU activation and 50 hidden units for the emission model. The RNNs in DKF use 20 hidden units. Dimensionality of latent state space is set to 10 across all models. The recognition network of fLDS and FNF models use 2 layer MLP with ReLU activation. For FFNF IFNF we respectively use 5 and 3 layers of normalizing flow transformations that share parameters with the recognition network.

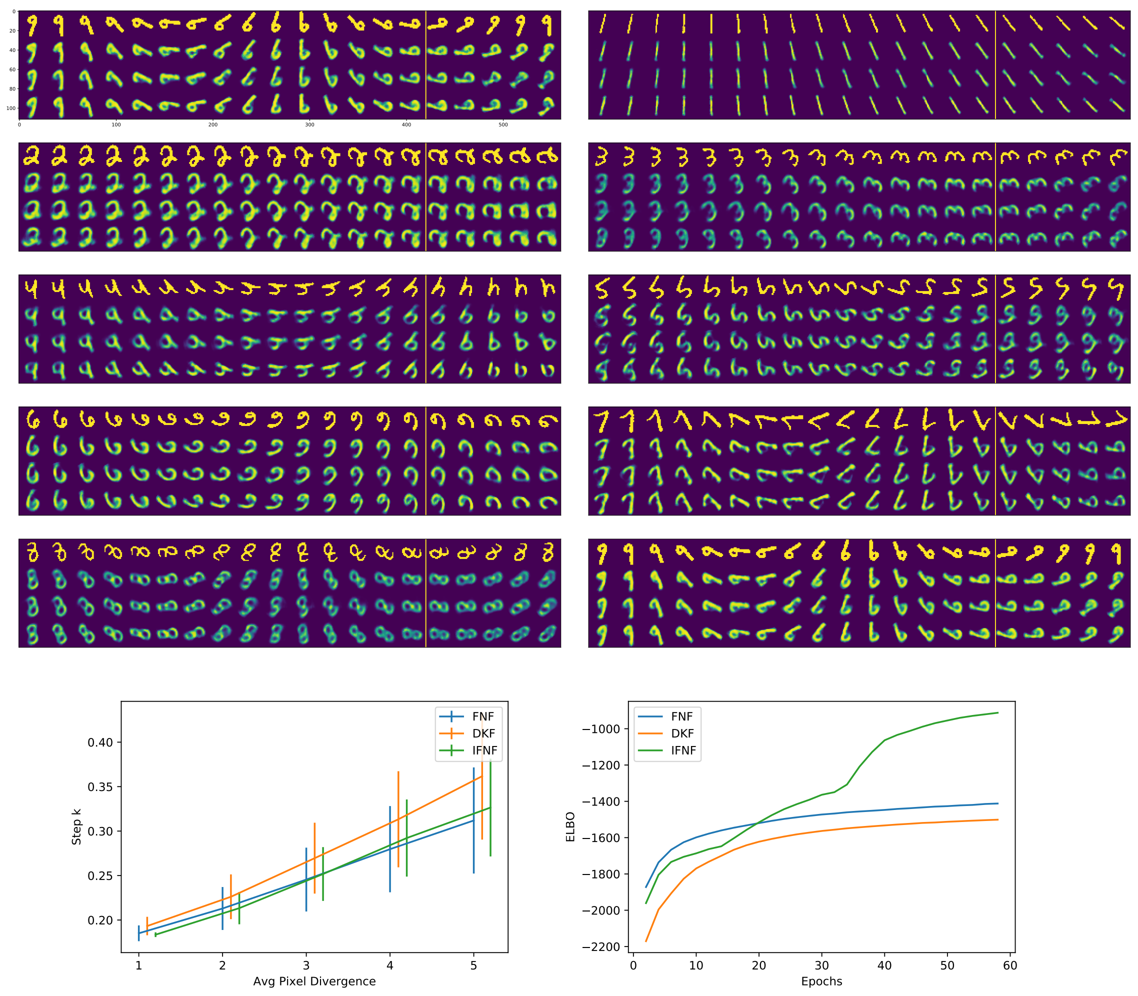


Figure 4.2: **Visualization of reconstructions and forward extrapolations for rotating MNIST images.** (**top**) 10 sequences of digits of test set being reconstructed and forward extrapolated (after solid yellow line). This demostrates how dynamics of rotation and latent representation of digits are separated since each digit has different rotaion value. Top row is true values and following that is respectively FNF, IFNF, and DKF. (**bottom left**) Inferred latent trajectory of digits for each model each color shows a different digit. 0 is blue and 1 is orange. (**bottom right**) ELBO of different models.

References

Jun, James J, Nicholas A Steinmetz, Joshua H Siegle, Daniel J Denman, Marius Bauza, Brian Barbarits, Albert K Lee, Costas A Anastassiou, Alexandru Andrei, Çağatay Aydın, et al. (2017a). "Fully integrated silicon probes for high-density recording of neural activity". In: Nature 551.7679, p. 232.

Franke, Felix, Michal Natora, Clemens Boucsein, Matthias H J Munk, and Klaus Obermayer (2010). "An online spike detection and spike classification algorithm capable of instantaneous resolution of overlapping spikes". In: J. Comp. Neuro. 29.1-2, pp. 127–148.

Carlson, David E, Joshua T Vogelstein, Qisong Wu, Wenzhao Lian, Mingyuan Zhou, Colin R Stoetzner, Daryl Kipke, Douglas Weber, David B Dunson, and Lawrence Carin (2014). "Multichannel electrophysiological spike sorting via joint dictionary learning and mixture modeling". In: IEEE TBME 61.1, pp. 41–54.

Kadir, Shabnam N, Dan F M Goodman, and Kenneth D Harris (2014). "High-dimensional cluster analysis with the masked EM algorithm." In: Neural computation 26.11, pp. 2379–94.

Ekanadham, Chaitanya, Daniel Tranchina, and Eero P Simoncelli (2014). "A unified framework and method for automatic neural spike identification." In: J. Neuro. Methods 222, pp. 47–55.

Muthmann, Jens-Oliver, Hayder Amin, Evelyne Sernagor, Alessandro Maccione, Dagmara Panas, Luca Berdondini, Upinder S Bhalla, and Matthias H Hennig (2015). "Spike detection for large neural populations using high density multielectrode arrays". In: Frontiers in neuroinformatics 9.

Pachitariu, Marius, Nicholas A Steinmetz, Shabnam N Kadir, Matteo Carandini, and Kenneth D Harris (2016). "Fast and accurate spike sorting of high-channel count probes with KiloSort". In: NIPS, pp. 4448–4456.

Yger, Pierre, Giulia LB Spampinato, Elric Esposito, Baptiste Lefebvre, Stephane Deny, Christophe Gardella, Marcel Stimberg, Florian Jetter, Guenther Zeck, Serge Picaud, et al. (2016). "Fast and accurate spike sorting in vitro and in vivo for up to thousands of electrodes". In: bioRxiv, p. 067843.

Hilgen, Gerrit, Martino Sorbaro, Sahar Pirmoradian, Jens-Oliver Muthmann, Ibolya Kepiro, Simona Ullo, Cesar Juarez Ramirez, Alessandro Maccione, Luca Berdondini, Vittorio Murino, et al. (2017). "Unsupervised spike sorting for large scale, high density multielectrode arrays". In: Cell Reports 18.10, 2521–2532.

Jun, James Jaeyoon, Catalin Mitelut, Chongxi Lai, Sergey Gratiy, Costas Anastassiou, and Timothy D Harris (2017b). “Real-time spike sorting platform for high-density extracellular probes with ground-truth validation and drift correction”. In: bioRxiv, p. 101030.

Lewicki, M S (1998). “A review of methods for spike sorting: the detection and classification of neural action potentials”. In: Network: Computation in Neural Systems.

Abeles, Moshe and Moise H Goldstein (1977). “Multispike train analysis”. In: Proceedings of the IEEE 65.5, pp. 762–773.

Harris, K D, D a Henze, J Csicsvari, H Hirase, and G Buzsáki (2000). “Accuracy of tetrode spike separation as determined by simultaneous intracellular and extracellular measurements.” In: J. Neurophysiology 84.1, pp. 401–14.

Shoham, Shy, Matthew R Fellows, and Richard A Normann (2003). “Robust, automatic spike sorting using mixtures of multivariate t-distributions”. In: Journal of neuroscience methods 127.2, pp. 111–122.

Hulata, Eyal, Ronen Segev, and Eshel Ben-Jacob (2002). “A method for spike sorting and detection based on wavelet packets and Shannon’s mutual information”. In: Journal of neuroscience methods 117.1, pp. 1–12.

Quiroga, R Quian, Z Nadasdy, and Y Ben-Shaul (2004). “Unsupervised spike detection and sorting with wavelets and superparamagnetic clustering.” In: Neural computation 16.8, pp. 1661–87.

Hurwitz, Cole, Kai Xu, Akash Srivastava, Alessio Buccino, and Matthias Hennig (2019). “Scalable Spike Source Localization in Extracellular Recordings using Amortized Variational Inference”. In: Advances in Neural Information Processing Systems, pp. 4726–4738.

Schmidt, Edward M. (1984). “Computer separation of multi-unit neuroelectric data: a review”. In: J. Neuro. Methods 12.2, pp. 95–111.

Gibson, Sarah, Jack W Judy, and Dejan Markovi (2012). “Spike Sorting: The first step in decoding the brain”. In: IEEE Signal Processing Magazine December 2011, pp. 124–143.

Rey, Hernan Gonzalo, Carlos Pedreira, and Rodrigo Quian Quiroga (2015). “Past, present and future of spike sorting techniques”. In: Brain research bulletin 119, pp. 106–117.

Souza, Bryan C, Vítor Lopes-dos Santos, João Bacelo, and Adriano BL Tort (2019). “Spike sorting with Gaussian mixture models”. In: Scientific reports 9.1, pp. 1–14.

Fournier, Julien, Christian M Mueller, Mark Shein-Idelson, Mike Hemberger, and Gilles Laurent (2016). “Consensus-based sorting of neuronal spike waveforms”. In: PloS one 11.8, e0160494.

Magland, Jeremy F and Alex H Barnett (2015). “Unimodal clustering using isotonic regression: ISO-SPLIT”. In: arXiv preprint arXiv:1508.04841.

Chung, Jason E, 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 (2017). “A fully automated approach to spike sorting”. In: Neuron 95.6, pp. 1381–1394.

Rodriguez, Alex and Alessandro Laio (2014). “Clustering by fast search and find of density peaks”. In: Science 344.6191, pp. 1492–1496.

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

Swindale, Nicholas and Martin Spacek (2014). “Spike sorting for polytrodes: a divide and conquer approach”. In: Frontiers in Systems Neuroscience 8, p. 6.

Pillow, Jonathan W, Jonathon Shlens, E J Chichilnisky, and Eero P Simoncelli (2013). “A model-based spike sorting algorithm for removing correlation artifacts in multi-neuron recordings.” In: PloS one 8.5, e62123.

Ekanadham, C, Daniel Tranchina, and EP Simoncelli (2011). “A blind deconvolution method for neural spike identification”. In: NIPS.

Carlson, D, V Rao, J Vogelstein, and L Carin (2013). “Real-Time Inference for a Gamma Process Model of Neural Spiking”. In: NIPS.

Dhawale, Ashesh K, Rajesh Poddar, Evi Kopelowitz, Valentin Normand, Steffen Wolff, and Bence Olveczky (2015). “Automated long-term recording and analysis of neural activity in behaving animals”. In: bioRxiv, p. 033266.

Calabrese, A and L Paninski (2010). “Kalman filter mixture model for spike sorting of non-stationary data”. In: J. Neurosci. Methods.

Kilosort2 (2019). `https://github.com/MouseLand/Kilosort2`.

IronClust (2020). `https://github.com/flatironinstitute/ironclust`.

Song, Andrew H., Francisco Flores, and Demba Ba (2018). Spike Sorting by Convolutional Dictionary Learning.

Ross, Stéphane, Geoffrey J. Gordon, and J. Andrew Bagnell (2010). “No-Regret Reductions for Imitation Learning and Structured Prediction”. In: CoRR abs/1011.0686.

Kingma, Diederik P and Max Welling (2013). “Auto-encoding variational bayes”. In: arXiv preprint arXiv:1312.611

Goodfellow, Ian, Yoshua Bengio, and Aaron Courville (2016). Deep learning. MIT Press.

Paninski, L and JP Cunningham (2018). "Neural data science: accelerating the experiment-analysis-theory cycle in large-scale neuroscience." In: Current Opinions in Neurobiology 50, pp. 232–241.

Smith, AC and EN Brown (2003). "Estimating a state-space model from point process observations". In: Neural Comput 15, pp. 965–991.

Yu, Byron M, John P Cunningham, Gopal Santhanam, Stephen I Ryu, Krishna V Shenoy, and Maneesh Sahani (2009). "Gaussian-process factor analysis for low-dimensional single-trial analysis of neural population activity." In: Journal of neurophysiology 102.1, pp. 614–35.

Goris, Robbe LT, J Anthony Movshon, and Eero P Simoncelli (2014). "Partitioning neuronal variability". In: Nature Neuroscience.

Petreska, Biljana, Byron M Yu, John P Cunningham, Gopal Santhanam, Stephen I. Ryu, Krishna V Shenoy, and Maneesh Sahani (2011). "Dynamical segmentation of single trials from population neural data". In: Advances in Neural Information Processing Systems 24. Ed. by J. Shawe-Taylor, R. S. Zemel, P. L. Bartlett, F. Pereira, and K. Q. Weinberger. Curran Associates, Inc., pp. 756–764.

Linderman, Scott, Matthew Johnson, Andrew Miller, Ryan Adams, David Blei, and Liam Paninski (2017). "Bayesian Learning and Inference in Recurrent Switching Linear Dynamical Systems". In: Proceedings of the 20th International Conference on Artificial Intelligence and Statistics. Ed. by Aarti Singh and Jerry Zhu. Vol. 54. Proceedings of Machine Learning Research. Fort Lauderdale, FL, USA: PMLR, pp. 914–922.

Archer, Evan, Il Memming Park, Lars Buesing, John Cunningham, and Liam Paninski (2016). "Black box variational inference for state space models". In: arXiv preprint arXiv:1511.07367.

Krishnan, Rahul G., Uri Shalit, and David Sontag (2016). "Deep Kalman Filters". In: arXiv preprint arXiv:1511.051

Gao, Yuanjun, Evan W Archer, Liam Paninski, and John P Cunningham (2016). "Linear dynamical neural population models through nonlinear embeddings". In: Advances in Neural Information Processing Syst
Ed. by D. D. Lee, M. Sugiyama, U. V. Luxburg, I. Guyon, and R. Garnett. Curran Associates, Inc., pp. 163–171.

Sussillo, David, Rafal Józefowicz, L. F. Abbott, and Chethan Pandarinath (2016). "LFADS - Latent Factor Analysis via Dynamical Systems". In: CoRR abs/1608.06315. arXiv: 1608.06315.

Hernandez, Daniel, A. Moretti, Ziqiang Wei, S. Saxena, John Cunningham, and Liam Paninski (2018). "A Novel Variational Family for Hidden Nonlinear Markov Models". In: CoRR abs/1811.02459.

Rezende, Danilo and Shakir Mohamed (2015). "Variational Inference with Normalizing Flows". In: Proceedings of the 32nd International Conference on Machine Learning. Ed. by Francis Bach and David Blei. Vol. 37. Proceedings of Machine Learning Research. Lille, France: PMLR, pp. 1530–1538.

Johnson, Matthew, David K Duvenaud, Alex Wiltschko, Ryan P Adams, and Sandeep R Datta (2016). "Composing graphical models with neural networks for structured representations and fast inference". In: Advances in Neural Information Processing Systems 29. Ed. by D. D. Lee, M. Sugiyama, U. V. Luxburg, I. Guyon, and R. Garnett. Curran Associates, Inc., pp. 2946–2954.

Le, Tuan Anh, Maximilian Igl, Tom Rainforth, Tom Jin, and Frank Wood (2018). "Auto-Encoding Sequential Monte Carlo". In: International Conference on Learning Representations.

Naesseth, Christian A., Scott W. Linderman, Rajesh Ranganath, and David M. Blei (2018). "Variational Sequential Monte Carlo". In: AISTATS.

Miller, Jeffrey W and Matthew T Harrison (2018). "Mixture models with a prior on the number of components". In: Journal of the American Statistical Association.

Hughes, Michael C. and Erik Sudderth (2013). "Memoized Online Variational Inference for Dirichlet Process Mixture Models". In: NIPS, pp. 1133–1141.

Pachitariu, Marius (2019). kilosort2. `https://github.com/MouseLand/Kilosort2`.

Parthasarathy, Nikhil, Eleanor Batty, William Falcon, Thomas Rutten, Mohit Rajpal, EJ Chichilnisky, and Liam Paninski (2017). "Neural networks for efficient bayesian decoding of natural images from retinal neurons". In: Advances in Neural Information Processing Systems, pp. 6434–6445.

Yoon, Young-Gyu, Peilun Dai, Jeremy Wohlwend, Jae-Byum Chang, Adam H Marblestone, and Edward S Boyden (2017). "Feasibility of 3D reconstruction of neural morphology using expansion microscopy and barcode-guided agglomeration". In: Frontiers in computational neuroscience 11, p. 97.

Weigert, Martin, Uwe Schmidt, Tobias Boothe, Andreas Müller, Alexandr Dibrov, Akanksha Jain, Benjamin Wilhelm, Deborah Schmidt, Coleman Broaddus, Siân Culley, et al. (2018). "Content-aware image restoration: pushing the limits of fluorescence microscopy". In: Nature methods 15.12, p. 1090.

Sun, Ruoxi and Liam Paninski (2018). "Scalable approximate Bayesian inference for particle tracking data". In: Proceedings of the 35th International Conference on Machine Learning.

Knox, Edwin M and Raymond T Ng (1998). "Algorithms for Mining Distance-Based Outliers in Large Datasets". In: VLDB. Citeseer, pp. 392–403.

Zhang, Tong (2011). "Adaptive forward-backward greedy algorithm for learning sparse representations". In: IEEE transactions on information theory 57.7, pp. 4689–4708.

Hartigan, John A, Pamela M Hartigan, et al. (1985). "The dip test of unimodality". In: The annals of Statistics 13.1, pp. 70–84.

Rossant, Cyrille, Shabnam N Kadir, Dan FM Goodman, John Schulman, Mariano Belluscio, Gyorgy Buzsaki, and Kenneth D Harris (2015). "Spike sorting for large, dense electrode arrays". In: bioRxiv, p. 015198.

Gregor, Karol and Yann LeCun (2010). "Learning Fast Approximations of Sparse Coding." In: ICML. Omnipress, pp. 399–406.

Sulam, Jeremias, Vardan Papyan, Yaniv Romano, and Michael Elad (2017). "Multi-Layer Convolutional Sparse Modeling: Pursuit and Dictionary Learning". In: CoRR abs/1708.08705.

Chambolle, Antonin, R.A. Vore, Nam-Yong Lee, and Bradley Lucier (Apr. 1998). "Nonlinear wavelet image processing: Variational problems, compression, and noise removal through wavelet shrinkage". In: Image Processing, IEEE Transactions on 7, pp. 319 –335.

Kiranyaz, Serkan, Turker Ince, Hamila Ridha, and Moncef Gabbouj (Aug. 2015). "Convolutional Neural Networks for Patient-Specific ECG Classification". In:

Ince, Turker, Serkan Kiranyaz, and Moncef Gabbouj (June 2009). "A Generic and Robust System for Automated Patient-Specific Classification of ECG Signals". In: Biomedical Engineering, IEEE Transactions on 56, pp. 1415 –1426.

Yu, Yang, Chaoyue Wang, Xiaoyu Gu, and Jianchun Li (2019). "A novel deep learning-based method for damage identification of smart building structures". In: Structural Health Monitoring 18.1, pp. 143–163.

Zaheer, Manzil, Satwik Kottur, Siamak Ravanbakhsh, Barnabas Poczos, Russ R Salakhutdinov, and Alexander J Smola (2017). "Deep Sets". In: Advances in Neural Information Processing Systems 30. Ed. by I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett. Curran Associates, Inc., pp. 3391–3401.

Kingma, Diederik and Jimmy Ba (2015). "Adam: A method for stochastic optimization". In: ICLR.

Kingma, Durk P, Tim Salimans, Rafal Jozefowicz, Xi Chen, Ilya Sutskever, and Max Welling (2016). "Improved Variational Inference with Inverse Autoregressive Flow". In: Advances in Neural Information Proces
Ed. by D. D. Lee, M. Sugiyama, U. V. Luxburg, I. Guyon, and R. Garnett. Curran Associates, Inc., pp. 4743–4751.

Hoffman, Matthew and David Blei (2015). "Stochastic structured variational inference". In: Artificial Intelligence ar
pp. 361–369.

De Boor, Carl (2001). A practical guide to splines. Springer.

Appendix A: Notation

We use the following conventions: scalars are lowercase italicized letters, e.g. x, constants such as max indices are represented by uppercase italicized letters, e.g. N, vectors are bolded lowercase letters, e.g. $\mathbf{x}$, and matrices are bolded uppercase letters, e.g. $\mathbf{X}$. Major notations used in the paper are summarized in Table A.1.

Notation	Explanation	Default value (if exists)
Data Constants		
T	Recording length (in timesteps)	
C	Total number of channels	
N	Number of detected spikes	
C_{neigh}	Number of neighboring channels (including the main channel)	
Parameters		
R	Temporal window size of spike	5 ms
R_{NN}	Temporal window size of spike used for NN detection and denoising	3 ms
τ	Threshold for spike detection under NN ouput	0.5
Data structures		
$\mathbf{V} \in \mathbb{R}^{T \times C}$	Recording	
$\mathbf{P} \in (0,1)^{T \times C}$	Sigmoid output of Neural Network Detection	
$\mathbf{X}, (\mathbf{X}_n)$	Waveform (of spike n)	
$\mathbf{W}_k$	Template of unit k	
Notations for MFM		
K	Number of components in the generative model	
$\pi = (\pi_1 \ldots \pi_k)$	Component weights in the generative model	
$\mu_k, \mathbf{\Lambda}_k$	Mean and precision matrix of component k	
$\mathbf{z} = (z_1, \ldots, z_n)$	Component membership	
$\mu_0, \lambda_0, \mathbf{V_0}, \nu_0$	Prior parameters for the Normal-Whishart	$\mathbf{0}, 0.01, \frac{1}{\nu_0}\mathbf{I}_D, D+2$ D: # of feature dims.
α	Prior concentration parameters for Dirichlet dist.	1
β	Prior parameter for Poisson Distribution	1
$\hat{K}$	Variational parameter for the point mass	
$\hat{\alpha}_k$	Variational concentration parameters for the posterior Dirichlet Distribution	
$\hat{\mu}_k, \hat{\lambda}_k, \hat{\mathbf{V}}_k, \hat{\nu}_k$	Variational parameters for the posterior Normal-Whishart	

Table A.1: **Summary table of notation used within the appendix.**

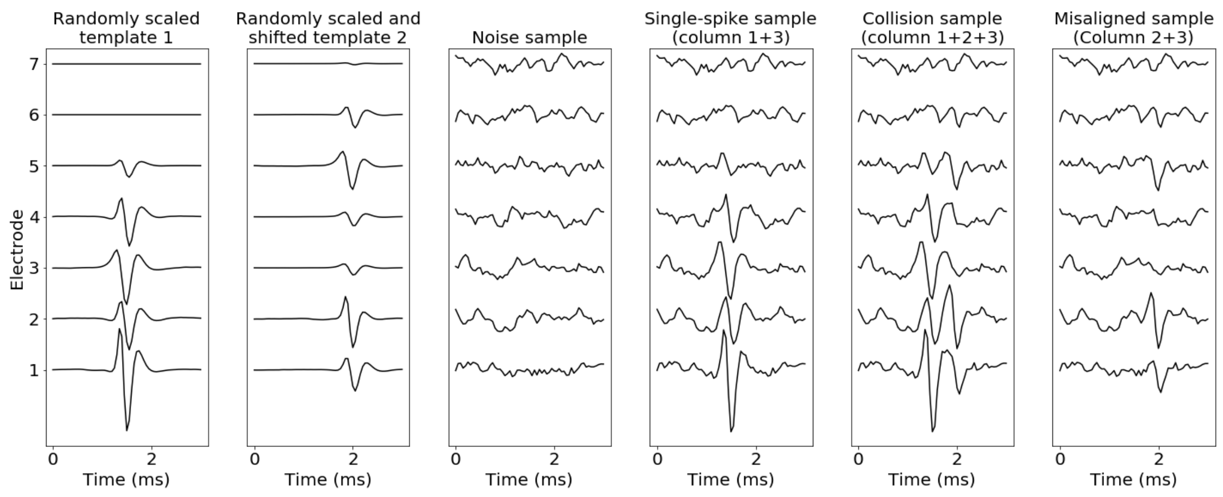


Figure B.1: **Illustration of Construction of NN Training Data**. (Column 1) One template is randomly chosen from a previous sort and randomly scaled. This template is centered temporally. (Column 2) Another template may be randomly chosen, randomly scaled, and randomly shifted. (Column 3) A Gaussian noise sample (both temporally and spatially correlated) is drawn. (Column 4) A single spike sample is created by superimposing template 1 and noise. (Column 5) A collision sample is constructed by adding template 1, template 2, and noise. No more than 2 templates are used for creating collision samples. (Column 6) A misaligned sample is created by adding noise to a randomly shifted template. When the convolutional neural network is applied to the recording, our goal is to output no duplicate detection of the same spike. Thus, during training, samples such as column 4 and 5 have label 1 and samples such as column 3 and 6 have label 0.

Appendix B: Additional details on the detection algorithm

B.0.1 Neural network training data

See Figure B.1 for an overview of how we construct each training sample for the detection neural network.

We start with a library of clean templates from previous sorts. (Since the detection network operates locally in space and time, we only need cleaned templates on local 7-electrode patches here: a center electrode and the six surrounding electrodes.) We choose a template randomly, scale it randomly, and center it, and then (optionally) corrupt this clean template with a collision from a different template (with a random scale and shift). (We found that it was not necessary to form collisions using >2 spikes.)

Next we add indepedent noise samples to these templates. For this we need to construct a distribution over noise samples of dimension $R_{NN} \times C_{neigh} = 61 \times 7$ here. Following previous work Pillow et al., 2013 we use a spatiotemporally separable, stationary Gaussian noise model. We find “noise" snippets in the raw data by excluding any spikes detected by simple thresholding with a low threshold (3 standardize units). We then rescale these snippets so that their standard deviation has unit standardized units and compute the temporal and spatial covariance matrix. We approximate the resulting temporal covariance with a Toeplitz (temporally stationary) matrix, and the spatial covariance with a rotation-symmetric matrix (to enforce spatial stationarity), then form the Kronecker product to obtain the full spatiotemporal covariance matrix.

Given an unlimited stream of samples from the above generative model, we train the neural network detector is trained as a classifier that takes as input waveforms $\mathbf{X} \in \mathbb{R}^{R_{NN} \times C_{neigh}}$, and returns a soft-max output between $(0, 1)$ that estimates the presence of a spike. Single-spike and collision samples have positive labels; noise and off-centered samples have negative labels. We use an equal proportion of positive and negative labels, and the misaligned spike is added 70% of the time to reflect the high collision rate of retina recordings.

B.0.2 Neural network structure

The proposed architecture is the detector is shown in Figure B.2. To build a faster network, temporal and spatial filters are separated. The first two layers extract temporal features and the third layer combines the features in spatially neighboring channels. The first layer applies a one-dimensional convolutional neural network using K_1 filters of size R_{NN}. The second layer maps K_1 features to K_2 features at each location. The third layer transforms $C_{neigh} \times K_2$ features into a single number. Rectified linear unit nonlinearities (ReLU) are used as activation functions and the sigmoid transform is applied at the end to get a value in $(0, 1)$. The network is learned by minimizing the cross-entropy loss using the Adam update rule (Kingma et al., 2015). For the retinal recordings

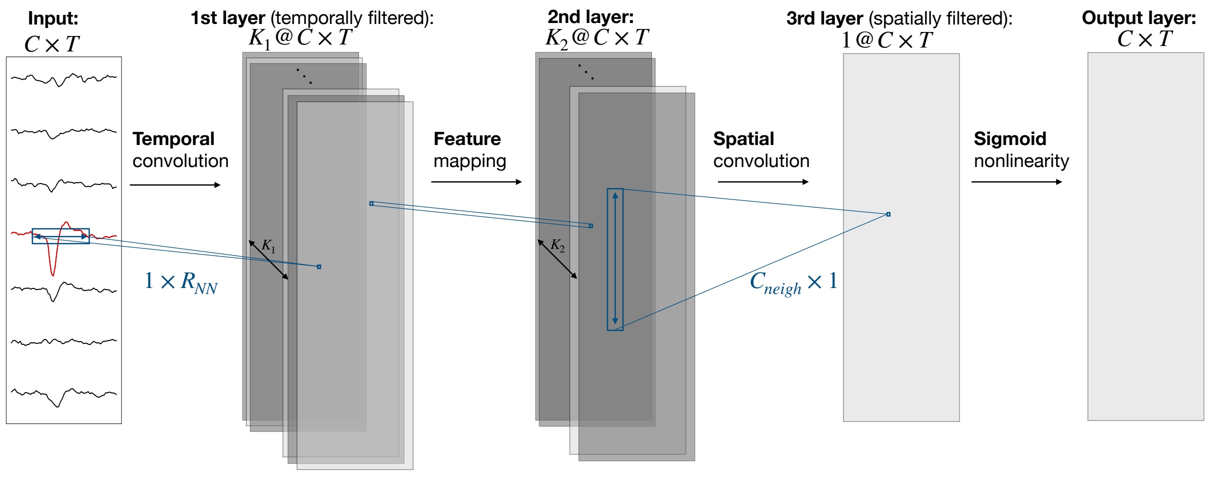


Figure B.2: Visualization of the architecture of Neural Network detector. ReLUs are used as activation functions.

analyzed here, K_1 and K_2 are set to 16 and 8. The learning rate of Adam is set to 0.0001.

After training, given a recording, $\mathbf{V} \in \mathbb{R}^{T \times C}$, the neural network outputs an array, $\mathbf{P} \in (0,1)^{T \times C}$, where $\mathbf{P}_{(t,c)}$ is the output of the neural network (applied convolutionally in space and time) with input a temporal and spatial snippet of $\mathbf{V}$ around t and c. Given the output, $\mathbf{P}$, temporal and spatial local maxima that cross a threshold, τ, are considered as detected events. As a result, the detector outputs pairs of spike times and corresponding channels, $(t_1, c_1), \ldots, (t_N, c_N)$.

Appendix C: Additional details on the denoising algorithm

The training data for the neural network denoiser is constructed as above for the neural network detector. Instead of the 0-1 label output by the detector, the denoiser output targets the ground truth template without any noise or collision superimposed.

We found that denoising on single channels already gave good performance. Thus the denoiser takes a waveform of size R_{NN} as input. The network consists of three hidden layers with one-dimensional convolutional neural network and ReLU activation functions. At the end, a linear mapping is applied to output a vector of size R_{NN}. The NN is trained to minimize the L2 distance between the output and the ground truth clean template. For the retina, the three hidden layers have 16, 8, and 4 filters of size 5, 11, 21 respectively. We use the same training parameters as described for the detector network above.

Appendix D: Additional details on the Mixture of Finite Mixture (MFM) model

In the clustering step we use a Mixture of Finite Mixture model (MFM) described in Miller et al., 2018. To estimate the posterior distribution we use a structured variational distribution (Hoffman et al., 2015) which preserves dependencies between parameters, leading to a more accurate posterior approximation than simpler unstructured variational approximations. Finally, we use split-merge steps based on Hughes et al., 2013 to estimate a reasonable number of clusters.

D.0.1 Generative Model

We assume the following generative model (Miller et al., 2018):

$$\begin{aligned}
K &\sim \text{Poisson}(\beta)\\
(\pi|K) &\sim Dir(\alpha, \dots, \alpha)\\
(\mu_k, \mathbf{\Lambda}_k|K) &\sim_{iid} \mathcal{NW}(\mu_0, \lambda_0, \mathbf{V}_0, \nu_0) \quad \text{for } k = 1:K\\
(z_n|\pi) &\sim_{iid} Cat(\pi_1, \dots, \pi_K) \quad \text{for } n = 1:N\\
(\mathbf{X}_n|z_n, \mu, \mathbf{\Lambda}) &\sim_{ind} \mathcal{N}(\mu_{z_n}, \mathbf{\Lambda}_{z_n}) \quad \text{for } n = 1:N.
\end{aligned}$$

The joint distribution of $K, \pi, \mu, \mathbf{\Lambda}, \mathbf{z}, \mathbf{X}$ is given by:

$$p(K, \pi, \mu, \mathbf{\Lambda}, \mathbf{z}, \mathbf{X}) = p(K|\beta)p(\pi|K, \alpha)\prod_{k=1}^{K} p(\mu_k, \mathbf{\Lambda}_k|\mu_0, \lambda_0, \mathbf{V}_0, \nu_0)\prod_{n=1}^{N} p(z_n|\pi)p(\mathbf{X}_n|z_n, \mu_{z_n}, \mathbf{\Lambda}_{z_n}).$$

D.0.2 Inference model and estimation

The following structured variational distribution (Hoffman et al., 2015) is used for inference:

$$\begin{aligned} q(K) &\sim \delta_{\hat{K}}(K) \\ q(\pi|K) &\sim Dir(\hat{\alpha}_1,\ldots,\hat{\alpha}_K) \\ q(\mu_k,\mathbf{\Lambda}_k|K) &\sim_{ind} \mathcal{NW}(\mu_k,\mathbf{\Lambda}_k|\hat{\mu}_k,\hat{\lambda}_k,\hat{\mathbf{V}}_k,\hat{\nu}_k) \quad \text{for } k=1:K \\ q(z_n|\mu_{(1:K)},\mathbf{\Lambda}_{(1:K)},\pi,K) &= p(z_n|\mathbf{X}_n,\mu_{(1:K)},\mathbf{\Lambda}_{(1:K)},\pi,K) \quad n=1:N. \end{aligned}$$

Also, the exact conditional is used for $q(z_n|\mu_{(1:K)},\mathbf{\Lambda}_{(1:K)},\pi,K)$, since it is tractable for the Gaussian mixture model. Given the global parameters, the conditional can be computed as:

$$p(z_n=k|\mathbf{X}_n,\mu_{(1:K)},\mathbf{\Lambda}_{(1:K)},\pi,K)=\frac{p(\mathbf{X}_n|\mu_{z_n},\mathbf{\Lambda}_{z_n})\pi_{z_n}}{\sum_{k=1}^{K}p(\mathbf{X}_n|\mu_k,\mathbf{\Lambda}_k)\pi_k}.$$

In summary, the variational joint distribution is then given by:

$$q(K,\pi,\mu,\mathbf{\Lambda},\mathbf{z})=\delta_{\hat{K}}(K)q(\pi|K,\hat{\alpha}_1,\ldots,\hat{\alpha}_{\hat{K}})\prod_{k=1}^{K}q(\mu_k,\mathbf{\Lambda}_k|\hat{\mu}_k,\hat{\lambda}_k,\hat{\mathbf{V}}_k,\hat{\nu}_k)\prod_{n=1}^{N}p(z_n|\mathbf{X}_n,\mu_{(1:K)},\mathbf{\Lambda}_{(1:K)},\pi,K).$$

The variational parameters are estimated by maximizing the Evidence Lower Bound (ELBO):

$$\mathcal{L}(q)=\mathbb{E}_q\Big[p(K,\mu_{(1:K)},\mathbf{\Lambda}_{(1:K)},\pi,\mathbf{z},\mathbf{X})-q(K,\mu_{(1:K)},\mathbf{\Lambda}_{(1:K)},\pi,\mathbf{z})\Big].$$

To estimate the variational parameters of $\mu_{(1:K)},\mathbf{\Lambda}_{(1:K)},\pi$, we use an approach similar to SSVI-A in Hoffman et al., 2015. However, instead of stochastic updates, where $\mu_{(1:K)},\mathbf{\Lambda}_{(1:K)},\pi$ are randomly sampled to estimate $q(z_n|\mu_{(1:K)},\mathbf{\Lambda}_{(1:K)},\pi,K)$, we use MAP estimation. This simplification is reasonable as the variational distributions are likely to have small variance due to large sample size. To infer the number of components, K, the exact birth and merge approach in Hughes et al., 2013 is used.

Appendix E: Additional details on spline interpolation within deconvolution

As noted in the main text, subsample alignment errors between templates and the observed sampled data can lead to large residuals in the deconvolution step. In order to correct for these alignment error, we begin by estimating the temporal offset between the true versus sampled objective function peaks. To estimate the true peak locations, we fit a quadratic function to the samples around each detected peak and solve for the location where the maximum is obtained. With this information, the convolved templates can be shifted via interpolation prior to subtraction from the objective in order to align their peaks with the estimated peaks.

We use a Basis Spline (De Boor, 2001) representation of each convolved template. Bsplines are capable of representing any spline function of a given order with sparsely supported basis functions whose support grows linearly with the order of the spline representation (i.e., the complexity of evaluation at a single location remains constant in the length of the templates). For example, a third order or “cubic" spline interpolation can be performed with the evaluation of only 4 basis functions at any given location, no matter the length of the function or number of knots. Therefore on the fly interpolation of a template represented as a combination of Bspline bases retains both the accuracy of standard spline interpolation and the linear computational complexity of linear interpolation. Moreover, evaluation of Bpline basis functions can be performed efficiently on the fly and lends itself to fast implementation on the GPU. We have implemented this functionality and in practice the cost incurred by performing interpolated subtraction is negligible relative to uncorrected subtraction.