New tools for electrophysiological data analysis and their application to a working memory study

vorgelegt von
Dipl.-Inform.
Robert Pröpper
geb. in Würzburg

von der Fakultät IV - Elektrotechnik und Informatik
der Technischen Universität Berlin
zur Erlangung des akademischen Grades

Doktor der Naturwissenschaften
- Dr. rer. nat. -

genehmigte Dissertation

Promotionsausschuss:

Vorsitzender:	Prof. Dr. Manfred Opper
Gutachter:	Prof. Dr. Klaus Obermayer
Gutachter:	PD Dr. Matthias H. J. Munk
Gutachter:	Prof. Dr. Gaute T. Einevoll

Tag der wissenschaftlichen Aussprache: 3. September 2015

Berlin 2015

Abstract

In order to understand how signals in the brain are related to observed behavior, it is essential to observe the activity of single neurons. Electrophysiology is one of the oldest and most important means to achieve recordings of the voltage fluctuations produced by active neurons. In extracellular electrophysiology, recording electrodes are placed in the intercellular medium and often record from multiple neurons at the same time. A central issue is to determine which neuron produced each recorded spike, solutions for this problem are called spike sorting.

The steady improvement in electrophysiological recording techniques requires advances in data analysis: the increasing amounts of recorded data need to be managed and appropriate analysis algorithms for large data sets need to be developed. As experiments grow more complex, interdisciplinary cooperation becomes more important. This necessitates data sharing and collaborative development of analyses. The first part of this thesis is concerned with a solution to the challenges arising from this trend: Spyke Viewer is a software platform for electrophysiological data management and analysis that supports many different data formats in a unified way. It is focused on usability and flexibility, so it is useful to both experimenters who can use it to browse and visualize data and theoreticians who develop new analysis algorithms.

With newer electrophysiology techniques, where multiple recording channels capture activity from an increasing amount of neurons, recorded spikes often overlap in time. The resulting waveforms are a particular challenge for spike sorting methods. In this thesis, a spike sorting algorithm that addresses the overlap problem is improved and evaluated on simulated and empirical data. In addition, a complete spike sorting pipeline from raw data to sorted spikes is described.

All methods were developed and tested using an empirical data set recorded from the prefrontal cortex of macaque monkeys. The monkeys performed a visual working memory task. Using Spyke Viewer and the improved spike sorting algorithm, large scale analyses on the coding of visual stimuli and experimental conditions were carried out. Reactions of individual neurons were examined and population codes were explored using a variety of decoding methods. Using time-resolved analyses, it was found that the neural coding of all experimental conditions changes quickly over the course of a trial, but sample stimuli and test stimuli elicit very similar neural response patterns at different times during the trial.

Zusammenfassung

Um zu verstehen, wie neuronale Signale mit Verhalten zusammenhängen, ist es essentiell die Aktivität von einzelnen Neuronen zu untersuchen. Elektrophysiologie ist eine der ältesten und wichtigsten Techniken um die Spannungsfluktuationen aufzunehmen, die von aktiven Neuronen produziert werden. In extrazellulärer Elektrophysiologie werden Elektroden im interzellulären Medium platziert. Diese Elektroden nehmen oft mehrere Neuronen gleichzeitig auf. Ein zentrales Problem ist es, herauszufinden welche aufgenommenen Spikes von welchen Neuronen produziert wurden. Lösungen für dieses Problem sind bekannt als Spike Sorting.

Die kontinuierlichen Verbesserungen von elektrophysiologischen Messtechniken machen Verbesserungen in der Datenanalyse notwendig: Die wachsenden Mengen an aufgenommen Daten müssen verwaltet werden und passende Analysealgorithmen für große Datensätze müssen entwickelt werden. Weil die Experimente komplexer werden, wird interdisziplinäre Kooperation wichtiger, was zu einem gesteigerten Interesse an neuen Methoden zur Datenfreigabe und kollaborativer Analyseentwicklung führt. Der erste Teil dieser Arbeit widmet sich einer Lösung für die Herausforderungen, die dieser Trend aufwirft: Spyke Viewer ist eine Softwareplattform zum Verwalten und Analysieren von elektrophysiologischen Daten. Der Fokus von Spyke Viewer liegt auf Nutzerfreundlichkeit und Flexibilität, was das Programm für Experimentatoren zum Visualisieren von Daten und für Theoretiker zum Entwickeln von Algorithmen nützlich macht.

Bei neueren elektrophysiologischen Techniken, die gleichzeitig die Aktivität von vielen Neuronen aufnehmen, überlappen aufgenommene Spikes häufig zeitlich. Die resultierenden Wellenformen sind eine besondere Herausforderung für Spike Sorting Methoden. In dieser Arbeit wird ein Spike Sorting Algorithmus verbessert und evaluiert, der das Problem von zeitlich überlappenden Wellenformen löst.

Alle Methoden wurden auf einem empirischen Datensatz entwickelt und getestet, der im präfrontalen Cortex von Makaken aufgenommen wurde, während die Affen eine visuelle Gedächtnisaufgabe lösen mussten. Mit Spyke Viewer und dem verbesserten Spike Sorting Algorithmus wurden großflächige Analysen zum Code der visuellen Stimuli und Experimentalkonditionen durchgeführt. Sowohl Reaktionen von einzelnen Neuronen als auch Populationscodes wurden mit einer Vielfalt von Dekodierungsmethoden untersucht. Mit Hilfe von zeitaufgelösten Analysen wird gezeigt, dass sich die Codierung von allen Experimentalkonditionen schnell ändert, während die Codierung der Stimuli zwischen Versuchs- und Test-Stimuli weitgehend gleich bleibt.

Acknowledgements

First, I would like to thank my supervisor Klaus Obermayer. He offered me a great scientific environment and guidance without imposing constraints. His feedback was always effective and to the point. Next, Matthias Munk for the dataset used in this thesis and many stimulating discussions. His enthusiasm for our projects was always contagious and I immensely enjoyed our interdisciplinary collaboration.

I also want to express my gratitude to the other members of my thesis committee, Martin Nawrot and Manfred Opper, for their feedback during our meetings and for helping to keep my research on track. In addition, Sonja Grün, Thomas Wachtler, and Jeff Teeters for believing in me and my work and for inviting me to their labs, workshops and working groups.

A very special thanks to Philipp Meier for the numerous ways in which he contributed to the completion of this project. He helped me get my bearings when I started out and continued to be a source of inspiration, support and fun throughout the whole process. We had many engaging discussions and he was a great colleague.

Furthermore, I am grateful to Felix Franke for his guidance and our very productive work together. Christian Donner was the best student assistant anyone could wish for and contributed a lot to the completion of this work. Jan Gosmann did diligent and great work during his lab rotation on the project.

I am thankful to all my colleagues from the Bernstein Center for our many long talks, scientific and otherwise. In particular, I am indebted to Janina Hesse, Katharina Wilmes, and Achim Meyer.

And of course a big thanks to all members of the Neural Information Processing Group for creating an enjoyable and stimulating work environment. In particular, I had great discussions with Timm Lochmann, Robert Meyer, Maziar Hashemi-Nezhad, Konstantin Mergenthaler, Moritz Augustin, and Wendelin Böhmer.

I also want to express my gratitude to Christian Kellner, Andrey Sobolev, Adrian Stoewer, and Tiziano Zito from the G-Node team for technical support and our discussions on neuroinformatics. And without Vanessa Casagrande, Margret Franke, Camilla Bruns, and Robert Martin, all the administrative details would have been a pain instead of a breeze.

You have actually read all of this? Then I'll finish by thanking Louis and Julia for giving everything to this research, and Scheherazade for getting me wherever I wanted to go. Finally, for being there when I needed them: my parents, Katharina, Romy, Sinan, and Janina.

Contents

List of Acronyms

API Application Programming Interface

BIC Bayesian Information Criterion

BOLD Blood-Oxygen-Level Dependent

BOTM Bayes Optimal Template Matching

CPU Central Processing Unit

DFT Discrete Fourier Transform

EEG Electroencephalography

EM Expectation Maximization

EPSP Excitatory Postsynaptic Potential

fMRI Functional Magnetic Resonance Imaging

FFT Fast Fourier Transform

GMM Gaussian Mixture Model

GUI Graphical User Interface

HDF5 Hierarchical Data Format 5

ICA Independent Component Analysis

IO Input Output

ISI Interspike Interval

JSON JavaScript Object Notation

LDA Linear Discriminant Analysis

LFP Local Field Potential

MAD Median Absolute Deviation

MEA Multielectrode Array

MRI Magnetic Resonance Imaging

MTEO Multiresolution Teager Energy Operator

PC Principal Component

PCA Principal Component Analysis

PFC Prefrontal Cortex

PSTH Peristimulus Time Histogram

PyPI Python Package Index

RBF Radial Basis Function

SEM Standard Error of the Mean

SIC Subtractive Interference Cancellation

SNR Signal to Noise Ratio

SVM Support Vector Machine

TEO Teager Energy Operator

List of Symbols

$\mathbf{X}$	Voltage data. A row per recording channel, a column per sample.
σ_X	Standard deviation of X.
x_t	Data point at sample t.
T	Length of data.
t	A time point in the data.
X_k	Fourier coefficient k.
N_C	Number of recording channels.
θ	A threshold value.
$\boldsymbol{\xi}^i$	Spike waveform template for neuron i.
T_f	Template length in samples.
$\mathbf{x}(t)$	A period of data with length T_f, starting at time t.
τ	Temporal offset in samples.
$\mathbf{C}$	Noise covariance matrix.
s^i	Spike train of neuron i.
v^i	Signal produced by neuron i.
η	Normally distributed noise.
f^i	Filter for neuron i.
r	Pearson product-moment correlation coefficient.
$\mathrm{K}(\mathbf{x}, \mathbf{y})$	A kernel function giving the similarity of feature vectors $\mathbf{x}$ and $\mathbf{y}$.

Introduction 1

Every interaction with the world around us requires a variety of tasks being performed by our nervous system. Information that is gathered by the sense organs needs to be encoded, processed, and potentially stored. Processing involves operations on previously stored information and new percepts have to be integrated with existing memories. Even for mildly complex tasks, memorizing and manipulating multiple pieces of information is essential. Finally, actions need to be planned and prepared before being executed by sending the appropriate signals to our muscles.

How all of these tasks are accomplished in the brain is one of the central questions in neuroscience. Despite decades of progress, numerous aspects of the neural correlates of behavior remain unsolved. Many techniques have been developed to measure neural activity while animals are performing tasks. One of the most important of these recording techniques is electrophysiology, because it allows researchers to measure the activity of single cells in the brain of behaving animals.

This thesis describes contributions to the state of the art in the analysis of electrophysiology data, including data management, signal processing and statistical analysis. The developed and improved methods are used on experimental data to yield insights into the coding of visual working memory in *Prefrontal Cortex* (PFC).

1.1 Recording brain activity

The generally accepted view is that neurons form the basic elements of information processing in the cortex. Neurons in the nervous system form a densely connected network: each neuron receives input via its dendrites and sends output along its axon. Axons and dendrites from different neurons are connected by synapses. To communicate, neurons use

action potentials (also called spikes): instantaneous changes in membrane potential that travel along the axon or dendrite and are transmitted at synapses. A neuron is said to fire when it generates a spike. A temporal sequence of action potentials, usually emitted by the same neuron, is called a spike train.

A variety of techniques are used in neuroscience to measure neural activity or a correlate thereof. All have distinct advantages, work at different spatial and temporal scales, and are continually improved (Marblestone et al., 2013). *Functional Magnetic Resonance Imaging* (fMRI) can record activity in the whole brain at a millimeter resolution and a sampling interval on the order of seconds. It relies on the *Blood-Oxygen-Level Dependent* (BOLD) contrast, so the measured signal is not the electrical activity of neurons but the oxygenation level of the surrounding blood following the activity. *Electroencephalography* (EEG) uses electrodes to record electrical activity on the scalp. Like fMRI, it is non-invasive and routinely used with human subjects. However, its spatial resolution is on the order of centimeters, while its temporal resolution reaches up to a few milliseconds.

Both fMRI and EEG are not suitable to investigate the activity of single neurons. For this purpose, invasive methods are needed. The most common are optical imaging and electrical recording. Optical imaging is a relatively recent technique (Kerr and Denk, 2008) that enables the experimenter to record the activity from hundreds or thousands of neurons on a timescale of around 100 milliseconds. Finally, electrodes allow direct recordings of the electrical field potential in the brain. When using intracellular recording techniques, the membrane potential and spikes from a single neuron can be recorded with very high accuracy. When using electrodes extracellularly, dozens or hundreds of neurons can be recorded in parallel. Both variants allow for sub-millisecond temporal precision. The methods and results described in this thesis deal with extracellularly recorded data.

1.2 Extracellular recordings

For extracellular recordings, small electrodes (typically 25-100 μm in diameter) are inserted into the brain and between neurons to record. Figure 1.1 shows a sketch of an electrode recording the activity of two nearby cells: spikes are typically initiated at the soma of a cell and then travel along its axon. This causes changes in the electrical field near the cell, which are recorded by the electrode. The signal from the electrode is amplified and then digitized before being stored and analyzed by a computer.

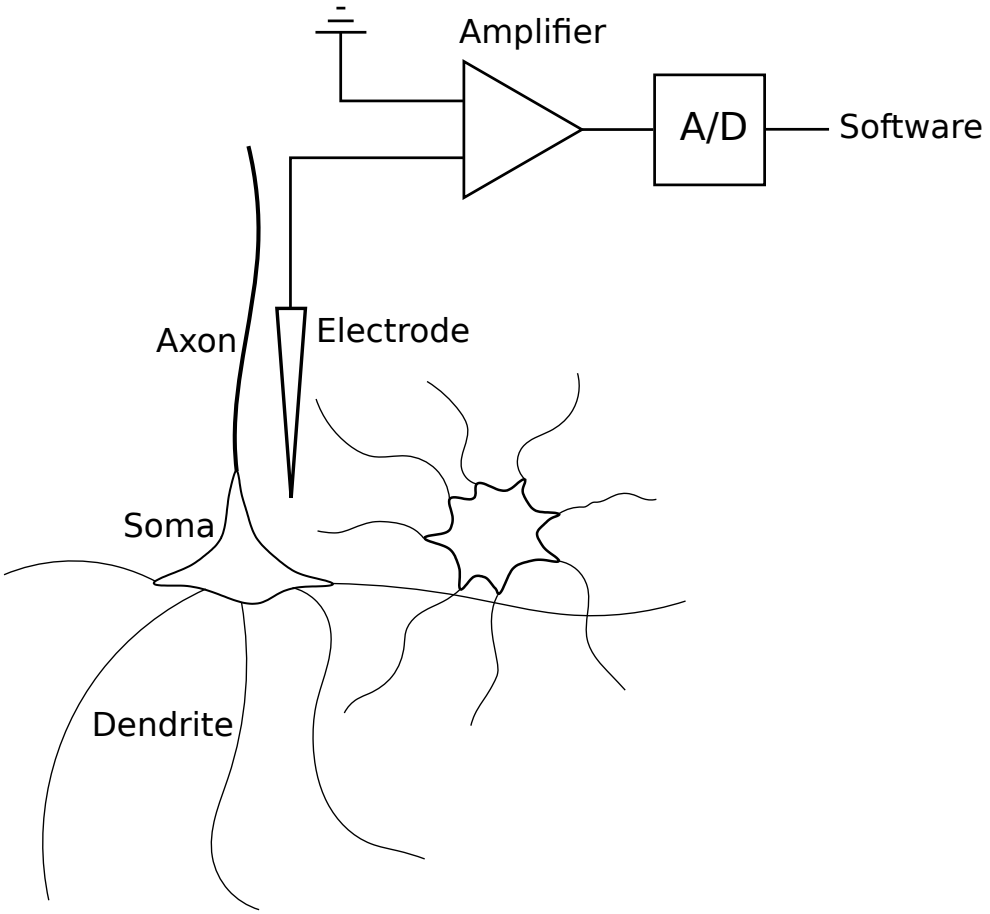

FIGURE 1.1: Simplified sketch of an extracellular recording setup.

The recorded signal is usually split into two components: *Local Field Potential* (LFP) and spike data. The LFP is generated by the combined electrical activity from nearby neurons, including synaptic activity, action potentials and graded membrane potentials (Buzsáki et al., 2012). The LFP is extracted from the recorded signal using a low-pass filter, usually with a cutoff at 300 Hz.

The spike data contains the frequency components above 300 Hz. This signal component reflects the action potentials of very close neurons and can be used to detect and identify single spikes. See Figure 1.2 for an illustration of the scales at which both signal components are generated.

Extracellular electrophysiological recordings have been one of the earliest techniques employed to investigate neural activity (Verkhratsky et al., 2006). They have been instrumental in many of the most important neuroscientific discoveries, from early breakthroughs like the receptive field and the properties of early visual information processing (Hubel and Wiesel, 1959, 1962) to more recent findings about the neural basis for

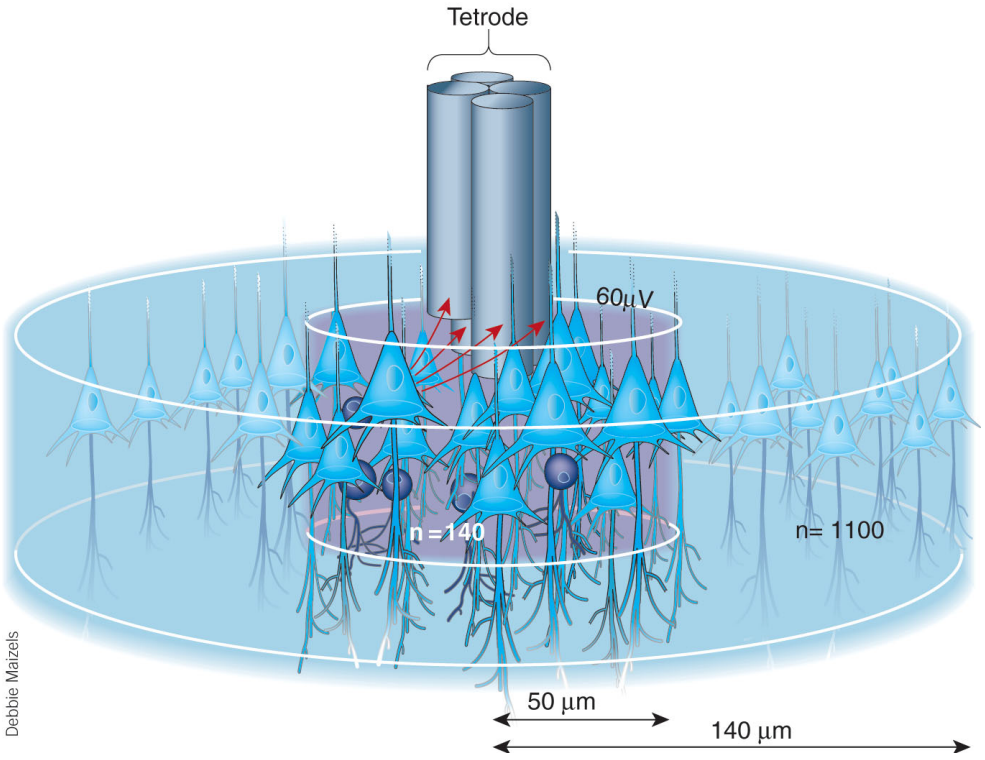

Figure 1.2: Sketch of a tetrode with four wires recording from surrounding neurons. Closer neurons contribute more strongly to the recorded signal and their action potentials can be detected and distinguished. The activity of more distant neurons is recorded as LFP and noise. Image from (Buzsáki, 2004), used with permission.

the sense of place in the form of place cells and grid cells (O'Keefe and Dostrovsky, 1971; Hafting et al., 2005).

While the technique has been in use for a long time, it continues to be improved. Early studies were only able to record the activity of a single neuron at a time. Today, even recording hundreds of units has become possible. In fact, the number of simultaneously recorded neurons has doubled every 7 years for the last 50 years (Stevenson and Kording, 2011). The associated exponential increase in recorded data enables and requires ever more sophisticated approaches to data management and analysis. The present work contributes to this development with the software and methods described in chapter 2 and chapter 3.

1.3 Spike sorting

Because the spiking activity of neurons is believed to be the main information transfer mechanism in the brain, extracting the time and source

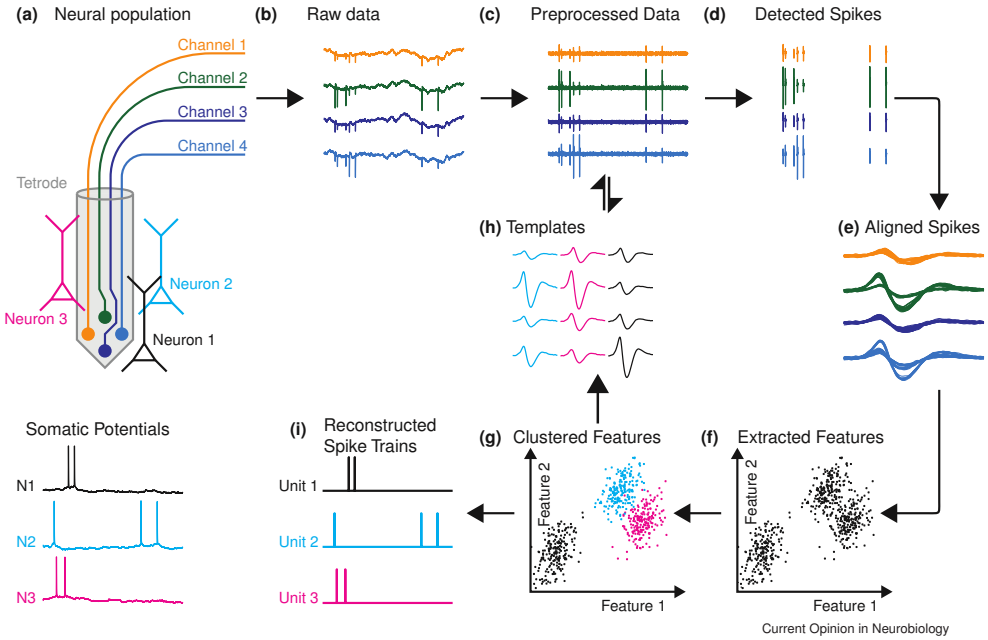

FIGURE 1.3: Illustration of the spike sorting process: **(a)** Electrodes records activity from three nearby neurons. **(b)** Unfiltered recorded data. **(c)** High-pass filtered spike data. **(d)** Parts of data where spikes were detected. **(e)** Small chunks of data cut out around spikes and aligned on amplitude minimum. **(f)** Features extracted from the spikes for clustering. **(g)** Output of a clustering algorithm: the detected spikes are labeled. **(h)** Each cluster has a prototypical waveform that can be extracted by averaging all waveforms assigned to the cluster. **(i)** Resulting spike trains for three single units. Image adapted from (Einevoll et al., 2012), used with permission.

of spikes in the recorded data is of particular importance. This process is called spike sorting and remains a technical challenge. Figure 1.3 illustrates the steps of a typical spike sorting pipeline from recording raw data to spike trains of three putative recorded neurons.

Because spike sorting is a statistical process and the resulting spike sources are not guaranteed to correspond to actual cells, they are referred to as units instead of neurons. There are two kinds of units: the results from a spike detection algorithm with only a single spike source for all spikes in a recording are called multi-units. The spikes from multi-units are assumed to be generated by multiple neurons. When a sorting step has been performed, its results are called single units. The spikes from each single unit are assumed to be generated by a unique neuron.

The *Signal to Noise Ratio* (SNR) of a recorded neuron depends on its distance to the electrode and is very low for most neurons (Buzsáki, 2004; Pedreira et al., 2012), which is a problem for both spike detection

and spike sorting. The SNR can be improved by using recording devices with multiple recording channels in close proximity (Gray et al., 1995). Such multielectrodes include stereotrodes (two channels) (McNaughton et al., 1983), tetrodes (four channels) (O'Keefe and Recce, 1993; Eckhorn and Thomas, 1993) and polytrodes (Blanche et al., 2005).

Multielectrodes record a compound signal from all active cells in their vicinity. Each cell has a characteristic amplitude profile across the recording channels because of the different relative location of each channel in respect to the cell. Each action potential of each cell is therefore represented by a multichannel spike waveform. Compared to spike waveforms recorded from single electrodes, multielectrode recordings reduce both false positives and false negatives when the spike sorting method takes the additional information into account (Gray et al., 1995; Harris et al., 2000).

Another challenge for spike sorting algorithms are temporally overlapping spikes: when action potentials of two neurons occur in quick succession and are recorded on the same channels, the resulting waveform is distorted and difficult to classify. This problem in particular will be addressed in chapter 3.

1.4 Working memory and the prefrontal cortex

While the ability to maintain relevant information in memory over short time spans (known as working memory) is an essential requirement for intelligent behavior, many questions remain about its neural basis. The brain needs to encode relevant information, maintain it and make it available for further processing. The PFC has long been thought to play an important role in working memory, as evidenced by early lesion (Jacobsen and Nissen, 1937; Mishkin, 1957) and electrophysiological studies (Fuster and Alexander, 1971; Kubota and Niki, 1971). The recordings often show persistent and task-dependent activity in prefrontal neurons during working memory tasks (Miller et al., 1996).

There are many proposed models of how working memory functions on an anatomical and cognitive level. Best known are the multicomponent model introduced by (Baddeley and Hitch, 1974) and the activated long-term memory model by (Cowan, 1988). In the multicomponent model, working memory is divided into separate components for phonological, visual, and spatial information. The model was later extended by another component for episodic information (Baddeley, 2000). Physiologically, evidence from electrophysiology points to the existence of such subsystems in the PFC (Wilson et al., 1993).

A more recent perspective holds that the actual maintenance and storage of information takes place in other areas (such as inferior temporal cortex) and the PFC is responsible for directing attention toward the appropriate areas, with working memory as an emergent property of this process (Postle, 2006). In such a model, each neuron takes part in many different tasks and specialization of areas is statistical rather than absolute (Duncan, 2001). This physiological view is compatible with Cowan's cognitive model of working memory as activated long-term memory. Again, there is electrophysiological evidence that the PFC functions more as a controller for attention than as a direct store of information in working memory (Lebedev et al., 2004).

In psychology, the importance of holding multiple items in working memory simultaneously has been recognized early. In his famous paper, (Miller, 1956) argued that there is a limit of around seven items that humans can keep in working memory. Later research points to a lower limit around four items (Cowan, 2001). The way in which working memory is searched also received early attention in psychology: the classic Sternberg task (Sternberg, 1966) measured the time in which a subject responded whether a test stimulus was part of a set of previously memorized sample stimuli. Sternberg found that the answer delay increased with the number of sample stimuli, which indicates that memory items are processed serially.

In contrast, the physiological basis of keeping multiple items in working memory simultaneously has received little attention. (Warden and Miller, 2007) trained two macaque monkeys to remember two visual items and respond after a delay whether a test stimulus was identical to one of the memorized items. They recorded neural activity in the PFC and found that a large fraction of neurons was sensitive to the identity of both objects. The patterns coding for a combination of two objects could not be explained by a linear combination of the patterns coding for each object individually.

A similar experiment was analyzed in (Warden and Miller, 2010): Again, the monkeys had to memorize two objects, but they had to either recognize them (releasing a bar when the sample sequence appeared again) or recall them (using saccadic eye movements in an array of presented stimuli). Most neurons encoded stimuli in both tasks, but generally not with the same activity patterns. This suggests that the neurons in PFC do not simply buffer sensory information but also reflect task demands.

1.5 Working memory experiment

To gain more insight into the neural code of working memory and in particular the coding of multiple working memory items, Matthias Munk

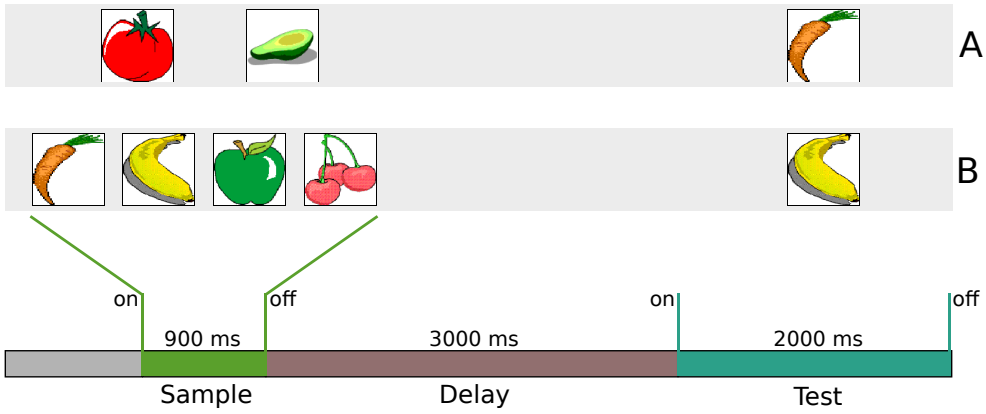

Figure 1.4: Timeline of a trial in the experiment. **(A)** Two sample stimuli are presented and and the test stimulus does not match any of the sample stimuli. **(B)** Four sample stimuli are presented and the test stimulus does match one of the sample stimuli.

and colleagues carried out an experiment: Two monkeys (Macaca mulatta) were trained to perform a visual working memory task (Figure 1.4). Each trial consisted of three periods: 900 ms sample stimuli presentation, 3 s delay, and 2 s test stimulus presentation. During the sample period, one to four different visual stimuli were successively presented.

The total sample presentation time remained the same regardless of the number of stimuli, so a single sample stimulus was shown for 900 ms, while each of a set of four sample stimuli remained on screen for 225 ms. The stimuli were randomly drawn from a set of 20 pictures of fruits and vegetables. The stimulus pictures were familiar to the monkey, although slightly modified pictures were presented in some recording sessions. In the test period, another stimulus drawn from the same set of pictures was presented. In 50% of all trials, the test stimulus matched one of the sample stimuli. Matching and non-matching trials were randomly distributed throughout each experiment. The monkey had to respond to matching trials by pressing the left of two buttons, while pressing the right button for non-matching trials. Correct answers were rewarded with juice.

Recording chambers were implanted in the ventral PFC guided by anatomical *Magnetic Resonance Imaging* (MRI) scans (T1-flash, 1 mm^3 isovoxel, 1.5 T). All procedures were approved by the local authorities (Regierungspräsidium) and are in full compliance with the guidelines of the European Community (EUVD 86/609/EEC) for the care and use of laboratory animals.

During the experiment, neural activity was recorded using up to 16 platinum-tungsten-in-quartz tetrodes (Thomas Recording, Giessen, Germany). Figure 1.5 (A) shows a photo of such a tetrode. The tetrodes

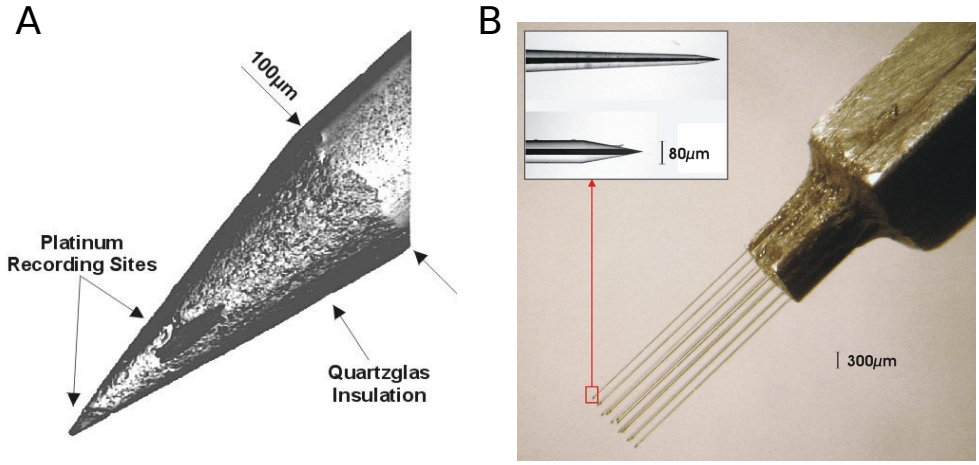

FIGURE 1.5: Photos of recording equipment. **(A)** The tip of one of the tetrodes used in the experiment. **(B)** A four by four electrode array. Images from Thomas RECORDING GmbH data sheet and newsletter.

were arranged in a square 4x4 grid with 1 mm distance between adjacent tetrodes. An example for such an electrode array is depicted in Figure 1.5 (B). Signals were digitized at 32 kHz and bandpass filtered 5-150 Hz for LFP, and 0.5-10 kHz for the spike data.

In total, 17 experimental sessions from the experiment are used in this thesis: 7 from monkey J and 10 from monkey L. Each session consists of 500 to 1500 trials. Over all recording sessions, 127 recording sites (tetrodes) provided data suitable for further processing.

1.6 Outline

The rest of this thesis is organized as follows: Chapter 2 describes novel tools for electrophysiology data management and analysis development. The core of the chapter is Spyke Viewer, a graphical interface to navigate and visualize large datasets in different formats that can be easily extended with custom analysis code. In chapter 3, all steps from recorded data to individual spike trains are described in detail. A focus of the chapter is the *Bayes Optimal Template Matching* (BOTM) algorithm, its improvements and extensive evaluation in particular with respect to overlapping spikes. Chapter 4 describes the analyses performed on the working memory experiment data and reports the findings. Finally, chapter 5 summarizes the results and provides a perspective on future lines of inquiry.

Data management and analysis tools for electrophysiology 2

This chapter introduces Spyke Viewer, an open source tool created to aid with the research presented in this thesis. What started as a small utility to help navigate the data from the working memory experiment has evolved into a mature and flexible application that can be used for various purposes in electrophysiology research. The program is being used by researchers in the Neural Information Processing Group and in other groups and is freely available.

Most parts of this chapter have been published in (Pröpper and Obermayer, 2013) and are reproduced here with minor modifications:

> Pröpper, R., Obermayer, K. (2013). Spyke Viewer: a flexible and extensible platform for electrophysiological data analysis. *Frontiers in Neuroinformatics*, 7(November):1-10.

In addition, newer developments and non-public extensions to the software specific to the data analysis pipeline for this thesis are described.

2.1 Introduction

The amount of data produced by electrophysiological experiments and neural simulations continues to grow. Accordingly, the complexity of analysis algorithms and infrastructure is increasing. Collaborations between research groups with different expertise are becoming the norm and the same data set is often analyzed by many scientists using diverse and dynamically evolving analysis methods.

This situation presents various challenges. Data sharing is difficult because electrophysiology data is very heterogeneous and no generally accepted standard formats exist yet (Teeters et al., 2013). Over the years, recording equipment vendors and simulation software authors have defined specific data formats and many research groups have created their own proprietary formats. Analysis code is often written for a particular dataset and then discarded or rewritten for other data. As a consequence, algorithms developed by theoreticians are difficult to use by experimenters and cannot easily be tested with newly gathered data.

2.1.1 Existing software

Many commercial and open source software packages have been developed to simplify such collaborative efforts and enhance code reuse. Chronux[1], FIND (Meier et al., 2008), and the Spike Train Analysis Toolkit[2] are algorithm toolboxes for MATLAB. However, while these toolboxes are open source, MATLAB itself is closed source and expensive. GNU Octave can be used as a free alternative, but it is not completely compatible with MATLAB and does not include all of its features. NeuroTools[3] is an algorithm toolbox for Python. All of these libraries restrict users to one or a few supported file formats; data from other sources has to be converted before it can be used. FIND uses Neuroshare[4] to support hardware vendor specific formats. Neuroshare is restricted to formats explicitly supported by the respective vendors and many formats only work on the Windows operating system. None of these libraries include a way of navigating and selecting data or using the analysis functions without programming.

Programs with a *Graphical User Interface* (GUI) provide such capabilities. There are various GUI programs available from hardware manufacturers and commercial vendors (e.g. Plexon[5], Tucker-Davis Technologies[6], or Nex Technologies[7]). However, the commercial programs are closed source, limited to the respective file formats of their creators, and only work on Windows. In addition, they cannot be extended by users. sigTOOL (Lidierth, 2009) is a MATLAB pogram with a graphical interface for data navigation that allows user extension. It supports several different data formats, but many of them only work in a 32-bit Windows environment and all data needs to be converted into a MATLAB format before it can be used by sigTOOL.

[1] `http://www.chronux.org/`
[2] `http://neuroanalysis.org/`
[3] `http://neuralensemble.org/NeuroTools/`
[4] `http://neuroanalysis.org/`
[5] `http://www.plexon.com`
[6] `http://www.tdt.com`
[7] `http://www.neuroexplorer.com`

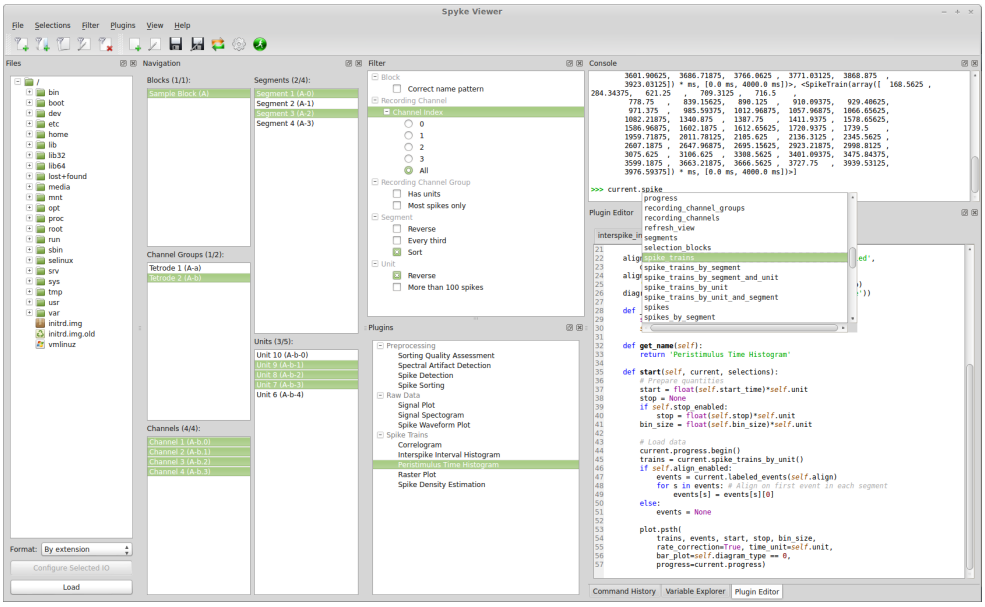

FIGURE 2.1: Screenshot of the Spyke Viewer GUI. The components of the program are arranged in dock elements. Most of the available docks are visible, the command history and variable explorer are tabbed together with the plugin editor. The dock layout is freely configurable and docks can be deactivated or detached to separate windows.

Noncommercial GUI programs that do not depend on MATLAB are rare. The program suite consisting of Dimstim, Spyke, and Neuropy (Spacek et al., 2008) supports data navigation and enables interactive work with a Python console, but does not support user extensions. The programs are very restricted in file format support and were written for the specific requirements of their authors. OpenElectrophy (Garcia and Fourcaud-Trocmé, 2009) includes a graphical interface and a library that users can use as a basis for their own analysis code. It has a strong focus on data management using a database: Many common file formats can be used but all data is loaded into a database before it can be accessed by OpenElectrophy. The GUI can only access tools provided by the developers (data visualization, oscillation detection, and spike sorting) – there is no interactive console and user extensions are limited to using the library for data access.

2.1.2 Spyke Viewer and Neo

Spyke Viewer is an open source GUI application to browse, visualize and analyze data from electrophysiology experiments or neural simulations. It is entirely written in Python and runs on Linux, OS X, and Windows with 32-bit and 64-bit architectures. Figure 2.1 shows a screenshot of the

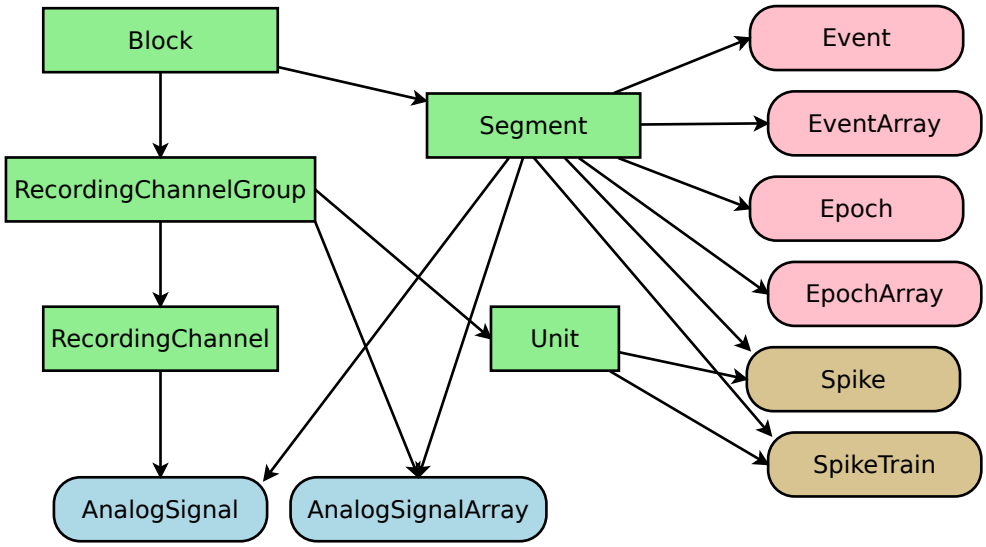

Figure 2.2: Objects defined by Neo. Rectangular items are container objects, data objects have rounded corners. Arrows indicate a "has zero or more" relationship. Colors of data objects denote to what containers they belong.

program. The GUI is organized in dock windows that can be configured into any desired layout.

Spyke Viewer is built on top of the Neo library (Garcia et al., 2014), an object model for electrophysiology data implemented in Python. Neo is based on numpy[8], the standard package for scientific computing in Python. In addition to numpy, Neo uses the quantities package[9] to represent data together with its physical units in a concise and user-friendly manner. Figure 2.2 shows the objects that Neo defines and their relationships. There are two general types of objects in Neo: containers and data objects. Containers define groupings of other objects:

Block
: The topmost object in the Neo hierarchy. Groups all objects that belong to the same experiment, recording session or simulation.

RecordingChannel
: A single channel that records a continuous signal, e.g. an electrode.

RecordingChannelGroup
: A semantical group of recording channels. This object could represent a tetrode, a *Multielectrode Array* (MEA) or an eye tracking device (with a channel each for the X and Y coordinates).

[8] `http://www.numpy.org`
[9] `http://packages.python.org/quantities`

Segment
: Groups data objects that are recorded simultaneously. Typically used for experimental trials.

Unit
: A source for spike events. Typically a neuron or a spike sorted unit.

All data objects exist in pairs, with one object representing a single entity, and an associated array:

AnalogSignal/AnalogSignalArray
: Continuous, analog signals with a fixed sampling interval. These signals always have a physical unit (e.g. Volt).

Event/EventArray
: Named points in time. A typical example is a button being pressed by an experimental animal.

Epoch/EpochArray
: Named periods of time, e.g. the stimulus presentation period.

Spike/SpikeTrain
: Neural action potentials. Includes at least the time when the spike occurred, but can also include information on the waveform.

Neo supports many file formats through custom Python classes, called Neo *Input Outputs* (IOs). Currently, there are IOs for Neuroshare, Matlab, a custom *Hierarchical Data Format 5* (HDF5)-based format and more than 15 others, many of them vendor specific[10]. I have co-authored the current implementation of the Neo HDF5 format and introduced new features to the library: when dealing with large files, it is often not desirable to load the entire data into memory at once. Loading only relevant parts of a file is referred to as lazy loading and Neo included a basic implementation. I extended the functionality to work transparently for users: in compatible IOs, activating lazy loading automatically only loads the parts of the file that are actually needed for the requested data.

With Spyke Viewer, users can load any data format supported by Neo and browse and select the contents. They can use an integrated Python console to interactively work with subsets of the loaded data, using the whole Python scientific stack and custom libraries. Python is also well suited to interface to other languages such as C/C++, Matlab, or R using appropriate modules.

Spyke Viewer is designed to be flexible and extensible. When working with large datasets, users often only care about a particular subset. A

[10]Full list at `http://neo.readthedocs.org/en/latest/io.html`

filtering system allows users to hide Neo objects they are not currently interested in. Filters are user defined Python functions and are manipulated directly from the GUI.

In addition to the console, analysis plugins provide the visualization and analysis capabilities of Spyke Viewer. Analysis plugins are Python classes and can be created and edited in the GUI or using an external editor. They perform operations on the selected data using arbitrary Python code (so they can use Python libraries or even other languages) and can range in complexity from showing the number of currently selected spikes to complex multi-stage analyses, possibly with their own GUI.

Many users want to analyze data stored in a nonstandard format. Using IO plugins, they can use Spyke Viewer with their existing data and do not need to convert it. If they want to share data with users of other tools, Spyke Viewer provides data export to multiple standard formats using Neo.

Because all data in Spyke Viewer is internally represented as Neo objects, analysis plugins work with data from any source without code changes. This standardization enables convenient sharing of algorithms and easier replication of results. It also facilitates testing algorithms on different data sets. On the Spyke Repository website, users can find existing plugins and share their own contributions.

Spyke Viewer and Neo are focused on LFP and spike data. Although data from other sources such as EEG can be processed, they are not explicitly supported: Neo does not include IO classes for many EEG data formats and the plugins included with Spyke Viewer do not offer specialized EEG visualizations.

2.2 Software architecture

Spyke Viewer consists of two Python packages: *spykeutils* and *spykeviewer*. *spykeutils* contains components related to data management, the core of the plugin system, analysis algorithms and plotting functions. It is independent of *spykeviewer*, has fewer dependencies and no graphical interface, so it can be installed on servers without a graphical environment (though interactive plots will not work in this case). Using *spykeutils*, plugins can be run on a compute server. The *spykeviewer* package contains the GUI program, including the navigation and filtering systems and plugin management. Here, the distinction is mostly ignored and both packages are referred to as Spyke Viewer.

The code of the most recent development version and all previous stable releases is available on GitHub[11]. The GitHub web page also

[11] `https://github.com/rproepp/spykeviewer`

includes an issue tracker for bug reports and feature requests. Using the Git version control system together with GitHub facilitates collaborative development and gives users of the program a convenient way to contribute code changes.

Software quality and documentation are an important aspect of software development that receive little attention in many scientific software packages. Spyke Viewer currently has almost 200 unit tests to ensure that new features do not break existing functionality. Continuous integration (Zaytsev and Morrison, 2012) delivers immediate feedback on code changes. In the same way, tze documentation is automatically updated and available online for all releases and the current development version[12].

2.2.1 Data selection

The first step in working with any dataset of sufficient size is selecting the particular subset of the data one wants to investigate. In Spyke Viewer, a selection describes a connected subgraph of Neo container objects (see Figure 2.2). On the top level, it consists of a list of block objects with information on where they can be found (e.g. a path in the file system). Objects further down in the hierarchy are then referenced by their relations to higher objects (e.g. the third segment of a block). Therefore, a selection is self-contained: it can be saved to disk and includes all necessary information to load the container and data objects it references. Because selections do not contain the data itself, they use little memory. In the Spyke Viewer GUI, users can create selections spanning data from multiple sources in an intuitive way using the navigation dock. Multiple selections can be managed from the GUI and saved to or loaded from human readable *JavaScript Object Notation* (JSON) files.

Usually, only a semantically defined subset of the data is relevant at a given time, for example all Segments that were recorded while a particular stimulus was presented. As it can be cumbersome to manually select such subsets from large datasets, Spyke Viewer includes a filtering system that determines which objects are shown in the navigation dock. A filter is a Python function that operates on a list of container objects. It returns which items are to be displayed and in what order. Multiple filters can be linked into filter chains that are processed in order. For convenience, it is also possible to define filters that only act on single objects and return whether the object should be displayed. Python functions are used to give maximum flexibility to the user because datasets are heterogeneous and will often need custom filters. Because Python has a syntax close to natural language and most filters are simple, it is easy to create them.

[12] `http://spyke-viewer.readthedocs.org/`

For example, a filter operating on single objects to only show blocks that contain at least 10 segments:

```
def filter(block):
  return len(block.segments) >= 10
```

When this filter is active, Spyke Viewer calls this function for each loaded block. The function then returns whether the list of segments in the block contains at least 10 items. Blocks for which the function returns false are not shown in the navigation dock.

2.2.2 Data provider

Once users have selected data of interest, they need a way to access this data. In Spyke Viewer, this is realized by the `DataProvider` class. Each `DataProvider` object encapsulates a selection and provides a large number of methods to query the selected Neo objects. This additional layer of abstraction between users and data has several advantages:

- Convenience for common usage patterns: For example, getting all selected analog signals ordered by recording channel requires just one method call.
- Management of data acquisition: Spyke Viewer supports transparent lazy loading by initially only reading container objects when a file is opened and then automatically loading the missing data as needed during calls to `DataProvider` methods.
- Specialization: `DataProvider` itself is an abstract base class. Spyke Viewer currently contains two implementations: one uses selections in memory as described above (e.g. loaded from a selection file), the other always uses what is currently selected in the GUI. Other implementations are possible: For example, database access can be implemented more efficiently with a `DataProvider` subclass than is currently possible using a Neo IO.

2.2.3 Console

One way to interact with the `DataProvider` objects is through an interactive console. Spyke Viewer provides two options: An internal console and IPython. The internal console is implemented using components from *spyderlib*[13] and integrates completely into the GUI as a dock. Spyke Viewer also includes a command history and a graphical variable explorer for the internal console in separate docks.

[13] `https://code.google.com/p/spyderlib`

IPython is a popular shell in the scientific Python community (Perez and Granger, 2007). An IPython kernel is embedded into Spyke Viewer, allowing external IPython consoles to attach to the program. The IPython consoles can then be used in the same way as the internal console.

Two important objects are defined in the console environment: `current`, a `DataProvider` object representing the presently selected objects in the GUI, and `selections`, a list of `DataProvider` objects representing stored selections. These objects, together with common scientific Python packages such as *scipy* and *matplotlib*, enable efficient exploratory data analysis.

2.2.4 Plugins

Extensibility is a central design goal of Spyke Viewer. It is achieved with a flexible plugin architecture that allows easy creation of new plugins that can be started directly from the GUI. The core of the plugin system is implemented in the *spykeutils* package, so plugins can also be executed without the Spyke Viewer program.

An important decision for every plugin architecture is how plugins in the file system are discovered and integrated. In order to keep the plugins as simple as possible, Spyke Viewer scans a list of user defined directories for Python files. The directories are scanned recursively and subdirectories are represented in the GUI, helping to organize plugins. In each of the Python files, Spyke Viewer searches for plugin classes. A plugin class inherits from an abstract plugin base class and implements two methods: `get_name` returns the name of the plugin used in the GUI, `start` contains the code to run when the plugin is executed.

Spyke Viewer includes a Python editor that can be used to create and edit plugins directly from the GUI. Like the internal console, the editor uses components from *spyderlib* and supports syntax highlighting, autocompletion and Python docstrings as tooltips. However, it is also possible to edit plugins in an external editor. The plugins are refreshed at runtime, so it is not necessary to restart Spyke Viewer when they change. Figure 2.3 shows an example of a complete plugin for Spyke Viewer that uses *matplotlib* to plot spike rates over time, with one data point per segment. This plugin and its creation are described in more detail in subsection 2.3.4.

Many analysis algorithms or plots have various parameters. Users should not have to change the plugin code every time they want to change a parameter, so Spyke Viewer uses the *guidata* package[14], to make creating graphical options very simple. See Figure 2.3, lines 6 to 8, for an example of two such options. Because *guidata* requires graphical

[14] `https://code.google.com/p/guidata`

packages, it is wrapped in *spykeutils* so plugins behave as expected when *guidata* is not available.

Analyses can require long execution times and may produce large results which are reused, e.g. in later stages of a processing pipeline. For convenience, Spyke Viewer provides an interface to save results connected to a parameter set (by default the current selection and plugin parameters) to HDF5 files. The resulting files are indexed using this information so they can be retrieved quickly. Because plugins are regular Python code, they can also employ arbitrary other persistence schemes such as saving results to a database. In addition, the Spyke Viewer *Application Programming Interface* (API) includes functions to execute plugins. Together, these features enable the creation of simple workflows through plugins calling and retrieving data from other plugins or the creation of dispatcher plugins that can implement more complex workflow scenarios.

Spyke Viewer provides multiple ways in which plugins can be executed. The default method starts the plugin in the same process as the program itself. This has the advantage of shared memory: the plugin can access all data that has been loaded into memory during execution of Spyke Viewer. This is the fastest method, but it has a drawback: the GUI can not be used while the plugin is working, except for progress display.

Another method is to start the plugin in its own process. In this case, the selection, plugin code and parameters are serialized and sent to a new process. The new process then creates the necessary `DataProvider` objects and starts the plugin. Because the plugin execution is independent of Spyke Viewer, the required data needs to be loaded in the new process. However, the GUI is unaffected by the plugin execution and multiple plugins can be run at the same time.

Finally, the execution of plugins can be completely decoupled from Spyke Viewer. A small Python script loads a plugin, stored selections and additional context information and then executes the plugin. Spyke Viewer can be configured to save the necessary information instead of directly starting a plugin in a new process. This can be used for reproducibility, batch processing or execution on a server (see subsection 2.2.5).

Despite the large number of file formats that Spyke Viewer supports through Neo, there are many custom data formats that Spyke Viewer cannot load directly. In order to use Spyke Viewer with such data, users have two choices: they can either convert the data to a standard format that Neo can read, or they can write an IO plugin. These plugins are implemented as Neo IO classes that load the format into the Neo object hierarchy. They inherit from `neo.io.BaseIO`, the abstract base class also used for all regular Neo IO classes. IO plugins are discovered in the same way as regular plugins when placed in one of the plugin directories. In addition to data reading, IO plugins can also support data writing.

```
from spykeutils.plugin import analysis_plugin, gui_data
import matplotlib.pyplot as plt

class SamplePlugin(analysis_plugin.AnalysisPlugin):
  # GUI options
  y_axis = gui_data.ChoiceItem(
    'Rate type', ('Spikes per segment', 'Spikes per second'))
  show_legend = gui_data.BoolItem('Show legend')

  def get_name(self):  # Name displayed in the GUI
    return 'Spike rates over segments'

  def start(self, current, selections):
    # Get a dictionary of selected spike trains
    spike_trains = current.spike_trains_by_unit_and_segment()

    # Iterate over units and plot spike counts for each
    for unit, trains in spike_trains.iteritems():
        if self.y_axis == 0: # Count the spikes in each segment
          rate = [len(st) for st in trains.itervalues()]
        else:  # Divide by the length of the spike train in seconds
          rate = [len(st) / (st.t_stop - st.t_start).rescale('s')
                  for st in trains.itervalues()]

        plt.plot(rate, label=unit.name)

    if self.show_legend:
      plt.legend()
    plt.show()
```

FIGURE 2.3: A complete plugin that plots spike rates over multiple segments. It includes two options that can be adjusted in the Spyke Viewer GUI.

Spyke Viewer offers data export using all available Neo IO classes and IO plugins that support writing.

Figure 2.4 shows how the different components of the system work together. Data is loaded into the Neo object hierarchy using Neo or IO plugins. The active filters determine which objects are displayed in the Navigation dock in the GUI. Users then select a subset of these objects. The selection from the navigation dock is used to create a `DataProvider` object. In addition, multiple selections can be stored and are used to create further `DataProvider` objects. These objects can then be used to access the selected data in the integrated console, in IPython or in analysis plugins.

2.2.5 Remote and cloud plugins

When computationally expensive algorithms are implemented as plugins, it is often inconvenient to execute them on a local machine. For this use case, Spyke Viewer includes a system to send plugins to remote servers for execution. It is implemented by sending a set of plugin code, parameters and data selections (referred to as task) from a Spyke Viewer instance

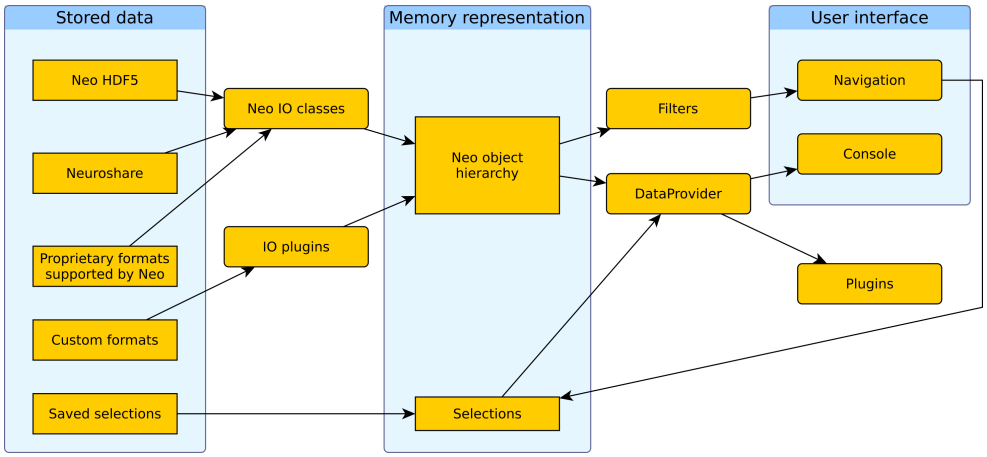

Figure 2.4: Illustration of the data flow in Spyke Viewer. Rectangles with sharp corners represent data on disk or in memory, rounded corners indicate Spyke Viewer components.

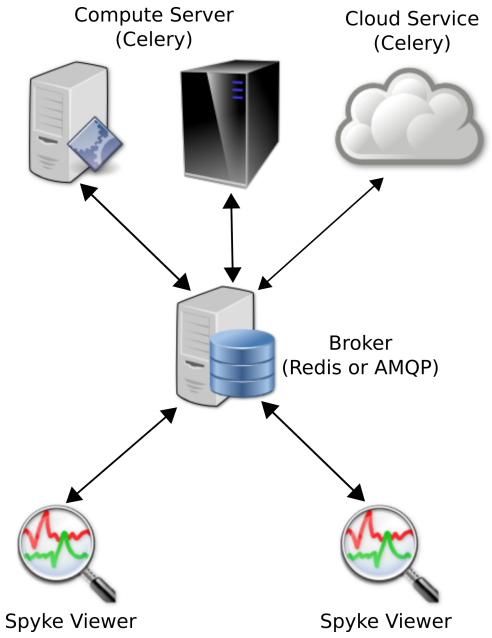

Figure 2.5: Schematic of a remote plugin execution scenario.

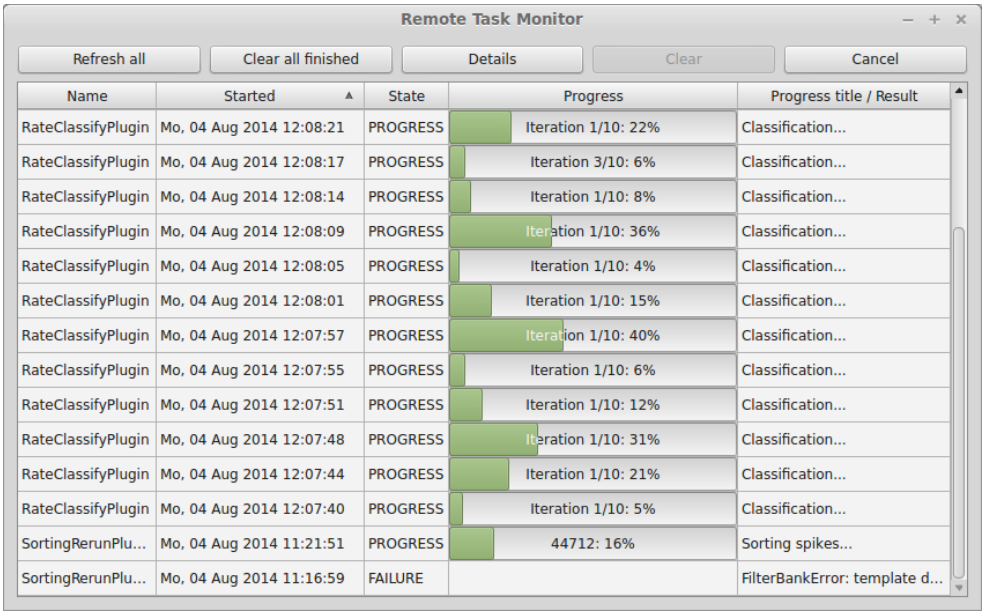

FIGURE 2.6: Remote tasks

to a machine running a task broker. The broker runs either Redis[15] or RabbitMQ[16]. It is responsible for storing tasks in a queue until they are executed and for relating current status, output, results and errors back to Spyke Viewer clients.

In addition to the broker, there are one or multiple worker instances responsible for executing the tasks. Each worker runs on a separate machine or cloud platform (e.g. Amazon EC2[17]) and can execute several tasks at once, which is useful if multiple *Central Processing Unit* (CPU) cores are available. Figure 2.5 shows how the various involved machines communicate: any number of Spyke Viewer clients can send their tasks to a single broker that distributes the tasks to an arbitrary number of workers for execution. New workers can be added and existing workers can be removed at any time without disrupting the system. Each worker runs a custom program based on the *Celery*[18] library that connects to the broker, receives task information and executes the contained plugin with the given parameters. Text output, exceptions and progress information are reported back to the broker and can be queried by clients.

Sending a plugin for remote execution is just a single click in Spyke Viewer. In addition, the application includes a manager for remote tasks (Figure 2.6). It shows the progress of running tasks and allows users to inspect parameters, results or errors of executed plugins and cancel

[15] `http://redis.io/`
[16] `http://www.rabbitmq.com/`
[17] `http://aws.amazon.com/ec2/`
[18] `http://www.celeryproject.org/`

queued or running tasks. In order for this setup to work, the data to be analyzed has to be available on the Spyke Viewer client in order create a selection and on all workers for the computation. This can be achieved by duplicating the data or storing it at a commonly accessible location.

2.3 Usage

2.3.1 Installation

Python is well suited for developing both scientific algorithms and large scale graphical applications such as Spyke Viewer thanks to the robust language design and availability of mature libraries for scientific computing, plotting, and GUI development. As an interpreted language, Python also lends itself to cross-platform development. By using PyQt as an interface to the Qt GUI library, the graphical interface runs and looks native on Linux, OS X, and Windows.

However, while developing a Python program that runs on different platforms works well, distributing the program can be difficult. The standard way to distribute a Python application is the *Python Package Index* (PyPI)[19]. However, PyPI requires an existing, properly configured Python installation. Because Spyke Viewer also targets users with little or no Python experience, further options apart from PyPI were needed.

For Debian-based Linux distributions (e.g. Ubuntu), the NeuroDebian project (Halchenko and Hanke, 2012) provides a large number of neuroscience research software packages. Spyke Viewer packages exist in NeuroDebian, so Debian users can install the program with their package manager using the NeuroDebian repository.

For Windows and OS X users, PyInstaller[20] is used to create binary packages. PyInstaller collects the Python interpreter, the Python files from Spyke Viewer, and all required libraries into one package that can be used without a regular Python installation. While these packages are easy to use, they have some drawbacks: they are larger than the source package provided by PyPI and users cannot simply install additional libraries as in a regular Python installation.

2.3.2 Filters

Once Spyke Viewer is installed, users can load supported files and explore their contents. For larger datasets, it is often useful to hide nonrelevant data using the filter system. The decision to implement filters as Python functions maximizes flexibility but could be less comfortable to use than a

[19] `http://pypi.python.org/`

[20] `http://www.pyinstaller.org`

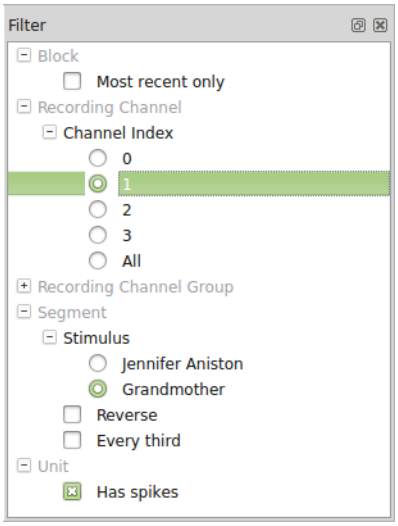

FIGURE 2.7: The filter dock allows users to order and activate filters.

more restricted solution. To alleviate this issue, Spyke Viewer integrates all aspects of filter management into the GUI. In the filter dock (see Figure 2.7), users can toggle filter activation, organize them into groups and order them using drag and drop.

Filters can be created or edited using a GUI editor. They are saved as Python files (with annotations in comments) so they can also be edited externally and shared with other users. Figure 2.8 shows such an annotated Python file for the segment filters in Figure 2.7. The first two filters check the annotations of a segment for a particular stimulus and are part of an exclusive filter group named "Stimulus". Exclusive groups can only include a single active filter. The other two filters are not mutually exclusive and operate on the complete list of segments instead of single items. They reverse the list or only admit every third item, respectively.

2.3.3 Included plugins

The Spyke Viewer package includes several plugins for common plots:

Correlogram
: Creates subplots with autocorrelograms and cross-correlograms for selected units or selections. Bin size and maximum offset can be specified. Figure 2.9 shows an example.

Interspike Interval Histogram
: Displays the interspike interval distribution for selected units or selections. Bin size and maximum offset can be specified. Creates bar or line histograms.

```
#GROUP Stimulus
#EXCLUSIVE

def filter(segment):
  """Jennifer Aniston"""
  return segment.annotations['stim'] == 'Jennifer'

def filter(segment): #ACTIVE
  """Grandmother"""
  return segment.annotations['stim'] == 'Grandmother'
#ENDGROUP

def filter(segments):
  """Reverse"""
  return segments[::-1]

def filter(segments):
  """Every third"""
  return segments[::3]
```

Figure 2.8: Code for segment filters in Figure 2.7. When filters are edited directly in the GUI, only the function body has to be written.

Peristimulus Time Histogram

Shows the *Peristimulus Time Histogram* (PSTH) for selected units. Spike trains can be aligned on an event before calculation. Bin size can be specified and users can choose between bar and line histograms.

Raster Plot

Displays the spike times for multiple units in one segment or multiple segments and one unit. Optionally overlays events or epochs.

Signal Plot

Shows analog signals for selected recording channels in subplots or one shared plot. Optionally overlays events, epochs, spike times or spike waveforms.

Signal Spectrogram

Creates subplots with spectrograms (time-frequency histograms) for selected recording channels. The number of samples used for the *Fast Fourier Transform* (FFT) can be specified as a trade-off between temporal and frequency resolution.

Spike Density Estimation

Shows a kernel density estimation of spike rates for selected units or selections. Spike trains can be aligned on an event before calculation. The kernel bandwidth can be specified. Alternatively, the plugin

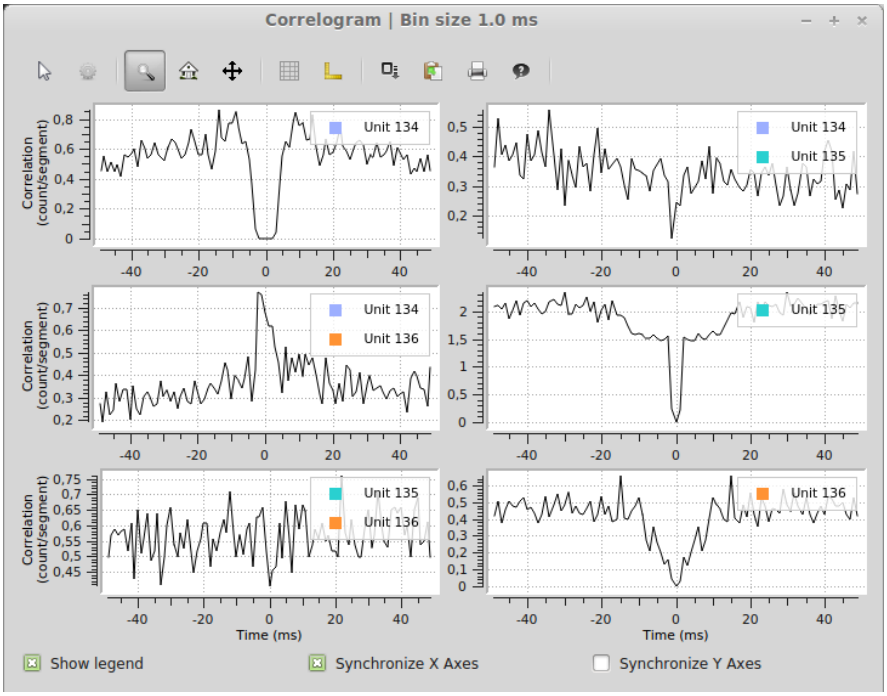

FIGURE 2.9: Autocorrelograms and cross-correlograms for three units. When plots include multiple subplots, the suplot axes can be synchronized independently.

can calculate the optimal bandwidth automatically (Shimazaki and Shinomoto, 2010). Shown in Figure 2.10.

Spike Waveform Plot

Displays spike waveforms for selected units. Offers various configurations to split recording channels or units into subplots.

All of the included plugins use the *guiqwt* plotting library[21]. Compared to *matplotlib*, which can be considered the standard library for plotting in Python, *guiqwt* offers better performance and more tools for interactive use. For publication-quality plots, *matplotlib* is preferable and available in Spyke Viewer from the console and in custom plugins.

The included plugins enable quick exploratory analyses of both analog signals and spike data for datasets of any size. For example, the PSTH and spike density estimation plugins can show average firing rates during experimental trials. Together with the filter system (e.g. the stimulus filters described above), these plugins could be used to compare the influence of various stimuli on the firing rate of neurons. Figure 2.10 shows an example of a spike density estimation plot comparing the influence of a stimulus on four neurons.

[21] `http://code.google.com/p/guiqwt`

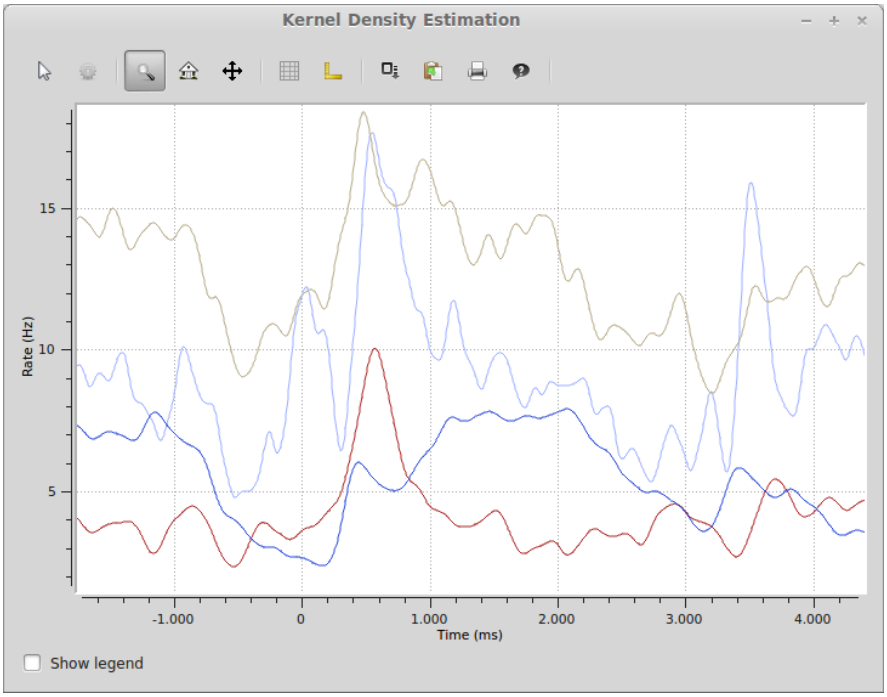

Figure 2.10: Spike density estimation of four units. Time 0 in the plot corresponds to an experimental stimulus event. The kernel bandwidths for the different neurons were optimized independently.

2.3.4 From console to plugin

After the initial exploration of the data, users usually want to perform custom analyses. A simple example illustrates how Spyke Viewer can be used to create a plot that is not available with the included plugins. Consider the case where users have analyzed the stimulus dependent firing rate changes and are now interested in how the activity of a neuron evolves over a longer time, spanning many segments of a recording. They want to visualize the spike count of a selected unit over segments. After selecting the unit and segments of interest, they use the internal console to query the currently selected spike trains:

```
>>> trains = current.spike_trains()
```

The `trains` variable could now be inspected using the variable explorer dock: it contains a list of Neo spike train objects. Now, the users can count and plot the number of spikes in each spike train. Because *matplotlib* is imported by default in the console under the name `plt`, this can be done in one line:

```
>>> plt.plot([len(st) for st in trains])
```

This code uses a Python list comprehension to concisely define the list of values to be plotted: The length of a spike train (i.e. the number

of spikes) for each spike train in the variable `trains`. After examining a few neurons in this way, the users might decide that this is a useful plot that they want to keep with convenient access. They might also want to add some additional features, such as multiple neurons in the same plot for easy comparison. These requirements are best satisfied by writing a plugin.

When writing a new plugin in the GUI, Spyke Viewer creates a template with code for a minimal functional plugin. The users can import *matplotlib* and copy the two lines they used to create the plot from the console history into the `start` method of the template plugin. Figure 2.3 shows the final plugin after adding the ability to plot multiple units. This plugin also has two graphical options that control if the number of spikes is shown as a count per segment or a rate in Hertz and if a legend is included in the plot. When the plugin is saved, it immediately appears in the Spyke Viewer plugin list and is ready for use.

2.3.5 Startup script

The startup script offers a way for users to customize Spyke Viewer. It is executed whenever Spyke Viewer is started, and before plugins or data are loaded. It can be used to set configuration options defined in the Spyke Viewer API, change global parameters like the colors used in plots or even manipulate the GUI itself. The following code would disable automatic loading of previously opened data files when starting the application and increase the font size in the console:

```
from spykeviewer import api

# Set configuration option
cfg = api.config
cfg.load_selection_on_start = False

# Change existing console font
f = api.window.console.font()
f.setPointSize(16)
con = api.window.console
con.set_pythonshell_font(f)
```

2.4 Extension repository

The Matlab File Exchange[22] is a popular resource for MATLAB users to find and share code. Currently, there is no equivalent for Python users. Although PyPI works well for sharing libraries, it requires users

[22] `http://www.mathworks.com/matlabcentral/fileexchange/`

to create setup scripts and the libraries are installed in a fixed directory, so it is not suited for plugins or other Spyke Viewer extensions. The Spyke Repository was created to foster sharing and grow a community around Spyke Viewer: a website where users can upload their Spyke Viewer extensions and search, download or comment existing extensions from others. Figure 2.11 shows a screenshot of the extension list page. The site is hosted at the G-Node (Herz et al., 2008) and can be reached at `http://spyke-viewer.g-node.org`.

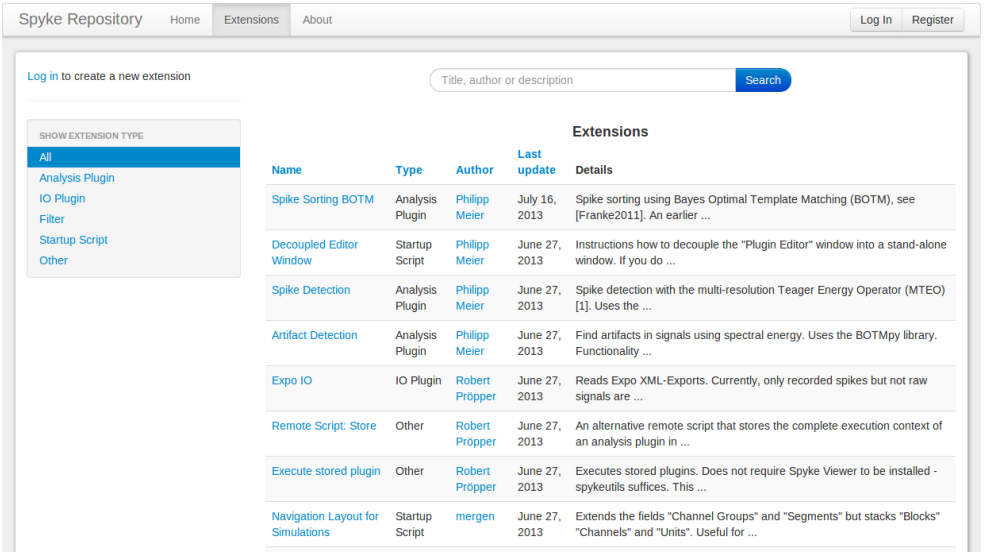

Figure 2.11: Extension list on the Spyke Repository website.

Some of the current content on the site are scripts used to execute plugins on servers. Other examples include plugins for artifact detection, spike detection, automatic spike sorting (Franke et al., 2010), and an IO plugin for Expo[23] files.

2.5 Database

The data from the experiment described in section 1.5 is hosted in a database at the G-Node. Figure 2.12 shows the schema of the database. It stores all recorded data: the high-pass filtered spike data, LFP, eye tracker, precise times for all events during each trial (e.g. stimulus onset or button press) and relevant metadata such as the length of a trial or what stimuli were shown. In addition, all results from the signal processing pipeline (described in chapter 3) are stored the database. Such an analysis has a type, for example artifact detection or spike sorting, and various attributes and parameters in addition to its primary

[23] `https://sites.google.com/a/nyu.edu/expo`

results: artifact periods or spike times and identities. An analysis can also reference other analyses. For example, a spike sorting can refer to its underlying spike detection. Using all of this information, the database provides full providence for every spike. A general note system allows comments with different scopes to be stored: for example, a remark on a particular tetrode or a comment about the behavior of a single unit during a specific trial are possible. The database layout is designed to be flexible and has also been used to store data from other electrophysiology experiments.

In order to work with the database effectively, I created a custom version of Spyke Viewer that includes extensions to the public version. In the GUI, there are additional docks for navigation, analysis information and comments (depicted in Figure 2.13). The database navigation dock looks and works similar to the regular Neo navigation. The most important difference is the analysis selection list: the Neo object hierarchy does not include a corresponding concept. This dock also includes a quick way to find an analysis, a unit or a trial by database id. The analysis info dock presents information about the currently selected analysis, such as the time and the trial range on which the analysis was executed and all of its parameters.

Another important extension is not visible to the user. Data is accessed using a custom `DataProvider` subclass. It would also have been possible to write an IO plugin for the database. While that approach would have been easier, creating a custom `DataProvider` subclass allows for various performance and memory optimization because it has access to more detailed knowledge of what data is needed at any given time. These optimizations make big difference because the total amount of data is much larger than the available main memory and the database is not hosted on the same computer as Spyke Viewer runs, but has to be accessed over the internet. Thanks to the Spyke Viewer architecture, the plugins written for the database can also work with other data loaded using Neo.

2.6 Summary

This chapter described Spyke Viewer, a flexible and extensible platform for analyzing data from electrophysiology experiments and computer simulations. It is based around the common object model provided by the Neo library. A central design goal was to restrict the user as little as possible while providing convenient functionality for selecting, accessing, visualizing, and analyzing data. The most important feature in this respect is the plugin system that facilitates exploratory data analysis and algorithm development, and includes features for reproducibility

and running time consuming analyses. No previous work has offered the combination of open source, convenience, and flexibility that Spyke Viewer provides its users.

Thanks to its GUI, Spyke Viewer provides a wide range of functionality for users who do not develop algorithms themselves. It includes several plugins for common visualizations used in neuroscience. Plugins developed by others can be applied to different datasets without writing code, considerably reducing the effort of sharing data and code between scientists and laboratories. The Spyke Repository website lists extensions created by users and allows developers to share their extensions with the community. Python has a large community of scientific users, including many neuroscientists. These researchers have produced a wide variety of useful Python packages for data analysis and visualization. Thanks to the flexibility Spyke Viewer offers to its users, these packages can be used from the internal console or turned into analysis plugins. By leveraging existing libraries and tools, small plugins can perform complex analysis steps such as spike sorting. Since Python also works well as a glue language, Spyke Viewer has the potential to unify existing tools by providing a central GUI from which independently developed analyses can be launched, regardless of whether they exist as Python libraries, in other programming languages or as standalone programs.

Spyke Viewer and its database extensions have been essential tools for the work presented in this thesis as well. All results described in chapter 3 and chapter 4 have been obtained using the software.

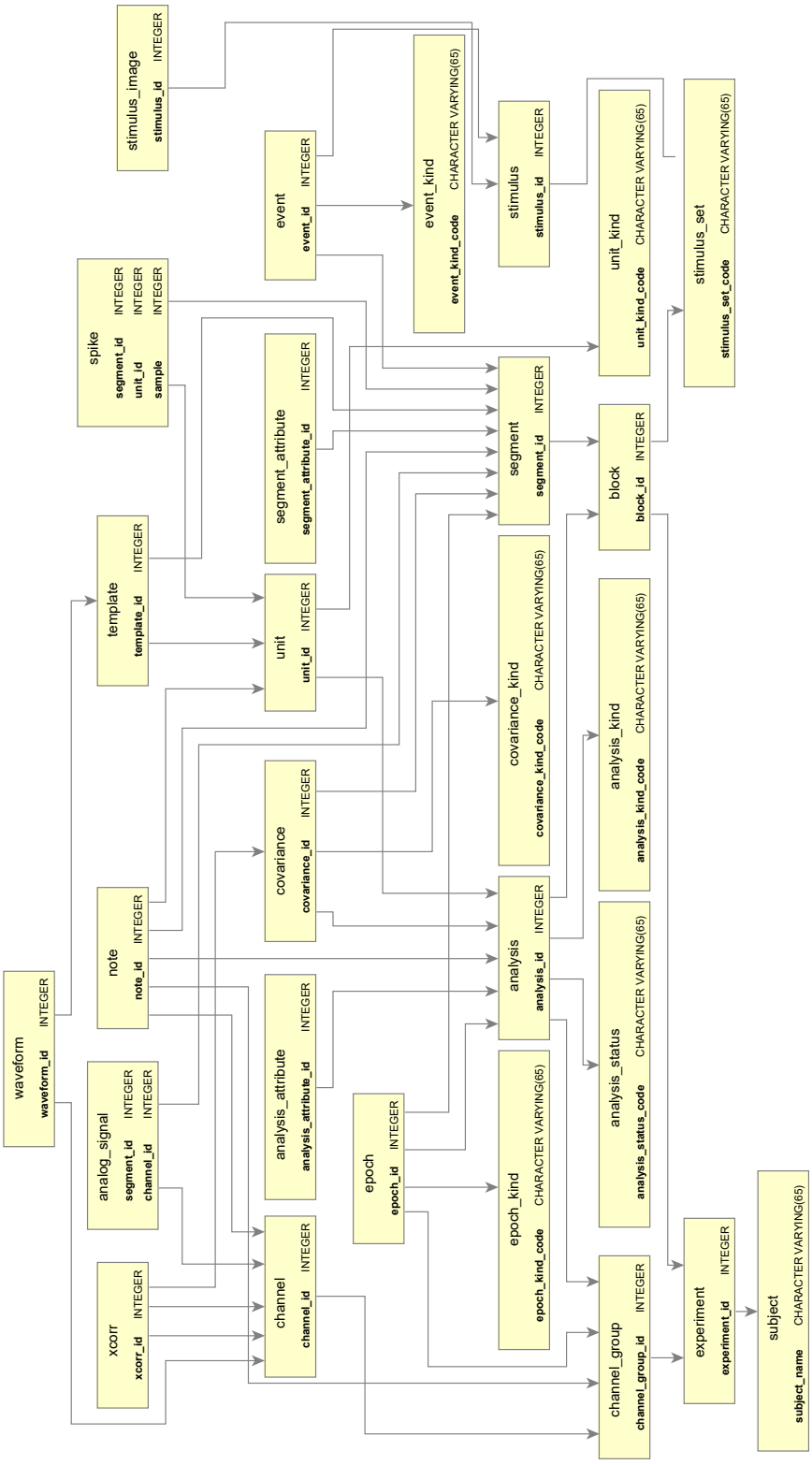

FIGURE 2.12: Schema of the database. Each box represents one table, including information on its primary key. Arrows represent relationships between tables.

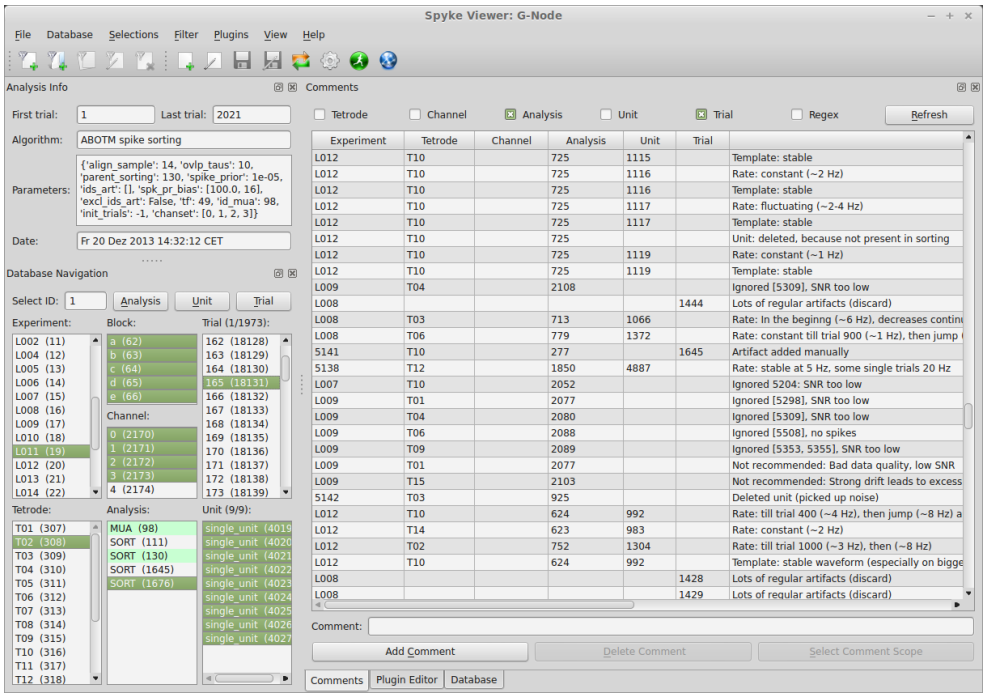

Figure 2.13: Database specific interface elements of Spyke Viewer.

Spike sorting pipeline 3

Action potentials or spikes are the most commonly analyzed information from electrophysiological experiments. However, there are a multiple processing steps involved in obtaining spike times and identities from the raw recorded data. This chapter describes the complete pipeline that was used to extract spikes in the data from the working memory experiment described in section 1.5. The spike sorting method *Bayes Optimal Template Matching* (BOTM) used here was originally described in (Franke, 2011) and more recently in (Franke et al., 2015b), and this chapter presents my enhancements and evaluations of the method.

During the recording in the working memory experiment, signals were digitized at 32 kHz and bandpass filtered to 5-150 Hz for the LFP, and 0.5-10 KHz for the spike data. This chapter is only concerned with the spike data, because the *Local Field Potential* (LFP) does not require similarly complex signal processing. However, some of the analyses in chapter 4 are also based on the LFP.

Parts of this chapter are based on material that will appear in (Franke et al., 2015a):

> Franke, F.*, Pröpper, R.*, Alle, H.*, Meier, P., Geiger, J., Obermayer, K., Munk, M.H.J. (2015). Spike Sorting of Synchronous Spikes from Local Neuron Ensembles. *Journal of Neurophysiology* (in revision).

3.1 Artifact detection

The first step of the data processing pipeline is the removal of artifacts, defined as periods of data that lead to errors in subsequent processing steps. In particular, artifacts would often be detected as spikes and

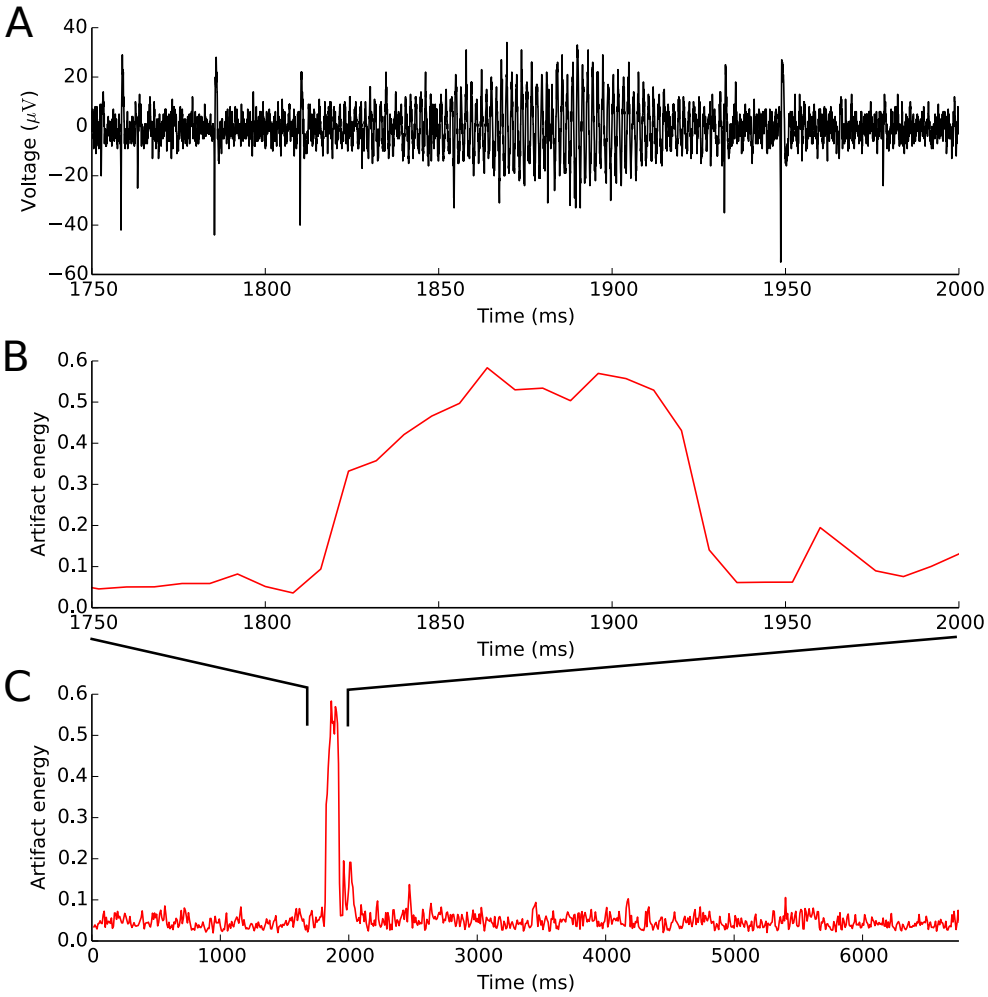

Figure 3.1: Example of a detected oscillation artifact. **(A)** Artifact in the raw data. **(B)** Artifact energy of the same period. **(C)** Artifact energy of the whole trial.

subsequently negatively impact spike sorting performance. The artifacts in the dataset fall into two categories: amplitude artifacts and oscillation artifacts. Detecting amplitude artifacts is straightforward: they are characterized by very large voltage deflections, much larger than those caused by spikes. To find them, it is sufficient to find periods in the data where the absolute voltage value crosses a very large threshold, e.g. 20 times the standard deviation of the recording: $\theta_{\text{Amp}} = 20 \cdot \sigma_{\mathbf{X}}$. Whenever the amplitude on any channel crosses this threshold, an artifact period with a margin of 10 ms from the first and last threshold crossing is defined. Artifacts of this kind are rare in the data and do not occur at all in most recording sessions.

The second kind of artifact is much more frequent. It is characterized by a a strong oscillation on a single frequency that can last from dozens

to hundreds of milliseconds. Figure 3.1 (A) shows an example of such an artifact surrounded by regular recorded data. The strong oscillation does not always occur on the same frequency. The following algorithm can automatically find these artifacts: First, short windows of data (512 samples, i.e. 16 ms, weighted by a Hanning window function to reduce spectral leakage) are converted into the frequency domain using the *Discrete Fourier Transform* (DFT):

$$X_k = \sum_{t=0}^{T-1} x_t \cdot e^{-i2\pi kt/T}, \quad k \in \mathbb{Z} \tag{3.1}$$

The coefficients X_k are T-periodic, so only coefficients with $0 \leq k < T$ need to be considered. Because the phase is not relevant to the artifact decision, the magnitude of each coefficient $|X_k|$ is used. An artifact energy E_{Osc} is then defined as the amplitude of the largest coefficient normalized by the total amplitude:

$$E_{\text{Osc}} = \frac{\max\limits_{0 \leq k < T} |X_k|}{\sum\limits_{0 \leq k < T} |X_k|} \tag{3.2}$$

E_{Osc} is a value between 0 and 1. It increases sharply in windows that contain an oscillation artifact and stays small for regular data (see Figure 3.1 (B) and (C)). A threshold of 0.25 is used to distinguish between artifact and regular windows. The artifact detector uses sliding windows with 50 % overlap, giving a resolution of 8 ms with the smallest possible artifact period being 16 ms long.

All artifact detectors work on each recording channel separately, but an artifact period is marked for the whole tetrode because spike detection and spike sorting use all channels simultaneously. The signals during artifact periods are not used in later processing steps.

3.2 Spike detection

Before spikes can be attributed to putative neurons (referred to as units) during spike sorting, the spikes have to be detected in the data. Though the BOTM method used in this thesis combines spike detection and sorting, it requires knowledge of the number and respective waveforms of all units. Because this information is not available in empirical recordings, it has to be estimated by detecting and clustering spikes on some of the data, the same procedure that many popular spike sorting algorithms employ (Lewicki, 1998; Harris et al., 2000; Rutishauser et al., 2006).

A common way to detect spikes in neural recordings is by a simple threshold on the instantaneous absolute amplitude. Typically, the thresh-

old value θ_{Amp} is calculated as a multiple of the standard deviation of the signal, e.g.

$$\theta_{\text{Amp}} = 4 \cdot \sigma_{\mathbf{X}}. \tag{3.3}$$

However, this method is sensitive to outliers in the data, such as the action potentials themselves. For high firing rates and large spike amplitudes, it leads to high threshold values that can miss spikes with smaller amplitudes. When calculating the threshold based on the *Median Absolute Deviation* (MAD), the influence of outliers is diminished:

$$\text{MAD}_{\mathbf{X}} = \text{median}\left(\left| \mathbf{X} - \text{median}(\mathbf{X}) \right| \right). \tag{3.4}$$

When the noise in the recorded data (excluding the outliers) is assumed to be Gaussian, the relationship between standard deviation and MAD is

$$\sigma_{\mathbf{X}} \approx 1.48 \cdot \text{MAD}_{\mathbf{X}}. \tag{3.5}$$

Therefore, to obtain a threshold comparable to θ_{Amp} in Equation 3.3, but more robust to high firing rates and large spike amplitudes:

$$\theta_{\text{Amp}} = 5.92 \cdot \text{MAD}_{\mathbf{X}}. \tag{3.6}$$

A similar method for calculating the threshold based on the same reasoning is presented in (Quiroga et al., 2004).

Another way to improve upon the standard method of detecting spikes using an amplitude threshold based on the standard deviation is to apply thresholding not on the signal itself, but some function of it (Kim and Kim, 2000). By using an appropriate function that encodes the knowledge of the general shape of a spike waveform, it is possible to detect more spikes in low SNR situations without increasing the number of false positives. Here, the *Multiresolution Teager Energy Operator* (MTEO) (Choi and Kim, 2002; Choi et al., 2006) is used. It is defined using a number of *Teager Energy Operator* (TEO) energies with varying k parameters, each of which is calculated as

$$\text{TEO}_k(x_t) = x_t^2 - x_{t-k}x_{t+k}. \tag{3.7}$$

Each TEO output is convolved with a Hamming window of size $4k+1$ (as suggested in (Choi and Kim, 2002)) and MTEO is defined as their maximum across k-values. To find the action potentials, a threshold needs to be defined on the resulting energy. As for the amplitude, the threshold should be calculated automatically and be robust against high firing rates and large spike waveforms. Because the energy is always positive, the threshold is one-sided and using the MAD is not appropriate. However, calculating the threshold based on the median confers the same advantage:

$$\theta_{\text{MTEO}} = b \cdot \text{median}(\text{MTEO}(\mathbf{X})) \tag{3.8}$$

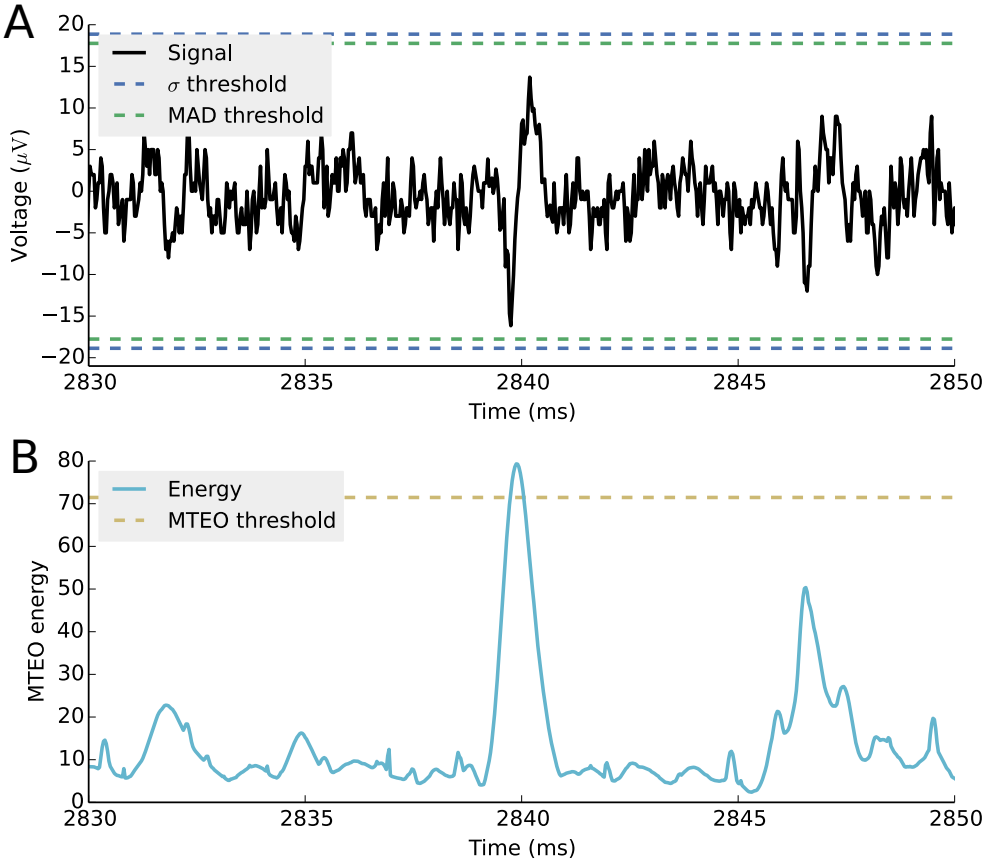

FIGURE 3.2: Spike detection thresholds. **(A)** A short period of recorded noise with an artificially inserted small spike waveform. **(B)** MTEO results for the same data.

where b can be chosen freely in a trade-off between sensitivity and false positives. Figure 3.2 illustrates the three spike detection variants. In the data processing pipeline described here, a MTEO based detector with $b = 5$ is used.

Figure 3.2 illustrates the presented spike detection methods. All thresholds were calculated on the complete recorded trial with a duration of about seven seconds. The MAD-based threshold and the σ-based threshold would miss the depicted spike, but it is detected using the MTEO.

When detecting spikes on multiple recording channels that potentially record action potentials from the same neurons, a separate threshold is calculated for each channel. Whenever the threshold is crossed on any channel, a period of all temporally connected samples above threshold is defined and only the first of these samples is considered as a spike to prevent the same spike being detected on multiple channels.

3.3 Spike sorting

The final and most complex step of the signal processing pipeline is spike sorting. Here, this step is performed using an improved version of the BOTM algorithm. An important advantage of BOTM compared to other algorithms is its ability to resolve overlapping spikes, i.e. action potentials that are recorded in close temporal proximity on the same channels.

3.3.1 The problem of overlapping spikes

A standard assumption for spike sorting algorithms is that all action potentials from a given neuron produce a similar, distinct waveform in the recorded signal. Most spike sorting pipelines work as follows (Einevoll et al., 2012): as a first step, a spike detection algorithm is run. Around each detected spike, a small segment of data is extracted. The data segments are aligned on a particular feature, for example the minimum amplitude. Then, the dimensionality of the data segments is reduced in preparation for the following clustering step. Data with too many dimensions can be problematic for clustering, a problem referred to as curse of dimensionality (see (Bishop, 2006)). Common techniques for dimensionality reduction include *Principal Component Analysis* (PCA) (Harris et al., 2000; Lewicki, 1998) and wavelets (Letelier and Weber, 2000; Quiroga et al., 2004). Another approach is to take features of the waveform shape, such as the amplitude at specific points or the spike height. Calculating such features requires little computation and has been used in early studies and for real-time applications (Simon, 1965; Dinning and Sanderson, 1981). Finally, the spikes are grouped using a clustering algorithm, where each cluster corresponds to a putative neuron.

Temporally overlapping waveforms of two or more neurons constitute a fundamental problem for these approaches to spike sorting, but precise spike times and synchronized neural activity can be an important mechanism for information transmission on various scales. Synchronous *Excitatory Postsynaptic Potentials* (EPSPs) have a larger chance of evoking an action potential in the postsynaptic neuron and different EPSP patterns can be distinguished even at the level of a single neuron (Branco et al., 2010). On a larger scale, concepts such as the synfire chain (Abeles, 1982) rely on neurons firing within milliseconds of each other. In local neuron assemblies neural activity is often highly correlated and the firing behavior of single cells can be predicted from the activity of surrounding neurons (Tsodyks et al., 1999). There is also evidence for functionally relevant synchronous firing in the prefrontal cortex (Pipa and Munk, 2011; Sakurai et al., 2013). But even for overlapping spikes that occur by chance, the ability to correctly resolve them can be very useful for further analyses, because the number of overlaps can form a significant

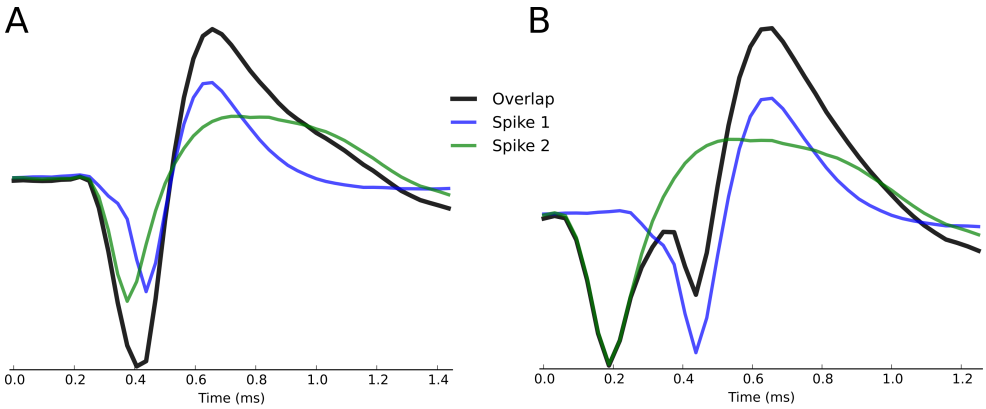

FIGURE 3.3: Examples for temporally overlapping spike waveforms. **(A)** Close overlap. **(B)** An overlap with a larger offset.

fraction of all recorded spikes depending on the number of neurons in the recording and their firing rates.

The joint waveform produced by the overlap will often not result in features that can be used to identify all or even one of the participating neurons. Figure 3.3 shows two examples: an overlap of two waveforms with a small offset often results in a waveform shape that could also be the result of a single spike (Figure 3.3 (A)). If the temporal offset is larger, such as in Figure 3.3 (B), the resulting waveform shape could not be produced by a single neuron, but both source waveforms are heavily distorted. Such overlaps can result in various errors during spike sorting: the new waveform could be classified as an uninvolved neuron or only a subset of the participating neurons could be identified. In some methods, the total overlap could be classified as a special unit that collects spikes that cannot be classified clearly. Finally the joint waveform as a whole could remain below the spike detection threshold so that no participating spike can be classified. Most methods are not explicitly evaluated on overlapping spikes. The wavelet-based spike sorting algorithm presented and evaluated in (Quiroga et al., 2004) performed much worse on overlaps than on single spikes. The simulated dataset included overlaps of two neurons on a single electrode.

The spike sorting results for overlapping spikes can be improved using electrodes with multiple contacts (recording channels) such as tetrodes. Action potentials of most identifiable neurons produce a waveform on more than one channel, depending on the relative position of the neuron to the contacts. Therefore, more robust features can be extracted by considering the spike waveform on multiple recording channels. However, corruption of features by temporal overlaps still takes place. Therefore, overlapping spike waveforms are a fundamental problem for the standard spike sorting approach presented so far. The resulting errors produce

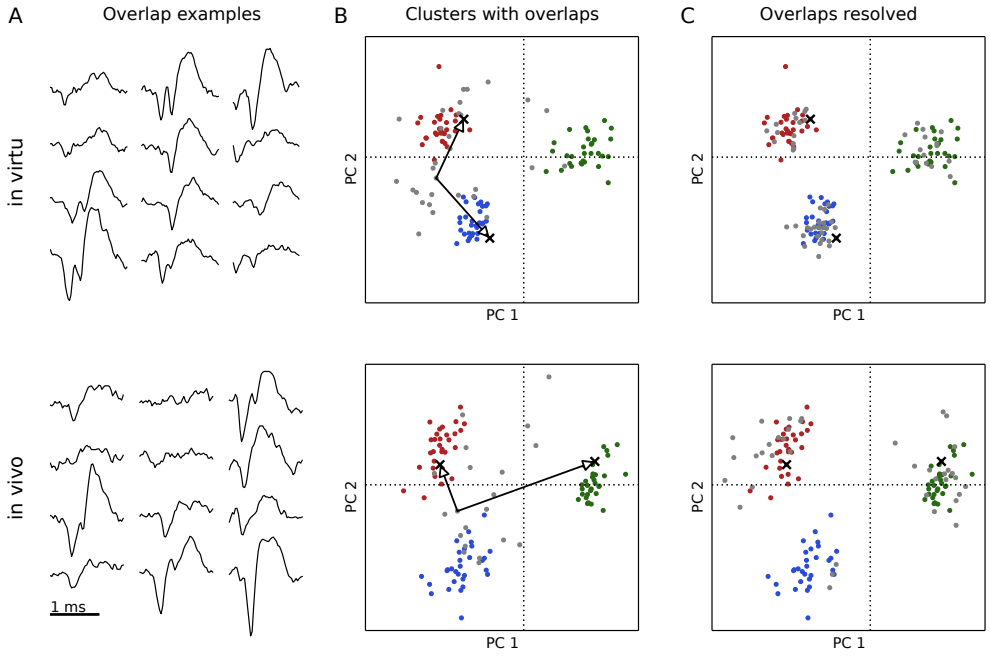

Figure 3.4: Temporally superimposed (overlapping) spike waveforms for simulated (in virtu) and in vivo data. **(A)** Three examples for overlaps of two spikes on four channels. **(B)** Results of a *Principal Component Analysis* (PCA). The first two *Principal Components* (PCs) are shown for overlapping spikes (gray) and single spikes (colored). **(C)** The same data after all overlaps have been resolved into single waveforms.

a bias in the sorted spike trains. Further analyses of the spikes can be strongly affected by this bias (Bar-Gad et al., 2001; Pazienti and Grün, 2006). Investigations based on correlations between nearby neurons such as functional connectivity and local cooperativity (Katsuki et al., 2013; Ohiorhenuan et al., 2010) are especially affected.

Figure 3.4 illustrates the difficulty that cluster-based spike sorting methods face with overlapping spikes. Figure 3.4 (B) shows three colored clusters for simulated and recorded data. Each gray point corresponds to an overlap of two single spikes. The example shows the two *Principal Components* (PCs) corresponding to the largest eigenvalues, other features exhibit similar behavior. For the recorded data, detected spikes from seven seconds of data with three units are included. The simulation is based on the recorded waveforms and noise. For more details on the simulation process see subsection 3.4.6.

Cluster-based methods operate on the points shown without the knowledge encoded in the coloring. For one overlap in both rows, arrows and crosses indicate where the two single waveforms comprising the overlap would be located in the depicted feature space. Clearly, assigning clusters to the gray overlaps is problematic: their features can be different

from all single spike clusters and even when one of the participating units is identified correctly, the other would be missed.

Figure 3.4 (C) shows the same data, but with two gray points for each overlap: the features of each participating spike after the respective other spike waveform has been removed from the data. Now the clusters are clearly separable and each spike, including those participating in overlaps, can be attributed to the appropriate unit.

Because of the central importance of temporally overlapping spikes for many analyses, there have been significant efforts to find methods that improve the spike sorting results for these events. The most common approach is based on template matching (Atiya, 1992; Lewicki, 1994; Prentice et al., 2011; Segev et al., 2004; Vargas-Irwin and Donoghue, 2007; Zhang et al., 2004): a prototypical waveform (the template) for each unit is compared to detected spikes and each spike is assigned to the unit with the smallest distance. By creating overlap templates of all possible combinations of neurons and temporal shifts, overlapping spikes can be sorted as well. However, this approach is computationally extremely expensive and therefore impractical for long recordings with a large number of units.

Another proposed method uses *Independent Component Analysis* (ICA) to separate the recorded signals into additive subcomponents generated by neurons (Brown et al., 2001; Madany Mamlouk et al., 2005). ICA is only possible with multielectrodes and is generally restricted to a single component and therefore at most one neuron per recording channel. This makes the method problematic for devices like tetrodes, which often record more than four neurons simultaneously, and impossible to apply for single electrodes. For multielectrodes, this problem can be mitigated through an additional regular clustering step (Takahashi et al., 2003). Another issue with ICA is the inherent assumption that the waveforms of each spike are linearly dependent on different recording channels, meaning the same shape is recorded on each channel with the only difference being their amplitude. This assumption does not generally hold (Gold et al., 2006; Jäckel et al., 2012; Shiraishi et al., 2009).

A promising approach to resolving overlapping spikes are filter-based methods (Roberts and Hartline, 1975; Stein et al., 1979; Vollgraf et al., 2005). As with template matching, a prototypical waveform for each unit is needed. This template is then used to construct a linear filter that is applied to the data. Using deconfusion (Franke et al., 2010) or a probabilistic model (Franke, 2011), it is possible to resolve overlapping spikes. Once filters are constructed, a separate spike detection step is no longer necessary because the filters detect and classify spikes simultaneously. Another recently proposed method (Ekanadham et al., 2014; Pillow et al., 2013) is based on a similar probabilistic model as (Franke,

2011) but does not use linear filters and therefore requires a previous spike detection step.

The remainder of this chapter describes the BOTM algorithm originally presented in (Franke, 2011) and the extensions to its overlap resolution method. In addition, extensive simulations and data recorded from monkey PFC show how the algorithm successfully resolves overlapping spikes of two and more neurons at arbitrary time shifts with the same classification performance as non-overlapping spikes.

3.4 Bayes Optimal Template Matching

Bayes Optimal Template Matching (BOTM) is a filter-based spike sorting method that is Bayes optimal if the assumptions of its underlying generative model hold. It assumes knowledge of the noise covariance matrix and spike templates for all recorded neurons.

3.4.1 Generative model

The generative model is the same that explicitly or implicitly underlies most spike sorting approaches (Pouzat et al., 2002). The general assumptions are:

1. The spike waveform $\boldsymbol{\xi}^i$ generated by each neuron i, called template, is distinct and constant over some time period.

2. The generated spike trains s^i are statistically independent of the noise η.

3. Spike waveforms and noise sum linearly.

4. The noise is normally distributed and described by a covariance matrix $\mathbf{C}$.

The spike templates $\boldsymbol{\xi}^i$ have a finite length of T_f samples on all N_C recording channels. Likewise, $\mathbf{C}$ is a symmetric $T_f \cdot N_C$ by $T_f \cdot N_C$ matrix consisting of N_C by N_C Toeplitz blocks (a banded matrix where all entries on each diagonal have the same value). The spike trains s^i are 1 at a single sample for each spike and 0 everywhere else.

The time period for assumption 1 can be quite small. Assuming that templates only change slowly, they could be re-estimated after every spike based on the waveforms of recently found spikes from the same unit. The recorded data X can then be expressed as

$$x_t = \sum_i \sum_\tau s^i_{t-\tau} \xi^i_\tau + \eta_t, \tag{3.9}$$

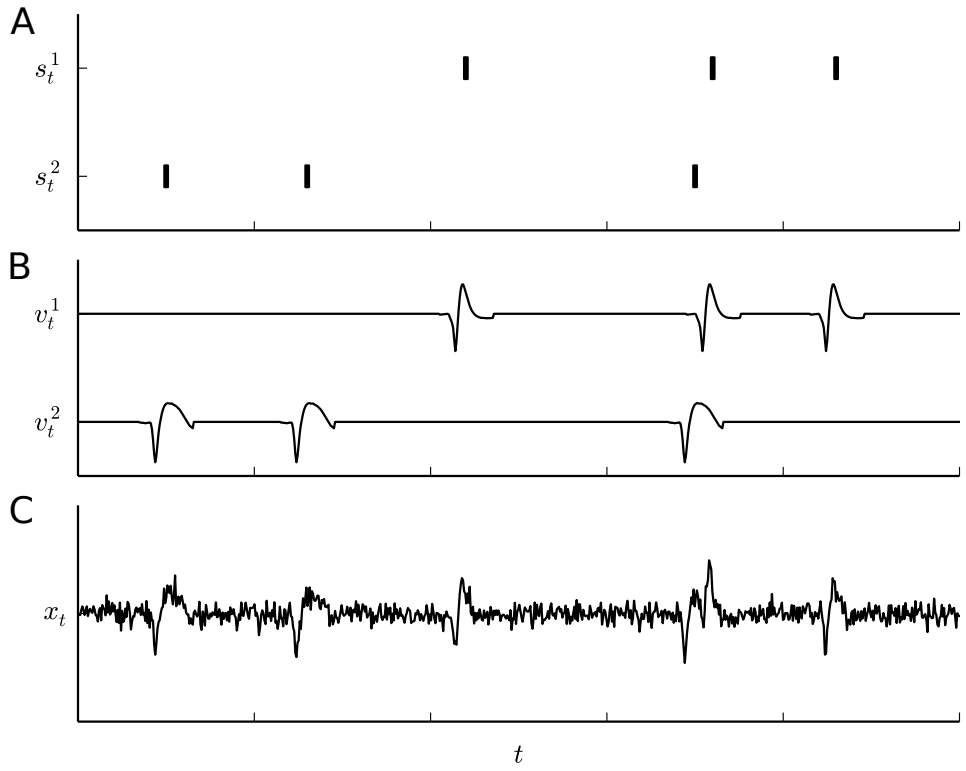

FIGURE 3.5: Illustration of the generative model. **(A)** Spike times of two neurons. **(B)** Signal produced by the two neurons. **(C)** The signal of both neurons combined with noise.

where t marks a time point in the recording, i iterates over neurons and τ iterates over template samples. Therefore, the recorded data is modeled as a convolution of the spike trains and templates for all units with additive colored Gaussian noise. x_t, ξ_τ^i and η_t can be scalars in case of a single recording channel or vectors when modeling multiple channels.

Figure 3.5 shows a simple example for the generative model. Two neurons 1 and 2 produce three spikes each. The corresponding signals are shown in Figure 3.5 (B), where $v_t^i = \sum_\tau s_{t-\tau}^i \xi_\tau^i$. The resulting data when both signals are combined and noise is added can be seen in Figure 3.5 (C).

The spike sorting problem is to invert the generative model: given the recorded data $\mathbf{X}$, find all spike trains s^i. However, the noise covariance matrix $\mathbf{C}$, the templates $\boldsymbol{\xi}^i$, and even the number of neurons are usually not known and have to be estimated from the data. For this estimation, BOTM requires an initialization step.

3.4.2 Initialization step

The initialization used on the recorded data from the macaque experiment described in section 1.5 resembles a regular clustering-based spike sorting. It runs on a short subset of data (around two minutes) to avoid problems caused by slowly changing spike waveforms from gradual changes of the relative positions of neurons and tetrode. The first step uses a spike detection based on MTEO as described in section 3.2. The detected spikes are aligned on their minimum amplitude. Next, a number of features are calculated for each spike: the first eight PCs, and the peak to peak amplitude of the waveform on each channel. The spike heights contain information that is not captured by the PCs (which are most commonly used as features for spike sorting) and their use improves clustering results in some situations.

The PCA to obtain the PCs is not computed directly on the data. First, the spikes are prewhitened by multiplying each with the square root of the inverse noise covariance matrix $\mathbf{C}^{-0.5}$. The prewhitening step ensures that the noise on the spikes is approximately standard normal distributed. This can improve PCA results by ensuring that the recording channels are uniformly scaled. In addition, it removes variance caused by noise so that the variance from waveform differences between units dominates, making it more likely that the PCs with the largest eigenvalues correspond to directions that separate the clusters well.

The clustering step on the extracted 12 features uses *Gaussian Mixture Models* (GMMs): a GMM is fitted using the *Expectation Maximization* (EM) algorithm. Since the EM algorithm needs to know the number of Gaussian components, i.e. the number of units in the data, a variety of cluster counts are used and the best model is chosen using the *Bayesian Information Criterion* (BIC). It is possible to manually discard or merge clusters when inspecting the sorting results with Spyke Viewer. This method is similar to the widely used KlustaKwik algorithm (Harris et al., 2000). The mean waveforms of the clusters from the best model are then used as the initial templates $\boldsymbol{\xi}^i$.

To estimate the noise covariance matrix, all detected spikes are cut from the data and the remaining periods are used to calculate the initial $\mathbf{C}$. However, this calculation contains a pitfall: Simply stitching the noise periods together and calculating the covariance from the combined data leads to an underestimation of the noise covariance, because the samples at the period borders were originally further apart and have an expected covariance of zero. Therefore, $\mathbf{C}$ is estimated separately for each noise period and the estimations are averaged, weighted by the length of their respective period.

During the sorting with BOTM, both the templates and the covariance matrix are continuously updated because both can change slowly over

time. After each sorted trial (about seven seconds), all new spikes are inserted into a circular buffer that contains the most recent spikes for each unit. The mean of all spikes in the buffer is then used as template for the next trial. For the covariance matrix, the periods without spikes from the most recent trial are weighted by their length and combined with data from previous recent trials to calculate $\mathbf{C}$.

3.4.3 Spike sorting by filtering

Now that the prerequisite information has been estimated, the actual spike sorting can take place. Starting at the beginning of the data and therefore reprocessing the period used for initialization, the BOTM algorithm is applied. It calculates a discriminant function $d(i,t)$ for each template at every sample of the recording:

$$d(i,t) := \mathbf{x}(t)^T \mathbf{C}^{-1} \boldsymbol{\xi}^i - \frac{1}{2} \boldsymbol{\xi}^{iT} \mathbf{C}^{-1} \boldsymbol{\xi}^i + \ln(p(i)), \tag{3.10}$$

where $\mathbf{x}(t)$ is a period of data of length T_f starting at time t and $p(i)$ is the prior probability that a spike of unit i occurs at any given sample. The influence $\ln(p(i))$ is small compared to the other terms when set to a value based on realistic firing rates so it is usually set to the same constant for all units. However, it is also possible to estimate the spiking probability using empirical Bayes methods: firing rates from spike sorting results earlier in a recording can give an estimation of the spiking probability.

Equation 3.10 is closely related to the discriminant function used in *Linear Discriminant Analysis* (LDA) (original derivation of BOTM using LDA in (Franke, 2011)): To decide to what class (i.e. unit) a spike $\mathbf{x}(\mathbf{t})$ belongs, calculate the conditional probability of each class given the data and choose the maximum: $\text{argmax}_i\, p(i|\mathbf{x}(\mathbf{t}))$. By using Bayes' rule, the likelihood of the data can be maximized instead of the conditional probability:

$$\text{argmax}_i\, p(i|\mathbf{x}(\mathbf{t})) = \text{argmax}_i \frac{p(\mathbf{x}(\mathbf{t})|i)p(i)}{\sum_j p(\mathbf{x}(t)|j)p(j)} \tag{3.11}$$

Since the denominator is the same for all classes, it can be ignored when finding the maximum. The LDA makes some assumptions that correspond with the generative model described in subsection 3.4.1: Each class is assumed to be normally distributed, with the same covariance for all classes. In the context of the generative model, the class means are the templates ξ^i and the covariance is $\mathbf{C}$. Using these assumptions with k the number of dimensions (i.e. the number of samples in $\mathbf{x}(t)$ and $\boldsymbol{\xi}^i$) and $|C|$ the determinant of C, a discriminant can be defined as

$$\begin{aligned}\mathrm{disc}(i,t) &:= (2\pi)^{-\frac{k}{2}}|\mathbf{C}|^{-\frac{1}{2}}e^{-\frac{1}{2}(\mathbf{x}(t)-\boldsymbol{\xi}^i)^T\mathbf{C}^{-1}(\mathbf{x}(t)-\boldsymbol{\xi}^i)}p(i)\\ &= (2\pi)^{-\frac{k}{2}}|\mathbf{C}|^{-\frac{1}{2}}e^{-\frac{1}{2}(\mathbf{x}(t)^T\mathbf{C}^{-1}\mathbf{x}(t)-2\mathbf{x}(t)^T\mathbf{C}^{-1}\boldsymbol{\xi}^i+\boldsymbol{\xi}^{iT}\mathbf{C}^{-1}\boldsymbol{\xi}^i)}p(i).\end{aligned} \tag{3.12}$$

The terms $(2\pi)^{-\frac{k}{2}}|\mathbf{C}|^{-\frac{1}{2}}$ and $\mathbf{x}(t)^T\mathbf{C}^{-1}\mathbf{x}(t)$ are independent of i and can be ignored when searching the maximum. Dropping the two terms and taking the logarithm results in Equation 3.10. Therefore, assigning a period of data $\mathbf{x}(t)$ to the class with the maximum discriminant is the same as performing an LDA classification.

The term $\mathbf{x}(t)^T\mathbf{C}^{-1}\boldsymbol{\xi}^i$ in Equation 3.10 is equivalent to a linear filtering operation with the matched filter

$$\mathbf{f}^i := \mathbf{C}^{-1}\boldsymbol{\xi}^i \tag{3.13}$$

and the term $-\frac{1}{2}\boldsymbol{\xi}^{iT}\mathbf{C}^{-1}\boldsymbol{\xi}^i$ is a normalization by the expected filter output of the template. It does not change as long as the template remains constant so it does not have to be recalculated at every sample and can therefore be treated as a constant

$$c^i := -\frac{1}{2}\boldsymbol{\xi}^{iT}\mathbf{C}^{-1}\boldsymbol{\xi}^i. \tag{3.14}$$

Finally, the term $\ln(p(i))$ does not have to remain constant: $p(i)$ remaining the same at every sample assumes that the spike train is generated by a homogeneous Poisson process. It could be generated by a more general inhomogeneous process that depends on the spike history, so the probability would become $p(i,t|s^i_{0\ldots t-1})$, where $s^i_{0\ldots t-1}$ is the spike train of unit i up to time $t-1$. It would even be possible to incorporate more complex models of spiking, for example including dependencies the spiking history of other units such as the generalized linear models used in (Pillow et al., 2008). However, realistic values for the spike probability only have a small influence on the discriminant except for situations with very small SNR where the other terms become small as well. In addition, the information needed for more complex probability models is usually not available during spike sorting. Still, there is one effect that is always present and comparatively simple to model: the absolute refractory period that neurons exhibit after generating an action potential. Modeling the refractory period only requires the information when the previous spike of the same unit occurred, defined here as $s^i(t)$. The probability becomes $p(i,t|s^i(t))$. The probability is set to a value ϵ close to 0 for a period of 0.5 ms after each spike because it is impossible for any neuron to fire again so quickly:

$$p(i,t|s^i(t)) = \begin{cases}\epsilon, & \text{if } 0 < t - s^i(t) \leq 0.5 \text{ ms}\\ p(i), & \text{otherwise.}\end{cases} \tag{3.15}$$

With these definitions and modifications, Equation 3.10 becomes

$$d(i,t) := \mathbf{x}(t)^T \mathbf{f}^i + c^i + \ln(p(i,t|s^i(t))). \tag{3.16}$$

The algorithm can now decide which unit i fired a spike at a given sample t by finding $\text{argmax}_i\, d(i,t)$. However, there are no spikes in many samples. Therefore, another template is introduced to account for noise. As the noise is assumed to have a mean of zero, the template is simply the zero vector $\mathbf{0}$ and its discriminant would be

$$d(i,t) := \mathbf{x}(t)^T \mathbf{C}^{-1}\mathbf{0} - \frac{1}{2}\mathbf{0}^T\mathbf{C}^{-1}\mathbf{0} + \ln(p(n)) = \ln(p(n)), \tag{3.17}$$

with $p(n)$ the prior of a sample not containing a spike. Since the probability for spikes now depends on time and previous spike times because the refractory period is considered, the noise prior is calculated as

$$p(n,t|s(t)) = 1 - \sum_i p(i,t|s^i(t)). \tag{3.18}$$

The discriminant for the noise template can be interpreted as a threshold: when any other discriminant is higher, that sample is a candidate for containing a spike. Therefore, the spike threshold for BOTM is

$$\theta_{\text{BOTM}} = \ln(p(n,t|s(t))). \tag{3.19}$$

3.4.4 Overlap resolution

When templates of multiple units have similar waveforms, multiple discriminants will often be above the noise threshold because the presence of different spikes might be a better explanation for the data than pure noise. Therefore, the fact that multiple discriminants reach the threshold within a short time is often not caused by overlapping spikes. To avoid false positive spikes, whenever the spike threshold is reached, the algorithm searches for the maximum discriminant value in a period of time from the first threshold crossing until all discriminants return below the threshold, and only registers a spike for the corresponding unit.

However, with this approach it would only be possible to detect at most one spike participating in an overlap. There are two ways for BOTM to deal with overlaps. First, additional templates can be created for all combinations of multiple units and time shifts. These overlap discriminants have to include spike priors for all participating units and can be calculated from the discriminants for single spikes. For two units

i, j and their combined template $\boldsymbol{\xi}^i + \boldsymbol{\xi}^j_\tau$ at offset τ, where $\boldsymbol{\xi}^j_\tau$ denotes the template $\boldsymbol{\xi}^j$ shifted by τ samples, the equation is (Franke, 2011):

$$\begin{aligned} d(i,j,\tau,t) &:= \mathbf{x}(t)^T\mathbf{C}^{-1}(\boldsymbol{\xi}^i + \boldsymbol{\xi}^j_\tau) - \frac{1}{2}(\boldsymbol{\xi}^i + \boldsymbol{\xi}^j_\tau)^T\mathbf{C}^{-1}(\boldsymbol{\xi}^i + \boldsymbol{\xi}^j_\tau) \\ &\quad + \ln(p(i,t|s^i(t))) + \ln(p(j,t+\tau|s^j(t+\tau))) \\ &= \mathbf{x}(t)^T\mathbf{C}^{-1}\boldsymbol{\xi}^i + \mathbf{x}(t)^T\mathbf{C}^{-1}\boldsymbol{\xi}^j_\tau \\ &\quad - \frac{1}{2}\boldsymbol{\xi}^{iT}\mathbf{C}^{-1}\boldsymbol{\xi}^i - \frac{1}{2}\boldsymbol{\xi}^{j\,T}_\tau\mathbf{C}^{-1}\boldsymbol{\xi}^j_\tau \\ &\quad - \frac{1}{2}(\boldsymbol{\xi}^{iT}\mathbf{C}^{-1}\boldsymbol{\xi}^j_\tau + \boldsymbol{\xi}^{j\,T}_\tau\mathbf{C}^{-1}\boldsymbol{\xi}^i) \\ &\quad + \ln(p(i,t|s^i(t))) + \ln(p(j,t+\tau|s^j(t+\tau))) \\ &= d(i,t) + d(j,t+\tau) - \boldsymbol{\xi}^{iT}\mathbf{C}^{-1}\boldsymbol{\xi}^j_\tau, \end{aligned} \tag{3.20}$$

assuming statistical independence of both units. Higher order overlap discriminants can be successively calculated from lower orders. Unfortunately, while this is the mathematically correct way of dealing with overlaps, it is prohibitively slow to calculate and analyze all the necessary discriminants. Even when only considering overlaps of two spikes, the computational cost is too high for most use cases.

Subtractive Interference Cancellation (SIC) is a much faster, but heuristic algorithm to deal with overlaps. It works by iteratively finding and removing the influence of spikes from the data. Because it starts by greedily selecting the spike with the highest discriminant, it is not exact and can sometimes fail where overlap discriminants work correctly. It can also be calculated directly based on the existing discriminants to improve performance, but there are multiple ways to recalculate the discriminants after each iteration. In all cases, the scenario is as follows: A spike from unit i is found in the data at sample t_0. The influence of the spike should be removed from the data and therefore from all discriminants. Afterwards, the updated discriminants are checked whether further spikes exist. If another spike is found, the process is repeated.

The first way (SIC A) to update the discriminants was introduced in (Franke, 2011): For each unit j (after a spike of unit i was found at t_0), calculate the new discriminant as

$$d^A_{i,t_0}(j,t) = d(j,t) - \boldsymbol{\xi}^{iT}\mathbf{C}^{-1}\boldsymbol{\xi}^i_\tau + \ln(p(i,t_0|s^i(t_0))), \tag{3.21}$$

with $\tau = t_0 - t$. The update only needs to take place in a period of T_f samples around t_0 because the discriminants do not change further away.

Equation 3.21 follows when spike i, which has already been found, is subtracted from the overlap discriminant as calculated in Equation 3.20:

$$\begin{aligned} d^A_{i,t_0}(j,t) := &(\mathbf{x}(t) - \boldsymbol{\xi}_i)^T \mathbf{C}^{-1}(\boldsymbol{\xi}^i + \boldsymbol{\xi}^j_\tau - \boldsymbol{\xi}^i) \\ &- \frac{1}{2}(\boldsymbol{\xi}^i + \boldsymbol{\xi}^j_\tau - \boldsymbol{\xi}^i)^T \mathbf{C}^{-1}(\boldsymbol{\xi}^i + \boldsymbol{\xi}^j_\tau - \boldsymbol{\xi}^i) \\ &+ \ln(p(i,t_0|s^i(t_0))) + \ln(p(j,t+\tau|s^j(t+\tau))) \\ = &\mathbf{x}(t)\mathbf{C}^{-1}\boldsymbol{\xi}^j_\tau - \boldsymbol{\xi}^T_i \mathbf{C}^{-1}\boldsymbol{\xi}^j_\tau - \frac{1}{2}{\boldsymbol{\xi}^j_\tau}^T \mathbf{C}^{-1}\boldsymbol{\xi}^j_\tau \\ &+ \ln(p(i,t_0|s^i(t_0))) + \ln(p(j,t+\tau|s^j(t+\tau))) \\ = &\, d(j,t) - {\boldsymbol{\xi}^i}^T \mathbf{C}^{-1}\boldsymbol{\xi}^i_\tau + \ln(p(i,t_0|s^i(t_0))). \end{aligned} \tag{3.22}$$

However, this update rule does not produce an identical result as removing the previously found spike from the data and recalculating all discriminants from the modified data. For this interpretation (SIC B), the found template has to be subtracted from the regular discriminants shown in Equation 3.10 and Equation 3.16:

$$\begin{aligned} d^B_{i,t_0}(j,t) := &(\mathbf{x}(t) - \boldsymbol{\xi}_i)^T \mathbf{C}^{-1}(\boldsymbol{\xi}^i + \boldsymbol{\xi}^j_\tau - \boldsymbol{\xi}^i) \\ &- \frac{1}{2}(\boldsymbol{\xi}^i + \boldsymbol{\xi}^j_\tau - \boldsymbol{\xi}^i)^T \mathbf{C}^{-1}(\boldsymbol{\xi}^i + \boldsymbol{\xi}^j_\tau - \boldsymbol{\xi}^i) \\ &+ \ln(p(j,t+\tau|s^j(t+\tau))) \\ = &\mathbf{x}(t)\mathbf{C}^{-1}\boldsymbol{\xi}^j_\tau - \boldsymbol{\xi}^T_i \mathbf{C}^{-1}\boldsymbol{\xi}^j_\tau - \frac{1}{2}{\boldsymbol{\xi}^j_\tau}^T \mathbf{C}^{-1}\boldsymbol{\xi}^j_\tau \\ &+ \ln(p(j,t+\tau|s^j(t+\tau))) \\ = &\, d(j,t) - {\boldsymbol{\xi}^i}^T \mathbf{C}^{-1}\boldsymbol{\xi}^i_\tau. \end{aligned} \tag{3.23}$$

The only difference between the two update rules is the existence of the prior probability for the previously found spike $\ln(p(i,t_0|s^i(t_0)))$. However, this term has already been applied to the discriminant of unit i when the spike was first found. The prior probability after the spike is found should be 1, because it is now known that there is a spike of unit i at t_0. Therefore, the additional prior term in SIC A does not correspond to the interpretation of removing a spike from the data and recalculating. Instead, the term can be thought of as an additional overlap prior that makes it less likely for spikes to participate in an overlap. It could be set to any value and does not have to be identical to the spike prior. Such an additional overlap prior is usually not desirable: it is not part of the generative model and can lead to missed overlapping spikes when the SNR is low (see Figure 3.6).

The third heuristic (SIC C) takes another perspective: when a spike is found at t_0, it calculates the values that overlap discriminants would have had around t_0. Again, this can be accomplished using the existing

discriminants. Following Equation 3.20, these new discriminants can be calculated as

$$d^C_{i,t_0}(j,t) = d(j,t) + d(i,t_0) - \boldsymbol{\xi}^{i^T}\mathbf{C}^{-1}\boldsymbol{\xi}^i_\tau. \tag{3.24}$$

Again, the difference to the other heuristics is only a single, constant term. Here, it is $d(i,t_0)$, i.e. the value of the discriminant for the spike that has already been found. This value is almost always positive: it has to be above the noise threshold, which is usually a very small negative number. Therefore, SIC C is more sensitive than the other variants. It can detect an overlap under the same circumstances as overlap discriminants can, provided that the first spike can be found with single discriminants. This can be useful for templates with a low SNR that can be easier to detect if they participate in an overlap, because overlaps often have a higher SNR than single spikes. On the other hand, the increased detection performance for small templates in overlaps can also be problematic. It can lead to increased false positives and it can introduce a slight bias towards more overlaps: The false negative rate of overlaps can become lower than that of single spikes, increasing the relative amount of overlapping spikes.

Figure 3.6 illustrates the effects of the three SIC heuristics with a simple example. The simulated overlap consists of two triangle shaped templates. For simplicity, both templates are identical, so there is only one artificial unit. While spikes of a single unit would not overlap in real data, the situation is methodically the same and makes the illustration clearer because fewer discriminants are involved. Figure 3.6 (B) shows the discriminants for the template and the overlap at the offset found in the data, as well as their peak values. The overlap discriminant is undefined in the early samples because the first spike would be outside of the data. In this situation, the second spike would be detected first, as its peak is higher. Figure 3.6 (D) depicts the discriminants for all SIC iteration types. SIC A would not find this spike as it stays below the noise threshold. The other variants detect the spike, and SIC C attains the exact same value at its peak as the overlap discriminant in Figure 3.6 (B). SIC B has been used for all the following analyses.

3.4.5 Hybrid

In the results of the overlap simulation described in subsection 3.4.6, most of the errors happened for overlaps with very small offsets of less than 0.5 ms. The distortion of the waveform is largest for small offsets, so there is a higher risk of misclassifying the first spike or finding the correct template but not at the correct sample, causing errors in later SIC iterations. Most of these errors occur because of the SIC heuristic and would be avoided when using overlap discriminants.

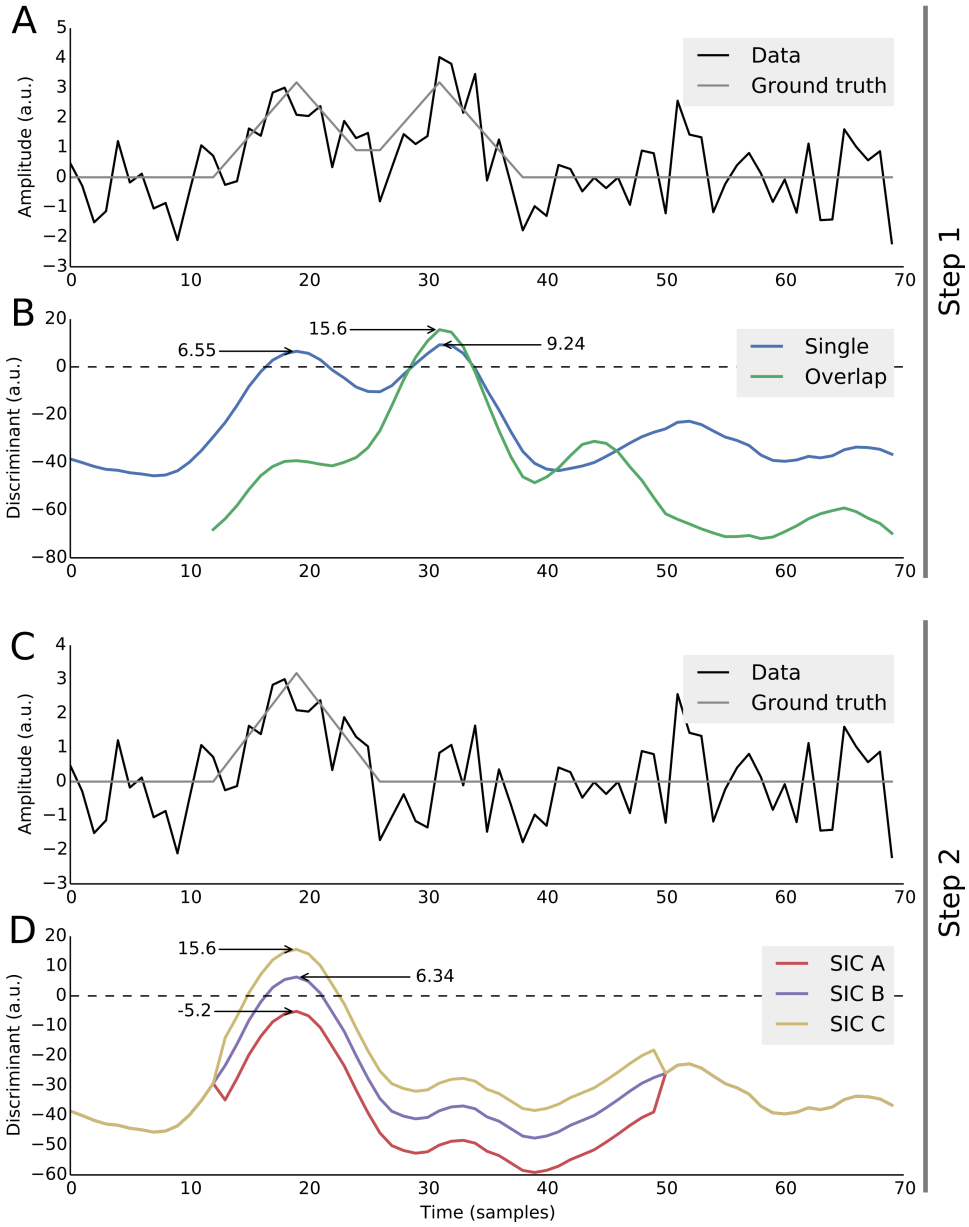

FIGURE 3.6: Toy example illustrating the different SIC heuristics. **(A)** Simulated data with two identical triangular spikes and Gaussian noise. **(B)** Single and overlap discriminants for data shown in (A). **(C)** Data after subtracting the second spike. **(D)** Discriminants for all SIC variants.

The idea behind the BOTM hybrid algorithm is to combine overlap discriminants and SIC to keep high performance but avoid errors at small overlap offsets. It works by calculating a relatively small number of overlap discriminants for close overlaps of two templates only. These discriminants are then used in the SIC iterations in the same way as the single discriminants. The method allows users to adjust the highest offset for which overlap discriminants are calculated as a trade-off between performance and accuracy.

An unwanted side effect occurs at the overlap discriminants with the largest offset: Because they combine two templates, their value is sometimes higher than that of single discriminants even if the offset between the participating spikes is actually larger. For example, an overlap discriminant for offset 10 could attain a higher maximum value than the single discriminants for the participating samples in an overlap with an actual offset of 12. The spikes would then be detected at the wrong samples. While this would not be a problem for most analyses as the offset is mostly smaller than 0.1 ms, it can cause errors in the following SIC iterations: the residual of a spike waveform that is subtracted from the data at the wrong sample is further away from pure noise. It can lead to a false positive detection or misclassification of another spike participating in the overlap. To remove this border effect, the hybrid algorithm temporarily disables the overlap discriminant when an overlap at the largest calculated offset is found. In that case, the overlap is resolved using SIC only. Both cases of overlap resolution are depicted in Figure 3.8.

For the macaque dataset, 10 samples (about 0.3 ms) around offset zero are enough to prevent most of the additional errors. Depending on the number of units, the hybrid algorithm with these parameters is three to ten times slower than regular BOTM with SIC only. On modern hardware, that can still be fast enough for sorting one tetrode per CPU core in real-time.

Figure 3.7 shows how the hybrid algorithm affects errors using simulated data based on a real recording (see subsection 3.4.6). For this example, 1000 overlaps of two templates were simulated for every offset. A false positive was defined as any template being found that was not part of the overlap, a false negative as one of the participating templates not being found. Correct examples contain neither false positives nor false negatives. The performance of the regular SIC algorithm is shown in Figure 3.7 (A). A clear drop in the accuracy is visible for offsets up to about 0.3 ms. The hybrid BOTM performance depicted in Figure 3.7 (B) shows no such drop. The accuracy even rises slightly at offsets for which the overlap discriminants are calculated. Over all offsets, the average error rate is 1.7 % for regular SIC is 1.0 % for hybrid SIC.

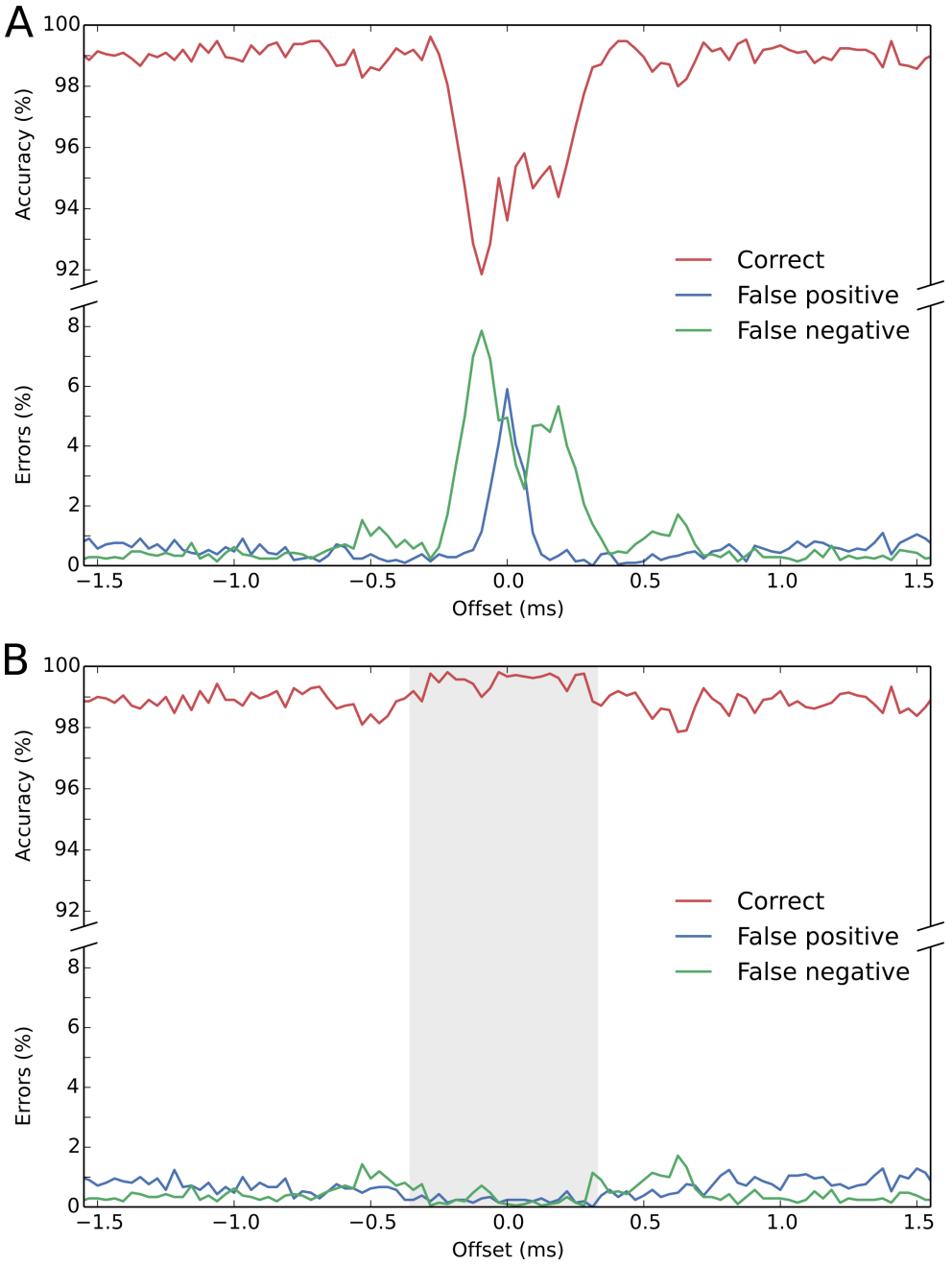

FIGURE 3.7: Example of performance on simulated overlaps at different offsets. **(A)** Regular SIC. **(B)** Hybrid method with overlap discriminants up to 0.3 ms. Offsets at which overlap discriminants are calculated are marked in gray.

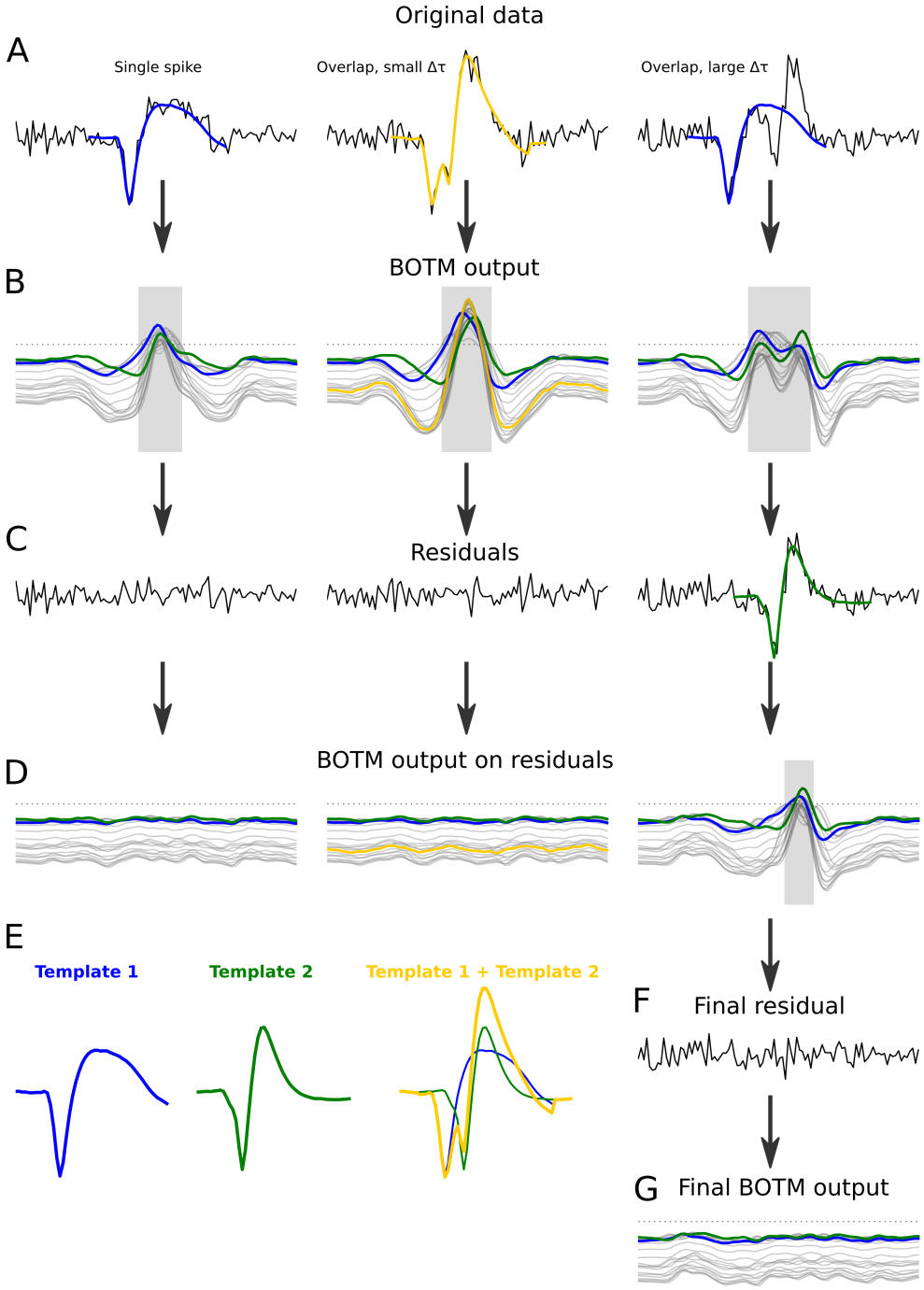

Figure 3.8: Example of spike and overlap resolution with BOTM. **(A)** Raw data of a single spike and two overlaps with superimposed spike templates for spikes found in (B). **(B)** BOTM output for two templates and all overlap templates used in hybrid BOTM. **(C)** Residuals after subtracting detected spikes. **(D)** BOTM output on the residuals. **(E)** The two templates used in the example and the overlap template at the offset used in the center column. **(F)** Residual after the second spike has been subtracted. **(G)** BOTM output on the second residual.

Figure 3.8 illustrates how the hybrid BOTM algorithm resolves a single spike and two overlaps at different offsets. First, the discriminants for both templates and all overlap discriminants (shown in gray) are calculated. The shaded gray area in Figure 3.8 (B) and (D) indicates the time period during which the discriminants exceed the spike threshold. The maximum discriminant at its highest point during this period determines which spikes are detected at what samples. In the left column, a single spike exists in the data and is detected. The data shown in the middle column contains a close overlap of two spikes, both of which are detected simultaneously by one of the overlap discriminants (marked in yellow). In the right column, an overlap with a larger offset is resolved in two steps using SIC because no overlap discriminants are calculated for such a large offset.

3.4.6 Simulating overlapping spikes

Analyses that assess the performance of spike sorting algorithms require a ground truth to compare the spike sorting results with. Such data could be obtained by simultaneously recording extracellular and ground truth data, for example with a tetrode and an intracellular recording technique such as patch clamp. For assessing sorting performance with overlaps, a single intracellularly recorded cell is not sufficient. Currently, the only dataset that includes two cells stimulated explicitly to produce overlapping spikes is used in (Franke et al., 2015a), but the results are not reproduced here because the analysis of that data was not part of this thesis. At the time of writing, datasets containing more than two cells with precise spike times are not available.

Simulation is an alternative to simultaneous recording for generating data with ground truth about spike times and identities. The recently developed ViSAPy (Hagen et al., 2015) is a flexible but complex way to generate such data. The simulation method used here is based on the macaque dataset (section 1.5) and illustrated in Figure 3.9. First, spike detection is used to find spikes in the recorded data. Periods that do not contain spikes are marked as noise. The detected spikes are then sorted using the GMM based algorithm described in section 3.4 to acquire the number of units and their cluster means. Because spikes do not only occur at exact sample boundaries, a total of four templates are created for each cluster mean: the mean waveform is upsampled by factor of four. Then, templates are created by downsampling shifted versions at four subsample shifts (0, 0.25, 0.5, 0.75 samples). Whenever the spike waveform of a unit is needed, one of the four shifted templates is used at random.

Figure 3.9 (B) shows how simulated spikes and overlaps are created. For each instance, a noise period is chosen at random. From the selected

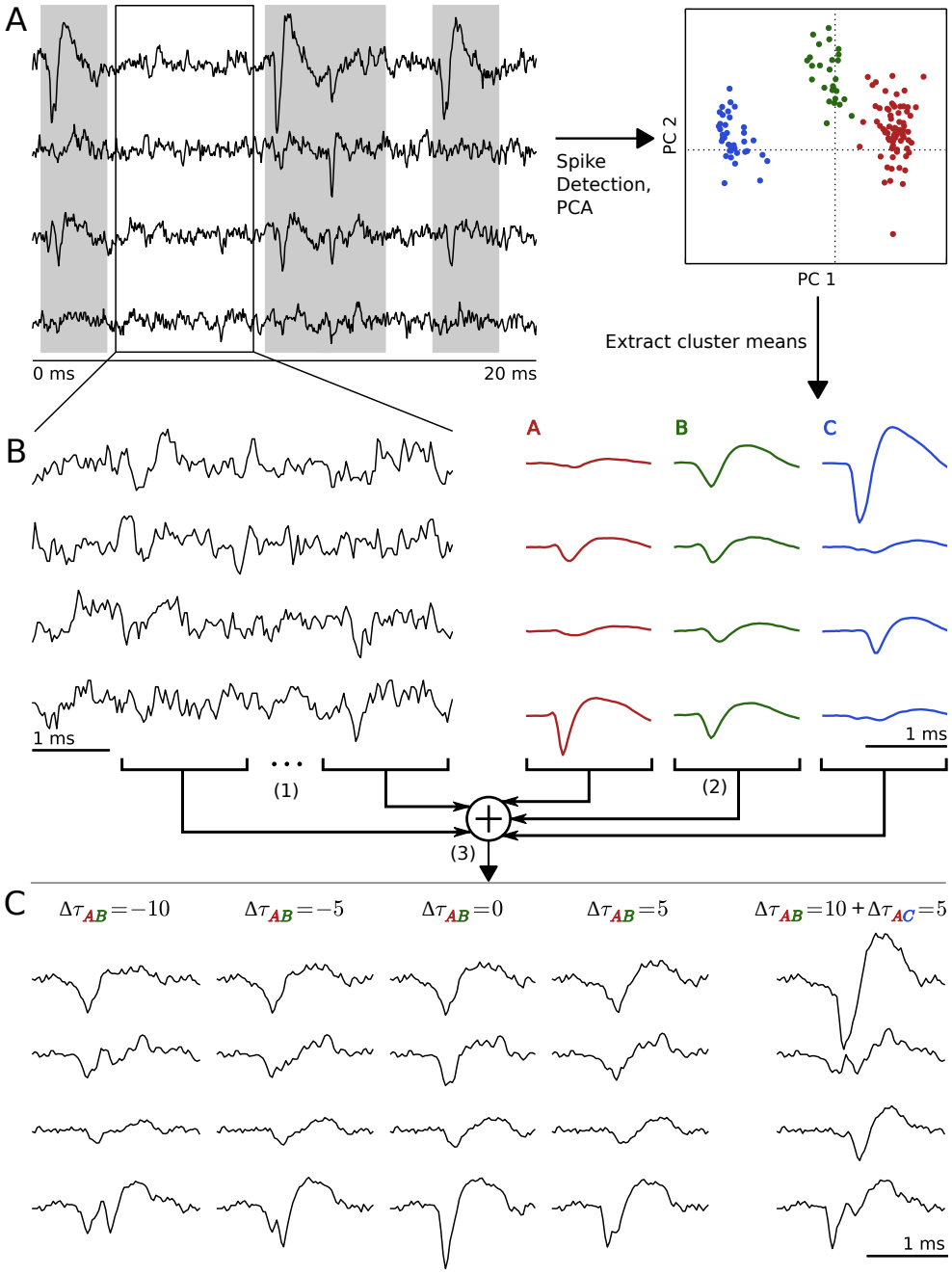

Figure 3.9: Sketch of the overlap simulation process. **(A)** Recorded data is divided into spikes and noise. **(B)** Noise and multiple spikes are combined. **(C)** Examples of resulting overlaps at various offsets. The $\Delta\tau$ values are given in samples for a sampling rate of 32 kHz.

period, a short patch of data long enough to contain the spikes is randomly selected (1). Next, spike waveforms of the desired number of units are selected, again randomly (2). Finally, the noise and spike waveforms at the desired offsets are summed (3), corresponding to the generative model that underlies most spike sorting algorithms (described in section 3.4). Figure 3.9 (C) shows examples of four overlaps of two spikes at different offset and one overlap of three spikes.

To create a sufficiently large set of simulated spikes and overlaps, a total of 40 recordings from different tetrodes were used as base data for the simulations. The spike sorting algorithm found between two and nine units per tetrode (4.92 ± 2.58). For each recording, 10000 single spikes were simulated. The maximum number of spikes in an overlap was constrained by the available number of units for that recording, with an upper limit of five. In overlaps of two spikes, all possible offsets between the two templates in the range -48 samples to 48 samples (-1.5 ms to 1.5 ms) were simulated 1000 times. For higher order overlaps, the number of possible combinations becomes too large to cover exhaustively, so 100000 random combinations of templates and their respective offsets were simulated for each overlap order above two.

3.4.7 Resolution of overlaps in simulated data

To evaluate the performance of BOTM on overlapping spikes, the simulated data was sorted using the hybrid BOTM algorithm described in subsection 3.4.5. The cluster means used in the simulation were set as templates, and the noise covariance matrix $\mathbf{C}$ was estimated from the noise periods. The simulated data deviates from the generative model in two respects: First, the simulated spikes are based on waveforms with subsample offsets, so the spikes shapes are not identical to the templates used by BOTM. Second, the noise is not artificially generated using $\mathbf{C}$, and it is unlikely that the estimated covariance perfectly describes the recorded noise that is used instead. Both deviations are desirable: they can increase the number of errors but the same difficulties arise when sorting real data.

For each instance, an error was defined as any false positive, false negative, or misclassification. For example, in a simulated overlap of order five, the algorithm had to correctly detect and identify all five spikes and report no further spikes, otherwise the instance was considered incorrect. The error rates reported here refer to the percentage of correct instances compared to all instances for a given condition.

Figure 3.10 shows the error rate over all simulations and template combinations for overlaps of two waveforms as a function of the offset $\Delta\tau$ between the two spikes. The average error is below 2% and only slightly depends on $\Delta\tau$. The reduced errors for small offsets result from

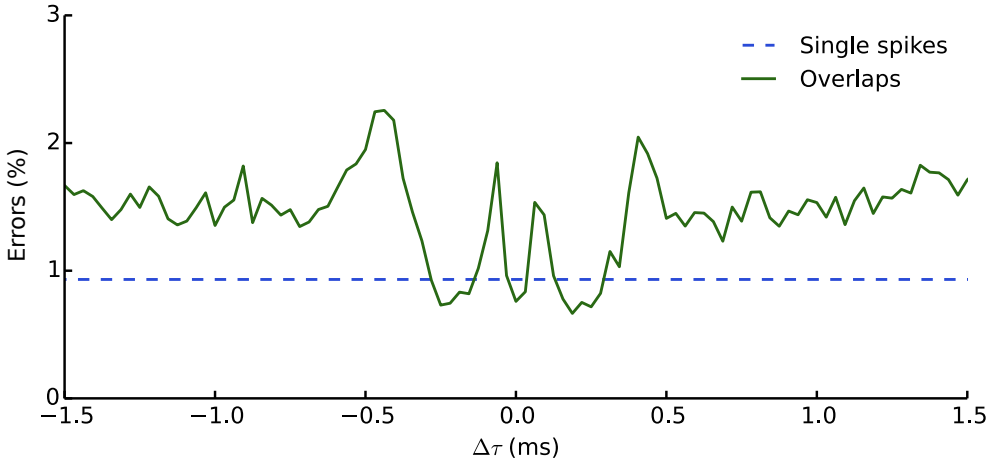

Figure 3.10: Error percentage depending on overlap offset.

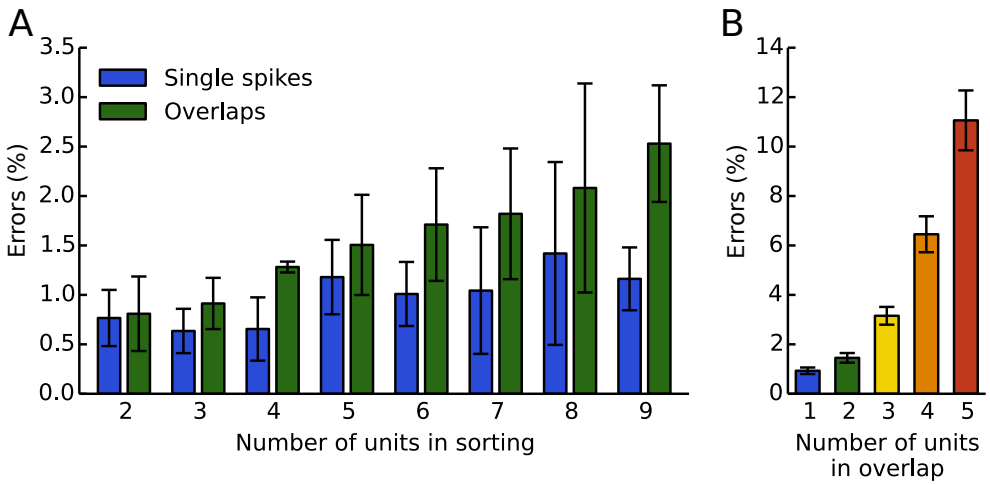

Figure 3.11: Sorting performance depending on number of units. Whiskers indicate SEM.

the hybrid BOTM algorithm. For some $\Delta\tau$, the performance for overlaps even exceeds the performance for single spikes. The two error peaks at -0.1 ms and 0.1 ms are caused by two particular base recordings that each contain a problematic triple of templates: the waveform of an overlap of two templates at offsets -0.1 ms or 0.1 ms is similar to a third template.

The number of units in the base data can also influence sorting performance: more possible templates carry a higher risk of misclassification. Figure 3.11 (A) illustrates this effect. For tetrodes where only two or three units were found the error for both single spikes and overlaps are similar and below 1 %. The error rate for overlaps rises with the number of available templates while the error rate for single spikes only shows a slight upward trend. Overlaps are more affected by the rising number of templates because they offer more opportunities for misclassification:

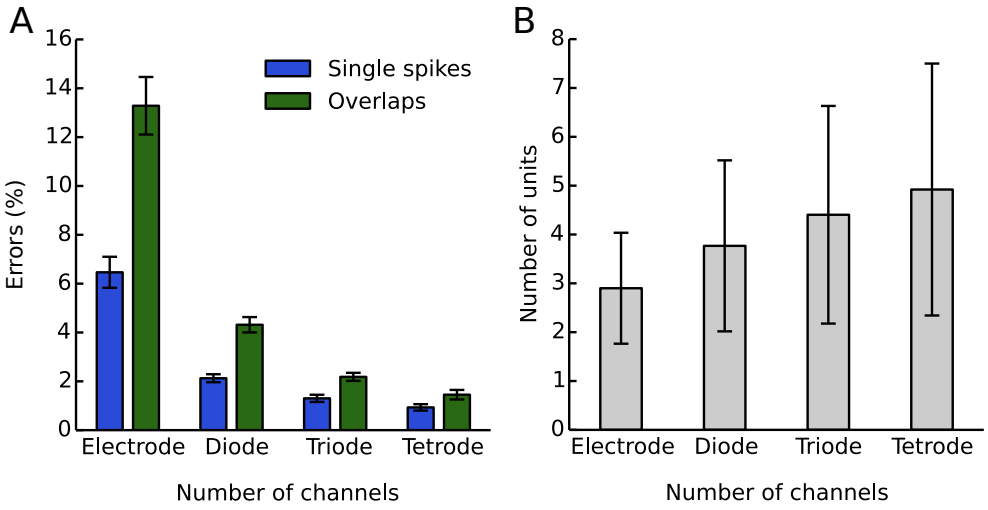

FIGURE 3.12: Influence of number of channels. **(A)** Sorting performance depends on the number of recording channels. Whiskers indicate SEM. **(B)** Number of detectable units increases with more recording channels. Whiskers indicate standard deviation.

each spike in the overlap could be confused with another template. In addition, the probability of an overlap waveform being similar in shape to a single template rises with the number of units. In this case, the whole overlap could be classified as a single spike.

Figure 3.11 (B) shows how the sorting performance depends on the overlap order. For single spikes, the average error percentage is 0.93 %. It rises to 1.45 % for overlaps with two spikes and continues to increase sharply for higher order overlaps. However, even for overlaps of five spikes, almost 90 % of all instances are correct, meaning all five spikes are attributed correctly. Overlaps of four or more spikes on a single tetrode are very rare in the macaque dataset.

To assess the impact of multiple recording channels on sorting performance, we removed electrodes from our simulated spikes and overlaps. For each recording, all permutations of channels were simulated. Because most templates have an unequal amplitude distribution over channels, they could become undetectable in certain channel permutations. In order to not artificially increase the error rate in these cases, templates that became undetectable because of a low SNR were discarded. The definition of SNR employed here is based on the Mahalanobis distance of a template $\boldsymbol{\xi}$ to the zero vector. It was introduced in (Franke, 2011):

$$\mathrm{SNR} = \sqrt{\frac{\boldsymbol{\xi}^T \mathbf{C}^{-1} \boldsymbol{\xi}}{T_f \cdot N_C}}. \tag{3.25}$$

Using this definition, units with a SNR < 0.8 on all remaining channels were not included in the simulation or the set of templates available for

BOTM. In a recording using only the respective number of channels, these units could not have been detected. Figure 3.12 (A) shows how additional channels improve performance for both single spikes and overlaps. As Figure 3.12 (B) illustrates, the number of units increases with more channels. Therefore, multielectrodes enable recordings from more neurons than single electrodes while also significantly reducing classification errors, including error rates for overlapping spikes.

3.4.8 Prevalence and resolution of overlapping spikes for in vivo Data

The results presented so far in this chapter show that BOTM can resolve overlapping spikes in tetrode recordings with high accuracy. The algorithm was also used to sort the macaque dataset. Using the recordings from the same 40 tetrodes that were used as basis for the simulations in subsection 3.4.6, the percentage of all detected spikes that participate in an overlap was assessed. Figure 3.13 shows how this percentage depends on the cumulative firing rate (blue dots). It ranges from below 1 % to more than 50 %, with a mean of 22 % (± 15 %).

In order to assess whether these percentages are unusually high, the firing rates of individual units in each recording were used to simulate independent Poisson processes with constant firing rates (green dots). While the rate of overlaps for these simulations also increases with the total firing rate, it remains mostly below the rate in the empirical recordings. This indicates that the spike trains in the recordings are not completely independent: multiple units tend to fire simultaneously or their firing rates covary. Because the data was recorded during a behavioral experiment, irregular and covarying firing rates are common. Therefore, another simple simulation was created, again with independent Poisson spike trains and the same average firing rate, but modulated by the same gamma oscillation (40 Hz). The amplitude of the oscillation ranged from 0.2 to 1.8 times the original firing rate of the respective unit (red dots). This simulation leads to similar overlap percentages as observed in most of the recordings.

However, some recordings have much higher overlap percentages. The most extreme example is marked in Figure 3.13 as (A) and shown in more detail in Figure 3.14. As visible from the spike trains depicted in Figure 3.14 (A), all four units on this tetrode tend to fire at similar times, while there are also long periods of silence during the trials. (B) shows a short patch of data with superimposed templates of detected spikes during a period of activity. Multiple overlaps are clearly visible during the short time window. In (C), one of these overlaps is depicted in more detail. On the left, the original data is shown in black together with the waveform of the complete overlap of three spikes in magenta. The three

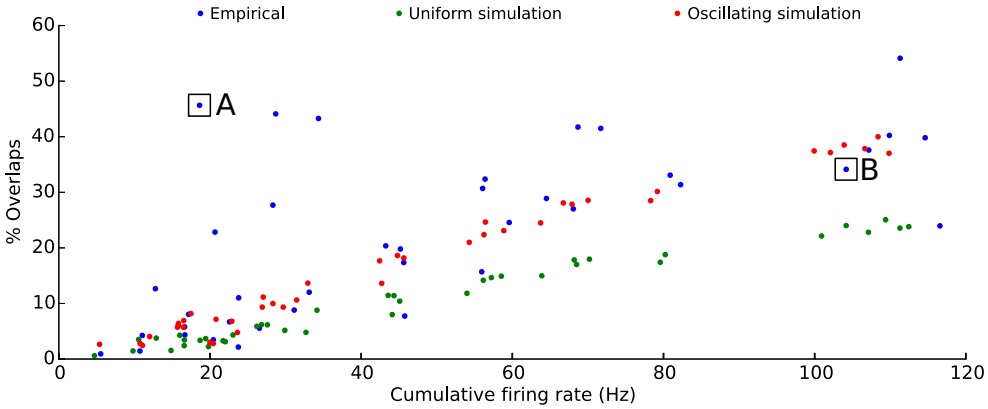

FIGURE 3.13: The percentage of spikes participating in overlaps depends on the total firing rate of all units in a given recording. The percentage of overlaps in in vivo recordings is compared to simulations of independent Poisson spike trains with a constant firing rates and co-modulated firing rates. The rectangles refer to tetrode recordings that are depicted in more detail in Figure 3.14 for (A) and Figure 3.15 for (B).

templates and the corresponding data when the other templates have been subtracted can be seen in the middle. It shows how all individual templates fit the respective residuals well even for higher order overlaps. The gray traces on the right show the residual data after all templates have been subtracted. Figure 3.15 shows another example from a tetrode with more units and a higher, more balanced cumulative firing rate.

3.5 Summary

This chapter presented the complete spike sorting pipeline that has been used on the working memory experiment data. The BOTM algorithm and its improvements were described in detail. The evaluations using simulated overlaps suggest that the algorithm can resolve temporally overlapping spikes with high precision, in some cases even exceeding the performance for single spikes. This suggests that the separation of spike sorting into two distinct phases is useful when considering temporal overlaps: an initial clustering step where sorting quality (in particular of overlaps) can be lower, with a following template matching that produces high quality results and can run in real-time. But even for studies in which spike synchrony is not a particular concern, correctly resolving overlaps should be an important issue, as the analysis of the prevalence of overlaps in the macaque dataset shows: if overlaps are ignored or classified incorrectly, a large percentage of spikes will be affected.

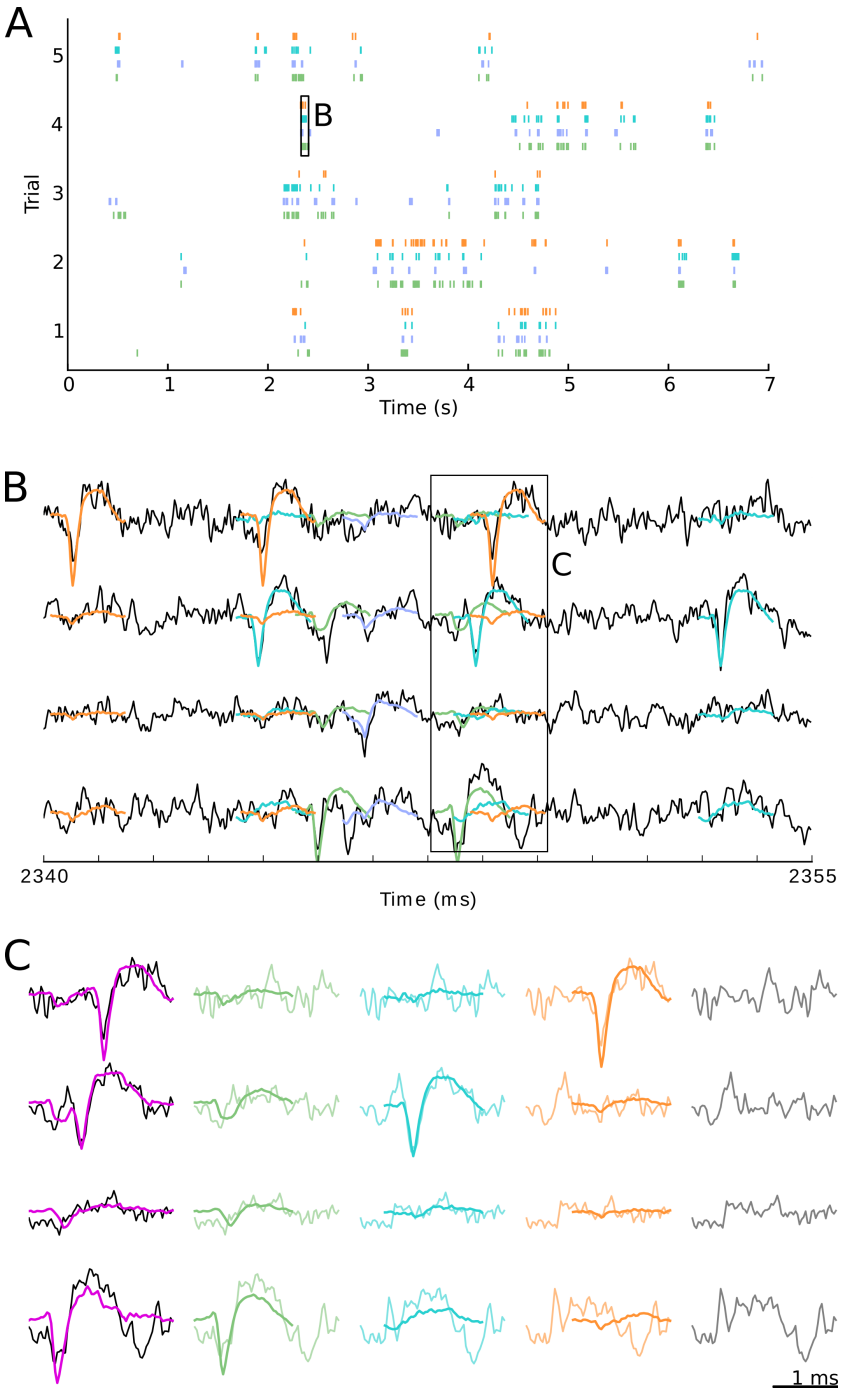

Figure 3.14: Example spike times and waveforms from a recording in which the four identified units fire in collective bursts and are silent most of the time which leads to a high prevalence of overlapping spikes. **(A)** Spike trains for five trials. **(B)** A small period of data with intense spiking activity. **(C)** A triple overlap.

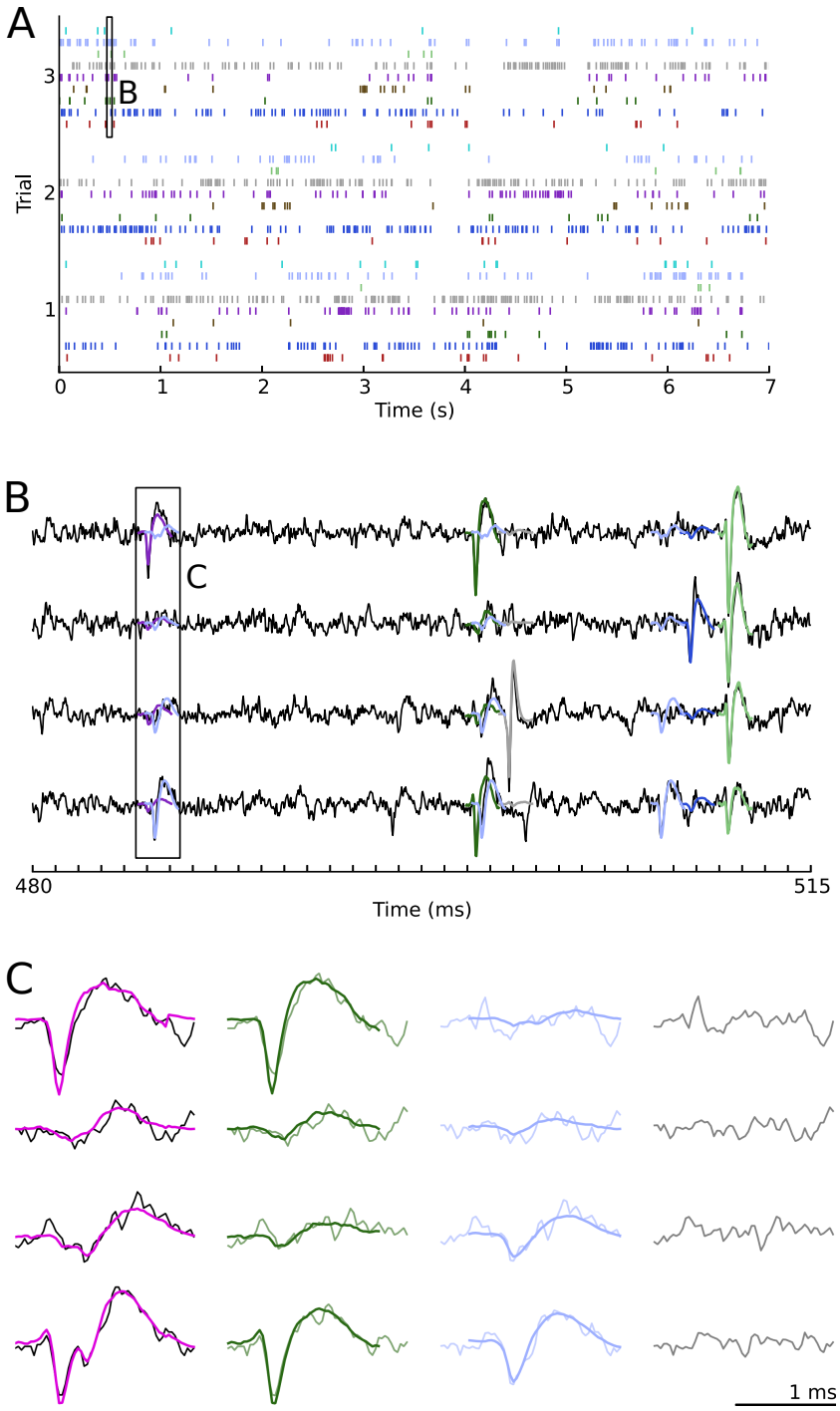

FIGURE 3.15: Example spike times and waveforms from a recording with nine identified units and a high cumulative firing rate. **(A)** Spike trains for three trials. **(B)** A typical period of data. **(C)** An overlap with two spikes.

Working memory analyses 4

With the technical infrastructure and the signal processing algorithm completed, the data from the working memory experiment described in section 1.5 could be analyzed in detail. Recorded data from 127 tetrodes in 17 recording session was spike sorted using the BOTM algorithm. Each recording session contains up to 5 blocks of trials that differ in their experimental protocol. For the analyses presented here, trial blocks were selected according to the following rules: Use the earliest block with a maximum memory load of at least three items (for most blocks, the maximum memory load was four items). Use only blocks with pictorial stimuli, ignore blocks with other stimuli. The rules were selected to maximize the similarity of the analyzed blocks and minimize the effect of electrode and tissue drift on spike sorting results. All analyses have been carried out using Spyke Viewer and are available as plugins.

The guiding question for this chapter is: How much information about stimuli and experimental conditions is contained in the recorded neural activity and how is it encoded? This includes inquiries on how many and what kind of neurons take part in coding, what the most effective decoder is, and how accurate the decoding is at different times during the trial.

4.1 Neuron type identification

There are two major functional classes of neurons in the PFC and other cortical areas: excitatory neurons (most commonly pyramidal cells) and inhibitory interneurons (Markram et al., 2004; Wilson et al., 1994). Pyramidal cells form long distance connections and can connect to different parts of the brain, while the axons of interneurons remain within restricted areas. Because these neuron types perform different physiological

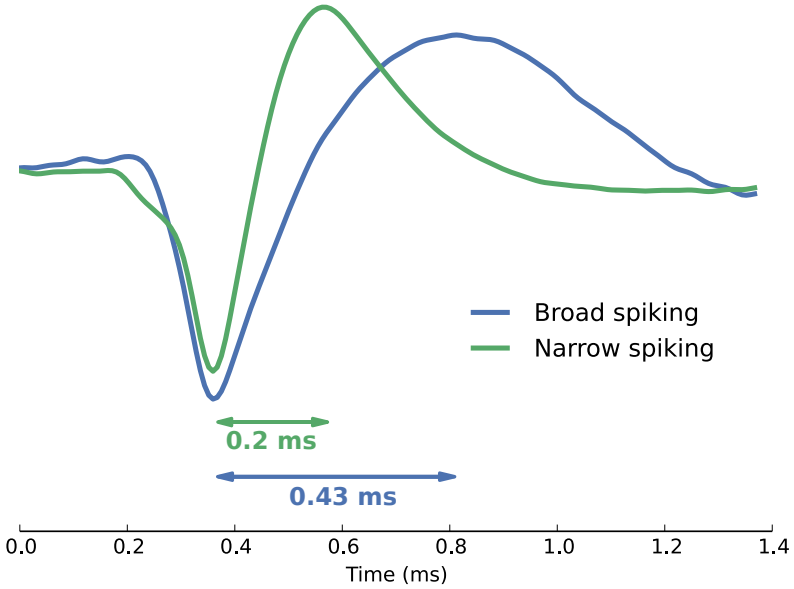

Figure 4.1: Mean waveform and duration for a typical narrow spiking (putative inhibitory) and a typical broad spiking (putative excitatory) neuron.

functions, it is likely that their contributions to the neural code differ as well. In extracellular recordings, it is generally not known to which class a recorded unit belongs.

Intracellular recording studies have shown that the shape of the generated spike waveform depends on the cell type: the action potentials of interneurons are narrower than those of pyramidal cells (McCormick and Connors, 1985). However, this categorization is not completely reliable: (Vigneswaran et al., 2011) found pyramidal neurons in macaque motor cortex that fire narrow spikes. Because the duration of the extracellularly recorded action potential is correlated with the duration of the intracellularly recorded waveform (Gold et al., 2006), putative interneurons and pyramidal cells can be inferred using the width of the extracellularly recorded waveform. Figure 4.1 illustrates typical waveform shapes for interneurons and pyramidal cells in extracellular recordings.

Previous studies have found two distinct populations with different functional properties in various areas of the macaque brain. (Chen et al., 2008) report different modulation by task difficulty in visual area V1, (Mitchell et al., 2007) found specific neural correlates for attention in visual area V4. Both (Diester and Nieder, 2008) and (Hussar and Pasternak, 2009) studied the PFC and found differences in numerical categorization and direction selectivity, respectively. All of these studies used the time from the trough to the peak of the waveform as measure

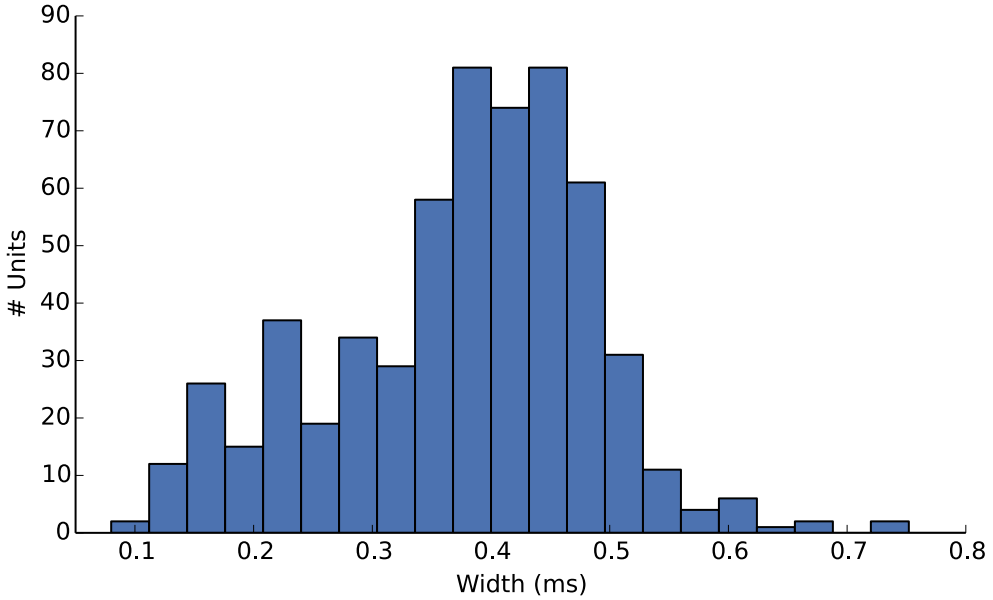

FIGURE 4.2: Histogram of spike waveform widths of all recorded units.

for the width and found a distinctly bimodal distribution of widths over all recorded units.

When calculating the spike waveform width for units recorded in the working memory experiment, two additional aspects had to be considered. First, each unit was recorded on four channels, producing a different waveform on each with potentially varying widths. Second, many units produced waveforms that slowly changed over the course of a recording session. For both cases, a way to decide which waveform and width to consider is required. Using the waveform with the highest *Signal to Noise Ratio* (SNR) ensures that the results are as little affected by noise as possible. Therefore, for every unit the width was calculated in the trial and on the channel with the maximum SNR.

All analyses are based on the spike templates used in the BOTM spike sorting algorithm (section 3.4). The templates are calculated as the mean of the 300 most recently detected spikes, decreasing noise but remaining responsive to slowly changing spike waveforms.

A histogram of the spike waveform widths of all recorded units is shown in Figure 4.2. While the distribution does have a left tail, it is not strongly bimodal and there is no hint in the data where a separation between the two neuron classes could be made. Other, additional features of the spike waveform, such as a symmetry index (difference between maxima in the normalized waveform to the left and to the right of the trough) or projections onto principal components did not yield two distinct classes of waveforms, either.

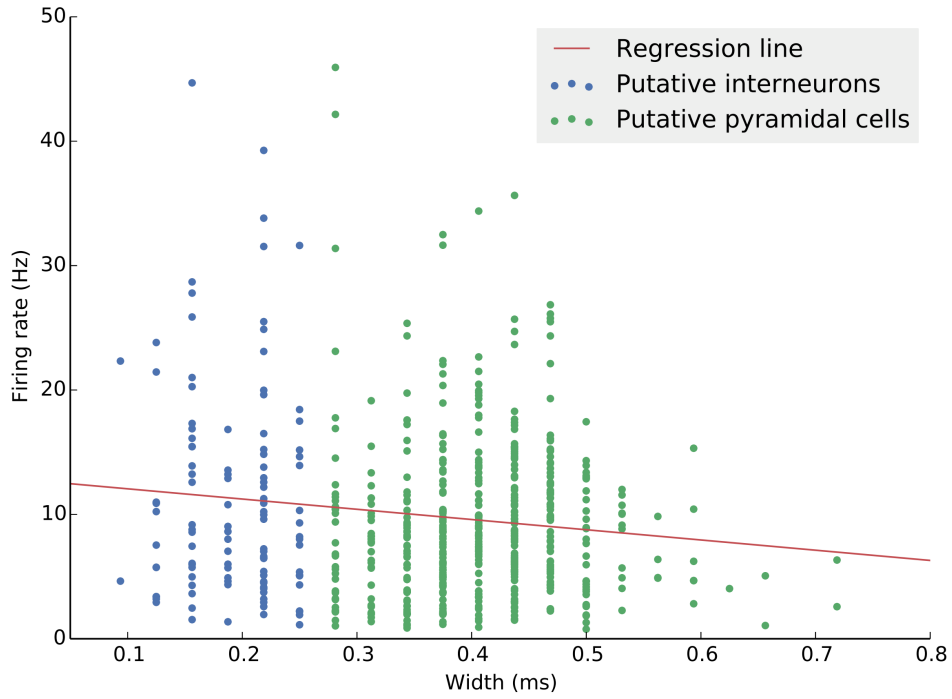

Figure 4.3: Spike waveform width and firing rate.

Inhibitory interneurons generally have a higher firing rate than excitatory neurons (Connors and Gutnick, 1990), making firing rate another possible indicator for neuron class. Figure 4.3 shows the relationship between firing rate and spike waveform width. A significant ($p < 0.05$) but small trend is visible for lower firing rates in units with broader spikes.

By assigning each unit to a class using a threshold of 0.25 ms (based on the values in the PFC literature: (Hussar and Pasternak, 2009) used 0.2 ms, (Diester and Nieder, 2008) clustered the waveforms and found a width threshold around 0.3 ms), it becomes clear that the firing rate does not allow for a clearer separation. The difference between the two groups is small (11.7 Hz average firing rate for putative interneurons, 9.4 Hz for putative pyramidal cells) and although the difference is significant ($p < 0.05$, Mann-Whitney U test), it is not enough to serve as an additional criterion in allowing a clear separation of two cell groups.

Because the data did not allow a clear separation of units into putative inhibitory and excitatory neurons, this line of inquiry was not developed further. The following analyses do not make use of neuron type information.

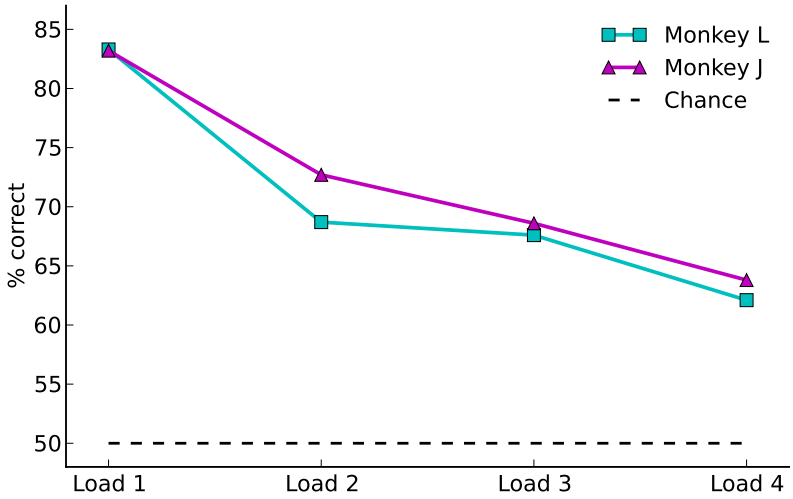

FIGURE 4.4: Percentage of correct responses: Behavioral performance decreases with rising memory load.

4.2 Experimental conditions

Many of the following analyses deal with differences in neural activity for experimental conditions that can vary from trial to trial. The most obvious of these conditions is the identity of the presented stimuli: up to four different stimuli during the sample period and another stimulus during the test period. Therefore, one of the investigated conditions is the test stimulus identity. For the sample stimuli, choosing useful conditions is more complex. For example, the third sample stimulus occurs only in trials with a memory load of three or four and the presentation periods starts and ends at different times depending on the memory load (600 ms to 900 ms after the onset of the sample stimulus presentation for load three, 445 ms to 675 ms for load four). Here, the first and the last sample stimulus identities are used as conditions. Both exist in every trial, the first sample always starts at the same time, the last sample always ends at the same time. For trials with a memory load of one item, the stimuli for both conditions are identical, when ignoring load one trials they are always distinct. Each stimulus is chosen from a set of 20 possible stimuli, so each condition in the stimuli category has 20 classes.

The load condition defines how many sample stimuli were presented in a trial. In most recording sessions, one to four sample stimuli presented so this category has four classes. However, not all recording sessions included trials with loads up to four. Therefore, the load condition was usually used with two classes: a single presented stimulus as one class and higher loads as the other class. The performance of the monkeys decreased with higher loads (see Figure 4.4).

Another condition is termed "match": it distinguishes whether the test stimulus matches one of the sample stimuli. It can also be defined as the button the monkey had to press, because the correct button depends on whether the stimuli match. While the other conditions depend directly on external events, the match condition has an internal component: the monkey has to compare the presented test stimulus to the memorized sample stimuli. In contrast to the previous conditions, analyses on the match condition only use rewarded trials. Finally, the reward condition divides trials into a class with rewarded trials and a class with trials in which the monkey did not receive a reward.

For all analyses presented in this chapter, the data was aligned to the onset of the first sample stimulus. When referring to a time during the trial, it is always relative to this onset, e.g. trial time -500 ms would be half a second before the start of the sample presentation period.

4.3 Single units

A classical approach in neurophysiology is to relate the firing rate of a neuron to external conditions, such as presented stimuli (Hubel and Wiesel, 1959). The firing rate can be determined by counting spikes in a defined period. But many neurons, especially in higher areas such as the PFC, exhibit fluctuating spontaneous activity independent of experimental conditions. How to decide whether a change in firing rate is not an unrelated fluctuation? Typically, statistical hypothesis testing is used to determine whether firing rate changes are significant.

An important decision for all rate-based methods is on what periods to calculate the firing rate. Using the whole trial offers the most stable estimation but ignores changes that occur during the trial. This is not a viable option because the working memory experiment trials contain several distinct periods (baseline, sample stimulus presentation, delay, and test stimulus presentation). Binning using short estimation periods offers a better temporal resolution but suffers from increased noise that can overshadow rate modulations, especially for units with low overall firing rates. Here, 200 ms windows are used as a compromise. By using a sliding window with a step size of 50 ms, a high temporal resolution and locality is possible. Larger window sizes yield slightly better results for some analyses but remove information on when during the trial an effect occurs. The analyses in this section are based on 629 recorded units.

4.3.1 Rate modulation during the trial

The first question to ask is whether the recorded units react to the experimental conditions at all, i.e. whether their firing rates change over

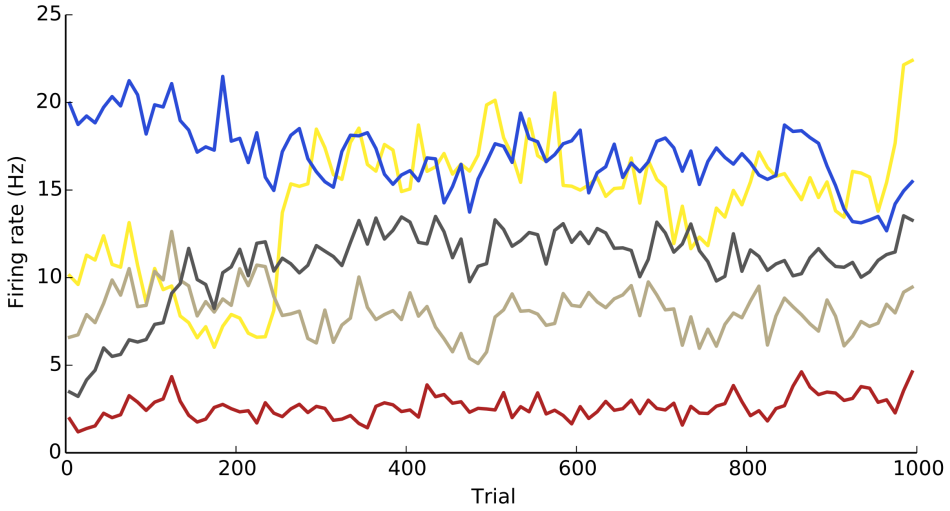

FIGURE 4.5: Firing rates of five units over the course of a recording session. Many units maintain a steady firing rate, but some change considerably.

the course of a trial. For this analysis, a baseline activity is defined as the firing rate before any stimulus is presented in a trial. The firing rate throughout the trial is then compared to the baseline. Because the firing rate of many neurons changes slowly over course of a recording session (see Figure 4.5 for a few examples), the baseline is recalculated for each trial. Using a paired t-test that pairs the baseline activity of every trial with the firing rate calculated in each bin during the same trial, the slowly changing firing rates do not affect the results.

The null hypothesis for the test is that the firing rate remains the same during the baseline and the respective bin. Figure 4.6 shows the results: for each bin, the fraction of units with a firing rate that significantly deviates from the baseline activity. Two significance levels are depicted to show that the results remain qualitatively the same, in the following, plots only include the $p < 0.05$ level to improve clarity. During the time before the trial starts, the number of significantly modulated units stays at the chance level. With the start of the sample stimulus presentation, the number increases quickly before dropping off during the middle of the delay period and increasing again toward the end of the trial. At all times after the baseline period, the number of units with significantly modulated firing rates stays far above chance level.

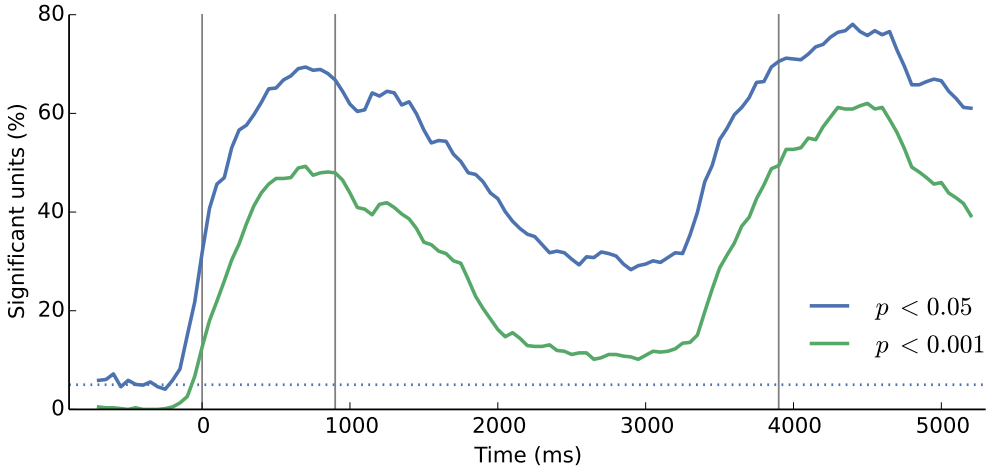

Figure 4.6: Percentage of units with a significantly modulated firing rate compared to baseline activity before the start of a trial. The dotted line marks the expected value under the null hypothesis for $p < 0.05$. The vertical lines mark the start of the sample, delay, and test periods, respectively.

4.3.2 Rate modulation by experimental conditions

Now that it is established that many neurons react to what happens during a trial, how many neurons are receptive to specific experimental conditions at what time during the trial? Again, this question can be answered using statistical tests with the null hypothesis that the firing rate does not change depending the investigated experimental condition.

As before, slow rate changes have to be taken into account: a regular t-test or analysis of variance could confound these changes with the rate modulations caused by changing experimental conditions. Using tests based on local groups of trials avoids this problem: by creating temporally local groups of trials that contain one trial for each class of the experimental condition, the effects of slow rate changes do not affect the comparisons. For most conditions, trials of different classes do not immediately follow one after another, so the grouping algorithm has a flexible maximum distance for finding groups that contain all classes: trials for the same group could be a maximum of 100 trials apart for the stimuli conditions, and 20 trials for all other conditions. The grouping procedure also results in class sizes that are equalized to the number of samples in the smallest class. Conditions with two classes were analyzed using the Wilcoxon signed-rank test. For conditions with more than two classes, the Friedman test was used.

To check whether the grouping was effective in eliminating the influence of slowly changing firing rates on the results, the tests during the baseline period can be compared to the chance level. For the match

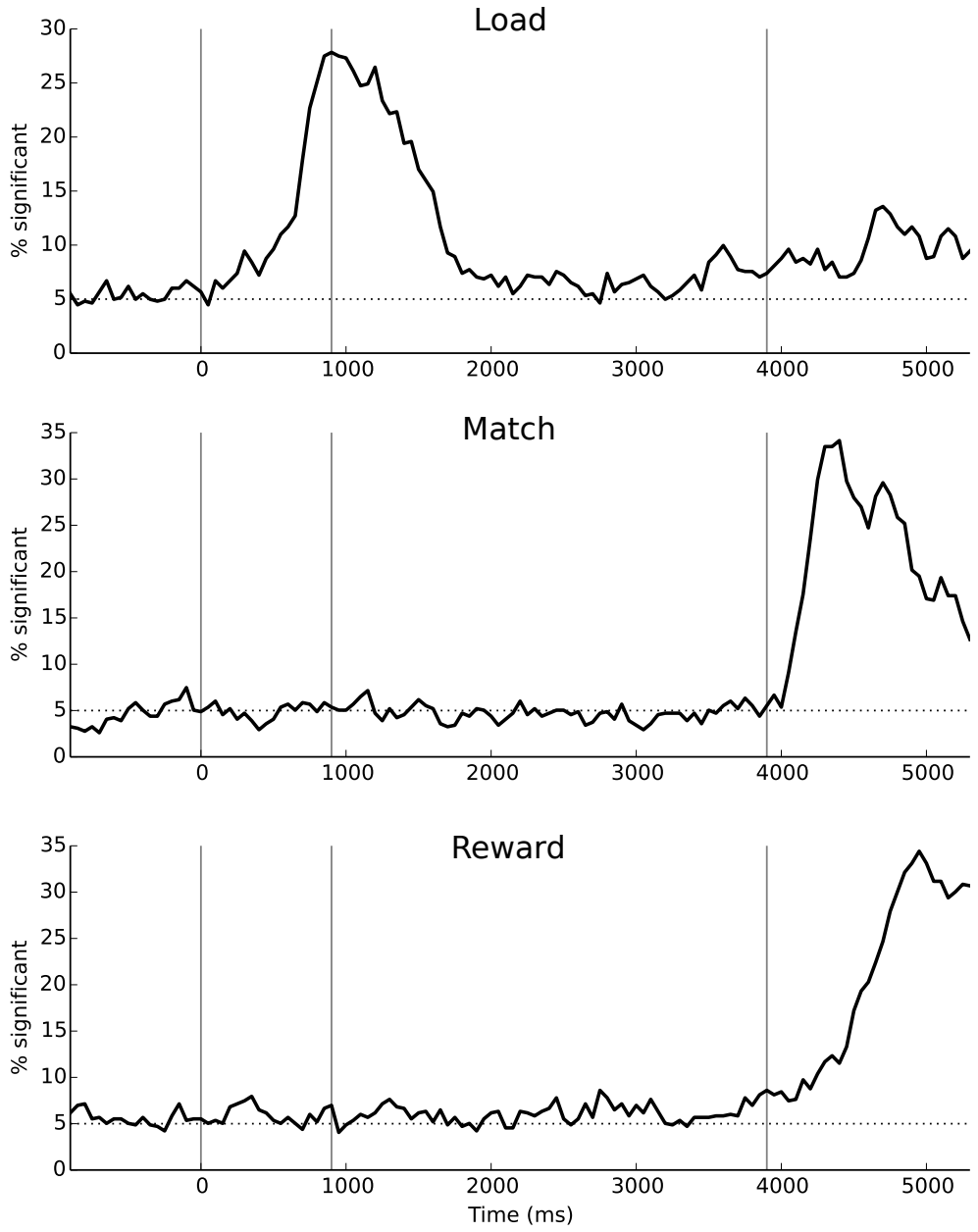

FIGURE 4.7: Percentage of units with significantly ($p < 0.05$) modulated firing rates for experimental conditions with two classes. The dotted line indicates the chance level.

condition, there is a much longer period that can be used to verify that no surplus information is being detected by the tests: until the test stimulus presentation, there can be no information about whether the trial is a match because matching trials are distributed randomly.

The results for the experimental conditions not related to stimulus presentations are shown in Figure 4.7. A large number of units react with significant rate modulation to all conditions, but at different times during

the trial. The percentage for the load condition rises during the sample stimulus presentation and returns almost to chance level in the early delay period. This indicates that the information about the memory load encoded in the firing rates of single units does not persist throughout the delay period. The small rise during the test stimulus presentation suggests that the processing during that period (e.g. comparison of test and sample stimuli) is influenced by the number of sample stimuli.

For the match condition, the number of units with significantly modulated firing rates remains at chance level until the test stimulus presentation, as expected. Then, about a third of the recorded units are sensitive to whether the test stimulus matches one of the memorized sample stimuli, indicating that these units take part in a comparison between the currently visible and memorized items. The picture is similar for the reward condition. However, the curve rises later in the trial because the monkey receives the reward only after a correct response. In addition, the number of significant units is slightly above chance level during the whole trial. This indicates that a few units take part in encoding an internal state (e.g. attention) that influences the success of the monkey.

Figure 4.8 shows the results for all stimulus conditions. As for the other conditions, the number of significant units rises above chance level for all conditions, although the percentage is much smaller for the stimulus conditions. The numbers are comparable for both sample stimuli and the test stimulus. As for the load condition, the number of significant units does not remain above chance during the whole delay period. This indicates that the recorded units do not encode the stimulus identity with their firing rates over the whole delay period. Since the monkeys achieved a high performance in the task, the information has to be stored in another way. This could be a different encoding mechanism or neurons in another area.

When looking at the results from each animal separately, some discrepancies appear. Examples for two conditions are shown in Figure 4.9. In general, the results are similar for both animals for the non-stimulus conditions. However, there is a clear difference for the stimulus conditions; the recordings from monkey J contain a much higher percentage of significant units. On the other hand, the data from monkey L has an overall better signal quality: a total of 496 single units were isolated for monkey L compared to only 115 for monkey J.

4.3.3 Selectivity for multiple conditions

The large percentages of units reacting with significant rate changes to all experimental conditions implies that some units are involved in coding for multiple conditions. But are units more likely to code for some condition pairs than for others? To answer this question, the previously determined

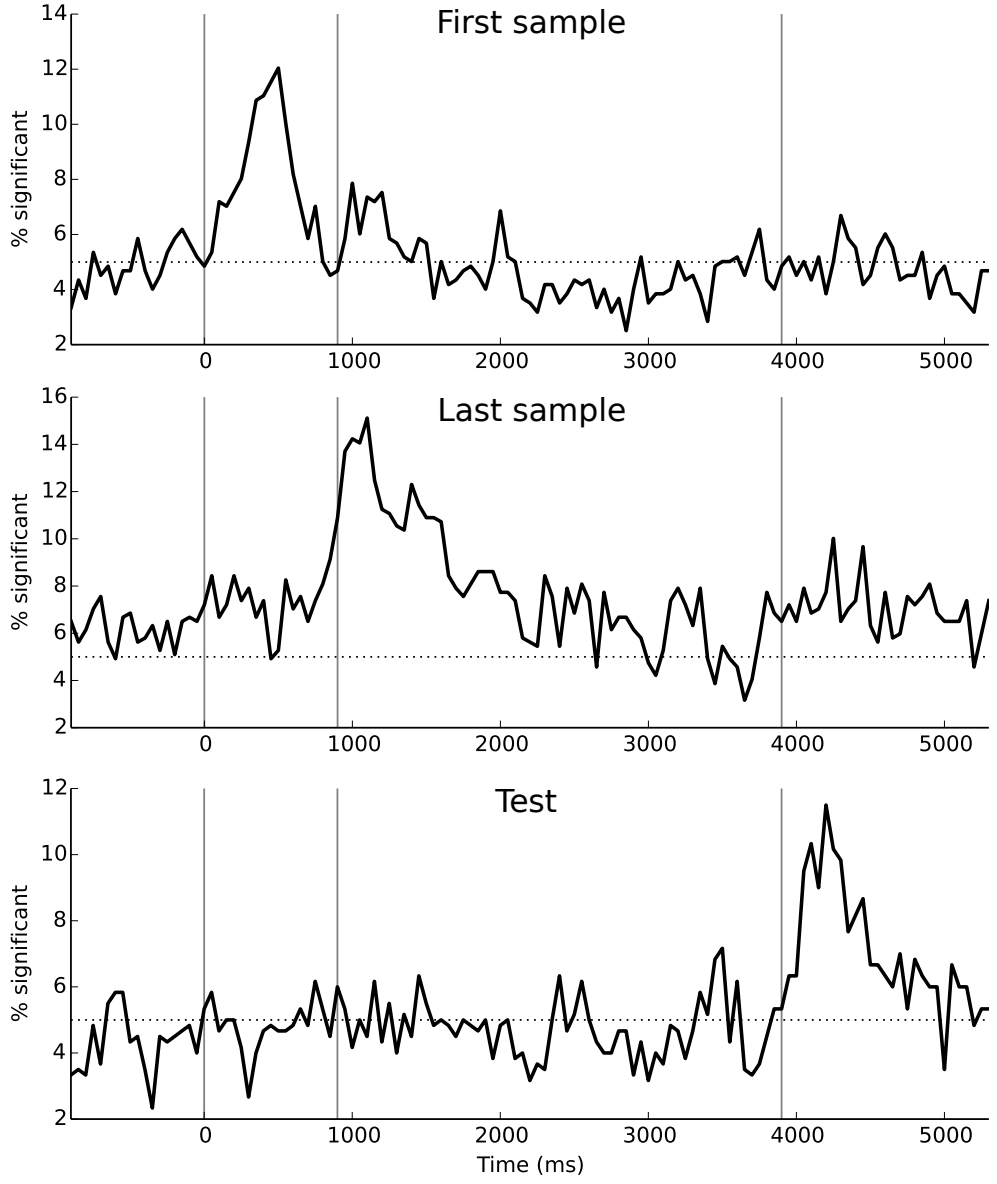

FIGURE 4.8: Percentage of units with significantly ($p < 0.05$) modulated firing rates for stimulus conditions.

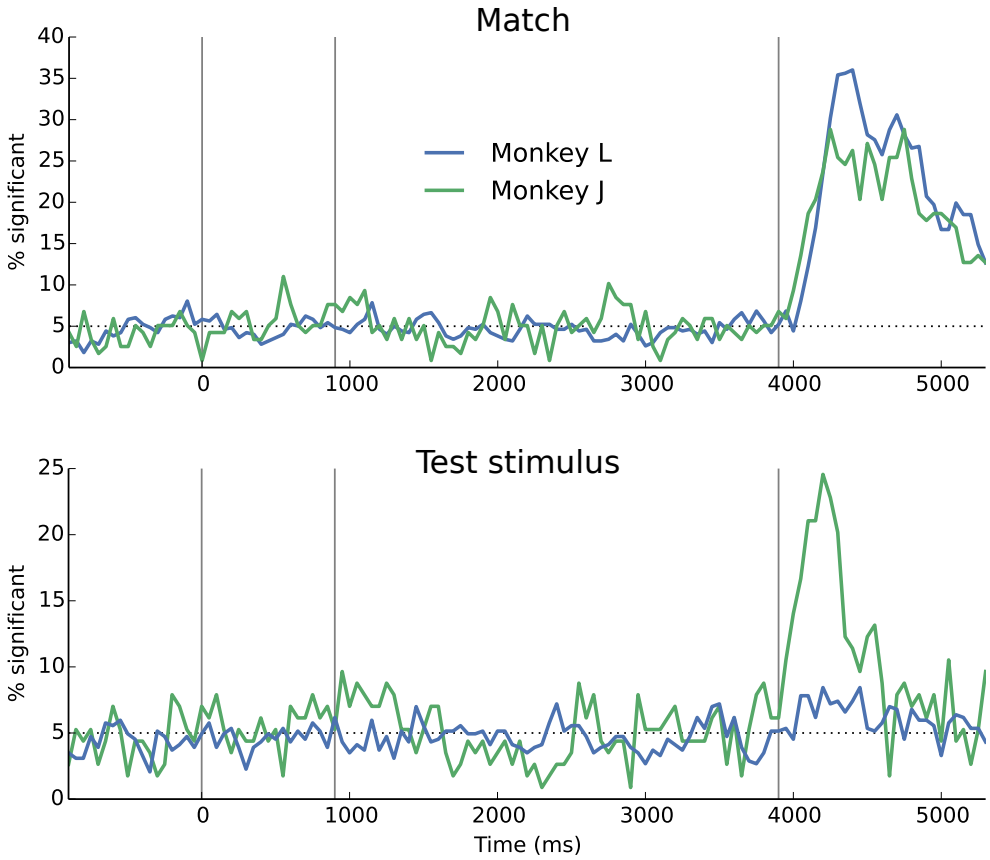

Figure 4.9: The difference in percentages of significant units between the two experimental animals varies between non-stimulus and stimulus conditions.

sets of significant units can be used. For two conditions at two given times during the trial, it has to be determined whether the number of units with significant rate modulations for both conditions (compared to just one or none) is significantly higher than expected by chance.

With k the observed number of units selective for both conditions, N the total number of units, M the number of units selective for the first condition and n the number of units selective for the second condition, this can be modeled using the hypergeometric distribution with the cumulative distribution function

$$H(k|N,M,n) := \sum_{y=0}^{k} \frac{\binom{M}{y}\binom{N-M}{n-k}}{\binom{N}{n}}. \tag{4.1}$$

The null hypothesis is that M and n are independent. The corresponding p value can be calculated from the survival function:

$$p = 1 - H(k|N;M;n). \tag{4.2}$$

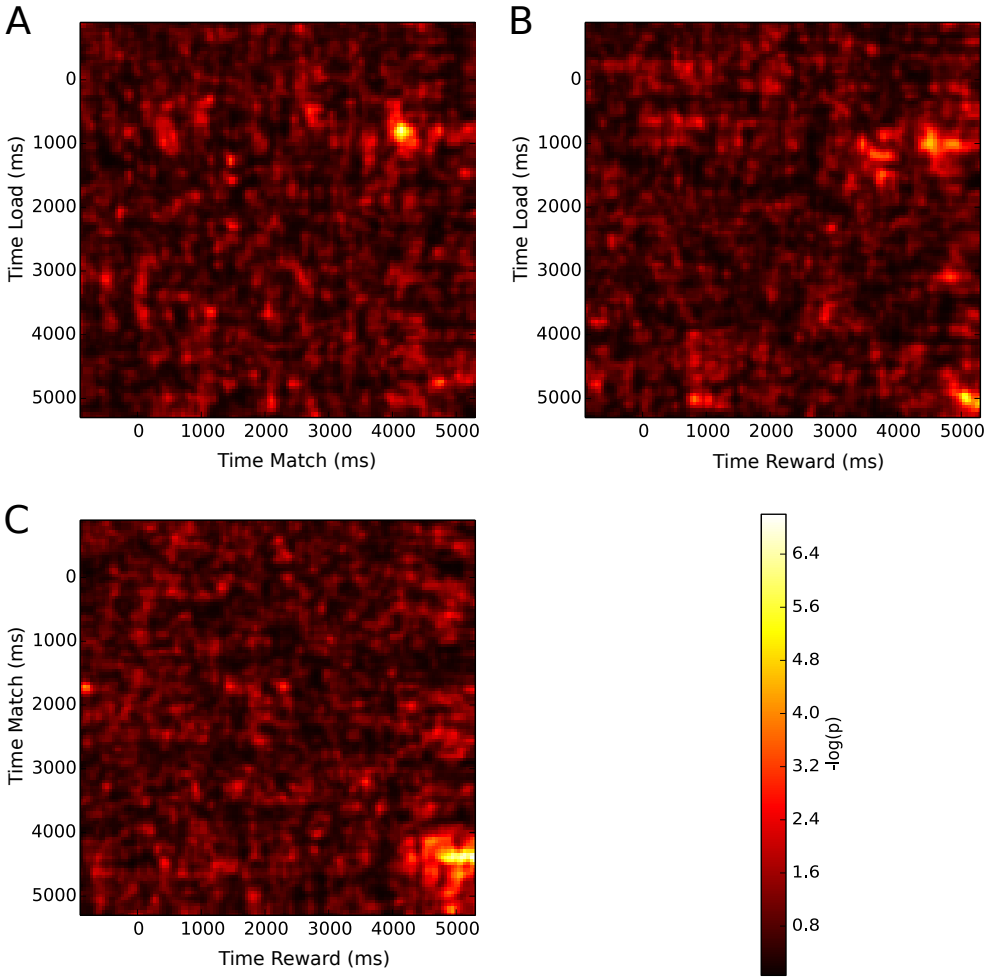

FIGURE 4.10: p values for shared subsets of significant units in non-stimulus conditions. The color scale is identical for all plots. **(A)** Load and match. **(B)** Load and reward. **(C)** Match and reward.

Determining the p values for all possible time shifts of two given conditions results in a two-dimensional p-map with trial time on both axes. Figure 4.10 shows the results for the two-class conditions. For all condition pairs, there are more units sensitive to both conditions than expected by chance during the time where each condition has its respective peak (as seen in Figure 4.7).

For load and match, there is a single sharp peak with the highest point at 750 ms for load and 4150 ms for match. At these coordinates, there are 132 significant units for load and 105 for for match, resulting in an expected 24 units that are significant for both conditions, compared to 50 observed units. In the load-reward pair, there are two distinct peaks: one for the big number of load-significant units in the early delay period, and another late in the trial during the smaller peak of load-significant units. The highest point for load-match is at 4950 ms and 5050 ms,

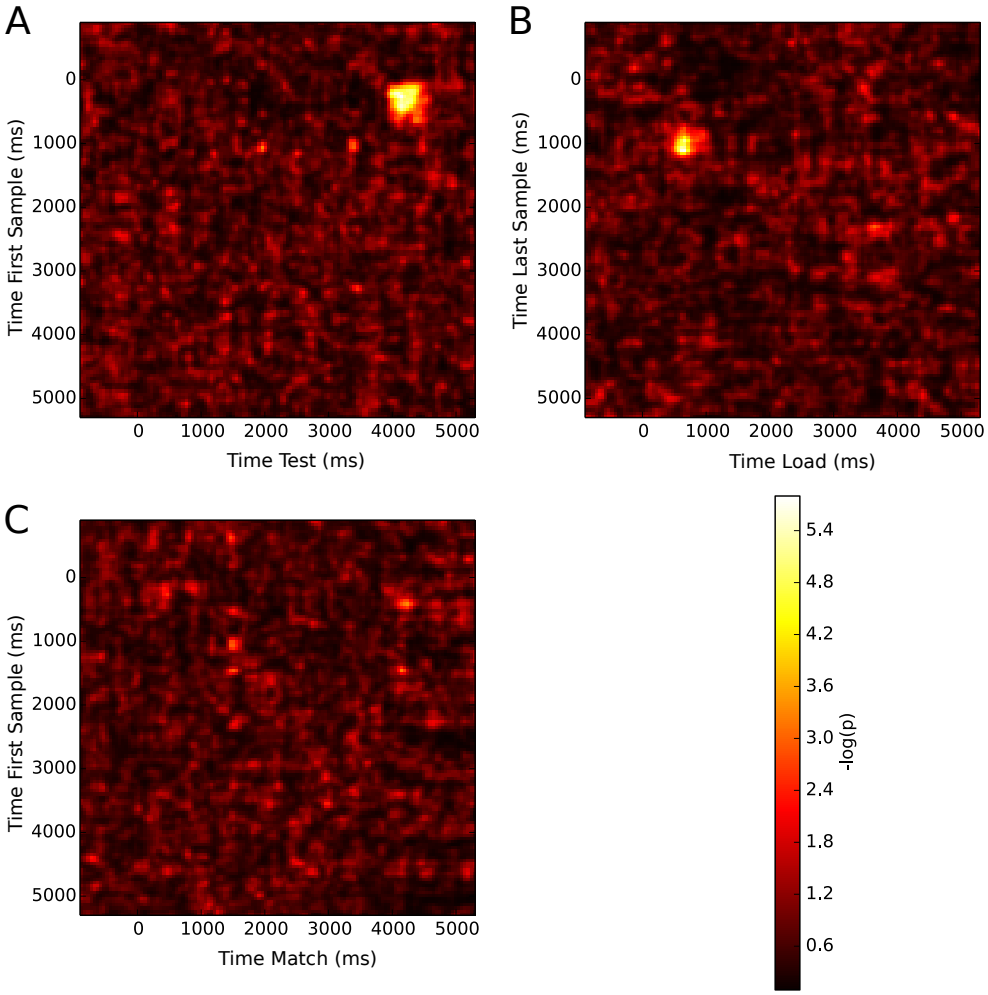

Figure 4.11: p values for shared subsets of significant units in condition pairs including stimulus conditions. The color scale is identical for all plots. **(A)** First sample stimulus and test stimulus. **(B)** Last sample stimulus and load. **(C)** First sample stimulus and match.

with 63 and 188 single significant units and 40 dual significant units compared to an expected 20. Finally, the area with low p values for match-reward is larger, encompassing all times where both conditions have a high number of significant units. It attains the maximum value at 4350 ms and 5150 ms, where 206 and 181 units are significant for each condition and 92 for both, while 61 would be expected.

Figure 4.11 shows examples for condition pairs involving stimulus conditions. For pairs of two stimulus conditions, such as the first sample and test stimulus pair shown in Figure 4.11 (A) (lowest p value at 250 ms and 4150 ms, 47 and 54 significant units with 17 doubly selective units opposed to 4 expected), there is always a strong peak at the times where both conditions have an increased number of selective units.

Combinations of a stimulus condition and the load condition, as shown in Figure 4.11 (B) (lowest p value at 1150 ms and 650 ms, 86 and 73 significant units with 29 doubly selective units compared to 11 expected), also always show a peak, as do stimulus-reward pairs (not shown). However, the match condition does not exhibit a surplus of doubly selective units with any of the stimulus conditions (for example, see Figure 4.11 (C)).

4.4 Population code

All analyses so far have considered each unit in isolation. They provide information about the number of units involved in coding for experimental conditions, a useful information that has also been investigated in other studies on working memory in PFC (Warden and Miller, 2007, 2010).

However, when analyzing individual units, population dynamics are not taken into account and information encoded in activity patterns of multiple neurons cannot be detected. Multivariate decoding methods offer a way to incorporate population codes in answering the question how much information can be extracted from the recorded neural activity (Hung et al., 2005; Meyers et al., 2008).

4.4.1 Classifiers

There are various issues to consider when choosing a decoding technique, including the selection of appropriate features and an appropriate classification algorithm. For the features, the most obvious choice is to use the same data that was analyzed on the single unit level: binned spike counts using a 200 ms rectangular window that slides in 50 ms steps. Instead of using the spike count of just one unit per analysis, all simultaneously recorded units from a recording session can contribute to a feature vector. For example, in a recording session with 50 isolated single units, the feature vector is 50-dimensional. At trial time 100 ms, it consists of the number of spikes for each unit in the period from 0 ms to 200 ms.

Only simultaneously recorded data can be used as features for classification. In some experimental designs, it is possible to pool data from multiple recording sessions (Meyers et al., 2008, 2012). However, in the working memory experiment the protocol varied in each recording session, so it is not possible to create a virtual population of neurons across sessions. Therefore, each classification analysis is run independently on 17 recording sessions (10 from monkey L, 7 from monkey J).

With these definitions, there is now a feature vector of spike counts every 50 ms for each trial. In the following, most analyses are performed independently for each time step (for the exceptions to this rule, see sub-

section 4.4.4). A number of different classifiers were compared on these features (using only one recording session from each animal to avoid overfitting) to determine if one consistently outperformed the others: *Linear Discriminant Analysis* (LDA), logistic regression, random forests and *Support Vector Machines* (SVMs) with various kernels (Bishop, 2006).

In order to assess the classification performance, a tenfold cross-validation procedure was used: the trials were divided into ten subsets. A single subset was used as test data on which the classification accuracy was measured, and the other nine were used as training data. The process was repeated ten times, each time with a different subset as test data.

In addition, the class sizes were equalized by randomly sampling trials from larger classes. For example, if there were 420 match trials and 390 non-match trials in a recording session, a random subset of 390 match trials would be used for classification. This process was also repeated ten times and occurred before cross-validation, so each sampled set was cross validated. The classification accuracy reported here is the average of these 100 determined accuracies. Finally, the SVM classifiers have free parameters. A nested tenfold cross-validation was used to determine the optimal parameter settings.

A number of kernels have been used for the SVM classifier. The simplest is the linear kernel, which computes the distance directly from the two feature vectors $\mathbf{x}$ and $\mathbf{y}$ as a dot product

$$K_{\text{lin}}(\mathbf{x}, \mathbf{y}) = \mathbf{x}^T \mathbf{y}. \tag{4.3}$$

This kernel results in a linear decision surface, similar to LDA or logistic regression. The quadratic kernel incorporates first order interactions between the features:

$$K_{\text{quad}}(\mathbf{x}, \mathbf{y}) = (\mathbf{x}^T \mathbf{y} + r)^2, \tag{4.4}$$

with a free parameter r. It can be used to learn curved decision surfaces. The *Radial Basis Function* (RBF) kernel can lead to more complex, but smooth decision surfaces:

$$K_{\text{RBF}}(\mathbf{x}, \mathbf{y}) = e^{-\gamma |\mathbf{x}-\mathbf{y}|^2}, \tag{4.5}$$

with the free bandwidth parameter γ. $|\mathbf{x} - \mathbf{y}|^2$ is the squared euclidean distance. If the quadratic or RBF kernel perform better than the linear kernel, additional information is contained in nonlinear interactions between the spike counts.

4.4.2 Spike train metrics

Spike train metrics are a way of quantifying similarity and dissimilarity of spike trains. They have been applied to investigate neural coding in a variety of systems (see (Victor, 2005) for an overview).

The euclidean distance between spike counts (as used in the RBF kernel) can be interpreted as a simple metric that only takes the firing rate into account. The specialized spike train metrics also account for the timing of individual spikes.

One such metric is the Victor-Purpura (VP) distance (Victor and Purpura, 1996). It is defined as the minimum cost of transforming one spike train A into a spike train B using a series of elementary steps. In each step, one of three operations can be performed: Insert a spike (cost 1), remove a spike (cost 1), or change the time of a spike t_a to a new time t_b with a cost based on the magnitude of the change multiplied by a free parameter q: $q|t_a - t_b|$. When using only insertion and deletion, the distance is equivalent to the difference between spike counts. The parameter q adjusts the sensitivity to spike count compared to spike timing. The VP distance follows a similar principle as the Levenshtein distance for strings and can be computed using the same dynamic programming principles.

The van Rossum metric introduced in (van Rossum, 2001) is another popular spike train distance measure. Here, the spike train is first converted into a function $f(t)$ using a kernel $h(t)$, usually defined as a decaying exponential function with a scale parameter τ:

$$h(t) = \begin{cases} 0, t < 0 \\ \frac{1}{\tau} e^{-t/\tau}, \text{otherwise.} \end{cases} \tag{4.6}$$

For a spike train s_a with spike times $t_1, t_2, ..., t_n$, the corresponding function is

$$f_a(t) = \sum_{i=1}^{n} h(t - t_i). \tag{4.7}$$

The van Rossum metric for two spike trains s_a and s_b is then defined as the L^2 distance between the two corresponding functions:

$$d(s_a, s_b) = \sqrt{\int_0^\infty (f_a(t) - f_b(t))^2 dt}. \tag{4.8}$$

The parameter τ has the same function as $\frac{1}{q}$ in the VP distance: a high value of τ leads to a spike count distance, a small τ means that coincident spikes contribute little to the distance value.

Other spike train metrics in the literature include measures based on correlation (Schreiber et al., 2003) and *Interspike Intervals* (ISIs)

(Kreuz et al., 2007). However, for many measures it is impossible or computationally very expensive to extend them to work on simultaneous spike trains from multiple units. For the VP distance, the extension is straightforward: add a fourth possible operation (with a cost k) that moves a spike from one unit to another (Aronov, 2003). The parameter k controls the importance of distinguishing the identity of units: a high k value indicates a "labeled line" code where units are treated separately, while a low k value leads to a "summed population" code where the identity of the unit firing a spike is of little importance.

Unfortunately, calculating the multi-unit van Rossum distance for data on the scale available for the working memory experiment is computationally intractable. However, a multi-unit extension for the van Rossum distance is also available by creating a mapping from a set of M spike trains $s_a^1, s_a^2, ..., s_a^M$ onto a function $\mathbf{f}$ that maps from time to a M-dimensional vector, where each unit is assigned a characteristic direction $\mathbf{e}^k$ (Houghton and Sen, 2008):

$$\mathbf{f_a}(t) = \sum_{k=1}^{M} (\mathbf{e}_k \sum_{i=1}^{n} h(t - t_i^k)). \tag{4.9}$$

The multi-unit distance between two sets of spike trains $\mathbf{s}_a$ and $\mathbf{s}_b$ from M units is then calculated using the L^2 distance in the same way as for single-unit case described above:

$$\mathbf{d}(\mathbf{s}_a, \mathbf{s}_b) = \sqrt{\int_0^\infty |\mathbf{f}_a(t) - \mathbf{f}_b(t)|^2 dt}, \tag{4.10}$$

where $|\mathbf{f}_a(t) - \mathbf{f}_b(t)|^2$ is the squared euclidean distance. As for the multi-unit VP distance, an angle parameter θ allows to interpolate between "labeled line" code (all unit directions $\mathbf{e}^k$ are orthogonal) and "summed population" code (all unit directions are identical). In combination with the efficient implementation of the van Rossum metric presented in (Houghton and Kreuz, 2012), this allows the calculation of distances from dozens of units and hundreds of trials in a tractable amount of time.

In order to use this metric as the basis for a SVM kernel, the euclidean distance of spike counts from Equation 4.5 is replaced with the metric:

$$K_{\text{Metric}}(\mathbf{x}, \mathbf{y}) = e^{-\gamma d(s_a, s_b)^2}. \tag{4.11}$$

When optimizing the metric parameters using cross validation, the optimal values could in principle allow for inferences on the importance of rate and coincidence coding (kernel width τ) or "labeled line" and "summed population" coding (unit angle θ). However, (Chicharro et al., 2011) found that such inferences are problematic because the classification accuracy is too unspecific to allow for reliable results when using it as a decision criterion.

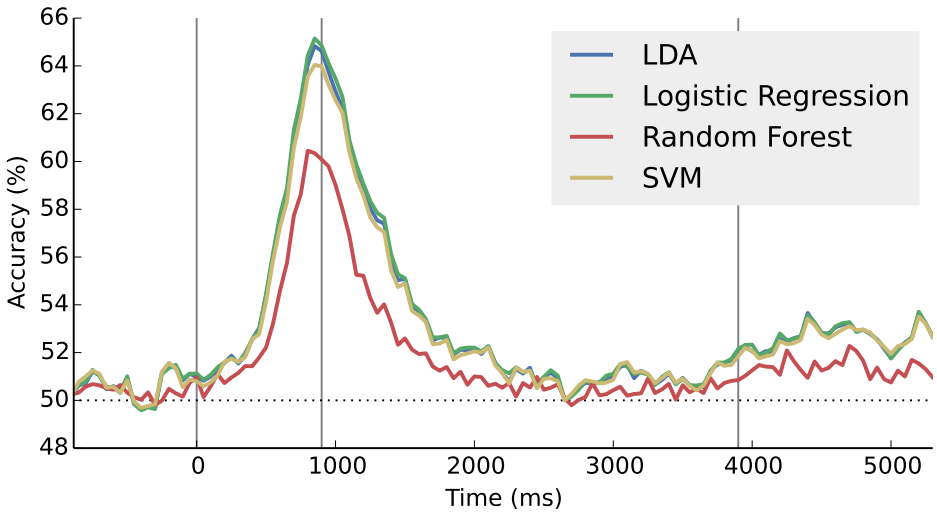

FIGURE 4.12: Comparison of classification performance of different classifiers for the load condition.

4.4.3 Classification results

How do the different classifiers and kernels compare in terms of classification accuracy? The accuracy is defined as the fraction of correctly classified trials. Because the class sizes are equalized, the chance performance for two-class conditions is 50 %.

Figure 4.12 shows the results for the load condition with different classifiers. LDA, logistic regression and linear SVM seem to learn the same decision surface and perform very similar while the random forest performs markedly worse. Qualitatively, all curves are very similar.

The classification accuracy for different SVM kernels is compared in Figure 4.13. The linear kernel and RBF kernel achieve a very similar, high accuracy. Using the van Rossum Kernel significantly reduces accuracy, indicating that the more precise temporal information and the increased flexibility of the spike train metric approach are not relevant for the neural code investigated here. Similarly, using up to three shorter bins instead of a single 200 ms spike count as features does not increase accuracy. In addition, the quadratic kernel offers the worst classification accuracy of all tested kernels. The first order interactions of spike counts from different neurons used by this kernel also do not appear to play a role in the investigated neural code. The linear SVM is used in all further classification analyses as it matches or outperforms all other methods in classification accuracy and is quick to train.

The classification results for all non-stimulus conditions are shown in Figure 4.14. The average classification accuracy reaches about 65 % during the relevant period in each condition, elevating the accuracy

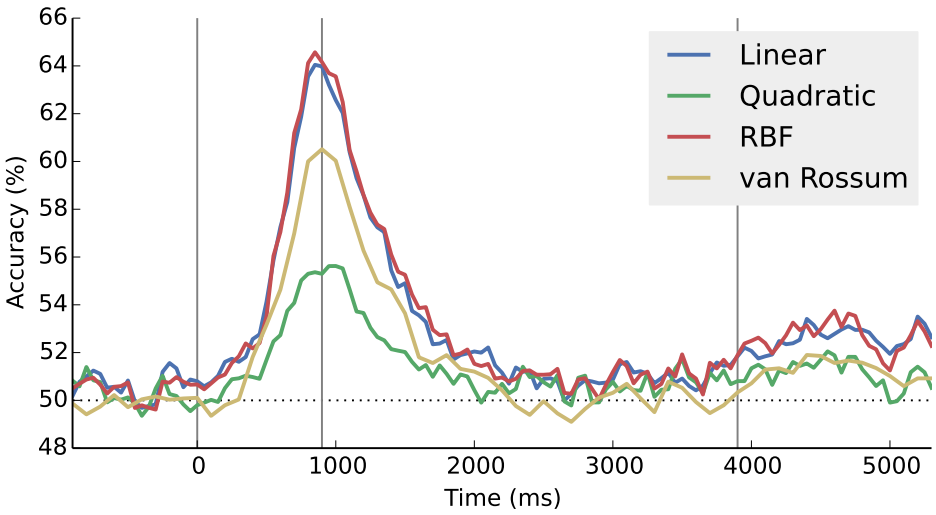

Figure 4.13: Comparison of classification performance of different SVM kernels for the load condition.

significantly above chance. The shape of the curves is quite similar to the results from single unit based statistical tests presented in subsection 4.3.2: the number of recorded units coding for a condition and the extractable information about the condition are highly correlated. With S the classification accuracy and C the number of significantly modulated units, the Pearson product-moment correlation coefficient r is defined as

$$r = \frac{\mathrm{Cov}(S, C)}{\sigma_S \sigma_C}. \tag{4.12}$$

For the load condition, $r = 0.917$. For the match condition it is $r = 0.984$ and for the reward condition $r = 0.988$.

Results for stimulus conditions are shown in Figure 4.15. There are 20 classes for each condition, so the baseline here is at 5 % accuracy. Note the big standard deviation: some recording sessions have very good results, others are poor. In general, as for the single units, the recordings from monkey J lead to better results in the stimulus condition than the recordings from monkey L. There are also large differences between the individual recording sessions. Some achieve a high accuracy but many remaining near chance level, as illustrated in Figure 4.16.

Again, the correlation between accuracy and the number of significantly modulated units is high, although not as pronounced as for the non-stimulus conditions: $r = 0.786$ for the first sample stimulus, $r = 0.716$ for the last sample, and $r = 0.83$ for the test stimulus condition.

Another question is whether the additional information provided by spike sorting is useful in decoding when compared to using spike detection

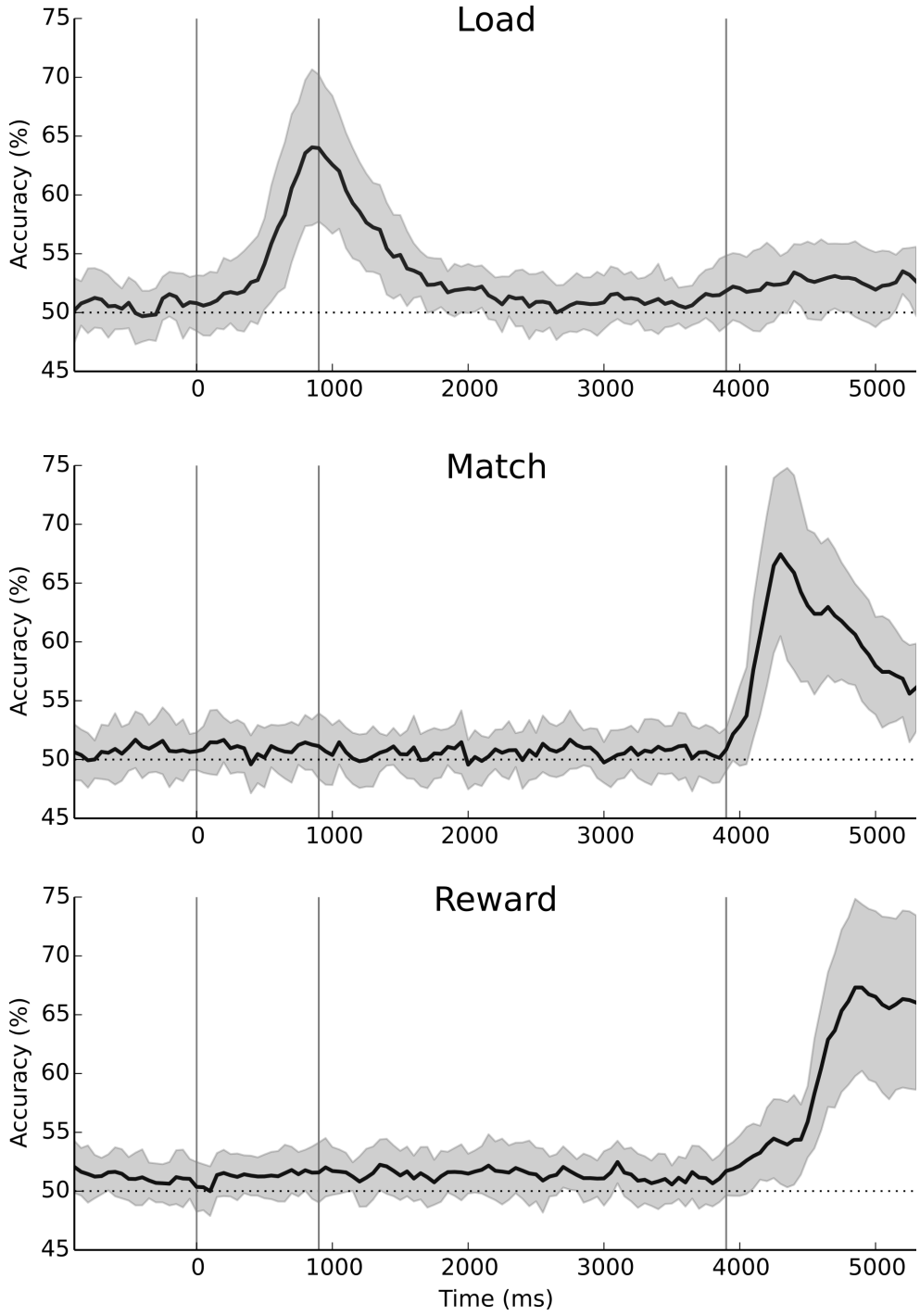

FIGURE 4.14: Classification accuracies for experimental conditions with two classes. Shaded areas indicate standard deviation.

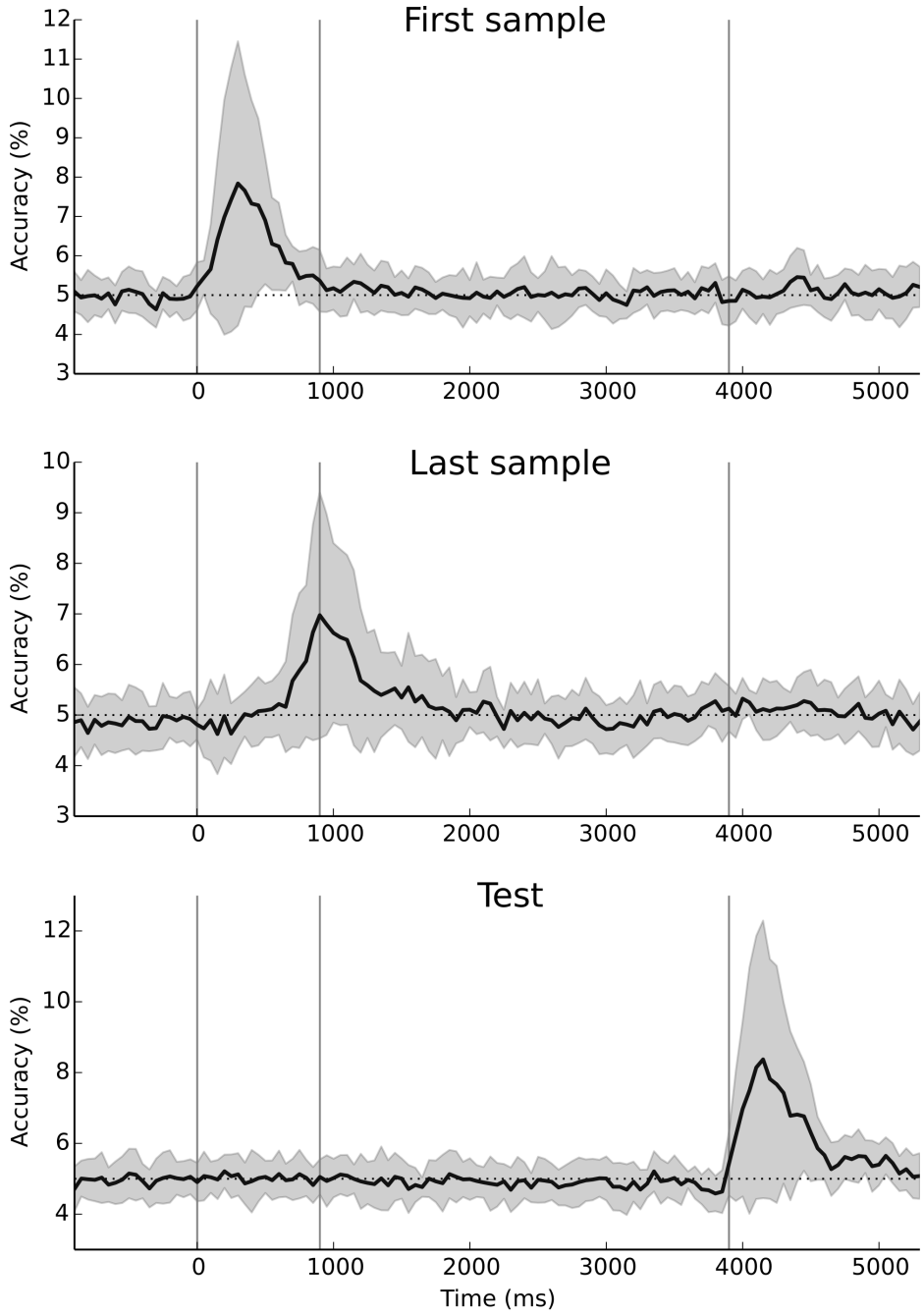

Figure 4.15: Classification accuracies for stimulus conditions. Shaded areas indicate standard deviation.

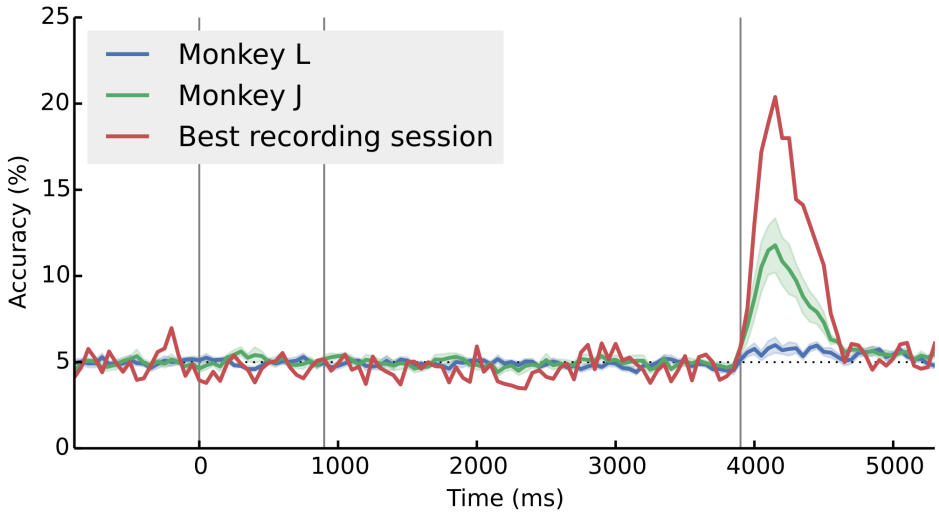

FIGURE 4.16: Comparison of average classification accuracy for the test stimulus condition for both animals and the best single recording session. The shaded areas indicate SEM. The best recording session is from monkey J and does not have a SEM because it corresponds to a single value at each time point.

results only. Figure 4.17 shows a comparison of classification accuracy in the match condition for different underlying features. For most of the period in which the accuracy is above chance, using the sorted units (one feature per unit) provides significantly higher classification accuracy than using detected spikes without additional identity information (one feature per tetrode, 3-12 in each recording session). The plot also shows the accuracy when using a compound firing rate: the sum of all individual firing rates (a single feature for each recording session). The accuracy also rises above chance but remains much lower than when using more features. The results are similar for all other conditions, confirming that spike sorting is indeed useful in increasing the information that can be extracted from spiking activity. The spike sorting used here is at least correct enough that these improvements are significant (Wilcoxon signed-rank test, $p < 0.05$).

4.4.4 Dynamic coding

An interesting question in neural coding is whether the firing rate patterns representing a particular condition change over the course of a trial or remain constant. This question can be addressed by training a classifier at one time point during a trial and evaluating the classification accuracy at other time points. If the patterns do not change, there should be little

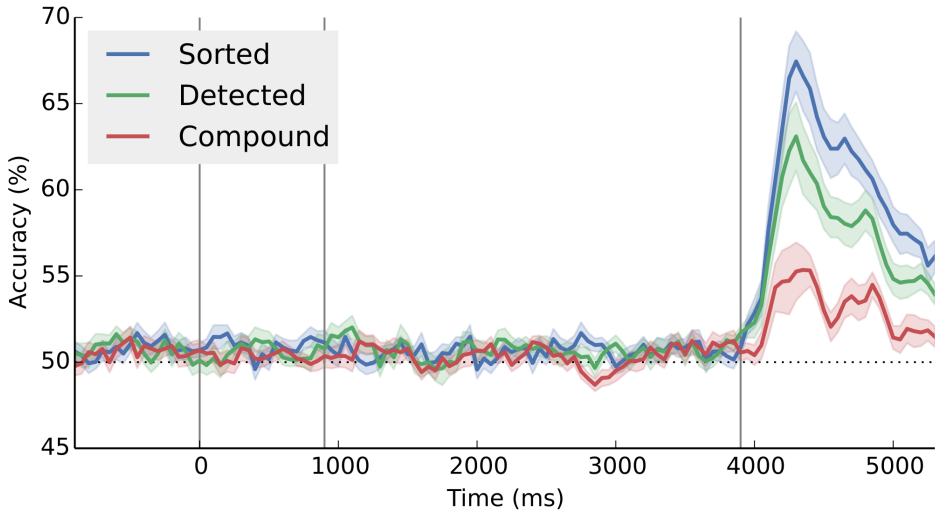

Figure 4.17: Classification accuracy for the match condition using sorted single, detected spikes per tetrode, or a single compound firing rate per recording session. Shaded areas indicate SEM.

or no decrease in accuracy when training at the time with the optimal accuracy (when training and testing at the same time). Potentially, the decoding accuracy at other times could even increase because periods where accuracy is high presumably contain more information that can be used for training. On the other hand, if the pattern changes, the accuracy should drop quickly when decoding further away from the training time.

In Figure 4.18, accuracies of all possible combinations of training and test times are shown for the non-stimulus conditions. Here, the diagonal corresponds to the accuracies with local training shown in previous plots. The highest decoding accuracies are found along the diagonal, so the best decoding performance is achieved by training and testing at the same time point. When comparing the results shown in Figure 4.18, the region of high accuracy for the load condition extends further away from the diagonal than the region for the match condition, indicating that the coding is more stable for the load condition. For the reward condition, the region of high accuracy is more rounded, so the patterns coding for reward change even more slowly.

Figure 4.19 shows more detail for some particular training times compared to the accuracy achieved with local training. The load condition has two peaks at different periods during the trial: the main peak at the end of the sample presentation and a much smaller peak during the test stimulus presentation. When training at the global maximum accuracy, the decoding performance stays close to the regular training but drops off during the delay period. The second peak later in the trial does not

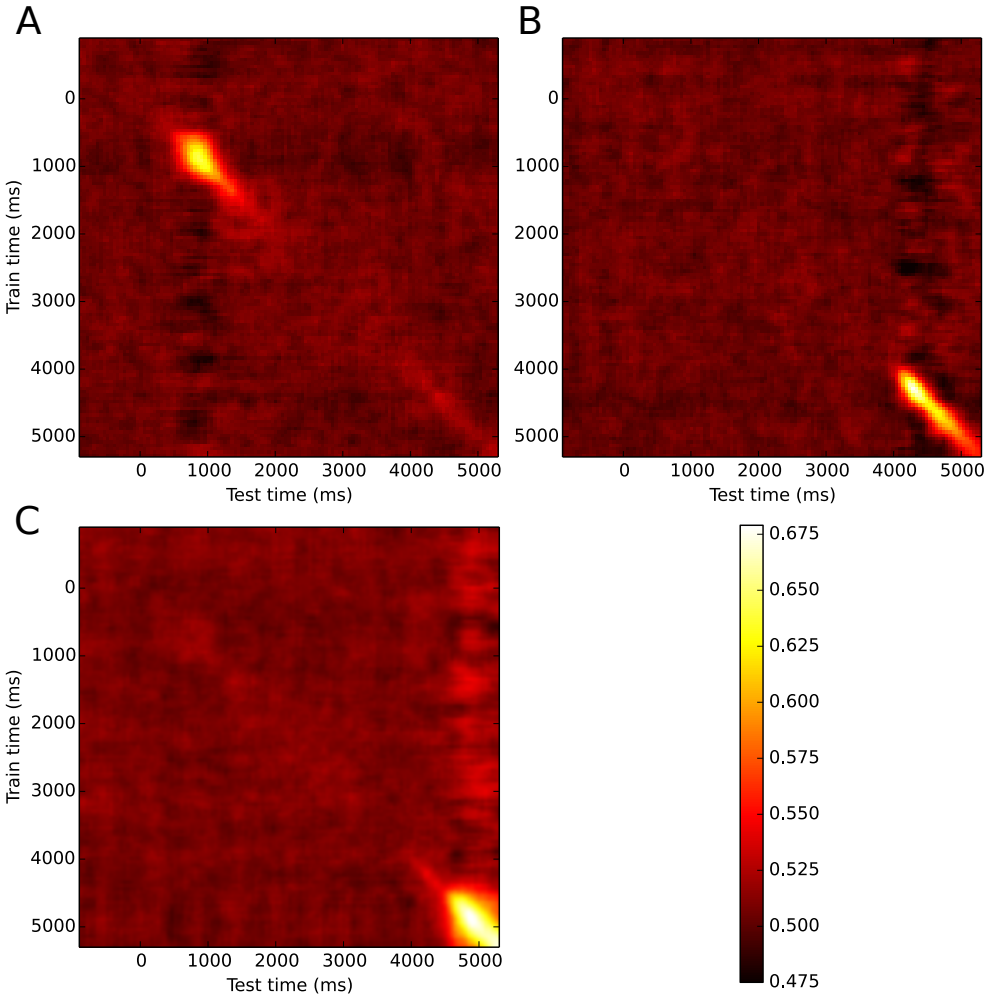

FIGURE 4.18: Classification accuracy with all time combinations for training and testing. **(A)** Load condition. **(B)** Match condition. **(C)** Reward condition.

appear when training during the initial peak. Conversely, when training at the local maximum later in the trial, there is no indication for a better than chance decoding performance during the main peak. It follows that while the activity patterns coding for load remain relatively stable during the sample stimuli presentation, they change later in the trial. The match condition only has a single peak with a long tail that lasts until the end of the trial. No matter at which point the training takes place, the decoding performance drops quickly, suggesting that the patterns coding for the match condition are volatile at all times. Finally, the coding for the reward condition is more stable. While the accuracy also drops off when compared to temporally local training and testing, it does so less quickly than for the other conditions.

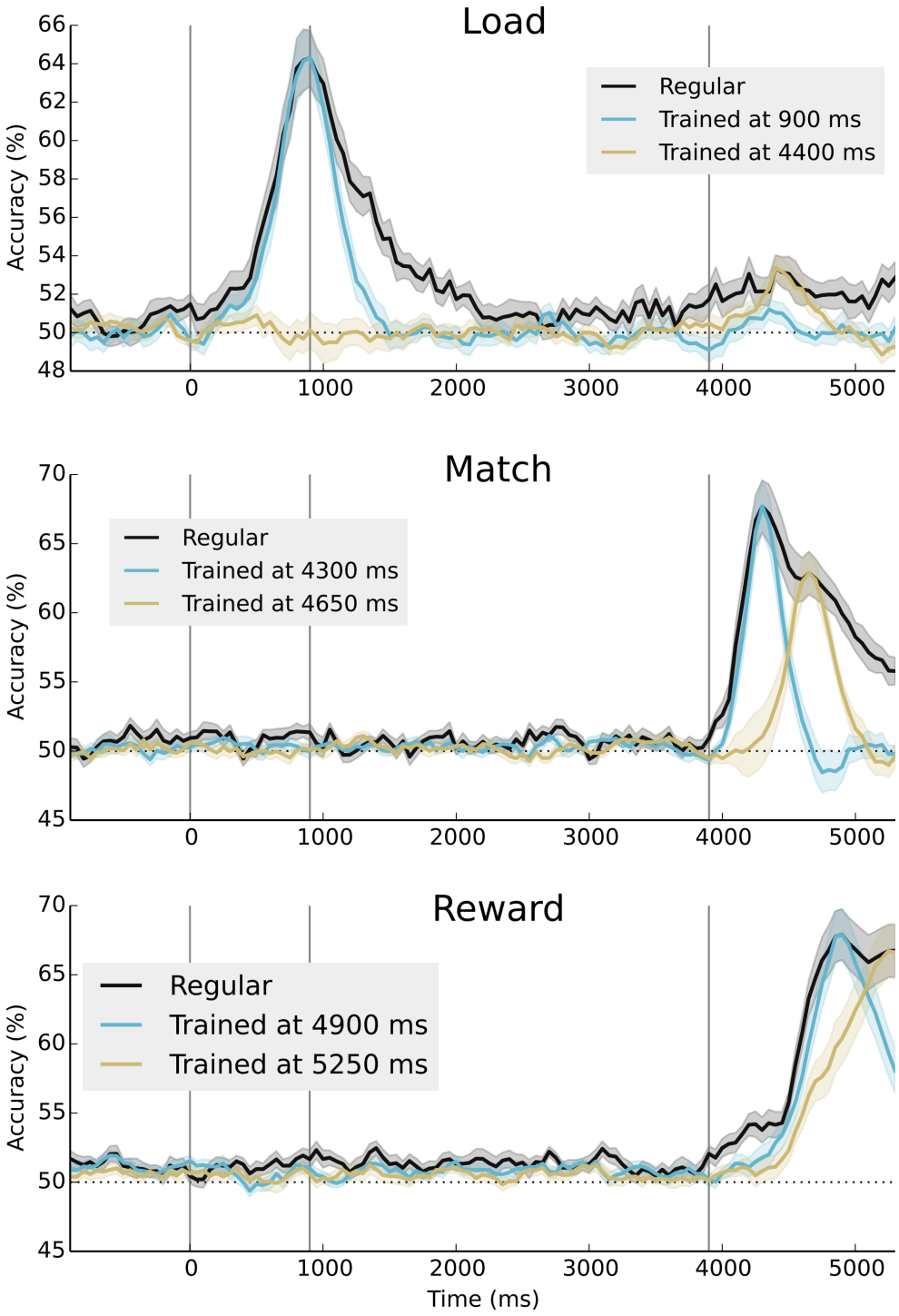

Figure 4.19: Comparison of regular, temporally local training and fixed training times. Shaded areas indicate SEM.

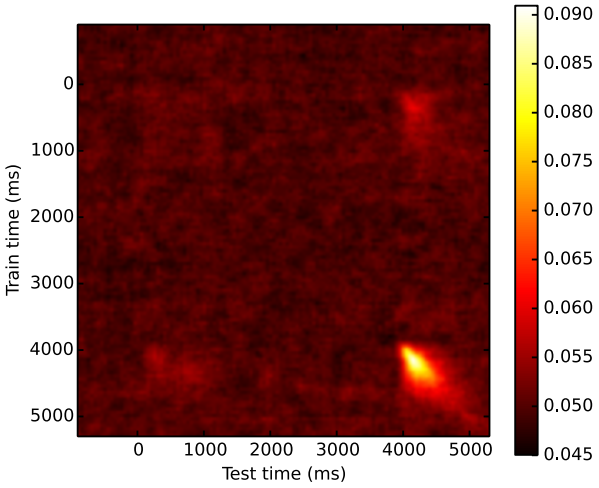

FIGURE 4.20: Classification accuracy with all time combinations for training and testing for the test stimulus condition.

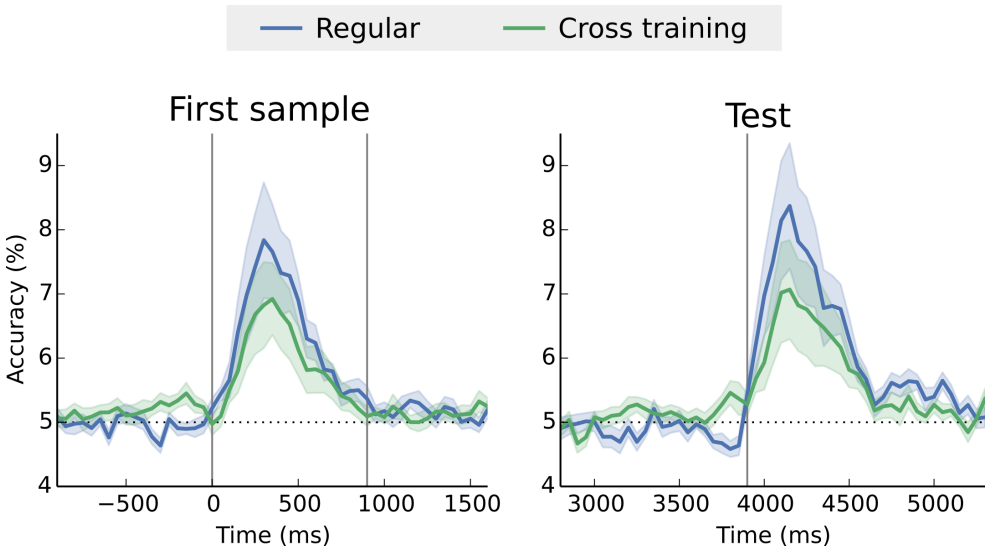

FIGURE 4.21: Comparison of regular training and cross training on a different stimulus condition. Shaded areas indicate SEM.

For the stimulus conditions, the code seems to be more stable (see Figure 4.20). However, the accuracy peaks for the stimulus conditions are short and it is therefore hard to make a confident judgment. But there is another way to investigate the stability of patterns for the stimulus conditions: because the classes (i.e. the stimulus identities) are identical for all stimulus conditions, a classifier can be trained with one condition (at a particular time point), and tested with another (at all time points). This fact is also hinted at by the regions of slightly elevated accuracy in Figure 4.20 for training or testing during the sample stimulus presentation.

Figure 4.21 shows the results for this approach using the first and test stimulus conditions. On the left, local training on the first stimulus condition is compared to training on the test stimulus condition at its peak accuracy. Both are tested on the first stimulus condition. On the right, the roles are reversed: the regular training on the test stimulus is compared to training on the peak accuracy of the first sample stimulus condition. In both directions, the decoding accuracy with cross training remains below regular training, but significantly above chance. The high accuracy suggests that the patterns that represent stimulus identities remain largely unchanged over the course of the trial and even among different conditions. The results are similar for the other stimulus condition pairs.

4.5 Firing rate variability

As an additional modality of interest apart from the firing rate itself, the trial-to-trial variability of the firing rate has recently been suggested to carry information about the participation of neurons in behavioral tasks. (Churchland et al., 2010) report that the onset of visual stimuli reduces the firing rate variability in multiple areas of cat and macaque brains. However, the areas investigated in this study do not include the PFC. (Hussar and Pasternak, 2010) found that in macaque PFC during a motion discrimination task, firing rate variability responded to experimental events, dropping strongly at stimulus onset and declining slowly before other experimental events. This section describes the analysis results for firing rate variability in the working memory experiment.

The most common way to quantify firing rate variability is the Fano factor. It is defined as

$$F = \frac{\sigma_C^2}{\mu_C}, \tag{4.13}$$

where σ_C^2 is the variance and μ_C is the mean of spike counts in a particular bin. Since the Fano factor (or any other measure of trial-to-trial variability) cannot be computed from a single trial, it cannot be used for decoding analyses like the spike count in the previous sections. As before, spike counts are computed in 200 ms bins that slide across the trial in 50 ms steps, which is also consistent with the binning used in (Hussar and Pasternak, 2010).

Figure 4.22 shows the results averaged over all units. The figure also includes the average firing rate because changes in the Fano factor can be influenced by changes in firing rate. The Fano factor drops during the presentation of sample and test stimuli, in agreement with reported findings. However, the temporal progression of the Fano factor differs between monkeys and recording session.

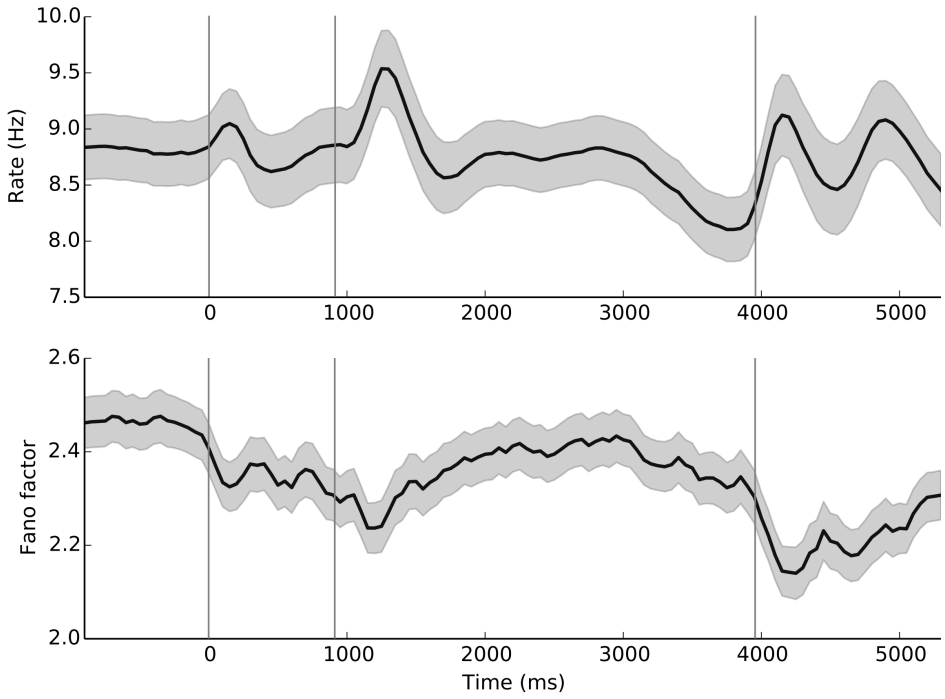

FIGURE 4.22: Firing rate and Fano factor averaged over all 629 units in all recording sessions. Shaded areas indicate SEM.

In Figure 4.23, average rates and Fano factors are plotted for the two monkeys individually. The drop in Fano factor that can be observed in Figure 4.22 is largely due to monkey L. For monkey J, the effect is reversed: the Fano factor rises during stimulus presentations. In addition, the average Fano factor is different for the two monkeys. The situation is similar when distinguishing between recording sessions: temporal progression and average value of the Fano factor differ (see Figure 4.24 for examples). Therefore, the results from (Hussar and Pasternak, 2010) could not be replicated in the working memory experiment and little evidence was found for a general Fano factor drop during stimulus presentation as reported in (Churchland et al., 2010).

4.6 Local field potential

The decoding analyses described so far are based on spike data only. However, recordings of slower neural oscillations, the *Local Field Potential* (LFP), are also available. A previous study with a similar dataset (Pipa et al., 2009) found that LFP oscillations were selective for both stimulus identity and behavioral success.

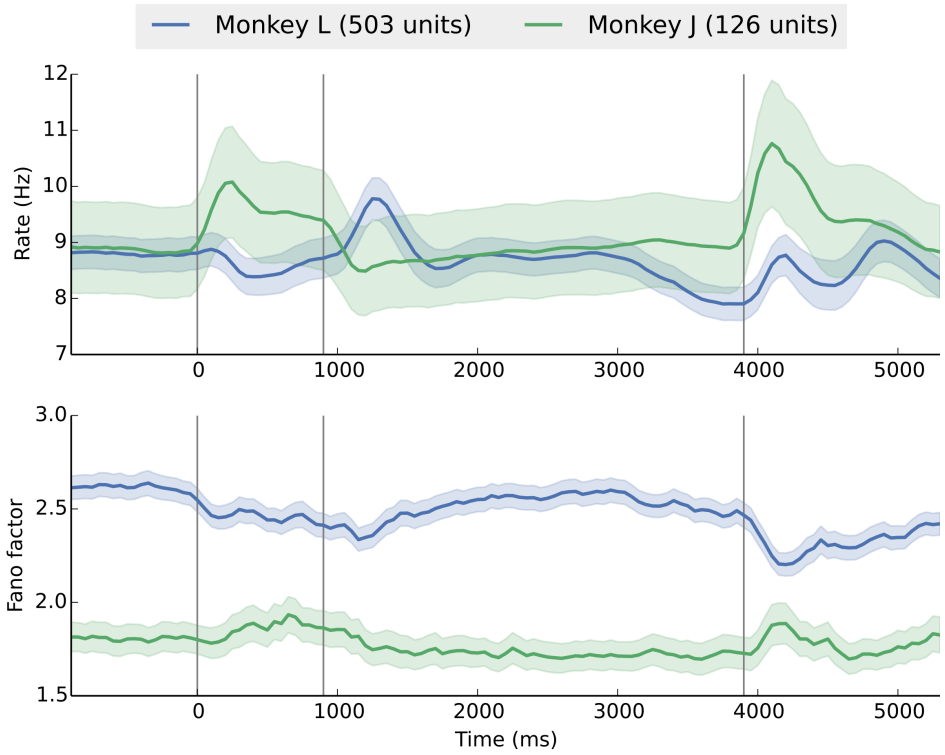

Figure 4.23: Firing rate and Fano factor averaged over all units from each monkey separately. Shaded areas indicate SEM.

In order to estimate how much information the LFP carried compared to the spike data, features were calculated from the LFP and used in a classifier in the same way as spike counts in section 4.4: using a *Fast Fourier Transform* (FFT) with a 200 ms window (multiplied by a Hanning window to reduce spectral leakage) shifted across the data in 50 ms steps results in 200 spectral power values per tetrode every 50 ms, just as counting spikes results in one feature per unit. Because the total number of features is quite large, spectral powers for frequencies above 100 Hz were ignored (in line with (Pipa et al., 2009)) in the following so that only 20 features per tetrode remained. Including the higher frequencies as features did not improve the classification accuracy.

Using the spectral features, classifiers can be used to decode experimental conditions in the same way as described in section 4.4, but with features derived from the LFP data. This enables a direct comparison of the classification accuracies: does the LFP carry more or less information than the spike data? Another interesting question is whether the information is complementary or redundant. It can be addressed by training a single classifier on both LFP and spike features. If this classifier achieves a higher accuracy than those using just one of the data types, each contains information not available in the other.

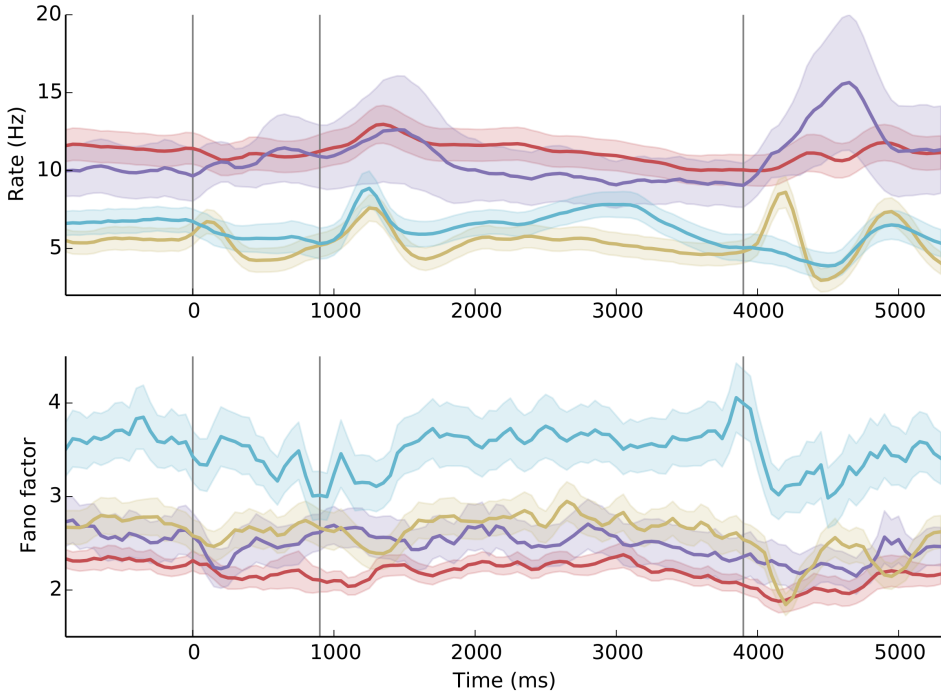

FIGURE 4.24: Average firing rate and Fano factor for four recording sessions from monkey L. Shaded areas indicate SEM.

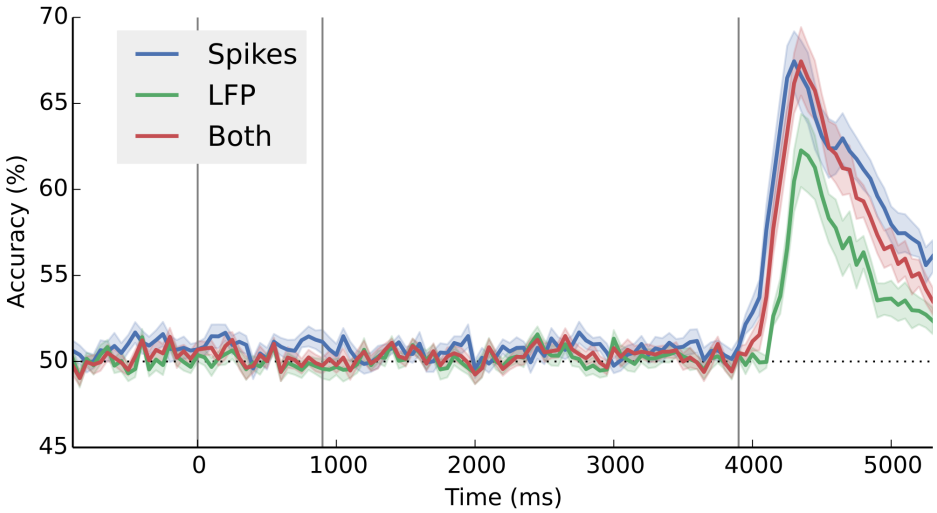

FIGURE 4.25: Spikes and LFP as features for classification in the match condition. Shaded areas indicate SEM.

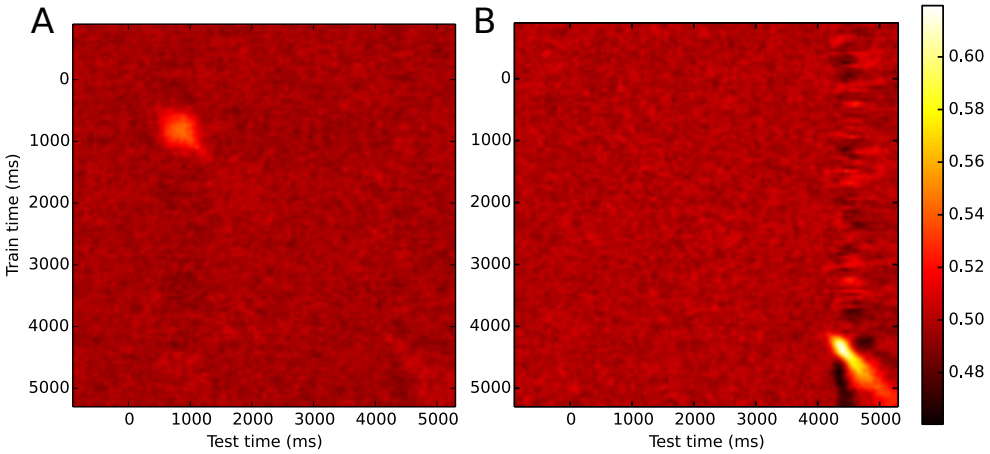

Figure 4.26: Classification accuracy with all time combinations for training and testing when using LFP features. **(A)** Load condition. **(B)** Match condition.

Figure 4.25 shows the decoding accuracies for the match condition. The accuracy of the decoder based on the LFP rises significantly above chance during the same time period as the spike-based decoder, but does not reach the same accuracy. A decoder based on both features improves upon the LFP-based accuracy but does not surpass the spike-based decoder. The results are similar for the other non-stimulus conditions. For the stimulus conditions, the spectral LFP features do not carry enough information to enable a classification accuracy above chance.

While the LFP does not carry additional information compared to the spike data, the patterns could be more stable over time. Figure 4.26 shows the accuracies in the load and match condition for independent train and test times. The total decoding accuracy is smaller than for the results using spike count data shown in Figure 4.18, but both results are comparable: the accuracy on the diagonals is always higher than off-diagonal, where it falls quickly. The region of elevated accuracy extends further from the diagonal for the load condition than for the match condition. Therefore, the behavior of LFP and spike patterns over time do not exhibit a clear difference.

4.7 Summary

This chapter presented analysis results from a variety of methods to illuminate the coding mechanisms in PFC during a working memory task. Most units react to the experiment with significantly changed firing rates at some time during the trial, which is in line with values reported in the literature (Miller et al., 1996). Many units also exhibit selective firing rate modulation for all experimental conditions in various periods during

the trial. However, the percentages of significantly modulated units, in particular for stimuli, are smaller than those previously reported (Warden and Miller, 2007). For most experimental condition pairs (with the exception of stimuli and match), the number of units selective for both conditions is significantly above chance, a fact that had not been reported before.

Different approaches were compared for decoding experimental conditions from single trials. Linear classifiers based on binned spike rates offered the highest classification accuracy. More complex, nonlinear methods could not improve decoding performance, similar to what was found in (Astrand et al., 2014). Using the LFP power as additional features also did not improve accuracy, while using multi-units instead of sorted single units significantly reduced it. In general, the decoding accuracy time course is highly correlated with the number of significant units, indicating that even large numbers of units contribute nonredundant information.

Another interesting examined aspect is the time dependence of coding. It was investigated by decoupling training and test times for classifiers. In agreement with (Meyers et al., 2008), the code was found to change dynamically over the course of a trial. In addition, code variations over the two between sample and test stimuli could be analyzed. Although there is a gap of three seconds between the stimulus presentations, the coding patterns showed a striking similarity: training at one stimulus period and testing at the other yields almost the same classification performance as training locally.

Unfortunately, some analyses did not lead to conclusive results, and corresponding claims from the literature could not be replicated with the dataset used here: a clear distinction of putative cell types by their extracellular waveform, as reported by (Diester and Nieder, 2008; Hussar and Pasternak, 2009), is not possible. Likewise, a consistent effect of stimulus presentation on the Fano factor that was reported by (Churchland et al., 2010; Hussar and Pasternak, 2010) was not found in the data. While the Fano factor does change over the course of the trial, the effect is not stable across monkeys and recording sessions.

Conclusion 5

Electrophysiological recording techniques are advancing rapidly. They can serve to answer more sophisticated research question but also require increasingly complex methods for data management and analysis. This thesis contributes new methods for data sharing, algorithm development and data analysis.

With Spyke Viewer, a new software tool was developed to help researchers using electrophysiology to effectively analyze the large datasets produced by current experiments. With a focus on quick algorithm development and effortless sharing of both data and analysis methods, it is an important part of a neuroscientist's toolbox. The program is publicly available, has been used by other researchers, and was instrumental in large parts of this work.

Since spikes are of great importance in most electrophysiology experiments, spike sorting is a critical step in many data analysis pipelines. With the advent and advancement of multichannel recording techniques, an increasing number of neurons can be recorded simultaneously such that coincident spikes become more prevalent. The *Bayes Optimal Template Matching* (BOTM) algorithm was originally developed to be able to resolve temporally overlapping spikes. In this thesis, improved methods for the overlap resolution were introduced and the algorithm was extensively verified on simulated data. In addition, empirical data was used to estimate the amount of spikes that would be missed or misclassified with other techniques. In many of the investigated tetrode recordings, these spikes account for a large fraction of all detected spikes.

The development Spyke Viewer and BOTM was motivated by the aim to analyze recordings from a working memory experiment. For all experimental conditions, a large fraction of the recorded units in PFC exhibit selectively modulated firing rates. Many units are selective for

multiple conditions. The firing rates can also be used to decode the conditions in a time-resolved manner using linear classifiers. The neural code does not remain constant over the whole trial, but the rate of change is different for the various experimental conditions. For the sample and test stimuli, the neural response is very similar.

While Spyke Viewer has been taken up by some researchers, there are still features to implement that would broaden its appeal. The main challenge stems from the many data formats in electrophysiology. Even though Spyke Viewer supports a multitude of formats, many potential users would need to invest some work converting their data or writing a custom IO class. This is also a problem for data sharing and has been recognized as an important issue for the electrophysiology community. It is currently being addressed by various initiatives, including the INCF electrophysiology task force in which I am also involved. Spyke Viewer will support the new standard once it is ready.

BOTM and its extensions presented in this thesis are a large step in providing accurate spike sorting results for temporally overlapping spikes, but there is still room for further improvements: Because the initial unit templates need to be estimated independently, a clustering step is still required. Clustering spikes is notoriously error prone and often requires human intervention. Progress in this area will also improve the results of subsequent BOTM sortings.

Another issue for spike sorting is the increase in parallel recording channels. While BOTM is well suited to work with multiple channels, it does not scale well to dozens or hundreds of channels. As such recording devices are becoming more popular, the algorithm has to be adjusted to adapt to this new challenge.

A common theme in the literature on working memory and PFC, selective neural activity during the delay period has been observed in many studies. Therefore, the absence of sustained information about stimuli or memory load during the delay period was a surprising finding. Neither statistical tests nor decoding revealed clear signs for such activity in the data. While it remains possible that the information is encoded in a more complex fashion than firing rate patterns, the analyses in this direction (e.g. using spike train metrics or LFP) have not yielded results beyond what was already achieved using binned firing rates. It seems that at least the neurons that were recorded in the experiment cannot be responsible for maintaining information on the sample stimuli during the delay period. This is compatible with the view that the PFC is not responsible for the actual storage of memory items, but for attention and control, which would not necessarily depend on the stimulus identity during the delay period.

Electrophysiology and neuroscience have made great progress in the recent past, but many challenges remain for the coming years. Driven by

new experimental techniques, the complexity of the required data analysis continues to increase. The software, methods, and results presented in this thesis will help to deal with the existing and arising challenges.

Bibliography

Abeles, M. (1982). *Local Cortical Circuits: an electrophysiological study*. Springer.

Aronov, D. (2003). Fast algorithm for the metric-space analysis of simultaneous responses of multiple single neurons. *Journal of Neuroscience Methods*, 124(2):175–179.

Astrand, E., Enel, P., Ibos, G., Dominey, P. F., Baraduc, P., and Ben Hamed, S. (2014). Comparison of classifiers for decoding sensory and cognitive information from prefrontal neuronal populations. *PLOS ONE*, 9(1).

Atiya, A. (1992). Recognition of multiunit neural signals. *IEEE Transactions on Biomedical Engineering*, 39(7).

Baddeley, A. D. (2000). The episodic buffer: a new component of working memory? *Trends in Cognitive Sciences*, 4(11):417–423.

Baddeley, A. D. and Hitch, G. (1974). Working memory. *Psychology of Learning and Motivation*, 8:47–89.

Bar-Gad, I., Ritov, Y., Vaadia, E., and Bergman, H. (2001). Failure in identification of overlapping spikes from multiple neuron activity causes artificial correlations. *Journal of Neuroscience Methods*, 107(1-2).

Bishop, C. M. (2006). *Pattern recognition and machine learning*. Springer New York.

Blanche, T. J., Spacek, M. a., Hetke, J. F., and Swindale, N. V. (2005). Polytrodes: high-density silicon electrode arrays for large-scale multiunit recording. *Journal of Neurophysiology*, 93:2987–3000.

Branco, T., Clark, B. A., and Häusser, M. (2010). Dendritic discrimination of temporal input sequences in cortical neurons. *Science*, 329(5999):1671–1675.

Brown, G. D., Yamada, S., and Sejnowski, T. J. (2001). Independent component analysis at the neural cocktail party. *Trends in Neurosciences*, 24(1):54–63.

Buzsáki, G. (2004). Large-scale recording of neuronal ensembles. *Nature Neuroscience*, 7(5):446–451.

Buzsáki, G., Anastassiou, C. a., and Koch, C. (2012). The origin of extracellular fields and currents – EEG, ECoG, LFP and spikes. *Nature Reviews Neuroscience*, 13(6):407–420.

Chen, Y., Martinez-Conde, S., Macknik, S. L., Bereshpolova, Y., Swadlow, H. a., and Alonso, J.-M. (2008). Task difficulty modulates the activity of specific neuronal populations in primary visual cortex. *Nature Neuroscience*, 11(8):974–982.

Chicharro, D., Kreuz, T., and Andrzejak, R. G. (2011). What can spike train distances tell us about the neural code? *Journal of Neuroscience Methods*, 199(1):146–165.

Choi, J. H., Jung, H. K., and Kim, T. (2006). A new action potential detector using the MTEO and its effects on spike sorting systems at low signal-to-noise ratios. *IEEE Transactions on Biomedical Engineering*, 53(4):738–746.

Choi, J. H. and Kim, T. (2002). Neural action potential detector using multi-resolution TEO. *Electronics Letters*, 38(12):541–543.

Churchland, M. M., Yu, B. M., Cunningham, J. P., Sugrue, L. P., Cohen, M. R., Corrado, G. S., Newsome, W. T., Clark, A. M., Hosseini, P., Scott, B. B., Bradley, D. C., Smith, M. a., Kohn, A., Movshon, J. A., Armstrong, K. M., Moore, T., Chang, S. W., Snyder, L. H., Lisberger, S. G., Priebe, N. J., Finn, I. M., Ferster, D., Ryu, S. I., Santhanam, G., Sahani, M., and Shenoy, K. V. (2010). Stimulus onset quenches neural variability: a widespread cortical phenomenon. *Nature Neuroscience*, 13(3):369–378.

Connors, B. W. and Gutnick, M. J. (1990). Intrinsic firing patterns of diverse neocortical neurons. *Trends in Neurosciences*, 13(3):99–104.

Cowan, N. (1988). Evolving conceptions of memory storage, selective attention, and their mutual constraints within the human information-processing system. *Psychological Bulletin*, 104(2):163–191.

Cowan, N. (2001). The magical number 4 in short-term memory: a reconsideration of mental storage capacity. *Behavioral and Brain Sciences*, 24(4):87–114.

Diester, I. and Nieder, A. (2008). Complementary contributions of prefrontal neuron classes in abstract numerical categorization. *The Journal of Neuroscience*, 28(31):7737–7747.

Dinning, G. and Sanderson, A. (1981). Real-time classification of multiunit neural signals using reduced feature sets. *IEEE Transactions on Biomedical Engineering*, 28(12):804–812.

Duncan, J. (2001). An adaptive coding model of neural function in prefrontal cortex. *Nature Reviews Neuroscience*, 2(11):820–829.

Eckhorn, R. and Thomas, U. (1993). A new method for the insertion of multiple microprobes into neural and muscular tissue, including fiber electrodes, fine wires, needles and microsensors. *Journal of Neuroscience Methods*, 49(3):175–179.

Einevoll, G. T., Franke, F., Hagen, E., Pouzat, C., and Harris, K. D. (2012). Towards reliable spike-train recordings from thousands of neurons with multielectrodes. *Current Opinion in Neurobiology*, 22(1):11–17.

Ekanadham, C., Tranchina, D., and Simoncelli, E. P. (2014). A unified framework and method for automatic neural spike identification. *Journal of Neuroscience Methods*, 222:47–55.

Franke, F. (2011). *Real-Time Analysis of Extracellular Multi Electrode Recordings*. PhD thesis, TU Berlin.

Franke, F., Natora, M., Boucsein, C., Munk, M. H. J., and Obermayer, K. (2010). An online spike detection and spike classification algorithm capable of instantaneous resolution of overlapping spikes. *Journal of Computational Neuroscience*, 29(1):127–148.

Franke, F., Pröpper, R., Alle, H., Meier, P., Geiger, J., Obermayer, K., and Munk, M. H. J. (2015a). Spike Sorting of Synchronous Spikes from Local Neuron Ensembles. *Journal of Neurophysiology*, (in revision).

Franke, F., Quiroga, R. Q., Hierlemann, A., and Obermayer, K. (2015b). Bayes optimal template matching for spike sorting – combining fisher discriminant analysis with optimal filtering. *Journal of Computational Neuroscience*.

Fuster, J. M. and Alexander, G. E. (1971). Neuron activity related to short-term memory. *Science*, 173(3997):652–654.

Garcia, S. and Fourcaud-Trocmé, N. (2009). OpenElectrophy: An Electrophysiological Data- and Analysis-Sharing Framework. *Frontiers in Neuroinformatics*, 3(May).

Garcia, S., Guarino, D., Jaillet, F., Jennings, T., Pröpper, R., Rautenberg, P. L., Rodgers, C. C., Sobolev, A., Wachtler, T., Yger, P., and Davison, A. P. (2014). Neo: an object model for handling electrophysiology data in multiple formats. *Frontiers in Neuroinformatics*, 8(February).

Gold, C., Henze, D. a., Koch, C., and Buzsáki, G. (2006). On the origin of the extracellular action potential waveform: A modeling study. *Journal of Neurophysiology*, 95(5):3113–3128.

Gray, C. M., Maldonado, P. E., Wilson, M., and McNaughton, B. L. (1995). Tetrodes markedly improve the reliability and yield of multiple single-unit isolation from multi-unit recordings in cat striate cortex. *Journal of Neuroscience Methods*, 63(1-2):43–54.

Hafting, T., Fyhn, M., Molden, S., Moser, M.-B., and Moser, E. I. (2005). Microstructure of a spatial map in the entorhinal cortex. *Nature*, 436(8):801–806.

Hagen, E., Ness, T. r. V., Khosrowshahi, A., Sø rensen, C., Fyhn, M., Hafting, T., Franke, F., and Einevoll, G. T. (2015). ViSAPy: A Python tool for biophysics-based generation of virtual spiking activity for evaluation of spike-sorting algorithms. *Journal of Neuroscience Methods*, 6.

Halchenko, Y. O. and Hanke, M. (2012). Open is Not Enough. Let's Take the Next Step: An Integrated, Community-Driven Computing Platform for Neuroscience. *Frontiers in Neuroinformatics*, 6(June).

Harris, K. D., Henze, D. A., Csicsvari, J., Hirase, H., and Buzsáki, G. (2000). Accuracy of tetrode spike separation as determined by simultaneous intracellular and extracellular measurements. *Journal of Neurophysiology*, 84(1):401–414.

Herz, A. V. M., Meier, R., Nawrot, M. P., Schiegel, W., and Zito, T. (2008). G-Node: An integrated tool-sharing platform to support cellular and systems neurophysiology in the age of global neuroinformatics. *Neural Networks*, 21(8):1070–1075.

Houghton, C. J. and Kreuz, T. (2012). On the efficient calculation of van Rossum distances. *Network: Computation in Neural Systems*, 23(1-2):48–58.

Houghton, C. J. and Sen, K. (2008). A new multi-neuron spike-train metric. *Neural Computation*, 20(6):1495–1511.

Hubel, D. H. and Wiesel, T. N. (1959). Receptive fields of single neurones in the cat's striate cortex. *The Journal of Physiology*, 148(3):574–591.

Hubel, D. H. and Wiesel, T. N. (1962). Receptive fields, binocular interaction and functional architecture in the cat's visual cortex. *The Journal of Physiology*, 160(1):106–154.

Hung, C. P., Kreiman, G., Poggio, T., and DiCarlo, J. J. (2005). Fast readout of object identity from macaque inferior temporal cortex. *Science*, 310(5749):863–866.

Hussar, C. R. and Pasternak, T. (2009). Flexibility of Sensory Representations in Prefrontal Cortex Depends on Cell Type. *Neuron*, 64(5):730–743.

Hussar, C. R. and Pasternak, T. (2010). Trial-to-trial variability of the prefrontal neurons reveals the nature of their engagement in a motion discrimination task. *Proceedings of the National Academy of Sciences*, 107(50).

Jäckel, D., Frey, U., Fiscella, M., Franke, F., and Hierlemann, A. (2012). Applicability of independent component analysis on high-density microelectrode array recordings. *Journal of Neurophysiology*, 108(1):334–348.

Jacobsen, C. F. and Nissen, H. W. (1937). Studies of cerebral function in primates. IV. The effects of frontal lobe lesions on the delayed alternation habit in monkeys. *Journal of Comparative Psychology*, 23:101–112.

Katsuki, F., Qi, X.-L. X.-L., Meyer, T., Kostelic, P. M., Salinas, E., and Constantinidis, C. (2013). Differences in Intrinsic Functional Organization Between Dorsolateral Prefrontal and Posterior Parietal Cortex. *Cerebral Cortex*, 24(9).

Kerr, J. N. D. and Denk, W. (2008). Imaging in vivo: watching the brain in action. *Nature Reviews Neuroscience*, 9(3):195–205.

Kim, K. H. and Kim, S. J. (2000). Neural spike sorting under nearly 0-dB signal-to-noise ratio using nonlinear energy operator and artificial neural-network classifier. *IEEE Transactions on Biomedical Engineering*, 47(10):1406–1411.

Kreuz, T., Haas, J. S., Morelli, A., Abarbanel, H. D. I., and Politi, A. (2007). Measuring spike train synchrony. *Journal of Neuroscience Methods*, 165(1):151–161.

Kubota, K. and Niki, H. (1971). Prefrontal cortical unit activity and delayed alternation performance in monkeys. *Journal of Neurophysiology*, 34:337–347.

Lebedev, M. A., Messinger, A., Kralik, J. D., and Wise, S. P. (2004). Representation of attended versus remembered locations in prefrontal cortex. *PLOS Biology*, 2(11).

Letelier, J. C. and Weber, P. P. (2000). Spike sorting based on discrete wavelet transform coefficients. *Journal of Neuroscience Methods*, 101(2):93–106.

Lewicki, M. S. (1994). Bayesian modeling and classification of neural signals. *Neural Computation*, 6(5):1005–1030.

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

Lidierth, M. (2009). sigTOOL: A MATLAB-based environment for sharing laboratory-developed software to analyze biological signals. *Journal of Neuroscience Methods*, 178(1):188–196.

Madany Mamlouk, A., Sharp, H., Menne, K. M., Hofmann, U. G., and Martinetz, T. (2005). Unsupervised spike sorting with ICA and its evaluation using GENESIS simulations. *Neurocomputing*, 65-66:275–282.

Marblestone, A. H., Zamft, B. M., Maguire, Y. G., Shapiro, M. G., Cybulski, T. R., Glaser, J. I., Amodei, D., Stranges, P. B., Kalhor, R., Dalrymple, D. a., Seo, D., Alon, E., Maharbiz, M. M., Carmena, J. M., Rabaey, J. M., Boyden, E. S., Church, G. M., and Kording, K. P. (2013). Physical principles for scalable neural recording. *Frontiers in Computational Neuroscience*, 7(October).

Markram, H., Toledo-Rodriguez, M., Wang, Y., Gupta, A., Silberberg, G., and Wu, C. (2004). Interneurons of the neocortical inhibitory system. *Nature Reviews Neuroscience*, 5(10):793–807.

McCormick, D. and Connors, B. W. (1985). Comparative electrophysiology of pyramidal and sparsely spiny stellate neurons of the neocortex. *Journal of Neurophysiology*, 54(10):782–806.

McNaughton, B. L., O'Keefe, J., and Barnes, C. (1983). The stereotrode: a new technique for simultaneous isolation of several single units in the central nervous system from multiple unit records. *Journal of Neuroscience Methods*, 8(4):391–397.

Meier, R., Egert, U., Aertsen, A., and Nawrot, M. P. (2008). FIND–a unified framework for neural data analysis. *Neural Networks*, 21(8):1085–1093.

Meyers, E. M., Freedman, D. J., Kreiman, G., Miller, E. K., and Poggio, T. (2008). Dynamic population coding of category information in inferior temporal and prefrontal cortex. *Journal of Neurophysiology*, 100(3):1407–1419.

Meyers, E. M., Qi, X.-L. X.-L., and Constantinidis, C. (2012). Incorporation of new information into prefrontal cortical activity after learning working memory tasks. *Proceedings of the National Academy of Sciences*, 109(12):4651–4656.

Miller, E. K., Erickson, C., and Desimone, R. (1996). Neural mechanisms of visual working memory in prefrontal cortex of the macaque. *The Journal of Neuroscience*, 16(16):5154–5167.

Miller, G. A. (1956). The psychological review. *Psychological Review*, 63(2):81–97.

Mishkin, M. (1957). Effects of small frontal lesions on delayed alternation in monkeys. *Journal of Neurophysiology*, 20(6):615–622.

Mitchell, J. F., Sundberg, K. a., and Reynolds, J. H. (2007). Differential attention-dependent response modulation across cell classes in macaque visual area V4. *Neuron*, 55(1):131–141.

Ohiorhenuan, I. E., Mechler, F., Purpura, K. P., Schmid, A. M., Hu, Q., and Victor, J. D. (2010). Sparse coding and high-order correlations in fine-scale cortical networks. *Nature*, 466(7306):617–621.

O'Keefe, J. and Dostrovsky, J. (1971). The hippocampus as a spatial map. Preliminary evidence from unit activity in the freely-moving rat. *Brain Research*, 34:171–175.

O'Keefe, J. and Recce, M. L. (1993). Phase relationship between hippocampal place units and the EEG theta rhythm. *Hippocampus*, 3(3):317–330.

Pazienti, A. and Grün, S. (2006). Robustness of the significance of spike synchrony with respect to sorting errors. *Journal of Computational Neuroscience*, 21(3):329–342.

Pedreira, C., Martinez, J., Ison, M. J., and Quiroga, R. Q. (2012). How many neurons can we see with current spike sorting algorithms? *Journal of Neuroscience Methods*, 211(1):58–65.

Perez, F. and Granger, B. (2007). IPython: a system for interactive scientific computing. *Computing in Science & Engineering*, 9(3):21–29.

Pillow, J. W., Shlens, J., Chichilnisky, E. J., and Simoncelli, E. P. (2013). A model-based spike sorting algorithm for removing correlation artifacts in multi-neuron recordings. *PLOS ONE*, 8(5).

Pillow, J. W., Shlens, J., Paninski, L., Sher, A., Litke, A. M., Chichilnisky, E. J., and Simoncelli, E. P. (2008). Spatio-temporal correlations

and visual signalling in a complete neuronal population. *Nature*, 454(7207):995–999.

Pipa, G. and Munk, M. H. J. (2011). Higher Order Spike Synchrony in Prefrontal Cortex during Visual Memory. *Frontiers in Computational Neuroscience*, 5(June).

Pipa, G., Städtler, E., Rodriguez, E., Waltz, J., Muckli, L., Singer, W., Goebel, R., and Munk, M. H. J. (2009). Performance-and stimulus-dependent oscillations in monkey prefrontal cortex during short-term memory. *Frontiers in Integrative Neuroscience*, 3(March).

Postle, B. R. (2006). Working memory as an emergent property of the mind and brain. *Neuroscience*, 139(1):23–38.

Pouzat, C., Mazor, O., and Laurent, G. (2002). Using noise signature to optimize spike-sorting and to assess neuronal classification quality. *Journal of Neuroscience Methods*, 122(1):43–57.

Prentice, J. S., Homann, J., Simmons, K. D., Tkačik, G., Balasubramanian, V., and Nelson, P. C. (2011). Fast, scalable, Bayesian spike identification for multi-electrode arrays. *PLOS ONE*, 6(7).

Pröpper, R. and Obermayer, K. (2013). Spyke Viewer: a flexible and extensible platform for electrophysiological data analysis. *Frontiers in Neuroinformatics*, 7(November).

Quiroga, R. Q., Nadasdy, Z., and Ben-Shaul, Y. (2004). Unsupervised spike detection and sorting with wavelets and superparamagnetic clustering. *Neural Computation*, 16(8):1661–1687.

Roberts, W. M. and Hartline, D. K. (1975). Separation of multi-unit nerve impulse trains by a multi-channel linear filter algorithm. *Brain Research*, 94(1):141–149.

Rutishauser, U., Schuman, E. M., and Mamelak, A. N. (2006). Online detection and sorting of extracellularly recorded action potentials in human medial temporal lobe recordings, in vivo. *Journal of Neuroscience Methods*, 154(1-2):204–224.

Sakurai, Y., Nakazono, T., Ishino, S., Terada, S., Yamaguchi, K., and Takahashi, S. (2013). Diverse synchrony of firing reflects diverse cell-assembly coding in the prefrontal cortex. *Journal of Physiology – Paris*, 107(6):459–470.

Schreiber, S., Fellous, J. M., Whitmer, D., Tiesinga, P., and Sejnowski, T. J. (2003). A new correlation-based measure of spike timing reliability. *Neurocomputing*, 52-54:925–931.

Segev, R., Goodhouse, J., Puchalla, J., and Berry, M. J. (2004). Recording spikes from a large fraction of the ganglion cells in a retinal patch. *Nature Neuroscience*, 7(10):1154–1161.

Shimazaki, H. and Shinomoto, S. (2010). Kernel bandwidth optimization in spike rate estimation. *Journal of Computational Neuroscience*, 29(1-2):171–182.

Shiraishi, Y., Katayama, N., Takahashi, T., Karashima, A., and Nakao, M. (2009). Multi-neuron action potentials recorded with tetrode are not instantaneous mixtures of single neuronal action potentials. In *Proceedings of the Annual International Conference of the IEEE Engineering in Medicine and Biology Society*, volume 2009, pages 4019–4022.

Simon, W. (1965). The real-time sorting of neuro-electric action potentials in multiple unit studies. *Electroencephalography and Clinical Neurophysiology*, pages 192–195.

Spacek, M., Blanche, T. J., and Swindale, N. (2008). Python for large-scale electrophysiology. *Frontiers in Neuroinformatics*, 2(January).

Stein, R., Andreassen, S., and Oğuztöreli, M. (1979). Mathematical analysis of optimal multichannel filtering for nerve signals. *Biological Cybernetics*, 24:19–24.

Sternberg, S. (1966). High-speed scanning in human memory. *Science*, 153(3736):652.

Stevenson, I. H. and Kording, K. P. (2011). How advances in neural recording affect data analysis. *Nature Neuroscience*, 14(2):139–142.

Takahashi, S., Anzai, Y., and Sakurai, Y. (2003). A new approach to spike sorting for multi-neuronal activities recorded with a tetrode—how ICA can be practical. *Neuroscience Research*, 46(3):265–272.

Teeters, J. L., Benda, J., Davison, A. P., Eglen, S., Gerhard, S., Gerkin, R. C., Grewe, J., Harris, K. D., Jackson, T., Moucek, R., Pröpper, R., Sessions, H. L., Smith, L. S., Sobolev, A., Sommer, F. T., Stoewer, A., and Wachtler, T. (2013). Considerations for developing a standard for storing electrophysiology data in HDF5. *Frontiers in Neuroinformatics. Conference Abstract: Neuroinformatics 2013*.

Tsodyks, M., Kenet, T., Grinvald, A., and Arieli, A. (1999). Linking spontaneous activity of single cortical neurons and the underlying functional architecture. *Science*, 286(5446):1943–1946.

van Rossum, M. (2001). A novel spike distance. *Neural Computation*, 3(4):8–13.

Vargas-Irwin, C. and Donoghue, J. P. (2007). Automated spike sorting using density grid contour clustering and subtractive waveform decomposition. *Journal of Neuroscience Methods*, 164(1).

Verkhratsky, A., Krishtal, O. A., and Petersen, O. H. (2006). From Galvani to patch clamp: The development of electrophysiology. *Pflügers Archiv - European Journal of Physiology*, 453(3):233–247.

Victor, J. and Purpura, K. P. (1996). Nature and precision of temporal coding in visual cortex: a metric-space analysis. *Journal of Neurophysiology*, 76(8).

Victor, J. D. (2005). Spike train metrics. *Current Opinion in Neurobiology*, 15(5):585–592.

Vigneswaran, G., Kraskov, A., and Lemon, R. N. (2011). Large identified pyramidal cells in macaque motor and premotor cortex exhibit "thin spikes": implications for cell type classification. *The Journal of Neuroscience*, 31(40).

Vollgraf, R., Munk, M. H. J., and Obermayer, K. (2005). Optimal filtering for spike sorting of multi-site electrode recordings. *Network: Computation in Neural Systems*, 16(1):85–113.

Warden, M. R. and Miller, E. K. (2007). The representation of multiple objects in prefrontal neuronal delay activity. *Cerebral Cortex*, 17:i41–i50.

Warden, M. R. and Miller, E. K. (2010). Task-dependent changes in short-term memory in the prefrontal cortex. *The Journal of Neuroscience*, 30(47):15801–15810.

Wilson, F. A. W., Scalaidhe, S. P., and Goldman-Rakic, P. S. (1993). Dissociation of object and spatial processing domains in primate prefrontal cortex. *Science*, 260(5116):1955–1958.

Wilson, F. A. W., Scalaidhe, S. P. O., and Goldman-Rakic, P. S. (1994). Functional synergism between putative gamma-aminobutyrate-containing neurons and pyramidal neurons in prefrontal cortex. *Proceedings of the National Academy of Sciences*, 91(4).

Zaytsev, Y. V. and Morrison, A. (2012). Increasing quality and managing complexity in neuroinformatics software development with continuous integration. *Frontiers in Neuroinformatics*, 6(January).

Zhang, P.-M., Wu, J.-Y., Zhou, Y., Liang, P.-J., and Yuan, J.-Q. (2004). Spike sorting based on automatic template reconstruction with a partial solution to the overlapping problem. *Journal of Neuroscience Methods*, 135(1):55–65.