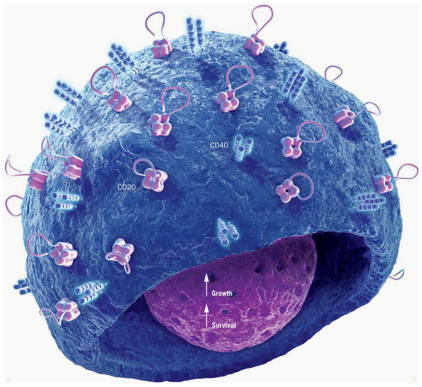
TÉCNICO LISBOA



Thesis Title

Subtitle

Full Name

Thesis to obtain the Master of Science Degree in

Biomedical Engineering

Supervisor(s): Prof./Dr. Lorem Ipsum

Examinatiom Committee

Chairperson: Prof. Lorem
Supervisor: Prof. Lorem Ipsum
Co-Supervisor: Prof. Lorem Ipsum
Members of the Committe: Dr. Lorem Ipsum
Prof. Lorem Ipsum

January 2015

Anyone who has never made a mistake has never tried anything new.

Albert Einstein

Acknowledgments

I would like to thank the Academy, bla bla bla..

Abstract

The Objective of this Work ... (English)

Keywords

Keywords (English)

Resumo

O objectivo deste trabalho ... (Português)

Palavras Chave

Palavras-Chave (Português)

Contents

List of Figures

List of Tables

Abbreviations

List of Symbols

1

Introduction

Contents

1.1 Motivation

Motivation Section.

1.2 State of The Art

State of The Art Section.

1.2.1 Dummy Subsection A

State of Art Subsection A

1.2.2 Dummy Subsection B

State of Art Subsection B

1.3 Original Contributions

Contributions Section.

1.4 Thesis Outline

Outline Section.

2

A Chapter

Contents

Present the chapter content.

2.1 Experimental Setup

In Neto et al., 2016 they described in detail a procedure for precisely aligning two probes for in vivo "paired-recordings" such that the spiking activity of a single neuron is monitored with both a dense extracellular silicon polytrode and a juxtacellular micro-pipette. A "ground truth" dataset was acquired from rat cortex with 32 and 128-channel silicon polytrodes and it is available online (http://www.kampff-lab.org/validating-electrodes). A brief description of the dual-recording setup design and protocol are presented below.

2.1.1 Set-up design and protocol

In Figure 2.1a is presented a schematic of the dual-probe recording station where two aligned, multi-axis micromanipulators (Scientifica, UK) and a long working distance optical microscope are required to reliably target neural cell bodies located within 100 μm of the polytrode electrode sites without optical guidance. A "PatchStar" (PS) and an "In-Vivo Manipulator" (IVM) are mounted on opposite sides of a rodent stereotaxic frame with different approach angles, 61 nd -48.61 rom the horizontal, respectively (Fig 2.1b).

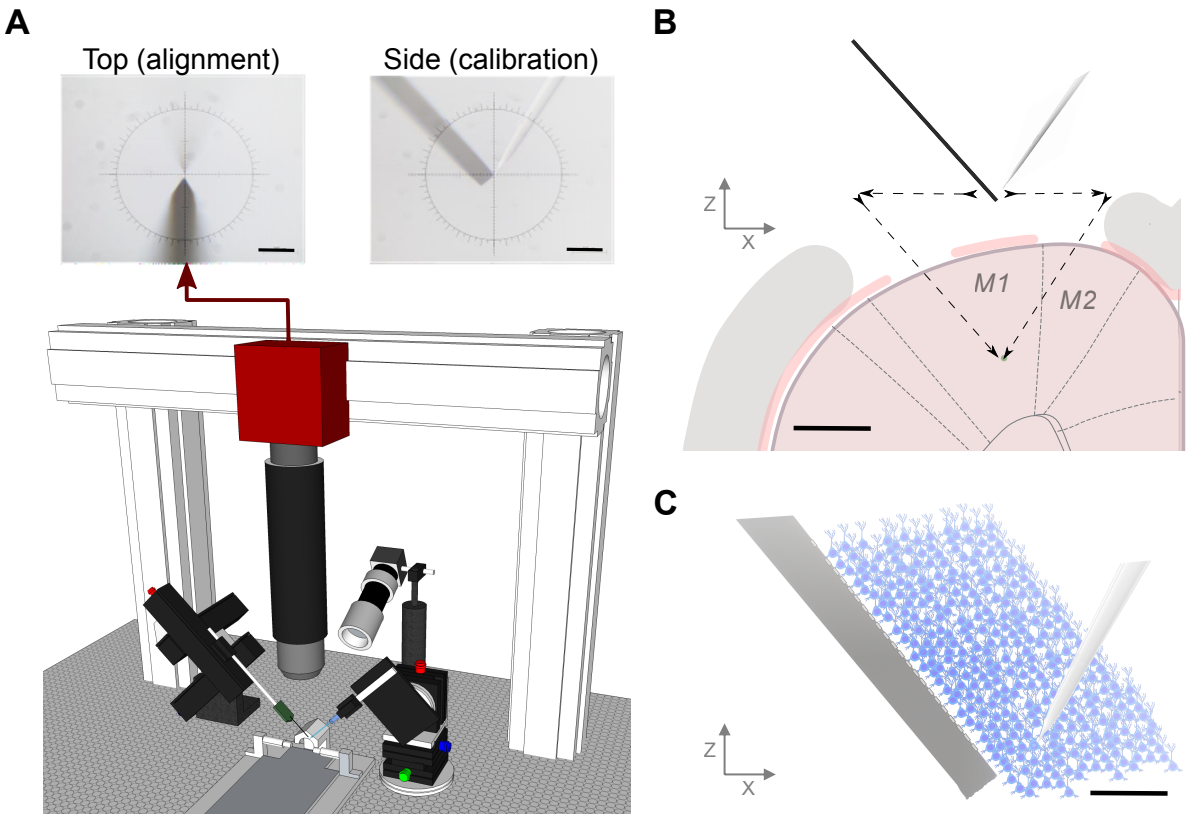


Figure 2.1: In vivo paired-recording setup: design and method. (a) Schematic of the dual-probe recording station. The PS micromanipulator drives the juxtacellular pipette and the IVM manipulator drives the extracellular polytrode. The setup includes a long working distance microscope assembled from optomechanical components mounted on a three-axis motorized stage. The alignment image provides a high-resolution view from above the stereotactic frame, upper left, however a side-view can also be obtained for calibration purposes, upper right (scale bar 100 μm). (b) Schematic of a coronal view of the craniotomy and durotomies with both probes positioned at the calibration point. The distance between durotomies, such that the probe tips meet at deep layers in cortex, was around 2 mm. The black arrows represent the motion path for both electrodes entering the brain (scale bar 1 mm). (c) Diagram of simultaneous extracellular and juxtacellular paired-recording of the same neuron at a distance of 90 μm between the micropipette tip and the closest electrode on the extracellular polytrode (scale bar 100 μm).

Rats (400 to 700 g, both sexes) of the Long-Evans strain were anesthetized with a mixture of Ketamine (60 mg/kg intraperitoneal, IP) and Medetomidine (0.5 mg/kg IP) and placed in the stereotaxic frame. Anesthetized rodents underwent a surgical procedure to remove the skin and the skull to expose the targeted brain region. Two reference electrodes Ag-AgCl wires (Science Products GmbH, E-255) were inserted at the posterior part of the skin incision on opposite sides of the skull.

Each paired-recording experiment began with the optical “zeroing” of both probes. Each probe was positioned, sequentially, at the center of the microscope image (indicated by a crosshair) and the motorized manipulator coordinates set to zero (Figure 2.1a). As shown in Figure 2.1b, this alignment is performed directly above the desired rendez-vous point inside the brain, as close as possible above dura, usually between 1 and 4 mm, but far enough to reduce background light reflected from the brain surface into the microscope image. The distance reported is the Euclidean distance between the tip of the pipette and the closest extracellular electrode. After both the extracellular probe and juxtacellular pipette positions were sequentially “zeroed” to the center of the microscope image, the extracellular probe was inserted first, at a constant velocity of 1 $\mu m.s^{-1}$, automatically controlled

by the manipulator software. When the extracellular probe was in place, the juxtacellular pipette, pulled from 1.5 mm capillary borosilicate glass (Warner Instruments, USA) and filled with PBS 1x, was then lowered through a second durotomy. The juxtacellular pipette with a long thin taper had typical tip diameter of 1-4 μm and resistance of 3-7 $M\Omega$. As the electrode was advanced towards a cell membrane, we observed an increase in the pipette resistance. If spikes were observed a slight suction was applied to obtain a stable attachment to the cell membrane. As the juxtacellular electrode was advanced through the brain, several neurons were encountered at different locations along the motion path and, consequently, at different distances from the extracellular polytrodes.

All experiments were performed with two different high-density silicon polytrodes. A commercially available 32-channel probe (A1x32-Poly3-5mm-25s-177-CM32, NeuroNexus, USA), with 177 μm^2 area electrodes (iridium) and an inter-site pitch of 22-25 μm, was used in the first experiments. In later experiments, they used a 128-channel probe produced in the collaborative NeuroSeeker project (http://www.neuroseeker.eu/) and developed by IMEC using CMOS-compatible process technology. These probe electrodes were 400 μm^2 (20 x 20 μm^2) large arranged at a pitch of 22.5 μm

Extracellular signals in a frequency band of 0.1-7500 Hz and juxtacellular signals in a frequency band of 300-8000 Hz were sampled at 30 kHz with 16-bit resolution and were saved in a raw binary format for subsequent offline analysis using a Bonsai interface.

2.2 Dataset

The dataset consists of twenty-three paired recording with a distance of less than 200 μm between the targeted neuron and the closest extracellular electrode. These were acquired from twenty-three cells, from the cortex of several anesthetized rats.

On Fig. 2.2a is an example of the signal acquired from using the juxtacellular pipette, which, with an amplitude of around 4mV, reveals the typical high signal-to-ratio that the signal this probe yields. On the figure (figure 2.2b), many of the spikes were aligned and plotted together. We can see that this waveform keeps it shape over the course of the recording. In this case, as is in most of the recording, it has a positive-before-negative biphasic waveform, which is indicative that there was a good coupling between the pipette and the neuron's soma (Herfst et al, 2012). However, in two case, that I used, the waveform has a negative-before-positive profile indicating incomplete contact between the cell membrane and the pipette, lowering the signal-to-ratio (SNR) significantly but remaining detectable. (2015_09_03_Pair9.0 and 2015_09_04_Pair5.0)

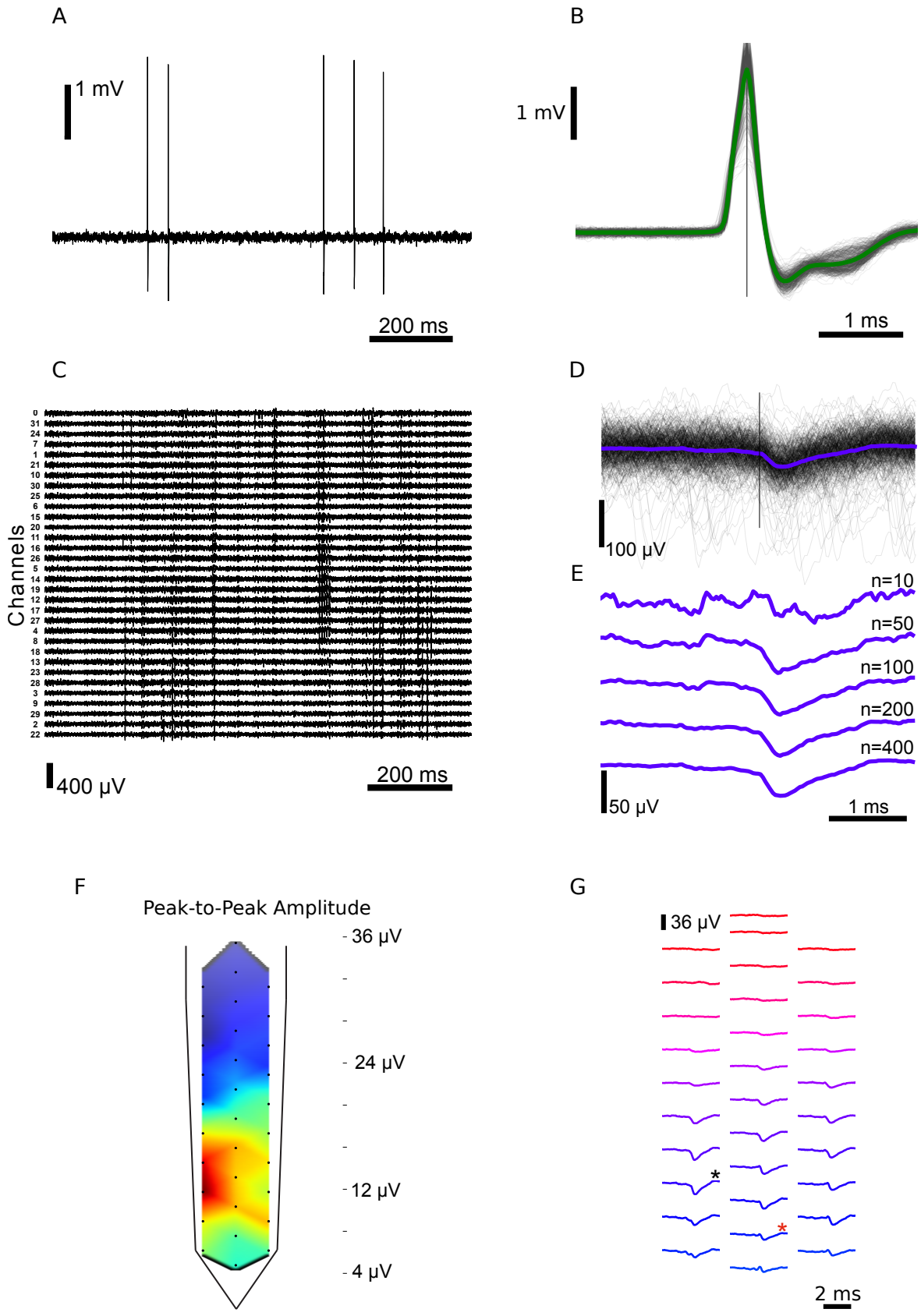


Figure 2.2: Paired extracellular and juxtacellular recordings from the same neuron (a) Representative juxtacellular recording from a cell in layer 5 of motor cortex, 68 μm from the extracellular probe (2014_10_17_Pair1.0), with a firing rate of 0.9 Hz. (b) The juxtacellular action potentials are overlaid, time-locked to the time of positive peak, with the average spike waveform superimposed in green (n= 442 spikes). (c) Representative extracellular recording that corresponds to the same time window as the recording in panel A. Traces are ordered from upper to lower electrodes and channel numbers are indicated. (d) Extracellular waveforms, aligned on the juxtacellular spike peak, for a single channel (channel 18). (e) the juxtacellular triggered average (JTA) obtained by including an increasing number of juxtacellular events (n as indicated). (f) Spatial distribution of the amplitude for each channel's extracellular JTA waveform. The peak-to-peak amplitude within a time window (+/- 1 ms) surrounding the juxtacellular event was measured and the indicated color code was used to display and interpolate these amplitudes throughout the probe shaft. (g) The JTAs are spatially arranged. The channel with the highest peak-to-peak JTA (channel 18) is marked with a black (*) and the closest channel (channel 9) is marked with a red

With such a high SNR, one can reliably use a simple threshold-based detector to calculate the times (hereafter juxta times) at which the juxta neuron spiked.The earliest extracellular recordings in the dataset were done using the 32-channel probe. Part of one of these recordings after the high-pass filter is illustrated in Fig. 2.2c. Each of these traces are plotted next to its neighbors, according the geometry of the probe. Most of the spikes are sensed by many electrodes revealing a coherent region of influence. This signal usually doesn't have a high SNR, as can be seen in Fig. 2.2d. To get the waveform of the EAP on this probe we perform Juxta-Triggered Averages (JTAs), where windows of 4 ms centered on the juxta spikes are averaged so that the noise decreases and the waveform becomes clear. In figure 2.2g) are represented the JTAs of each electrode in its correct position in the 32-channel probe. It is possible to see that the EAP has a different waveform on different electrode sites. They are also displaced in time: on electrodes farther way, the waveform is delayed with respect to one on a electrode closer to the neuron. The JTA peak-to-peak amplitude for each channel interpolated within the electrode site geometry, sometimes called "the cell footprint" (Delgado Ruz and Schultz, 2014), is shown in Figure (figure 2.2f)

During the course of this project I focused on 5 recordings where the 128-channels probe was used. These are presented in Fig. 2.3 and summarized in Table 2.1.

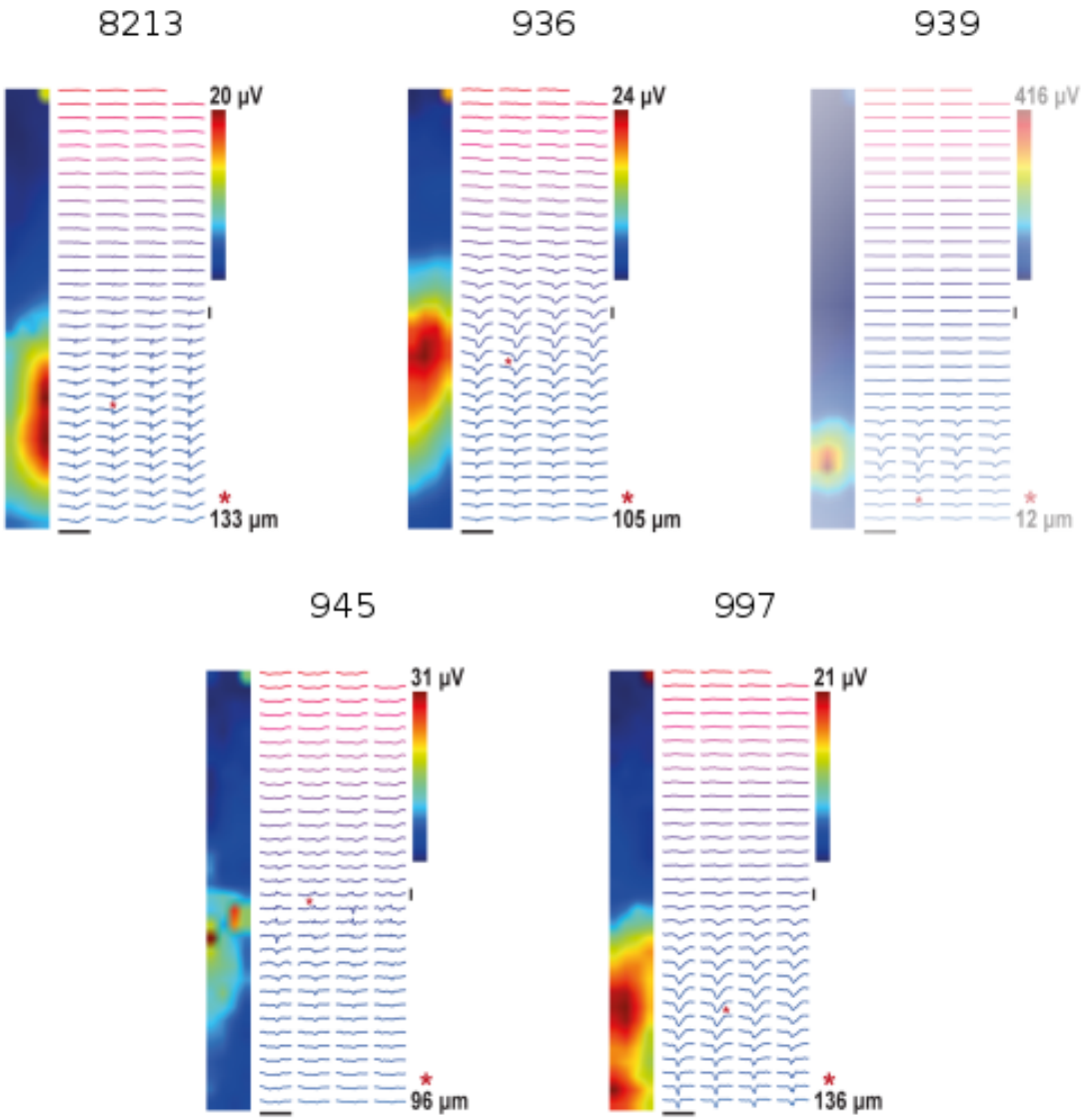


Figure 2.3: Presentation of the recording used in this project. Here are presented the spatial distribution of the peak-to-peak amplitude of the Juxta-Triggered Averages, illustrated as a interpolated heatmap. In addition, the extracellular JTA waveforms for all the extracellular electrodes are spatially arranged

Recording ID	Short ID	Distance (μm)	P2P (μV)	Depth (μm)	# Juxta spikes
2015_09_09_Pair7.0	997	136.2 $\pm$ 40	20.7	1032.8	1082
2015_09_04_Pair5.0	945	96.1 $\pm$ 40	30.8	1185.5	185
2015_09_03_Pair6.0	936	153.3 $\pm$ 40	24.1	1063.2	3329
2015_09_03_Pair9.0	939	11.5 $\pm$ 40	416.3	1152.8	5007
2015_08_21_Pair3.0	8213	132.8 $\pm$ 40	19.4	1286.0	8117

Table 2.1: Information about the recordings used. The values on the "Recording ID" are conform the dataset provided by Neto et al (2016). For convenience, a Short ID will be used throughout this document. P2P stands for Peak-to-Peak Amplitude calculated as the $\max_i \left(\max_t \left(JTA_i\right) - \min_t \left(JTA_i\right)\right), i = 0, \ldots, 127$. In the fifth column are the values of the depth in the cortex. In the last column are the number of spikes detected in the signal from the Juxtacellular pipette.

We have some variability in this ensemble. The recording 939 was recorded very closed to the neuron and therefore has a very large P2P amplitude and lies above the noise; it also recorded many spikes. The recording 945 has a very low count of spikes and relatively low P2P amplitude. For this

reason its JTA is not very well defined. Despite its low P2P amplitude, the recording 8213 is the one with the most events, making its JTA reasonable.

In Fig. 2.3, the spatiotemporal profile of each recording is noticeable and centered around the closest electrode in the probe.

2.3 Methods

2.3.1 Spike Detekt

To my knowledge (REFERENCE), klustakwik is the one that yields the best results, in reasonable human and computational time.

To deal with the problem of overlapping spikes KlustaKwik uses (user-provided) information about geometry of the probe. This information consists of the electrodes positions and the adjacency graph that defines ”neighbour” electrodes.

This method uses a Butterworth filter on the first stage. Then it uses a double-threshold detection: the user defines a ”weak threshold” and ”strong threshold”. The parts of the signal whose amplitude exceeds the weak threshold must be define a contiguous region in time as well as in space (according to the adjacency graph) and at least one data point must exceed the strong threshold. This region is called a connected component. To define this region KlustaKwik uses the flood fill algorithm (commonly used in computer vision). This approach avoids both spurious detection of noise events and prevents the same spikes from being detected more than once.

After detection, spike times are calculated as the center of mass of the signal that lies above the weak threshold for each channel. This results in several spike times therefore it is necessary to align the waveforms, i.e., shift each window

With the aligned waveforms, KlustaKwik performs Principal Component Analysis as Feature Extraction method and the three most significant components are kept. Along with this feature vector a mask vector is calculated. With this vector, to each detected event and each channel a number between zero and one is assigned: zero if the peak amplitude of the waveform sensed by that channel doesn’t reach the weak threshold and one if the peak amplitude exceeds the strong threshold. Otherwise this number is assigned according to a function of the peak amplitude, for example, a linear function.

The mask vector ensures that temporally overlapping spikes with similar feature vector are distinguishable, and treated individually.

In this new representation space, spikes are sorted with an unsupervised learning algorithm called Expectation-Maximization algorithm. This stage defines a number of putative neurons (classes) with spikes assigned to them.

Finally, a human operator inspects the results and manually sorts if necessary.

KlustaKwik requires the user to define many parameters prior to running: most significantly the weak and strong thresholds. In [Rossant et al. 2016] Rossant et al. report that the optimal values for these parameters are $\theta_w = 2\sigma_{noise}$ and $\theta_s = 4.5\sigma_{noise}$ where σ_{noise} is the standard deviation of

the noise for each channel which is estimated as the standard deviation estimator s_{n-1} of a few time windows of the filtered signal spread across the whole recording.

These values were obtained evaluating the detection performance against hybrid labeled data (data composed by several labelled dataset acquired by the same 32-channel probe).

2.3.2 phy

As of the summer of 2015, Klusta-Team made phy available. phy is a python package for python 3.4 that allows researchers to use the API from KlustaKwik in a modular way importing only what is necessary. phy can also be run as a command-line operation.

In the context of this project, before running phy it was necessary to convert the binary data from Neto et al. into a binary file with the struture and data type that phy expects to read. The binary file must be a flat array of 16-bit integer with the following structure:

$t_1C_1, t_1C_2, \ldots, t_1C_N, \ldots t_TC_1 \ldots t_TC_N$

The command "phy detect filename.prm" was used. The .prm file stores all the user-defined parameters necessary for phy to work, in particular, the weak and strong thresholds. The relevant output of this command is a .kwik file which is an HDF5 file containing the extracted PCA features and the masks for each detected spike.

2.3.3 Cross-Correlograms

To analyze neural activity in the brain, scientists often look into the temporal correlation of the recorded signals. If The correlation of the signal f and signal g is defined as:

$$(f * g)(\tau) = \int_{-\infty}^{+\infty} f(t)\, g(\tau + t)\, dt \tag{2.1}$$

where τ is called the time delay.

If the signals are uncorrelated, their correlation will appear flat. If there exists some correlation (or even causality) between the two signals their correlation will behave interestingly (not trivial), e.g., if the second neuron always spikes 1 ms after the first one, there will be a peak when $\tau = -1$.

When performing this kind of analysis, spike signals are usually represented as a sequence of times when the neuron spiked or as a time series of 0's and 1's, if a sparse representation is preferred. These are called spike trains.

This is usually done by means of calculating cross-correlograms. These are the graphical form of the cross-correlation between to signals, typically in the form of histograms.

If there exists a strong temporal correlation between the two signals, the cross-correlograms should peak when τ equals the time difference between the correlated spikes of the two cells.

Another useful calculation is the auto-correlogram which the cross-correlograms of a signal with itself. This should have a very high count when which is uninformative and usually omitted. It is used to verify if there are a significant number of events with delays smaller than the refractory period; if so, the detection of spikes or the sorting was unsuccessful.

In this document, spike trains were sequences of times at which events occurred. To calculate the cross-correlograms, the two spike trains were compared by subtracting every element of one spike train to every element of the second. Then, values of this difference that were larger than a certain value (referred to as lag) were discarded. Finally the histogram of this sequence is plotted.

AND BACK!!

2.4 Results

To be sure about the quality of the juxtacellular recording, auto-correlograms were computed for each recording (Fig. 2.4).

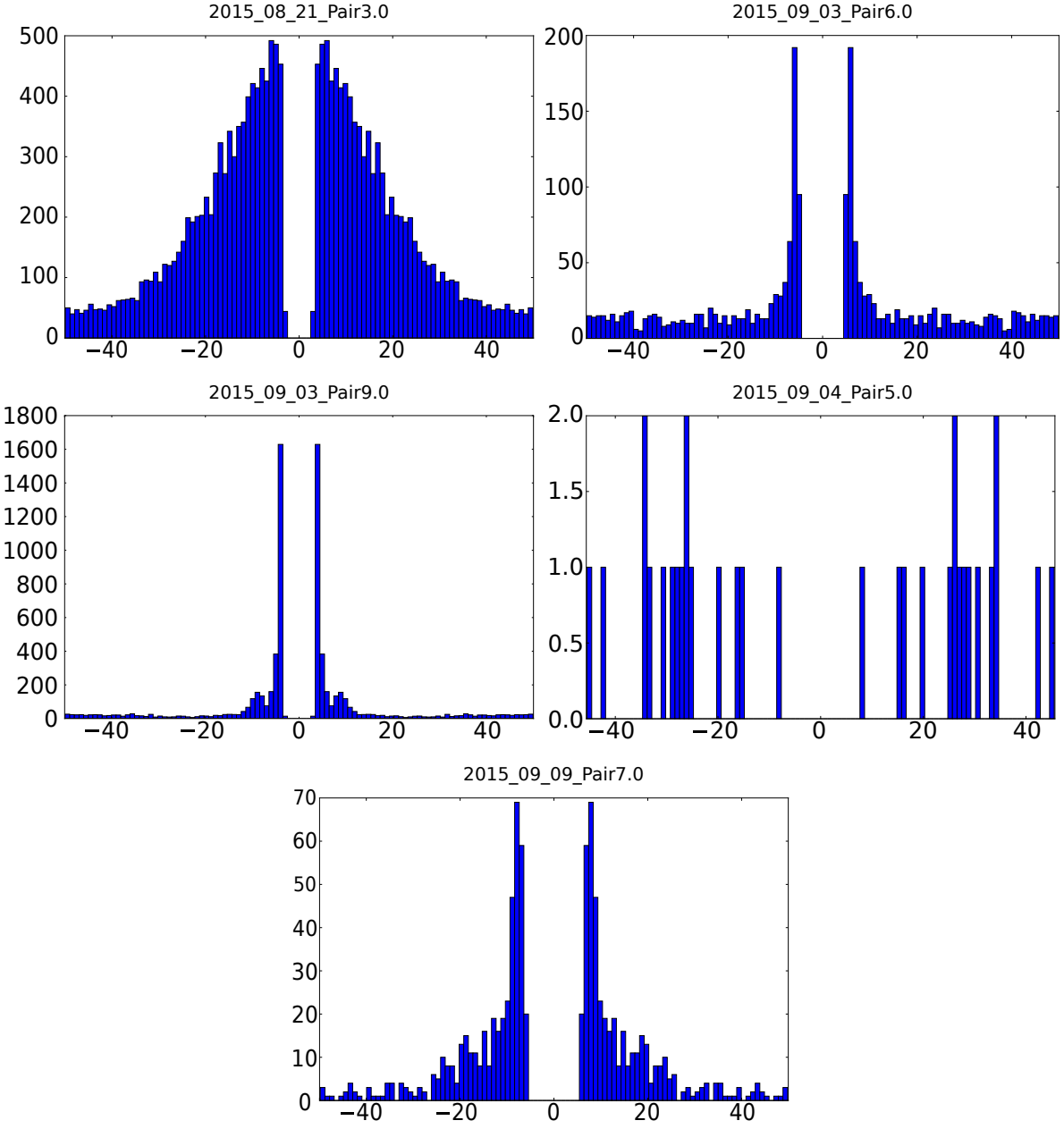


Figure 2.4: Auto-Correlograms for all the recordings. The size of the bins in the histograms is 1 ms and the value for the lag is 50ms.

All the auto-correlograms display a usual distribution, clearly showing a gap in the interval from -5ms to 5ms, assuring that the cell never spiked twice within a 5ms interval, which conforms with the typical values of refractory period of 2-3ms.

It is worth noting that on the auto-correlogram corresponding to the recording 939 a second peak is resolved around $\tau = -9ms$ and $\tau = 9ms$.

For each recording, phy was run with the following parameters: The data was filtered with a forwards-backwards Butterworth filter of order 3 with cutoff frequency set to 500Hz. The noise standard deviation was evaluated in 50 excerpts of 1 second each. The weak threshold was $\theta_w = 2\sigma_{noise}$ and the strong threshold was $\theta_s = 4.5\sigma_{noise}$.

The results are presented in table 2.2.

Recording ID	# detected Spikes	σ_{noise} (μV)	θ_W (μV)	θ_S (μV)
8213	148762	12.95	25.91	58.30
936	323629	10.76	21.52	48.43
939	265476	10.51	21.02	47.29
945	126234	10.92	21.84	49.14
997	156932	11.47	22.93	51.60

Table 2.2: Summary of the output from phy. In this table are the values of the estimated standard deviations of the noise, and the calculated weak and strong thresholds for each recording. These values were converted into μV.

In Fig. 2.5 are the whole-probe cross-correlograms for the recordings, where for each detected spike, all electrodes whose corresponding mask value was non-zero were used.

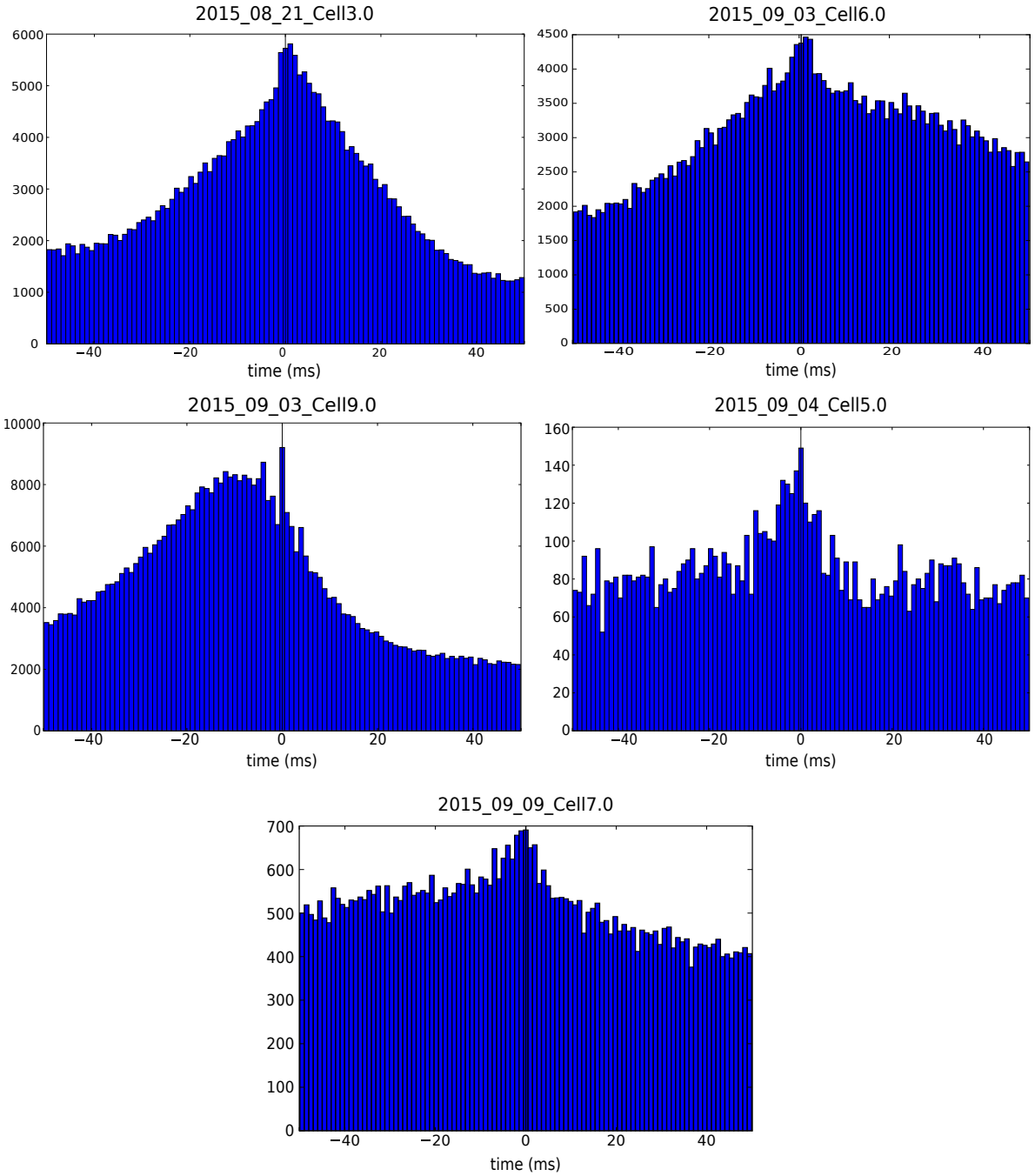


Figure 2.5: Cross-Correlograms for all the recordings. The size of the bins in the histograms is 1 ms and the value for the lag is 50ms.

In Appendix NUMBEROFAPPENDIX are the cross-correlograms per channel, where the spike train output by phy was split according to the electrode the spike was detected on using the masks.

On Fig 2.5, all cross-correlograms present a somewhat coherent distribution. This means that for every value τ in the considered interval, there exists some temporal correlation between the juxta neuron and the activity of the rest of the neurons in the recorded volume. This is to be expected. According to Ruiz-Mejias et al., the use of ketamine as anesthesia in rats provokes the synchronization in the population activity in many cortical areas, including the motor cortex. This gives rise to ”up” and ”down” states, where most neurons in the population are firing or silent, respectively. In addition, they

report that the frequency of oscillation of these states is, on average, 0.97Hz, which is close to the firing rates observed in the juxtacellular recordings from Neto et al.

Only on the cross-correlogram corresponding to the recording 939 can we see a distinct peak when $\tau = 0ms$, on top of the correlation with the background activity. This means that phy managed to find juxta neuron.

On the rest of the cross-correlograms in Fig 2.5, the peak around the central bin is never very clear. In fact, in the case of the recordings 8213 and 936, the peak is even shifted to $\tau = 1ms$. This could justify a more careful examination setting the size of the bin used in the histograms to a smaller value.

To calculate the number of events corresponding to the juxta, it is necessary to remove the counts from the correlation with the background activity. To estimate this value, the average of the counts in the bin neighboring bins ($\tau = -1ms$ and $\tau = 1ms$) was computed and subtracted to the counts in the central bin. The results are in table 2.3.

Recording ID	Bin Counts			Corrected Counts	#juxtaSpikes	Accuracy
	$\tau = 0$	$\tau = -1$	$\tau = 1$			
8213	5725	5642	5810	-1	7760	-0.01%
936	4377	4357	4465	-34	3329	-1.02%
939	9202	6701	7092	2305.5	4947	46.60%
945	144	137	120	15.5	185	8.38%
997	691	689	650	21.5	1082	1.99%

Table 2.3: Correction of the cross-correlograms central peak.

Even in the recording 939 the detection rate is fairly low, considering it has a very large P2P amplitude. In the other cases, phy yields a detection accuracy close to zero. This is not surprising considering the algorithm used by phy. The maximum P2P amplitude of JTAs of these case lies between lies between 19.4μV and 30.8μV and the strong threshold is always larger than 48.43 μV. Since it is required that at least one sample in a connected component be larger than the strong threshold these spikes are never detected. Two possible explanations exist for this number of detected events. First, the neuron may have spiked simultaneously to a large noise fluctuation causing it to be detected. Secondly, the connected components of these spikes could have been connected with the connected component of other spike which exceeded the strong threshold. This would result in the detection one single spike where the computed spike time was closer to the corresponding juxta spike time and therefore contributed to the central bin in the cross-correlograms.

2.5 Conclusion

Using five recordings from the labeled dataset acquired by Neto et al. it was possible to identify limitations in KlustaKwik. In this dataset:

blah blah blah

noise,motor cortex, neuroseeker probes, blind motorized targeting

3

A Chapter

Contents

Present the chapter content.

To address the issues presented in the previous chapter and to take advantage of the labeled paired recordings in Neto et al.,

In order to overcome the issues presented in the previous chapter, a different approach was tried. In this chapter is report the attempt of employing recent techniques of deep learning to perform spike detection.

3.1 Methods

For millennia, humans (and other animals) have tried to understand the rules that govern the phenomena surrounding them based on observations. This knowledge allowed them to expand the reach of their predictions and develop inventions to improve their way of living, as well as the empirical laws that support all the fields of fundamental science, and consequently applied science. In the last 5 decades, with the advent of several kinds of sensors, fast electronics and large storage capacity, quantitative observations have become more and more numerous at a great level of detail (the ”deluge of data”) [ref] and it appears that this way of learning is becoming more and more necessary in the present than ever before.

Datasets became very large in the number of elements as well as very high in its dimensionality (the ”Deluge of Data”), in a such way that they are no longer amenable for a human operator to analyze them. To be able to deal with this problem, researchers and engineers started using computers in particular methods of Machine Learning. Machine Learning is a subfield of computer science that studies and develops algorithms meant to find new structures or rules a given experiment or phenomenon is based on. Arthur Samuel defined machine learning as a ”Field of study that gives computers the ability to learn without being explicitly programmed” (reference Phil Simon (March 18, 2013). Too Big to Ignore: The Business Case for Big Data. Wiley. p.89. ISBN 978-1-118-63817-0.)

However, conventional machine learning techniques often require careful and domain-specific design in order to achieve good results. For example, it usually takes a considerable amount of expertise and experience with a phenomenon of interest in order to determine an algorithm and/or model that would be a good approximation of what was actually happening in the experiment under study

In the field of statistical machine learning, a dataset consists of m points (examples), each one with a set of n features defining a n-dimensional space (input space). Depending on the specific kind of model we would like to extract from the data, there are a number of different approaches for machine learning.

The most commonly used form of machine learning uses labeled datasets. This is called supervised learning.

3.1.1 Supervised Learning

In supervised learning, each input example has an extra feature that represents the “true” value, the ”right answer” we would like to obtain from the machine for this instance. If the dataset has such

features, it is said to be labeled, otherwise, it is unlabeled.

Supervised learning algorithms are usually iterative methods. They are usually presented with a large number of labeled examples beforehand, which they then use to modify the evaluation model in order to output the best possible answer [what is a model? a definição devia aparecer antes]. These modifications follow a set of operations (the training or learning algorithm), that depends on some measure of dissimilarity between the outputs of the model in its current configuration (the model predictions) and the corresponding labels. This function is usually called a loss function and is most often thought of as a distance. Examples of loss functions are the Mean-Squared Error of Cross Entropy [others?]. At each iteration, the training algorithm will calculate the necessary adjustments in order to make the loss function smaller. After a certain number of iterations, the algorithm will, possibly, converge on a solution that is a good enough approximation of the results a human supervisor would produce when analyzing the same dataset.

In machine learning, we must choose a model (or framework) to work with. In the case of linear regression, the output of the algorithm is a linear function of the form $y = mX + b$, where m and b are the adjustable parameters. This choice strongly conditions the power of the algorithm. If, for instance, y depends on X quadratically, linear regression may not yield the best results; however, depending on the situation it may provide a good enough approximation. Another choice the user must define a priori is the training algorithm. A simple example of such an algorithms is gradient descent.

In supervised learning we are not only interested in producing a model for the data we do have but more generally in predicting the outcome of the experiment in untested situations that we have yet to observe.

More formally, the computer receives a number, m, of examples of input as a (n-dimensional) feature vectors $\vec{x_i}, i = 1, 2, \ldots, m$ and corresponding (p-dimensional) target ("true") value $\vec{y_i}, i = 1, 2, \ldots, m$ and tries to find the element in the family of functions (hypothesis class) h_θ parameterized by θ such that $\vec{y_i} \approx h_\theta\left(\vec{x_i}\right)$ for each example.

3.1.1.A Linear Regression

In the context of linear regression, we restrict ourselves to a family of function such that:

$$h_{\vec{\theta},b}(\vec{x}) = \sum_{j=1}^{m} \theta_j x_j + b = \theta \cdot \vec{x} + b \tag{3.1}$$

where θ and b are the parameters to be learnt.

A common choice for the loss function is the Mean Squared Error (MSE), defined as:

$$J(\theta, b) = \frac{1}{2} \sum_{j=1}^{m} \left[h_{\theta,b}\left(\vec{x}^{(j)}\right) - y^{(j)} \right]^2 = \frac{1}{2} \sum_{j=1}^{m} \left(\theta \cdot \vec{x}^{(j)} + b - y^{(j)} \right)^2 \tag{3.2}$$

Most training algorithms require the knowledge of the gradient of the loss function with respect to the model's parameters to determine the update rule. In this situation, the gradient would be:

$$\frac{\partial J}{\partial \theta_i} = \sum_{j=1}^{m} x_i^{(j)} (h_{\theta,b}(x^{(j)}) - y^{(j)}) \tag{3.3}$$

$$\frac{\partial J}{\partial b} = \sum_{j=1}^{m} (h_{\vec{\theta},b}(x^{(j)}) - y^{(j)}) \tag{3.4}$$

And the parameters would be updated as:

$$\theta_i^{n+1} = \theta_i^n - \eta \left[\nabla_\theta J(\theta^n, b^n)\right]_i \tag{3.5}$$

$$b^{n+1} = b^n - \eta \frac{\partial J}{\partial b}(\theta^n, b^n) \tag{3.6}$$

where η is the learning rate defined by the user.

3.1.1.B Logistic Regression

Machine learning is also often used to perform a classification task where we are trying to assign a class (discrete value) to some input vector. For instance, we could apply linear regression and set a threshold value to define the boundary between two class. However this method is very sensitive to extreme values of the input. In the case of binary classification $y^{(i)}$ can only be either 1 or 0. In this situation Logistic Regression is usually a better choice. With logistic regression we try to find the predictor choosing a different hypothesis class:

$$h_{\theta,b}(x) = \sigma(\theta \cdot x + b) = \frac{1}{1 + exp(-\theta \cdot x - b)} \tag{3.7}$$

Note that this function (called the sigmoid function or logistic function) is a continuous for all values of x. It is always positive, monotonically increasing from zero to one. This leads to the interpretation of the output of the logistic regression as the probability of the class labeled as "1" happening given the input vector x:

$$P(Y = 1 \mid X = x) = \frac{1}{1 + \exp(-\theta \cdot x - b)} \leqslant 1 \tag{3.8}$$

$$P(Y = 0 \mid X = x) = 1 - P(Y = 1 \mid X = x) \tag{3.9}$$

The cost function in this case is usually defined as:

$$J(\theta, b) = -\sum_{j=1}^{m} (y^{(j)} \log(h_{\theta,b}(x^{(j)})) + (1 - y^{(j)}) \log(1 - h_{\theta,b}(x^{(j)}))) \tag{3.10}$$

Since in this setup $y^{(j)}$ can only be either 1 or 0, only one of the terms inside the summation is non-zero.

The gradient of this loss function is:

$$\nabla_{\theta,b} J(\theta, b) = \sum_{j=1}^{m} x^{(j)} (h_{\theta,b}(x^{(j)}) - y^{(j)}) \tag{3.11}$$

In classification tasks it is common to have more than two classes that we are interested in. In this case, we can generalize logistic regression to many-classes using Softmax Regression, where the probabilistic interpretation is applied

In classification task, we are looking for the boundary between classes in the feature space. The techniques I mentioned above can only resolve problems in which the classes are linearly separable (where the boundary is an hyper-plane in the feature space). But this is not always the case. Sometimes the region corresponding to a particular class may even be disjoint. In such case linear classifiers are not powerful enough to solve the problem. (As an example consider the Exclusive OR function where the inputs are two binary valued variables. There is not straight line in the input space that separates the class “0” and the class “1”.)

3.1.2 Artificial Neural Networks

A much more powerful concept is that of Artificial Neural Networks. An artificial neuron (hereafter neuron unless stated otherwise) is a computational unit that takes as input the vector x and outputs

$$h_{w,b}(x) = f\left(w \cdot x + b\right) = f\left(\sum_{i=1}^{n} w_i x_i + b\right) \tag{3.12}$$

where $f\colon \mathbf{R} \to \mathbf{R}$ is called the activation function. The vector **w** and the value of b (called the bias or intercept term), as before, can be tuned according to some algorithm to perform a designated task as good as possible.

If the activation function is the sigmoid function we recover the logistic regression.

Another example of activation function is the hyperbolic tangent, which increases from -1 to 1. Lately, researchers and engineers have started using the rectified linear function (RELU) particularly in the context of deep neural network (which I’ll talker later in this document). This function is defined as

$$RELU(z) = \max(z, 0) = \begin{cases} z, & \text{if } z \geq 0 \\ 0, & \text{if } z < 0 \end{cases} \tag{3.13}$$

This activation function is significantly different from the ones referred before because it is not bounded as z increases. Moreover, it is not differentiable when $z = 0$ although this doesn’t become a problem in practice because it is differentiable at any point arbitrarily close to 0.

A Artificial Neural Network (ANN) is put together by hooking together many of these simple neurons by means of function composition where the output of one neuron is the input of another.

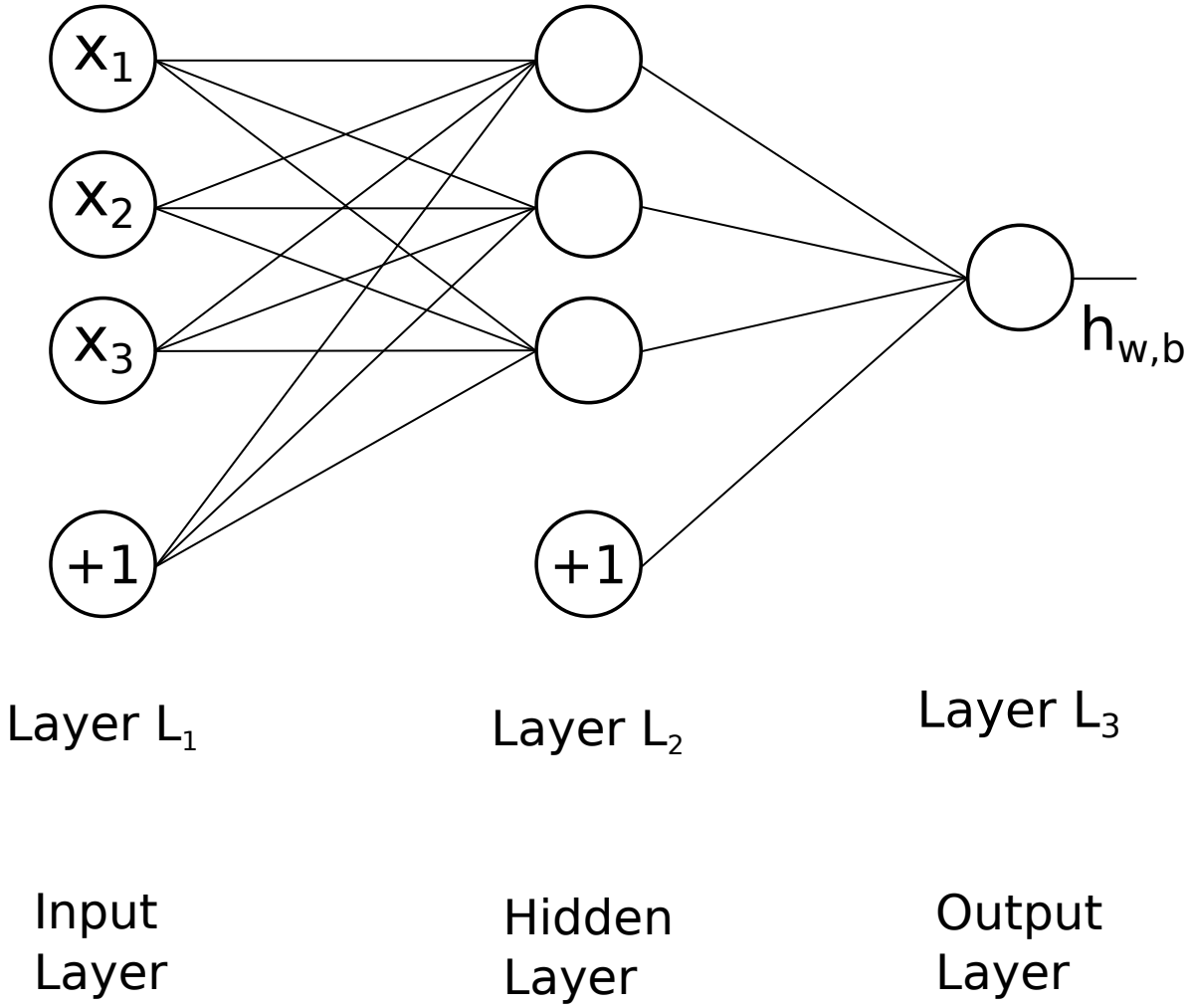


Figure 3.1: Graphical visualization of an illustrative example of an artificial neural network. This ANN is composed by three layers with three neurons on the first two and one neuron on the output layer. The connections between all the neurons are also represented, in addition to the connection to the bias term represented by the extra nodes on the bottom with the label "+1".

In the Fig. 3.1 is a graphical representation of what was just mentioned. On the left, we have the input layer (layer L_1) where the input vector (x_1, x_2, x_3) is fed. In the middle there are three neurons. These are called hidden neurons and they compose one hidden layer (L_2). Finally, there is the output layer with only one neuron. There are also two nodes on the bottom that represent the bias term.

In this example, there are connections between all neurons of one layer to the neurons in the next layer. This ANN is said to be fully connected (or densely connected). For each connection there is an associated parameter (weight) and also a bias parameter. The input of each neurons in the hidden layer is a linear combination of the output of the neurons in the previous layer weighted by the parameters of the corresponding connections and summed with the bias term. In this case, the weights for the connections coming in to the first neuron in the layer L2 are $w_{1,1}^{(1)}, w_{1,2}^{(1)}, w_{1,3}^{(1)}, b_1^{(1)}$ and its input is:

$$z_1^{(2)} = \sum_{i=1}^{3} w_{1,i}^{(1)} a_i^{(1)} + b_1^{(1)} \tag{3.14}$$

This input is then fed into the activation function which reveals the outputs of this neuron as

$$a_1^{(2)} = f\left(z_1^{(2)}\right) \tag{3.15}$$

We denoted the number of layer as n_l and the number of neurons in the layer L_l as S_l. In the example above $n_l = 3$, $S_1 = S_2 = 3$ and $S_3 = 1$

In general, when the network is densely connected, the input of the neuron i in the layer j is:

$$z_i^{(j)} = \sum_{k=1}^{S_{l-1}} w_{i,k}^{(j)} a_k^{(j-1)} + b_i^{(j-1)} \tag{3.16}$$

And its output is:

$$a_i^{(j)} = f(z_i^{(j)}) \tag{3.17}$$

It is also convenient to introduce a matrix notation such that:

$$\mathbf{z}^{(j)} = \mathbf{W}^{(j)}\mathbf{a}^{(j-1)} + \mathbf{b}^{(j-1)} \tag{3.18}$$

where the vector $\mathbf{z}^{(j)}$, $\mathbf{a}^{(j)}$ and $\mathbf{b}^{(j)}$ represent the input, output and bias parameter of all neurons in Layer L_j

The output of the output layer is denoted $\mathbf{h}_{\mathbf{W},\mathbf{b}}(\cdot) = \mathbf{a}^{(L)}(\cdot)$

Using Artificial Neural Networks, we now have a much more powerful and flexible framework to create models that can be trained to compute much more complex functions than the ones computable with traditional machine learning algorithms.

However, there's still the need to define the training algorithm. In order to use the gradient descent algorithm, it is necessary to compute the gradients of the loss function with respect to all the adjustable parameter of ANN. These gradients are usually computed using the back-propagation algorithm. In this method, the label is subtracted from the output value as an estimate of the gradient on the output layer. This value is then propagated backwards layer by layer by applying the chain rule for derivatives, which will depend on the chosen activation functions. With the gradients estimated, the parameters are update in the way defined earlier in equation 3.6

The key aspect of learning with ANN is that, due to its generalization power, the form of the final output function is not directly designed by a human: they are learned from the data.

But this framework leads to questions of how exactly to define the model. In other words, how to define the architecture of the ANN, i.e., how many neurons should there be in the ANN and how should they be connected? There are two basic topologies: shallow networks and deep networks. In shallow networks, there is at most one hidden layer, whereas a network is said to be deep if it has two or more hidden layers. This distinction has become very important over the last three decades. On the one hand, it has been shown that a shallow ANN with only one arbitrarily large hidden layer could approximate a function to any level of precision (REFERENCE Horniket al.,1989). Nonetheless, this level of precision would only increase with exponentially increasing number of neurons, becoming computationally very demanding.

On the other hand, deep neural networks are conceptually more interesting because each layer can be thought of as a representation of the input in a higher and higher level of abstraction, similar

to how the visual processing hierarchy in cortex is thought to operate to construct human visual perception. However, using back-propagation and gradient descent with the usual sigmoid or tanh function can very quickly run into the problem of “vanishing gradients”, when the activation function saturates and the training will not proceed any further. Moreover, even in the cases where training is possible, deep networks were originally found to perform worse than shallow networks.

However, in the past decade there have been several theoretical and technological advances that brought deep neural network back to life.

3.1.3 Deep Learning

Representation learning is a family of methods that allows machines to find new ways of representing the raw data it was fed with. Deep Learning tries to accomplish this in a ”layer by layer” manner. In a deep neural network, each layer holds a new representation of the input data, by transforming the output of the previous layer into a new representation, a more abstract way of perceiving the input data. Composing many of these layer, it should be possible to compute very complex function. For the case of classification task, higher layers of representation may amplify aspects of the input that are important for the discrimination and suppress irrelevant features.

In this section, the techniques used in this project will presented such as training (or optimization), loss function, initialization methods and regularization

3.1.3.A Stochastic Gradient Descent

Stochastic Gradient Descent (SGD) is a stochastic approximation of the gradient descent algorithm described in the previous section. In SGD, only utilizes a subset (called a batch) of the provided examples to compute the gradient. This ”noisy” approximation of the gradient is then used to update the parameters of model. It should be noted that all the examples in the dataset are utilized: the entire dataset is split in batches and for each batch there is one update in the parameters. One pass through the entire dataset is called an epoch. This means that there will be more (but faster) iteration than in the standard gradient descent. Surprisingly, this simpler method is known to yield good results much faster than more sophisticated algorithms.

Stochastic learning is often much faster than classic learning particularly when using large redundant datasets: if, for instance, the dataset is composed of the ten repetitions of a smaller set of examples, then estimating the gradient using the whole dataset would have the same result as using one tenth of the dataset. Of course in practice two examples are rarely the same. However many example may be acquisition of the same pattern and therefore will contain approximately the same amount of information.

Networks learn the fastest from examples that are most distant from what the network predicted. Therefore it would preferable to choose such examples in each iteration. Of course, there is no simple way to know which are the ”good” examples to train the network with in each iteration. There is a relatively simple way of applying this idea. Assuming successive examples do not differ much,

shuffling the dataset before splitting would make the batches ”richer” in terms of the information they contain.

Another interesting characteristic of stochastic learning is that it is less prone to get the network stuck in local minima of the loss function. The noise introduced in the gradient estimation generates updates on the parameters that allow easier ”jumps” of the parameters from one local minima to another, possibly deeper than the previous. On the other hand, this noise also prevents the full convergence to the minimum: the parameter will always have stochastic fluctuation. This can be address by adaptively changing the size of the batch or the learning rate.

3.1.3.B AdaGrad

One solution for the problem just mentioned is AdaGrad (Adaptive Gradient optimization), proposed in 2011 by Duchi et al. REFERENCE. In this algorithm, the learning rate is adapted in each iteration according to the geometry of the loss function in the vicinity of the current values of the parameters. For each element in the parameter matrix, the user-defined learning rate η is weighted by factor that makes it larger or smaller according to a (fast) approximation of the Hessian Matrix. In other words, the update for each parameter is more coarse when far from a local minimum and finer when close to a local minimum.

It should be noted that this algorithm was proposed for convex optimization problems. However, even in non-convex situations AdaGrad usually performs better that standard SGD. (REFERENCE Gupta, Maya R.; Bengio, Samy; Weston, Jason (2014). ”Training highly multiclass classifiers” (PDF). JMLR 15 (1): 1461–1492.)

3.1.3.C Parameter Initialization

The starting values of the weights can have a significant effect on the training process. For instance, in an ANN with the hyperbolic tangent as activation function, if all parameters were initialized with the same value all neurons would output the same value and get the same updates during training. This would render the network useless. For this reason it is necessary to break these symmetries from the onset.

To get the most out a certain ANN, the initial parameters should be as uncorrelated as possible. However, if the weights are initialized with very high values, a sigmoid or tanh activation function would start saturated, gradients would vanish and the training wouldn't be possible. To solve this problem, the initial parameters are usually sampled from a uniform distribution $U([-a, a])$, where a is some small value. There are several proposals for determining the value of a depending on the particular model chosen by the user: He Uniform, Xavier Uniform(also known as Glorot Uniform) or LeCun Uniform. REFERENCES

For example, LeCun Uniform was defined with sigmoid activation functions in mind. With this initialization method, starting values for the weights are drawn from a uniform distribution with zero mean and standard deviation such that the input of each neuron has a standard deviation close to 1.

3.1.3.D Loss Function

MSE

cross entropy

3.1.3.E Regularization

L1 and L2 regularization

Dropout

4

Conclusions and Future Work

Conclusions Chapter

Title of AppendixA