Data Compression of Neural Spike Signals

DISSERTATION

zur Erlangung des akademischen Grades eines

DOKTOR-INGENIEURS
(Dr.-Ing.)

der Fakultät für Ingenieurwissenschaften, Informatik und Psychologie der Universität Ulm

von

Matteo Pagin
aus Venezia

Gutacher: Prof. Dr.-Ing. Maurits Ortmanns
Prof. Dr.-Ing. Matthias Kuhl
Amtierender Dekan: Prof. Dr.-Ing. Maurits Ortmanns

February 15, 2021
Declaration of Authorship

I, Matteo Pagin, declare that this thesis titled, “Data Compression of Neural Spike Signals” and the work presented in it are my own. I confirm that:

- This work was done wholly or mainly while in candidature for a research degree at this University.

- Where any part of this thesis has previously been submitted for a degree or any other qualification at this University or any other institution, this has been clearly stated.

- Where I have consulted the published work of others, this is always clearly attributed.

- Where I have quoted from the work of others, the source is always given. With the exception of such quotations, this thesis is entirely my own work.

- I have acknowledged all main sources of help.

- Where the thesis is based on work done by myself jointly with others, I have made clear exactly what was done by others and what I have contributed myself.

Signed: 

Date: 
“A few observation and much reasoning lead to error; many observations and a little reasoning to truth.”

Alexis Carrel
In this thesis an analysis of various compression methods for neural data is carried out. Reading the electrical activity from neurons is a crucial step to study the working principles of the brain, treat neuronal disorders and directly control prosthetic limbs. Independently from the application the more neurons are covered in the recording the better. In the past years brain machine interfaces have seen an increase in the maximum number of recording channels to satisfy this need. Wiring the recording sites through the skull creates a high possibility of inflammation, wireless data transmission is then beneficial in this regard. But in order not to damage surrounding neural tissue, implantable recorders have limited power available which makes wireless data transmission even more challenging. To combine the need for a high channel count and safety of the surrounding tissue compression becomes critical in implantable neural recorders. For all compression methods a trade-off between algorithm complexity, information loss and data reduction has to be evaluated. In this work in-depth analysis of how compression affects neural signal processing is carried out. Publicly available state of the art software for spike sorting is used to determine the effects of compression. After evaluating the minimum signal resolution, more sophisticated compression methods are analyzed. Predictor based compression schemes with a focus on lossless compression are compared with lossy compressed sensing. In the analysis machine learning elements are used to enhance compression or carry out elaboration of neural signals. This work uses simulated data as well as recorded data to provide the most realistic scenario as possible. At the end hardware evaluation of different methods is carried out to provide insight on which method is the most suitable to be implemented, depending on the available technology. From the analysis it turns out that when compressing whole neural signals predictive encoding achieves the best performance providing lossless compression. Out of the examined predictors linear neural networks offer the best compression but their hardware requirements might be too demanding for implantable devices. In this case delta compression used as a lossy compressor can offer comparable data reduction at lower hardware cost. When spikes are extracted on-chip a novel approach using autoencoders is shown to achieve high accuracy in sorting for very high compression while offering a low power implementation.
Acknowledgements

A PhD is extremely tough at times and very demanding, the writing of this dissertation would have not been possible without all the people who, in various ways, supported and guided me during the journey.

A huge thank you to my supervisor Prof. Maurits Ortmanns for opening his lab and allowing me to be part of it. His guidance and invaluable advice accompanied me during this journey making me grow in the academic career and outside of it. I’m very grateful for all that I have learned.

I would like to thank Prof. Matthias Kuhl for making time out of his schedule to be my second examiner for this dissertation.

Maintaining a working infrastructure to allow the PhD students to focus on their dissertations requires a lot of work and often goes unrecognised: thanks to Joachim Becker for helping with the synthesis of digital circuits and his housekeeping work, Markus Prokein for helping with all pc related issues, Doris Wanicki and Simone Litzinger for their patience and help with bureaucracy.

Thanks to all my colleagues who made a wonderful family both at the university and outside. In particular Patrick for being the coolest office mate, having good taste in music and being a very good friend. Thank you to Andy, Mokhtar, Jiazuo and Johannes for the travels, bike trips and wonderful moments together. I would have not survived the mensa if I had not had the "late lunch group" people as company.

I want to thank my friends outside the university for keeping life interesting and being there during happy and difficult times alike. Thank you to my friends in Ulm with whom I’ve shared many experiences and beautiful days and that made life much better, and my Italian friends for being there despite the distance. Finally a huge thank to my family for being always so caring, loving and supportive, I owe to them my education and this very opportunity I had to pursue a PhD abroad.
Contents

Declaration of Authorship iii
Abstract vii
Acknowledgements ix

1 Introduction 1
  1.1 Brain Machine Interfaces ........................................... 1
    1.1.1 How Neurons Communicate .................................. 2
  1.2 Different Recorders, Different Signals and Applications ............ 4
  1.3 Thesis Outline ..................................................... 6

2 Background and Motivation 7
  2.1 State of the Art of Neuronal Recorders ......................... 7
    2.1.1 In-vitro and In-vivo Measurements .......................... 9
  2.2 Neural Signal Processing ........................................ 9
    2.2.1 Filtering .................................................. 13
    2.2.2 Spike Detection ........................................... 13
    2.2.3 Feature Extraction ......................................... 14
    2.2.4 Clustering ................................................ 16
    2.2.5 Spike Sorting Summary ..................................... 16
  2.3 The Need for Compression ...................................... 17
  2.4 State of the Art for Compression Schemes ...................... 20
    2.4.1 Spike Detection On Chip .................................... 20
    2.4.2 Compressed Sensing ......................................... 21
    2.4.3 Delta Compression Schemes .................................. 21

3 Methods and Data 23
  3.1 Comparison of Spike Sorting Results ........................... 23
    3.1.1 Metrics Used for Comparison ............................... 26
  3.2 Spike Sorting Programs ......................................... 27
    3.2.1 Osort ..................................................... 28
    3.2.2 Waveclus ................................................ 28
  3.3 Used Neural Data Sets .......................................... 29
    3.3.1 Simulated Data Sets ....................................... 29
    3.3.2 Recorded Data ............................................ 30

4 Quantization for Neural Signals 33
  4.1 Resolution Requirements for Neural Signals .................... 33
  4.2 Logarithmic Quantization ..................................... 36
  4.3 Non-uniform Quantization ................................... 39
    4.3.1 Non-uniform Quantization Architectures .................. 40
    4.3.2 Non-uniform Quantization Effectiveness ................. 42
5 Compression Schemes for Neural Signals

5.1 Entropy Coding

5.1.1 Huffman Encoding

5.1.2 Arithmetic Encoding

5.1.3 Prediction by Partial Matching

5.1.4 Entropy Encoding of Neural Spikes

5.2 Differential Encoding

5.2.1 Delta Compression

5.2.2 Slope Predictor

5.2.3 Linear Neural Network

5.3 Compressed Sensing

5.3.1 Compressed Sensing for Neural Signals

5.4 Machine Learning for Data Compression

5.4.1 The Perceptron

5.4.2 Training and Feed-forward Operation

5.5 Autoencoders

5.5.1 Autoencoders for Spike Frames Compression

5.5.2 Autoencoders for Classification

5.6 Summary

6 Comparison of Compression Schemes for Neural Signals

6.1 Comparison of Lossless Compression Using Predictor Schemes

6.2 Comparison of Lossy Compression Schemes

6.2.1 Compression of Spike Frames

6.2.2 Compression of Unframed Neural Signals

6.3 Comparison of Autoencoders for Compression and Classification

6.3.1 Compression of Spike Frames with Autoencoders

6.3.2 Spikes Classification with Autoencoders

6.4 Summary Evaluation of Compression Schemes

7 Hardware Evaluation

7.1 Hardware Requirements

7.2 Feasibility of Hardware Implemented Linear Predictors

7.2.1 Implementation of Delta Compression

7.2.2 Implementation of a Linear Neural Network

7.3 Feasibility of Hardware Implemented Autoencoders

7.4 FPGA Proof of Concept

8 Conclusion

8.1 Outlook

Bibliography

List of Author’s Publications
List of Figures

1.1 Scheme of a BMI where an implantable unit, with possibly more shanks records neuronal activity in the brain. ......................... 2
1.2 Conceptual scheme of a neuron and its axon for communication with other neuron cells. .................. 3
1.3 Typical shape of an action potential showing main phases and characteristics. .......................... 4

2.1 Number of channels against year for SoA neural implants. Implants are divided in 3 categories indicating their application and transmission method. ......................... 8
2.2 Two architectural approach to record neural signals using one shared ADC or multiple ADCs. ........... 10
2.3 Spike sorting scheme from data recorded in anesthetized mice showing the main steps for spike detection and classification. .......................... 12
2.4 SoA for Tx on BMI implants, categorized depending on their application and measurement case. ........ 18

3.1 Compression scheme for neural signals showing different approaches for spike sorting and compression with respective way of comparing compression results. .......................... 24

4.1 Average CA for linear quantization by varying the number of bits on simulated data sets in [55] and for recorded datasets hc-1 and hc-2 [9, 126]. .............................. 34
4.2 Plots of neural signals showing the different noise levels between low noise simulated data Easy1 (top left) and its high noise counter part (top right) from [55]. In the bottom part low noise recorded data from hc-1 [9] (left) and high noise recorded data from hc-2 [126] (right). .......................... 35
4.3 Quantization of a Spike for a 4 bits linear ADC (top) and 4 bits logarithmic ADC (middle), and the absolute value for the respective quantization errors (bottom). .......................... 37
4.4 LSB voltages normalized to \(1V_{fs}\) for the corresponding resolution are reported for both linear and logarithmic ADCs .......................... 37
4.5 Average CA for logarithmic quantization by varying the number of bits on simulated data sets in [55] and for recorded datasets hc-1 and hc-2 [9, 126]. .............................. 38
4.6 CA comparison: difference between average CA for logarithmic ADC and linear ADC is plotted for different resolutions .......................... 39
4.7 Illustration of threshold based windowing with fine quantization inside the high resolution window, and coarse quantization outside .......................... 40
4.8 Block diagrams to realize non-uniform quantization for neural signals .......................... 41
PCAS (a) and CR (b) for simulated data in [55] with different thresholds (1..5σ) over different number of bits (2..8) for background noise and 16 bits resolution for spikes ................................................. 43

Huffman tree construction steps for Example 5.1.1 ............................................. 47

Average spike shapes from simulated data and various spikes showing their similarities in shape ................................................................. 50

Average spike shapes for simulated data in Easy1 dataset with bars indicating one std. ................................................................. 51

Compression of spikes using PPM and arithmetic encoding .................................... 52

Tetrode recording of a spike event from data in hc-1 [9] ........................................... 53

Auto-correlation (AC) for one channel a), normalized cross-correlation (CC) for channels 1 and 2 b), normalized cross-correlation function for channels 1 and 8 c) ........................................ 54

Compression schemes using a predictor and Huffman encoder with and without a threshold block to improve compression .................................. 55

Comparison of neural signal \( x_i(n) \) and the error to be transmitted \( e(n) \) .................. 56

\( X_i \): quantized sample of channel \( i \). \( \hat{X}_i \): previous quantized sample of channel \( i \), with \( i = 1, \ldots, N \) where \( N \) is the number of channels ........................................ 57

Linear neural network layer without bias used as predictor, \( x_1(n), \ldots, x_N(n) \) is the serialized data from \( N \) channels sampled at time \( n \) ........................................ 59

Illustration of input selection for different operation modes of the LNN .................. 60

CR plotted against number of taps in the LNN using only temporal information (LNNT) ................................................................. 61

Diagram of a perceptron ........................................................................ 64

Various activation functions used in ANN, the x-axis represent the inputs to the functions while the y-axis are the outputs values ................................................. 65

Example of AE with 5 neurons at the input layer and 3 neurons in the hidden layer ................................................................. 68

a) Original Spikes from Easy1 low noise given as input to a b) AE output spikes with 2 neurons in the middle layer, c) AE output spikes with 1 neurons in the middle layer ........................................ 70

Linear predictors CR lossless comparison plot on tetrode data from low-noise recorded datasets in hc-1 for different number of memory depth (taps) of the LNN ................................................................. 76

Linear predictors CR for lossless comparison plot on tetrode data from dataset in hc-2 for different number of memory depth (taps) of the LNN ................................................................. 77

Linear predictors CR lossless comparison plot on tetrode data from in-vitro recordings of retinal ganglion cells for different number of memory depth (taps) of the LNN ................................................................. 78

Lossless CR achieved by LNNT on 4225 channels from in-vitro recordings of retinal ganglion cells ................................................................. 79

Classification Accuracy (CA) for delta compression and CS (SDNCS) for extracted spike frames from high noise simulated data Easy1_noise01 for 3 clusters ........................................................................................................ 80

Classification Accuracy (CA) for delta compression and CS (SDNCS) for extracted spike frames from high noise simulated data Easy1_noise035 for 3 clusters ........................................................................................................ 81
<table>
<thead>
<tr>
<th>Section</th>
<th>Title</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>6.7</td>
<td>Comparisons of different lossy compression schemes for data in hc-1 [9] when compressing the whole neural signal. CA is obtained by analyzing the data with Osort [65].</td>
<td>82</td>
</tr>
<tr>
<td>6.8</td>
<td>Comparisons of different lossy compression schemes for data in hc-1 [9] when compressing the whole neural signal. CA is obtained by analyzing the data with Waveclus [55].</td>
<td>83</td>
</tr>
<tr>
<td>6.9</td>
<td>Comparisons of different lossy compression schemes for data in hc-2 [126] when compressing the whole neural signal. CA is obtained by analyzing the data with Osort [65].</td>
<td>84</td>
</tr>
<tr>
<td>6.10</td>
<td>Comparisons of different lossy compression schemes for data in hc-2 [126] when compressing the whole neural signal. CA is obtained by analyzing the data with Waveclus [55].</td>
<td>85</td>
</tr>
<tr>
<td>6.11</td>
<td>CA of spike frames with Waveclus as reference sorting for all simulated datasets compressed using different number of neurons in a non-deep AE and deep AE.</td>
<td>86</td>
</tr>
<tr>
<td>6.12</td>
<td>CA on all simulated data sets using AE for direct classification of spike frames vs. the ground truth of the simulated data sets.</td>
<td>88</td>
</tr>
<tr>
<td>6.13</td>
<td>CA on all simulated datasets using AE for direct classification of spike frames without full-precision floating point and hidden layer output quantization.</td>
<td>89</td>
</tr>
<tr>
<td>7.1</td>
<td>Transmitter and receiver scheme, including compression and decompression</td>
<td>93</td>
</tr>
<tr>
<td>7.2</td>
<td>Effect on CR when varying the number of bits of the coefficients for LNNT predictor.</td>
<td>97</td>
</tr>
<tr>
<td>7.3</td>
<td>FPGA prototype utilization summary and power report.</td>
<td>101</td>
</tr>
</tbody>
</table>
List of Tables

<table>
<thead>
<tr>
<th>Table</th>
<th>Title</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>2.1</td>
<td>LFP and spike signals electrical properties [3]</td>
<td>11</td>
</tr>
<tr>
<td>2.2</td>
<td>Overview of different spike sorting methods</td>
<td>15</td>
</tr>
<tr>
<td>6.1</td>
<td>Summary of compression methods in literature and presented in this work</td>
<td>91</td>
</tr>
<tr>
<td>7.1</td>
<td>State of the art specifications of transmitters used in BMIs</td>
<td>95</td>
</tr>
</tbody>
</table>
## List of Abbreviations

<table>
<thead>
<tr>
<th>Abbreviation</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>AE</td>
<td>AutoEncoder</td>
</tr>
<tr>
<td>AFE</td>
<td>Analog Front End</td>
</tr>
<tr>
<td>ANN</td>
<td>Artificial Neural Network</td>
</tr>
<tr>
<td>AP</td>
<td>Action Potential</td>
</tr>
<tr>
<td>ADC</td>
<td>Analog to Digital Converter</td>
</tr>
<tr>
<td>BMI</td>
<td>Brain Machine Interface</td>
</tr>
<tr>
<td>BCI</td>
<td>Brain Computer Interface</td>
</tr>
<tr>
<td>CA</td>
<td>Classification Accuracy</td>
</tr>
<tr>
<td>CR</td>
<td>Compress Ratio</td>
</tr>
<tr>
<td>CS</td>
<td>Compressed Sensing</td>
</tr>
<tr>
<td>DWT</td>
<td>Discrete Wavelet Transform</td>
</tr>
<tr>
<td>FoM</td>
<td>Figure of Merit</td>
</tr>
<tr>
<td>FPGA</td>
<td>Field Programmable Gate Array</td>
</tr>
<tr>
<td>IDC</td>
<td>Interchannel Delta Compression</td>
</tr>
<tr>
<td>MEA</td>
<td>Multielectrode Array</td>
</tr>
<tr>
<td>ML</td>
<td>Machine Learning</td>
</tr>
<tr>
<td>LFP</td>
<td>Local Field Potential</td>
</tr>
<tr>
<td>LNN</td>
<td>Linear Neural Network</td>
</tr>
<tr>
<td>LNNS</td>
<td>Linear Neural Network Spatial</td>
</tr>
<tr>
<td>LNNST</td>
<td>Linear Neural Network Spatial Temporal</td>
</tr>
<tr>
<td>LNNST</td>
<td>Linear Neural Network Temporal</td>
</tr>
<tr>
<td>LSTM</td>
<td>Long Short Term Memory</td>
</tr>
<tr>
<td>NEO</td>
<td>Non-linear Energy Operator</td>
</tr>
<tr>
<td>PCA</td>
<td>Principal Component Analysis</td>
</tr>
<tr>
<td>PCAS</td>
<td>Percentage of Correctly Assigned Spikes</td>
</tr>
<tr>
<td>pdf</td>
<td>probability density function</td>
</tr>
<tr>
<td>PPM</td>
<td>Prediction by Partial Matching</td>
</tr>
<tr>
<td>ReLU</td>
<td>Rectified Linear Unit</td>
</tr>
<tr>
<td>Rx</td>
<td>Receiver</td>
</tr>
<tr>
<td>RNN</td>
<td>Recurrent Neural Network</td>
</tr>
<tr>
<td>SDC</td>
<td>Standard (intrachannel) Delta Compression</td>
</tr>
<tr>
<td>SDNCS</td>
<td>Signal Dependent Neural Compressed Sensing</td>
</tr>
<tr>
<td>SDNCS-P</td>
<td>Signal Dependent Neural Compressed Sensing approach with Prior recovery</td>
</tr>
<tr>
<td>SoA</td>
<td>State of the Art</td>
</tr>
<tr>
<td>SoC</td>
<td>System on Chip</td>
</tr>
<tr>
<td>std</td>
<td>standard deviation</td>
</tr>
<tr>
<td>Tx</td>
<td>Transmitter</td>
</tr>
<tr>
<td>UWB</td>
<td>Ultra Wide Band</td>
</tr>
</tbody>
</table>
## List of Symbols

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\alpha$</td>
<td>Switching activity factor</td>
</tr>
<tr>
<td>$\sigma_n$</td>
<td>Noise standard deviation</td>
</tr>
<tr>
<td>CA</td>
<td>Classification accuracy</td>
</tr>
<tr>
<td>$C_L$</td>
<td>Capacitive load</td>
</tr>
<tr>
<td>CR</td>
<td>Compress ratio</td>
</tr>
<tr>
<td>$D$</td>
<td>Datastream</td>
</tr>
<tr>
<td>$D_c$</td>
<td>Compressed datastream</td>
</tr>
<tr>
<td>$e(n)$</td>
<td>Error signal at sample $n$</td>
</tr>
<tr>
<td>$E_b$</td>
<td>Energy efficiency per bit</td>
</tr>
<tr>
<td>$E_{Cx}$</td>
<td>Energy efficiency at the compressor</td>
</tr>
<tr>
<td>$E_{Tx}$</td>
<td>Energy efficiency at transmitter</td>
</tr>
<tr>
<td>$f_s$</td>
<td>Sampling frequency</td>
</tr>
<tr>
<td>$N$</td>
<td>Number of samples</td>
</tr>
<tr>
<td>$N_b$</td>
<td>Number of bits per sample</td>
</tr>
<tr>
<td>$N_c$</td>
<td>Number of channels</td>
</tr>
<tr>
<td>$N_o$</td>
<td>Number of bits of uncompressed data</td>
</tr>
<tr>
<td>$M$</td>
<td>Number of measurements for compressed sensing</td>
</tr>
<tr>
<td>$P$</td>
<td>Electrical power</td>
</tr>
<tr>
<td>$P_{dyn}$</td>
<td>Dynamic electrical power</td>
</tr>
<tr>
<td>$V_{DD}$</td>
<td>Voltage supply</td>
</tr>
<tr>
<td>$w_i$</td>
<td>$i$-th weight</td>
</tr>
<tr>
<td>$x$</td>
<td>Input neural signal</td>
</tr>
<tr>
<td>$\hat{x}$</td>
<td>Estimated neural signal</td>
</tr>
</tbody>
</table>
A mia mamma e mio papà
Chapter 1

Introduction

In the last decades a lot of research effort has gone into developing brain machine interfaces (BMIs), also referred to as brain computer interfaces (BCI). These neural interfaces allow neuroscientists to access the inner mechanisms of our brains and shed light on how information is processed. The knowledge acquired also offers very practical applications: by accessing the electrical activity, implants can be used to suppress an incoming seizure or allow paralyzed people to move a robotic prosthetic limb as if it were their own, just with thoughts. In this thesis the problem of compressing and sending neural data to the outside world is addressed. In this very first chapter an overview of the problem is provided to better understand the challenges and requirements for a BMI.

1.1 Brain Machine Interfaces

A typical BMI is shown in Fig. 1.1, the interface consists of two blocks: an implanted side and a non-implanted side. The implanted side is where electrodes are inserted into the brain tissue to record neuronal activity. Close to the electrodes the analog front end (AFE) and recording electronics is so placed. To avoid inflammation and path of easy infections this unit should be also implanted, especially for chronic use.

Depending on the architecture, the non-implanted device can be wearable and carried on the body directly and usually has some further signal processing and transmission blocks incorporated, or alternatively it can be a workstation directly receiving the processed information from neural signals, hence the whole data processing and transmission is done directly on-chip. Many implants and recording methods are available hence the categories are often blurred and mixed up together. What makes a huge difference for the electrical specification is whether a certain function of the BMI is implanted into biological tissue or not.
1.1.1 How Neurons Communicate

What is of interest to neuroscience is how neurons organize themselves into neural networks and process information by observing the biological processes neurons use to communicate and connect with each other \[1-4\]. In Fig. 1.2 a neuron is depicted, the nucleus is surrounded by the cell membrane and is connected to an axon. The axon provides the meaning for communication between neurons, every neuron receives signals from others through their axons, and also from and to organs and muscles. Neural activity is similar to digital communication in the sense that there is either a signal or there is none (all-or-none) \[4\]. When inputs to a neuron reach a certain threshold the neuron will fire a so called Action Potential (AP) or spike \[4\]. This consists in a sudden change of voltage across the cell membrane that propagates across the axon to the other connected neurons. Since the axons can be quite long, e.g. in the spinal cord neurons can extend to a meter of length, the axon provides an active electrical transmission system, where the spikes are regenerated along its length. Other than allowing for far electrical communication, this suppresses all the spurious electrical activity that might arise.

In resting state neurons present an inner cell potential with typical values ranging from \(-40mV\) to \(-80mV\) depending on the cell type and the organism \[4\]. The concentration of negatively charged ions is kept higher inside the cell by ion pumps in its membrane \[6\]. When this potential is altered, by endogenous or exogenous current stimuli, there are two possible situations arising. If the potential across the
1.1. Brain Machine Interfaces

Figure 1.2: Conceptual scheme of a neuron. The axon connects different neurons and provide a means for action potential to propagate between neurons for communication. Image adapted from [5] under license CC BY-SA 3.0 (https://creativecommons.org/licenses/by-sa/3.0/deed.en).

membrane is made more negative the cell becomes hyperpolarized and presents a passive response where the voltage simply follows the stimulus. For a current stimulus making the voltage more positive, the neuron begins to depolarize. Up to a threshold value a passive response is still present where the voltage again follows its stimulus. Interestingly when the threshold voltage is exceeded the neuron is depolarized and emits a spike, presenting now an active electrical response to the stimulus.

A typical action potential shape is shown in Fig. 1.3 When the membrane potential exceeds its resting state a spike is triggered. This mechanism is sustained by voltage depended conductivity of selected ions providing positive feedback. The Na channels are opened, allowing Na\(^+\) ions to enter the membrane, the increase in potential opens up more channels which allow more Na\(^+\) ions to enter the membrane and the voltage shoots upwards. A second delayed feedback loop regulates the K\(^+\) ions by letting them out of the cell membrane and ending in deactivating the voltage dependent Na channels and hyperpolarizing the neuron. After this peak downward a refractory period follows, where the neuron is inhibited and does not react to input stimuli even if they would trigger an AP in resting potential condition. After the refractory period the cell membrane is back to its resting condition and the neuron can become active again when properly stimulated.

The positive feedback provided by the Na\(^+\) channels makes the communication between neurons an all-or-none signalling, there is no in-between state; once
Chapter 1. Introduction

Figure 1.3: Typical shape of an action potential showing the depolarization phase after the stimulus exceed the threshold voltage, followed by repolarization and a refractory period where the potential is below resting potential and the neuron is inactive. Failed initiations don’t trigger APs and the voltage is restored to the resting potential afterwards. Image adapted from [8] under license CC BY-SA 3.0 (https://creativecommons.org/licenses/by-sa/3.0/deed.en).

enough channels are active the voltage will reach its peak until the potassium channels become active. This mechanisms makes the shapes of APs stereotyped since it does not depend on the amount of the current received, as long as it is above the threshold a spike is triggered, otherwise not. This process was modelled for the first time by Nobel prize winners Hodgkin and Huxley quantifying the effect of the different ion channels and providing equations for quantifying the generation of APs [7].

1.2 Different Recorders, Different Signals and Applications

The generation of spikes briefly described in the previous section always considers the voltage across the membrane of a neuron and the outside of the cell. Inside the brain many neurons will instead be active at the same time to coordinate neural activity, and while it is possible to record voltages inside a single neuron, so called intracellular recordings [9], more often a conductor is inserted into the tissue and hence will be surrounded by many neurons and will pick up the electrical activity of all of those in its vicinity [10].
To distinguish one single neuron, the conductor needs to be placed close to only one neuron or more electrodes have to be used in order to differentiate between the same kind of neurons [4, 10–12]. These recordings are commonly known as extracellular recordings, in fact the electrodes recording them happen to be outside the cell membrane.

When the acquired signal is lowpass filtered only the slowly changing components are kept, these signals are called Local Field Potential (LFP), representing the neural activity of a whole brain region, hence spikes are no longer distinguishable and only the result of super-imposed APs is recorded. Less invasive techniques to record LFP are: Electrocorticography (ECoG) and Electroencephalography (EEG). ECoG records from electrodes directly under the skull, but not penetrating the brain tissue, hence being less invasive than electrodes inserted close to specific areas and neurons; these signals contain information on activity of brain areas and can contain, e.g. movement information, which was used to control prosthetics [13–16]. EEG signals are non-invasive recordings recorded from the scalp, they have also been used for motion detection but with less accuracy and greater training time compared to the more invasive ECoG [13].

EEG and ECoG represent a smoothed version of the LFP, which propagates towards the superficial electrodes [11, 12]; when recorded locally LFP can provide important information about a restricted area of the brain. This can be exploited in a Deep Brain Stimulator (DBS): these devices use LFP information to detect e.g. epileptic seizures and inject current in the selected region to inhibit the seizure [17, 18]. DBS is also used for Parkinson’s disease and other neuropathies with some treatments being already FDA approved and commercially available [18].

Other important clinical applications of neural recorders are represented by Brain Computer Interfaces (BCIs) or Brain Machine Interfaces (BMIs) used for prosthetic controls in patients with reduced mobility; these can use ECoG and EEG and hence being based on LFP. More interesting they can also use information coming from multi electrodes arrays (MEAs) which can record single/few neuron activity as described earlier. These present more complex signal processing but offer greater range of movement decoding [19]. Being able to study how single neurons interact with each other also allows scientists to understand the inner mechanisms of the brain and hence it is crucial for research to develop implants being able to record from more channels with a high spacial resolution in order to pick up spikes from single neurons [1, 4, 10–12].
1.3 Thesis Outline

After introducing the basic biological principles underlying neural signals in the brain and state of the art applications and research that is possible by accessing this data this thesis aims to show that data compression is crucial in such applications and provides data on which compression method is suited best for different applications.

Chapter 2 introduces state of the art recorders over the past years and covers in detail the steps needed for neural signal processing and acquisition. Eventually, limitations of these steps are introduced followed by the benefits of data compression for such applications.

Chapter 3 defines the methodology used for this work consisting in different programs for spike sorting operation and the data used to verify the claims on compression efficacy.

Chapter 4 treats quantization for neural signals as a preliminary step, in fact the workflow of the thesis assumes that data is already quantized and hence quantization is a preliminary step to compression.

Chapter 5 describes state of the art compression schemes in detail, introducing their strengths and weaknesses and preliminary analysis to their usage when needed.

Chapter 6 presents an extensive analysis of compression algorithms, where accuracy for neural signals is evaluated for different compression ratios to evaluate, not only effectiveness in data reduction, but as well signal quality after compression.

Chapter 7 talks about the strict hardware requirements of brain implants and defines a methodology to verify when compression is beneficial for the whole system. A detailed analysis of various compressor hardware requirements is shown. Feasibility discussion of compressors on an implant concludes the chapter.

Chapter 8 concludes the thesis highlighting and summarizing the findings of this research work.
Chapter 2

Background and Motivation

After a brief review of the biological mechanism underlying the electrical transmission in the brain, in this chapter the main challenges of BMIs and their related signal processing are presented with focus on benefits and challenges for data compression in neural recorders.

2.1 State of the Art of Neuronal Recorders

In neuroscience researchers use electrophysiology to understand the inner mechanisms of the brain. Electrophysiology consists, as seen in Chapter 1, in the study of voltages and/or currents across cell membranes to understand how biological systems work. For neuroscience this focus on the observation of how electrical activity of neurons serves to build up complex processing neural networks. Studying the brain poses several challenges for neuronal recorders: number of recording sites, data acquisition, transmission and processing.

To understand how neurons in different areas of the brain contribute to the whole function of our nervous system we want to be able to observe their electrical signals in detail, and furthermore to be able to follow the signal propagation from one neuron to the other. Approximately every 7 years the number of neurons that can be recorded has doubled for the past decades [20].

To meet the spatial resolution need, a brain recorder has to provide simultaneous recording from multiple recording sites: the more and the closer the electrodes the better. In Fig. 2.1 the number of channels, that can be recorded simultaneously, from recent works are plotted for both in-vivo and in-vitro recorders; it’s clear from Fig. 2.1 that in the last decades the research on neural recorders focused also in increasing the number of channels available.

In fact both higher percentage of area covered by the electrodes and density are desirable. The first allows scientists to study larger areas of tissue, providing clues how information is processed by the neuronal network of neurons. Also beneficial is having a high density of electrodes/recording sites, as it will be covered later in
Chapter 2. Background and Motivation

Section 2.2 since it helps to better recognize spikes from neurons when they change shape during long term recordings or due to electrodes shift [21, 22].

Recording from many electrodes in parallel is already challenging in terms of area and power for the analog front end and the digitization circuitry, on top of that transmitting the generated data presents its own challenges as well. This is true for devices to be used in-vitro since they have the highest channel counts, Fig. 2.1, transmitting and especially storing up to some Gbit/s becomes non-trivial. For in-vivo application data transmission itself becomes critical. For this class of applications a wireless link is necessary to avoid possible paths of infection caused by cable links; unfortunately power is constrained to avoid excessive tissue heating that will kill the cells adjacent to the implant [3, 23], hence compression methods to reduce the data rate are sought. Finally, since the huge amount of data recorded has to be processed, possibly in realtime, and eventually stored, data compression techniques can ease the requirements of these operations.

It is clear from Fig. 2.1 that wireless BMIs are the most challenging ones and technology is improving to allow their realization but the number of published implants is still very low. In fact for the few implants having a wireless link only 2 feature more than 10 channels: in [24] only timestamps are retained from the neural signals; while [2] uses an implanted and external device to achieve the channel count, but making it suitable only for studies in primates. Compression before transmitting the data would then be necessary to achieve high channel counts and a desirable wireless transmission.
2.1.1 In-vitro and In-vivo Measurements

When comparing different implants it is necessary to distinguish between in-vitro and in-vivo recordings; in the first, as the name implies, measurements are made in a slice of the neural tissue kept in saline solution while in the latter case measurements are carried on live subjects.

While the requirement for a large number of channels is present in both kinds of recorders, in-vitro recorders have less stringent power constraints, especially for signal transmission since they can use wires and bulkier table equipment (still power requirements on the AFE require specific solutions to integrate a large number of active electrodes in small area). Having more area and power available, in-vitro recording setups can offer higher number of channel counts compared to their in-vivo counterparts, Fig. 2.1, as high as several thousands [26, 29, 33]. Compression is then beneficial for in-vitro recordings to ease the, otherwise, huge amount of data to be transferred and stored.

Data compression is also useful for implanted neural recorders used for in-vivo applications to increase the number of channels and enable wireless transmission above all. As mentioned earlier a wireless transmission would avoid infection paths through cables but due to the limited amount of power available, to avoid tissue damage and prolong battery operation, the number of channels one can record simultaneously is strongly limited. Recent in-vivo wireless neural recorders have in fact a limited channel count [47, 48] and for high number of channels the recorders are bulky and not at all implantable inside the skull [2].

Hence data compression is beneficial for both use cases, in-vitro and in-vivo, although not for the same reasons and provides different benefits specific to the application. There is also an implementation difference for in-vitro setups, where having more power and area available, the compression algorithm can be more complex and power hungry. In contrast for in-vivo, the algorithm should be kept simple enough in order to meet the power constrain for an implantable device.

2.2 Neural Signal Processing

In this section the signal acquisition and processing chain for neural signals is described with particular focus on spike sorting. The process is described in detail, since before talking about compression it is important to understand the nature of the signals to compress and what are the most common processing steps.

After picking up the electrical signal from an electrode, this has to be amplified with a low noise amplifier (LNA) to avoid adding excessive electronic noise when the signal has still small amplitude. After amplification the signal is usually filtered to isolate the band of interest and avoid aliasing components; this is referred as the
Figure 2.2: Two architectural approaches to record neural signals in (A) one shared ADC converts all of the neural channels serialized at once. In (B) more ADCs are used to digitize data from a portion of channels and the digital signals are then all multiplexed and streamed to the processing block.

analog front end (AFE). Filtering is then followed by an ADC for digitization and eventually the digital signal is sent, possibly through wireless transmission, to an external processing unit [3].

In the case of MEAs, different architectures were used for the AFE; since many channels are available routing and multiplexing is necessary; it would be impractical or even impossible to have one output per channel resulting in chips with thousands of pads and wires for the outputs. Hence usually the data stream from different electrodes is ordered in a single stream, e.g. using time division where every sample of the final stream will belong to a different channel [26, 34].

In Fig. 2.2, two common approaches for the channel architecture are shown. In the top part of the figure every channel has its own LNA and filter. Amplified and filtered analog signals are then routed using an analog multiplexer to one ADC, which is then shared across all channels [24, 42]. Being shared across all the channels the ADC needs to be able to work at a frequency $f_s = N \cdot f_{\text{sample}}$ where $N$ is the number of channels and $f_{\text{sample}}$ the sampling frequency required for each neural signal.

To ease the speed requirements on the ADC another approach can be used, Fig. 2.2b, where the ADC is now shared across groups of channels, hence now the sampling frequency of each ADC is relaxed to $f_s = L \cdot f_{\text{sample}}$, where $L$ is the number of channels per group [24, 31, 32, 34]. This second architecture is hence more popular.
in recorders with high channel count.

Both architecture shown in Fig. 2.2 output a digital stream with all the channels multiplexed. Therefore this can be assumed to be the input to all the digital processing happening after the recording of neural signals, independent from the chosen AFE architecture.

As discussed in section 1.2, there are different kinds of signals that can be recorded from brain tissue, of most importance in this work the LFP and spikes. LFPs are the results of multi-neuron activity in the recorded region, specifically due to the superimposition of extracellular currents [11] and are hence slowly changing and present high amplitudes; spikes result from activity of one neuron and present higher bandwidth and lower amplitude. For convenience LFPs' and spikes' electrical properties are summarized in Table 2.1.

<table>
<thead>
<tr>
<th></th>
<th>Bandwidth</th>
<th>Amplitude</th>
</tr>
</thead>
<tbody>
<tr>
<td>LFP</td>
<td>0.5 – 300Hz</td>
<td>500μV – 5mV</td>
</tr>
<tr>
<td>Spikes</td>
<td>300Hz – 10kHz</td>
<td>50 – 500μV</td>
</tr>
</tbody>
</table>

Table 2.1: LFP and spike signals electrical properties [3]

In this work LFP processing is not considered hence it will be assumed that the recorded neural signal is filtered for spikes [26, 31, 42, 49]. In section 2.3 it will be shown that this is not affecting the resulting data rate but if the neural signal is comprised of both LFP and spikes, then the LFP component should be filtered before compression.

As stated earlier the spike signals are the effect of single neuron activity; in neural research these signal are then needed to study how single neurons interact and respond to stimuli [50–52]. To process this information the steps are to identify when a neuron is active (detection) and which one it is (sorting). This is referred to as spike detection and sorting or in short spike sorting [53]. A neuron creates an electric potential difference across its membrane and lets this signal propagate through the axons to other neurons, hence a neuron is active when such a voltage spike is detected. Then the first step is to identify such events by isolating them from electrical and biological noise. Afterwards the spike shape is analyzed and assigned to a neuron under the assumption that the spike characteristics are similar for the same neuron and different for others, Fig. 2.3. These steps will be analyzed in the next sections:

• Filtering
• Spike Detection
• Feature Extraction
• Clustering.
Chapter 2. Background and Motivation

Figure 2.3: Spike sorting scheme from data recorded in anesthetized mice used in [54] and available at [9].

a) bandpass filtered signal.

b) spikes are detected, here using wavelet transform prior to actual detection.

c) feature extraction using PCA, two well separated clusters are identified and spike shapes are shown per cluster with the average shape.
2.2. Neural Signal Processing

2.2.1 Filtering

The first filtering operation is to separate the low frequency LFP component from the high frequency spikes, to enhance spike detection another filtering operation is applied to the signal to reduce its bandwidth and reducing the false positives due to noise. Typically this second band-pass filter has a bandwidth of 300 – 3000 Hz [55] but care is required since phase distortion can lead to errors by preventing distinction of different kind of neurons [56] and distort noise artifacts to look like a real spike [57].

To avoid phase distortion a non-causal filter should be implemented, but clearly this would require some data buffering to introduce a delay in the signal or storing the whole recording and post-filter it forward and backward [58]. If real time is required by the application then, an IIR filter with nearly linear phase can be used [50]. Alternatively the data can be recorded with wide band filters (300 – 7000 Hz), introducing negligible phase distortion, then spike times are detected on narrow band data (300 – 3000 Hz) and finally spike shapes are extracted from the wide band original data [59].

2.2.2 Spike Detection

After filtering, the time information of the spikes can be extracted by distinguishing them from the background noise. Since spike amplitudes are larger than the noise, a common way to detect them is by applying a thresholding technique which detects a spike when the signal is above or below a certain threshold. This is simple and easily implemented in hardware [24, 60], although the problem how to choose the threshold is not an easy one [50, 55, 59]; in fact, a too high threshold will miss lower amplitude spikes, and a too low level will detect noise pulses as spikes. A common way is to pick a threshold as a multiple of the standard deviation of the noise $\sigma_n$, with typical values between $3\sigma_n$ and $5\sigma_n$. E.g. if $\sigma_n$ denotes the standard deviation of the noise the probability that a noise sample will exceed a threshold of $5\sigma_n$ is $5.7 \cdot 10^{-7}$ when assuming a white gaussian noise; on average approximately one event every second of recording at 20 kSamples/s.

Still, calculating the RMS value of the noise requires to record some part of the signal and distinguish the spikes from the noise floor, in fact when there are too many spikes, e.g. in a burst of activity, the RMS of the whole signal would not be an accurate measure of $\sigma_n$ [55]. In [55], also a method to approximate $\sigma_n$ using the median of the signal was proposed; although more accurate this method does not seem to be very practical to be implemented in the recorder, since to calculate the median a whole segment of interest in the signal needs to be stored.
Different threshold methods were also proposed in the literature: instead of measuring only one amplitude, a two point method can also include information of the duration of the putative spike, or two level threshold applied to one and/or more channels for MEAs [61–63].

To increase the detection accuracy, more sophisticated methods involving extra signal processing before detection were developed as well. The Non-linear Energy Operator (NEO) can be applied to the filtered data to facilitate the detection of the high frequency components in the signal, hence the position of the spikes [64, 65], and it allows to find spikes also when the power of the spikes is comparable with the noise floor [66]. Finally, other methods for detection have been proposed based on different algorithms including Wavelets [67, 68], template matching [69], fuzzy logic [70] and hidden Markov models [71].

It becomes thus clear how spike sorting is -until today- not well defined and many methods are used already in the first steps. Table 2.2 provides a list of previous works, available software and the methods used for spike sorting where it is already possible to see various detection algorithms implemented and used.

### 2.2.3 Feature Extraction

After being detected the putative spikes are to be assigned to a neuron. To do so features are extracted from the spike shapes to reduce their dimensionality and to be able to cluster these features. Geometric properties such as the spike amplitude and duration can be used [53]. Albeit easy to extract, these simple features don’t provide enough differentiation for different neurons [50]. E.g the spike amplitude is influenced by other neuronal activity superimposing on it, and also spikes from different neurons with different shapes can have similar amplitudes, making them indistinguishable when looking at amplitude as a feature.

A common and more advanced method for feature extraction is the Principal Component Analysis (PCA) [53, 54, 65]. PCA provides orthogonal components where the first principal component accounts for most of the variation in the data, and the following components capture as much variation as possible while being orthogonal to the preceding components. The spike waveforms can then be expressed as a linear combination of the PCA components where the first few coefficients accounts for almost all of the spikes energy. Hence, only the first PCA coefficients are needed for a clustering algorithm. In Fig. 2.3 c) an exemplary plot of the the first two PCA features for extracted spikes from [9] are shown. It is evident that for these two neurons there is a clear separation when using only the first two features, i.e. the dots form two well separated clusters. It is also possible to utilize more features to make the clusters more separable at the cost of more computations.
Wavelet coefficients can also be used as features to differentiate spikes from different neurons [55]; in fact, wavelet coefficients are also localized in time, as opposed to PCA which does not consider time information and can be chosen to obtain maximum separation of clusters, which is not assured by the direction of maximum variation provided by PCA. On the other hand PCA components are ordered in magnitude while the most significant wavelet coefficients have to be chosen prior to clustering. It was shown in [55] that wavelet decomposition can achieve better clustering results when compared with PCA for some spike shapes.

To provide a solution to the problem of choosing the feature which is best suited to classify spikes in [72] a spike sorting algorithm calculating a variety of features (geometric, PCA and wavelets) is proposed. This method assigns then a score to every family and pick the highest scoring feature to be used for clustering, hence combining the advantages of different methodologies, but requiring more calculations.

<table>
<thead>
<tr>
<th>Program Name/Reference</th>
<th>Detection Method</th>
<th>Features and Clustering Algorithm</th>
</tr>
</thead>
<tbody>
<tr>
<td>[66] NEO</td>
<td></td>
<td>Neural Network</td>
</tr>
<tr>
<td>Waveclus [55]</td>
<td>Amplitude Threshold</td>
<td>Wavelet Coefficients and Super Paramagnetic Clustering</td>
</tr>
<tr>
<td>Osort [65]</td>
<td>Energy operator</td>
<td>Online Sorting based on clusters distance</td>
</tr>
<tr>
<td>KlustaViewa [63]</td>
<td>Double Threshold Flood Fill</td>
<td>PCA</td>
</tr>
<tr>
<td>[73] NEO</td>
<td></td>
<td>Wavelet-PCA features with Watershed algorithm</td>
</tr>
<tr>
<td>[74]</td>
<td>Maximum a posteriori criterion and threshold</td>
<td>Continuous Basis Pursuit</td>
</tr>
<tr>
<td>[75]</td>
<td>-</td>
<td>Independent Component Analysis</td>
</tr>
<tr>
<td>[76]</td>
<td>Bayesian Optimal Template Matching</td>
<td>Bayesian Optimal Template Matching (requires prior Clustering)</td>
</tr>
<tr>
<td>[77]</td>
<td>Energy and Wavelet Detection</td>
<td>PCA and Wavelet features with Expectation Maximization for Clustering</td>
</tr>
<tr>
<td>Spyking Circus [78]</td>
<td>Amplitude Threshold</td>
<td>Density-based clustering and template matching</td>
</tr>
</tbody>
</table>

**Table 2.2: Overview of different spike sorting methods**
2.2.4 Clustering

The last step in the sorting procedure is to differentiate between the spike shapes to individuate and label the different neurons that generated them. E.g. in Fig. 2.3c, two neurons are found, red and green spikes; after detection and feature extraction the clustering algorithm will decide if the spike is more similar to the red or the green shape, hence, labelling it as coming from one or the other identified neurons, noise or an unforeseen neuron. The simplest approach to this is probably manual clustering, called cluster cutting \[53\], where a user defines the boundaries of the clusters. E.g. in Fig. 2.3c, a vertical line could easily be drawn in the middle of the PCA plot and this would define a boundary for the two neurons. Although very simple in principle this has several drawbacks. The first obvious drawback is the necessity of a person trained to perform the operation. Another problem derives from the dimensionality of the data and the need to find cluster borders in more than 2-dimensional spaces. Furthermore user bias can lead to clustering errors \[54, 79\].

Also used in the literature for spikes clustering is the nearest-neighbor method or k-mean clustering. The principle is to find the closest cluster’s center to a spike and assign that spike to this cluster \[53, 65\]. E.g. in Fig. 2.3c) on the right this is used to assign the spikes, represented by their first and second principal components, to a specific neuron, color coded in the figure. While the algorithm works best when the clusters are separated, like the one shown in Fig. 2.3c) the algorithm breaks down when the clusters are overlapping \[53\].

This can be overcome with Bayesian clustering where the contours of the clusters are defined by calculating the probability that a certain spike belongs to a specific cluster \[50, 53\], which works nicely when clusters have Gaussian shapes. Finally also a method based on statistical mechanics like super-paramagnetic clustering \[55\] which doesn’t make any assumption on the clusters has been proposed and can deal with clusters with more exotic shapes. Also other methods based on neural networks and different methodologies have been introduced \[66\].

2.2.5 Spike Sorting Summary

Spike sorting is in theory a simple operation, but as seen in the last paragraphs there are many implementation details that make it not standardized at all, and depending on the application and user preferences different methods for detecting and sorting the spikes are used. To emphasise the huge number of different ways that are actually being used to process neural information, Table 2.2 lists different methods from the literature and available software packages. For a more comprehensive review of the process and the state of the art there are many works and online resources published \[50, 53, 59, 72, 74, 80-83\]. A great variety of methods then exists for spike
sorting and moreover the research on this topic extends even further, since there are main problems related to the recordings in biologic tissues, e.g. spike overlapping, electrodes drift, amplitude variations and how to deal with bursts, to name a few.

Spike sorting algorithms are not the focus of this manuscript but since data compression is strictly related to the application, it is very important to understand what the processing of neural signals involves. Especially, and most important of all, for the sake of data reduction the knowledge that there is not yet a standard procedure, and spike sorting is highly dependent on user and application, is essential. Hence before throwing away, supposedly, not useful data points to reduce the overall data it must be considered what the targeted application will need. In some applications like BMI the usefulness of spike clustering is even doubted and there are works trying to answer this question [84–86]; therefore, for some applications, it might be possible to even retain only spike timing information. But in general spike sorting is still a very needed operation and still under research by neuroscientists as the amount of literature presented in this section shows.

2.3 The Need for Compression

Before any further analysis the first question that need to be answered is why compression is necessary in BMI. Looking at Fig. 2.1 it is clear that neural recorders and BMIs present an always higher number of channels. Increasing the number of channels is required for a better understanding of the inner working principles of the brain, with more channels scientists can observe neural activity from an increasing number of neurons and understand how neural networks work. To properly observe neural activity, typical reported signal resolutions in the state of the art range from 8 to 16 bits; this combined with sampling frequencies between $20 - 30\text{kHz}$ and the high number of channels can generate a very high amount of data to be transferred and processed.

E.g. for the device in [2] with 512 channels, 12 bits resolution and a sampling frequency of $31\text{kHz}$ the generated data rate amounts to $\approx 190\text{Mbit/s}$. Transmitting this data rate is rather challenging and it was achieved in [2] with non-implanted devices, that covered up a large part of the monkey head. Also for a wired implantable neural recorder [41] with parallel readout from 384 channels with 10 bits resolution and $30\text{kHz}$ sampling frequency the amount of data results in a quite large $\approx 115\text{Mbit/s}$.

As stated earlier for a BCI it is highly desirable to avoid cable connections through the skull, since these will easily lead to infections. One of the main challenges in providing this wireless link is the power dissipation, in fact exposure to excessive heat
can induce necrosis and angiogenesis in biological tissue [87]. In guinea pig olfactory slices with an increase in temperature of 2°C aberrant activity was observed [88] and spreading depression was elicited in rats by increasing the cerebral cortex temperature by 3.5°C. To avoid brain tissue damage in the long term the maximum temperature increase in the cortex must be limited to about 1−2°C or a maximum power density of 80mW/cm² or 800μW/mm² [23, 87].

In Fig. 2.4 power consumption against data rates is shown for various transmitters designed for neural implants, with transmitter areas in the range from 0.015 to 2.25mm² [97–101]. The maximum power density often exceeds the safety limit. In order to keep the number of channels high without damaging the surrounding tissue, compression is then necessary. In fact a reduced data rate translates directly into a lower amount of energy, and subsequently less heat dissipation. E.g. for the previously considered implant generating 115Mbit/s, a state of the art UWB neural transmitter with an efficiency of $E_b = 4.38\text{nJ/bits}$ [89] would consume approx 500mW. From the heat dissipation simulation and experiment in [102] for a brain implant about 10−20mW would increase the temperature in the tissue by more than 1°C. Hence compression is critical for implanted devices in order to meet the safety requirements.

When considering the high number of channels, only few transmitters in Fig. 2.4 can handle greater than 100Mbps datarates, and only one was actually implanted in-vivo [89]. Hence for the datarates generated in [2] and [41] only 2 state of the art Tx are published.

Furthermore from Fig. 2.4 a big discrepancy can be seen between measurements,
that were carried out at a bench in a lab, and in-vivo measurements. The former usually report power consumption even 2 orders of magnitude lower than implanted Tx for comparable data rates. One of the problem is the skin and moisture on and in it, which greatly attenuates the signals over $3 - 5\, \text{GHz}$ \cite{103,104}. Furthermore the alignment can be carefully adjusted in laboratory conditions but could become impossible to fully control in actual implanted devices. To avert this problem recent publications present a laboratory setup including biological tissue to better simulate the environment of the implant \cite{100,101,103}. Hence, also very power efficient Tx, that are measured only under ideal conditions, will most probably require compression too.

Lastly from Fig. 2.4 it is shown that in-vivo tested Tx achieving more than $10\, \text{Mbps}$ consume from about a hundred to several hundreds of mW for transmission, which would allow transmission of barely more than 40 channels sampled at $20\, \text{kHz}$ with 12 bit resolution. Indeed compression would be beneficial here in order to increase the number of channels.

Clearly reducing power and allowing more data to be transmitted for the same energy consumption are two sides of the same coin and both quite desirable for BMI to overcome the limitations provided by a biological environment.

But compression is also highly beneficial for in-vitro recorders; here constraints on the power are more relaxed and wireless transmission is not necessary but the amount of data that can be generated by these recorders is even higher. E.g. in \cite{26} 65536 channels can be recorded simultaneously, resulting in a staggering high almost $16\, \text{Gbit/s}$ data stream. Even with the price of hard-drives decreasing, storing this amount of data can be expensive, especially for long and multiple experiments. Once again being able to reduce the amount of data provides a more relaxed transmission system and reduced amount of storage needed.

After providing explanation and examples that compression is highly required for implants and very desirable for in-vitro recorders too, there are two major approaches to it:

- lossy compression - only information relevant to the application is kept
- lossless compression - the whole data is kept in a compressed representation.

State of the art for both approaches are illustrated in the following sections while the implementation details of the algorithms are provided in the next chapters.
2.4 State of the Art for Compression Schemes

2.4.1 Spike Detection On Chip

Perhaps the most intuitive way of compressing data is reducing its size by discarding parts of it that are not useful. Hence, as the name implies, introducing losses, this method can then achieve very high compress ratios and is very depended on the application requirements. The difficulty implementing this way of compression is in finding a way that will not alter the processing after compression.

Very different methods have been proposed in literature for this approach: in [24] only spike timestamps are transmitted offering online spike detection, hence the data is greatly reduced and wireless transmission for 100 channels was achieved. Unfortunately, this method prevents any further elaboration by means of spike sorting algorithms, hence all the methods seen in Section 2.2 cannot be applied. The threshold for deciding, when a spike is happening can be calculated on the fly using the circuitry described in [3] using basic probability properties, or more complex modeling using Sigma-Delta modulation [60].

Since spike timestamps don’t provide information about distinct single neuron activity, other work provided ways to detect spikes on chip and to transmit spike shapes and not only time information. In [105], spikes are detected using an absolute value threshold detector and their whole shapes are transmitted, achieving compress ratios from 4 to 40, depending on the spikes activity on the channels. To avoid losing the spike shape before the threshold crossing, a digital memory introducing a delay is needed. A similar approach is used in [106] where the delay and the memory are substituted by an analog buffer in order to quantize only samples that belong to spike shapes to save power in the ADC. This latter approach introduces more complexity and area consumption due to the high number of high-resolution sampling capacitors needed for the analog buffer. Furthermore the power saving in the ADC would not be significant when compared to the more power hungry transmitter in an implant.

All these approaches rely on spike detection on chip; as seen in Section 2.2 this might not be ideal for neuroscience research. In fact implementing sophisticated spike-detection methods on chip might not even be possible due to power and area constraints; furthermore on-chip implementations prevent scientists to apply different sets of parameters before deciding, which one yields the best results. On the other hand these methods reduce the data rate by a huge amount, when neural activity is low. Different methods, reported in the following subsections, allow to compress the spikes further and compress the background noise in order not to lose information about neural activity that might be present there.
2.4.2 Compressed Sensing

In the last years compressed sensing (CS) became very popular and has been used for biomedical signal compression [107–109]. CS aims at reconstructing the original data based on an under-sampled version of it provided that the data is sparse in some representation.

For neural signals CS was used in [110] and proved to be able to compress spike shapes by a factor of 3.2. In [111] CS methods were evaluated and found not suitable for compressing neural data: when the whole neural data was compressed most of the information after spike sorting was completely lost.

In more recent years, advanced methods to create data based dictionaries and improve sparsity were developed and led to compression ratios from 8 to 16 times data reduction [112]. Dictionaries were also improved by introducing spike classification information, which led to up to 50 times data reduction for spike frames [113] and 11 times using block sparsity [114]. Even data based dictionaries require training, which is usually computationally expensive and time consuming. Training-free dictionaries were developed and resulted in data reduction rates between 3 – 5 depending on the desired reconstruction quality [115].

Not only improving the dictionary is possible but also the sensing matrix can be optimized; a two-step method was proposed in [116] compressing simulated and recorded spikes by a factor of 5. A deep learning framework for compressed sensing was utilized to optimize the sensing and the reconstruction together and led to high compression rates between 30 and 40 times for simulated and recorded spikes [117].

As it can be seen, there are many publications implementing CS for neural signals and the reported results are very different. In the state of the art there is a lack of a unified way to compare compression methods for neural signals and depending how the analysis is conducted, the results can largely vary. Part of this presented thesis is dedicated in comparing these compression methods under the same working conditions to effectively evaluate them.

2.4.3 Delta Compression Schemes

Delta compression consists in transmitting the differences between two samples (deltas) instead of the whole signal. For signals that are very correlated, like neural signals, the deltas will have a lower dynamic range and a skew distribution, making them easier to be compressed [118]. In [40] delta compression is implemented in an analog fashion; a threshold block allows to have a trade-off between quality of the recorded signal and data rate. Successive work implemented delta compression followed by an entropy encoder in the digital domain [119] and proved effective for lossless and lossy compression with a data rate reduction of 2 for lossless
compression. Delta compression can also be modified to use samples from different channels, reducing the memory requirements and achieving comparable results to standard delta compression on low-noise data \cite{120}. Advantages of delta compression are its adaptability to be lossy or lossless and low power requirements, while the disadvantage is a modest data reduction.
Chapter 3

Methods and Data

In this chapter the figure of merit for comparing compression results, in both regards of its effective data reduction and quality of reconstructed signal are introduced. Understanding how compression affects the reconstructed signal requires the use of spike sorting programs and a method to compare sorting results before and after compression [111, 121]. Two different programs are used in this thesis, they are both publicly available, one is suited for online sorting and the other for offline sorting. They also cover different methods for detection and clustering in order to make the analysis more general and not dependent on one specific spike sorting approach. Various data is also used from simulated to recorded in-vivo and in-vitro experiments. Except for one data set, the data is also publicly available in order for the results to be reproducible by others.

The analysis will focus only on the spikes or action potentials (APs). Unless specified it is assumed that the two components, LFP and spikes are already separated and compression is applied only to the high-pass filtered data. This is in fact a likely scenario for implants, [49, 122]. Furthermore the bandwidth and resolution requirements for the LFP results in a very modest data rate, which is about 2 orders of magnitude smaller than the data rate required for AP data. In addition the methods explored for compression are implemented in the digital domain, hence the data is already digitized before compression.

3.1 Comparison of Spike Sorting Results

To assess how compression affects the signal, the method first presented in [111] is used and extended for two different approaches to neural recordings depicted in Fig. 3.1 [121]. In the top part spikes are extracted on chip and then compressed, while in the bottom part the whole neural signal is transmitted. When spikes are extracted on chip all other parts of the signal are lost; theoretically these components include only noise, electronic and biological noise due to far-away neurons. Spikes extraction on-chip is already a form of compression itself, as described in the previous
Chapter 3. Methods and Data

Figure 3.1: Compression scheme for neural signals. (top) Spikes shapes are extracted on-chip, compressed and transmitted. (bottom) Spike detection is done off-chip hence the whole neural signal needs to be compressed and transmitted. In both spike sorting evaluation is then done with compressed and non-compressed spikes or whole neural signal.

chapter, but spike shapes can then be further compressed by means of a compression algorithm. This scenario is common in neural recorders, especially for BMIs, where spike occurrences are only needed and detection errors are mitigated by the use of many close channels [10, 21]. Despite this, detecting and compressing spikes directly on-chip represent an highly ideal scenario; as seen in Section 2.2, there are many different methods to detect spikes with tunable parameters and some of them requiring access to long chunk of signal.

Some of the most sophisticated techniques might not even be feasible for an implant due to the power constraints, hence limiting the choice to only the most simplistic methods. Furthermore spike detection relies most often on a threshold, which has to be carefully chosen to provide the best results for detection. In fact a poor choice for this parameter can lead to too many false positive spikes, which can then lead to spurious neurons, and missed spikes. While false positives can be dealt with to some extent during clustering, for false negative there is no way to recover these spikes as the information is forever lost. At last, after selecting one threshold value it wouldn’t be possible to experiment by varying the threshold parameter and select the optimal one.

To make the application scenario in the top part of Fig. 3.1 an even more ideal one, for most compressors the spikes should be aligned in order to have better performance; due to noise the alignment is again non trivial.
3.1. Comparison of Spike Sorting Results

In the bottom part of Fig. 3.1, a different approach is depicted; as in the previous case, the neural signal is assumed to be already high-pass filtered and digitized. In this case there is no spike detection and the whole signal is compressed and transmitted. This is done to overcome the problems with spike detection seen previously. The disadvantage of this approach is that there is much more data to compress, and since also biological noise in between the spikes must be compressed the efficiency of the compressor is likely to be less when compared to the previous approach. The big advantage is that it is then possible to do both spike detection and sorting off-chip and apply all sorts of dedicated algorithms for it.

In both cases the compression efficiency can be quantified by the compression ratio (CR) defined as:

\[ CR = \frac{N_o}{N_c}, \]  

where \( N_o \) and \( N_c \) are the number of bits for the original and compressed data, respectively. Hence, CR provides a measure of how much storage/bandwidth the uncompressed data requires compared to the compressed version.

After having quantified how much information is saved through compression it is crucial to assess how much information is retained from the original data for lossy compression. This is clearly not needed for lossless compression where there is no information loss. The comparison methodology is derived from [111]:

1. the neural signal is first elaborated by a spike sorting algorithm and results are kept for later.
2. After the neural signal is compressed and then reconstructed,
3. the reconstructed neural signal is elaborated by the same spike sorting algorithm of step 1,
4. the two spike sorting results are compared.

The comparison is slightly different in case the spikes are already extracted on chip or not, Fig. 3.1. When spikes are already extracted only this information is processed by the spike sorting program, hence only clustering takes place. The spike shapes are then compressed, transmitted, reconstructed and eventually given as input to the spike sorting program.

A similar procedure happens for the whole neural data with the difference that spike sorting includes both detection and clustering. While compression is performed before any elaboration to compress the whole signal, Fig. 3.1.

With this procedure the effects of compression on the neural signal are clear also in terms of retained information. Data reduction should not affect the content of useful information present in the signal; hence, for neural recorders the spike sorting algorithm should yield the same, or very close, results when processing both
uncompressed and compressed data. Otherwise, as an extreme case, it would be possible to throw away everything and obtaining an infinite CR; but in this case there would be no information left. Thus, this equivalence check becomes obviously necessary, even though this is often omitted in literature.

### 3.1.1 Metrics Used for Comparison

After obtaining the two sorting results from the original data and the reconstructed version after compression, these two can be compared in terms of spikes belonging to the same neuron. When comparing two spike sorting results, the original sorting from uncompressed data, is considered as the reference. In most cases there is no ground truth available; thus, the scope of this analysis is to assess how much information loss happens after compression and not how good a spike sorting algorithm performs compared to ground truth. The following parameters are considered:

- \( t_{p,ca} \) number of true positive spikes correctly assigned to the same cluster/neuron in both sortings,
- \( t_{p,cr} \) number of correctly recovered/detected spikes but assigned to different clusters/neurons in the sorting from compressed data,
- \( fp \) number of false positive spikes, i.e. additional spikes found only in the compressed data,
- \( fn \) number of false negative spikes, i.e. spikes that are present in the original data but are not detected from the compressed data.

These parameters can be used individually or also combined together to provide a general accuracy measure defined as Classification Accuracy (CA) [121]:

\[
CA = \frac{t_{p,ca}}{t_{p,ca} + t_{p,cr} + fp + fn}.
\]  

This number ranges from 0 to 1, where \( CA = 1 \) means a perfect match between the two sortings, hence no information loss at all, while a number closer to 0 means a great number of spikes mis-classified, mis-detected or missed. In cases where false positive spikes are low only the number of correctly detected spikes can suffice and the Percentage of Correctly Assigned Spikes (PCAS) is defined as:

\[
PCAS = \frac{t_{p,ca}}{t_{p,ca} + t_{p,cr} + fn} \times 100.
\]

To determine if a spike is the same in both the reference and measured sorting, the comparison algorithm goes through all the spike timestamps in the reference
3.2. Spike Sorting Programs

For each of these it looks for the correspondent in the measured sorting, two timestamps are considered the same if their difference is less than 1 ms to account for timing difference that can occur after compression. The value is justified since spikes last about 1 – 2 ms, hence the detection has to occur inside a spike window and not too far away such that it could be actually another spike.

In the scenario of spike detection on chip, there can’t be any false positive, since the spike shapes are already extracted and given as input to the sorter, but there can be false positives when a spike is distorted and is then discarded as noise by the classifier. Again this scenario is easier than dealing with the whole signal, also for the comparison, since one of the negative parameters can be readily omitted. Furthermore this comparison doesn’t consider how much the spikes are distorted, resulting in a perfect match as long as the spikes are distorted in the same way and hence assigned to the same neuron/cluster. In fact the clustering information is the final result of the whole spike sorting process and is what matters. However when data is highly distorted, but only clustering is done, it is possible that heavily distorted data would still be clustered together, while, when detection has to be carried out, the distortion is more easily discarded as noise artifacts. Once again extracting spikes on chip results in simpler work for the spike sorting programs because the extracted spike frames are assumed to be the only available information.

This proposed comparison clearly considers spike sorting as the heart of neural signal processing; another measure of quality, which is independent of data processing, is the Signal to Noise and Distortion Ratio (SNDR) [112, 117], defined as:

\[
\text{SNDR} = 20 \log_{10} \left( \frac{\| x \|^2}{\| x - \hat{x} \|^2} \right)
\]

where \( x \) is the neural signal and \( \hat{x} \) is the reconstructed signal after compression. This measure is mostly useful for comparing different works, which very often don’t have a standard way of assessing neural signal quality, but it also lacks a clear relationship with the spike sorting process. In case of lossless compression the two signals, \( x \) and \( \hat{x} \), are the same, hence the denominator of Eq. 3.4 goes to 0 and the \( \text{SNDR} \rightarrow \infty \).

3.2 Spike Sorting Programs

As discussed, a comparison of spike sorting before and after compression is needed; but for this a spike sorting algorithm/program is required. In this work two different programs, implementing different methods for spike sorting, are utilized: Osort [65] and Waveclus [55]. Both programs are freely available online at [123, 124] respectively, allowing for a re-engineering of the results presented in the thesis. Osort is an online spike sorting, meaning that the sorting process can start processing data and
progressively update its results and also does not require interaction and the processing is completely automated. Waveclus on the other hand is an off-line sorter and requires the whole data to be recorded and then processed all at once. Another difference is that Waveclus can do sorting of already extracted spikes for the scenario pictured in Fig. 3.1 while Osort does not provide the option to separate spike detection and clustering operations. In the next subsections their working principles are briefly described.

### 3.2.1 Osort

There are different methods implemented by default in Osort for spike detection: level and absolute value threshold, energy based [66] and wavelet detection [67].

For this work the wavelet detection is used, since it is reported to be more effective than the other methods [67, 68].

In order to detect a putative spike with the wavelet method, first a discrete wavelet decomposition at different scales is performed. Then this information is used to assess if a spike is present at each scale and eventually the decisions at different scales are used to determine if a spike is present or not [68]. When a spike is detected the Multi-resolution Teager Energy Operator (MTEO) is used to align the spikes. This is particularly effective for signals where high level of noise distort the shape of the spikes and alignment based on peak amplitude is not reliable anymore [125].

After detecting and aligning a spike, Osort calculates its distance between waveforms of the already detected spikes and determines to which neuron the spike belongs or if the spike represents a new possible cluster. Being an online spike sorting tool, after assigning the new spike, the clusters need to be updated. When clusters are found to be too close to each other, a merging operation is carried out, otherwise clusters are left as they are. Osort’s algorithm is also meant to be unsupervised, hence it is a great match for a comparison tool.

### 3.2.2 Waveclus

Spike detection in Waveclus can be done only using value threshold, and its level is calculated by estimating the noise standard deviation, $\sigma_n$. The absolute value is used in this work as it enables the program to equally identify spikes which begin with an upward or downward peak. Then, a spike is detected when:

$$|x(t)| \geq N\sigma_n,$$  \hspace{1cm} (3.5)
where $x$ is the neural signal, $N$ is a parameter usually between $3 – 5$ and with:

$$
\sigma_n = \text{median} \left\{ \frac{|x(t)|}{0.6745} \right\}
$$

(3.6)

as described in [55]. Using the median instead of the standard deviation provides a better estimate for the noise level, especially for recording with neurons with high firing rates [55]; the drawback is that in order to be able to calculate the median all possible values for the neural signals need to be considered together for the calculation.

After detecting the spikes, Waveclus uses the discrete wavelet transform (DWT) to reduce dimensionality of the spikes before clustering. After feature extraction a super paramagnetic clustering (SPC) algorithm is used to cluster the spikes. SPC is inspired by principles of statistical mechanics to provide robustness to noise and flexibility [55].

Waveclus allows the user to perform spike detection and clustering on the whole neural data, or to process already extracted spikes performing clustering only. Both scenarios discussed earlier and shown in Fig. 3.1 can then be covered with this program.

Unlike Osort, Waveclus is not meant to be completely unsupervised but will provide the user with a set of clusters, which can then be discarded or kept. To make the algorithm useful for our analysis the settings are kept quite conservative, in order to avoid having too many noisy clusters. This proved to be reliable in the sense that when running the program twice on the same data with the same parameters, the sorting results will match. This makes also Waveclus suited for comparing different compression methods.

### 3.3 Used Neural Data Sets

To verify different compression algorithm, both synthetic and pre-recorded data are used. As it is the case for the spike sorting algorithms, the data, with the exception of one dataset, is publicly available and downloadable at www.CRCNCS.org after registration. This is done to make it possible to re-engineer and verify the results presented in the analysis. Most often in literature custom algorithms are implemented along with "closed source" data, and this renders it impossible to verify the claims of the compression algorithm.

### 3.3.1 Simulated Data Sets

Simulated data is synthesized in order to have recordings with known spikes from a known number of different neurons and with known, but different noise levels. The
data was used in [55] and is available at https://www2.le.ac.uk/centres/csn/software as well as the spike sorting software Waveclus.

The data is organized in Easy and Difficult datasets presenting easy to sort spikes, which have very different shapes and features, and harder to sort spikes with features that are very similar to each other. The datasets are meant to be challenging for spike sorting programs, hence the ground truth information is used only for spike extraction when compressing spike frames.

For easy and difficult data sets different noise levels are available, e.g. Difficult1_noise005 indicates a noise, having a variance of 0.05 relative to the spike amplitudes, is added to the simulated data set; thereby amplitudes are normalized to 1 and the noise is composed of superimposed spike shapes in order to make a good approximation of neural noise. As will be shown later, low noise data is very useful for testing spike sorting algorithms but seems to be too optimistic when compared to actual recorded data.

The data is provided with a sampling rate \( f_s = 24kHz \) and is rescaled to 16bits and bandpass filtered \((300Hz - 3kHz)\) before being used in this thesis. Every dataset consists of 1 minute of data.

### 3.3.2 Recorded Data

Recorded data is publicly available at www.CRCNCS.org after previous registration. The datasets used are from hc-1 [9, 21] and hc-2 [126]. The first set is recorded in anaesthetized mice with tetrodes implanted in the hippocampus. For this data, intracellular and corresponding extracellular recordings are available. This will be usually referred to as low noise recorded data, since the mice are asleep and spike sorting and compression are much easier compared to the awoken animals. In hc-2, in fact mice are carrying behavioral tasks and the recordings are much more challenging since neural activity is increased and so are noise levels in the recording. These recordings also provide shanks resulting in 32/64 channels of extracellular recordings.

A third dataset, the only not publicly available, is from in-vitro recorded data and was provided by NMI Natural and Medical Sciences Institute at the University of Tübingen. The recording consists of about 5.6s of data from CMOS-based MEAs of 65x65 channels of in-vitro retinal ganglion cell [127]. This dataset will be labelled NMI in the remainders of the thesis.

For all the recorded datasets, the sampling frequency is \( f_s = 20kHz \) with 16 bits resolution and the data bandpass filtered \((300Hz - 3kHz)\). Furthermore to avoid problems due to long recording all the sets are split in 1minute segments which are then individually compressed and processed. For hc-1 and hc-2, sessions ‘d1891102’,
3.3. Used Neural Data Sets

’d561106’, ‘d561107’, and ‘ec013_527’, ‘ec014_793’ are used. In total this thesis relies on

• 46 minutes of recorded data from hc-1 [9],

• 5.37 hours of recorded data from hc-2 [126],

• 6.57 hours from in-vitro recorded data NMI [127].
Chapter 4

Quantization for Neural Signals

In this chapter the effects of signal quantization are analysed. This is a very first and needed step before compression. Even though reducing the number of bits can not provide a great advantage in terms of data reduction, setting the inferior resolution limit helps to design more efficient compression methods. In fact, by reducing the number of bits required to a bare minimum, the risk of having bits representing only thermal noise is mitigated and therefore compression can be more effective as it will not have to deal with incompressible white noise. Furthermore different non-linear quantization approaches are investigated.

4.1 Resolution Requirements for Neural Signals

In the reported literature, it is common to see neural signals recorded with resolutions ranging from 8 to 12 bits for the spikes alone and 10 to 16 bits for LFP and spikes. Especially lab instrumentation offers usually 14 to 16 bits resolution and record the wideband signal (1 Hz – 10 kHz) including spikes and LFP components. In this analysis, the LFP component is assumed to be already separated from the spikes, either with filtering on-chip or off-chip processing.

Before analyzing any compression method for the spikes, the minimum number of bits required has to be discussed. To quantify the effects of quantization on spike sorting algorithms, the procedure described in section 3.1 is performed using a reference sorting based on the neural signal with the maximum resolution of 16 bits. Subsequently, the neural signal is requantized with lower resolutions and the sorting is compared with the reference. In Fig. 4.1 the effects of quantization can be seen for recorded and simulated data, for both Osort and Waveclus. As it can be seen, simulated data appears to be robust to quantization: decreasing the number of bits from 16 to about 5 does not produce an appreciable loss in spike sorting information when using Osort. For Waveclus slightly higher resolution between 6 and 7 bits is needed. Due to the different algorithm, Waveclus is able to better distinguish between subtle differences in the spikes than Osort. Hence some spikes are no longer assigned to different neurons if their differences consist in small features of
few LSBs in amplitude. Being not able to distinguish between these features, Osort clusters are more stable for low resolutions. Below a resolution of 5 bits, the number of correctly assigned spikes drops abruptly for both programs.

For the recorded data, results are highly different! The spike sorting quality with Osort drops more linearly with the reduction in resolution, and already below 8 bit CA ≤ 0.9 for hc-2, meaning that at least 10% of the original information is lost. Interestingly, for Waveclus between 8 and 6 bits the accuracy stays quite close to the ideal value. Since for recorded data the noise levels are much higher, also the amplitude threshold to detect spikes is higher. Therefore only the high amplitude spikes are detected, which are less prone to the decrease in resolution until the point when their features are too distorted to correctly assign them.

The causes of such different behaviors between simulated and recorded data are to be identified in the nature of neural data. The simulated data sets consist of typical spike shapes from 3 different neurons summed with white and simulated neural noise, Fig. 4.2. These conditions make it easy for the program to obtain the same classification information, even for low resolutions. On the other hand recorded data is much more challenging. By looking to an exemplary extract of different datasets from simulated and recorded data in Fig. 4.2, it is already quite clear that the noise levels are usually much higher than the ones in simulated data and the number of neurons, which can be detected, and hence the number of clusters is higher too. By comparing the top half of Fig. 4.2 with the bottom half it is seen that the simulated data has much lower noise levels than the recorded data. For noisy recorded data in hc-2 [126], putative spike events have no significant difference from the rest of
4.1. Resolution Requirements for Neural Signals

Figure 4.2: Plots of neural signals showing the different noise levels between low noise simulated data Easy1 (top left) and its high noise counter part (top right) from [55]. In the bottom part low noise recorded data from hc-1 [9] (left) and high noise recorded data from hc-2 [126] (right).

the signal, which is in contrast to simulated data. Hence the resulting difference between CA for recorded and simulated data. Having more neurons means that there will be some spikes with small amplitudes, since some of these neurons must be further away from the recording electrode. These smaller spikes are also more likely to be covered up in noise. What is thus expected is that these smaller spikes are gradually lost by the reduced resolution, which explains the first linear decrease in accuracy. Below the 8 bit threshold also the bigger spikes, coming from closer neurons, are getting misclassified since the increase in noise and variance of recorded spikes compared to the simulated data.

Already one of the leitmotifs of the thesis is presented in this section: compressing and analysing simulated data is usually much more simple than dealing with recorded data. It is very important to highlight this since simulated data is used rightly for validation of the algorithms. But in the end it is not enough when an algorithm can highly compress simulated data without providing how it performs with recorded data, representing a much more realistic (and utmost harder) scenario for compression.
4.2 Logarithmic Quantization

The first non-linear quantization investigated is logarithmic quantization. Logarithmic ADCs (log-ADCs) are employed in voice communication applications in order to reduce the data rate while preserving the signal quality. Also, they were proposed for neural implants [128, 129]. Logarithmic ADCs seem appealing for this application, since they allow to reduce the number of bits while preserving similar resolution for small amplitudes as well as large dynamic range for larger amplitudes. Still the effectiveness of logarithmic quantization for neural signal needs to be investigated in terms of data reduction and classification accuracy [130].

The idea of a logarithmic ADC is to vary the quantization step width logarithmically over the whole dynamic range. The output $D_{out}$ for an input $V_{in}$ is given by

$$D_{out} = \left[ 2^N \cdot \log_B \left( \frac{V_{in}}{V_{fs}} \cdot B^C \right) \right]$$

(4.1)

where $N$ is the number of bits, $V_{fs}$ the full scale voltage, and $B$ is the chosen logarithm base and $C$ is the code efficiency factor. Thanks to the companding characteristic, given by the log function in Eq. (4.1), the logarithmic ADC achieves good resolution for small input signals, but still allows coarsely quantized large input signals. By setting $C$ in Eq. (4.1) to large values the logarithmic characteristic will be more accentuate. The result is that quantization noise is lower when the signal amplitude is small, and it grows with the signal amplitude. Comparing with a linear ADC, where the steps are always equal, for the same number of bits, a logarithmic ADC can better resolve small signal amplitudes. In Fig. 4.3 this is shown for a neural signal. Here a segment containing a spike is shown; in the upper part a linear ADC is used and due to the dynamic range requirements the small amplitude part is poorly represented showing rather large quantization error (4.3 bottom). On the other hand the high amplitude spike is well preserved, including all details of its shape. In the middle part of Fig. 4.3 a logarithmic ADC is used, and it is obvious that here the low amplitude is well preserved, whereas due to the higher quantization steps the spike (high amplitude) shows worse quantization instead, which is again seen for the quantization error in Fig. 4.3 bottom at the position of the spike.

From this the motivation to use logarithmic ADCs for neural signals is clear, since the large dynamic range is compressed into fewer number of bits, thus saving potentially both area and power in the ADC and moreover saving bits to be wirelessly transmitted. Nonetheless, the logarithmic ADC distorts the shape of a spike, and thus the question arises, if this distortion has a critically negative effect on spike sorting algorithms, which determine from the form of the spike also the corresponding neuron.

To be noted is also, that - as Eq. (4.1) predicts - for the same number of bits...
and the same full-scale range, the least significant bit (LSB) of a log-ADC is much smaller compared to that of a linear ADC, shown in Fig. 4.4. Thus, it requires much smaller $kT/C_S$ sampling noise. Consequently, an advantage is only achieved, if the log-ADC can provide similar signal quality for a significantly lower number of bits and comparable LSB as the linear ADC.

Similar to Fig. 4.1 in Fig. 4.5, the effects of quantization for log ADCs are shown. The same trend observed for linear ADC is also found here: simulated data can be easily requantized to a low number of bits without, almost, no loss in the spike sorting process. A logarithmic ADC provides better reconstruction for less than 6 bits when compared to the linear ADC (Fig. 4.1), which originates from its ability to also
reconstruct small spike amplitudes with a low number of bits. While this resolution can be good enough for certain simulated data sets, for real recorded data, as expected, a significantly higher number of bits is required to provide acceptable sorting and the advantage of a log-ADC for low resolution quantization becomes only marginal while its increased hardware requirements are still much more demanding in terms of LSB resolution as shown in Fig. 4.4.

To conclude the analysis on logarithmic quantization and to answer the question if a log-ADC would be beneficial to BMIs, the difference in CA between log- and lin-ADCs is plotted in Fig. 4.6 for various resolutions. For lower resolutions the log-ADC presents a clear improvement, which results more prominent when Osort is being used; in fact the logarithmic function can represent the small amplitude spikes detected by Osort with higher precision. Despite the clear advantage, in previous sections it was shown that for lower than 6 bits the overall accuracy for both ADCs is not good enough for all considered data sets, except for the simulated data, when Osort is used. This prevents the log-ADC from being a better choice compared to a linear quantization; in fact the increased hardware cost to achieve LSBs at least an order of magnitude smaller than the linear counterpart (Fig. 4.4) is not compensated by an increase in accuracy. In fact for higher than 6-8 bits resolutions, the CA becomes $\geq 0.85$, but no accuracy gain for the logarithmic ADC in that range is seen anymore, Fig. 4.6. The advantage in CA even vanishes for some data with Waveclus, meaning that linear quantization is providing a more suited representation of the signals for spike sorting. Hence for neural signals a linear ADC is preferable and effective in preserving the accuracy for both Osort and Waveclus. As for the minimum number of bits, above 8 bits CA becomes generally greater than
4.3 Non-uniform Quantization

As outlined in Chapter 2, the spike sorting process is not standardized; but it is obvious that the useful information in neural signals is based on the spikes. The quantization could thus be adapted to have a maximum available resolution on putative spikes, in order to preserve the spike characteristics and to allow a precise spike sorting operation [132].

In principle the non-putative spike part of the signal can be completely omitted, but the caveat lies in the definition of a spike, or how to detect one. In order to have sophisticated methods based on energy and wavelets to detect spikes with greater accuracy off-chip, the background noise can still be transmitted with a lower resolution to save bandwidth. Furthermore methods have been developed to use information about the noise background to improve spike sorting accuracy [133].

The working of non-uniform quantization is depicted in Fig. 4.7: a detected spike is quantized using a high-resolution ADC while the remaining of the signal is kept but with a lower resolution, hence saving bandwidth. The spikes can be detected on-chip using a simple threshold detector; while this might not be the choice of the neuroscientists for analyzing the data, its purpose is to only find the most obvious APs. More sophisticated methods or simply different settings can be tested off-line thanks to the background information being kept.

Threshold crossing detection is suitable for such role of detecting putative spikes since it requires an easy implementation. Since the threshold should be set high

![Figure 4.6: CA comparison: difference between average CA for logarithmic ADC and linear ADC is plotted for different resolutions.](image)

0.85 but it might still be on the lower end for such data sets, especially for recorded data. Between 12 and 16 bits the accuracy is almost unchanged for all the considered data sets.
4.3.1 Non-uniform Quantization Architectures

Non-uniform quantization can be implemented in both analog and digital domains. In Fig. 4.8, the analog implementation is shown. The input consists of $N$ multiplexed channels. Then, the signal passes through an analog discrete-time buffer to insert a delay, during which the signal is compared with the threshold level. The threshold block then decides which ADC to enable: if the condition in (4.2) is met the high resolution is triggered, e.g. 12 bit or more, otherwise the low resolution,

$$|x(t)| \geq th,$$

where $x$ is the neural signal and $th$ is the threshold value in multiples of the noise standard deviation: $th = N\sigma$, with $\sigma = median\{\frac{|x(t)|}{0.6745}\}$ calculated as described in [55, 68].
4.3. Non-uniform Quantization

(a) Non-uniform quantization architecture for analog implementation

(b) Non-uniform quantization architecture for digital implementation

Figure 4.8: Block diagrams to realize non-uniform quantization for neural signals

e.g. 2 to 8 bits, ADC is used. Thanks to the high resolution analog buffer (with correspondingly large sampling capacitors), data points before the threshold crossing can be processed with the higher resolution, too. This operation ensures that the whole spike shape is preserved with high precision. A delay in the threshold block makes it possible to achieve the same for samples after the threshold crossing as well.

The scheme in Fig. 4.8a without the low resolution ADC (0 bit) is equal to the architecture proposed in [106]. Disadvantageously, completely eliminating the information about background noise also eliminates any possibility of recovering spikes that may fall below the threshold level. Furthermore realizing an analog buffer implies high area usage since the length of the buffer translates to the required number of sampling capacitors, which is then multiplied by the number of channels. Especially, if higher resolution is required in order to reduce $kT/C_S$ noise the individual sampling capacitors are between 1..10 pF. This can dominate the area if the buffer length is large. This structure is also inherently not very flexible due to the analog implementation and the maximum number of costly analog delay (S&H) blocks can not be widely adjusted.

The proposed alternative is to implement the algorithm in the digital domain, Fig. 4.8b. [132]. Here, a high resolution ADC converts the neural signals coming from N channels. The delay is then implemented with a digital register. This is not as costly as sampling capacitors, and provides much more flexibility and area reduction compared to the analog buffer, as well as a simpler design and technology scalability. The threshold block works in the same way as Fig. 4.8a, only digitally implemented and its output enables a truncation block. This reduces the resolution of the acquired signal if (4.2) is not met, otherwise let the high resolution version unaltered. Thanks to the delay provided by the register, the spikes’ shape is preserved as well.
In the digital implementation, the high resolution ADC is always active, but the larger power consumed in this block is negligible compared to the power used at the transmitter. In fact, state of the art transmitters require 1...1000\(pJ/\)bit, where the most power efficient transmitters usually provide only small data rates, or results are measured through air or do not even provide antenna measurements, as seen from Fig. 2.4. In contrast, ADCs require 1...100\(fJ/\)conv, making their efficiency per bit at least an order of magnitude better. Hence, if compression is provided at expense of more ADC usage, the system will still benefit in terms of saved power, as we reduce the data to be transmitted, while we pay for more digital flexibility with the increased - but still far from dominant - ADC power consumption. It has to be noted that also for an analog implementation, as in [106], the high resolution full data rate ADC has to convert signals at a conversion rate of \(N \cdot f_s\), where \(f_s\) is the sampling frequency of a single channel. In fact, in case of a spike burst, it might happen that all channels meet the condition in (4.2), and need to be digitized with high resolution. Thus, area and peak power consumption for the ADC in the analog approach are not lower, but only the time-multiplexed operation would reduce its averaged power consumption.

### 4.3.2 Non-uniform Quantization Effectiveness

Non-uniform quantization is applied here to neural signals from simulated datasets described in [55] to assess the effects of varying the threshold level as well as the number of bits used for the background activity. This data will then provide information how to setup the parameters for non-uniform quantization in the next chapters in order to properly evaluate its effectiveness against other methods on both synthetic and recorded data.

In Fig. 4.9 non-uniform quantization is applied to all the simulated data sets available in [55] and Osort is used for spike sorting, while results with Waveclus will be presented in the following chapters. Average results with one standard deviation error bars are shown. The effect of using different resolution for background activity is analyzed by varying the number of bits used for the low resolution quantization, and the results are also shown for different threshold levels in multiples of the noise standard deviation \(\sigma\) in (4.2). While the spikes are kept at the maximum resolution available of 16 bits. As it is expected, increasing the number of bits increases the Percentage of Correctly Assigned Spikes (PCAS) from Eq. (3.3) and reduces the error bar, meaning that the algorithm preserves most of the information for all data sets. On the other hand the CR is obviously reduced by that. Similarly, the CR drops when the threshold is reduced, since now, more parts of the signal are inside the high resolution window. When using a threshold of only \(1\sigma\), the reconstruction is good because most of the signal is quantized with high resolution, and for this reason the
CR is practically non-existent. For 0 bit background information, as done in [106] and other similar thresholding methods, the reconstruction was found good only for very low noise data, but with an extremely high error bar. The CR was obviously found really high, but the reconstruction quality too poor.

To be mentioned, this improved result using coarse background signal quantization is not due to the spike sorting program needing some background activity: this was proven by replacing the background noise by a pseudorandom noise with the same variance and the achieved PCAS is far from what is shown in Fig. 4.9. Furthermore by changing only the threshold level, either high CR can be achieved or high PCAS, while when changing the resolution as well, a better compromise or adaptation to the recording conditions is possible. E.g. in Fig. 4.9 with 4 bits for the low resolution ADC and \( th = 3\sigma \), reconstruction of more than 95\% of all spikes and 3 times data reduction can be achieved.

Performances of non-uniform quantization will be further investigated in Chapter 6 and will also compared to the other compression methods presented in this thesis and results are provided for both Osort and Waveclus using the methodology.
described in Chapter 3. Non-uniform quantization is also an appealing technique that when implemented in the digital domain does not require implementation of big capacitors array and help preserving the CA of the neural signals. After presenting in Fig. 4.9 how this quantization method behave by varying its parameters it will be considered in the next chapters as a proper compression method because of its effectiveness and easiness of implementation.
Chapter 5

Compression Schemes for Neural Signals

In the previous chapter quantization effects were investigated; quantization is not considered in this work as proper compression scheme but as a preliminary step for digital compression techniques, with the exception of non-uniform quantization that when realized in the digital domain can be compared to the other compression schemes. In this chapter various other approaches are presented with preliminary results of their effectiveness for neural signals and feasibility for implanted devices.

5.1 Entropy Coding

Entropy coding is a lossless compression method, which uses only signal statistics. The two most famous algorithms are known as Huffman and arithmetic encoding and will be discussed in more detail in the next sections. These methods are not specific for neural signals but are combined in this thesis with other source coding techniques to enhance data compression. Before describing in more detail the working principles of these two methods, in the remaining of this section the fundamental definitions from information theory and notation are introduced.

One of the key definitions in information theory is the following:

**Definition 1.** Let $X$ be a random variable over an alphabet $\mathcal{X}$, its entropy $H(X)$ is defined by:

$$H(X) = - \sum_{x \in \mathcal{X}} p(x) \log p(x),$$  \hspace{1cm} (5.1)

where $p(x) = P(X = x) = p_X(x)$ for $x \in \mathcal{X}$ represents the probability mass function.

The entropy is a measure of an uncertainty of a random variable and plays a major role in compression since it defines the maximum achievable CR on a source for **lossless** compression given the signal statistics.
A transformation is now considered, such that for every element in the alphabet of a random variable it provides a string of finite elements from a dictionary $D$. To provide a more concrete example, neural signals can be seen as the realization of a random variable. When recorded from the electrode, the values are analog and continuous time; after the ADC the signal is now discrete time and discrete amplitude, the possible values it can assume are now in fact limited to the levels of the ADC. The values are represented in the alphabet $X$ and the ADC performs the coding by providing strings of binary symbols. A particularly useful property of the emitted codes is that every value $x \in X$ is represented by a unique string of bits and can be interpreted independently of the previous or future values. A code with this property is called a prefix or instantaneous code \[118, 134, 135\] and for such the following theorem holds true.

**Theorem 1.** Given a prefix code for a random variable $X$, the expected length, $L$, is greater or equal to the entropy $H(X)$:

$$L \geq H(X).$$  \hfill (5.2)

The proof of the theorem can be found in \[134, 135\]. From this theorem it is clear that if compression is lossless the best achievable is a code with an average length $L = H(X)$.

E.g. consider and ADC with 12 bit resolution, then there are $2^{12}$ symbols in $X$ and $l(x) = 12$, where $l(x)$ is the length of the codeword $x$, $\forall x \in X$. The average length is given by:

$$L = \sum_{x \in X} p(x)l(x),$$  \hfill (5.3)

and in this specific example

$$L = 12 \sum_{x \in X} p(x) = 12$$

since $\sum p(x) = 1$; this is indeed not very surprising for an ADC with an output fixed to 12 bits length! Although trivial, the example hints that to reduce the number of bits needed to represent a signal generated by some random variable, the length of the codewords $l(x)$ has to change accordingly to the symbols’ probability. Upon this idea lossless compression of the source can be achieved. In the next following sections, the two most popular methods are reviewed.

### 5.1.1 Huffman Encoding

Proposed by Huffman in 1952 \[136\], Huffman codes are optimal prefix codes for a given probability model \[118, 134\]. The code is built on two principles:
5.1. Entropy Coding

• more probable symbols should have shorter codewords,

• the two least probable symbols have codewords of the same length.

By following these two ideas, Huffman showed that it is possible to create a decodable code; formal algorithm is provided in the original paper and many modern books on data compression [118, 134, 136], hence here only one example of such procedure is given.

Example 5.1.1. Consider a random variable with alphabet \( X = \{-2, -1, 0, 1, 2\} \) and probabilities 0.2, 0.2, 0.3, 0.2, 0.1 respectively. If a fixed length coding is to be used then \( \lceil \log_2 |X| \rceil \) bits are needed, for this example with 5 elements this means 3 bits. Since the symbols are not equiprobable, using a fixed length code is not optimal; in fact the entropy \( H(X) \) is only 2.246 bits/symbol. Meaning that for a signal generated by this random variable a CR of 1.33, or about 33% data reduction can be achieved by using an optimal code.

Such code is shown in Fig. 5.1 where either 0 or 1 is assigned to least probable symbols. Then the operation is repeated but this time the least probable symbols are considered together, as shown in Fig. 5.1 by the internal tree nodes, where the probability of two symbols are summed up. For this example from Fig. 5.1 the following code is obtained:

\[-2 \rightarrow 01, \ -1 \rightarrow 00, \ 0 \rightarrow 10, \ 1 \rightarrow 111, \ 2 \rightarrow 110.\]

The average length of this code is only 2.3 bits/symbol, very close to the entropy of the source, hence showing that the signal generated with a distribution \( p_X \) is compressible when using length-varying codes.

Huffman coding is widely used in many applications and in specific also for biomedical signals [119, 137, 143] since its simplicity and the possibility to be combined with different methods, as explained in the following sections. One of the good properties about this encoding procedure is that the code generated is optimal.
and its average length \( L \) will be bounded by:

\[
H(X) \leq L \leq H(X) + 1.
\] (5.4)

Proof of this is again left to the many scholar texts that cover the topics \([118, 134]\).

Despite being optimal there is no guarantee that the code will be optimal when considering sequences of symbols, in fact for sources with memories often have sequences that are more probable than others. In this case the number of elements in the Huffman code would grow exponentially with the sequence length, making it unpractical \([118]\). Also it becomes unpractical to have different Huffman codes depending on the previous samples of the signal. E.g. there could be one code made to account for the typical statistic of background noise and another code for putative spikes, since one can expect that these will have different distributions; this would require to store different dictionaries for correct encoding and again can lead to large memory requirements, especially for neural recorders.

### 5.1.2 Arithmetic Encoding

Huffman encoding is very intuitive and yet effective way of compressing data, in fact it can be proven to be an optimal code when the probabilities of symbols are a power of two \([118, 134]\). One big limitation is that the code considers only one symbol at a time. Considering that spikes originated from neurons have similar shapes it makes sense to account for the history of the signal as well, instead of treating every symbol individually. Unfortunately if there are \(2^N\) symbols (e.g. samples from the ADC) the alphabet for encoding needs to have one entry per symbol, so when a sequence of two symbols together is to be compressed, the Huffman dictionary will grow to \(2^{2N}\). The more symbols are to be considered together the bigger the dictionary, making it practically impossible to manage such huge dictionary already for 2 symbols.

Arithmetic encoding solves the problem of those exponentially growing dictionaries by elaborating the symbols as a stream; the input data is processed by the encoder and a number is outputted. Every sequence of symbols will be represented by a real number lying in the interval \([0, 1)\).

The working principle of arithmetic encoding is to start with the whole interval \([0, 1)\); when a symbol is processed the interval is reduced by a factor given by the probability of that symbol to happen. At the end of the process the midpoint of the interval can be taken and at the receiver the symbol sequence can be reconstructed \([118, 144]\). The conditions for reconstructing the right sequence are obviously that the correct number representing the sequence is received, and that the encoder and decoder are using the same probability model for every symbol.
There are two advantages coming from this technique:

- the whole sequence is considered when encoding, hence taking advantage of most probable sequences and not individual symbols alone; but still encoding is done symbol by symbol;

- the probability model can be changed at every symbol, as long as it changes in both encoder and decoder.

These two advantages allow arithmetic encoding to achieve optimal codes even when the symbols probability are not power of 2 and the second advantage allows for better data representation without necessarily having to use multiple dictionaries, which would require increased memory storage compared to a single dictionary. The last point is further explained in the next section when a technique is provided to better model the data probability.

The last point regarding arithmetic encoding is its implementation; in the scope of the thesis it is used only on software level since it requires divisions and complex modeling. Although its complexity can be reduced and actually used in hardware this would be topic for future improvement. Here only the effects of compression on neural signals are investigated.

### 5.1.3 Prediction by Partial Matching

Prediction by Partial Matching (PPM) can be used to model the conditional probability of data sequences. For example in figure 5.2 spikes from different neurons are shown: in the top part the average shapes and in the bottom 100 waveforms from the two neurons are plotted. The two spikes have different waveforms, in particular the left one has first a positive peak followed by a smaller negative one. The right spike has the opposite and higher amplitude for the second peak.

This could be useful for compression: after seeing the first part of one of the spikes, the encoder can recognize which family it belongs to. E.g. after seeing some samples going upwards it makes sense for the encoder to assume that the first spike shape is being recorded and to change the probability accordingly. So if usually after the positive peak a negative one follows it is convenient to assign high probability to samples with decreasing amplitude after the peak.

This method of pattern matching is known in literature as PPM \[118, 144\] and is used to model the conditional probability of a symbol given a past sequence: \( p(x_m | x_{m-1}, ..., x_0) \). PPM combines nicely with arithmetic encoding because the probability modeling is separated by the encoding itself. Since accounting for all the sequences would yield again exponentially growing storage space seen for Huffman dictionaries, a solution adopted in PPM is to prune the model by keeping only
the most frequent symbols and sequences and output an escape character for un-
seen new symbols. When an escape is outputted the probability model resort to a
more basic model, either a fixed one or one with conditional probability on shorter
sequences. This offers a tradeoff between the complexity of the model and its effec-
tiveness.

5.1.4 Entropy Encoding of Neural Spikes

As it will be shown later entropy encoders are a building blocks enabling compres-
sion of various signals. As a very first analysis Huffman and arithmetic encoding
with PPM can be used to compress neural spikes. This gives an idea on the limits
of this kind of lossless compression and why it is usually combined with different
signal processing techniques.

In Fig.5.3, average spikes for dataset Easy1_noise005 from [55] are plotted with
error bars covering one standard deviation (std) of the data for that particular data
point. If the mean shape of the spikes is known, from previous sorting or from
some modeling, it is possible to obtain an estimate on the maximum compression
achievable on the spikes using only entropy encoders.

The spike shapes are constant when emitted from the same neuron [133], know-
ing the average shape of the spikes it is possible to subtract it from the shapes in
figure 5.2 and the unknown element to be transmitted is now only the spike vari-
ance. The probability density function (pdf) can be assumed to be Gaussian with
mean as the mean value of a spike (solid line in Fig. 5.3) and std from the variance
calculated for every point (error bars in Fig. 5.3). After subtracting the average shape
values from the pdf the entropy can be calculated, and using Eq. (5.2) a maximum
achievable CR can be estimated.
5.1. Entropy Coding

From the entropy alone it turns out that for these kind of spikes $CR = 2.21$ is the best obtainable when the average shape is known in advance and lossless compression is sought. This is only slightly better than $CR = 2.05$, which is obtained for the same dataset by using Huffman encoding, and assigning each symbol a probability $p = \frac{N_{counts}}{N_{tot}}$, where $N_{counts}$ is the number of appearances of that particular value and $N_{tot}$ is the total count of samples.

These results would be difficult to obtain in a BMI because it requires to know the average spike shapes, recognize the most probable one, subtract its average shape from the actual values and then encode the differences. To make this feasible a better probability model is needed. As seen in Section 5.1.3, PPM with the conditional probability accounting for repetitive patterns in the spike shapes and combined with arithmetic encoding can provide this better model for the spikes. The conditional probability does not require knowledge about the class of the spikes but will try to infer it from previous values.

In figure 5.4 CR for different contexts used to calculate the conditional probability $p(x_m|x_{m-1}, ..., x_0)$ are plotted. The contexts consist of the sequence of past symbols taken into account for the conditional probability, e.g. a context of 2 means $m = 2$ in the aforementioned probability formulation and so on for different values.

In Fig. 5.4 it is shown that the CR, when using PPM on spikes with very little noise, is slightly higher than the entropy limit of $CR = 2.21$. This means that PPM can be applied without prior knowledge of the spike classes and the pdf obtained is better suited for the data than subtracting the mean, hence that conditional probability is better suited at modeling this data than counting symbol appearances.

Unfortunately, lossless compression is very limited also when conditional probability is used to model the data. The CR in Fig. 5.4 is decreasing for increased context.
lengths, albeit not very significantly. This fact is not explained by the number of points used to make the statistic, which shows the same behavior for an increased amount of samples, and hence it is likely that it comes from the statistical properties of spike shapes themselves.

Unless a modeling function, that can better predict the spike shapes and its variability, is found, in this section it was shown that for lossless compression of spike frames the CR is about 2.5, depending on the used modeling. To go beyond this limit it is possible to pre-process the signals to skew their probability distribution and/or resort to lossy compression; in the latter not the whole information of the shape is going to be encoded and a more aggressive compression can be achieved at the expense of some information loss.

5.2 Differential Encoding

Differential encoding is based on sending differences between actual signal samples and a prediction of these samples. The key idea is that many signals present high correlation in space and time [145]. In the case of neural signals one can expect a memory effect, in fact the signals are not randomly generated by the neurons and the shape of the action potentials, as seen in Chapter 1, depends on the neuron and presents very similar characteristics. It is also very unlikely that biological signals can change arbitrarily fast. In fact they are generated by ions moving from one side to the other of a cell membrane, hence charge transfer is limited and depends on the previous state of the membrane. Concerning spatial correlation, this can be expected if the electrodes are close enough and the recording is not too noisy, either because of electrical noise or background neurons. Under this condition the signal picked up
by an electrode will be recorded by other close-by electrodes as well. It is expected to see different signal amplitudes depending on the electrodes relative positions due to the tissue partly absorbing the signal.

In Fig. 5.5, a tetrode recording of a neural spike from dataset in hc-1 is shown. From the extracellular channels it is clear that, when the electrodes are close to each other and to the neuron, they pick up the same spike, but scaled depending on their distance. It is possible to quantify the correlation in time and space of neural signals by calculating the auto-correlation function and the cross-correlation between channels. The auto-correlation in Fig. 5.6 a) indicates that the signal has dependency on its past values which decreases rapidly after a few samples, and becomes almost negligible after 6-8 samples. In b) and c) in Fig. 5.6 the cross correlation between an adjacent channel and a far away one is shown from recordings with groups of 8 electrodes in hc-2 data. These plots indicate (in a quantitative way as opposed to the qualitative plot of Fig. 5.5) that close channels have some correlation and, as expected, that it decreases with the distance between the channels.

Understanding that neural signals are correlated in time and space allows to introduce what makes differential encoding appealing: using past and adjacent channel samples in order to predict the next incoming sample. When doing so, only the difference between the predicted and actual value needs to be transmitted. A typical scheme is shown in Fig. 5.7. When a sample $x_i(n)$ from the $i$-th channel at sampling time $n$ arrives, the predictor block will make a guess based on the previous values of the signal. The prediction $\hat{x}_i$ is then subtracted from the actual value:

$$e(n) = x_i(n) - \hat{x}_i(n).$$

(5.5)

For compressing neural signals it is possible to exploit the correlation with past and
adjacent samples to obtain an accurate prediction. Since the prediction is based only on past values of the signal, it can also be made at the receiver and it is then information that is represented in the model of the predictor and does not need to be transmitted; hence this reduces the amount of data that needs to be sent.

Actually subtracting predicted values from the signal is not compressing the signal itself; but if the predictor is accurate then the error will have smaller dynamic range than the original signal and a more skewed probability density function. Hence, compressing the error becomes easier with an entropy encoder [118] such as a Huffman encoder. In Fig. 5.8 we can appreciate how much the dynamic range of the signal is reduced by using a temporal predictor; for the specific case this leads to more than two times increase in data reduction compared to compression applied directly on the signal.

At the receiver end the value of $e(n)$ is summed with the value given by the predictor to obtain again the original signal $x_i$. The scheme in Fig. 5.7a) can be modified to obtain lossy compression, where original and reconstructed signals are not mathematically the same. Lossy compression can then trade reconstruction accuracy for higher CR. E.g. in Fig. 5.7b) a threshold block is included in order to transmit a zero when the error is below a certain threshold. Therefore, when the prediction is close to the actual signal, the system transmits a zero and the reconstructed signal will be equal to the prediction. Thresholding will then increase the CR by reducing the number of symbols to be transmitted for the error. Since the predictor at the decoder
5.2. Differential Encoding

Differential encoding with predictors is an appealing technique for compressing neural signals by allowing the entropy encoder to compress them further than what would be possible without the prediction. Furthermore the scheme can be realized to be lossless or lossy by setting a threshold before transmitting the error. Appealing to BMIs is that the predictor can be simple and doesn’t require complex feature extraction like PCA or wavelet techniques, which might be too power and area consuming for neural recorders. In the following sections different kind of predictors, suitable for differential encoding in a neural implant are presented.

5.2.1 Delta Compression

The Standard Delta Compression (SDC) scheme is illustrated in Fig. 5.9a [119]. It is composed of a delta predictor/modulator followed by an Huffman encoder. The predictor is as simple as possible: one register storing the previous value of the neural signal. The error of the prediction at a given time \( n \) for one channel is given by:

\[
e(n) = \Delta(n) = x_i(n) - x_i(n-1)
\]
The differences are all done using samples from the same channel under the assumption that biological signals don’t vary randomly and too fast. Then this predictor, although very basic, provides already some compression in the order of $2 - 3 \times 10^6$ data reduction as will be shown in detail in the results in Chapter 6. Compared to the scheme in Fig. 5.7, the predictor is positioned in a slightly different way and a threshold block can also be inserted. The threshold block zeros values below a given threshold; when this block is enabled the compression becomes lossy resulting in a reconstruction error at the receiver side. This loss is justified by the increase of the CR due to an increased probability of transmitting a zero, which helps having a more skewed distribution of the error. Thresholding small values is effective in neural signals, which present isolated spikes rising from a lower noise floor. Disadvantageously, this can lead to loss of information and quality of the reconstructed signal, thus the choice of the threshold level is crucial. To avoid the threshold error to sum up, a modified scheme can be implemented \[118\], where the reconstructed value is subtracted to the actual signal sample instead of the previous plain sample \[118\]. At the receiver we decode the Huffman stream and then reconstruct the signal by summing the received data.

When this structure is used in a multichannel recorder a register of length $N$ is needed to store the previous samples from all channels and do time multiplexing. This has a couple of disadvantages:

- **Power Consumption:** Extra $N$ registers are needed on the implantable side
- **Flexibility:** The compressor and the transmitter must be designed for a fixed number of channels.

In Fig. 5.9b a modified architecture is shown, referred as Interchannel Delta Compression (IDC). The idea is to eliminate the extra registers: now the previous sample
5.2. Differential Encoding

Figure 5.9: $X_i$: quantized sample of channel $i$. $\hat{X}_i$: previous quantized sample of channel $i$, with $i = 1, ..., N$ where $N$ is the number of channels.

used for the prediction belongs to a different channel instead of the same one. For $N$ channels the output of the modulator will change to:

$$
\Delta_1(n) = X_1(n) - \hat{X}_N(n - 1), \quad \hat{X}_N(0) = 0
$$

$$
\Delta_i(n) = X_i(n) - \hat{X}_{i-1}(n - 1), \quad \forall i = 2, ..., N.
$$

Ideally the different channels are recorded from close by probes, the two signals then share most of the waveforms and delta compression could become even more effective, when the channels present a higher spatial correlation than temporal correlation. This architecture reduces the hardware requirements by reducing the number of registers on the implantable side and it allows for a flexible compressor since the number of channels can be changed without modifying the compressor structure.

5.2.2 Slope Predictor

The implicit assumption made on delta encoding is that the signal is not fast changing and therefore the differences between adjacent samples have a lower dynamic range than the original signal. One could go further and assume that the derivative of the signal is slowly changing and assume that the speed and direction of the
growth of the signal remain constant, hence, the prediction is modified to:

\[
\hat{x}_i(n) = x_i(n-1) + \frac{dx_i}{dt}(n)T_s
\]  

(5.7)

where \( T_s \) is the sampling period, hence assuming a linear approximation for the neural signal. The discrete derivative is calculated as:

\[
\frac{dx_i}{dt}(n) = \frac{\Delta x}{\Delta T} = \frac{x_i(n-1) - x_i(n-2)}{(n-1)T_s - (n-2)T_s} = \frac{x_i(n-1) - x_i(n-2)}{T_s}.
\]  

(5.8)

It is worth noting that the derivative at time \( n \) is defined by past samples only. This is necessary to ensure that the predictor at the receiver will be able to output the same value as the transmitter predictor. Inserting Eq. (5.8) into Eq. (5.7) the prediction is given by:

\[
\hat{x}_i(n) = x_i(n-1) + x_i(n-1) - x_i(n-2) = 2x_i(n-1) - x_i(n-2).
\]  

(5.9)

This predictor model still results in a low complexity system; in fact it only requires a multiplication by 2, consisting of only a shift to the left in digital domain, and a subtraction. This is a very desirable feature as the less complex the predictor, the less power will be required to compress the signal on the implant side.

### 5.2.3 Linear Neural Network

A linear neural network (LNN) can be used as a predictor [145]. Its structure can also be seen as a FIR filter, where the weights are variable and will be adapted to the signal to be compressed, in a machine learning fashion. The inputs to a linear neural network are multiplied by weights, determined during the training phase and adapted to a specific signal, the results are then summed together to obtain the output of the net, hence the name linear network since the output is not further processed by non-linear functions. This is illustrated in Fig. 5.10 for exemplary 4 inputs. The shown network features only one layer and no bias to reduce the hardware requirements to implement the network. In fact implementing multiplications could be too demanding in terms of area and power for an implantable device and therefore a detailed analysis of this predictor’s hardware implementation is required.

A temporal and spatial selection of all \( N \) neural signals \( x_i(n), i = 1, \ldots, N \), is stored in the memory making a certain number of delayed samples, from one up to all \( N \) channels, available as inputs \( u_i \) of the neural network. After being multiplied with the weights \( w_i \), the signals \( u_i \cdot w_i \) are summed together. For \( N \) channels, this can be expressed as scalar product between two vectors: \( \langle W, U \rangle \), with \( U = (u_1, \ldots, u_N) \) being the input vector and \( W = (w_1, \ldots, w_N) \) the weight vector. Since the neural
5.2. Differential Encoding

\[
\sum_{i=1,...,N} w_i u_i \quad w_2 \quad w_3 \quad w_4 \quad u_2 \quad u_3 \quad u_4
\]

**Memory Array**

\[
\begin{aligned}
\mathbf{x}(n) |_{i=1,...,N} \\
\widehat{\mathbf{x}}(n) |_{i=1,...,N}
\end{aligned}
\]

**Figure 5.10:** Linear neural network layer without bias used as predictor, \(x_1(n), ..., x_N(n)\) is the serialized data from \(N\) channels sampled at time \(n\).

Signals are high-pass filtered, a bias is not necessary, as the mean of the signals is assumed close to zero.

It is possible to use the LNN predictor to exploit temporal or spatial redundancy, or both, by modifying the memory behaviour accordingly, i.e. which channels are stored and for how long. Three different working modes are then considered:

- **Temporal only information (LNNT)** - only samples from the same channel are used to make the prediction.
- **Spatial only information (LNNS)** - only samples from adjacent channels are used to make the prediction.
- **Spatial and Temporal information (LNNST)** - samples belonging to the same channel as well as samples from adjacent channels are used for the prediction.

In Fig. 5.11 these different modes are depicted. E.g. in order to predict the third sample for channel 1, \(x_1(3)\), using only temporal information (LNNT), the neural network uses two previous samples from channel 1: \(x_1(2), x_1(1)\). In a spatial mode (LNNS) only information belonging to the neighboring channel 2, \(x_2(2), x_2(1)\), is used to predict the next sample \(x_1(3)\) in channel 1. Combining both methods, the linear network can use both spatial and temporal (LNNST) information, hence using previous samples coming from both channels, e.g. \(x_1(2)\) and \(x_2(2)\) to predict \(x_1(3)\).

More generally, when only samples from the same channel are used in LNNT mode, then the input \(U = (x_i(n-M), .., x_i(n-1))\), for a positive delay of \(M\) samples on the \(i\)-th channel, and the prediction exploits the temporal correlation of that channel over time. In spatial mode LNNS, in order to predict the next sample of channel 1, \(x_1(n)\), the neural network input vector \(U\) is setup by \(L - 1\) neighboring
Chapter 5. Compression Schemes for Neural Signals

channels from a total of $N$ recording channels, $U = (x_2(n-1), .., x_L(n-1))$; consequently only spatial information from a maximum of $N-1$ channels is used for the prediction. Concerning complexity, more memory of course requires more registers for storage. E.g. in temporal mode using $M$ previous samples:

$$\hat{x}_i(n) = w_i^1 x_i(n-1) + ... + w_i^M x_i(n-M) \quad \forall i = 1, ..., N,$$

thus $N \cdot M$ samples must be saved for an $N$-channel recorder, but the predictor only uses $M$ of them at a time, while predicting the next sample of one of the channels. Similar reasoning can be done for the spatial mode. More profitable is to use the information from all the $N \cdot M$ stored samples, since they anyhow are stored and thus can be beneficially used. This enables prediction on all channels taking full advantage of spatial and temporal redundancy, LNNST. For LNNST there will be a maximum of $N \cdot M$ multiplications per channel since every $M$ stored values, from at most all $N$ channels, influence the estimation of the next sample of a specific channel $\hat{x}_i(n)$.

To determine the weights of the neural network, the built-in Matlab function \textit{newlind} was used. The function uses the least mean square (LMS) algorithm, which minimizes the mean square error between a training input sequence and a target sequence. The training of the network is done offline and requires short training sequences. For a sampling frequency $f_S = 20kHz$, less than one second per channel was used for training. This training method provides very similar coefficients as when using the correlation matrix method as described in [118], which is commonly used in differential pulse code modulation as well.

Different resolutions have been tested for the weights $w_i$ to ensure it was possible to implement the method with reasonable resolutions. A fixed point multiplication and 12 bit resolution for the coefficients are used. Increasing the resolution further did not improve the compression performance; reducing it would allow to compromise CR and hardware resources. In Chapter 7 this will be investigated and explained in detail.

Figure 5.12 shows the CR for 3 different datasets; quite clearly, the CR saturates to a maximum value. Increasing the number of taps further, hence the number of
Compressed Sensing (CS) allows to reconstruct a signal with arbitrary precision from under-sampled measurements. A formal description of the sensing problem is to reconstruct an $n$-sample signal $x \in \mathbb{R}^N$ using $M < N$ measurements. What CS requires is that $x$ has a sparse representation on some basis $\Psi \in \mathbb{R}^{N \times P}$, i.e. the signal can be written as a linear combination of the columns in $\Psi$ [148]:

$$x = \Psi \alpha$$  \hspace{1cm} (5.10)

with $\alpha \in \mathbb{R}^P$ having only a few $K < P$ non-zero coefficients. In other words what Eq. (5.10) implies is that to represent the signal $x$ only a few columns in $\Psi$ are needed since the coefficients vector $\alpha$ consists of mostly zero and few non-zero coefficients. When this condition is met for some base or dictionary $\Psi$, then the signal $x$ can be reconstructed from $M$ measurements. Formally the measurements are given by
linear combinations of the original signal $x$:

$$y = \Phi x = \Phi \Psi \alpha$$

(5.11)

with $\Phi \in \mathbb{R}^{M \times N}$ called sensing matrix. Provided that $x$ has a K-sparse representation and

$$M \geq C \mu^2(\Phi, \Psi) K \log N$$

(5.12)

holds true for some constant $C > 0$ and with $\mu(\Phi, \Psi)$ defined as the coherence between $\Phi$ and $\Psi$, which measure how much the two matrices are correlated and has unitary value for totally uncorrelated matrices. Then, the most important result of CS is that $x$ is obtained from the measurement $y$ by solving the following minimization problem for $\alpha$ [148, 149]:

$$\min_{\alpha \in \mathbb{R}^P} \|\alpha\|_1 \text{ subject to } y = \Phi \Psi \alpha$$

(5.13)

and then applying Eq. (5.10). Hence the signal can be reconstructed by minimizing the vector $\alpha$, meaning taking the sparsest vector $\alpha$.

This result presents a problem: while signals can be sparse, in a real world scenario there will be noise corrupting the measurement. Hence $\alpha$ will not be exactly $K$-sparse but will present $K$ large coefficients, while the others will be smaller in value. Luckily, also in this scenario it is possible to reconstruct the original signal although with some error.

One extra condition is needed: the so called restricted isometry property (RIP). One of the reasons why CS became popular is that one can choose as sensing matrix $\Phi$ a random Bernoulli matrix, consisting of random sequences of $\pm 1$, including or excluding zero as possible value, and this sensing matrix will be incoherent with most of the dictionaries, hence minimizing $\mu(\Phi, \Psi)$ in Eq. (5.12), while fulfilling the RIP properties. Since these matrices fulfill the CS requirements and are also easy to be implemented, they became popular in energy-constrained environments, e.g. BMIs, where the complexity of reconstructing the signal by solving Eq. (5.13) is outsourced to the non-implanted side, where power is not a critical issue anymore, while at the same time this saves bandwidth and power by transmitting only $M < N$ measurements.

### 5.3.1 Compressed Sensing for Neural Signals

The major problem with CS techniques is then to find a suitable base $\Psi$ that allows the signal to be sparsely represented. In this thesis, the technique presented in [112] is used to generate a data-dictionary. Signal dependent neural compressed sensing (SDNCS) [112, 150] is chosen for the analysis, because it can be used to compress
both already extracted spike frames and whole neural signals, hence it works in both
cenarios depicted in Fig. 3.1. Other similar methods presented in [113] and [117]
showed to perform slightly better in terms of reconstruction quality and compres-
sion, but these methods work only on extracted spike frames.

The hypothesis of SDNCS is to decompose a neural signal containing a spike as:

\[ x = x_c + x_f, \]  

(5.14)

where \( x_c \) and \( x_f \) are the coarse and the fine shapes of the spike and \( x \) the whole
spike frame. The coarse shape \( x_c \) is the pattern that a spike from a particular neuron
has, as if it wasn’t affected by external noise factors. The fine component accounts
for the distortion due to spikes from distant neurons superimposing, and electrical
noise in the recording chain. The two components are reconstructed using different
dictionaries: the coarse shape is represented by a data dictionary trained using K-
SVD algorithm [151] on a set of already extracted spike frames; the fine component
is reconstructed using a more generic Wavelet dictionary.

To compress the data, a Bernoulli random matrix with ±1 entries for lightweight
hardware implementation is used. On the receiver side, for the reconstruction first
\( x_c \) is found using Orthogonal Matching Pursuit (OMP) [152], then the residual \( x_f \) is
found using Compressive Sampling Matching Pursuit (CoSAMP) [153].

In order to improve the reconstruction, when the whole neural signal is to be
compressed, the temporal information of a spike is required. Therefore a threshold
detector is utilized and, the crossing point is used to shift the spike dictionary in or-
der for the shapes to be aligned in the frame and in the dictionary. When the thresh-
old detector doesn’t detect a spike, the reconstruction is carried out using only the
wavelet dictionary since it wouldn’t make sense to reconstruct noise using a spike
dictionary. This improved version -suitable for whole neural signal compression- is
called SDNCS with Prior recovery information (SDNCS-P) [112]. Both SDNCS and
SDNCS-P frameworks are used in the later analysis to compress spike frames and
whole neural signals respectively.

5.4 Machine Learning for Data Compression

The Linear Neural Network presented in section 5.2.3 is a so-called adaptive filter.
As shown, the LMS algorithm used to adapt the filter, is a building block for machine
learning. In fact a training part of the data is used to find proper coefficients for the
filter to minimize the error between prediction and actual sample value, Eq. (5.5),
without requiring user’s input. The filter used in section 5.2.3 can also be seen as the
basic version of the perceptron or artificial neuron. Indeed, in this work it is called
LNN to highlight its commonality with neural networks.
Given how effective LNN can be, it follows that perhaps more sophisticated artificial neural networks (ANN) could achieve even better performance. The drawback will be again complexity and the loss of feasibility especially for implants utilizing older and more power hungry technology nodes. It is anyhow worth investigating if and how effective more complex neural networks can be used to compress neural signals. The investigated architecture consists of Autoencoders (AEs): networks with more elements that directly try to achieve compression by gradually reducing the number of processing elements. In the following an overview of the basic concepts of neural networks is presented.

### 5.4.1 The Perceptron

Independently of the approach the basic block of most neural network is the perceptron as depicted in Fig. 5.13. The perceptron’s diagram is similar to the LNN in Fig. 5.10; the inputs \( x_i \) are multiplied by their respective weights \( w_i \) and summed altogether. A constant element called bias \( b \) is introduced and added to the previous products’ sum. Eventually the result is passed through the activation function, which will be described in more detail in the following paragraphs. As in a biological network, single neurons are combined in more layers by connecting the output of a neuron from a previous layer to the synapses/inputs of the successive layer.

To better understand the details of the perceptron it is useful to compare the artificial neural networks to the biological nets. When considering the scheme in Fig. 5.13, the inputs are signals coming from other neurons, hence inspired by the synapses of the biological neural networks. The output of a neuron can be seen as high or low signal, or better a non-active or active signal. Indeed, when biological...
neurons are active a spike is fired, otherwise not. The condition for a neuron to become active is that there are enough high signals coming from its synapses. Every synapse is weighted, modeling the influence these particular neuronal connections have. Then the signals altogether will determine the behavior of this particular neuron, hence the activation function will determine whether an active signal is outputted or not.

There are different kinds of activation functions depending on the data and the desired behavior of the neuron. In Fig. 5.14 some common types of activation functions are depicted. In case of the LNN the activation is the identity function, in fact no further operation is carried out and the output of LNN is a continuous-value signal.

A step function can be used to achieve the active/non-active behavior described before, by properly setting a bias; e.g. with negative bias the output will be active only when the weighted sum of the synapses is larger than the bias. Instead of clipping positive values, a rectified linear unit (ReLU) can be used to make a neuron not active for negative values, and when active, the neuron will preserve the weighted sum of its inputs. To introduce non-linearities, activation functions like the sigmoid, tangent and hyperbolic tangent functions are widely used as well. Their behavior in ANN is similar and they can be chosen depending on the required range of the neuron’s output.
There are many more activation functions used in literature and various ML applications; in this work, linear and tanh activation functions are used for neurons that have to reproduce the real values of neural signals, usually ranging from both negative to positive values those two activation functions are the most indicated to replicate real values. When more layers of neurons will be stacked, in the middle layers also ReLU is used to extract features from the signal, which can then help the compression.

### 5.4.2 Training and Feed-forward Operation

The peculiarity of neural networks is that their configuration can be learned and adapted to the task at hand. This process of adapting to a task is referred as training. To train a network the back propagation algorithm is used. After the architecture of the neural network is decided, input data is given to the network and the output is computed. The output is processed by a cost function that quantifies how far the actual output is from the desired output. E.g. for an estimator, after feeding some past sample of the neural signal, the desired output is the next sample at the input. To calculate how far the actual output is from the ideal output, the Euclidean distance can be used as cost function (as it is done in section 5.2.3) and the training will try to minimize it.

Once the cost or error is calculated, back propagation uses the gradient descent method to do the actual minimization. This works by calculating the partial derivatives of the output with respect of one of the parameters in the network. E.g. the output of the perceptron in Fig. 5.13 can be written as:

$$\hat{y} = f \left( \sum_{i=1}^{n} w_i \cdot x_i \right),$$  \hspace{1cm} (5.15)

and for a desired output $y$, we want to minimize the error function $E(y, \hat{y})$. In order to do so for the gradient descend the derivatives $\frac{\partial E}{\partial w_i}, \forall i = 1, ..., n$, have to be computed. Starting with $\frac{\partial E}{\partial \hat{y}}$ and applying the derivation chain rule it turns out that:

$$\frac{\partial E}{\partial w_i} = \frac{\partial E}{\partial \hat{y}} \cdot \frac{\partial \hat{y}}{\partial w_i}.$$

Introducing $z = \sum_{i=1}^{n} w_i \cdot x_i$ Eq. (5.16) becomes:

$$\frac{\partial E}{\partial \hat{y}} \cdot \frac{\partial f}{\partial z} \cdot \frac{\partial z}{\partial w_i} = \frac{\partial E}{\partial \hat{y}} \cdot \frac{\partial f}{\partial z} \cdot x_i.$$

This quantifies how much a change in the input will change the error. Hence the weights can be changed to a proportional factor, called the learning rate $\epsilon$, in order
to achieve the desired minimization \[154\]:

\[
\delta w_i = -\epsilon \frac{\partial E}{\partial w_i}.
\]  \hspace{1cm} (5.18)

From this simple derivation, it is already possible to observe some of the important elements for training a network. First of all, neurons can be combined into layers and the chain rule can be extended to the previous layers. But the more layers, the smaller the final gradient will be, which makes the training computationally hard.

The second fact is that activation and cost function need to have a well defined and non-zero derivative; hence some functions, e.g. the step function, cannot be used unaltered but require some modification. Also when activation functions have a zero gradient Eq. (5.18) becomes \(\delta w_i = 0\), hence the weights are not changed and the net will not learn; some functions can be modified to avoid this problem, e.g. the leaky ReLU \[155\] shown in Fig. 5.14. In this thesis, the one function that is useful for such a model would be a quantization function to mimic already in training the hardware limited resources. Unfortunately a quantization function does not have a continuous gradient, hence it is not possible to insert it in the training phase.

To circumvent this problem the networks in this thesis are trained using full available resolution. After the training is completed a quantization layer is introduced at the output of selected layers, e.g. before data transmission in order to achieve proper compression. The quantization will then only appear in the feedforward operation of the network, hence only when the output is being computed. To also reduce the computing cost of the multiplications, weights can be quantized too. Training with reduced resolutions and purging the weights in order to save computation power for embedded systems is a very young ongoing research topic \[156\] \[157\]. Since the application of this thesis is for medical neural implants, the training part can be computed using full resolution on a system with virtually no power limitations and then the net can be optimized to run on an implantable device.

5.5 Autoencoders

Autoencoders (AEs) are a particular kind of neural network consisting of an encoder and a decoder part. When the net is trained by imposing the desired output signal to be equal to the input it learns how to reproduce the input by learning salient features to distinguish and reconstruct the original signals. This training of the net is unsupervised since only the signals have to be provided without any further information from the user.

Another useful scenario is when spike frames are directly classified by an AE, also here features from the input signals are learned but they are used to classify
the various inputs in categories. This training is supervised since training inputs are associated with a class, in order for the net to classify features from the input signals into classes. This will be covered later in Section 5.5.2 while in the next section AE working principles will be illustrated along with its use for compressing spike frames.

5.5.1 Autoencoders for Spike Frames Compression

A neural network outputting a signal equal to its input might not seem useful for compression purposes. It becomes relevant for compression when such a net is designed to have a hidden layer with less neurons than the input layer, hence becoming a bottleneck, Fig. 5.15, the input has to be recreated at the output after creating a compressed representation of it. In fact, the information is passed from the encoder layer to the decoder layers through the hidden layer, which has reduced dimensionality. Then, in a compression scenario this first layer of the net acts as an encoder to provide a compact representation of the data. Afterwards, only this new signal is transmitted to the remaining layers of the network, which act then as a decoder. In Fig. 5.15 an exemplary net with a single hidden layer dimension of 3 neurons is depicted. Here the input layer consists of 5 elements, which are connected to the middle layer hidden units, since these are not seen outside of the AE. The output of the hidden layer is then sent to the decoder part of the network. While the encoder
is placed on chip and is highly constrained on power and area, the layers forming
the decoder are virtually not power constrained at all. These last layers can then
be implemented using full precision arithmetic and many more layers in order to
reconstruct the input with great accuracy, while keeping the computational cost low
on the implanted encoder part.

This can be seen as similar to what happens in CS in Section 5.3 in fact if the
synapse weights of layer 2 are chosen to be $\pm 1,0$ values at random, with a linear
activation function the same effect as random sampling can be obtained.

The output of the second layer is then passed to the last layers in the decoder,
which will try to reconstructure the original input as best as possible. For the example
in Fig. 5.15 when the same resolution for the input and the middle layer’s output is
kept, the data is reduced by a factor of 1.66. Hence it is possible to use this structure
to transmit the compressed representation and still reconstruct the original data at
the receiver. Furthermore, the CR is determined by the net topology and given by
the ratio of the number of neurons in the hidden and input layers, when assuming
the same resolution for input and hidden layer. This is again very similar to the
situation of CS in Section 5.3 where the size of the sampling matrix determines by
how much the data is compressed.

The autoencoder is then forced to learn an approximation of the identity function
by trying to achieve $\hat{x} = x$. Clearly this task is not always possible depending on
the data and the net structure. E.g. for a random Gaussian noise input signal and
a lower number of neurons in the intermediate layer, the autoencoder will not be
able to reproduce all of the possible inputs. On the other hand, if the data has some
structure, the net can learn how to exploit the data to provide a representation with
reduced dimensionality. Indeed in many cases a simple autoencoder will produce
a similar representation to Principal Component Analysis [158]. In case of neural
spikes, an AE could learn how to do some internal classification and reproduce the
spike shapes. This method of network training is also called unsupervised learning
because it does not require a user to label the different kind of signals, but it relies
on the network ability to find a compact representation of the data.

Example 5.5.1. In Fig. 5.16 an output example of two autoencoders for the same
input is shown. In Fig. 5.16 a) the input spikes from Easy1 (low noise) [55] data set
are plotted, in Fig. 5.16 b) and c) the output of AEs with 2 and 1 hidden units are
plotted, respectively.

From Fig. 5.16 a) and b) the AE can preserve the shape of the spikes nicely even
if only two hidden states are retained. With the input spike frames having a length
of 64 samples, and the input and hidden layer maintaining the same sample res-
olution, this means a data reduction of a factor 32. What can be observed is that
the spike shape is well preserved but the noise tends to be poorly represented. Not
surprisingly the AE can learn how to distinguish the spikes and plot their average shape, but it can not represent noise well. Interestingly enough when the number of hidden states is reduced to 1 (hence doubling the data reduction factor from the previous case to 64) the shapes become mixed with one another (Fig. 5.16 c)), suggesting that the internal representation of the autoencoder is similar to a codification of the class of the spikes. Hence the internal layer will determine, to which class the spike belongs, and based on that the output layer outputs the corresponding average shape for that class. For only one neuron in the hidden layer the AE cannot properly represent all three classes and the output ends up mixing them. Also when looking at 5.16 b), the two spikes on the right belong to the same class, and despite having slightly different shapes in 5.16 a) they have mostly the same shape at the AE’s output. This indicates that for this case the spike’s class is correctly identified by the AE, which can then nicely represent an average spike shape at the output, but not enough information is provided to also reconstruct the noise part.

The AE in Fig. 5.15 is called a dense autoencoder, because it utilizes only one layer, and neurons in the hidden layer have connections to all the neurons in the input layer. To model more complex data, the number of layers can be increased, making it a deep AE. The number of layers is usually balanced at the encoder and
decoder part, but for BMI the computation cost, which is much higher at the encoder than the decoder side, can be alleviated by skewing the distribution of the number of layers by putting more on the decoder side. Also the use of linear neurons is related to PCA features [158], but different features can be used in the neurons to make the AE learn non-linear, more interesting, features about the data. The ReLU activation function, Fig. 5.14, is a good candidate as it does not require computationally heavy calculations and is also one of the most common choices for feature extraction.

In [117] such an approach was used, where one layer would compute the random sensing and then a deep decoder would reconstruct the signal with great accuracy. In [117], the first layer of the AE is limited by the number of neurons and by the weights that can be ±1 or zero; this makes the computation lightweight on the implanted side. In fact this situation is analog as using a binary sensing matrix for CS. On the non-implantable side more and more complex layers are used.

### 5.5.2 Autoencoders for Classification

Since this thesis focuses on compression of neural signals, it is very interesting to answer if AEs are actually good at classifying neural spikes. This would in fact result in an extreme CR because no information about the time-domain shape of the spikes has to be transmitted but only an identifier of a class; this scenario would be particularly useful when spike frames are extracted already on-chip.

AEs are known for their denoising properties when properly trained [159]: during training an AE will try to learn a compact representation of its input. This results to be quite robust against noise corrupting the input. Hence when trained to classify spikes, the AE should also be able to recognize the different shapes under different noise levels.

In this case, for the hidden layer the ReLU activation function (Fig. 5.14) is used. This is in fact more suited for recognizing features, which are useful for classification. The hidden layer in the encoder will then try to learn features that are useful for classification of its inputs and the ReLU penalizes all the features that are negative or to small by thresholding them to zero. This effect is beneficial because the output of a neuron is going to be mostly zero, making it sparse. Similar to what was seen in Section 5.3, when the net is forced to find sparse representation, it will find the most useful information for classification and turn it into a set of features at the output of the hidden layer.

On the receiver side of the AE, a deep net is used to reconstruct the input and classify it. The first layers are usually dense layers with or without non-linear application, similar to the previous scenario where the AE tries to reliably reconstruct the whole spike shape. The last layer has now a different kind of activation function: the aim of this layer is no longer to output a real number representing the input sample
but to determine to which cluster/neuron this particular spike belongs. Assuming there are $K$ classes, e.g. spike sorting was ran on an initial recording and $K$ different neurons are identified; then the task of the AE is to correctly classify future spikes into these $K$ classes based on the hidden layer outputs. Hence, the last layer will consist of $K$ neurons and the softmax activation function is used, this ensures the output will be a probability distribution for that particular spike. The softmax function is defined as:

$$p_n = \frac{e^{z_n}}{\sum_{j=1}^{K} e^{z_j}}$$  \hspace{1cm} (5.19)

where $z_n = \sum_{i=1}^{n} w_i \cdot x_i$ as in Eq. (5.17) represents the inputs of the $n$-th neuron in the layer multiplied by its weights. The output of the last layer will consist of a vector $P \in \mathbb{R}^K$ with $p_i \in (0, 1) \forall i = 1, ..., K$ and $\sum_i p_i = 1$ hence, where every $p_i$ represents the probability that the spike belongs to the $i$-th class. Since classification implies that a certain spike belongs only to one class, the output class is taken as $i = \text{argmax}(p)$, with $\text{argmax}$ being the function that indicates the index of the maximum element in $P$. This activation function is particularly expensive; but again it is implemented on the non-critical non-implanted side with the rest of the decoder layers.

The training of AE for classification purposes is a supervised learning task; in fact a list of spike classes associated with spike shapes has to be provided. Hence a short period of the neural signal can be recorded and analyzed with the desired spike sorting program; afterwards the results are used to train an AE and from there on the classification will happen on-chip. Due to the nature of the spikes, the AE might require to be retrained when new spikes appear. But thanks to its denoising ability, it should stay stable against changes due to the electrodes drift and different noise conditions. Also thanks to the softmax function, when the vector $P$ doesn’t have a clear maximum for too many spikes, it can be taken as an indication that spike sorting need to be done again, because a new neuron might be active now.

## 5.6 Summary

The following summary provides an overview of the different compression schemes presented in this chapter, along with an evaluation of their properties.

**Entropy Encoding:** Compression using only signal’s statistic.

+ Lossless,
- CR dependent on probability modeling,
- difficult to achieve high CR due to modeling,
  + can be combined with differential encoding for improved performance.
Differential Encoding: Combines predictor, possibly a threshold, and an entropy encoder.

+ Flexible, can be lossless or lossy,
+ trade-off between predictor complexity and CR,
- relies on signals temporal and/or spatial correlation,
+ compression of spike frames and whole signals.

Compressed Sensing: Direct sampling at reduced data rate on implant with computationally heavier reconstruction at the receiver.

- Inherently lossy,
+ CR determined by the size of the sampling matrix,
- relies on sparse representation of the signals,
+ compression of spike frames and whole signals.

Machine learning: Neural networks trained on neural signals:

+ Flexible can be used as a predictor, for direct compression or spike classification,
- requires computationally intense training on data.

Autoencoder: Neural network with reduced complexity input layer on implant and computationally heavier reconstruction layers at the receiver.

- Inherently lossy,
+ CR determined by the network structure,
+ extrapolates features from data using machine learning,
- compression/classification of spike frames only,
+ can be used for compression or direct classification.
Chapter 6

Comparison of Compression Schemes for Neural Signals

In this chapter, the results from comparing the different compression schemes are reported. In particular a large part of this chapter is dedicated in the comparison of lossy compression methods. In literature, the most common scenario is that only spikes are extracted and compressed; then these spike frames are processed by means of custom made programs to do spike sorting. Although the algorithms implemented are the ones used and described in spike sorting programs (see Chapter 3) the implementation details are not known. Since in many neuroscientists labs, algorithms are not likely to be rewritten but available spike sorting programs are more likely to be used, the following analysis uses available spike sorting programs, without modifications, to provide a realistic use case of compression for neural signal acquisition.

6.1 Comparison of Lossless Compression Using Predictor Schemes

In the previous chapter differential encoding with predictors was introduced in Section 5.2 in the following performances in terms of CR for different predictors are investigated further. In this section the aim is to find, which predictor is more suitable for neural data compression and how many memory elements and arithmetic operations are needed in order to provide meaningful CR. This analysis is run on a subset of the recordings used in the following Section 6.2 due to the amount of simulations required for sweeping through the parameters of the LNN predictors and only lossless compression is considered. Once suitable predictors are found they will be compared to other compression methods in the remaining of this chapter. Lossy compression is investigated in the subsequent sections, when the results from lossless compression are used to compare the complexity of the predictors and different algorithms for compression are compared against each other.

The plots presented in the rest of this subsection compare CR against different number of taps/memory elements for LNN. This is the only predictor of the ones
Chapter 6. Comparison of Compression Schemes for Neural Signals

Figure 6.1: Linear predictors CR lossless comparison plot on tetrode data from low-noise recorded datasets in hc-1 for different number of memory depth (taps) of the LNN.

Presented in Chapter 5, that has a customizable parameter. The other methods have no such degree of freedom, and hence they appear as a straight line in the plots. The taps in Fig. 6.1 indicate memory elements used by LNN. Depending on the configuration (LNNT, LNNS, LNNST) the taps can store a sample from one or more channels, depending if spatial information is used in the prediction. For LNNS and LNNST, though the memory elements/taps are the same used by LNNT, still the number of multiplications would increase since they utilize samples from neighboring channels, too. For the following plots the training of LNN is done on 250 ms of neural data.

In Fig. 6.1 the CR obtained with different linear predictors for low-noise recorded datasets in hc-1 is shown. For delta and slope predictors there are no parameters to be set, hence the plot has the taps used for the linear neural network setup on the x-axis. Delta compression is the simplest predictor, achieving a CR ≈ 3.5 when using samples from the same channel (SDC) and slightly more than CR = 3, when using only spatial information on different channels (IDC). Although complexity is not the focus of this chapter, SDC requires additional registers in order to de-serialize the input and compute the differences on samples belonging to the same channel.

The slightly more complex slope predictor can achieve CR ≈ 4.4, about 22% better than delta compression, at the cost of a shift and an additional subtraction operation. Further increasing the computations and the complexity of the predictor, LNN can be used. The number of taps in Figure 6.1 represents the number of past samples that are being used by the network for the prediction. Depending on the
6.1. Comparison of Lossless Compression Using Predictor Schemes

configuration of the net, samples can be from the same channel (LNNT), from different channels (LNNS) or both (LNNST). When using also spatial information the net requires more multiplications since the prediction is made using the same number of taps but for more channels. Thereby the number of memory elements is the same for LNNT and LNNST because both nets have to store past samples for all channels.

From Fig. 6.1 LNNS is clearly the worst performing of the linear predictors, implying that spatial information alone is not very useful for these predictors, at least for the used datasets. LNNT and LNNST outperform all the other methods. It seems there is no significant difference for these datasets between the two methods, confirming that spatial information is not providing large benefit for this data set. When using LNNT with about 5 memory elements, the $CR \approx 6.2$, about 70% increase compared to SDC. Further increasing the number of taps does not significantly increase the CR. This is quite beneficial as it might be possible to implement LNNT using only as much as 5 multiplications per channels and still achieving almost the maximum CR for this method.

The same analysis results are plotted in Fig. 6.2 for recorded datasets in hc-2 [126] obtained from awoken mice during activities. The overall CR is lower than Fig. 6.1 due to the increased amount of noise in the hc-2 sets. The same considerations as the previous results are still valid in this scenario, with perhaps the only difference being that LNNS and IDC are far inferior to the other predictors. Again LNNT and LNNST are the best performing with a negligible improvement of compression for LNNST and the CR saturating for both after 5 taps. This makes it possible also for this dataset to achieve the maximum compression with $CR \approx 5$ at

![Figure 6.2: Linear predictors CR for lossless comparison plot on tetrode data from dataset in hc-2 for different number of memory depth (taps) of the LNN.](image)
Chapter 6. Comparison of Compression Schemes for Neural Signals

In Fig. 6.3 the predictor based schemes are compared using tetrode recording from in-vitro recordings of retina ganglion cells as well. As seen for the previous datasets spatial information does not help to increase the compression and again interchannel delta compression and LNNS are the worst performing predictors. LNNST is even slightly lower than LNNT alone, and its increased number of multiplications to include spatial information for the prediction makes it again a sub-optimal choice. This indicates that the channels are becoming less correlated making it harder for the predictor to use spatial information while autocorrelation (LNNT) is more effective for prediction. Also for this scenario the slope predictor offers more compression compared to SDC but not as significant as it did in case of hc-1 data, Fig. 6.1 although its hardware cost is still comparable to delta compression.

To fully validate LNNT as a predictor, the whole in-vitro data from the 65 × 65 channel MEA was used and the CR for single channel lossless compression with LNNT is reported in Fig. 6.4. An average data reduction of 5.6 times is achieved with a standard deviation of approximately 0.28; this demonstrates that the compression can effectively reduce the data rate and is overall quite stable over the whole array. One row, 57, appeared to be broken in the recording and it is possible to see that here the compression is worse than the average. Using spatial information combined with temporal information (LNNST) in this dataset does provide a slight advantage resulting in an average compression of 5.8 times, only slightly better than compression per single channel. Still, for the large array at least it could be proven, that spatial correlation can be useful for compression, as LNNST is slightly better than
6.2. Comparison of Lossy Compression Schemes

In the following, different SoA schemes for compressing neural signals are compared between each other. Since the approaches are inherently lossy (with the exception of delta and LNNT when the threshold is set to zero) the evaluation method defined in Chapter 3 needs to be used to evaluate compressors effectiveness in retaining information, while significantly compressing the data. The two different scenarios from Fig. 3.1 are separately examined with extended usage of neural data from simulation, in order to be sure about the presence of spikes and actual recordings for the whole signal processing.
6.2.1 Compression of Spike Frames

In this section compression of only spike frames is analyzed. As shown in Fig. 3.1 when spike detection is done on-chip, only the spike frames need to be compressed. Here only the effects of compression of these spikes are considered and it is assumed that spike detection is already ideally done. For the analysis, only Waveclus is used for spike sorting, since Osort does not offer a direct way to sort only the spikes without first detecting them. Also only simulated data is used since for recorded data no ground truth of the spike locations is available, meaning that there is no sure way to tell if a spike is a real spike or just a noise frame.

Not all compression schemes presented are useful in this scenario, in fact only delta compression (DC) and compressed sensing (CS) will be compared here. For the other schemes: non-uniform quantization will not be suited since spikes are already detected, hence there is no need to use different resolutions. Out of the linear predictors delta compression makes most sense; due to the nature of the data training, a LNN is not effective as it is for the whole neural signal. Since spikes are fast changing signals the slope prediction does not fit nicely and offer sub-optimal performance compared to delta compression. Autoencoders are well suited, being more similar to CS and they will be investigated later in this chapter; because to properly evaluate their compression performances, architecture and hardware implementations have to be investigated, too.

The spikes used for the evaluation are extracted from Easy1 simulated datasets in [55], since it offers a high noise variance to explore the effects of noise directly on the compression scheme. In the case of CS where training is needed half of the spikes are used for training the dictionary.
6.2. Comparison of Lossy Compression Schemes

In Fig. 6.5, classification accuracy (CA) from Eq. (3.2) is plotted against various CRs for spike frames and for low noise data. In this scenario, with spikes locations known and their shapes aligned, CS offers superb CR, achieving above $30 \times$ data reduction for almost unaltered classification, which are similar numbers as reported in prior work [112]. Lossy delta compression performs poorly and its accuracy drops almost immediately for CR above 3. Since the idea of lossy delta compression is to threshold away the smaller differences, high CR can not be expected, if the noise is basically eliminated and only spikes are extracted for compression. Furthermore, in order to compress the data at all, the threshold for DC has to be chosen high, which results in a very distorted spike shape, hence low CA.

In Fig. 6.6, equivalent results as in Fig. 6.5 are shown for simulated dataset Easy1 with 0.35 noise variance, which features a $3 \times$ higher noise as the previous example. In this scenario, CS still achieves much better results than delta compression, but its CA performance is heavily impared by the increase in noise. If high accuracy is sought the CR for $CA \geq 0.9$ drops by more than a factor of 6 compared with the low noise case. Also interesting to note, CS does not achieve 100% CA any more, not even for small compression rates, which is in contrast to DC with low or no losses at low CR.

6.2.2 Compression of Unframed Neural Signals

In this scenario the spikes are not detected but the whole neural signal goes into the compressor. Again the method illustrated in Chapter 3 is used to assess the retain of information, quantified by the classification accuracy (CA) from Eq. (3.2).
In Fig. 6.7, CA against CR is plotted. CS based methods with data trained dictionaries using K-SVD [151], SDNCS and SDNCS-P with threshold detection aid [112], are the worst performing schemes for this kind of scenario. CS can achieve $CA \geq 0.9$, but at the cost of very modest CR. For higher CR it is not possible to obtain useful spike information from the compressed data. It is surprising that SDNCS-P doesn’t achieve better results than CS but the reason lies in the choice of the threshold detector used for the compression. While Osort uses a wavelet method to detect spikes, SDNCS-P uses an absolute value threshold, for reduced hardware complexity. Hence the spikes, that are detected by the compression algorithm, can differ from the ones detected from Osort. This can happen, e.g. due to low amplitude spikes not detected by the threshold. Furthermore spur spikes from noise can be detected and wrongly treated as actual spikes, which results in using the spike dictionary and reconstructing a known spike shape for a noise segment.

Standard CS with data dictionary is more suited since it can reconstruct low amplitude spikes in segments with high noise levels, which comes at the price that sparsity of these segments is not low anymore and they require more samples, making CR quite modest.

Non-uniform quantization (Nuq) has a limited range of compression but preserves the spike sorting accuracy (CA) very nicely. For LNNT, lossless compression (CA=1) provides already a $CR \approx 5.6$. When a threshold is used CR increases by 50% in Fig. 6.7, but the CA significantly drops below 0.9, even when the threshold is set to 1 LSB, only. Quite interesting is the behavior of delta compression, which can achieve the same CR of lossless LNNT by sacrificing only 10% accuracy. Since
delta compression is expected to be much less power hungry than LNNT, this can be used to compromise little accuracy for CR without using more complex and power hungry predictors.

There are also some compression methods in Fig. 6.7 and successive figures in this section for which the CR stops early, e.g. Delta compression, this is due to the threshold. After moderately low threshold of a few LSBs the reconstructed signals is completely distorted and almost no useful information is retained. E.g. this is similar to Figures 6.5 and 6.6 where delta performance drops abruptly. In the comparison in this section these points have not been reported since they take a considerable amount of resources and don’t provide further useful information.

Similarly to Fig. 6.7 CA is reported for different CR, when using Waveclus as spike sorter [55] in Fig. 6.8 CA is much higher for all methods when compared to the Osort results in Fig. 6.7. This is explained by the different detection mechanisms. While more complex wavelet detection can find small amplitude spikes in high noise frames, the simpler threshold detection can only detect spikes with signal power much above the noise level. Hence the spikes that are eventually detected by Waveclus have high power and are easier to be preserved by compression. E.g. thresholding after prediction does not heavily affect high amplitude spikes.

The situation in Fig. 6.8 is similar to Fig. 6.7 with an important difference that SDNCS-P is now working much better than before. Note that now both compression and spike sorting algorithm are using the same detection method. The situation is then very similar to what happens when only spike frames are compressed, and thus SDNCS-P achieves much better results. Still, also in this scenario the CA is barely
above 0.9 for very modest CR. In fact the problem of reconstructing noise spikes as actual spikes is still present. While Waveclus can discard a spike because its features are correlated to noise and not an actual neuron, the SDNCS-P on-chip detector cannot distinguish between the two cases. The threshold detection in Waveclus is also helpful for delta compression and LNNT, since thresholding does not affect the high amplitude spikes too much. While for LNNT the lossy approach degrades CA very much for delta compression, the loss is almost negligible here with the same maximum threshold used for Fig. 6.7.

Note, it stayed unclear, why for $CR \leq 2$, CA drops but it seems to be due to the software and not actual degradation of the compressed signal. Although it was verified that for the same input the software outputs the same sorting results.

The various compression methods have been extensively tested also on data from hc-2 [126] and the results are report in Fig. 6.9 and 6.10 for Osort and Waveclus respectively.

Overall the results are similar to what was obtained for hc-1. But with the more noisy data, the CR is generally lower and the CA drops faster. Again Osort is more prone to data loss, when using CS and SDNCS-P than Waveclus, but the CA is only marginally better. Also for the other methods, even low values of thresholds for LNNT and delta compression make the CA drop fast. The increased noise level requires higher thresholds to achieve the same CR as could be obtained in hc-1, which results in higher distortion of the signal, hence lower CR for the same CA or lower CA for the same CR. The same happens with Nuq, since spike frames are flooded with noise and identifying them is not trivial anymore. This combined with
6.3 Comparison of Autoencoders for Compression and Classification

In this section different architectures for autoencoders (AEs) are investigated in order to find the key parameters that allows it to provide meaningful compression. As seen in Section 5.5 there are two main use cases for AEs: compression of already extracted spike frames and direct classification. In the following sections these are individually addressed. For training the AE half of the spike frames for every set are used. This ensures that the results are obtained on signals that the network didn’t actually see during the training phase.

6.3.1 Compression of Spike Frames with Autoencoders

When compressing spike frames the same procedure used in Section 6.2.1 and described in Chapter 3 is adopted. The spikes are extracted, using ground truth, from simulated datasets in [55], then the frames are compressed and reconstructed with an AE and processed by Waveclus and finally the sorting is compared to the uncompressed spike frames sort results. In this scenario the AE is trained using unsupervised learning.

The threshold makes Nuq to quantize the noise part with higher resolution, hence with low CR, or too many segments of a spike quantized with too low accuracy, therefore the CA drops.

Figure 6.10: Comparisons of different lossy compression schemes for data in hc-2 [126] when compressing the whole neural signal. CA is obtained by analyzing the data with Waveclus [55].
One of the key parameters in determining the CR and the reconstruction ability of an autoencoder (AE) is its hidden dimension. This refers to the number of neurons in its hidden layer (Fig. 5.15) and represents the bottleneck of the network. In Fig. 6.11 AEs with different hidden states are used to compress spikes coming from simulated data of [55]. The first AE analyzed is composed of only 3 layers: input, hidden and output. This is the most simple architecture possible for such a network. A deep autoencoder with 3 layers at the decoder side, instead of just one, is included in the analysis.

In Fig. 6.11 CA = 1 is obtained for datasets with low noise, but CA drops abruptly for all sets, when the noise levels are increased. Given the low number of hidden units, for all used AEs the spike shapes at the output can not be preserved well enough to allow Waveclus to do proper classification. From Fig. 6.11 it turns out that having a deep autoencoder, using more layers on the decoder side, does not in general provide a reasonable advantage over using non-deep architectures for this scenario. In fact from Fig. 6.11 the non-deep AEs perform better compared to their deeper counterpart. This, could indicate that the deeper AEs require more spike frames for training than what is available in the simulated data sets.
6.3.2 Spikes Classification with Autoencoders

In spike classification the spike frames are extracted and the ground truth labels provided with the simulated data are used to train the AE in classifying correctly the spikes (supervised learning), the AE classification results are then directly compared to the ground truth.

In classification, AE has superb performances. In Fig. 6.12 the CA for simulated data sets for all different noise levels from [55] is shown. The spike frames are extracted with a length of 64 samples around the spike event and then classified with the AE. As it can be seen, the AE is able to correctly classify against ground truth more than 90% of the spikes for almost all data sets with as little as only one neuron in the hidden layer. Using two neurons a CA > 95% is achieved for all but 2 datasets.

A very interesting result comes from the fact that the CA is calculated against the ground truth provided and the AE shows its denoising function. In fact the classification remains very high also in presence of high noise levels, meaning that the network can extract some stable features for every cluster also when having only the noisy version of the spikes used for training. From these features it is also able to correctly classify the other noisy version of the spikes provided as a test set after training. The classification accuracy of the AE using ground truth exceeds even the one for the used spike sorting algorithms [55], which suffers much more from the increase of noise levels. This encouraging result could also indicate that a method of using AEs to classify spikes without using any prior information is possible. Once a pre-spike sorting operation is carried out, the AE can recover the spikes with great accuracy, Fig. 6.12.

In terms of compression this can be used to directly obtain the classification of the spike by using the output of the hidden layer. Even when using a full precision autoencoder, the hidden layer outputs are at most 32 bits; hence the transmitted data results in 32/64/96 bits respectively for 1/2/3 hidden neurons. In the performed simulations, the data was quantized with 12 bits per sample and 64 samples are used per spike. This results in $CR = 8 - 24$. While being already a great reduction in data rate, this is still a pessimistic assumption: it is in fact not very practical to run 32 bits operations on the implanted side, hence when reducing the precision the expected CR should increase. At the same time no big loss of performance is expected since the AE performances are stable under presence of noise.

Hence to make AE competitive against other methods like CS, it has to be simulated using lower resolution for the coefficients in the encoder layer and its output has to be quantized to resolutions using less than the standard 32 bits single precision used in ML problems.
In a BMI interface scenario, the input to the net will consist of neural data coming from the ADC, hence it is represented by integers with typical resolutions of 12 to 16 bits. This is problematic for optimizers as they run easily into numerical problems and the training is not effective. This can be solved by training the network on normalized floating point input, and then rescale its weights. There are two benefits from this procedure:

- Standard optimizers provided with ML frameworks can be readily used.
- After scaling, the weights are rounded to the nearest integer, eliminating the need for floating point operations on-chip.

This last point can be pushed even further by having the weights not only rounded but also approximated by a power of 2. Thus the multiplications are substituted by only one shift and a summation/subtraction operation, greatly reducing the hardware complexity and storage requirements for the coefficients.

In Fig. 6.13, classification results are shown, similar to Fig. 6.12 but for AE structures with quantization for the encoding layer and weights scaling. As it can happen in CS that some choices of the random sensing matrix don’t lead to optimal performances, also here this was noticed in regards with the random initialization of the network weights. This can greatly affect its performances after scaling. On average the process was found to be quite stable and the std was at a maximum of 17% for worst performing datasets. For Fig. 6.13, the average performance for 12 runs on
every dataset are reported. It is shown that classification results are quite good for Easy datasets, maintaining $CA \geq 0.9$ for almost all of them even in the presence of noise, excluding high noise Easy2 dataset. For Difficult2 datasets the CA degrades rapidly in the presence of noise although $CA \geq 0.8$ for all AEs in Fig. 6.13. The DriftEasy2 dataset is the most problematic one for all considered AE, here the spikes change shape in time to simulate the effect of an electrode’s drift [55]; a good part of the spikes are then not recognized anymore by the AEs since their shapes, and hence features, are changing. Still, when provided with enough resolution the AEs are able to properly deal with the dataset, as shown in Fig. 6.12.

A visible trend for classification in Fig. 6.13 is that higher number of hidden units fits the data better than an increase of bits used for every unit. In fact when using 4 neurons with only 2 bits output the AE performs better than 3 neurons and 3 bits per output. Also there is no significant change in performance when using 3 neurons but with different quantizations at the output; the only observable difference comes when only one bit is used, then the CA drops for most datasets.

This fact, that it is better to increase the number of output layer neurons instead of the number of bits for their outputs, can indeed be explained, when considering the working of the AE: due to the bottleneck by the hidden layer, the encoding side is forced to learn a compact representation of features for the data [160]. Unlike the data samples, the features don’t need an accurate quantization since their information is more an on-off representation saying the feature is meaningful, or not, for a specific input. Hence, even when only 2 bits are used to represent the output of a
hidden layer, these are enough to indicate when a feature is present and, if so, with
a high or low value. These facts make the AE a good candidate for hardware imple-
mentation. In fact the number of neurons in the hidden layer can be kept as low as
4, reducing the number of operations and the number of bits to be transmitted.

6.4 Summary Evaluation of Compression Schemes

In this chapter a comparison of the performance of neural data compression schemes
has been presented. In Table 6.1 main features for different compression schemes
from SoA and this work are summarized. Having many different approaches justi-
fies the need to have a framework for comparing the compression methods.

When only spike timestamps are transmitted no further analysis is possible but
the CR is the highest of all methods in Table 6.1. When only spike epochs are retained
the efficacy of this method has to be compared to other spike detection methods.
Hence it cannot be applied to the comparison method developed in this thesis.

As in literature, CS is the accepted best performing algorithm to compress neural
signals. It is crucial to verify, which conditions allow to use its maximum potential.
It turns out that, when knowledge about the spikes is provided and compression is
done only on these segments, CS achieves great compression performances. CS can
in fact very well compress the spike shapes given that their temporal information
is provided. When dealing with the raw neural signal, temporal information about
the spikes needs to be retrieved before the compression. Implementing a threshold
detector on chip is easy in principle but the choice of the threshold value is critical,
as shown by our simulations. Spurious noise spikes have a detrimental effect on
the overall sorting accuracy since they will be reconstructed as spikes. Furthermore
high noise levels impair the reconstruction of spike shapes which results in spikes
being missed.

In transmission of the whole neural data scenario, LNNT is the best perform-
ing method for lossless compression, although it does not cope well with the intro-
duction of a threshold that will alter the effectiveness of the predictor, leading to
an abrupt drop of performance. An interesting case for low noise recorded data is
lossy delta compression, which achieves high accuracy and better CR as LNNT. This
is indeed interesting, since delta compression requires much less hardware, when
compared to LNNT.

When comparing the two spike sorting programs, there is not a huge difference.
Only Osort is less robust to compressed signals, which is due to the more sophisti-
cated detection method being implemented, which is able to resolve neural spikes
even in presence of high amplitude noise.
6.4. Summary Evaluation of Compression Schemes

<table>
<thead>
<tr>
<th>Compression Method</th>
<th>Harrison ’07 [24]</th>
<th>Biederman ’15 [161]</th>
<th>Zhang ’15 [150]</th>
<th>LNNT (This Work)</th>
<th>AE (This Work)</th>
</tr>
</thead>
<tbody>
<tr>
<td>CR</td>
<td>150×</td>
<td>8.3×</td>
<td>4 − 16×</td>
<td>4 − 7×</td>
<td>128× †</td>
</tr>
<tr>
<td>Compression</td>
<td>Timestamps</td>
<td>Epochs</td>
<td>Lossy</td>
<td>Lossless/Lossy</td>
<td>Classification</td>
</tr>
<tr>
<td>Reconstructed SNDR</td>
<td>-</td>
<td>-</td>
<td>&lt; 10dB</td>
<td>'Inf'</td>
<td>-</td>
</tr>
<tr>
<td>Spike Sorting</td>
<td>×</td>
<td>Only Clustering</td>
<td>✓</td>
<td>✓</td>
<td>Only Clustering</td>
</tr>
<tr>
<td>CA</td>
<td>-</td>
<td>-</td>
<td>95% @ 10× *</td>
<td>100% @ 7×</td>
<td>≥ 95% *</td>
</tr>
</tbody>
</table>

Table 6.1: Summary of compression methods in literature and presented in this work.

†Compression of spike frames
* On spikes from synthetic data

One final consideration regards AEs: it was proven that the CS based methods are superior to delta compression for compression of spike frames, providing about > 30× data reduction on low noise data, while AEs could only offer a maximum of CR = 24. When transmitting only spike frames the aim is to be able to do classification off-chip, the AEs can be used for this purpose directly. The spike features are extracted by the hidden layer and transmitted to the decoder which classifies them. Only the features need to be sent, hence achieving data reduction of the spike frames. Furthermore this method proved to be superior to CS in terms of CA and allows for large hardware reduction, which in turn reduces the amount of data to be transmitted for features, resulting in a CR = 128 for almost perfect classification accuracy for very low noise data sets.

Shown in Table 6.1 the AE classification achieves a CR not much lower than retaining only spike timestamps, but providing information about the spikes class as well. Hence, perhaps sending only spike timestamps or compressing spike frames might not be very beneficial in the end and direct classification can greatly reduce the data rate within very acceptable losses of CA.
Chapter 7

Hardware Evaluation

In this chapter an analysis is carried out to identify which of the presented methods is feasible to be realized in custom hardware and also for implantation. So far the algorithm complexity was kept as low as possible, by realizing only linear filters and for the AE by limiting the operations to only shifts in order to reduce complexity. Since the final goal is to have hardware that can actually be implemented in an implantable circuit, here also power estimations for various compression schemes, are reported with an analysis of their feasibility in a 180nm CMOS technology.

7.1 Hardware Requirements

The first issue to be assessed is when it is advantageous to use compression in neural implants. The whole point of compression in an implant is to reduce the power required for transmission in order to enable wireless communication. Shown in Fig. 7.1 is the transmitter scheme for a neural interface. After the ADC, the data at full data rate is elaborated by the compressor, which delivers a compressed data stream to the transmitter. In inverse order the receiver feeds the compressed stream to the decompressor and eventually the un-compressed data stream is available.

As discussed in Chapter 2, transmitters can easily be the most power hungry component on-chip, hence the power saved in the transmitter can be allocated to the compressor. The very generic condition that needs to be met would then be:

\[ P_{C \text{Stream}} < P_{\text{RawStream}}, \]

Figure 7.1: Transmitter and receiver scheme, including compression and decompression
Chapter 7. Hardware Evaluation

with \( P_{\text{CxStream}} \) and \( P_{\text{RawStream}} \) being the power for transmission with the compressor and without, respectively. Hence \( P_{\text{CxStream}} \) includes the reduced power of the transmitter and the compressor power, while \( P_{\text{RawStream}} \) is the power required to transmit the full stream without any compression. For saving power, the inequality makes sense only in strict sense. In fact if \( P_{\text{CxStream}} = P_{\text{RawStream}} \), it is not worth to bother with compression, which would not provide power reduction. In the case, where compression is used to increase the maximum number of channels that can be transmitted, then the equality makes sense.

In the power analysis, the receiver end is not included, since it will not be implanted and hence power is virtually not an issue. For the transmitter, the first observation is that the compressor and the Tx work on two different data streams. The first elaborates the full data stream \( D = N_c \cdot f_s \cdot N_b \) with \( N_c \) and \( N_b \) number of channels and bits per sample, respectively, and \( f_s \) sampling frequency, while the Tx receives a reduced data stream: \( D_c = D_{\text{CR}} \).

The common energy per bit FoM for the transmitter \( E_{\text{Tx}} \) can be used, and an equivalent one can be defined for the compressor: \( E_{\text{Cx}} = \frac{P_{\text{Cx}}}{D} \), where \( P_{\text{Cx}} \) is the power of the compressor. \( E_{\text{Tx}} \) and \( E_{\text{Cx}} \) are then the energy to elaborate one bit in the transmitter and the compressor, respectively. By substituting this Eq. (7.1) becomes:

\[
E_{\text{Tx}} \frac{D}{CR} + E_{\text{Cx}} D < E_{\text{Tx}} D.
\]

And after manipulating this equation, it can be rewritten as:

\[
E_{\text{Cx}} < E_{\text{Tx}} \left( \frac{CR - 1}{CR} \right). \tag{7.2}
\]

Eq. (7.2) is the condition that needs to be met by the compressor in order for it to be useful power wise, i.e. reducing the total power required for transmission.

When there is no compression \( CR = 1 \), and Eq. (7.2) gives \( E_{\text{Tx}} < 0 \), which confirms the intuition that if no compression is done then the compressor should also require no power. When CR becomes large then \( E_{\text{Cx}} \approx E_{\text{Tx}} \), in fact the more the compressor reduces the stream, the more power it can use. But the compressor is never allowed to have the same energy per bit as the transmitter (or even higher); since \( E_{\text{Cx}} = E_{\text{Tx}} \) would mean the compressor is consuming as much power as the transmitter would to transmit the uncompressed data stream. Since both compressor and Tx are needed this would violate the condition in Eq. (7.1).

Eq. (7.2) allows for comparing different architectures for Tx and compressors by requiring only their energy efficiency per bit. Assuming that these quantities scale linearly with the data rate they will then be independent of the number of channels.
### 7.1. Hardware Requirements

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Data rate [Mbps]</td>
<td>128</td>
<td>24</td>
<td>1</td>
<td>0.346</td>
<td>200</td>
<td>1</td>
</tr>
<tr>
<td>Tx Power [mW]</td>
<td>561</td>
<td>90</td>
<td>24</td>
<td>0.72</td>
<td>0.3</td>
<td>5.52</td>
</tr>
<tr>
<td>Efficiency [nJ/bit]</td>
<td>4.383</td>
<td>3.75</td>
<td>24</td>
<td>2.081</td>
<td>0.0015</td>
<td>5.52</td>
</tr>
<tr>
<td>Modulation</td>
<td>OOK</td>
<td>FSK</td>
<td>ASK</td>
<td>FSK</td>
<td>Return to Zero</td>
<td>OOK</td>
</tr>
<tr>
<td>Measurement</td>
<td>Body-equivalent liquid</td>
<td>Implanted</td>
<td>Implanted</td>
<td>Implanted</td>
<td>Bench with bio tissue</td>
<td>Bench with bio tissue</td>
</tr>
</tbody>
</table>

**Table 7.1:** State of the art specifications of transmitters used in BMIs.

and sampling frequency\(^1\) Still CR needs to be accounted for, and as seen in the previous chapter, this is dependent on the kind of data being compressed.

State of the art Tx specifications for implantable devices are reported in Table 7.1. Only one transmitter achieves an efficiency below the \(nJ\/bit\) and higher than 100Mbps data rate, although measurements are still carried in a fully-controlled environment and the antenna occupies a fairly big area of 10mm \(\times\) 10mm. From the Table and using Eq. (7.2), it is possible to calculate the maximum allowed power consumption for a compression method given its CR for these state of the art Tx. Preliminary, since the power of the compressor has to be strictly smaller than the power of the transmitter, to be efficient enough a neural compressor needs to use less than few \(nJ\) per bit for most of the SoA transmitters in Table 7.1.

Equivalently when having an estimation for the energy efficiency of the compressor it is possible to find for which implants it would be suited, in terms of power savings. By the condition given in Eq. (7.2) it is interesting to calculate the minimum CR necessary to obtain power savings. By inverting and manipulating Eq. (7.2) the following relation is obtained:

\[
CR > \frac{E_{Tx}}{E_{Tx} - E_{Cx}}. \tag{7.3}
\]

When developing the compressor a target implant [42, 103] is considered. From this, key features for compression are determined:

- spectral separation of APs and LFP
- 16 bit resolution ADC

\(^1\)Even though this assumption might be hard to prove in Tx it is not strictly required as \(E_{Tx}\) and maximum data rate can be fixed and then the compressor can be chosen based on its achievable CR and \(E_{Cx}\). For digital compressors it turns out that \(P \propto f^{162}\) when considering the most dominant dynamic power. This will be further discussed in Section 7.2.2.
Chapter 7. Hardware Evaluation

- 32 channels
- \( f_s = 20\text{kHz} \).

Having the AP and LFP separated on-chip allows to do compression only on the high frequency component, without caring about the LFP. Regarding wireless data transmission an UWB transmitter and miniaturized antenna \([163, 164]\) were realized to achieve 100\(\text{Mbps}\) data rate; the power consumption was found to be between 3 and 4\(\text{mW}\), depending on the antenna and the distance between the Tx and Rx leading to an efficiency of \( E_{Tx} = 30 - 40 \frac{\mu\text{J}}{\text{bit}} \), a similar range as in \([100]\).

In the remaining of this chapter, feasibility and hardware implementations are discussed for the various compressors presented in this thesis with particular focus on LNN since they offer the opportunity to increase the CR substantially compared to other linear compressors, e.g. delta compression.

7.2 Feasibility of Hardware Implemented Linear Predictors

Linear predictors are based on simple arithmetic operations, hence the assumption that these predictors will not be too power hungry and simple enough to be used for implementation in an implantable device.

7.2.1 Implementation of Delta Compression

Delta compression has been synthesized in a 180nm CMOS technology and consumes about 44\(\mu\text{W}\) for a 32 channels implant, including the Huffman encoder. This results in an efficiency \( E_{Cx} = 4.3\frac{\mu\text{J}}{\text{bit}} \). From Table 7.1 this compressor is not suitable to be used with the transmitter in \([100]\) since \( E_{Cx} > E_{Tx} \). When comparing with Tx in \([94]\), which is also the work with best efficiency measured in-vivo in Table 7.1, and by using Eq. (7.3), it turns out that a CR slightly greater than 1 provides beneficial effects for power savings. From previous chapters, delta compression could achieve \( CR \geq 2 \), which makes it very desirable in BMI to reduce the overall power. The advantage will be proportionally greater for implants with less efficient Tx as for most other SoA implementations in Table 7.1.

7.2.2 Implementation of a Linear Neural Network

Out of the linear predictors highlighted in this thesis, only the LNN requires multiplications, which are quite expensive operations on a power constrained hardware. Hence, the first step is to analyze the power required for a multiplication.

From Fig. 5.12 it was already shown that the performances of LNNT begin to decrease when the number of taps is \( \leq 4 \). In order to reduce the power of the compressor, only the minimum required amount of multiplications should be used. The
7.2. Feasibility of Hardware Implemented Linear Predictors

Figure 7.2: Effect on CR when varying the number of bits of the coefficients for LNNT predictor.

The number of taps for the power analysis is fixed at 5 or 10 allowing to choose a slightly sub-optimal inexpensive setting or a more costly almost full performance version, respectively.

Binary multiplication between two numbers (called multiplier and multiplicand) consists in shifting the multiplicand by a proper amount and summing it to the product, when the bits in the multiplier are 1s. There are then two main approaches to this: the first is a combinational approach, which is also called the array multiplier [162]. This approach is quite expensive: for N bits in the multiplicand and M bits in the multiplier then the following resources are required:

- \( N \times M \) 2-bit AND gates for generating the partial products
- \( N - 1 \) M-bit adders to sum the partial products.

To reduce the requirements, a second approach called shift and accumulate using only one adder but requiring more cycles is also possible. Instead of calculating everything at once, the multiplicand is shifted and depending on the multiplier bits summed to a partial product register, which at the end of the multiplication yields the desired result. This approaches will take instead only a \( (N+M) \) bits adder and \( M \) cycles [162].

Independently from the architecture, there are already two parameters that will determine the number of operations and complexity of the multiplier: \( N \) and \( M \).

Fig. 7.2 shows the CR against different depth of the taps used in LNN for lossless compression. Tetrode recordings from hc-1, hc-2 and MEA recordings from NMI where taken and the coefficient resolution was lowered to see how this affects the CR. It is clear from Fig. 7.2 that below 8 bits resolution there is a significant drop in compression for both the 5 and the 10 taps case. This means that while the resolution...
of the input is fixed (16 bits for the target implant) the coefficients can be encoded with half the bits.

The two different architectures have been synthesized using VHDL in a 180nm CMOS technology to evaluate their power consumption and feasibility for a neural implant. Unfortunately, being a technology developed for high-voltage and implantable devices, its featured size doesn’t make it competitive enough for digital processing.

For the combinational approach the efficiency was found to be \( E_{\text{bit}} = 26 \text{pJ/bit} \) and for the shift and accumulate approach \( E_{\text{bit}} = 36 \text{pJ/bit} \). It is not surprising that the combinational approach is more efficient since it can run at lower frequency. This is because the dynamic power consumption is given by [162]:

\[
P_{\text{dyn}} = \alpha f C_L V_{DD}^2
\]

where \( \alpha \) is the switching activity factor accounting for the probability that a signal transition will draw power from the supply (generally when switching from 0 to 1), \( f \) is the frequency of the logic, \( C_L \) the capacitive load and \( V_{DD} \) the power supply voltage.

To extend the analysis, it is possible to apply low power techniques to the multiplier in order to reduce its power consumption. One possible solution for this logic could be to dynamically turn off the multiplier when it is not needed. While this approach helps greatly in smaller node technologies it is almost useless for the considered 180nm technology node since it cuts the static power consumption. In fact the static power consumption is estimated to be about 722 \( \mu \text{W} \) while the dynamic power is 373 \( \mu \text{W} \). Thus static power is completely irrelevant to the overall power consumption.

To reduce the switching activity a ROM based multiplier, which uses look up tables to retrieve pre-computed partial products, could also be used at the expense of area. It was shown that this implementation can reduce power consumption by 40% [165].

One further problem in the realization of LNN is the implementation of the input registers. For an \( N_c \) channel implant with \( N \) bit resolution ADC and using \( T \) taps in the LNN, \( N_c \cdot N_b \cdot T \) registers are needed to store the input values for every channel. For a 32 ch implant with 16 bit resolution and 20kHz sampling frequency, 5 taps in LNN would require 2560 flip-flops, consuming 146\( \mu \text{W} \) at 20kHz in a 180nm technology. This alone result in an efficiency per bit \( E_{b} = 14 \text{pJ/bit} \).

The total power estimation for LNNT, including Huffman encoder, is \( P = 564 \mu \text{W} \) for 5 taps on a 32 ch implant hence with an efficiency for the compressor of \( E_{Cx} = 55 \text{pJ/bit} \).
Given the efficiency it is clear that this kind of compressor might be too power hungry for the Tx used in [100], but it should be remembered that [100] reports orders of magnitude better efficiency than most other implantable Tx. In fact its efficiency is already more than 3× worse than the one of the best Tx.

For the implant in [94], which has the best Tx efficiency for an in-vivo implant from Table 7.1, although the smaller datarate, the minimum CR for power efficient application of compression would be \( CR_{\text{min}} \approx 1.03 \). This was proven to be easily achievable for LNNT compressors and also other linear predictors in the previous chapter.

### 7.3 Feasibility of Hardware Implemented Autoencoders

In this section the hardware cost for the autoencoder classifier seen in Section 5.5.2 and 6.3.2 is analyzed. As seen in Fig. 6.11 for compressing spike frames, the autoencoder does a good job for low noise datasets. But its performance is heavily affected by noise, although double-precision floating-point format was used for the computations of Fig. 6.11.

An approach leading to much better CA was shown in Fig. 6.12, where the AE is trained to perform the classification itself and where it is not compared to results from the spike sorting program but against the ground truth provided with the simulated data. With this approach already with only 2 hidden states, the accuracy is \( CA \geq 0.9 \) except for one dataset, and with 3 hidden states \( CA \geq 0.95 \) for all the sets.

This second use of AE seems then much better suited to be implemented in hardware. Although different from the lossless approach this method would still suit the scenario, where only spike frames are transmitted. In fact the spike shape would be lost, since the AE only outputs the class, which the spike belongs to, but anyhow when only spike frames are kept the only operation possible is to sort them. Hence the autoencoder would fit into the whole signal chain by directly classifying the spikes.

From Fig. 6.13 the classification is worse compared to the ideal case, where full-precision floating point operations are used, but remains of \( CA \geq 0.9 \) for most of the datasets, when 4 neurons are used in the hidden unit.

An autoencoder with 4 hidden units with 2 bits quantized output working at a frequency of 1.28 MHz was designed and synthesized in the 180nm CMOS technology. The weights are quantized in order to be a power of 2, hence requiring only shift operation for multiplication. In this configuration the AE can then elaborate 64 samples of a spike in 50\( \mu s \), allowing elaboration to be done in one data cycle. For this configuration the AE estimated power consumption is of \( P = 27.8 \mu W \). The efficiency per bit yields \( E_c = 2.71 \mu J/\text{bit} \). This makes the AE less power efficient of the
Chapter 7. Hardware Evaluation

Tx in [100], but more efficient than all the other transmitters. For 16 bits quantized data and 64 samples spike window, it turns out that \( CR = \frac{64 \times 16}{4 \times 2} = 128 \).

For the Tx in [94], which has the best transmitter efficiency in Table 7.1 for in-vivo, this would turn in a power saving of about 710\(\mu W\) or almost 99\% power saving compared to the Tx alone. Such power saving is way above what can be offered by CS for spike frames. In fact, a CS implementation would consume 270\(nW\) per channel [112] but as shown in figures 6.5 and 6.6 CS is only able to achieve \( CR \leq 40 \) and especially for high noise data, even lower CR. This difference comes at the price that for AE there is no spike that can be reconstructed. Only a cluster index is output, indicating the neuron firing the spikes.

7.4 FPGA Proof of Concept

An FPGA implementation was also carried out during this PhD thesis. The prototype could be used for laboratory experiments in-vivo or in-vitro, where the amount of channels becomes critically high. Also for in-vitro experiments, a real-time compression is highly beneficial to reduce storage directly at the source.

The board available was a Zybo board from Digilent [166], which is equipped with a Xilinx Zynq-7010 All Programmable SoC [167]. Concerning the main specifications, the chip has 4400 logic slices, 80 DSP slices, 240KB of on-chip block RAM clocked at 450\(MHz\) and 512MB of external DDR3 RAM on-board.

To compress neural data, two interfaces with high throughput are needed for receiving the data from the implant and to feed it to the workstation. Unfortunately, on the Zybo board there is only one ethernet interface able to sustain a data rate of 1Gbit/s. For this reason the FPGA has been used on a few seconds of data to show that real-time elaboration is possible, but is limited by the board in this implementation.

The data was copied to DDR, then accessed using the DMA block provided with the Zynq processing system [168]. The DMA block directly connects the programmable logic to the DDR, hence avoiding to involve the ARM core and to allow real time operation on the neural data, provided the DDR RAM is fast enough. Once the data is transferred from the DDR, it is processed in the programmable logic using a custom block implementing the multiplications, which are needed to make the prediction for LNNT. When the compression is done the data is gathered by the DMA and transferred from the programmable logic back into the DDR. After completely compressing the whole pre-recorded data, the DDR is accessed again with the slower processing system and the data correction is verified.

The programmable logic runs at 100\(MHz\) and is implemented in a serial fashion for 10 taps of LNNT. The board could process 10\(MS/s\), which would be enough to
process 500 channels at 20kS/s. In Fig. 7.3, the utilization of the FPGA resources is shown along with the power report. For the DMA and prediction logic 70% of the look up tables (LUTs) are taken while only 1 DSP slice is used. The DSP slices are implemented into the programmable logic and used for the multiplication for the LNNT. In order to make better use of the resources, more could be deployed to parallelize operation. This was not done in this first proof of concept as the problem on how to simultaneously provide the input data with the DMA was not addressed here. In successive versions this improvement would directly lead to an increase of the maximum number of channels that can be elaborated.

The process would be suitable for real-time data compression. But having to store the neural data from a PC into the DDR and then from the DDR to the PC slows the process and its operation speed is limited by the maximum amount of memory available on board. For a board that provides e.g. 2 Gigabit ethernet interfaces or 2 HDMI ports, a design being able to compress the data and stream it on the fly would then be possible.
Chapter 8

Conclusion

Compression for neural signals was investigated in this PhD dissertation. An approach to evaluate signal quality after the compression is extended from previous work and extensively used to test different compression schemes on more than 10 hours of various neural recordings.

In literature it is not obvious to find a standard framework to evaluate neural signals, hence the same is found for compression schemes, making it hard, and sometimes impossible, to compare different methods. The approach used in this work is to evaluate compressed data by means of spike sorting software in order to find how much of the information is lost in compression.

A first step before talking about compression is to evaluate the minimum number of bits. It seems that for APs the provided 16bits might not be always necessary and about 10 to 12 should suffice. Linear quantization is still the best choice for simplicity and accuracy of spike sorting. While using different resolutions for spike and noise does not provide any benefit compared to a Linear Neural Network or Delta Compression.

Out of the various predictors examined in this work, Linear Neural Network represents probably the best choice provided that the transmitter efficiency is above $55\mu J/\text{bit}$, or that a smaller technology node is available and Linear Neural Network Temporal requires less power than what was estimated for a 180nm CMOS technology. In fact, the Linear Neural Network was estimated to consume $564\mu W$, mostly dynamic power. Hence this kind of compressor would benefit greatly by a smaller technology node with smaller $V_{DD}$ and less capacitive load per cell. The Linear Neural Network Temporal is the predictor of choice for this work because it requires less multiplications than Linear Neural Network Spatial and Temporal although the latter has been proved to be slightly more effective on a large array of electrodes for in-vitro recording. The additional computational cost of Linear Neural Network Spatial and Temporal is represented only by the increase of multiplications but memory requirements are the same for Linear Neural Network Temporal and Linear Neural Network Spatial and Temporal.
Delta compression is interesting because it can be effectively combined with a threshold block and provides high accuracy and a CR that is comparable to Linear Neural Network Temporal for low noise recorded data. Furthermore its implementation requires very little hardware and it can be combined with almost every Tx since it has the lowest energy per bit for all the examined compressors, \( E_{Cx} = 4.3\text{pJ/bit} \).

An other useful scenario for compression is when the transmitter capacity has been reached; then compressing the signals allows to record from more sites. In this case the power of the compressor should not be compared to the power of the transmitter and the compressor should be chosen based on the desired performances from the implant, but always inside the safety limits of brain implanted devices.

When only spikes are to be transmitted Compressed Sensing, achieves great performance and does not require very complicated implementations. In this scenario it was proven that an autoencoder can also be used for direct classification. The autoencoder achieves a \( CA \geq 0.9 \) for most of the simulated datasets and provides up to a \( CR = 128 \). The only operations needed are shifts and sums resulting in a very low energy efficiency for the autoencoder, \( E_{Cx} = 2.71\text{pJ/bit} \). Hence making it highly desirable for Brain Machine Interfaces where only classification of spikes is needed.

### 8.1 Outlook

Recording from thousands of channels in brain machine interfaces allows neuroscientists to investigate the working principles of the brain, to treat neuronal diseases and opens up the door to a whole new field of applications. Compression is one key element in the success of this new technology and it must be tailored to the application at hand.

For neural research where there is not yet a clear defined standard for spike sorting, lossless compression will still be needed and the differential encoding approach with predictors offers a good solution. To increase its performance new predictors can be engineered. Non linear neural network could be used and specifically ones which retains memory using Gated Recurrent Units as neurons. The non linearity and the memory element could better model the neural signal shape. With a more power friendly technology node the predictor could be even made of more blocks, each one specialized in different segment of the signal, e.g. predicting spike shapes from different neurons.

For applications where detection and classification of spikes is needed on-chip the autoencoder presented in this work could be improved by having more complex network and a 2-step training where the net is first trained in an unsupervised
fashion and then supervised, this would enhance the ability of the net to find useful features from the data. Furthermore the autoencoder’s performance could be evaluated against various spike sorting software as well. A spike detector could be also be implemented using again neurons with memory retaining ability, e.g. gated recurrent units.

Spatial information provided some better compression for lossless approach, while in spike detection only very recent works start to use multi channel data for detecting spikes. This could improve detection accuracy already on-chip. In the near future lossless compression is probably going to be more relevant but in the coming years when requirements for spike sorting will be more standardized in the scientific community spike detection and sorting directly on-chip will be the main way of compressing data in brain machine interfaces, especially for clinical and commercial applications.
Bibliography


[21] Kenneth D. Harris et al. “Accuracy of Tetrode Spike Separation as Determined by Simultaneous Intracellular and Extracellular Measurements”. In: *Journal of Neurophysiology* 84.1 (2000). PMID: 10899214, pp. 401–414. DOI: [10.1152/jn.2000.84.1.401](https://doi.org/10.1152/jn.2000.84.1.401) eprint: [https://doi.org/10.1152/jn.2000.84.1.401](https://doi.org/10.1152/jn.2000.84.1.401) URL: [https://doi.org/10.1152/jn.2000.84.1.401](https://doi.org/10.1152/jn.2000.84.1.401).


[38] S. Kim et al. “A Sub-µW/Ch Analog Front-End for δ-Neural Recording with Spike-Driven Data Compression”. In: IEEE Transactions on Biomedical Circuits and Systems (2018), pp. 1–1. ISSN: 1932-4545. DOI: 10.1109/TBCAS.2018.2880257

[39] Antal Berényi et al. “Large-scale, high-density (up to 512 channels) recording of local circuits in behaving animals”. In: Journal of Neurophysiology 111.5 (2014). PMID: 24353300, pp. 1132–1149. DOI: 10.1152/jn.00785.2013 eprint: https://doi.org/10.1152/jn.00785.2013 URL: https://doi.org/10.1152/jn.00785.2013


[54] Kenneth D. Harris et al. “Accuracy of Tetrode Spike Separation as Determined by Simultaneous Intracellular and Extracellular Measurements”. In: *Journal of Neurophysiology* 84.1 (2000). PMID: 10899214, pp. 401–414. DOI: 10.1152/jn.2000.84.1.401 eprint: https://doi.org/10.1152/jn.2000.84.1.401 URL: https://doi.org/10.1152/jn.2000.84.1.401


[82] 2016. URL: https://www.sciencedirect.com/topics/neuroscience/spike-sorting


[126] Mizuseki K, Sirota A, and Buzsáki G. Pastalkova E. *Multi-unit recordings from the rat hippocampus made during open field foraging*. 2009. URL: http://dx.doi.org/10.6080/K0Z60KZ9


Bibliography


[160] G. E. Hinton and R. R. Salakhutdinov. “Reducing the Dimensionality of Data with Neural Networks”. In: *Science* 313.5786 (2006), pp. 504–507. ISSN: 0036-8075. DOI: [10.1126/science.1127647](https://doi.org/10.1126/science.1127647). eprint: [https://science.sciencemag.org/content/313/5786/504.full.pdf](https://science.sciencemag.org/content/313/5786/504.full.pdf). URL: [https://science.sciencemag.org/content/313/5786/504](https://science.sciencemag.org/content/313/5786/504).


List of Author’s Publications

During this work the results obtained led to several publications, which are here listed.


