A New Approach to Information Processing with Filters and Pulses

Author(s):
Neff, Sarah

Publication Date:
2016

Permanent Link:
https://doi.org/10.3929/ethz-a-010748553

Rights / License:
In Copyright - Non-Commercial Use Permitted
A New Approach to Information Processing with Filters and Pulses

A thesis submitted to attain the degree of Doctor of Sciences of ETH Zurich (Dr. sc. ETH Zurich)

presented by

Sarah Neff
MSc ETH, Dipl.-Ing. Supélec
born on February 16, 1988
citizen of France and Germany

accepted on the recommendation of
Prof. Dr. Hans-Andrea Loeliger, examiner
Prof. Dr. Giacomo Indiveri, co-examiner

2016
Acknowledgments

First I want to thank my supervisor, Hans-Andrea Loeliger, for giving me the opportunity to work at the Signal and Information Processing Laboratory. Time passed at an incredible speed during my PhD. I cannot remember how many discussions we had, sometimes longer, sometimes shorter ones, often going in many directions, but always very inspiring and motivating. I found it amazing how small things, which first seemed insignificant, could all of the sudden sound so fascinating and lead to totally new ideas. For example, working on Morse code started as an idea, among many others coming up during a meeting, and turned out to be in hindsight the first milestone reached in this thesis.

I also want to thank Giacomo Indiveri for accepting to be the co-examiner of this thesis. It combines elements of neurocomputing, signal processing and machine learning, so that finding someone having experience in all these fields was not easy. Thanks again!

Of course, a big thanks also to all my colleagues from the institute. I started here with my master thesis, and immediately liked the working atmosphere, so that I decided to stay for a PhD. I really enjoyed our discussions, our weekly badminton sessions and even the yearly Sola running!

Last but not least, I want to thank my friends and family for their support during my studies. Stephan deserves a special mention, for all he did during this adventure. He was there to say the right words I needed to hear in difficult times, but also to open a beer in good times!
Abstract

This thesis presents a hierarchical network for the analysis of structured, complex time signals.

This network can be viewed as a new kind of artificial neural network working with time signals, without extra synchronization. It consists of multiple layers of feature-detection filters. We focus on pulse-domain processing, where we define an inner-product filter, looking for some pulse pattern in its input signal. The filters work by projecting the input into a one-dimensional subspace, and producing a pulse if the projection exceeds some threshold. They can be implemented efficiently using simple, forward only recursions. Among other models, the filters can be built with biologically plausible neurons. We demonstrate that inner-product filters work well with pulse-domain signals, by implementing two examples of networks for Morse code parsing.

We also show how learning can be done in such networks. We first propose algorithms for supervised learning. Their main ingredient is a new type of backpropagation, where starting on the highest layer, “good” pulse positions for the intermediate layers are recursively defined and used in the cost function. We evaluate the performance of the algorithm with examples of 3 and 4-layer networks, which can distinguish 20 piano tunes. We conclude the topic with proposals for unsupervised learning.

Finally, we broaden the framework by introducing loops, and emphasize the potential of such a recurrent network.

Keywords: State-space model; factor graph; Artificial Neural Network; hierarchical feature-detection filters; pulse-domain signal processing; inner-product filter; gradient descent; Expectation Maximization.
Kurzfassung

Diese Dissertation beschreibt ein hierarchisches Netzwerk für die Analyse strukturierter, komplexer Zeitsignalen.


**Stichworte:** Zustandsraummodell; Faktorgraf; Künstliches Neurona-les Netzwerk; hierarchisches Feature-Detection-Filter; Signalverarbei-tung mit Pulsen; Innenprodukt-Filter; Gradientenverfahren; Expectati-on Maximization.
Contents

Abstract v

Kurzfassung vii

1 Introduction 1

1.1 Background ......................................................... 1
  1.1.1 Pattern Recognition with Deep Neural Networks ............... 1
  1.1.2 Pulses and Spiking Neural Networks .......................... 2
  1.1.3 Link to Previous Work ......................................... 3

1.2 Motivation .......................................................... 3

1.3 Contributions ...................................................... 4

1.4 Overview ............................................................ 4

1.5 Preliminaries ......................................................... 5
  1.5.1 Notation .......................................................... 5
  1.5.2 Factor Graphs ................................................... 6
  1.5.3 State-Space Models ............................................. 6

I Pulse-Domain Signal Analysis: Principle and First Examples 7

2 Hierarchical Filtering 9

2.1 Introduction ......................................................... 9

2.2 Format of the Intermediate Signals ................................ 9
  2.2.1 Different Possibilities ......................................... 9
  2.2.2 A Pragmatic Argument ......................................... 10
  2.2.3 Working with Pulses ........................................... 11

2.3 Inner-Product Filters and Score Signal ............................ 12
  2.3.1 Definition ....................................................... 12
<table>
<thead>
<tr>
<th>Section</th>
<th>Title</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>2.3.2</td>
<td>A Simple Example</td>
<td>14</td>
</tr>
<tr>
<td>2.3.3</td>
<td>Pulses and Filters Work Well Together</td>
<td>15</td>
</tr>
<tr>
<td>2.4</td>
<td>Recursive Computation of the Score Signal</td>
<td>16</td>
</tr>
<tr>
<td>2.4.1</td>
<td>Update Equations</td>
<td>16</td>
</tr>
<tr>
<td>2.4.2</td>
<td>Input Estimation</td>
<td>16</td>
</tr>
<tr>
<td>2.4.3</td>
<td>Guard Interval</td>
<td>17</td>
</tr>
<tr>
<td>2.5</td>
<td>Conclusion</td>
<td>18</td>
</tr>
<tr>
<td>3</td>
<td>Detecting Precise Temporal Patterns</td>
<td>19</td>
</tr>
<tr>
<td>3.1</td>
<td>Monomial Pulse Patterns</td>
<td>19</td>
</tr>
<tr>
<td>3.1.1</td>
<td>Definitions</td>
<td>19</td>
</tr>
<tr>
<td>3.1.2</td>
<td>Relationship to Latency Codes</td>
<td>20</td>
</tr>
<tr>
<td>3.1.3</td>
<td>Beyond Monomials</td>
<td>20</td>
</tr>
<tr>
<td>3.2</td>
<td>Design of Filters</td>
<td>20</td>
</tr>
<tr>
<td>3.2.1</td>
<td>Sinusoidal Model</td>
<td>20</td>
</tr>
<tr>
<td>3.2.2</td>
<td>Exponential Model</td>
<td>23</td>
</tr>
<tr>
<td>3.2.3</td>
<td>Neuron Model</td>
<td>24</td>
</tr>
<tr>
<td>3.3</td>
<td>Properties of the Filters</td>
<td>33</td>
</tr>
<tr>
<td>3.3.1</td>
<td>Robustness</td>
<td>33</td>
</tr>
<tr>
<td>3.3.2</td>
<td>Redundancy</td>
<td>33</td>
</tr>
<tr>
<td>3.4</td>
<td>Conclusion</td>
<td>35</td>
</tr>
<tr>
<td>4</td>
<td>First Example - Morse Code with Sinusoidal Model</td>
<td>37</td>
</tr>
<tr>
<td>4.1</td>
<td>Background</td>
<td>37</td>
</tr>
<tr>
<td>4.1.1</td>
<td>Introduction</td>
<td>37</td>
</tr>
<tr>
<td>4.1.2</td>
<td>About the Morse Code</td>
<td>38</td>
</tr>
<tr>
<td>4.1.3</td>
<td>Training and Test Datasets</td>
<td>38</td>
</tr>
<tr>
<td>4.2</td>
<td>Description of the Network</td>
<td>39</td>
</tr>
<tr>
<td>4.2.1</td>
<td>First Layer</td>
<td>39</td>
</tr>
<tr>
<td>4.2.2</td>
<td>Higher Layers</td>
<td>40</td>
</tr>
<tr>
<td>4.2.3</td>
<td>Remarks</td>
<td>41</td>
</tr>
<tr>
<td>4.3</td>
<td>Conclusion and Outlook</td>
<td>46</td>
</tr>
<tr>
<td>5</td>
<td>Second Example - Morse Code with Neuron Model</td>
<td>47</td>
</tr>
<tr>
<td>5.1</td>
<td>Introduction</td>
<td>47</td>
</tr>
<tr>
<td>5.2</td>
<td>Building Block</td>
<td>48</td>
</tr>
<tr>
<td>5.2.1</td>
<td>General Description</td>
<td>48</td>
</tr>
<tr>
<td>5.2.2</td>
<td>Parameters Adapted to Morse Code Example</td>
<td>49</td>
</tr>
<tr>
<td>5.3</td>
<td>Building Simple Features</td>
<td>49</td>
</tr>
<tr>
<td>5.3.1</td>
<td>Counter - Dah</td>
<td>50</td>
</tr>
<tr>
<td>Chapter</td>
<td>Section</td>
<td>Page</td>
</tr>
<tr>
<td>---------</td>
<td>---------</td>
<td>------</td>
</tr>
<tr>
<td>II</td>
<td>Learning Algorithms</td>
<td>57</td>
</tr>
<tr>
<td>6</td>
<td>Supervised Learning with Sinusoidal Model</td>
<td>59</td>
</tr>
<tr>
<td>6.1</td>
<td>Introduction</td>
<td>59</td>
</tr>
<tr>
<td>6.2</td>
<td>Data and Goal</td>
<td>60</td>
</tr>
<tr>
<td>6.2.1</td>
<td>Recordings</td>
<td>60</td>
</tr>
<tr>
<td>6.2.2</td>
<td>Architecture</td>
<td>62</td>
</tr>
<tr>
<td>6.3</td>
<td>Implementation of the Learning Algorithm</td>
<td>64</td>
</tr>
<tr>
<td>6.3.1</td>
<td>Cost Function</td>
<td>65</td>
</tr>
<tr>
<td>6.3.2</td>
<td>Equations</td>
<td>65</td>
</tr>
<tr>
<td>6.3.3</td>
<td>Pseudo-Code of the Algorithm</td>
<td>67</td>
</tr>
<tr>
<td>6.3.4</td>
<td>Some Implementation Details</td>
<td>68</td>
</tr>
<tr>
<td>6.4</td>
<td>Results</td>
<td>69</td>
</tr>
<tr>
<td>6.4.1</td>
<td>Accuracy</td>
<td>69</td>
</tr>
<tr>
<td>6.4.2</td>
<td>An Example of Learned Feature</td>
<td>69</td>
</tr>
<tr>
<td>6.4.3</td>
<td>Conclusion and Outlook</td>
<td>71</td>
</tr>
<tr>
<td>7</td>
<td>Supervised Learning with Neuron Model</td>
<td>73</td>
</tr>
<tr>
<td>7.1</td>
<td>Introduction</td>
<td>73</td>
</tr>
<tr>
<td>7.2</td>
<td>Architecture</td>
<td>73</td>
</tr>
<tr>
<td>7.3</td>
<td>Implementation of the Learning Algorithm</td>
<td>75</td>
</tr>
<tr>
<td>7.3.1</td>
<td>Cost Function</td>
<td>75</td>
</tr>
<tr>
<td>7.3.2</td>
<td>Equations</td>
<td>75</td>
</tr>
<tr>
<td>7.3.3</td>
<td>Pseudo-Code</td>
<td>77</td>
</tr>
<tr>
<td>7.3.4</td>
<td>Implementation Details</td>
<td>78</td>
</tr>
<tr>
<td>7.4</td>
<td>Results</td>
<td>79</td>
</tr>
<tr>
<td>7.4.1</td>
<td>Accuracy</td>
<td>79</td>
</tr>
<tr>
<td>7.4.2</td>
<td>Going Deeper</td>
<td>79</td>
</tr>
<tr>
<td>7.5</td>
<td>Relationship to Other Learning Algorithms in the Literature</td>
<td>80</td>
</tr>
<tr>
<td>7.5.1</td>
<td>SpikeProp</td>
<td>80</td>
</tr>
<tr>
<td>7.5.2</td>
<td>Tempotron</td>
<td>81</td>
</tr>
<tr>
<td>7.5.3</td>
<td>Chronotron</td>
<td>81</td>
</tr>
</tbody>
</table>
7.5.4 Another Framework for Spatio-Temporal Spike Pattern Recognition .................. 81
7.5.5 Conclusion ............................................................................. 81

8 Unsupervised Learning .................................................................. 83
  8.1 Introduction .............................................................................. 83
  8.2 Sinusoidal Model ..................................................................... 83
    8.2.1 Setup ............................................................................. 83
    8.2.2 Derivation of EM-Update Rules ........................................ 84
  8.3 Neuron Model .......................................................................... 87
    8.3.1 Learning $C$ .................................................................... 88
    8.3.2 Learning $\lambda$ ............................................................... 88
  8.4 Conclusion ............................................................................... 90

III Beyond Feedforward Networks ................................................... 91

9 Introducing Loops ......................................................................... 93
  9.1 Introduction ............................................................................. 93
  9.2 First Examples ......................................................................... 93
    9.2.1 Neuron Model .................................................................. 94
    9.2.2 Sinusoidal Model ............................................................. 96
  9.3 Potential Computational Power for Large Networks ................... 97
    9.3.1 Notation ........................................................................... 97
    9.3.2 Sharing Neurons ............................................................. 97
    9.3.3 Increasing the Number of Neurons ..................................... 98

10 Conclusion .................................................................................. 101

A A State-Space Model of a Neuron .................................................. 103
  A.1 A Model with Dendrites ......................................................... 103
  A.2 LIF with Exponential Input Currents ....................................... 107

B Proofs for Chapter 3 ..................................................................... 109
  B.1 Sinusoidal Model .................................................................... 109
    B.1.1 Version I .......................................................................... 112
    B.1.2 Version II ......................................................................... 112
  B.2 Exponential Model ................................................................... 113
  B.3 Neuron Model ......................................................................... 115
    B.3.1 Version I .......................................................................... 116
    B.3.2 Version II ......................................................................... 117
B.3.3 Version III .......................... 117

C Networks for Parsing Morse Code 119
    C.1 About Morse Code ....................... 119
       C.1.1 International Morse Code .......... 119
       C.1.2 Speed of Transmission and Dit Length 119
    C.2 Datasets ................................ 121
       C.2.1 Training Set ......................... 121
       C.2.2 Test Set ............................. 121
    C.3 Sinusoidal Model ....................... 122
       C.3.1 Layer 2 ................................ 122
       C.3.2 Layer 3 ................................ 123
       C.3.3 Layer 4 ................................ 125
    C.4 Neuron Model ........................... 125

D Learning Algorithms 129
    D.1 Melodies ................................. 129
    D.2 Architecture for Learning: Fixed Parameters 129
       D.2.1 First Layer ............................ 129
       D.2.2 Second Layer ......................... 132
       D.2.3 Third Layer ............................ 132
    D.3 Derivation of the Update Equations - Sinusoidal Model 133
       D.3.1 Highest Layer ......................... 133
       D.3.2 Computation of Endpoints for Intermediate Layers 135
    D.4 Derivation of the Update Equations - Neuron Model ....... 136
       D.4.1 Update of $\lambda$ ..................... 136
       D.4.2 Computation of Endpoints for Intermediate Layers 138

Bibliography 141

About the Author 147
Chapter 1

Introduction

1.1 Background

This thesis presents a hierarchical architecture for pattern recognition, as shown in Figure 1.1. Each block, called feature-detection filter, is looking for a feature in its input signal, and produces an output which in turn is analyzed by subsequent layers. In this section, we first develop the similarities of this architecture with deep neural networks, then with spiking neural networks. Finally we precise the links to previous work.

1.1.1 Pattern Recognition with Deep Neural Networks

The idea of using a deep layered network, each layer producing an output taken as an input to the next layer, has been very popular for pattern recognition in recent years (“deep learning” [30]). Evidence suggesting that deep architectures could be more powerful than shallow ones also emerged in [2,10]. Deep feedforward networks are nowadays an important tool in machine learning [17] and have improved the state of the art in many domains. Popular examples are image recognition [11,29] and speech recognition [24,43].

The main advantage of such an architecture over more traditional methods is the automated feature extraction: there is no need for hand-crafted feature design, the architecture processes raw signals directly. Multiple layers lead to a hierarchy of signals, with lower levels close to the input signal, and higher levels representing more abstract features.
For a long time, only shallow architectures showed successful \cite{51,52}. One of the first “working” (i.e. trainable) deep architectures are convolutional neural networks for image recognition \cite{32}. There, the input are raw pixel values and the layerwise procedure of feature extraction can be - literally - seen: the first layer detects edges, at different locations and with different orientations. The second layer looks for different arrangements of these edges and thus detects a somewhat larger feature. This procedure goes on, each layer looking for combinations of features of its input layers, until the last layer, which can detect complete objects or even scenes.

When the input is not an image, the role of the intermediate layers may not be so clear. In this thesis, we focus on the concrete example of music tunes. We present a learning algorithm, and show some intermediate features.

1.1.2 Pulses and Spiking Neural Networks

The architecture presented in this thesis has the particularity to use only pulsed signals (time signals that are either 0 or 1 at each time step), starting from the second layer. In other words, only the raw input signal is not in the pulse domain. We will show that the feature-detection filters can be built with biologically plausible neurons, making an interesting link to Spiking Neural Networks (SNN, \cite{14}).

Contrary to traditional neural networks, working with continuous values representing the firing rate of a neuron, SNN take into account the
precise timing of the individual input spikes. This fundamental difference makes the training of such networks difficult. In this thesis we propose a new method, suitable for deep networks, and show its performance on the example of music tunes.

1.1.3 Link to Previous Work

This thesis is a continuation of the structured methodical approaches proposed in [48]: The factor graph representation of a state-space model, together with a glue factor expressing some additional constraint, were successfully applied to estimation and detection tasks [34, 48, 49], and had the advantages of being time-invariant and working without external synchronization. Of special interest is the forward only scenario, where the glue factor is at the end of the signal: It allows for online signal analysis, whereas traditional architectures usually use block processing based methods.

In situations where the glue factor is too complicated, a logical next step is to develop a hierarchical model. We show that this approach is well suited for multiscale signal analysis. Signal structure on a longer timescale is captured by levels higher up in the hierarchy. The resulting network can be viewed as a new kind of artificial neural network working with signals (functions of time), without extra synchronization or chopping into frames.

1.2 Motivation

The main motivation of this thesis is to present a general framework for structured signal analysis, which is self-timed and could be implemented by clockless continuous-time analog circuits.

Moreover, this thesis can be seen as a proof of concept for some ideas in [48]: We present concrete, working examples of the architecture, and show first results in learning such networks. We even extend the architecture by introducing loops, which could potentially keep information over very long time spans.

Finally, a main novel point is that the architecture works with pulse-domain signals, and can be implemented with biologically plausible neurons. As such, it can be viewed as a new mode of computation using
linear filters and well-separated unit pulses that may help to understand how information is processed in the brain.

1.3 Contributions

We extend the framework for information processing in feedforward networks proposed in [48]:

- We show that pulses work well in a layered network of feature-detection filters.
- We introduce robust inner-product filters, and describe how to design filters for detecting precise temporal pulse patterns.
- We show that the filters can be built with biologically plausible neurons.
- We demonstrate the use of the inner-product filters with two (large) examples of multiscale pattern detection.

Concerning learning such networks:

- We devise a gradient descent algorithm, working with a new type of backpropagation rule and show its performance in a supervised learning scheme.
- In a network using the neuron model, we introduce a learning scheme based on delays and weights, as opposed to traditional learning methods which use weights only. We show the performance of this learning in a second example.
- We present an algorithm for unsupervised learning, based on Expectation Maximization.

1.4 Overview

In the first part, we start from the architecture proposed in [48] and precise the framework of hierarchical feedforward filtering: We restrict the signals to be in the pulse domain and present the inner-product filter for detecting precise temporal pulse patterns. This filter produces a score signal, which can be computed with different models; we focus on a sinusoidal and a neuron model. We conclude this part with two examples of
larger networks, which illustrate the principles and show that complex pattern recognition is possible.

The second part introduces learning algorithms: We develop mainly the supervised setting, and show how learning can be done in networks using the sinusoidal model. We then turn to the neuron model, where learning is more difficult, but can still be done. In both cases we illustrate the algorithms in learning a network that can distinguish 20 short piano tunes. We conclude the second part with a glance on unsupervised learning, where we propose algorithms for single-layer networks.

The last and shortest part is mainly an outlook for future work. We show how the architecture can be extended by introducing loops, and emphasize the potential of such a recurrent network.

1.5 Preliminaries

1.5.1 Notation

We write matrices in boldface $\mathbf{A}$, vector in italic boldface $\mathbf{v}$. We use following symbols:

- $\triangleq$ Equal by definition
- $\approx$ Approximately equal
- $\propto$ Proportional to
- $\mathbf{I}_n$ $n \times n$ identity matrix
- $\mathbf{A}^\top$ Transpose of matrix $\mathbf{A}$
- $\mathbf{A}^{-1}$ Inverse of matrix $\mathbf{A}$
- $\mathbf{A}^{-\top}$ Inverse transpose of matrix $\mathbf{A}$
- $\mathbf{A} \otimes \mathbf{B}$ Kronecker product of matrices $\mathbf{A}$ and $\mathbf{B}$
- $\text{cvect} \, \mathbf{A}$ Matrix-to-vector operation by stacking matrix columns
- $\text{diag}(\mathbf{a})$ Diagonal matrix, with elements of $\mathbf{a}$ on the diagonal
- $\text{rotm}(\Omega)$ Rotation matrix $\begin{pmatrix} \cos \Omega & -\sin \Omega \\ \sin \Omega & \cos \Omega \end{pmatrix}$
1.5.2 Factor Graphs

We use Forney-style factor graphs [35], with additions from [48].

- A forgetting factor can be used as defined in [49]: the factor graph of Figure 1.2a represents the function \( f(x) = h(x)^\gamma g(x) \). The symbol \( \gamma \) indicates that every factor to the left of the bracket is taken to the power of \( \gamma \).

- We draw a small filled box at the end of an edge whose value is fixed. For example, in Figure 1.2b, the value of \( Y \) is fixed to \( y \).

1.5.3 State-Space Models

We define a linear discrete-time state-space model (SSM) with state \( x_k \), input \( u_k \), output \( y_k \) by following equations:

\[
\begin{align*}
x_{k+1} &= A_k x_k + B_k u_k \\
y_k &= C_k x_k
\end{align*}
\]  

(1.1)  

(1.2)

Such a SSM is time-invariant if \( A_k = A \), \( B_k = B \) and \( C_k = C \) for all \( k \in \mathbb{Z} \).

We will mainly work with linear time-invariant and autonomous SSM, i.e. \( B = 0 \).
Part I

Pulse-Domain Signal Analysis: Principle and First Examples
Chapter 2
Hierarchical Filtering

2.1 Introduction

In this first chapter, we describe and extend the hierarchical feedforward network for detection and estimation of complex, structured time signals presented in [48].

Figure 2.1 shows the structure of this layered network. Each block, called feature-detection filter, is looking for a feature in its input signal, and produces an output which in turn is analyzed by subsequent layers.

As the architecture was only roughly sketched, we first concretize the suggestions of [48]: we investigate which type of signals give a robust functionality and then turn to the computation of these intermediate signals.

2.2 Format of the Intermediate Signals

2.2.1 Different Possibilities

The network of Figure 2.1 is constructed on the principle of using the output of one layer as an input to the next layer.

Assume a filter produces as output the score signal in the top of Figure 2.2. There exist different ways of reusing this signal as an input to the next layer. We compare three of them:

- using the raw signal (this was first proposed in [48]),
- defining a threshold and using a rectangular signal,
defining a threshold and using a pulsed signal.

These three possibilities are described in the top, middle, respective bottom part of Figure 2.2 with two possible values for the threshold.

\subsection*{2.2.2 A Pragmatic Argument}

After some trials, it turned out only a pulsed score signal, producing a unit pulse whenever the threshold is reached, followed by a suitable guard interval (to separate two consecutive pulses) could be used in a hierarchical architecture.

As the score is based on linear filters, the input signal is (locally) integrated. The score then depends on the width of the rectangular signal, which varies with the threshold, as illustrated in Figure 2.2 and this hinders robust functionality. The local integration of the raw signal poses similar problems.

In contrast, pulses have the same (unit) weight, but as can be seen in Figure 2.2b, we need to forbid the second, dashed pulse, so that the output of the filter is the same for high and low thresholds. We solve this by introducing a suitable guard interval after a pulse, i.e. a time interval where a subsequent pulse is forbidden.

The width of the raw and rectangular signals informs about the duration of a feature. This information is not lost when working with pulses: if a feature is present over a time longer than the guard interval, a pulse
2.2 Format of the Intermediate Signals

Figure 2.2: Possible intermediate signals for two different thresholds: raw (top), rectangular (middle) and pulsed score signal (bottom)

In conclusion, we chose to work with pulses because they showed both the highest robustness and versatility.

2.2.3 Working with Pulses

The input of the network in Figure 2.1 is not necessarily pulsed. The first layer extracts pulse-domain feature signals from the input, all subsequent layers (see Figure 2.3) work with pulses. We will not discuss the first layer in this thesis, but only focus on the higher layers. These layers are looking for precise temporal pulse patterns, where only a few pulses are present (as for example in Figure 2.4).

This problem is not common: Most pulse-based approaches are focusing on firing rates [7], and these methods cannot be applied when there are too few pulses [54].

This problem is also not only theoretical: There is evidence that this kind of signaling occurs in the brain [21, 25, 27, 42], which means the
Hierarchical Filtering

Figure 2.3: Layered network of feature-detection filters, working with pulses after layer 1

precise timing of pulses carry information.

Finally, pulses are also interesting because they have finite energy, and working with few pulses can give rise to energy efficient architectures.

2.3 Inner-Product Filters and Score Signal

In this section, we define the pulse-domain feature-detection filters. We emphasize in which regime the filters work well and how to compute them efficiently.

2.3.1 Definition

The core idea of the feature-detection filters is to compare the signal to be analyzed \( y \) with a model signal \( \tilde{y} \) coming from an autonomous state-space model [8,37].

More formally:

- Let \( y_1, y_2, \ldots \in \{0,1\}^L \) be the (pulse-domain) multichannel input signal of some feature-detection filter.
2.3 Inner-Product Filters and Score Signal

Figure 2.4: An example of three channel pulse pattern (inside the box), a possible target for a feature-detection filter

- Let $\tilde{y}_1, \tilde{y}_2, \ldots \in \mathbb{R}^L$ be the model signal originating from an autonomous deterministic state-space model:

$$\tilde{y}_k = CA^{k-n}x_n \quad (2.1)$$

with $A \in \mathbb{R}^{m \times m}$, $C \in \mathbb{R}^{L \times m}$, and $x_n \in \mathbb{R}^m$. The matrix $A$ is assumed to be regular, which means that the time-$n$ state $x_n$ completely determines $\tilde{y}_k$ for all $k$.

At each time instant, we compute the inner-product signal

$$\langle y, \tilde{y} \rangle = \langle y, \tilde{y}_1 \rangle, \langle y, \tilde{y}_2 \rangle, \ldots \in \mathbb{R}$$

defined as

$$\langle y, \tilde{y} \rangle_n \triangleq \sum_{k=1}^{n} y_k^T \tilde{y}_k \quad (2.2)$$

for some fixed $x_n = s$. The model signal $\tilde{y}$ thus serves as a weighting kernel for the pulses in $y$.

With (2.1), this inner-product signal can also be written as:
\( \langle y, \tilde{y} \rangle_n = \sum_{k=1}^{n} y_k^T C A^{k-n} x_n \) \hspace{1cm} (2.3)

\[ = \sum_{k=1}^{n} x_n^T (A^T)^{k-n} C^T y_k \] \hspace{1cm} (2.4)

We will assume that all eigenvalues of \( A \) are strictly larger than 1, which implies that the sum (2.2) converges for \( n \to \infty \). We will only consider the stationary case \( n \gg 1 \) where border effects can be neglected.

If \( \langle y, \tilde{y} \rangle_n \) exceeds some threshold, the feature-detection filter produces a unit pulse at time \( n \). Then, any further pulses are suppressed for the duration of some guard interval.

### 2.3.2 A Simple Example

Suppose we want to detect the three channel pulse pattern shown in Figure 2.5 at time instant \( n = 300 \), for given

- \( A = \lambda \text{rotm}(\Omega) \), where \( \lambda = 1.005 \), \( \Omega = 2\pi/400 \)
- \( x_n = s = [1 \ 0]^T \)

From (2.1), it is clear that every component of the weighting signal is a damped sinusoid, with amplitude and phase determined by \( C \) and \( s \).

To maximize the inner-product, we need a weighting signal which is maximal at the positions of the pulses. In our case (see Figure 2.5) the first component of the weighting signal should be maximal at \( t = 100 \), the second at \( t = 300 \) and the third at \( t = 200 \).

As \( s \) is fixed, we choose \( C \) in consequence:

\[ C = \begin{pmatrix} -1 & 0 \\ 1 & 0 \\ 0 & -1 \end{pmatrix} \] \hspace{1cm} (2.5)

The corresponding weighting signals are drawn in red in Figure 2.5 and the score function is given in the last plot. Taking a threshold of 2 would work well for detecting the pattern at \( n = 300 \).

The chosen \( C \) may not be optimal (a careful analysis of the weighting signals shows that the maxima in channels 1 and 3 are slightly shifted to
the right, such that other pulse patterns produce an even higher score). We will give further examples and describe how to find an optimal $C$ in Chapter 3.

### 2.3.3 Pulses and Filters Work Well Together

From this simple example, we can already see some advantages of the inner-product filters:

- The network is self-timed and each filter allows small variations around the exact position of the pulses.
- It can handle large inputs ($L \gg 1$), because the size of the state can be kept small (it is not necessarily bounded to the size of the input).
- For large $L$, the filter tolerates some missing and extra pulses: As the score signal is the projection of the input signal into a one-dimensional subspace, it produces a pulse whenever the overall

![Score signal for three channel input pattern](image.png)

**Figure 2.5:** Score signal for three channel input pattern
direction of the input signal is correct. This will be shown in detail in Section 3.3.1.

2.4 Recursive Computation of the Score Signal

In this section we detail how to compute the score signal of the inner-product filters efficiently. We suppose $x_n = s$ (some given vector).

2.4.1 Update Equations

In the factor graph of Figure 2.6, the weighted mean $\mathbf{W}_{X_n} \mathbf{m}_{X_n} = \mathbf{\xi}_n$ can be updated with (see [35]):

\[
\mathbf{\xi}_n = A^{-T} \mathbf{\xi}_{n-1} + C^T y_n
\]  
\[
= \sum_{l=0}^{n-1} A^{-lT} C^T y_{n-l}
\]  
\[
= \sum_{k=1}^{n} (A^T)^{k-n} C^T y_k
\]  

We will mainly be interested in models for which the forward message $\mathbf{\xi}_n$ can be computed without any inversion.

From (2.4), the score signal is simply $s^T \mathbf{\xi}_n$.

2.4.2 Input Estimation

The score signal can be seen as an estimation of a sparse input in Figure 2.6. With $B = (-1 \ 0 \ldots \ 0)^T$, the backward message on edge $U_n$ is:

\[
\mathbf{m}_{U_n} = (B^T \mathbf{W}_{X_n} B)^{-1} B^T (-\mathbf{W}_{X_n} \mathbf{m}_{X_n})
\]  
\[
= (B^T \mathbf{W}_{X_n} B)^{-1} s^T \mathbf{\xi}_n
\]  

In the stationary case ($n \gg 1$), $\mathbf{W}_{X_n}$ is constant (and can be computed with (4.5)), thus

\[
\mathbf{m}_{U_n} \propto s^T \mathbf{\xi}_n
\]
2.4.3 Guard Interval

After a pulse is produced in the output, all subsequent pulses have to be suppressed for the duration of a guard interval. There are two possibilities for the implementation:

- set $\xi = 0$ and use a counter

- set the state $\xi$ to a very large value (either positive or negative). This ensures that the filter needs some time to "recover" before it produces a new pulse.

We used the first strategy in our implementations.
2.5 Conclusion

We have detailed the architecture for signal analysis. It needs pulses as intermediate signals to work robustly.

These pulses are produced by inner-product filters, which correspond to the projection of the input signal into a one-dimensional subspace.

The score signal of the inner-product filters can be computed efficiently with a simple recursion.

We have shown an example of detection, but have given only a first glance on finding optimal filter parameters. In the next chapter, we dive deeper into this subject.
Chapter 3
Detecting Precise Temporal Patterns

In this chapter, we show how to design filters for detecting a given pulse pattern. First, we define and restrict the problem to monomial pulse patterns. The concrete design problem is then to find suitable $C$ and $s$ for a given monomial target pattern and given $A$.

We present results for three different models: a sinusoidal model (as suggested in [48]), an exponential model, and a neuron model.

We conclude with general remarks concerning the performance of the designed filters.

3.1 Monomial Pulse Patterns

3.1.1 Definitions

Definition 3.1.1: Monomial pulse pattern
A multichannel signal will be called monomial if it is zero everywhere except for (at most) one unit pulse in every channel.

For example, the three channel pulse pattern of Figure 3.1 is monomial.

Definition 3.1.2: Preferred monomial pattern
The preferred monomial pattern of a given inner-product filter is the monomial signal $y$ that maximizes $\max_n \langle y, \tilde{y} \rangle_n$ (the peak amplitude of the inner-product signal) among all monomial patterns.
3.1.2 Relationship to Latency Codes

A latency (or time-to-first-spike) code is based on the idea that the first spike after the onset of a stimulus contains all information needed to be transmitted [15]. In other words, in each channel, only the timing of the first spike after a reference signal is taken into account. All subsequent pulses are forbidden by inhibition, which holds as long as there is no new stimulus. This corresponds exactly to our definition of a monomial pattern.

There is experimental evidence that this type of coding is used, for example in the sensory system (to encode the strength and direction of touch in the fingertip [27]) and in the visual system [16]. One advantage of latency codes is the speed of transmission: As only one spike is needed, the information could be provided faster than with rate codes [7, 27, 55].

3.1.3 Beyond Monomials

The results below are derived in a purely monomial setting. In practice, only pulses that are sufficiently well separated are required.

3.2 Design of Filters

Assume given an input \( y_1, y_2, \cdots \in \{0, 1\}^L \), having a pulse in every channel \( \ell \in \{1, \ldots, L\} \), at time \( k_\ell \). Also given is \( A \), for which we will distinguish three different cases.

In each case, we want to recognize the pulse pattern at time \( n \) (where \( n \geq k_\ell, \forall \ell \)), i.e. we want to find \( C \) and \( s \) such that the resulting filter prefers the given monomial pulse pattern \( y \).

3.2.1 Sinusoidal Model

Setup

We take \( A \) of size \( m \times m \) (with \( m = 2M \)), that models a signal consisting of a sum of exponentially increasing sinusoids:

\[
A = \begin{pmatrix}
J_1 & 0 & 0 & \ldots & 0 \\
0 & J_2 & 0 & \ldots & 0 \\
\vdots & & & \ddots & \vdots \\
0 & 0 & \ldots & 0 & J_M
\end{pmatrix},
\]

(3.1)
where \( \mathbf{J}_j \triangleq \lambda_j \text{rotm}(\Omega_j) \) with \( \lambda_j \geq 1, j = 1, \ldots, M \), and

\[
\text{rotm}(\Omega) \triangleq \begin{pmatrix}
\cos(\Omega) & -\sin(\Omega) \\
\sin(\Omega) & \cos(\Omega)
\end{pmatrix}.
\]  

(3.2)

We have presented an example of such a model, together with a suitable matrix \( \mathbf{C} \) for a given input pulse pattern in Subsection 2.3.2. Note that even if \( \mathbf{C} \) is acceptable, it is not optimal in the sense of Definition 3.1.2: \( \mathbf{C} \) seemed correct (the pulse pattern is recognized), but we did not check if there was not a better one.

**Optimal \( \mathbf{C} \) and \( \mathbf{s} \)**

For \( \mathbf{A} \) as in (3.1), the optimal matrix \( \mathbf{C} \) with rows \( \mathbf{c}_1, \ldots, \mathbf{c}_L \) is given by

\[
\mathbf{c}_\ell \triangleq \mathbf{s}^T \begin{pmatrix}
\mathbf{R}_1 & 0 & 0 & \cdots & 0 \\
0 & \mathbf{R}_2 & 0 & \cdots & 0 \\
\vdots & \vdots & \vdots & \ddots & \vdots \\
0 & 0 & \cdots & 0 & \mathbf{R}_M
\end{pmatrix} \mathbf{A}^{n-k_\ell},
\]  

(3.3)

where \( \mathbf{s} \triangleq (1, 0, 1, 0, \ldots, 1, 0)^T \in \mathbb{R}^m \) and where \( \mathbf{R}_j \triangleq \begin{pmatrix} 1 & -\phi_j \\ \phi_j & 1 \end{pmatrix} \) with \( \phi_j \triangleq \frac{\ln(\lambda_j)}{\Omega_j} \).

Note that the target pulse pattern enters the filter design only by the term \( \mathbf{A}^{n-k_\ell} \) in (3.3).

**Theorem 3.2.1: Optimal \( \mathbf{C} \) for Sinusoidal Model - Version I**

If

\[
\max_{\ell \in \{1, \ldots, L\}} (n - k_\ell) < \min_{j \in \{1, \ldots, M\}} \frac{3\pi/2 - \arctan \phi_j}{\Omega_j}
\]  

(3.4)

then this inner-product filter prefers the given pulse pattern and the corresponding maximum value of the inner-product signal is \( Lm/2 \).\(^{1}\)

The proof is done by maximizing the inner-product separately for each channel and each frequency. It is given in Section B.1 of the Appendix.

Figure 3.1a shows the score function and the weighting functions for each channel. A careful comparison with Figure 2.5 shows that the maximum of the score function is now exactly 3, and that the maximum of the weighting signal in each channel is exactly at the position of the pulse.

\(^{1}\)The term \(- \arctan \phi_j\) in (3.4) has been forgotten in 36.
Figure 3.1: Score signal and weighting signals for sinusoidal model
What if Some Frequencies Don’t Satisfy Equation (3.4)?

Strictly speaking, it is not necessary that all frequencies satisfy (3.4).

**Theorem 3.2.2 : Optimal C for second order filters - Version II**

Let \( s \triangleq (1, 0, 1, 0 \ldots, 1, 0)^T \in \mathbb{R}^m \).

Suppose there exists \( j_0 \) such that

\[
\max_{\ell \in \{1, \ldots, L\}} (n - k_\ell) < \frac{3\pi/2 - \arctan \phi_{j_0}}{\Omega_{j_0}}
\]

(3.5)

For \( C \) with rows given by

\[
c_\ell \triangleq s^T R A^{n-k_\ell},
\]

(3.6)

where \( R \) has only two nonzero elements, the rotation matrices \( R_{j_0} \) and \( R_i \) (the position of \( i \neq j_0 \) is arbitrary) on the diagonal. This inner-product filter prefers the given pulse pattern and the corresponding maximum value of the inner-product signal is \( 2L \).

The proof is given in Section [B.1] of the Appendix. An example is given in Figure 3.1b (the weighting signal is divided by 2 for better readability).

Allowing some frequencies which do not respect condition (3.4) has a cost: local maxima are present in the score signal.

### 3.2.2 Exponential Model

In this example, we show that complex poles are not needed: The theorem can be rewritten for filters with simple real poles.

Let \( A \) be a real diagonal matrix with different diagonal elements \( \lambda_j > 1, j = 1, \ldots, m \). Let \( s \triangleq (1, 1, \ldots, 1)^T \in \mathbb{R}^m \) and let \( C \) be a matrix with rows \( c_1, \ldots, c_L \in \mathbb{R}^m \) as follows.

- If \( k_\ell = n \), then
  \[
c_\ell \triangleq (0, \ldots, 0, 1, 0, \ldots, 0)^T,
  \]
  (3.7)

  where the position of the single nonzero entry is arbitrary.

- If \( k_\ell < n \), then \( c_\ell \) has exactly two nonzero entries at arbitrary positions \( i \) and \( j \) with coefficients
  \[
c_{\ell,i} \triangleq \frac{\ln \lambda_j}{\ln \lambda_i - \ln \lambda_j} \lambda_i^{n-k_\ell}
  \]
  (3.8)

  and vice versa with \( i \) and \( j \) exchanged.
The choice of the positions of the nonzero elements of $C$ does affect the shape of the weighting signal $\tilde{y}$, but not the position of the maximum.

**Theorem 3.2.3 : Optimal C for Exponential Model**

Such a filter prefers the given pulse pattern, and the corresponding maximum value of the inner-product signal is $L$.

The proof is given in Section B.2 of the Appendix.
An example is given in Figure 3.2.

### 3.2.3 Neuron Model

In this section, we show that the inner-product filter can be built with a biologically plausible neuron model.

**Deriving the Model**

Neither the classical neuron model, where the inputs are simply weighted and summed [41] nor the Leaky Integrate-and-Fire (LIF, [14]), described
in Figure 3.3 are suitable models: We need delays, so that a pulse arriving on a synapse arrives with some delay at the soma.

It has been shown that the location of a synapse has an influence on the shape of the postsynaptic potentials \[39,47\]. In the LIF of Figure 3.3, if the input current \( I \) results from the activity of presynaptic neurons, we can turn it into a model that has delays:

We consider

\begin{itemize}
  \item \( m \) dendritic trees (typically \( 1 \leq m \leq 20 \) \[12\]),
  
  \item on each tree \( i \) there are \( n_i \) synapses (such that \( L = \sum_i n_i \), i.e. there are \( L \) inputs in total).
\end{itemize}

We model the dendritic membrane as a parallel RC-circuit, with an input current source for each synapse \[14\]. The complete circuit is drawn in Figure 3.4.

From this electrical model, we can derive a state-space model for a neuron, supposing there is an input on synapse \( j \) of the \( i \)th dendritic tree at time instant \( k_{ij} \) (\( 1 \leq i \leq m \) and \( 1 \leq j \leq n_i \)). The complete derivation is in Appendix A.1 and gives
Figure 3.4: Electrical circuit corresponding to a LIF with synaptic inputs on $m$ dendrites
\[
\begin{pmatrix}
V_S[k+1] \\
V_1[k+1] \\
\vdots \\
V_m[k+1]
\end{pmatrix}
= 
\begin{pmatrix}
\tilde{\lambda}_0 & \tilde{\alpha}_1 & \tilde{\alpha}_2 & \cdots & \tilde{\alpha}_m \\
0 & \tilde{\lambda}_1 & 0 & \cdots & 0 \\
\vdots & \vdots & \ddots & \ddots & \vdots \\
0 & \cdots & \cdots & 0 & \tilde{\lambda}_m
\end{pmatrix}
\begin{pmatrix}
V_S[k] \\
V_1[k] \\
\vdots \\
V_m[k]
\end{pmatrix}
+ C^T 
\begin{pmatrix}
\delta[k-k_{11}] \\
\delta[k-k_{12}] \\
\vdots \\
\delta[k-k_{1n_1}] \\
\delta[k-k_{21}] \\
\vdots \\
\delta[k-k_{2n_2}] \\
\vdots \\
\delta[k-k_{mn_m}]
\end{pmatrix}
\]

with

\[
C^T = 
\begin{pmatrix}
0 & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots & 0 \\
c_{11} & \cdots & c_{1n_1} & 0 & \cdots & 0 & 0 & \cdots & 0 \\
0 & \cdots & 0 & c_{21} & \cdots & c_{2n_2} & 0 & \cdots & 0 \\
\vdots \\
0 & \cdots & 0 & 0 & \cdots & 0 & c_{m1} & \cdots & c_{mn_m}
\end{pmatrix}
\]

We compare this equation with Equation 2.6 and define:

\[
A^{-T} = 
\begin{pmatrix}
\tilde{\lambda}_0 & \tilde{\alpha}_1 & \tilde{\alpha}_2 & \cdots & \tilde{\alpha}_m \\
0 & \tilde{\lambda}_1 & 0 & \cdots & 0 \\
\vdots & \vdots & \ddots & \ddots & \vdots \\
\vdots & \vdots & \ddots & 0 & \tilde{\lambda}_m \\
0 & \cdots & \cdots & 0 & \tilde{\lambda}_m
\end{pmatrix}
\]

In this neuron model:

- The first component of the state vector \( \tilde{\xi}_n \) is the potential in the soma.
- The other components represent the potential in different dendritic trees.
- Each synapse has its own weight in \( C \). The weights can be positive (excitatory synapse) or negative (inhibitory).
An Example

To be biologically plausible, the time constant of the soma should be larger than the dendrites’ time constants.

We can take, for example:

\[
A^{-T} = \begin{pmatrix}
0.99 & 1 & 1 & 1 \\
0 & 0.95 & 0 & 0 \\
0 & 0 & 0.94 & 0 \\
0 & 0 & 0 & 0.92
\end{pmatrix}
\]

\[
C^T = \begin{pmatrix}
0 & 0 & 0 \\
1 & 0 & 0 \\
0 & 1 & 0 \\
0 & 0 & -1
\end{pmatrix}
\]

\[
s = (1 \ 0 \ 0 \ 0)^T
\]

This is a model of a neuron with three dendritic trees. On each tree is only one input. Two inputs are excitatory, one is inhibitory.

The score signal shows that the model reproduces two types of summation known to be performed by neurons [46]:

- Spatial summation (inputs arriving at roughly the same time are summed in the soma) is shown in Figure 3.5a.

- Temporal summation occurs when the same input fires in small time intervals (see Figure 3.5b).

Relationship to Other Models

Over the years, many neuron models have been proposed. We compare the model presented with similar ones from the literature.

In Appendix A.2 we derive the equations of a SSM for a LIF neuron driven by exponential input currents. The equations are very similar, but \(C\) has only one element per row, because each synaptic input has its own delay.

We point out that:
Figure 3.5: Summation examples performed by the neuron model
• By comparing both SSM, one could consider that the neuron model is a special case of a leaky integrate and fire driven by exponential inputs: The synaptic inputs of the same dendritic tree are summed, each having an individual weight, but all sharing the same decay. There is evidence that dendrites could perform even more complicated operations [38,50], but in a first step, we keep the simplest model possible.

• In terms of biological mechanisms, the two models are very different: the time constant of the potential decay comes from the location of the dendrites in one case, whereas they are due to synaptic properties (for example neurotransmitter concentrations) in the other case.

As both models give similar equations, we simply exploit the fact that delays and amplitude variations are present.

A LIF driven by exponential inputs is a special case of the Spike Response Model (SRM, [14]). Especially, we have the same writing as a temporal kernel and not a differential equation as usual for the LIF. Compared to the SRM, our model has following simplifications:

• the exact form of the action potential is not considered (the output of the filter is 0 or 1)

• there is no kernel to take into account external currents

• we model refractoriness only through the guard interval after a spike (the form of the impulse response does not depend on the time to previous spikes)

• the threshold is constant.

Optimal Filters

We now turn to the design of $C$ and $s$ to detect the given monomial input pulse pattern. Let $A$ be:

$$
A = \begin{pmatrix}
\lambda_0 & 0 & \cdots & \cdots & 0 \\
\alpha_1 & \lambda_1 & 0 & \cdots & 0 \\
\vdots & & \ddots & & \vdots \\
\alpha_m & 0 & \cdots & 0 & \lambda_m
\end{pmatrix}
$$

(3.11)
with different diagonal elements $\lambda_i > 1$.

We distinguish three cases, depending on the form of matrix $C$.

The simplest version of the theorem comes if we have no constraints on the form of $C$:

Let $s \triangleq (1, 0, \ldots, 0)^T \in \mathbb{R}^{m+1}$ and let $C$ be a matrix with rows $c_1, c_2, \ldots, c_L$ given by:

- if $k_\ell = n$, $c_\ell = (1 \ 0 \ \ldots \ 0)$
- if $k_\ell \neq n$: $c_\ell$ has exactly two nonzero entries at position 0 and $i$ ($i \geq 1$, arbitrary) with

$$
\begin{align*}
  c_{\ell,0} &= \frac{\ln \lambda_0 \lambda_i^{n-k_\ell} - \ln \lambda_i \lambda_0^{n-k_\ell}}{\ln \lambda_0 - \ln \lambda_i} \\
  c_{\ell,i} &= \frac{1}{\alpha_i} \frac{\lambda_i - \lambda_0}{1 - \ln \lambda_i / \ln \lambda_0} \lambda_i^{n-k_\ell}
\end{align*}
$$

(3.12)

(3.13)

**Theorem 3.2.4 : Optimal C for Neuron Model - Version I**

Such a filter prefers the given pulse pattern, and the corresponding maximum value of the inner-product signal is $L$.

An example is given in Figure 3.6a. The proof is given in Section B.3 of the Appendix. The only condition to respect in this case is $m \geq 1$.

Obviously, this $C$ is not consistent with the model derived above: the first column of $C$ should be zero, otherwise it would mean we allow a double connection (one from a presynaptic neuron to a synapse, and another directly to the soma).

We keep, as before, $s \triangleq (1, 0, \ldots, 0)^T \in \mathbb{R}^{m+1}$. If the first column of $C$ is zero, its rows $c_1, c_2, \ldots, c_L$ are given by $c_\ell$, which has exactly two nonzero entries at position $i$ and $j$ ($i, j \geq 1$, arbitrary) with

$$
\begin{align*}
  c_{\ell,j} &= \frac{\lambda_j - \lambda_0}{\alpha_j} \frac{\ln \lambda_0 \lambda_i^{n-k_\ell} - \ln \lambda_i \lambda_0^{n-k_\ell}}{f(\lambda_j, \lambda_i, \lambda_0)} \lambda_j^{n-k_\ell}
\end{align*}
$$

(3.14)

and

$$
\begin{align*}
  f(\lambda_j, \lambda_i, \lambda_0) &= (\lambda_j^{n-k_\ell} - \lambda_0^{n-k_\ell}) \ln \lambda_i - (\lambda_i^{n-k_\ell} - \lambda_0^{n-k_\ell}) \ln \lambda_j \\
  &\quad + (\lambda_i^{n-k_\ell} - \lambda_j^{n-k_\ell}) \ln \lambda_0
\end{align*}
$$

($c_{\ell,i}$ has the same expression with $i, j$ interchanged).
Theorem 3.2.5: Optimal C for Neuron Model - Version II

Such a filter prefers the given pulse pattern, and the corresponding maximum value of the inner-product signal is $L$.

An example is given in Figure 3.6b. It is interesting to note that in this case, we need $n - k_\ell > 0$ and $m \geq 2$ in order to detect a pattern. The proof is given in Section B.3 of the Appendix.

We still have a matrix $C$ with two elements per row, which means there are double connections: Suppose for example that on the first row, the elements in columns $j_1$ and $j_2$, called $c_{1j_1}$ and $c_{1j_2}$ are not zero. This means a pulse is arriving at the same time (i.e. from the same presynaptic neuron) on dendrite $j_1$ and $j_2$. This is more plausible than before (a connection directly into the soma), because given the density of neurons, double connections surely exist, but can we also find optimal parameters for the model where we allow only one element per row of $C$?

In this case, we have to set $\lambda_i$ and $c_i$ jointly.

Theorem 3.2.6: Optimal C for Neuron Model - Version III

Let $s \triangleq (1, 0, \ldots, 0)^T \in \mathbb{R}^{m+1}$,

$$C = 
\begin{pmatrix}
0 & c_1 & 0 & \cdots & \cdots & 0 \\
0 & 0 & c_2 & 0 & \cdots & 0 \\
\vdots & \vdots & \vdots & \ddots & \ddots & \vdots \\
0 & \cdots & 0 & \cdots & 0 & c_m
\end{pmatrix}
$$

(3.15)

where

$$c_\ell = \frac{\lambda_\ell - \lambda_0}{\alpha_\ell (\lambda_\ell^{k_\ell-n} - \lambda_0^{k_\ell-n})}
$$

(3.16)

and

$$\lambda_\ell = \exp \left[ W \left( (n - k_\ell) \ln \lambda_0 \exp((n - k_\ell) \ln \lambda_0)) / (n - k_\ell) \right)^2 \right]
$$

(3.17)

Such a filter prefers the given pulse pattern, and the corresponding maximum value of the inner-product signal is $m$.

---

$^2$W denotes the Lambert W-function, see Appendix B.3
An example is given in Figure 3.6c. The proof is given in Section B.3 of the Appendix. We need again $n - k_\ell > 0$ and $m \geq 2$ in order to detect a pattern. Moreover, $C$ has $m$ rows, which means we need $m = L$ (the order of the state-space model is given by the number of input channels) for taking into account all input channels.

### 3.3 Properties of the Filters

#### 3.3.1 Robustness

Because of the linearity of Equation (2.4), we can see that erasures and extra pulses are not an issue for a large number of input pulses:

Suppose the input is not exactly the preferred pattern $y_{\text{pref}}$, there are some extra pulses and erasures (summed up in $y_{\text{noise}}$):

$$y = y_{\text{pref}} + y_{\text{noise}} \quad (3.18)$$

Then the score is:

$$s^T \xi_n = s^T \sum_{k=1}^{n} (A^{k-n})^T C^T y_{k,\text{pref}} + s^T \sum_{k=1}^{n} (A^{k-n})^T C^T y_{k,\text{noise}} \quad (3.19)$$

The first term is maximal over all monomial pulses, and as we consider $L \gg 1$, the second term is negligible:

$$s^T \xi_n \approx s^T \sum_{k=1}^{n} (A^{k-n})^T C^T y_{k,\text{pref}} \quad (3.20)$$

i.e. the filter can cope well with erasures and extra pulses.

#### 3.3.2 Redundancy

If only a few pulses are present, the threshold determines if the filter is looking for exactly one pulse pattern or some variations of it: With a high threshold (Figure 3.7a), all pulses in the pattern should be present. With a lower threshold (Figure 3.7b), we allow one missing pulse. In this case, there will be an output spike for different input patterns. This can be seen as multiple sparse representations of the same feature.
Figure 3.6: Score signal and weighting signals for neuron model
Interestingly, the idea that the same feature can be represented in sparse, different ways depending on the context, has been developed in the last years in the Hierarchical Temporal Memory, and is one of its main advantages [23].

3.4 Conclusion

This chapter has introduced monomial pulse patterns and design techniques for the inner-product filters: We have found optimal $C$ and $s$ such that the filter prefers the given pattern over all other monomial patterns.
This has been done with three different models. For each, we have presented different possibilities for $C$ and $s$, depending on the requirements of the problem.

The filters have two interesting regimes: Either they are looking for a monomial pattern with many pulses, in which case they are very robust, or they are looking at a pattern consisting of few pulses, and it is possible to introduce redundant representations.

In the next chapter, we show a first example of a complete, multi-layer network built on the principles introduced so far.
Chapter 4

First Example - Morse Code with Sinusoidal Model

4.1 Background

4.1.1 Introduction

We have introduced a theoretical architecture for multiscale pulse pattern detection, and we have shown small examples, limited to one layer. In this chapter, we present a larger example: A four-layer network that can parse Morse code.

We chose Morse code as a toy example, because it seemed to have the “right complexity”: Sequences of letters in Morse code are complex enough to be more interesting than the examples we had so far, without being as complicated as speech or music [22].

Besides being a proof of concept, this chapter also shows that the network presents interesting properties: It is self-timed, and the efficient forward only implementation is adapted to online processing.
### Table 4.1: Main elements of Morse code

<table>
<thead>
<tr>
<th>Name</th>
<th>Symbol</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Dit</td>
<td>●</td>
<td>Short signal</td>
</tr>
<tr>
<td>Dah</td>
<td>■</td>
<td>Long signal</td>
</tr>
<tr>
<td>Inter-element gap</td>
<td>□</td>
<td>Pause between dit/dah</td>
</tr>
<tr>
<td>Inter-letter gap</td>
<td>■■</td>
<td>Pause between letters</td>
</tr>
<tr>
<td>Inter-word gap</td>
<td>■■■■□□□□□□</td>
<td>Pause between words</td>
</tr>
</tbody>
</table>

#### 4.1.2 About the Morse Code

In the Morse code [56], text information is transmitted by on-off signaling. The main elements of the code are short and long signals, called “dits” and “dahs”. The duration of a dit is the basic unit of time measurement.

Three types of pauses separate dits and dahs: short (inter-element gap), medium (inter-letter gap) or long pauses (inter-word gap). These main elements are summarized in Table 4.1.

Each letter of the alphabet is represented by a unique sequence of dits and dahs.

#### 4.1.3 Training and Test Datasets

We used the International Morse Code, as described in Table C.1 of the Appendix, to transmit acoustic signals as on-off keying of a 440 Hz-tone. We used the iPad application “Morse-It” [5] to create Morse signals, recorded with the microphone of a computer. This means the signals considered in this chapter (and the next one) were transmitted over a real acoustic channel, between two different devices. Three examples of letters, as recorded with the microphone, are shown in Figure 4.1. These letters show that the complexity and time scale vary from one letter to another.

We adjusted the parameters of the network by hand on the training set. It consists of the complete alphabet, recorded four times with different orderings of the letters, assembled in small “words” (with no meaning). Then we tested the network on another recording of the alphabet, as well as some common words and short sentences.

Both sets are described in detail in Appendix C.2.
4.2 Description of the Network

In the following, we describe the different layers of the architecture. They use exclusively the sinusoidal model presented in Section 3.2.1. We applied, whenever possible, the formulas presented there. If this was not possible, we built the filters such that the highest accuracy on the training set was reached.

4.2.1 First Layer

The first layer detects the 440Hz-tone present in the input. As said in Section 2.2.3, this layer is not the main point of this thesis. Therefore, we only sketch the principle, details can be found in [8].

We apply forward message passing in the graph of Figure 4.2, where $y_k$ is the (one-dimensional) input signal at time instant $k$ (sampled at $f_s = 4000$ Hz) and

$$
A = \text{rotm} \left( 2\pi \frac{440}{f_s} \right) \quad (4.1)
$$

$$
C = \begin{pmatrix} 1 & 0 \end{pmatrix} \quad (4.2)
$$

$$
\gamma = 0.891 \quad (4.3)
$$

To detect if a tone is present, we use the score $s_n$ computed at time instant $n$ with:

Figure 4.1: Some example letters as recorded from the microphone
\[ s_n = \frac{\overrightarrow{m_x}^T \overrightarrow{W_x} \overrightarrow{m_x} - y_n^2}{\kappa_n} \] (4.4)

where \( \kappa_n = \gamma \kappa_{n-1} + y_n^2 \).

Notice that \( \overrightarrow{W_x} \) can be computed offline with (see [48]):

\[ \text{cvec} \overrightarrow{W_x} = (I_4 - \gamma (A^{-T} \otimes A^{-T}))^{-1} \text{cvec} (C^T C) \] (4.5)

Whenever the score function reaches a threshold, a pulse is produced. An example of the pulsed output of this layer is presented in Figure 4.3.

4.2.2 Higher Layers

The feature-detection filters in layers 2, 3, and 4 are roughly described in Tables 4.2, 4.3, and 4.4, respectively.

All these filters use a single frequency, as indicated in the tables (\( T \) is the duration of a dit, approximately 0.1s).

To understand how these filters work, we take the example of filter C5 (see Figure 4.4), which detects the succession of B5 (dit) and B4 (dah at the end of a letter), in that order and with a delay of 0.5s approximately:

- We start by setting \( A \), which needs to fulfill condition (3.4): a period of 1.2s (with \( \lambda = 0.9989 \)) is fine.
Now we can find $C$ and $s$ as we did in Subsection 3.2.1. This is not exactly a monomial situation, but the principle of detection can still be applied: We set $C$ and $s$ such that the filter would prefer a monomial pattern with one pulse in B5, followed by one in B4 with a delay of 0.5s.

A complete description of the filters is in Section C.3 of the Appendix.

4.2.3 Remarks

A First Working Example

We observe that this network works quite robustly. Even if it was not optimized, the network already shows good results: The filters were built such that 95% accuracy on training files (containing in total 104 letters, listed in Appendix C.2) was reached.

Table 4.5 summarizes the performance on the test files: less than 5% of the letters were missed or falsely detected.

Short Time Memory

It is interesting to note that the filters built with the sinusoidal model have a natural way of keeping information over some (short) time: Delaying pulses is the key element for recognizing a pulse pattern.
### Table 4.2: Second-Layer Feature-Detection Filters

<table>
<thead>
<tr>
<th>id</th>
<th>feature, letter</th>
<th>input</th>
<th>freq.</th>
</tr>
</thead>
<tbody>
<tr>
<td>B1</td>
<td>pause – dit</td>
<td>A1</td>
<td>1/3T</td>
</tr>
<tr>
<td>B2</td>
<td>pause – dah</td>
<td>A1</td>
<td>1/7T</td>
</tr>
<tr>
<td>B3</td>
<td>dit – pause</td>
<td>A1</td>
<td>1/9T</td>
</tr>
<tr>
<td>B4</td>
<td>dah – pause</td>
<td>A1</td>
<td>1/8T</td>
</tr>
<tr>
<td>B5</td>
<td>dit</td>
<td>A1</td>
<td>1/2.5T</td>
</tr>
<tr>
<td>B6</td>
<td>dah</td>
<td>A1</td>
<td>1/6.5T</td>
</tr>
<tr>
<td>B7</td>
<td>E</td>
<td>A1</td>
<td>1/4T</td>
</tr>
<tr>
<td>B8</td>
<td>T</td>
<td>A1</td>
<td>1/6T</td>
</tr>
<tr>
<td>B9</td>
<td>pause between words</td>
<td>A1</td>
<td>1/10T</td>
</tr>
</tbody>
</table>

### Table 4.3: Third-Layer Feature-Detection Filters

<table>
<thead>
<tr>
<th>id</th>
<th>letter</th>
<th>feature</th>
<th>input</th>
<th>freq.</th>
</tr>
</thead>
<tbody>
<tr>
<td>C1</td>
<td></td>
<td></td>
<td>B1, B6</td>
<td>1/4T</td>
</tr>
<tr>
<td>C2</td>
<td></td>
<td></td>
<td>B1, B5</td>
<td>1/9T</td>
</tr>
<tr>
<td>C3</td>
<td></td>
<td></td>
<td>B2, B5</td>
<td>1/8T</td>
</tr>
<tr>
<td>C4</td>
<td></td>
<td></td>
<td>B2, B6</td>
<td>1/18T</td>
</tr>
<tr>
<td>C5</td>
<td></td>
<td></td>
<td>B5, B4</td>
<td>1/12T</td>
</tr>
<tr>
<td>C6</td>
<td></td>
<td></td>
<td>B5, B3</td>
<td>1/2T</td>
</tr>
<tr>
<td>C7</td>
<td></td>
<td></td>
<td>B6, B3</td>
<td>1/21T</td>
</tr>
<tr>
<td>C8</td>
<td></td>
<td></td>
<td>B6, B4</td>
<td>1/4T</td>
</tr>
<tr>
<td>C9</td>
<td>A</td>
<td></td>
<td>B1, B4</td>
<td>1/12T</td>
</tr>
<tr>
<td>C10</td>
<td>I</td>
<td></td>
<td>B1, B3</td>
<td>1/9T</td>
</tr>
<tr>
<td>C11</td>
<td>N</td>
<td></td>
<td>B2, B3</td>
<td>1/9T</td>
</tr>
<tr>
<td>C12</td>
<td>M</td>
<td></td>
<td>B2, B4</td>
<td>1/15T</td>
</tr>
</tbody>
</table>
### Table 4.4: Fourth-Layer Feature-Detection Filters

<table>
<thead>
<tr>
<th>id</th>
<th>letter</th>
<th>input</th>
<th>freq.</th>
</tr>
</thead>
<tbody>
<tr>
<td>D1</td>
<td>U</td>
<td>C2, C5</td>
<td>1/12T</td>
</tr>
<tr>
<td>D2</td>
<td>S</td>
<td>C2, C6</td>
<td>1/8T</td>
</tr>
<tr>
<td>D3</td>
<td>R</td>
<td>C1, C7</td>
<td>1/12T</td>
</tr>
<tr>
<td>D4</td>
<td>W</td>
<td>C1, C8</td>
<td>1/16T</td>
</tr>
<tr>
<td>D5</td>
<td>K</td>
<td>C3, C5</td>
<td>1/12T</td>
</tr>
<tr>
<td>D6</td>
<td>D</td>
<td>C3, C6</td>
<td>1/8T</td>
</tr>
<tr>
<td>D7</td>
<td>G</td>
<td>C4, C7</td>
<td>1/12T</td>
</tr>
<tr>
<td>D8</td>
<td>O</td>
<td>C4, C8</td>
<td>1/16T</td>
</tr>
<tr>
<td>D9</td>
<td>Q</td>
<td>C4, C5</td>
<td>1/18T</td>
</tr>
<tr>
<td>D10</td>
<td>Z</td>
<td>C4, C6</td>
<td>1/18T</td>
</tr>
<tr>
<td>D11</td>
<td>L</td>
<td>C1, C6</td>
<td>1/18T</td>
</tr>
<tr>
<td>D12</td>
<td>P</td>
<td>C1, C7</td>
<td>1/24T</td>
</tr>
<tr>
<td>D13</td>
<td>V</td>
<td>C2, C5</td>
<td>1/20T</td>
</tr>
<tr>
<td>D14</td>
<td>H</td>
<td>C2, C6</td>
<td>1/15T</td>
</tr>
<tr>
<td>D15</td>
<td>F</td>
<td>C2, C7</td>
<td>1/16T</td>
</tr>
<tr>
<td>D16</td>
<td>X</td>
<td>C3, C5</td>
<td>7/120T</td>
</tr>
<tr>
<td>D17</td>
<td>B</td>
<td>C3, C6</td>
<td>1/15T</td>
</tr>
<tr>
<td>D18</td>
<td>C</td>
<td>C3, C7</td>
<td>1/18T</td>
</tr>
<tr>
<td>D19</td>
<td>J</td>
<td>C1, C8</td>
<td>1/30T</td>
</tr>
<tr>
<td>D20</td>
<td>Y</td>
<td>C3, C8</td>
<td>7/120T</td>
</tr>
</tbody>
</table>
Figure 4.4: Example of third layer pulsed output
Table 4.5: Performance of hand-crafted network with sinusoidal model
From one layer to the next, the length of the features grows. This explains why the two shortest letters (E and T) are already detected in Layer 1, whereas most letters are detected only in Layer 4. It can even happen that a feature in one layer is twice as long as its individual components (for example letter C, the combination of the outputs of filters C3 and C7).

Thus, this network already achieves a kind of multiscale pattern recognition.

4.3 Conclusion and Outlook

We have shown a first working example of a multi-layer architecture with inner-product filters.

The network was not optimized, but already works quite well.

It would be easy to add a fifth layer to detect short sequences of letters (for example “SOS”), and additional layers for longer words.

Before turning to the learning of such a network, which will be discussed in Chapter 6, we show that the network can also be built with biologically plausible neurons.
Chapter 5

Second Example - Morse Code with Neuron Model

5.1 Introduction

This chapter introduces a hand-crafted network using exclusively the
neuron model introduced in Section 3.2.3. The task of the network is
the same as before (transcript an acoustic Morse signal into a sequence
of letters), and we use the same training and test datasets.

A main difficulty when using filters built with the neuron model are
long reset times. Figure 5.2 shows the impulse response of such a fil-
ter. The score is maximal approximately 0.1s after the pulse, but then
needs about 0.5s to decay. Filters built with second-order, rotation ma-
trices have an impulse response with infinitely many oscillations, which
in practice lead to shorter reset times.

This observation triggered a new idea: for the reset time to be short,
the filters should implement only very short delays, for example the
duration of a dit. As this is the fundamental time unit of all elements of
the Morse code, it should be possible to build a complete network based
on this single delay. This is the network presented here.
5.2 Building Block

5.2.1 General Description

The basic building block of our network is a neuron with only a single dendrite, all synapses on that dendrite having the same delay.

We use following graphical notation:

- a circle represents a fixed delay,
- an arrow with a sign represents an input (the sign indicates the type of synapse).

For example, Figure 5.1 corresponds to a neuron with two inputs:

\[ y = \begin{pmatrix} x_1 \\ x_2 \end{pmatrix} \]  \hspace{1cm} (5.1)

and from the weight signs:

\[ C = \begin{pmatrix} 0 \\ 1 \\ -1 \end{pmatrix} \] \hspace{1cm} (5.2)

So far, we did not specify the delay, but we know it is the same for both inputs:

\[ A^{-T} = \begin{pmatrix} \lambda_0 & \alpha_1 & \alpha_2 \\ 0 & \lambda_1 & 0 \\ 0 & 0 & \lambda_1 \end{pmatrix} \] \hspace{1cm} (5.3)

The parameters \( \alpha_1 \) and \( \alpha_2 \) represent the strengths of the synapses. This information does not appear graphically.
5.2.2 Parameters Adapted to Morse Code Example

We now design the building block implementing a delay of the length of a dit. The delay should be adapted to our training samples: they are played at a speed of 13 Words Per Minute (WPM), which corresponds approximately to 92ms per dit (see Appendix C.1.2).

We choose $A$ as follows: we first fix $\lambda_0 = 0.9984$ and $\alpha_1 = 1$. Then we set $\lambda_1$ with (3.17) (we fix the delay to 92ms) and $c_1$ with (3.16) (we fix the maximal amplitude to be 1).

The complete filter parameters are:

$$A^{-T} = \begin{pmatrix} \lambda_0 & \alpha_1 \\ 0 & \lambda_1 \end{pmatrix}$$ (5.4)

$$C = \begin{pmatrix} 0 \\ c_1 \end{pmatrix}$$ (5.5)

$$s^T = \begin{pmatrix} 1 & 0 \end{pmatrix}$$ (5.6)

with

$$\lambda_0 = 0.9984$$ (5.7)

$$\lambda_1 = 0.9958$$ (5.8)

$$\alpha_1 = 1$$ (5.9)

$$c_1 = 0.0076$$ (5.10)

The impulse response of the basic building block of our architecture is given in Figure 5.2.

5.3 Building Simple Features

In this section, we present how to build simple features with the basic building block.

We start with a counter, which can be used for detecting a dah (three pulses in a row). Then we show how to detect the end of a symbol (small pause). The end of a letter and the end of a word can be built in the same manner. We combine dah and a small pause to detect the dits.

These main elements (dit, dah, end of word, end of letter), can be combined in a simple way to detect letters. We show the principle with one example.
5.3.1 Counter - Dah

The basic building block delays a pulse of about 92ms. As a dah is simply three pulses in a row, delayed by 92ms approximately (the duration of a dah is exactly 3 dits). This means three building blocks can be used to detect a dah (see Figure 5.3).

This construction seems elementary, but is very robust. In Figure 5.4a, we show the score signal for a dah followed by 3 dits. Even if the pulse spacing seems similar (distinguishing 3 dits from a dah with the eye is difficult), the precise impulse response allows only small variations of the delays in the pulses. The delay between dits is too large, such that there is no confusion possible.

Moreover, there is a saturation: the contribution of each new dit to the score signal is always smaller. This means that not only 3, but even 4 or more dits will not be confused with a dah. A good example for this phenomenon is letter H, see Figure 5.4b.

From this simple example, it is also clear that there is more than one version possible for some filters. For example, as the Morse code has only dit or dah, there exists no element of length 2 (2 times the duration of a dit). Therefore, two building blocks in a row could also be used as a dah detector. This could be used to introduce redundancy.
Figure 5.3: Filter for detecting dah

(a) Dah versus 3 dits in a row

(b) Dah versus 4 dits in a row

Figure 5.4: Robustness of the filter detecting dah
5.3.2 Pauses

End of Symbol - Small Pause

To see the difference between dit and dah, we need to detect pauses. After each symbol (dit or dah), there is a small pause (of the length of one dit), which we want to detect.

The construction is based on following observation: if we delay the input by one time unit, a pulse will “sit” in each interval where there was a small pause in the original input signal. Thus, the difference between the delayed input and the original one shows the position of the small pauses after a pulse in the original signal, or in other words, the end of a pulse train (of length 3 or 1).

The corresponding filter is described in Figure 5.5a.

Note that this simple operation can only be done if negative synapses are allowed.

Other Pauses

The two other types of pauses, medium (end of a letter, 3 times the length of a dit) and large (7 time units) can be built according to the
same principle as the small pause. The filters are presented in Figure 5.5b and 5.5c.

For the medium pause, we stopped after a pause of length 2. This was chosen to minimize the error on the training data. In the same manner, for the large pause, we stopped after 5 elements (see Figure 5.5c: $E_2$, a pause of length 2, is delayed 3 times).

### 5.3.3 Dit

The dits are detected every time there is a small pause (indicating the end of a dit or dah) but no dah. This is the construction of Figure 5.6 ($C_1$ is the small pause, $D_1$ is the dah).

### 5.3.4 Letter A

So far, the construction principle should be clear: combining the right elements with appropriate delays. For completeness, we describe how we detect the letter A, which is simply a dit, followed by a dah and a medium pause.

The filter is shown in Figure 5.7. The elements with positive weights are obviously dit, dah and medium pause. Figure 5.8 shows how we found the correct delays.

If a dit or a dah is present before the first dit (see Figures 5.8b and 5.8c), as for example in the letters U or K, we should not detect an A. Therefore, we add two negative components (with delays as in Figures 5.8b and 5.8c).

### 5.4 Complete Network

The complete network, for all letters of the alphabet, is described in Table C.7 of the Appendix. The construction of the filters for each letter is similar to the one described for letter A.
For the longest letters, we do not always need negative components. In total, the network turns out to be deep: it has 18 layers in total. The longest letter, which is detected in the highest layer, is J (dit, dah, dah, dah).

The network shows good performance on the test data (see Table 5.1), comparable to the example built with the sinusoidal model. It proves that the architecture proposed can be implemented in a biologically plausible way and works well as a Morse code parser.

5.5 Conclusion

We have shown a second example of a network able to understand Morse code.

As for the example built on the sinusoidal model, the network was hand-crafted and shows good performance on the test dataset.

The differences to the first example lie in the number of layers, the type and usage of the feature-detection filters: The network based on the neuron model uses a single, simple filter (even the simplest filter possible), but this filter is replicated many times.

The network based on the sinusoidal model was built with more complex filters, which were reused for different letters (for example, most filters of layer 3 are used for the detection of 4 or 5 different letters).

In the following, we will show how to learn the parameters of the inner-product filters in such networks.
Figure 5.8: Construction of the filter to detect A

(a) Delays to detect A (positive weights)

(b) Delay to avoid confusion with U (negative weight)

(c) Delay to avoid confusion with K (negative weight)
<table>
<thead>
<tr>
<th>Letter</th>
<th>Occurrence</th>
<th>False Negative</th>
<th>False Positive</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>20</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>B</td>
<td>11</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>C</td>
<td>10</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>D</td>
<td>14</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>E</td>
<td>40</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>F</td>
<td>6</td>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td>G</td>
<td>9</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>H</td>
<td>14</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>I</td>
<td>27</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>J</td>
<td>2</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>K</td>
<td>3</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>L</td>
<td>22</td>
<td>4</td>
<td>1</td>
</tr>
<tr>
<td>M</td>
<td>5</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>N</td>
<td>18</td>
<td>0</td>
<td>3</td>
</tr>
<tr>
<td>O</td>
<td>19</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>P</td>
<td>10</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Q</td>
<td>2</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>R</td>
<td>21</td>
<td>0</td>
<td>4</td>
</tr>
<tr>
<td>S</td>
<td>16</td>
<td>4</td>
<td>1</td>
</tr>
<tr>
<td>T</td>
<td>24</td>
<td>4</td>
<td>0</td>
</tr>
<tr>
<td>U</td>
<td>6</td>
<td>2</td>
<td>0</td>
</tr>
<tr>
<td>V</td>
<td>5</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>W</td>
<td>7</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>X</td>
<td>2</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>Y</td>
<td>7</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>Z</td>
<td>2</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>space</td>
<td>47</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>Total</td>
<td>369</td>
<td>18 (4.9%)</td>
<td>16 (4%)</td>
</tr>
</tbody>
</table>

**Table 5.1:** Performance of hand-crafted network with neuron model
Part II

Learning Algorithms
Chapter 6

Supervised Learning with Sinusoidal Model

6.1 Introduction

The Morse code example showed that the multi-layer architecture of inner-product filters can work in a concrete situation. Nevertheless, setting all parameters by hand was obviously not ideal. For more complex examples, it may even not be possible. We therefore turn to the idea of learning, and develop it for a network that can differentiate 20 short piano tunes.

In this chapter, we focus on the sinusoidal model, i.e. we use inner-product filters as defined in Section 2.3 with

$$A = \lambda \text{rotm}(\Omega)$$  \hspace{1cm} (6.1)

where $\lambda$, $\Omega$ and $x_n = s$ are given.

We will show how to learn $C$ with a simple supervised learning method, stochastic gradient descent [3]. This method has been widely applied in the field of pattern recognition, and was especially the first one successfully applied to a deep architecture [31,32].
6.2 Data and Goal

6.2.1 Recordings

We played short (between 2s and 5s duration) piano tunes on an iPad and recorded them with a microphone. In total, there are 20 different tunes, each one labeled with a number between 1 and 20 (see Appendix D.1 for a detailed description). We recorded each tune 50 times.

For each tune, we divide the 50 samples in three categories:

- 30 samples are used for training,
- 10 samples are used for evaluating the performance of the algorithm during training (stopping criterion),
- 10 samples are used for testing, when the training is done.

The total training set $S$ consists of 600 samples, the validation set $S_V$ and the test set $S_T$ have 200 samples.

Goal

We want to train a network that has 20 output channels. If we play tune $i$, it should produce a pulse at the end of the tune in channel $i$, and no pulse anywhere else.

First Layer

As for the Morse code example, the first layer is a tone detector. We build it exactly as described in Subsection 4.2.1, with 13 different frequencies. The parameters are in Appendix D.2.1. This layer is fixed, the learning is done on the higher layers.

An Example

An example of a melody is given in Figure 6.1. The first row shows the notes played, the second is the input of the microphone, the next rows are the tones (between an A of frequency 220 Hz and A of frequency 440 Hz), and the last row shows the goal of our learning algorithm: We want to detect the tune at the position of the last pulse (shown is only the score of the input tune, all other scores should be 0 everywhere).
Figure 6.1: First layer for Melody 1, and goal score function (after learning)
Variability

The tunes were played by hand, therefore there are little variations between the elements of the training set. We did not add any simulated noise. Figure 6.2 shows a superposition of all 50 recordings for the first melody. It is clear that time variations in the positions of the pulses are present. There are also some extra pulses on nearby channels (for example $D\#$ instead of $E$).

6.2.2 Architecture

About Feature Sharing

We would like to train a hierarchical architecture where the first layers are shared, and only the highest layers are specifically trained for each input tune. The low, shared layers should capture simple features that are present in multiple tunes, for example, two or three successive notes.

Detailed Description

We consider as given (see Appendix D.2 for all exact values):

- the number of layers: 3 (we will go deeper in the next chapter);
- all matrices $A$ on each layer, using only one frequency (i.e. $m = 2$);
- $s = (1\ 0)^T$;
- the thresholds (they are all set to 1) and guard spaces.

Concerning the number of filters (see Figure 6.3):

- We use 13 filters in the first layer (for detecting 13 different notes). These are shared and fixed as explained above.
- We use 8 intermediate filters (shared).
- We use 20 filters on the last layer (each filter is trained individually for each tune).

We want to learn:

- the matrices $C_{2,i} \in \mathbb{R}^{13 \times 2}$ in the second layer, $1 \leq i \leq 8$,
- the matrices $C_{3,j} \in \mathbb{R}^{8 \times 2}$ in the third layer, $1 \leq j \leq 20$.

This gives 528 parameters in total.
Figure 6.2: Superimposed recordings of Melody 1
Score Signal

We use the pulsed score signal presented in Section 2.3: \( s^T \xi_n \). As this score is linear with respect to \( C \), we can use a fix threshold.

6.3 Implementation of the Learning Algorithm

For clarity, we show the update equations for our 3-layer network. The generalization to more layers is straight-forward.

We also restrict the description to learning one matrix of layer 3, called \( C_{3,X} \), together with all matrices of layer 2 (\( C_{2,i}, 1 \leq i \leq 8 \)). A positive sample in this setting is tune \( X \). A negative sample could be any other tune.

In practice, as we use a stochastic method, we learn all matrices of layer 3 simultaneously (see Section 6.3.3).
6.3 Implementation of the Learning Algorithm

6.3.1 Cost Function

Consider as input tune $X$, which is passed through the first layer of the network. The output of this layer is a pulse-domain signal having 13 components. We denote $t_P$ the time index of the last pulse present in this signal; this pulse could be in any channel. In the last layer of the network, we want the score function to be zero everywhere, except at $t_P$ (where it should be 1). This corresponds to a cost function:

$$J_P = \sum_{i \neq P} s_c(t_i)^2 + (s_c(t_P) - 1)^2$$

where $s_c$ is the (pulsed) score function in the last layer of the network.

If the input was instead a negative training sample, the score function in the last layer should be zero everywhere:

$$J_N = \sum_i s_c(t_i)^2$$

This gives the total cost function:

$$J = J_P + J_N$$

6.3.2 Equations

**Highest layer - Update Equations for $C_{3,X}$**

From now on, we drop the index ($C_{3,X} = C$) so that the notation is not overloaded. An element of $C$ is written $c_{kl}$ where $k, l$ run over all rows, resp. columns of $C$.

The computation of the derivative has to be done carefully, because of the thresholding. We use a symmetric derivative to simplify the computations and introduce $\tilde{s_c}^{kl}$, the (non-thresholded) score in the last layer, obtained when $C$ of the last layer is replaced by $1_{kl}$ (an all-zero matrix except for a 1 in row $k$ and column $l$).

The complete derivation is shown in Section D.3 of the Appendix:

$$c_{kl}^{new} = c_{kl}^{old} - \epsilon \left( \frac{\partial J_P}{\partial c_{kl}} + \frac{\partial J_N}{\partial c_{kl}} \right)$$

(6.2)

with
\[
\frac{\partial J_P}{\partial c_{kl}} = \sum_{i \neq P} \frac{sc(t_i)}{\tilde{sc}(t_i) - th} \frac{\tilde{sc}^{kl}(t_i)}{th - \tilde{sc}(t_P) + \eta} + \frac{(sc(t_P) - 1)}{th - \tilde{sc}(t_P) + \eta} \tilde{sc}^{kl}(t_P) 
\]

(6.3)

where \(\eta = 0.001\), and

\[
\frac{\partial J_N}{\partial c_{kl}} = \sum_i \frac{sc(t_i)}{\tilde{sc}(t_i) - th} \tilde{sc}^{kl}(t_i) 
\]

(6.4)

**Lower layer - Update Equations for \(C_2\)**

Each \(C_{2,i}\), for \(1 \leq i \leq 8\), is used in the computation of the score of one channel in layer 2.

For this channel \(\ell\), we need to know where we would like to have a pulse (we call this time instant an “intermediate endpoint”, and denote it \(t_\ell\)), in order to produce a pulse at \(t_P\) in the last layer (i.e. in order to detect the tune). Once the intermediate endpoints are known, we can define a similar cost function as for the highest layer.

We use \(A = \lambda \text{rotm}(\Omega)\), \(C\) and the endpoint \(t_P\) (all of layer 3), to set the endpoints of the previous layer:

If \(c_\ell\) is the \(\ell\)th-row of \(C\), we write \(c_\ell = (c_{\ell1} \ c_{\ell2}) = \mu_\ell (\cos \beta_\ell \ \sin \beta_\ell)\) with \(\mu_\ell > 0\) and \(\beta_\ell \in [-\pi, \pi]\).

As shown in Appendix D.3.2:

\[
t_\ell = t_P + \frac{\beta_\ell + \arctan \left( \frac{\ln \frac{\lambda}{\Omega}}{\Omega} \right)}{\Omega} 
\]

(6.5)

If \(t_\ell > t_P\), we use \(t_\ell = t_P\). This strategy could be used recursively to set intermediate endpoints until the lowest layer.

The cost function for channel \(\ell\) is the sum of the cost for a positive sample:

\[
J_P^\ell = \sum_{i \neq \ell} sc(t_i)^2 + (sc(t_\ell) - 1)^2 
\]

and the cost for a negative sample:

\[
J_N^\ell = \sum_i sc(t_i)^2
\]
6.3 Implementation of the Learning Algorithm

We compute the derivation of the cost with respect to $C_{2,\ell}$. The expression is the same as for the highest layer, but with a different endpoint.

A subtle difference comes from implementing shared features: As layer 2 is shared for all tunes, we have to update $C_{2,\ell}$ with an average over a subset of $N_b$ tunes.

If we keep both terms in the cost function, $J_\ell^N$ will tend to suppress too many pulses in the intermediate layers ($J_\ell^P$ “suppresses” already all pulses except one). From trials, we found out that updating the intermediate layers only with $J_\ell^P$ worked best for shared features.

The final formula to update $c_{kl}$, an element of $C_{2,\ell}$ is:

$$c_{kl}^\text{new} = c_{kl}^\text{old} - \epsilon \sum_{i \neq \ell} \frac{\partial J_\ell^P}{\partial c_{kl}}$$

with

$$\frac{\partial J_\ell^P}{\partial c_{kl}} = \sum_{i \neq \ell} \frac{sc(t_i)}{sc(t_i) - th} \tilde{sc}^{kl}(t_i) + \frac{(sc(t_\ell) - 1)}{th - sc(t_\ell)} - \tilde{sc}^{kl}(t_\ell) + \eta \tilde{sc}^{kl}(t_\ell)$$

and $\eta = 0.001$.

The sum in (6.6) runs over a subset of $N_b$ training samples.

We apply this strategy for all $\ell \in \{1, \ldots, 8\}$. As the highest layer can give satisfying results even if not all pulses are present in all channels of the intermediate layers, we update the lower layers only if the highest changed.

6.3.3 Pseudo-Code of the Algorithm

Given are all $\lambda, \Omega$ and guard spaces. The learning rates (one per layer) are also fixed (they were chosen such that the algorithm had a possibly fast convergence).

As the first layer of the network is fixed, we compute the output of layer 1 offline. We take these pulsed signals as the input for our learning algorithm. For an individual tune, the training set $S$ consists of 30 positive samples ($S_+$), and 19*30 negative samples (all other tunes: $S_-$).
Algorithm 6.3.1 : Learning Algorithm for Second Order Filters

1. Initialize all $C_{2,i}$ for $1 \leq i \leq 8$ and $C_{3,j}$ for $1 \leq j \leq 20$.

2. For $l = 1 : n_{\text{steps}}$
   
   (a) Update for all $i$: $C_{2,i}^{\text{old}} \leftarrow C_{2,i}$
   
   (b) For $k = 1 : N_b$
   
   i. Take a positive sample $y_p \in S_+$, and a negative sample $y_n \in S_-$
   
   ii. Determine the position of the last pulse in $y_p$, use it (with $C_{3,p}$ and the corresponding $A$) to set intermediate endpoints according to (6.5)
   
   iii. Do message passing with all $C_{2,i}^{\text{old}}$ for $y_p$ and $y_n$, store thresholded and non-thresholded score functions, and the non-thresholded scores $\tilde{s}_{kl}$, $k,l$ running over all indices of $C$ (for all $C_{2,i}$ and for $C_{3,p}$)
   
   iv. Update $C_{3,p}$ with (6.2)
   
   v. If $C_{3,p}$ has been changed: update all $C_{2,i}$ with (6.6)

(c) Every 50 steps:
   
   i. Evaluate the cost on 10 positive sample of $S_V$ and 10 negative samples of $S_V$
   
   ii. If cost($l$) $\leq 0.05$ and cost($l - 50$) $\leq 0.05$, stop training

6.3.4 Some Implementation Details

Parameters

We used following parameters:

- Mini-batch size: $N_b = 20$
- Learning rate for the second layer: $\epsilon_2 = 0.005$
- Learning rate for the third layer: $\epsilon_3 = 0.02$
- We used random numbers between -1 and 1 for initialization of $C_{2,i}$ for $1 \leq i \leq 8$ and $C_{3,j}$ for $1 \leq j \leq 20$. 
6.4 Results

About the “Endpoint” of the Tune

Requiring the score to be 1 exactly at the position of the last pulse is very hard. Especially for tunes where the last note had a duration of two pulses or more (and the number of pulses can vary depending on the recording), this lead to “false errors”: The tune is recognized at the onset of the last note, and this is too early according to our learning rule, meaning the cost is the same as if the tune was not detected!

We solved this issue by detecting the melody once it is “approximately” over, as follows: If there is a pulse in the last layer for a positive sample, and if this pulse is close to (meaning occurring less than 0.5s before or after) the wanted endpoint, we use it as a new endpoint.

About the Threshold

It is an advantage of the algorithm that the threshold can be set beforehand. We noticed that the learned values in $C$ are bounded. We conjecture that this is a nice side-effect of the constant threshold (there is no need to continue increasing $C$ once the score has reached a threshold).

6.4 Results

6.4.1 Accuracy

Table 6.1 presents the performance of the algorithm on the test set, which was not used for training. The overall performance is good, showing only 3% of false negatives and 4% of false positives.

6.4.2 An Example of Learned Feature

It is interesting to have a look at the intermediate features, to have an idea of what was learned. We took the learned $C_{2,6}$ of layer 2 and computed the preferred pattern of the output pulses. They are shown in Figure 6.4.

The filter produces two pulses. We marked them in different colors. The preferred patterns are superimposed in color to the pulses present in the input channels.

The first (red) pulse fires because of the two pulses in channel E and G. This means the filter looks for the note E followed by G, with a delay
\begin{table}
\centering
\begin{tabular}{|c|c|c|}
\hline
Tune & False Negative & False Positive \\
\hline
1 & 0 & 0 \\
2 & 0 & 0.1 \\
3 & 0 & 0 \\
4 & 0 & 0 \\
5 & 0.1 & 0 \\
6 & 0 & 0.06 \\
7 & 0 & 0 \\
8 & 0.1 & 0.08 \\
9 & 0 & 0.06 \\
10 & 0 & 0.02 \\
11 & 0.2 & 0 \\
12 & 0 & 0 \\
13 & 0 & 0.06 \\
14 & 0 & 0.08 \\
15 & 0.2 & 0.02 \\
16 & 0 & 0.08 \\
17 & 0 & 0.03 \\
18 & 0 & 0.01 \\
19 & 0 & 0.24 \\
20 & 0 & 0 \\
\hline
Avg & 0.03 & 0.04 \\
\hline
\end{tabular}
\caption{Errors for training algorithm with second order filters}
\end{table}
of about one second. Note that one of the red dashed pulse is almost exactly at the optimal position.

The second pulse (blue) fires because of the pulses in channels E and D, delayed by about 0.5s.

This shows that the intermediate features seem to be small subsets of a melody: Some notes, in a precise order and with a precise delay.

### 6.4.3 Conclusion and Outlook

This chapter showed a first example of learning the layered network of feature-detection filters.

The pulses are helpful in various respects: First, they help to define a simple quadratic cost function. Second, based on the notion of preferred pulse pattern, we could develop a new method of “backpropagating endpoints”, where intermediate endpoints are recursively set from the highest to the lowest layer. Each channel then has its own cost function, which is nevertheless dependent on the higher layers, through the dependencies of the endpoints.

The performance of the network is good, even if the gradient could not be computed exactly. We also showed an example of learned intermediate features, which confirms that the network learns basic features present in the inputs.

This was only a first step. There are many ways in which this work can be continued: other examples, with larger datasets have to be tested. We also need to go deeper, as 3 layers are still a shallow architecture. Online learning would also be very interesting, as the scores are computed only with forward messages.
Figure 6.4: Filter 6 of layer 2, in dashed two preferred patterns for the output pulses
Chapter 7

Supervised Learning with Neuron Model

7.1 Introduction

We have shown learning in a network built on the sinusoidal model. In this chapter, we show how to learn $C$ in a network using the neuron model, and we focus on the case without double connections, in which we need to learn both $C$ and $A$ jointly.

We use the same dataset as in the previous chapter and demonstrate a similar supervised learning algorithm based on gradient descent. We start with 3 layers to show that the method is feasible. Then we extend the network to 4 layers.

As the network is a kind of Spiking Neural Network [14, 19, 45], we precise the links to similar algorithms in the literature.

7.2 Architecture

In Section 3.2.3, where we derived the neuron model and computed $C$ and $s$ for detecting a given monomial pulse pattern, we encountered three types of matrices $C$: a full one (which means there are double connections with direct paths to the soma), one with the first column
being zero (meaning double connections only between synapses) and one with the first column being zero and only one element per row.

The first two cases would amount to learning $C$, as we did in the previous chapter. The last case is more interesting: the coefficients in $C$ change only the scaling of the impulse response. To change also the shape, we need to learn the diagonal elements in $A$.

Thus we consider layers of filters with:

$$A^{-T} = \begin{pmatrix} 0 & \lambda_1 & 0 & \cdots & 0 \\ \lambda_0 & 1 & 0 & \cdots & 1 \\ \vdots & \vdots & \ddots & \ddots & \vdots \\ 0 & 0 & \cdots & 0 & \lambda_m \end{pmatrix}$$

$$C^T = \begin{pmatrix} 0 & 0 & 0 & \cdots & 0 \\ c_1 & 0 & 0 & \cdots & 0 \\ \vdots & \vdots & \ddots & \ddots & \vdots \\ 0 & 0 & \cdots & 0 & c_m \end{pmatrix}$$

We consider as given:

- the number of layers: 3
- all coefficients $\lambda_0$ in the matrices $A$ (they are all taken equal to 0.9999)
- $s = (1, 0, \cdots, 0)^T \in \mathbb{R}^{m+1}$;
- the thresholds (they are all set to 1).

Concerning the number of filters (see Figure 6.3):

- We use 13 filters in the first layer (for detecting 13 different notes). These are shared and fixed as before.
- We use 20 intermediate filters (shared), of size $14 \times 14$.
- We use 20 filters on the last layer (each filter of size $21 \times 21$ is trained individually for each tune).

We want to learn:

- the nonzero elements in $C_{2,i}$ in the second layer, as well as all $\lambda$ in $A_{2,i}$, except the first one, $1 \leq i \leq 20$
- the nonzero elements in $C_{3,j}$ in the third layer, all $\lambda$ in $A_{3,j}$, except the first one, $1 \leq j \leq 20$.

The number of parameters to learn is 1320 ($13 \times 2 \times 20 + 20 \times 2 \times 20$).
7.3 Implementation of the Learning Algorithm

7.3.1 Cost Function

We work with the same cost function as in the previous chapter. We consider tune $X$ as input. It is passed through the first layer of the network, and we denote $t_P$ the time index of the last pulse present in this signal.

In the last layer of the network, we want the score function to be zero everywhere, except at $t_P$ (where it should be 1). This corresponds to a cost function:

$$J_P = \sum_{i \neq P} sc(t_i)^2 + (sc(t_P) - 1)^2$$

where $sc$ is the (pulsed) score function in the last layer of the network.

If the input was instead a negative training sample, the score function in the last layer should be zero everywhere:

$$J_N = \sum_i sc(t_i)^2$$

This gives the total cost function:

$$J = J_P + J_N$$

7.3.2 Equations

Update of $C$

The update rule is the same as in the previous chapter (this rule depends only on messages, which can be computed with any $A$), but we need to learn only the nonzero elements of $C$: $c_1, \ldots, c_m$.

We simplify the indices of the update equations given in (6.2), (6.7) and (6.4).

For $k \in \{1, \ldots, m\}$

$$c_k^{new} = c_k^{old} - \epsilon \left( \frac{\partial J_P}{\partial c_k} + \frac{\partial J_N}{\partial c_k} \right)$$ (7.1)
Supervised Learning with Neuron Model

with

\[
\frac{\partial J_P}{\partial c_k} = \sum_{i \neq P} \frac{sc(t_i)}{\tilde{sc}(t_i)} \tilde{sc}^k(t_i) + \frac{(sc(t_P) - 1)}{th - \tilde{sc}(t_P) + \eta} \tilde{sc}^k(t_P) \tag{7.2}
\]

where \(\eta = 0.001\), and

\[
\frac{\partial J_N}{\partial c_k} = \sum_i \frac{sc(t_i)}{\tilde{sc}(t_i)} \tilde{sc}^k(t_i) \tag{7.3}
\]

with \(\tilde{sc}^k(t_i)\) defined as before (it is the non-thresholded score for \(c_k = 1\), all others being 0)

**Update of \(\lambda\)**

We need to control that the value of the learned \(\lambda_i\) does not exceed 1, otherwise we have stability issues. One (classic) way is to introduce a sigmoid, denoted \(\sigma\).

\[
\lambda = \sigma(x) = \frac{1}{1 + \exp(-x)} \tag{7.4}
\]

The computation of the update rule for \(x\) is detailed in [D.4.1]:

\[
\lambda^{new} = \sigma(x^{new}) \tag{7.5}
\]

where

\[
x^{new}_k = x^{old}_k - \epsilon \frac{\partial J}{\partial \lambda_k} \sigma(x_k)(1 - \sigma(x_k)) \tag{7.6}
\]

with

\[
\frac{\partial J}{\partial \lambda_k} = \frac{\partial J_P}{\partial \lambda_k} + \frac{\partial J_N}{\partial \lambda_k} \tag{7.7}
\]

and
\[
\frac{\partial J_P}{\partial \lambda_k} = \sum_{i \neq P} \frac{sc(t_i)}{\hat{sc}(t_i)} \left( \frac{(t_i - m_k)r_k}{\lambda_k(\lambda_k - \lambda_0)} - \frac{c_k\hat{sc}_k(t_i)}{\lambda_k - \lambda_0} \right) + \frac{sc(t_P) - 1}{th - \hat{sc}(t_P)} \left( \frac{t_P - m_k}{\lambda_k(\lambda_k - \lambda_0)}r_k - \frac{c_k\hat{sc}_k(t_P)}{\lambda_k - \lambda_0} \right)
\]
(7.8)

\[
+ \frac{sc(t_P)}{\hat{sc}(t_P)} \left( \frac{t_P - m_k}{\lambda_k(\lambda_k - \lambda_0)}r_k - \frac{c_k\hat{sc}_k(t_P)}{\lambda_k - \lambda_0} \right) - c_k\hat{sc}_k(t_P)
\]
(7.9)

\[
\frac{\partial J_N}{\partial \lambda_k} = \sum_i \frac{sc(t_i)}{\hat{sc}(t_i)} \left( \frac{(t_i - m_k)r_k}{\lambda_k(\lambda_k - \lambda_0)} - \frac{c_k\hat{sc}_k(t_i)}{\lambda_k - \lambda_0} \right)
\]
(7.10)

\(r_k\) is the \((k + 1)th\) component in the state vector, \(\hat{sc}_k\) is the non-thresholded score function for \(c_k = 1\) (all other elements being 0), and \(m_k\) is the position of the last pulse before \(t_i\) (resp. \(t_P\)) in channel \(k\).

**Computation of Intermediate Endpoints**

As before, we need intermediate endpoints for all channels of layer 2. We use \(A\) of layer 3 to set

\[
t_i = t_P - \frac{\ln \left( \frac{\ln \lambda_i}{\ln \lambda_0} \right)}{\ln \lambda_i - \ln \lambda_0}
\]
(7.11)

for \(i \in \{1, \ldots, 20\}\).

The computation is shown in [D.4.2].

### 7.3.3 Pseudo-Code

Given are all \(\lambda_0\) and guard spaces. The learning rates are also fixed: \(\epsilon_2^C, \epsilon_3, \epsilon_3^C\)

We denote the filters of the intermediate layer \((A_{2,i}, C_{2,i})\), and those of the highest layer \((A_{3,j}, C_{3,j})\).

The algorithm has the same steps as the one for the sinusoidal model. We repeat it for clarity and to refer to the right update equations.

**Algorithm 7.3.1**

1. Initialize all \(A_{2,i}, C_{2,i}\) for \(1 \leq i \leq 20\) and \(A_{3,j}, C_{3,j}\) for \(1 \leq j \leq 20\)

2. For \(l = 1 : n_{steps}\)
(a) Update for all $i$: $C_{2,i}^{\text{old}} \leftarrow C_{2,i}$ and $A_{2,i}^{\text{old}} \leftarrow A_{2,i}$

(b) For $k = 1 : N_b$

i. Take a positive sample $y_p \in S_+$, and a negative sample $y_n \in S_-$.

ii. Determine the position of the last pulse in $y_p$, use it (with $A_{3,p}$) to set intermediate endpoints according to (7.11).

iii. Do message passing with all $A_{2,i}^{\text{old}}, C_{2,i}^{\text{old}}$ for $y_p$ and $y_n$, store thresholded and non-thresholded score functions, and the non-thresholded scores $\tilde{s}_c k$, $k$ running over all nonzero indices of $C$ (for all $C_{2,i}$ and for $C_{3,p}$)

iv. Update $C_{3,p}$ with (7.1) and $A_{3,p}$ with (7.6)

v. If $C_{3,p}$ or $A_{3,p}$ has been changed: update all $C_{2,i}$ and $A_{2,i}$.

(c) Every 50 steps:

i. Evaluate the cost on 10 positive sample of $S_V$ and 10 negative samples of $S_V$

ii. If $\text{cost}(l) \leq 0.05$ and $\text{cost}(l - 50) \leq 0.05$, stop training

7.3.4 Implementation Details

Parameters

We used

- Mini-batch size: $N_b = 20$
- Learning rates for the second layer: $\epsilon_{C2} = 0.005$, $\epsilon_2 = 0.0005$
- Learning rates for the third layer: $\epsilon_{C3} = 0.02$, $\epsilon_3 = 0.003$

Endpoints

We use an “approximate” endpoint as for the sinusoidal model (see Section 6.3.4).

Initialization

- In layer 2: $1/\lambda_i \in [0.999; 0.9995]$
- In layer 3: $1/\lambda_i \in [0.9995; 0.9998]$
- The nonzero values in all $C$ are random between -1 and 1.
### 7.4 Results

#### 7.4.1 Accuracy

The overall performance is summarized in Table 7.1.

Even if we had to learn more parameters as in the sinusoidal model, the training results are comparable.

#### 7.4.2 Going Deeper

The main difference to the previous chapter is the number of intermediate layers (20 instead of 8).

As we have seen in the Morse code of Chapter 5, it could be helpful to add layers.

This would also confirm that the algorithm is working for more than

---

**Table 7.1:** Results for training algorithm with neuron model

<table>
<thead>
<tr>
<th>Tune</th>
<th>False Negative</th>
<th>False Positive</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0.4</td>
<td>0.24</td>
</tr>
<tr>
<td>2</td>
<td>0.1</td>
<td>0.12</td>
</tr>
<tr>
<td>3</td>
<td>0</td>
<td>0.08</td>
</tr>
<tr>
<td>4</td>
<td>0</td>
<td>0.01</td>
</tr>
<tr>
<td>5</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>6</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>7</td>
<td>0</td>
<td>0.05</td>
</tr>
<tr>
<td>8</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>9</td>
<td>0</td>
<td>0.1</td>
</tr>
<tr>
<td>10</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>11</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>12</td>
<td>0</td>
<td>0.06</td>
</tr>
<tr>
<td>13</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>14</td>
<td>0.1</td>
<td>0.43</td>
</tr>
<tr>
<td>15</td>
<td>0</td>
<td>0.04</td>
</tr>
<tr>
<td>16</td>
<td>0</td>
<td>0.16</td>
</tr>
<tr>
<td>17</td>
<td>0.3</td>
<td>0.01</td>
</tr>
<tr>
<td>18</td>
<td>0</td>
<td>0.13</td>
</tr>
<tr>
<td>19</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>20</td>
<td>0</td>
<td>0.06</td>
</tr>
<tr>
<td>Avg</td>
<td>0.04</td>
<td>0.07</td>
</tr>
</tbody>
</table>
one hidden layer. As a first step, we implemented a learning on 4 layers (i.e. two hidden layers). We used following architecture:

- 13 filters in the first layer, fixed, shared
- 10 filters in the second layer, shared
- 3 filters per tune in the third layer
- 1 filter per tune in the fourth layer

We used Algorithm 7.3.1 to learn layers 2, 3 and 4. There was no major difficulty, except that we needed to find suitable learning rates for each layer (one rate for A and another for C). We followed the recipe in [1]: Set the learning rate as large as possible while still observing convergence. This resulted in many trials. For more layers, we would probably need to find more elaborate strategies. Some interesting suggestions can be found in [6].

7.5 Relationship to Other Learning Algorithms in the Literature

Spiking Neural Network (SNN) are a large research topic. We found a number of topics related to our approach, which we describe in the following.

7.5.1 SpikeProp

Probably the first proposal of a backpropagation rule in a SNN taking the temporal aspect into account was the SpikeProp algorithm [4].

The setting of the SpikeProp algorithm is slightly different: There is only one impulse response, shifted multiple times to obtain different delays. The neuron model allows for different delays but also different shapes of the impulse response. The adaptation of the delays is a key element in the learning.

Moreover, learning in the SpikeProp algorithm is done on actual pulses (i.e. pulses present in the output). If no spike is present, the weights are not updated. This means that a silent neuron will never be updated. In contrast, the learning algorithm presented here will update weights also if no pulse is present.
7.5.2 Tempotron

The Tempotron uses a LIF neuron driven by exponential synaptic inputs (i.e. the same model as we use). The learning is done on the weights, and double connections are allowed. The main difference is in the learning goal: There should be at least one firing while the pattern is presented. The time index of detection can be anywhere. In practice, the time instant where the potential in the highest layer is maximal is chosen: the weights are updated such that a pulse occurs there. This has the advantage of simplifying computations, but also means the learned output spikes cannot be anywhere. An online setting is also not possible.

7.5.3 Chronotron

Many papers deal with the learning of an input-output relationship. For example, which is a continuation, with more theoretical justification of .

The learning is done by comparing two spike trains (the actual one and the wanted one). This would be hard to apply for the intermediate layers, because it is not clear what they should do exactly. Our algorithm suggests points where pulses should be present, but these points are updated in each step.

7.5.4 Another Framework for Spatio-Temporal Spike Pattern Recognition

Closer to our work is , which presents a framework for spatio-temporal spike pattern recognition. It is interesting to note that the architecture presented there is different: a dendrite is modeled as a summation followed by a non-linearity.

First, this produces a very flat output, and an optimal threshold for the output signal has to be found. Our algorithms work with a single threshold, as the weights in are adapted.

Second, the use of delays is only implicit: they are crucial in the learning, but the reason is only clear if one adapts the delays to correspond to the input pulse pattern.

7.5.5 Conclusion

The algorithm presented in this chapter was strongly inspired by the algorithm for the sinusoidal model, which is probably the main reason
it is hard to compare it with algorithm from the SNN literature. Never-
theless, we found some papers using a similar neuron model as ours and
similar ideas.

But the learning of the delays ($\lambda$) and weights ($C$) as presented in
this chapter is new: The key element in our algorithm are the inter-
mediate endpoints, which are “good” positions for pulses. The learned
intermediate features presented in the previous chapter show that this
approach, based on preferred patterns, makes sense.

We believe this algorithm is suitable for learning deep networks,
which can deal with highly complex signals.
Chapter 8

Unsupervised Learning

8.1 Introduction

So far, we have presented supervised learning, where each training sample comes with a label. In this chapter, we propose algorithms for unsupervised learning.

Expectation Maximization (EM) has been formulated as a message-passing algorithm in [9]. As we have seen in the previous chapter, finding the learning rates of each layer may be a problem in a deep scenario. One of the advantages of EM is that it does not need a learning rate.

For a start, we consider only single-layer networks. We begin with the sinusoidal model, where we can formulate an EM learning for $C$. In the second part of this chapter, we look at the same algorithm for the neuron model. There, the problem is more complicated, but we propose some solutions.

8.2 Sinusoidal Model

8.2.1 Setup

In this section, we focus on the sinusoidal model, i.e. we use inner-product filters as defined in Section 2.3 with

$$A = \lambda \text{rotm}(\Omega)$$  \hspace{1cm} (8.1)
where $\lambda$, $\Omega$ and $s_n = s$ are given. We show how to learn $C$ with EM.

### 8.2.2 Derivation of EM-Update Rules

We start with the factor graph of Figure 8.1 (the same as in Chapter 2, redrawn here for clarity).

In this graph:

$$Y_n = C\tilde{X}_n + Z_n$$  \hspace{1cm} (8.2)

$$= \begin{pmatrix} c^1 \tilde{X}_n + z^1_n \\ \vdots \\ c^L \tilde{X}_n + z^L_n \end{pmatrix}$$  \hspace{1cm} (8.3)

where $c^i$ ($1 \leq i \leq L$) denotes a row of $C$ and
From these computations, it is clear that Figure 8.2 is an equivalent formulation of Figure 8.1. Estimating $C$ then amounts to estimating all its rows separately.

We follow the rules described in [9]. The factor graph used for computing messages needed for the EM-updates is in Figure 8.3.

For the update equations of $c^i$, we need the backward messages out of the multiplicative nodes (drawn as blue arrows in Figure 8.3).

For $i = 1, \ldots, L$

$$c^i_{new} = \left(\sum_{k=1}^{n} \gamma^{n-k} \hat{W}_{c_k^i}\right)^{-1} \left(\sum_{k=1}^{n} \gamma^{n-k} \hat{W}_{c_k^i} \hat{m}_{c_k^i}\right)$$  \quad (8.5)

and from [9]:

$$\hat{W}_{c_k^i} = V_{\hat{X}_k} + m_{\hat{X}_k} m_{\hat{X}_k}^T$$  \quad (8.6)
Figure 8.3: Factor graph for deriving EM-updates
\[ \hat{W}_{c_k}^i \hat{m}_{c_k}^i = m_{\hat{X}_k} y_k^i \] (8.7)

(Note that \( \hat{W}_{c_k}^i \) is the same for all \( i \)).

As \( U_n \) is either 0 or 1 (fixed value), the variance \( \hat{V}_{\hat{X}_k} \) is always zero. Thus, the equations simplify to:

\[ \hat{W}_{c_k}^i = m_{\hat{X}_k} m_{\hat{X}_k}^T \] (8.8)
\[ \hat{W}_{c_k}^i \hat{m}_{c_k}^i = m_{\hat{X}_k} y_k^i \] (8.9)

We distinguish two cases:

- If \( U_n = 0 \): \( m_{\hat{X}_n} = 0 \), which means there is no change in \( C \).
- If \( U_n = 1 \): All values of \( m_{\hat{X}_k} \) need to be recomputed (for \( k = 1, \ldots, n \)).

The pseudo-code is given in Algorithm 8.2.1.

**Algorithm 8.2.1 : EM for Sinusoidal Model**

- Fix the threshold \( th \), initialize \( C = \hat{C}_0 \) (non zero), \( \xi_0 = 0 \)
- For \( k = 1, \ldots, n \)
  - Compute \( \hat{m}_{U_k} \) with \( \hat{C}_{k-1} \)
  - If \( \hat{m}_{U_k} < th \), \( \hat{C}_k = \hat{C}_{k-1} \)
  - Else:
    * For \( l = k, \ldots, 1 \): Compute \( \hat{W}_{c_l}^i \) and \( \hat{W}_{c_l}^i \hat{m}_{c_l}^i \) (\( \forall i \))
    * Update \( \hat{C}_k \) with Equation [8.5].

### 8.3 Neuron Model

If we consider \( A \) given, we could learn \( C \) with the same algorithm as for the second order case. This learned \( C \) would probably not have the form derived in Section 3.2.3 (first column all zero, no more than one input per row).

As for the supervised case, we use a more biologically plausible neuron, and derive a new algorithm for learning \( C \) and \( A \). We fix \( \lambda_0 \) and use
Unsupervised Learning

\[
\mathbf{A}^{-\top} = \begin{pmatrix}
\lambda_0 & 1 & 1 & \ldots & 1 \\
0 & \lambda_1 & 0 & \ldots & \\
\vdots & & & & \\
0 & 0 & & & \lambda_m \\
\end{pmatrix}
\]  
(8.10)

\[
\mathbf{C}^\top = \begin{pmatrix}
0 & 0 & 0 & \ldots & 0 \\
0 & c_1 & 0 & 0 & \ldots \\
\vdots & & & & \\
0 & & 0 & & c_m \\
\end{pmatrix}
\]  
(8.11)

We want to learn \(c_1, \ldots, c_m\) and \(\lambda_1, \ldots, \lambda_m\). Note that for this model, we need \(m = L\), otherwise not all input channels are considered.

### 8.3.1 Learning \(C\)

There is only one parameter to learn per row of \(C\). Therefore, the update rule is simpler: we learn a scalar instead of a complete vector.

The factor graph used to derive the update rules is given in Figure 8.4.

We can simplify the equations derived previously to:

\[
c_{\text{new}}^i = \left( \sum_{k=1}^{n} \gamma^{n-k} \hat{W}_{c_k^i} \right)^{-1} \left( \sum_{k=1}^{n} \gamma^{n-k} \tilde{W}_{c_k^i} \tilde{m}_{c_k^i} \right)
\]  
(8.12)

where again:

\[
\hat{W}_{c_k^i} = m_{\tilde{X}_k^i} m_{\tilde{X}_k^i}^\top
\]  
(8.13)

\[
\tilde{W}_{c_k^i} \tilde{m}_{c_k^i} = m_{\tilde{X}_k^i} y_k^i
\]  
(8.14)

Depending on the value of \(U\), we distinguish again two cases:

- \(m_{\tilde{X}_n^i} = 0\), which means there is no change in \(c\).
- All values of \(m_{\tilde{X}_k^i}\) need to be recomputed (for \(k = 1, \ldots, n\)).

### 8.3.2 Learning \(\lambda\)

It is not enough to learn only on \(C\), because we also need to have an influence on the delays (in \(A\)) so that they are adapted to the pattern we want to learn.
Figure 8.4: EM learning of \( C \) for neuron model
Learning $\mathbf{A}$ with EM is not straight-forward, because there is no input noise.

A simpler possibility could be to set the values of $\mathbf{A}$, because we have derived a closed form solution for the optimal $\lambda_i$ in Theorem 3.2.6. If at time index $n$, $\hat{m}_{U_n} > th$ (i.e. we update $c_i$), we could also update $\lambda_i$ with

$$\lambda_i^{new} = \lambda_i^{old} + \epsilon(\lambda_i - \lambda_i^{old}) \tag{8.15}$$

where $\lambda_i$ is given by (3.17) (for for $i = 1, \ldots, m$), and for some small value of $\epsilon$ ($\epsilon = 1$ would mean we replace the old value of $\lambda_i$ with the new one, but we only want to make one step in the right direction).

All steps are summed up in Algorithm 8.3.1.

**Algorithm 8.3.1 : EM for C and A with Neuron Model**

- Fix the threshold $th$, initialize $\mathbf{C} = \hat{\mathbf{C}}_0$ (non zero), $\hat{\xi}_0 = 0$
- For $k = 1, \ldots, n$
  - Compute $\hat{m}_{U_k}$ with $\hat{\mathbf{C}}_{k-1}$
  - If $\hat{m}_{U_k} < th$, $\hat{\mathbf{C}}_k = \hat{\mathbf{C}}_{k-1}$
  - Else: For $l = k, \ldots, 1$:
    - Compute $\hat{\mathbf{W}}_{c_i}$ and $\hat{\mathbf{W}}_{c_i}$, $\hat{m}_{c_i}$ ($\forall i$)
    - Store the positions $\ell_i$ of the last pulses before $k$ ($\forall i$)
  - Update $\hat{\mathbf{C}}_k$ according to (8.12), and $\hat{\mathbf{A}}_k$ according to (8.15).

**8.4 Conclusion**

We have shown that unsupervised learning seems possible, both for the sinusoidal and for the neuron model. We derived learning rules, but had to be creative for the learning of $\mathbf{A}$ in the neuron model.

The algorithms proposed in this chapter are only propositions, simulations are on-going. If they show good results, it could be interesting to adapt the algorithm for more than one layer.
Part III

Beyond Feedforward Networks
Chapter 9

Introducing Loops

9.1 Introduction

So far, we considered only feedforward networks. We have seen that it is possible to delay information over a short time in such networks. Adding layers is one option to achieve a form of multiscale pulse pattern analysis, but this strategy is not really adapted for long term memory.

Here we present an extension of the architecture, such that output connections are taken as inputs of the same layer. Such graphs have cycles, used for keeping information over a long time span.

As such, the architecture presented here belongs to the class of Recurrent Neural Network, which have been applied successfully in the last years in multiple fields, as for example language translation [53] and handwriting recognition [18]. For an extensive overview, see [33].

We present the general framework, and give details for the neuron model mainly. It turns out the idea is similar to the notion of a polychronous groups, introduced a few years ago in [26].

9.2 First Examples

We consider again the simple pattern of Figure 2.5, three pulses in three different channels. In this section, we present a simple strategy to repeat this pattern over a long time. We use the idea of a preferred pattern, introduced in Definition 3.1.2.
9.2.1 Neuron Model

Method

Suppose initial pulses in channels 1 and 3 are given (they are inside the red box in Figure 9.1a). We can build a filter which prefers this pattern over all other monomial patterns, and produces a pulse in channel 2 exactly 100 time steps after the pulse in channel 3 (see Figure 9.1a).

This is possible with (for example):

\[
A = \begin{pmatrix}
1/0.9988 & 0 & 0 \\
1.2 & 1/0.978 & 0 \\
-0.3 & 0 & 1/0.985
\end{pmatrix}
\] (9.1)

\[
s = (1 \ 0 \ 0)^T
\] (9.2)

and \(C_2\) found with (3.14). Numerically, in this example:

\[
C_2 = \begin{pmatrix}
0 & 0.01 & 0.09 \\
0 & 0 & 0 \\
0 & -0.05 & -0.07
\end{pmatrix}
\] (9.3)

The output of this filter is the dotted pulse in channel 2 in Figure 9.1a.

We have now three pulses (see Figure 9.1b), and can build a filter looking for the pulse pattern of channels 3 and 2. The filter should produce a pulse on channel 1, again exactly 100 time steps after the pulse in channel 2. With use \(A\) and \(s\) as defined above, and find \(C_1\) with (3.14):

\[
C_1 = \begin{pmatrix}
0 & 0 & 0 \\
0 & -0.05 & -0.07 \\
0 & 0.01 & 0.09
\end{pmatrix}
\] (9.4)

Finally, we apply the same technique to design a filter producing a pulse in channel 3 (see Figure 9.1c), using the same \(A\) and \(s\). We find:

\[
C_3 = \begin{pmatrix}
0 & -0.05 & -0.07 \\
0 & 0.01 & 0.09 \\
0 & 0 & 0
\end{pmatrix}
\] (9.5)

With an initialization in channels 1 and 3, the network, summarized in Figure 9.2 produces the pattern shown in Figure 9.1d.
Figure 9.1: Preferred patterns for each filter and total resulting pattern
Relationship to Polychronous Groups

In [26], Izhikevich introduced the notion of “polychronization”, i.e. a network in which “reproducible time-locked but not synchronous firing patterns with millisecond precision” appear. Certain patterns had finite durations, other were cyclic. The example we built in this section is a cyclic polychronous group.

9.2.2 Sinusoidal Model

The method works independently of the model chosen, as long as a closed form solution for $C$ can be derived. Thus, we could apply the technique described above to produce the pattern of Figure 9.1d with a network of second order filers. To find $C$, we would simply use Equation (3.3).
9.3 Potential Computational Power for Large Networks

9.3.1 Notation

In this section, we do not represent the height of a pulse (as they are all equal). For example, the cyclic pattern of Figure 9.1d, where neurons were firing in the order 1-3-2-1-3-2-... is represented in Figure 9.3.

To indicate that neuron 2 fires at time instant $k$ if a pulse in channel 1 is present at time instant $k - d_{12}$ and a pulse in channel 3 at time instant $k - d_{32}$, we write

$$ n_2[k] = n_3[k - d_{32}] \land n_1[k - d_{12}] $$

(9.6)

For simplicity, we use unit delays, so that for $d_{32} \leq d_{12}$, the equation above can be simplified to

$$ n_2[k] = n_3[k - 1] \land n_1[k - 2] $$

(9.7)

This equation means that neuron 2 is looking for a pulse in channel 1 followed by a pulse in channel 3.

Inhibition is represented with a bar: $n_2[k] = n_3[k - 1] \land \bar{n}_1[k - 2]$ would mean that neuron 2 fires only if a pulse is present in channel 1 but not in channel 3.

9.3.2 Sharing Neurons

The first example we derived may suggest that the network cannot store a large number of patterns: we built only one cyclic pattern with 3 neurons.
Introducing Loops

We augment the number of patterns by introducing sharing: a neuron can be involved in more than one pulse pattern. An example is given in Figure 9.4.

The equations for these patterns are:

\[
\begin{align*}
   n_1[k] &= n_3[k - 1] \lor n_2[k - 1] \\
   n_2[k] &= n_1[k - 1] \land n_3[k - 2] \\
   n_3[k] &= n_1[k - 1] \land n_2[k - 2]
\end{align*}
\] (9.8)

The threshold is 1, meaning a single neuron can initialize the appearance of a polychronous group.

- With the equations above, an initialization at neuron 3 gives a first cyclic polychronous group, 3-1-3-1-... (the black circles in Figure 9.4).

- With an initialization at 2 a second cyclic pattern appears (2-1-2-1-...), represented as blue crosses.

- With an initialization at neuron 1, we obtain a the polychronous group shown in Figure 9.5 which is not cyclic.

### 9.3.3 Increasing the Number of Neurons

We add a fourth neuron in the network. If we use following equations:

\[
\begin{align*}
   n_1[k] &= (n_3[k - 1] \lor n_2[k - 1]) \land n_4[k - 2] \\
   n_2[k] &= (n_1[k - 1] \lor n_4[k - 1]) \land n_3[k - 2] \\
   n_3[k] &= (n_1[k - 1] \lor n_4[k - 1]) \land n_2[k - 2] \\
   n_4[k] &= (n_3[k - 1] \lor n_2[k - 1]) \land n_1[k - 2]
\end{align*}
\] (9.9)

we obtain 4 cyclic patterns (see Figure 9.6).

In other words, in this example, we doubled the number of cyclic patterns by adding one shared neuron.

More generally, it has been shown that adding (shared) neurons can increase the storage capacity of the network. For example, in [26], an example with 5 neurons is given, where adding one neuron doubles the number of polychronous patterns. Simulations also showed that the
9.3 Potential Computational Power for Large Networks

(a) Initialization: Neuron 3

(b) Initialization: Neuron 2

Figure 9.4: Two cyclic patterns, sharing neuron 1

Figure 9.5: Non cyclic pattern
number of polychronous patterns increases exponentially with the number of inputs connections of the neurons [40]. These results focus on polychronous patterns in general, which include cyclic patterns as a special case.

Clearly, adding loops in the network is a simple way to retain information over long a long time. It is interesting that the principle of a preferred pattern can be used to build simple recurrent network. Learning such networks could be a promising direction for further research.
Chapter 10

Conclusion

We have mainly developed further the feedforward architecture, whose analysis started in [48].

We first defined suitable intermediate signals for the network, well-separated unit pulses which work robustly over many layers. We then precised the feature-detection filters: We defined the inner-product filter, looking for some pulse pattern in its input signal. The filters work by projecting the input into a one-dimensional subspace, and producing a pulse if the projection exceeds some threshold.

Finally, we showed how to design filters for detecting precise temporal pulse patterns, and applied the rules in building a first larger example, a network for parsing Morse code. The example came in two versions, one using a sinusoidal model and the other a biologically plausible neuron model. The two networks turned out to be quite different: Whereas the first used a rich bank of second order filters and only 4 layers, the second worked with a simple filter implementing a delay and a deep network (18 layers). Both proved the feasibility of the proposed methods for self-synchronizing signal analysis.

In the second part of the thesis, we developed algorithms for learning such networks.

The main contribution in supervised learning is an algorithm working with a new type of backpropagation, where starting on the highest layer, “good” pulse positions for the intermediate layers are defined. This algorithm has some similarities with existing work, but takes advantage
of the specificity of pulses and the position of the maximum in the impulse response. We developed the algorithm for two models again, the sinusoidal and the neuron model, and both showed good results.

We also touched upon the topic of unsupervised learning: We derived an algorithm for a single-layer architecture, but further work is needed to assess its performance.

The main drawback of the feedforward architecture is that information can only be retained over short time scales. In the final part, we showed how the architecture could be adapted to store information over longer time spans, by introducing loops in the network. The basic idea of preferred pulse patterns can be reused in an original way, and the implementation with the neuron model bears similarities with existing work.

We mainly proved that pattern recognition with a hierarchical architecture, as introduced in [48], is possible. While there is certainly room for improvement, the self-synchronizing network presented in this thesis works robustly with pulses, and first results have been shown in learning such a network, thus achieving a first step in the direction of structured, methodical multiscale signal analysis.
Appendix A

A State-Space Model of a Neuron

A.1 A Model with Dendrites

We consider $m$ dendritic trees, on each tree $i$ there are $n_i$ synapses (such that $L = \sum_i n_i$). We derive a state-space model for the electrical circuit of Figure A.1.

For $V_S$, we have:

$$\sum_{j=1}^{m} \frac{V_j(t) - V_s(t)}{R_j} = \frac{V_S(t)}{R_S} + C_S \frac{dV_S(t)}{dt} \quad (A.1)$$

In discrete-time, this can be written as:

$$\sum_{j=1}^{m} \frac{V_j[k] - V_s[k]}{R_j} = \frac{V_S[k]}{R_S} + C_S \frac{V_S[k+\Delta] - V_S[k]}{\Delta} \quad (A.2)$$

Thus:

$$V_S[k+\Delta] = \left(1 - \frac{\Delta}{C_S} \left(\frac{1}{R_S} + \sum_{j=1}^{m} \frac{1}{R_j}\right)\right) V_S[k] + \sum_{j=1}^{m} \frac{\Delta}{R_j C_S} V_j[k] \quad (A.3)$$

$$= \tilde{\lambda}_0 V_S[k] + \sum_{j=1}^{m} \tilde{\alpha}_j V_j[k] \quad (A.4)$$
Figure A.1: LIF with synaptic inputs on $m$ dendrites
For the potential in dendrite $i$, with an input pulse coming on synapse $j$ (the synapse strength is $\beta_j$) at time instant $k_j$, we have:

$$C_i \frac{dV_i}{dt} + \frac{V_i(t)}{R_i} = \sum_{j=1}^{n_i} \beta_j \delta(t - k_j)$$ (A.5)

We rewrite this equation in discrete-time:

$$C_i \frac{V_i[k + \Delta] - V_i[k]}{\Delta} + \frac{V_i[k]}{R_i} = \sum_{j=1}^{n_i} \beta_j \delta[k - k_j]$$ (A.6)

Thus:

$$V_i[k + \Delta] = \left(1 - \frac{\Delta}{R_i C_i} \right) V_i[k] + \sum_{j=1}^{n_i} \frac{\Delta}{C_i} \beta_j \delta[k - k_j]$$ (A.7)

$$\begin{bmatrix} V_S[k + 1] \\ V_1[k + 1] \\ \vdots \\ V_m[k + 1] \end{bmatrix} = \begin{bmatrix} \tilde{\lambda}_0 & \tilde{\alpha}_1 & \tilde{\alpha}_2 & \cdots & \tilde{\alpha}_m \\ 0 & \tilde{\lambda}_1 & 0 & \cdots & 0 \\ \vdots & \ddots & \ddots & \ddots & \vdots \\ 0 & \cdots & \cdots & 0 & \tilde{\lambda}_m \end{bmatrix} \begin{bmatrix} V_S[k] \\ V_1[k] \\ \vdots \\ V_m[k] \end{bmatrix} + C^T \begin{bmatrix} \delta[k - k_{11}] \\ \delta[k - k_{12}] \\ \vdots \\ \delta[k - k_{1n_1}] \\ \delta[k - k_{21}] \\ \vdots \\ \delta[k - k_{2n_2}] \\ \vdots \\ \delta[k - k_{mn_m}] \end{bmatrix}$$

with

$$C^T = \begin{bmatrix} 0 & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots & 0 \\ c_{11} & \cdots & c_{1n_1} & 0 & \cdots & 0 & 0 & \cdots & 0 \\ 0 & \cdots & 0 & c_{21} & \cdots & c_{2n_2} & 0 & \cdots & 0 \\ \vdots & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots & \vdots \\ 0 & \cdots & 0 & 0 & \cdots & 0 & c_{m1} & \cdots & c_{mn_m} \end{bmatrix}$$ (A.8)
Figure A.2:  LIF with $L$ synaptic inputs
A.2 LIF with Exponential Input Currents

We consider $L$ synaptic inputs. The complete circuit is shown in Figure A.2.

On synapse $\ell$, an input pulse of amplitude $\beta_\ell$ arrives at time instant $k_\ell$. This gives:

$$C_\ell \frac{dV_\ell}{dt} + \frac{V_\ell}{r_\ell} = \beta_\ell \delta[t - k_\ell] \quad \text{(A.9)}$$

From there we obtain:

$$V_\ell[k + \Delta] = \left(1 - \frac{\Delta}{r_\ell C_\ell} \right) V_\ell[k] + \frac{\Delta \beta_\ell}{C_\ell} \delta[k - k_\ell] \quad \text{(A.10)}$$

The total input current arriving at the soma is $\sum_{\ell=1}^{L} I_\ell$:

$$C_S \frac{dV_S}{dt} + \frac{V_S}{R_S} = \sum_{\ell=1}^{L} I_\ell \quad \text{(A.11)}$$

From Figure A.3, we know that $I_\ell = \frac{V_\ell}{R_\ell}$:

$$C_S \frac{dV_S}{dt} + \frac{V_S}{R_S} = \sum_{\ell=1}^{L} \frac{V_\ell}{R_\ell} \quad \text{(A.12)}$$

As before, we write this equation in discrete-time and obtain:
\[ V_S[k + \Delta] = \left(1 - \frac{\Delta}{C_S R_S} \right) V_S[k] + \sum_{\ell=1}^{L} \frac{\Delta}{R_\ell C_S} V_\ell \]  

(A.13)

Summing up:

\[
\begin{pmatrix}
V_S[k + 1] \\
V_1[k + 1] \\
\vdots \\
V_L[k + 1]
\end{pmatrix} =
\begin{pmatrix}
\tilde{\lambda}_0 & \tilde{\alpha}_1 & \tilde{\alpha}_2 & \cdots & \tilde{\alpha}_L \\
0 & \tilde{\lambda}_1 & 0 & \cdots & 0 \\
\vdots & \ddots & \ddots & \ddots & \vdots \\
0 & \cdots & \cdots & 0 & \tilde{\lambda}_L
\end{pmatrix}
\begin{pmatrix}
V_S[k] \\
V_1[k] \\
\vdots \\
V_L[k]
\end{pmatrix} + C^T \begin{pmatrix}
\delta[k - k_1] \\
\delta[k - k_2] \\
\vdots \\
\delta[k - k_L]
\end{pmatrix}
\]

with

\[
C^T =
\begin{pmatrix}
0 & \cdots & \cdots & 0 \\
c_1 & 0 & \cdots & 0 \\
0 & c_2 & 0 & \cdots & 0 \\
\vdots & \ddots & \ddots & \ddots & \vdots \\
0 & \cdots & 0 & c_L
\end{pmatrix}
\]

(A.14)

Note that in this case, the order \((m)\) is \(L + 1\).
Appendix B

Proofs for Chapter 3

B.1 Sinusoidal Model

We want to find $C$ and $s$ such that $s^T \xi_n$ is maximal at point $n$ for the given input pulse pattern, and such that no other monomial pulse pattern can have a higher score.

We write

$$C = \begin{pmatrix} c_1 \\ \vdots \\ c_L \end{pmatrix} \in \mathbb{R}^{L \times m}$$

$$C^T = (c_1^T \cdots c_L^T)$$

where $c_l = (c_{l,1} c_{l,2} \cdots c_{l,m}) \in \mathbb{R}^{1 \times m}$ with $m = 2M$, and choose

$$s = (1, 0, 1, 0, \ldots, 1, 0)^T \in \mathbb{R}^m \quad (B.1)$$

We start by rewriting the score function:

$$s^T \xi_n = s^T \sum_{k=1}^{n} (A^{k-n})^T C^T y_k \quad (B.2)$$

$$= s^T \sum_{k=1}^{n} \sum_{l=1}^{L} (A^{k-n})^T c_l^T y_k^l \quad (B.3)$$
\[ s^T \sum_{l=1}^{L} (A_{k_l} - n)^T c^T_l \]  
\[ = s^T \sum_{l=1}^{L} (A^T)_{k_l} - n c^T_l \]  

(B.4)

(B.5)

In the last line, we have used the monomial property of the input signal:

\[ y^l_k = \begin{cases} 
1 & \text{if } k = k_l \\
0 & \text{else} 
\end{cases} \]  

(B.6)

For matrix \( A \) given as in the theorem

\[ A = \begin{pmatrix} 
J_1 & 0 & 0 & \ldots & 0 \\
0 & J_2 & 0 & \ldots & 0 \\
\vdots & \vdots & \ddots & \ddots & \vdots \\
0 & 0 & \ldots & 0 & J_M 
\end{pmatrix}, \]  

(B.7)

where \( J_j \triangleq \lambda_j \text{rotm}(\Omega_j) \) with \( \lambda_j \geq 1, j = 1, \ldots, M \), and

\[ \text{rotm}(\Omega) \triangleq \begin{pmatrix} 
\cos(\Omega) & -\sin(\Omega) \\
\sin(\Omega) & \cos(\Omega) 
\end{pmatrix}. \]  

(B.8)

we know that for \( n \in \mathbb{Z} \):

\[ A^n = \begin{pmatrix} 
J^n_1 & 0 & 0 & \ldots & 0 \\
0 & J^n_2 & 0 & \ldots & 0 \\
\vdots & \vdots & \ddots & \ddots & \vdots \\
0 & 0 & \ldots & 0 & J^n_M 
\end{pmatrix}, \]  

(B.9)

where \( J^n_j = \lambda^n_j \text{rotm}(n\Omega_j) \).

Therefore:

\[ s^T \xi_n = \sum_{l=1}^{L} \sum_{j=1}^{M} \lambda^{k_l - n}_j (c_{l,2j-1} \cos((k_l - n)\Omega_j) + c_{l,2j} \sin((k_l - n)\Omega_j)) \]  

(B.10)
We proceed as follows: We set the (global) maximum of each channel to be 1 at time instant $n$. To guarantee that no other monomial pulse pattern has a higher score, we need the weighting signal in each channel to be highest at $k_\ell$, and this maximum should be the first one before $n$ (as shown in Figure 3.1a), for each frequency taken separately. The frequencies in $\mathbf{A}$ therefore need to satisfy a constraint: all input pulses need to be strictly inside one period of each frequency. To ensure that, we call $T_{\Omega_j}$ the time difference between the first maximum and the second zero for a given frequency $\Omega_j$ (see Figure B.1). We want all input pulses inside this window, i.e.

$$\max_\ell(n - k_\ell) < \min_j T_{\Omega_j} \quad (B.11)$$

We now compute $T_{\Omega_j}$ for the function $f(k) = \lambda^k \cos(k\Omega)$.

$$f'(k) = \ln(\lambda)\lambda^k \cos(k\Omega) - \lambda^k \Omega \sin(k\Omega) \quad (B.12)$$

$$f'(k) = 0 \iff \tan(k\Omega) = \frac{\ln \lambda}{\Omega} \quad (B.13)$$

$$\iff k\Omega = \arctan\left(\frac{\ln \lambda}{\Omega}\right) + k' \pi, \quad k' \in \mathbb{Z} \quad (B.14)$$

$$\iff k = \frac{1}{\Omega} \arctan\left(\frac{\ln \lambda}{\Omega}\right) + k' \frac{\pi}{\Omega}, \quad k' \in \mathbb{Z} \quad (B.15)$$
The first extremum corresponds to $k' = 0$: $k = \frac{1}{\Omega} \arctan \left( \frac{\ln \lambda}{\Omega} \right)$. The positions of the zeros of $f$ are unchanged by the factor $\lambda^k$, which means they are at multiples of $\frac{\pi/2}{\Omega}$. The second zero is at $\frac{3\pi/2}{\Omega}$.

Therefore, we deduce:

$$T_\Omega = \frac{3\pi/2 - \arctan \left( \frac{\ln \lambda}{\Omega} \right)}{\Omega} \quad \text{(B.16)}$$

**B.1.1 Version I**

If all frequencies fulfill the condition stated above, we can compute an optimal $C$ using all frequencies. We define one component of the score:

$$f_{jl}[k] = \lambda_j^{k_{l+1} - k} (c_{l,2j_1} \cos((k_{\ell+1} - k)\Omega_j) + c_{l,2j} \sin((k_{\ell+1} - k)\Omega_j)) \quad \text{(B.17)}$$

The two conditions

$$f'_{jl}[k] = 0 \quad \text{(B.18)}$$
$$f_{jl}[k] = 1 \quad \text{(B.19)}$$

give two equations in $c_{l,2j-1}$ and $c_{l,2j}$ which we can solve to find:

$$c_{l,2j-1} = \lambda_j^{n-k_{\ell+1}} \left( \cos((n - k_{\ell+1})\Omega_j) - \frac{\ln \lambda_j}{\Omega_j} \sin((n - k_{\ell+1})\Omega_j) \right) \quad \text{(B.20)}$$
$$c_{l,2j} = \lambda_j^{n-k_{\ell+1}} \left( -\frac{\ln \lambda_j}{\Omega_j} \cos((n - k_{\ell+1})\Omega_j) - \sin((n - k_{\ell+1})\Omega_j) \right) \quad \text{(B.21)}$$

This can be rewritten as in the theorem, where the second row of $R$ is chosen such that $R$ is a rotation matrix (this row can be chosen arbitrarily, because of the zeros in $s$).

The value of the maximum is $Lm/2$, because all $L$ channels are used and there are $m/2$ components in the rotation matrix.

**B.1.2 Version II**

If only one frequency fulfills the condition, we can use this frequency and any other one to compute the optimal $C$. 
The computations are similar to version I, with a simpler expression of the score signal (it has only two components instead of \( M \)). For the component satisfying the constraint, we set the global maximum of the weighting signal as before at \( k_\ell \). For the other component, we set a local maximum at \( k_\ell \). As these components are summed in the score function, we still have a global maximum at \( k_\ell \) (as can be seen in Figure 3.1b), meaning the score signal will be maximal at \( n \).

The value of the maximum of the score signal is \( 2L \) (we use only 2 frequencies instead of \( m/2 \)).

\section{B.2 Exponential Model}

We want to find \( C \) and \( s \) such that \( s^T \xi_n \) is maximal at point \( n \), with maximum \( L \) for the given input pulse pattern, and such that no other monomial pulse pattern can have a higher score.

\[
C = \begin{pmatrix} c_1 \\ \vdots \\ c_L \end{pmatrix} \in \mathbb{R}^{L \times m}
\]

\[
C^T = (c_1^T \cdots c_L^T)
\]

where \( c_l = (c_{l,1} \cdots c_{l,m}) \in \mathbb{R}^{1 \times m} \)

We start by rewriting the score function:

\[
s^T \xi_n = s^T \sum_{k=1}^{n} (A^{k-n})^T C^T y_k
\]

\[
= s^T \sum_{k=1}^{n} \sum_{l=1}^{L} (A^{k-n})^T c_l^T y_k^l
\]

\[
= s^T \sum_{l=1}^{L} (A^{k_l-n})^T c_l^T
\]

In the last line, we have used the monomial property:

\[
y_k^l = \begin{cases} 
1 & \text{if } k = k_l \\
0 & \text{else}
\end{cases} \tag{B.22}
\]

Now we have a closer look at this expression. We know that \( A = \text{diag}(\lambda_1, \ldots, \lambda_m) \), \( \lambda_i > 1 \) and we choose \( s = (1 \cdots 1) \).
In that case the score function can be written as:

\[ s^T \xi_n = \sum_{l=1}^{L} \sum_{j=1}^{m} c_{l,j} \lambda_j^{k_l-n} \]

To have a maximum at time index \( n \) that is equal to \( L \), a simple solution consists in taking each channel separately: for each channel \( l = 1, \ldots, L \), the maximum should be at \( t = n \) and equal to 1. This means, \( \forall l = 1, \ldots, L \), we want to find:

\[
\begin{align*}
\text{argmax}_i f_l[i] &= n \\
\max_i f_l[i] &= 1
\end{align*}
\]

where \( f_l[i] = \sum_{j=1}^{m} c_{l,j} \lambda_j^{k_l-i} \).

We distinguish two cases:

- If \( k_l = n \), we take \( c_{l,j} = 1 \) for one \( l = 1, \ldots, L \), all others being 0 (the position does not matter)

- If \( k_l \neq n \), taking \( c_{l,j} = 1 \) for one \( l = 1, \ldots, L \) will result in a maximum at \( t = k_l \), which is not what we want. Taking two non-zero coefficients will give us a system with 2 equations and 2 unknowns. W.l.g we take \( c_{l,1} \neq 0 \) and \( c_{l,2} \neq 0 \), all other being 0, which gives us following system of equations :

\[
\begin{align*}
\frac{f_l[i]}{1} &= 1 \iff c_{l,1} \lambda_1^{k_l-i} + c_{l,2} \lambda_2^{k_l-i} = 1 \\
\frac{f'_l[i]}{0} &= 0 \iff c_{l,1} (-\ln \lambda_1) \lambda_1^{k_l-i} + c_{l,2} (-\ln \lambda_2) \lambda_2^{k_l-i} = 1
\end{align*}
\]

Solving this system gives:

\[
\begin{align*}
c_{l,1} &= \frac{\ln \lambda_2}{\ln \lambda_2 - \ln \lambda_1} \lambda_1^{n-k_l} \\
c_{l,2} &= \frac{\ln \lambda_1}{\ln \lambda_2 - \ln \lambda_1} \lambda_2^{n-k_l}
\end{align*}
\]

We can check that the solution found is indeed a maximum by computing \( f''_l[n] \):

\[ f''_l[n] = -\ln \lambda_2 \ln \lambda_1 \]

As all \( \lambda_i > 1 \), we have \( \ln \lambda_i > 0 \), which means the above expression is negative. Thus we have indeed found a maximum of our function.
B.3 Neuron Model

We have:

$$\mathbf{C} = \begin{pmatrix} c_1 \\ \vdots \\ c_L \end{pmatrix} \in \mathbb{R}^{L \times (m+1)}$$

$$\mathbf{C}^T = (c_1^T \ldots c_L^T)$$

where $$\mathbf{c}_l = (c_{l,0} \ldots c_{l,m}) \in \mathbb{R}^{1 \times (m+1)}$$, and

$$\mathbf{s} = (1 \ 0 \ 0 \ldots 0)^T \in \mathbb{R}^{m+1} \quad \text{(B.23)}$$

We start by rewriting the score function:

$$s^T \xi_n = s^T \sum_{k=1}^n (\mathbf{A}^{k-n})^T \mathbf{C}^T y_k$$  \quad \text{(B.24)}$$

$$= s^T \sum_{k=1}^n \sum_{l=1}^L (\mathbf{A}^{k-n})^T \mathbf{c}_l^T y_k$$  \quad \text{(B.25)}$$

$$= s^T \sum_{l=1}^L (\mathbf{A}^{k_l-n})^T \mathbf{c}_l^T \quad \text{(B.26)}$$

$$= s^T \sum_{l=1}^L (\mathbf{A}^{T})^{k_l-n} \mathbf{c}_l^T \quad \text{(B.27)}$$

(From B.25 to B.26, we have used the monomial property of the input signal).

For matrix $$\mathbf{A}$$ given as in the theorem ($$\lambda_i > 1$$):

$$\mathbf{A}^T = \begin{pmatrix} \lambda_0 & \alpha_1 & \alpha_2 & \ldots & \alpha_m \\ 0 & \lambda_1 & 0 & \ldots & 0 \\ \vdots & \vdots & \ddots & \ddots & \vdots \\ \vdots & \vdots & \ddots & 0 \\ 0 & \ldots & \ldots & 0 & \lambda_m \end{pmatrix} \quad \text{(B.28)}$$

we can compute, for $$n \in \mathbb{Z}$$:
\[(A^T)^n = \begin{pmatrix}
\lambda_0^n & \tilde{\alpha}_{1,n} & \tilde{\alpha}_{2,n} & \cdots & \tilde{\alpha}_{m,n} \\
0 & \lambda_1^n & 0 & \cdots & 0 \\
\vdots & \ddots & \ddots & \ddots & \vdots \\
\vdots & \ddots & \ddots & 0 & 0 \\
0 & \cdots & \cdots & 0 & \lambda_m^n
\end{pmatrix} \quad (B.29)\]

with \(\forall i \in 1, \ldots, m\)

\[\tilde{\alpha}_{i,n} = \alpha_i \sum_{k=0}^{n-1} \lambda_0^k \lambda_i^{n-1-k} = \alpha_i \frac{\lambda_i^n - \lambda_0^n}{\lambda_i - \lambda_0} \quad (B.30)\]

We now can continue after Equation \(B.27\):

\[s^T \xi_n = \sum_{l=1}^{L} (\lambda_0^{k_\ell-n} \tilde{\alpha}_{1,k_\ell-n} \tilde{\alpha}_{2,k_\ell-n} \cdots \tilde{\alpha}_{m,k_\ell-n}) c_\ell^T \quad (B.32)\]

\[= \sum_{l=1}^{L} \left( \lambda_0^{k_\ell-n} c_{l,1} + \sum_{j=2}^{m} c_{l,j} \tilde{\alpha}_{j,k_\ell-n} \right) \quad (B.33)\]

To have a maximum at time index \(n\) that is equal to \(L\), we again consider each channel separately: for each \(l = 1, \ldots, L\), the maximum should be at \(t = n\) and equal to 1.

**B.3.1 Version I**

One solution could be to take \(c_\ell^T = (c_{\ell,0} 0 c_{\ell,i} 0 \cdots 0)^T\), for \(i \in \{1, \ldots, m\}\). The position of \(i\) is arbitrary. For each channel, we have to maximize:

\[\lambda_0^{k_\ell-n} c_{\ell,0} + c_{\ell,i} \alpha_i \frac{\lambda_i^{n-k_\ell} - \lambda_0^{n-k_\ell}}{\lambda_i - \lambda_0} \quad (B.34)\]

The maximum should be one and the derivative zero:

- if \(k_\ell = n\), \(c_\ell = (1 0 \cdots 0)\)
B.3 Neuron Model

• else: \( c_\ell \) has exactly two nonzero entries at position 0 and \( i \) (\( i \geq 1 \) arbitrary)

\[
c_{\ell,i} = \frac{\lambda_i - \lambda_0}{\alpha_i \frac{1}{1 - \ln \lambda_i / \ln \lambda_0}} \lambda_i^{n-k_\ell} \tag{B.35}
\]

\[
c_{\ell,0} = \frac{\ln \lambda_0 \lambda_i^{n-k_\ell} - \ln \lambda_i \lambda_0^{n-k_\ell}}{\ln \lambda_0 - \ln \lambda_i} \tag{B.36}
\]

B.3.2 Version II

If we want to force \( c_{\ell,0} = 0 \) for all \( \ell \), we need \( n - k_\ell > 0 \). Then we can choose \( c_\ell \) to have exactly two nonzero entries at position \( i \) and \( j \) (\( i,j \geq 1 \) arbitrary). In this case:

\[
s^T \xi_n = \sum_{\ell} c_{\ell,i} \tilde{\alpha}_{i,k_\ell-n} + c_{\ell,j} \tilde{\alpha}_{j,k_\ell-n} \tag{B.37}
\]

We again take each channel separately: it should be maximal at \( n \), with value 1. This gives:

\[
c_{\ell,j} = \frac{\lambda_j - \lambda_0 \ln \lambda_0 \lambda_i^{n-k_\ell} - \ln \lambda_i \lambda_0^{n-k_\ell}}{\alpha_j f(\lambda_j, \lambda_i, \lambda_0)} \lambda_j^{n-k_\ell} \tag{B.38}
\]

with

\[
f(\lambda_j, \lambda_i, \lambda_0) = (\lambda_j^{n-k_\ell} - \lambda_0^{n-k_\ell}) \ln \lambda_i - (\lambda_i^{n-k_\ell} - \lambda_0^{n-k_\ell}) \ln \lambda_j + (\lambda_i^{n-k_\ell} - \lambda_j^{n-k_\ell}) \ln \lambda_0
\]

B.3.3 Version III

Another possibility if the first column of \( C \) is zero (and \( n - k_\ell > 0 \)) is to take only one component in each row of \( C \):

\[
C = \begin{pmatrix}
0 & c_1 & 0 & \cdots & \cdots & 0 \\
0 & 0 & c_2 & 0 & \cdots & 0 \\
\vdots & & \ddots & \ddots & \vdots & \vdots \\
0 & \cdots & 0 & 0 & \cdots & 0
\end{pmatrix} \tag{B.39}
\]
Now we compute optimal values for both \(\lambda_j\) and \(c_j\):

\[
\mathbf{s}^T \xi_n = \sum_{\ell} c_{\ell} \tilde{\alpha}_{\ell, n - k_\ell}
\]

\(= \sum_{\ell} c_{\ell} \alpha_{\ell} \frac{\lambda_{\ell}^{n - k_\ell} - \lambda_0^{n - k_\ell}}{\lambda_\ell - \lambda_0}
\]

We want the score to be one in each channel at time instant \(n\), therefore:

\[
c_{\ell} = \frac{\lambda_\ell - \lambda_0}{\alpha_{\ell} (\lambda_\ell^{k_\ell - n} - \lambda_0^{k_\ell - n})}
\]

Taking the derivative of the score in one channel w.r.t. \(n\) to be 0 gives:

\[
c_{\ell} \alpha_{\ell} \frac{(\ln \lambda_\ell) \lambda_\ell^{n - k_\ell} - (\ln \lambda_0) \lambda_0^{n - k_\ell}}{\lambda_\ell - \lambda_0} = 0
\]

We define \(x = (n - k_\ell) \ln \lambda_\ell\), i.e. \(\lambda_\ell = \exp(x/(n - k_\ell))\). \(x\) is the solution to:

\[
x \exp(x) = (n - k_\ell) \ln(\lambda_0) \exp((n - k_\ell) \ln(\lambda_0))
\]

The right hand side of this equation is known, therefore \(x\) can be written as a Lambert-W-function:

\[
x = W((n - k_\ell) \ln(\lambda_0) \exp((n - k_\ell) \ln(\lambda_0)))
\]

The obvious solution \(\lambda_0 = \lambda_i\) has to be discarded, otherwise the score is zero everywhere.
Appendix C

Networks for Parsing Morse Code

C.1 About Morse Code

C.1.1 International Morse Code

We used only the letters (no numbers) of the International Morse code [56], as shown in Table C.1.

C.1.2 Speed of Transmission and Dit Length

The usual way to state the transmission speed of Morse code is in Words Per Minute (WPM).

One standard word used to compute the dit length is “PARIS” followed by an inter-word gap [56]. The total length of this word (see Figure C.1) is 50 elements. We call the length of one element \(d\) (i.e. the length of a dit), 1 WPM corresponds to 50\(d\) per minute. Thus

\[
d = \frac{60}{50} = 1.2s
\]

Obviously, 1 WPM is quite slow. Our data was recorded at 13 WPM, which is a common value.

For 13 WPM, the length of a dit is \(1.2/13 \approx 92.3\text{ms}\).
Table C.1: Letters of International Morse Code

<table>
<thead>
<tr>
<th>Letter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td></td>
</tr>
<tr>
<td>B</td>
<td></td>
</tr>
<tr>
<td>C</td>
<td></td>
</tr>
<tr>
<td>D</td>
<td></td>
</tr>
<tr>
<td>E</td>
<td></td>
</tr>
<tr>
<td>F</td>
<td></td>
</tr>
<tr>
<td>G</td>
<td></td>
</tr>
<tr>
<td>H</td>
<td></td>
</tr>
<tr>
<td>I</td>
<td></td>
</tr>
<tr>
<td>J</td>
<td></td>
</tr>
<tr>
<td>K</td>
<td></td>
</tr>
<tr>
<td>L</td>
<td></td>
</tr>
<tr>
<td>M</td>
<td></td>
</tr>
<tr>
<td>N</td>
<td></td>
</tr>
<tr>
<td>O</td>
<td></td>
</tr>
<tr>
<td>P</td>
<td></td>
</tr>
<tr>
<td>Q</td>
<td></td>
</tr>
<tr>
<td>R</td>
<td></td>
</tr>
<tr>
<td>S</td>
<td></td>
</tr>
<tr>
<td>T</td>
<td></td>
</tr>
<tr>
<td>U</td>
<td></td>
</tr>
<tr>
<td>V</td>
<td></td>
</tr>
<tr>
<td>W</td>
<td></td>
</tr>
<tr>
<td>X</td>
<td></td>
</tr>
<tr>
<td>Y</td>
<td></td>
</tr>
<tr>
<td>Z</td>
<td></td>
</tr>
</tbody>
</table>

Figure C.1: Standard word “PARIS ” with element count
C.2 Datasets

All files were recorded at a speed of 13 WPM. We used an Ipad and the application “Morse-it” [5] to produce the encoded signals.

C.2.1 Training Set

We record the alphabet four times in total. The letters are grouped together to form small “words”:

- ABCDEDCBAE
- FGHIJHGFJ
- KLMNOONMLK
- PQRSTSTRQP
- UVWXYZVXZYWU

Each word is recorded twice. The words have approximately the same length (between 10 and 15s). Some noise is present due to the recording. An example of training sample is given in Figure C.2.

C.2.2 Test Set

We used single words (the same as those for training, but recorded separately), as well as other words which were never presented to the network before. To test the functioning of pauses, we also recorded short
<table>
<thead>
<tr>
<th>Type</th>
<th>Description</th>
<th>Note</th>
</tr>
</thead>
<tbody>
<tr>
<td>Single words</td>
<td>• ABCDEDCBAE</td>
<td>3 recordings</td>
</tr>
<tr>
<td></td>
<td>• FGIJIHGFJ</td>
<td></td>
</tr>
<tr>
<td></td>
<td>• KLMNOONMLK</td>
<td></td>
</tr>
<tr>
<td></td>
<td>• PQRSTSTRQP</td>
<td></td>
</tr>
<tr>
<td></td>
<td>• UVWXYZVXYZYWU</td>
<td></td>
</tr>
<tr>
<td></td>
<td>• ABRICOT</td>
<td></td>
</tr>
<tr>
<td></td>
<td>• APPLETREE</td>
<td></td>
</tr>
<tr>
<td></td>
<td>• VISION</td>
<td></td>
</tr>
<tr>
<td>Short sequences</td>
<td>• HELLO WORLD</td>
<td>3 recordings</td>
</tr>
<tr>
<td></td>
<td>• HELLO AGAIN</td>
<td>3 recordings</td>
</tr>
<tr>
<td></td>
<td>• THIS IS A TEST</td>
<td></td>
</tr>
<tr>
<td></td>
<td>• ANOTHER TEST</td>
<td></td>
</tr>
<tr>
<td></td>
<td>• ULTRA WIDE BAND</td>
<td></td>
</tr>
<tr>
<td></td>
<td>• DIFFERENT SPEEDS</td>
<td></td>
</tr>
<tr>
<td></td>
<td>• A BIG WHITE CAT</td>
<td></td>
</tr>
<tr>
<td>Long sequences</td>
<td>• BRUSHY MORSE ALPHABET SERVILENESS DECEPTIVE SKIMMER</td>
<td></td>
</tr>
<tr>
<td></td>
<td>• ENTERING HONEY BAG GITH DIFFERENT EUPHONICON</td>
<td></td>
</tr>
<tr>
<td></td>
<td>• DUPLICABILITY RENEGADE OELIGNANTHYLIDENE ENTRAILS PYROGALLIC</td>
<td></td>
</tr>
</tbody>
</table>

Table C.2: Words and expressions in the test set

sequences. Finally, to be sure that the network does not accumulate some errors, we also tested longer sequences (most of the words were selected randomly). Sometimes we recorded a word or expression more than once.

Table C.2 summarizes the complete test set.

C.3 Sinusoidal Model

For all layers, $s = (1 0)$.

C.3.1 Layer 2

$C \in \mathbb{R}^{1 \times 2}$ can be written as
\[ C = \mu (\cos \alpha \sin \alpha) \quad (C.1) \]

The threshold is 0.9. The complete parameters are in Table C.3.

C.3.2 Layer 3

\( C \in \mathbb{R}^{9 \times 2} \) but there are only 2 rows of nonzero elements. We write those as

\[ c_i = \mu_i (\cos \alpha_i \sin \alpha_i) \quad (C.2) \]

We have 2 rows per filter, the ordering corresponds to the ordering in the column input. In all our examples, the two rows had the same weight. Therefore, we sum them up in \( \mu, \alpha_1, \alpha_2 \). The threshold is 1.9.

Example: Filter C1 has \( (\mu, \alpha_1, \alpha_2) = (1/1.03, \pi, 0) \) and takes B1 and B6 as inputs. This means

\[
\begin{pmatrix}
\frac{1}{1.03} \cos(\pi) & \frac{1}{1.03} \sin(\pi) \\
0 & 0 \\
0 & 0 \\
0 & 0 \\
0 & 0 \\
\end{pmatrix}
\]

\[
\begin{pmatrix}
\frac{1}{1.03} \cos(0) & \frac{1}{1.03} \sin(0) \\
0 & 0 \\
0 & 0 \\
0 & 0 \\
\end{pmatrix}
\]

\[
= \begin{pmatrix}
\frac{1}{1.03} & 0 \\
0 & 0 \\
0 & 0 \\
0 & 0 \\
\end{pmatrix}
\]

The complete parameters are in Table C.4.
### Table C.3: All Parameters for Second-Layer Feature-Detection Filters

<table>
<thead>
<tr>
<th>id</th>
<th>feature, letter</th>
<th>input</th>
<th>freq.</th>
<th>$\lambda^{-1}$</th>
<th>$\mu, \alpha$</th>
</tr>
</thead>
<tbody>
<tr>
<td>B1</td>
<td>pause – dit</td>
<td>A1</td>
<td>$\frac{1}{3}T$</td>
<td>0.9986</td>
<td>2.5, $\frac{7\pi}{6}$</td>
</tr>
<tr>
<td>B2</td>
<td>pause – dah</td>
<td>A1</td>
<td>$\frac{1}{7}T$</td>
<td>0.9984</td>
<td>$\frac{1}{1.17}, \frac{\pi}{4}$</td>
</tr>
<tr>
<td>B3</td>
<td>dit – pause</td>
<td>A1</td>
<td>$\frac{1}{9}T$</td>
<td>0.9937</td>
<td>0, $\frac{4\pi}{3}$</td>
</tr>
<tr>
<td>B4</td>
<td>dah – pause</td>
<td>A1</td>
<td>$\frac{1}{8}T$</td>
<td>0.9984</td>
<td>$\frac{1}{0.21}, \pi$</td>
</tr>
<tr>
<td>B5</td>
<td>dit</td>
<td>A1</td>
<td>$\frac{1}{2.5}T$</td>
<td>0.9956</td>
<td>20, $\frac{3\pi}{2}$</td>
</tr>
<tr>
<td>B6</td>
<td>dah</td>
<td>A1</td>
<td>$\frac{1}{6.5}T$</td>
<td>0.9969</td>
<td>$\frac{1}{0.7}, \frac{\pi}{3}$</td>
</tr>
<tr>
<td>B7</td>
<td>E</td>
<td>A1</td>
<td>$\frac{1}{4}T$</td>
<td>0.9984</td>
<td>$\frac{1}{0.23}, \frac{4\pi}{3}$</td>
</tr>
<tr>
<td>B8</td>
<td>T</td>
<td>A1</td>
<td>$\frac{1}{6}T$</td>
<td>0.999</td>
<td>2, $\pi$</td>
</tr>
<tr>
<td>B9</td>
<td>pause betw. words</td>
<td>A1</td>
<td>$\frac{1}{10}T$</td>
<td>0.9984</td>
<td>100, $\frac{3\pi}{2}$</td>
</tr>
</tbody>
</table>

### Table C.4: All Parameters for Third-Layer Feature-Detection Filters

<table>
<thead>
<tr>
<th>id</th>
<th>letter</th>
<th>feature</th>
<th>input</th>
<th>freq.</th>
<th>$\lambda^{-1}$</th>
<th>$\mu, \alpha_1, \alpha_2$</th>
</tr>
</thead>
<tbody>
<tr>
<td>C1</td>
<td></td>
<td></td>
<td>B1, B6</td>
<td>$\frac{1}{4}T$</td>
<td>0.9984</td>
<td>$\frac{1}{1.15}, \pi, 0$</td>
</tr>
<tr>
<td>C2</td>
<td></td>
<td></td>
<td>B1, B5</td>
<td>$\frac{1}{9}T$</td>
<td>0.9978</td>
<td>1, $\frac{\pi}{2}, \frac{\pi}{4}$</td>
</tr>
<tr>
<td>C3</td>
<td></td>
<td></td>
<td>B2, B5</td>
<td>$\frac{1}{8}T$</td>
<td>0.9984</td>
<td>$\frac{1}{0.6}, \pi, 0$</td>
</tr>
<tr>
<td>C4</td>
<td></td>
<td></td>
<td>B2, B6</td>
<td>$\frac{1}{18}T$</td>
<td>0.9989</td>
<td>1, $\frac{3\pi}{4}, \frac{\pi}{4}$</td>
</tr>
<tr>
<td>C5</td>
<td></td>
<td></td>
<td>B5, B4</td>
<td>$\frac{1}{12}T$</td>
<td>0.9989</td>
<td>$\frac{1}{1.06}, \pi, 0$</td>
</tr>
<tr>
<td>C6</td>
<td></td>
<td></td>
<td>B5, B3</td>
<td>$\frac{1}{2}T$</td>
<td>0.9989</td>
<td>$\frac{1}{2.475}, \pi, 0$</td>
</tr>
<tr>
<td>C7</td>
<td></td>
<td></td>
<td>B6, B3</td>
<td>$\frac{1}{21}T$</td>
<td>0.9984</td>
<td>$\frac{1}{0.53}, \pi, 0$</td>
</tr>
<tr>
<td>C8</td>
<td></td>
<td></td>
<td>B6, B4</td>
<td>$\frac{1}{4}T$</td>
<td>0.9991</td>
<td>$\frac{1}{1.323}, \frac{3\pi}{2}, \frac{\pi}{4}$</td>
</tr>
<tr>
<td>C9</td>
<td>A</td>
<td></td>
<td>B1, B4</td>
<td>$\frac{1}{12}T$</td>
<td>0.9989</td>
<td>$\frac{1}{1.09}, \pi, 0$</td>
</tr>
<tr>
<td>C10</td>
<td>I</td>
<td></td>
<td>B1, B3</td>
<td>$\frac{1}{9}T$</td>
<td>0.9989</td>
<td>$\frac{1}{1.1}, \pi, 0$</td>
</tr>
<tr>
<td>C11</td>
<td>N</td>
<td></td>
<td>B2, B3</td>
<td>$\frac{1}{9}T$</td>
<td>0.9989</td>
<td>$\frac{1}{1.09}, \pi, 0$</td>
</tr>
<tr>
<td>C12</td>
<td>M</td>
<td></td>
<td>B2, B4</td>
<td>$\frac{1}{15}T$</td>
<td>0.9989</td>
<td>$\frac{1}{1.05}, \pi, 0$</td>
</tr>
</tbody>
</table>
C.4 Neuron Model

Table C.5: All Parameters for Fourth-Layer Feature-Detection Filters

<table>
<thead>
<tr>
<th>id</th>
<th>letter</th>
<th>input</th>
<th>freq.</th>
<th>( \lambda^{-1} )</th>
<th>( \mu, \alpha_1, \alpha_2 )</th>
</tr>
</thead>
<tbody>
<tr>
<td>D1</td>
<td>U</td>
<td>C2, C5</td>
<td>(1/12T)</td>
<td>0.9995</td>
<td>4/5, (\pi, 0)</td>
</tr>
<tr>
<td>D2</td>
<td>S</td>
<td>C2, C6</td>
<td>(1/8T)</td>
<td>0.9992</td>
<td>4/5, (\pi, 0)</td>
</tr>
<tr>
<td>D3</td>
<td>R</td>
<td>C1, C7</td>
<td>(1/12T)</td>
<td>0.9995</td>
<td>4/5, (\pi, 0)</td>
</tr>
<tr>
<td>D4</td>
<td>W</td>
<td>C1, C8</td>
<td>(1/16T)</td>
<td>0.9995</td>
<td>1/1.05, (\pi, 0)</td>
</tr>
<tr>
<td>D5</td>
<td>K</td>
<td>C3, C5</td>
<td>(1/12T)</td>
<td>0.9995</td>
<td>4/5, (\pi, 0)</td>
</tr>
<tr>
<td>D6</td>
<td>D</td>
<td>C3, C6</td>
<td>(1/8T)</td>
<td>0.9992</td>
<td>4/5, (\pi, 0)</td>
</tr>
<tr>
<td>D7</td>
<td>G</td>
<td>C4, C7</td>
<td>(1/12T)</td>
<td>0.9995</td>
<td>1/1.15, (\pi, 0)</td>
</tr>
<tr>
<td>D8</td>
<td>O</td>
<td>C4, C8</td>
<td>(1/16T)</td>
<td>0.9995</td>
<td>1/1.1, (\pi, 0)</td>
</tr>
<tr>
<td>D9</td>
<td>Q</td>
<td>C4, C5</td>
<td>(1/18T)</td>
<td>0.9995</td>
<td>1/1.1, (\pi, 0)</td>
</tr>
<tr>
<td>D10</td>
<td>Z</td>
<td>C4, C6</td>
<td>(1/18T)</td>
<td>0.9995</td>
<td>1/1.15, (\pi, 0)</td>
</tr>
<tr>
<td>D11</td>
<td>L</td>
<td>C1, C6</td>
<td>(1/18T)</td>
<td>0.9995</td>
<td>1/1.1, (\pi, 0)</td>
</tr>
<tr>
<td>D12</td>
<td>P</td>
<td>C1, C7</td>
<td>(1/24T)</td>
<td>0.9995</td>
<td>1/1.1, (\pi, 0)</td>
</tr>
<tr>
<td>D13</td>
<td>V</td>
<td>C2, C5</td>
<td>(1/20T)</td>
<td>0.9992</td>
<td>1/1.1, (\pi, 0)</td>
</tr>
<tr>
<td>D14</td>
<td>H</td>
<td>C2, C6</td>
<td>(1/15T)</td>
<td>0.9992</td>
<td>1/1.2, (\pi, 0)</td>
</tr>
<tr>
<td>D15</td>
<td>F</td>
<td>C2, C7</td>
<td>(1/16T)</td>
<td>0.9992</td>
<td>1/1.05, (\pi, 0)</td>
</tr>
<tr>
<td>D16</td>
<td>X</td>
<td>C3, C5</td>
<td>(7/120T)</td>
<td>0.9995</td>
<td>1/1.13, (\pi, 0)</td>
</tr>
<tr>
<td>D17</td>
<td>B</td>
<td>C3, C6</td>
<td>(1/15T)</td>
<td>0.9995</td>
<td>1/1.2, (\pi, 0)</td>
</tr>
<tr>
<td>D18</td>
<td>C</td>
<td>C3, C7</td>
<td>(1/18T)</td>
<td>0.9995</td>
<td>1/1.1, (\pi, 0)</td>
</tr>
<tr>
<td>D19</td>
<td>J</td>
<td>C1, C8</td>
<td>(1/30T)</td>
<td>0.9997</td>
<td>1/1.2, (\pi, 0)</td>
</tr>
<tr>
<td>D20</td>
<td>Y</td>
<td>C3, C8</td>
<td>(7/120T)</td>
<td>0.9996</td>
<td>1/1.2, (\pi, 0)</td>
</tr>
</tbody>
</table>

C.3.3 Layer 4

\( \mathbf{C} \in \mathbb{R}^{12 \times 2} \): Again, we have only 2 rows which are nonzero. We apply the same representation as for layer 3. The threshold is again 1.9.

The complete parameters are in Table C.5.

C.4 Neuron Model

\( A_1 \) is the pulsed output of the first layer.

For Table C.7, we separate positive and negative weights (columns “Positive” and “Negative”) and adopt a simplified notation: We keep only dit, dah and “end” (denoting the end of a letter), and give the number of delays in the exponent.
For example, the letter A which is repeated in Figure C.3 is described as

- Positive: dit$^5$ (5 because of the 5 delays from $E_1$ to the + input in the neuron), dah$^3$, end
- Negative: dit$^7$, dah$^9$
### Table C.6: All parameters for the main elements of Morse code, architecture with neuron model

<table>
<thead>
<tr>
<th>id</th>
<th>feature, letter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>C1</td>
<td>Small pause</td>
<td><img src="image" alt="Diagram" /></td>
</tr>
<tr>
<td>D1</td>
<td>Dah</td>
<td><img src="image" alt="Diagram" /></td>
</tr>
<tr>
<td>E1</td>
<td>Dit</td>
<td><img src="image" alt="Diagram" /></td>
</tr>
<tr>
<td>E2</td>
<td>Medium pause</td>
<td><img src="image" alt="Diagram" /></td>
</tr>
<tr>
<td>H1</td>
<td>Large pause</td>
<td><img src="image" alt="Diagram" /></td>
</tr>
</tbody>
</table>
**Table C.7:** All parameters for detecting letters, architecture with neuron model

<table>
<thead>
<tr>
<th>id</th>
<th>letter</th>
<th>Positive</th>
<th>Negative</th>
</tr>
</thead>
<tbody>
<tr>
<td>F1</td>
<td>A</td>
<td>dah$^2$, dit$^5$, end</td>
<td>dit$^7$, dah$^8$</td>
</tr>
<tr>
<td>F2</td>
<td>B</td>
<td>dit, dit$^2$, dit$^4$, dah$^7$, end</td>
<td>dit$^6$</td>
</tr>
<tr>
<td>F3</td>
<td>C</td>
<td>dit, dah$^4$, dit$^6$, dah$^{10}$, end</td>
<td>dit$^8$</td>
</tr>
<tr>
<td>F4</td>
<td>D</td>
<td>dit, dit$^2$, dah$^6$, end</td>
<td>dit$^4$, dit$^6$, dit$^8$, dah$^9$</td>
</tr>
<tr>
<td>F5</td>
<td>E</td>
<td>dit, end</td>
<td>dit$^2$, dah$^4$</td>
</tr>
<tr>
<td>F6</td>
<td>F</td>
<td>dit, dah$^4$, dit$^6$, dit$^8$, end</td>
<td>dit$^2$, dah$^{10}$</td>
</tr>
<tr>
<td>F7</td>
<td>G</td>
<td>dit, dah$^4$, dah$^8$, end</td>
<td>dit$^2$, dit$^{10}$, dah$^{11}$</td>
</tr>
<tr>
<td>F8</td>
<td>H</td>
<td>dit, dit$^2$, dit$^4$, dit$^6$, end</td>
<td></td>
</tr>
<tr>
<td>F9</td>
<td>I</td>
<td>dit, dit$^2$, end</td>
<td>dit$^4$, dah$^5$</td>
</tr>
<tr>
<td>F10</td>
<td>J</td>
<td>dah$^3$, dah$^7$, dah$^{11}$, dit$^{13}$, end</td>
<td>dit, dit$^5$, dit$^9$</td>
</tr>
<tr>
<td>F11</td>
<td>K</td>
<td>dah$^3$, dit$^5$, dah$^8$, end</td>
<td>dit$^8$, dit$^9$, dah$^{12}$</td>
</tr>
<tr>
<td>F12</td>
<td>L</td>
<td>dit, dit$^2$, dah$^6$, dit$^8$, end</td>
<td>dit$^4$, dah$^9$</td>
</tr>
<tr>
<td>F13</td>
<td>M</td>
<td>dah$^3$, dah$^7$, end</td>
<td>dit, dit$^5$, dit$^9$, dah$^{11}$</td>
</tr>
<tr>
<td>F14</td>
<td>N</td>
<td>dit, dah$^4$, end</td>
<td>dit$^2$, dit$^3$, dit$^6$, dah$^8$</td>
</tr>
<tr>
<td>F15</td>
<td>O</td>
<td>dah$^3$, dah$^7$, dah$^{11}$, end</td>
<td>dit, dit$^5$, dit$^9$, dit$^{11}$</td>
</tr>
<tr>
<td>F16</td>
<td>P</td>
<td>dit, dah$^4$, dah$^8$,dit$^{10}$, end</td>
<td>dit$^2$, dah$^{11}$</td>
</tr>
<tr>
<td>F17</td>
<td>Q</td>
<td>dah$^4$, dit$^6$, dah$^{10}$, dah$^{14}$, end</td>
<td></td>
</tr>
<tr>
<td>F18</td>
<td>R</td>
<td>dit, dah$^4$, dit$^6$, end</td>
<td>dit$^8$, dah$^{10}$</td>
</tr>
<tr>
<td>F19</td>
<td>S</td>
<td>dit, dit$^2$, dit$^4$, end</td>
<td>dit$^6$, dah$^7$</td>
</tr>
<tr>
<td>F20</td>
<td>T</td>
<td>dah$^3$, end</td>
<td>dit$^5$, dah$^7$, dit</td>
</tr>
<tr>
<td>F21</td>
<td>U</td>
<td>dah$^3$, dit$^5$, dit$^7$, end</td>
<td>dit$^9$, dah$^{10}$</td>
</tr>
<tr>
<td>F22</td>
<td>V</td>
<td>dah$^3$, dit$^5$, dit$^7$,dit$^9$, end</td>
<td></td>
</tr>
<tr>
<td>F23</td>
<td>W</td>
<td>dah$^3$, dah$^7$, dit$^9$, end</td>
<td>dah$^{12}$</td>
</tr>
<tr>
<td>F24</td>
<td>X</td>
<td>dah$^3$, dit$^5$, dit$^7$, dah$^{10}$, end</td>
<td></td>
</tr>
<tr>
<td>F25</td>
<td>Y</td>
<td>dah$^3$, dah$^7$, dit$^9$, dah$^{12}$, end</td>
<td></td>
</tr>
<tr>
<td>F26</td>
<td>Z</td>
<td>dit, dit$^2$, dah$^6$, dah$^9$,end</td>
<td>dit$^4$, dit$^8$</td>
</tr>
</tbody>
</table>
Appendix D

Learning Algorithms

D.1 Melodies

The data for learning are simple melodies (extracted from more or less well-known pieces, see D.1), played on an iPad with the application “Pianist Pro” [44]. There are 20 different melodies and 50 recordings of each.

D.2 Architecture for Learning: Fixed Parameters

D.2.1 First Layer

Both learning algorithms (sinusoidal and neuron model) use the same first layer. It is built with:

\[ A_f = \text{rotm} \left( \frac{2\pi f}{f_s} \right) \]  \hspace{1cm} (D.1)

\[ C = \begin{pmatrix} 1 & 0 \end{pmatrix} \]  \hspace{1cm} (D.2)

\[ (D.3) \]

with \( f_s = 4000\text{Hz} \). For each filter there is one \( \gamma \) used in the computation of the score (cf. Equation 4.4). The parameters are given in Table D.3.
<table>
<thead>
<tr>
<th>Nr</th>
<th>Composer</th>
<th>Title</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Beethoven</td>
<td>Ode to Joy</td>
<td><img src="image" alt="Beethoven Ode to Joy" /></td>
</tr>
<tr>
<td>2</td>
<td>Offenbach</td>
<td>French Cancan</td>
<td><img src="image" alt="Offenbach French Cancan" /></td>
</tr>
<tr>
<td>3</td>
<td>Dvořák</td>
<td>Largo, New World Symphony</td>
<td><img src="image" alt="Dvořák Largo, New World Symphony" /></td>
</tr>
<tr>
<td>4</td>
<td>Strauss</td>
<td>The Blue Danube</td>
<td><img src="image" alt="Strauss The Blue Danube" /></td>
</tr>
<tr>
<td>5</td>
<td>Mozart</td>
<td>Variations K.300e</td>
<td><img src="image" alt="Mozart Variations K.300e" /></td>
</tr>
<tr>
<td>6</td>
<td>Boatner</td>
<td>When the Saints Go Marching In</td>
<td><img src="image" alt="Boatner When the Saints Go Marching In" /></td>
</tr>
<tr>
<td>7</td>
<td>Vivaldi</td>
<td>Spring</td>
<td><img src="image" alt="Vivaldi Spring" /></td>
</tr>
<tr>
<td>8</td>
<td>Mendelssohn</td>
<td>Wedding March</td>
<td><img src="image" alt="Mendelssohn Wedding March" /></td>
</tr>
<tr>
<td>9</td>
<td>Vivaldi</td>
<td>Winter</td>
<td><img src="image" alt="Vivaldi Winter" /></td>
</tr>
<tr>
<td>10</td>
<td>Tchaikovsky</td>
<td>Swan Lake</td>
<td><img src="image" alt="Tchaikovsky Swan Lake" /></td>
</tr>
</tbody>
</table>

**Table D.1:** Melodies 1 to 10 used for learning
<table>
<thead>
<tr>
<th>Nr</th>
<th>Composer</th>
<th>Title</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>11</td>
<td>Bizet</td>
<td>Toreador’s Song</td>
<td>(\text{\text音符}})</td>
</tr>
<tr>
<td>12</td>
<td>Ravel</td>
<td>Bolero</td>
<td>(\text{\text音符}})</td>
</tr>
<tr>
<td>13</td>
<td>Vangelis</td>
<td>Chariots of Fire</td>
<td>(\text{\text音符}})</td>
</tr>
<tr>
<td>14</td>
<td>Händel</td>
<td>Lascia ch’io pianga</td>
<td>(\text{\text音符}})</td>
</tr>
<tr>
<td>15</td>
<td>Morricone</td>
<td>Here’s to You</td>
<td>(\text{\text音符}})</td>
</tr>
<tr>
<td>16</td>
<td>Trénet</td>
<td>La Mer</td>
<td>(\text{\text音符}})</td>
</tr>
<tr>
<td>17</td>
<td>Gluck</td>
<td>Orfeo ed piangia</td>
<td>(\text{\text音符}})</td>
</tr>
<tr>
<td>18</td>
<td>Yepes</td>
<td>Romance</td>
<td>(\text{\text音符}})</td>
</tr>
<tr>
<td>19</td>
<td>Youmans</td>
<td>Tea for Two</td>
<td>(\text{\text音符}})</td>
</tr>
<tr>
<td>20</td>
<td>Kander</td>
<td>New York, New York</td>
<td>(\text{\text音符}})</td>
</tr>
</tbody>
</table>

**Table D.2:** Melodies 11 to 20 used for learning
### Table D.3: Parameters for tone detectors

<table>
<thead>
<tr>
<th>Tone</th>
<th>( f )</th>
<th>( \gamma )</th>
</tr>
</thead>
<tbody>
<tr>
<td>( A_3 )</td>
<td>220</td>
<td>0.3631</td>
</tr>
<tr>
<td>( A_3^# ) or ( B_3^b )</td>
<td>233.1</td>
<td>0.3418</td>
</tr>
<tr>
<td>( B_3 )</td>
<td>246.9</td>
<td>0.3208</td>
</tr>
<tr>
<td>( C_4 )</td>
<td>261.6</td>
<td>0.2998</td>
</tr>
<tr>
<td>( C_4^# ) or ( D_4^b )</td>
<td>277.2</td>
<td>0.2790</td>
</tr>
<tr>
<td>( D_4 )</td>
<td>293.7</td>
<td>0.2586</td>
</tr>
<tr>
<td>( D_4^# ) or ( E_4^b )</td>
<td>311.1</td>
<td>0.2387</td>
</tr>
<tr>
<td>( E_4 )</td>
<td>329.6</td>
<td>0.2192</td>
</tr>
<tr>
<td>( F_4 )</td>
<td>349.2</td>
<td>0.2003</td>
</tr>
<tr>
<td>( F_4^# ) or ( G_4^b )</td>
<td>370</td>
<td>0.1820</td>
</tr>
<tr>
<td>( G_4 )</td>
<td>392</td>
<td>0.1644</td>
</tr>
<tr>
<td>( G_4^# ) or ( A_4^b )</td>
<td>415.3</td>
<td>0.1477</td>
</tr>
<tr>
<td>( A_4 )</td>
<td>440</td>
<td>0.1318</td>
</tr>
</tbody>
</table>

### D.2.2 Second Layer

For the sinusoidal model, we use:

\[
A = \lambda \, \text{rotm} \left( 2\pi f \right) \tag{D.4}
\]

The values of \( f \) and \( \lambda \) for each filter are in Table D.4 where \( T = 4000 \).

### D.2.3 Third Layer

For the sinusoidal model, all filters in this layer use

<table>
<thead>
<tr>
<th>Filter</th>
<th>( f )</th>
<th>( \lambda )</th>
<th>Guard space</th>
</tr>
</thead>
<tbody>
<tr>
<td>F1</td>
<td>( 2\pi/0.25T )</td>
<td>1.00115</td>
<td>0.125( T )</td>
</tr>
<tr>
<td>F2</td>
<td>( 2\pi/0.5T )</td>
<td>1.00057</td>
<td>0.25( T )</td>
</tr>
<tr>
<td>F3</td>
<td>( 2\pi/T )</td>
<td>1.00029</td>
<td>0.5( T )</td>
</tr>
<tr>
<td>F4</td>
<td>( 2\pi/1.5T )</td>
<td>1.00019</td>
<td>0.75( T )</td>
</tr>
<tr>
<td>F5</td>
<td>( 2\pi/2T )</td>
<td>1.00014</td>
<td>1.5( T )</td>
</tr>
<tr>
<td>F6</td>
<td>( 2\pi/2.5T )</td>
<td>1.00012</td>
<td>2( T )</td>
</tr>
<tr>
<td>F7</td>
<td>( 2\pi/1.5T )</td>
<td>1.00019</td>
<td>0.75( T )</td>
</tr>
<tr>
<td>F8</td>
<td>( 2\pi/2T )</td>
<td>1.00014</td>
<td>1.5( T )</td>
</tr>
</tbody>
</table>

**Table D.4: Parameters for second layer (sinusoidal model)**
D.3 Derivation of the Update Equations - Sinusoidal Model

\[ \mathbf{A} = \lambda \, \text{rotm}(2\pi f) \]  
\[ \text{(D.5)} \]

with

\[ f = \frac{2\pi}{4T} \]  
\[ \text{(D.6)} \]

\[ \lambda = 1.00012 \]  
\[ \text{(D.7)} \]

and a guard space of \(2T\), where \(T = 4000\).

D.3 Derivation of the Update Equations - Sinusoidal Model

D.3.1 Highest Layer

We show here the update equation for the elements in \( \mathbf{C} \) of the highest layer. We write an element \( c_{kl} \) where \( k, l \) run over all rows, resp. columns of \( \mathbf{C} \).

\[ c_{kl}^{\text{new}} = c_{kl}^{\text{old}} - \epsilon \frac{\partial J}{\partial c_{kl}} \]

We compute:

\[ \frac{\partial J}{\partial c_{kl}} = \frac{\partial J_P}{\partial c_{kl}} + \frac{\partial J_N}{\partial c_{kl}} \]  
\[ \text{(D.8)} \]

and

\[ \frac{\partial J_P}{\partial c_{kl}} = \sum_{i \neq P} 2.sc(t_i) \frac{\partial sc}{\partial c_{kl}|t=t_i} + 2.(sc(t_P) - 1) \frac{\partial sc}{\partial c_{kl}|t=t_P} \]  
\[ \text{(D.9)} \]

\[ \frac{\partial J_N}{\partial c_{kl}} = \sum_{i=1}^{P} 2.sc(t_i) \frac{\partial sc}{\partial c_{kl}|t=t_i} \]  
\[ \text{(D.10)} \]

\( sc \) is the pulsed score function in the highest layer. The derivative is not straightforward, because of the threshold. We introduce \( \tilde{sc} \), the non-thresholded score function. Then, with the chain rule:
\[
\frac{\partial sc}{\partial c_{kl}} = \frac{\partial sc}{\partial \tilde{sc}} \frac{\partial \tilde{sc}}{\partial c_{kl}} \tag{D.11}
\]

We first compute the second factor:

\[
\frac{\partial \tilde{sc}}{\partial c_{kl} \mid t = t_i} = s^\top \frac{\partial \xi_n}{\partial c_{kl} \mid n = t_i} \tag{D.12}
\]

\[
= s^\top \sum_{k=1}^{n} (A^{n-k})^\top \frac{\partial C^\top}{\partial c_{kl} \mid n = t_i} y_k \tag{D.13}
\]

For \( \frac{\partial C^\top}{\partial c_{kl}} = \left( \frac{\partial C}{\partial c_{kl}} \right)^\top \), we know that \( \frac{\partial C}{\partial c_{kl}} \) is a matrix of same size as \( C \), containing 1 at position \((i, j)\), and zeros everywhere else. This term can thus be computed as the (non-thresholded) score in the last layer, obtained when \( C \) of the last layer is \( 1_{kl} \). We call this term \( \tilde{s}_c^{kl} \):

\[
\frac{\partial \tilde{sc}}{\partial c_{kl} \mid t = t_i} = \tilde{s}_c^{kl}(t_i) \tag{D.14}
\]

For the first term, we have to be more creative. We use a symmetric derivative:

\[
\frac{\partial sc}{\partial \tilde{sc}} \approx \frac{sc(\tilde{sc} + h) - sc(\tilde{sc} - h)}{2h}
\]

for some small (positive) value of \( h \).

In the cost function \( J_P \), for \( i \neq P \), only the terms where a pulse is present in the last layer before the endpoints matter (the others are multiplied by the thresholded score function, which is 0): for those points, \( \tilde{sc} \) was already higher than the threshold, so \( \tilde{sc} + h \) is even higher \( (sc(\tilde{sc} + h) = 1) \). If \( sc(\tilde{sc} - h) = 1 \), the derivative is 0 and will have no influence. For \( sc(\tilde{sc} - h) \) to be different from 1, the smallest value possible is \( h = \tilde{sc}(t_i) - th \). (This cannot be zero, because the pulse score is set to 1 only if \( \tilde{sc} > th \)).

For the endpoint \( t_P \): if \( sc(t_P) = 1 \), the contribution from the endpoint to \( \frac{\partial J_P}{\partial c_{kl}} \) disappears (it is multiplied by \( sc(t_P) - 1 \), which is 0, in the gradient). Thus, we compute \( \frac{\partial sc}{\partial \tilde{sc}} \) only if no pulse is present at \( t_P \).
D.3 Derivation of the Update Equations - Sinusoidal Model

- $\tilde{s}c - h$ is smaller than $\tilde{s}c$ at $t_P$, thus $sc(\tilde{s}c - h) = 0$
- $\tilde{s}c + h$ is larger than $th$ for $h = th - \tilde{s}c + \eta$ (in practice we took $\eta = 0.001$)

Therefore:

$$\frac{\partial J_P}{\partial c_{kl}} = \sum_{i \neq P} \frac{sc(t_i)}{\tilde{s}c(t_i) - th} \tilde{s}c^{kl}(t_i) + \frac{(sc(t_P) - 1)}{th - \tilde{s}c(t_P) + \eta} \tilde{s}c^{kl}(t_P)$$  \hspace{1cm} (D.15)

The reasoning for $J_N$ is similar. We obtain:

$$\frac{\partial J_N}{\partial c_{kl}} = \sum_i \frac{sc(t_i)}{\tilde{s}c(t_i) - th} \tilde{s}c^{kl}(t_i)$$  \hspace{1cm} (D.16)

In conclusion, to update $c_{kl}$ in the last layer, we need the thresholded and non-thresholded score function, and the non-thresholded score for $c_{kl} = 1$, all other elements of $C$ are 0. The nice thing is that we do not need to keep track of the points where score is higher than a threshold, because the factor $sc(t_i)$, the pulsed score function takes care of this automatically.

The final formula to update $c_{kl}$ is:

$$c_{kl}^{new} = c_{kl}^{old} - \epsilon \left( \frac{\partial J_P}{\partial c_{kl}} + \frac{\partial J_N}{\partial c_{kl}} \right)$$  \hspace{1cm} (D.17)

with $\frac{\partial J_P}{\partial c_{kl}}$ given by D.15 and $\frac{\partial J_N}{\partial c_{kl}}$ given by D.16.

D.3.2 Computation of Endpoints for Intermediate Layers

We want to find the maximum of the weighting signal given the endpoint $n$ of the layer directly above:

$$\tilde{y}_k = CA^{k-n}s$$

We denote $(c_{i1} \ c_{i2}) = \mu_i(\cos(\beta_i) \ \sin(\beta_i))$ a row of $C$.

For each channel $i$, we want to find the maximum of

$$f[k] = \lambda^{k-n}(c_{i1} \cos((k-n)\Omega) + c_{i2} \sin((k-n)\Omega))$$  \hspace{1cm} (D.18)

$$= \lambda^{k-n} \mu_i(\cos((k-n)\Omega - \beta_i))$$  \hspace{1cm} (D.19)
The derivation w.r.t \( k \), set to 0 gives:

\[
\tan((k - n)\Omega - \beta_i) = \ln\left(\frac{\lambda}{\Omega}\right)
\]  
(D.20)

From there, we find the endpoint

\[
k = n + \frac{\beta_i + \arctan\left(\ln\frac{\lambda}{\Omega}\right)}{\Omega}
\]  
(D.21)

D.4 Derivation of the Update Equations - Neuron Model

D.4.1 Update of \( \lambda \)

We have

\[
\lambda_k = \sigma(x_k) = \frac{1}{1 + \exp(-x_k)}
\]

then

\[
\frac{\partial \lambda_k}{\partial x_k} = \sigma(x_k)(1 - \sigma(x_k))
\]

Thus the learning rule is:

\[
x_k^{\text{new}} = x_k^{\text{old}} - \epsilon \frac{\partial J}{\partial \lambda_k} \frac{\partial \lambda_k}{\partial x_k}
\]  
(D.22)

\[
= x_k^{\text{old}} - \epsilon \frac{\partial J}{\partial \lambda_k} \sigma(x_k)(1 - \sigma(x_k))
\]  
(D.23)

and

\[
\lambda_k^{\text{new}} = \sigma(x_k^{\text{new}})
\]

We now detail the derivative of the cost function. We use the same notation as for the sinusoidal model: \( \tilde{s}c \) is the score function before applying the threshold.

\[
\frac{\partial J}{\partial \lambda_k} = \frac{\partial J_P}{\partial \lambda_k} + \frac{\partial J_N}{\partial \lambda_k}
\]  
(D.24)

where
\[
\frac{\partial J_P}{\partial \lambda_k} = \sum_{i \neq P} 2sc(t_i) \frac{\partial sc}{\partial \lambda_k | t=t_i} + 2(sc(t_P) - 1) \frac{\partial sc}{\partial \lambda_k | t=t_P} \tag{D.25}
\]

\[
= \sum_{i \neq P} 2sc(t_i) \frac{\partial sc}{\partial \tilde{sc}} \frac{\partial \tilde{sc}}{\partial \lambda_k | t=t_i} + 2(sc(t_P) - 1) \frac{\partial sc}{\partial \tilde{sc}} \frac{\partial \tilde{sc}}{\partial \lambda_k | t=t_P} \tag{D.26}
\]

\[
\frac{\partial J_N}{\partial \lambda_k} = \sum_{i=1}^{P} 2sc(t_i) \frac{\partial sc}{\partial \tilde{sc}} \frac{\partial \tilde{sc}}{\partial \lambda_k | t=t_i} \tag{D.27}
\]

The computation of \( \frac{\partial sc}{\partial \tilde{sc}} \) is not dependent on \( \lambda_k \). We only need to compute \( \frac{\partial \tilde{sc}}{\partial \lambda_k} \).

For \( m \) input channels, with one pulse at position \( m_i \) in channel \( i \), we have shown in [B.3] that the score signal at time instant \( n \) is:

\[
\tilde{sc}_n = \sum_{i=1}^{m} \alpha_i c_i \lambda_i^{n-m_i} - \lambda_0^{n-m_i} \frac{\lambda_i - \lambda_0}{\lambda_k - \lambda_0}
\]

From there, we compute the partial derivatives:

\[
\frac{\partial \tilde{sc}_n}{\partial \lambda_k} = \frac{\alpha_k c_k (n - m_k) \lambda_k^{n-m_k}}{\lambda_k (\lambda_k - \lambda_0)} - \frac{\alpha_k c_k \lambda_i^{n-m_k} - \lambda_0^{n-m_k}}{(\lambda_k - \lambda_0)^2}
\]

Remark I: In the state vector, the \((k+1)th\) row is

\[
r_k = \alpha_k c_k \lambda_k^{n-m_k}
\]

and the score function, if \( c_k \) is 1, all other elements of \( C \) being 0, is \( \tilde{sc}^k \):

\[
\tilde{sc}^k_n = \alpha_k \frac{\lambda_k^{n-m_k} - \lambda_0^{n-m_k}}{\lambda_k - \lambda_0}
\]

This means the expression can be rewritten as:

\[
\frac{\partial \tilde{sc}_n}{\partial \lambda_k} = \frac{n - m_k}{\lambda_k (\lambda_k - \lambda_0)} r_k - \frac{c_k \tilde{sc}^k}{\lambda_k - \lambda_0}
\]

The complete update rule is:

\[
x_{new} = x_{old} - \epsilon \frac{\partial J}{\partial \lambda_k} \sigma(x_k)(1 - \sigma(x_k)) \tag{D.28}
\]
with
\[
\frac{\partial J}{\partial \lambda_k} = \frac{\partial J_P}{\partial \lambda_k} + \frac{\partial J_N}{\partial \lambda_k} \quad (D.29)
\]

and
\[
\frac{\partial J_P}{\partial \lambda_k} = \sum_{i \neq P} \frac{sc(t_i)}{\tilde{sc}(t_i)} - th \left( \frac{(t_i - m_k)r_k}{\lambda_k(\lambda_k - \lambda_0)} - \frac{c_k\tilde{sc}^k(t_i)}{\lambda_k - \lambda_0} \right) + \frac{sc(t_P) - 1}{th - \tilde{sc}(t_P) + \eta} \left( \frac{t_P - m_k}{\lambda_k(\lambda_k - \lambda_0)}r_k - \frac{c_k\tilde{sc}^k(t_P)}{\lambda_k - \lambda_0} \right) \quad (D.30)
\]
\[
\frac{\partial J_N}{\partial \lambda_k} = \sum_i \frac{sc(t_i)}{\tilde{sc}(t_i)} - th \left( \frac{(t_i - m_k)r_k}{\lambda_k(\lambda_k - \lambda_0)} - \frac{c_k\tilde{sc}^k(t_i)}{\lambda_k - \lambda_0} \right) \quad (D.32)
\]

Remark II: We derived the equations in a purely monomial setting. In practice, if there are more than one pulse in channel \( k \), we use the last one before \( t_i \) as \( m_k \).

D.4.2 Computation of Endpoints for Intermediate Layers
We want to find the maximum of the weighting signal given the endpoint \( n \) of the layer directly above:
\[
\tilde{y}_k = CA^{k-n}s
\]
We denote \( c_i \) the nonzero element in row \( i \) of \( C \).
For each channel \( i \), we want to find the maximum of
\[
f[k] = c_i \frac{\lambda_i^{m_i-k} - \lambda_0^{m_i-k}}{\lambda_i - \lambda_0} \quad (D.33)
\]
The derivation w.r.t \( k \), set to 0 gives:
\[
\ln \lambda_i \lambda_i^{m_i-k} = \ln \lambda_0 \lambda_0^{m_i-k} \quad (D.34)
\]
From there, we find the endpoints:

\[ m_i = k - \frac{\ln \left( \frac{\ln \lambda_i}{\ln \lambda_0} \right)}{\ln \lambda_i - \ln \lambda_0} \]  

(D.35)
Bibliography


Series in Signal and Information Processing

edited by Hans-Andrea Loeliger


Vol. 5: Markus Erne, Signal Adaptive Audio Coding using Wavelets and Rate Optimization. ISBN 3-89649-625-5


Vol. 7: Heinz Mathis, Nonlinear Functions for Blind Separation and Equalization. ISBN 3-89649-728-6

Vol. 8: Daniel Lippuner, Model-Based Step-Size Control for Adaptive Filters. ISBN 3-89649-755-3

Vol. 9: Ralf Kretzschmar, A Survey of Neural Network Classifiers for Local Wind Prediction. ISBN 3-89649-798-7


Vol. 11: Pascal O. Vontobel, Algebraic Coding for Iterative Decoding. ISBN 3-89649-865-7

Vol. 12: Qun Gao, Fingerprint Verification using Cellular Neural Networks. ISBN 3-89649-894-0


Hartung-Gorre Verlag Konstanz – http://www.hartung-gorre.de