A Neural Network Based Brain–Computer Interface for Classification of Movement Related EEG

Pontus Forslund

INSTITUTE OF TECHNOLOGY
LINKÖPING UNIVERSITY

LiTH-IKP-EX-2107
Linköping, December 2003

Supervisor: **Torbjörn Alm**
Thomas Karlsson

Examiner: **Kjell Ohlsson**

Abstract

A brain–computer interface, BCI, is a technical system that allows a person to control the external world without relying on muscle activity. This thesis presents an EEG based BCI designed for automatic classification of two dimensional hand movements. The long-term goal of the project is to build an intuitive communication system for operation by people with severe motor impairments. If successful, such system could for example be used by a paralyzed patient to control a word processor or a wheelchair.

The developed BCI was tested in an offline pilot study. In response to an external cue, a test subject moved a joystick in one of four directions. During the movement, EEG was recorded from seven electrodes mounted on the subject's scalp. An autoregressive model was fitted to the data, and the extracted coefficients were used as input features to a neural network based classifier. The classifier was trained to recognize the direction of the movements. During the first half of the experiment, real physical movements were performed. In the second half, subjects were instructed just to imagine the hand moving the joystick, but to avoid any muscle activity.

The results of the experiment indicate that the EEG signals *do* in fact contain extractable and classifiable information about the performed movements, during both physical and imagined movements.

Keywords: Brain–Computer Interface, Neural Networks, EEG, Autoregressive modeling

Acknowledgments

Many people contributed to this thesis, in one way or another. First and foremost, I would like to thank my advisors, Professor Kjell Ohlsson, Dr Thomas Karlsson and Torbjörn Alm for introducing me to the fascinating field of Brain–Computer Interfacing and for guiding me through the project. I am also deeply indebted to Staffan Magnusson who provided invaluable help during the data acquisition phase of the experiment. The four people, who volunteered to participate as test subjects in this study, deserve a special thank. Thank you. You know who you are. I would also like to express my appreciation to my friends and former colleagues at Virtual Technology AB for their support and understanding. Last, but not least, I would like to thank Katarina for her endless love and support and for helping me improve the report by providing helpful comments and thorough proofreading.

This thesis is submitted in partial fulfillment of the requirements for the degree of Master of Science in Applied Physics and Electrical Engineering at Linköping University, Sweden. The project was made in cooperation with Virtual Technology AB and The Swedish Defense Research Agency.

Contents

Notation

Throughout this thesis, the following notation is used. In general, scalar variables are set in italics, a_{ij}, vectors and matrices are denoted by bold letters, $\mathbf{y} = \mathbf{A}^T\mathbf{x}$, and calligraphic symbols are used to represent sets and spaces, $p_1 \in \mathcal{D}$.

Acronyms

ALS	amyotrophic lateral sclerosis
ANN	artificial neural network
AR	autoregressive model
BCI	brain–computer interface
CNS	central nervous system
DSP	digital signal processor
EEG	electroencephalogram
EOG	electrooculogram
FFT	fast Fourier transform
IIR	infinite duration impulse response
LAR	lagged autoregressive model
MLP	multilayer perceptron
RMP	resting membrane potential

Operations

$\mathbf{a}^{\mathrm{T}}$	transpose of vector **a**
$\mathbf{x}^{\mathrm{T}}\mathbf{w}$	inner product of vectors **x** and **w**

∇	gradient operator
$\wedge$	logical and
$\vee$	logical or
$\neg$	logical not
$\lvert x \rvert$	absolute value of x
$\lVert x \rVert$	Euclidean norm of x
$\cap$	intersection
$\cup$	union
$\subseteq$	subset
$\subset$	proper subset, a subset which is not the entire set
$\in$	member of
$[x]$	concentration of ion x

Symbols

a_{pi}	AR-model coefficient
α	momentum term
b_{pi}	backward prediction error
β	sigmoid scale parameter
$\mathcal{C}$	set of EEG electrodes used for classification
$\mathcal{C}_{tot}$	set of all EEG electrodes used in experiment
c_{win}	window center
$\mathcal{D}_{tes}$	set of test patterns
$\mathcal{D}_{trn}$	set of training patterns
$\mathcal{D}_{val}$	set of validation patterns
δ_j	local gradient of neuron j
e_{pi}	forward prediction error
E	error function
E_p	total prediction error
η	learning rate parameter
F	Faraday's constant
f_c	filter cut-off frequency

f_{max}	highest frequency component
f_s	sample frequency
$\varphi(\mathrm{z})$	logistic sigmoid function
$\mathrm{H(z)}$	Heaviside threshold function or filter transfer function
$\mathbf{H}_{\mathrm{ann}}$	ANN node configuration vector
L	AR model lag
n	discrete time
p	AR model order
P	relative permeability
$\mathrm{P(f)}$	power spectrum
R	autocorrelation function
R	gas constant
R_c	classification rate
$\rho(\mathcal{C})$	information content in electrode set
t	continuous time
T	sample period or absolute temperature
V_m	cell membrane potential
$V_{\mathbf{p}}$	potential at point $\mathbf{p}$
$\mathbf{w}$	ANN weight vector
w_{win}	window width
w_0	threshold value
$\tilde{x}_n$	predicted value of x_n
$\mathbf{x}$	ANN input vector
$\mathbf{y}$	ANN output vector
$\mathbf{y}_{\mathrm{d}}$	desired ANN output vector

Chapter 1

Introduction

In the classic action book Firefox, written by Craig Thomas in 1977, an American fighter pilot is sent off to the Soviet Union on a secret mission. The task is to infiltrate the Russian Air Force and steal a new, highly advanced, fighter aircraft, the MIG31 Firefox. What makes this fighter so special is not that it is invisible to all radar systems or that it can reach a top speed of five times the speed of sound. Instead, the most desirable feature of the Firefox is its *thought controlled* combat computer. Using a special helmet equipped with electrodes that detects his brain waves, the pilot can engage the plane's weapons on a target simply by using his thoughts.

The Firefox system is just one of many examples of people having their minds coupled to computers that have appeared in different works of fiction. As often depicted, the user simply thinks of a command, and the computer responds, just as if the command had been entered on a keyboard. Such system would arguably be the most intuitive interface between the user and a computer, acting as a direct extension of the human nervous system. In accordance with the terminology established by other researchers in this field, we will refer to such system as a *brain–computer interface, BCI.*

1.1 Background

The dream of creating a direct link between the human brain and an electronic device was born in the mid 1960's when the United States Department of Defense initiated a program to help pilots interact with their aircraft. The idea was to reduce the mental workload of the pilot by providing a new intuitive channel of communication with the plane's computer. Unfortunately, the technology of the time was not sophisticated enough to be used for such complex tasks, and the program was cancelled after only a few years. However, even if the immediate success was limited, the project laid the ground for other research programs and, since then, the area has grown extensively and attracted researchers from many different disciplines.

1.2 Motive

The perhaps most interesting and important application of the BCI technology today, is medical rehabilitation. As the operation of a true brain–computer interface does not require any muscle activity, a communication system based on BCI techniques could be operated even by people with severe motor impairments. A typical example is patients suffering from *amyotrophic lateral sclerosis*, ALS, an incurable, neurological disease that affects the ability to control muscles and gradually leads to paralysis. ALS does not affect the brain cells or any intellectual functions, but it causes the nerve cells that carry information from the brain to the muscles to cease functioning and eventually die off. In late stages of ALS, no voluntary muscle activity can occur, even though the brain still generates the corresponding control signals. The purpose of a brain–computer interface in this case, is to capture those signals directly from the brain and convert them into a form that can be understood by a computer or other device. The signals could then be used to operate for example a wheelchair, an artificial limb or a word processor, thereby providing the patient with communication channels previously cut-off by the disease.

This type of applications is now the major motivation for most researchers, and during the last decade, great advances have been made. The progress is partly due to the remarkable advances in computer technology, but also, due to an improved understanding of the functionality of the brain. Despite the encouraging results of many programs, one should remember that the human brain is an immensely complex organ, and the information that can be extracted from it is very difficult to decipher. Designing a reliable link between the human brain and an electronic device therefore remains a formidable challenge.

1.3 What is a Brain–Computer Interface?

In general terms, a brain–computer interface is a technical system that allows a person to control the external world without relying on muscle activity. Rather than depending on the body's normal output pathways of nerve cells and muscles, the input control signals are represented by electrophysiological impulses recorded directly from the brain.

1.3.1 BCI Systems are not Mind Readers

A brain–computer interface is designed to recognize patterns in data extracted from the brain and associate the patterns with commands. Very often these patterns, or states, are referred to as *thoughts,* and accordingly, systems that rely on BCI techniques for input are described as being *thought controlled.* This is somewhat unfortunate because it easily leads to the misconception that BCI systems can read minds. They cannot. There is no way any technical system today could tell *what a person is thinking,* and there are several reasons for that. First, the sensing techniques that can be used to extract information from the brain are imprecise, and they can only represent a microscopic fraction of the brains total activity. Second, even if we were able to extract the states from all the cells in the brain, that information would be way too complex to handle, even for the most powerful super-computer. Third, brains are individual and unique. If two people think of the exact same thing, the action schemes of their brain cells are completely different. So, let us state it once and for all, *computers cannot read minds.*

1.3.2 Techniques of Brain–Computer Interfacing

So, how does it really work then? Well, that of course depends on the techniques used. Since the BCI area is rather new as a scientific field, there are no rights or wrongs, and much of the research is based on trial-and-error. Hence, many different approaches have been tried to solve similar problems. Still, a few things are common to all brain–computer interfaces. For example, practically all BCI systems consist of at least three modules: a *signal recorder*, a *signal preprocessor* and a *classification module*, as described in Figure 1.1.

Signal Recording

The majority of all existing BCI systems, including the one developed in this project, use recordings of the electrical activity in the brain, so called electroencephalogram, EEG, as the source of input. There are other methods, but most of them have some kind of serious drawback associated with them that makes them inappropriate for this type of job. EEG recorders, on the other hand, are typically fast, relatively inexpensive, and do normally

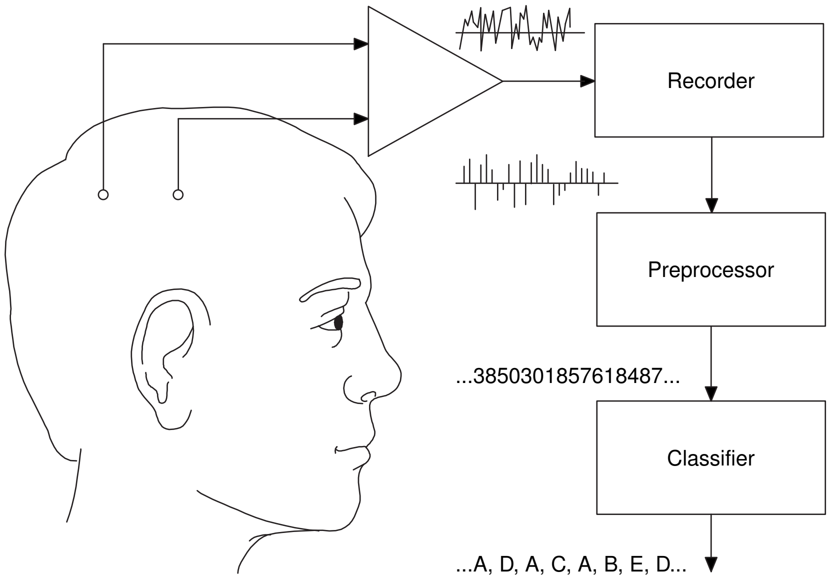


Figure 1.1: *The principle of a brain–computer interface. Most BCI systems consist of at least three modules: a recorder, a signal preprocessor and a classifier.*

not require any invasive procedures. We will therefore focus on EEG inputs from now on. The purpose of the recorder module is to measure, amplify and filter the EEG signal, before converting it to digital form and passing it on to the preprocessor module.

Signal Preprocessing

In the preprocessor, the digital input signal is converted to a form that makes it easier to classify. This transformation may include further filtering, noise reduction, combination of different input channels or other forms of digital signal processing.

Classification

The last step in the BCI chain is the classifier module. Here, the features extracted by the preprocessor are used to sort consecutive input signal segments into categories. All categories generate different outputs from the system. Often the classifier is based on some sort of adaptive self-learning software that is trained to recognize the important patterns. In this project, we are using a special form of artificial neural network for that part.

1.3.3 Training Process

Since all brains are different, no BCI system can be developed that works perfectly with all users, right from the beginning. Consequently, connecting a brain–computer interface to a new user is an integration process that has to involve some degree of adaptation. There are two ways to do this: have the user adapt to the system or design the software to adapt to the specific user. Both ways have their pros and cons.

The former approach requires less complex systems, since most of the adaptation work is done by the user. In this case, the user learns to operate a static system through a biofeedback process. The downside of this approach is that the learning process may be long, typically requiring months of daily training to reach acceptable accuracy.

The other way to go is to incorporate adaptive self-learning software into the classifier module. This is the approach taken in this project. The main idea is to place the burden of adjusting on the system rather than on the user. Interfaces using this technique are typically faster, more general, more extensible, and easier to use. On the other hand, they are also technically more complex, and designing a good BCI based on this approach is a much more challenging task. Despite the technical difficulties, many researchers agree [18, 4] that this type of system has greater potential in the long run, and much effort is therefore put into solving the problems associated with it. A summary of the achievements of different BCI research groups can be found in [27] and [7].

1.3.4 Problems

The biggest problem with most brain–computer interfaces is low accuracy. Sometimes the output of the system does not match the input. This, of course, can be more or less serious depending on the application. If used for moving the cursor on a computer screen an erroneous output every now and then might be tolerable, but if used for controlling the motion of a wheelchair such behavior is, of course, unacceptable.

Another problem associated with many BCI paradigms is too long input–output delays. Today, the most successful systems work at a transfer rate of less than 30 bits per minute [28]. That might be enough to operate a simple word processor system, but it is definitely too slow to control a wheelchair.

Most research today, including the project described in this thesis, therefore focuses on improving the two factors of speed and accuracy of BCI communication.

1.4 Purpose of this Work

The purpose of this project is twofold: to design and implement an offline brain–computer interface, and then to test this system in a real BCI study. The work on the project can accordingly be divided into two phases: one theoretical and practical and one experimental.

1.4.1 BCI Framework Design

The first phase comprises the design and implementation of a fully functional offline BCI system. The system should include functionality for recording, preprocessing, and classifying multidimensional EEG data, as already described in Figure 1.1. The preprocessor and the classifier should be implemented as separate modules within the program to allow for easy modification and testing of new design ideas. To make the system design successful, this first phase also has to include a thorough investigation of the theory behind the different parts of a BCI.

1.4.2 BCI Experiment

In the second phase of the project, the developed system should be tested in a pilot study of movement related EEG, that is, brain waves recorded during voluntary limb movement. The purpose of this part is to see if it is possible to detect and categorize two dimensional joystick movements, both when the subject performs real physical actions and when the movements are merely imagined. The procedure of the experiment is as follows. The test subject sits in front of a computer, with one hand on a joystick. In response to a stimulus given by the computer, the joystick is moved in one of four directions: left, right, up and down. During the movement, EEG data is recorded by the BCI. After the experiment is completed, the data is processed, and the output of the BCI system, one of five classes: *left*, *right*, *up*, *down* or *rest,* is compared to the movements actually performed. The accuracy of the system is defined as the number of correct classifications divided by the total number of movements. In the first half of the experiment, real physical movements are performed. In the second half, subjects are instructed just to imagine the movements, but to remain relaxed.

1.4.3 Questions

The short-term aim of this research is to find suitable algorithms for preprocessing and classification of movement related EEG data. In the project described in this thesis, we are taking a first step towards that by developing a framework for BCI algorithm evaluation and by testing that system in a pilot study. The long-term research goal is to design an *online* brain–computer interface for use by people with motor impairments.

The tool developed in this first step operates *offline,* that is, no classification is done in real-time. There are two reasons for that. First, the main purpose of the system is to serve as a research tool for studies on different preprocessor and classification procedures. It is therefore important that data can be recorded once and then analyzed multiple times to compare and evaluate different algorithms. This would not be possible in a real-time system that requires a continuous stream of new input data. The second reason is that designing a successful *online* BCI system is a very complicated task that involves a number of non-technical questions that are beyond the scope of this work. For example: "how does the brain respond and adapt to real-time feedback?" and "how does that adaptation affect the EEG?". All of these problems are added on top of the technical difficulties involved in building a brain–computer interface. Due to the complexity of the problem, an online system design process is therefore likely to be fruitless if not preceded by a thorough investigation about its associated technical problems. In other words, before spending any time and effort on developing a real-time classification system we should make sure that we have evaluated the algorithms in an offline study and solved any problem associated with them. This is exactly what we are about to do in this project.

Hence, the bottom line question asked in this thesis is – *can we find a set of BCI techniques for extracting information from EEG that performs well enough to motivate further research in an on-line study?* The conclusion is heavily dependant on the answers of a couple of other key-questions.

- Is it at all possible to discriminate between two-dimensional joystick movements based on EEG recordings? And if so, with what accuracy?

- Is it possible to make the same classification even if the movements are only imagined?

- How much data is needed to be able to make a reliable classification, that is, what is the speed of the system?

- What parts of the brain are important sources of EEG for such classifications? Where on the skull should the electrodes be placed?

1.5 Limitations

In the experiment part of the project, EEG data is collected from four test subjects, and one of them is selected for a deeper analysis. The reason we are focusing on one test subject only is that the whole data analysis is very time consuming and that the purpose of the project is to investigate the *possibility* of extracting information from the EEG. To reach statistical validity we would have to test a large number of subjects. However, such investigation would not be meaningful unless we know that the selected method has potential.

The technical investigation, in this first step, was limited to one type of preprocessor and one type of adaptive classifier.

1.6 Thesis Disposition

As described, the work on this project has been naturally divided into two phases. The first phase contains the design and implementation of a BCI system, and the second part the conduction of an experiment using that system. To make the work foreseeable the same structure has been applied on the thesis.

The first part of the report therefore focuses on the theoretical background of the modules that make up a brain–computer interface. Chapter 2 discusses the first and most important component in any BCI, the human brain. That discussion is followed by a chapter on EEG and EEG recording. The next link in the BCI chain is the preprocessor. Consequently, Chapter 4 is devoted to preprocessing of EEG data. The last module in any BCI is the classifier. In this project, we focus on classifiers based on Artificial Neural Networks. This is done in Chapter 5.

The second part of the thesis describes conducted experiment. Chapter 6 describes the method used, including the experiment design, the equipment used etc. The analyses of the data, including the results, are presented in Chapter 7. Chapter 8 contains a discussion of the findings and suggestions for follow-up experiments.

The software developed in phase one is briefly described in Appendix A.

Chapter 2

The Human Brain

The central nervous system, CNS, consists of two main components, the spinal cord and the brain, where the latter is defined as the part that is located inside the skull. In this chapter, we will focus on the structure of the human brain, its basic building blocks, *the neurons*, and the electrical activity inside these neurons.

2.1 Biological Neurons

The human brain is one of the most complicated objects ever studied and, on the whole, it is still poorly understood. High-level questions like "what is a thought?" and "how does the mind work?" remain unanswered, and probably will for a long time. Instead, the assembled knowledge about the brain is focused on low-level operations like, "what kind of cells make up the different parts of the brain?" and "how are these cells interconnected?".

The most fundamental component of the brain, and in fact of the whole nervous system, is the neuron. The concept was introduced by Ramón y Cajál in 1911 and led to a breakthrough in the understanding of the nervous system.

One can distinguish two different kinds of neurons, *interneuron cells* that are the dominant type of neurons in the CNS, and *output cells* that connect the brain to muscles and sensory organs and different areas of the brain to each other. Even if the two types of neurons serve different purposes and differ from each other on a detailed level, the general structure of most neurons is the same, as illustrated in Figure 2.1.

In the center of the neuron is the cell body, the *soma*. Attached to the soma are two types of branches: the *dendrites* that serve as inputs and the output *axon*.

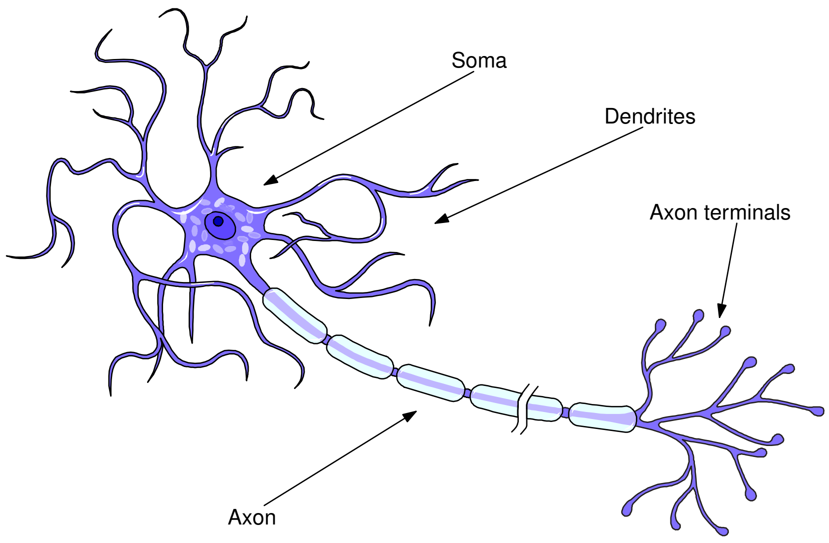


Figure 2.1: *The biological structure of a neuron. A typical neuron consists of three distinct parts: the soma, the dendrites and the axon.*

2.1.1 Dendrites

The term dendrite is derived from *dendron*, the Greek word for tree, reflecting that their shape resembles that of a tree with branches that fork and fork again into finer and finer structure. The number of branches varies from only a few to hundreds of thousands. The dendrites act as input connections through which all the information to the neuron arrives. An important property of the dendrites is their ability to change over time, to break connections with some nerve cells and form new connections with other cells. This fundamental property is essential to the learning process.

2.1.2 Axon

The signal produced in the dendrites and the cell body is transmitted away from the neuron through the output axon. The axon is a structure specialized in carrying information over distances in the nervous system and may reach up to a meter or more in length. The information is transmitted in form of electrical impulses called *action potentials*.

The axon begins at the part of the cell body called the *axon hillock* and usually ends in a network of branches, *collaterals*, and endpoints, the so-called, *pre-synaptic terminals*.

2.1.3 Synapses

There are no anatomical connections between neurons. Instead, the cell membranes of the neurons are separated by very small physical gaps, called *synapses*. These junctions allow impulses from one cell to transmit to another cell, electrically or using a chemical transmitter substance.

2.1.4 Models of the Neuron

Anatomically and functionally, a neuron is a stand-alone processing unit specialized for transmitting and receiving electro-chemical signals, so called *action potentials*. It accepts inputs from other cells through its dendrites and adds them up in some way. If the result of the addition meets some, neuron-dependent, condition, determined by the trigger zone near the axon hillock, the cell "fires" by sending out a new signal through its output axon. Figure 2.2 shows a simple mathematical model of a neuron.

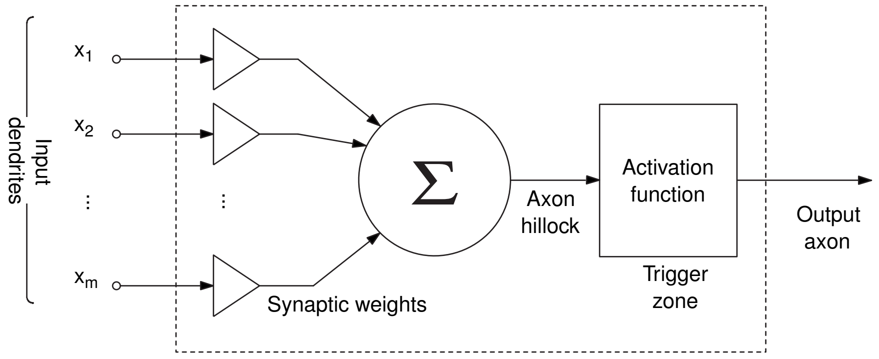


Figure 2.2: *Simple mathematical model of a neuron.*

2.2 The Electrical Activity in the Neurons

Like all cells in the nervous system, the neurons are composed mainly of fluids contained within very thin membranes. These membranes are *semi-permeable,* meaning that the interchange of molecules and ions between the inside and the outside is restricted, but not entirely shut off. The fluid inside of a neuron contains a high concentration of potassium K^+ and low concentrations of sodium Na^+ and chloride Cl^- ions. Conversely, outside the cell, the concentration of Na^+ and Cl^- is high and the number of K^+ ions is low. This difference in ionic concentration gives rise to an electrical voltage that can be described by the Goldman equation.

$$V_m = \frac{RT}{F} \ln\left(\frac{P_K\left[K^+\right]_o + P_{Na}\left[Na^+\right]_o + P_{Cl}\left[Cl^-\right]_o}{P_K\left[K^+\right]_i + P_{Na}\left[Na^+\right]_i + P_{Cl}\left[Cl^-\right]_i}\right) \approx -70\,\mathrm{mV} \qquad (2.1)$$

Here, R symbolizes the gas constant, T the absolute temperature and F, Faraday's constant. The terms P_K, P_{Na} and P_{Cl} are the relative permeability of the membrane to these three ions. The concentration of the different ions inside and outside the cell is written as $[ion]_i$ and $[ion]_o$ respectively. For a typical neuron at rest, the membrane potential is about $-70\,\mathrm{mV}$. This is referred to as the *resting membrane potential*, RMP.

2.2.1 Disturbed Electrochemical Equilibrium

It is important to remember that the forces of nature strive to move free particles from a higher concentration to a lower. Hence, if not controlled, the ions will move through the cell membrane until the concentration inside the cell equals the concentration outside. This kind of equilibrium is called a *chemical equilibrium*. Since the ions around the cell membrane are electrically charged they are also affected by the electrical forces trying to move positively charged ions to areas of negative charge and negatively charged ions to positive areas. We say that nature is working to achieve *electrical equilibrium* by eliminating the voltage over the cell membrane. The total forces acting on the system due to the chemical and electrical imbalances is called an *electrochemical gradient*.

The total gradient of the cell is maintained by the so-called sodium–potassium pump that continuously pumps K^+ ions into the cell and Na^+ ions out through the membrane. Without this pump, the gradient would level out and thereby, as we will see, remove the cell's ability to transmit signals to other cells.

2.2.2 Action Potential

The equilibrium in the neuron can be disturbed in many ways, electrically, chemically and even mechanically. When the cell is stimulated in any of these ways, the permeability of its membrane changes, making it possible for ions to flow in and out of the cell. If the flow of ions through the membrane is low, the sodium-potassium pump will quickly eliminate the disturbance and restore the concentration rates. If the flow is high, however, the capacity of the ion pump may not be sufficient to reset the electrochemical gradient. This, of course, results in a change of the resting potential.

If the voltage exceeds –55 mV, a special form of protein molecules opens up the membrane to sodium ions, allowing Na^+ to flow freely into the cell for a fraction of a millisecond before it closes again. This process is called *depolarization* and results in a large change of the membrane potential to about +30 mV. Shortly after the depolarization, other proteins open up the membrane for potassium ions resulting in a flow of K^+ out of the cell. This is called *repolarization*. The net flow of positively charged ions in to the cell is therefore positive at first and then negative resulting in the membrane potential change depicted in Figure 2.3. This impulse is called *action potential*.

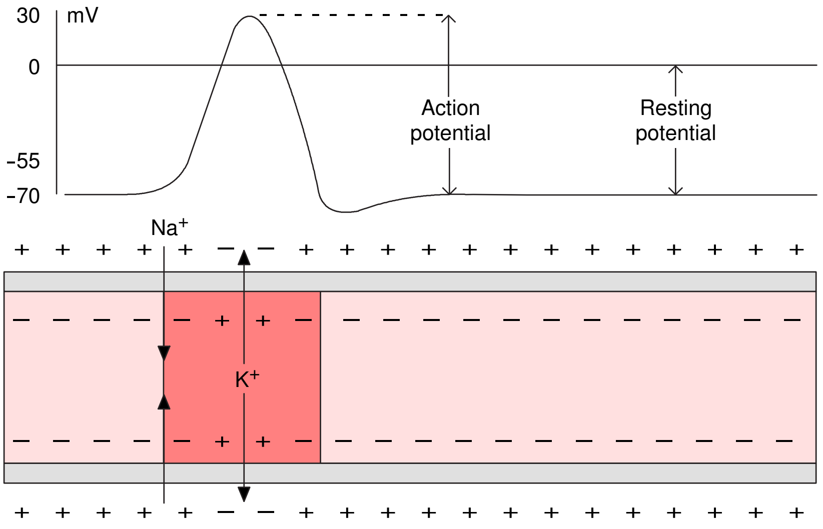


Figure 2.3: *The action potential in a nerve cell is generated by a flow of ions through the cell membrane.*

2.2.3 Action Potential Propagation

As sodium start to flow into the axon during the depolarization phase the concentration of Na^+ in that region will increase dramatically. Since the Na^+, like the K^+, has a positive electrical charge, the ions have a repulsive effect on each other, as on all positively charged particles. Accordingly, the excess of Na^+ causes all free particles with a positive charge to diffuse away in the fluid inside the axon. This flow of positive charges causes a new depolarization of the membrane close to the original depolarization site as described in Figure 2.4. Hence, the action potential has become self-propagating. Analogously, when the membrane opens up for K$^+$ the reversed situation occurs, thus a wave of repolarization will chase the wave of depolarization down the axon.

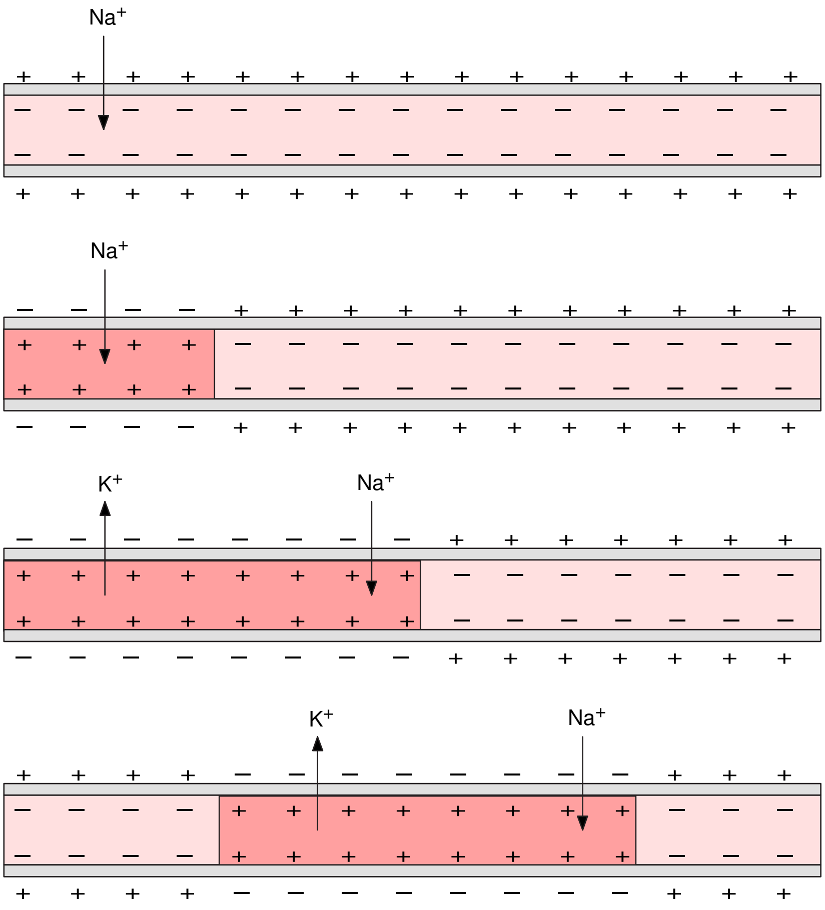


Figure 2.4: *Propagation of the action potential along the axon.*

Action potentials flow in only one direction. This is because they are normally generated at the trigger region, the start of the axon close to the soma.

2.2.4 Saltatory Propagation

The type of impulse propagation considered so far is called *continuous propagation*, reflecting that the potential flows along the axon in a continuous manner with approximately constant velocity. This form of conduction is typically that of muscle fibers or, so called, *unmyelinated* nerve axons. The major part of the axons in the nerve systems, like the one presented in Figure 2.1, are surrounded by a fatty substance called *myelin sheath* that drastically changes the way impulses are transmitted. The myelin acts as a type of electrical insulator and effectively blocks the flow of ions through the cell membranes. At regular intervals the sheath is interrupted by neurofibral nodes called the nodes of Ranvier. At these nodes the concentration of voltage gated ionic channels is very high allowing for a very efficient exchange of Na^+ ions. The direct consequence is that the action potential appears to jump from node to node in a discrete manner. This type of conduction is called saltatory propagation from the Latin *saltare; to jump.*

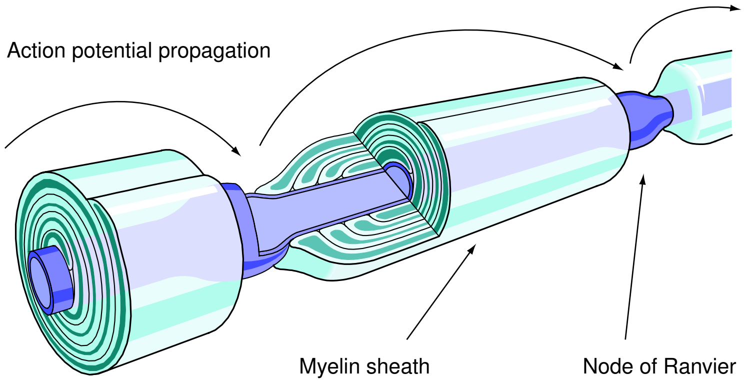


Figure 2.5: *Saltatory propagation.*

Because of the jumping, the propagation velocity of the action potential is very high in myelinated axons, typically 12–120 m/s, as opposed to 0.2–2 m/s in unmyelinated axons.

2.3 The Structure of the Human Brain

Now that we know how the neurons in the brain communicate with each other on a molecular level, we take a great leap in abstraction and focus on the brain as a whole. By examining the drawing presented in Figure 2.6, we can easily distinguish three distinct parts of the human brain: the large, convoluted *cerebrum*, the rippled *cerebellum* and the *brain stem.*

In this thesis, we are mostly interested in the analysis of electrical signals emanating from the cerebrum. We will therefore focus on that part with a special interest in the layer surrounding it, the *cerebral cortex.*

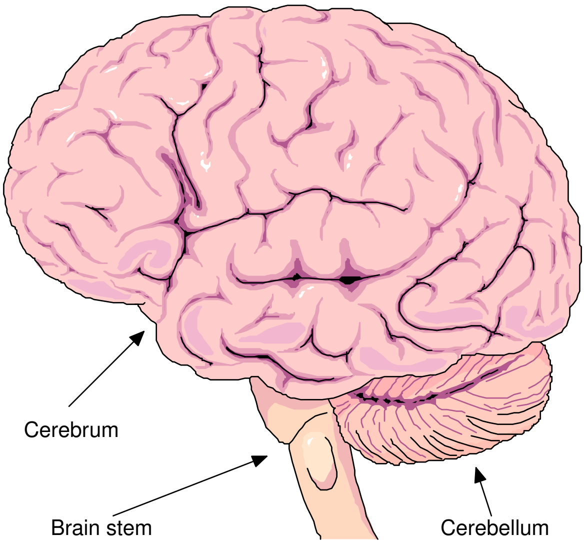


Figure 2.6: *The human brain consists of three different parts.*

2.3.1 The Cerebral Cortex

The cerebrum is divided into two similar structures, the left and the right hemisphere. The left hemisphere senses information from the right side of the body and controls movements on the right side. Analogously, the right hemisphere is connected to the left side of the body. The brain is said to process information *contra-laterally*.

Together, the two hemispheres weigh about 1.4 kg and occupy most of the interior of the skull. Both halves are covered with a thin layer of substance that is extremely densely packed with neurons. Some estimates place the number at more than 100 billion (10^{11}) neurons. Knowing that each neuron can be connected to as many as 10 thousand (10^4) other neurons gives an idea of the complexity of the network. The layer holding this network together is called the *cerebral cortex*. The cerebral cortex has been extensively studied for many years, but due to its complexity, it is far from fully understood. Researchers do agree, however, that the cortex seems to be the center for higher order functions of the brain such as vision, hearing, motion control, sensing and planning. It is also accepted that these functions are *localized* so that different areas of the cortex are responsible for different functions [14]. In fact, since the cortex is only about 5 millimeters thick it can, essentially, be seen as a two-dimensional map of the functions of the brain. Figure 2.7 shows the map with a few important areas marked out.

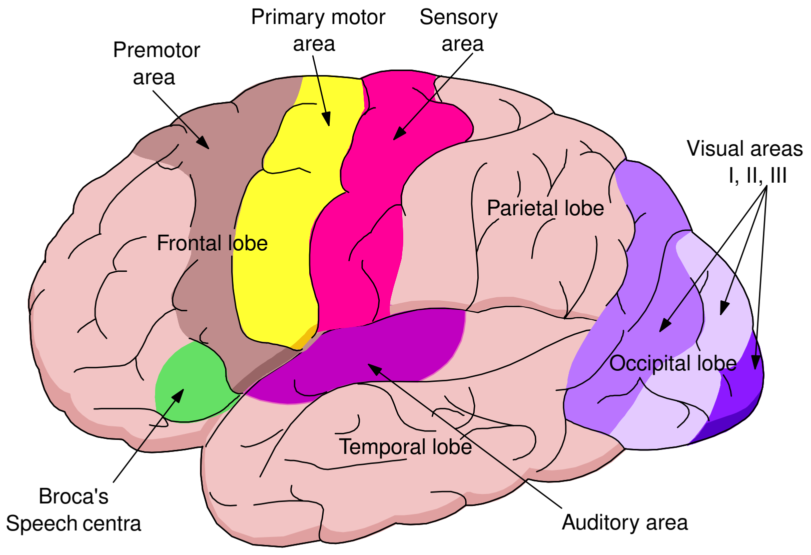
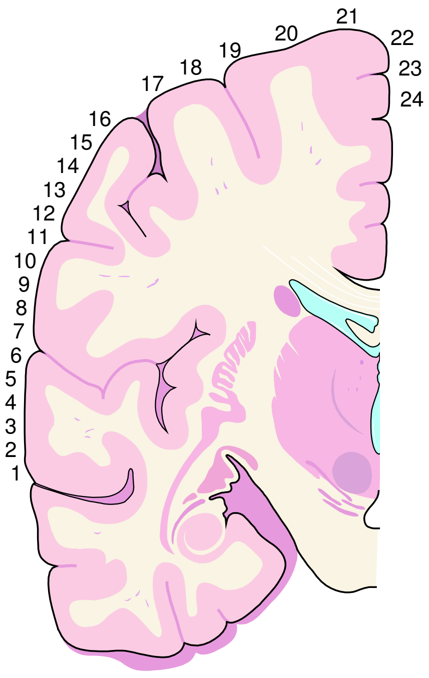


Figure 2.7: *The functionality of different areas of the cerebral cortex.*

2.3.2 The Motor Homunculus

The purpose of this thesis is to analyze the electrical activity in the brain during voluntary movement. Such movements are believed to be initiated in the primary motor area, as shown in Figure 2.7. It is therefore instructive to study that part of the cortex in more detail. It turns out that even the primary motor area can be divided into regions depending on what parts of the body they control. Figure 2.8 is a simplification of the classical *motor homunculus map* drawn in 1950 by Penfield and Rasmussen [16]. They showed that activity of particular parts of the primary motor cortex causes movements of particular parts of the body.

Later, Penfield and Rasmussen also showed that a similar map could be drawn for the sensory area of the cortex. For a designer of a brain–computer interface this is important information, since it gives an indication on where on the scalp the EEG electrodes should be located.



1 –	chewing
2 –	swallowing
3 –	tounge
4 –	teeth, jaw
5 –	speech
6 –	lips
7 –	face
8 –	eyes
9 –	forehead
10 –	neck
11 –	thumb
12 –	index finger
13 –	middle finger
14 –	ring finger
15 –	little finger
16 –	hand
17 –	wrist
18 –	elbow
19 –	shoulder
20 –	trunk
21 –	hip
22 –	knee
23 –	ankle
24 –	toes

Figure 2.8: *The motor homunculus map. (After Penfield and Rasmussen, 1950.)*

Chapter 3

EEG Recording

In the previous chapter, we studied the human brain, the cerebral cortex and how information is processed internally in this important structure. It is now time to see how that information can be extracted and recorded by an external device. Doing that also means taking the first step in the design of our brain–computer interface.

3.1 History of the EEG

The first discovery of electrical potentials generated in the brain was made by the English physician Richard Caton, in 1875. Caton studied the brains of cats and monkeys using electrodes probing directly on the exposed cortexes of the animals. Since there were no electronic amplifiers available at that time, the probes were connected to simple galvanometers with optical magnification. Considering the equipment available and the knowledge of electricity at that time, the results of the experiments were impressive and inspired many researchers to come and work in this new field.

Yet, it was not until more than fifty years later that the first recordings from a human brain were made by the German psychiatrist Hans Berger. In 1929, Berger announced that "it is possible to record the electric currents generated in the brain, without opening the skull, and to depict them graphically on paper". This form of recording was named *electroencephalogram*, EEG. Later Berger also found that the EEG varies with the mental state of the patient. This was a revolutionizing discovery that led to the foundation of a brand new field of medical science, *clinical neurophysiology*.

3.2 The Origin of the EEG Signal

As impressive as the results of Caton and Berger were, none of them could explain where the recorded brain waves actually came from. Today, the advances in neurology have brought us a lot closer to the answer.

The major part of the signals that can be observed in the EEG emanates from neurons in the cerebral cortex. As described in Chapter 2.3, the cortex is a thin layer of densely packed neurons that surrounds the two hemispheres in the cerebrum.

Several types of neurons in the cortex contribute to the EEG, but the most important is the pyramidal cell depicted in Figure 3.1.

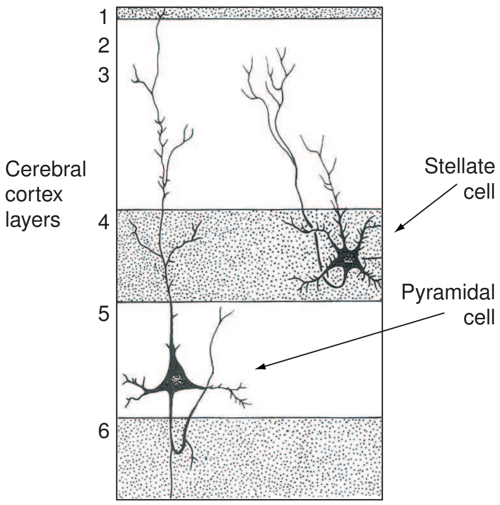


Figure 3.1: *Pyramidal cell in the cortex.*

The pyramidal cell can be recognized by its triangularly shaped cell body and its long parallel dendrites that extend through all layers of the cortex, perpendicular to the surface.[1]

3.2.1 Dipole Potential

When a dendrite of a pyramidal cell is triggered by an axon connected to it, the cell membrane opens up, and a flow of positively charged ions enters the cell. That flow leaves an excess of negative charges in the fluid surrounding the "top of the pyramid". The current that entered the dendrite spreads

[1]Other cells in the cortex usually have star-shaped dendrites. These cells are called *stellate cells*.

down the cell and escapes out of its deeper part creating a positive charge in the extracellular fluid around the “base of the pyramid” as described by Figure 3.2. The process is similar to the generation of the action potential described in Section 2.2.

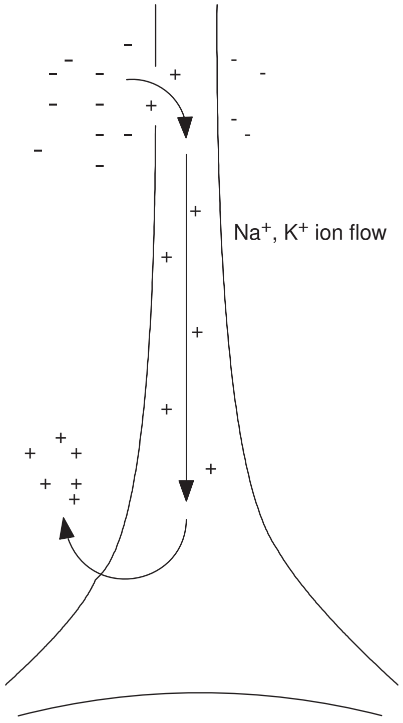


Figure 3.2: *Current flow within and around a triggered pyramidal cell.*

From the outside, the activated cell can be seen as two electrical charges of equal magnitude and opposite signs, separated by a distance, as shown in Figure 3.3. Such system is referred to as an *electric dipole*. From the theory of electromagnetism, we know that the electrical potential that can be sensed from a dipole is a function of the magnitude of its charges, their separation and the distance to the dipole.

For a simple system with only one dipole located in vacuum the potential can be described as

$$V_{\mathbf{p}} = \frac{q\mathbf{d}\cdot\mathbf{r}}{4\pi\varepsilon_0\,|\mathbf{r}|^3} = \frac{qd\cos\theta}{4\pi\varepsilon_0 r^2}. \tag{3.1}$$

This expression is an extreme oversimplification of the relationship between the surface potential and the neurons causing it. The enormous number of

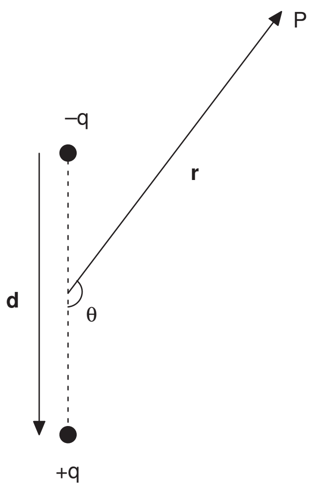


Figure 3.3: *Schematic view of an electric dipole.*

cells contributing to the electrical field, and the intricate mixture of different materials the signal has to penetrate before it reaches the electrode, make the reality far more complex. Still, we can learn a few interesting things from Equation 3.1.

First, we see that the contribution from each dipole drops as the square of the distance from the electrode. In practice, this means that only the neurons located directly underneath the electrode and a few centimeters around it can be detected in EEG.

Further, the cosine term in the nominator indicate that the alignment of the pyramidal cell is important to the result. This is unfortunate because it means that only neurons with dendrites perpendicular to the skull will be measured accurately. As we know, the surface of the cerebrum is highly convoluted with ridges and valleys, much like a walnut. This is a clever biological solution that makes it possible to increase the area of the cortex without increasing the size of the head. In fact, about two-thirds of the cortex are hidden within the valleys between the folds of the cortex. In the case of EEG, however, this is unfortunate because it means that electrical activity of the neurons on the walls of the ridges is undetectable from the surface. The reason is that the dipoles generated by these cells will be parallel, not perpendicular, to the surface of the skull. Hence, the cosine term will be zero. To reach these electrical fields we would have to advance electrodes into the skull.

3.3 Rhythms of the Brain

The neurons in the cerebral cortex are constantly active, and it is therefore possible to observe changes in EEG at any time, even when the subject for example is asleep. In fact, quite often, the amplitude of the recorded EEG is larger at deep sleep than when the subject is fully awake. Why is that? The answer is also somewhat related to Equation 3.1. If we were to evaluate that expression numerically for a typical neuron, we would get a potential contribution from the single cell of a few nanovolts, or less. Such signals cannot be reliably detected by any, today existing, EEG recorder. Hence, the EEG signals that *can* be observed by electrodes on the scalp have to be the sum of many thousands of neurons activated at the same time. The interesting consequence of this is that the magnitude of the EEG signal is more or less independent on the total neural activity in the brain. The important factor is instead how *synchronized* the activity is. At deep sleep the brain is resting, body movement is minimal, eyes are closed, and experiments indicate that most sensory inputs do not even reach the cortex. In this phase when there are no external disturbances, the neurons in the cortex enter a rhythmic state that is believed to be synchronized from the thalamus. That explains why the EEG is larger in amplitude when you are asleep than when you are awake. It is simply just more synchronized. The same phenomenon can be observed in relaxed subjects who open and close their eyes during a recording, as shown in Figure 3.4.

The different rhythms that can be detected in EEG have been shown to correlate with different states of behavior in the subject. The waves are usually categorized based on their frequency content. The first rhythm to be discovered was the alpha rhythm; then followed the beta, theta and delta rhythms. Figure 3.5 illustrates the different rhythms.

3.3.1 Alpha

The alpha waves lie in the frequency spectra between 8 and 13 Hz and usually have an amplitude of about 50 μV. The waves are mostly found in the occipital region, at the back of the head, in people who are awake and resting with their eyes shut. During sleep, the alpha waves disappear completely.

3.3.2 Beta

If a resting person suddenly opens the eyes or engages in some sort of mental activity, like doing arithmetic in the head, the alpha waves normally dis-

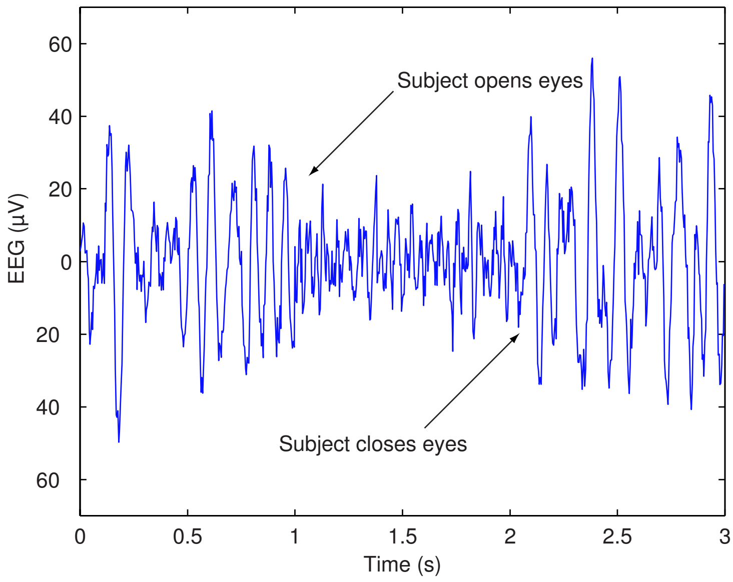


Figure 3.4: *EEG recorded when opening and closing the eyes.*

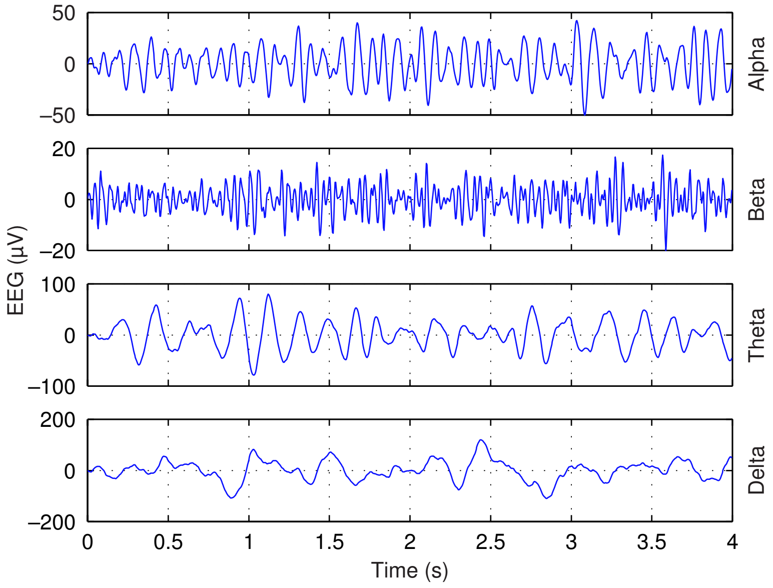


Figure 3.5: *Brain waves are categorized based on their frequency content.*

appear and are replaced by less synchronized waves of higher frequencies, 13–30 Hz, so called beta waves. The beta rhythms have a maximum amplitude of 20 μV and are mostly found in the parietal and frontal regions, see Figure 2.7.

3.3.3 Theta

The theta rhythm, 4–8 Hz, is commonly found in children and adults in light sleep. Sometimes, emotional stress and frustration can trigger periods of theta activity. Most of the waves can be found in the parietal and temporal regions, and their amplitude is usually less than 100 μV.

3.3.4 Delta

The delta waves are associated with deep sleep, but can also be found in the EEG from infants. Delta waves have the lowest frequency 0.5–4 Hz, but also the highest amplitude, sometimes over 100 μV.

3.4 Recording EEG

The principle of recording EEG is actually relatively simple. Two or more electrodes are attached to the scalp of the subject. The electrodes are connected to the inputs of an amplifier, which filters and magnifies the signal. The output from the amplifier is then, either presented on paper by an analog curve writer or sampled and stored for digital processing.

3.4.1 Electrodes

The purpose of an electrode in general is to transfer electrical impulses from a recording site to the input of the recorder. In clinical EEG, the most commonly used electrode type is *surface electrodes* consisting of small metal discs that are applied directly on the scalp of the patient. *Needle electrodes* that are inserted under the skin are not recommended due to the risk of infection. The *clip electrode* is a special form of surface electrode that can be used to detect signals from, for example, the earlobes. Figure 3.6 shows the two types of disc electrodes.

The discs of a surface electrode are usually made of gold or silver, coated with a thin layer of silver chloride, platinum or some other metal that does not interact chemically with the scalp. The diameter of the discs has to be somewhere between 4 and 10 millimeters to yield good electrical and

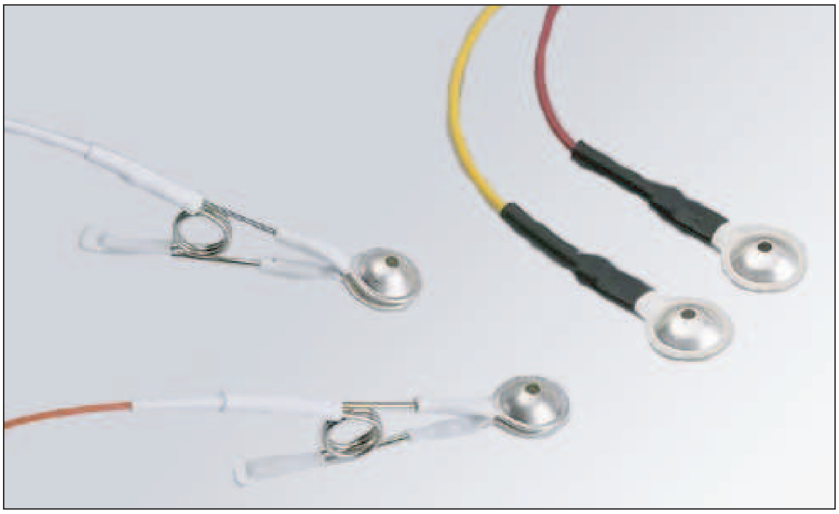


Figure 3.6: *The electrodes most commonly used in clinical EEG are surface electrodes in form of metal discs that are attached directly on the skull of the patient or in form of clips used for recordings from the earlobes.*

mechanical contact. The electrical contact is very important to the results of the recording. If the impedance between the electrode and the skin is too large, the recorded signal is attenuated, and therefore it easily drowns in surrounding noise. As a rule-of-thumb it is desirable to keep the impedance below 10 kΩ. To minimize the electrode impedance it is important to clean the application site on the scalp carefully before applying the electrodes. Lightly scrubbing the skin and wiping it with alcohol usually does the job. To further reduce the impedance several types of contact gels and pastes have been developed that can be applied between the skin and the electrode. The paste increases the conductivity of the skin and helps keeping the electrode in place.

3.4.2 Filters

Almost any EEG amplifier available on the market has a set of filters integrated with the amplifiers. A *high-pass* filter is used to remove DC-components, and a *low-pass* filter removes high frequency noise. Most EEG machines also provide a special *notch-filter* that eliminates frequency components around 50/60 Hz. The notch filter reduces the most common electrical artifact, interference from equipment powered by alternating current.

3.4.3 Amplifiers

As described, the EEG signals that can be detected on the scalp have maximum amplitude of a few hundred microvolts. Consequently, the overall gain of the amplifier has to be very high, typically 10.000 or more. Most EEG amplifiers are so called differential amplifiers where the output is generated by the difference between two inputs that are related to the same reference. This property makes the amplifier less sensitive to noise.

3.5 The 10–20 System for Electrode Placement

Since different regions of the cortex have different functionality, the electrical activity recorded by electrodes on the scalp can vary greatly depending on the position of the electrode. To make it possible to compare recordings made by different researchers and be able to repeat previously made experiments, an international group of neurophysiologists in 1947 set out to develop a standard for the placement of EEG electrodes. Several important design principles were agreed upon.

- The electrode positions should be measured from standard positions on the skull that can be easily detected in any subject, for example the nasion, the point where the nose meets the forehead.
- All parts of the head should be represented with name-given positions.
- The names should be in terms of brain areas instead of just numbers to make it possible for a non-specialist to understand.
- Studies should be made to determine the functionality of the part of the cortex underlying each electrode. The electrode should be named thereafter.

The work on the design of the system was led by Herbert Jasper and was presented at a conference in Paris, 1949 [13]. Jasper named the work "the 10–20 system", and it is now the most widely used standard for EEG electrode placement. The system works as follows.

The positions in the anterior–posterior direction is based on the distance over the center of the scalp between the *nasion*, the root of the nose, and the *inion*, the small protuberance of the skull at the back of the head. Along this line, five points are marked as depicted in Figure 3.7. The first point is called the frontal pole (Fp) and is placed 10% of the nasion–inion distance from the nasion. The following points are named: frontal (F), central (C), parietal (P) and occipital (O), and are positioned 20% of the distance from each other with the F-point 20% away from the Fp-point. That leaves 10% between the O-point and the inion.

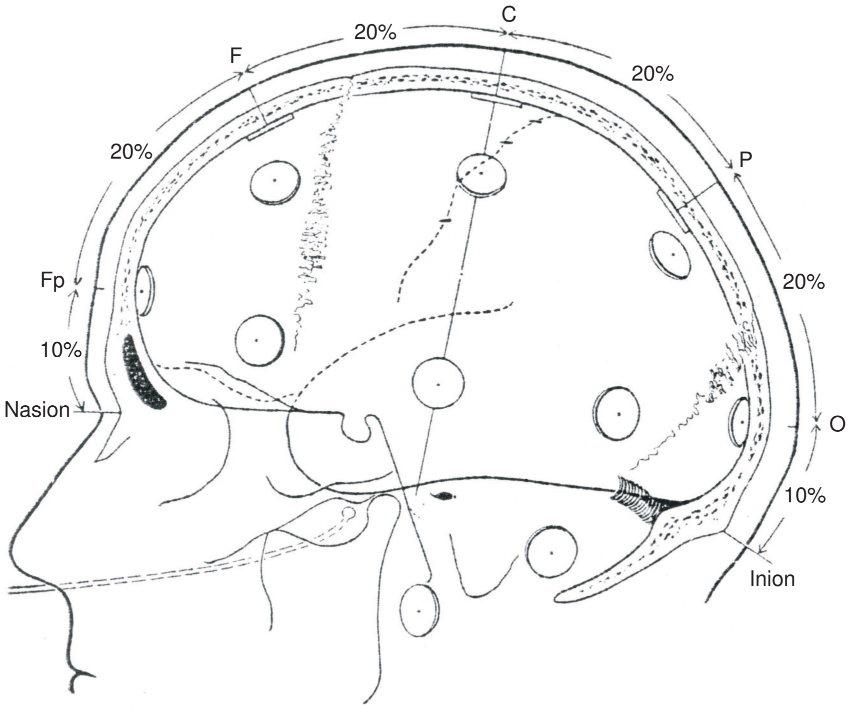


Figure 3.7: *Division of the midline between nasion and inion according to the 10-20 system. (From Jasper, 1958.)*

The measurements in the left–right direction is based on a imagined line between the so called *preauricular points*[2], just in front of the left and the right ear, passing through the previously determined central point on the top of the head. That line is divided in the same 10–20-way as the nasion–inion line, and the five points are named from left to right: T3, C3, Cz, C4, and T4, as illustrated in Figure 3.8.

The following electrodes are placed along two lines between the frontal point and occipital point, passing through the T3 electrode on the left side and the T4 electrode on the right side, as shown in Figure 3.9. As before

[2]The preauricular point can be felt as a small depression at the root of the zygoma just in front of the tragus (the small piece of cartilage near the opening of the auditory canal in the ear).

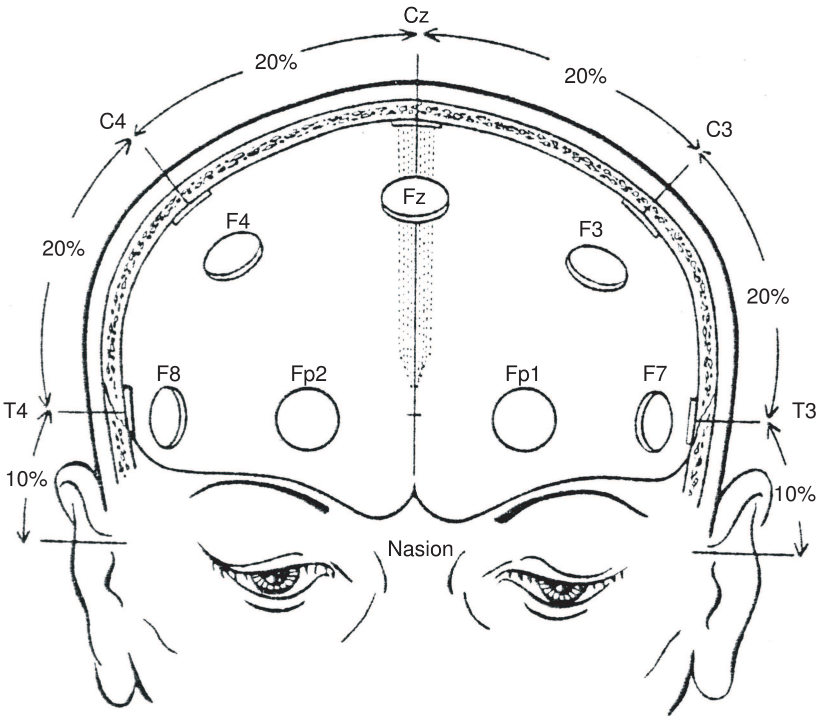


Figure 3.8: *Frontal view of the head showing electrode positions along the central line. (From Jasper, 1958.)*

this line is divided in 10% and 20% sections, and the electrodes placed on these new positions are called O1, T5, T3, F7 and Fp1 on the left side and O2, T6, T4, F8 and Fp2 on the right side.

Next, the previously defined frontal point is assigned an electrode, Fz. Through this point, a line is drawn from F7 to F8. On this line, two new electrodes are placed, F4, equidistant from F8 and Fz, and F3, equidistant from F7 and Fz. Finally, in exactly the same way, the three last electrodes P3, Pz and P4 are positioned between T5 and T6.

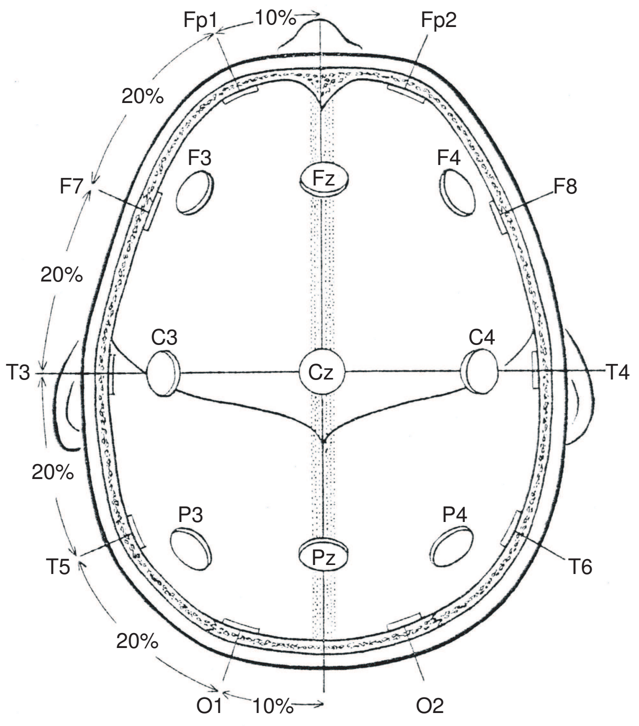


Figure 3.9: *Top view of the skull illustrating electrode positions according to the 10–20 system. (From Jasper, 1958.)*

Chapter 4

Preprocessing of EEG Data

In the introduction of this thesis, we described the general structure of a brain–computer interface as a system composed of three different modules: a *recording module*, a *preprocessor stage* and a *classifier*. The process of recording the electrical activity in the brain has already been discussed. In this chapter, we focus on how to convert, or preprocess, that raw signal to a form that makes it better suited for the classification.

The boundaries between the different stages of a BCI are often fuzzy, and it is not always easy to tell where one stage ends and another begins. For example, the classifier module can act as a part of the preprocessing. Often, but not always, the preprocessor stage performs fixed transformations of the data, while the classifier contains parameters that are adapted through a training process.

4.1 Feature Extraction and Selection

The concept of preprocessing in this context is actually a matter of two operations, feature extraction and feature selection. The former is the process of acquiring data, usually numerical values, about the object to be classified. The operation can be a simple measure of physical properties of the object like length and weight. It can also be more complex like calculating the Fourier transform of a radio signal to find the power in a certain frequency band. Feature selection is the operation of choosing what features to use for classification. As we shall see later, it is desirable to keep the number of features needed for the classification at a minimum. There are several reasons for this. First, the generalization capability of many classifiers is known to deteriorate if the dimension of the input increases above a certain point. This is related to the problem called *the curse of dimensionality* [11]. Second, very large adaptive classifiers may be cumbersome to work with as the training time normally grows very fast with the dimension of the data.

Sometimes feature selection means calculating new patterns by combining two or more selected features. When used this way feature selection can be regarded as a form of feature extraction. As we see, the distinction between selection and extraction is fuzzy, and very often the concepts overlap. In the subsequent text, we will refer to both concepts as feature extraction, or sometimes simply preprocessing.

4.2 Time Series

In its most general form, a brain–computer interface is a system that measures a continuous multidimensional signal and converts it to a symbol, for example, an integer, corresponding to one of a set of classes.

Since the input to the system is a continuous signal, usually an analog voltage, and the processing of the signal is performed by a digital computer, the input has to be converted into a digital form. This operation is called sampling and means that the current value of the input is measured periodically. If the measurements are ordered according to the sample time the set is called a time series and is denoted

$$x\,[n] = (x_0, x_1, x_2, \ldots, x_N)\,. \tag{4.1}$$

The samples $x[n]$ can be scalars or, if two or more channels are sampled simultaneously, vectors of dimension d, where d is the number of channels. Often, the sample period, that is, the time between two measurements is fixed. The sampling is then called uniform. In order not to lose any information contained in the signal due to the sampling, the sample period, denoted T, has to meet the requirement

$$T = \frac{1}{f_s} < \frac{1}{2f_{\max}}. \tag{4.2}$$

The frequency f_s is called the sample frequency, and f_{max} represents the highest frequency component contained in the signal to be sampled. Equation 4.2 is often referred to as the Nyquist-Shannon sampling theorem.

In this thesis, we are dealing primarily with multidimensional EEG signals. As described in Chapter 3, such signals are believed to have most of its information content in the spectrum below 30 Hz. Most studies on brain–computer interfaces therefore focus on that part by filtering out all components above some maximum frequency using a low-pass filter. For all recordings in this project we have set the filter cut-off frequency to $f_c = 30\,\text{Hz}$ and the sample frequency to $f_s = 256\,\text{Hz}$.

4.3 Temporal Signal Processing

As stated, a time series is an ordered set of d-dimensional measurements $x[n]$, where n denotes the position within the sequence. The classification of such series can be done on a sample-by-sample basis by treating each sample individually. Hence, the task is reduced to a classification of a set of static patterns $\{\mathbf{x}_1, \mathbf{x}_2, ..., \mathbf{x}_n\}$ resulting in an equally sized set of labels $\{C_1, C_2, ..., C_n\}$. Usually, we are only interested in one label, the one that classifies the time series as a whole. The produced labels can then be combined by some sort of post processing procedure, for example majority voting to form the overall label. This method does not consider any information about subsequent samples in the series. Therefore, it is mostly useful if the different samples in the time series are uncorrelated.

Generally, when the time series is produced by a physical process, for example EEG, there is significant correlation between the different samples. This information can be very useful in the classification procedure, and just ignoring it as proposed above will most certainly decrease the classification rate of the system [9]. The solution to the problem is to treat the samples not just as points in a d-dimensional space, but as parts of a trajectory, where the points are connected by the order given by the extra dimension time. Hence, the goal of the classification is to label the trajectory instead of the individual points. Figure 4.1 illustrates the difference between non-temporal and temporal classification. For the remainder of this chapter we will discuss temporal preprocessing only.

The purpose of the temporal preprocessing stage is to convert a set of successive samples into a static pattern in a way that preserves as much as possible of the information in the signal. Creating a good preprocessor module is a non-trivial problem and one of the most important parts in the design of a system for time series classification, such as a brain–computer interface. Several important design issues have to be addressed.

4.3.1 Selecting Window Width

One question is how many samples to process at the time. If using too many, the latency of the system, that is, the delay from input to output, may become too large. In a real-time BCI system, controlling for example

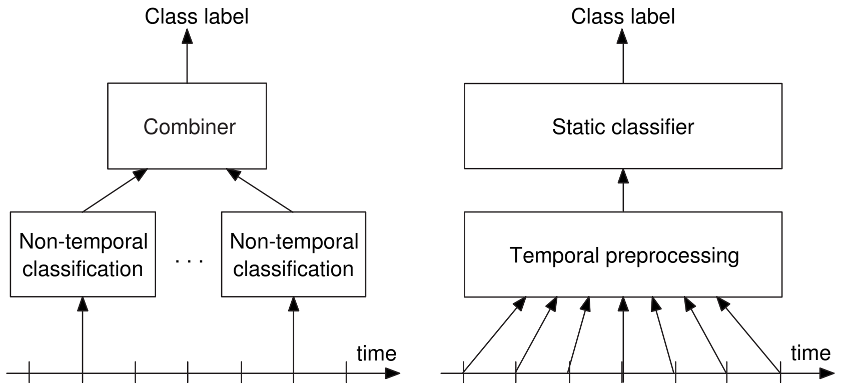


Figure 4.1: *Principles of non-temporal versus temporal preprocessing.*

a wheelchair an input-output delay of one second is probably too much. If making the window too short important information about the relations between samples may be lost resulting in bad classification performance and a high error rate. In the extreme case, only one sample is used. The system then degenerates to a non-temporal classifier. The optimal window width is a compromise that depends on the input signal, the preprocessing method and the classifier.

4.3.2 Linear or Non-linear Models

The dominating way of analyzing physiological signals, like the EEG, is by classical linear modeling. The main reason for this is that linear models have been studied for centuries, and therefore the theory on these methods is comprehensive.

From a signal processing perspective, little is known about the physiological factors generating the EEG. It is accepted that the signals are non-stationary[1], and there are indications that they may be non-linear as well [8], although this has not been conclusively demonstrated.

In [12] Hazarika, Tsoi and Sergejew show that a non-linear preprocessing technique based on Newton-Rhapson iterations can produce better classifications results on specific types of EEG than the corresponding linear model. However, the differences are small, and the method presented has several drawbacks like potential instability and long processing times. Still

[1]A non-stationary signal has statistical properties, e.g. mean value, variance or frequency spectrum that varies over time.

the idea is interesting, and the topic deserves further investigation. In this thesis however, we focus on the more reliable linear methods.

4.4 Time Series Modeling

The procedure of extracting classifiable features from a time series is very closely related to a larger group of problems referred to as *parametric signal modeling*. As we shall see, many techniques from the signal-processing field can be successfully applied to our problem. Parametric modeling deals with the problem of representing a signal, usually a time series, by using only a small number of carefully chosen parameters.

Say, for example that we want to transmit a signal in form of a sine wave across some sort of communication channel. The signal has the form

$$x(t) = A \sin(2\pi f t + \varphi) . \tag{4.3}$$

One way of doing this is of course to sample the signal, with a sample frequency higher than twice the frequency of the wave itself, and then transmit each sample individually. This would be clumsy however and use far more bandwidth than actually needed. A more efficient way would be just to transmit the parameters A, f and φ. The receiver could then reconstruct the signal perfectly, without noise. Figure 4.2 illustrates the concept.

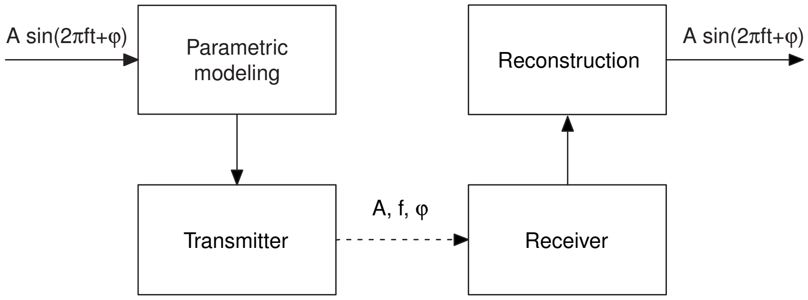


Figure 4.2: *Example of parametric signal modeling.*

4.4.1 Adaptive or Non-adaptive Models

The example above illustrates the principle of non-adaptive parametric modeling, that is, the estimated parameters do not change in time. Because of this, the model requires that the signal is *stationary*. In other words, the statistical properties of the signal like average amplitude and frequency content must not vary from one time to another. If the signal is not stationary, the parameters will have to be adapted as the properties of the input changes. Such models are therefore called adaptive parametric models.

The electroencephalogram is non-stationary in that it has a time varying frequency spectrum. This, however, does not mean that non–adaptive models are useless for EEG modeling. The spectrum varies only slowly so the EEG can in fact be considered stationary over short intervals. In other words if we use segments of recorded EEG shorter than a certain length we can consider the signal in that segment to be stationary and thereby still use the less complex non-adaptive techniques. Pardey *et al* [15], recommend windows *no longer than one second* for non-adaptive modeling.

4.5 Autoregressive Modeling

The by far most common way of modeling time series like the EEG is by fitting a so-called autoregressive model, AR, to the data. Mathematically this means that the sample x_n, at a certain point in time n is described as a linearly weighted sum of the previous p values.

$$\tilde{x}_n = -\sum_{i=1}^{p} a_{pi} x_{n-i} \tag{4.4}$$

The weights a_{pi} are the parameters to be estimated. We realize that if the model was perfect for all n we could use the predicted value $\tilde{x}_n$ together with the $p-1$ most recent values of x to predict the value of $\tilde{x}_{n+1}$. Then, this new value can be used to estimate $\tilde{x}_{n+2}$ and so on, recursively, until the whole time series was defined. It would thus be possible to completely reconstruct the time series x given the coefficients a_{pi} and an appropriate set of starting points $\{x_0, \ldots, x_p\}$. Different starting points correspond to different realizations of the signal. Normally, in an EEG signal classification system we are only interested in the frequency content of the input signal. And, as we soon will see, all that information is contained in the AR coefficients a_{pi}.

In the normal case, p is much shorter than the window length N, and hence determining the coefficients has to be a compromise between the coefficients obtained for each n. The difference between the predicted value $\tilde{x}_n$ and the true value x_n is denoted e_{pn} and is defined by

$$e_{pn} = x_n + \sum_{i=1}^{p} a_{pi} x_{n-i}. \tag{4.5}$$

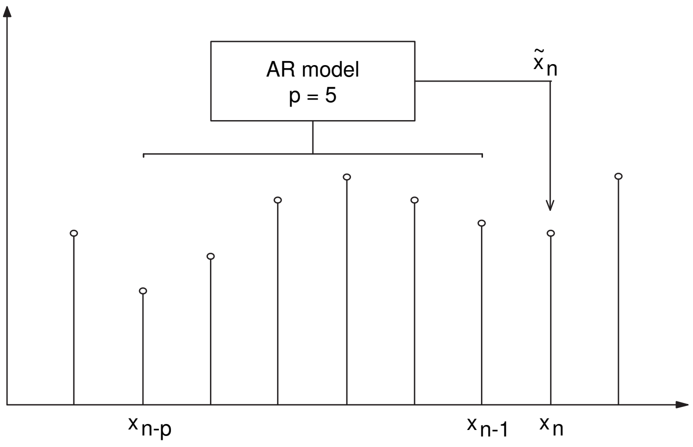


Figure 4.3: *AR model forward prediction.*

The total prediction error can be written as the mean value of the square of the local errors.

$$E_p = \frac{1}{N}\sum_{n=1}^{N} e_{pn}^2 = \frac{1}{N}\sum_{n=1}^{N}\left(x_n + \sum_{i=1}^{p} a_{pi}x_{n-i}\right)^2 \tag{4.6}$$

The best set of AR parameters is taken to be the one that minimizes the total prediction error E_p. Since we know that the optimum for such problem has to be an extreme point, all partial derivatives of E_p with respect to the coefficients has to be zero.

$$\frac{\partial E_p}{\partial a_i} = 0 \quad \text{for } i = 1, 2, \ldots, p \tag{4.7}$$

Applying this constraint to the definition of E_p gives us the following set of equations.

$$\begin{bmatrix} \frac{1}{N}\sum\limits_{n=1}^{N} x_{n-1}x_{n-1} & \cdots & \frac{1}{N}\sum\limits_{n=1}^{N} x_{n-p}x_{n-1} \\ \frac{1}{N}\sum\limits_{n=1}^{N} x_{n-1}x_{n-2} & \cdots & \frac{1}{N}\sum\limits_{n=1}^{N} x_{n-p}x_{n-2} \\ \vdots & \ddots & \vdots \\ \frac{1}{N}\sum\limits_{n=1}^{N} x_{n-1}x_{n-p} & \cdots & \frac{1}{N}\sum\limits_{n=1}^{N} x_{n-p}x_{n-p} \end{bmatrix} \begin{bmatrix} a_{p1} \\ a_{p2} \\ \vdots \\ a_{pp} \end{bmatrix} = - \begin{bmatrix} \frac{1}{N}\sum\limits_{n=1}^{N} x_n x_{n-1} \\ \frac{1}{N}\sum\limits_{n=1}^{N} x_n x_{n-2} \\ \vdots \\ \frac{1}{N}\sum\limits_{n=1}^{N} x_n x_{n-p} \end{bmatrix} \tag{4.8}$$

A more compact way of writing this system is by using so called truncated autocorrelation functions $R_0, \ldots, R_p$ defined by

$$R_{|i-j|} = \frac{1}{N} \sum_{n=1}^{N} x_{n-i} x_{n-j} \quad \text{for } 0 \leq i \leq p,\ 1 \leq j \leq p. \tag{4.9}$$

Substituting 4.9 into 4.8 yields the *Yule-Walker equations.*

$$\begin{bmatrix} R_0 & R_1 & \cdots & R_{p-1} \\ R_1 & R_2 & \cdots & R_{p-2} \\ \vdots & \vdots & \ddots & \vdots \\ R_{p-1} & R_{p-2} & \cdots & R_p \end{bmatrix} \begin{bmatrix} a_{p1} \\ a_{p2} \\ \vdots \\ a_{pp} \end{bmatrix} = - \begin{bmatrix} R_1 \\ R_2 \\ \vdots \\ R_p \end{bmatrix} \tag{4.10}$$

The method of calculating the AR coefficients from 4.10 is termed the *autocorrelation method* and can be efficiently implemented using the recursive *Levinson-Durbin* algorithm [15].

4.5.1 Backward Prediction

Just as the AR coefficients can be used to predict the next value of a time series, the same set of parameters can be used to retrospectively predict past values in terms of future values. Figure 4.4 illustrates the concept.

Consequently, there are two different ways of estimating the optimal AR coefficients, using normal and reversed time. Generally, they do not give the same results even if the underlying time series is the same.

A natural idea is therefore to modify the forward prediction method described above to try to make it optimal even for the reversed prediction. We name the prediction error defined in 4.5 *forward prediction error* and define a similar term for the *backward prediction error.*

$$b_{pn} = x_{n-p} + \sum_{i=1}^{p} a_{pi} x_{n-p+i} \tag{4.11}$$

It can be shown [15] that the forward and backward prediction errors are related in a recursive way.

$$\begin{aligned} e_{pn} &= e_{(p-1)n} + a_{pp} b_{(p-1)(n-1)} \\ b_{pn} &= b_{(p-1)(n-1)} + a_{pp} e_{(p-1)n} \end{aligned} \tag{4.12}$$

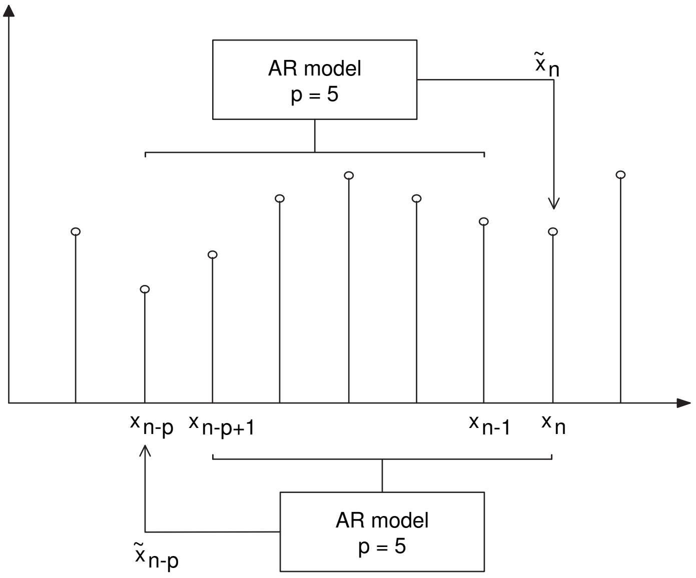


Figure 4.4: *AR model forward and backward prediction.*

We rewrite the total prediction error to account for the backward prediction error as well.

$$E_p = \frac{1}{2\left(N-p\right)} \sum_{n=p+1}^{N} \left(e_{pn}^2 + b_{pn}^2\right) \tag{4.13}$$

Differentiating and setting all partial derivatives to zero gives a set of equations just like 4.10 that can be solved for a_{pi}. The most efficient way of solving the system is by using the recursivity defined by the error terms in 4.12 together with another important property of the autoregressive coefficients.

$$a_{mi} = a_{(m-1)i} + a_{mm} a_{(m-1)(m-i)} \quad \text{for } \; 1 \leq i \leq m-1 \tag{4.14}$$

This relationship is termed *Levinson recursion*, and it means that if we know all coefficients for a p–1 order model together with the last coefficient of order p, the other parameters a_{ip} can be determined. In other words if we know a_{11} and a_{22} we can calculate a_{21}, and then if we know a_{33}, we can determine a_{31} and a_{32} etc. This is important because it leads up to a very efficient way of recursively determine all AR coefficients. So all we need is a way of determining a_{pp} based on the coefficients of models of order p–1.

Substituting 4.12 into 4.13 differentiating yields

$$\begin{aligned} E_p &= \frac{\sum\limits_{n=p+1}^{N}\left(\left[e_{(p-1)n}+a_{pp}b_{(p-1)(n-1)}\right]^2+\left[b_{(p-1)(n-1)}+a_{pp}e_{(p-1)n}\right]^2\right)}{2(N-p)} \\ \frac{\partial E_p}{\partial a_{pp}} &= \frac{\left[\sum\limits_{n=p+1}^{N}\left(2e_{(p-1)n}b_{(p-1)(n-1)}\right)+a_{pp}\sum\limits_{n=p+1}^{N}\left(b^2_{(p-1)(n-1)}+e^2_{(p-1)n}\right)\right]}{2(N-p)} \end{aligned} \tag{4.15}$$

We set the partial derivative to zero and solve for a_{pp}

$$a_{pp} = \frac{-2\sum\limits_{n=p+1}^{N}\left(e_{(p-1)n}b_{(p-1)(n-1)}\right)}{\sum\limits_{n=p+1}^{N}\left(b^2_{(p-1)(n-1)}+e^2_{(p-1)n}\right)} \tag{4.16}$$

We now have all the ingredients we need to recursively calculate all AR coefficients. For $p = 0$, no prediction takes place so both the forward and backward prediction error equals the true value of that sample. This gives us a starting point for the algorithm

$$\begin{aligned} e_{0n} &= x_n \quad \text{for } 1 \le n \le N \\ b_{0n} &= x_n \quad \text{for } 1 \le n \le N \end{aligned} \tag{4.17}$$

Equation 4.12, 4.16 and 4.17 shows how to extend the estimation to calculate any coefficient. So, for m = 1,2...,p we simply repeat the following steps.

$$\begin{aligned} a_{mm} &= \frac{-2\sum\limits_{n=m+1}^{N} b_{(m-1)(n-1)}e_{(m-1)n}}{\sum\limits_{n=m+1}^{N}\left[b^2_{(m-1)(n-1)}+e^2_{(m-1)n}\right]} \\ a_{mi} &= a_{(m-1)i} + a_{mm}a_{(m-1)(m-i)} \quad \text{for } 1 \le i \le m-1 \\ e_{mn} &= e_{(m-1)n} + a_{mm}b_{(m-1)(n-1)} \quad \text{for } 1 \le n \le N-m \\ b_{mn} &= b_{(m-1)(n-1)} + a_{mm}e_{(m-1)n} \quad \text{for } 1 \le n \le N-m \end{aligned} \tag{4.18}$$

The procedure described by 4.17 and 4.18 is called *Burg's algorithm* and was formulated by John Parker Burg in 1960. It is very efficient. We note that the loop in 4.18 will be repeated p times and that the number of operations each time is on the order N. Hence, the complete algorithm require on the order of Np operations. This should be compared to solving the equations 4.10 by, for example, Gaussian elimination that require Np^2 operations to set up the equations and then p^3 operations to solve them.

What makes the Burg method so attractive is not only its efficiency, but also the fact that the so-called *reflection coefficients* are bounded by one according to the Cauchy-Schwartz inequality.

$$|a_{mm}| = \frac{2\left|\sum\limits_{n=m+1}^{N} b_{(m-1)(n-1)}e_{(m-1)n}\right|}{\sum\limits_{n=m+1}^{N}\left[b^2_{(m-1)(n-1)} + e^2_{(m-1)n}\right]} = \frac{2\left|\mathbf{b}^T_{m-1}\mathbf{e}_{m-1}\right|}{\left|\mathbf{b}_{m-1}\right|^2 + \left|\mathbf{e}_{m-1}\right|^2} \leq 1 \quad (4.19)$$

This makes the model numerically stable. For a more thorough discussion about model stability, see [10].

4.5.2 Frequency Representation of the AR Model

We said before that, all of the interesting frequency content in the input signal x is embedded in the AR coefficients, if the model is good enough. To understand this we start by repeating the actual definition of the AR model with the forward prediction error included

$$x_n = -\sum_{i=1}^{p} a_{pi}x_{n-i} + e_{pn} \quad (4.20)$$

In terms of digital signal processing, Equation 4.20 defines an infinite impulse response filter, IIR, fed by the error term e_p. Figure 4.5 illustrates a schematic realization of a third order IIR-filter corresponding to an AR model with $p = 3$. Rearranging the recurrence relation 4.20 and taking the z-transform gives the transfer function, $H(z)$, for the IIR-filter.

$$e_{pn} = x_n + \sum_{i=1}^{p} a_{pi}x_{n-i} \Leftrightarrow E(z) = \left(1 + \sum_{i=1}^{p} a_{pi}z^{-i}\right) X(z) \quad (4.21)$$

$$H(z) = \frac{1}{1 + \sum\limits_{i=1}^{p} a_{pi}z^{-i}} \quad (4.22)$$

The z-transform of the output signal is thus determined by the ratio

$$X(z) = H(z)E(z) = \frac{1}{1 + \sum\limits_{i=1}^{p} a_{pi}z^{-i}} E(z). \quad (4.23)$$

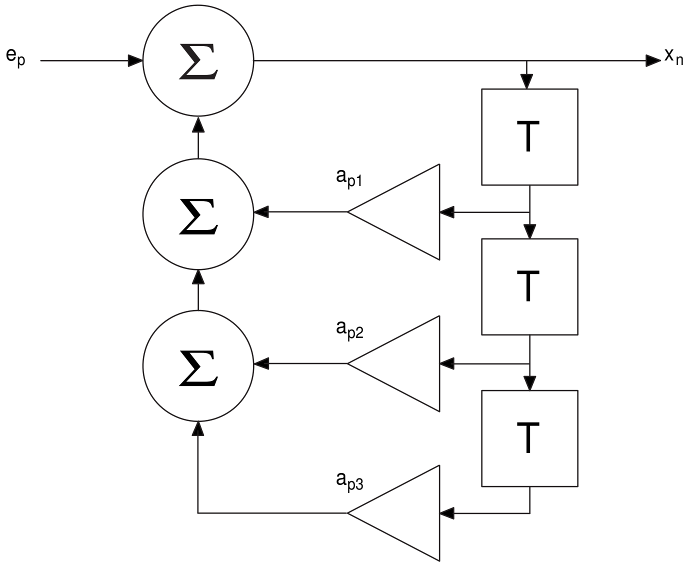


Figure 4.5: *Third order AR model represented as an IIR filter.*

As we know the frequency spectrum of the output signal is determined by its Fourier transform. In this case the Fourier transform is actually the z-transform restricted to the unit circle in the z-plane, $\mathrm{z} = \mathrm{e}^{j2\pi fT}$, where T is the sample period. In practice if the AR model is good, the error term $e(n)$ can be approximated with a white noise sequence with a constant frequency spectrum. Consequently, the power spectrum of the output signal depend solely on the AR coefficients

$$P\left(f\right) = \frac{K}{\left(1 + \sum\limits_{i=1}^{p} a_{pi} e^{-i2\pi jfT}\right)\left(1 + \sum\limits_{i=1}^{p} a_{pi} e^{+i2\pi jfT}\right)} \tag{4.24}$$

where K is some positive real constant. In other words, a good autoregressive model of a signal effectively summarizes everything there is to know about its frequency content. That is exactly what we are interested in here.

4.6 Lagged Autoregressive Models

We note that a factorization of the denominator in 4.24 gives a total number of $2p$ zeros. Since the coefficients in the polynomials are real, the roots are either real or complex conjugated. Consequently, both factors in the denominator in the rightmost part of 4.24 have the same p roots. At the most, $p/2$ of these roots are located in the upper half of the unit circle, that is, in the frequency range

$$0 \leq f \leq \frac{1}{2T} \tag{4.25}$$

which is the part of the spectra we are interested in. This shows that an autoregressive model of order p can pick up to p/2 distinct frequency peaks of the signal and represent frequencies up to $f = f_s/2$.

Sometimes it is necessary to use a sample frequency that is significantly higher than twice the maximum frequency component. The reason for this might be that the recorder only has fixed sample periods or that the built in band pass filter has bad performance. Using a higher sample frequency increases the width of the frequency spectrum, but it also reduces the resolution, and therefore it is known to make the model more sensitive to noise [18]. The solution to this problem is to down sample the signal by a factor, L, before applying the Burg algorithm and thus reduce the effective sample frequency by the same factor. Of course, the sequence also has to be low pass filtered before down sampling to avoid aliasing. Remember the Nyquist-Shannon sampling theorem. Effectively this technique means that the taps of the autoregressive model are separated by L samples. The model is said to have a lag of L, and the technique is referred to as lagged autoregressive modeling, LAR. This is the technique we will be using for the rest of this project.

Chapter 5

Classification with Neural Networks

The last and arguably most important part of a BCI system is the classifier module. The purpose of the classifier is to sort sequences of data from the preprocessor into distinct categories. As described in the thesis introduction, the recorder and the preprocessor typically perform static transformations while the classifier often includes adaptive self-learning software that is taught to produce the correct classifications based on a set of training examples.

Several paradigms of adaptive software have been developed, as parts of research in artificial intelligence, AI. One of the most successful for classification tasks is so called *artificial neural networks*, ANN, a paradigm that is related to biological networks and tries to mimic the structure of the human brain. This chapter contains a brief discussion about the history behind the evolution of ANNs and a detailed description of the type of network used in this project, the *multilayer perceptron*.

5.1 What are Neural Networks?

Neural Networks, or artificial neural networks to be precise, represent a technology with many applications. Therefore, it has attracted researchers from a wide variety of disciplines like computer science, psychology, physics and neuroscience. The work on ANNs has been inspired by the way biological nervous systems, such as the human brain, process information. The ultimate aim is to design a computer paradigm that can handle problems which the biological brain solves easily, but where conventional computers usually do a poor job. Vision, speech recognition and the ability to learn from training data are only a couple of examples.

5.2 Biological versus Artificial Neural Nets

The human brain, as described in Chapter 2, is one of the most complicated things man has ever studied, and it is still understood only on a very basic level. In computer science terms, the brain is a highly complex, non-linear, parallel computer with the capability to organize itself to adapt to its environment. The approach of most research on neural networks is to try to capture the low-level structure of the brain and apply it to computer systems. Haykin [11] suggests the following definition of a neural network originally proposed by Aleksander and Morton (1990).

Definition 5.1 *"A neural network is a massively parallel, distributed processor made up of simple processing units, which has a natural propensity for storing experiential knowledge and making it available for use. It resembles the brain in two respects:*

1. *Knowledge is acquired by the network from its environment through a learning process.*
2. *Interneuron connection strengths, known as synaptic weights are used to store the acquired knowledge."*

5.3 Properties of Neural Networks

Different types of artificial neural networks have different structures and, therefore, different advantages and disadvantages. Still some important properties are common to most neural network paradigms.

5.3.1 Learning

One of the most important properties of neural networks is their ability to learn from examples, that is, learn to produce a certain output when presented with a certain input. The learning process involves modification of the internal parameters in the net, the connection weights, to make its overall behavior correspond to a desired behavior defined by a set of training examples. Each example in the training set consists of an input pattern and a desired output pattern. To train the network we pick an example from the training set, feed it to the network and see what output it produces. If the output is not what we expected, we modify the internal weights of the network according to some training algorithm, so as to minimize the difference between the desired and the actual output. The training is then continued with another training example and so on, until the network has reached steady state where no more significant changes to the weights are made, and hopefully the system produces correct outputs for all examples in the training set.

5.3.2 Generalization

The capability to learn is one of the main reasons why neural networks have become popular in so many different disciplines. Even if we have little or no knowledge about the actual process producing the input–output mapping we can still learn it from examples.

More important than the actual learning, however, is the network's capability to generalize, that is, to produce good outputs even for inputs not encountered during training. We will come back to the topic of generalization performance in the last section of this chapter.

5.3.3 Non-linearity

Mathematically a neural network defines a mapping from an input space to an output space. This mapping can be described as a vector-valued function $\mathbf{y} = \mathbf{f}(\mathbf{x})$ that transforms the input vector $\mathbf{x}$ to an output vector $\mathbf{y}$. Both $\mathbf{x}$ and $\mathbf{y}$ can be of any dimensionality. The mapping $\mathbf{f}$ itself is a combination of mappings performed in parallel by simpler units, the neurons. The information processing in each neuron is non-linear, and hence the resulting mapping is non-linear. The property of non-linearity has proven to be very important, especially if the physical mechanism that generates the input signal is non-linear.

5.3.4 Fault Tolerance

When we talk about robust- and fault tolerant systems, we usually mean systems that keep on functioning even if parts of them stop working. The human brain is a beautiful example of a fault tolerant system. Every day a number of neurons in the brain die off, as a part of the natural course of events. Nevertheless, the brain as a whole keeps on working just as if nothing has happened. Only if the damage is severe, changes in the performance can be noticed. Even then, the performance falls gradually to a lower level. It does not drop to zero and start producing nonsense outputs. This is known as *graceful degradation*. The good property of fault tolerance in the brain is much due to its massively parallel, distributed structure. Since artificial neural networks try to mimic that structure, they also inherit a lot of the robustness the brain comprises.

5.4 Layered Networks

The perhaps most well known form of Neural Networks is the multilayer perceptron, MLP. Its structure was proposed by Rumelhart and McClelland in 1986, and it is now the most widely used of all kinds of ANNs. The multilayer perceptron consists of a number of neurons organized in what the authors describe as layers. It has an input layer, an output layer and in between one or more hidden layers as shown in Figure 5.1.

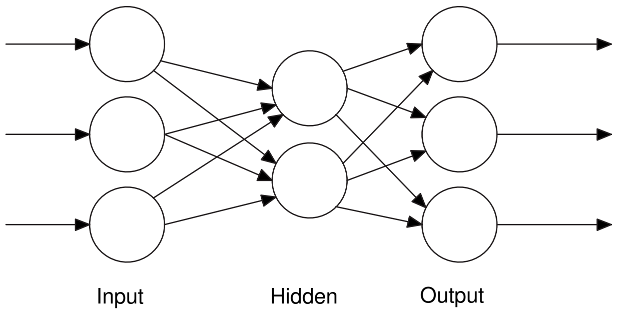


Figure 5.1: *The structure of a multilayer perceptron.*

The perceptron is an example of a feedforward neural network. By this, we mean that the signals are always passed forward through the network, from the input layer to the output layer, via the hidden layers. This ensures stability in the system. We will discuss this further in the following sections along with a very common training algorithm. But, before we go into detail, we should take a brief look at the building blocks that make up the network, the perceptrons.

5.4.1 Perceptrons

The concept of perceptrons was first introduced by Rosenblatt, 1962. His work was based on earlier models of the biological neuron by McCulloch and Pitts from the mid-forties. A single perceptron is a processing element with a number of inputs and one output. The output is usually defined to be a function of the weighted sum of the inputs. Mathematically we can express this as

$$y = f\left(\sum_{i=1}^{N} w_i x_i - w_0\right) = f\left(\sum_{i=0}^{N} w_i x_i\right) = f(z) \tag{5.1}$$

where y is the output term, x_i is the ith of N inputs and w_i its associated weight. In the last step we have included the term w_0 in the sum by setting $x_0 = -1$. The function, f, is always some sort of monotone threshold function for example the Heaviside function, $H(z)$, or the logistic sigmoid, $\varphi(z)$.

$$H(z) = \begin{cases} 1 & z \geq 0 \\ 0 & z < 0 \end{cases} \qquad \varphi(z) = \frac{1}{1 + e^{-\beta z}} \tag{5.2}$$

Both functions share the property that their outputs tend to zero for negative inputs and to unity for positive ones. This means that the function in 5.1 will switch from zero to one when the weighted sum of the non-constant inputs becomes higher than w_0. The term w_0 is therefore called threshold value.

5.4.2 Example

So, how can we use this to produce something useful? Well, say for example that we have a perceptron with two binary inputs and two weights $w_1 = 0.5$ and $w_2 = -0.5$. Further, we set the output function to be the Heaviside function and the threshold to $w_0 = 0.25$. This will give us the simple network presented in Figure 5.2.

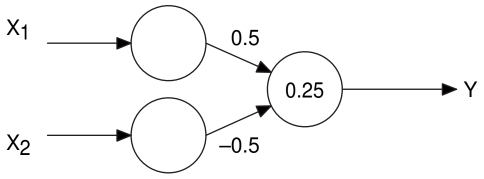


Figure 5.2: *Rosenblatt single layer perceptron network.*

If we present this network with the inputs $x_1 = 1$ and $x_2 = 0$ we get the output

$$y = H(0.5 \cdot 1 + (-0.5) \cdot 0 - 0.25) = H(0.25) = 1 \tag{5.3}$$

whilst if we change any of the input values we will get $y = 0$. In other words, we have constructed a network that knows how to compute the

Boolean function

$$y = x_1 \wedge \neg x_2. \tag{5.4}$$

Now, say that we change the second weight to $w_2 = 0.5$. The network will then produce the output $y = 0$, if and only if both inputs are zero. That is, we have altered the behavior of the net to produce, not the expression in 5.4, but a Boolean OR.

This example might seem trivial, but it shows something important: we can change the behavior of our network simply by varying the weights. This is fundamental to all forms of neural networks and the principle learning is based upon.

5.5 Learning in Single Layer Networks

So far, we have seen how a network can produce useful outputs when the weights have been preset to proper values. This is good, but normally we expect a lot more from a neural network. We expect it to be able to learn.

The perhaps most straightforward way of learning is learning by mistakes. That is, if we present the network with a certain input and it produces an erroneous output we want it to adjust internally to reduce the risk of making that same mistake again. We then hope that the next time we present the same or similar input the network will respond correctly. In this form of training we have to guide the network through the process by providing the right answers to all presented inputs. We therefore call it *supervised learning*.

There are several algorithms for supervised training of a single layer network. Most methods, however, follow the same simple principles. If an incorrect output is encountered during training, the weights of the inputs that contributed to that output are adjusted in a direction towards a correct solution. The weight changes are usually taken to be proportional to the corresponding input. If the network responds correctly, no changes are made at all. Hence, given a set of examples in the form of input–output pairs, we can train the net by simply presenting one input at the time, adjust the weights if necessary, and then go on with the next input. One loop through the whole set of examples is called an *epoch*. It can be proven that the network will converge towards a solution – if a solution exists [11, 23].

5.5.1 Linear Separability

The examples we have seen so far all belong to the class of pattern classification problems. We have a set of inputs and one output. For some of the inputs we want the network to produce a certain output, for the others, another output. We want the network to classify the input patterns in two distinct categories.

One serious limitation to the single layer perceptron is that it is unable to solve classification problems where the classes are not *linearly separable.* To understand this we express the inputs and weights as vectors and make use of some theorems from linear algebra. If a perceptron has N inputs we can describe them as an $(N+1)$-dimensional vector $\mathbf{x} = (-1, x_1, x_2, \ldots, x_N)^T$ and the associated weights as another $(N+1)$-dimensional vector $\mathbf{w} = (w_0, w_1, \ldots, w_N)^T$. With this terminology, the sum in 5.1 can be written compactly as the matrix product $\mathbf{x}^T\mathbf{w}$. We now see that the matter of classifying the input vector $\mathbf{x}$ comes down to finding out whether or not $\mathbf{x}^T\mathbf{w}$ is positive or negative. The decision boundary can be expressed by the linear equation

$$\mathbf{x}^T\mathbf{w} = 0 \tag{5.5}$$

which in terms of linear algebra defines an N-dimensional hyperplane. This means that a two-dimensional pattern space will be divided by a straight line, a three-dimensional space is partitioned by a plane etc.

5.5.2 The XOR Problem

However, if we try to train a single layer perceptron to learn the Boolean function XOR we will quickly run into trouble. The reason for this is that the two classes, $\mathbf{x}^T\mathbf{w} < 0$ and $\mathbf{x}^T\mathbf{w} > 0$, are not linearly separable. Learning this problem corresponds solving the set of simultaneous equations

$$\begin{cases} 0 \cdot w_1 + 0 \cdot w_2 < w_0 \\ 0 \cdot w_1 + 1 \cdot w_2 > w_0 \\ 1 \cdot w_1 + 0 \cdot w_2 > w_0 \\ 1 \cdot w_1 + 1 \cdot w_2 < w_0 \end{cases} \tag{5.6}$$

that obviously has no solution. It is impossible to determine w_1 and w_2, that is, to draw a straight line trough the two-dimensional pattern space, so that all four inequalities are satisfied. Figure 5.3 illustrates the problem.

5.6 The Multilayer Perceptron

The XOR-problem clearly demonstrates the shortcomings of the single layer perceptron. Unfortunately, this drawback, when first proven by Minsky and

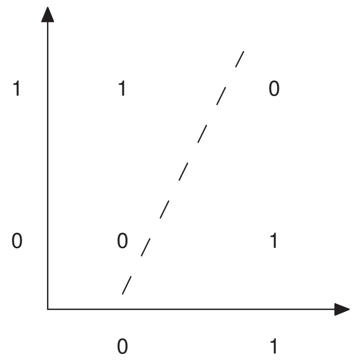


Figure 5.3: *The XOR problem of linear separability.*

Papert in their influential book Perceptrons from 1969, killed a lot of the interest in neural systems. It was not until almost twenty years later when Rumelhart and McClelland proposed their idea of the multilayer perceptron that the area caught attention again.

The multilayer perceptron does not suffer from the linear separability problem. In fact it can be proven that a three-layer network is sufficient to solve any classification problem provided the number of nodes is high enough. This important finding is often referred to as the Kolmogorov theorem. We will discuss this further in the next section, but first we have to straighten out a few questions regarding the structure of the multilayer perceptron.

When we talk about a three-layer network in this context, we mean a network with three layers of variable weights, three active layers. This might be a bit confusing, since most multilayer networks also contain a separate input layer that has fixed weights and only one input per node. Hence, this input layer does not contribute to the training of the network; it just passes its input on to the next layer, as shown in Figure 5.4. To be consistent with the terminology for the single layer perceptron we shall still refer to this as a three-layer network.

To summarize, a three-layer network has three active layers: one output layer and two hidden layers. Each node in these layers correspond to the Rosenblatt perceptron, that is, their output is the weighted sum of their inputs passed through some sort of activation function, normally the logistic function. In addition to these layers, the network also contains an input layer that does nothing but distributes the inputs to the other layers.

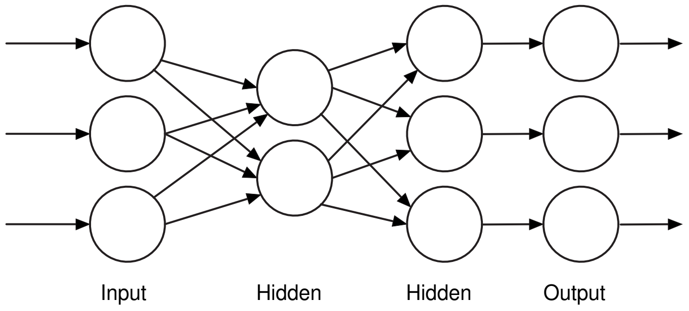


Figure 5.4: *A three-layer perceptron network; one input layer and three active layers.*

5.7 Learning in Multilayer Networks

As for the Rosenblatt perceptron many different algorithms for training multilayer feedforward networks have been developed. Here we focus on one of them, the backpropagation algorithm or delta rule. However, before we describe the actual training algorithm we shall try to state mathematically what we mean by learning.

For a perfectly trained network, that always gives the desired output no matter the input pattern, we can say that the error is zero. For any other net, we need to define a function that estimates how far from a perfect solution we are. We call this an error function and denote it E.

The error function is hence a real-valued function that, for a fixed input, measures the distance between the actual output, y, and the desired output, d, for all active nodes in the network. An intuitive way of defining such function is to use the Euclidean distance

$$E = \frac{1}{2}\sum_{i=1}^{N}(y_i - d_i)^2. \tag{5.7}$$

We see that E is always positive, as we expected, so our aim is to achieve a situation where $E \rightarrow 0$. If we again think in terms of linear algebra the actual outputs and the desired outputs can be arranged as N-dimensional vectors, $\mathbf{y}$ and $\mathbf{d}$, and the error function can be written as

$$E = \frac{1}{2}(\mathbf{y} - \mathbf{d})^T(\mathbf{y} - \mathbf{d}) \tag{5.8}$$

Hence, the problem of training the network is nothing but a multi-dimensional, non-linear and unconstrained optimization problem.

5.7.1 Steepest Descent Algorithm

Standard procedure for solving problems like this is by some sort of iterative search algorithm. Our object function is the error, E, and the variables are of course the weights in the active layers $\mathbf{w}$. For a two-dimensional problem, that is a network with only two variable weights, we can visualize the error function as an energy landscape, like the one in Figure 5.5.

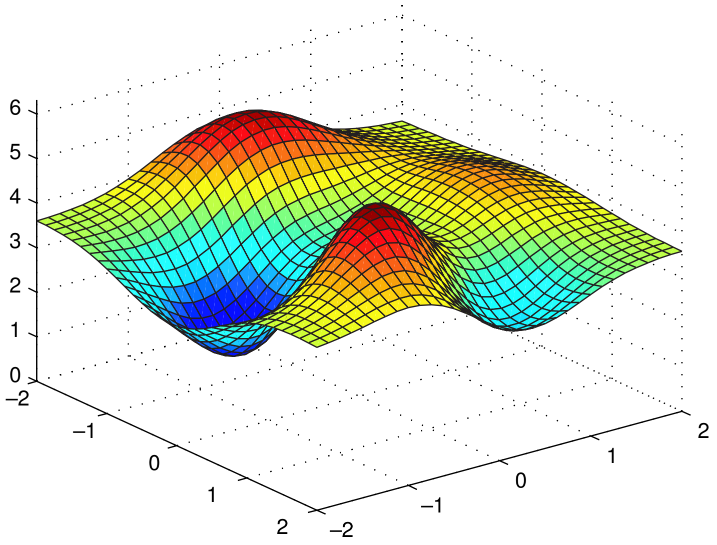


Figure 5.5: *Error function of a two-dimensional problem.*

The idea behind most search algorithms is to start in some initial point in the landscape and then repeatedly work the way downward by taking small steps in well-chosen directions.

To find these directions we start by making a first order Taylor expansion of E around some initial point $\mathbf{w}_0$ and se how we can minimize that linear function instead.

$$E(\mathbf{w}) = E(\mathbf{w}_0) + \nabla E(\mathbf{w}_0)^T(\mathbf{w} - \mathbf{w}_0) \tag{5.9}$$

The first term in the approximation is a constant so we cannot do much about that. The second term is in fact a dot product, which has its minimum when the two vectors are parallel but with opposite signs. That is,

$$\mathbf{w} = \mathbf{w}_0 - \eta \nabla E(\mathbf{w}_0) \tag{5.10}$$

where η is some small constant. Consequently, we are ensured to decrease the value of the error function by adjusting the weights according to 5.10, provided the gradient is non-zero, and the step-length, η, is small enough. With this done, we simply recalculate 5.10 from the new point and so on, until the solution is good enough. This method is to mathematicians known as *the steepest descent algorithm.*

5.8 The Backpropagation Rule

Now that we have a method of adjusting the weights, all we have to do is to express 5.10 in terms of single weights. As the vector ∇E is a collection of partial derivatives arranged as a column vector, focusing on a single weight in the system corresponds to extracting one row from the weight update formula.

$$w_{ij} \leftarrow w_{ij} - \eta \frac{\partial E}{\partial w_{ij}} \tag{5.11}$$

This will be the update rule for the weight that connects node i and node j. We recall that the output from node i is

$$y_i = f\left(\sum_{k\to i} w_{ki} y_k\right) = f(z_i) \tag{5.12}$$

where the sum is taken over all nodes k that connect to node i. We rewrite the partial derivative by the chain rule and develop.

$$\frac{\partial E}{\partial w_{ij}} = \frac{\partial E}{\partial z_i}\frac{\partial z_i}{\partial w_{ij}} = \frac{\partial E}{\partial z_j} y_i = y_i (y_j - d_j) \frac{\partial f(z_j)}{\partial z_j} = y_i \delta_j \tag{5.13}$$

Note that E is differentiable if and only if the activation function f is differentiable, so from now on we have to forget about using the Heaviside function as our threshold. Focusing instead on the logistic sigmoid function the last derivative yields

$$\frac{\partial f}{\partial z_i} = \frac{\partial}{\partial z_i}\left(\frac{1}{1+e^{-\beta z_i}}\right) = \frac{\beta}{\left(1+e^{-\beta z_i}\right)^2} e^{-\beta z_i} = f(1-f). \tag{5.14}$$

Combining these results gives us the final update rule for the nodes in the output layer.

$$w_{ij} = w_{ij} - \eta \beta y_j^2 (1 - y_j)(y_j - d_j) \tag{5.15}$$

However, this formula is only applicable to nodes for which we know the desired output value d. For the nodes in the hidden layers, we have to select a different approach. By reviewing 5.13 and noting that

$$\frac{\partial E}{\partial y_j} = \sum_k \frac{\partial E}{\partial z_k}\frac{\partial z_k}{\partial y_j} = \sum_k \frac{\partial E}{\partial z_k}\frac{\partial}{\partial y_j}\sum_i w_{jk}y_i = \sum_k \frac{\partial E}{\partial z_k} w_{jk} = \sum_k \delta_k w_{jk} \tag{5.16}$$

and therefore

$$\delta_j = \frac{\partial f\left(z_j\right)}{\partial z_j}\sum_k \delta_k w_{jk} \tag{5.17}$$

we obtain the update rule for hidden nodes

$$w_{ij} \leftarrow w_{ij} - \eta\beta y_j^2\left(1 - y_j\right)\sum_k \delta_k w_{jk} \tag{5.18}$$

The last sum is taken over all nodes k that are connected to the output of node j. This is principally important because it means that the update of the weights has to be performed backwards through the network. Before we can update the weights in node j, we have to calculate the error term δ_k for all its successors. This is the reason why the algorithm is called backpropagation.

5.8.1 Improving the Algorithm

Although the backpropagation algorithm works very well in most cases, it has some general problems associated with it. The steps through the weight space are always taken in the direction of the steepest descent of the error function, with a length proportional to the slope at that point. This means that as soon as we get close to a local minimum we most likely get stuck there because the slope is zero at all extreme-points, that is, $\nabla E = 0$.

However, we have no way of knowing if that extreme point is in fact the global minimum we are looking for, all we know is that it is a local extreme point. Sometimes if the error in the extreme point is close enough to the global minimum we can accept that non-optimal solution, but occasionally we might end up in a hollow that produces an unacceptable output. Unfortunately, there is no universal solution to this problem. Sometimes the network will settle in a local minimum, and all we can do is try to minimize the probability of this happening.

Several modifications to the backpropagation algorithm have been suggested to achieve this. One of the most successful is the introduction of a momentum term in the weight update formula.

$$w_{ij}^{(k+1)} = w_{ij}^{(k)} - \eta \frac{\partial E}{\partial w_{ij}^{k}} + \alpha \left(w_{ij}^{(k)} - w_{ij}^{(k-1)} \right) \tag{5.19}$$

This recursivity adds a form of inertia to the motion in the energy landscape and can sometimes effectively speed up convergence. It also decreases the risk of being trapped in an early local minimum. For stability reasons we see that α must be positive and less than unity, $0 \leq \alpha < 1$.

We can now summarize the modified backpropagation algorithm in a few steps.

Algorithm 5.1 Backpropagation

1. Initialize **w** by setting all weights to small random numbers
2. Present a training pattern to the input layer and calculate the outputs from all nodes in the network in a feedforward way according to

$$y = \frac{1}{1 + e^{-\beta \sum_{i=0}^{N} w_i x_i}}.$$

3. Adjust the weights in the output layer

$$w_{ij}^{(p+1)} = w_{ij}^{(p)} - \eta \beta y_j^2 \left(1 - y_j\right) \left(y_j - d_j\right) + \alpha \left(w_{ij}^{(p)} - w_{ij}^{(p-1)} \right)$$

where η is the learning rate, β the spread of the threshold function and α the momentum term. The actual output from node j is denoted y_j and the desired output d_j.

4. Work the way backwards through the network, and update the rest of the weights according to the backpropagation rule

$$w_{ij}^{(p+1)} = w_{ij}^{(p)} - \eta \beta y_j^2 \left(1 - y_j\right) \sum_k \delta_k w_{jk} + \alpha \left(w_{ij}^{(p)} - w_{ij}^{(p-1)} \right).$$

The error term δ_k is the error of the successor node k given by

$$\delta_k = \beta y_k \left(1 - y_k\right) \left(y_k - d_k\right)$$

for output nodes and

$$\delta_k = \beta y_k \left(1 - y_k\right) \sum_m \delta_m w_{jm}$$

for the nodes in the hidden layers.

5. Repeat from 2 until result is good enough.

5.9 Generalization Performance

In the last step of the backpropagation algorithm, we vaguely defined the looping criterion as, "repeat until result is good enough". What exactly is good enough – and what is the result we are talking about? Obviously, we need to clarify the concepts and define some measure that tells us the performance of the network. As previously pointed out, the most important property of a neural network is its capability to generalize. In practice, this means to perform correct input–output mappings for inputs not used when training the network.

5.9.1 Overfitting

In many ways, training a neural network is equal to fitting a curve to a set of data points as described in Figure 5.6. The curve represents the input–output mapping and is a result of the network learning the *training data points*. The training error defined in 5.7 correspond to the total distance between the curve and the points. Analogously, we can define a test error E_{tes} as the distance between the curve and a set of *test data points* drawn from the same population as the training data but not previously seen by the network,

$$E_{tes} = \sum \|\mathbf{y}_p - \mathbf{d}_p\|. \tag{5.20}$$

The summation above is taken over all points p in the test set, and thus, the test error can be seen as an estimation of the networks total generalization error. A neural network with good generalization capability will produce good input–output mappings even for data points that differ slightly from the ones in the training set. Hence, it will yield a low test error. However, this does not necessarily mean that the training error is at its minimum.

The continuous line, a, in Figure 5.6, represents the output of a properly trained network. It does not pass through all points in the training set, but it yields a good approximation of the data set as a whole. The dotted line, b, on the other hand, fits all training points exactly, but it gives a poor representation of data points not in the training set. This behavior is referred to as *overfitting.*

Practically all types of neural networks are prone to overfitting, and the multilayer perceptron is no exception. Remember that the aim of the back-propagation algorithm is to minimize the training error at any cost. This means that at some point during training, the network will reach a state where the generalization does not improve anymore. If the training proceeds past that point, the net will begin to overfit and thus inevitably lose

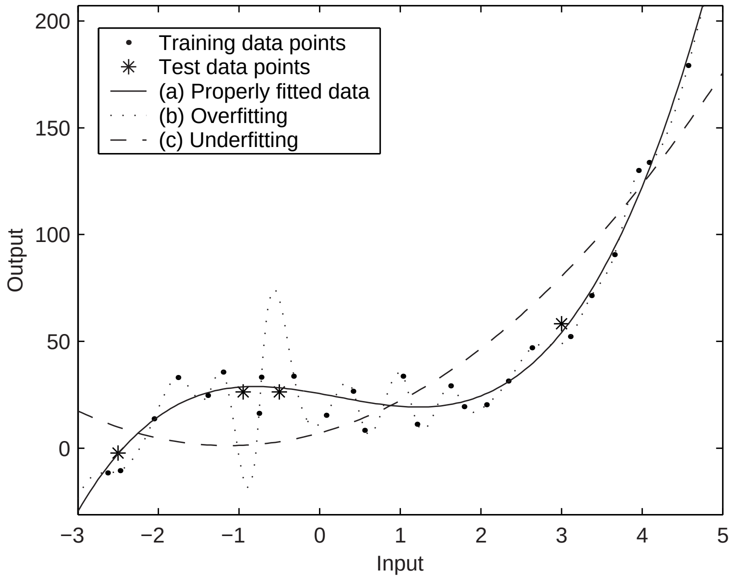


Figure 5.6: *Overfitting problem.*

its ability to generalize. Numerous methods have been proposed to solve the overfitting problem.

5.9.2 Early Stopping

The most intuitive way of dealing with overfitting is to stop training when the generalization error has reached its minimum. This technique is known as *early stopping.* It sounds simple in theory, but in reality, it is not. We have no way of knowing when we have reached the optimal point. Also, remember that the test error is only an estimation of the networks generalization error. In most introductory books on neural networks, the error curves produced by the training set and a test set are presented as smooth, continuous functions like the ones in Figure 5.7.

If this were the case, we would simply interrupt the training process as soon as the error on the test set begins to increase. Unfortunately, reality is almost always more complex than the schoolbook examples. In most real-world situations, the error curves have a number of local minima, and it is impossible to tell when we have reached the best one. In the classification package written in this project, we have worked our way around the problem by saving the internal state of the network every time E_{tes} reaches a

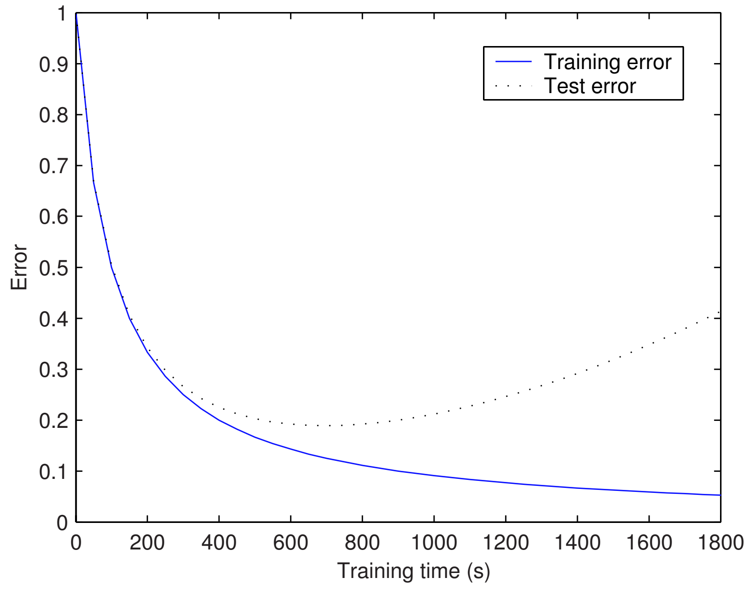


Figure 5.7: *Training and test error as a function of training time.*

minimum. That allows us interrupt the training at any time and always be able to go back to the best state encountered so far. Of course, one could argue that the test data used to measure the performance of the classifier should not be used for creating the model. This is a sound viewpoint since the main idea behind using E_{tes} as an approximation of the generalization error is that the test set should consist of unseen data only. To solve this dilemma, the package also includes functionality for monitoring a third set of data, a *validation set*, similar to the test set. This makes it possible to train the network on the training data, interrupt the training when the error on the validation data reaches its minimum and finally use the classification error on the test data to measure the performance of the network. For a more thorough analysis on backpropagation stopping criteria see [22].

5.9.3 Dimensionality Control

Another way of preventing overfitting is to reduce the number of degrees of freedom of the network. This again is analogous to the curve-fitting problem. Let us review Figure 5.6. The training data in this example comes from a physical process $y = P(x) + \varepsilon$ where $P(x)$ is a polynomial and ε some random noise. It is clear that the best approximation of the data set is achieved by curve a, which is in fact a least-mean-square fit of a third order polynomial. The more complex dotted curve b shows the typical symptoms of overfitting. Instead of fitting just the underlying signal $P(x)$, it uses all its extra degrees of freedom to fit the noise in the training data with poor generalization as result. The dashed curve c, on the other hand, does not have sufficient amount of free parameters to fit the data and therefore suffers from *underfitting.*

In a multilayer perceptron network, every weight represents a degree of freedom, and consequently, the problem of improving generalization through dimensionality control is a problem of choosing an appropriate number of nodes in the hidden layers of the network. A small network may not be complex enough to partition a complicated pattern space, and a large network may be overfitting the data, leading to poor generalization. These are the consequences of the *curse of dimensionality.*

5.9.4 Amount of Training Data

The analogy with the curve-fitting problem can be drawn even further. We realize that the oscillating behavior of curve b in Figure 5.6 is because it has enough degrees of freedom to fit every point in the test set individually. It does not have to compromise like the other two graphs. However, say that we had access to more training data from the same process and that we start adding points to the training set. Eventually, we will reach the point when there are not enough free parameters in curve b to fit them all, and hence, it will too have to start compromising. This will effectively reduce the oscillations and thereby improve the curves generalization. In the multilayer perceptron case, this means that large networks require more training data to maintain good generalization capability. Haykin [11] suggests the following rule-of-thumb for the relation between the number of training points N and the number of weights in the network W. The ε denotes the fraction of permitted errors on test data.

$$N = \mathrm{O}\left(\frac{W}{\varepsilon}\right). \tag{5.21}$$

The point here is that generalization depends on both the size of the training set and the size of the network and that finding a good setup is a critical and non-trivial issue.

Chapter 6

Practical Experiment

So far, we have discussed the theory behind brain–computer interfacing. We have looked at the functionality of the human brain and how its electrical activity can be extracted in form of EEG signals. We have also discussed appropriate algorithms for preprocessing and classifying the recorded information. In other words, we have concluded the design of our BCI system and thereby fulfilled the first purpose of this thesis.

The second purpose is to implement the design ideas and test them in a pilot experiment. The implemented software is described in Appendix A, and this chapter is devoted to the actual experiment.

6.1 Background

The task selected for this part deals with classification of brain waves related to voluntary limb movement. The purpose is to see if it is possible to detect and categorize two dimensional joystick movements, based on recorded EEG alone. The experiment is performed both for real physical movements and for imagined movements.

The motive for such experiment is primarily medical rehabilitation. If successful, a BCI that can detect two dimensional joystick movements without relying on muscle activity can be operated even by patients with severe motor disabilities. The output of the system could be used to control, for example, a word processor or a wheelchair, and thereby greatly improve the quality of life for people with ALS or spinal injuries that make other forms of communication impossible.

6.2 Procedure

Four healthy test subjects, three females and one male, volunteered to participate in the study. The subjects were between 20 and 30 years old and free of any form of medication. The experiment took place in a quiet room with dim lighting at The Swedish Defense Research Agency (FOI) in Linköping, Sweden. Only the test subject and the experimenter were allowed in the room during the time of the experiment. The session consisted of two parts, one per experimental condition. Each part lasted about twenty minutes and the first part was preceded by a five-minute introduction, to help the subject to get used to the equipment and the task to perform. The entire experiment, including introduction and mounting of electrodes took about four hours to complete. Figure 6.1 illustrates the experiment disposition.

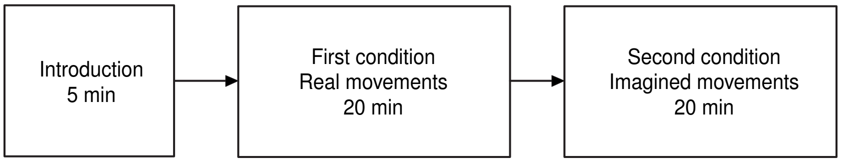


Figure 6.1: *The procedure of the experiment session. The test consisted of two experimental conditions plus a brief introduction round.*

In the beginning of the session, the test subject was seated in a comfortable chair in front of a table with a portable computer. The computer presented the stimuli during the experiment. On the table was also a joystick, located so that it could be operated by either the left or the right hand. One of the subjects was left-handed and therefore performed the operations with the left hand while the other subjects used their right hand. Throughout the whole experiment a fixation cross was presented at the center of the computer screen. The subjects were instructed to keep their eyes at the cross and to relax and move as little as possible. The reason for this is that the EEG can be very sensitive to muscle activity and that any movement, even eye blinks, may distort the recorded signal and make it useless for classification.

Both experimental conditions consisted of 100 trials divided into five categories. All stimuli started with a short acoustic warning tone, to tell the subject to be alert. After one second the outline of an arrow was presented in one of four positions, left, right, above or below the fixation cross, as depicted in Figure 6.2.

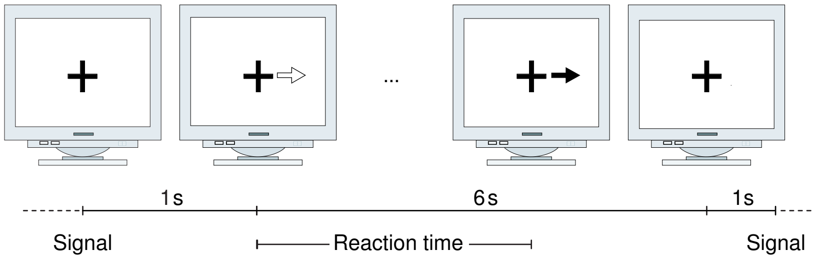


Figure 6.2: *A stimulus consisted of an acoustic signal followed by an indication of direction in form of an arrow on the computer screen. Each stimulus lasted eight seconds and was followed by another acoustic signal.*

6.2.1 First Condition, Real Movements

The task in the first experimental condition was simply to move the joystick in the direction indicated by the arrow and to keep it there. The joystick was connected to the computer that presented the stimulus, and its position was continuously logged by the program. Once the joystick was moved to the indicated position the outline of the arrow on the screen was filled in, to form an all black arrow, as a form of feedback. After six seconds, the arrow was removed and one second later, another warning tone sounded, indicating the end of the trial. After another four seconds, the procedure started all over again with a new trial. In twenty percent of the trials, no arrow was presented. The subjects were then instructed to remain relaxed and not to move the joystick at all. These trials were used to record a baseline signal.

6.2.2 Second Condition, Motor Imagery

In the second experimental condition, the subjects were asked to merely imagine moving the joystick, but not to perform the actual movement. Apart from that, the procedure was identical to the first condition.

6.3 EEG Signal Recording

During each trial, EEG data was recorded from seven gold plated electrodes attached to the skull of the subject and from two EOG electrodes mounted above and below one eye. The EEG electrodes were positioned at C3, C4, Cz, P3, P4, O1, and O2, according the international 10–20 system, as depicted in Figure 6.3. The EOG electrodes around the eye were used to detect eye blinks.

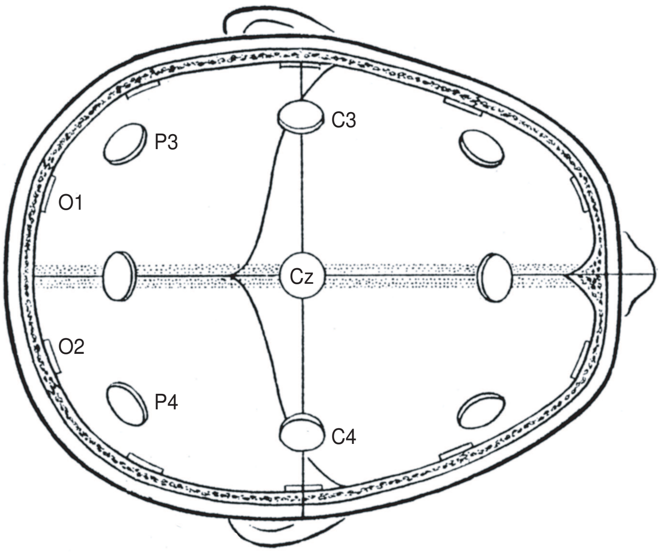


Figure 6.3: *A total of seven EEG electrodes was used in the recording.*

Data was recorded for eight seconds starting at the same time as the first warning tone, that is, one second before the arrow was presented. The signals were sampled at a rate of 256 Hz using a National Instruments NI-DAQ 6312 12-bit A/D converter, mounted in a standard PC. All electrodes were connected to the computer through a Grass model 12 EEG amplifier with built-in band-pass filters. The amplification was set to 10^4 for the first seven channels and to 500 for the EOG signal. All signals were band-pass filtered between 0.3 and 30 Hz, before the sampling. The nine recorded signals were combined to form eight channels of data according to Table 6.1. To ensure good signal–noise ratio each electrode pair was required to have an impedance below 10 kΩ.

6.3.1 Artifact Removal

To minimize the risk of artifacts like eye blinks and other movements distorting the data, all trials were visually inspected for abnormalities before the feature extraction. Trials that contained contaminated data were discarded. Figure 6.4 shows an example of an eye blink and how that affects

Channel	Electrode	Reference	Amplification	Impedance
1	C3	Cz	10^4	$\leq 10\text{k}\Omega$
2	C4	Cz	10^4	$\leq 10\text{k}\Omega$
3	P3	Cz	10^4	$\leq 10\text{k}\Omega$
4	P4	Cz	10^4	$\leq 10\text{k}\Omega$
5	O1	Cz	10^4	$\leq 10\text{k}\Omega$
6	O2	Cz	10^4	$\leq 10\text{k}\Omega$
7	C3	C4	10^4	$\leq 10\text{k}\Omega$
8	X	Y	500	$\leq 10\text{k}\Omega$

Table 6.1: *Eight channels combined of signals from nine electrodes.*

the data of all channels[1]. This procedure was repeated for all test subjects, and the subject with the lowest number of discarded trials, a left-handed female, was selected for the detailed analysis.

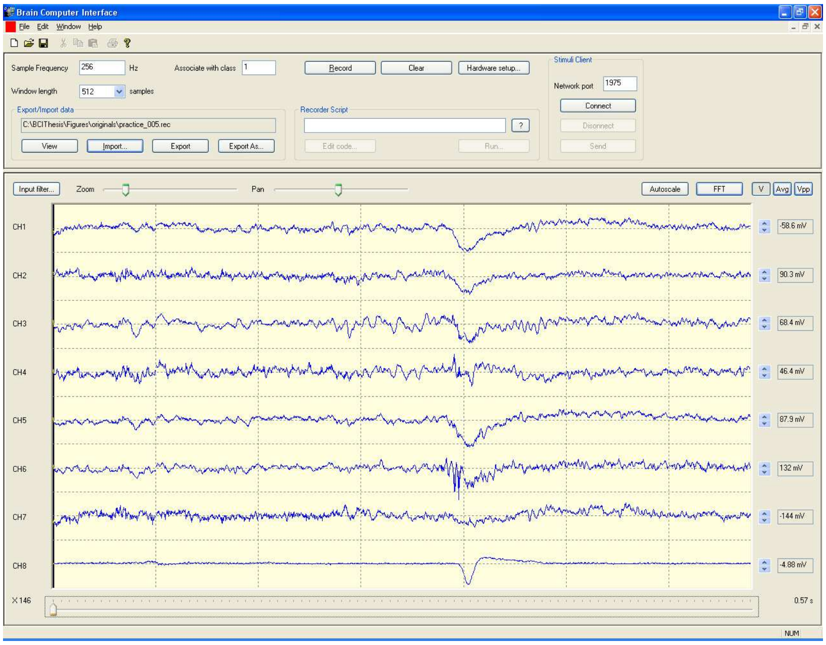
6.4 Preprocessing

Since data was recorded for eight seconds with a sampling rate of 256 Hz, each channel produced 2048 samples per trial. Totally, all seven EEG channels produced over 14 thousand samples per recording. Clearly, that amount of data is too much to handle for any neural network based classifier. Hence, we have to do something to reduce dimensionality of the data, but without loosing any valuable information of course. How do we do that?

6.4.1 Channel Selection

First, some channels might be unusable for classification. As described in Figure 6.3 and Table 6.1 we have used data from seven EEG electrodes distributed across the back of the head. The reason we selected these particular positions is that they overlay regions of the cortex that we hope will be actively involved in the movements we want to classify. The same electrode positions have also been used in other classification experiments

[1]The screenshot is taken from recorder module of the developed software. A more thorough description of the application can be found in Appendix A.



Sample Frequency 256 Hz
Associate with class 1
Window length 512 samples
Export/Import data C:\BCIThesis\Figures\originals\practice_005.rec
Network port 1975

CH1 -58.6 mV
CH2 90.3 mV
CH3 68.4 mV
CH4 46.4 mV
CH5 87.9 mV
CH6 132 mV
CH7 -144 mV
CH8 -4.88 mV

X146 0.57 s

Figure 6.4: *Artifacts caused by an eye blink. The eighth channel shows the muscle activity around the eye and is used to detect blink artifacts.*

with good results [5]. However, we do not know if all channels contain classifiable information. It is not unlikely that one or more of the electrodes records information that is completely uncorrelated with the movement we are interested in. In such case, that electrode can be removed to decrease the complexity of the data, but without degrading the performance of the classifier. We will make a more thorough analysis on electrode information content in the next chapter. As a preparation for that analysis, we formalize the problem of finding an optimal subset of channels by defining the set consisting of all seven channels as

$$\mathcal{C}_{tot} = \{C3, C4, P3, P4, O1, O2, C3C4\} \tag{6.1}$$

and a subset $\mathcal{C} \subseteq \mathcal{C}_{tot}$ as the set of c channels selected for feature extraction.

6.4.2 Dividing Data

Second, the information content in the signal might vary over the eight seconds, and we want to keep the amount of samples needed to perform a classification as low as possible. From all channels, $c_i \in \mathcal{C}$, a set of M windows was extracted. Each window consisted of N samples and overlapped the previous window by T samples according to Figure 6.5. In the analysis in Chapter 7, we will find out how the windows should be distributed to give optimal classification results, that is, what part of the eight seconds contain the most important information about the joystick movement.

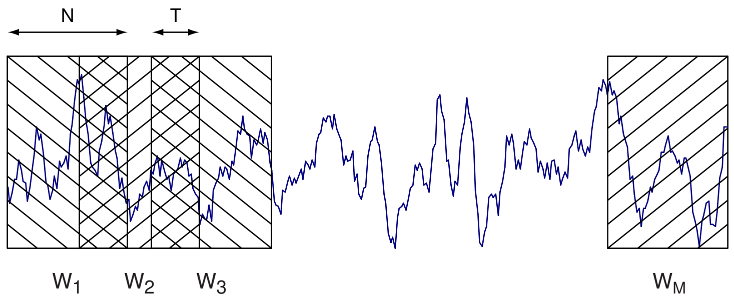


Figure 6.5: *Each sample is divided into M windows of length N overlapping the prevoius window with T samples.*

6.4.3 Parameterize Data

To model the signal contained in the M windows a pth order lagged autoregressive model, LAR, was applied to each channel according to

$$x_c(k) = -\sum_{i=1}^{p} a_c(i) x_c(k - iL) . \tag{6.2}$$

This produced c vectors, $\mathbf{a}_c$, of length p per window and, hence, $c{\cdot}M$ vectors per trial. The LAR model had a lag of L meaning that the samples used in the AR model were separated by L samples. In practice, this was achieved by down-sampling the recorded signals by a factor L and then applying a standard non-lagged AR model. The reason for using the lagged model is that it is known to make the model less sensitive to noise [18]. The coefficients were estimated using Burg's algorithm described in Chapter 4.

6.4.4 Creating Feature Vectors

The AR coefficients produced by the parameterization were concatenated to form one single feature vector, of length $c{\cdot}p$, per window w.

$$\mathbf{x}_w = (a_{1,1}\ a_{1,2}\ \cdots\ a_{1,p-1}\ a_{1,p}\ a_{2,1}\ a_{2,2}\ \cdots\ a_{c,p-1}\ a_{c,p})^T \tag{6.3}$$

These vectors were used as inputs to the neural network classifier.

6.5 Classification

The final categorization of the extracted feature vectors was done using a classifier based on artificial neural networks. In this experiment, we used multilayer perceptrons with none, one or two hidden layers.

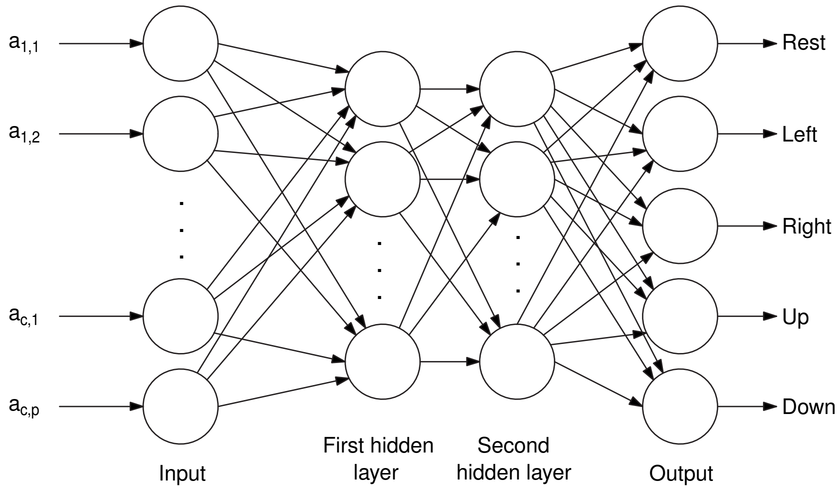


Figure 6.6: *Multilayer perceptrons were used as the basis of the classifier module.*

The number of inputs of the network corresponds to the length Cp of the feature vector, and the number of outputs was set to five, one node per category: *rest*, *left*, *right*, *up* and *down*. Each node produces a real-valued output between zero and unity. Hence, with the five outputs of the network organized as a vector $\mathbf{y}$ the desired output $\mathbf{y}_d$ for each category was set according to Table 6.2. For example, if we present the network with an input belonging to class 2, that is, a feature vector extracted from a trial where the test subject moved the joystick to the right, we want the network to produce the output $(0\ 0\ 1\ 0\ 0)^T$. If the subject was just resting, we want the output to be $(1\ 0\ 0\ 0\ 0)^T$. The network was trained using the backpropagation algorithm presented in Chapter 5.

6.5.1 Generalization

To test the classifiers ability to generalize, that is, to produce good output values even for patterns not previously presented to it, we used the following simple cross-validation procedure. The complete set of artifact-free trials, $\mathcal{D}$, was divided into three distinct subsets: one training set $\mathcal{D}_{trn}$, one validation

Class	Category	Desired output
0	Rest	$(1\ 0\ 0\ 0\ 0)^T$
1	Left	$(0\ 1\ 0\ 0\ 0)^T$
2	Right	$(0\ 0\ 1\ 0\ 0)^T$
3	Up	$(0\ 0\ 0\ 1\ 0)^T$
4	Down	$(0\ 0\ 0\ 0\ 1)^T$

Table 6.2: *Desired output vectors from the network.*

set $\mathcal{D}_{val}$, and one test set $\mathcal{D}_{tes}$ such that each pattern was represented in exactly one set.

$$\begin{cases} \mathcal{D} = \mathcal{D}_{trn} \cup \mathcal{D}_{val} \cup \mathcal{D}_{tes} \\ \mathcal{D}_{trn} \cap \mathcal{D}_{val} = \emptyset \\ \mathcal{D}_{trn} \cap \mathcal{D}_{tes} = \emptyset \\ \mathcal{D}_{tes} \cap \mathcal{D}_{val} = \emptyset \end{cases}$$

The trials in the validation set and the test set were selected randomly from the original pool of trials but controlled to ensure that each set contained an equal amount of trials corresponding to each category.

Both experimental conditions produced 100 trials, 20 per category. In the first condition, we had to discard 9 patterns due to eye blinks and other artifacts. The data produced by the second condition contained 18 contaminated trials. In both cases, we selected ten trials for the test set and ten for the validation set, each containing two trials per category according to Table 6.3.

Movements	Physical	Imagined
Training, $\mathcal{D}_{trn}$	71	59
Validation, $\mathcal{D}_{val}$	10	10
Test, $\mathcal{D}_{tes}$	10	10

Table 6.3: *Distribution of artifact free trials.*

The training set was used to train the network and the test set to check its performance. The validation set can be used to define stopping criteria for the network training, as described in Section 5.9. It can also be used to weight the output of a network when combining two or more classifiers [26, 11]. However, since the classifier in this experiment consisted of a single network, the validation set was used only to decide when to interrupt the training process.

6.6 Post Processing

As we said, the output of each node in the network is a value between zero and unity. To make it easier to compare this value to the desired output value, which only contains binary values, we applied a simple threshold filter to the output of the network as shown in Figure 6.7. The output node that produced the largest value was set to unity and the rest to zero. This ensures that the classifier always produces outputs that correspond to one of the predefined categories. A classification was therefore considered correct if and only if the filtered output exactly matched the desired output.

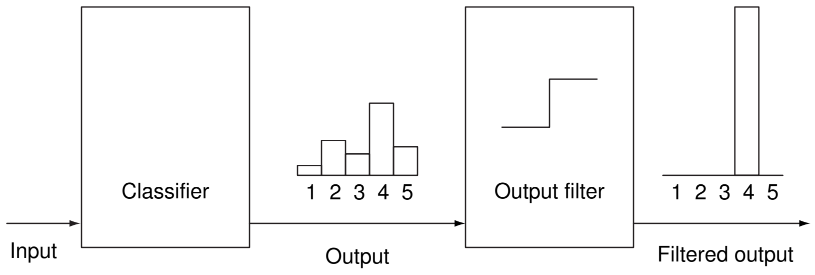


Figure 6.7: *Principle of the threshold function used for post processing.*

6.7 Classification Performance

As a direct measure of the performance of the classifier, we used the amount of correctly classified patterns in the test set divided by the total number of patterns in the set.

$$R_c = \frac{\text{number of correctly classified test patterns}}{\text{total number of patterns in the test set}} \tag{6.4}$$

This gives us a value between 0% and 100%. We shall refer to this value as the *classification rate* or *accuracy* of the classifier. Note that an accuracy of 20% corresponds to a complete random classification, since the number of categories is five.

Chapter 7

Analysis and Results

In this chapter, we present the analysis of the recorded data and the final results of the study. The major part of the chapter focuses on the results of the first experimental condition, the one where the subject actually moved the joystick. The data from the second condition is analyzed in exactly the same way, and therefore, the results from that investigation are commented primarily when they differ from the output of the first one.

7.1 Analysis Procedure

As described in the previous chapter, the classification of the recorded EEG data depends on a number of parameters. The set of electrodes used, the distribution of the windows utilized for feature extraction, the LAR model parameters and the size of the neural network are only a few examples. All of these parameters play important roles in the final classification, and it is therefore interesting to see how altering them changes the final result, the classification rate. The ultimate goal is of course to find a specific set of parameters that gives us the highest accuracy possible. In other words, we may regard this analysis as a multidimensional optimization problem, where the classification rate is our object function and the parameters that control the classification are the free variables. Mathematically, we express this with an equation on the form

$$R_c = f\left(c_{win}, w_{win}, p, L, \mathbf{H}_{ann}, \mathcal{C}, \ldots\right). \tag{7.1}$$

Here we let R_c denote the classification rate, c_{win} and w_{win} the center and with of the recorded window respectively and p and L the model order and lag of the preprocessor filter. The vector $\mathbf{H}_{ann}$ represents the configuration of nodes in the neural network and $\mathcal{C}$ the set of electrodes used. The ellipsis at the end of the parameter list indicate that the model depend of even more parameters. However, we hope that the six variables described are the ones that affect the result the most, and for the rest of this analysis we will therefore focus on these parameters. Despite that simplification, the optimization procedure is far from trivial. There is no way of knowing beforehand how the six parameters depend on each other so the best we can do is to try to maximize R_c locally for each parameter and hope that this brings us close to the global optimum.

We will start by analyzing the data from the first experimental condition, the one where the subject performed real physical movements.

7.2 First Condition, Real Movements

In the first experimental condition, a stimulus in form of an arrow on a computer screen, pointing in one of four directions, was presented to the subject. The task was simply to move a joystick in the direction indicated by the arrow. During each trial, EEG data was recorded for eight seconds, starting one second before the stimulus was presented and lasting until one second after the stimuli was removed. Figure 7.1 illustrates the recording during one trial.

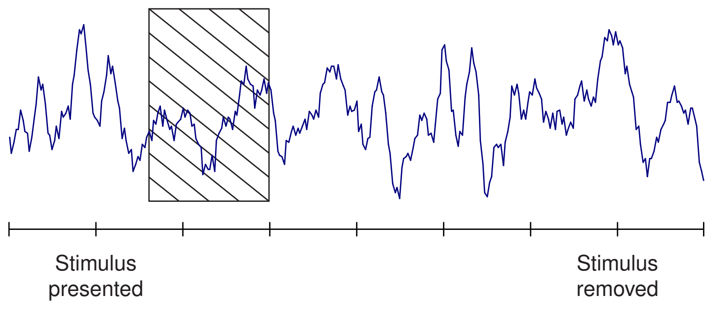


Figure 7.1: *Optimal recording period. What part of the recorded eight seconds contains the most important information?*

7.3 Optimal Recording Period

Obviously, in a real world brain–computer interface, having to collect data for eight whole seconds before being able to make a single classification is too slow. Controlling a wheelchair would be out of the question, and even operating a word processor would be cumbersome. Besides, we know from Chapter 4 that the EEG can be considered stationary only for windows shorter than about one second. If using longer windows we have to replace our static LAR preprocessor with some adaptive method, and this does not only make the classification procedure more complex, it is also known to sometimes decrease performance, since the amount of data produced by the preprocessor increases [15].

So, if we can use no more than one second of the eight seconds recorded, which part will give us the most information? There is only one way to find out, trial-and-error. We start by extracting LAR features from 16 windows per trial. The window length is 0.25 seconds or 64 samples, and each window overlaps the previous window by 60 samples. This gives us the data distribution presented in Figure 7.2.

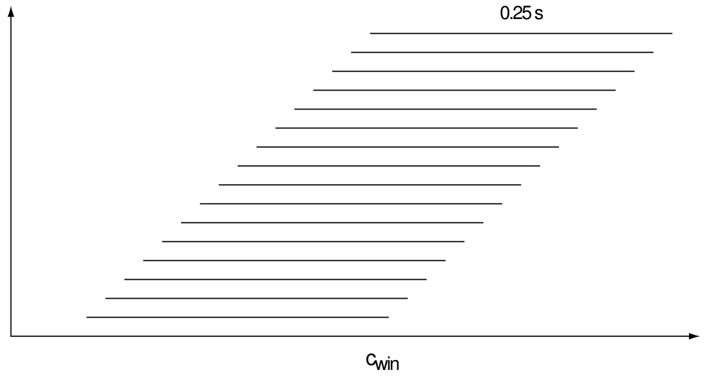


Figure 7.2: *Window distribution. Each eight second recording is divided into 16 quarter second windows overlapping by 60 samples.*

The windows are equally distributed around the time c_{win}, and hence, optimizing R_c with respect to c_{win} will give us a clear indication of what part of the recorded eight seconds contain the most information. The feature extraction was repeated for 17 different values of c_{win} with the model order set to $p = 6$ and the lag to $L = 1$. The results were then used to train a multilayer perceptron with 40 hidden nodes in the first layer and 15 in the second hidden layer. Figure 7.3 shows the classification rate as a function of c_{win}. Here, the time is counted from when the stimulus was presented.

It is clear that the most important period for the classification is in the segment between 1 and 1.5 seconds. We can explain this by looking at the average reaction time for the subject. The histogram in Figure 7.4 shows the distribution of the reaction time[1] over the 80 trials that involved movements. The similarity with Figure 7.3 is noticeable. The average reaction time for this test subject was 1.37 seconds meaning that the actual movement, in most cases, took place in the interval between 1 and 1.5 seconds after the stimulus was presented. This corresponds very well to the maximum peak in classification accuracy.

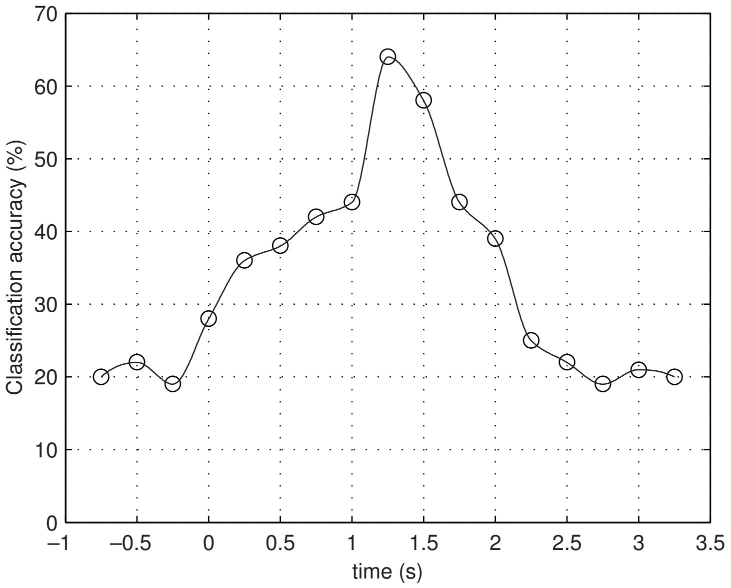


Figure 7.3: *Classification result as a function of window center.*

Returning to Figure 7.3 we note that the classification rate in the interval between –1 and 0 seconds is around 20%, that is, no better than a random guess. That makes sense, since we cannot expect to get any output before we present the input. Also, in the interval above 2.5 seconds the accuracy is no better than random. Consequently, just holding the joystick in a fixed position does not produce classifiable EEG. At least not using the classification method presented in this thesis.

[1]The reaction time is in this case measured from the moment when the stimulus was presented until 80% of the movement was completed, that is, until the position of the joystick reached 80% of its maximum value. This is also the point when the feedback (filling in the arrow) was presented to the subject.

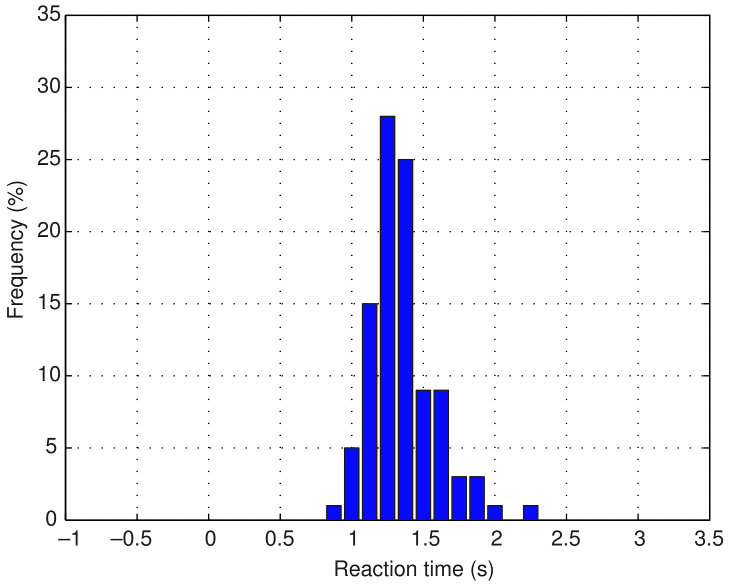


Figure 7.4: *Histogram of reaction times.*

7.4 Window Width

Now that we know what part of the recorded data contains the most information, we have to determine how much data is needed to make a good classification. We said before that windows longer than one second require a non-static preprocessor model, so we cannot go over that. But, how short can we make our windows before we start loosing information. Is half a second enough? One eighth of a second? Or, maybe just a couple of samples? As before, trial-and-error is the only way to find out.

Since the accuracy seems to have its optimum for windows centered at 1.25 seconds we repeat the first analysis, but with the window center fixed to $c_{win} = 1.25$. Instead, we vary the window width w_{win} in the interval 0 to 1 seconds. The results are presented in Figure 7.5.

It seems that we were lucky when we selected a window with of 0.25 seconds in the first analysis. In fact, there is no other value we could have picked that gives better performance. If making the window shorter we would rapidly loose valuable information. If making it longer we would only get a slower classifier without gaining any accuracy.

To summarize, it seems that the most important section of the recorded EEG data is the segment between 1.1 and 1.4 seconds the part where the

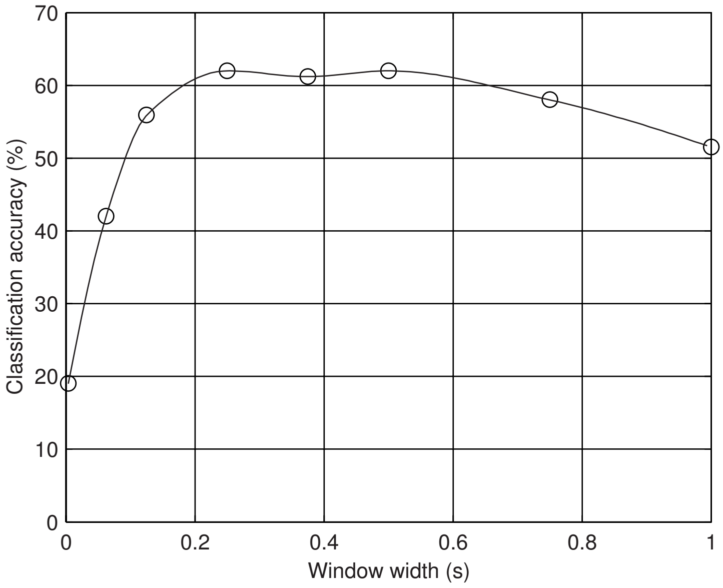


Figure 7.5: *Accuracy as a function of window width.*

actual movement took place. We will therefore focus on this section in the following analyses.

7.5 Neural Network Size

We know from Chapter 5 that the number of layers in a multilayer perceptron strongly affects its ability to recognize certain patterns. For example, the XOR-problem can be easily solved using a two- or three-layer perceptron, while a single layer network is bound to fail since the classes are not linearly separable. Even the number of nodes within the layers affects the performance, and in a way that is very difficult to predict. Each node contributes to the edges that make up the boundaries between classes. Hence, a larger network means more edges and possibly a finer separation of the categories. However, using too many nodes makes the learning process slow, and it might also lead to over-fitting since the number of degrees-of-freedom in the network increases. Choosing the optimal network size for a specific task is a non-trivial problem. There have been several books written on this topic alone. To investigate what network size is best suited for our problem we create 13 nets, all of different sizes according to Table 7.1. All networks are trained five times, and the classification rate is noted each time. The average accuracy obtained by each net is presented in Figure 7.6.

Network	Input	First	Second	Output
A1	42	–	–	5
B1	42	5	–	5
B2	42	10	–	5
B3	42	20	–	5
B4	42	30	–	5
B5	42	40	–	5
B6	42	100	–	5
C1	42	5	5	5
C2	42	10	7	5
C3	42	20	10	5
C4	42	30	12	5
C5	42	40	14	5
C6	42	80	20	5

Table 7.1: *Node configuration of the tested networks. Column three and four represent the number of nodes in the first and the second hidden layer.*

Even though the differences in accuracy between the 13 networks are small, we can see that the best results are produced by network B4, a two-layer network with 30 nodes in the hidden layer. This is not so surprising. What is somewhat surprising though is that the single layer perceptron A1 performs so well. An average classification rate of 60% is actually better than most three-layer networks tested. This is interesting because it indicates that the whole problem of classifying the LAR processed EEG data is very close to linear.

7.5.1 Training Time

Looking at the classification results in Figure 7.6, we note that the three layer networks, C1–C6, seem to perform better the bigger they are. It is therefore tempting to try to build even larger networks than the 147-node C6 in hope of further improving the performance. However, before we resort

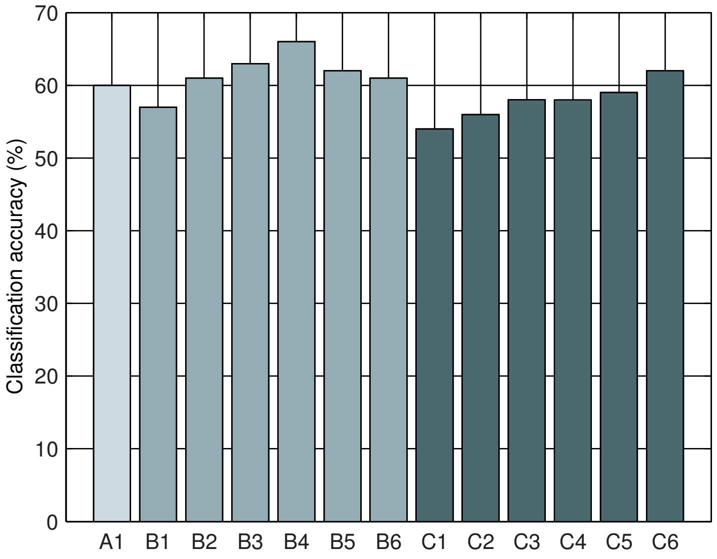


Figure 7.6: *Classification rate for a number of networks of different sizes. Network A1 is a single layer network, B1 to B6 have two active layers, and C1 to C6 are three layer networks.*

to that, we should consider the downsides of working with larger networks. One is that generalization usually deteriorates as a result of overfitting. This is probably why performance drops when adding more than 30 hidden nodes to a two-layer network, and we will most likely see a similar behavior in the group of three-layer networks if we start adding nets larger than C6. However, the main reason why we chose not to work with very large networks in this case is that the training time explodes as the net grows. Figure 7.7 shows the relative training time of the nets we already tested. Since the time needed obviously depends on the computer power allocated for the task, all values have been normalized against the training time of the best network B4.

To put things into perspective, a standard PC with a single 1.3 GHz Pentium 4 processor needed about 20 minutes to complete the training of a B4 network. Even though it might be a bit off the topic of brain–computer interfaces, it is instructive to see how the training time is affected by the size of a neural network. Figure 7.8 shows the relative training time as a function of the number of weights, that is, the number of free parameters

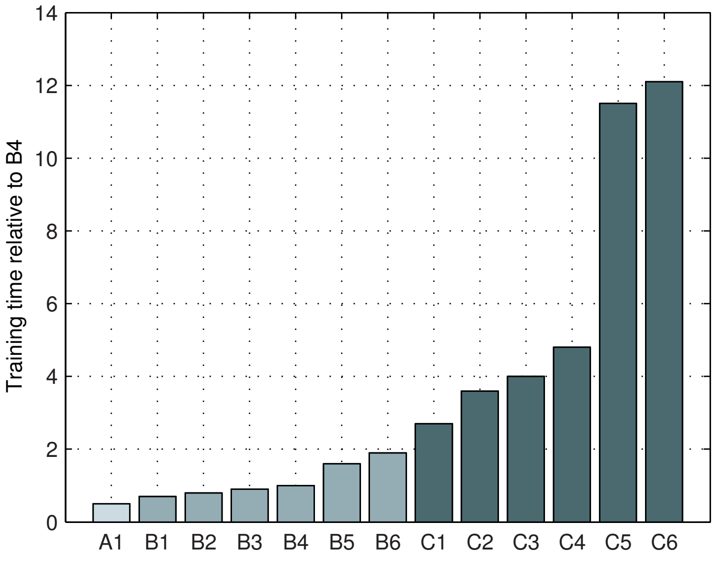


Figure 7.7: *Relative training time for the tested networks.*

in the network. The relation is very close to linear, and we can probably expect it to continue that way even for networks with somewhat more than 5000 weights. This means that a network with twice the number of nodes compared to C6 would need over 12 hours to complete one single training session. Also, remember that each net is trained multiple times and that this is only one of a number of analyses performed on the data. With that in mind it should not be too hard to resist the temptation of building very large networks, just to gain one or two percent in classification rate.

7.6 LAR Model Parameters

The technique of using autoregressive modeling as a preprocessing step in a BCI system is not new. In [3, 2] Anderson et al. compared classification results of four different representations of EEG signals, including raw data, low-pass filtered raw data, Karhunen-Loève decomposition and a frequency-band representation calculated from AR coefficients. The results indicated that the AR-based representation is superior to the other models. Later, the same group extended their study [5] to include three new representa-

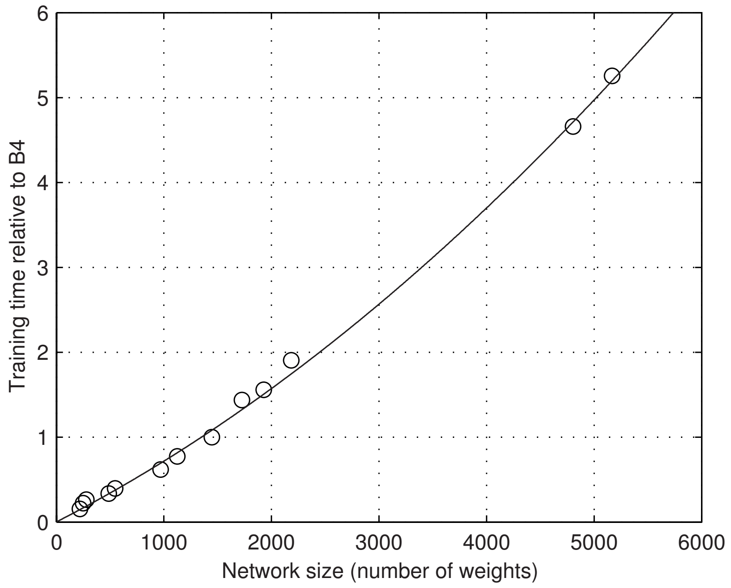


Figure 7.8: *Relative training time as a function of the number of weights in a neural network.*

tions, AR coefficients only, multivariate AR coefficients[2] and feature vectors composed by eigenvalues of the correlation matrix of the data. The best results were achieved for the representations based on scalar and multivariate AR coefficients. If computation time is an issue, Anderson recommends the use of scalar AR coefficients, since the multivariate model involves more complex calculations.

In this project, we have followed Anderson's recommendation and represent our recorded EEG data with the coefficients of a pth order scalar AR model. The parameters were estimated using Burg's algorithm on a per-channel basis and then concatenated to form a set of $p{\cdot}c$ dimensional feature vectors, where c is the number of channels used. In another BCI study [25], Penny and Roberts used a similar approach and showed that the model order p strongly affects the final classification rate. Using a linear classifier, they achieved the best results when the data was fitted to a 9th-order model, but the results were almost identical for model orders between 3 and 10.

[2]Multivariate models consider not only relations between subsequent samples in the same channel but also correlations between samples of different channels.

Repeating that analysis on our own EEG data and using a three layer neural network as classifier we find that a model order of $p = 2$, gives the best results. All remaining analyses will therefore use a second order model. However, as shown in Figure 7.9, the results deviate only slightly from the maximum in the range 2 to 10.

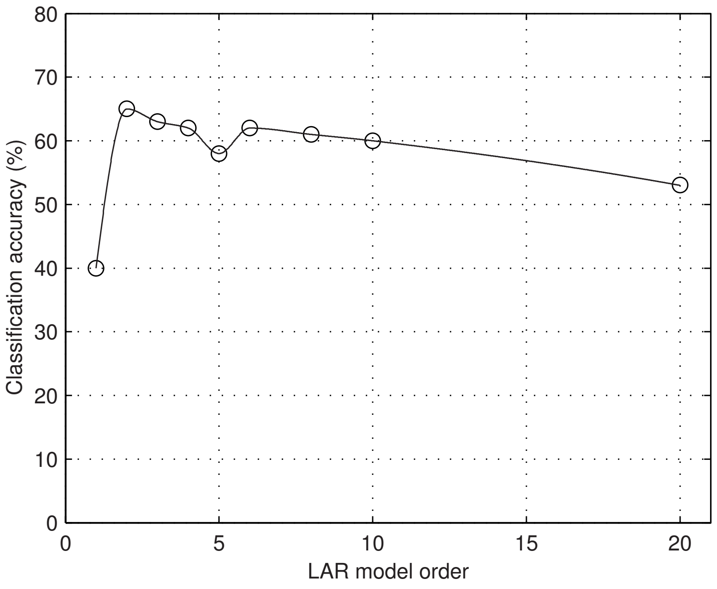


Figure 7.9: *Classification accuracy as a function of the order p of the LAR model.*

The idea of introducing a lag in the AR model was also presented by Penny and Roberts [19, 18]. To their experience, lagged AR models are superior to non-lagged in being less sensitive to noise. Our experiments, on the contrary, show that the classification rate drops rapidly when the lag is introduced and keeps on falling as the lag is increased. This is interesting and we will come back to this result in the next chapter. Throughout the rest of this analysis we will use AR models without lag, $L = 1$.

7.7 Optimal Electrode Distribution

So far, the analyses have included data derived from all EEG electrodes, that is, channel 1 to 7 as described in Table 6.1. In a real-world BCI system, we would like to keep the number of electrodes as low as possible

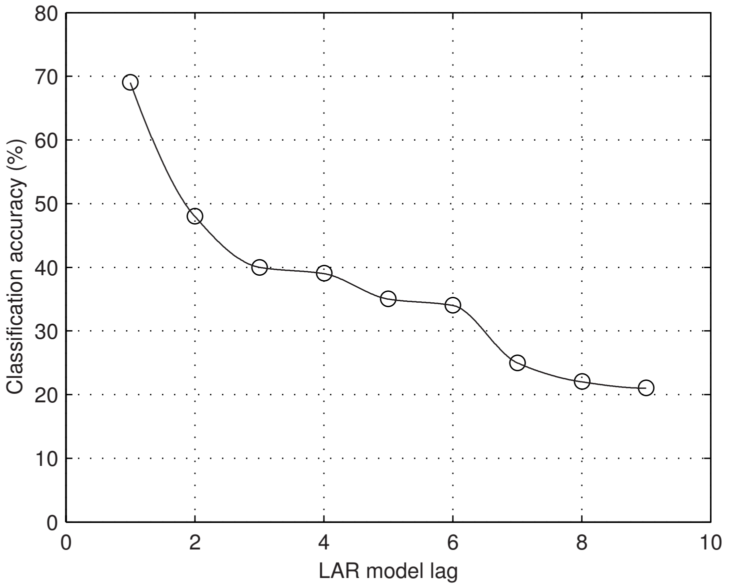


Figure 7.10: *Classification rate versus LAR model lag.*

to minimize the amount of data to process and also to avoid unnecessary and time-consuming electrode mounting. We know from Chapter 2 that different regions of the cerebral cortex are responsible for different functions. It is therefore very interesting to see how much of the information used in movement classification is extracted from each region. The goal is of course to eliminate electrodes that do not contribute to the final result and perhaps suggest adding electrodes to important regions not covered by the original set.

Using the terminology we established in Chapter 6, we refer to the set of all recorded EEG channels as

$$\mathcal{C}_{tot} = \{C3, C4, P3, P4, O1, O2, C3C4\} \tag{7.2}$$

and as before, we define a subset

$$\mathcal{C} \subseteq \mathcal{C}_{tot} \tag{7.3}$$

as the set of channels actually used in the classification. To determine the optimal subset we systematically remove one channel at the time from $\mathcal{C}_{tot}$

forming seven new sets.

$$\begin{aligned}
\mathcal{C}_{C3} &= \{C4, P3, P4, O1, O2, C3C4\} \\
\mathcal{C}_{C4} &= \{C3, P3, P4, O1, O2, C3C4\} \\
\mathcal{C}_{P3} &= \{C3, C4, P4, O1, O2, C3C4\} \\
\mathcal{C}_{P4} &= \{C3, C4, P3, O1, O2, C3C4\} \\
\mathcal{C}_{O1} &= \{C3, C4, P3, P4, O2, C3C4\} \\
\mathcal{C}_{O2} &= \{C3, C4, P3, P4, O1, C3C4\} \\
\mathcal{C}_{C3C4} &= \{C3, C4, P3, P4, O1, O2\}
\end{aligned} \tag{7.4}$$

Data from all sets are preprocessed using a second order AR model and classified by a two-layer multilayer perceptron with 30 hidden nodes. The results are compared to the classification rate produced by the complete set $\mathcal{C}_{tot}$ creating a relative measure ρ of the information content in each electrode.

$$\rho(\mathcal{C}) = \frac{R_c(\mathcal{C}_{tot}) - R_c(\mathcal{C})}{R_c(\mathcal{C})} \tag{7.5}$$

What $\rho(\mathcal{C})$ really says is that, compared to the classification using subset $\mathcal{C}$ we can gain a factor $\rho(\mathcal{C})$ in accuracy by adding the missing channel forming the total set $\mathcal{C}_{tot}$. Figure 7.11 presents the result of the comparison.

For three of the electrodes, P3, O1 and O2 the relative classification rate ρ is negative, meaning that removing any of them from the set used for classification would actually make the accuracy better. The electrode that ruins the results the most is the O2 electrode overlying the visual cortex. Consequently, the best set of channels of size six is formed by removing channel O2 from the original set.

Repeating the analysis by systematically removing channels from $\mathcal{C}_{O2}$ and calculating $\rho(\mathcal{C})$ for the six sets of remaining electrodes

$$\begin{aligned}
\mathcal{C}_{O2C3} &= \{C4, P3, P4, O1, C3C4\} \\
\mathcal{C}_{O2C4} &= \{C3, P3, P4, O1, C3C4\} \\
\mathcal{C}_{O2P3} &= \{C3, C4, P4, O1, C3C4\} \\
\mathcal{C}_{O2P4} &= \{C3, C4, P3, O1, C3C4\} \\
\mathcal{C}_{O2O1} &= \{C3, C4, P3, P4, C3C4\} \\
\mathcal{C}_{O2C3C4} &= \{C3, C4, P3, P4, O1\}
\end{aligned} \tag{7.6}$$

gives us the results in Figure 7.12. The most noticeable difference to the data presented in Figure 7.11 is that the relative classification rate is now positive

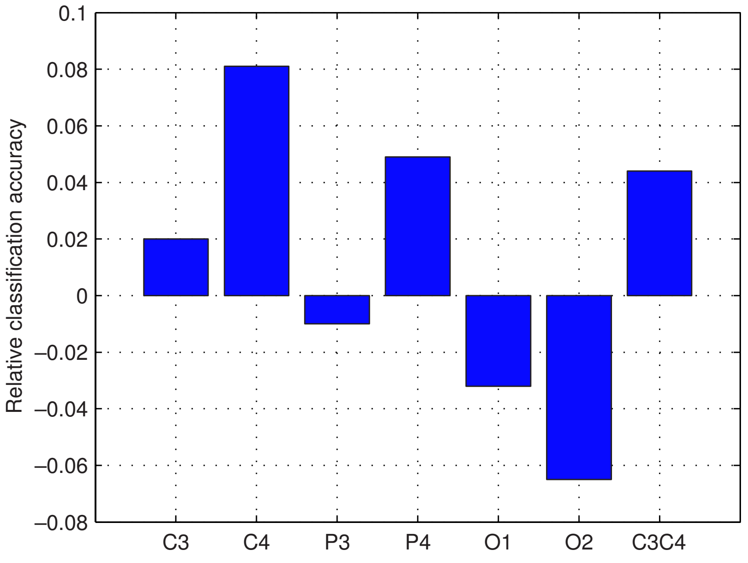


Figure 7.11: *Relative information content in the original set of seven channels.*

for all channels. This of course means that all electrodes make positive contributions to the accuracy and that removing any one electrode from the set, $\mathcal{C}_{O2}$, will degrade the overall performance of the system. Electrode C3 alone does not add much information not already contained in the other electrodes, but the combination with C4 in the last channel does, so we can still not remove C3 completely.

There is more to say about the information content in each electrode, and we will discuss this matter further in the next chapter. But first, we should analyze the data from the second experimental condition.

7.8 Second Condition, Imagined Movements

The second condition in this experiment was in all essentials equal to the first one, except for one important detail. Instead of physically moving the joystick in the direction indicated by the stimulus arrow, the subject was told to just imagine the movement, to visualize the hand moving the joystick, but still trying to avoid all muscle activity. In some sense, this part is both more and less interesting than the first one. It is exciting

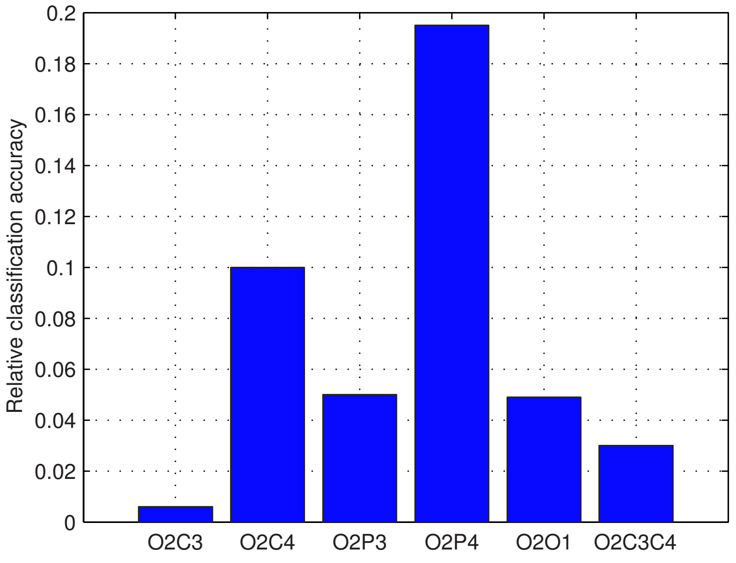


Figure 7.12: *Relative information contents in different channels when the channel O2 has been removed.*

because it mimics a real-world situation. A brain–computer interface that detects hand movements is of little use if the person employing it is able to actually perform the physical movements. In such case, other more reliable sensor systems can be used for detection, joysticks for example. One of the basic ideas behind the research on brain–computer interfaces is that they should be operable even by people not able to produce any voluntary muscle movements. The question is, how do we simulate a disability like that on healthy, non-disabled test subjects? Unfortunately, there is no simple answer. In the first experiment, we allowed the subject to perform the joystick movements freely, without restrictions. In this second experiment, we add the restriction that no muscle movements must occur.

In doing so, we also introduce a number of problems in the paradigm. The first is that the task suddenly becomes very difficult. It is extremely hard to focus on one thought only for several seconds and not to think of anything else. Try it. The slightest lapse in concentration might ruin the trial. The second problem is that it is impossible to monitor the actions of the subject during the experiment. In the first experiment, we continuously logged the position of the joystick, and so, we could always check if the subject responded as expected to the stimuli. Erroneous responses, for example if the joystick was moved in the wrong direction could accordingly be filtered out before training the classification system.

In the second experiment, we have no way of knowing if the subject responds correctly. Remember that the neural network classifier learns by examples. With a total number of only twenty trials per class, a couple of undiscovered erroneous trials could be devastating to the training process and then in the end, the performance of the system.

7.9 Analysis of the Second Condition

The analyses performed on the data from the second experimental condition were exactly the same as for the first one. First, the optimal recording period was found, then the LAR model parameters and the neural network size. Finally, the optimal set of electrodes was created by removing channels that did not contribute to the classification rate. In four of the analyses, the results were equal to the first condition. The window width used for LAR modeling was $w_{win} = 0.25$ seconds and the optimal LAR model was of order $p = 2$, with no lag, $L = 1$. As in the first experimental condition, the differences in classification rate between various neural network sizes were small. Therefore, the same two-layer network with 30 hidden nodes was used in this analysis.

Focusing instead on the differences between the two conditions, we see in Figure 7.13 that the optimum in classification rate occurred much earlier in the process this time. The average reaction time in the first condition was 1.37 seconds, and we saw in Figure 7.3 that this corresponds very well to the optimal center of recording period. Here, the classification rate reaches its optimum for quarter second windows centered around 0.5 seconds, in other words almost a second faster than when the movements were actually performed. This is a very interesting result, which deserves, and will be given, a more thorough discussion in the next chapter.

7.9.1 Channel Selection

The other analysis that yields different results in the two conditions is the investigation of optimal channel selection. In the first condition, we found that removing one channel at the time from the classifier input actually increased the performance in three of the seven channels: P3, O1 and O2. In the second condition only one channel, O2, gives a negative contribution to the classification result, see Figure 7.14.

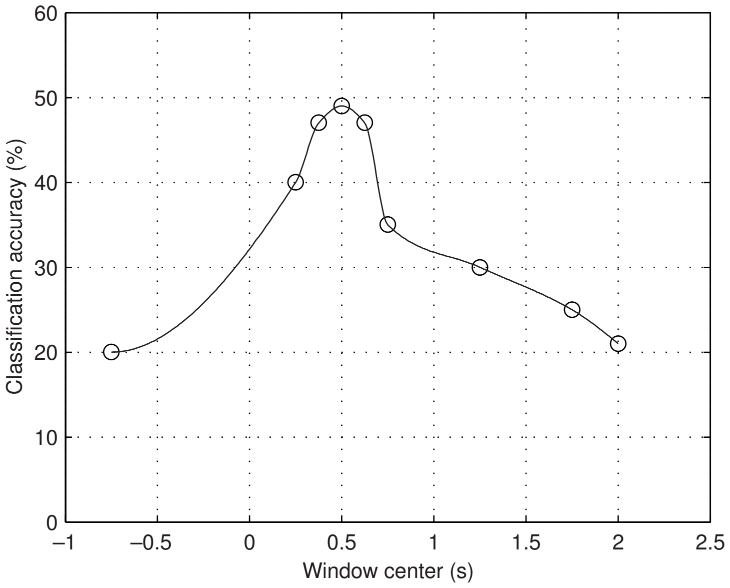


Figure 7.13: *The maximum peak in accuracy occurs earlier in this second experiment.*

Furthermore, the relative drop in accuracy when removing one channel is higher in this second condition. It seems like the level of redundancy in the information between channels is lower, now when movements are only imagined. The average change in relative classification rate was 4.26% in the first condition and 10.12% in the second.

As in the first experimental condition, this analysis was repeated after removing the electrode that gave the most negative contribution, O2. However, the only noteworthy result of that test was that all relative accuracies came out positive, indicating that the optimal electrode configuration in this second condition is the same as in the first case.

7.10 Final Results

Remember how we described the classification rate of our brain–computer interface as a real-valued function of a set of six parameters.

$$R_c = f\left(c_{win}, w_{win}, p, L, \mathbf{H}_{ann}, \mathcal{C}\right) \tag{7.7}$$

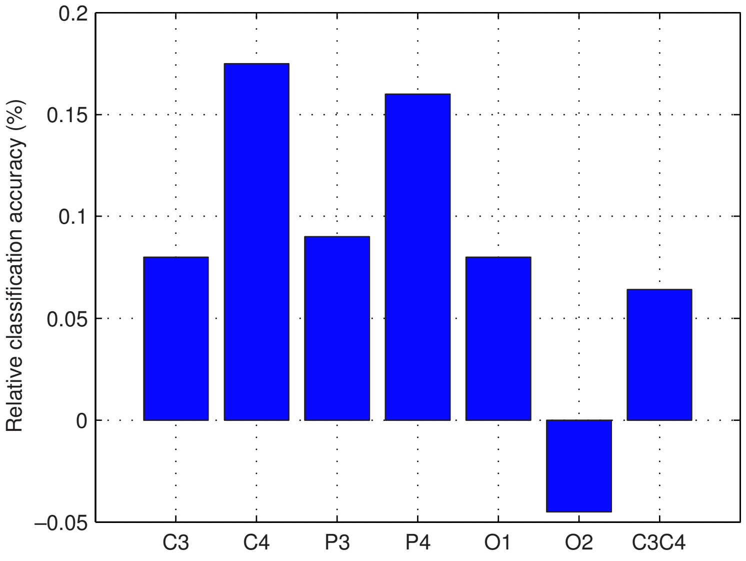


Figure 7.14: *Relative information content in the original set of channels, second condition – imagined movements.*

In the analyses of the data produced by both experimental conditions, we have now optimized f locally for each parameter. We have found the optimal recording periods, optimal LAR model parameters and the best sets of electrodes to use. We have also found neural network sizes that give a good compromise between training time and classification rate. Table 7.2 summarizes the findings.

Of course, it is not possible to guarantee that the locally optimized accuracies are equal to the global optima. And, since we do not know how the six parameters depend on each other, there is not much we can do to improve the situation either. Without resorting to random searching, that is. As the matter of fact, the only way of systematically improving the parameter settings is to go through all analyses again with the values in Table 7.2 as starting points. And then again, and again, to iteratively try to make the settings converge to stable values. However, such method would be very time consuming and most likely improve the classification rate with a few percent, or less, since the values used in the first run actually were quite close to the ones presented in Table 7.2. The important point here is that we have shown that all of the variables investigated so far, affect the final

Parameter	Real movements	Imagined movements
Window center	1.25 s	0.5 s
Window width	0.25 s	0.25 s
Model order	2	2
Lag	1	1
ANN nodes	(12 30 5)	(12 30 5)
Electrodes	{C3 C4 P3 P4 O1 C3C4}	{C3 C4 P3 P4 O1 C3C4}

Table 7.2: *Summary of the optimized parameter settings in the BCI system, based on the two experimental conditions.*

classification result, in one way or another. Some more, some less but they all contribute, and hence, the analyses of each one of them are important ingredients in the design of a successful brain–computer interface.

7.10.1 Classification Rate

The rest of this chapter will discuss the output of our classifier system, the overall performance and a few other properties. We base the discussion on two final analyses, one per experimental condition, with all parameters set according to Table 7.2. This procedure is repeated eight times for statistical validity, and the final classification rate for each repetition is noted. For the first condition, the maximum classification rate is **77.5%** correctly classified patterns. In the second one, the corresponding value is **54.4%**. These are, arguably, the two most important results of this thesis.

7.10.2 Confusion Matrix

The reason we lay so much stress on the classification rate in the discussion of the system performance is that it is a simple, well-defined and widely accepted measure that allows us to compare our result to other researchers. The truth is however, that just the classification rate alone is a quite inadequate way of presenting the performance of a classifier system; at least for systems with more than two output classes. A much more informative way is to use so called *confusion matrices.*

A confusion matrix is a class-by-class arrangement of the classifications made by the system. Each row in the grid corresponds to the *correct* class and the columns represent the *output* from the classifier. Hence, an element

η_{ij} in the grid indicate the amount of patterns belonging to class i that was recognized as class j. Ideally, a confusion matrix would be an identity matrix with ones in the diagonal and zeros otherwise. That would correspond to a perfect classification of all test patterns in all classes. This of course is hardly ever the case in a real-world situation.

The confusion matrices from the two experimental conditions are presented in Table 7.3 and 7.4. Examining the data in the matrices can tell us a great deal about the behavior of the classifiers, not only the average accuracy. We will study the data further in the discussion in the next chapter.

(%)	Base	Left	Right	Up	Down
Base	99.2	0.4	0.0	0.0	0.4
Left	3.5	89.8	0.0	5.5	1.2
Right	0.0	0.0	90.2	3.9	5.9
Up	19.1	17.2	6.3	55.9	1.6
Down	6.6	51.2	3.1	1.2	37.9

Table 7.3: *Confusion matrix of the classifications performed by the optimized BCI system, real movements.*

(%)	Base	Left	Right	Up	Down
Base	63.1	15.6	0.0	15.6	5.6
Left	3.7	6.3	38.1	33.1	18.8
Right	14.4	0.0	80.6	5.0	0.0
Up	2.5	18.7	11.9	55.0	11.9
Down	13.8	10.7	0.0	13.7	61.9

Table 7.4: *Confusion matrix of the classifications performed by the optimized BCI system, imagined movements.*

Chapter 8

Discussion

In this chapter, we discuss the results of the performed experiments and the validity of the results. We also present some method critique and talk about what could have been done differently. The final sections contain the conclusions drawn in the project and some suggestions for future work.

8.1 Interpretation of the Results

The analyses presented in Chapter 7 produced many interesting results. Some of them are fairly easy to understand and explain, some of them need, and deserve, further discussion. In this section, we will go through the achieved results one-by-one and elaborate on some of them.

8.1.1 Optimal Recording Period

The first step in the analysis procedure dealt with what part of the recorded data to use for classification. Inspecting the outcome of that analysis immediately brings us to one of the most interesting results of this thesis. The first experimental condition clearly indicate that the most important part of the data for classifying real physical movement is the period corresponding to when the movement was actually performed. The period just before the movement did not contain any classifiable information and, neither did the period following immediately after, when the joystick was just held in the indicated position.

In the second condition, when the movements were only imagined a similar peak in classification accuracy was detected. Only this time the maximum occurred almost one second earlier. Figure 8.1 illustrates the differences.

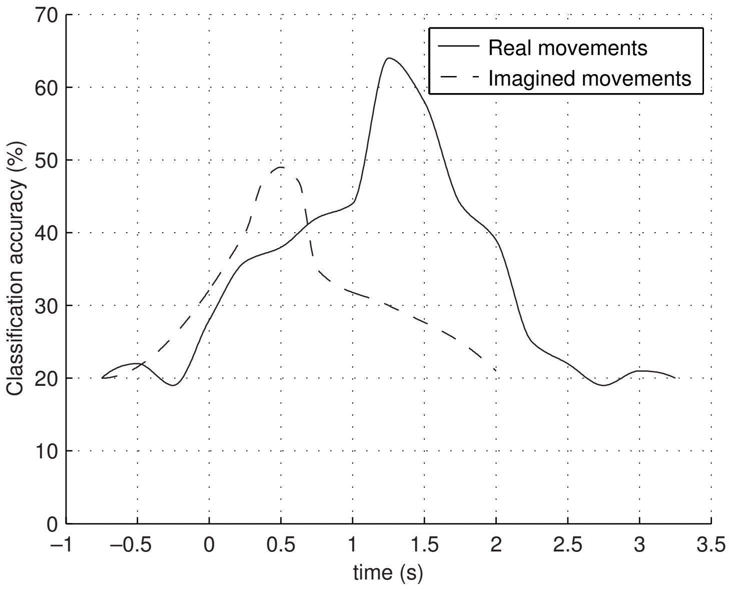


Figure 8.1: *Information content at different time intervals.*

One interpretation of this phenomenon is that the overall signal recorded during a movement, consists of several different sub-signals. One signal is correlated with the planning of the movement and can be detected when the subject is about to move the joystick, but before any real muscle contraction occurs. This part of the signal is present both during imaginary and real movements. The existence of such signal has been observed in other studies [20, 6]. Another sub-signal is related to the actual limb movement. This part is stronger from a classification point-of-view and is only present when the muscle activity occurs. In the second condition, this signal is inhibited by the user, since no physical movements were allowed. Consequently, the peak in classification accuracy for imagined movements is due to the signal extracted during the planning of the motion. This signal is detected for physical movements as well, but due to the difference in information content, it is overshadowed by the stronger, movement-related sub-signal.

8.1.2 Window Width

The optimal amount of data to use for classification was 0.25 seconds in both conditions. This is an encouraging result, since one of the major problems with existing BCI systems is the operational speed. Wolpaw [7] estimates that the most successful systems provide a transfer rate of 25 bits per minute, or less. Discrimination between four classes performed in 0.25 seconds would correspond to a theoretical transfer rate of 480 bits per minute, if we were able to perform the classifications back-to-back. In reality, successive categorizations will most likely have to be separated in time to avoid data confusion, and that will of course lower the transfer rate. Also the fact that the classification accuracy is not perfect means that the overall speed of a real-time system would be lower than the suggested maximum of 480 bits per minute. However, it is impossible to say how much lower before actually testing our ideas in an online experiment.

8.1.3 Neural Network Size

We know from Chapter 5 that the size and structure of a neural network is very important to its performance as a classifier. Small networks are bound to fail on complex non-linear problems, and large networks are prone to overfitting if the problem at hand is not very complex. Studying the architecture of a successful classifier can therefore tell us a lot about the actual problem. The best results in this study, for both real and imagined movements, were achieved using a network with 30 nodes in a single hidden layer. This is pretty much what we expected to find, and similar networks have been used in numerous other studies with good results [5, 2]. What we did not expect, however, was to get comparable results using a single layer network, since this correspond to a linear classification, and we expected our problem to be essentially non-linear. Nevertheless, a neural network with no hidden layers reached a classification rate of 60% in the first condition. This is only about 10% lower than the best performing network. The fact that the linear classifier performs so well is a positive result and means that the preprocessor is doing a good job. Remember that single layer perceptrons are only able to classify categories that can be separated by a single hyperplane. Consequently, a high classification rate using a linear classifier indicates that the preprocessor successfully transformed the input data into a pattern space where the classes are almost linearly separable. The direct consequence of this finding is a recommendation for future studies to consider linear classifiers as well as non-linear techniques.

8.1.4 LAR Model Parameters

The LAR model preprocessor is affected by two parameters: the model order p that specifies the number of extracted parameters and the lag L, which describes the separation of the taps in the model. The most interesting result of the LAR analysis is the fact that the introduction of a lag in the model ruined the classification performance completely. This is a bit confusing. Technically, using a LAR model with lag L compared to a non-lagged model corresponds to re-sampling the data with a sample rate L times lower than the original.

Throughout this experiment, we have used a sample frequency of 256 Hz. In other words, we should be able to detect components up to 128 Hz, according to the sampling theorem. However, the data received from the Grass 12 amplifier was already band pass-filtered between 0.3 and 30 Hz[1], before sampling. Hence, we should theoretically be able to go down to a sampling rate of about 60 Hz before starting to loose information. Of course, no filter is perfect, and the sampled data will most certainly contain some frequency components over 30 Hz. To avoid aliasing we therefore applied a digital anti-aliasing filter with a cutoff frequency of $f_c = 128/L$ before calculating the LAR coefficients. Still, we should be able to use a lag of up to $L = 4$ before removing any information in the frequency range below 30 Hz. Nevertheless, the accuracy drops dramatically already at $L = 2$, as illustrated in Figure 7.10. This might indicate that the recorded EEG data contain important information even in the spectra above 64 Hz. However, the matter has to be investigated in more detail before drawing any definitive conclusions. Since an experiment like that would benefit from new EEG data be collected using a higher sampling frequency and different filter settings we leave that question as a proposition for follow-up experiments.

8.1.5 Optimal Electrode Distribution

The search for the optimal electrode configuration revealed an interesting connection between the information content in an electrode and its position on the scalp. In both experimental conditions, the C4 electrode made the strongest contribution to the classification results. To understand why, we compare the location of that electrode to the map of the cerebral cortex presented in Chapter 2. The C4 electrode is mounted on the right side of the head, about 10 cm above the ear. That corresponds to the sensorimotor area of the cortex, the part of the brain that controls movements of the left side of the body. Remembering that the test subject was left-handed and hence controlled the joystick with the left hand it suddenly all makes sense.

[1]According to the technical specification of the Grass 12 amplifier about 15% of the amplitude of the frequency components around 100 Hz, passes through the internal band pass-filter when the upper cut-off frequency is set to 30 Hz.

The most important electrode for detecting movements is the one located closest to the part of the cortex that initiates the movement.

Furthermore, it appears that the information content in the other electrodes is related to their geometrical distance from the motor area, in that the relative classification rate drops as the distance increases. Figure 8.2 shows a map of the head where the intensity corresponds to the information content in different regions[2]. High intensity indicates important areas and dark color point out regions not useful for classification.

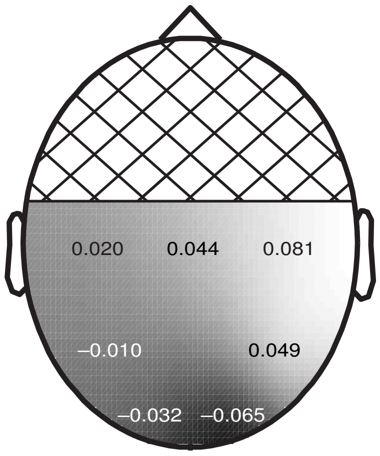


Figure 8.2: *Information content in different parts of the cortex.*

Another phenomenon that is interesting to observe is that the electrodes located close to O2, for example O1, P3 and P4, became much more important after the removal of O2. This indicates that at least some information is shared between electrodes geometrically close to each other, in a way that do not apply to electrodes on opposite sides of the head. Thus, when removing only one electrode in a region, the surrounding electrodes can partially make up for that loss due to the redundancy in information. However, removing two adjacent electrodes, for example O2 and P4, leaves a too large area of the cortex uncovered. Hence, the remaining electrodes cannot replace the loss of information, resulting in a drop in classification rate.

[2]The data presented in this case was taken from the first experimental condition. However, the result of the second condition is similar. The intensity in between the electrodes is interpolated using two-dimensional spline functions.

8.1.6 Final Results

As stated, one of the best ways to examine the performance of a classifier system is by studying the confusion matrix produced by a set of test data. Figure 8.3 presents a graphical representation of the confusion matrices generated by the two experimental conditions.

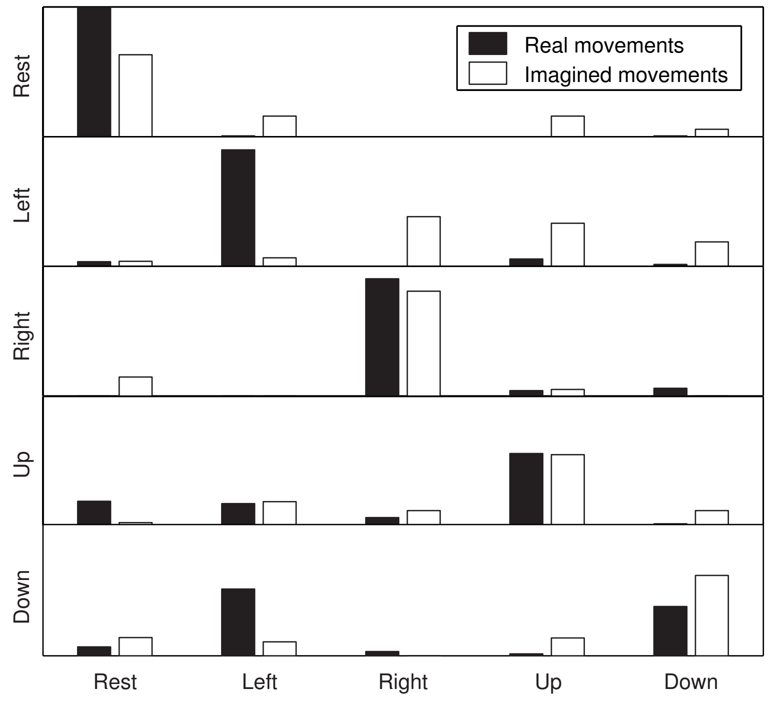


Figure 8.3: *Graphical representation of the confusion matrices obtained in the two experimental conditions. The rows in the matrix represent the correct categories and the columns the output of the classifiers.*

Focusing on real movements, we see that the classification of patterns belonging to the base class, that is, when the joystick was not moved, is almost perfect. For the test patterns recorded when the user held the joystick still our BCI system responds correctly in 99.2% of the cases. Hence, separating physical movements from rest is no problem. Neither is classification of left–right movements. Left movements are recognized with 89.8% accuracy and right movements with 90.2%. The last two classes on the other hand seem to be much harder to distinguish. When the subject moves the joystick forward, the system recognizes the produced pattern in 55.9% of the cases, and when moving it backward the accuracy drops to 37.9%.

It is a bit hard to explain why the last two classes are so much harder to recognize than the first three. One, somewhat far fetched but still possible, explanation is that the number of trials that had to be rejected due to eye blinks and other artifacts was somewhat higher in the up and down categories resulting in smaller training sets for these classes. As we saw in Section 5.9.4, reducing the number of training patterns can cause an increase in generalization error. Table 8.1 presents the number of trials in the training set and the test set for each category.

Category	Total	Rejected	Training	Test
Rest	20	0	16	2
Left	20	1	15	2
Right	20	1	15	2
Up	20	2	14	2
Down	20	3	13	2

Table 8.1: *Distribution of non-rejected trials in training and test set.*

As implied, this is a rather strained explanation, since the number of rejected trials is still very low. Nevertheless, the correlation is clear, the more rejected trials the lower the classification rate. Could there be another explanation to this? Maybe the overall quality of the recorded data is just lower in the last two classes. Maybe the distortion from muscle activities is higher when the joystick is pressed forward and backward due to activation of other muscle groups. After all, in the first class, when the user performed no movement at all, we did not have to remove any trials. This could also explain why more trials had to be rejected in the last two categories. Either way, the problem should be investigated further before we try to draw any definitive conclusions.

Instead, we turn our attention to the second experimental condition and see what results it produced. The obvious first observation is that the classification rate is in fact lower than in the first case. The maximum number of correctly classified test patterns is 54.4%, as opposed to 77.5% for the first condition. This should not be a surprise. In [17] Penny and Roberts compared classification results of a neural network based BCI system tested on both real and imagined finger movements. Their results showed that the average classification accuracy on imagined movement was significantly lower than on real movements.

The second apparent difference is the very low classification rate of patterns belonging to the category "left" in this second condition. This is what brings down the overall performance. There is no obvious explanation to this. Why should the imagined left movements be so much harder to recognize than for example right movements?

In almost forty percent of the cases patterns belonging to class left are misclassified as belonging to class right. Why is that? Is it because the two thoughts are too similar? No, it is not likely, since the reversed situation never occurs. Right movements are never mistaken for left movements.

One reasonable explanation is that one, or both, of the data sets used to train and test the neural network for left movements contained bad data. Remember that the experiment only contain 20 trials of each category. In the left-class case, four of them had to be removed initially due to eye blinks or other detectable artifacts. Twelve of the remaining trials were used for training and two for testing. With so little data, it is possible that just a couple of bad trials can corrupt the training process and make the classification accuracy drop like in the left class case here.

8.2 Are the Results Correct?

One question we should ask ourselves when presenting the results of a study like the present is, can we trust the data, are the results correct? The somewhat ambivalent answer is – yes and no.

Yes, the experiment show that EEG related to two-dimensional joystick movements can be automatically categorized both during real and imagined movements. At least, this was true for the test subject in this pilot study. However, since we have only analyzed data from one subject it is impossible to draw any conclusions about the general usefulness of the presented techniques. The EEG can vary greatly between subjects, and there is no guarantee that the method will be successful on all people.

Moreover, we do not know anything about what type of information found in the EEG signal the classifier is basing its decision on. On the other hand, that is not the important issue here. The indisputable fact is, when we are feeding the system with raw unprocessed EEG data recorded during voluntary joystick movement, the system can effectively tell us the direction of the movement. The problem of analyzing the actual classifier to see what features are the most important in the EEG is an interesting but difficult topic that is beyond the scope of this thesis. In this study, we have verified that the system has potential, and that is what we set out to prove.

8.3 Method Critique

In all experiments consisting of a data acquisition phase and an analysis phase, experimenters are likely to find in the analysis that things should have been done differently in the first phase. The present study is no exception. This section discusses a couple of problems encountered in the analysis, and what can be done to prevent them in future studies.

8.3.1 Shorter Data Recordings

The first analysis clearly demonstrates that collecting data for eight whole seconds per trial is overkill. In fact, if we knew when to initiate the readings the results indicate that a recording period of 0.25 seconds would be enough. A reasonable compromise would be to record two seconds of data starting at the point when the stimulus is presented.

8.3.2 More Training Data

Reducing the time required per stimulus would also enable us to increase the number of trials per experiment run. This would not only increase the classifiers ability to generalize, see Section 5.9, it would also reduce the risk of bad training or test data corrupting the process.

8.3.3 Reference Condition

To make it easier to discuss the validity of the results one could argue that the experiment should include a third reference condition where the same sequence of stimuli is presented to the subject, but where the task is just to relax and not performing or imagining any movements. Such data sequence can be used to ensure that the supposed movement related features extracted by the system are in fact due to movements and not induced directly by the stimuli.

8.4 Conclusions

The long-term goal of this research is to build an intuitive and reliable brain–computer interface that can be operated even by people with severe motor disabilities. In this first step of the project, we have developed an offline system for evaluation of different BCI techniques and algorithms. The system was utilized in a pilot study to investigate the feasibility of using EEG for automatic classification of voluntary hand movements. The immediate aim of the study was to see if we could find a set of BCI techniques that performed well enough to motivate further research. In the thesis introduction, we listed a number of related questions that we have strived to answer during the project. We will now briefly review those questions and summarize our findings.

- **Is it at all possible to discriminate between two-dimensional joystick movements based on EEG recordings? And if so, with what accuracy?** Yes, the results of the study indicate that the EEG signals recorded during physical joystick movements do in fact contain classifiable information about the performed movement. That information can be extracted and classified with an accuracy of 77.5%, using the techniques presented in this thesis.
- **Is it possible to make the same classification even if the movements are only imagined?** Yes, we have showed that the information can be extracted when the subject only imagines performing the motion but inhibits the actual muscle activity. This is principally important, since it means that a fully functional system could be operated even by people with motor impairments.
- **How much data is needed to be able to make a reliable classification, that is, what is the speed of the system?** The analyses indicate that a recording period of 0.25 seconds is necessary and sufficient to be able to make reliable classifications. Theoretically, this corresponds to a maximum transfer rate of 480 bits per minute.
- **What parts of the brain are important sources of EEG for such classifications? Where on the skull should the electrodes be placed?** There appears to be a connection between EEG electrode location and information content. Electrodes located near the motor centra of the cortex were observed to make a much stronger contribution to the movement classification than other electrodes on the scalp. The information content seemed to be correlated with the geometrical distance from the motor centra.

It should be stressed again that all of these results are based on the analysis on one test subject, and we cannot guarantee that the suggested techniques work for all people. Moreover, we should remember that this is an offline feasibility study with no intention of covering all problems related to real-time classifications.

Nevertheless, the results clearly demonstrate that the presented techniques of AR modeling and ANN based categorization can be used for classifying movement related EEG, given the right conditions. The final conclusion in this project is therefore that: *the suggested techniques have shown great potential, and further research on this topic can be recommended.*

8.5 Further Research

As concluded, brain–computer interfacing is an interesting area that deserves further research. The remainder of this chapter is devoted to suggestions for follow-up experiments.

8.5.1 Online System with Feedback

A natural continuation of the project described in this thesis is to extend the developed system to make it possible to perform classifications in real-time. The output of the online system should then be used to perform some observable task like moving a cursor on a computer screen or controlling a robotic arm. Previous experiments have shown that the classification accuracy of the system often increases when the user is provided with some form of direct feedback on the movements [24].

8.5.2 Higher Sample Rate and Shorter Recording Periods

In our experiment, we used a sample frequency of 256 Hz to record eight seconds long segments of EEG data per trial. The analyses of that data indicate that quarter-seconds segments contain enough information to make reliable classifications. In a new experiment, we should therefore strive to shorten the recording time per trial to be able to collect more trials per test subject. We know from Section 5.9 that more training data usually result in better generalization performance.

The recorded EEG data was band-pass filtered between 0.3 and 30 Hz before sampling. This is generally believed to be the most interesting part of the frequency spectra in an EEG signal. The analysis of LAR model parameters on the contrary indicates that the signal might contain interesting information well above 30 Hz. To be on the safe side we therefore recommend using a higher cut off frequency, and consequently, a higher sample frequency, for example 1 kHz, in future experiments.

8.5.3 Committee Machines

Another trick that can be applied to our problem to increase classification accuracy is so-called output averaging. In practice, this means that the outputs of a number of classifier systems are combined in some way to produce an overall output. The systems can be working in parallel or with a bias in time. In mathematics, this type of construction is called a *committee machine*. The combination algorithm can be simple addition, majority voting or more sophisticated statistical techniques. Simple committee machines have been used in other BCI applications with good results [21, 1]. More advanced techniques include *boosting* and so-called *hierarchical mixture of experts* [11, 26]. Applying any of these techniques on our BCI problem may help to improve classification accuracy even further.

References

[1] C.W. Anderson. Effects of variations in neural network topology and output averaging on the discrimination of mental tasks from spontaneous electroencephalogram. *Journal of Intelligent Systems*, 11(4):423–431, 1997.

[2] C.W. Anderson, S.V. Devulapalli, and E.A. Stolz. Determining mental state from EEG signals using neural networks. *Scientific Programming*, pages 171–183, 1995.

[3] C.W. Anderson, S.V. Devulapalli, and E.A. Stolz. EEG signal classification with different signal representations. In F. Girosi, J. Makhoul, E. Manolakos, and E. Wilson, editors, *Neural Networks for Signal Processing*, pages 475–483. IEEE Service Center, Piscataway, NJ, 1995.

[4] C.W. Anderson and Z. Sijercic. Classification of EEG signals from four subjects during five mental tasks. In A.B. Bulsari, S. Kallio, and D. Tsaptsinos, editors, *Solving Engineering problems with Neural Networks, Proceedings of the Conference on Engineering Applications in Neural Networks (EANN'96)*, pages 407–414, 1996.

[5] C.W. Anderson, E.A. Stolz, and S. Shamsunder. Multivariate autoregressive models for classification of spontaneous electroencephalogram during mental tasks. *IEEE Transactions on Biomedical Engineering*, pages 277–286, 1998.

[6] B. Blankertz, G. Curio, and K. Muller. Classifying single trial EEG: Towards brain computer interfacing. In S. Becker T. G. Diettrich and Z. Ghahramani, editors, *Advances in Neural Information Processing Systems (NIPS 01)*, volume 14, pages 157–164, 2002.

[7] J.R. Wolpaw et al. Brain–computer interface technology: A review of the first international meeting. *IEEE Transactions on Rehabiltation Engineering*, 2(8):164–173, June 2000.

[8] A.S. Gevins. and N.H. Morgan. Applications of neural-network (NN) signal processing in brain research. *IEEE ASSP Transactions*, 7(36):1152–1161, 1988.

[9] E. Haselsteiner. *Neural Based Methods for Time Series Classification.* PhD thesis, Technischen Universität, Graz, June 2000.

[10] M.H. Hayes. *Statistical Digital Signal Processing and Modeling.* John Wiley & Sons, Inc., 1996.

[11] S. Haykin. *Neural Networks: A Comprehensive Foundation.* Prentice Hall, 1999.

[12] N. Hazarika, A. Chung Tsoi, and A. Sergejew. Nonlinear considerations in EEG signal classification. *IEEE Transactions on Signal Processing*, (45):829–836, 1997.

[13] H. Jasper. The ten-twenty electrode system of the international federation. *Electroencephalography and Clinical Neurophysiology*, (20):371–375, 1958.

[14] A.R. Luria. *Higher Cortical Functions in Man.* Basic Books, New York, 1966.

[15] J. Pardey, S.J. Roberts, and L. Tarassenko. A review of parametric modelling techniques for EEG analysis. *Medical Engineering and Physics*, 18(1):2–11, 1996.

[16] W. Penfield and T. Rasmussen. *The Cerebral Cortex of Man.* Macmillan, New York, 1950.

[17] W.D. Penny and S.J. Roberts. Bayesian neural networks for classification: how useful is the evidence framework? *Neural Networks*, 12(6):877–892, 1999.

[18] W.D. Penny and S.J. Roberts. EEG-based communication via dynamic neural network models. In *International Joint Conference on Neural Networks (IJCNN)*, 1999.

[19] W.D. Penny and S.J. Roberts. Experiments with an EEG-based computer interface. Technical report, Imperial College, Albany, USA, June 1999.

[20] W.D. Penny, S.J. Roberts, and M. Stokes. Imagined hand movements identified from the EEG mu-rhythm. *Journal of Neuroscience Methods*, 1998.

[21] B.O. Peters, G. Pfurtscheller, and H. Flyvbjerg. Automatic differentiation of multichannel EEG signals. *IEEE Transactions on Biomedical Engineering*, 1(48):111–116, 2001.

[22] L. Prechelt. Early stopping – but when? In G.B. Orr, editor, *Neural Networks: Tricks of the Trade*, pages 55–69. Springer-Verlag, 1996.

[23] T. Jackson R. Beale. *Neural Computing an Introduction.* Institute of Physics Publishing, Bristol, 1990.

[24] H. Ramoser, J. Müller-Gerking, and G. Pfurtscheller. Optimal spatial filters for discrimination of single trial EEG recorded during imagined hand movement. *IEEE Transaction on Rehabilitation Engeneering*, 8:441–446, December 2000.

[25] S.J. Roberts and W.D. Penny. Real-time brain-computer interfacing: A preliminary study using bayesian learning. *Medical and Biological Engineering and Computing*, 1999.

[26] A.J.C. Sharkey. *Combining Artificial Neural Nets.* Springer, 1999.

[27] T.M. Vaughn, J.R. Wolpaw, and E. Donchin. EEG-based communication: Prospects and problems. *IEEE Transactions on Rehabilitation Engineering*, 4(4), 1996.

[28] J.R. Wolpaw. Brain–computer interfaces for communication and control. Technical report, Wadsworth Center, New York State Department of Health and State University of New York at Albany, 2001.

Appendix A

The Developed BCI System

One of several purposes of this project was to develop a general tool for further research on the techniques of brain–computer interfacing. The requirements on the system were that it should be fully functional, easy to extend and modify, and include all essential modules normally contained in an offline BCI system. The result of the work is a Windows based[1] analysis tool that handles all the links in the chain of brain–computer interfacing from recording, via preprocessing and classification to post-processing. In this chapter, we will take a brief look at the functionality of the software and the hardware used with it. We will do that by reviewing the steps of the first experimental condition from an operator's point-of-view.

A.1 Structure of the System

The developed software actually consists of two separate programs: the main BCI application and a stimuli presentation program. The system can be run on a single PC or on two separate computers communicating over a TCP/IP network, as described in Figure A.1.

A.1.1 BCI Server

The major part of the system is controlled by the server application. The server tells the stimuli client when to present a stimulus, and it also initiates the recordings of the data through the EEG recorder. After the reading is completed, the BCI server handles the preprocessing and the classification

[1]Both the server application and the stimuli client are written in C++. The software was developed using Microsoft Visual Studio 6.0 under Windows 2000. The total amount of code is about 30.000 lines.

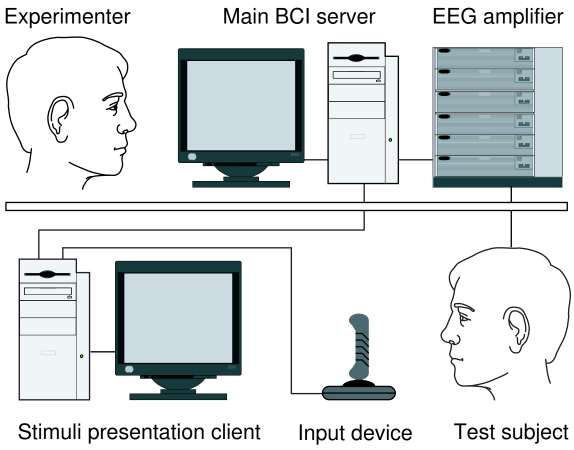


Figure A.1: *The structure of the BCI system used in the experiment.*

of the data. The server is the most important part of the system, and therefore, this chapter will be very much focused on the functionality of this application.

A.1.2 Stimuli Presentation Client

The main purpose of the stimuli client is to present visual stimuli to the test subject, as dictated by the server. The client has built-in support for stimuli consisting of simple geometrical figures, like the arrows used in this project, but it can also be extended through a simple script language to show arbitrary pictures represented as bitmaps. The script language can also be used to incorporate acoustic signals in the stimuli. In our experiment, we used this feature to indicate the beginning and the end of a trial, to tell the subject when to be alert and when to relax.

A.1.3 EEG Recorder

The recorder system used in this project was a Model 12 amplifier from Grass-Telefactor combined with a NIDAQ 6023E data acquisition card from National Instruments. The Grass system, shown in Figure A.2, has 16 single-channel amplifiers with built-in band-pass filters and automatic electrode impedance control. The actual recording of the amplified and filtered data was handled by the I/O card, which was mounted in the server PC. The card simultaneously samples up to 16 channels of data with 12-bit precision and communicates with the BCI application through the PCI bus of the computer.

Figure A.2: *A Grass model 12 unit was used for amplifying and filtering the EEG.*

A.2 Experiment Walkthrough

To demonstrate some of the functionality of the BCI application we will now go back to the point of the first experimental condition and walk through the setup, the experiment and the data analysis, step-by-step. Remember that the purpose of the experiment is to categorize real two-dimensional joystick movements based on EEG recordings.

A.2.1 Hardware Setup

First, the system has to be set up according to Figure A.1 with two computers and an EEG amplifier. Seven EEG electrodes are mounted on the skull of the test subject at the positions Cz, C3, C4, P3, P4, O1 and O2 as defined by the 10–20 system. Two additional EOG electrodes mounted close to the subject's right eye are used to detect eye blink artifacts. All electrodes are connected to the Grass amplifier and combined to form eight channels of data according to Table 6.1.

The stimuli presented to the test subject consist of arrows pointing in four different directions *left*, *right*, *up* and *down* as exemplified in Figure A.3. As described, the task for the test subject is to move a joystick in the direction indicated by the arrow. If no arrow is presented the task is to simply relax and try not to move at all.

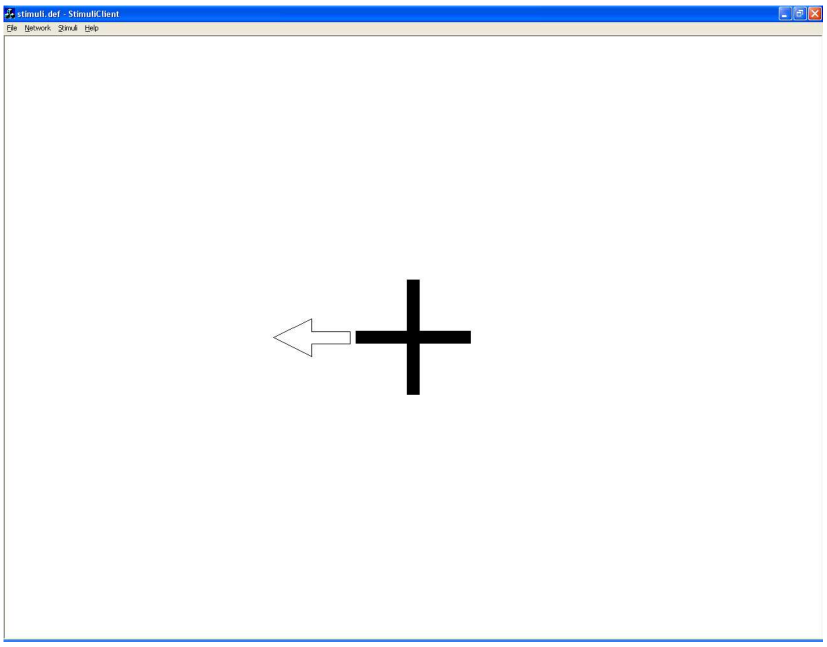


Figure A.3: *The stimuli presentation client.*

A.2.2 Offline Experiment

The BCI system described in this thesis is an offline analysis tool. This means that the process of collecting the EEG data is separated from the rest of the analysis. No classification is done in real-time. Figure A.4 illustrates the procedure of a typical offline experiment.

First, the EEG data is collected in the recording process. This is the only phase of the experiment that involves the test subject and, hence, it is the only part that has to be performed in real-time. During the recording, a number of stimuli are presented to the subject. He or she responds by

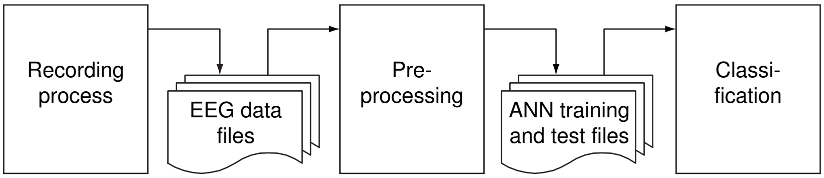


Figure A.4: *Principle of an offline analysis tool.*

moving the joystick and, during that movement, the subject's brain waves are recorded. The collected EEG is stored in files for later processing, one file per stimulus. To avoid distorted recordings corrupting the classification process all files are manually inspected for artifacts. Trials that contain for example eye blinks are removed.

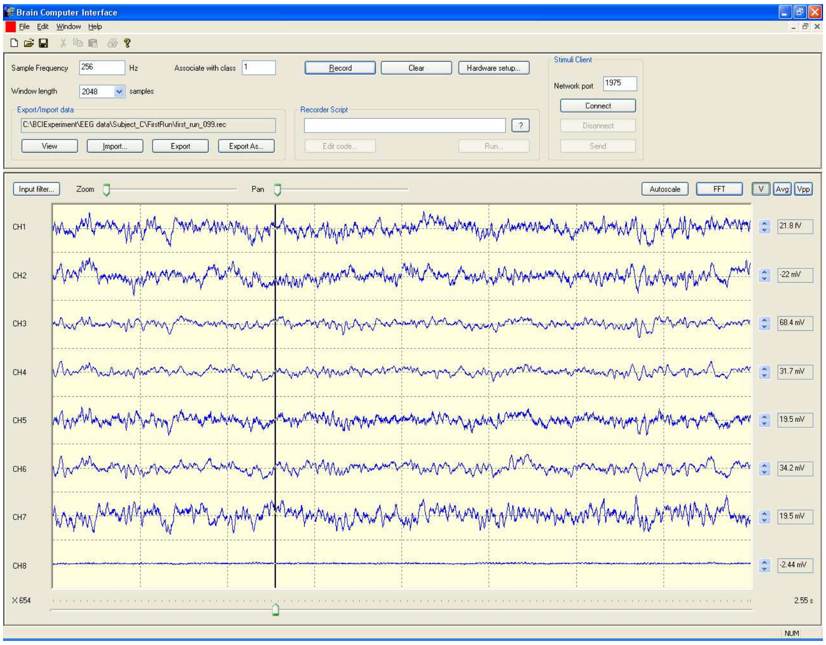
The purpose of the preprocessing phase is to load the remaining EEG data, convert it, file by file, and store the output as training examples for the classification module.

Finally, the classifier module loads the training files and starts learning the examples contained within it. Throughout the learning process, the performance of the classifier is tested against a set of test examples produced in the same way as the training patterns. The result is presented continuously, and the training is stopped when a sufficient classification performance is reached.

A.3 The Recorder Module

The recording phase of the experiment is controlled by the recorder module in the server application. Figure A.5 shows a screenshot of the module just after the reading of eight seconds of data.

The recorder is the control panel from where the first part of the experiment is managed. Here we can setup the hardware, initiate data readings and send commands to the stimuli client. The recorded data is presented graphically in the same way as on a paper strip chart. A built-in DSP tool makes it possible to analyze data online. Analyzes include digital low-pass, high-pass, band-pass and band-stop filters, Fourier transforms (FFT), as well as functionality for rescaling, calculation of statistical properties etc. The processed data can be saved to file using a simple text-based format that can be easily imported into other signal processing software like Matlab and LabView. It is of course possible to import data the same way.



Sample Frequency 256 Hz
Associate with class 1
Window length 2048 samples
Network port 1975
C:\BCIExperiment\EEG data\Subject_C\FirstRun\first_run_099.rec
21.8 fV
-22 mV
68.4 mV
31.7 mV
19.5 mV
34.2 mV
19.5 mV
-2.44 mV
X 654
2.55 s

Figure A.5: *The recorder module.*

A.3.1 Manual Initiation of Recordings

As described, each trial in the experiment can be started from the recorder module. To do this we first connect to the stimuli client through the port specified in the field "Network port". When the connection has been established we can remotely present stimuli to the subject by entering the predefined number of the stimulus in the field marked "Associate with class" and pressing the "Send" button. Exactly what stimulus will be presented and how long it will last is defined by the script file running on the stimuli client, as described in the previous section. The actual recording is initiated by entering the sample frequency and the total number of samples to read in the two fields in the upper left corner of the module and then pressing the "Record" button. When the scanning is completed, the data can be saved to file using the "Export" function.

A.3.2 Automatic Initiation of Recordings

In a real experiment, we typically want to present the subject with a large number of stimuli in a sequence and record the EEG during every one of them. That would mean repeating the procedure described above; presenting stimuli, recording and saving the data, over and over again. To simplify this task the server includes a simple automation functionality that makes it possible to program the behavior of the system through a script file. The listing A.1 below shows what such file might look like for our experiment.

Program A.1 BCI Server Automation Script

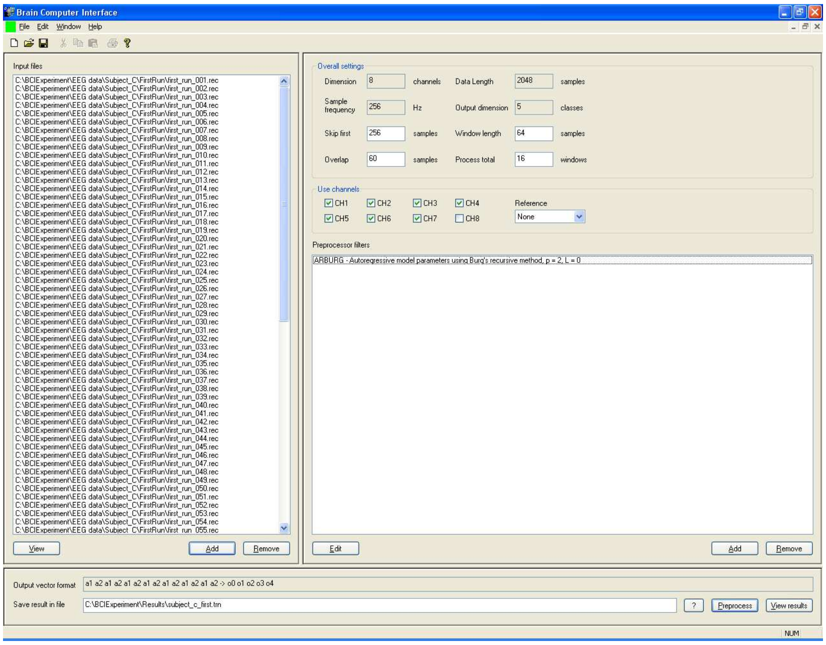
```
# Script file for BCI Experiment
# Author:  Pontus Forslund
# Date:  2001-07-08
RESET # Reset counter
CONNECT 1975 # Connect to stimuli client through port 1975
MESSAGE "Press OK to start " # Wait for experimenter to start session
SLEEP 10000 # 10s delay.  To give experimenter a chance to leave the room
# Start recording session
AUTOREC 2048 256 2 E:\BCIProject\Result\result_\@.rec 4000
AUTOREC 2048 256 3 E:\BCIProject\Result\result_\@.rec 4000
AUTOREC 2048 256 1 E:\BCIProject\Result\result_\@.rec 4000
AUTOREC 2048 256 0 E:\BCIProject\Result\result_\@.rec 4000
AUTOREC 2048 256 3 E:\BCIProject\Result\result_\@.rec 4000
AUTOREC 2048 256 0 E:\BCIProject\Result\result_\@.rec 4000
AUTOREC 2048 256 1 E:\BCIProject\Result\result_\@.rec 4000
AUTOREC 2048 256 2 E:\BCIProject\Result\result_\@.rec 4000
AUTOREC 2048 256 1 E:\BCIProject\Result\result_\@.rec 4000
...
DISCONNECT # Close the network connection
MESSAGE "Experiment completed" # Experiment finished
```

The script code is rather self-explaining, except for the perhaps most important command, `AUTOREC`. Each call to `AUTOREC` corresponds to one trial in the experiment. In this case a total of 2048 samples of data are recorded each trial with a sample frequency of 256 Hz. The stimulus presented to the subject is defined by the third argument, and the fourth argument tells the server where to save the results. The tag `\@` is an integer variable that is reset at the beginning of the program and then incremented on each encounter to give all result files unique names. All trials are followed by a 4000 milliseconds delay before the next trial.

A.4 The Preprocessor

Once the recording is completed and all data is saved to files, the preprocessing phase can begin. The fist step is normally to remove all trials that contain data contaminated by artifacts like eye blinks or other distortions. The easiest way to do this is by visually inspecting the data in the recorder module and simply delete all files that contain corrupted samples.

The remaining files are then loaded into the preprocessor module for feature extraction. Figure A.6 shows a screenshot of the module. The fixed parameters in the box "overall settings", tells us that each loaded file contain eight channels of data consisting of 2048 samples recorded with a frequency of 256 Hz. All trials are associated with one of five outputs corresponding to the movement performed by the subject during the recording.



Input files

```
C:\BCIExperiment\EEG data\Subject_C\FirstRun\first_run_001.rec
C:\BCIExperiment\EEG data\Subject_C\FirstRun\first_run_002.rec
C:\BCIExperiment\EEG data\Subject_C\FirstRun\first_run_003.rec
C:\BCIExperiment\EEG data\Subject_C\FirstRun\first_run_004.rec
C:\BCIExperiment\EEG data\Subject_C\FirstRun\first_run_005.rec
C:\BCIExperiment\EEG data\Subject_C\FirstRun\first_run_006.rec
C:\BCIExperiment\EEG data\Subject_C\FirstRun\first_run_007.rec
C:\BCIExperiment\EEG data\Subject_C\FirstRun\first_run_008.rec
C:\BCIExperiment\EEG data\Subject_C\FirstRun\first_run_009.rec
C:\BCIExperiment\EEG data\Subject_C\FirstRun\first_run_010.rec
C:\BCIExperiment\EEG data\Subject_C\FirstRun\first_run_011.rec
C:\BCIExperiment\EEG data\Subject_C\FirstRun\first_run_012.rec
C:\BCIExperiment\EEG data\Subject_C\FirstRun\first_run_013.rec
C:\BCIExperiment\EEG data\Subject_C\FirstRun\first_run_014.rec
C:\BCIExperiment\EEG data\Subject_C\FirstRun\first_run_015.rec
C:\BCIExperiment\EEG data\Subject_C\FirstRun\first_run_016.rec
C:\BCIExperiment\EEG data\Subject_C\FirstRun\first_run_017.rec
C:\BCIExperiment\EEG data\Subject_C\FirstRun\first_run_018.rec
C:\BCIExperiment\EEG data\Subject_C\FirstRun\first_run_019.rec
C:\BCIExperiment\EEG data\Subject_C\FirstRun\first_run_020.rec
C:\BCIExperiment\EEG data\Subject_C\FirstRun\first_run_021.rec
C:\BCIExperiment\EEG data\Subject_C\FirstRun\first_run_022.rec
C:\BCIExperiment\EEG data\Subject_C\FirstRun\first_run_023.rec
C:\BCIExperiment\EEG data\Subject_C\FirstRun\first_run_024.rec
C:\BCIExperiment\EEG data\Subject_C\FirstRun\first_run_025.rec
C:\BCIExperiment\EEG data\Subject_C\FirstRun\first_run_026.rec
C:\BCIExperiment\EEG data\Subject_C\FirstRun\first_run_027.rec
C:\BCIExperiment\EEG data\Subject_C\FirstRun\first_run_028.rec
C:\BCIExperiment\EEG data\Subject_C\FirstRun\first_run_029.rec
C:\BCIExperiment\EEG data\Subject_C\FirstRun\first_run_030.rec
C:\BCIExperiment\EEG data\Subject_C\FirstRun\first_run_031.rec
C:\BCIExperiment\EEG data\Subject_C\FirstRun\first_run_032.rec
C:\BCIExperiment\EEG data\Subject_C\FirstRun\first_run_033.rec
C:\BCIExperiment\EEG data\Subject_C\FirstRun\first_run_034.rec
C:\BCIExperiment\EEG data\Subject_C\FirstRun\first_run_035.rec
C:\BCIExperiment\EEG data\Subject_C\FirstRun\first_run_036.rec
C:\BCIExperiment\EEG data\Subject_C\FirstRun\first_run_037.rec
C:\BCIExperiment\EEG data\Subject_C\FirstRun\first_run_038.rec
C:\BCIExperiment\EEG data\Subject_C\FirstRun\first_run_039.rec
C:\BCIExperiment\EEG data\Subject_C\FirstRun\first_run_040.rec
C:\BCIExperiment\EEG data\Subject_C\FirstRun\first_run_041.rec
C:\BCIExperiment\EEG data\Subject_C\FirstRun\first_run_042.rec
C:\BCIExperiment\EEG data\Subject_C\FirstRun\first_run_043.rec
C:\BCIExperiment\EEG data\Subject_C\FirstRun\first_run_044.rec
C:\BCIExperiment\EEG data\Subject_C\FirstRun\first_run_045.rec
C:\BCIExperiment\EEG data\Subject_C\FirstRun\first_run_046.rec
C:\BCIExperiment\EEG data\Subject_C\FirstRun\first_run_047.rec
C:\BCIExperiment\EEG data\Subject_C\FirstRun\first_run_048.rec
C:\BCIExperiment\EEG data\Subject_C\FirstRun\first_run_049.rec
C:\BCIExperiment\EEG data\Subject_C\FirstRun\first_run_050.rec
C:\BCIExperiment\EEG data\Subject_C\FirstRun\first_run_051.rec
C:\BCIExperiment\EEG data\Subject_C\FirstRun\first_run_052.rec
C:\BCIExperiment\EEG data\Subject_C\FirstRun\first_run_053.rec
C:\BCIExperiment\EEG data\Subject_C\FirstRun\first_run_054.rec
C:\BCIExperiment\EEG data\Subject_C\FirstRun\first_run_055.rec
```

Overall settings

Dimension	8	channels	Data Length	2048	samples
Sample frequency	256	Hz	Output dimension	5	classes
Skip first	256	samples	Window length	64	samples
Overlap	60	samples	Process total	16	windows

Use channels

✓ CH1	✓ CH2	✓ CH3	✓ CH4	Reference
✓ CH5	✓ CH6	✓ CH7	☐ CH8	None

Preprocessor filters

ARBURG - Autoregressive model parameters using Burg's recursive method, p = 2, L = 0

Output vector format: a1 a2 a1 a2 a1 a2 a1 a2 a1 a2 a1 a2 a1 a2 -> o0 o1 o2 o3 o4

Save result in file: C:\BCIExperiment\Results\subject_c_first.trn

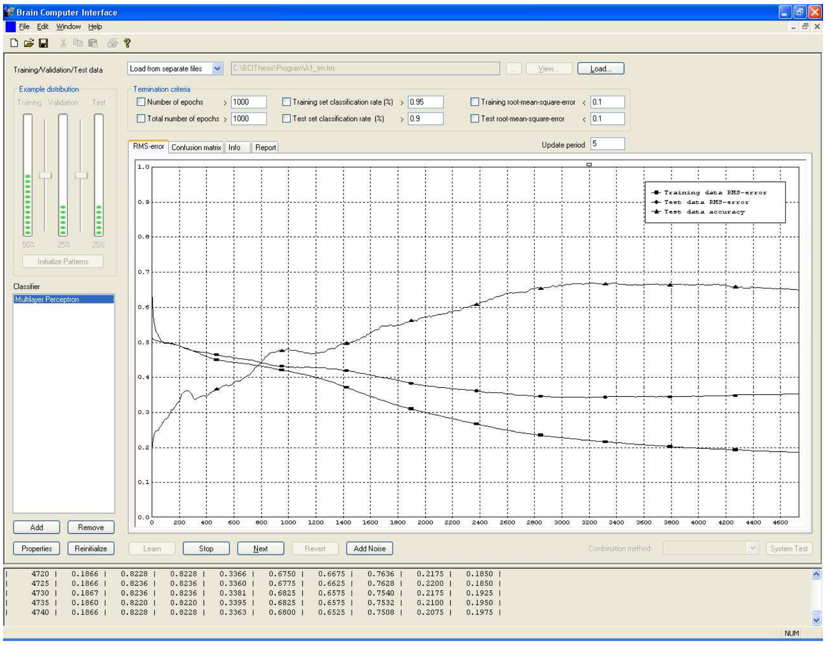
Figure A.6: *The preprocessor module.*

In this example the data recorded during each trial is divided into 16 windows of 64 samples. Each window overlaps the previous by 60 samples, and the first window starts 256 samples into the recorded sequence. The last channel is only used to detect blink artifacts and is therefore excluded from further processing.

The module can handle several preprocessor filters in parallel, but in this example, we are using only one; a second order autoregressive model with no lag computed using Burg's algorithm, as described in Chapter 4. Burg's model extracts two AR coefficients from each channel producing a total of fourteen features per window. These features are concatenated into a vector and stored in a file together with the movement class associated with the trial. Hence, the output from the preprocessor is a file containing a number of input–output pattern pairs, one pair per window. This file can later be used to train or test the classifier in the last module. Normally the set of artifact free trials is divided into two or three parts that are preprocessed separately. One set is used to produce a file of training examples. The other sets, which are usually smaller, produces test and validation patterns that are used to measure the performance of the classifier as training progresses.

A.5 The Classifier

The last component of the BCI software is the classifier module, shown in Figure A.7. Here the actual classification of the extracted features takes place. The classifier is the most important, but also the most complex module, and in this section, we will only describe a subset of its features.



Training/Validation/Test data: Load from separate files — C:\BCIThesis\Program\A1_trn.trn — ... View... Load...

Number of epochs > 1000
Total number of epochs > 1000
Training set classification rate (%) > 0.95
Test set classification rate (%) > 0.9
Training root-mean-square-error < 0.1
Test root-mean-square-error < 0.1
Update period 5
Classifier: Multilayer Perceptron

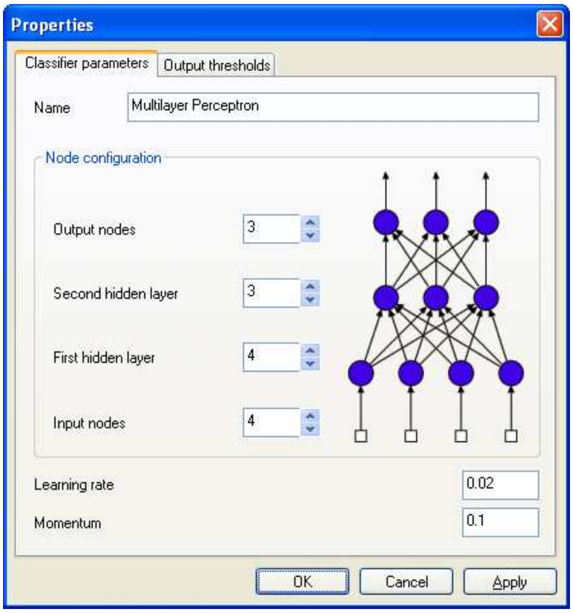
```
|      4720 |    0.1866 |    0.8228 |    0.8228 |    0.3366 |    0.6750 |    0.6675 |    0.7636 |    0.2175 |    0.1850 |
|      4725 |    0.1866 |    0.8236 |    0.8236 |    0.3360 |    0.6775 |    0.6625 |    0.7628 |    0.2200 |    0.1850 |
|      4730 |    0.1867 |    0.8236 |    0.8236 |    0.3381 |    0.6825 |    0.6575 |    0.7540 |    0.2175 |    0.1925 |
|      4735 |    0.1860 |    0.8220 |    0.8220 |    0.3395 |    0.6825 |    0.6575 |    0.7532 |    0.2100 |    0.1950 |
|      4740 |    0.1866 |    0.8228 |    0.8228 |    0.3363 |    0.6800 |    0.6525 |    0.7508 |    0.2075 |    0.1975 |
```

Figure A.7: *The classifier module.*

Let us assume that the preprocessing phase produced two files of input–output patterns; one file based on 90 percent of the artifact free trials and one file based on the remaining 10 percent. The first file is used to train the classifier and the other to test its performance.

After loading the training and test files, we should decide what type of classifier to use. The program is designed to handle several types of adaptive classifiers, like different kinds of neural networks, genetic algorithms etc, and even combinations of them. However, in this first version of the software the only implemented paradigm is the multilayer perceptron described in Section 5.6.

To create an MLP network we simply select the “Add” button and then pick the MLP from an appearing list of available classifiers. Upon creation of a classifier, a dialog box with the properties associated with it appears, as shown in Figure A.8. In the MLP case, we are asked to specify the learning rate and the momentum as well as the number of nodes in the hidden layers. The number of nodes in the input and the output layers are already specified by the dimension of the training patterns. In our example, the preprocessor produced 14 features per window, and therefore, the network has to have 14 input nodes. The output nodes correspond to the five target classes for the classification; rest, left, right, up and down, and hence, the network has five output nodes.


Properties

Name: Multilayer Perceptron

Node configuration

Output nodes: 3

Second hidden layer: 3

First hidden layer: 4

Input nodes: 4

Learning rate: 0.02

Momentum: 0.1

Figure A.8: *The multilayer perceptron setup.*

Once the classifier is created, the training of the network can be initiated by pressing the “Learn” button. Figure A.7 shows a screenshot of the classifier module after the training has proceeded for a while. The lines marked with boxes and circles in the graph window show the networks training and test error respectively. As we expect, both error curves start at about 0.5 for a completely untrained network and then decrease as the training proceeds. The third curve, marked with triangles, describes the classification accuracy based on the test set, and it can accordingly been seen as the generalization performance of the classifier. Naturally, the accuracy at the beginning of the training session is around 20 percent, since it corresponds to a random guess between five categories. However, as the network learns the problem at hand, the generalization capability quickly improves, and after about 3200 epochs, it reaches its maximum. After that point, the network starts to overfit the training data, resulting in a drop in test accuracy. To avoid the problem of overfitting the state of the network is automatically saved every time the generalization accuracy, based on either the test set or the validation set, reaches a maximum. This point is marked out with a small square just above the graph area. This way we can always go back to the best state encountered so far by pressing the "Revert" button.

The button marked "Add noise" is used for rescuing networks that have been trapped in local minima. Remember that the backpropagation algorithm always looks for an optimum in the direction of the steepest descent of the training error. Accordingly, if the algorithm finds a local non-optimal minimum, chances are that it will never find its way out of there – a least not without help. The add-noise-functionality provides that help by effectively pushing the operating point of the algorithm a small step in some random direction. The purpose is of course to get the network out of the local minimum and hopefully on the track towards a global minimum.

After the training is completed, the program automatically generates a report with essential information about the session such as network settings, classification rates, generalization errors and confusion matrices. It is also possible to study the behavior of the network for each input pattern individually. This functionality can be an invaluable tool when analyzing the actual classifier and trying to extract information from the trained neural network. Such analysis, however, is regarded a highly complicated task [4] and was considered out of the scope of this project.

	Avdelning, Institution Division, Department Industrial Ergonomics, Dept. of Mechanical Engineering	**Datum** Date December 16, 2003

Språk Language ☐ Svenska/Swedish ☒ Engelska/English ☐ ______________	**Rapporttyp** Report category ☐ Licentiatavhandling ☒ Examensarbete ☐ C-uppsats ☐ D-uppsats ☐ Övrig rapport ☐ ______________	**ISBN** — **ISRN** — **Serietitel och serienummer** Title of series, numbering **ISSN** — LITH-IKP-EX-2107
URL för elektronisk version		

Titel	Klassificering av rörelserelaterade EEG signaler med neurala nätverk
Title	A Neural Network Based Brain–Computer Interface for Classification of Movement Related EEG
Författare Author	Pontus Forslund

Sammanfattning
Abstract

A brain–computer interface, BCI, is a technical system that allows a person to control the external world without relying on muscle activity. This thesis presents an EEG based BCI designed for automatic classification of two dimensional hand movements. The long-term goal of the project is to build an intuitive communication system for operation by people with severe motor impairments. If successful, such system could for example be used by a paralyzed patient to control a word processor or a wheelchair.

The developed BCI was tested in an offline pilot study. In response to an external cue, a test subject moved a joystick in one of four directions. During the movement, EEG was recorded from seven electrodes mounted on the subject's scalp. An autoregressive model was fitted to the data, and the extracted coefficients were used as input features to a neural network based classifier. The classifier was trained to recognize the direction of the movements. During the first half of the experiment, real physical movements were performed. In the second half, subjects were instructed just to imagine the hand moving the joystick, but to avoid any muscle activity.

The results of the experiment indicate that the EEG signals *do* in fact contain extractable and classifiable information about the performed movements, during both physical and imagined movements.

Nyckelord
Keywords

Brain–Computer Interface, Neural Networks, EEG, Autoregressive modeling