NeuCube Spiking Neural Network Algorithm for EEG Signals Decoding on SpiNNaker Neuromorphic Hardware

MASTER THESIS

submitted by
Jan Behrenbeck

born on: 27.10.1992

Schleißheimer Straße 88
80797 München
Tel.: +49 15255645640
E-Mail: jan.behrenbeck@tum.de

NEUROSCIENTIFIC SYSTEM THEORY
Technische Universität München

Prof. Dr. Jörg Conradt

Supervisor:	Dipl.-Ing. Zied Tayeb
Co-Supervisors:	Dr. Christoph Richter
	Dr. Oliver Rhodes
Start:	09.10.2017
Intermediate Presentation:	29.01.2018
Final Submission:	09.04.2018

In your final hardback copy, replace this page with the signed exercise sheet.

Abstract

Spatio-temporal brain data collected via Electroencephalography can be used in Motor Imagery-based Brain Computer Interfaces to support disabled patients and improve their mobility. Subjects can either use rehabilitation robots for active physio therapy without professional therapeutic assistance, or directly control robot arms or prostheses with their thoughts. Nonetheless, the classification of brain activity is challenging as discriminative features must be extracted for classification and knowledge about these features is limited.

NeuCube is a specialized, biologically inspired liquid state machine for modelling and recognition of complex spatio-temporal signals correlation using a network of spiking neurons. It is advantageous as it neither requires an upstream feature extraction stage nor a-priori knowledge about the input data.

This work provides an adapted, publicly available Python implementation of the NeuCube algorithm using SpiNNaker neuromorphic hardware. The implementation comprises three stages: an encoding stage, which converts Electroencephalography signals into spike trains; a three dimensional brain-shaped recurrent Spiking Neural Network reservoir, which filters spatio-temporal correlations from the data; and an output classifier, which is trained on class-specific reservoir dynamics and predicts class labels for unknown samples. This thesis discloses the internal structure and characteristics of the model and introduces metrics to evaluate the functionality of the system. The reported experiments examine and evaluate the system on a modular as well as an integrated level and result in a successful classification of both 4-class Electromyography and 2-class Electroencephalography data with a mean accuracy of 75% and 65% respectively.

Notwithstanding the functionality of the system was validated, the results remain inferior to those of traditional classifiers and other NeuCube implementations. The biggest challenge in improving performance is tuning a large set of parameters that govern the internal dynamics of the system. Finally, a path towards the application of NeuCube-based Brain Computer Interfaces in a rehabilitation context is suggested and commented with respect to its ethical implications.

I would like to thank my supervisors Zied Tayeb, Christoph Richter and Oliver Rhodes for guiding and inspiring me during this project. You gave me valuable feedback and helped to advance my academic way of thinking.

Contents

Chapter 1

Introduction

1.1 Motor Imagery-based Brain Computer Interfaces for Rehabilitation

The motor system of the human body is based on the complex interaction between the central nervous system, the peripheral nervous system and the anatomical constitution of bones, muscles, and tendons. From a technological point of view the system is comprised of information processing, signal propagation, and task execution. However, those stages are distributed and overlap anatomically, which makes it difficult to locate and isolate functional elements particularly within the information processing chain.
Due to illnesses and neurological disorders such as muscular dystrophy, nerve infections, or strokes, or the impact of external forces for example during accidents the motor system can be damaged [44]. Specific consequences reach from degenerated or interrupted nerve connections and muscles to the loss of entire limbs. Depending on the location and severity of the damage, it can lead to a major impairment of the subject's mobility. Some of those impairments are irreversible and result in permanent functional limitations. In this case patients either need support of other people to compensate their inability or use prostheses or orthoses to regain their mobility to some extent. In other cases, these impairments can be partially or completely reversed by physio therapeutic treatment [18, 41].
The treatment is most effective when the patient takes an active part in training. Although patients may have some residual muscle activity, they often fail to recruit enough motor units at an appropriate speed and pattern to generate sufficient force to complete the desired movement on their own. Professional physiotherapists can help by moving the patient's limbs manually. However, the employment of physio therapists is time consuming and expensive. As the imagination of movements results in similar cortical activity as their execution [80], Motor Imagery (MI), or the mental rehearsal of movements, is another approach used by rehabilitation professionals to encourage motor practice in the absence of sufficient muscle activity. Even

though recovery of movement control is greater after motor execution training than after MI training, the combination of MI and the appropriate passive movement can improve training effectiveness [60]. Therefore, rehabilitation robots can be used to move patient's limbs via functional electrical muscle stimulation, which reduces the need for therapeutic professionals, while the patient can take an active part in training by controlling the robot using a Brain Computer Interface (BCI) [10, 50].
BCIs are a specialization of human computer interfaces that enable direct communication between the user's brain and a computer without employing the peripheral nervous system [87]. BCIs use signals recorded from the brain via functional Magnetic Resonance Imaging (fMRI), Electroencephalography (EEG), or Electrocorticography (ECoG). These signals contain information about neuronal activity in the brain. They can be analyzed employing signal processing tools as well as machine learning techniques to learn about, interpret and classify brain data. MI-based BCIs focus on the decoding of imaginative movements and can not only be applied in motor rehabilitation for paralyzed people [5] but might also serve as a direct communication channel for the control of active prostheses, orthoses, or robots. Finally, BCIs can not only be used to restore lost abilities but augment them as well. Avoiding the relatively slow signal propagation in the peripheral nervous system, a BCI could potentially provide a fast and invisible information transfer path from the human brain and serve as an additional communication channel for example for gaming applications.
This work focuses on MI-based BCIs for functional recovery, motor-rehabilitation, and prosthetic control.

1.2 Signal Source

A variety of sensors for monitoring brain activity exist and could in principle provide the basis for a BCI. These include, besides non-invasive and invasive electrophysiological methods such as EEG and ECoG, more technologically advanced methods such as Magnetoencephalography (MEG) or fMRI. The properties of a BCI depend on the characteristics of the signal source. For example, information transfer rate and reaction time in a BCI depend on the temporal resolution of the input signal. Other important factors are price, health risk for the patient, spatial resolution, and manageability/size of the measurement device. Table 1.1 compares EEG, ECoG and fMRI with respect to these criteria. Although studies have shown the value of technologically advanced methods such as fMRI or MEG as they possess a high spatial resolution and are non-invasive, they are still too expensive and slow and the measurement devices too bulky to be used in mobile BCI applications in the context of rehabilitation [72]. ECoG is comparably inexpensive, compact, and shows a high spatial and temporal resolution [47]. Notwithstanding, the surgical implantation of electrodes on the cortex poses a significant health risk for the subject. Finally, EEG is a non-invasive electrophysiological method that has a high temporal resolution, is

	fMRI	**EEG**	**ECoG**
Price	high	low	low
Manageability	bulky	compact	compact
Health risk	low	low	high
Spatial resolution	high	low	high
Temporal resolution	low	high	high
Signal to noise ratio	high	low	high

Table 1.1: Qualitative comparison of brain activity measurement techniques functional Magnetic Resonance Imaging (fMRI), Electroencephalography (EEG) and Electrocorticography (ECoG) [47, 72].

inexpensive, compact and portable. It measures the Local Field Potential (LFP) on the surface of the head. Due to synaptic activity within the brain, primary transmembranous currents generate LFPs that, in turn, induce secondary ionic currents along the cell membranes in the intra- and extra-cellular space within a small volume of nervous tissue. The summed extra-cellular electrical potential (voltage) is transmitted through the skull and measured via electrodes on the scalp [57]. As EEG only measures the effect of superposed neuronal activity outside the head, the spatial resolution is limited. In addition to LFPs generated by neurons, the electrode measurements include any other electrical potential within reach. Therefore, EEG signals contain noise and artifacts caused by muscular activity, blood flow and other sources, resulting in a signal-to-noise ratio that is lower than for ECoG and fMRI.
This work focuses on EEG-based interfaces as it aims at portable, scalable BCI applications for wide-spread use.

1.3 Information Coding and Classification

The classification of brain data using standard classification algorithms such as Support Vector Machines (SVM) or k-Nearest Neighbor (KNN) usually requires an upstream feature extraction stage to reduce the dimensionality of the data and isolate important components from the signal. In order to select and extract the most discriminative features, a-priori knowledge about information representation within the signal and the signal source is beneficial.

Information Coding

The human brain consists of more than 10 billion densely packed neurons that are interconnected in an intricate network [3]. The synapse is the connection for the one-directional communication between pre- and post-synaptic neurons. If the membrane potential of a neuron depolarizes above a certain threshold, for example due to

an external sensory stimulus or pre-synaptic spikes, an action potential is triggered. An action potential is defined as the very fast depolarization and re-polarization of the membrane potential of a neuron. Figure 1.1 shows the schematic course of the membrane potential during the forwarding of an action potential.
As the depolarization phase is very short and the membrane potential always rises to a similar value, action potentials can be abstracted as binary temporal events, also known as spikes. Action potentials are triggered at the cell body due to incoming spikes at thousands of dendritic synapses, and forwarded along the axon to stimulate other neurons, as shown in Figure 1.1. Synapses can be excitatory or inhibitory. Ex-

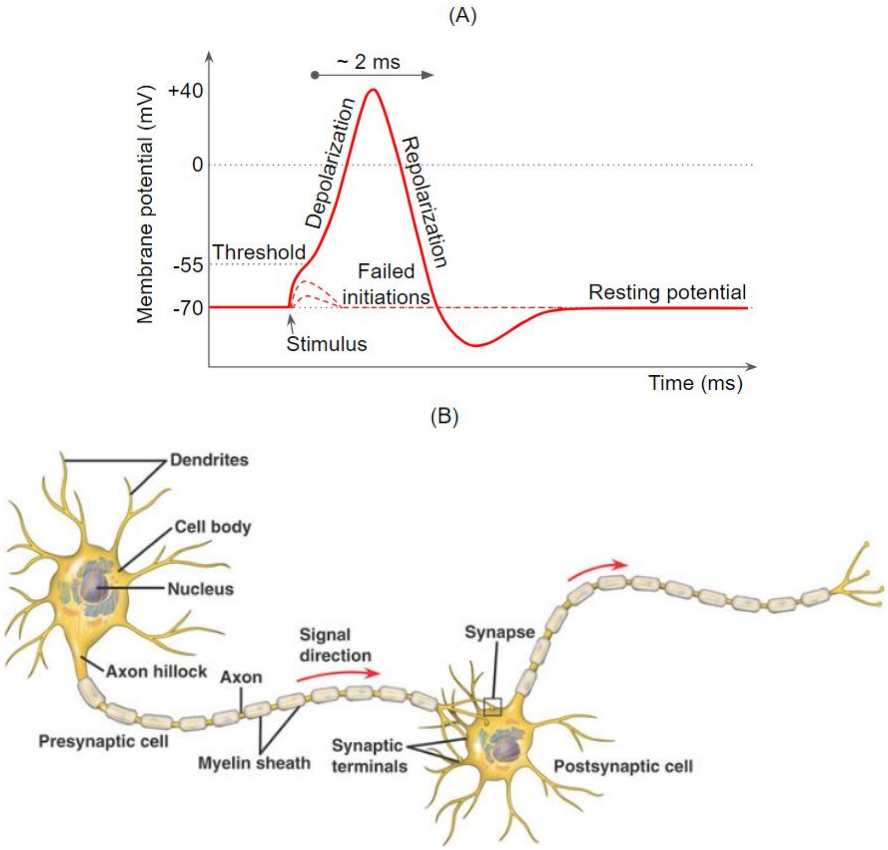

Figure 1.1: A): Schematic course of the neuron membrane voltage during the forwarding of an action potential. The membrane potential starts out at its resting potential $v_{rest} = -70\,mV$. A stimulus is applied, which raises the membrane potential. As soon as the membrane potential exceeds the threshold potential $v_{thresh} = -55\,mV$, it rapidly rises to a peak potential of $v_{peak} \approx +40\,mV$. Just as quickly, the potential drops and overshoots until it finally reaches the resting potential again. The time frame for an action potential spans around 2 to 4 ms. B): Schematic drawing of connected neurons. An action potential is generated at the cell body and forwarded along the axon to the synaptic terminals. A detailed explanation is provided in the text. From Grau [26].

citatory spikes depolarize the post-synaptic membrane and thereby contribute to the generation of a new spike, while inhibitory synapses hyper-polarize the post-synaptic membrane and thereby hinder post-synaptic action potentials. The strength of a synapse defines the contribution of pre-synaptic spikes to evoke or inhibit action potentials in the post-synaptic neuron. The time between spike generation at the cell body and spike arrival at the synaptic terminals correlates with the axon length and thickness and its isolation with a myelin sheath and will be referred to as synapse delay in this work. The exact number of pre- and post-synaptic connections as well as the length of a neuron depend on the specific neuron type and its location in the nervous system. Only the right combination of pre-synaptic spikes, synapse delays, and synapse strengths can cause a depolarization above the threshold potential and thereby generate a post-synaptic spike. Usually it needs multiple spikes on multiple synapses withing a small time window to generate a post-synaptic spike [77, 79]. In this way, information is processed and transmitted in the brain. Nonetheless, synapse strengths are not constant. Synaptic plasticity mechanisms such as Long-Term Potentiation (LTP) and Long-Term Depression (LTD) adjust the synapses to information that is processed in the brain. The modification of synapse strengths is what we call learning.
Although there are hundreds of thousands of spikes emitted in the brain each millisecond, it is possible to record the spiking activity of both single neurons and neural populations. However, it is not yet fully understood how information is encoded in the recordings. There are two main theories about information coding in neural spike trains:

1. **Rate Coding:** The traditional concept of rate coding assumes, that information in the brain is defined by the mean firing rate of one or multiple neurons. The average can be calculated over time, over several repetitions of an experiment, or over a population of neurons.

2. **Pulse or Temporal Coding**: The concept of temporal coding focuses on the exact timing of spikes in a train. There are different theories that emphasize the importance of the first spike time after an applied stimulus, the phase of first spikes in a periodic stimulation, or the relative correlation and synchronization of spike times between neural populations.

While it has been shown empirically, that there are neurons applying both concepts, any one on its own cannot sufficiently explain how the brain processes information in spike trains [23, 71].
In addition to the temporal information within spike trains, recorded brain activity data contains spatial information as well. Based on its anatomical and physiological structure the brain can be divided into functional areas, such as the primary visual cortex, the Broca-area, the somatosensory cortex, and the primary motor cortex. Within the primary motor cortex there is a topographic mapping between body parts and cortical areas [67, 55], as shown in Figure 1.2, that can help to identify and

assign motor actions to different body parts. Although studies have shown that MI mainly activates the primary sensorimotor area in humans [66], recent publications indicate that other areas take part in action related information encoding as well, which complicates the mapping [22]. Consequently, the information in a spike train does not only depend on the internal temporal composition but also on its origin or measurement location.
The topic of spike information coding is still under discussion. If it was at all possible to define a universal neural coding scheme, it would probably combine temporal and spatial coding, as all concepts show empirical justification, as well as other concepts, which have not been identified yet.

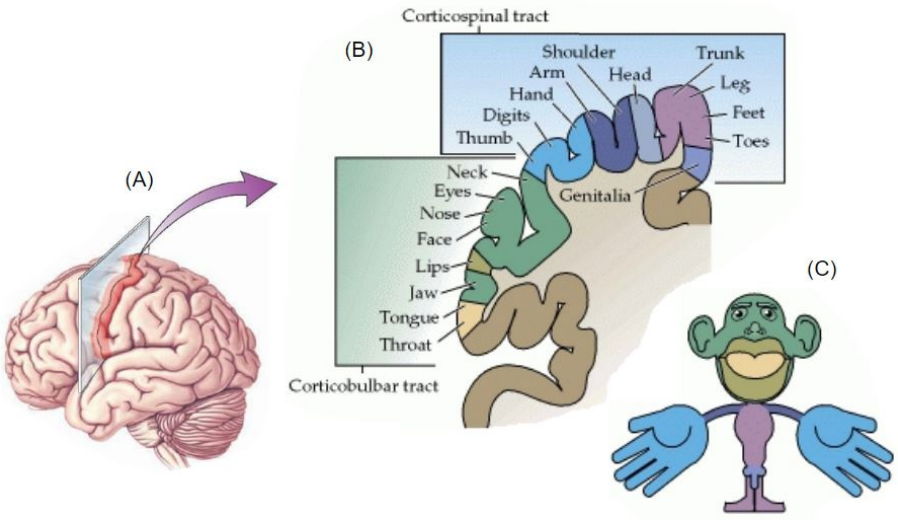

Figure 1.2: Spatial Coding in the human brain with a topographic map in the primary motor cortex of neural regions responsible for the control of respective body musculature. A): Location of primary motor cortex in the precentral gyrus. B): Section along the precentral gyrus, illustrating the somatotopic organization of the motor cortex. The most medial parts of the motor cortex are responsible for controlling muscles in the legs; the most lateral portions are responsible for controlling muscles in the face. C): Disproportional representation of various portions of the body musculature in the motor cortex. Representations of parts of the body that exhibit fine motor control capabilities (such as the hands and face) occupy a greater amount of space than those that exhibit less precise motor control (such as the trunk). From Purves [67].

Classification

The limited knowledge about information coding in spatio-temporal brain data (STBD) makes selection and extraction of meaningful features extremely difficult. So far, the following signal characteristics have been identified and prominently used

for feature extraction: Event-Related Potentials (ERP) such as P300 describe a positive or negative peak potential that is time locked to a specific stimulus event. For large delays and enduring potential changes of more than 300 ms they are called slow cortical potentials (SCP) [6, 65]. Event-Related Synchronization or Desynchronization (ERS/ERD) describe the synchronization or desynchronization of neuronal activity due to external or internal stimuli, which shows in an increase or decrease of power in given frequency bands [72, 65]. Nonetheless, as both ERP and ERS/ERD amplitudes are comparably small, it needs huge data sets to detect these characteristics in brain recordings and extensive, time-consuming, and demanding training by the user to reliably apply control via a BCI using the respective feature classifiers. Furthermore, most features exclusively consider either spatial or temporal information in the data. Thereby, important information can be lost or ignored.
Another challenge in EEG classification is the variability of the input data. EEG measurements can vary due to neural dynamics in the signal source (plasticity in the human brain due to adaptation of the patient to the task), day-to-day variability, or subsequently changing recording conditions [15, 52]. Most standard classifiers are static during classification and do not adapt to changes in the input data. Notwithstanding, this would be necessary to guarantee a reliable and robust classification, in particular in non-clinical applications.

This work employs a promising Spiking Neural Network (SNN) algorithm called NeuCube to handle or avoid these problems. It analyzes spatio-temporal correlations, does not require a-priori knowledge about the data or training of the user, and is adaptable to changes in the input data during the entire usage period as it can learn on-line. A detailed introduction to this model is given in Chapter 2.

1.4 Hardware Infrastructure

The NeuCube algorithm is based on the concept of SNN, which are inspired by the signal processing in the human brain and focus on the exact timing of spikes [24]. To execute such artificial networks in real-time means simulating the asynchronous spiking behavior and synaptic communication between thousands of neurons at the same time. This either requires hardware that is capable of massive parallel processing or software that can simulate parallel processes. In addition, to embed the hardware in mobile applications and enable an independent, off-grid power supply requires low power consumption. Standard hardware, such as Central Processing Units (CPUs), Graphical Processing Units (GPUs), or Field Programmable Gate Arrays (FPGAs), do not satisfy these conditions, because they either consume too much energy or are not able to run complex simulations of SNN in real-time.
Neuromorphic hardware has been developed specifically to simulate large networks of spiking neurons in real-time with low energy consumption. With its asynchronous, decentralized architecture, the brain consumes only 20 W as only those units con-

sume energy that are actually used. As in the brain ($10\,\frac{\text{fJ}}{\text{conn.}}$) the energy for a single connection of artificial, spiking neurons within neuromorphic systems is minuscule (TrueNorth: $25\,\frac{\text{pJ}}{\text{conn.}}$, SpiNNaker: $10\,\frac{\text{nJ}}{\text{conn.}}$), which results in a low over-all energy consumption [19]. Functional complexity is achieved by connecting single units in a complex way (this concept is elaborated in Section 2.3).

TrueNorth, developed by IBM, is optimized for low power consumption using only 70 mW for the simulation of 100 million artificial neurons [1]. Nonetheless, it is not programmable and fixes the synaptic connections after initialization, which prevents run-time plasticity within the network [19].

SpiNNaker (Spiking Neural Network Architecture) has been developed by Steve Furber's Advanced Processing Technologies Research Group (APT) at the University of Manchester [20]. While SpiNNaker maintains a comparably low power consumption of 1 W per chip (up to 10 k neurons), neuron and synaptic plasticity models are still programmable and neural connections can change during run-time enabling learning in the network [19, 61]. Each SpiNNaker chip consists of 18 ARM 968 cores running at 200 MHz clock speed, a router, and 128 MByte shared SDRAM. One core is always blocked for monitoring, e.g. controlling communications across the machine. Therefore, users can use 17 cores to simulate their neuron models with each core simulating up to 1.000 neurons. However, as increasing the number of synaptic connections reduces the number of neurons that can be simulated on each core, the simulated neurons should be distributed on the available cores. There are different SpiNNaker boards available with 4 (SpiNN-3) or 48 (SpiNN-5) chips respectively. To simulate large networks multiple boards can be easily connected to increase the computational power. Although the processor architecture is crucial for parallel processing, it is the specialized, asynchronous communication system using address-event representation that makes SpiNNaker neuromorphic. The lightweight multicast packet-routing mechanism is the key innovation of SpiNNaker as it supports the very high connectivity found in biological brains and is the basis for the remarkable scalability of the system. [20, 19].

The use of neuromorphic hardware enables adaptable, portable, low-power, real-time applications for BCI that are based on spiking networks. In this work we use SpiNNaker as the NeuCube algorithm requires synaptic plasticity and the programmability of the board allows for quick changes to the simulations.

1.5 Related work

This section presents an overview of related work in the field of MI BCI research. First, the state of the art of invasive ECoG and non-invasive EEG BCIs is shortly described. Then, applications of the novel EEG decoding algorithm NeuCube and the neuromorphic hardware SpiNNaker are reviewed. Lebedev and Nicolelis have reviewed state-of-the-art BCIs in more detail [47].

Invasive ECoG MI BCIs

The past two decades brought significant performance improvements in MI-based invasive BCIs. Based on the use of ECoG in apes for bi-manual hand movement and whole body navigation [31, 69], invasive BCI have become relevant for human applications as well. Schwartz et al. and Hochberg et al. achieved the first major breakthroughs in human ECoG MI BCI research in 2008 and 2012 respectively [28, 85]. They implanted micro electrode arrays into the skull of paralyzed patients and used the recorded signals to decode multidimensional motor intentions from a small, local population of motor cortex. The subjects were able to perform reach and grasp tasks using a neurally controlled robotic arm with seven degrees of freedom. Although robotic reach and grasp actions were not as fast or accurate as those of able-bodied people, these results demonstrate the feasibility for patients with tetraplegia or paraplegia, years after injury to the central nervous system, to recreate useful multidimensional control of complex devices directly from a small set of neural signals.
In 2016, for the first time Flesher et al. recreated natural sensation in paralyzed patients by using intra-cortical micro-stimulation of human somatosensory cortex. A classification accuracy of up to 100% shows the advances in bidirectional control [17].
Invasive BCI help to understand the human brain, prove the feasibility and performance limits of BCI applications in a research environment, and make the application potential of BCI tangible. Nonetheless, the usability and scalability of ECoG MI BCI is highly limited due to the health risks that come with the necessary surgical intervention and the associated costs, which raises the need for inexpensive, non-invasive solutions.

Non-invasive EEG MI BCIs

Due to its usability and comparably low price, EEG has become a popular brain signal source for MI BCI applications. Various approaches for decoding EEG signals have been investigated. Most of the decoding methods are based on traditional machine learning algorithms, such as SVM and linear discriminant analysis (LDA) [91]. Schlögl et al. applied and compared five different classification approaches: single-channel minimum distance analysis (MDA); MDA based on the three best channels; LDA using 60 channels; SVM using 60 channels; and KNN using 60 channels. Accuracy ranged between 52 and 70% with SVM performing best [73].
Other authors applied different versions of neural networks. Yang at al. used convolutional neural networks on a 4-class MI data set resulting in a cross-validated accuracy of 67% with a standard deviation of 16%, which is the state of the art for 4-class MI classification using statistical machine learning methods [90].
Applications for EEG BCI include multi-dimensional movement control of robot arms or prostheses [88, 40]. Notwithstanding, due to the low spatial resolution and noise in EEG signals the applications are limited to four classes or less. Standard

classification methods are executed on a general purpose computer, which results in a low processing speed and high energy consumption. Furthermore they require large data sets to achieve good results. This reduces the suitability of such techniques for mobile, real-time BCI applications.
Lotte et al. provide a detailed review of classification algorithms for EEG-based BCI and a discussion of their limitations [52].

NeuCube applications

The NeuCube model has been successfully applied and evaluated in different studies. In an EEG context the system was tested to recognize cognitive processes and compared with other traditional classifiers such as SVM and multi-layer perceptron (MLP) for EEG classification [39, 36]. Doborjeh et al. tested the model in a similar study on fMRI data [16]. Chen et al. and Taylor et al. performed two feasibility studies for the use of NeuCube for an MI BCI resulting in accuracy values of up to 81% for recognition of wrist flexion, extension, and rest [82, 10]. The performance measures of NeuCube are generally higher than for traditional classifiers reaching up to 100% for individual subjects, which shows the potential of the model. However, the publications do not include information about the chosen parameters or the code itself which limits the usability of the model. Scott et al. present an implementation architecture but do not provide access to the code [76, 75]. Furthermore, the network parameters were optimized manually for individual data sets which limits the applicability of the model.
Due to good results in EEG BCI studies, the NeuCube algorithm has been adapted and successfully applied to different types of source data as well, such as traffic and sleep data [84], earth quake data [76], EMG measures [63], to perform ecological forecasts [83], and to analyze and understand the mechanisms in the human brain [38].
These positive results encouraged our choice of the NeuCube algorithm for this study.

SpiNNaker applications

The SpiNNaker board has already been applied in robotic contexts and connected to other neuromorphic hardware such as the eDVS silicon retina sensor to control an autonomous mobile robot, which can identify and approach a signal in real time [14]. In another project SpiNNaker was used to control a musculoskeletal robot in real time [70]. Recently it was used to decode two-class MI movements from EEG signals using a SNN with standard input features (ERS,ERD,ERP) [81].

1.6 Ethics

With respect to the recent development and the current efforts, which are made to improve BCI performance in terms of reliability, communication bandwidth, and speed, as well as the increased publicity due to initiatives such as Elon Musk's Neuralink, it seems reasonable to imagine a not too distant future in which the brain is completely understood and accessible. We may think of scenarios in which humans are able to use BCI to reliably and safely control complex technical systems with their thoughts. They may merge with intelligent prostheses to regain or augment their abilities intuitively, communicate with each other from brain to brain, or even learn new things by simply updating their brain with external programs. The aim of this work is to help reach this goal by providing a small step in progress and insight into the big picture of BCI and neuro-scientific research. However, just as every technological advancement so far has opened up new opportunities to improve our lives and the world around us, progress always carries new risks and dangers as findings may be abused for criminal, harmful or antisocial purposes. Applications of BCI research may be categorized in three classes: thought reading; controlling with thoughts; and controlling of thoughts.

Thought reading: Assuming a future, in which the function of the brain is mostly understood, reading and interpreting people's thoughts will most likely be technically possible. This will open up a new category in personal data protection and privacy, which should be anticipated and prepared by legislators. Concepts such as ethical approvals should be discussed for neuro-scientific research projects and potential users should be educated on the access they provide and the information they share via BCIs.

Controlling *with* thoughts: As soon as humans are able to control complex robots with their thoughts, BCIs could also be applied in military contexts or terrorist endeavors to harm other human beings or to break into people's privacy. This new control path might reduce the cognitive threshold for committing a crime or harming humans as a person does not have to physically act to cause damage. Mechanisms for the prevention of BCI abuse should therefore be part of the general research strategy.

Controlling *of* thoughts: Advances in brain measurements and BCI will most likely deepen the knowledge about information coding in the human brain. At the same time, neural stimulation techniques will improve becoming more precise and subtle. Combining knowledge about the brain and its stimulation may increase the danger of mind or body control. People could be influenced in their decisions and actions without their knowledge, which may have major consequences on society, politics, law and all social interactions.

In order to shape and ensure a safe and peaceful future it is of utmost importance that researchers are aware of the consequences of their work and users are educated about the consequences of their actions when opening their brain to technology.

Chapter 2

NeuCube SNN Algorithm

The NeuCube architecture has been developed by Nikolai Kasabov at the Knowledge Engineering and Discovery Research Institute (KEDRI) at Auckland University of Technology, New Zealand. It is biologically inspired by the human brain, although it does not lay claim to biological plausibility, and can be used to perform both classification and regression tasks on complex signals [35, 39]. The current work solely considers classification as it is focused on the interpretation and recognition of MI tasks in EEG BCI applications.
The basic NeuCube model comprises three modular stages, preceded by a signal acquisition and pre-processing step, as shown in Figure 2.1:

1. Signal to Spike Encoder
2. SNN Reservoir Filter
3. Output Classifier

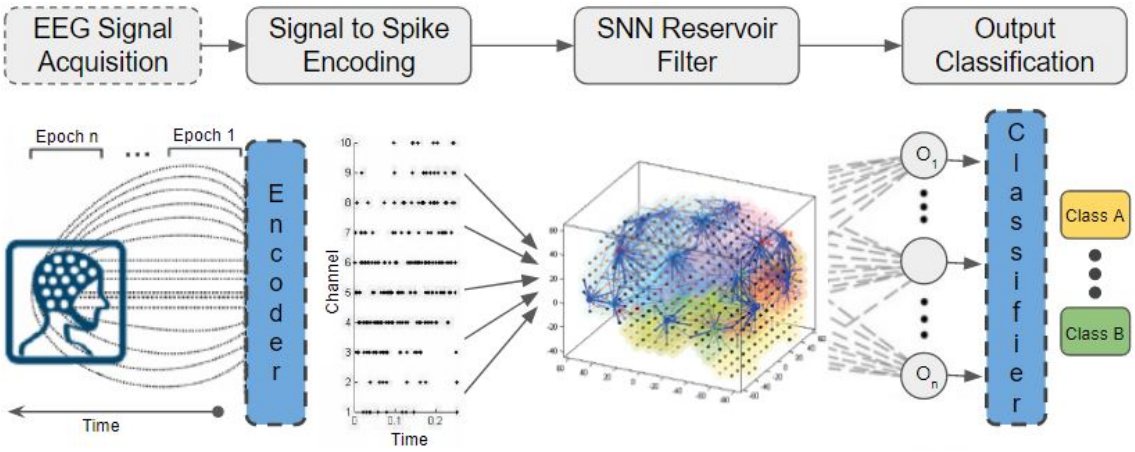

Figure 2.1: Modular structure of NeuCube [2, 10]

At its core, NeuCube is a liquid state machine (LSM) [25]. In general, LSMs are composed of two basic parts, a liquid - usually a recurrent SNN, and a readout function. The LSM performs pattern recognition as follows: First, an input signal $x(t)$ is acquired and encoded with some encoding function $\mathrm{e} : \mathbb{N} \times \mathbb{R}^{\mathrm{n}} \rightarrow \mathbb{N} \times \mathbb{B}^{\mathrm{n}}$ into a discrete time series of binary spikes, a spike train. The liquid is then stimulated with the input spike train and transforms the signal into another signal using a function $\mathrm{l} : \mathbb{N} \times \mathbb{B}^{\mathrm{n}} \rightarrow \mathbb{N} \times \mathbb{B}^{\mathrm{p}}$, which encapsulates the spiking dynamics of the liquid. Due to the internal structure and properties of the liquid, different input signals induce different dynamic reactions of the liquid that are captured by a sampling function $\mathrm{s} : \mathbb{N} \times \mathbb{B}^{\mathrm{p}} \rightarrow \mathbb{R}^{\mathrm{p}}$ creating a state vector for each input signal. Finally, a readout function $\mathrm{r} : \mathbb{R}^{\mathrm{p}} \rightarrow \{0, 1, ...\mathrm{N}\}$ is trained on the state vectors to represent N input classes. In the end, the LSM can predict the class c of a new input signal x as $c(x) = \mathrm{r}(\mathrm{s}(\mathrm{l}(\mathrm{e}(x))))$.

This chapter describes how different functions of the LSM are conceptually realized in the NeuCube model and explains how this enables the recognition of EEG related signal features.

2.1 EEG Signal Acquisition and Pre-Processing

In order to systematically accumulate spatio-temporal brain data (STBD) for training MI-based decoding systems, de-facto standardized recording protocols have been designed. A protocol defines and schedules tasks that have to be performed by the subject. Experimental paradigms try to objectify and quantify human behavior. The subject has to perform the predefined tasks while the EEG signals are recorded on multiple electrodes at a device-dependent sampling frequency. The recorded samples are labeled with their respective class label and stored in the general data format for bio-medical signals (GDF) incorporating meta data such as age and gender of the subject [49]. The electrode positions on the scalp of the subject are defined by the international 10/20 system [29]. It was developed to foster reproducibility and comparability of results by standardizing the electrode positions. The electrodes are located on a regular grid as shown in Figure 2.2, in which the electrode distances are relative to the maximum lateral/sagittal (front-back) or coronal/frontal (left-right) dimension [42].

After recording, the raw EEG signals can be filtered to remove noise and independent component analysis can be used to remove artifacts such as Electrooculography (EOG). A band-pass filter can be applied to filter frequencies below and over the biologically reasonable limits and a notch filter can be used to remove power grid artifacts. NeuCube does not require any further signal processing or feature extraction, which is one noticeable advantage. The filtered signal $x(t)$ serves as its input. The pre-processing steps that are used depend on the specific implementation of the model.

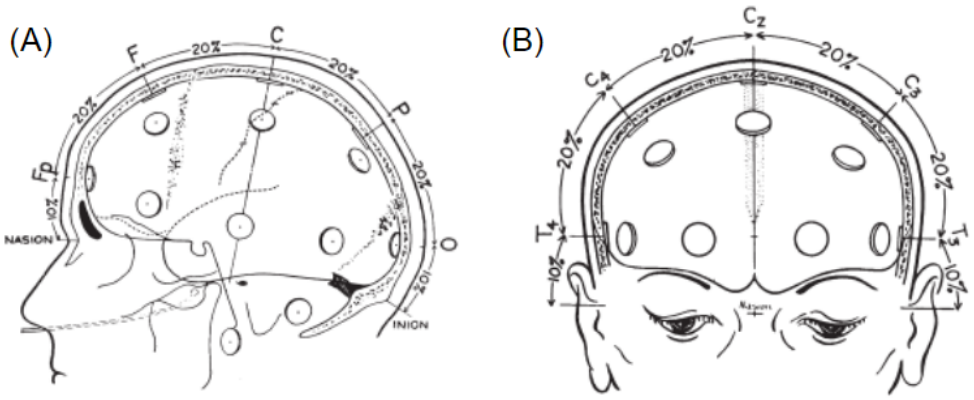

Figure 2.2: International 10/20 system for standardized EEG electrode placement. (A) Lateral view of the human skull showing the standardized electrode locations relative to the distance between nasion and inion. (B) Transverse view of the human skull showing the standardized electrode locations relative to the distance between left and right ear. From Jasper et al. [42]. Please note that the electrodes are erroneously located inside the skull on the surface of the cortex.

2.2 Stage 1: Signal to Spike Encoding

The first stage realizes the encoding function $e : \mathbb{N} \times \mathbb{R}^n \rightarrow \mathbb{N} \times \mathbb{B}^n$ and converts the input STBD signals that have been recorded from the brain using EEG into spike trains. A spike train is a time discrete, binary signal which can only take the values 0 (no spike) and 1 (spike). Depending on the context and encoding scheme, negative spikes with a value of -1 are also permissible. Nonetheless, positive and negative spikes can never occur at the same time. Mathematically a spike train can be described as:

$$spike\_train(t) = \begin{cases} +1 & \text{if there is an excitatory spike at time } t, \\ -1 & \text{if there is an inhibitory spike at time } t, \\ 0 & \text{otherwise} \end{cases} \tag{2.1}$$

There are different methods for signal encoding that focus on different aspects of neural coding. In general terms, each method defines rules and equations that define for each time step, if there shall be a spike or not. Within NeuCube only those encoding methods are admissible that maintain the dimensionality of the data, meaning that the number of input channels for each sample has to be equal to the number of spike trains generated by the encoder. Therefore, the output of the encoder consists of one spike train per input EEG channel.

2.3 Stage 2: SNN Reservoir

The second stage realizes the liquid function $\mathrm{l} : \mathbb{N} \times \mathbb{B}^{\mathrm{n}} \rightarrow \mathbb{N} \times \mathbb{B}^{\mathrm{p}}$. The liquid consists of a three dimensional recurrent SNN reservoir which extracts spatio-temporal information from the data [56, 25] and employs an unsupervised learning rule to increase the effectivity of this process [58].

Spiking Neural Networks and Learning

SNNs are the third generation of artificial neural networks and comprise of spiking neurons, which are connected via artificial synapses. Each synapse is defined by a pre- and post-synaptic neuron, a synapse delay, and a weight value representing the synapse strength, which is modified during learning processes. SNNs are, with regard to the number of required neurons, computationally more powerful than classical neural network models [53]. Information transfer in these neurons mimics the information transfer in biological neurons, e.g. via the precise timing of spikes or a sequence of spikes [24]. There are different models for artificial spiking neurons that describe how action potentials in neurons are initiated and propagated. Neuron models differ in their degree of biological realism and computational efficiency with the Hodgkin-Huxley neuron model describing neuronal behavior on a particle level [48], the Leaky-Integrate-and-Fire (LIF) neuron model sacrificing realism for computational efficiency [51], and the Izhikevich neuron model combining both aspects [32, 33]. Based on findings about learning in the human brain, different rules have been designed to facilitate synaptic plasticity in SNNs. Spike-Time Dependent Plasticity (STDP) is one biological realization of Hebbian learning that can be implemented in spiking networks [9, 34]. It describes the phenomenon that 'cells that fire together wire together' [27]. In detail this means that a synapse increases its efficacy if it persistently takes part in firing the post-synaptic target neuron.
In spiking networks the STDP weight change mechanism works as follows. A synapse connects a pre-synaptic neuron $\mathrm{N_{pre}}$ to a post-synaptic neuron $\mathrm{N_{post}}$ with a weight of $w_{\mathrm{pre}}^{\mathrm{post}}$. If $\mathrm{N_{pre}}$ fires a spike at $\mathrm{t_{pre}}$ and $\mathrm{N_{post}}$ fires a spike at $\mathrm{t_{post}}$, the weight change depends on the spike time difference $\Delta t = \mathrm{t_{pre}} - \mathrm{t_{post}}$. If $\mathrm{N_{pre}}$ fires before $\mathrm{N_{post}}$ ($\Delta t < 0$), the weight is increased as it is assumed that $\mathrm{N_{pre}}$ contributed to the spiking of $\mathrm{N_{post}}$. This is called potentiation. If $\mathrm{N_{pre}}$ fires after $\mathrm{N_{post}}$ ($\Delta t \geq 0$), the weight is decreased as $\mathrm{N_{pre}}$ can not have caused the spiking of N_{post}. This is called depression. Equation 2.2 specifies one function for the amount of synaptic modification, which is depicted in Figure 2.3 [78].

$$\frac{\Delta w_{\mathrm{pre}}^{\mathrm{post}}(\Delta t)}{w_{\mathrm{pre}}^{\mathrm{post}}} = \begin{cases} \mathrm{A_+} \times \mathrm{e}^{\Delta t/\tau_+} & \text{if } \Delta t < 0 \\ -\mathrm{A_-} \times \mathrm{e}^{-\Delta t/\tau_-} & \text{if } \Delta t \geq 0 \end{cases} \tag{2.2}$$

with $\Delta w_{\mathrm{pre}}^{\mathrm{post}}(\Delta t)$ being the synaptic modification arising from a single pair of pre- and post-synaptic spikes separated by a time Δt. Time constants τ_+ and τ_- describe the exponential decay of modification with Δt. The factors $\mathrm{A_+}$ and $\mathrm{A_-}$ determine

the respective maximum amount of synaptic modification during potentiation and depression.

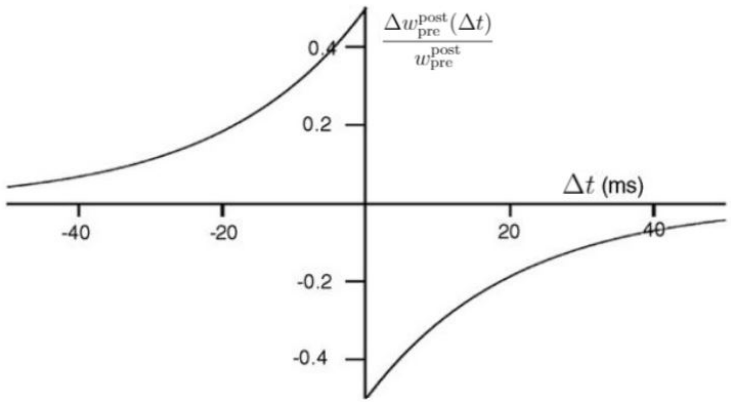

Figure 2.3: Weight change in STDP. The relative weight change $\frac{\Delta w_{\text{pre}}^{\text{post}}(\Delta t)}{w_{\text{pre}}^{\text{post}}}$ at a synapse due to a single pre- and post-synaptic spike with spike time difference Δt. For a detailed explanation see main text. From Song [78].

Structure of the SNN Reservoir

Within the NeuCube reservoir, the locations and connections between spiking neurons are modeled after the 3D-geometry of the human brain. The neuron density and the employed neuron model are chosen by the user. Neurons are connected following a small world connectivity approach, which means that close neurons are connected with a higher probability than neurons that are further apart. The proportion of excitatory and inhibitory synapses is defined by the inhibitory split. Furthermore, the synapse delays are proportional to the Euclidean distance between the connected neurons. This design ensures that the spatial information and relative timing of the spikes in the input data is maintained within the model.

SNN Spatio-Temporal Filter for EEG Features

After the neurons and their connections are initialized, the network has to find spatio-temporal correlations between different samples within the input signals. The ultimate goal is to initialize an SNN reservoir ('liquid') that can produce significantly different spiking dynamics for samples from different classes, and similar spiking behavior for samples from the same class. The 'amount of separation between trajectories of internal states of the system that are caused by two different input streams' [54] is an important quality that ensures the efficacy of an LSM. In NeuCube it measures the distance between state vectors belonging to different classes. More details about state vector sampling and separation measures can be found in Section 2.4 and 3.5.

In order to use traditional classification algorithms, features have to be extracted from the input signal. When working with EEG data, the challenge at this point is that the signal features that actually encode the relevant information to distinguish samples from different classes are not fully understood. Based on the limited knowledge about brain processes and EEG features that has been identified so far, it is assumed that potential features are:

- firing or not-firing of a neuron with a specific delay after imagination onset, as observed in event-related potentials (e.g. P300),
- synchronous periodic firing of neurons with a certain frequency and a well-defined delay/phase, as observed similarly in event-related synchronization (ERS),
- asynchronous firing of neurons with a certain delay after imagination onset/offset, as observed similarly in event-related desynchronization (ERD) [64].

Notwithstanding, it is highly probable that there are more complex features that have not been identified so far. Due to its complex and meaningful connectivity, the reservoir is able to identify and highlight such features.
This mechanism shall be demonstrated with a simplified example, which is illustrated in Figure 2.4. If the significant feature for a class A was the synchronization of spike trains from two input neurons N_i and N_j with a characteristic spike time delay of $\Delta t_A = t_i - t_j$, the presentation of a sample from that class A will cause periodic firing in reservoir neurons N_r, if the overall synapse delay difference $\Delta d_r^{i,j} = d_{i,r} - d_{j,r}$ compensates the spike time difference Δt_A.

$$\Delta t_A = t_i - t_j \stackrel{!}{=} d_{j,r} - d_{i,r} = -\Delta d_r^{i,j} \tag{2.3}$$

In other words, the sum of spike time and overall synapse delay has to be the same for the respective pre-synaptic neurons in order for their spikes to reach the post-synaptic neuron at the same time.

$$t_i + d_{i,r} \stackrel{!}{=} t_j + d_{j,r} = t_r \tag{2.4}$$

The time $d_{i,r}$ that a spike needs to travel from neuron N_i to neuron N_r is defined by the summation delays of all synapses that connect N_i to N_r. As the synapse delay is effectively determined by the spatial distance between the neurons, the spatial arrangement is responsible for this mechanism to work.
If another sample from class B with a different characteristic spike time delay $\Delta t_B \neq \Delta t_A$ was presented to the same system, there would be a different set of neurons firing in response, as the spike time difference Δt_B would no longer match the travelling time difference $\Delta d_r^{i,j}$ to neurons N_r. In other words, the spikes arrive at N_r at different times.

$$\Delta t_B \neq d_{j,r} - d_{i,r} = -\Delta d_r^{i,j} \tag{2.5}$$

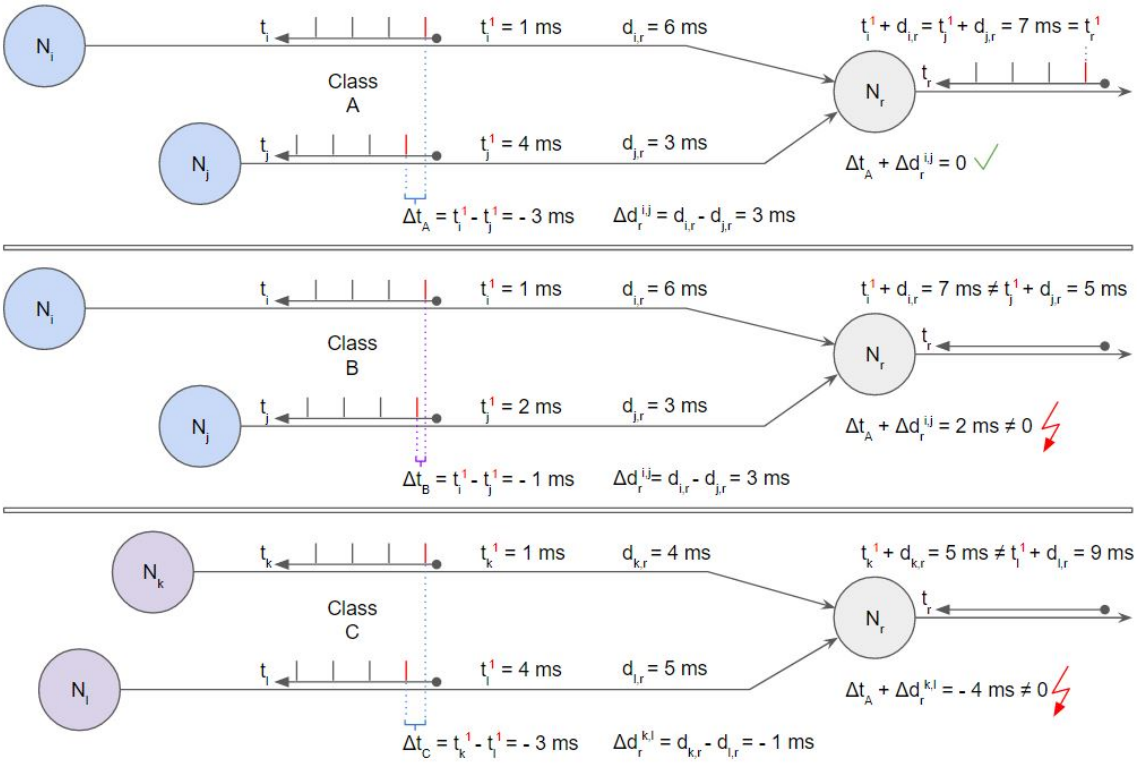

Figure 2.4: Example for the spatio-temporal filter function of SNN. Only if the spike time difference between input neuron spikes is compensated by the synapse delay difference, a spike is fired in the post-synaptic neuron. This is influenced by the exact spike timing and spatial arrangement of the neurons. Every reservoir neuron consequently filters some spatial and temporal information out of the data. The set of firing neurons in the reservoir finally characterizes the input class.

If another sample from class C with the same characteristic delay $\Delta t_\mathrm{C} = \Delta t_\mathrm{A}$ but different input electrodes $\mathrm{N_k}$ and $\mathrm{N_l}$ was presented to the same system, there would be a different set of neurons firing in response as well, as the new synapse delay difference $\Delta d_\mathrm{r}^{\mathrm{k,l}} = d_{\mathrm{k,r}} - d_{\mathrm{l,r}}$ would not compensate the spike time delay Δt_C.

$$\Delta t_\mathrm{C} = t_\mathrm{k} - t_\mathrm{l} \neq d_{\mathrm{l,r}} - d_{\mathrm{k,r}} = \Delta d_\mathrm{r}^{\mathrm{k,l}} \tag{2.6}$$

In general, post-synaptic spikes are only triggered in those neurons, whose spatial arrangement matches the specific timing of pre-synaptic spikes. This example shows how a simple network identifies and extracts spatio-temporal features from input spike trains by creating sets of firing neurons that represent specific classes. In NeuCube the spiking network is more complex. The recurrence within the network, the complex connectivity with excitatory and inhibitory neurons, and the internal dynamics enable the system to recognize more complex features in the input data [8]. This property of unsupervised, spatio-temporal filtering without a-priori knowledge is the crucial benefit of NeuCube. However, the filter function or degree

of separation of an SNN is highly dependent on the parameters for network creation.

In order to improve the separation, learning rules can be applied within the network [58]. NeuCube applies STDP learning to strengthen synapses that are involved in the creation of the class-specific neuron sets and weakens connections that take no role in class-specific neuron pattern creation. Thereby, discriminative spatio-temporal correlations are amplified and isolated. Furthermore, synaptic potentiation increases the impact of pre-synaptic spikes on post-synaptic neurons, which can accelerate the firing of spikes in a neuron after sample presentation onset. This improves the classifier as, within the sampling function, the first spike time has a major influence on the separation of the system. STDP can also make the network robust towards incomplete input samples, in which individual channels are missing, and adaptable to changes in the input data streams by modifying synapses on-line during application. However, learning does not always improve separation. If the parameters are set incorrectly, STDP can overshoot and cause the system to loose sensitivity to different input signals resulting in a lower separation [58]. Therefore, learning parameters have to be tuned very carefully.

Summing up, the spatially meaningful neuron structure and its complex connectivity enable the SNN liquid to filter class-specific spatio-temporal features from EEG data by creating class-specific sets of firing neurons and isolating these sets with STDP-learning, as shown in Figure 2.5.

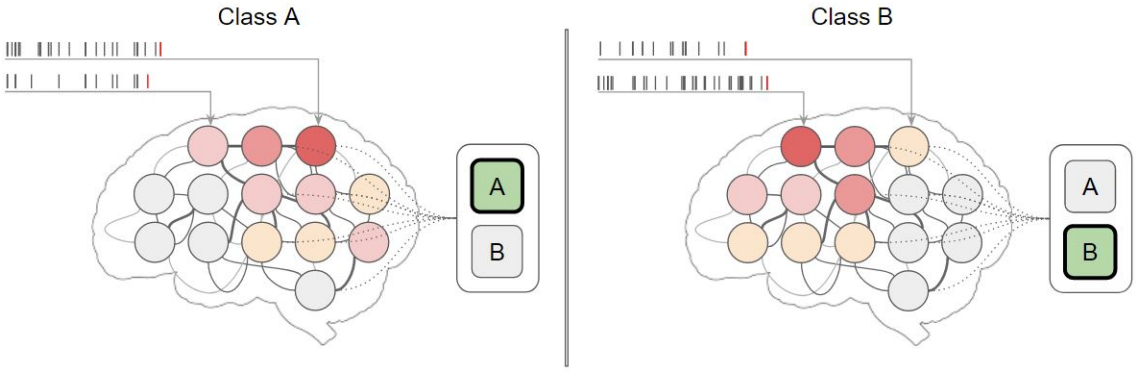

Figure 2.5: Schematic reservoir behavior. Due to the complex reservoir structure different input signals cause activity in different sets of neurons. These sets represent the respective classes and can be used for classification. Figures 2.6 and 2.7 illustrate the classification step in more detail.

2.4 Stage 3: Output classification

The third stage realizes both the sampling function $s : \mathbb{N} \times \mathbb{B}^p \rightarrow \mathbb{R}^p$, which converts the liquid dynamics into state vectors, and the readout function $r : \mathbb{R}^p \rightarrow \{0, 1, ...N\}$, which is trained with and finally classifies the state vectors. While the NeuCube algorithm in general is not restricted to a specific sampling or readout function, this work focuses on the use of a dynamic, evolving SNN (deSNN) with a downstream KNN classifier that classifies samples based on the synapse weights of the network. To avoid ambiguity we refer to neurons within the liquid as reservoir neurons and to neurons in the deSNN as output neurons. Kasabov et al. discuss alternative output classifiers in [37].
In NeuCube the deSNN is created during a training phase that can be interrupted for classification and continued at any time making the system adaptable to new input classes and changes in the input data. For every training sample a new output neuron is added to the classification network, labeled with the respective class label, and connected to all reservoir neurons. While a sample is fed into the reservoir, the resulting spikes within the reservoir are obtained by the output neuron. The initial weights for the synaptic connections between the reservoir neurons and the output neuron are computed using the rank-order (RO) learning rule, which sets the weights according to the timing of the first arriving spike on each synapse:

$$w_{\text{init}}(\mathrm{N}_n, \mathrm{O}_m) = \alpha \times \text{mod}^{order(\mathrm{N}_n, \mathrm{O}_m)} \tag{2.7}$$

with $w_{\text{init}}(\mathrm{N}_n, \mathrm{O}_m)$ the initial weight of the synapse between reservoir neuron N_n and output neuron O_m, $order(\mathrm{N}_n, \mathrm{O}_m)$ the rank of the first incoming spike on this synapse compared to all spikes within the reservoir, α defining the maximum initial weight, and a constant mod, which takes values between 0 and 1, determining the importance of the order of spikes [37].

Once a synaptic weight is initialized based on the first spike at the pre-synaptic reservoir neuron, it gets dynamically adjusted based on the remaining spikes in the reservoir using a semi-supervised learning algorithm called Spike Driven Synaptic Plasticity (SDSP) [21]. SDSP learning is a variant of the STDP rule. Instead of modifying synaptic weights based on both pre- and post-synaptic spikes, it rather depends on pre-synaptic spikes and the membrane potential of the post-synaptic neuron. If a pre-synaptic spike arrives at the synapse and the post-synaptic membrane potential is above a threshold, the synaptic weight is increased by adding a potentiation constant. Otherwise the weight is decreased by subtracting a depression constant [7]. The NeuCube model simplifies this rule. Instead of examining the post-synaptic potential of the output neuron for every incoming reservoir spike, it assumes that, once the first spike arrives at a synapse, the respective synaptic threshold is reached and synaptic plasticity is enabled. It then adds a small value drift_{up} at any time a reservoir spike arrives at this synapse and decreases the weight

by a small value $\text{drift}_{\text{down}}$ if there is no spike [37]:

$$w_{\text{final}}(\text{N}_n, \text{O}_m) = w_{\text{init}}(\text{N}_n, \text{O}_m) + \text{drift}_{\text{up}} \times n_{\text{spikes}} - \text{drift}_{\text{down}} \times n_{\text{no\_spikes}} \quad (2.8)$$

with $w_{\text{final}}(\text{N}_n, \text{O}_m)$ the final weight of the synapse between reservoir neuron N_n and output neuron O_m, n_{spikes} the total number of spikes at the synapse after the first spike, and $n_{\text{no\_spikes}}$ the number of time steps without a spike after the first spike.
The similarity of samples can be computed using the Euclidean distance between the weight vectors of the respective output neurons. This metric can be used to merge similar output neurons that belong to the same class. The weight vectors reflect both temporal and rate coding aspects by combination of RO-learning, which focuses on the importance of early spikes and thus exact spike timing in the network, and SDSP-learning, which incorporates the average firing rate per reservoir neuron. The weight vectors of the output neurons serve as state vectors for the readout function and resemble characteristic reservoir dynamics for different samples. The classification network evolves and dynamically adjusts to every training sample.
Finally, a KNN classifier is trained on the state vectors. A simple linear classifier suffices as the recurrence in the reservoir leads to an integration of information over time. The classification of reservoir dynamics gets easier if the reservoir can distinctly separate reservoir dynamics for different input signals. With respect to the applied sampling function desirable spiking behavior in the liquid can be characterized by a class-specific set of neurons with an early first spike and a high average firing rate, and the remaining neurons firing late with a low firing rate or not at all.

2.5 Integration

The algorithm comprises two phases: the training phase and the classification phase. In both stages the input spike trains are fed into those reservoir neurons that are spatially closest to the respective recording electrodes. During the training phase, the input signals are encoded and fed into the reservoir, where spatial and temporal information from the data are extracted and network connections are adjusted accordingly via STDP learning. Afterwards the spike trains are fed into the reservoir again and the reservoir spikes are used to create the deSNN and train the KNN classifier on the provided data. Figure 2.6 illustrates schematically the creation of output neurons in the deSNN.
During the classification phase new input samples are encoded and fed into the trained reservoir. For each sample a new classification neuron is created and trained outside the classifier network using the same learning strategy as for the output neurons. The new validation sample is assigned to the class that is predicted by the KNN classifier, which associates the classification neuron with the closest output neuron based on the minimum distance between the weight vectors. Figure 2.7 visualizes the creation of a classification neuron and its class assignment by the KNN

classifier. The KNN classifier can use different features to fit the data. It can use the initial or final weight vectors, the total number of spikes per reservoir neuron, or a user-specified combination of the aforementioned.
The training and classification phases can overlap as STDP learning of the reservoir may stay active during classification to adapt the reservoir to changes in the input data stream. Furthermore, the classification phase can be paused to train the model on new samples from potentially new classes.

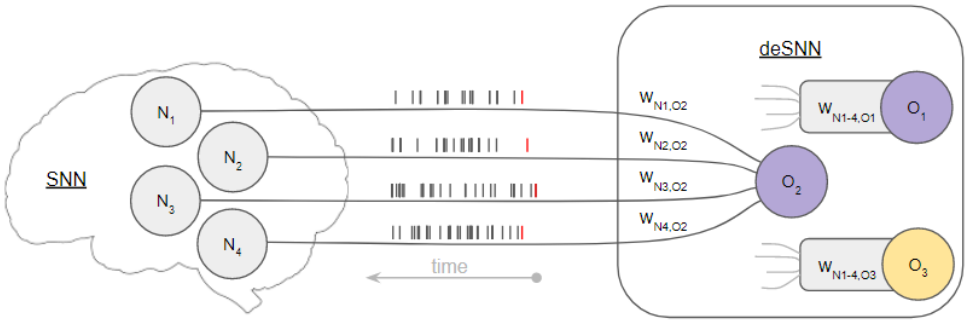

Figure 2.6: Training of a deSNN for sample classification. Each output neuron O_j is connected to all reservoir neurons N_i. The first spikes in each channel (red) define the initial synaptic weights, and the consecutive spikes (black) cause synaptic potentiation and depression during training via SDSP learning. Each output neuron is characterized by its synaptic weight vector. Output neurons with the same color belong to the same class.

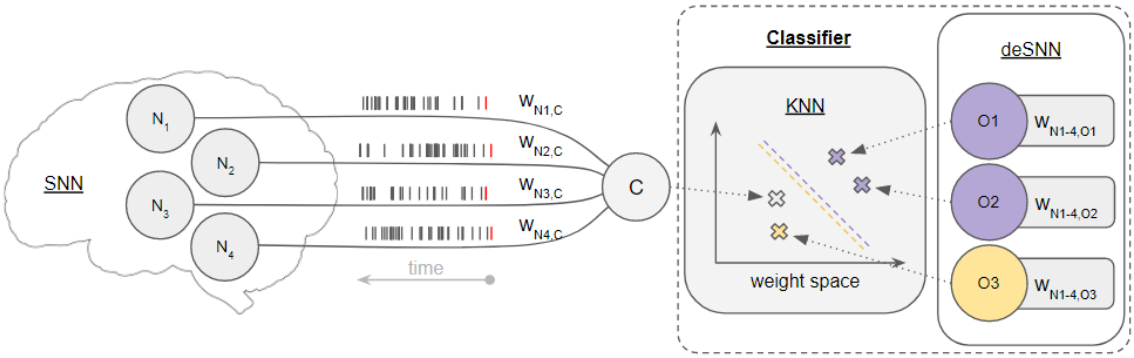

Figure 2.7: Classification of a neuron C. The weights between classification neuron C and reservoir neurons N_i is calculated. The resulting weight vector is compared to all weight vectors in the deSNN employing a KNN classifier. The class of the closest output neuron O_j determines the class prediction of the sample.

Chapter 3

Implementation

This chapter describes the implementation of NeuCube that was developed in this work. It serves as a reference for anyone who wants to understand or use the implementation in the future.

3.1 Tools

The NeuCube model has been implemented on a personal computer running the operating system Windows 10, which was connected via Ethernet to a SpiNN-3 board. Tests were performed on a personal computer running Ubuntu 16.04 LTS (Linux), which was connected via Ethernet to 6 SpiNN-5 boards.
We used the version control and repository hosting service GitHub for maintaining and publishing our code. The repository for this thesis is publicly available at: `https://github.com/behrenbeck/NeuCube_SpiNNaker.git`.

Software	Version	Links
Python	2.7.6	Download
PyNN	0.8.3	Download + Installation Guide
sPyNNaker8	1!4.0.0	Download
		Documentation
		Manual for PyNN on SpiNNaker
SciKit-Learn	0.19.1	Download

Table 3.1: Software versions and links for download and documentation.

This implementation is written in **Python**. The open source Python package for neural networks specification **PyNN** is a Python-based programming interface that enables computational neuroscientists to write a network model and simulation script once and run it on any PyNN-supported simulator as well as on neuromorphic hardware such as SpiNNaker [12]. Running PyNN scripts on SpiNNaker requires

the installation of the front end interface **sPyNNaker8** from the Manchester SpiNNaker group's GitHub repository page. **SciKit-Learn** is a community based Python toolbox for machine learning [62]. In this work we used the KNeighborsClassifier class from the package neighbors as part of the deSNNs implementation. Table 3.1 specifies respective versions that were used and provides links to download and documentation.

3.2 EEG Signal Acquisition and Pre-Processing

For this work, we used the **Graz Data Set B** [49] to train and evaluate the model. This data set consists of EEG data collected from 9 subjects. The subjects were right-handed, had normal or corrected-to-normal vision and were paid for participating in the experiments. All volunteers were sitting in an armchair, watching a flat screen monitor placed approximately 1 m away at eye level. Three bipolar recordings at electrode positions C3, Cz, and C4 were recorded with a sampling frequency of 250 Hz. The recordings had a dynamic range of ±100 µV. They were band-pass filtered between 0.5 Hz and 100 Hz, and a notch filter was applied at 50 Hz in order to remove power line noise. The resulting signals were normalized to a range between 0 and 1 for each channel:

$$v_{\text{norm}} = \frac{v_{\text{raw}} - \min(v_{\text{raw}})}{\max(v_{\text{raw}}) - \min(v_{\text{raw}})} \tag{3.1}$$

with v_{norm} being the normalized potential, v_{filt} being the raw potential after band pass and notch filtering, and min and max being functions to extract the minimum and maximum value of a signal.

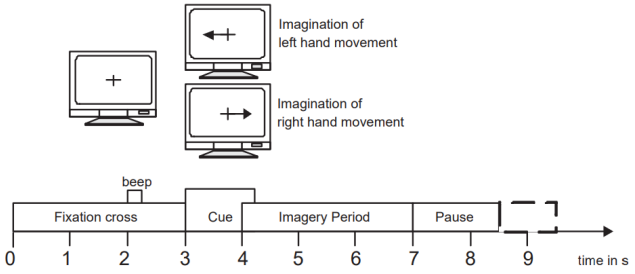

Figure 3.1: Cue-based screening paradigm as used for Graz Data Set B consisting of two classes: MI of left hand and right hand. From Leeb et al. [49].

The cue-based screening paradigm, which is visualized in Figure 3.1, consists of two classes, namely the MI of left hand (class 1) and right hand (class 2) movements. Each subject participated in two screening sessions without feedback recorded on two different days within two weeks. Each session comprised six runs with ten trials each and two classes of imagery. This resulted in 20 trials per run and 120 trials per

session. Data of 120 repetitions of each class were available overall for each person. Prior to the first MI training subjects executed and imagined different movements for each body part and selected that motion for recording that they could imagine best (e. g., squeezing a ball or pulling a brake). Each trial started with a fixation cross and an additional short acoustic alert (1 kHz, 70 ms). Some seconds later a visual cue (arrow pointing either to the left or right according to the requested class) was presented for 1.25 s. Afterwards the subjects had to imagine the corresponding hand movement over a period of 3 s. Each trial was followed by a short break of at least 1.5 s and a randomized time of up to 1 s to avoid adaptation [49].
Although NeuCube in general allows input signals with different duration, the normalized EEG-data was cut into multiple samples of the same length to improve comparability between test results. Three samples with 1 s duration were extracted from each trial containing 250 data points each. One sample was extracted right after imagination onset between ${\tau_1}^{\text{start}} = 4\,\text{s}$ and ${\tau_1}^{\text{stop}} = 5\,\text{s}$, one during imagination between ${\tau_2}^{\text{start}} = 5\,\text{s}$ and ${\tau_2}^{\text{stop}} = 6\,\text{s}$, and one at the end of imagination between ${\tau_3}^{\text{start}} = 6\,\text{s}$ and ${\tau_3}^{\text{stop}} = 7\,\text{s}$, resulting in matrices of size 3×250 (number of electrodes $\times$ number of data points). For each time window and subject, the samples were reordered in an alternating structure with 5 samples of one class being followed by 5 samples of the other class and stored in separate files with the naming convention 'sam_X.csv' where X denotes the number of the sample. Each sample set is contained in a folder specifying subject and time window in 'input_stage_1' (e.g. 'B04T_4-5' for samples from Graz Data Set 'B', subject '04', training trials 'T' within the time window $\tau_1 = [4, 5]$).
For testing purposes the data set was duplicated thrice and filtered with additional band-pass filters (cut-off frequencies of $\text{f}_1 : 0.5-30\,\text{Hz}$, $\text{f}_2 : 7-30\,\text{Hz}$, $\text{f}_3 : 0.1-60\,\text{Hz}$). The resulting sample sets were stored in separate folders in 'input_stage_1' as well (e.g. 'B04T_4-5_f05-30' for samples from subject 4, time window τ_1, and filter f_1).

3.3 Stage 1: Signal to Spike Encoding

The spike encoding stage has been implemented in Python and runs on the host computer. It converts normalized EEG samples into sets of spike trains. For this project, two existing encoding algorithms have been implemented: Temporal Difference (TD) and Bens Spiker Algorithm (BSA). Additionally, a new encoding algorithm has been developed, which is a modification of the TD algorithm inspired by the BSA (modTD).
To describe spike trains we use arrays that are similar to spike source arrays (SSA) as used in PyNN. An SSA is an array containing time points of spikes. Notwithstanding, SSAs can only contain either excitatory or inhibitory spikes, which is not sufficient for TD encoding. In order to distinguish between excitatory and inhibitory spikes within the same array, we define that the sign of the time point specifies the spike mode: if the time point is positive, the spike is excitatory; if the time point is

negative, the spike is inhibitory. Time points are specified in milliseconds.
Example: If there are excitatory spikes at $t = 12\,\text{ms}$ and $t = 36\,\text{ms}$ and an inhibitory spike at $t = 33\,\text{ms}$, a spike train is defined as spike_train $= [12, -33, 36]$.

For visualization the spike times and spikes can be extracted as follows:

1: time_points = [abs(x) for x in spike_train]
2: spikes = [sign(x) for x in spike_train]

To evaluate performance in the encoder and ensure the preservation of information in the signal, the encoded spike trains are decoded again and the reconstructed signals are compared to the original signals. Hu et al. present an error metric, which is used to evaluate the performance and compare different encoders [30]:

$$error = \frac{\sum |s_{\text{ori}} - s_{\text{rec}}|}{\sum s_{\text{ori}}} \tag{3.2}$$

where *error* is the evaluation metric, s_{ori} is the original input EEG signal, and s_{rec} is the reconstructed signal from the encoded spike train.

Temporal Difference (TD) Algorithm

The Temporal Difference encoding algorithm is computationally simple. It allows both excitatory and inhibitory spikes in a spike train. It has been included in this work as it is easy to understand and implement and can build the baseline for any comparison. It works as follows. Every time step τ, the difference $\Delta v = v(\tau) - v(\tau - 1)$ between the current and the previous value is calculated and compared to a threshold. If the absolute difference exceeds the threshold, a spike is generated. The sign of the difference Δv defines the sign of the spike and thereby its mode. If the absolute difference is smaller than the threshold, there is no spike. The threshold can be optimized for different input signals.

Pseudo-code for TD-Encoding:

1: *spike_train* = [ ]
2: *last_value* = 0
3: **for** time step τ in signal duration **do**
4: *diff* = *current_value* - *last_value*
5: **if** abs (*diff*) $\geq$ threshold **then**
6: *spike_train*.add (sign (*diff*) $\times$ time step τ)
7: **end if**
8: *last_value* = *current_value*
9: **end for**

The decoding algorithm is also straight-forward. The recreated signal starts off at 0. For every spike in the spike train, all the values in the signal after the spike time

change by the threshold value depending on the mode of the spike. If the spike is excitatory, the signal rises. If the spike is inhibitory, the signal falls.

Pseudo-code for TD-Decoding:

```
signal = zeros(signal_duration)
for spike in spike_train do
    if spike is excitatory then
        signal [spike time τ : end] += threshold
    else
        signal [spike time τ : end] -= threshold
    end if
end for
```

Bens Spiker Algorithm (BSA)

Schrauwen et al. developed the Bens Spiker Algorithm (BSA) based on the Hough Spiker Algorithm (HSA)[74]. It shows better performance with regard to information loss and is more robust as it shows less susceptibility to changes in the threshold and different input signals. This is why it is included in this work. The BSA uses solely excitatory spikes and makes use of a finite impulse response (FIR) reconstruction filter. It works as follows. Every instant in time τ two error metrics are calculated:

$$error_1 = \sum_{i=0}^{M} \text{abs}(\text{eeg}_{\text{ori}}(i+\tau) - \text{h}(i)) \tag{3.3}$$

$$error_2 = \sum_{i=0}^{M} \text{abs}(\text{eeg}_{\text{ori}}(i+\tau)) \tag{3.4}$$

with eeg_{ori} being the original signal, h the FIR filter, and M the filter size.
If the first error metric is smaller than the second minus a threshold, a spike is generated and the filter is subtracted from the input signal; else there is no spike. The threshold can be optimized based on the input signal and the filter size. For

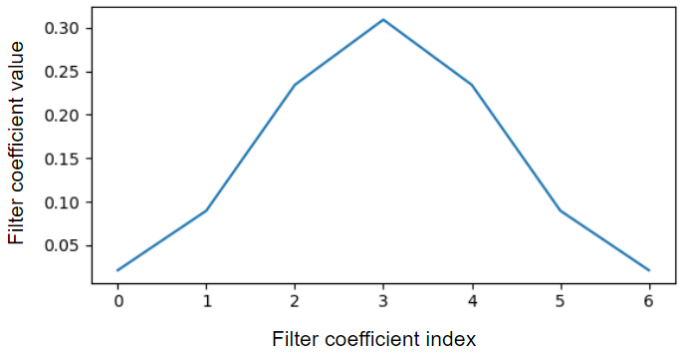

Figure 3.2: 7-tap filter as used for the BSA spike encoder generated by Python function 'scipy.signal.firwin'.

this work, a 7-tap FIR filter was used as suggested by Hu [30] as it produced better results for our data than longer filters suggested in other publications[74, 13, 59]. This is because our signals contained high-frequency elements and fast fluctuations are deleted by large FIR filters. Figure 3.2 plots the filter coefficients of the employed filter.

Pseudo-code for BSA-Encoding:

1: *spike_train* = [ ]
2: **for** time step τ in signal duration **do**
3: calculate $error_1$ according to Equation 3.3
4: calculate $error_2$ according to Equation 3.4
5: **if** $error_1 \leq error_2 -$ threshold **then**
6: *spike_train*.add (time step τ)
7: *signal* [spike time $\tau : \tau$ + M] -= h
8: **end if**
9: **end for**

The decoding algorithm works as follows. The recreated signal starts off at 0. For every spike in the spike train the filter is added to the signal.

Pseudo-code for BSA-Decoding:

1: *signal* = zeros(signal_duration)
2: **for** *spike* in spike_train **do**
3: *signal* [spike time $\tau : \tau$ + M] += h
4: **end for**

Modified Temporal Difference (modTD) Algorithm

The modified Temporal Difference algorithm upgrades the TD algorithm by adding a memory variable that accumulates the residual error for each time step and integrates it into the spiking decision. This feature has been inspired by the BSA, which considers a wider part of the signal for each spiking decision.
The modTD algorithm works as follows. A memory variable *residual* is initialized to 0. Every time step τ, the difference between the current and the last value is added to the *residual*, which is then compared to a threshold: if the absolute value of the *residual* is bigger than the threshold, a spike is added to the spike train and the *residual* is reduced by the threshold. The sign of the *residual* defines the sign of the spike and thereby its mode. If the absolute *residual* is smaller than the threshold, there is no spike. The threshold can be optimized for different input signals.
The integration of a memory variable improves the encoding performance as large local errors due to steep slopes are no longer propagated but can be resolved in the following time steps.

Pseudo-code for modTD-Encoding:

1: *spike_train* = [ ]
2: *last_value* = 0
3: *residual* = 0
4: **for** time step τ in signal duration **do**
5: *residual* += *current_value* - *last_value*
6: **if** abs (*residual*) $\geq$ threshold **then**
7: *spike_train*.add (sign (*residual*) $\times$ time step τ)
8: *residual* = sign (*residual*) $\times$ (abs (*residual*) - threshold)
9: **end if**
10: *last_value* = *current_value*
11: **end for**

The decoding works as for the TD algorithm.

3.4 Stage 2: SNN Reservoir

The second stage can be separated into the generation of the network structure and the simulation of the network, which includes both training and classification. The network generation, meaning the positioning and connection of neurons, is executed on the host computer. Then the network is mapped and trained on a SpiNNaker machine using the sPyNNaker software.

Network Generation

In order to position the neurons in a meaningful way, the Talairach stereotactic brain atlas was used, which is a 3-dimensional coordinate system for the human brain, which is used to map the location of brain structures abstracting individual differences in the size and overall shape of the brain [45, 46]. The reservoir neurons are positioned on a regular, three dimensional grid on this coordinate frame, bound by the outer shape of the brain. The resolution of the grid, meaning the distance between two consecutive neurons in one dimension, was chosen to be 10 mm as this reflects the resolution of an EEG measurement when using the 10/20 system for electrode positioning [76]. The resulting reservoir comprises 1471 neurons and has a maximum expansion of 120 mm $\times$ 140 mm $\times$ 100 mm in x, y, and z. The 10/20 electrode positions are converted to Talairach coordinates using a mapping by Kössler [43]. The reservoir neuron positions and electrode positions are stored as lists of Talairach coordinates in the files 'neuron_positions.txt' and 'electrode_positions.txt' in the folder 'setup_stage_2'. Figure 3.3 shows the position of the Talairach atlas in the brain and visualizes a resulting reservoir with 1471 neurons and the respective 10/20 electrode positions.

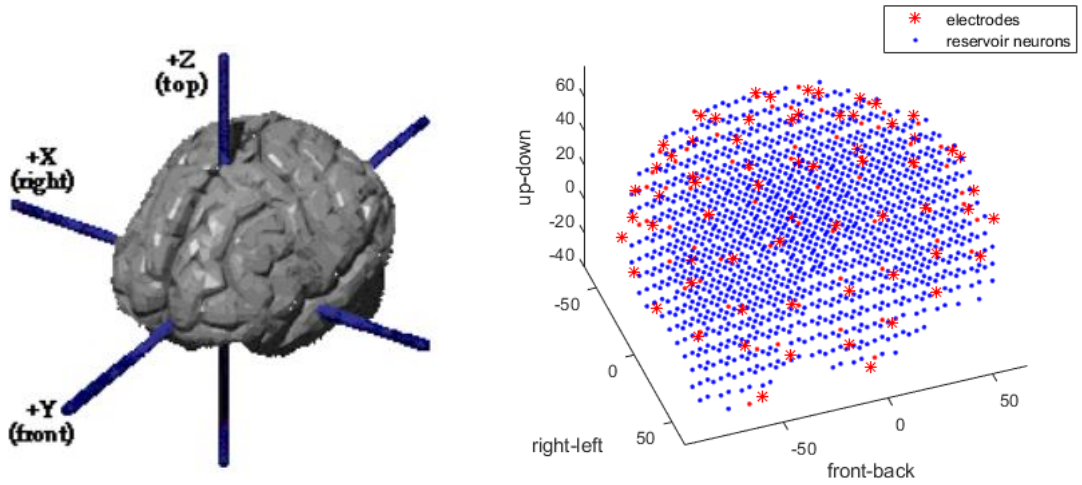

Figure 3.3: Talairach coordinate frame (left), from `talairach.org`. And a NeuCube reservoir with a grid density of 10 mm (right). Blue dots depict 1471 reservoir neurons. Red stars depict 65 electrode positions within the 10/20 system. Red dots depict reservoir neurons that serve as potential entries for the respective electrode measurements due to their spatial proximity.

After the neurons are positioned, the synaptic connections are initialized in a small world connectome [30]. Small world connectivity means that neurons that are close are connected with a higher probability than neurons that are further apart. For every neuron pair $\mathrm{N}_i = (x_i, y_i, z_i)$ and $\mathrm{N}_j = (x_j, y_j, z_j)$, the Euclidean distance is computed as $d_{i,j} = \sqrt{(x_i - x_j)^2 + (y_i - y_j)^2 + (z_i - z_j)^2}$. All distances are normalized to a range between 0 and 1 with respect to the maximum neuron distance in the network. Then, for each neuron pair, a connection probability $P_{i,j}$ is calculated according to Equation 3.5 [86]. Moreover, the maximum relative distance for connected neurons is set by a threshold value $\mathrm{d}_{\mathrm{thresh}}$ between 0 and 1, over which no connection is established.

$$P_{i,j} = \begin{cases} \mathrm{C} * \mathrm{e}^{-(d_{i,j}^{\mathrm{norm}}/\lambda)^2} & \text{if } d_{i,j}^{\mathrm{norm}} \leq \mathrm{d}_{\mathrm{thresh}} \\ 0 & \text{otherwise} \end{cases} \tag{3.5}$$

with C specifying the maximum connection probability and λ defining the small world connection radius. Figure 3.4 shows the relation between connection probability and inter-neuron distance for different small world parameters λ.

Based on the calculated connection probability $P_{i,j}$ and the user-specified inhibitory split parameter f, which defines the relative amount of inhibitory synapses in the network, the excitatory and inhibitory synapses within the reservoir are probabilistically determined, considering the following rules:

1. Input neurons cannot be solely inhibitory as this would prohibit a stimulation

of the network. Nonetheless, input spike trains that contain both inhibitory and excitatory spikes are permitted.

2. Neurons are not self-connected. Recurrence involves at least two neurons to prevent infinite spiking of a neuron by self-excitation.

3. There are no connections to input neurons in order to prevent a direct muting of input signals. Input neurons can only serve as pre-synaptic neurons.

4. The connection probability C_{inp} for input neurons is set higher than for reservoir neurons and their synapses are initialized with a higher weight than inter-reservoir synapses to enable and amplify the initial penetration of the network at sample presentation onset: $C_{inp} = amp \times C$. The input amplification factor amp is dependent on the number of input electrodes and the average firing rate per electrode and has to be adjusted to the data. For the Graz Data Set B it is chosen to be 5.

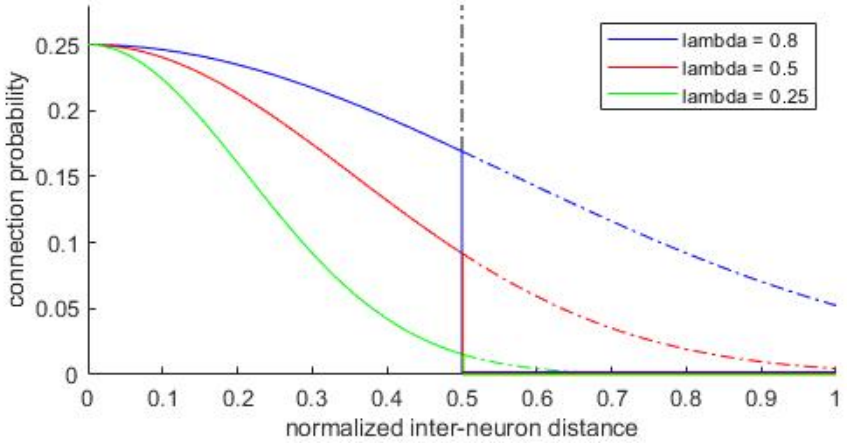

Figure 3.4: Connection probability function according to Equation 3.5 with a maximum connection probability $C = 0.25$ and a maximum inter-neuron distance of $d_{thresh} = 0.5$.

A synapse is defined by its pre- and post-synaptic neuron, a weight parameter specifying the contribution of a pre-synaptic spike to the excitation of the post-synaptic neuron, and a synapse delay parameter. The synapse delay is set proportional to the neuron distance, specifying the time a pre-synaptic spike needs after being fired to arrive at the post-synaptic neuron. This preserves the spatial arrangement and is an essential component of NeuCube. Figure 3.5 shows the distribution of synapse delays and therefore connection lengths in an example reservoir. Just as the surface of a sphere grows quadratically with its radius, the number of potential post-synaptic neurons for a synaptic connection grows quadratically with the distance from the pre-synaptic neuron. The distribution of synapse delays is the result of multiplying

the exponential connection probability function with the function for the number of potential synaptic connections within a certain distance. It should therefore resemble the shape of a function $f(x) = \mathrm{e}(-x) \times x^2$ with an abrupt cut-off at the maximum connection length defined by $\mathrm{d}_{\mathrm{thresh}}$.

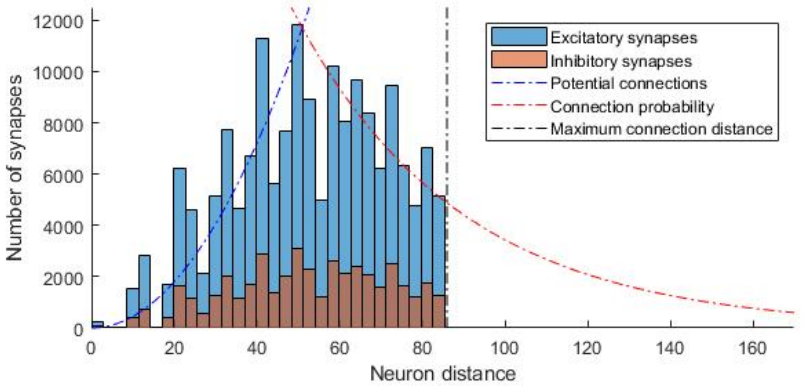

Figure 3.5: Distance distribution for an example reservoir showing the number of synapses with respect to the distance between pre- and post-synaptic neurons. The peaks at distances 10, 20, 30 and so on are caused by the positioning of neurons on a regular grid with a distance of 10 mm.

In order to generate characteristic spiking dynamics for different input classes, the network has to avoid typical bad spiking behaviours that are known in LSM and can significantly decrease performance in the reservoir [58]. Pathological synchrony describes the existence of infinite positive feedback loops which lock the network into a state of constant firing ('network explosion'). Thereby, sensitivity to input spikes is lost. Networks showing this behavior are also called unstable or chaotic. On the contrary, over-stratification describes a state in which input spikes are not propagated long enough to achieve an integration of inputs over time. The system loses its memory and thereby its ability to distinguish temporal patterns in a memory-less readout function. In a mechanical analogy these two states would be described as under-damped and over-damped. Figure 3.6 shows example spike raster plots for both pathological synchrony and over-stratification. A spike raster plot contains the spikes of neurons within a network over time and can be used to visualize network activity.
Reservoir stability can be analyzed by calculating the eigenvalues of the internal weight matrix, which contains the synapse weights of all connections within the reservoir [68]. The weight matrix is an asymmetric square matrix connecting pre-synaptic neurons (rows) to post-synaptic neurons (columns). According to matrix theory, the eigenvalues of a stable system lie within the unit circle in the complex plane. However, the eigenvalues of the synaptic strength matrix lie in a circle within a radius of $r = \sqrt{1 + (1 - f) \times \mu_{\mathrm{ex}}^2 + f \times \mu_{\mathrm{inh}}^2}$, with μ_{ex} and μ_{inh} being the mean excitatory and inhibitory synaptic weights respectively and f the inhibitory split.

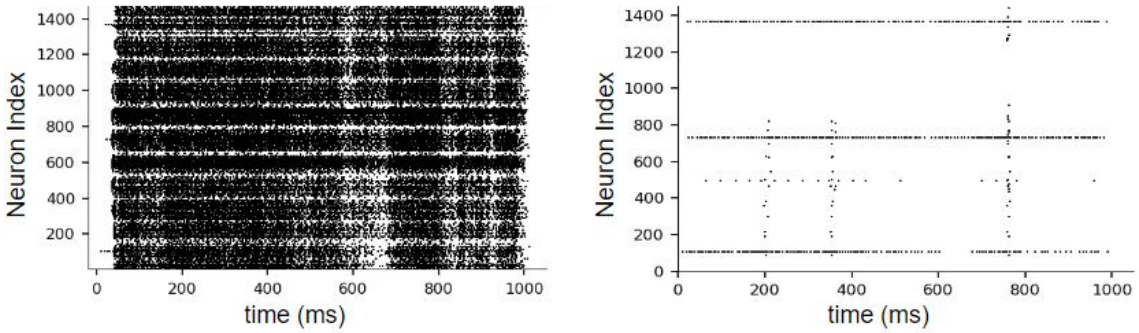

Figure 3.6: Bad reservoir behavior: pathological synchrony (left) and over-stratification (right).

As shown in Figure 3.7, this radius is larger than 1 if the excitatory and inhibitory means are different. Once the eigenvalues of the weight matrix grow beyond the unit circle, the network becomes increasingly unstable.
Rajan and Abbott suggest a stabilization approach that forces the eigenvalues to lie inside the unit circle [68]. By scaling the excitatory and inhibitory weights of every post-synaptic neuron in a way that their sum equals 0 it is ensured that, even if every reservoir neuron starts firing, the inhibitory spikes compensate the excitation in the system. This approach was not implemented as it would require a normalization of the weights and a time-consuming re-initialization of the network after every STDP learning step.
Instead, stability is approximated by initialization of a balanced network. The excitatory and inhibitory synapse weights are drawn from Gaussian distributions with different normalized means $\frac{\mu_{\text{ex}}}{\sqrt{\text{N}}}$ and $\frac{\mu_{\text{inh}}}{\sqrt{\text{N}}}$ but the same variance $var = \frac{1}{\text{N}}$ with

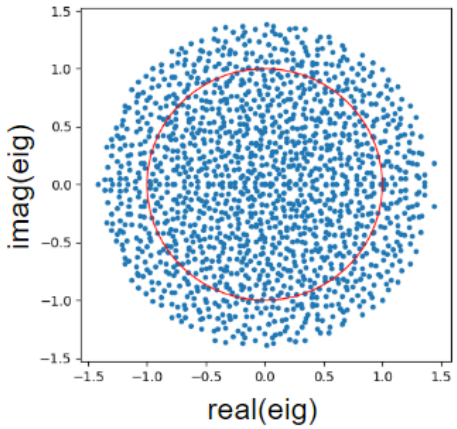

Figure 3.7: Eigenvalues of the synaptic weight matrix plotted in the complex plane for a balanced system with an inhibitory split of $f = 0.2$, excitatory synapse weight mean of $\mu_{\text{ex}} = 3.0$, and inhibitory synapse weight mean of $\mu_{\text{inh}} = 12.0$.

N being the number of reservoir neurons. In a balanced network the average of the combined excitatory and inhibitory distributions, from which synapse weights are drawn, has to be 0, resulting in a balance condition [68]:

$$(1 - f) \times \mu_{ex} - f \times \mu_{inh} = 0 \tag{3.6}$$

Once the user specifies the excitatory mean μ_{ex}, the inhibitory mean μ_{inh} is set automatically by the system according to the balance condition. If STDP is used in the system, it has to be tuned so that the network stays stable.
Once a network has been initialized, it can be saved and reloaded to be used for further training or classification. The required files can be saved automatically in the folder 'memory_stage_2'. A fully initialized SNN reservoir is depicted in Figure 3.8. In the end, the excitatory and inhibitory synapses are summarized in two lists to be processed in network simulation.

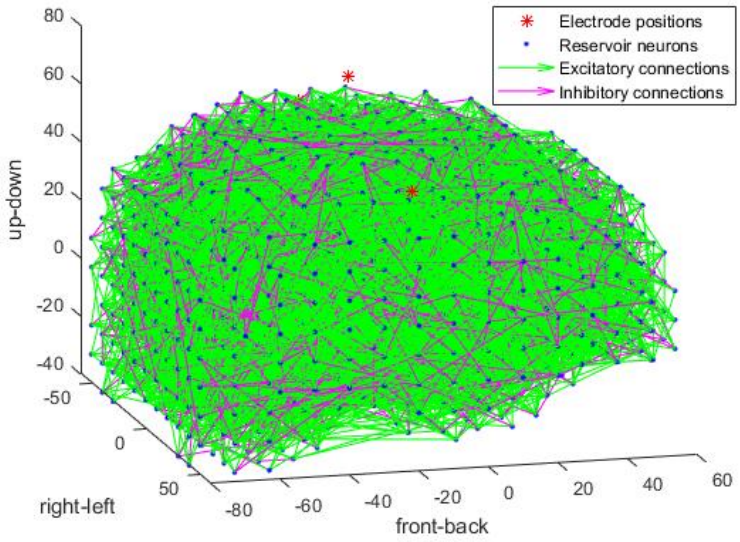

Figure 3.8: Fully initialized NeuCube reservoir. The inhibitory split for this network structure is $f = 0.2$.

Network Simulation

Once a network structure is initialized or loaded, it has to be translated and mapped onto SpiNNaker using the sPyNNaker software package. For real time simulations the simulation time step should be set to $t = 1\,\text{ms}$. The number of simulated neurons per core n_{core} should be distributed to minimize the workload on each core resulting in $n_{\text{core}} = \frac{1471\text{ neurons}}{72\text{ cores}} \approx 21\frac{\text{neurons}}{\text{core}}$. In sPyNNaker the user can choose between different neuron models to simulate spiking behavior. For this work we chose the exponential current-based LIF model (PyNN: *IF_curr_exp*) as it is computationally

efficient and we are not interested in a biologically plausible simulation on a particle level.
In PyNN, neurons are clustered in populations and connected by projections between populations. Projections can be defined from one population to itself or to other populations. In order to map the reservoir structure on SpiNNaker, a population of 1471 *IF_curr_exp* neurons is created. Although the neurons in the population have no three dimensional structure in the form of Talairach coordinates, the spatial information is preserved through the small world connectivity and the synapse delays.
For solely excitatory input spike trains, an input population is generated with the number of neurons matching the number of input electrode channels. The encoded input spike trains are fed into the input neurons using PyNN-supported *SpikeSourceArrays*. The input population is projected to the reservoir population connecting the input neurons to those reservoir neurons that are closest to the respective electrode positions with respect to their Talairach coordinates. If the input spike trains contain both excitatory and inhibitory spikes, the input population is automatically duplicated and the spike train is split into one excitatory and one inhibitory. Each spike train is fed into one population and both populations are projected equally onto the reservoir.
To enable plasticity in the reservoir, an STDP mechanism is specified by a timing rule and a weight update rule. As timing rule we chose the *SpikePairRule*, which means that the relative timing between pairs of pre- and post-synaptic spikes determines the weight updates. This rule has four parameters: τ_+, τ_-, A_+, A_-, which have been explained in detail in Section 2.3. As weight update rule we chose *AdditiveWeightDependence*, which means that the weight updates are added to the current values, as we want the reservoir to stay adaptive during the whole usage period. This rule requires the parameters $w_{\min}$ and $w_{\max}$, which limit the minimum and maximum weight of the synapse. The system can be easily modified to use different timing or weight update rules such as *MultiplicativeWeightDependence*. Fine tuning of the parameters is important for the system to achieve good performance.
As soon as the reservoir population and the STDP mechanism are set up, two projections are created from the reservoir to itself using a *FromListConnector* based on the excitatory and inhibitory synapse lists, which have been determined during network generation.
During the STDP and deSNN training phase, the training samples are sequentially linked to be presented in one simulation. To avoid interference between samples the input channels are muted for half a sample duration between two consecutive sample presentations.
While the simulation is running and the reservoir is adapting to the input data, the reservoir spikes and membrane voltages of the neurons are recorded. Furthermore, between every sample presentation the synaptic weights, both for the excitatory and inhibitory projections, can be saved to aid monitoring of synaptic plasticity. After the simulation has finished, the reservoir behavior, meaning the membrane voltages

and generated spikes, can be extracted from the board and visualized in a spike raster plot. The resulting reservoir network can be saved to reload it at a later stage for consecutive training or classification. In deSNN training, the recorded spikes are converted to a list which contains all the spikes that have been fired during the simulation. Each element of the list contains the neuron index and the spike time. The list is then returned to be processed by the deSNN classifier to generate the output classification network.
During the deSNN classification phase, single samples are simulated on a trained network. STDP-learning can be turned on and off by the user to enable or disable plasticity. An input population is generated and connected as in the training phase and the data is fed into the system. The reservoir spikes are recorded, extracted, and converted to a reservoir spike list as in deSNN training. The list is then returned to be processed by the deSNN classifier to generate a classification neuron and predict a class label for the presented sample.

3.5 Stage 3: Output classification

The third stage employs a deSNN for the final classification task and can be separated into network generation and sample classification. Stage 3 runs on the host computer. This section also elaborates on the separation metric, its calculation and meaning for the overall system.

Network Generation

For every training sample a new neuron is created within the output network and connected to every reservoir neuron except for the input neurons. All output neurons are independent as there are no connections within the deSNN. In this implementation there is no direct, live communication between reservoir and output neurons as the reservoir spikes can only be read out after the simulation has stopped. Every output neuron contains three feature vectors of 1471 elements based on its connections to the reservoir neurons: one for the initial weights; one that contains the total number of spikes on each synapse; and one for the final weights. The index of the reservoir neuron defines the respective location of the synapse in the feature vectors of the output neuron.
To calculate the initial weights based on the RO-learning rule a list of spikes, containing the reservoir neuron index and the spike timing for each spike in the reservoir, is extracted from stage 2. First, this list is sorted chronologically and the rank for each synapse is determined. The rank of a synapse is defined as the position of the first spike at this synapse in the chronologically sorted spike list. Based on the rank order the initial weights are calculated following Equation 2.7. After the initial weights are calculated, the remaining spikes on each reservoir neuron are counted to calculate the final weight vector following Equation 2.8. The network can be

extended and trained with new samples from both known and new classes at any time.

Classification

The actual class prediction task is performed by a KNN classifier from the SciKit-Learn library. The user can specify the number of neighbors k to perform the class vote and the feature to be used for fitting the model. The available features are the initial weight vector, the vector that contains the total number of spikes per synapse, and the final weight vector.
For the classification of a sample, a new neuron is trained outside the output network but with the same algorithm as the output neurons, meaning that all three feature vectors are calculated based on the reservoir behavior that has been induced by the test sample. For each classification task, the KNN classifier is fitted to the feature vectors of the training data using the built-in function *KNN.fit* and the sample feature vectors are classified using *KNN.predict*. The target labels for all training samples are provided in the folder 'input_stage_3'.

Evaluation

For the deSNN to work as a sampling function and the KNN classifier to work as a readout function, the SNN reservoir has to generate separable spike behavior. The separation of a liquid can be calculated using the metric *Sep* [25], which is specified in Equation 3.7 and 3.8. In LSM terminology, separation measures the average Euclidean distance between representative state vectors of each class that have been sampled from the liquid. In NeuCube terminology, separation is the average Euclidean distance between feature vectors of different classes. For each class, the respective feature vectors of the output neurons in the deSNN are merged to a representative center-of-mass (COM) vector. For every class pair the distance between their representative COM vectors is calculated. Finally, all class pair distances are averaged to obtain the separation metric. The further the COM vectors are apart, the easier it is for the KNN classifier to make correct class predictions. Goodman suggests that an increase of separation results in a higher classification accuracy [25]. This problem is similar to the margin optimization problem in SVM.

$$C_m(\mathrm{O}_i) = \frac{\sum_{o_j \in \mathrm{O}_i} o_j}{|\mathrm{O}_i|} \tag{3.7}$$

$$Sep(\psi, \mathrm{O}) = \sum_{i=1}^{\mathrm{N}} \sum_{j=1}^{\mathrm{N}} \frac{||C_m(\mathrm{O}_i) - C_m(\mathrm{O}_j)||_2}{\mathrm{N}^2} \tag{3.8}$$

where $C_m(\mathrm{O}_i)$ is the representative center of mass vector for class i, O_i the subset of state vectors assigned to class i with $|\mathrm{O}_i|$ elements, O a set containing all state vectors from N classes, and ψ denoting the liquid.

Notwithstanding, the separation metric does not consider the expansion of a class in the weight space, meaning the average distance of weight vectors to their representative COM vector. This is an important measure because if the separation is high (meaning the representative vectors are far apart) but the weight vectors of both classes overlap, a good classification will be difficult for a nearest neighbor classifier, as shown in Figure 3.9. The expansion metric Ex is introduced in this work to measure the average Euclidean distance between a state vector and the respective COM vector over all classes.

$$Ex(\psi, \mathrm{O}) = \sum_{i=1}^{\mathrm{N}} \frac{1}{\mathrm{N}} \frac{\sum_{o_j \in \mathrm{O}_i} ||o_j - C_m(\mathrm{O}_i)||_2}{|\mathrm{O}_i|} \tag{3.9}$$

The denser the state vectors of one class and the further the representative center of mass vectors of different classes are apart, the easier it should be for the KNN classifier to distinguish between different classes. The corrected separation S of the reservoir is expected be able to evaluate the internal dynamics of the model:

$$S(\Omega) = Sep(\psi(\Omega), \mathrm{O}) - 2 \times Ex(\psi(\Omega), \mathrm{O}) \tag{3.10}$$

where Ω is the parameter set for the reservoir creation.

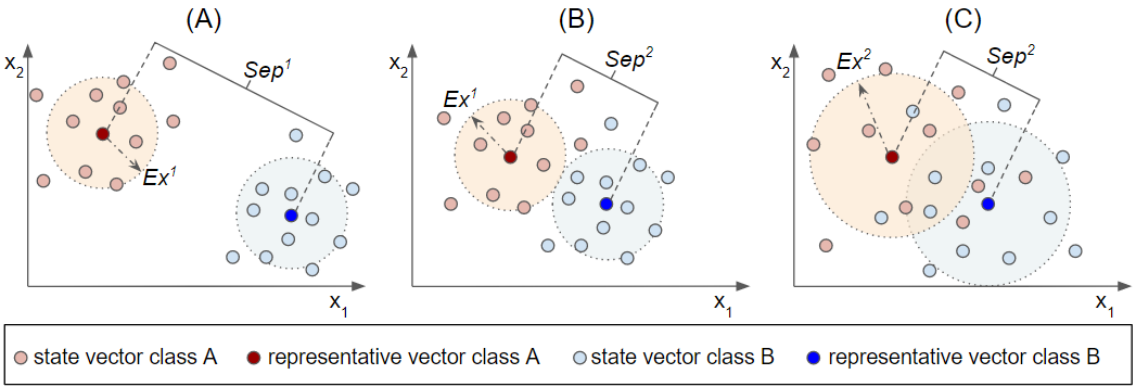

Figure 3.9: Conceptual influence of metrics *Sep* and *Ex* on the separation of a liquid. The state and center of mass vectors are represented in a 2D space. The light red/blue dotted circles denote the expansion of the respective classes A (red) and B (blue). Graph (A) shows a desirable example with a large separation Sep^1 and a small expansion Ex^1. In example (B) the classes have the same expansion Ex^1 but a smaller separation Sep^2, which decreases the overall contrast between classes. Example (C) finally has a low separation Sep^2 and a bigger expansion Ex^2 of the classes, which further impedes classification.

3.6 Code Architecture

The imlpemented code has been written following an object oriented design approach. Figure 3.10 depicts an UML class diagram. The class *NeuCube* implements the overall model and provides abstract methods to encode EEG-signals, initialize new SNN reservoirs as well as save and reload existing ones, train the reservoir and the output classifier, and finally classify new samples. A *NeuCube* object serves as the functional interface between model and user. It contains three objects, which represent the stages of the model and implement the methods of the NeuCube object: The *Encoder*, the *NeuCubeReservoir*, and the *Classifier*.

In the *Encoder*, the input training data is loaded, stored and converted into spike train data using the method *encode*, which takes the encoding method as an argument. The *Encoder* stores the spike data and measures performance in the encoding method for the respective samples. It can process all samples contained in the training data at once to train the reservoir but also encode single samples during the classification phase.

Every *NeuCubeReservoir* object contains a *NetworkStructure* object, which contains the positions of reservoir neurons, input neurons and the synapse lists for excitatory, inhibitory and input connections. The *NeuCubeReservoir* can either initialize a new reservoir network based on the given parameters for neuron connectivity, or load an existing network. The reservoir can then be trained on the input data via STDP learning or directly filter the input data to train the deSNNs classifier and classify new samples.

The *Classifier* object contains a list of *OutputNeuron* objects resembling the deSNN, a list with the respective class labels, and a KNN classifier, which classifies the synapse state vectors. Every *OutputNeuron* is defined by a set of three vectors: one for the initial weights for the reservoir connections; one for the numbers of spikes per synapse; and one for the final weights. A new *OutputNeuron* is initialized using the reservoir spike data of the respective sample, the target class label, and the specified learning parameters for RO and SDSP learning. The *Classifier* object has methods to add new output neurons to the deSNN and to classify neurons based on the trained network and an KNN classifier.

The implementation is portable to any operating system and has been successfully tested on Windows and Linux without changes to the code. In order to run NeuCube, execute the following steps (Links in Section 3.1):

- Install and setup Python, PyNN, sPyNNaker and the SciKit-Learn toolbox.
- Connect and setup a SpiNNaker board.

- Clone the NeuCube repository and set the design and control parameters in *NeuCube.py*.
- Run the file *NeuCube.py*.

Tables 3.2, 3.3 and 3.4 list and explain the design and control parameters that can be set by the user within the file *NeuCube.py*. All data that is needed for the model, i.e. the input EEG data as well as structural information to set up the reservoir, has to be stored in respective folders within the path of the NeuCube code. Tables 3.5 and 3.6 describe the naming convention for the folder structure.

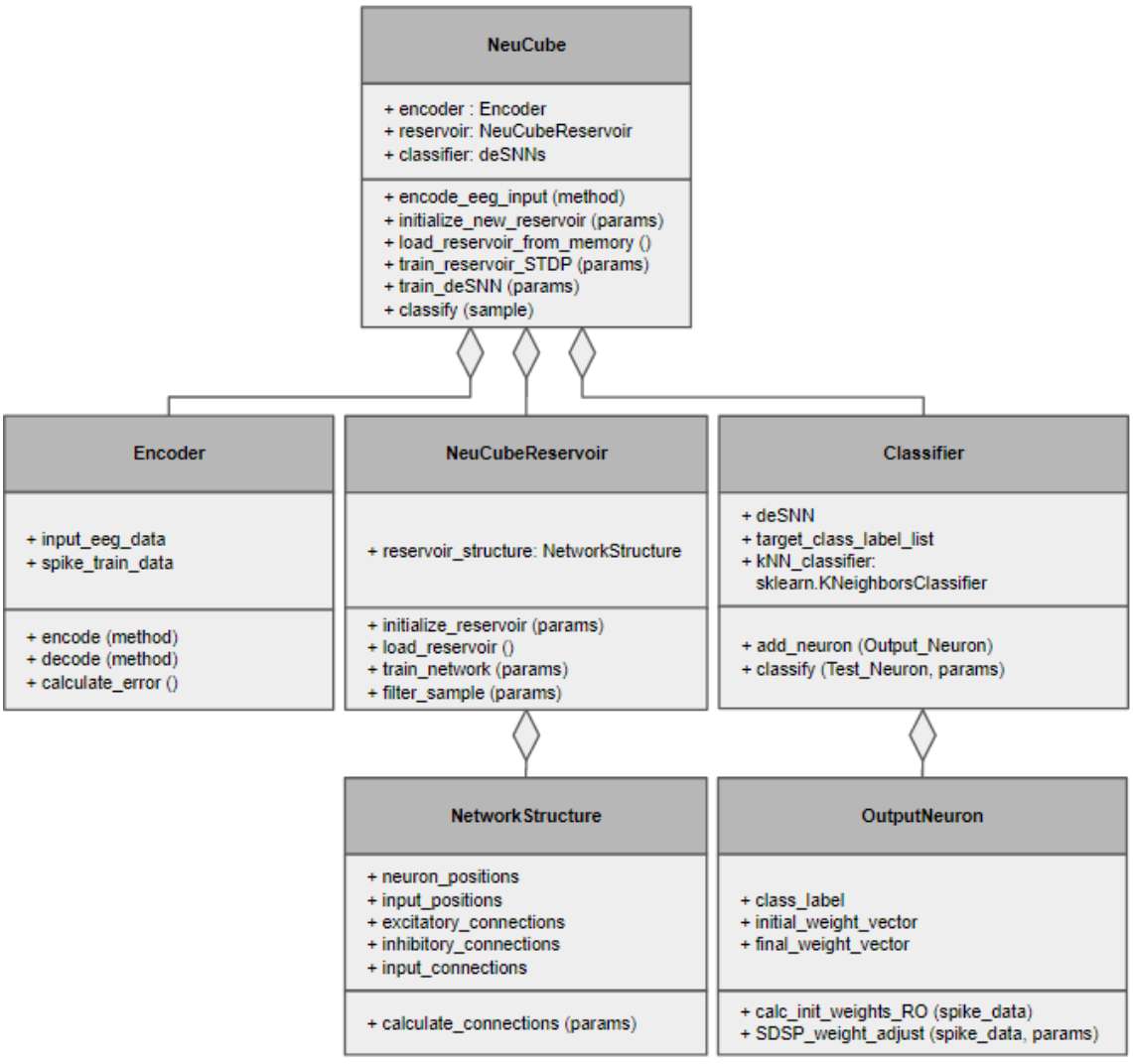

Figure 3.10: Simplified UML class diagramm for the implementation of the NeuCube model.

	Parameter	Example	Explanation
General	input_electrodes	['C3','Cz','C4']	Input electrode positions according to the 10-20 system. Input positions in the Talairach coordinate frame are automatically loaded.
	number_of_train_samples	80	Number of training samples for both reservoir and deSNN training.
	subject	'B04T_4-5'	Name of subject data folder in 'input_stage_1'.
	number_of_classes	2	Number of classes in the input data set.
	signal_duration	1000	Duration of the input samples in [ms]. In this implementation all samples must have the same length.
	signal_timestep	4	Time step in the input signal in [ms]. The time step is the reciprocal value of the sampling frequency.
Encoding	encoding_method	'BSA'	Method used for signal to spike encoding. Available methods are 'BSA', 'TD', 'mod_TD'. The system is automatically adjusted to the utilized spike modes.
Reservoir	inhibitory_split	0.2	Percentage of inhibitory synapses in the reservoir.
	connection_probability	0.25	Connection probability C for reservoir synapses from Equation 3.5.
	small_world_conn_factor	0.5	Small world connection radius λ from Equation 3.5.
	max_syn_len	0.5	Maximum relative connection distance over which no connection is allowed.
	w_dist_ex_mean	3.0	Mean of the Gaussian distribution from which initial excitatory synapse weights are drawn. It is automatically normalized by the suqare root of the number of reservoir neurons N. The variance is automatically set to 1/N. The distribution mean for inhibitory synapses is automatically calculated to fulfill the balanced condition in Equation 3.6
STDP	tau_plus	10	STDP design parameter τ_+ from Equation 2.2
	tau_minus	10	STDP design parameter τ_- from Equation 2.2
	A_plus	0.02	STDP design parameter A_+ from Equation 2.2
	A_minus	0.02	STDP design parameter A_- from Equation 2.2
	w_min	0.0	Limit for minimum weight value
	w_max	0.25	Limit for maximum weight value

Table 3.2: Design parameters of the NeuCube implementation (1/2).

	Parameter	Example	Explanation
SpiNNaker	neurons_per_core	30	Number of simulated neurons on each core. Setting this value too low leads to memory errors on a Spinn-3 board. The SpiNN-5 board has an on-board memory management system avoiding these errors.
	simulation_timestep	1	Time step of the simulation in [ms]. This value should be set to 1 for real-time execution.
	simulation_time	1000	Total time of the simulation per sample. This value should not exceed the input signal duration.
Classifier	k_neighbors	1	Number of neighbors integrated in the class-vote within the KNN classifier.
	alpha	1	RO learning design parameter from Equation 2.7
	mod	0.9	RO learning design parameter from Equation 2.7
	drift_up	0.01	SDSP design parameter from Equation 2.8
	drift_down	0.01	SDSP design parameter from Equation 2.8
	feature	'final_weights'	Classification feature defining the state vector to be used by the KNN classifier: 'initial_weights','spike_count','final_weights',
Testing	first_test_sample_index	81	Index specifying the first sample to be classified. This is usually set to number_of_train_samples + 1.
	number_of_test_samples	20	Number of samples to be classified.
	number_of_simulations	5	Number of simulations on the given parameter set. Multiple simulations have to be performed for each parameter set to average the results and obtain reliable information that is not dependent on the probabilistic uncertainly in the system.

Table 3.3: Design parameters of the NeuCube implementation (2/2).

	Parameter	**Example**	**Explanation**
Encoding	save_data	False	Saves the generated spike trains and performance measures to Â´memory_stage_1'.
	plot_data	True	Plots the original and recreated signal and the generated spike train for the first 3 samples.
Reservoir	new_reservoir	True	If true, it initializes a new reservoir. If False, it loads a predefined structure from 'memory_stage_2'.
	plot_stability	False	Plots the eigenvalue distribution in the complex plane for stability analysis.
	save_structure	True	Saves the generated reservoir structure (neurons and synapses) to 'memory_stage_2' to be analyzed or reloaded.
STDP	use_STDP	False	Turns on STDP learning within the reservoir. If False, the system directly trains the deSNN.
SpiNNaker	save_training_result	False	Saves reservoir spikes and synaptic weights to 'memory_stage_2'
	plot_spikes	True	Generates a spike raster plot to visualize reservoir spiking behavior.
	plot_voltage	False	Plots the membrane voltage of all reservoir neurons for the time of the simulation.
Classifier	load_reservoir_spikes	False	If True, it loads the reservoir spikes directly from 'memory_stage_3' to build the deSNN. If False, it runs a simulation in the input signals and extracts the spikes from the reservoir.
	save_reservoir_spikes	False	Saves reservoir spikes to 'memory_stage_3' after simulation.
	save_neurons	False	Saves the feature vectors of all deSNN output neurons to 'memory_stage_3'.

Table 3.4: Control parameters of the NeuCube implementation.

Folder Name	**Contained Files/Folders**	**File/Folder Fescription**
input_stage_1	B0ST_τ^{start}-τ^{stopp} $*$	Folders containing normalized EEG data files for subject S and time window [τ^{start},τ^{stopp}]. The samples within the folders are specified as 'sam_X.csv' containing an N $\times$ M array, where N denotes the number of input electrode channels and M the number of data points and X is the sample number.
input_stage_2	sam_X.csv	List of N encoded spike trains for sample X, where N denotes the number of input electrode channels and each spike train is a list of spike times in milliseconds.
input_stage_3	tar_class_label.txt	List of target class labels for training samples. The list index matches the respective sample number.
	reservoir_spikes_sam_X.txt	List of reservoir spikes for sample X. Each element is defined by the spiking neuron index and the spike time in milliseconds.
setup_stage_2	neuron_positions.csv $*$	N $\times$ 3 array, containing the Talairach coordinates (x,y,z) for N reservoir neurons.
	electrode_positions.txt $*$	65 $\times$ 3 array, containing the Talairach coordinates (x,y,z) for the 65 electrodes within the 10/20 system as published by Kössler [43].
	electrode_order.txt $*$	List of 10/20 system electrode labels. The list index indicates the order of electrode positions in the file electrode_positions.txt

Table 3.5: Folder structure and naming conventions for the NeuCube implementation (1/2). These folders must have the same path as the NeuCube Code files *NeuCube.py*, *Reservoir.py*, *Encoder.py* and *Classifier.py*. Folders and files that are marked with a $*$ are necessary to run the simulation. All other files are stored by the model and used for reloading or analysis.

Folder Name	**Contained Files/Folders**	**File/Folder Fescription**
memory_stage_1	error.csv	File containing the channel-specific encoding errors in an N × M array, where N denotes the number of training samples and M the number of input channels.
	sam_X.csv	Reconstructed signal for sample X in an N × M array, where N denotes the number of input electrode channels and M the number of data points
memory_stage_2	ex_syn_list_pre_training.txt inh_syn_list_pre_training.txt inp_conn_list.txt	Reservoir synapse lists for excitatory, inhibitory and input synapses before STDP training. Each list element is a synapse defined as *[pre_neuron,post_neuron,weight,delay]*
	ex_syn_list_post_X_training.txt inh_syn_list_post_X_training.txt	Reservoir synapse lists for excitatory and inhibitory synapses after STDP training with sample X.
memory_stage_3	X_neuron_rank_list.txt	Lists the ranks for firing reservoir neurons for sample X. Each list element is defined as *[neuron_index,first_spike_time,rank_order]*.
	X_spikes_per_neuron.txt X_initial_weights.txt X_final_weights.txt	Feature vectors generated by simulation of sample X containing the initial weights and final weights for synapses between reservoir and deSNN and the total number of spikes per synapse respectively.
results	test-subject-date-time-result.txt	Simulation results for a specific parameter set and multiple runs. The file contains the respective accuracy, variance, separation and the used parameter set.

Table 3.6: Folder structure and naming conventions for the NeuCube implementation (2/2). These folders must have the same path as the NeuCube Code files *NeuCube.py*, *Reservoir.py*, *Encoder.py* and *Classifier.py*. Folders and files that are marked with a *) are necessary to run the simulation. All other files are stored by the model and used for reloading or analysis.

Chapter 4

Results

The NeuCube implementation in this work was tested with two main purposes. First, we wanted to test the functionality of individual stages to validate the implementation. Second, we wanted to test the overall performance of NeuCube as a classifier for EEG data. Therefore, multiple experiments were designed during different stages of the project, which are presented in Section 4.1. The results of the experiments are discussed in Section 4.2, which elaborates on general design aspects of the implementation as well as the basic NeuCube model.

4.1 Experimental Results

When testing a complex system such as NeuCube, a major challenge is model calibration. Due to the complex interplay between different stages, the multitude of parameters involved in the process, and the probabilistic uncertainty that is introduced with every new reservoir, it is hard to analyze and establish the influence of individual parameters on the overall system performance.

Methodical Approach

One way to approach this issue is to perform an extensive test series that explores the whole parameter space at once and, in a best-case scenario, delivers the optimum parameter set for the provided input data. Nonetheless, this requires large data sets, time for simulation and training, and extensive computational resources. Furthermore, even though the parameter set might produce good results for the system, the tests do not necessarily provide information about the internal structure and processes that are responsible for system operation: the system is treated like a black box. We did not choose this approach as we are interested in understanding the system in order to adapt it to different applications and did not have the necessary resources.
Another approach is to break down the system and test individual stages in isolation. The individual stages have less parameters and are less complex. Therefore,

optimization of the stages is expected to be easier than optimization of the whole system. Notwithstanding, the individual optimization results do not necessarily apply for the overall system as inter-stage interactions are not considered during stage optimization tests. After modular optimization, the results can be assembled to test the integrated system. This is expected to be easier as findings of the modular tests can be included in integrative test design. We used this approach in this work as modular tests require less resources, offer more insights, and are better suited to validate the basic implementation.
The outcome of an optimization is highly dependent on the starting point. Therefore, the choice of initial parameters should be justified. Although parameter sets have been identified that produce biologically plausible behavior in small spiking networks between individual neurons [78], these sets are not directly transferable to more complex networks to produce good results as most artificial spiking networks differ too much from the human brain in terms of size and connectivity. For example, in a small network the firing threshold for post-synaptic neurons has to be set lower than in nature as neurons only receive input from a small number of pre-synaptic neurons compared to up to 10 k neurons in nature. The same result could also be obtained by increasing the weights of the individual synapses, which shows the interdependence of parameters. As a result we have to sacrifice biological plausibility for technical feasibility. We therefore started with an initial parameter set composed of parameters that have already been used in recent publications [25, 76, 10, 82]. Parameter changes were made based on the observed network behavior, justified by experience, and trial and error.

Modular Test Stage 1: Spike Encoding Performance

Initially, 7 randomly chosen, normalized, single-channel EEG-signals from the Graz Data Set B were analyzed to evaluate and optimize the encoding performance of the implemented methods. Each test signal was encoded into a spike train, which was again decoded to reconstruct the signal. Example graphs for all methods showing the original signal plotted against the reconstructed signal and the generated spike train

	sample								
Method	1	2	3	4	5	6	7	**Mean**	Var
TD	0.23	0.79	0.88	0.54	1.17	0.85	0.91	**0.77**	0.09
modTD	0.06	0.18	0.17	0.24	1.16	0.08	0.15	**0.15**	0.004
BSA	0.12	0.14	0.14	0.28	1.15	0.28	0.13	**0.18**	0.005

Table 4.1: Error metric as defined in Equation 3.2 for 7 randomly chosen, normalized, single-channel EEG samples from the Graz Data Set B. Results are shown for the Temporal Difference (TD) algorithm, the modified Temporal Difference (modTD) algorithm and the Bens Spiker Algorithm (BSA) in each row. The last two columns show the average performance of the encoders.

are depicted in Figure 4.1. The original and reconstructed signal were compared using the error metric introduced in Equation 3.2.
Our hypothesis was that all encoders are able to encode the data and reconstruct the signal with a mean error of less than 0.2. For the BSA encoder we used a 7-tap filter with cut-off frequency $f_{\text{cut-off}} = 0.1\,\text{Hz}$ and a threshold of 0.679. For both TD and modTD we used a threshold of 0.05. Table 4.1 provides an overview of all encoding results. The modTD and BSA encoder reached a mean encoding error of 0.15 and 0.18 and a variance of 0.004 and 0.005 respectively. The TD encoder reached a mean error of 0.77 and a variance of 0.09.

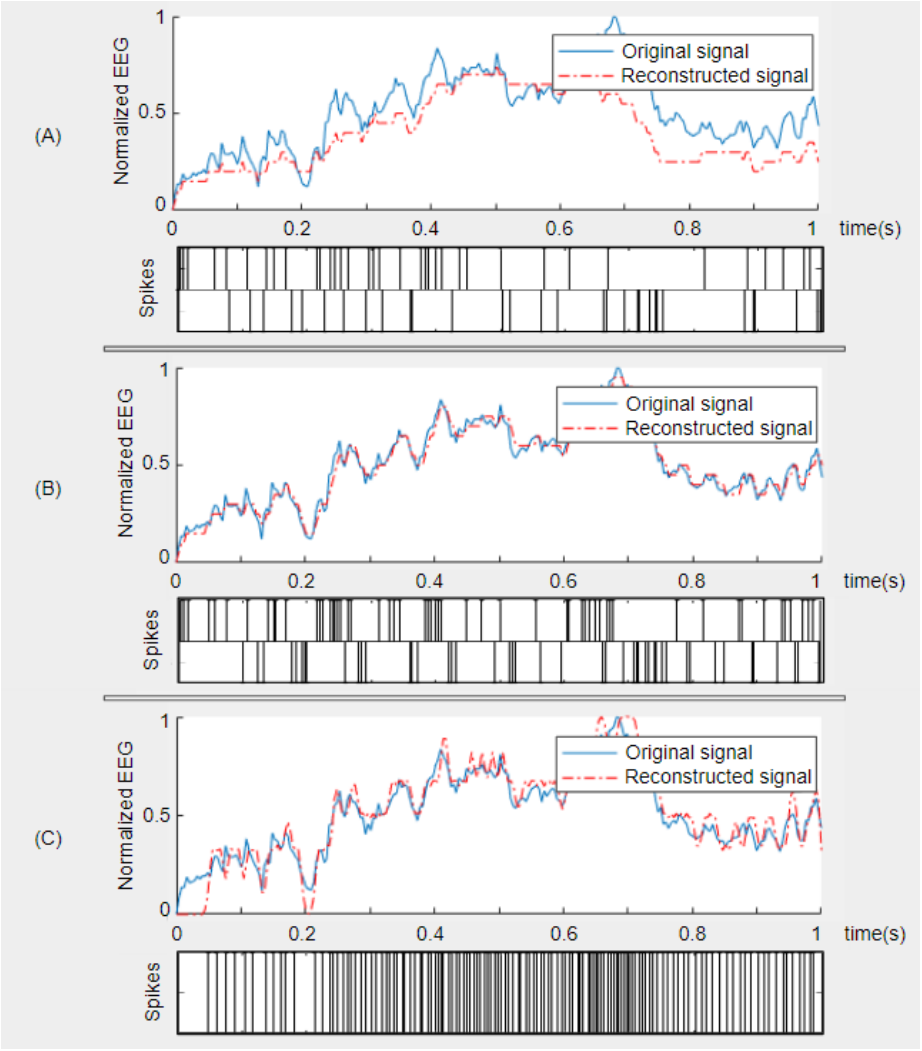

Figure 4.1: Signal to spike encoding and signal reconstruction for a randomly picked sample from Graz Data Set B. In the upper plots, the blue continuous lines show the original, normalized EEG-signals. The red dotted lines show the reconstructed signals. The lower plots show the respective spike trains for the encoding methods TD (A), modTD (B), and BSA (C). Table 4.1 lists all performance metrics in detail.

Modular Test Stage 2: Reservoir Activity and STDP Learning

In order to validate stage 2, we performed one test with manually generated spike trains to monitor STDP learning in the reservoir and one test with encoded EEG data to analyze stability. In the STDP test we designed Spike Source Arrays and reservoir synapses in a small network to cause potentiation and depression for specific synaptic weights. Our hypothesis was that the synapses are able to adjust according to relative spike timing in incoming spike trains. Comparing the synaptic weights before and after simulation showed that potentiation and depression modified the weights as expected.
In the second test we disabled STDP and excited the reservoir with encoded EEG signals and plotted the activity in a spike raster graph, as shown in Figure 4.2. Our hypothesis was that the raster graphs clearly display a separation in the reservoir by showing different spike activities for samples from different classes. The stability of a network appeared to depend on the right balance between connection density, inhibitory split, and synaptic weights. While a high connection density and high weights led to pathological synchrony, few connections and low weights resulted in over-stratification. An optimization of the parameters in isolation was not possible as visual inspection of the raster graph only allowed us to observe instability but not to inspect separation. Nonetheless, we noticed that the application of STDP on large networks led to run-time errors on SpiNNaker, which was presumably caused by increasing instability of the system. As the simulation was interrupted before synapse weights could be extracted, we were not able to analyze the eigenvalues of the weight matrix after learning. This is why we disabled STDP learning for the other optimization tests.

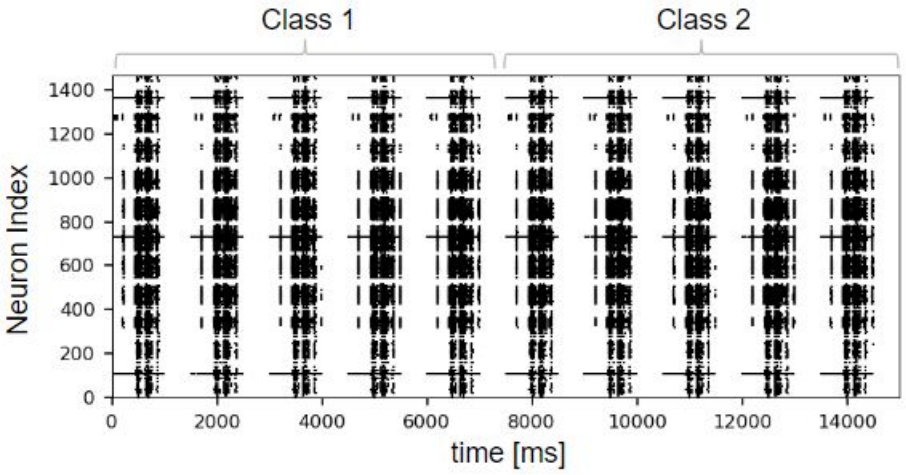

Figure 4.2: Spike raster graph for reservoir activity during presentation of 10 EEG samples of subject 4 from the Graz Data Set B. Samples are clustered by classes. A separation of the liquid is not perceivable.

Modular Test Stage 3: Approximation of the Classifier

To validate stage 3, the classifier was tested for approximation. Approximation refers to 'the reading function's ability to classify the state vectors acquired from the liquid' [58]. To validate approximation in NeuCube, the KNN classifier has to be able to distinguish between different feature vectors created by the RO- and SDSP-learning rules based on the reservoir spiking behavior after stimulus presentation. Our hypothesis was that the deSNN is able to learn and identify different reservoir dynamics under the assumption of separation in the liquid.
To validate the assumption for the test, separable reservoir behavior was generated by designing a set of four test samples. Three functional signals were created representing a 1 Hz cosine, 7 Hz sine, and a linear function. An additional idle signal was created containing mostly zeros to represent quiet channels. In order to match the number of electrode channels in Graz Data Set B, each sample comprised three channels, each containing one of the test signals. Three tests were performed with a sample structure increasing in complexity according to Table 4.2. Each test was conducted once for each classification feature. Regardless of the sample complexity the system was able to recognize all training samples with 100% accuracy.

	Test 1			Test 2			Test 3		
Sample	ch1	ch2	ch3	ch1	ch2	ch3	ch1	ch2	ch3
1	**cos**	*idle*	*idle*	**cos**	*idle*	*idle*	**cos**	**lin**	**sin**
2	*idle*	**sin**	*idle*	**sin**	*idle*	*idle*	**sin**	**cos**	**lin**
3	*idle*	*idle*	**lin**	**lin**	*idle*	*idle*	**lin**	**sin**	**cos**
4	*idle*	*idle*	*idle*	*idle*	*idle*	*idle*	*idle*	*idle*	*idle*

Table 4.2: Sample structure for approximation tests. For each sample and test the Table shows which signal was used in which channel. **cos** denotes a 1 Hz cosine, **sin** denotes a 7 Hz sine, **lin** denotes a linear signal, and *idle* describes a quiet channel.

Integration Test 1: Electromyography (EMG) Data

A first integration test was performed on bipolar, 4-class Electromyography (EMG) data to demonstrate correct operation of the system. Our hypothesis was that the system is able to classify the recorded and pre-processed EMG signals with a significantly better-than-chance accuracy. The data was recorded in a student project at NST, TUM.
The EMG signal contained four channels sampled at 512 Hz from 3 subjects. The classes corresponded to the movements of grasping with: two fingers (thumb and pointer); three fingers (thumb, pointer and middle finger); the whole hand; and a resting state. There was no pre-processing for artifact removal or sorting of bad data. The signals were band-pass and notch filtered, rectified and normalized to a range between 0 and 1 with respect to the highest value in a recording session, in

order to preserve relative DC amplitude differences between channels and samples. Afterwards, for each class and subject 36 samples were created containing 200 data points, resulting in 144 samples per subject. The envelopes of the filtered, normalized signals were used as input signals to the encoder. We chose the electrode positions of C5, Pz, C6, and Fz as input positions to obtain a maximum spatial distribution within the reservoir.
We used EMG data in this initial test because it is easier to classify than EEG data as the features are well known and the data structure is less complex. The amplitude of an EMG signal correlates with the muscle strength at the electrode position. Therefore, for different grasping movements we expect different amplitude patterns on the electrodes. As the amplitude gets converted into spike density by the encoder, we expect a clear and perceivable separation in the reservoir.
Without tuning the encoder the generated input spike trains had an average encoding error of 0.52 with a variance of 0.169 for all 144 samples. After a very short period of network parameter tuning with respect to data from subject 1, the model was trained on 120 samples (30 per class) without STDP and tested on the remaining 24 samples. Multiple tests were conducted for which different sample sets were chosen for training and testing. Depending on the sample split the model achieved an average balanced accuracy of 75% (62%, 79%, 79%, 79%). Figure 4.3 shows example reservoir spike behavior in a raster graph. The separation of the reservoir is clearly visible: every class has a different characteristic reservoir firing pattern. The same parameter set was used to classify the data from subject 2 and 3 resulting in lower a accuracy (both 67%). Other tests with minor parameter changes resulted in a significantly worse classification performance for subject 1 (accuracy between 37% and 54 %) as well as turning on STDP learning (62%).

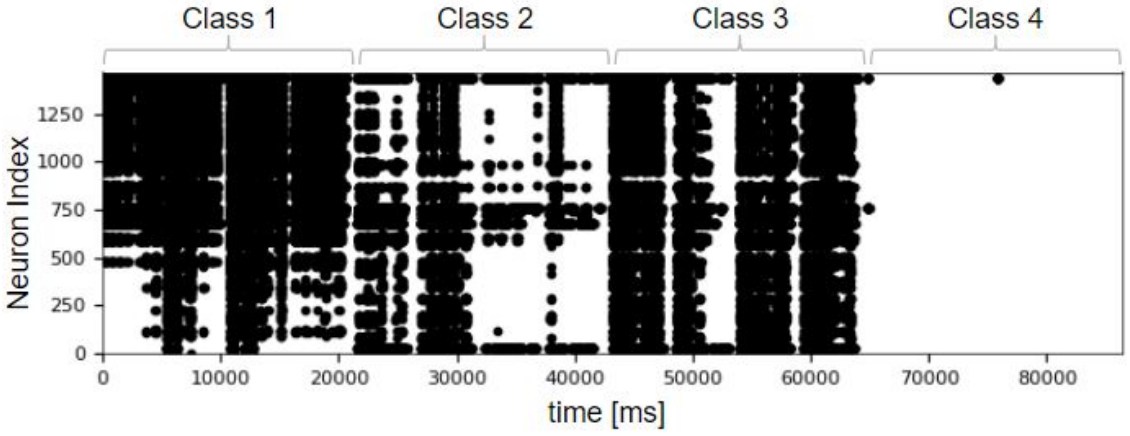

Figure 4.3: Spike raster graph for reservoir activity during presentation of 144 EMG samples of subject 1, clustered by classes. Both the separation of the liquid as well as bad input data is clearly visible.

Integration Test 2: Graz Data Set B (EEG)

For the final parameter optimization and performance testing of the integrated model we used primarily training data B04T collected in subject 4 within the Graz Data Set B and a time window of $\tau = [4-5]\ s$ as it showed good results for traditional classification methods. After removing the artifact afflicted samples, noise filtering, and normalization, we ended up with 399 samples. The sample set showed a mean encoding error of $\mu_{\mathrm{BSA}} = 0.16$ with a variance of $var_{\mathrm{BSA}} = 0.001$, $\mu_{\mathrm{TD}} = 1.02$ with a variance of $var_{\mathrm{TD}} = 1.133$, and $\mu_{\mathrm{modTD}} = 0.20$ with a variance of $var_{\mathrm{modTD}} = 0.002$. Due to these results we used the BSA encoder for further testing. Our hypothesis was that the implementation is able to classify 2-class EEG data with an accuracy of significantly more than 50% (chance).

Recognition is the easiest version of classification as the classifier does not have to perform any abstraction of the samples. Therefore, in a first test the NeuCube model was trained with 10 samples on the initial parameter set. Afterwards the model was tested on the same 10 samples from the training data. The algorithm was able to identify the training samples with an accuracy of 100%.

Next, we tried to classify new samples. The data was split into bunches of 5 samples per class. To train and test the system, we used the same number of samples for each class to obtain a balanced classification accuracy. In order to reduce the number of parameters, keep the network stable, and to be able to assess initial separation in the reservoir, STDP learning was turned off. As first classification tests on the initial parameter set did not result in better-than-chance accuracy, we performed a grid search on the parameters *connection_probability*, *inhibitory_split*, and *mean_excitatory_weight* using an 80-20 data split for training and testing. For each parameter 6 values were tested while holding the others constant. Each data set was simulated 10 times to compensate for probabilistic variability resulting in 180 simulations and an overall simulation time of around 30 hours. Although individual simulations reached accuracy values of up to 70%, averaged accuracy measures did not significantly exceed chance. Some parameter sets showed accuracy deviations of more than 25% from the mean for individual simulations.
As the initial parameter set did not lead to reliable results, we explored the parameter space further by choosing 16 parameter combinations based on our understanding of the network. Some parameter sets were designed to create reservoirs with a very dense connectivity but a small maximum connection length and low initial weights. Other sets were designed to have fewer but longer and stronger connections. Each set was simulated five times resulting in 80 simulations in total. Among all sets only one resulted in an average accuracy above 55% and at the same time no single simulation accuracy fell below 50%.
Next, we focused on this parameter set and created another 25 parameter sets with only one variable differing from the identified set. In addition, 12 slightly modified

parameter sets were applied on data from the same subject but using a different time window $\tau = [5 - 6]\, s$. Each set was simulated 5 times resulting in 185 simulations. For both time windows the starting parameter set gave the maximum mean balanced accuracy of 64% with a standard deviation of 3.7%. Notwithstanding, most modified sets were not significantly better than chance.

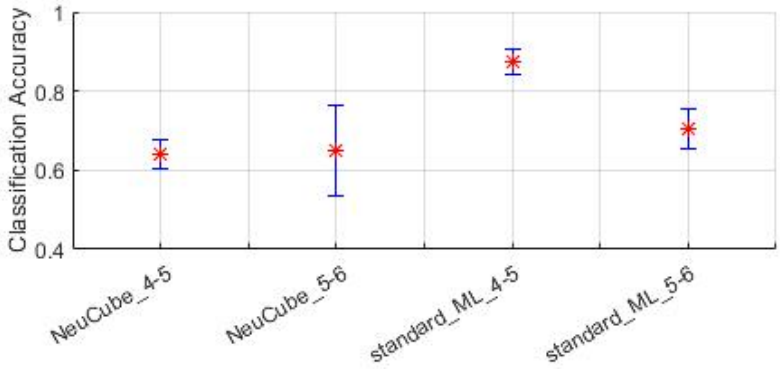

Figure 4.4: Results of the EEG performance test comparing the mean accuracy of NeuCube simulations on the final parameter set for two different time windows with the averaged accuracy over standard machine learning techniques as listed in Table 4.3. For the NeuCube simulation we used the parameter set as listed in Tables 3.2 and 3.3.

As the change of network parameters did not appear to improve the accuracy significantly, we modified other parameters. A change of classification parameters or encoding method did not improve performance in the system. However, moving the input positions for electrode spike trains resulted in an average accuracy of 65% with a standard deviation of 11.4% and a peak accuracy for individual simulations of 85% for an 80-20 training-testing data split and a time window $\tau = [5 - 6]\, s$.
The final parameter set resulting in the best classification accuracy is given in Tables 3.2 and 3.3. Figure 4.2 shows example reservoir spike behavior for EEG data.

4.2 Discussion

4.2.1 Experimental Results

This section discusses the experimental results. We differentiate between conceptual and functional validation. Conceptual validation means that the implementation has been examined and tested using manually created or simplified test data and is running as expected. Functional validation means that the implementation has been successfully tested on real EEG data and is ready to use.

Modular Test Stage 1: Spike Encoding Performance

For the BSA and modTD encoding methods the hypothesis was verified. Both methods could reliably encode and decode the provided signal with mean errors smaller than 0.2. Figure 4.1 shows that the original signals could be successfully reconstructed.
For the TD method the hypothesis could not be verified. The low encoding performance can be methodically explained: as the algorithm only considers two consecutive data points and a constant threshold in its spiking decision, inclination that can be encoded and reconstructed by the algorithm is limited to $inc_{\max} = \frac{\text{threshold}}{\text{time step}}$. If the threshold is set large, low-frequency high-amplitude components can be encoded but fast and small changes are filtered. For small thresholds the contour of the plot can be encoded but particularly if the signal starts with a large offset, the initial DC offset error can be propagated through the entire encoding process. Especially TD and modTD might be useful for data which is normalized to 0 mean and contains negative values as they allow for inhibitory spikes and can recreate negative signals. All in all, the first stage was conceptually and functionally validated. The BSA and modTD methods were used for further testing.

Modular Test Stage 2: Reservoir Activity and STDP Learning

Our hypothesis for STDP learning in a test network was verified as the measured weight changes matched with our expectation. Notwithstanding, for larger and more complex networks STDP led to run-time errors on SpiNNaker. This is most probably caused by neurons getting out of balance. If the sum of all synapse weights at a neuron exceeds 0 for most neurons in the network, then the overall level of excitation will increase, and could lead to instability. If the sum falls below 0, then the excitation of the network will be inhibited, which could decrease the integration of spikes over time. Only if STDP is tuned correctly can it improve the separation in a reservoir [58]. In order to analyze this effect, one could examine the eigenvalue distribution of the weight matrix during STDP learning and test if a normalization of the weights at each neuron to maintain a sum of 0, as suggested by Rajan [68], keeps the network in balance. However, on SpiNNaker it is not trivial to extract and modify synaptic weights during a simulation. Therefore, integrating an autotune mechanism on SpiNNaker that keeps neurons during STDP learning in balance might be a reasonable task for future applications.
The second hypothesis on visibility of separation in a raster graph was not validated. For EEG data we were not able to distinguish between samples from different classes. Furthermore, when tuning the parameters it was difficult to decide, when a desired spiking activity in terms of number of spikes per channel or total number of firing neurons had been reached. We expect a more sophisticated visualization as used by Kasabov [39] to give the reservoir spikes more meaning and make manual parameter tuning by visual separation observations easier. Visualizing the network behavior and structure in 3D might also help to understand the development of synaptic

connections during STDP and help to find find proper learning parameters.

Modular Test Stage 3: Approximation of the Classifier

The approximation test was able to verify the stated hypothesis as the classifier was able to recognize all test behaviors of the reservoir. This shows that, under the condition of separable reservoir behavior, the RO and SDSP learning are in principle suitable to create representative state vectors in the weight space, and the KNN classifier is able to compare such vectors to recognize and predict class labels. This conceptually validates the third stage.
Nonetheless, this test does not allow any conclusions regarding the effectiveness of RO and SDSP learning in EEG applications but only validates the basic implementation. This is because, even if the reservoir was able to separate different classes, we would expect the reservoir spike dynamics to be significantly more complex. However, adjusting the sampling function in order to create state vectors that reflect the discriminative information from the SNN is hard, as it is unknown which information is relevant to decode specific MI classes and how this information is encoded in the spike trains of the reservoir. The tuning of the sampling function is therefore one crucial aspect to include in future research.

Integration Test 1: Electromyography (EMG) Data

The successful classification of EMG data verifies our hypothesis and validates the implementation of the overall system. The high encoding error and decreased performance in tests with different subjects and parameters show the high sensitivity of the system towards different input sources and subjects, and emphasizes the importance of a properly tuned parameter set for encoder, reservoir and classifier. As the discriminative features within the data and therefore the desired behavior of the reservoir were known, the parameters could be easily adjusted by visual inspection of a spike raster plot, which showed clear separation for correct parameter sets as shown in Figure 4.3. Although a general class-specific firing pattern is visible, there are samples that do not match their respective class pattern. This may be due to recording problems and bad input signals, encoding errors, or general fluctuation of the EMG signal as people may not always perform the same movement with the same muscle strength. A higher force leads to a higher spike rate, which in turn may induce spikes in the reservoir that are not typical for a class. These atypical samples explain incorrect classifications and the comparably low accuracy for EMG classification tasks. For comparison, an SVM classifier with upstream feature extraction achieved an accuracy of 86.5% on the same data set.
In general, it is questionable if NeuCube gives any advantages for the classification of non-brain data. As the brain-like spatial arrangement of neurons has no meaning in other contexts, a standard LSM with a regular neuron grid would probably suffice to achieve the same results. Instead, in order to use the properties of NeuCube, for every data source one could create a new reservoir structure imitating the respective

spatial nature of the input data source. This might help to extract spatial information in non-brain related input data as well. Nevertheless, the model was able to classify the presented EMG samples with significantly higher-than-chance accuracy and, thus, the overall system was conceptually validated.

Integration Test 2: Graz Data Set B (EEG)

There are certain aspects that differentiate the classification of EEG data with NeuCube from regular classification problems and classification of different input signals such as EMG: the features that are significant for the decoding of different classes are partly unidentified; in turn, the nature of desired liquid dynamics for different classes is unclear; and therefore the requirements and necessary properties for a sampling function and output classifier are unknown. Therefore, it is extremely difficult to manually tune and understand the influence of interdependent parameters on the internal processes of the system.
Although out tests verified the hypothesis as there were simulations that reached an average and peak accuracy of up to 65% and 85% respectively, they also showed that performance in the system highly depends on the parameter set, the input data split, and the subject. Using solely excitatory input or allowing both excitatory and inhibitory spikes had no effect on the performance. Some parameter sets even showed a variability of up to 25% when applied on the same data multiple times. We expect this performance variability to be caused by the probabilistic initialization of the reservoir.
Every neuron in the reservoir can conceptually represent some spatio-temporal information from the input data based on its connectivity and spatial arrangement. However, only a limited number of these connections is initialized probabilistically resulting in a different network for each initialization. Only if the randomly created connections match the discriminative features within the input spike trains - which we do not know - the reservoir is able to represent this information by spiking neurons. Based on this assumption we recommend future projects to start off with high initial connectivity, a high neuron density, and a low average synapse weight to enable the system to represent as many features as possible while staying stable. Afterwards, synaptic plasticity mechanisms should be used to strengthen these connections and neurons that contribute to the representation of class-specific features and to remove those connections that are not involved and do not transmit spikes. Although STDP is a learning mechanism for potentiation and depression of synaptic weights, it should be supplemented by another mechanism for structural plasticity that can actually delete connections and prune the network. As long as the variability of test results is high, multiple simulations have to be executed to obtain representative results, which is time-consuming. Nonetheless, variability should be small to guarantee reliable operations.
In order to reduce the number of parameters for network initialization we suggest to analyze the interdependence of parameter pairs in more detail. While the mean

for the synaptic weights is already normalized to the number of reservoir neurons, it should also be auto-tuned to the connectivity in the network. An increased connection probability results in a higher number of synapses per neuron. Therefore, the average synaptic weight must be reduced to maintain the overall spiking behaviour as more spikes can arrive at a neuron at a time. We expect the eigenvalue distribution of the weight matrix to be a reliable indicator for the stability of the network. Another aspect to elaborate on is the structure, number and mapping of neurons in the reservoir. Single tests have indicated that a spatial distribution of the input signals may lead to higher accuracy. This seems reasonable as an increased distance between input neurons results in a higher number of neurons in between to represent spatio-temporal information. The same concept indicates the benefit of increasing neuronal density in the system, as the number of neurons in the brain involved in motion coding is probably higher than the used 1471 neurons in the reservoir. However, if we only want to simulate neuronal activity in the motor cortex, another basic shape of the reservoir could provide a better spatial representation.
Lastly, although EEG measures the LFPs and not the activity of single neurons, the encoded spike trains are assigned to single input neurons in the reservoir. Developing a mapping between 10/20 electrode positions and the reservoir that better mirrors the nature of EEG, might also improve performance in the system. One approach to measure the information loss in electrode mapping is to calculate LFPs for the appropriate parts of the reservoir to reconstruct the EEG measurements and compare the results with the original recordings. However, the calculation of LFPs requires detailed information about the cellular structure and orientation of neurons, which is hard to obtain as measurement techniques with the necessary resolution do not exist yet and the specific neuronal structure varies among people.

Comparison with State-of-the-Art Results

Table 4.3 shows classification results from standard classifiers for data from Graz Data Set B, subject 4, and time windows $\tau = [4 - 5]\,\mathrm{s}$ and $\tau = [5 - 6]\,\mathrm{s}$. All classifiers used an upstream feature extraction to filter ERS and ERD from the signal. Subject 4 has been chosen because ERD and ERS features are distinctively prominent in the respective signals. For a time window directly after imagination onset, classification accuracy reached between 80 and 90%, which is comparable to state-of-the-art results in the literature for 2-class EEG classification [89, 4]. As we know that ERD and ERS features can only be seen around imagination onset and offset, we expect the accuracy to drop if a time window between on- and offset is used. The tests confirmed this expectation. For input signals that were recorded during imagination the results dropped to 65 to 75% as shown in Figure 4.4. These results show that standard classifiers perform well on data of which the internal information structure is known and can be extracted, but forfeit performance if the features are unknown, which was our initial assumption and motivation to use NeuCube.

In comparison, our implementation reached an averaged accuracy of 64 and 65% for the respective time windows. State-of-the-art results for EEG classification with NeuCube show an accuracy (without STDP learning) of 80% for 3-class MI data [10], which is comparable to standard approaches [11], showing that a big performance loss is probably due to incorrect tuning of the parameters resulting in a poor SNN feature filter and an inappropriate sampling function.

Time Window	KNN	LDA	Log. Regr.	MLP	Naive Bayes	Quadr. LDA	Rand. Forest	SVM	Tree
t = [4,5] s	86.6	90	89.2	87.5	90	81.7	89.2	90	83.3
t = [5,6] s	64.2	75.8	75	73.3	66.7	66.7	73.3	75	64.2

Table 4.3: Classification accuracy (%) for subject 4 from Graz Data Set B after 10-fold cross validation with shuffling. The results were generated using an open source EEG classification library called gumpy (`www.gumpy.org`), which has been developed at NST, TUM.

4.2.2 Implementation and NeuCube Model

This section discusses the implementation on SpiNNaker and the NeuCube model itself and suggests future research to improve performance and reliability of the implementation in a MI context.

Implementation

The SpiNNaker architecture was used in this work due to its energy efficiency and parallel processing capabilities to enable portable, real-time BCIs. Nonetheless, as only the reservoir stage is directly executed on SpiNNaker, there are still some aspects that limit real-time application and mobility. First, as the encoding and classification stage are still performed on a standard PC, the low energy principle is not yet fulfilled on all stages of the implementation. In order to make these stages more efficient, they would have to be implemented on a specialized microprocessor or FPGA, or directly ported onto SpiNNaker. However, we expect the implementation of the encoder and classifier on SpiNNaker to be difficult and time-consuming. As the filter and sampling function are not yet performing as desired we suggest to only transfer the stages to SpiNNaker if the optimization is finished and the system works as expected.
As long as the stages are executed on different platforms, the communication between SpiNNaker and the host computer poses a major challenge for real-time applications. Compiling the executable files for SpiNNaker as well as simulating and extracting all spikes from the board is time consuming as SpiNNaker uses distributed memory and does not store the spikes in one central storage. One solution might be to develop and use a more efficient and faster interface for SpiNNaker to extract

spikes in real time. However, as the reservoir is supposed to integrate spikes over time due to its recurrent character, it may not be necessary to extract all spikes but only the last ones, which should contain information about previous spikes as well. This could further accelerate the communication between SpiNNaker and the host, and get closer to real-time performance. To incorporate this change into the model would require a different state vector sampling approach, which allows for real time use. In this implementation the state vectors are generated after the sample presentation is finished. Instead, one could calculate the initial state vectors and then update them over time to be able to perform class predictions at any time during sample presentation. A simpler sampling function would also be more easily implementable on neuromorphic hardware.

Signal Source

A major share of information processing in the human nervous system is not performed in the cortex but the cerebellum and the spinal cord. In particular, nonlinear muscle control is influenced by the feedback loops within the peripheral nervous system. By using EEG data in BCI applications this information is lost as we record only cortical activity. A potential way to include this information in classification models is adding another signal source to the BCI such as EMG. Hybrid BCI combine cortical neural information from EEG and peripheral mechanisms from EMG to achieve higher classification performance. However, in rehabilitation applications the combination of EMG and FES is not recommended as the FES would contaminate the EMG measurements.
The low spatial resolution and low SNR of EEG measurements limit the number and complexity of distinguishable classes, which in turn reduces the intuitiveness of the control paradigm. For example, at this time in robotic or prosthetic control scenarios patients have to think about left or right hand movements to respectively flex or extend the elbow. To achieve a natural and intuitive control it is necessary to be able to identify and differentiate between complex and specific movements. As studies in ECoG BCI research have shown, a high spatial resolution in recording is necessary to be able to decode more specific movements such as arm flexion/extension [28]. Therefore, in order to improve EEG BCIs an improvement in EEG measurement techniques is necessary. A reduction of noise and a higher spatial resolution could also increase the use of denser neural networks in models such as NeuCube and potentially improve classification of complex movements.

SNN Reservoir

Our tests have shown that the setup of the reservoir is the crucial step for NeuCube to achieve good performance. There are different ways to create the model and to connect neurons. In addition to the connectivity rules applied in this work, there are more constraints and connection probability equations that could potentially be applied [30]. While it would be helpful to understand the effect of different connectomes

on overall performance, it is very difficult to quantify due to the interdependence of parameters. In this work we applied plausible rules, such as the prohibition of self-connections in the reservoir, while keeping complexity to a minimum. In future work we suggest to build reservoirs that mirror the structure of the human brain in more detail, for example by inhibiting connections between left and right hemisphere, and using as many electrodes as possible to consider all available information in the system. We predict results that emerge from such a system would allow more insights into the function of the human brain. A digital model of the human brain with high plausibility is developed in the Virtual Brain (`http://www.thevirtualbrain.org`).

Section 2.3 explains how ERS can be detected within the reservoir. Nonetheless, a desynchronization of input spike trains can not be explained with the current model. One possible approach to extend the capability of the reservoir could be to add artificial inputs with continuous, periodic firing behaviours to the network. Combining these inherently firing neurons with inhibitory synapses would extend the behavioural range of the reservoir to recognize desynchronization in input spikes. This is particularly interesting for MI applications as ERD is an important feature for movement encoding in the brain [65]. However, adding such neurons to the reservoir would require giving them a spatial location. As these locations have no biological meaning, the additional neurons might interfere with the spatial coding within NeuCube.

Output Classification

One feature of deSNN, which is described in the literature is the merging of similar state vectors/classification neurons. We did not implement this feature as we expect it to counteract the idea of KNN learning for $k > 1$.
In general, the plausibility of the implemented SDSP learning equations should be reconsidered as they do not reflect the idea of SDSP in detail. For now, the synaptic plasticity does not depend on the membrane potential of the post-synaptic neuron. We simply assume that, as soon as there has been one first spike arriving at the respective neuron, SDSP is enabled. As the suggested rule does not include spike train characteristics such as local firing rates or pauses between spike bursts, the implementation can only interpret the reservoir behavior in very basic terms, which are the time order of first spikes and the total spike count. This does not appear to be representative as it is easy to come up with two spike trains that have the same first spike time and the same spike count but a totally different overall distribution. Although the design of an appropriate sampling function is extremely important, it is hard to develop as the discriminative spike train characteristics that must be reflected in the state vectors remain unknown.

Chapter 5

Conclusion

This work presents a self-contained, detailed description and publicly accessible implementation of the NeuCube algorithm for decoding of EEG signals using SpiNNaker neuromorphic hardware. The implementation was designed for the classification of MI data to be used in rehabilitation applications. It comprises three stages: an encoding stage with three different encoding methods, which converts EEG signals into spike trains; a three dimensional brain-shaped recurrent SNN reservoir, which filters spatio-temporal correlations from the data; and an output classifier, which contains a deSNN and a KNN classifier that is trained on class-specific spiking behavior and predicts class labels for unknown samples. All stages have been conceptually and functionally validated. In a final performance test the implementation was applied on 2-class MI EEG data provided by the Graz Data Set B resulting in a maximum mean accuracy of 65% and an individual peak accuracy of 85%. However, the obtained measurements are still inferior to state-of-the-art results using traditional methods and other NeuCube implementations. This is most probably due to a poor separation in the reservoir and a sampling function that does not properly reflect reservoir spike dynamics.
We believe that our work will be valuable for the BCI community as it discloses the internal characteristics of a NeuCube reservoir, which have not been divulged in previous publications in detail, and provides an implementation that is easy to use and portable. This is of utmost importance for fellow researchers to work with the model and to improve system calibration. We believe that the public sharing of information is the fastest way to advance research.

In order to build a NeuCube-based BCI for rehabilitation purposes the system reliability in terms of correct classifications must be increased and real-time communication and execution must be enabled. Therefore, the following steps are suggested:

1. The NeuCube implementation of this work on SpiNNaker shall be used to further test the influence of parameters on the overall system performance. The improvement of the filter function of the SNN reservoir to extract discriminative signal features is especially important as it is the crucial feature

and benefit of NeuCube over traditional classifiers. Interesting aspects to be analyzed are: the size and shape of the SNN reservoir; the spatial distribution and number of input spike trains/electrodes and the mapping of neurons according to different brain atlases; the connectivity within the reservoir and its interdependence with the synapse weight distributions; and the influence of STDP learning on reservoir stability. Furthermore, the state vector sampling function should be modified to better reflect characteristic reservoir spike dynamics. The introduced corrected separation metric can be used to quantify filtering and sampling performance. Nonetheless, a more sophisticated visualizer should be employed to make internal reservoir processes tangible. SpiNNaker is a suitable research platform at this stage as it combines the energy-efficiency and multi-core architecture of neuromorphic hardware with the programmability of a personal computer.

2. With the gained understanding for the influence and interaction of different parameters a mechanism should be developed to tune networks automatically towards provided input data, as manual parameter tuning is time-consuming and poses a major hurdle when new data sets are classified.

3. After the SNN filter and the state vector sampling are understood and tuned, the encoder and classifier stages should be implemented on SpiNNaker to enable real-time communication between the stages and enable in-field testing while sustaining maximum programmability to optimize the parameters for real-life applications.

4. The validated and optimized NeuCube model should be transferred to TrueNorth in order to take advantage of its minimum power consumption to build a commercially usable BCI classifier.

As soon as the network is thoroughly tuned, we expect it to successfully classify multi-class EEG data, which is necessary for complex and intuitive prostheses or robot control. As the implementation can be easily applied on different input data and different neuron structures, we expect the model to be used in varying fields of application.

Referring to the ethical aspects that were presented in the beginning of this work, if all our suggestions for future research were executed and all our predictions for resulting performance came true, we would be a step closer to understanding and interfacing the human brain. Although our primary fields of application for BCI systems such as motor rehabilitation are peaceful and meant to help people, we need to start thinking seriously about the ethical implications of our work. For now, the presented future scenarios are mostly speculation as we still know little about the human brain, and maybe we will never understand the brain in its entirety. As the neuro-scientist Moran Cerf said: ‘If the human brain were so simple that

we could understand it, we would be so simple that we couldn't.'. Nevertheless, only by fostering an intense exchange between and among researchers, politicians, educators, and legislative representatives, can a sustainable and friendly path for the application of BCI technologies be guaranteed.

List of Figures

Bibliography

[1] F. Akopyan, J. Sawada, A. Cassidy, R. Alvarez-Icaza, J. Arthur, P. Merolla, N. Imam, Y. Nakamura, P. Datta, G.-J. Nam, et al. Truenorth: Design and tool flow of a 65 mw 1 million neuron programmable neurosynaptic chip. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 34(10):1537–1557, 2015.

[2] F. B. Alvi, R. Pears, and N. Kasabov. An evolving spatio-temporal approach for gender and age group classification with spiking neural networks. *Evolving Systems*, pages 1–12, 2017.

[3] B. B. Andersen, L. Korbo, and B. Pakkenberg. A quantitative study of the human cerebellum with unbiased stereological techniques. *The Journal of Comparative Neurology*, 326(4):549–560, 1992.

[4] K. K. Ang, Z. Y. Chin, H. Zhang, and C. Guan. Filter bank common spatial pattern (fbcsp) in brain-computer interface. In *Neural Networks, 2008. IJCNN 2008.(IEEE World Congress on Computational Intelligence). IEEE International Joint Conference on*, pages 2390–2397. IEEE, 2008.

[5] K. K. Ang, C. Guan, K. S. G. Chua, B. T. Ang, C. Kuah, C. Wang, K. S. Phua, Z. Y. Chin, and H. Zhang. Clinical study of neurorehabilitation in stroke using eeg-based motor imagery brain-computer interface with robotic feedback. In *Engineering in Medicine and Biology Society (EMBC), 2010 Annual International Conference of the IEEE*, pages 5549–5552. IEEE, 2010.

[6] N. Birbaumer, T. Elbert, A. G. Canavan, and B. Rockstroh. Slow potentials of the cerebral cortex and behavior. *Physiological reviews*, 70(1):1–41, 1990.

[7] J. M. Brader, W. Senn, and S. Fusi. Learning real-world stimuli in a neural network with spike-driven synaptic dynamics. *Neural computation*, 19(11):2881–2912, 2007.

[8] D. V. Buonomano and W. Maass. State-dependent computations: spatiotemporal processing in cortical networks. *Nature Reviews Neuroscience*, 10(2):113, 2009.

[9] N. Caporale and Y. Dan. Spike timing–dependent plasticity: a hebbian learning rule. *Annu. Rev. Neurosci.*, 31:25–46, 2008.

[10] Y. Chen, J. Hu, N. Kasabov, Z. Hou, and L. Cheng. Neucuberehab: A pilot study for eeg classification in rehabilitation practice based on spiking neural networks. In *International Conference on Neural Information Processing*, pages 70–77. Springer, 2013.

[11] D. Coyle, J. Garcia, A. R. Satti, and T. M. McGinnity. Eeg-based continuous control of a game using a 3 channel motor imagery bci: Bci game. In *Computational Intelligence, Cognitive Algorithms, Mind, and Brain (CCMB), 2011 IEEE Symposium on*, pages 1–7. IEEE, 2011.

[12] A. P. Davison, D. Brüderle, J. M. Eppler, J. Kremkow, E. Muller, D. Pecevski, L. Perrinet, and P. Yger. Pynn: a common interface for neuronal network simulators. *Frontiers in neuroinformatics*, 2:11, 2009.

[13] H. De Garis, N. E. Nawa, M. Hough, and M. Korkin. Evolving an optimal de/convolution function for the neural net modules of atr's artificial brain project. In *Neural Networks, 1999. IJCNN'99. International Joint Conference on*, volume 1, pages 438–443. IEEE, 1999.

[14] C. Denk, F. Llobet-Blandino, F. Galluppi, L. A. Plana, S. Furber, and J. Conradt. Real-time interface board for closed-loop robotic tasks on the spinnaker neural computing system. In *International Conference on Artificial Neural Networks*, pages 467–474. Springer, 2013.

[15] J. Dethier, P. Nuyujukian, C. Eliasmith, T. C. Stewart, S. A. Elasaad, K. V. Shenoy, and K. A. Boahen. A brain-machine interface operating with a real-time spiking neural network control algorithm. In J. Shawe-Taylor, R. S. Zemel, P. L. Bartlett, F. Pereira, and K. Q. Weinberger, editors, *Advances in Neural Information Processing Systems 24*, pages 2213–2221. Curran Associates, Inc., 2011.

[16] M. G. Doborjeh, E. Capecci, and N. Kasabov. Classification and segmentation of fmri spatio-temporal brain data with a neucube evolving spiking neural network model. In *Evolving and Autonomous Learning Systems (EALS), 2014 IEEE Symposium on*, pages 73–80. IEEE, 2014.

[17] S. N. Flesher, J. L. Collinger, S. T. Foldes, J. M. Weiss, J. E. Downey, E. C. Tyler-Kabara, S. J. Bensmaia, A. B. Schwartz, M. L. Boninger, and R. A. Gaunt. Intracortical microstimulation of human somatosensory cortex. *Science translational medicine*, 8(361):361ra141–361ra141, 2016.

[18] K. Fox. Experience-dependent plasticity mechanisms for neural rehabilitation in somatosensory cortex. *Philosophical Transactions of the Royal Society of London B: Biological Sciences*, 364(1515):369–381, 2009.

[19] S. Furber. Large-scale neuromorphic computing systems. *Journal of neural engineering*, 13(5):051001, 2016.

[20] S. B. Furber, F. Galluppi, S. Temple, and L. A. Plana. The spinnaker project. *Proceedings of the IEEE*, 102(5):652–665, 2014.

[21] S. Fusi, M. Annunziato, D. Badoni, A. Salamon, and D. J. Amit. Spike-driven synaptic plasticity: theory, simulation, vlsi implementation. *Neural computation*, 12(10):2227–2258, 2000.

[22] J. P. Gallivan and J. C. Culham. Neural coding within human brain areas involved in actions. *Current opinion in neurobiology*, 33:141–149, 2015.

[23] W. Gerstner and W. M. Kistler. *Spiking neuron models: Single neurons, populations, plasticity*. Cambridge university press, 2002.

[24] S. Ghosh-Dastidar and H. Adeli. Spiking neural networks. *International journal of neural systems*, 19(04):295–308, 2009.

[25] E. Goodman and D. Ventura. Spatiotemporal pattern recognition via liquid state machines. In *Neural Networks, 2006. IJCNN'06. International Joint Conference on*, pages 3848–3853. IEEE, 2006.

[26] J. Grau Moya. Integration of the information in complex neural networks with noise. 2011.

[27] D. O. Hebb. Distinctive features of learning in the higher animal. *Brain mechanisms and learning*, pages 37–46, 1961.

[28] L. R. Hochberg, D. Bacher, B. Jarosiewicz, N. Y. Masse, J. D. Simeral, J. Vogel, S. Haddadin, J. Liu, S. S. Cash, P. van der Smagt, et al. Reach and grasp by people with tetraplegia using a neurally controlled robotic arm. *Nature*, 485(7398):372, 2012.

[29] R. W. Homan, J. Herman, and P. Purdy. Cerebral location of international 10–20 system electrode placement. *Electroencephalography and clinical neurophysiology*, 66(4):376–382, 1987.

[30] J. Hu, Z.-G. Hou, Y.-X. Chen, N. Kasabov, and N. Scott. Eeg-based classification of upper-limb adl using snn for active robotic rehabilitation. In *Biomedical Robotics and Biomechatronics (2014 5th IEEE RAS & EMBS International Conference on*, pages 409–414. IEEE, 2014.

[31] P. J. Ifft, S. Shokur, Z. Li, M. A. Lebedev, and M. A. Nicolelis. A brain-machine interface enables bimanual arm movements in monkeys. *Science translational medicine*, 5(210):210ra154–210ra154, 2013.

[32] E. M. Izhikevich. Simple model of spiking neurons. *IEEE Transactions on neural networks*, 14(6):1569–1572, 2003.

[33] E. M. Izhikevich. Which model to use for cortical spiking neurons? *IEEE transactions on neural networks*, 15(5):1063–1070, 2004.

[34] X. Jin, A. Rast, F. Galluppi, S. Davies, and S. Furber. Implementing spike-timing-dependent plasticity on spinnaker neuromorphic hardware. In *Neural Networks (IJCNN), The 2010 International Joint Conference on*, pages 1–8. IEEE, 2010.

[35] N. Kasabov. Neucube evospike architecture for spatio-temporal modelling and pattern recognition of brain signals. In *IAPR Workshop on Artificial Neural Networks in Pattern Recognition*, pages 225–243. Springer, 2012.

[36] N. Kasabov and E. Capecci. Spiking neural network methodology for modelling, classification and understanding of eeg spatio-temporal data measuring cognitive processes. *Information Sciences*, 294:565–575, 2015.

[37] N. Kasabov, K. Dhoble, N. Nuntalid, and G. Indiveri. Dynamic evolving spiking neural networks for on-line spatio-and spectro-temporal pattern recognition. *Neural Networks*, 41:188–201, 2013.

[38] N. Kasabov, N. M. Scott, E. Tu, S. Marks, N. Sengupta, E. Capecci, M. Othman, M. G. Doborjeh, N. Murli, R. Hartono, et al. Evolving spatio-temporal data machines based on the neucube neuromorphic framework: design methodology and selected applications. *Neural Networks*, 78:1–14, 2016.

[39] N. K. Kasabov. Neucube: A spiking neural network architecture for mapping, learning and understanding of spatio-temporal brain data. *Neural Networks*, 52:62–76, 2014.

[40] K. Kiguchi, T. D. Lalitharatne, and Y. Hayashi. Estimation of forearm supination/pronation motion based on eeg signals to control an artificial arm. *Journal of Advanced Mechanical Design, Systems, and Manufacturing*, 7(1):74–81, 2013.

[41] J. A. Kleim and T. A. Jones. Principles of experience-dependent neural plasticity: implications for rehabilitation after brain damage. *Journal of speech, language, and hearing research*, 51(1):S225–S239, 2008.

[42] G. H. Klem, H. O. LuÈders, H. Jasper, C. Elger, et al. The ten-twenty electrode system of the international federation. *Electroencephalogr Clin Neurophysiol*, 52(3):3–6, 1999.

[43] L. Koessler, L. Maillard, A. Benhadid, J. P. Vignal, J. Felblinger, H. Vespignani, and M. Braun. Automated cortical projection of eeg sensors: anatomical correlation via the international 10–10 system. *Neuroimage*, 46(1):64–72, 2009.

[44] K.-H. Kong, K. S. Chua, and J. Lee. Recovery of upper limb dexterity in patients more than 1 year after stroke: frequency, clinical correlates and predictors. *NeuroRehabilitation*, 28(2):105–111, 2011.

[45] J. Lancaster, L. Rainey, J. Summerlin, C. Freitas, P. Fox, A. Evans, A. Toga, and J. Mazziotta. Automated labeling of the human brain: a preliminary report on the development and evaluation of a forward-transform method. *Human brain mapping*, 5(4):238, 1997.

[46] J. L. Lancaster, M. G. Woldorff, L. M. Parsons, M. Liotti, C. S. Freitas, L. Rainey, P. V. Kochunov, D. Nickerson, S. A. Mikiten, and P. T. Fox. Automated talairach atlas labels for functional brain mapping. *Human brain mapping*, 10(3):120–131, 2000.

[47] M. A. Lebedev and M. A. Nicolelis. Brain-machine interfaces: From basic science to neuroprostheses and neurorehabilitation. *Physiological reviews*, 97(2):767–837, 2017.

[48] S.-G. Lee, A. Neiman, and S. Kim. Coherence resonance in a hodgkin-huxley neuron. *Physical Review E*, 57(3):3292, 1998.

[49] R. Leeb, C. Brunner, G. Müller-Putz, A. Schlögl, and G. Pfurtscheller. Bci competition 2008–graz data set b. *Graz University of Technology, Austria*, 2008.

[50] J. Li, R. Zheng, Y. Zhang, and J. Yao. ihandrehab: An interactive hand exoskeleton for active and passive rehabilitation. In *Rehabilitation Robotics (ICORR), 2011 IEEE International Conference on*, pages 1–6. IEEE, 2011.

[51] Y.-H. Liu and X.-J. Wang. Spike-frequency adaptation of a generalized leaky integrate-and-fire model neuron. *Journal of computational neuroscience*, 10(1):25–45, 2001.

[52] F. Lotte, M. Congedo, A. Lécuyer, F. Lamarche, and B. Arnaldi. A review of classification algorithms for eeg-based brain–computer interfaces. *Journal of neural engineering*, 4(2):R1, 2007.

[53] W. Maass. Networks of spiking neurons: the third generation of neural network models. *Neural networks*, 10(9):1659–1671, 1997.

[54] W. Maass, T. Natschläger, and H. Markram. Real-time computing without stable states: A new framework for neural computation based on perturbations. *Neural computation*, 14(11):2531–2560, 2002.

[55] A. Nakamura, T. Yamada, A. Goto, T. Kato, K. Ito, Y. Abe, T. Kachi, and R. Kakigi. Somatosensory homunculus as drawn by meg. *Neuroimage*, 7(4):377–386, 1998.

[56] T. Natschläger and B. Ruf. Spatial and temporal pattern analysis via spiking neurons. *Network: Computation in Neural Systems*, 9(3):319–332, 1998.

[57] E. Niedermeyer and F. L. da Silva. *Electroencephalography: basic principles, clinical applications, and related fields*. Lippincott Williams & Wilkins, 2005.

[58] D. Norton and D. Ventura. Preparing more effective liquid state machines using hebbian learning. In *Neural Networks, 2006. IJCNN'06. International Joint Conference on*, pages 4243–4248. IEEE, 2006.

[59] N. Nuntalid, K. Dhoble, and N. Kasabov. Eeg classification with bsa spike encoding algorithm and evolving probabilistic spiking neural network. In *International Conference on Neural Information Processing*, pages 451–460. Springer, 2011.

[60] S. J. Page, P. Levine, S. Sisto, and M. V. Johnston. A randomized efficacy and feasibility study of imagery in acute stroke. *Clinical rehabilitation*, 15(3):233–240, 2001.

[61] E. Painkras, L. A. Plana, J. Garside, S. Temple, F. Galluppi, C. Patterson, D. R. Lester, A. D. Brown, and S. B. Furber. Spinnaker: A 1-w 18-core system-on-chip for massively-parallel neural network simulation. *IEEE Journal of Solid-State Circuits*, 48(8):1943–1953, 2013.

[62] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830, 2011.

[63] L. Peng, Z.-G. Hou, N. Kasabov, G.-B. Bian, L. Vladareanu, and H. Yu. Feasibility of neucube spiking neural network architecture for emg pattern recognition. In *Advanced Mechatronic Systems (ICAMechS), 2015 International Conference on*, pages 365–369. IEEE, 2015.

[64] G. Pfurtscheller, C. Brunner, A. Schlögl, and F. L. Da Silva. Mu rhythm (de) synchronization and eeg single-trial classification of different motor imagery tasks. *Neuroimage*, 31(1):153–159, 2006.

[65] G. Pfurtscheller and F. L. Da Silva. Event-related eeg/meg synchronization and desynchronization: basic principles. *Clinical neurophysiology*, 110(11):1842–1857, 1999.

[66] G. Pfurtscheller and C. Neuper. Motor imagery activates primary sensorimotor area in humans. *Neuroscience letters*, 239(2-3):65–68, 1997.

[67] D. Purves, G. J. Augustine, D. Fitzpatrick, L. C. Katz, A.-S. LaMantia, J. O. McNamara, and S. M. Williams. *Neuroscience*. Sinauer Associates, 2001.

[68] K. Rajan and L. Abbott. Eigenvalue spectra of random matrices for neural networks. *Physical review letters*, 97(18):188104, 2006.

[69] S. Rajangam, P.-H. Tseng, A. Yin, G. Lehew, D. Schwarz, M. A. Lebedev, and M. A. Nicolelis. Wireless cortical brain-machine interface for whole-body navigation in primates. *Scientific reports*, 6:22170, 2016.

[70] C. Richter, S. Jentzsch, R. Hostettler, J. A. Garrido, E. Ros, A. Knoll, F. Rohrbein, P. van der Smagt, and J. Conradt. Musculoskeletal robots: scalability in neural control. *IEEE Robotics & Automation Magazine*, 23(4):128–137, 2016.

[71] F. Rieke. *Spikes: exploring the neural code*. MIT press, 1999.

[72] G. Schalk and J. Mellinger. *A practical guide to brain–computer interfacing with BCI2000: General-purpose software for brain-computer interface research, data acquisition, stimulus presentation, and brain monitoring*. Springer Science & Business Media, 2010.

[73] A. Schlögl, F. Lee, H. Bischof, and G. Pfurtscheller. Characterization of four-class motor imagery eeg data for the bci-competition 2005. *Journal of neural engineering*, 2(4):L14, 2005.

[74] B. Schrauwen and J. Van Campenhout. Bsa, a fast and accurate spike train encoding scheme. In *Neural Networks, 2003. Proceedings of the International Joint Conference on*, volume 4, pages 2825–2830. IEEE, 2003.

[75] N. Scott, N. Kasabov, and G. Indiveri. Neucube neuromorphic framework for spatio-temporal brain data and its python implementation. In *International Conference on Neural Information Processing*, pages 78–84. Springer, 2013.

[76] N. M. Scott. *Evolving Spiking Neural Networks for Spatio-and Spectro-Temporal Data Analysis: Models, Implementations, Applications*. PhD thesis, Auckland University of Technology, 2015.

[77] S. Silbernagl. *Taschenatlas der physiologie*. Georg Thieme Verlag, 2007.

[78] S. Song, K. D. Miller, and L. F. Abbott. Competitive hebbian learning through spike-timing-dependent synaptic plasticity. *Nature neuroscience*, 3(9):919, 2000.

[79] M. Spitzer. *The mind within the net: Models of learning, thinking, and acting*. Mit Press, 1999.

[80] A. J. Szameitat, S. Shen, and A. Sterr. Motor imagery of complex everyday movements. an fmri study. *Neuroimage*, 34(2):702–713, 2007.

[81] Z. Tayeb, E. Erçelik, and J. Conradt. Decoding of motor imagery movements from eeg signals using spinnaker neuromorphic hardware. In *Neural Engineering (NER), 2017 8th International IEEE/EMBS Conference on*, pages 263–266. IEEE, 2017.

[82] D. Taylor, N. Scott, N. Kasabov, E. Capecci, E. Tu, N. Saywell, Y. Chen, J. Hu, and Z.-G. Hou. Feasibility of neucube snn architecture for detecting motor execution and motor intention for use in bciapplications. In *Neural Networks (IJCNN), 2014 International Joint Conference on*, pages 3221–3225. IEEE, 2014.

[83] E. Tu, N. Kasabov, M. Othman, Y. Li, S. Worner, J. Yang, and Z. Jia. Neucube (st) for spatio-temporal data predictive modelling with a case study on ecological data. In *Neural Networks (IJCNN), 2014 International Joint Conference on*, pages 638–645. IEEE, 2014.

[84] E. Tu, N. Kasabov, and J. Yang. Mapping temporal variables into the neucube for improved pattern recognition, predictive modeling, and understanding of stream data. *IEEE transactions on neural networks and learning systems*, 28(6):1305–1317, 2017.

[85] M. Velliste, S. Perel, M. C. Spalding, A. S. Whitford, and A. B. Schwartz. Cortical control of a prosthetic arm for self-feeding. *Nature*, 453(7198):1098, 2008.

[86] D. Verstraeten, B. Schrauwen, M. d'Haene, and D. Stroobandt. An experimental unification of reservoir computing methods. *Neural networks*, 20(3):391–403, 2007.

[87] J. R. Wolpaw, N. Birbaumer, D. J. McFarland, G. Pfurtscheller, and T. M. Vaughan. Brain–computer interfaces for communication and control. *Clinical neurophysiology*, 113(6):767–791, 2002.

[88] J. R. Wolpaw and D. J. McFarland. Control of a two-dimensional movement signal by a noninvasive brain-computer interface in humans. *Proceedings of the National Academy of Sciences of the United States of America*, 101(51):17849–17854, 2004.

[89] W. Wu, X. Gao, B. Hong, and S. Gao. Classifying single-trial eeg during motor imagery by iterative spatio-spectral patterns learning (isspl). *IEEE Transactions on Biomedical Engineering*, 55(6):1733–1743, 2008.

[90] H. Yang, S. Sakhavi, K. K. Ang, and C. Guan. On the use of convolutional neural networks and augmented csp features for multi-class motor imagery of eeg signals classification. In *Engineering in Medicine and Biology Society (EMBC), 2015 37th Annual International Conference of the IEEE*, pages 2620–2623. IEEE, 2015.

[91] X. Yong and C. Menon. Eeg classification of different imaginary movements within the same limb. *PloS one*, 10(4):e0121896, 2015.

License