Doctoral Dissertation
Academic Year 2019

Neuromorphic Networks for Prediction Applications

Keio University
Graduate School of Media Design

Cedric Caremel

A Doctoral Dissertation
submitted to Keio University Graduate School of Media Design
in partial fulfillment of the requirements for the degree of
Ph.D. in Media Design

Cedric Caremel

Dissertation Advisory Committee:

Professor Kai Kunze	(Supervisor)
Professor Kouta Minamizawa	(Co-Supervisor)
Professor Akira Kato	(Co-Supervisor)

Doctoral Dissertation Review Committee:

Professor Kouta Minamizawa	(Chair)
Professor Akira Kato	(Member)
Professor Keiko Okawa	(Member)
Professor Yoshihiro Kawahara	(Member, The University of Tokyo)

Abstract of Doctoral Dissertation of Academic Year 2019

Neuromorphic Networks for Prediction Applications

Category: Science / Engineering

Summary

Artificial neural networks represent a powerful class of machine learning algorithms, well suited for any type of technical application: from engineering applications to scientific computing. However, artificial neural networks designs are increasingly deviating from the functional architecture of brain circuits they originated from, focusing on very sophisticated yet very segmented implementations, at the opposite end of a multipurpose intelligence. Instead, current advances in neuroscience converge toward models of the encoding of sensory signals as well as rewards, learning and behavioral dynamics, indicating that, in the near future, tools such as artificial neural networks should be capable of providing better insights about the brain architecture. Here, the main objective is to provide a few key concepts and methods to leverage the power of predictive, biologically plausible, neural networks.

The expert implementation of task-specific neural networks versus the practical needs of innovators, engineers, physicists or biologists to analyze their models regardless of the complexity or type of data they are working with, also emphasize the critical importance of designing general-purpose algorithms, and as such, bio-inspired artificial neural networks represent a viable solution. Prediction can be used to control complex hardware, validate experiments or create innovative interactions. Biological plausibility brings in flexibility and adaptability to different situations and desired outputs, thus facilitating data processing for experts and non-experts alike.

Keywords:

Artificial Neural Networks, Brain Circuits, Human-Machine Interfaces, Neurocomputing.

Keio University Graduate School of Media Design

Cedric Caremel

Contents

List of Figures

List of Tables

Acknowledgements

I would like to thank the Professors at Keio University for their guidance, and in particular my main supervisor, Prof. Kai Kunze, for his constant support, unique approach and invaluable insights along the course of my doctoral program. Many thanks to Prof. Kato and Prof. Minamizawa as well, Dr. Benucci at RIKEN CBS (Center for Brain Science), Prof. Yoshihiro Kawahara at the University of Tokyo (Graduate School of Engineering, Department of Electrical Engineering and Information Systems), Kinya Tagawa, Hisato Ogata and Kotaro Watanabe, partners/directors at Takram Design Engineering, for the many inspiring discussions and advises. A very special thanks to my colleagues, friends and family, for their indefectible support along the years.

The work at Keio University is partly supported by JST Presto, Grant No: JPMJPR16D4.

All plots, images and graphs are my own, unless stated otherwise. In case of any inquiries, please contact me: cedric@keio.jp

Chapter 1
Introduction

1.1. Definition of Artificial Intelligence

The idea of an 'electronic brain' appears as early as 1943 in the seminal paper "A Logical Calculus of Ideas Immanent in Nervous Activity" (McCulloch and Pitts 1943), by Warren McCulloch, an established American neurophysiologist, and Walter Pitts, his young assistant, a self-taught logician eager to follow the path of the British philosopher Bertrand Russel (Russel and Whitehead 1910-13), at the university of Chicago. From 1942, Pitts and MacCulloch worked extensively on the nervous system and together, they laid the foundations of what would be later called the McCulloch–Pitts neuron model:

$$y_k = \varphi(\sum_{i=1}^{n} w_{ji} x_i)$$

where y_k represents the output of an artificial neuron, as the function of a sum of its weighted inputs. This groundbreaking model of the mathematical expression of a biological neuron led, a few decades later, to the development of a flourishing new field, computational neurosciences, first in 1969, with Minsky (Perceptron (Minsky M. 1969), 1969), showing that the perceptron architecture as intended by Rosenblatt (Rosenblatt 1958) could not solve the XOR problem (Minsky M. 1969); then in 1987, when the Lisp machine, specifically designed for a new generation of high-level computer language, failed to impose its paradigm to the market. The resurgence came in the early 2000, when the exponential development of powerful workstations and additional resources (GPUs) at lower manufacturing costs, in a newly globalized world, favored the conditions for some spectacular achievements in machine learning, led by a few private (Facebook, Google, IBM, Tencent, NVidia) and public initiatives around the world.

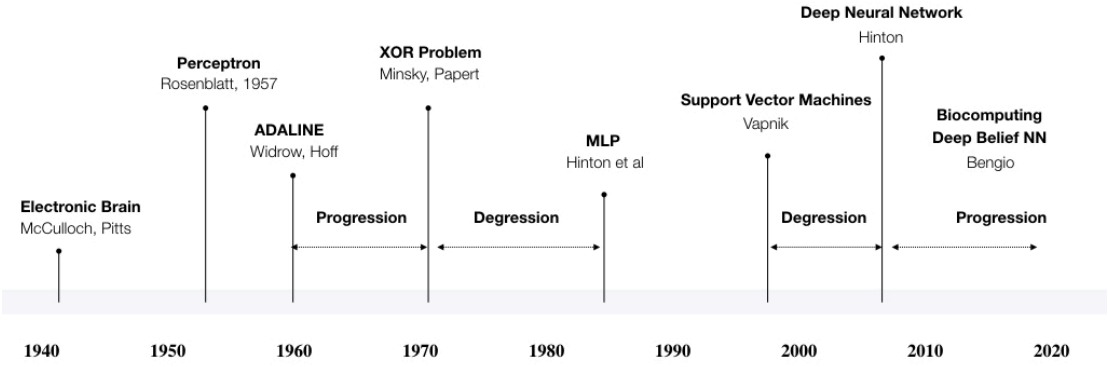

Figure 1.1 Machine learning milestones.

The concept of artificial intelligence slowly gained acceptance by analysts and defense experts, after accelerating its pace in the last decades (IBM DeepBlue (Crichton et al. 2017)), and reaching an overwhelming visibility in the literature, in the press and in movies. Now that more complex games can be solved (AlphaGo (Silver et al. 2017)), artificial intelligence is under scrutiny virtually everywhere, a subject of both fascinations and tensions. As we have already briefly seen, the term "artificial intelligence" encompasses a myriad of sub-fields and various concepts across history, borrowing from physiology, computational sciences, physics, mathematics, philosophy, fiction. In that view, artificial intelligence represents a collective dream for our modern societies. In a strict sense though, it is an academic field that requires rigor and method and, as we will see, it has yet to deliver all its promises. The question that arises then, is for the immediate future: will artificial intelligence develop into fully intelligent machines that can help us solve seemingly intractable problems?

1.2. Machine Learning

To answer this question, I will explore today's available technology. After reviewing some important methods in the field, along with the state-of-the-art, I will expose the theoretical foundations of this thesis.

As Figure 1.3 shows, the term "machine learning" has turned to be an exponential trend in the literature since 1980. It denotes a practical and effective subset of artificial intelligence that can be characterized by the actual algorithmic implementation of the core principle of any intelligent system: the ability to learn and discover by itself. There is an important distinction operating there: while artificial intelligence regroups a variety of concepts, stretching from computer science to ontological questions on the nature of the human mind, machine learning should be considered as a programmatic attempt to answer those questions rationally and practically (Bishop 2007).

In 1970, John Conway's Game of Life (Gardner 1970), after John Von Neumann and Stanislaw Ulam's initial discovery, marks one of the very first effort to model life as a self-replicating program that evolves based on initial inputs and a set of rules. This concept known as "cellular automaton" laid the groundwork

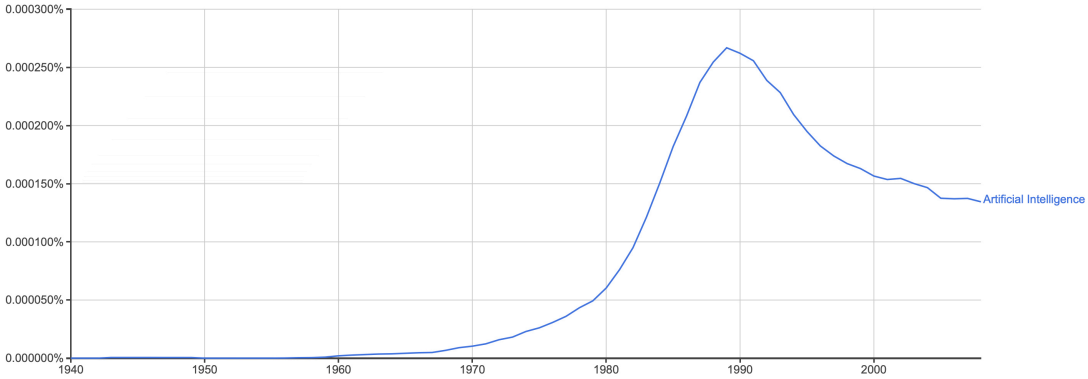

Figure 1.2 Google N-GRAM of the term "artificial intelligence" in English literature since 1940. In computational linguistics, the N-GRAM refers to the number of occurrences of a word recorded over a given period of time.

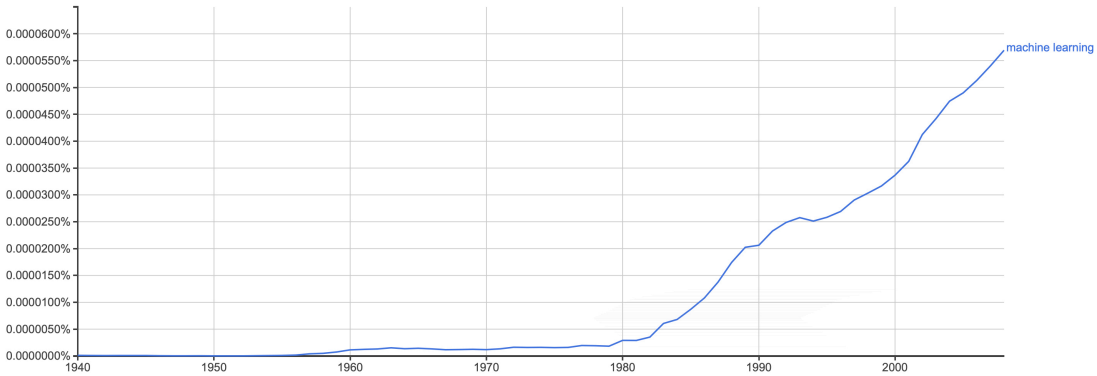

Figure 1.3 Google N-GRAM of the term "machine learning" in the literature (English language) since 1940.

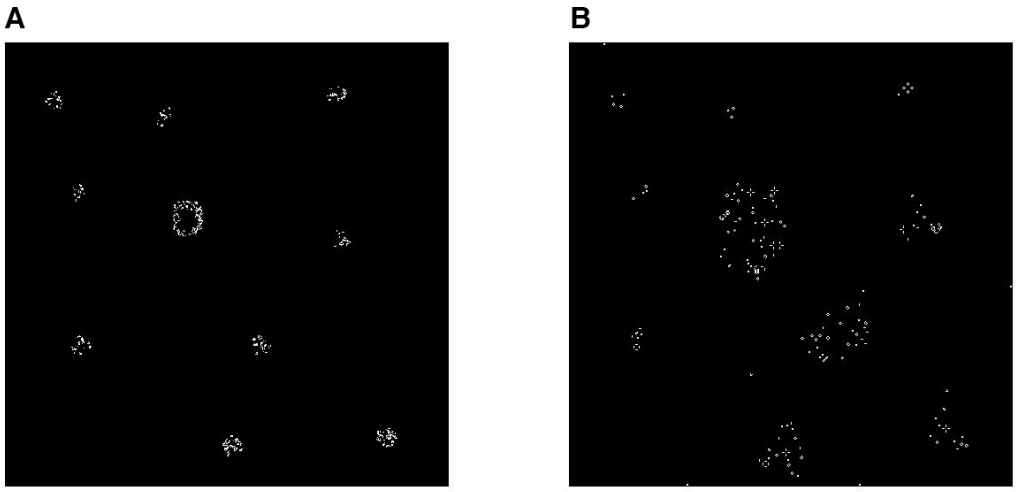

Figure 1.4 A MATLAB implementation of the Game of Life (©Ibraheem, 2010). The initial state (A) is defined by a 500x500 pixel black and white picture with random noise represented by the white pixels. A cell is said to be alive if white, black otherwise. The program computes the state of a cell based on its alive neighbors. After initialization, different types of patterns arise. After 1000 epochs, the final state (B), represents an ensemble of stable patterns that will not evolve any further. Each pattern can be classified and there only exist a finite number of variations. However, the game is undecidable: given (A) and (B), it is not possible to tell whether (B) will ever exist, other than by running the algorithm until it reaches its final state.

for machine learning. The important result this simple game provided is that a system does not preclude complexity to emerge from a set of simple rules; here, the program is undecidable (Figure 1.4). This idea that we, as human beings doted of an intellect, are capable of producing pure, perfect and ideal constructions, represented by mathematical properties, always confronts the harsh reality of an ever-changing, seemingly chaotic world (Strogatz 2000), that also seems to increase in complexity as we increase our understanding. This strange paradox (increasing our knowledge does not appear to simplify the state of the world, but rather, render it harder to apprehend) led us to build tools powerful enough to overcome our own limitations, but simple enough to let us control them with ease. If artificial intelligence can respond to the first part of our need (understanding the world), its interpretability (Sussillo and Barak 2013, Sussillo et al. 2015), or lack thereof, remains: we may be capable of producing models that can solve intractable problems, but we may not fully grasp their internal representations. Indeed, if the theoretical background that underlies the algorithm we are implementing is well-known, the solutions may be non-trivial, since the number of parameters and dimensions explodes as the difficulty of the task increases. In October 2015, AlphaGo, a program that plays Go, beat Lee Sedol, the world best player, with odd-looking (from a human perspective) moves (Silver et al. 2017). This unprecedented feat is, of course, reminiscent of Kasparov's loss against IBM DeepBlue, but should not be mistaken for it: Chess is essentially an entropic system, where the number of successful combinations decreases as the game develops. Playing pieces are removed, allowing the algorithm to brute-force a decision tree. The board game Go follows instead a negative entropy development, where the combinations in play rise exponentially as playing pieces (white stones and black stones) are added to the game. A set of sophisticated statistical models, based on the neural networks introduced earlier, were used in combination to enormous computer resources, in order to train the network on millions of game variations and approximate the optimum policy that would eventually allow the program to beat the world champion. More impressively, AlphaZero, a second iteration of AlphaGo, superseded its predecessor in just under 4 hours, learning by self-play reinforcement learning. Although classic tree search algorithms are still present in the most recent machine learning implementations and can be easily represented

in terms of choice decisions, the idea that an artificial network may surpass the expert knowledge, not only in terms of outcome but also in terms of optimization, is mesmerizing: what did the network learn that the expert didn't? Are those remarkable achievements only a question of processing speed, or could it be that the internal representations in play there, are fundamentally uninterpretable from a human standpoint?

1.3. Artificial Neural Networks: State of the Art

Whether we use them in games, the automotive industry or the health-care sector, artificial neural networks represent a special set of machine learning algorithms and can be classified in three distinct categories:

- Supervised Learning
- Unsupervised Learning
- Reinforcement Learning

Deep Learning, the strongest proponent in machine learning to achieve AGI, regroups under this denomination the three methods stated above, by sharing a common architecture: the addition of a middle layer between the input and the output, a standard implementation (Gallicchio et al. 2018) that can be traced back to the 1990s as a replacement to first-generation classifiers, such as Support Vector Machines (SVN) (Cortes and Vapnik 1995). Deep learning, as a generic method to implement artificial neural networks, can be designed either in feed-forward fashion or recurrently. In feed-forward fashion, the data are transiting through the network in one pass, from the lower to the upper layers, each layer being specialized in a type of data feature extraction, making it an ideal candidate for resource intensive and image processing applications: the hidden layers between the input and output units allow to stack as many kernels as needed to filter images and extract the much needed features to compute, for instance, driving. AlexNet, GoogleLeNet, Inception and more recently VGG are all pretrained convolutional neural networks (CNN) with a deep neural network architecture that have been massively adopted by the automotive industry (Luckow et al. 2016).

Recurrent Neural Networks are another well-known class of deep learning algorithms, where the output is fed back to the network as an input (hence, recurrent). Generally suitable for time-series prediction, Long-Short Term Memory (LSTM) neural networks have long provided good results for natural language-processing applications. However, new approaches, combining both of those or introducing novel ideas inspired by biology, such as echo-state networks and reservoir computing (Gallicchio et al. 2018), promise to unleash the next wave of AI-powered applications, as we will see in the next section.
If deep learning has gained a lot of traction in recent years, it is mostly due to its very active research community, along with groundbreaking, language and vision-based applications, provided, sometimes for free, by some of the world largest companies and R&D centers: self-driving vehicles, face recognition systems, personal "smart" assistants, translators, but also tool-kits, programming languages and frameworks - there is almost no limit to the number of visionary concepts that could involve machine learning. In the automotive industry particularly, this revolution, in an otherwise very conservative and competitive business landscape, proposes for example "full autonomy" in three future steps (level 3-5):

- Level 0: No autonomy.
- Level 1: Drive assist (brake-assist, parking-assist, GPS).
- Level 2: Partial autonomy (automated safety measures, speed-limit, self-parking).
- Level 3: Conditional autonomy (autopilot point-to-point on highways, changing lanes). Current state-of-the art.
- Level 4: Near autonomous (autopilot in a populated, semi-urban environment, safe decision-making based on road signals and dynamic traffic).
- Level 5: Fully autonomous (autopilot in urban environment with maximum safety).

As it appears above, safety is a key variable in leveling autonomy, which could not be achieved without a fully integrated hardware-software development. It is worth noting though, that the convolutional (Nayebi et al. 2018) and recurrent

neural networks (Gallicchio et al. 2017, Salehinejad et al. 2017) tool-kits and libraries currently implemented by machine learning developers (Keras, TensorFlow by Google, PyTorch, Caffe by Facebook) share the same type of initial architecture and philosophy proposed by Pitts-McCulloch, Rosenblatt and others during the past half-century: essentially, a multilayer perceptron, which I will formalize in the next chapter (Related Work).
Resource-demanding, convoluted networks, for instance, rely heavily on two critical points: first, a dedicated system of optical sensors, either lasers (LIDAR) or cameras, to acquire an accurate representation of the environment. Second, an expensive training over a very large amount of data. Because the concept of learning is intrinsically bonded to the concept of training, complex dynamic systems may not be correctly approximated with a limited amount of data.

In that sense, major advances are coming from the medical research (Esteva et al. 2017, Christiansen et al. 2018), where data may be sparse or not publicly available. Transfer learning showed, for example, very promising results in skin cancer detection: one neural net is first trained with a very large amount of data, and then applied to a second-related task with a more restricted amount. In a letter issued in 2017 on Nature, Stanford researchers (Esteva et al. 2017) proved that their CNN's performance was on par with the diagnosis of pro-eminent dermatologists for detecting malignant melanomas. Although the training was applied over a large 129,450 clinical images dataset, it is still far less compared to the pretrained architecture they utilized for their approach, the GoogleNet Inception net, which was trained over 1.28 million images.

1.4. Aim and Objectives

This research focuses on the recent developments in artificial intelligence, more specifically biologically plausible artificial neural networks, to create and envision hardware and software that can be fully integrated to human activity. The aim of this research is not to provide an exhaustive review of all the possible implementations in machine learning, as there would be too many, but rather to select a few relevant ones, belonging to the neural network class, and propose a novel network design framework, inspired by biology and relying on sensory inputs, acquired via

innovative interfaces. The main objectives are the following:

- Presenting a novel method using the cortical circuits of the brain as a prime model for the structural and functional architecture of artificial networks. The main advantage is to propose flexibility and adaptability to different experimental conditions. Its relevancy w.r.t. biological networks is limited by the current knowledge in brain science, but the architecture is agile enough to be trained over large data sets, in a reasonable amount of time.

- Establishing a comparative analysis using core concepts of machine intelligence such as agency, prediction and control, in different applications areas. Such implementations have been first designed with conventional neural networks, to confirm and explore their potential (Caremel et al. 2018, Chernyshov et al. 2018a). In parallel, artificial neural circuits have been designed and tested extensively for a bio-inspired, general-purpose architecture.

- Envisioning the potential of neural data to automate decision-making tasks. A new experiment with neural data recording has be implemented for this purpose. Indeed, if training an artificial neural network on neural signals for cognitive tasks is common practice in brain science (Nayebi et al. 2018), (Sussillo and Barak 2013), (Guerguiev et al. 2017), it has been a forsaken path in HCI, due to the lack of framework and tools. In that sense, a brain-based neural network architecture represents a legitimate and desirable route to offer new perspectives to an ever-growing number of innovators, researchers and engineers interested in predictive simulations and computational models.

1.5. Contributions

This thesis constitutes a major advance in the field of human-computer interfaces by bridging the gap between neuroscience and machine learning.

- In this thesis, “neuromorphic networks”, a novel paradigm, is defined at the intersection of machine learning, HCI and neurosciences. To this day,

the computations occurring in the brain are not fully understood. However, predicting outcomes and making choices based on simulations are at the core of the human intellect, shaping our experiences. On the other end, programming languages and interfaces are widely available, and in recent years, state-of-the-art artificial neural networks have been extremely efficient at modeling a large range of problems. A multitude of techniques and research works have been published and discussed. Among those available, the two well-established methods of Classification and Regression are exposed in this thesis as complementary. While classification is by definition well-suited when the dataset can be labelled, regression, in contrast, is generally used when the underlying function is not well-known. This work proposes to re-unite those methods in a novel "neuromorphic" paradigm, by mapping the known structural and functional connectivity of the brain to an artificial neural network. An important finding was that the brain connectivity could serve as an example to describe the neuromorphic model in a comprehensive, formalized way.

- By carefully developing several applications, this thesis demonstrates how to leverage the predictive power of neuromorphic networks to facilitate system modeling. Whether it is for determining the resting parameters of a complex shape-memory alloy, anticipating the decision-making process involved in trajectory predictions and body motions, or predicting neurosignals associated with a specific task, the techniques that were designed and tested provided better insights on the understanding of the models internal dynamics and representation. Comparative analysis on error and performance led to innovative solutions for refining and improving the models over existing techniques.

- Finally, an experimental setup with EEG recording was designed to envision knowledge transfer and task automation using neural models. Training an artificial neural network on neural data poses several challenges: the learning halts at a local minimum or a plateau, overfitting may occur, weights are not properly distributed. An exploration phase is often needed to derive the correct parameters and validate the network. Based on the current corpus of

knowledge in neuroanatomy and computational neuroscience, a "blueprint" was devised to design a predictor-classifier for a working-memory task. The thesis also represents a detailed guideline and an invitation to build such networks, paving the way for future brain-computer interfaces and neural-based machine learning.

1.6. Overview

Today, neuroscience research is increasingly relying on machine learning models to understand the encoding of sensory signals as well as rewards, learning and behavioral dynamics, indicating that, in the near future, tools such as artificial neural networks should be essential to provide better insights about the brain architecture. Interestingly, the converse did not happen: machine learning engineers are less inclined to use biology as a referral model, while still aiming for general intelligence. This thesis proposes to bridge the gap between the current knowledge of biological models and the state-of-the-art machine learning techniques. In life, we base our choice on predictions, simulations of the future, and we rank the possible solutions to adjust our behaviors, and take actions to the world. Similarly, artificial neural networks can be designed to encode inputs and predict patterns, so as to model a behavior and predict outcomes. Regression and classification should then be used in conjunction, rather than separately, as it is classically done. This novel paradigm is at the center of this thesis, and a new model, 'BioNN' (for 'Biology-inspired Neural Network'), is proposed for several application examples to verify this approach.

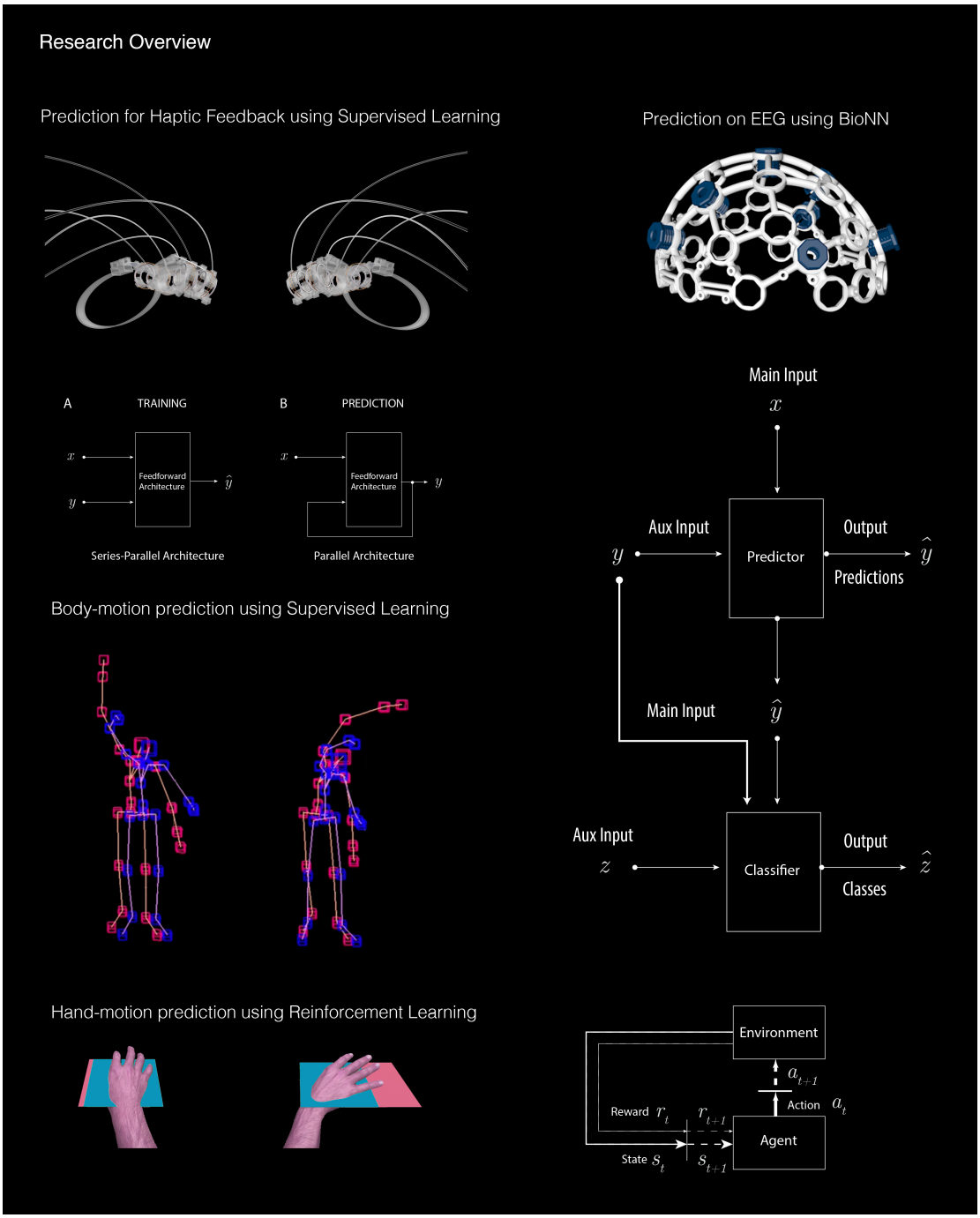

Figure 1.5 Overview of the implemented neural networks models. Left: prediction model for haptic feedback, prediction model for body motion, hand motion prediction in virtual reality. Right: neuromorphic network use case based on EEG recording.

Chapter 2
Related Work

"All models are approximations. Essentially, all models are wrong, but some are useful. However, the approximate nature of the model must always be borne in mind."
— George E. P. Box.

2.1. Hybrid Approaches

Some related works have been published in recent years, such as multiregressor models, or for instance converting classifier to predictors. Generative Adversarial Networks (GANs) is an obvious example (Ian J. Goodfellow 2014): as two neural networks compete with each other, a generative model can, for example, generate plausible and convincing yet fake images of faces and people[1]. Dynamic Routing Between Capsules (Capsule networks) by Hinton was an important contribution to the field in 2017. First, because of the stature of Hinton in the machine learning community. More importantly perhaps, capsule networks offer a novel approach to model the hierarchical spatial relationships of objects. Indeed, the pooling operations that have made Convolutional Neural Networks (CNNs) so popular and efficient this last decade, actually loose a large part of the internal representation of the data. Capsule networks rely on multiple predictions to activate "capsule layers" in a tree-like structure, "solving the problem of assigning parts to wholes" (Sabour et al. 2017). Spiking Neural Network Architecture (SNNs) is another emergent approach which gained traction very recently: such networks

rely on trains of pulses rather than gradient descent to adjust their weights, as spikes trains are discrete and non-differentiable (a requisite for backpropagation). Some attempts to combine deep learning and spiking neural architectures have been proposed to study biological models, using classification/regression of the activated spiking patterns (Doborjeh et al. 2018). However, it is not yet clear if supervised learning is achievable with this approach.
Another interesting hybrid technique consists in using CNNs for time series analysis, for example price prediction in financial data (Sezer and Özbayoglu 2019). 2-D images are generated from stocks charts and a trading model representing the market conditions is run to define a buy-sell strategy.

2.2. Fundamentals

While regression models are generally used to predict values, classification is preferred to assign classes to data. To better understand how we can leverage different types of artificial neural network architecture to combine regression and classification in a biologically plausible paradigm, I will provide in the following sections an overall related work. Indeed, artificial neural networks are fundamentally statistical models. As such, it is important to understand their structure at the mathematical level, how they can possibly relate to a biological model, and what were the latest contributions and limitations for each type of architecture. In this chapter, I will explain some of the most fundamental concepts in machine learning. Those detailed explanations, and the literature that follows, should contribute to a better understanding of the neuromorphic network paradigm. In the following section, the reader should assume the following: an artificial neural network consists of layers (input, hidden, output), each of them consisting in nodes (Rumelhart 1986, Salehinejad et al. 2017). Each node, sometimes also referred as unit, can be considered the computational equivalent of a neuron. As such, a node is said to be "activated" (firing) when its input passes a certain threshold. This threshold is set by an activation function. In its simplest form, a step-function centered around 0 can be set to return 1 on its y-axis when the input values on its x-axis are positive, 0 otherwise. Since the objective of a neural network is to approximate any type of function, random weights and biases are

attributed to each nodes during an initialization phase, shaping (weights) and shifting (biases) the curvature of each activation function (Figure 2.2) for each node, so as to provide as many local approximators as possible, and find the best compound of functions that approximates a given input function.
The mathematically-inclined reader may refer to the "Stanford notations for Deep Learning",[2] for further explanations on the notations rules used in this chapter.

2.2.1 Supervised Learning

Supervised learning has been very popular (and still is) in data mining applications: given a set of labeled data, the network maps an input to an output and infer functional rules during a training phase, to generalize over a new set of data during a testing (or simulation) phase. For classification, targets can be defined as the supervised component, and labeled data provided to the network will represent the features that should be approximated. In the case of a regression for prediction, supervised learning can also be used, in that case univariate or multivariate time-series will be re-framed as an input set and a target set, and passed through the network to forecast approximate values. This approximation takes place as the outputs diverge from the targets, and the network aims to minimize the error between those two sets.
Classically (Rumelhart 1986), a deep learning artificial neural network consists in an input, as many middle layers as needed, and an output layer (Figure 2.2). Each layer contains a number of units, or nodes, or "neurons" (in the following, those terms are used interchangeably).

The output layer will yield $\{y_1, ..., y_m\}$ as an output dataset to be compared with the initial input dataset $\{x_1, ..., x_m\}$, where m is the number of the dataset samples. To create this network, each layer is connected to the next, such that the input for each layer's unit is the sum of the dot product of its weights by its previous layer's outputs, where a is known as the activation input of a neuron, w represent its weight and b, the bias (Figure 2.1[3]). This input is activated by another function, the "activation function", denoted g. As shown in Figure 2.2, the bias shifts the activation function to the right or to the left, while its steepness is changed by the weight. A logistic sigmoid can be used (Salehinejad et al. 2017), as it introduces non-linearity, which allows us to map out any type of values. It

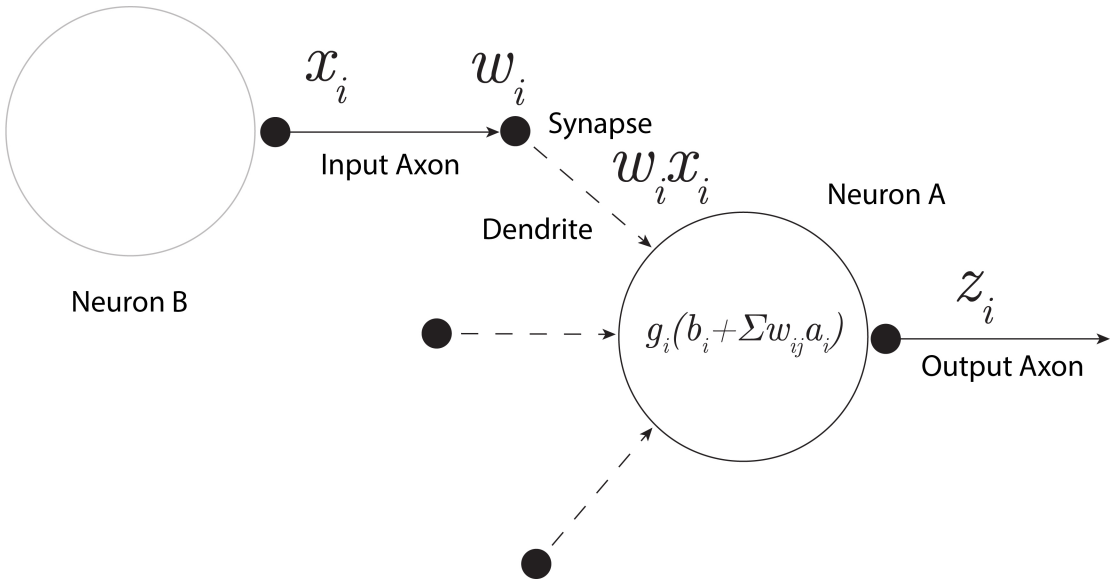

Figure 2.1 Mathematical expression of a neuronal cell: Neuron A and B propagate an electric signal unidirectionally from their neuronal cell body to their axon terminal, characterized by the output vector z_i in A. From B to A: to sustain neuronal firing, synaptic transmission defined as the weight w_i is ensured for the input vector x_i; the excitatory input $w_i x_i$ is provided on the dendrite of Neuron A to produce a firing pattern modulated by the activation function g_i.

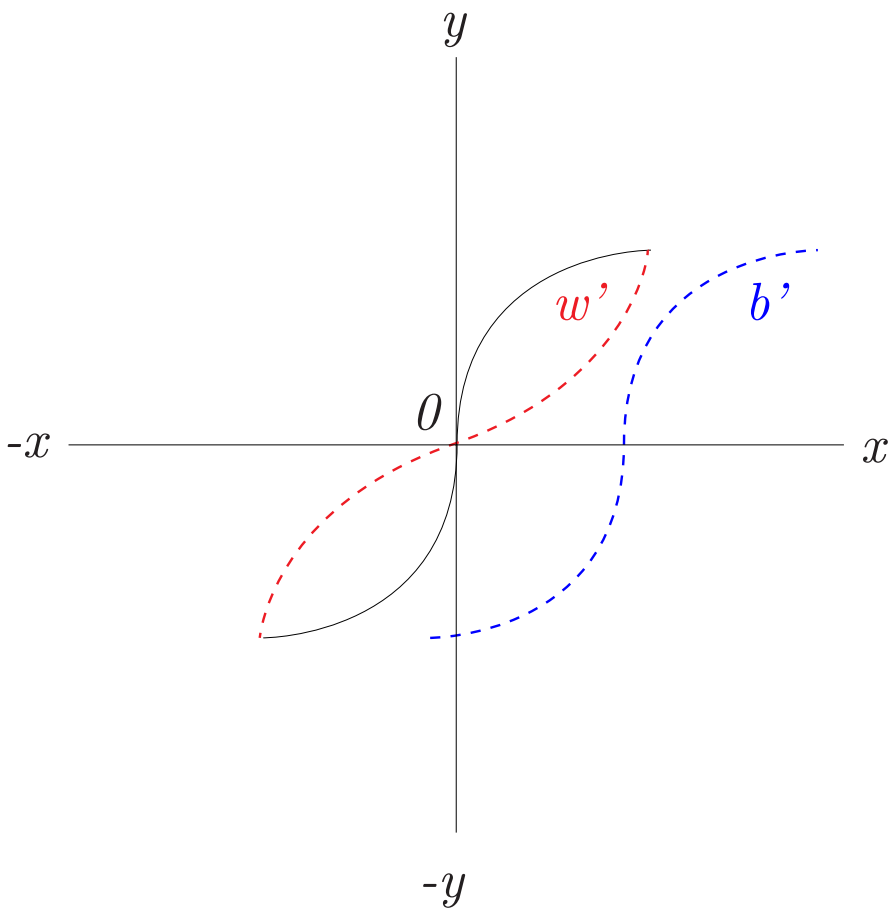

Figure 2.2 A sigmoid function g admits a weighted input and its bias, such that $g(wx + b)$ is shifted to the right when $b = b'$ and its steepness is changed when $w = w'$.

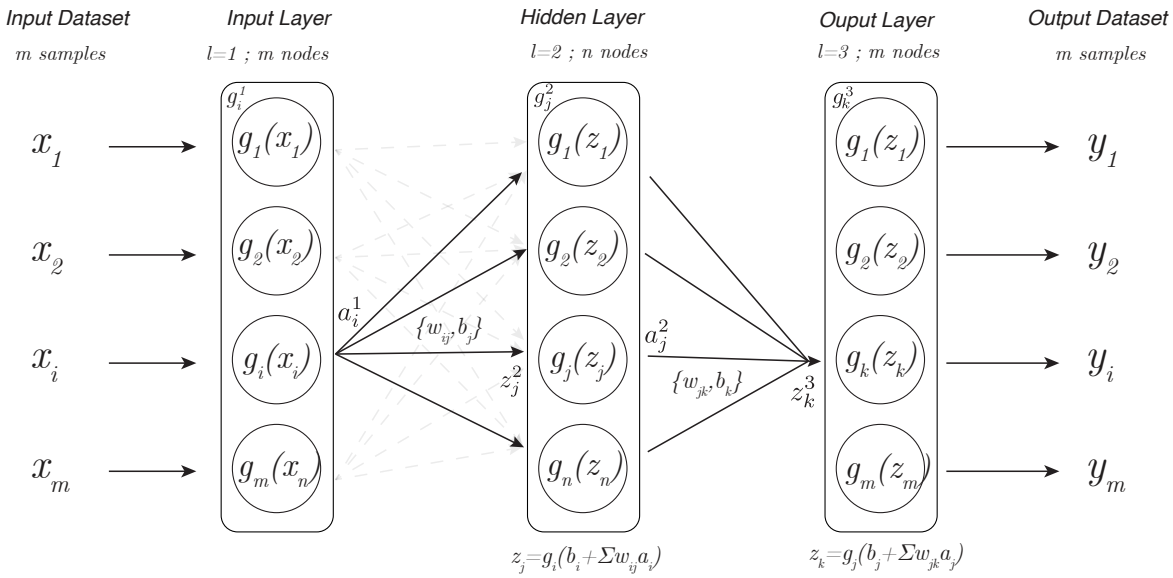

Figure 2.3 Example of a custom neural network architecture for a feed-forward, supervised learning algorithm, with 1 input layer/m nodes, 1 hidden layer/n nodes, and 1 output layer/m nodes. The error is computed via backpropagation using gradient descent.

is written as:

$$g_{logistic} = \frac{1}{1 + e^{-x}}$$

Also, its derivative is well-known:

$$\frac{\mathrm{d}g_{logistic}}{\mathrm{d}x} = g_{logistic}(1 - g_{logistic})$$

Once the network is set, and its weights and biases randomly assigned for each node of each layer, the input-target pair is passed through the network, as highlighted in Figure 2.3.

Its aims is to minimize a cost function J between its targets $\hat{y}_k$ and its outputs y_k so as to fit the input set x^k. To compute this minimization, the mean squared error function (MSE) is a standard, but many other cost functions may be implemented instead. Here, for the sake of simplicity, we will only refer to the MSE

when computing the error. It is written:

$$J(\hat{y}, y) = \frac{1}{m} \sum_{1}^{m} (y_k - \hat{y}_k)^2$$

Classically, the gradient descent (Yann LeCun 1989) (or convex optimization) algorithm has been used to find the parameters that minimize the cost function: as the cost decreases, we can better approximate the two parameters $\{w, b\}$ for each node of each layer. As suggested in Figure 2.4, this method requires to calculate the derivative for the multivariate cost function and find its local minimum. This cost function is different for each layer.

For each layer then, we need to calculate the partial derivative w.r.t to the weights set in the previous layer. The gradient will then be first computed from the output layer to the middle layer, then from the middle layer to the input layer, and at each step, the signal error will be used to update the weights, hence the term "backpropagation" (Yan LeCun 1998, Training 2006).
From the output layer to the middle layer, the cost function evaluation is formalized as follow, w.r.t to its weights w_{jk} (1):

$$\frac{\partial J}{\partial w_{jk}} = \delta_k a_j$$

where:

$$\delta_k = (y_k - \hat{y}_k) g'_k(z_k)$$

And as follow, w.r.t. to its biases b_k (2):

$$\frac{\partial J}{\partial b_k} = \delta_k$$

where a_j in equation (1) represents the activation input for the layer l+1 (or the output of node j from the layer l to layer l+1) and $(y_k - \hat{y}_k)$ in equation (2) compares the targets $\hat{y}_k$ to the corresponding outputs y_k.
$g'_k(z_k)$ is the derivative of the activation function in the output of layer l+1, and z_k is the input from node k in the previous layer l-1 to node j in layer l.

Next, the weights and biases parameters will be updated, such that we can better approximate our targets, and find a global function that generalize well over other inputs. Regarding the weights of the connection between the output

layer and the middle layer, they are updated such that $w_{ij} = w_{ij} - \eta \frac{\partial J}{\partial w_{jk}}$, where η is the learning parameter, defining how much the error contributes to update the weights. The error signal can then back-propagated further toward the previous (input) layer. The biases are updated analogously and the computation is reiterated for the error signal of the connection between the input layer and middle layer. Again, w.r.t to its weights w_{ij}, J is written as:

$$\frac{\partial J}{\partial w_{ij}} = \delta_j a_i$$

where:

$$\delta_j = (y_j - \hat{y}_j) g'_j(z_j)$$

And w.r.t. to its biases b_j:

$$\frac{\partial J}{\partial b_j} = \delta_j$$

Once the weights and biases are updated for each layer, the error optimization is reiterated, so that the neural network can converge further toward a better solution. A well-known issue, however, is the vanishing gradient problem (Hochreiter 1991), where the gradient is so small that it prevents the weights to be updated. Historically, other methods have been developed in parallel, such as the momentum and Nesterov momentum approaches, Adaptive Gradient, Conjugate Gradient (used in one of the implementation described in the next chapter) and were proved to be faster and more reliable, but are not fully covered here for the sake of brevity.

In 2011, an important contribution by Xavier Glorot (Glorot and Bordes 2011) proved that ReLU (Rectified Linear Unit) activation functions may mitigate that issue, also showing that they may be closer to biological models, computationally faster and more efficient. For instance, non-significant outliers in the training set may be removed early in the process to improve the performance, using techniques such as PRISM, outlined by Smith in 2011 (Smith and Martinez 2011). However, a major drawback, inherent to the supervised component of the learning process, is that the dataset needs to be labelled "by hand", an almost impossible task for a very large amount of data. Very recently, frameworks have been developed to ease this expensive process, such as automated tagging using CNN (Keunwoo Choi 2016) or crowd sourcing (the Backyard Worlds project)[4].

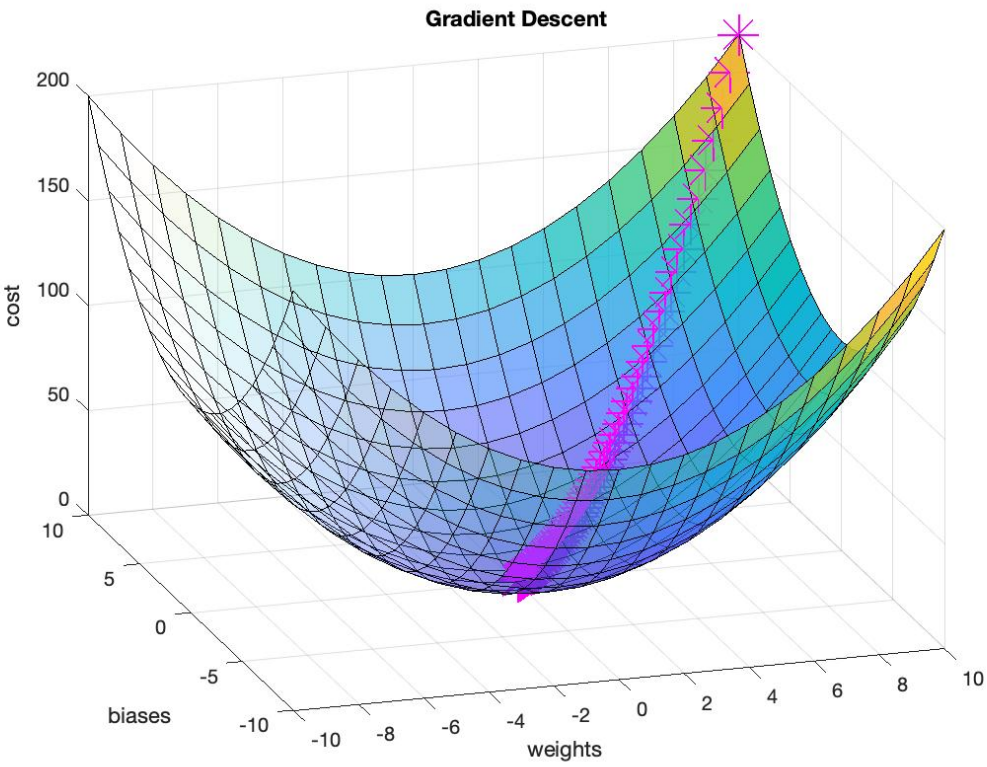

Figure 2.4 The gradient descent algorithm (©simar, 2012), marching downhill to find the local minimum of the function $z = x^2 + y^2$ where x represents the weights, y represents the biases, and z, the cost.

2.2.2 Unsupervised Learning

Unsupervised learning does not need labeled data to infer new rules, and is therefore usually preferred for classification when labels are not available or labelling is too time-consuming. Self-organizing maps are one such example: the input space is considered as a continuous network of nodes, where a neighborhood function aims to reduce the dimensionality of the data while preserving its topology. Generally, unsupervised learning algorithms refer, directly or indirectly, to Hebbian principles (Pearlmutter and Hinton 1986), where if a neuron A spike activity is strongly correlated to a neuron B spike activity, A and B are most likely physically connected. Applied to machine learning, the Hebbian learning function is defined as follows: the weight change Δ of the connection between a given neuron input A_i (i corresponding biologically to one if its axon) and a neuron output B_j is computed according to the rule:

$$\Delta w = \eta * A_i * B_j$$

where η is the learning rate (See Chapter 2.1: Supervised Learning).

Because the parameters $\{w, b\}$ are unknown to the network, in the case of Unsupervised Learning, several methods can be used to find them without any targets. An interesting approach is the method of moments, widely used in statistics and initially developed by Pearson in 1936 (K. 1936).

Practically, however, as stated by Ruffini (Matteo Ruffini 2017), the adoption of this method has been very limited, despite its very strong theoretical foundation, and a more robust and comprehensive approach, such as the log-likelihood, is generally preferred. For this type of method, a family of distribution is selected (such as the natural logarithmic distribution, given that the population of the model follows this type of distribution) so as to find the values of the parameters (for example the weights and biases) that makes the model more probable. This technique is very often encountered in the literature when it is not possible to label the data, or when it is not precisely known what type of features should be learned within the set. Recently, eminent researchers in the machine learning community, such as Andrew Ng, Yann LeCun, or Hinton, have all emphasized that Unsupervised Learning presents a real challenge and an important opportunity for the development of artificial intelligence. In games in particu-

lar, Generative Adversarial Networks (GANs) have been cited frequently since 2014 (Ian J. Goodfellow 2014), making the use of Unsupervised Learning techniques to let one neural network compete against another in a zero-sum game, a very promising framework for the next few years.

2.2.3 Reinforcement Learning

Reinforcement Learning, popularized by the TensorFlow library release in 2017, can be summarized as a reward system informing the agent about its states to optimize its policy function. From this definition, we should note that three important concepts are highlighted:

- Observations or Markovian state variables : the network records a set of states of the system.
- Agency: an agent takes action over selected variables of the system (the control variables) and,
- Policy: the agent gets rewarded or penalized for its action, in order to optimize the policy function of the system i.e. the hidden rules that govern the states of the system.

This system is best described by a recursive function maximizing the reward of the agent (Figure 2.5[5]) and as such can be better formalized by the Bellman equation (R. 2003):

$$n \in \mathbb{R}_{>0}, v(s) = r_t + \gamma r_{t+1} + ... + \gamma^n r_{t+n}$$

where $v(s)$ is the value of all the states of the system, computed as the sum of the rewards r_t at every step t, discounted by a factor γ that increases exponentially as the agent takes action over the next step. Hence, the equation can be written recursively as:

$$v(s) = r_t + \gamma v(s_{t+1})$$

Because the function should be maximized w.r.t. the actions taken by the agent (as the agent should collect as many rewards as possible), the Bellman function is often seen as: $v(s) = argmax(r_{s,a} + \gamma v(s_{t+1}))$ where s and a denote, respectively,

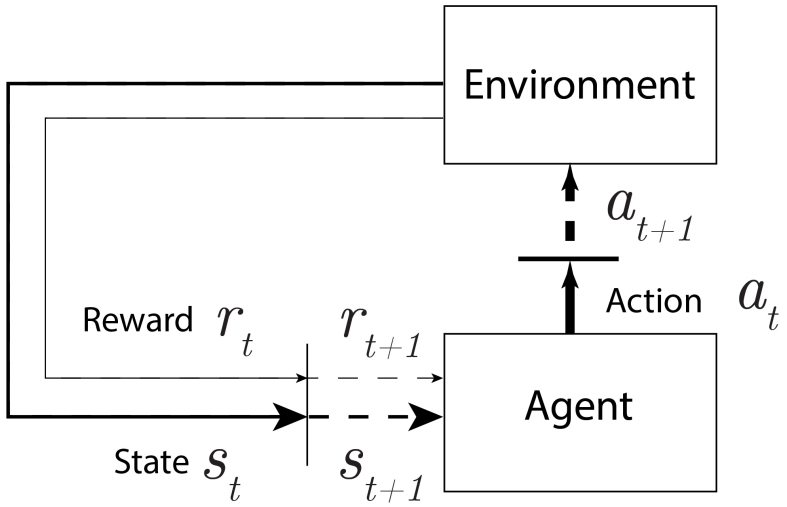

Figure 2.5 Diagram of the reward system in Reinforcement Learning: the agent acquires the state s_t of the environment at time t, takes action a_t over the environment, and get rewarded with r_t for its action. The process is looped recursively.

the state of the system, and the action of the agent at step t. Artificial Neural Networks can approximate this function with a method known as Q-learning, where Q denotes $v(s)$. Similarly to Supervised Learning methods, the objective function aims to minimize the difference between the Q value and its target value. This target value corresponds to an optimal Q value i.e. the expected total reward for the agent. Q-learning has proved to be a major advance, especially in game-oriented applications, but has nevertheless lost favor recently, with the introduction of interesting new supervised and unsupervised techniques, such as GANs (Zuo et al. 2018), as previously seen, or echo-state networks, a supervised technique where the output weights, which drive the change of the adaptation, are reused to produce new outputs (Schiller UD 2005). Similarly, recent research work in biologically-based computation, subsumed under the name of "Reservoir Computing", have also shown interesting results (Y. Paquot 2012) and is a source of inspiration for my current and future work.

Notes

1 https://www.thispersondoesnotexist.com

2 Standard notations for Deep Learning, an interesting initiative to set a new standard for deep learning mathematical notations: https://cs230.stanford.edu/files/Notation.pdf

3 Adapted from: http://cs231n.github.io/neural-networks-1/

4 The Backyard Worlds project: a collaboration between between NASA and UC Berkeley, the American Museum of Natural History in New York, Arizona State University, the Space Telescope Science Institute in Baltimore, and Zooniverse, a platform for developers and educators to manage science projects: https://www.zooniverse.org/projects/marckuchner/backyard-worlds-planet-9

5 Adapted from http://www.wildml.com/author/dennybritz/

Chapter 3
Neuromorphic Networks

3.1. Paradigm

Etymologically, "Neuromorphic" is a closed compound word, combining Ancient Greek *νεῦρο-* (Latin *neuro-*, originally "nerve")[1] and *-μορφή* (Latin *-morphḗ*, "form")[2]. A neuromorphic system refers to a system mimicking the neural system. For instance, the notion of "Neuromorphic electronic systems" was exemplified at Caltech by Carver Mead in 1990 (Monroe 2014), to describe a novel computing paradigm in hardware design that would make use of analog signals, instead of digital signals, to decrease the cost of computational power. Directly inspired by the "elementary functions", "representation of information", and "organizing principles" of the nervous system (Mead 1990), the paper echoes the progress in neurosciences at the time.

In 1982, David Marr's posthumous publication "Vision" (Marr 1982) presented a pivotal work in computational neurosciences. By carefully detailing how perception can be defined in terms of computational, algorithmic, and implementational processes (notoriously, his three "levels of analysis"), Marr developed the idea that the way we perceive the world can be formalized by a set of functional rules following the computational logic in the physical world. Here, "Neuromorphic networks" refers to biologically plausible artificial neural networks, that is, algorithms functionally and structurally inspired by biological models. Perception is therefore a central theme in the creation of such networks, as modeling biomimetic neural circuits implies a body of knowledge on how neural signals effectively translates sensory-motor information with predictive power. Accordingly, neuromorphic networks could not be totally envisioned without interactions to the physical world, and hardware implementation with feedback loops, as I will develop in the last chapter, should be the final design objective of such networks. Simulating and,

ultimately, predicting physical features, or physiological signals, will confirm the model.

To design biologically plausible networks, one very important concept to highlight is the notion of percept, the mental objects our brain relentlessly produce from sensory information (even fragmentary, see Fig. 3.1). A perception, as a mode of presentation of a particular stimulus, does not arise from the brain, but from a dedicated organ: vision, for example, is acquired via electric signals propagating from the retinal ganglion cells to neuronal cells in the visual cortex, and visual information is processed in parallel cortical circuits by specialized networks (identifying contrasts, edges, colors...), eventually linking vision to decision and interactions with the physical world (Luo 2016).

Perception is therefore an active process and in neurobiology, behavioral assays are constantly designed to reveal how behavior and decision-making patterns relate to neuronal activity. A well-known experiment (Luo 2016) consists in presenting moving dots to a monkey and monitoring the saccades (the motion of the eye fixation point) to correlate the task variables to his behavioral response. Classically, electrodes (Niell and Stryker 2008) were also implanted in the brain, so as to record and perturb the neuronal activity, in order to further investigate how it relates to behavior. An important result is that the perturbations induced in the brain did influence the monkey's behavior, advancing the interpretation of the activity (linking neuronal dynamics and behavior) from correlation to causation. The consensus is that neurons in the visual cortex have preferred orientations (Swindale 1998), and that their firing rates can predict the direction of the sensory information. Recently, more advanced techniques, such as optogenetics, utilize genetically modified protein indicators to express light-sensitivity in targeted neurons. Perturbations may be introduced as well to take control over the electrical activity of the neuronal circuits. This new class of sensors and actuators represent a powerful tool set to study the brain (Aoki et al. 2017).
In machine learning, new types of artificial neural networks are constantly designed as well. However, the computations involved in such circuits usually do not take into account the specifics of an environment. Those networks are created ad hoc, processing large streams of data to obtain a statistical model that will be used to achieve a given task. This may be useful for industrial and commercial

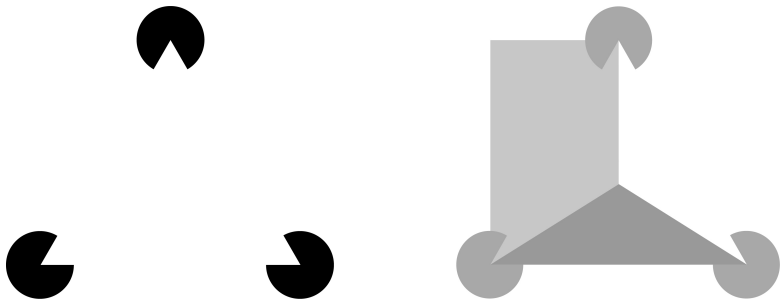

Figure 3.1 A 3D variation of the Kanizsa triangle, an illusory contour first described by the Italian psychologist Gaetano Kanizsa, and covered in David Marr's Vision work, emphasize how we perceive information. Here, a tetrahedron can be visually inferred by both the negative space created by the troncated black dots and the contrasted background, telling us that we are biologically inclined to make sense of our environment, our neural cells compensating for any lack of sensory information.

applications, but does not represent an ideal solution to achieve general-purpose intelligence. Instead, learning in biological neural networks presupposes a structured collection of cells in continuous interaction with the physical world. This is where the core concept of "neuromorphic networks" takes shape: an artificial neural network inspired by biological systems and principles.

Indubitably, artificial neural networks are promising for developing human-centered applications. However, the objective of this thesis is not only to introduce novel applications, but also to provide a theoretical background on how intelligence, or at the very least decision-making processes, may arise from fundamental and functional components that may be modeled after the brain. Historically, machine learning core principles have been theorized on a biologically plausible brain architecture. However, lately, bioinspired approaches have been facing some criticism: first, there are now countless examples of implementation that clearly diverge from a biologically valid model and yet, impressive performances in solving a wide range of problems have been reported in the literature on an almost daily basis, showing that there might be different types of intelligence required for different types of task, and the research in the field does not burden itself anymore with its neurophysiological roots, at the notable (and expected) exception of the neu-

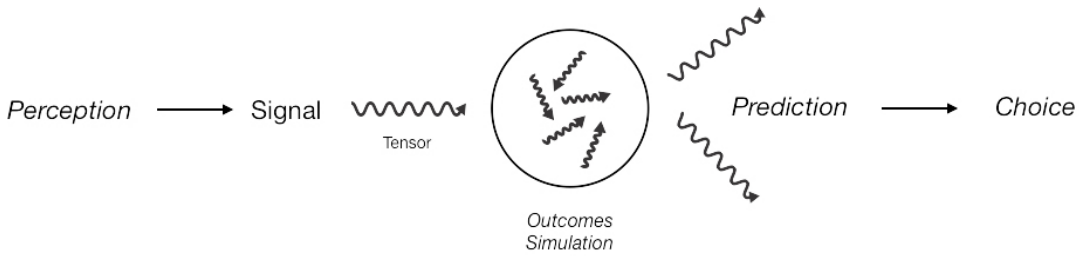

Figure 3.2 Diagram of a neuromorphic network with prediction and classification.

rosciences (Guerguiev et al. 2017). Second, to this day, artificial neural networks are still struggling to explain a realistic brain model. The two most common difficulties in neuroscience to validate those computational models are the following: the first difficulty is to accurately simulate the activity of large cortical networks, which has yet to be done. The second difficulty is that an accurate simulation will not suffice: the results need to be interpreted. At present, the artificial neural network model is often used as a "black box" (Sussillo and Barak 2013) to compute a stimulus-response mechanism: the internal functions are not accounted for and it is hard, then, to figure out exactly how it has learned to do tasks. Without a comprehensive framework to interpret data, our understanding of the most fundamental principles of functional connectivity in the brain will remain opaque (Yang et al. 2017). On the opposite side, if artificial neural networks can be proved to be a solid representation of how the brain work, effectively validating the statistical models, they will provide us with a novel theoretical background. In return, more powerful applications, biologically compatible, will be developed, opening entirely new perspectives in the field of brain-computer interfaces.

Here, I propose to design a Neural Network to model and test the hypothesis that behavior can be modeled based on predicted outcomes. Classification and Regression will be used as complementary approaches to attempt to mimic biological models' learning.

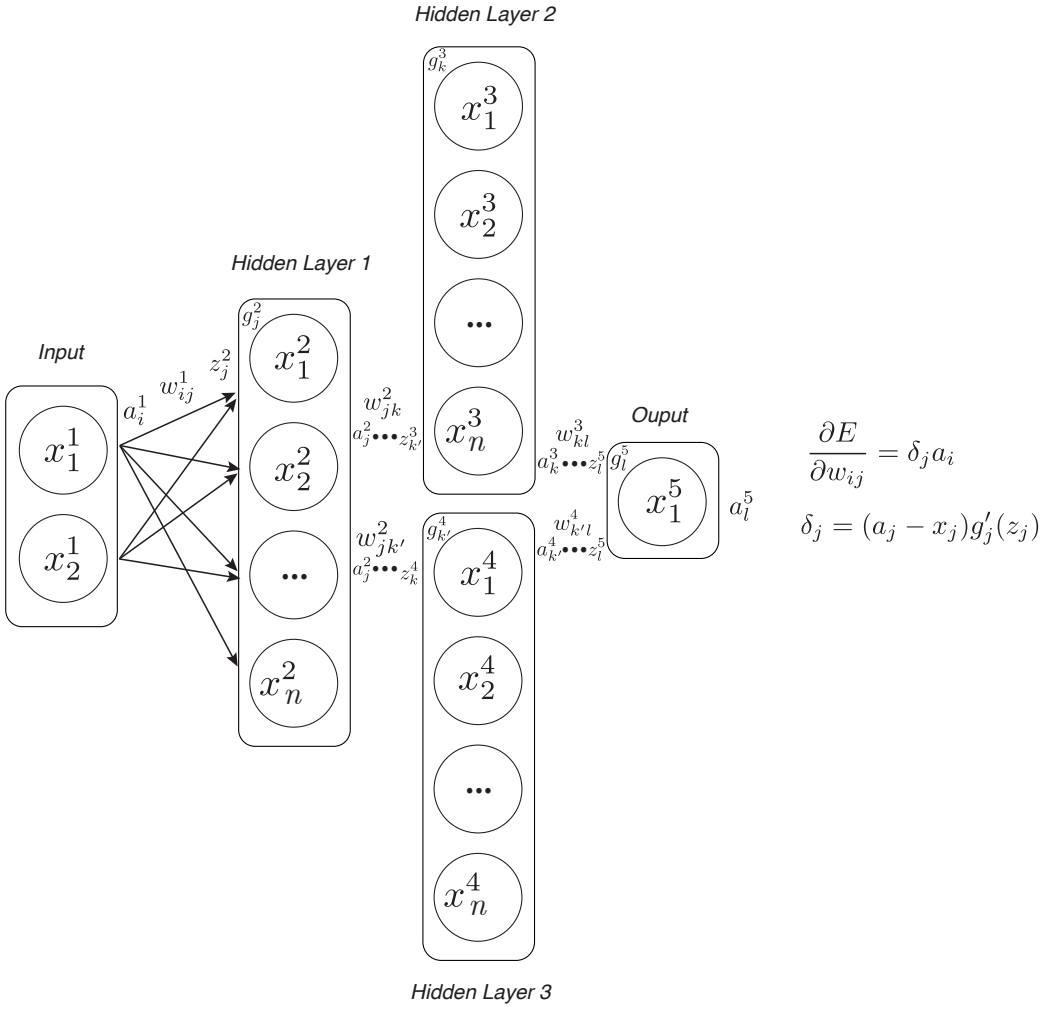

Figure 3.3 Functional structure of a stacked architecture. The output, evaluated to compute the network error, is passed through the network's layer.

3.2. Modeling Neural Circuits for Prediction

The custom artificial neural network should admits a series of inputs, for instance a sequence of keystrokes or images, while any signals and possible behavioral components can be added as targets. The initial inputs yield artificial outputs, corresponding to the signal components and behavioral components of the network. The neural net aims to minimize the objective function such that the network output error is:

$$\frac{\partial E}{\partial w_{jk}} = \delta_k a_j$$

where:

$$\frac{\partial E}{\partial w_{jk}}$$

represents the derivative of the error, a_j represents the output of node j from the layer l to the output layer l+1 and:

$$\delta_k = (a_k - t_k) g'_k(z_k)$$

where

$$(a_k - t_k)$$

compares the targets a_k (the recorded neural activity in different regions) to the corresponding outputs, where:

$$g'_k(z_k)$$

is the derivative of the activation function in the output layer l+1, and where z_k is the input to node j in layer l. The final output layer represents the behavioral component.

The architecture is designed as follows: a LSTM-based neural network, referred as a **predictor**, predicts based on the task variables only. A multilayer **classifier** then identifies possible actions over the predictions. This necessitates a training in two steps, as summarized in Fig. 3.5. First, the main input, such as the task variables, is passed sequentially (over time) through the network, along with auxiliary inputs, to help the optimizer computing the objective function. The weights and biases matrices may be computed during the training phase over thousands of trials to set the predictor. Once this initial training has been completed, each target is converted to a main input, and the desired outputs, for

instance a behavioral component, is set as targets. It is important to note here that the structure of the hidden layers should be loose and organic, as we will see in detail in the next section. Similar in that to echo-state networks, where the output weights, which drive the change of the adaptation, are reused to produce new outputs (Schiller UD 2005), this type of implementation represents a more flexible approach for training over complex time-series, as recent research work in biologically inspired computation, subsumed under the name of "Reservoir Computing", have shown (Y. Paquot 2012). The spatial configuration follows the biological model, where information streams from the retina pathways, engage in LGN layers to the primary visual cortex, then from the visual cortex to other highly specialized areas. Those areas support various features of the sensory input, such as, in the case of the visual cortex, form and color, or motion and depth (Luo 2016). In a neuromorphic network, the behavioral layer returns the final output, a vector representing the network decision over time. This implementation allows to predict the model's response given a specific stimulus, a property inherent to recurrent neural networks (Hammer et al. 2009)), and this type of architecture may be structurally more representative of the biological model's cortex (Spoerer et al. 2017).

One method, to explore further the mechanism by which the network performs the tasks, consists in linearizing the dynamics of the system, following a top-down approach, in a reverse engineering fashion. However, because these networks are trained and not designed following a number of constraining assumptions, the long-term goal of developing neural networks is to achieve a bottom-up design where better insight is gained from experimental data instead of being provided solely by theory-bounded methods. My central hypothesis is that, if artificial neural networks are essentially data-driven constructs and can learn most of the dynamics present in the data, the representation of the data must be computed a posteriori. Yet, after being trained, the phase space of the network, which represents the internal states of the network, must be interpreted with linear methods, as the total number of parameters (weights and biases) in multilayer neural nets can be extremely large for the data sets we are considering. A promising method to study such dynamical elements in the phase space of our trained networks will be to approximate the full nonlinear system with a succession of more easily inter-

pretable linear systems defined by fixed and slow points (Sussillo, D. and Barak, O., 2013).

3.3. Biologically plausible neural network

The bio-inspired artificial neural net described in this section represents an elementary mapping of the structure and functions of several regions of the brain. Here, the structure refers to the spatial connectivity between different areas, as mapped from the MSDL atlas [3]. The functions of each region are determined by the functional connectivity between those same regions. Otherwise put, it represents the activation of population of neurons over time, and unlike the structural connectivity, the functional connectivity may describe activation patterns occurring in physically unrelated areas. The correlation matrices were extracted from the signals in the ADHD200 dataset [4]. 30 participants were selected to build a robust connectivity matrix, Fig. 3.4.

Spatially, the neural net is designed to map the connections of large brain regions. The net can be expanded or re-configured with other regions, depending on the task and experimental conditions.

3.4. Structure and Function

The structural and functional connectivity can be extracted from correlation plots or maps, providing measures of the connection between population of neurons, and helping to analyze how the brain regions are interconnected. The methods to compute those measures may differ according to the experimental conditions, preferred toolboxes and computational models, but they are usually a conjunction of derived data from neural recording for the functional analysis part (for instance by calculating Pearson's correlations over time or testing different permutations between groups of neurons against random networks with the BRAPH toolbox (Mijalkov et al. 2017) and anatomical work, imaging physical routes between neurons from one region to another. Over the last decades, several atlases have been proposed to establish the topology and connectivity of mammals brain (Hashikawa Atlas, Allen Human Brain Atlas, The Brain/MINDS 3D digital

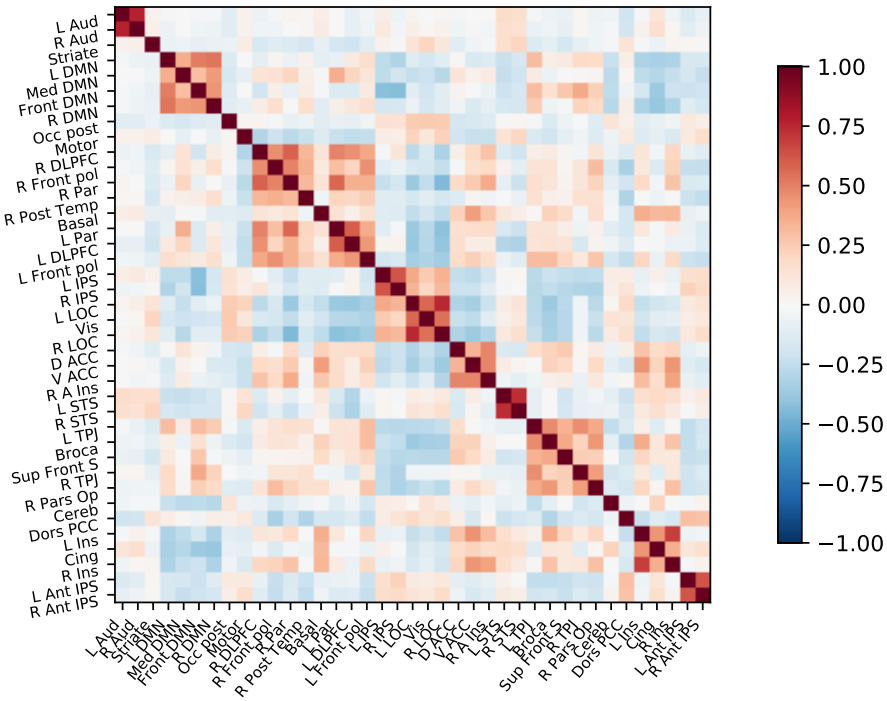

Figure 3.4 Pictured is the connectivity matrix from the ADHD dataset, with the atlas data from MSDL for the labelling. The rows and columns of the matrix represent the extracted regions. The gradient bar indicates the normalized strength of the correlations. This matrix will help define the connections between each region in the neuromorphic network.

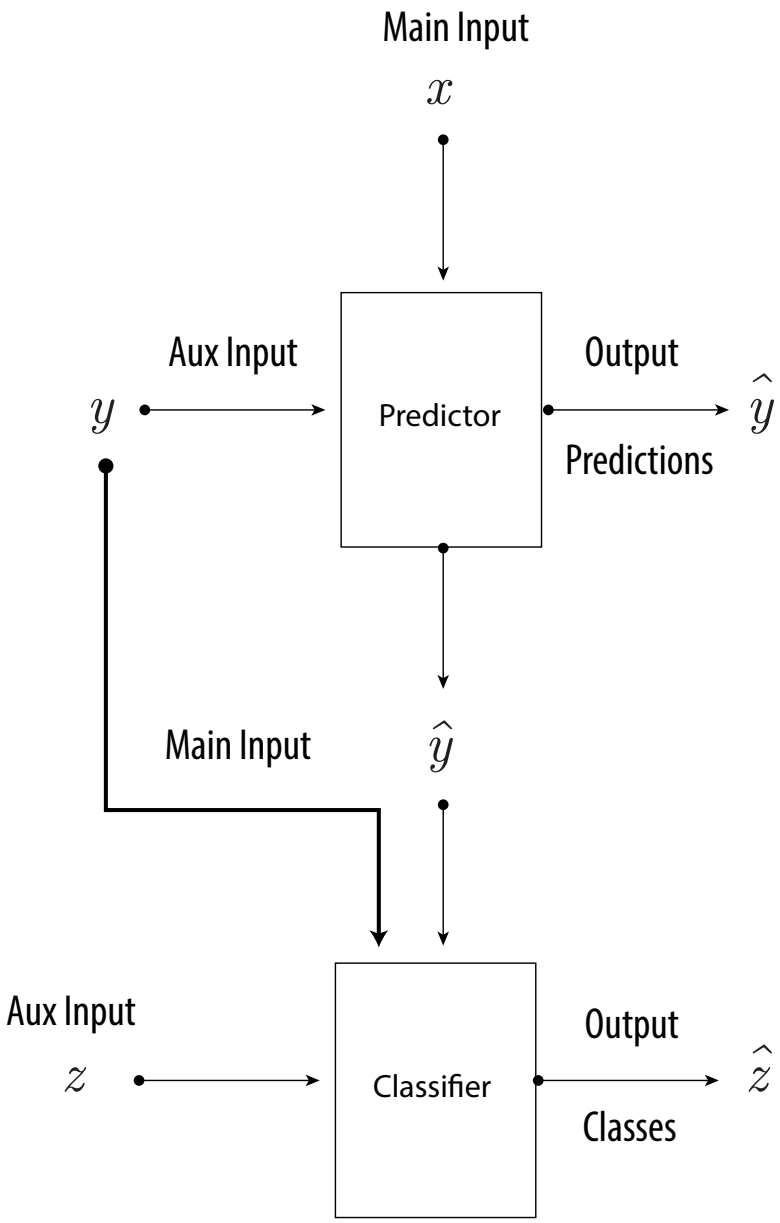

Figure 3.5 Diagram of the predictor/classifier network.

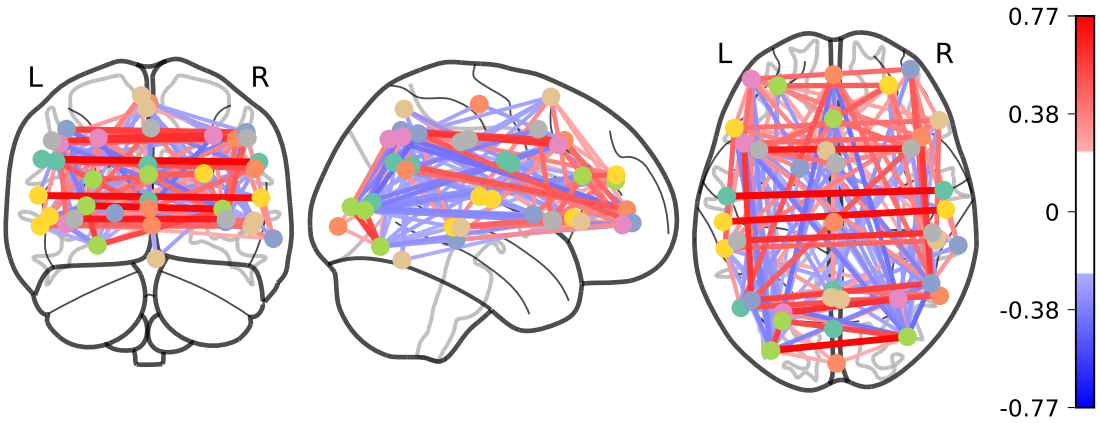

Figure 3.6 Brain Atlas extracted from the MSDL data set.

marmoset brain atlas to name a few). For this thesis, I used the MSDL/ADHD200 public dataset to design the first version of a bio-inspired artificial neural network. The MSDL atlas and ADHD200 dataset presents several advantages, being well-documented and open to public, they can be processed in python relatively easily thanks to the Nilearn module.

In the following paragraphs, I will describe the bio-inspired neural network in details, also shown in Fig. 3.7 as a preliminary study in SIMULINK. On the method of the implementation, the network must be designed with its two core components in mind: structural and functional connectivity mapped after the biological brain. After designing the experimental protocol, the arrangement of those meta-parameters must be taken in consideration. However, they offer a practical advantage over conventional artificial neural networks designs, as any arrangement can be easily conceived and tested depending on the experimental requirements.

Spatially, the neural net can been designed with the connections established by the MSDL atlas as a reference point: Fig. 3.6 shows the connectivity map based on the published data. The extracted map determines the spatial arrangement of the neuromorphic layers. A totally different arrangement may be used depending on the desired output. For instance, audio signal processing may be treated by adding two layers that represent the mapping of the L/R Aud connections. This

constitutes the basic "blueprint" for the neuromorphic network architecture.
The rows and columns of the matrix in fig. 3.4 represent individual cortical areas. The gradient bar represents the estimate of the central tendency for the correlation coefficients. Only a few regions may be selected, for instance the parietal, DMN, occipital and frontal lobes. Those regions generally correspond to the treatment of information with a high selectivity for a specific task, (e.g. the visual cortices have high selectivity for orientation (Solomon and Rosa 2014) (Solomon and Rosa 2014)). Functionally, this connectivity matrix is used to define the loss weights for each layer. Indeed, as we have seen, the neural net aims to minimize the objective function of its hidden and output layers. The loss weights determine the loss contribution (after backpropagation) to different outputs. As the final loss for the model is the weighted sum of all the sub-losses, using the correlation coefficients as weight coefficients allows to fine-tune the network according to the biological model. The weights for each layer will be updated differently, based on the functional connectivity extracted from the ADHD dataset. For example, the loss weights should be low for the connections between the R Aud layer and the L DMN layer (as shown in the matrix 3.4), while obviously very high between R Aud and L Aud, strengthening the connections between the corresponding layers in the artificial model.

Finally, the neural net has been designed to capture the hidden dynamics of the recorded regions with sparse data, so as to reveal the encoding of task variables, despite the high variability of responses within the data set. As explained, on the data input structure, the neuromorphic network admits two categories of inputs. The first class should be been designed specifically for the predictor (e.g. the task variable and corresponding signals), and the second class for the classifier (e.g. the behavior and corresponding signals). The final output layer represents the result, or the observable variables derived from the task.

3.5. Reduction, Simulation and Prediction.

To prototype and test a neuromorphic network in ideal conditions, I first designed a randomized dataset for each layer. This "noise" dataset represent the task variables, the recorded signals, and a behavior component, and is a conve-

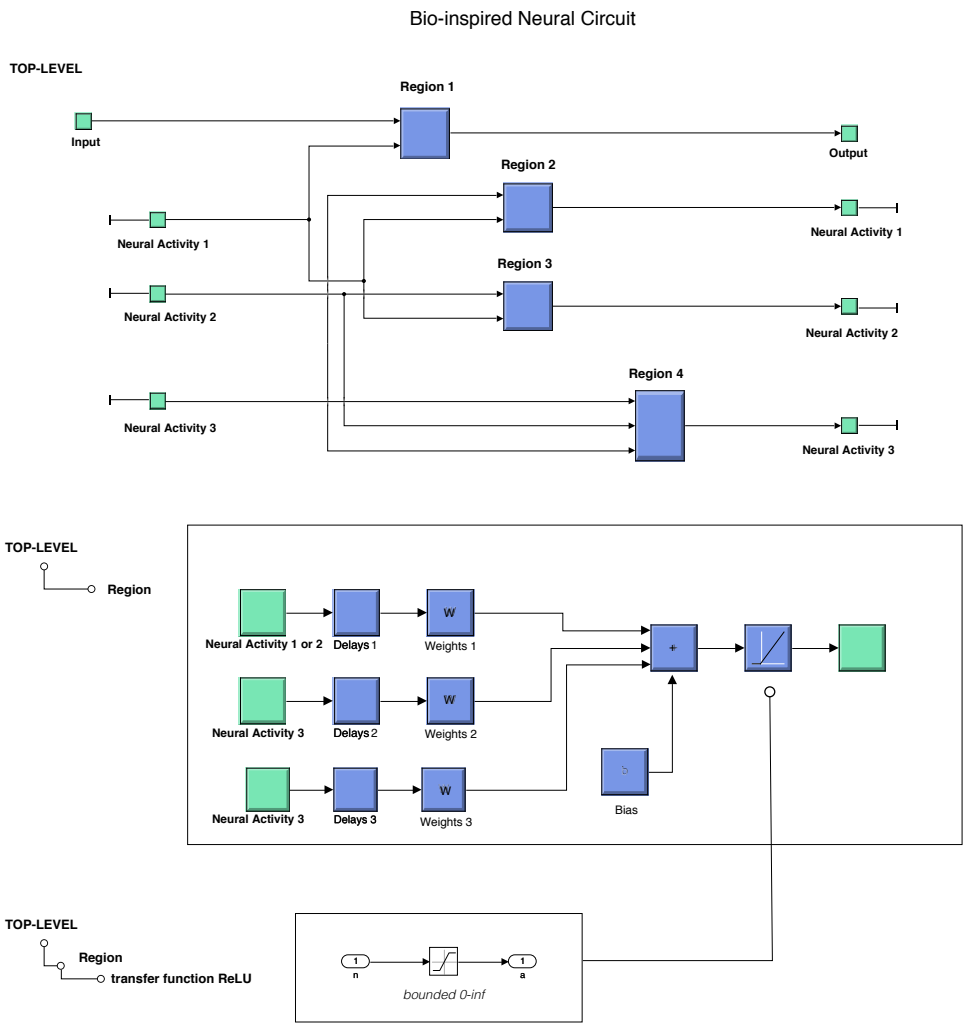

Figure 3.7 The bio-inspired neural circuit shown above was generated with SIMULINK. There are three levels: the top-level refers to the overall structure of the artificial network. In this example, one input (the task-variable) and three targets (the neural data) can be passed through the network, i.e. through Region 1,2,3,4, each representing, for instance, a visual cortex area. Each output for each region may be fed back for memory-based tasks. The second level concerns the functional component of the network: signals are integrated to compute weights and biases of the network before backpropagation. The lowest level simply describes the type of activation function, here, a ReLU function.

nient way to test the network without worrying about missing values, data errors 3.8.

Mocanu et al recently published a paper [5] about the scalable training of artificial neural networks. Their objective was to reduce quadratically the number of parameters, without sacrificing to quality. For example, RNN have a square number of connections compared to their neurons. This characteristic is inherent to their structure. However, a sparse topology, which is by default for biological models, is built on a very different paradigm: small worldness. This concept, developed in graph theory (and retrieved in many human construct and natural phenomenon), proposes that small world network nodes can be reached from every other node by a small number of steps. As such, they follow a power law distribution: they are scale-free. As the number of connections of a node to other nodes increases, the distribution of its degree decreases. Otherwise said, in that configuration, a few nodes become "connection hubs", with a large amount of connections to other nodes with very few connections. The advantage of those small-world networks, as opposed to random networks, is, as we saw it, scalability. The main issue is that ANNs have been design with opposite paradigms in mind (a task-dependent structure), and topological features are usually a lesser concern: as stated by Mocanu et al., their weights distribution tend to drop to 0. In my functional-structural approach, inspired by biological networks, I noticed that the newly conceived network does remove some weights by setting some weight layer matrices to 0. This feature is obviously a corollary of the structural aspect of the approach: because several routes are possible for the information to flow from one layer to another, the number of parameters greatly decreases, along with the computation time. The step function in the behavioral layer was chosen based on the binary type of data fed to this layer (noise, left or right choice); the cross-entropy function, which is employed for logistic regression, is above all for classification problems (as the 12AX task developed later to simulate the prototype is one of them); all those functions are responsible for minimizing the weights of some layers, and especially in the behavioral-output layer. To illustrate this argument, Fig. 3.9 shows some comparison between functionalities. On the left subplot, the ReLU function was applied to all layers and RMSE was used for the cost function. The weights are well distributed across the layers but the Pearson value p indicates a

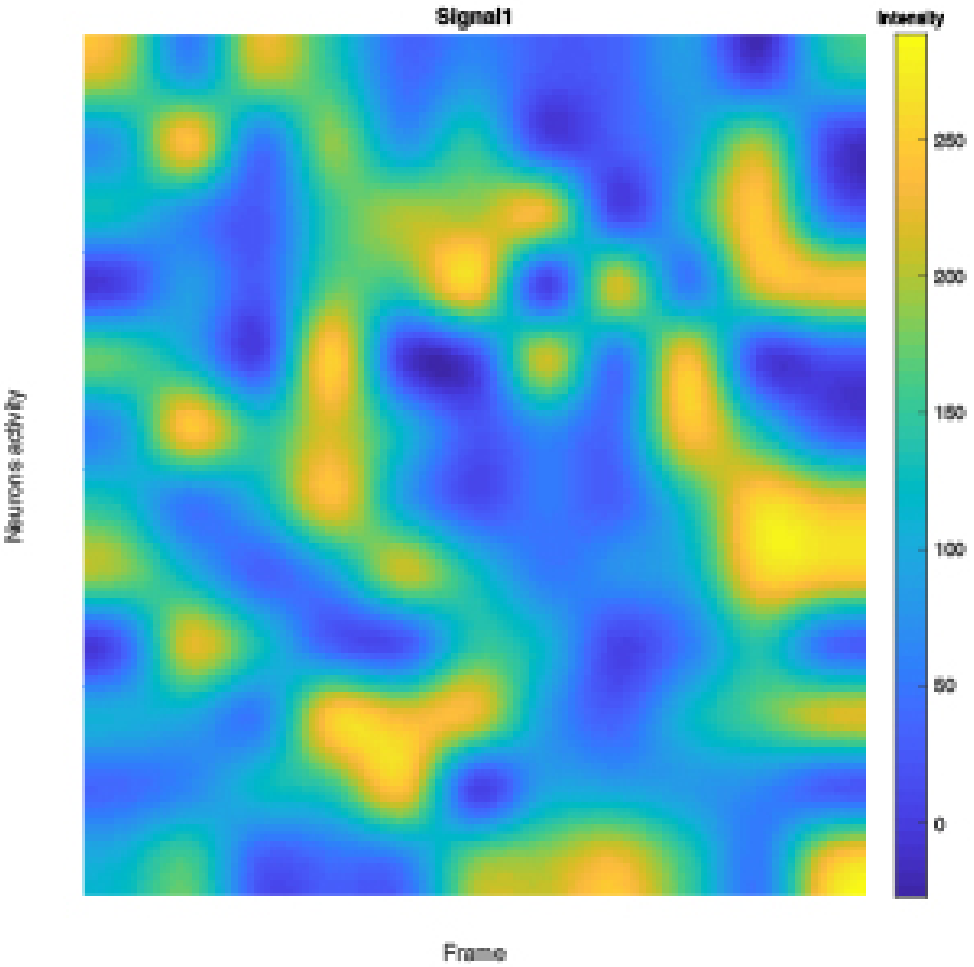

Figure 3.8 Noise matrix for the BioNN design testing.

negative correlation (low score) between the raw EEG signal and its prediction. In the middle, the step function in the behavioral layer did nullify some of its weight matrices. However, the matrices correlations are still negative. On the right, cross-entropy is finally introduced as the objective function: the final layers matrices indicate weights close to 0 but the p value is now clearly positive. Those pretests are consistent across multiple trials.

3.6. "BioNN": a Custom Neural Network in MATLAB

Hassoun, in his "Fundamentals of Neural Networks" [6] define the solution to an algorithmic problem as a set of requirements for a certain number of steps (time complexity), memory size (space complexity) and algorithm length (Kolmogorov complexity). In a neural net, it is equivalent to the number of computations, number of units, and number of weights (or degrees of freedom) where the algorithm is stored. Obviously, the objective here is to design a neural net that minimizes the complexity for each. Functional Programming (subsequently abbreviated FP) is generally considered as best suited for writing machine learning code. Indeed, machine learning applications require a heavy use of matrices manipulation; parallelisation of the processes (a core feature of FP languages), although not within the scope of this thesis, leverages the power of GPU computing for faster training. FP languages, such as Python, R, and to a lesser extent Scala, allow to write concise code, relying on scientific libraries and packages to call functions and perform operations generally in a vectorized fashion i.e. without the use of a loop or conditional statements. As of today, Python is probably the most popular language for machine learning, due to its open-source core, with a large and dedicated community, fostered by technology hubs around the world, and granted of dozen of powerful scientific libraries. Google's Tensorflow and the Keras API created by François Chollet, for instance, were used for some of the predictive applications developed in this research work, as mentionned in the next chapter. MATLAB, as a matrix-oriented language[7], is especially suited, albeit not free (commercial IDE licence, not open-source)- a point of friction often heard in the computer science community. Yet, largely spread in the academic world, its integrated IDE

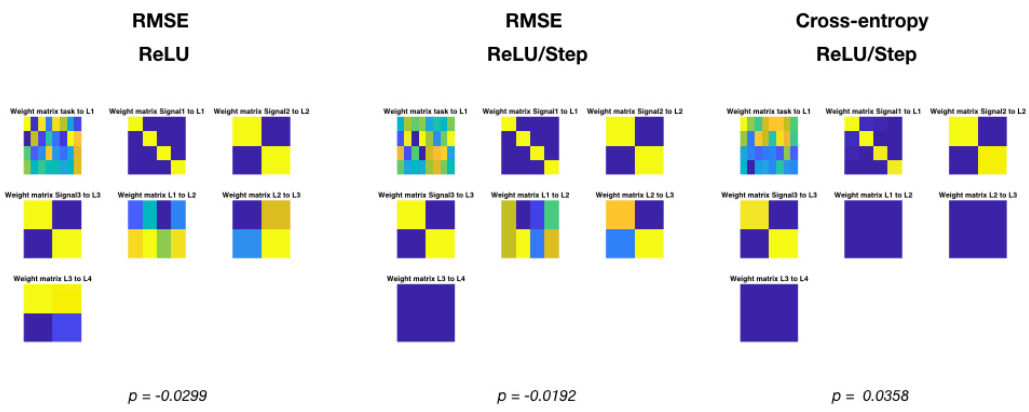

Figure 3.9 On the left, the ReLU function was applied to all layers and RMSE was used for the cost function. The weights are well distributed across the layers but the Pearson value indicates a negative correlation (low score) between the raw EEG signal and its prediction. In the middle, the step function in the behavioral layer nullify its weights matrix. However, the matrices correlations are still negative. On the right, cross-entropy is introduced as the objective function: the final layers matrices indicates weights close to 0 but the p value is now clearly positive. Those tests are consistent across multiple trials (>10).

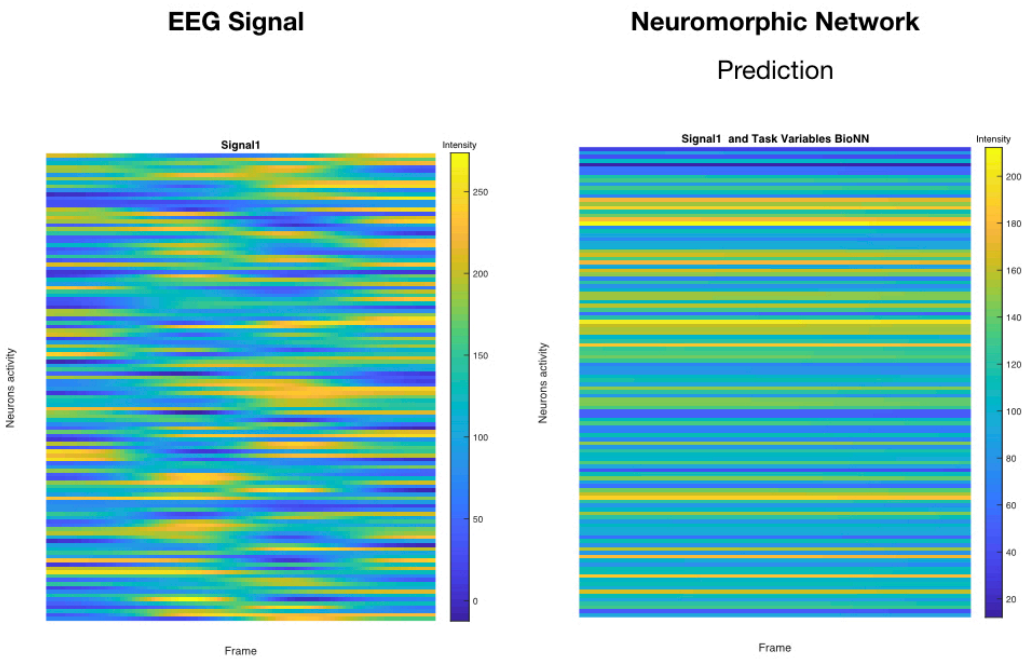

Figure 3.10 Depicted left, the original signal, depicted right, the predicted signal. The network was trained on previous tasks, but was never presented a signal, only the same sequence of characters the participant was shown on screen.

offers a huge and robust set of mathematical tools, to build and analyze complex projects. For the sake of simplicity and brevity, we will use here the MATLAB language and its Neural Network Toolbox (machine learning package) to convey the main concepts and ideas behind the build of a simple bio-inspired neural network, namely “BioNN”. We will also explain the core properties of the functions imported from the package, a necessary endeavor to understand the granular aspects of the program processes, so that more sophisticated tasks may be built upon those general guidelines. The code is also accompanied by diagrams, that follow the build step-by-step. A more sophisticated model will then be developed with TensorFlow/Keras (Python) in the evaluation part, to leverage the full potential of the correlation matrices and brain connectivity maps extracted from the MSDL and ADHD dataset.
Below are a few introductory steps:

Create a network:

First, per the language documentation, an obvious ’network’ function can be called to return a neural network with the following properties defined:

- numInputs - Number of inputs
- numLayers - Number of layers
- biasConnect - numLayers-by-1 Boolean vector, zeros.
- inputConnect - numLayers-by-numInputs Boolean matrix, zeros.
- layerConnect - numLayers-by-numLayers Boolean matrix, zeros.
- outputConnect - 1-by-numLayers Boolean vector, zeros.

The network starting point is a class of functions without any input/ouput 3.11.

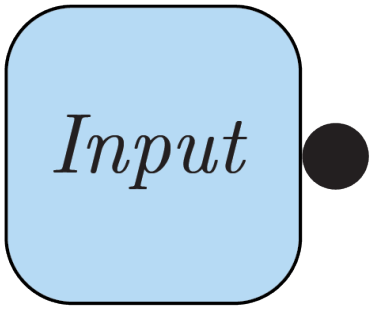

Figure 3.11 Functional programming of the bio-inspired architecture: create the first node.

```
net = network;
net.name = 'BioNN';
```

Next, the number of inputs is defined as follow: here, only one input, that will represent a 8 states cell e.g. 8 possible choices between 8 alphanumeric characters (12AX task, or an image, a text). What is defined as “input to the network” is the task variables.

```
net.numInputs = 1;
```

The number of layers is then defined according to some elementary anatomical mapping, e.g. each layer may correspond to large areas following the 10-20 system (in the case of EEG recording) described earlier. Here, the 3 layers will refer to Frontal, Parietal, Temporal (3 regions) and a behavioral component that spatially receive all the signals from the previous layers (output layer). The behavioral layer can be considered as the interface to the real world. In the 12AX task,

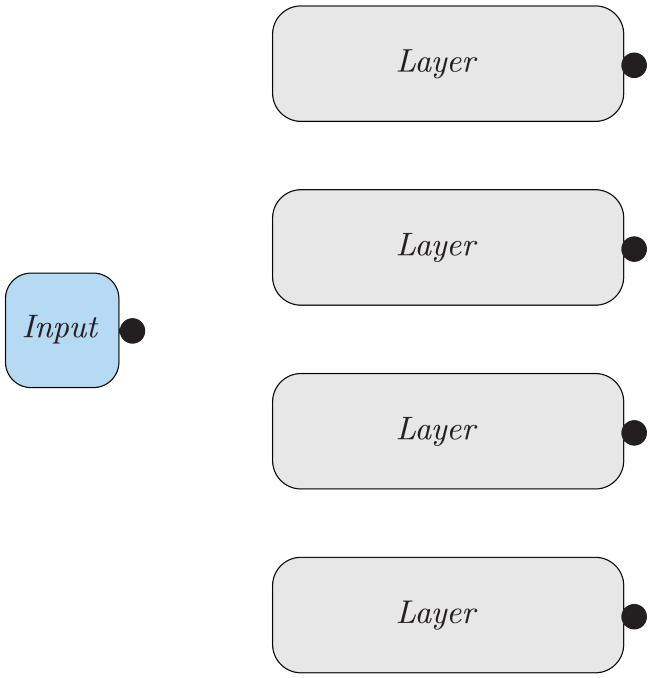

Figure 3.12 Functional programming of the bio-inspired architecture: create the first nodes.

this component registers the user’s decision (left or right keystroke input). The Frontal, Parietal and Temporal layers are the network targets. The network will aim to minimize the error between those targets, for instance raw EEG data, and the produced estimates. Once the network is trained, the targets can be removed; the network will be able to produce artificial outputs, for instance neural data, with only the task variables as input. The network will stimulate its own “neural” data and can be used for other tasks. Fig 3.12

```
layer_count = 4;
net.numLayers = layer_count;
```

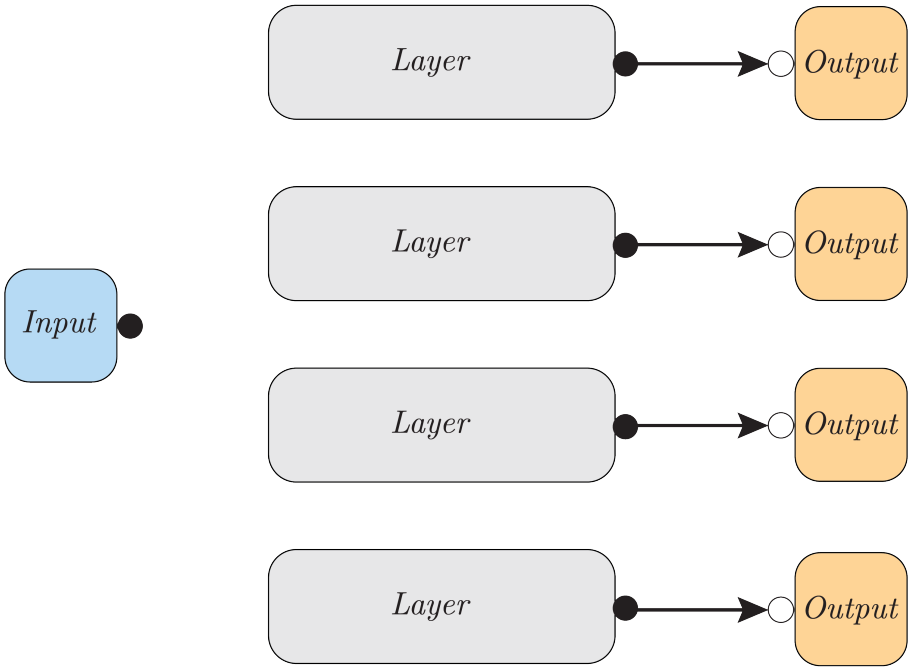

Figure 3.13 Functional programming of the bio-inspired architecture: first connections to outputs.

Define the connections:

Each layer has structurally at least 2 entry points and 1 exit point: 2 for the weights and biases and 1 for the output. The connections between layers, to biases and from outputs, are all defined by 2D Boolean matrices. Fig 3.13.

```
net.biasConnect = ones(layer_count,1);

net.outputConnect = ones(1,layer_count);
```

Connect the layers

Regarding the layers, in rows (matrix first index) are the layer indices for the incoming connection; in column (second index) are the layer indices for the outgoing connection. Fig. 3.14.

```
%Connect Layers From L1 to L2
net.layerConnect(2,1) = 1;

%Connect Layers From L2 to L3
net.layerConnect(3,2) = 1;

%Connect Layers From L3 to L4
net.layerConnect(4,3) = 1;
```

Inputs process functions:

Each function can be customized, depending on the type of task e.g. classification problems (involving binary choices) generally require step-function in the output layer; here, the default functions including in the nn packages are shown for the example. ’mapminmax’ normalize the values between -1 and 1 by default; however, it is generally advised to normalize the input values beforehand (with norm(), rescale(),...) so that the functional boundaries are clearly defined and controlled by the user.

```
net.inputs{:}.processFcns = {'mapminmax'};
```

At present, the feedback input matrix should be considered as a critical piece for a bio-inspired architecture and be set to true. When the signals are fed-back to the layers, some recurrent dynamics can be captured by the network, a property that is associated to task memorization processes. Each hidden state (non-observable

information) at time t is modified by a corresponding weight matrix: when the information contained in those states is looped over the network process, the next hidden state contains partial information regarding the previous states. Besides, the network should be trained in feed-forward fashion, by setting its outputs to ’open’: this step in the network configuration allows the network to be supervised during the training. The targets will be removed in a second step, as explained earlier.

```
net.outputs{:}.feedbackInput = 1;
net.outputs{:}.feedbackMode = 'open';
```

Normalize Input:

```
net.inputs{1}.name = 'Task Variable';
net.inputs{2}.name = 'Frontal Region';
net.inputs{3}.name = 'Parietal Region';
net.inputs{4}.name = 'Temporal Region';
net.inputs{5}.name = 'Behavioral Component';
```

The input preprocesses may be refined if need, although this step is not necessary if the data have been normalized as advised.

```
net.inputs{1}.processParams{1}.ymin = 0;
net.inputs{1}.processParams{1}.ymax = 1;
```

Connect the layers:

The layers can now be interconnected, see Fig. 3.15. In a biological model, each anatomical region “project” to another. Some connections are sparse, other

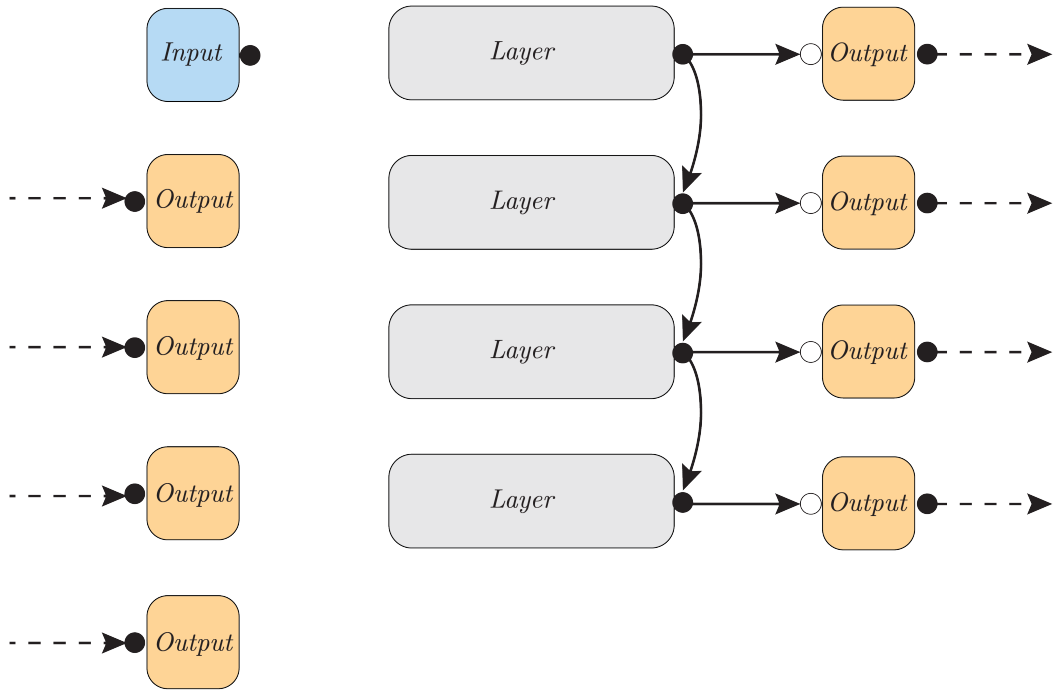

Figure 3.14 Functional programming of the bio-inspired architecture: connect the layers.

are dense. A basic routing can be done via inputConnect().

```
%row=Target-Layer -> col=Input:
%Task Variable = 1,
%Target-Layers:
%Signal 1 = 2, Signal 2 = 3, Signal 3 = 4 and Behavioral = 5
%Layer 1 to Task Variable and Signal 1
net.inputConnect(1,1) = 1;
net.inputConnect(1,2) = 1;

%Layer 2 to Signal 2
net.inputConnect(2,3) = 1;

%Layer 3 to Signal 3
net.inputConnect(3,4) = 1;

%Layer 4 to Behavioral
net.inputConnect(4,5) = 1;
```

Customize delays and weights:

Finally, delays are entered. Feedback delays may follow a specific sequence, for instance Frontal-Temporal/Occipital-Behavioral.

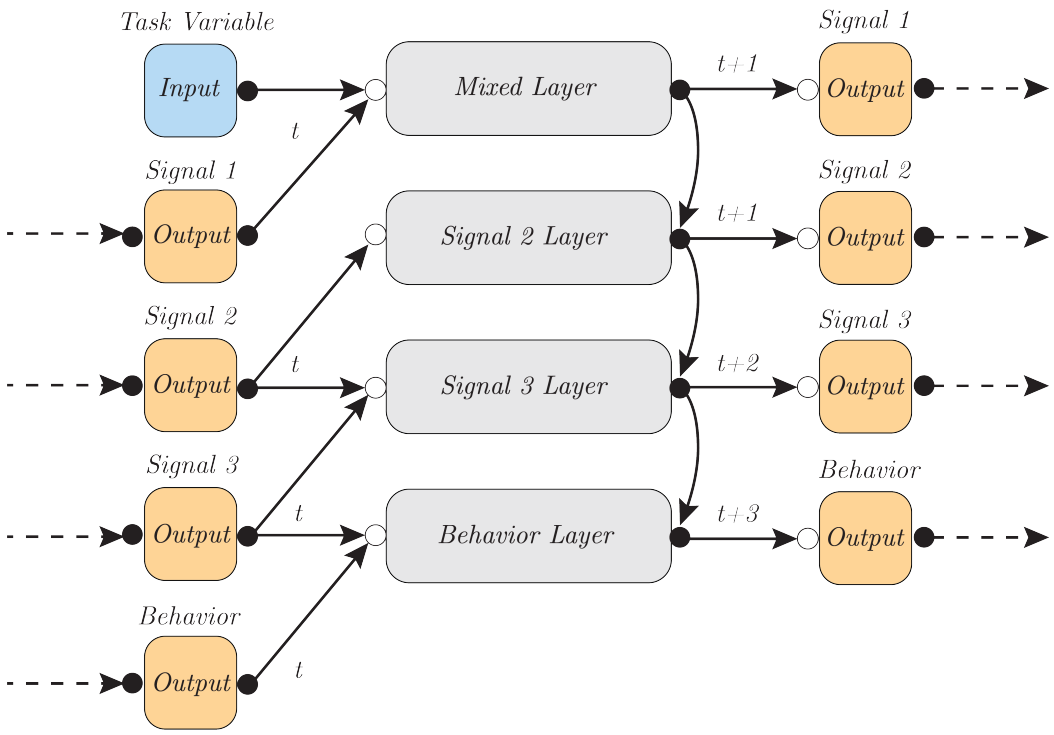

Figure 3.15 Functional programming of the bio-inspired architecture: overview.

```
%net.LayerWeights{2,1}.delays = 1;
%net.LayerWeights{3,2}.delays = 1;
%net.LayerWeights{4,3}.delays = 10;
%net.inputWeights{2,2}.delays = 1;
%net.inputWeights{3,2}.delays = 1;
%net.inputWeights{4,2}.delays = 10;
net.outputs{1}.feedbackDelay = 1;
net.outputs{2}.feedbackDelay = 2;
net.outputs{3}.feedbackDelay = 2;
net.outputs{4}.feedbackDelay = 3;
```

Define the activation functions:

Although there is no formal proof that such functions actually perform complex computations in the brain, activation functions are also critical to pass values from one node to another. Sigmoid can be used to simulate the firing rate of neurons, with the function ceiling as the maximum rate. Above a certain threshold (the function ceiling or asymptote), neuron-units are deactivated. Sigmoid functions output a finer probabilistic result: as weighted values pass through the functions, they follow a logistic distribution, assuring non-binary outputs. The sigmoid, Fig. 3.16, can be defined as:

$$f'_{Sig}(x) = \frac{1}{1 + e^{-x}}$$

ReLU (Rectified Linear Unit, Fig. 3.17) functions are sometimes preferred for a network that requires a faster convergence, as the positive part of the function is updated more rapidly during training due to its linearity:

$$f'_{ReLU}(x) = \begin{cases} 0 & \text{for} \quad x < 0 \\ 1 & \text{for} \quad x \geq 0 \end{cases}$$

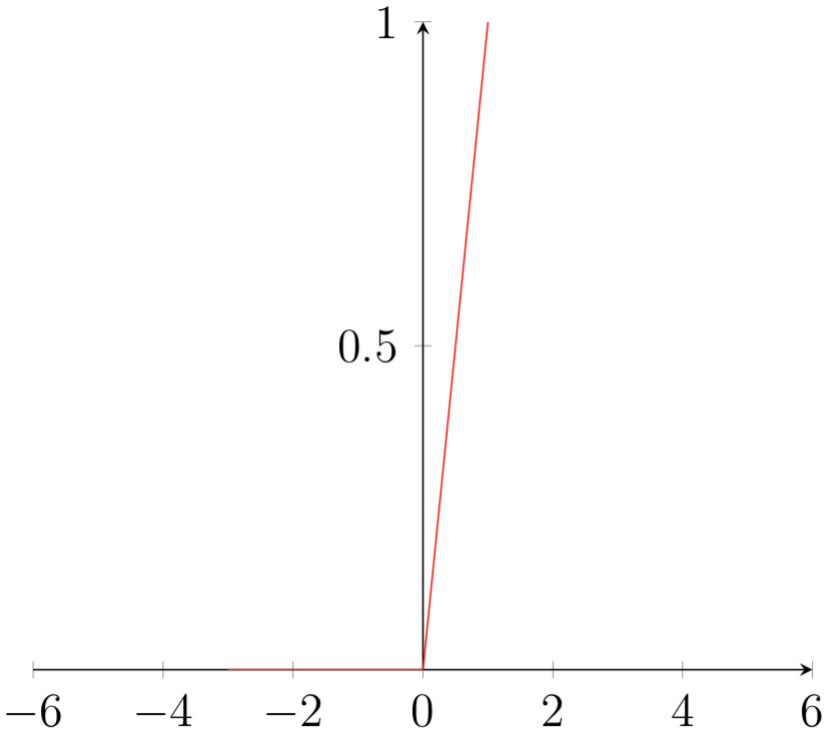

Figure 3.16 Functional programming of the bio-inspired architecture: ReLU activation function.

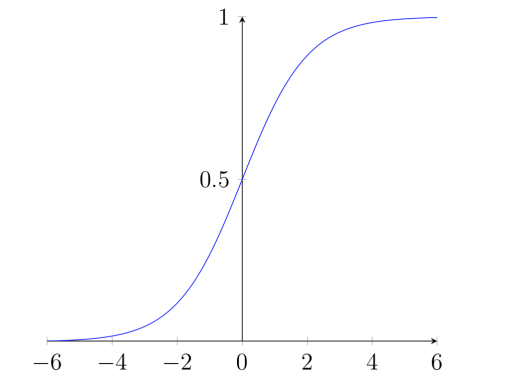

Figure 3.17 Functional programming of the bio-inspired architecture: Sigmoid function.

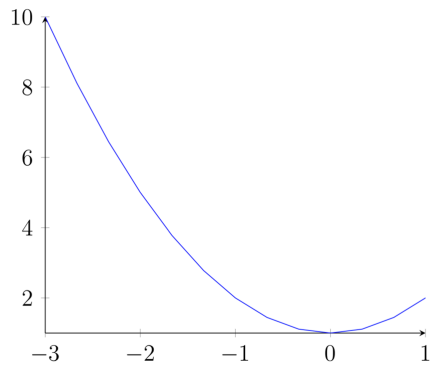

Figure 3.18 Functional programming of the bio-inspired architecture: objective function.

```
net.layers{:}.transferFcn = 'poslin';
net.layers{4}.transferFcn = 'hardlim'; %step function
net.layerWeights{:}.learnFcn = 'learngdm';
```

As seen previously, the objective function, Fig. 3.18, also known as “loss function” or “cost function” (mainly used when referring to the Mean Squared Error, as the MSE is applied to the entire dataset) calculates the error between the outputs and the targets. Here, we apply a cross-entropy function, as it suits well classification problems with probabilistic activation functions in the output layer such as the ’hardlim’ step function:

$$\sum_i H(y_i', y_i)$$

where y_i' is the probability of the estimated value for class i and y_i' is the true probability of the value. In other words, the cross-entropy function measures the minimum average binary encoding size for each data point, when following the true probability distribution and when following the estimated one. The resulted value is an indicator of how good the model is.

```
net.performFcn = 'crossentropy';
net.performParam.regularization = 0.1;
net.performParam.normalization = 'none';
```

An adaptive function may be chosen among those; particularly, the Hebb function, that follows D. Hebb’s theory that “cells that fire together wire together” [8]. This function presents the advantage to link structure and function, a desirable goal for a “neuromorphic” network.

```
%Hebb with decay weight learning rule.
net.adaptFcn = 'learnhd';

%Conscience bias learning function
%net.adaptFcn = 'learncon';

%Gradient descent weight/bias learning function
%net.adaptFcn = 'learngd';

%LVQ 2.1 weight learning function
%net.adaptFcn = 'learnlv2';

%Perceptron weight/bias learning function
%net.adaptFcn = 'learnpn';

%Self-organizing map weight learning function
%net.adaptFcn = 'learnsom';
```

As detailed in the previous chapter, backpropagation is applied, and although it may not have been particularly relevant w.r.t neuroscience literature, some more recent research hints in that direction, for example published in the neuroscience journal Cell this year[9]. The Scaled Conjugate Gradient (SCG) is a fast, robust

gradient descent as it does not perform line search at each iteration [10], but rather use a step scaling mechanism especially suited for supervised learning, and detailed by Møller in his initial paper.

```
%Scaled conjugate gradient backpropagation
net.trainFcn = 'trainscg';

epoch=100; %number of planned training steps
net.trainParam.epochs = epoch;
```

Define the sampling:

The sampling is set as ’time’ since the data are time series.

```
net.divideMode = 'time';
net.divideFcn = 'dividerand';
net.divideParam.trainRatio = 60/100;
net.divideParam.valRatio = 20/100;
net.divideParam.testRatio = 20/100;
```

Define the final plots and diagram:

Finally, the regression values R for the training, validation and testing sets should be plotted. They are considered as a reliable indicator of the network fitting between the outputs and the targets: a value too close to 1 may indicate overfitting (the network may not be able to generalize over unseen data) while a value too close to 0 shows under-fitting (the network did not capture the dynamics of the data).

```
net.plotFcns = {'plotregression'};
```

Diagram Labels:

```
net.layers{1}.name = 'Signal 1 Layer';
net.layers{2}.name = 'Signal 2 Layer';
net.layers{3}.name = 'Signal 3 Layer';
net.layers{4}.name = 'Behavioral Layer';
```

Training and results:

The training is done over the preprocessed Input, Signals and Behavioral dataset.

```
net = init(net);
[net, tr] = train(net,{[Input_cell],...
[Signal1_cell],[Signal2_cell],[Signal3_cell],...
[Behavioral_cell]},,...
{[Signal1_cell],[Signal2_cell],[Signal3_cell],...
[Behavioral_cell]});
save('net','net')
```

The final weight matrices can also be plotted for reference, as plotted on 3.10.

```
view(net)
%Plot Weights
W_i_weight=net.IW;
figure,
imshow(imresize(normalize(W_i_weight{1,2}),[800,800],...
'nearest'),[])
colormap parula
```

Notes

1 -neuro, E.Littré, public domain, https://gallica.bnf.fr/ark:/12148/bpt6k58019485/f254

2 -morphé, E.Littré, public domain, https://gallica.bnf.fr/ark:/12148/bpt6k5460034d/f638. *Littré* is one of the oldest etymology book (1863–77), providing thousands of Ancient Greek and Latin roots. Interestingly, when the last edition was published in 1877, Camillo Golgi had just discovered (1873) a revolutionary silver staining technique to characterize nerve cells, later famously known as the Golgi method, but the wording “neuron” had yet to be invented. The “neuron theory” was established later, in the 1950s, and the term “neuron” became the modern acception for “nerve cells”.

3 “Multi-subject dictionary learning to segment an atlas of brain spontaneous activity”, Varoquaux et al, Information processing in medical imaging 2011, p 562-573

4 “The ADHD-200 Sample is a grassroots initiative, dedicated to accelerating the scientific community’s understanding of the neural basis of ADHD through the implementation of open data-sharing and discovery-based science.” http://fcon$_1$000.$projects.nitrc.org/indi/adhd$200/

5 https://www.nature.com/articles/s41467-018-04316-3

6 https://books.google.co.jp/books/about/Fundamentals$_o f_A rtificial_N eural_N etwor.html?id = Otk32Y3QkxQCredir_e sc = y$

7 https://www.mathworks.com/help/matlab/language-fundamentals.html

8 Donald Hebb in his 1949 book: “The Organization of Behavior”, Hebb, D.O. (1949). New York: Wiley Sons.

9 https://www.cell.com/trends/cognitive-sciences/fulltext/S1364-6613(19)30012-9

10 Neural Networks, Vol. 6, 1993, pp. 525–533

Chapter 4
Neuromorphic Networks: a Use Case

4.1. Design of the Experiment

Regarding the goal of the experiment, the plan was to run a simulation where neurosignals can be used to train a biologically plausible network for task automation. The central hypothesis was that, if artificial neural networks are essentially data-driven constructs and can learn most of the dynamics present in the data, a biologically accurate representation of the data will retrieve the neural dynamics, model the task and therefore may facilitate knowledge transfer by humans using neural recording. To test this, we aimed to design a task-dependent activity and record neural data to train and simulate the bio-inspired neural net.

The electrodes were placed according to the 10–20 system nomenclature (Fig. 4.2), highlighted in the previous chapter. Each electrode is connected on a pin (code in parenthesis) to an Arduino Cython board (16 channels with the Daisy module), which includes an on-board RFDuino radio module connected to the workstation. The signals are collected via UDP at 128Hz.

- Channel 1(N1P) - Fp1 - Pre-frontal (planning, decision-making)
- Channel 2(N2P) - Fp2 - Pre-frontal (planning, decision-making)
- Channel 3(N3P) - C3 - Central (no central lobe per say: partially integrated activity of contiguous lobes, user dependent)
- Channel 4(N4P) - C4 - Central (no central lobe per say: partially integrated activity of contiguous lobes, user dependent)
- Channel 5(N5P) - P7 - SIMPRED (language processing)

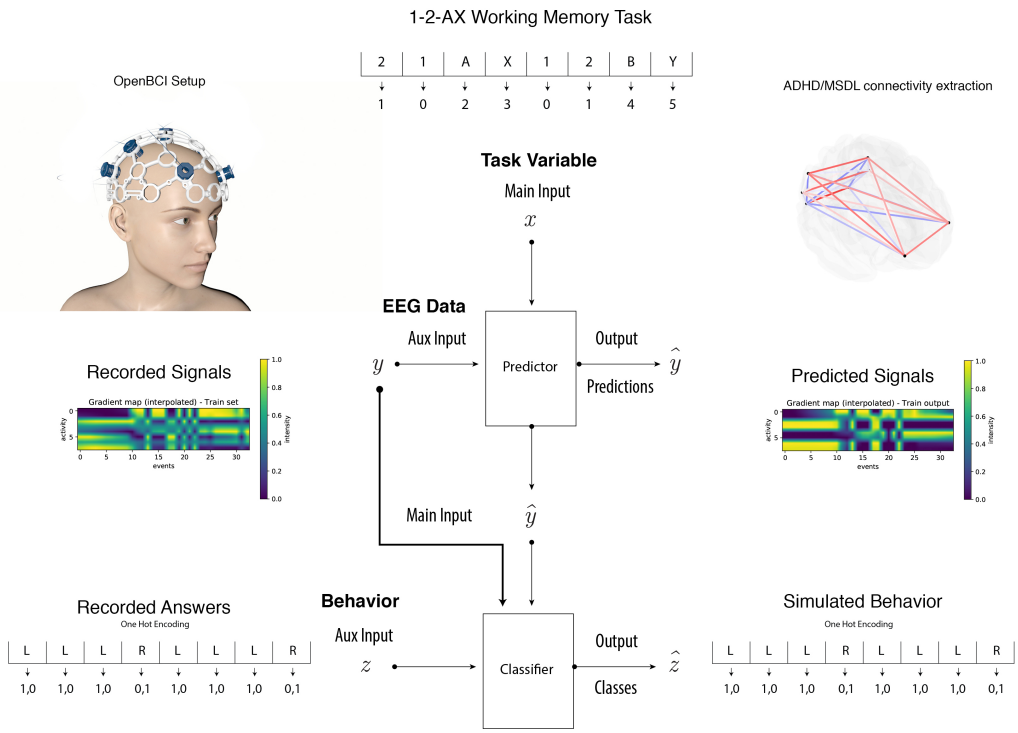

Figure 4.1 EEG-based neural interface and machine learning algorithm for the 12AX task.

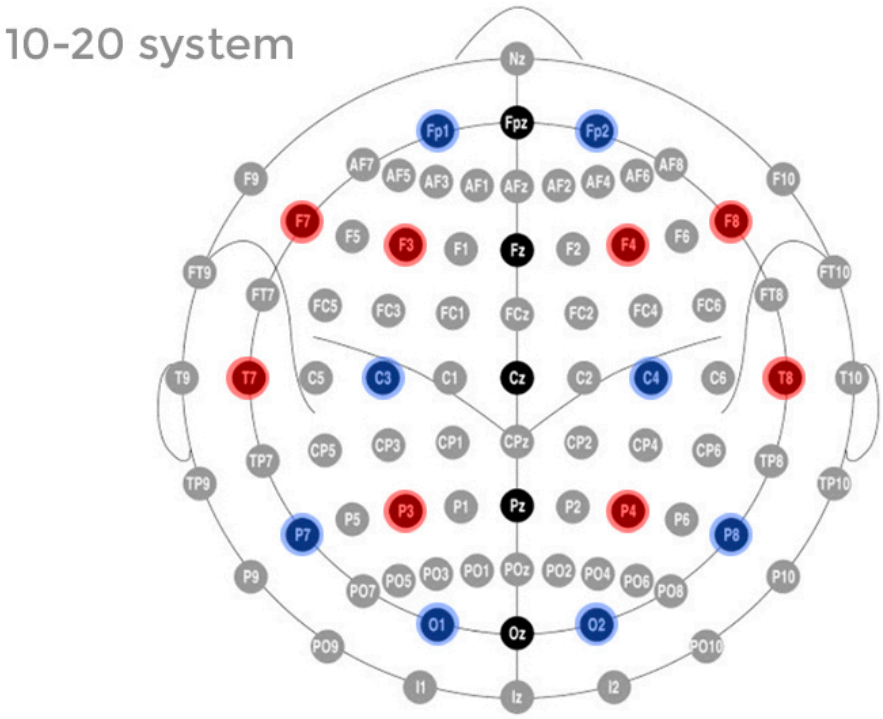

Figure 4.2 The 10-20 system, describing the positioning of EEG electrodes relative to brain anatomical regions. Image source: OpenBCI (wwww.openbci.com).

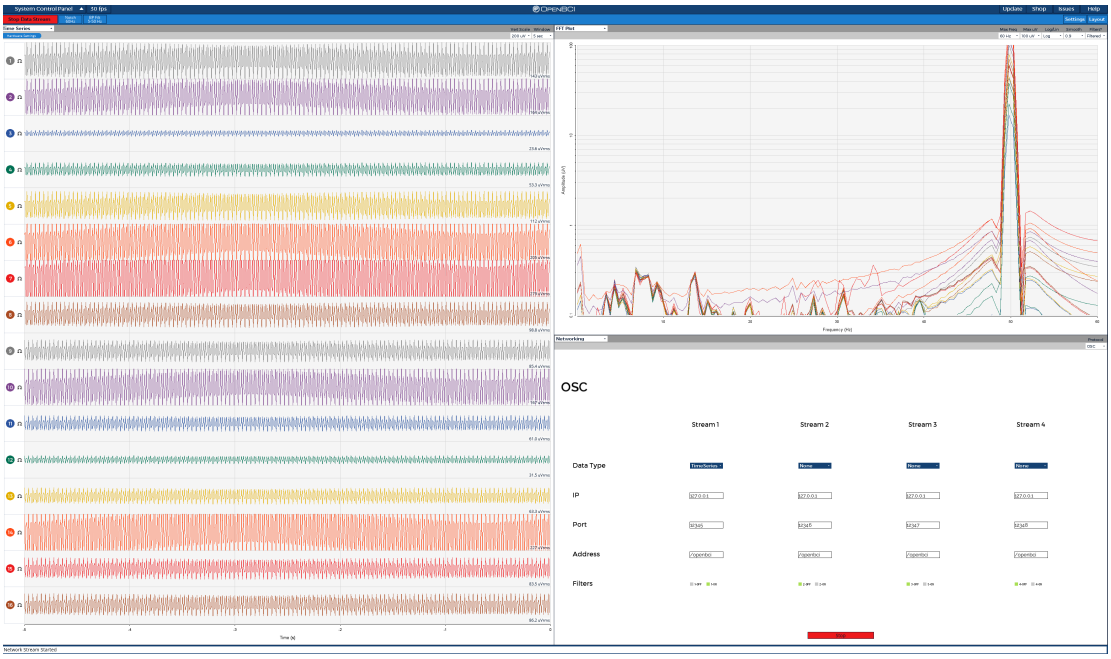

Figure 4.3 The OpenBCI Interface.

- Channel 6(N6P) - P8 - SIMPRED (language processing)
- Channel 7(N7P) - O1 - Occipital (visual processing)
- Channel 8(N8P) - O2 - Occipital (visual processing)
- Channel 9(BN1P) - F7 - Frontal (action, body control such as ocular movement, speech)
- Channel 10(BN2P) - F8 - Frontal (action, body control such as ocular movement, speech)
- Channel 11(BN3P) - F3 - Frontal (action, body control such as ocular movement, speech)
- Channel 12(BN4P) - F4 - Frontal (action, body control such as ocular movement, speech)
- Channel 13(BN5P) - T7 - Temporal (visual memory, emotion associations)

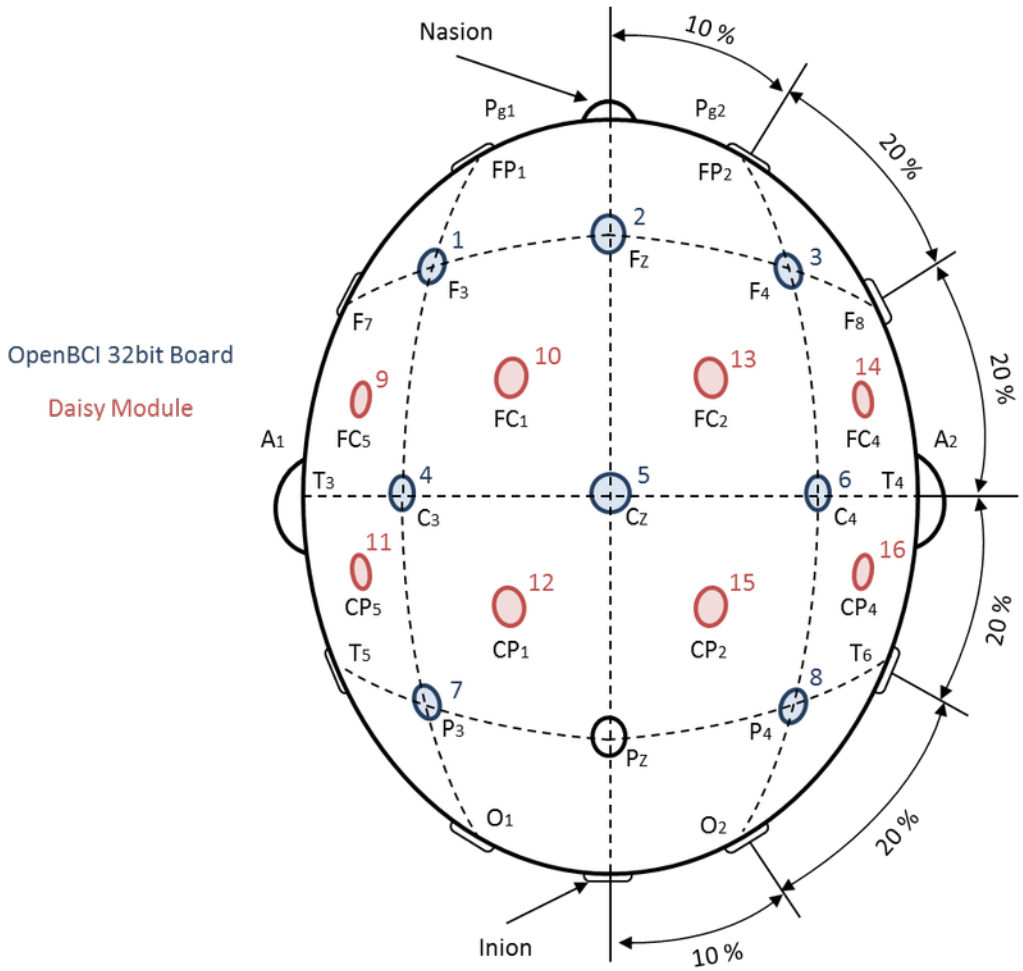

Figure 4.4 The OpenBCI electrodes placement (image source: www.openbci.com).

- Channel 14(BN6P) - T8 - Temporal (visual memory, emotion associations)
- Channel 15(BN7P) - P3 - SIMPRED (language processing)
- Channel 16(BN8P) - P4 - SIMPRED (language processing)

Here, the preferred method for recording was EEG for its non-invasiveness and accessibility. In that, the OpenBCI toolkit represented a reasonable, portable solution, being open-source, easily configurable and a relatively inexpensive yet seemingly reliable EEG recording device. Alternative tests may be conducted in

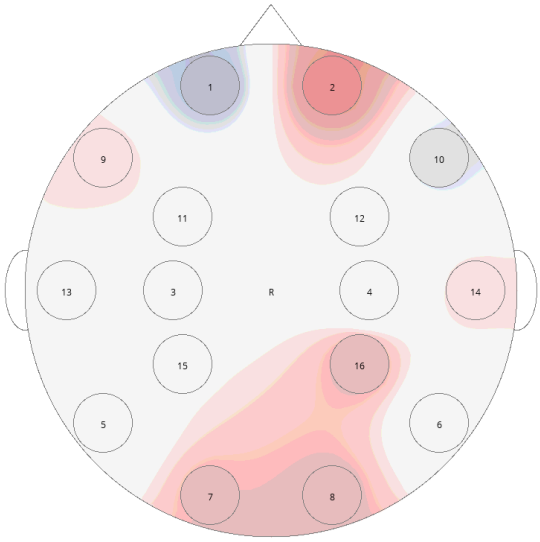

Figure 4.5 Sample of the heat map of the active signals while performing the task. Each circle represents an electrode; warmer color indicates higher signal intensity.

the future on a highly reliable and reputable source of neurosignal data, such as vim-1 (Lescroart et al. 2011), a fMRI dataset of human visual areas in response to natural images. However, EEG-recording was the preferred setup for this experiment, as it allows to freely design the experiment, as opposed to using a database with predefined experimental conditions. An intermediate solution was to use fNIR, a near-infrared spectroscopy system, recording oxygen concentration levels in the brain, and presenting the same temporal accuracy as EEG but with a better spatial resolution, similar in that to fMRI. A peak in the amount of scientific publications using fNIR imaging has surged in recent years, along with the release of commercial, portable versions, making the technology more attractive.

4.2. Method: BioNN, Structure and Function

As described in length in the previous chapter, a neuromorphic network "BioNN" has been designed, here using Python's Keras functional API to Tensorflow [1] for the implementation part, so as to keep it free and open-source. Keras is a powerful deep learning library, specifically designed for research. The neuromorphic network requirements were structural and functional. Here, the structure refers to the spatial connectivity between different regions. The functions of each region are determined by the functional connectivity between those same regions. In Fig. 4.6, a selected connectivity map, based on the MSDL dataset, is shown connecting eight different regions structurally: Frontal left and right, Parietal left and right, Occipital left and right, Default Mode Network left and right. Additionally, the eight rows and columns of the corresponding matrix Fig. 4.7 represent the functional connectivity of each mapped region. As explained in the previous chapter, this map and this matrix helped define the specifications of a biologically plausible network. Functionally, the correlation matrix from the ADHD atlas was used for the sub-losses weights between each layer in the neural net. Structurally, the connectivity map from the MSDL brain atlas, as shown in diagram Fig. 4.8, was used for the layers arrangement. It also helped as a reference point for the OpenBCI electrodes placement, which follows the 10-20 system (Fig. 4.2).

Regarding the design of the circuit, the neural net is a compound of two networks, one network consists of a predictor and the other network acts as a classifier. A similar predictor network architecture was used for the experiment on prediction model for thermo-haptic feedback. The main difference is that there is no rewarding system in this design, as supervised learning is preferred for classification and regression, as opposed to clustering or learning a policy (unsupervised learning). The dual design approach was strongly influenced by my research work on NARX for body motion prediction in Virtual Reality, with a series-parallel architecture for the training and a closed network for prediction. Besides, the NARX model, a multilayer perceptron, served as the basis to build the BioNN classifier.

In this example, one input (the task-variable) and eight targets (from the eight OpenBCI channels) were passed through the predictor, i.e. through eight interconnected LSTM layers. Each layer receives a channel signal as an auxiliary input

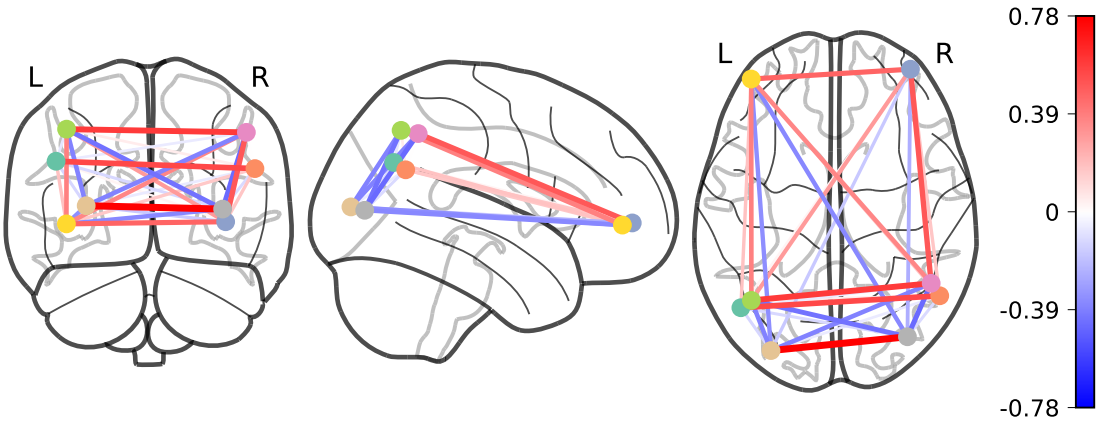

Figure 4.6 Connectivity mapping for 8 selectivity regions of the brain, based on the correlation matrix from the functional ADHD dataset. Labeling of the regions was extracted from the MSDL data set.

(i.e. as a target). Each channel refers to a specific region: for instance, a visual processing region, a language processing region. The predictor computes the task variables (1,2,A,X,B or Y) and the EEG signals, and generates new signals as an output. A second network, the classifier, is then trained, by computing the observed variables, that is: left or right answers ((1,0) or (0,1) in one hot encoding), and the corresponding signals. Importantly, the signals that were targets in the predictor are now inputs in the classifier. This transitive design allows the signals to be passed from one network to the other. As a result, the trained model only requires a task variable as an input, and can generate a simulated behavior (left or right answer) without any additional data. Synthetic signals are generated by the first network and passed though the second network to generate a choice based on the initial task variable. ReLU and Sigmoid (in the output layer) are used for the activation functions, as it is classically done.

4.3. Experimental Results

To implement this architecture in the EEG setup, the plan was to train and simulate a network on some elementary computations involved in memory tasks,

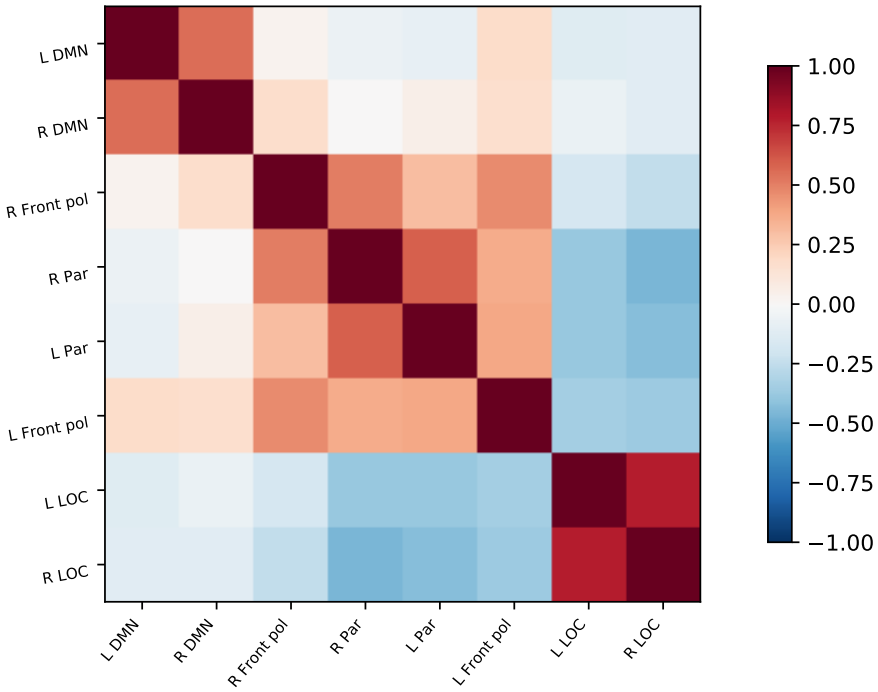

Figure 4.7 Correlation matrix from the functional ADHD dataset with the atlas data from MSDL for the labelling.

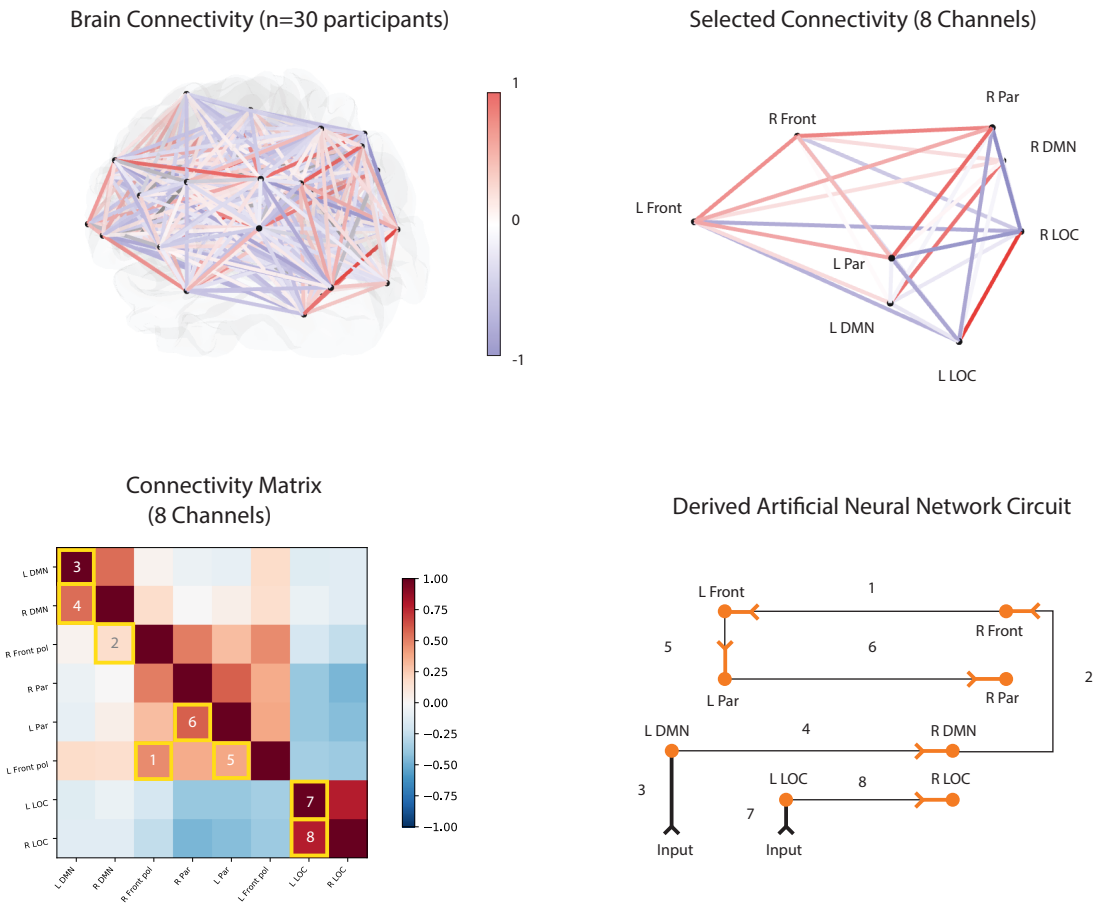

Figure 4.8 Design of the BioNN network. Top left quadrant: brain connectivity mapping over 30 participants. Top right: selected connectivity corresponding to 8 OpenBCI channels. Bottom left: extracted connectivity matrix. Bottom right: neural network circuit derived from the brain connectivity mapping.

such as the 1-2-AX working memory test described by P. Dayan (2008), O'Reilly (2006). This relatively recent test has been used with different neural networks in the literature and represent a relevant choice for comparing with base-line architectures. It can be described as a simple algorithm to solve for humans, but a notoriously difficult one for neural networks. To my knowledge, it has never been tested with a biologically plausible neural network trained on the neural traces acquired from a human subject performing the task.

```
#Python 3
def nextOutput(nextInput):
    global lastNum, lastLetter
    if nextInput in ["1", "2"]:
        lastNum = nextInput
        lastLetter = ""
        return "L"
    elif nextInput in ["A", "B"]:
        lastLetter = nextInput
        return "L"
    elif nextInput in ["X" , "Y"]:
        seq = lastNum + lastLetter + nextInput
        lastLetter = nextInput
        if seq in ["1AX","2BY"]:
            return "R"
        if seq not in ["1AX","2BY"]:
            return "L"
    return None
```

The algorithm above can be briefly described as follow: given a random sequence composed of letters 1,2,A,B,X,Y if the last numeral is 1, the target sequence is AX and L should be returned, if it is 2, the sequence is BY and L should be returned, otherwise R should be returned; only the last number and last letter count. For example, the sequence "21AAXBYAX" returns "LLLLRLLLR". The number of

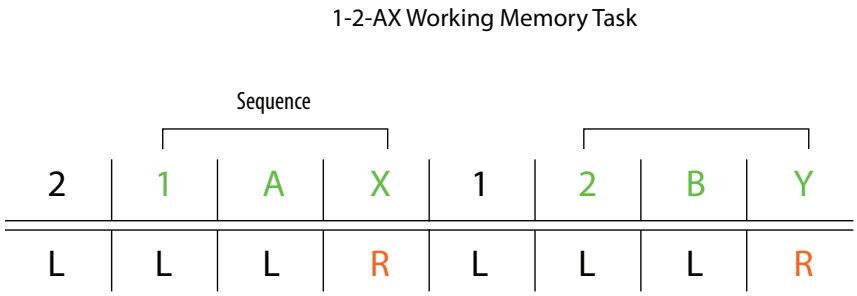

Figure 4.9 The 1-2-AX working memory task.

combinations for a specific number of alphanumeric characters (sequence length) was determined before testing the task, this way we may test all the combinations, for a reasonable length. As wrote earlier, this task is a non-trivial problem for artificial neural networks, even with a long training, but a trivial one for humans, with minimum training. Resolution of this problem have already been successfully demonstrated, initially in 2008 using Gabor wavelet functions (Kay, 2008) or more recently with state-of-the-art Generative Adversarial Networks (StYves, 2018, not peer-reviewed), but not directly with neural data.

EEG recording was done over n=12 participants (8 males, 4 females): each participant was instructed about the 1-2-AX and a trial test was given. Once completed, the recorded session started. A sequence was displayed on a computer screen and each participant could answer by clicking left or right, with 10 sequences (of variable lengths) in total. On average, the participants answered correctly 94.38% of the time, with 3 participants yielding a perfect score (100%). As shown in Fig. 4.10, each participant presents different neural patterns while

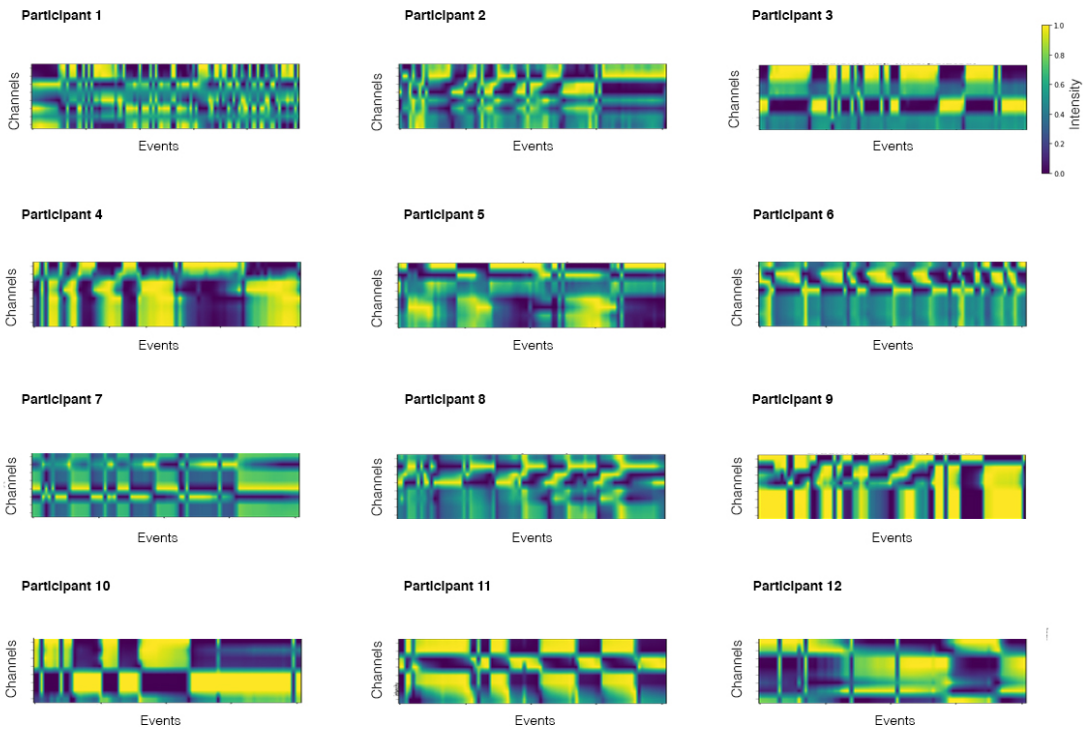

Figure 4.10 Recorded EEG signals for n=12 participants.

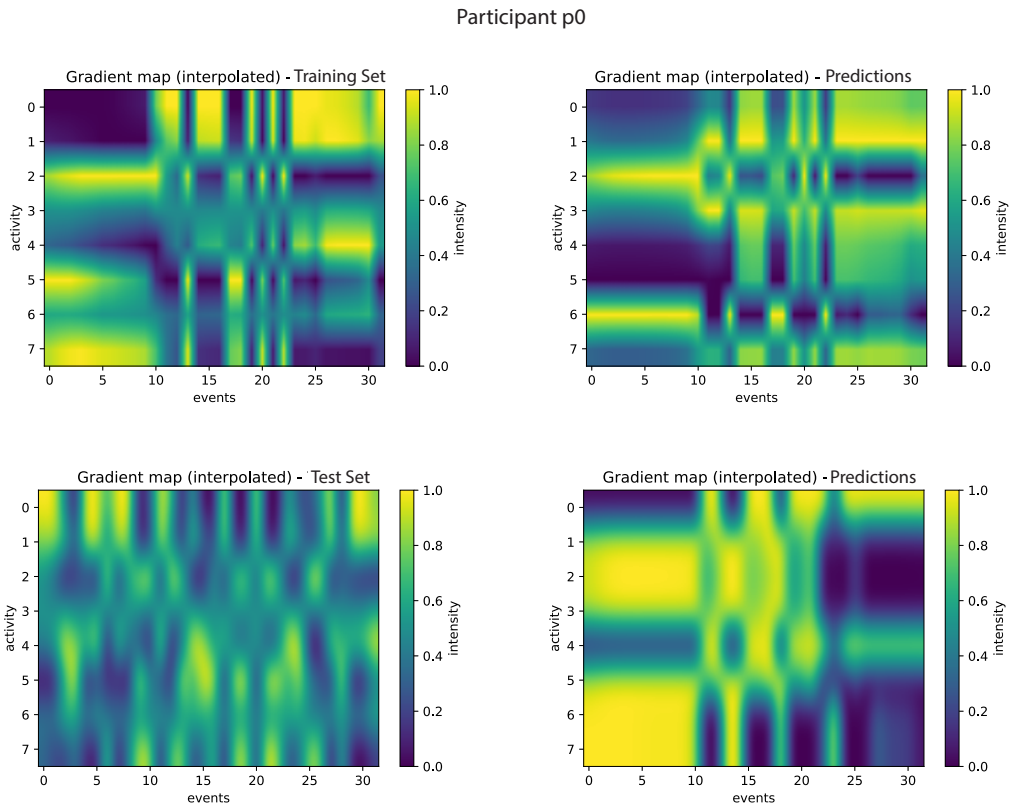

Figure 4.11 Pictured on the left column, the recorded EEG signals. On the right, the predicted signals.

performing the same memory task.

Based on our data collection, an important result is that we could simulate an artificial network that can retrieve and predict some of the non-linear dynamics present in the data, as shown in Fig. 4.11. On the training set, the signals outputs of the neuromorphic network could explain on average 33.65% (and as high as 70.21% and 70.41% for participant 1 and participant 10) of the neural activity recorded on the OpenBCI. The highest explained variance recorded on a training session for a specific channel was on channel 3 (Default Mode Network) with participant 1 (98.31%, average over all participants: 64.35%) and should be

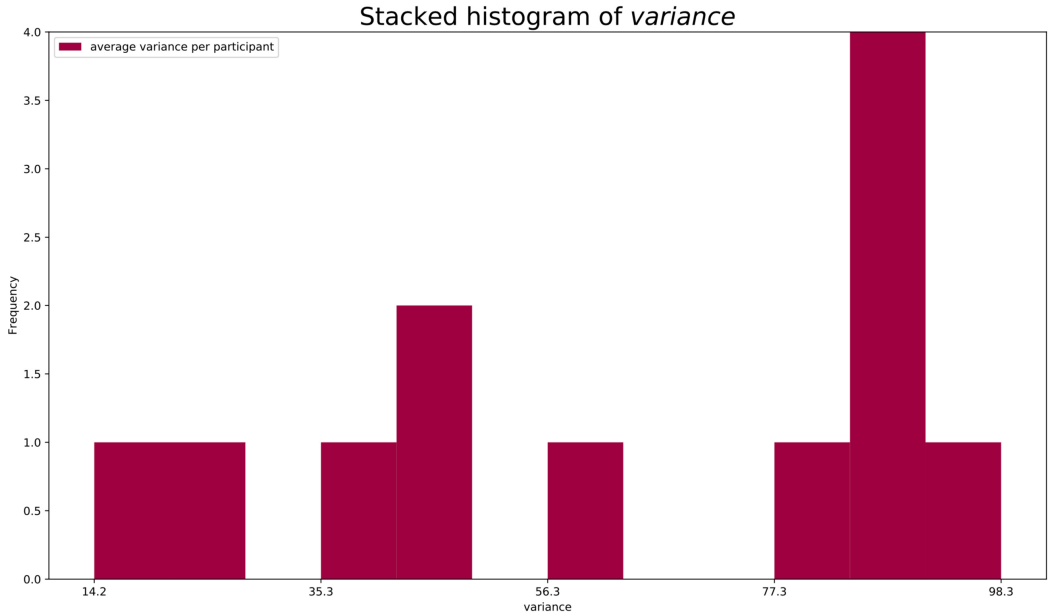

Figure 4.12 Stacked histogram of the highest variance for a specific channel (1 channel), n participants = 12, range 14.24%-98.31%.

considered an outlier: this set translated to an overfitting and a lower predictive power on this participant data overall. However, participants 3, 4, 5 and 9, 10 had also good scores on the training set (highest:48.92-88.05%, average: 28.26-70.41%) and relatively higher predictive power on the test set as well.

Fig. 4.11 shows conclusive results: the neuromorphic network did retrieve a significant part of the activation despite the variability of the neural data between participants, as a visual inspection of the matrices side-by-side, user generated and artificially generated, can confirm. Note that those matrices represent the activity of the Frontal (left, right, involved in planning, decision-making), Parietal (left, right, involved in language processing), Occipital (left, right, visual processing) regions and DMN (Default Mode Network, widespread brain regions with functional connectivity). By designing an experiment where EEG (electroencephalograms) signals could be used to train the network, the network, which is interpretable both functionally and structurally, could in return provide interesting insights

about each participants neural dynamics.

Additionally, I then tested the trained network in prediction, without providing any EEG signals. The only input was the alphanumeric sequence, for instance: ['2', '1', 'A', 'A', 'Y', 'A', 'X', '1', 'A', 'A']

This sequence was never part of the training session. Remarkably, the closed-network produced similar auto-generated EEG pattern, when comparing to the existing user's EEG signals (also not used in the training session).

The synthetic signals that were produced, visible on the right column (predictions) in Fig. 4.11 are pure constructs of the network with the 12AX sequence as the only input. Apart from those performance tests, the EEG experiment served as a point of comparison for a qualitative assessment. Indeed, contrary to classical supervised learning, where the targets are essentially the correct answers to output, the bio-inspired neural network also had to search the spatial and frequency space of the neural data to output the correct signals. Because the network was capable of partially modeling the neural dynamics, if there exist strong correlations of patterns between the neural data provided by the EEG and the correctness of the answers, the neural net may find, in a future work, the optimum solution to the task itself, formulated as a random sequence of letters with a hidden algorithmic solution.

4.4. Neuromorphic Network Evaluation

Benchmark for the BioNN architecture shown in Fig 4.15 was performed against a standard LSTM network (as represented in Fig 4.16): for fair comparison, they share the same input/target structure, with the task variables as inputs, the signals as targets/inputs and the observed variables (behavior) as outputs. The observed variables were formatted with one hot encoding, so that the categorical variables could be represented as binary vectors (e.g. [1,0],[0,1] for Left, Right) and categorical cross-entropy could be applied as the objective function for the classifier. In order to train over the signals as auxiliary targets, LSTM layers are ordered in parallel, each line computing a signal. The total amount of parameters for this LSTM-based network is 2,173,000 as the summary below reveals.

In comparison, BioNN has 1 order of magnitude less parameters, here 271,950.

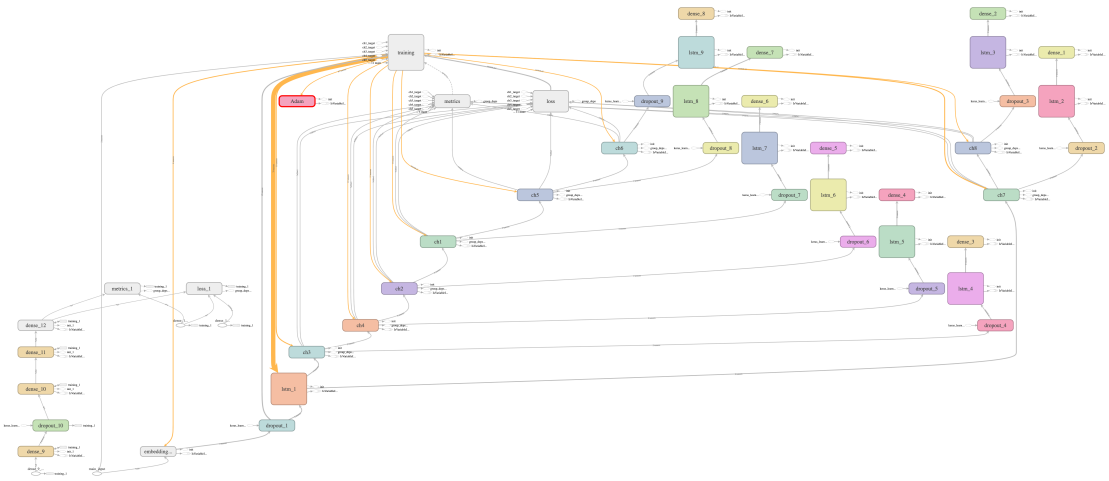

Figure 4.13 Tensorflow diagram of the first part of the BioNN network (LSTM-based predictor).

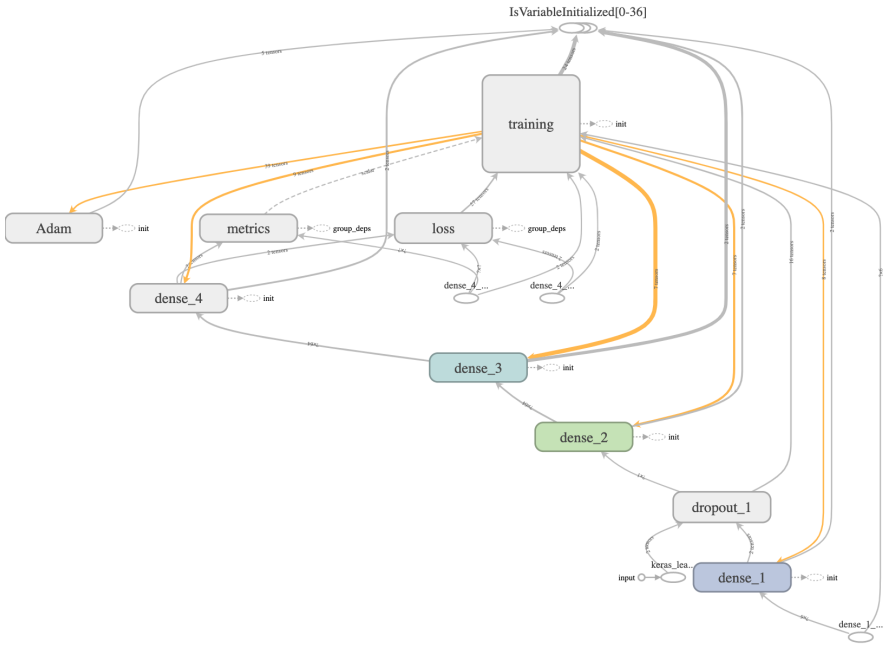

Figure 4.14 Tensorflow diagram of the second part of the BioNN network (classifier).

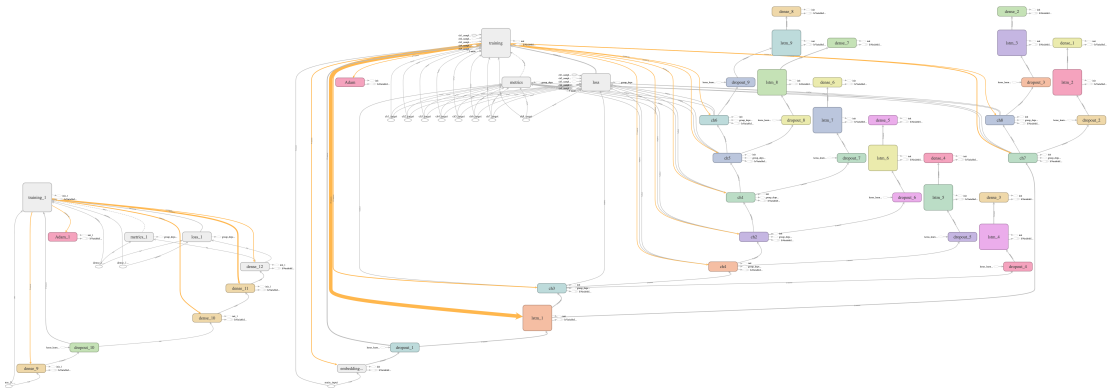

Figure 4.15 Tensorflow diagram of the BioNN network, complete with the LSTM-based predictor and the sequential classifier.

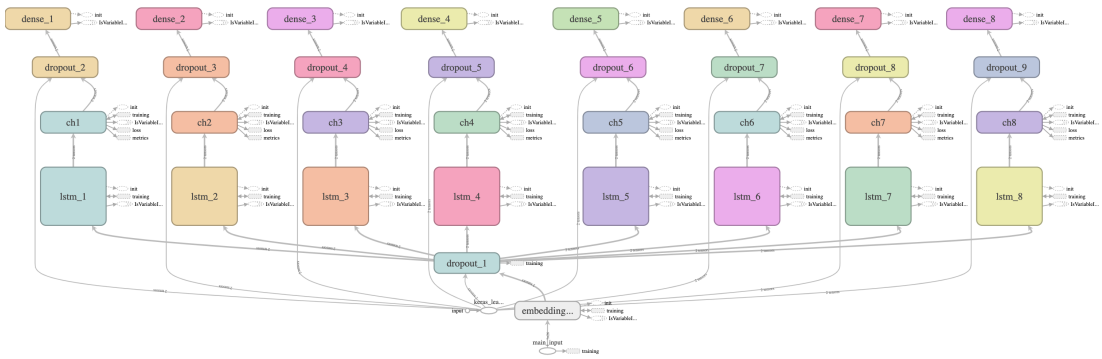

Figure 4.16 Tensorflow diagram of the LSTM network, for benchmarking.

LSTM		
Layer (type)	Parameters	Connected to
Main input	0	
Embedding	64	Main input
Dropout	0	Embedding
LSTM 1	271360	Dropout
LSTM 2	271360	Dropout
LSTM 3	271360	Dropout
LSTM 4	271360	Dropout
LSTM 5	271360	Dropout
LSTM 6	271360	Dropout
LSTM 7	271360	Dropout
LSTM 8	271360	Dropout
Channel 1	257	LSTM 1
Channel 2	257	LSTM 2
Channel 3	257	LSTM 3
Channel 4	257	LSTM 4
Channel 5	257	LSTM 5
Channel 6	257	LSTM 6
Channel 7	257	LSTM 7
Channel 8	257	LSTM 8
Total params: 2,173,000		
Trainable params: 2,173,000		
Non-trainable params: 0		

Table 4.1 LSTM Parameters and Connections Summary.

BioNN		
Layer (type)	Parameters	Connected to
Main input	0	
Embedding	64	Main input
Dropout	0	Embedding
Input	271360	Dropout
Channel 3	257	Input
Channel 4	2	Channel 3
Channel 2	2	Channel 4
Channel 1	2	Channel 2
Channel 5	2	Channel 1
Channel 7	257	input
Channel 6	2	Channel 5
Channel 8	2	Channel 7
Total params: 271,950		
Trainable params: 271,950		
Non-trainable params: 0		

Table 4.2 BioNN Parameters and Connections Summary.

This allows a much faster training time compared to a conventional LSTM. Regarding the losses, Fig 4.17 indicates the backpropagated error between the target tensors and the output tensors, the lower the error the better the network performs. BioNN losses, in blue, decrease in validation for a majority of channels, suggesting that overfitting is less likely to occur, whereas the LSTM validation losses increase in most cases. Indeed, the interconnection of the BioNN layers reducing the number of parameters, the backpropagation of the error is computationally less expensive, and less epochs are necessary to decrease the cost of the objective function. Besides, relatively low loss weights are assigned to each layer, in the range 0.17-1: as explained earlier, those coefficients were obtained from the extracted connectivity on the ADHD data set; they reflect the contribution of each region to the others. As the BioNN layers are connected in similar fashion, it seems plausible that the contribution of the sub-losses relative to each layer helps optimize the network.

As stated by François Chollet in "Deep Learning with Python"[2]: "The mean squared error (MSE) loss used for the age-regression task typically takes a value around 3–5, whereas the crossentropy loss used for the gender-classification task can be as low as 0.1. In such a situation, to balance the contribution of the different losses, you can assign a weight of 10 to the crossentropy loss and a weight of 0.25 to the MSE loss." Here, the MAE objective function (Mean Absolute Error, a variant of Mean Squared Error which presents the advantage of being outlier tolerant) was applied to the predictor, while crossentropy was applied to the classifier. Since the training is done in two-fold, the losses of the classifier do not impact the predictor's hidden layers weights.

Notes

1 "Keras is a high-level neural networks API, written in Python and capable of running on top of TensorFlow, CNTK, or Theano. It was developed with a focus on enabling fast experimentation. Being able to go from idea to result with the least possible delay is key to doing good research." https://keras.io

2 "Deep Learning with Python, author: François Chollet, November 2017, ISBN 9781617294433"

Figure 4.17 Validation graphs for the BioNN and LSTM. BioNN losses, in blue, decrease in validation for a majority of channels, suggesting that overfitting is less likely to occur, whereas the LSTM validation losses increase in most cases.

Chapter 5
Evaluations

Different neuromorphic architectures can be designed for different applications. Whether they are rewarding models or feedback loop, recurrent models, if we acknowledge that similar processes occur in the brain, and that those networks implement some of the neural dynamics features, we can also use them to predict the hysteresis of a device or non-linear human behaviors. In essence, we can use neuromorphic networks to augment our perceptions. In this chapter, I advance the hypothesis that neuromorphic artificial nets cannot be achieved without the notion of embodiment and physical interactions. Specifically, embodiment should be defined here as the implementation of artificial neural networks in hardware. Training embodied artificial neural networks implies that they will interact with our environment. Not taking into account our environment may result in a "vat" model, with lower predictive power. Conversely, physical interactions (environmental parameters, behavioral components, physiological signals) may be used as a feedback loop for the training. The main motivation is the following: as we have seen, the field of machine learning provides powerful methods to implement functional computations, and artificial neural networks particularly, can learn any function. If we ever wanted to build a type of intelligence that is fully compatible with the perception we have of our environment, it seems sensible, then, to build biologically valid models that are fully immersed into the physical world.
In the following sections, I will detail three setup, in virtual reality, haptic force feedback and thermal feedback, that highlight the notions of feedback control and prediction by supervised learning networks and reinforcement learning networks. Each of this setup represents a specific perceptual modality: Visual perception in VR, Pressure detection and Thermoception with haptic devices. Regarding visual information processing, an overwhelming amount of publications has already been produced in computer sciences, physiology, neurosciences. However,

cognitive augmentation through agency in virtual reality spaces represents a novel approach. Virtual reality systems (head-mounted displays, control and tracking) have gained significant improvements only recently, and learning algorithms can now be implemented in such systems. Regarding haptic feedback, it also represents an interesting field to investigate, as there are not so many devices that can accurately reproduce the sense of touch (Minamizawa et al. 2012): tactile data are difficult to acquire; texture, weight or pressure can be subjective.

Many other types of perception (auditive, olfactive, gustative, internal, to mention only a few) could be further studied, but this body of work will focus on very specific perceptual properties to build neural nets, rather than presenting an exhaustive list of possible implementations.

5.1. Prediction Model in Virtual Reality

Predictive models could possibly help the user compensate for the reaction time or prevent motion sickness, extending the range of our cognitive abilities. Preliminary research was conducted at Sony CSL with Dr Shunichi Kasahara (Kasahara and Rekimoto 2014), interested in exploring human cognition and the concept of body ownership in VR. Although recurrent neural networks (RNNs) for data modeling have been extensively used, the implementation of RNNs for tracking complex behaviors in a VR environment remains under-investigated. The application of my work was done over the decision-making process involved in trajectory predictions, and how neural nets could anticipate such decisions: body movements in particular, and the large set of motions that can be performed, can be described as a nonlinear, dynamical system with uncertain parameters. Artificial RNNs have traditionally been used for time series prediction (Xie et al. 2009) where parametric uncertainty is part of the model and non-trivial. However, accurately predicting the future behavior of such a system requires learning long-term dependencies: a difficult problem for standard RNNs (Martens and Sutskever 2011), as the gradient descent algorithm used in those networks tends to “vanish” with time (Hochreiter 1991). In addition, forecasting methods require stationarity where most of the real-time series are non-stationary.

The architectural approach we chose was based on a nonlinear autoregressive

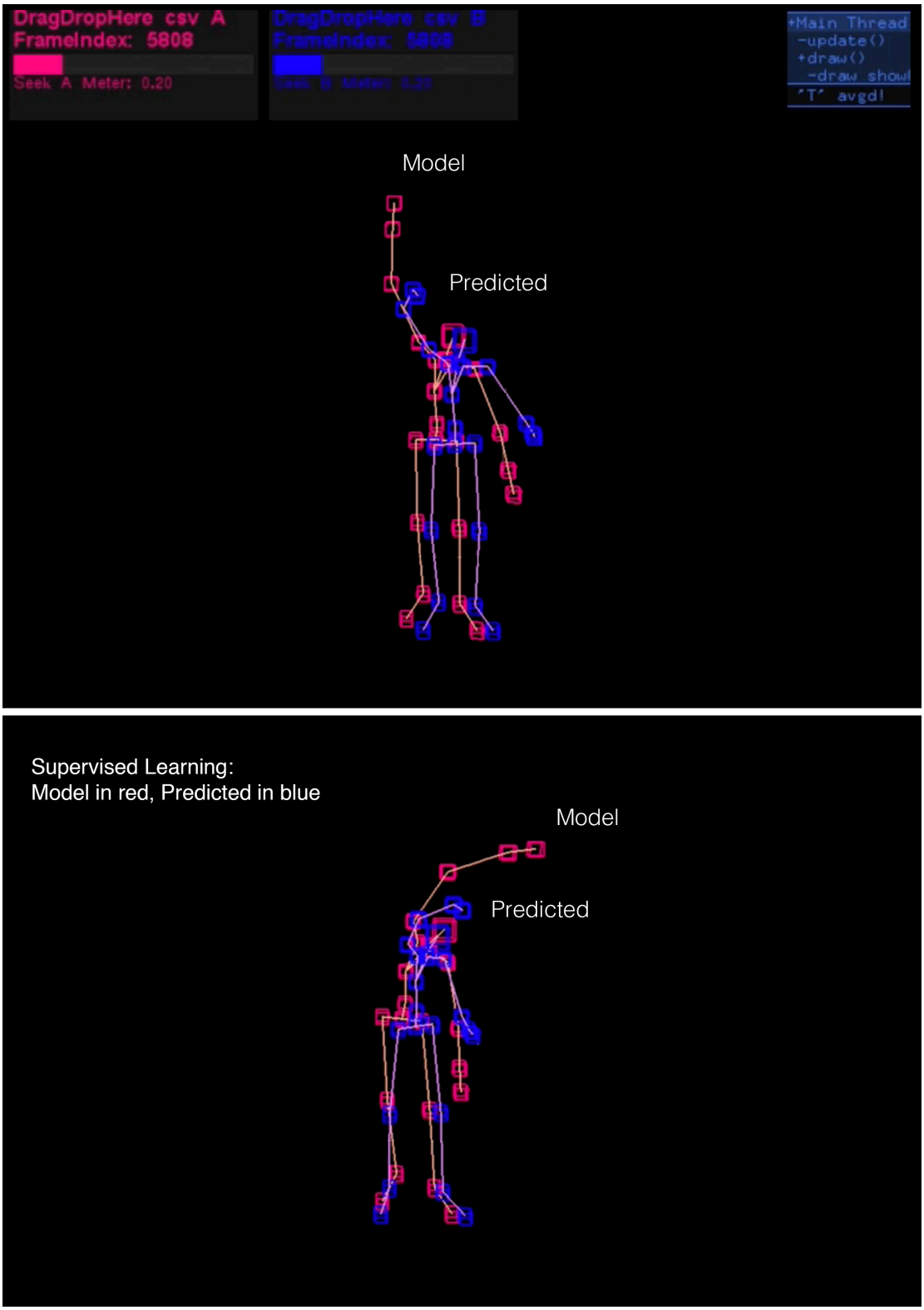

Figure 5.1 ExoBrain testing in collaboration with Dr Shunichi Kasahara using his RAMActor setup.

exogenous (NARX) model: essentially a Multilayer Perceptron that takes its past outputs as inputs instead of estimated ones (Fig. 5.3[1]). The main advantage of this architecture, compared to a standard RNN, was to achieve a much faster convergence, an important characteristic for real-time processing. Another advantage, regarding the non-linearity, relied on the network capability to learn long-term dependencies between the time series components. Finally, the possibility to cope with non-stationarity and therefore process real-time data with a good performance made this model particularly well-suited to our approach. We used the NARX network as a nonlinear tool for successfully predicting human motions, both in batch processing and real-time. In the experiments we conducted, the apparatus consisted in a full-body suit with motion capture markers placed around 23 articulation points and an optical system tracking their tridimensional position in real-time. The device transfers the movements to our processing unit via UDP in the short 20-50ms range. The stimuli were either physical or virtual objects presented to the actor; the tests consisted in grasping, avoiding or catching those as fast as possible. In order to validate our RNN model, the first tests were done in batch processing. Both the reflex movement and its predicted trajectory were classified as, respectively, reflex and anticipated movements; the plotted sequence of anticipated movements and the reflex one were superimposed; we then monitored the accuracy of our predictions over time.

Eventually, the performance of the RNNs was evaluated on the number of false positives/negatives, and we improved both the apparatus and the RNN, until an acceptable margin of error was achieved.
In order to avoid overfitting, the cross-validation was done by randomly dividing the dataset between a training, a validation, and a test set: the splitting was found (by trial-and-error) most effective at, respectively, 70%, 15%, and 15%. We used the Levenberg-Marquardt algorithm for the training part, as it converges quickly. The results show a good performance: the final mean squared error was small as shown in the prediction plot. Also, overfitting did not occur as the test set error and the validation set error had similar characteristics. For the real-time processing, the model was optimized. The method was very similar to batch processing, only the mode of acquisition changed. We processed the data within 100-200ms, before the motor response to the stimulus reached the actor's muscles

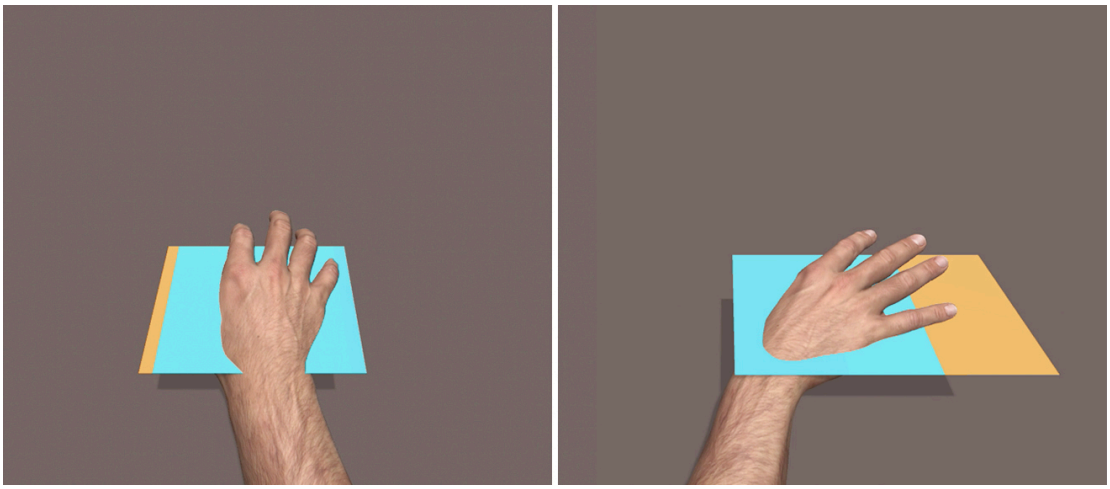

Figure 5.2 Hand-motion prediction in VR: in blue the current position of the hand; in orange the predicted position computed in real-time at 100fps (TensorFlow to Unity Reinforcement Learning framework).

or, simply put, under the reaction time. We obtained a lower yet acceptable accuracy. One of the main limitation we had was that, in order to process the data in such a short period of time, we could only model some specific parts of the body, such as the arm, and not the whole body. The possible next steps would be to increase the accuracy using unsupervised learning methods that were proven efficient for the type of systems we are simulating. Learning human actions, based on real-time gesture acquisition, is one of the direction we think have some great potential for achieving general-purpose intelligence.

The demo was coined "ExoBrain" as an implicit reference to the Exoskeletons known as "Shells" or "Protective Body": the predictive model could possibly help the user compensate for his relatively slow reaction time or prevent him to experience motion sickness, extending the range of his cognitive abilities.

5.2. Prediction Model for Haptic Feedback

In this experiment, we designed for engineering applications an innovative force feedback interface using Shape-Memory Alloys and we demonstrated that arti-

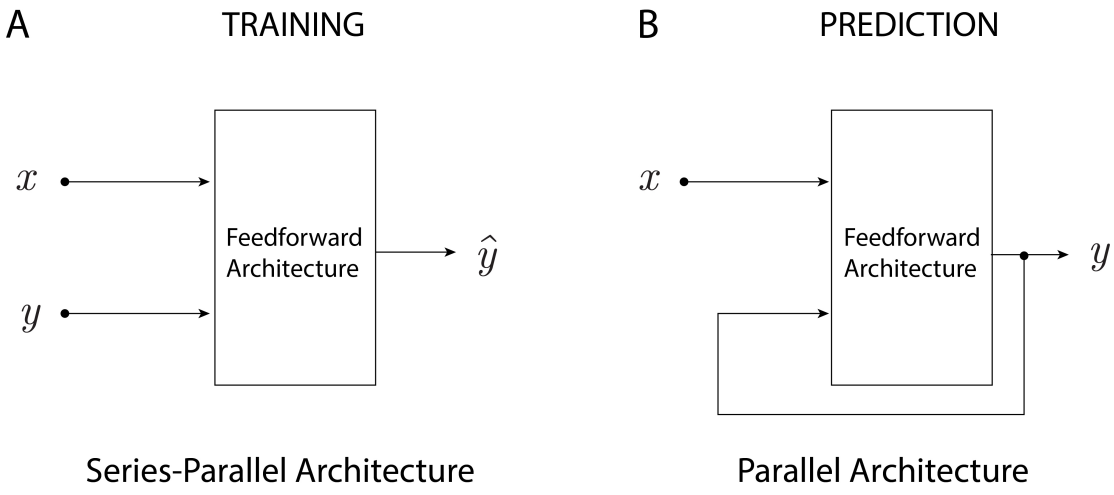

Figure 5.3 Difference between the Series-Parallel Architecture (diagram A, open-loop) and the purely Parallel Architecture (diagram B, closed-loop). The NARX network is first trained in open-loop (supervised with targets), then switched to a closed-loop for prediction based on the estimated outputs.

ficial nets could also be used for modeling the non-linear behavior of those actuators (Chernyshov et al. 2018b, Caremel et al. 2018). It appears that SMA applications are very limited, especially for haptic interface implementations, as it is difficult to precisely control them. Here, we could emulate a fully automated force feedback control scheme that did not require a complex analysis of the SMA depending on environment variables such as temperature, humidity, or the hysteresis of the system.

Classically, several sophisticated models can be used to simulate the behavior of SMA, such as the Preisach model and the Jiles-Atherton model, both used for ferromagnetism; however, the recent advances in neural networks allowed us to simulate nonlinear systems with a minimal set of parameters. In our current setup, the difficulty was that the resting time needed after each contraction of the ring is hardly predictable with conventional methods. This resting parameter was critical in our testing sessions: understanding the hysteresis of the SMA can help us triggering adequately timed haptic feedback. Indeed, if the contraction peak was reached instantly after a current was applied to the SMA, it took more time

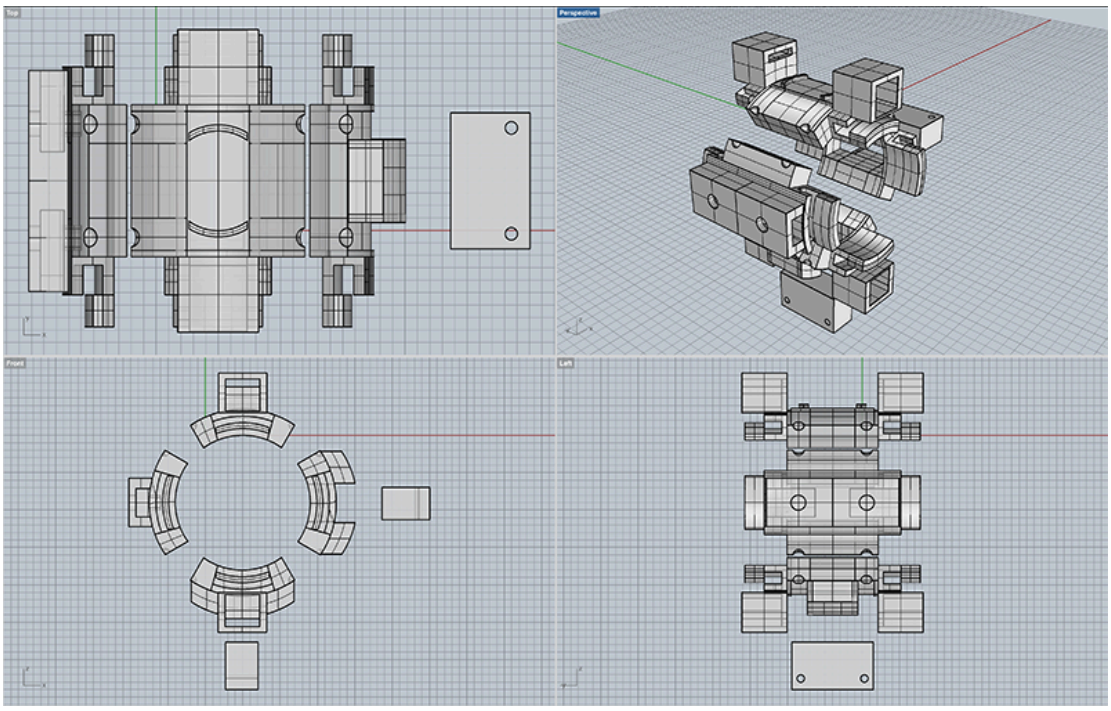

Figure 5.4 Technical drawing of the mechanical engineering of the ring structure. In clockwise direction from the top right quadrant of the screen: perspective view, left cross-section, front cross-section, top cross-section.

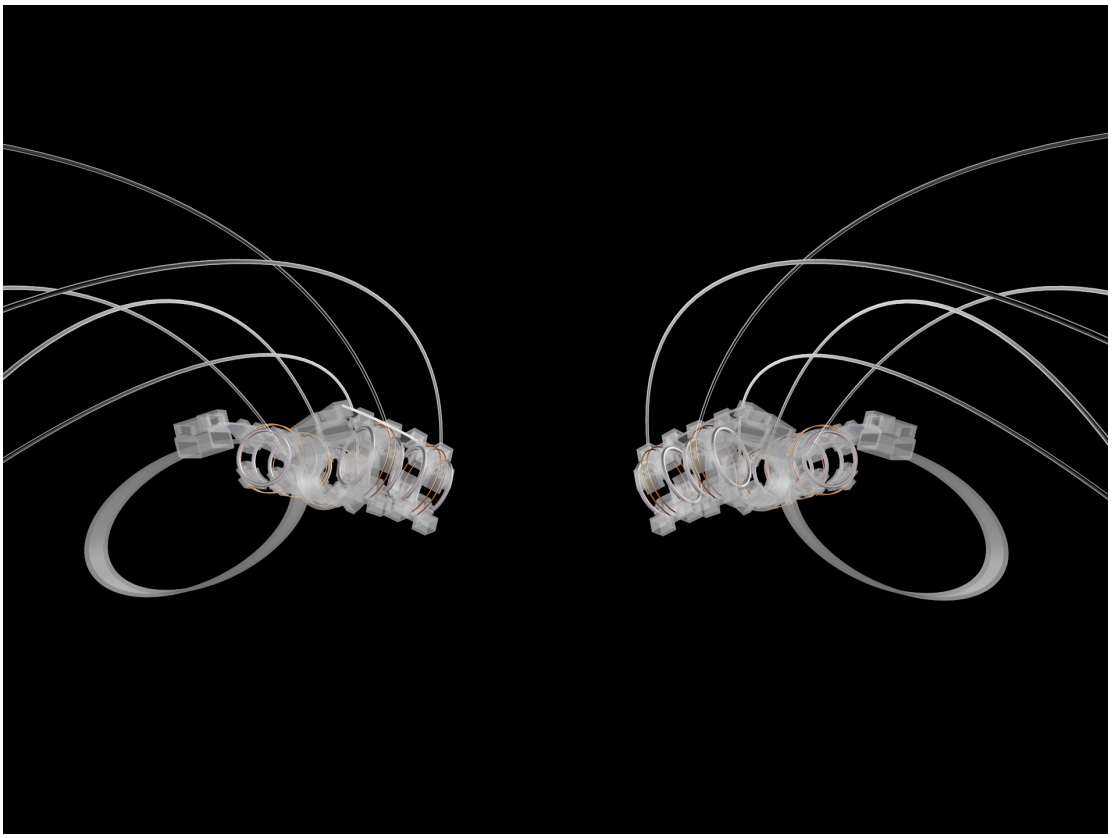

Figure 5.5 Rendering of the haptics glove, complete with the connectors, handles and ring structure.

Figure 5.6 Picture of the first prototype developed for haptic feedback with Takram. The SMA are visible on the gloves. When electric current is applied, the SMA applies pressure to the index and thumb.

for the ring to retrieve its original resting shape. The SMA cannot be quickly elongated unless it is forced-cooled, which would have drastically increased the complexity of the design of our wearable device. Therefore, the state of the system was highly dependent on variables such as the room temperature and the devices' history. Although the room temperature could be easily controlled, the previous states of the system did not exhibit linearity, which made prediction a difficult challenge. Therefore, anticipating the future states of the system was of crucial importance for a fully-controllable device.

First, we designed a setup consisting of a high-speed/high accuracy laser displacement meter (Keyence LC-2400 with LC-2440 measuring head) measuring the contraction displacement relative to the sensor head, in a horizontal plane. As the silicone tube, containing the SMA wire, goes through four FR-4 supports (exerting pressure on the finger), a reflector was placed on one of them. This support was

placed in front of the laser and the sensor was calibrated based on its reflectance. The three other supports were fixed on the horizontal plane so that each contraction would move the reflector only farther away from the sensor. The frictions to the horizontal plane, a slick plastic plate, were negligible at this scale, as the reflective marker could move freely without any significant difference compared to a test done in mid-air. The whole experiment was recorded at 240 fps. First, a constant current of 850mA, at 5V was applied to the SMA for 1s. The ring instantly contracted and we measured the displacement in mm, in the range 0.8004-0.881 (minimum-maximum). We repeated this procedure multiple times to evaluate the resting duration of the SMA ring, i.e. when the displacement measured by the sensor drops to near 0 (Fig. 5.2). It is worth noting the peak displacements by contraction were remarkably invariable over successive trials. However, the resting time was still problematic there: after each short contraction, it took more time for the alloy to recover its original shape, which we assessed here over 10 successive trials, with 9 resting intervals.

The data we collected in order to build a model consist in a time series that was acquired following the protocol described previously: we monitored the displacement dynamics of the ring by noting its magnitude and the associated time reference. Every time we triggered a contraction, the time it took for the ring to recover its original shape is not the same as previously noted. The resulting data was, as expected, very noisy and therefore very difficult to model.

Here, we wanted to use the right method to correctly assess the behavior of our SMA-actuated ring. Usually, smoothing methods are used to fit noisy data. We decided to test a conventional "smoothing spline" model, using the MATLAB Curve Fitting toolbox: when calling the fit function, it returns a vector of values defined by the spline interpolation of x (time) and y (displacement). However, only the values collected after about 1 min were properly fitted by this model; the displacement dynamic between 0 and 1 min is not well-captured. Indeed, the degree of the polynomial of the spline was high, with a relatively high RMSE of 0.3391. A value closer to 0 would have indicated that the fit is better for prediction. Likewise, the R-Squared error being weak at 0.6845, it performed better than a horizontal line, but the proportion of variance in the dataset was not fully captured by the fit.

Figure 5.7 Final setup exhibited by Takram (Photograph: Yuki Shinohara).

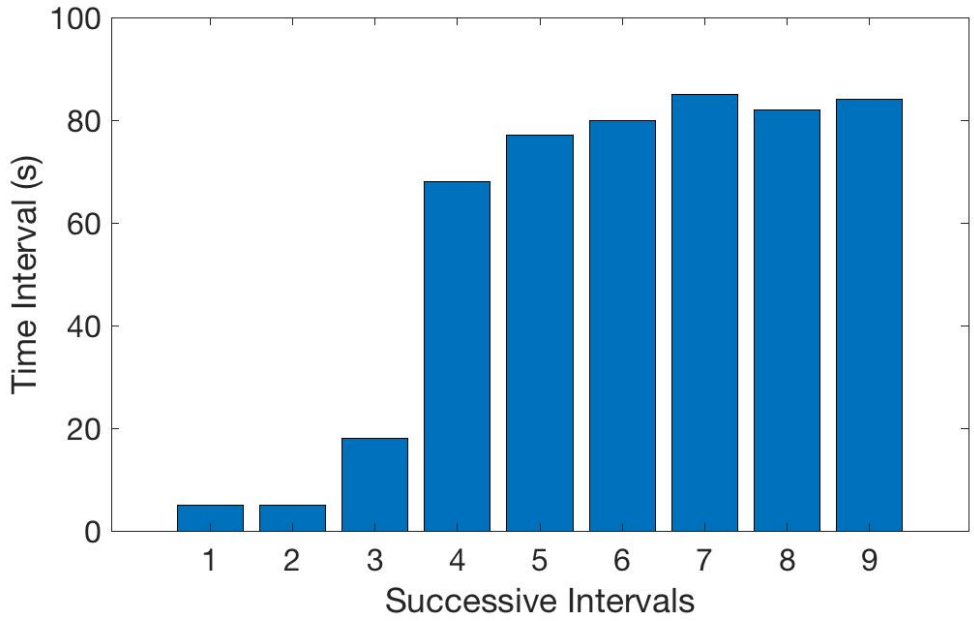

Figure 5.8 Resting time after displacement: it takes more time for the SMA to recover after successive contractions.

By stark contrast, we applied a NARX to the model and it largely outperformed the simple fit, making it useful for prediction. The RMSE was evaluated at 0.0851 (1 order of magnitude smaller compared to the previous results with simple fit) and the R-squared error was 0.832, a clear indicator that the predictive model is robust. Plotting the results of the NARX model compared to the original data, we monitored how the early trials were then fully captured by the model. Besides, the main advantage of this model architecture was that we could implement a predictive function. Unlike Wang *et al.* (Han Wang 2014), we didn't use the Levenberg-Marquardt algorithm for the gradient descent but the Fletcher-Reeves conjugate method, giving fast results. Also, we only needed n=8 units in our hidden layers which we found was the optimal configuration for a better generalization across the repeat of the trained network. We divided the data into three subsets: the training set (60%), the validation set (20%) and the test set (20%), using the 'dividerand' function in MATLAB, with its 'divideMode' property set

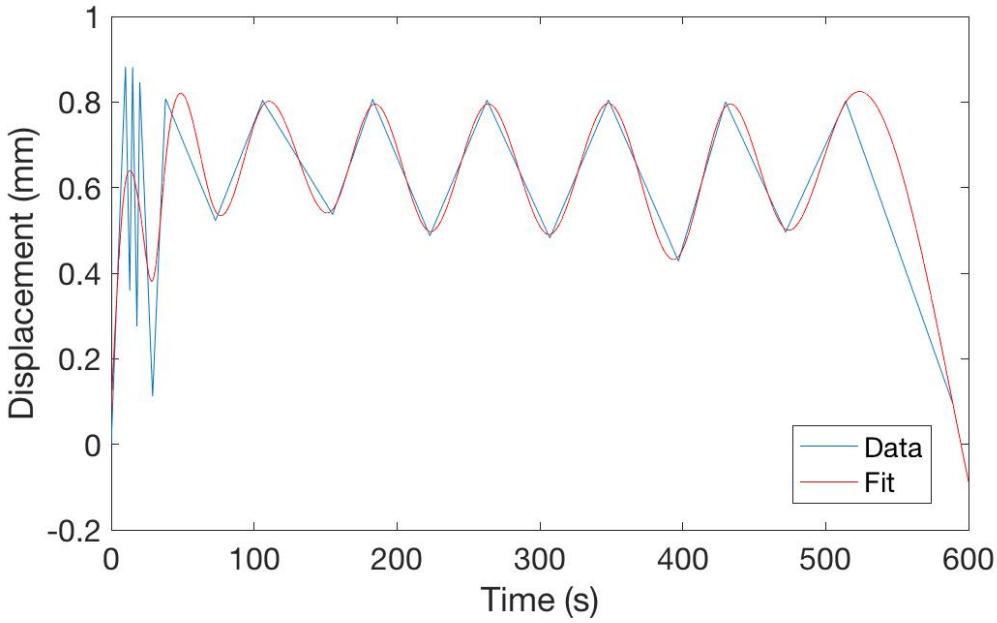

Figure 5.9 The simple fit, a spline interpolation between peak values, try to capture the nonlinear resting behavior of the SMA but does not properly describe all the characteristics of the alloy, especially the early stage (left values on the graph).

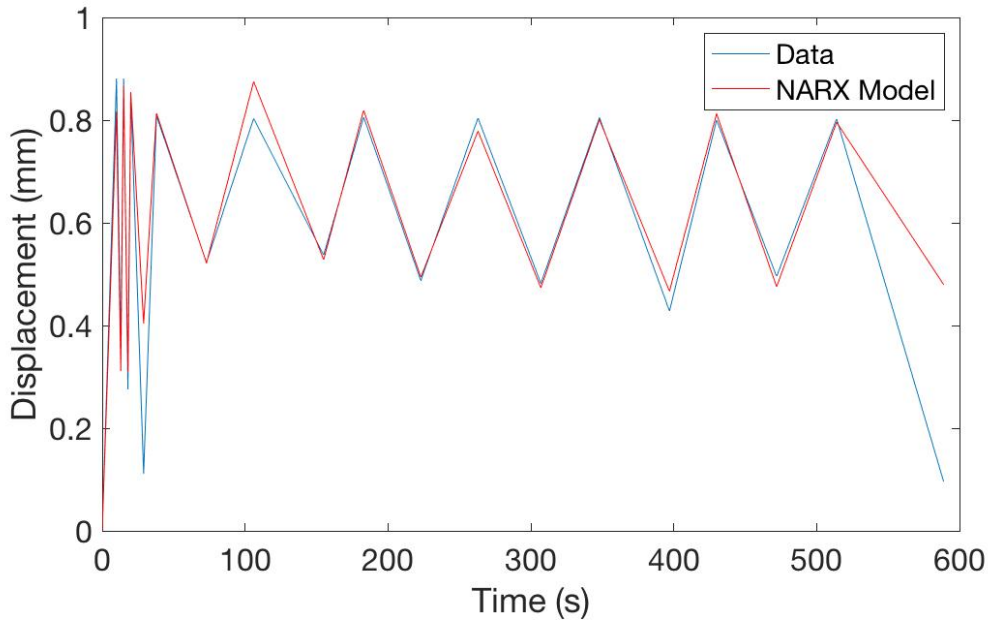

Figure 5.10 The Neural Network models and predicts the correct values much more accurately than the simple fit.

as 'time' for dynamic network. The neural network computed its weights and biases based on the training set. As the training progressed, the error on this set decreased. Meanwhile, a second set is used for validation: when the network started to over-fit the data, the error on the validation set, instead, increased: the training was stopped.

The third set, the testing set, was never used while training to ensure that we could objectively evaluate the performance of the trained model over a new set of data. We run the neural network multiple times until we obtained a satisfying model that generalizes well, without overfitting. The linear regression value R between outputs and targets on the training, validation and test sets was R=0.97 (1 is ideal) for the test set. There was no overfitting as both the training error and the testing error converge to a small mean square error (Fig. 3.3) and, more importantly, the predicted values were consistently in the range previously mentioned, around 0.8-0.9 mm displacement. Obviously, the features of the system

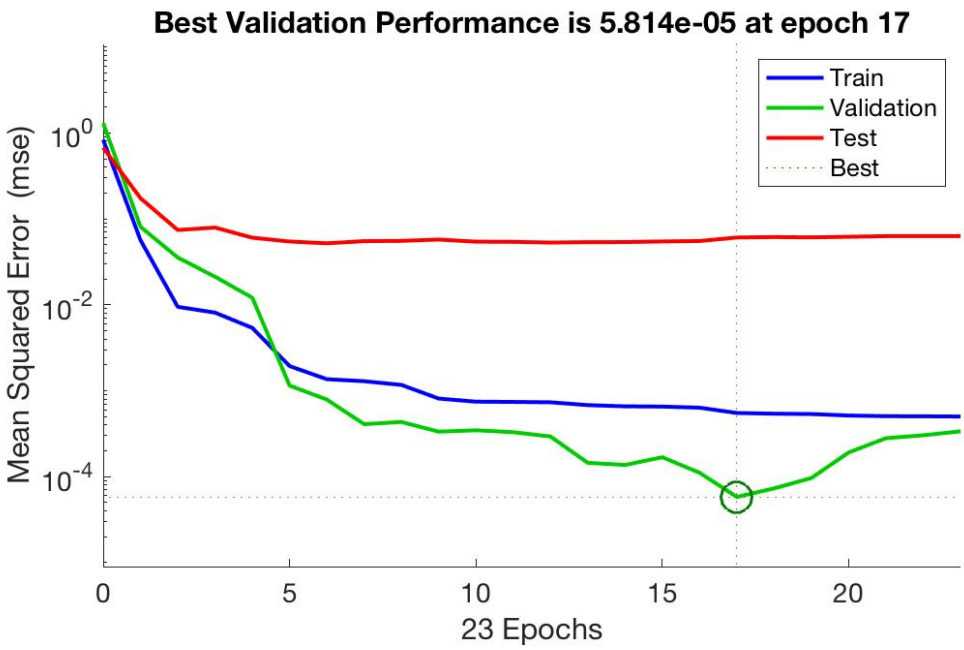

Figure 5.11 Best validation performance (mse) of the training, validation and test set. The error on the validation set increases after epoch 17, signaling the network starts to overfit the data. The weights and biases are selected at this minimum for the training set. The error on the test set, although higher, also decreases, showing similitudes with the training set trend.

were correctly assessed by the model: short burst of contraction with a displacement at a maximum peak below 1 and relax state duration based on nonlinear dependencies relative to the history of the system. We concluded that, if this setup showed artificial neural networks could enable accurate control over a complex haptic feedback, a wide range of novel type of sensors and actuators could be designed specifically using a model produced by artificial neural networks.

5.3. Prediction Model for Thermo-haptic Feedback

In biological systems, neural networks learn how to interact with the real world in real-time. The current application of my research on thermo-haptic feedback involved a reinforcement learning algorithm for motion prediction in a real-time virtual reality simulator (Chernyshov et al. 2018a), also described as a novel approach to study intelligent systems (Norman 2018). The hand position could then be accurately projected in the user's 3D space, enabling a thermal feedback of the user's visual perception of the effect (hot and ice materials handled in VR). The hand movements were tracked using infrared sensors mounted on a Vive (VR) headset and the TensorFlow toolkit was added to design the machine learning part, a LSTM with a TensorGraph depicted on Fig. 5.15. The objective of the simulation was to track, predict and display hand movements ahead of time, acquiring perceptual features related to thermoception. We used TensorFlow in Unity to implement a reinforcement learning algorithm for motion prediction. Unity is used as a simulator and the TensorFlow toolkit for the machine learning part. The hand movements were tracked using LeapMotion mounted on a Vive headset. The environment requires an agent to record observations of the hand behavior. The agent takes actions, such as changing the direction of the predicted hand. The predicted hand position is extrapolated by using the hand previous positions and superimposing the agent action. For each fairly correct assumption, the agent gets rewarded, otherwise it is penalized. This reward system informs the agent about the task, as it learns to optimize its policy to collect rewards, hence the name "reinforcement learning".

First, the simulation asked an agent to record observations of the hand behavior

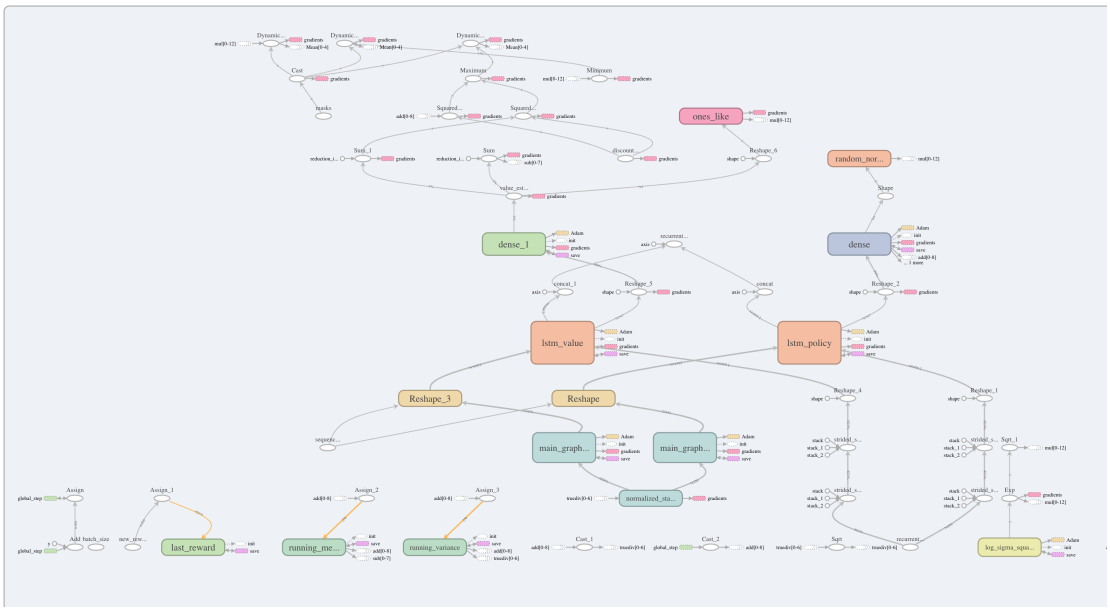

Figure 5.12 The Google TensorFlow Graph for the LSTM implemented in thermo-haptic feedback.

in the real world. During this training process, the agent took actions, such as changing the direction of the virtual hand in the simulator. The future positions were predicted by extrapolating the past and current state of the observational system. As described in the previous chapter, for each correct assumption, the agent got rewarded, otherwise it was penalized. This reward system informed the agent about the task, as it learned to optimize its policy to collect rewards. Our results showed how the agent was rewarded over time, with a small reward attributed for each fairly correct prediction. Hence, during the training session, the network accumulated rewards over time. However, the increase was non-linear, as the agent had to adapt its action to non-linear behaviors, as already described in section 3.2.1. For instance, a sudden change in the motion pattern may have temporarily penalized the agent. Nevertheless, because the network also adapted its learning rate (which represents the search for the optimal policy and steadily decrease over time), the prediction model could more reliable as the training progressed and the estimate increased over time.

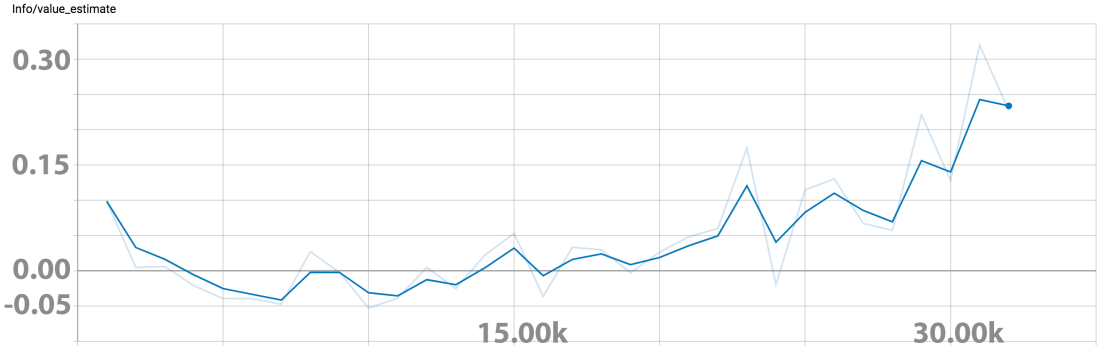

Figure 5.13 Value estimate graph demonstrating how prediction accuracy increases over the training.

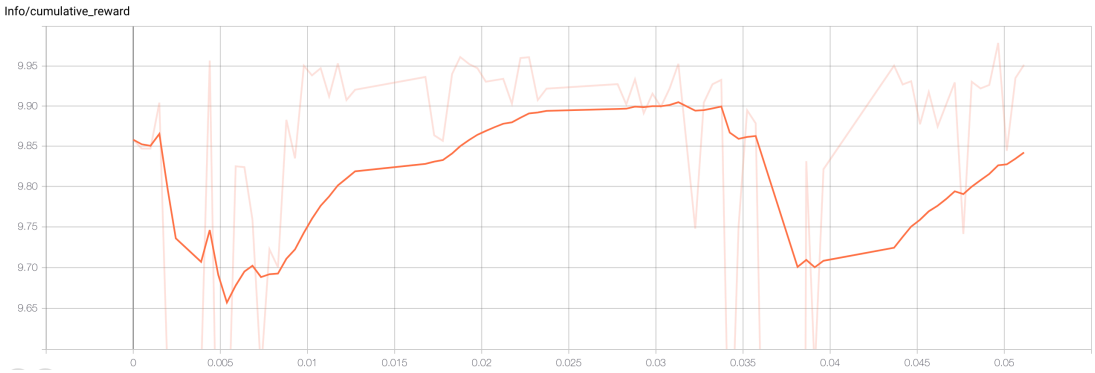

Figure 5.14 Cumulative rewards indicating how the agent is favored over time.

Figure 5.15 Thermo-haptic feedback demo exhibited at VRST (Photo credit: K. Ragozin).

More specifically, the cumulative reward indicates how the agent is retributed over time, with a reward of 0.01 attributed every frame to each fairly correct prediction, otherwise -0.001. Those scalars were estimated best based on the simulator frame rate set at 100fps. The increase showed that the training session was successful, as it accumulated rewards. Besides, the learning rate had decrease over time, a good indicator that the task was learned. It was set at 0.0003 and steadily decreased from 0.0003 to below 0.00012. Finally, as shown in Fig. 5.14, as the prediction model got more reliable, the value estimate increased over time. We concluded that the newly learned rules of our specific environment were mainly related to the range of sensations expressed by the users (from cold to warm), rather than the virtual representation of their hand position. Although the network did learn the features purely based on behavior, our assumption is that it actually learned second-order features, directly correlated to the user's thermoception, as it was aiming to maximize its rewards.

Notes

1 Adapted from: https://www.mathworks.com/help/deeplearning/ug/design-time-series-narx-feedback-neural-networks.html

Chapter 6
Conclusion and Future Work

"Chaos: when the present determines the future, but the approximate present does not approximately determine the future."

— Lorentz Edward.

6.1. Conclusion

The aim for the BioNN network was to output the correct associations between neural data and visual inputs. Although it has already been successfully demonstrated, initially in 2008 using Gabor wavelet functions (Kay et al. 2008) or more recently with state-of-the-art Generative Adversarial Networks (St-Yves and Naselaris 2018), such a demonstration using EEG represented a powerful proof-of-concept for my thesis research. Additional behavioral components, such as pupil dilation, may be recorded in a next development phase. Those components may be beneficial to help the network optimization when the neural data are too noisy and correlation patterns difficult to establish. Overall, non-invasive neurosignal recording applications constitute a desirable goal for the future of machine learning and BCI. First, for machine learning: training neural network can be sometimes fastidious and time-consuming. Neural data may be seen as a compressed version of the required amount of data to achieve a specific task. Currently, human experts can outperform most of the AI-powered algorithms if we only consider the training time as a performance indicator. Indeed, in the case

of AlphaGo Lee, the first version of AlphaGo, it took over 160,000 game patterns from the KGS data set to train the neural net (Silver et al. 2016). The second version, AlphaGo Zero, only took 36 hours (Silver et al. 2017), but while it certainly represents an impressive feat, one should consider that the training was executed over a pretrained Alpha Lee. With a game lasting on average 40 minutes, training Alpha Lee represents 6,400,000 minutes or over 12 years of uninterrupted practice. In that regard, the best reinforcement learning algorithms applied to video games still require an incommensurate amount of data, such as the one developed by Vinyals et al (Vinyals et al. 2017), which surpassed experts knowledge in a rather complex strategy game, but represents 200 years of game experience. What if the training data were only sparsely available, or observable actions, such as moving pawns on a game board, could not be derived? The bio-inspired neural network should be regarded as a different yet complementary approach: cooperation rather than competition is the highlight of such a network, as it capitalizes on the expert knowledge to learn the task. This type of training could be useful to model concepts or knowledge that cannot be easily explained or expressed by words or actions. More importantly, it is designed in its core to be flexible depending on the experimental conditions. Whether it is for image processing, time-series predictions or simple problem-solving, its adaptability, based on structural and functional brain connectivity, may be seen as a step closer to general intelligence. By developing biologically plausible neural networks, one can also envision the future of BCI applications, where information can transit from biological circuits to artificial ones and reciprocally. Eventually, databases of task-dependent neural signals could be constituted to download training sequences suitable for specific tasks.

6.2. Limitation and Future Work

Regarding my latest EEG setup, for further improvement, medical research equipment could be beneficial to get more granular, detailed, data. Another desiderata would be to test the network in real-life scenarios, by automating some simple tasks, such as guessing a participant intention based on their neural data. Ethical considerations should be taken into account, so as to enlarge the per-

spectives that neuromorphic applications offer. Yet, after testing several types of neural network along the years, the results of my last EEG setup proved to be conclusive for creating a "neuromorphic network", delineating a very promising framework for future developments in HCI and BCI. Although the main language used for the first implementation were MATLAB, the core concepts were developed with Keras/Tensorflow (Python) to offer a free, open-source alternative to researchers interested in building bio-inspired networks. One of the main limitations I encountered during my research along those years, was to find the right methodology to explore artificial neural networks from a bottom-up approach, as opposed to a top-down approach, where a strong theoretical corpus is already constituted, and one may derive irrefutable properties from the main principles. However, because machine learning is still a young discipline w.r.t. to biology, physics or engineering, the profuse amount of methods, techniques, ideas, theories, annotations and standards, make it virtually impossible to grasp all the nuances, historical points, latest trends, mathematical corners and ethical concerns that come along the way. Because machine learning is inherently a vast, interdisciplinary field, which requires solid notions in various domains, the amount of technical challenges and theoretical "impasses" that one must face to build upon the existing framework is sometimes hard but exhilarating; yet, learning machines demand data. This year, a turning point in my research was to, carefully, methodologically, create synthetic, perfect, random data before each new network simulation, so as to avoid the burden of broken sources, discrepancies within the dataset, format issues and other NaN. More importantly, designing synthetic dataset to fit some network requirements allows a deeper understanding of how the inner gears and processes interplay. Instead of laboriously iterating over machine learning algorithm versions to match the data structure, I designed the data to fit the network I wanted to build, so as to simulate the network on real data in a subsequent step. The result, a neuromorphic, simple yet flexible, fast network of nodes and layers, capable of prediction and simulation, provided me with a better control over the parameters than I had imagined. One of the most exciting, perhaps intriguing, perspective, was, and still is, to build models for searching patterns in seemingly noisy data. Obviously, all computer generated random numbers are in fact, pseudo-random numbers, and finding hints of

the existence of the traces of a programmatic structure in non-deterministic generators represents for me a much sought-after research direction, with possible applications to a multitude of fields.

6.3. Summary

At the start of my doctoral program, I was honored to be invited to the Schloss Dagstuhl seminar (Leibniz Center for Informatics) in Germany, where I had the chance and the privilege to meet and discuss some of my early results with international senior researchers, providing me with additional references and valuable resources on the topic, and confirming the potential significance of my thesis for computer applications in science. I worked extensively with engineers, innovators and researchers at Sony CSL and Takram Design Engineering on novel interfaces and hardware. Last year, I was part of the NII Shonan Meetings, the first Dagstuhl-style seminar held in Asia by the National Institute of Informatics in Japan, where I could refine my research work direction on predictive modeling. At Riken CBS, I focused my own research on artificial neural networks applied to biological models, as this burgeoning field is traversed by highly important yet unanswered questions.

After finalizing both the set up of a EEG recording device and the design of a task to acquire and exploit neural data, I established a map of regions of interest in the brain, according to the task. I conducted preliminary data recording and analysis to verify that the set-up was in line with my final objective and implemented a task-oriented, bio-inspired artificial neural network. In conclusion, this research work represents a detailed approach, based on artificial neural networks, to capture and predict goal-oriented behaviors. Inspired by engineering core principles, I proposed an original formulation on how to integrate machine learning to novel hardware and applications, and as artificial neural networks will gain in interpretability, I am convinced they will extend the range of our tools for revisiting or defining novel theories.

References

A., Turing (1937) "On computable numbers, with an application to the Entscheidungsproblem," *Proceedings of the London Mathematical Society, Series 2, Volume 42*.

Acharya, U. Rajendra, Shu Lih Oh, Yuki Hagiwara, Jen Hong Tan, Hojjat Adeli, and D. P Subha (2018) "Automated EEG-based screening of depression using deep convolutional neural network," *Computer Methods and Programs in Biomedicine*, Vol. 161, pp. 103–113.

Aoki, Ryo, Tadashi Tsubota, Yuki Goya, and Andrea Benucci (2017) "An automated platform for high-throughput mouse behavior and physiology with voluntary head-fixation," *Nature Communications*, Vol. 8, No. 1.

Bakker, Rembrandt, Paul Tiesinga, and Rolf Kötter (2015) "The Scalable Brain Atlas: Instant Web-Based Access to Public Brain Atlases and Related Content," *Neuroinformatics*, Vol. 13, No. 3, pp. 353–366.

Bishop, Christopher (2007) *Pattern Recognition and Machine Learning.*: Springer.

Caremel, Cedric, Gemma Liu, George Chernyshov, and Kai Kunze (2018) "Muscle-Wire Glove: Pressure-Based Haptic Interface," in *IUI Companion*, pp. 52:1–52:2: ACM.

Carreiras, Manuel, Ileana Quiñones, Juan Andrés Hernández-Cabrera, and Jon Andoni Duñabeitia (2014) "Orthographic Coding: Brain Activation for Letters, Symbols, and Digits," *Cerebral Cortex*, Vol. 25, No. 12, pp. 4748–4760.

Chaisangmongkon, Warasinee, Sruthi K Swaminathan, David J Freedman, and Jing Wang (2017) "network performs sequential category-based decisions," Vol. 93, No. 6, pp. 1504–1517.

Chernyshov, George, Kirill Ragozin, Cedric Caremel, and Kai Kunze (2018a) "Hand motion prediction for just-in-time thermo-haptic feedback," in *VRST*, pp. 52:1–52:2: ACM.

Chernyshov, George, Benjamin Tag, Cedric Caremel, Feier Cao, Gemma Liu, and Kai Kunze (2018b) "Shape memory alloy wire actuators for soft, wearable haptic devices," in *UbiComp*, pp. 112–119: ACM.

Christiansen, Eric M., Samuel J. Yang, D. Michael Ando, Ashkan Javaherian, Gaia Skibinski, Scott Lipnick, Elliot Mount, Alison O'Neil, Kevan Shah, Alicia K. Lee, Piyush Goyal, William Fedus, Ryan Poplin, Andre Esteva, Marc Berndl, Lee L. Rubin, Philip Nelson, and Steven Finkbeiner (2018) "In Silico Labeling: Predicting Fluorescent Labels in Unlabeled Images," *Cell*, Vol. 173, No. 3, pp. 792–803.e19.

Cortes, Corinna and Vladimir Vapnik (1995) "Support-vector networks," *Machine learning 20.3 (1995): 273-297.*

Crichton, Ellika M., Marie Noël, Esther A. Gies, and Peter S. Ross (2017) "A novel, density-independent and FTIR-compatible approach for the rapid extraction of microplastics from aquatic sediments," *Analytical Methods*, Vol. 9, No. 9, pp. 1419–1428.

Dayan, Peter (2008) "Simple substrates for complex cognition," *frontiers in Neuroscience*, Vol. 2, No. 2, pp. 255–263.

Doborjeh, Gholami, Z., N. Kasabov, Gholami Doborjeh, M., and A. Sumich (2018) "Modelling Peri-Perceptual Brain Processes in a Deep Learning Spiking Neural Network Architecture.," *Scientific reports, 8(1), 8912.*

Esteva, Andre, Brett Kuprel, Roberto A. Novoa, Justin Ko, Susan M. Swetter, Helen M. Blau, and Sebastian Thrun (2017) "Dermatologist-level classification of skin cancer with deep neural networks," *Nature*, Vol. 542, No. 7639, pp. 115–118.

Gallicchio, Claudio, Alessio Micheli, and Luca Pedrelli (2017) "Deep reservoir computing: A critical experimental analysis," *Neurocomputing*, Vol. 268, No. April, pp. 87–99.

Gallicchio, Claudio, Alessio Micheli, and Luca Pedrelli (2018) "Design of deep echo state networks," *Neural Networks*, Vol. 108, No. February, pp. 33–47.

Gardner, Martin (1970) "Mathematical Games: The fantastic combinations of John Conway's new solitaire game "Life"."," *Scientific American. 223: 120â123.*

Glorot, Xavier and Antoine Bordes (2011) "Deep Sparse Rectifier Neural Networks," *Proceedings of the Fourteenth International Conference on Artificial Intelligence and Statistics*, Vol. 15, pp. 315–323.

Goulas, Alexandros, Matteo Bastiani, Gleb Bezgin, Harry B. M. Uylings, Alard Roebroeck, and Peter Stiers (2014) "Comparative Analysis of the Macroscale Structural Connectivity in the Macaque and Human Brain," *PLoS Computational Biology*, Vol. 10, No. 3, p. e1003529.

Guerguiev, Jordan, Timothy P. Lillicrap, and Blake A. Richards (2017) "Towards deep learning with segregated dendrites," *eLife*, Vol. 6, pp. 1–37.

Hammer, Barbara, Benjamin Schrauwen, and Jochen J. Steil (2009) "Recent advances in efficient learning of recurrent networks," in *ESANN 2009, 17th European Symposium on Artificial Neural Networks, Bruges, Belgium, April 22-24, 2009, Proceedings.*

Han Wang, Gangbing Song (2014) "Innovative NARX recurrent neural network model for ultra-thin shape memory alloy wire," *Elsevier, Neurocomputing*, Vol. 134.

Hebb, D.O. (1949) *The Organization of Behavior, New York, Wiley.*

Hochreiter, Sepp (1991) "Untersuchungen zu dynamischen neuronalen Netzen. Diploma thesis, TU Munich."

Hopfield, J. J. (1982) "Neural networks and physical systems with emergent collective computational abilities.," *Proceedings of the National Academy of Sciences*, Vol. 79, No. 8, pp. 2554–2558.

Ian J. Goodfellow, Mehdi Mirza Bing Xu David Warde-Farley Sherjil Ozairâ Aaron Courville Yoshua Bengio, Jean Pouget-Abadieâ (2014) *Generative Adversarial Nets.*

K., (Pearson (1936) *Method of Moments and Method of Maximum Likelihood.*

Kasahara, S. and J. Rekimoto (2014) "JackIn: integrating first-person view with out-of-body vision generation for humancaugmentation," *J. In Proceed- ings of the 5th Augmented Human International Conference, ACM , 46.*

(2009) "Synaptic Activity and the Construction of Cortical Circuits Author (s): L . C . Katz and C . J . Shatz Published by : American Association for the Advancement of Science Stable URL : http://www.jstor.org/stable/2891573," *Advancement Of Science*, Vol. 274, No. 5290, pp. 1133–1138.

Kay, Kendrick N., Thomas Naselaris, Ryan J. Prenger, and Jack L. Gallant (2008) "Identifying natural images from human brain activity," *Nature*, Vol. 452, No. 7185, pp. 352–355.

Keunwoo Choi, Mark Sandler, George Fazekas (2016) "Automatic tagging using deep convolutional neural networks," *ISMIR (International Society of Music Information Retrieval) Conference 2016.*

Kirchhoff, Michael, Thomas Parr, Ensor Palacios, Karl Friston, and Julian Kiverstein (2018) "The markov blankets of life: Autonomy, active inference and the free energy principle," *Journal of the Royal Society Interface*, Vol. 15, No. 138.

Kobak, Dmitry, Wieland Brendel, Christos Constantinidis, Claudia E. Feierstein, Adam Kepecs, Zachary F. Mainen, Xue Lian Qi, Ranulfo Romo, Naoshige Uchida, and Christian K. Machens (2016) "Demixed principal component analysis of neural population data," *eLife*, Vol. 5, No. APRIL2016, pp. 1–36.

Lescroart, Mark, Kendrick Kay, Thomas Naselaris, Ryan Prenger, Michael Oliver, and Jack Gallant (2011) "fMRI of human visual areas in response to natural images."

Liang, D. (2017) "Automated Multi-task Learnin," *UC San Diego.*

Libonati, F., C. Colombo, and L. Vergani (2013) "Design and characterization of a bio-inspired composite," *Engineering Against Failure - Proceedings of the 3rd International Conference of Engineering Against Failure, ICEAF 2013*, pp. 30–38.

Luckow, Andre, Matthew Cook, Nathan Ashcraft, Edwin Weill, Emil Djerekarov, and Bennie Vorster (2016) "Deep learning in the automotive industry: Applications and tools," *Proceedings - 2016 IEEE International Conference on Big Data, Big Data 2016*, pp. 3759–3768.

Luo, Liqun (2016) *Principles of Neurobiology*: Garland Science.

Majka, Piotr, Tristan A. Chaplin, Hsin-Hao Yu, Alexander Tolpygo, Partha P. Mitra, Daniel K. Wójcik, and Marcello G.P. Rosa (2016) "Towards a comprehensive atlas of cortical connections in a primate brain: Mapping tracer injection studies of the common marmoset into a reference digital template," *Journal of Comparative Neurology*, Vol. 524, No. 11, pp. 2161–2181.

Marr, David (1982) *Vision.*: The MIT Press, Cambridge, Massachusetts.

Martens, James and Ilya Sutskever (2011) "Learning recurrent neural networks with Hessian-free optimization," *Proceedings of the 28th International Conference on Machine Learning, ICML 2011*, pp. 1033–1040.

Martinolli, Marco, Wulfram Gerstner, and Aditya Gilra (2018) "Multi-Timescale Memory Dynamics Extend Task Repertoire in a Reinforcement Learning Network With Attention-Gated Memory," *Frontiers in Computational Neuroscience*, Vol. 12.

Matteo Ruffini, Borja Balle, Guillaume Rabusseau (2017) "Hierarchical Methods of Moments," *NIPS*.

McCulloch, W.S. and W. Pitts (1943) *A Logical Calculus of Ideas Immanent in Nervous Activity.*

Mead, Carver (1990) "Mead C (1990, October) Neuromorphic electronic systems. Proc IEEE," *Proceedings of the IEEE*, Vol. 78, pp. 1629 – 1636.

Meier, Jil, Prejaas Tewarie, Arjan Hillebrand, Linda Douw, Bob W. van Dijk, Steven M. Stufflebeam, and Piet Van Mieghem (2016) "A Mapping Between Structural and Functional Brain Networks," *Brain Connectivity*, Vol. 6, No. 4, pp. 298–311.

Mijalkov, Mite, Ehsan Kakaei, Joana B. Pereira, Eric Westman, and Giovanni Volpe and (2017) "BRAPH: A graph theory software for the analysis of brain connectivity," *PLOS ONE*, Vol. 12, No. 8, p. e0178798.

Minamizawa, Kouta, Yasuaki Kakehi, Masashi Nakatani, Soichiro Mihara, and Susumu Tachi (2012) "TECHTILE toolkit: a prototyping tool for design and education of haptic media," in *VRIC*.

Minsky M., Papert S. (1969) *Perceptrons: an introduction to computational geometry.*

Monroe, Don (2014) "Neuromorphic Computing Gets Ready for the (Really) Big Time," *Commun. ACM*, Vol. 57, No. 6, pp. 13–15.

Nayebi, Aran, Daniel Bear, Jonas Kubilius, Kohitij Kar, Surya Ganguli, James J Dicarlo, Daniel L K Yamins, Cognitive Sciences, Google Brain, and Mountain View (2018) "Task-Driven Convolutional Recurrent Models of the Visual System," pp. 1–14.

Niell and Stryker (2008) "Highly selective receptive fields in mouse visual cortex."

Norman, M.D. (2018) *Applying Complexity Science with Machine Learning, Agent-Based Models, and Game Engines: Towards Embodied Complex Systems Engineering.*

Orban, Guy A., David Van Essen, and Wim Vanduffel (2004) "Comparative mapping of higher visual areas in monkeys and humans," *Trends in Cognitive Sciences*, Vol. 8, No. 7, pp. 315–324.

O'Reilly, Randall C. and Michael J. Frank (2006) "Making Working Memory Work: A Computational Model of Learning in the Prefrontal Cortex and Basal Ganglia," *Neural Computation*, Vol. 18, No. 2, pp. 283–328.

Pearlmutter, Barak A and Geoffrey E Hinton (1986) "G-Maximi,zation: an Unsupervised Learning Procedure," *Snowbird Conference on Neural Networks for Computing*, No. 151, pp. 333–338.

Pennachin, Ben GoertzelCassio (2007) *Artificial General Intelligence*: Springer.

R., Bellman (2003) *Dynamic Programming.*

Rosenblatt, Frank (1958) *The perceptron: A probabilistic model for information storage and organization in the brain.*

Rumelhart, Geoffrey E.; Williams Ronald J., David E.; Hinton (1986) "Learning representations by back-propagating errors," *Volume 323, Issue 6088, pp. 533-536 (1986).*

Russel, B. and A. N. Whitehead (1910-13) *Principia Mathematica.*

Sabour, Sara, Nicholas Frosst, and Geoffrey E. Hinton (2017) "Dynamic Routing Between Capsules," *CoRR*, Vol. abs/1710.09829.

Salehinejad, Hojjat, Sharan Sankar, Joseph Barfett, Errol Colak, and Shahrokh Valaee (2017) "Recent Advances in Recurrent Neural Networks," pp. 1–21.

Scarselli, Franco, Marco Gori, Ah Chung Tsoi, Markus Hagenbuchner, and Gabriele Monfardini (2009) "The graph neural network model," *IEEE Transactions on Neural Networks IEEE Transactions on Neural Networks IEEE TRANSACTIONS ON NEURAL NETWORKS*, Vol. 20, No. 1, pp. 61–80.

Schiller UD, Steil JJ (2005) "Analyzing the weight dynamics of recurrent learning algorithms," *Neurocomputing 63: 5-23.*

Seth, A. K., A. B. Barrett, and L. Barnett (2015) "Granger Causality Analysis in Neuroscience and Neuroimaging," *Journal of Neuroscience*, Vol. 35, No. 8, pp. 3293–3297.

Sezer, Omer Berat and Ahmet Murat Özbayoglu (2019) "Financial Trading Model with Stock Bar Chart Image Time Series with Deep Convolutional Neural Networks," *CoRR*, Vol. abs/1903.04610.

Silver, David, Aja Huang, Chris J. Maddison, Arthur Guez, Laurent Sifre, George van den Driessche, Julian Schrittwieser, Ioannis Antonoglou, Veda Panneershelvam, Marc Lanctot, Sander Dieleman, Dominik Grewe, John Nham, Nal Kalchbrenner, Ilya Sutskever, Timothy Lillicrap, Madeleine Leach, Koray Kavukcuoglu, Thore Graepel, and Demis Hassabis (2016) "Mastering the game of Go with deep neural networks and tree search," *Nature*, Vol. 529, No. 7587, pp. 484–489.

Silver, David, Thomas Hubert, Julian Schrittwieser, Ioannis Antonoglou, Matthew Lai, Arthur Guez, Marc Lanctot, Laurent Sifre, Dharshan Kumaran, Thore Graepel, Timothy Lillicrap, Karen Simonyan, and Demis Hassabis (2017) "Mastering Chess and Shogi by Self-Play with a General Reinforcement Learning Algorithm," pp. 1–19.

SivasankariK. and ThanushkodiK. "n Improved EEG Signal Classification Using Neural Network with the Consequence of ICA and STFT.," *Journal of Electrical Engineering and Technology*, Vol. 9, No. 3, pp. 1060–1071.

Smith, Michael R. and Tony Martinez (2011) "Improving Classification Accuracy by Identifying and Removing Instances that Should Be Misclassified," *Conference: Neural Networks (IJCNN), The 2011 International Joint Conference on.*

Solomon, Samuel G. and Marcello G. P. Rosa (2014) "A simpler primate brain: the visual system of the marmoset monkey," *Frontiers in Neural Circuits*, Vol. 8.

Spoerer, Courtney J., Patrick McClure, and Nikolaus Kriegeskorte (2017) "Recurrent Convolutional Neural Networks: A Better Model of Biological Object Recognition," *Frontiers in Psychology*, Vol. 8.

St-Yves, Ghislain and Thomas Naselaris (2018) "Generative Adversarial Networks Conditioned on Brain Activity Reconstruct Seen Images."

Strogatz (2000) *Nonlinear Dynamics and Chaos.*

Sussillo, David, "LFADS - Latent Factor Analysis via Dynamical Systems."

Sussillo, David and Omri Barak (2013) "Opening the black box: Low-dimensional dynamics in high-dimensional recurrent neural networks," *Neural Computation*, Vol. 25, No. 3, pp. 626–649.

Sussillo, David, Mark M. Churchland, Matthew T. Kaufman, and Krishna V. Shenoy (2015) "A neural network that finds a naturalistic solution for the production of muscle activity," *Nature Neuroscience*, Vol. 18, No. 7, pp. 1025–1033.

Swindale, N. V. (1998) "Orientation tuning curves: Empirical description and estimation of parameters," *Biological Cybernetics*, Vol. 78, No. 1, pp. 45–56.

Training, Backpropagation (2006) "1 . 2 On the origin and development of neurocomputing," No. 1958.

Vinyals, Oriol, Timo Ewalds, Sergey Bartunov, Petko Georgiev, Alexander Sasha Vezhnevets, Michelle Yeo, Alireza Makhzani, Heinrich Küttler, John Agapiou, Julian Schrittwieser, John Quan, Stephen Gaffney, Stig Petersen, Karen Simonyan, Tom Schaul, Hado van Hasselt, David Silver, Timothy P. Lillicrap, Kevin Calderone, Paul Keet, Anthony Brunasso, David Lawrence, Anders Ekermo, Jacob Repp, and Rodney Tsing (2017) "StarCraft II: A New Challenge for Reinforcement Learning," *CoRR*, Vol. abs/1708.04782.

Whittington, James C.R. and Rafal Bogacz (2019) "Theories of Error Back-Propagation in the Brain," *Trends in Cognitive Sciences.*

Xie, Hang, H A O Tang, and Yu-he Liao (2009) "Time Series Prediction Based on NARX Neural Networks: An Advanced Approach," *Proceedings of the Eighth International Conference on Machine Learning and Cybernetics*, No. July, pp. 12–15.

Y. Paquot, A. Smerieri J. Dambre B. Schrauwen M. Haelterman S. Massar, F. Duport (2012) "Optoelectronic Reservoir Computing," *Scientific Reports volume 2, Article number: 287.*

Yan LeCun, J. S. Denker D. Henderson R. E. Howard W. Hubbard L. D. Jackel, B. Boser (1998) "Backpropagation Applied to Handwritten Zip Code Recognition."

Yang, Guangyu Robert, H. Francis Song, William T. Newsome, and Xiao-Jing Wang (2017) "Clustering and compositionality of task representations in a neural network trained to perform many cognitive tasks," *bioRxiv*, p. 183632.

Yann LeCun, Patrick Haffner, LÃ©on Bottou (1989) "Gradient Descent Algorithm Applied to Document Recognition."

Zuo, Yan, Gil Avraham, and Tom Drummond (2018) "Generative Adversarial Forests for Better Conditioned Adversarial Learning."

Appendices

A. Toy Model - Sample Code (MATLAB)

```
---------STRUCTURE-----------
net = network;
net.numInputs = 1;
layer_count = 4;
net.numLayers = layer_count;

net.biasConnect = ones(layer_count,1);
net.layerConnect(2,1) = 1;
net.layerConnect(3,2) = 1;
net.layerConnect(4,3) = 1;
net.outputConnect = ones(1,layer_count);

net.outputs{:}.feedbackInput = 1;
net.outputs{:}.feedbackMode = 'open';
net.inputs{1}.name = 'Task Variable';
net.inputs{2}.name = 'Signal 1';
net.inputs{3}.name = 'Signal 2';
net.inputs{4}.name = 'Signal 3';
net.inputs{5}.name = 'Behavioral Component';
```

```
%Layer 1 to Task Variable and Signal 1
net.inputConnect(1,1) = 1;
net.inputConnect(1,2) = 1;
%Layer 2 to Signal 2
net.inputConnect(2,3) = 1;
%Layer 3 to Signal 3
net.inputConnect(3,4) = 1;
%Layer 4 to Behavioral
net.inputConnect(4,5) = 1;

%net.LayerWeights{2,1}.delays = 1;
%net.LayerWeights{3,2}.delays = 1;
%net.LayerWeights{4,3}.delays = 10;
%net.inputWeights{2,2}.delays = 1;
%net.inputWeights{3,2}.delays = 1;
%net.inputWeights{4,2}.delays = 10;
net.outputs{1}.feedbackDelay = 1;
net.outputs{2}.feedbackDelay = 2;
net.outputs{3}.feedbackDelay = 2;
net.outputs{4}.feedbackDelay = 3;

dim_1 = size([Input_cell{1,1}],1);
dim_2 = size([Signal1_cell{1,1}],1);
dim_3 = size([Signal2_cell{1,1}],1);
dim_4 = size([Signal3_cell{1,1}],1);
dim_5 = size([Behavioral_cell{1,1}],1);

%Number of neurons for Input_cell
net.inputs{1}.size = dim_1;
%Number of neurons for Signal1_cell
net.inputs{2}.size = dim_2;
%Number of neurons for Signal2_cell
net.inputs{3}.size = dim_3;
%Number of neurons for Signal3_cell
net.inputs{4}.size = dim_4;
%%Number of neurons for Behavioral_cell
net.inputs{5}.size = dim_5;
```

Architecture and training parameters:

```
---------FUNCTION--------
net.layers{:}.transferFcn = 'poslin';
net.layers{4}.transferFcn = 'hardlim'; %'logsig'
net.layerWeights{:}.learnFcn = 'learngdm';
net.performFcn = 'crossentropy'; %'msereg'
%net.performParam.regularization = 0.01;
%net.performParam.normalization = 'none';
net.adaptFcn = 'learnhd'; %Hebb with decay weight
%net.adaptFcn = 'learncon'; %Conscience bias learning
%net.adaptFcn = 'learngd'; %Gradient descent
%net.adaptFcn = 'learnlv2'; %LVQ 2.1 weight learning
%net.adaptFcn = 'learnpn'; %Perceptron
%net.adaptFcn = 'learnsom'; %Self-organizing map

net.trainFcn = 'trainscg'; %SCG backpropagation

epoch=100; %number of planned training steps
net.trainParam.epochs = epoch;

net.divideMode = 'sampletime';
net.divideFcn = 'dividerand';
net.divideParam.trainRatio = 70/100;
net.divideParam.valRatio = 15/100;
net.divideParam.testRatio = 15/100;

net.plotFcns = {'plotregression'};

net.layers{1}.name = 'Signal 1 Layer';
net.layers{2}.name = 'Signal 2 Layer';
net.layers{3}.name = 'Signal 3 Layer';
net.layers{4}.name = 'Behavioral Layer';

save('net','net')
```

```
----------TRAINING---------
net = init(net);

row_start = 1;
row_end = sessions_n; %Number of trained sessions.
col_start = 1;
col_end = 1;
data_input = {[Input_cell],...
[Signal1_cell],...
[Signal2_cell],...
[Signal3_cell],...
[Behavioral_cell]}';
data_target = {[Signal1_cell],...
[Signal2_cell],...
[Signal3_cell],...
[Behavioral_cell]}';

[net, tr] = train(net,data_input,data_target);

view(net)
W_i_weight=net.IW;
W_l_weight=net.LW;
```

```
closednet=closeloop(net);

%Connect Layer 1 to Layer 2
closednet.layerConnect(2,1) = 1;
%Connect Layer 1 to Layer 3

closednet.layerConnect(3,1) = 1;
%Connect Layer 2 to Layer 4
closednet.layerConnect(3,2) = 1;
%Connect Layer 3 to Layer 4
closednet.layerConnect(4,3) = 1;
view(closednet)

data_input_test = {[Input_cell]}'
[outputs] = closednet(data_input_test); %Test unseen session
```

B. BioNN Code (Python Sample)

```
#!/usr/bin/env python
# coding: utf-8
##########################################################
import keras
from keras.models import Sequential, Model
from keras.layers import (Input, Embedding, LSTM, Dense,
Dropout, Activation, BatchNormalization)
from keras.optimizers import SGD, Adam
from keras.callbacks import (TensorBoard, ModelCheckpoint,
EarlyStopping,ReduceLROnPlateau)
from keras.utils import to_categorical
```

```
from sklearn.preprocessing import normalize, minmax_scale

import pandas as pd
import numpy as np
import time
import matplotlib.pyplot as plt
import matplotlib
import os

channel_num = 8

epoch1 = 2000
epoch2 = 200

#Neural architecture parameters
act = 'sigmoid' #hidden layer activation e.g. sigmoid
#Predictor parameters
actLSTM = 'relu' #hidden layer activation e.g. relu
n_LSTM = 256 #number of units in hidden layers
n_do = 0.1 #dropout in dense layers [0,1]
n_DenseLSTM = 1 #number of units in output layers

lr_n=0.001 #optimizer
decay_n=1e-5 #optimizer
loss_fc='mae' #objective function
b_size = 1 #Batch size

#Classifier parameters
actDense_c = 'relu'
actOut_c = 'sigmoid' #output layer activation
n_Dense = 64 #number of units in hidden layers
n_do_c = 0.1 #dropout in dense layers [0,1]
```

```
lr_c=0.0001 #optimizer
decay_c=1e-5 #optimizer
loss_fc_c='binary_crossentropy' #objective function
b_size_c = 12 #Batch size

print(keras.__version__)
from tensorflow.python.client import device_lib
print(device_lib.list_local_devices())
keras.backend.clear_session()

#####################DATA FORMAT############################

#Data should be formatted as follows (Pandas DataFrame main_df):
print(main_df.head())
   timestamp        CH1        CH2        CH3        CH4        CH5       CH6
0     262263 -5957.8276 -4735.3843  6936.8374  1477.7177 -1137.4503  8699.506
1     166260 -6102.6440 -5061.7305  8240.2400  1742.7611 -1489.5394  9228.946
2     661257 -5902.5000 -5057.8936  8845.1260  1865.3721 -1682.4323  9307.123
3     760260 -5402.4263 -4869.0654  9659.1690  2028.9210 -1983.9315  8889.895
4     558259 -4858.0415 -4563.2583  9769.4380  2042.3453 -2092.5664  8151.338

         CH7        CH8  Task Variable  Behavioral
0  2848.1594  7123.8335              1           0
1  3178.9612  8410.9260              0           0
2  3341.8276  8983.0940              2           0
3  3555.8054  9523.4880              3           1
4  3436.3738  9333.0030              5           0


#Dataset split length
main_df_np = np.array(main_df)
len_timeseries = main_df_np.shape[0]
n_ = round(len_timeseries/3)-1
```

```
#Get data for Classifier
#Channels are inputs x, Behavioral is target y
x_train_Neural = main_df_np[0:n_,1:9] #Channels
y_train_Behavior = main_df_np[0:n_,-1] #Behavioral 1 or 0 (Left or Right)
x_validation_Neural = main_df_np[n_:n_*2,1:9]
y_validation_Behavior = main_df_np[n_:n_*2,-1]
x_test_Neural = main_df_np[n_*2:n_*3,1:9]
y_test_Behavior = main_df_np[n_*2:n_*3,-1]

#Get data for Predictor
#Rescale [0,1]:
main_df_np = minmax_scale(main_df_np,feature_range=(0, 1), axis=0)

#Task Variable is input x, Channels are targets y
x_train_Task = main_df_np[0:n_,-2] #Task Variable 1 to 6 classes
y_train_Neural = main_df_np[0:n_,1:9] #Channels
x_validation_Task = main_df_np[n_:n_*2,-2]
y_validation_Neural = main_df_np[n_:n_*2,1:9]
x_test_Task = main_df_np[n_*2:n_*3,-2]
y_test_Neural = main_df_np[n_*2:n_*3,1:9]

#Format data for Predictor

#LSTM encoding
x_train_Task = np.reshape(x_train_Task,(-1,n_))
y_train_Neural = np.reshape(y_train_Neural,(1,n_,8))

x_validation_Task = np.reshape(x_validation_Task,(-1,n_))
y_validation_Neural = np.reshape(y_validation_Neural,(1,n_,8))

x_test_Task = np.reshape(x_test_Task,(-1,n_))
y_test_Neural = np.reshape(y_test_Neural,(1,n_,8))
```

```
##############################################################

#Predictor architecture

#8 Channels
input_len = x_train_Task.shape[1]
main_input = Input(shape=(input_len,), dtype='int32', name='main_input')

x_ = Embedding(output_dim=8, input_dim=8, input_length=input_len)(main_input)

x = Dropout(n_do)(x_)

lstm_out = LSTM(n_LSTM, activation=actLSTM, return_sequences=True)(x)

ch7 = Dense(1, activation=act, name='ch7')(lstm_out) #L LOC (to In)
d7 = Dropout(n_do)(ch7)
lstm_out7 = LSTM(n_LSTM, activation=actLSTM, return_sequences=True)(d7)
de7 = Dense(n_DenseLSTM, activation=act)(lstm_out7)

ch8 = Dense(1, activation=act, name='ch8')(ch7) #R LOC (to L LOC)
d8 = Dropout(n_do)(ch8)
lstm_out8 = LSTM(n_LSTM, activation=actLSTM, return_sequences=True)(d8)
de8 = Dense(n_DenseLSTM, activation=act)(lstm_out8)

ch3 = Dense(1, activation=act, name='ch3')(lstm_out) #L DMN (to In)
d3 = Dropout(n_do)(ch3)
lstm_out3 = LSTM(n_LSTM, activation=actLSTM, return_sequences=True)(d3)
de3 = Dense(n_DenseLSTM, activation=act)(lstm_out3)


ch4 = Dense(1, activation=act, name='ch4')(ch3) #R DMN (to L DMN)
d4 = Dropout(n_do)(ch4)
lstm_out4 = LSTM(n_LSTM, activation=actLSTM, return_sequences=True)(d4)
```

```
de4 = Dense(n_DenseLSTM, activation=act)(lstm_out4)


ch2 = Dense(1, activation=act, name='ch2')(ch4) #R Front (to R DMN)
d2 = Dropout(n_do)(ch2)
lstm_out2 = LSTM(n_LSTM, activation=actLSTM, return_sequences=True)(d2)
de2 = Dense(n_DenseLSTM, activation=act)(lstm_out2)


ch1 = Dense(1, activation=act, name='ch1')(ch2) #L Front (to R Front)
d1 = Dropout(n_do)(ch1)
lstm_out1 = LSTM(n_LSTM, activation=actLSTM, return_sequences=True)(d1)
de1 = Dense(n_DenseLSTM, activation=act)(lstm_out1)


ch5 = Dense(1, activation=act, name='ch5')(ch1) #L Par (to L Front)
d5 = Dropout(n_do)(ch5)
lstm_out5 = LSTM(n_LSTM, activation=actLSTM, return_sequences=True)(d5)
de5 = Dense(n_DenseLSTM, activation=act)(lstm_out5)


ch6 = Dense(1, activation=act, name='ch6')(ch5) #R Par (to L Par)
d6 = Dropout(n_do)(ch6)
lstm_out6 = LSTM(n_LSTM, activation=actLSTM, return_sequences=True)(d6)
de6 = Dense(n_DenseLSTM, activation=act)(lstm_out6)

model_neural = Model(inputs=main_input,
outputs=[ch1, ch2, ch3, ch4, ch5, ch6, ch7, ch8])

opt_neuro = Adam(lr=lr_n, decay=decay_n) #optimizer
```

```
NAME = f"Model-BioNN-neural-{int(time.time())}"


tensorboard_p = TensorBoard(log_dir="logs/{}".format(NAME))

filepath_p = "Predictor_BioNN-Final-{epoch:02d}"
checkpoint_p = ModelCheckpoint("models/{}.model".format(filepath_p,
verbose=0, save_best_only=True, mode='max'))

stopping_p = EarlyStopping(patience=200, verbose=0)
reduce_lr_p = ReduceLROnPlateau(factor=0.2,patience=5, min_lr=0)

loss_weights_ = [0.4732719 , 0.17619088, 1.        , 0.55852309, 0.38801036,
       0.60008576, 1.        , 0.78297974]

model_neural.compile(optimizer=opt_neuro,
loss={'ch1': loss_fc, 'ch2': loss_fc, 'ch3': loss_fc,
'ch4': loss_fc,'ch5': loss_fc, 'ch6': loss_fc, 'ch7': loss_fc,'ch8': loss_fc},
loss_weights={'ch1': loss_weights_[0], 'ch2': loss_weights_[1],
'ch3': loss_weights_[2], 'ch4': loss_weights_[3],
'ch5': loss_weights_[4], 'ch6': loss_weights_[5],
'ch7': loss_weights_[6], 'ch8': loss_weights_[7]},
metrics=[loss_fc])



# reshape each target for individual input:
ch1 = np.reshape(y_train_Neural[0,:,0],(1,input_len,1))
ch2 = np.reshape(y_train_Neural[0,:,1],(1,input_len,1))
ch3 = np.reshape(y_train_Neural[0,:,2],(1,input_len,1))
ch4 = np.reshape(y_train_Neural[0,:,3],(1,input_len,1))
ch5 = np.reshape(y_train_Neural[0,:,4],(1,input_len,1))
ch6 = np.reshape(y_train_Neural[0,:,5],(1,input_len,1))
```

```
ch7 = np.reshape(y_train_Neural[0,:,6],(1,input_len,1))
ch8 = np.reshape(y_train_Neural[0,:,7],(1,input_len,1))

ch1_v = np.reshape(y_validation_Neural[0,:,0],(1,input_len,1))
ch2_v = np.reshape(y_validation_Neural[0,:,1],(1,input_len,1))
ch3_v = np.reshape(y_validation_Neural[0,:,2],(1,input_len,1))
ch4_v = np.reshape(y_validation_Neural[0,:,3],(1,input_len,1))
ch5_v = np.reshape(y_validation_Neural[0,:,4],(1,input_len,1))
ch6_v = np.reshape(y_validation_Neural[0,:,5],(1,input_len,1))
ch7_v = np.reshape(y_validation_Neural[0,:,6],(1,input_len,1))
ch8_v = np.reshape(y_validation_Neural[0,:,7],(1,input_len,1))


history = model_neural.fit({'main_input': x_train_Task},
{'ch1': ch1 , 'ch2': ch2, 'ch3': ch3 , 'ch4': ch4,
'ch5': ch5 , 'ch6': ch6, 'ch7': ch7 , 'ch8': ch8},
epochs=epoch1,
batch_size=b_size,
validation_data=(x_validation_Task, [ch1_v,ch2_v,ch3_v,ch4_v,
ch5_v,ch6_v,ch7_v,ch8_v]),
verbose=1,
shuffle=False,
callbacks=[tensorboard_p, checkpoint_p])


score_neural = model_neural.evaluate(x_validation_Task, [ch1_v,ch2_v,ch3_v,
ch4_v,ch5_v,ch6_v,ch7_v,ch8_v], batch_size=b_size)

print(score_neural)


model_neural.save("saved_models/{}".format(NAME))
model_neural.summary()
```

```
pred_neuro = model_neural.predict(x_test_Task)

pred_neuro_np = np.reshape(pred_neuro,(8,n_)).transpose()

print(pred_neuro_np[0,:])


#Classifier architecture

model = Sequential()

model.add(Dense(8, input_dim=8, activation=actDense_c))
model.add(Dropout(n_do_c))
model.add(Dense(n_Dense, activation=actDense_c))
model.add(Dense(n_Dense, activation=actDense_c))
model.add(Dense(1, activation=actOut_c))

opt = Adam(lr=lr_c, decay=decay_c)

model.compile(loss=loss_fc_c,
              optimizer=opt,
              metrics=[loss_fc_c])


NAME = f"Model-BioNN-{int(time.time())}"

tensorboard_c = TensorBoard(log_dir="logs/{}".format(NAME))

filepath_c = "Classifier_BioNN-Final-{epoch:02d}"
checkpoint_c = ModelCheckpoint("models/{}.model".format(filepath_c,
verbose=0, save_best_only=True))
```

```
history_c = model.fit(x_train_Neural, y_train_Behavior,
          epochs=epoch2,
          batch_size=b_size_c,
          validation_data=(x_validation_Neural, y_validation_Behavior),
          verbose=1,
          shuffle=False,
          callbacks=[tensorboard_c, checkpoint_c])

score = model.evaluate(x_validation_Neural, y_validation_Behavior,
batch_size=b_size_c)



model.save("saved_models/{}".format(NAME))
model.summary()


pred_neuro_np = minmax_scale(pred_neuro_np,feature_range=(x_test_Neural.min(),
x_test_Neural.max()), axis=0)

pred_behavior = model.predict(pred_neuro_np)

pred_behavior_int = pred_behavior.transpose()
pred_behavior_int = np.round(pred_behavior_int)

print(pred_behavior_int) #Predicted behavior
print(y_test_Behavior) #Baseline behavior
p_binary=((pred_behavior_int==y_test_Behavior).sum()/y_test_Behavior.shape[0])
p_binary=p_binary*100
print("percentage of successful predictions: %.2f%%" % (p_binary))
```

```
def perf_measure(y_actual, y_hat):
    TP = 0
    FP = 0
    TN = 0
    FN = 0

    for i in range(len(y_hat)):
        if y_actual[i]==y_hat[i]==1:
           TP += 1
        if y_hat[i]==1 and y_actual[i]!=y_hat[i]:
           FP += 1
        if y_actual[i]==y_hat[i]==0:
           TN += 1
        if y_hat[i]==0 and y_actual[i]!=y_hat[i]:
           FN += 1

    return(TP, FP, TN, FN)

#TP, FP, TN, FN = perf_measure([0,1], [0,1])

y_actual = pred_behavior_int
y_actual = np.reshape(y_actual,(y_actual.shape[1])).astype(bool)
y_hat = y_test_Behavior.transpose().astype(bool)
TP, FP, TN, FN = perf_measure(y_actual, y_hat)
print('True Positives: %.2f' % (TP))
print('False Positives: %.2f' % (FP))
print('True Negatives: %.2f' % (TN))
print('False Negatives: %.2f' % (FN))



print('Neural scores on evaluation:')
```

```
for n in range(len(score_neural)):
    print("%s: %.2f" % (model_neural.metrics_names[n], score_neural[n]))
print('Final scores on evaluation:')
for n in range(len(score)):
    print("%s: %.2f" % (model.metrics_names[n], score[n]))



#%matplotlib widget
get_ipython().run_line_magic('matplotlib', 'inline')

#Predictor
# list all data in history
print(history.history.keys())

# summarize history for loss
fig1 = plt.figure()
plt.plot(history.history['loss'])
plt.plot(history.history['val_loss'])
plt.title('Neural model loss')
plt.ylabel('loss')
plt.xlabel('epoch')
plt.legend(['train', 'validation'], loc='upper left')
plt.savefig('Figs/Neural model loss.pdf')
plt.show()

#Classifier
# list all data in history
print(history_c.history.keys())

fig2 = plt.figure()
plt.plot(history_c.history['loss'])
plt.plot(history_c.history['val_loss'])
```

```
plt.title('Final model loss')
plt.ylabel('loss')
plt.xlabel('epoch')
plt.legend(['train', 'validation'], loc='upper left')
plt.savefig('Figs/Final model loss.pdf')
plt.show()



#Plot 1 Signal (quick check)
get_ipython().run_line_magic('matplotlib', 'inline')

ch_ = 0

X_pred = np.arange(0, n_, 1)
X_pred = np.reshape(X_pred,(n_))
X_test = X_pred

Y_pred = pred_neuro_np[:,ch_]
Y_test = x_test_Neural[:,ch_]

fig5 = plt.figure()
ax = plt.axes()
ax.set(xlabel='events', ylabel='neural activity',
title='predictions 1 channel')
plt.plot(X_pred, Y_pred, alpha=1)
plt.savefig('Figs/predictions 1 channel.pdf')
plt.show()


fig6 = plt.figure()
ax = plt.axes()
ax.set(xlabel='events', ylabel='neural activity',
```

```
title='test set 1 channel')
plt.plot(X_test, Y_test, alpha=1)
plt.savefig('Figs/test set 1 channel.pdf')
plt.show()


#%matplotlib inline

#Plot all signals
Y_p = minmax_scale(pred_neuro_np,feature_range=(0, 1), axis=1).transpose()
Y_t = minmax_scale(x_test_Neural,feature_range=(0, 1), axis=1).transpose()

def heatmap2d(arr: np.ndarray, str_title):
    ax = plt.axes()
    ax.set(xlabel='events',
    ylabel='activity',
    title=str_title)
    im = plt.imshow(arr, cmap='viridis', interpolation='bicubic',
    aspect='auto')
    cb = plt.colorbar()
    cb.set_label('intensity')
    plt.savefig('Figs/'+str_title+'.pdf')
    plt.show()


fig7 = plt.figure()
heatmap2d(Y_p, 'Gradient map (interpolated) - Predicted')

fig8 = plt.figure()
heatmap2d(Y_t, 'Gradient map (interpolated) - Test')


#Plot regions by regions
```

```
def heatmap2d_b(arr: np.ndarray, str_title):
    ax = plt.axes()
    ax.set(xlabel='events',
    ylabel='Channels',
    title=str_title)
    plt.imshow(arr, cmap='viridis', interpolation='bicubic',
    aspect='auto')
    #plt.imshow(arr, cmap='viridis')
    cb = plt.colorbar()
    cb.set_label('intensity')
    plt.savefig('Figs/'+str_title+'.pdf')
    plt.show()

fig7b = plt.figure()

Fp_p = minmax_scale(pred_neuro_np[:,0:2],feature_range=(0, 1),
axis=1).transpose()
heatmap2d_b(Fp_p, 'Fp Left,Right - Predicted')

Fp_t = minmax_scale(x_test_Neural[:,0:2],feature_range=(0, 1),
axis=1).transpose()
fig8b = plt.figure()
heatmap2d_b(Fp_t, 'Fp Left,Right - Test')

C_p = minmax_scale(pred_neuro_np[:,2:4],feature_range=(0, 1),
axis=1).transpose()
heatmap2d_b(C_p, 'Central - Predicted')

C_t = minmax_scale(x_test_Neural[:,2:4],feature_range=(0, 1),
axis=1).transpose()
fig8b = plt.figure()
heatmap2d_b(C_t, 'Central - Test')
```

```
Par_p = minmax_scale(pred_neuro_np[:,4:6],feature_range=(0, 1),
axis=1).transpose()
heatmap2d_b(Par_p, 'Parietal Left,Right - Predicted')

Par_t = minmax_scale(x_test_Neural[:,4:6],feature_range=(0, 1),
axis=1).transpose()
fig8b = plt.figure()
heatmap2d_b(Par_t, 'Parietal Left,Right - Test')

Occ_p = minmax_scale(pred_neuro_np[:,6:8],feature_range=(0, 1),
axis=1).transpose()
heatmap2d_b(Occ_p, 'Occipital Left,Right - Predicted')

Occ_t = minmax_scale(x_test_Neural[:,6:8],feature_range=(0, 1),
axis=1).transpose()
fig8b = plt.figure()
heatmap2d_b(Occ_t, 'Occipital Left,Right - Test')



#Generate outputs based on training targets for evaluation:
output_tr_neuro = model_neural.predict(x_train_Task)
output_tr_neuro_np = np.reshape(output_tr_neuro,(8,n_)).transpose()



#%matplotlib inline

#Plot all signals (training)
Y_p = minmax_scale(output_tr_neuro_np,feature_range=(0, 1), axis=1).transpose()
Y_t = minmax_scale(x_train_Neural,feature_range=(0, 1), axis=1).transpose()
```

```
def heatmap2d(arr: np.ndarray, str_title):
    ax = plt.axes()
    ax.set(xlabel='events',
    ylabel='activity',
    title=str_title)
    im = plt.imshow(arr, cmap='viridis', interpolation='bilinear',
    aspect='auto')
    cb = plt.colorbar()
    cb.set_label('intensity')
    plt.savefig('Figs/'+str_title+'.pdf')
    plt.show()


fig7 = plt.figure()
heatmap2d(Y_p, 'Gradient map (interpolated) - Train output')

fig8 = plt.figure()
heatmap2d(Y_t, 'Gradient map (interpolated) - Train set')


def explained_variance(arr1,arr2,str_):
    cor_arr = np.corrcoef(arr1, arr2,rowvar=str_)
    cor_arr_output_target = cor_arr[8:,:8]
    diag_ = np.diag(cor_arr_output_target)
    p_var = (diag_**2)*100
    p_var[np.isnan(p_var)] = 0
    print("variance explained by each output channel:")
    print(p_var)
    print("highest variance explained by outputs: %.2f%%" % (p_var.max()))
    ch_ = np.where(p_var==p_var.max())[0][0]+1
    print("EEG Channel: ", ch_)
    return cor_arr, ch_
```

```
cor_arr, ch_ = explained_variance(Y_p,Y_t,True)



#Plot 1 Signal (quick check)
get_ipython().run_line_magic('matplotlib', 'inline')

ch_ = ch_-1

X_pred = np.arange(0, n_, 1)
X_pred = np.reshape(X_pred,(n_))
X_test = X_pred

Y_pred = pred_neuro_np[:,ch_]
Y_test = x_test_Neural[:,ch_]

fig5 = plt.figure()
ax = plt.axes()
ax.set(xlabel='events', ylabel='neural activity',
title='predictions channel '+ch_.astype(str))
plt.plot(X_pred, Y_pred, alpha=1)
plt.savefig('Figs/predictions 1 channel.pdf')
plt.show()


fig6 = plt.figure()
ax = plt.axes()
ax.set(xlabel='events', ylabel='neural activity',
title='test set 1 channel '+ch_.astype(str))
plt.plot(X_test, Y_test, alpha=1)
plt.savefig('Figs/test set 1 channel.pdf')
plt.show()
##################################################
```

C. Glossary

A (non-alphabetic) glossary condensing some study notes in the context of machine learning.

- **Activation Function**. An activation function, as the name suggests, is used by a unit in a neural network to decide what the activation value of the unit should be based on a set of input values. The activation value of many such units can then be used to make a decision based on the input (classification) or predict value of some variable (regression).

  The activation functions are typically non-linear. Non-linear mappings applied to inputs are able to capture interesting properties of the input.

  There are different types of units based on the activation functions like sigmoid units, rectified linear units (ReLU), tanh units.

  - 1. Sigmoid maps input to a value in the range 0 to 1.
  - 2. Tanh maps the input to a value in the range -1 to 1.
  - 3. ReLU maps the input x to max(0,x), i.e. it maps negative inputs to 0 and positive inputs are output without any change.

- **Saddle point**. A saddle point admits a local minimum in one axis and a local maximum in another (hyperbolic or paraboloid).

- **Gradient map**. A gradient map, or heat map, gives the direction of the change and the strength or magnitude of the change. It is a convenient visual representation of tensors or scalar fields, showing the rate of change and direction (e.g. vectors pointing to red area as the highest values).

- **Jacobian**. The Jacobian gives the best linear approx of a distorted figure at a point x (partial derivatives). It is the determinant of the Jacobian matrix. Since matrix represents the coefficients in a systems of linear equations, the Jacobian is the scaling factor of the transformation of the matrix i.e. if V maps to W, addition and scalar multiplications are preserved by this factor. Scalar product changes the scale (or magnitude) of the vector, not its direction (see inner or dot products of 2 vectors).

- **MSED**. Minimum Square Cartesian Distance: a Cartesian system being an euclidian space with coordinates, the euclidean distances can be squared and the smallest is the MSED.
- **SSE**. Sum of Squares Due to Error. A value closer to 0 indicates that the model has a smaller random error component, and that the fit will be more useful for prediction.
- **R-Square**. Because R-square is defined as the proportion of variance explained by the fit, if the fit is actually worse than just fitting a horizontal line then R-square is negative.
- **Degrees of Freedom Adjusted R-Square**. The adjusted R-square statistic can take on any value less than or equal to 1, with a value closer to 1 indicating a better fit
- **Principal Component Analysis (PCA)**. Useful if the data has high variance. The Principal Components are equivalent to an orthogonal transformation that preserves the inner product of the linear transformation of the data points. PCA reduces the dimensionality of the data, useful for denoising.
- **Tensor**. A tensor is a geometric object that describes a linear relation between vectors.
- **Estimator**. An underlying function estimating the model i.e. its expected outputs. Overfitting occurs when the model provided by the estimator is too complex while underfitting occurs when the model is not sophisticated enough. In that case, a deviation to the true estimator is observed, and large.
- **Standardization**. Standardisation is not normalization. Standardisation is centered to mean=0 and SD=1, following a normal distribution. It is a sub-class of normalization.
- **Predictor values**. Refers to independent variables or input values to the network.

- **Regularization**. Regularization reduces the variance of the estimator by increasing its bias, so that error decreases.
- **Hessian matrix**. The Hessian matrix contains the second partial derivatives to determinate the local min or max or saddle point of a surface, thus helping the search of the direction to go for the gradient descent.
- **Bayesian inference**. Inferring the posterior probability based on the antecedents ("likelihood" Baye's function).
- **Probability mass function / probability density function**. Functions that give the probability of a variable value in a discrete / continuous distribution.
- **Dynamic Causal Modeling**. DCM describes how dynamics are manifest in the data, what is the physical-causal mechanism.
- **Perceptron**. An object, with features and associated weights, indicating their importance.