Politecnico di Torino

POLITECNICO DI TORINO

Master's Degree in Biomedical Engineering
A.a. 2022/2023
October 2023

Time robustness of deep learning models for real-time neural decoding of arm movement

Master's Degree Thesis

Supervisors:
Prof. Gabriella Olmo

Co-supervisor:
Dr. Paolo Viviani

Candidate:
Marta Bono (s292185)

List of Figures

List of Tables

Abstract

In the past years, it has been proven that the use of AI algorithms is useful to improve the control on Brain-Machine Interfaces (BMI). Through the search for patterns in the neural signal by Machine Learning (ML) algorithms, it was possible to derive information about the brain activity, leading leading to a greatly improved accuracy in neural decoding. BMIs were proposed that allowed people to compose sentences by selecting letters on a virtual keyboard, others that allowed people to control chronic pain through stimulation of the brain in a virtual environment, and others that allowed control of a robotic arm. However, it has been proven that neural signals change over time, mainly due to neuroplasticity, but also due to inflammatory processes and the displacement of electrode microarrays in the case of intracortical signal retrieval (ECoG). Unfortunately, the change in signals inevitably leads to a deterioration in decoding capabilities and performance and it is still an open research problem.

The aim of this Master Thesis work is to find an optimal Transfer Learning (TL) strategy that maintain high decoding performance and increase robustness without requiring a large number of recording sessions to re-train the algorithm. This work fits in the context of a larger project aiming building a BMI to let a Non-Human Primate (NHP) control a robotic arm.

The first step was to find a public dataset that provided several recordings of an NHP performing a task involving an arm or hand movement. In particular, it was essential that the neural signals were ECoG retrieved in the primary motor cortex (M1) and that several recording sessions were provided over a longer or shorter period. Four different datasets were analysed and the Zenodo dataset was chosen, which provided 37 sessions of recording an NHP while reaching a random target in the transversal plan, allowing for the analysis of coherent neural data over a long time span. Subsequently, it was necessary to apply pre-processing techniques to derive multi-unit activity (MUA) signals, spike counts and hand velocity. Next, three neural networks (Bidirectional Recurrent Neural Network, Bidirectional Long Short-Term Memory, and Bidirectional Gated Recurrent Unit) were trained and optimized on the signals of the first recording session, in terms of root-mean squared error (RMSE) and Pearson's Correlation Coefficient (CC). This

model was used to predict the hand velocity signal for all the following sessions; therefore, with no adaptation to the variability of the signal in time, the result represented the lower bound for the decoding performance of the model through time. After that, the upper bound of the decoding performance was obtained by training a dedicated model on each single session, maintaining the weights of the previous training session.

At this point, several TL strategies were evaluated to explore the spectrum in between the previous approaches. Most strategies involved re-training the model on a small amount of data less frequently that at each session, to reduce the need for labelled data. Eventually, a self-supervised approach was proposed to evaluate if a solution that did not require kinematic signals as labels could achieve comparable performance to supervised learning approaches.

Contents

Chapter 1

Introduction

Nowadays, numerous people suffer from complete or incomplete spinal cord injuries (SCI) or from lower motor neurons diseases (MNDs), such as amyotrophic lateral sclerosis (ALS). Both situations lead to the loss of control of the upper and lower limbs movements. Recent studies have proved how the use of Deep Learning (DL) in Brain-Machine Interfaces (BMI) improves the translation of neural signals into control commands for prostheses or exoskeleton compared to the use of traditional machine-learning methods. However, the analysis of the performances of the Artificial Intelligence (AI) algorithm over the time is still an open research problem. This is a crucial step, since the electroencephalographic signal can change very quickly due to neuroplasticity, displacement of the microelectrode array, inflammatory processes, etc. Usually, changes in the electroencephalographic signal can lead to performance degradation of the AI algorithm. To prevent this, Transfer Learning (TL) technique can be used. Transfer Learning is a supervised technique that reuses parts of a previously trained model on a new network. It is, in simple terms, the ability to "transfer" one network knowledge to a new network.

This work intends to identify is to find an optimal Transfer Learning strategy approach for enabling a non-human primate (NHP) to operate a prosthetic hand with two degrees of freedom (DOF) overtime. The primary objective is to overcome the challenge of electroencephalographic signal changes experienced over time during the use of the prosthetic hand and to increase the generalization skills of the AI model.

1.1 Introduction to the neuromuscular system

The neuromuscular system is responsible for the voluntary muscle contraction. It itself is composed by two main actors: the Nervous System (NS) and the skeletal muscles. Together with the skeletal system, they are responsible for the voluntary

movements in vertebrates.

1.1.1 Anatomy and physiology of the Nervous System

The Nervous System is composed by two main parts:

- The Central Nervous System (CNS), composed by the brain and the spinal cord. It is the body's processing centre, so it receives and elaborates the information coming from the body's receptors and it creates the response in the form of a nerve depolarization, known as Action Potential (AP).
- The Peripheral Nervous System (PNS), composed by nerves that branch off from the spinal cord and extend to all parts of the body. It collects sensory information coming from the receptors and transfer them to the CNS. It also does transmit the AP from the CNS to the skeletal muscles through specific neurons called motor neurons.

The base unit of the nervous system is the nerve cell or neuron. As shown in Figure 1.1, it is composed by four main elements:

- Cell body or soma: it contains the nucleus of the cell and it is the place in which the protein synthesis occurs.
- Dendrites: they are the extensions of the cell with many branches. This part of the neuron is responsible for receiving the stimulus or AP.
- Axons: it is the longest part of the neuron. It can be seen as a cable which propagates the stimulus from the dendrites to the axon terminals. It can be covered by myelin, a material produced by oligodendrocytes (in the CNS) or by the Schwann cells (in the PNS). The myelin layer is not continuous, but it is interrupted in points called nodes of Ranvier. This discontinuity increases the conduction velocity of the AP through the axon thanks to the saltatory conduction. If the axon has myelin, the conduction velocity is generally about 50-70 m/s, but it can be up to 120 m/s.
- Axon terminals: it is the terminal part of the neuron. It is the portion of neuron in which the synapses occur: synaptic boutons are structures where neurotransmitters are released to communicate with the target neurons.

One way to classify neurons is based on their function. Sensory neurons, also called afferent neurons, conduct signals from sensory receptors to the brain or the spinal cord. Motor neurons, also known as efferent neurons, propagate the stimulus from the brain and spinal cord to the muscle fibers, resulting in muscle contraction. Interneurons connect two different neurons within specific regions on the CNS, forming the neural circuit.

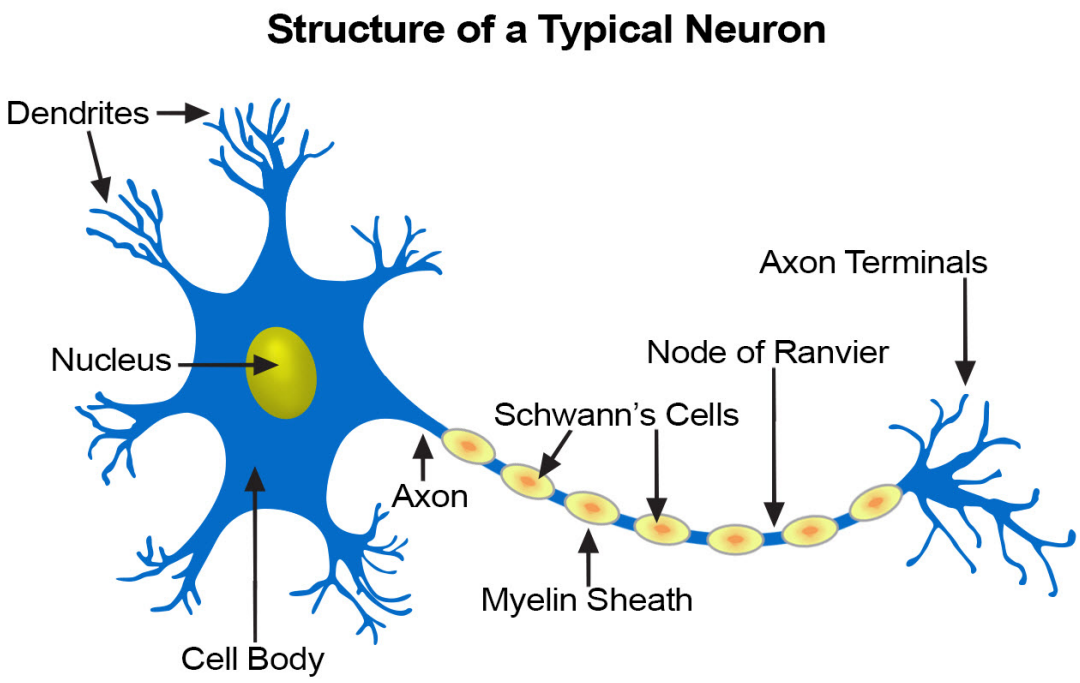


Figure 1.1: Structure of the nerve cell. Figure taken from [1]

Anatomy and physiology of the Brain

The brain is the most complicated yet important part of the human body and it is part of the CNS. It is composed by more than 100 billion of neurons and other cells (such as oligodendrocytes and astrocytes) and blood vessels. As shown in Figure 1.2, the brain is divided in three parts: cerebrum, cerebellum, and brainstem.

The cerebrum is the largest part of the brain and it is divided into the left and the right hemispheres, linked by the corpus callosum. It is characterized by many folds, known as gyri and sulci. The external part is called cerebral cortex and it is composed by grey matter, while the inner subcortical portion is made of white matter. The motor and sensory neurons coming from one hemisphere control the specular part of the body: for example, the right arm movement is controlled by the left hemisphere. This is because the motor and sensory neurons cross to the opposite side in the brainstem. Both the hemispheres are composed by four lobes:

- Frontal lobe: it is responsible for the motor function (motor cortex), language (Broca area) and all the cognitive processes, such as mood, memory, and attention.
- Parietal lobe: it is responsible for elaborating the sensory signals, such as taste, touch, pain, and proprioception.

- Temporal lobe: it is important for the elaboration of language (Wernicke area), and to retain memory and emotions.

- Occipital lobe: it is responsible for the interpretation of visual information.

The cerebellum is the lower part of the brain and it is responsible for the motor coordination, posture, and balance. It increases the precision and accuracy of the motor activity. The brainstem contains the midbrain, pons, and medulla. It is responsible for the regulation of autonomic functions, such as breathing, temperature regulation, heart rate, etc.

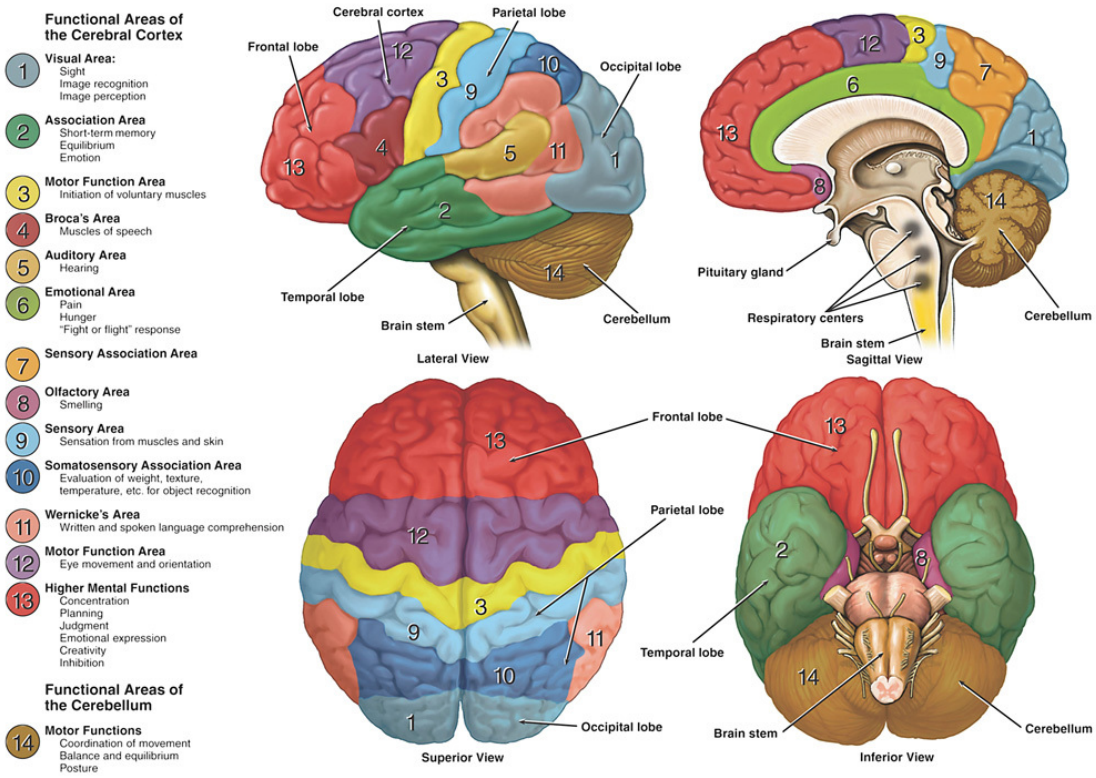


Figure 1.2: Structure and functions of the brain. Figure taken from [2]

The motor cortex, a crucial portion of the cerebral cortex, is responsible for the planning, control, and execution of voluntary movements. Several studies have been conducted to identify which areas of the cortex generate movement for specific body parts. Among these studies, one of the most significant is the creation of a "cortical homunculus" by Penfield and Boldrey [18], which serves as a map for the motor cortex. This map was established through cortical stimulation on 126 patients and represents the amount of cortical area dedicated to motor or somatosensory functions of each body part, as shown in Figure 1.3. The primary motor cortex

(M1) is the specific area responsible for generating the neural impulses required for executing movement.

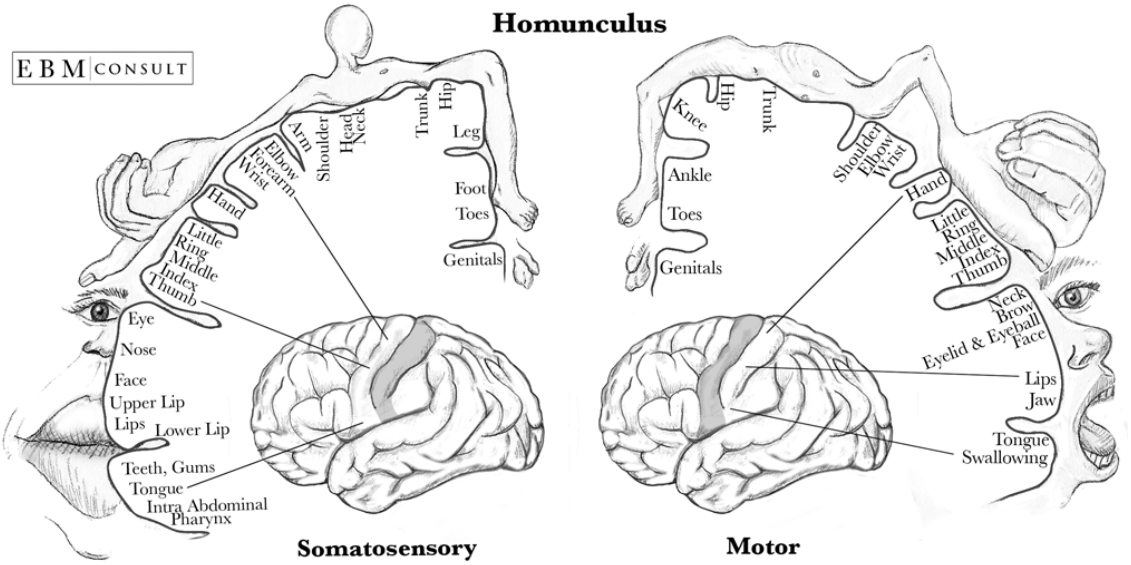


Figure 1.3: Representation of the cortical homunculus. Figure taken from [3]

One of the important processes of the brain is called neuroplasticity or brain plasticity. It is defined as the ability of the nervous system to change its activity or reorganize its structure in response of different stimuli. Neuroplasticity is a very common phenomenon, especially during childhood, when most of the abilities are acquired. It can change through growth and reorganization of the structures. Changes can range from new connection of singular neurons to wider adjustments like cortical remapping. Neuroplasticity is one of the crucial aspects to consider when talking about brain signal analysis, as it is one of the main factors of the change of signals themselves.

1.1.2 Different signals from the Nervous System

As was previously said, the stimulus in the Nervous System is created and then transmitted through a specific electrical signal called Action Potential. The detection of the AP is an essential indicator for many applications. It is useful for understanding the neural communication, diagnosing neural disorders, mapping neural circuits and much more. Since the CNS has about 100 billion of neurons, it is not possible to record the activity of each neuron individually, so the depolarization of one or more population of neurons is recorded instead. There are several methods of signal acquisition:

- Electroencephalography (EEG): it is a non-invasive method to record the activity of the brain. It requires numerous electrodes placed on the scalp using, typically, the International 10-20 system. The resulting signal, called electroencephalography, has normally an amplitude that ranges between 20 and 100 μV and a spectrum between 1 and 30 Hz. Depending on the frequency of the signal, it is possible to derive the state of activity of the brain: alpha (8-13 Hz) for a state of relaxed wakefulness, beta (13-30 Hz) for intense mental activity, delta (0.5-4 Hz) and theta (4-7 Hz) for deep sleep or for neural dysfunction. EEG signal is often used to diagnose diseases such as epilepsy, sleep disorders, vegetative states, or brain death. An example of normal and pathological EEG is shown in Figure 1.4
- Electrocorticography (ECoG): also known as intracranial electroencephalography (iEEG), it is an invasive method for recording the brain activity. It requires the electrodes to be placed directly on the interested area of the motor cortex, an operation (craniotomy) is therefore necessary. Through this technique, it is possible to retrieve different signals: Local Field Potentials (LFP), Multi-Unit Activity (MUA) or Single-Unit Activity (SUA). Particularly, MUA derive from band-passing the raw signal in the high frequency region (300 to 6000 Hz) and it has been recently considered rich of information for the motor-related brain activity [19]
- Magnetoencephalography (MEG): it is a non-invasive functional neuroimaging technique for mapping the brain activity. It records the magnetic fields produced by the current generated by the AP.

1.1.3 Anatomy and physiology of skeletal muscles

If the nervous system is responsible for generating and transmitting the stimuli of movement, the muscular system is the actuator that enables movement through contraction. The skeletal muscle is one of the three muscle tissues in the human body and it is responsible for the actuation of the voluntary movements. Typically, it is connected to two bone segments by tendons and performs the desired movement by contracting.

From the structural point of view, skeletal muscle tissue consists of bundles of muscle fibers called fasciculi, which are covered in a connective tissue called perimysium. The innermost sheath surrounding each muscle fiber is known as endomysium. Each muscle fiber contains myofibrils comprised of multiple myofilaments, made up of the proteins actin and myosin. The orderly arrangement of these protein fibers within the myofibrils is called a sarcomere, and it forms the fundamental contractile unit of skeletal muscles. When a muscle receives a stimulus, its protein fibers begin to slide over one another, resulting in muscle contraction.

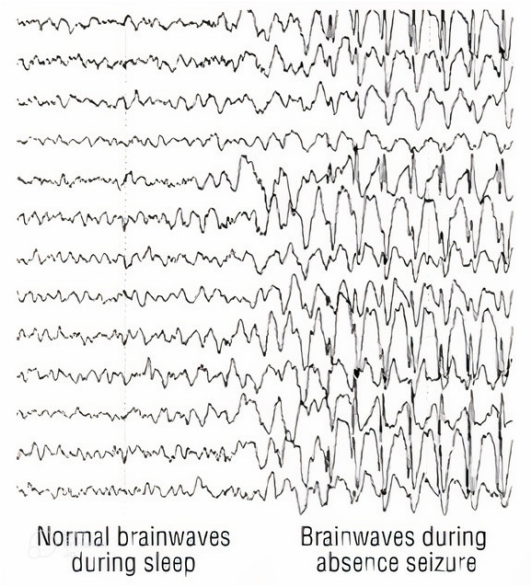


Figure 1.4: Example of normal and pathological EEG signal. Figure taken from [4]

There are two types of muscle fibers: white fibres are used to generate a movement that requires a type of impulsive force, therefore with high strength and short duration; red fibres, on the other hand, are recruited when the movement to be performed is of long duration, as the resistance of the red fibres is greater than that of the white fibres.

1.1.4 Motor unit and voluntary muscle contraction

While the neuron and sarcomere serve as the base unit for the Nervous and Skeletal Muscle System respectively, the Neuromuscular System operates through the motor unit (MU), which involves three elements:

- Motor neuron, from the Nervous System;
- Muscle fibers, from the and Skeletal Muscle System;
- Neuromuscular junction, formed by the terminal axons of motor neurons connecting to multiple muscle fibers.

Different types of motor units exist and each of them has a specific role: smaller MUs are used to actuate precise movements that do not require high force; bigger MUs are stimulated to perform movements that require higher forces and smaller precision. There are also MUs made of white fibers, red fibers or both.

When a healthy patient wants to move a body part, a stimulus is created in the motor cortex. This action potential is then propagated through the neurons to the

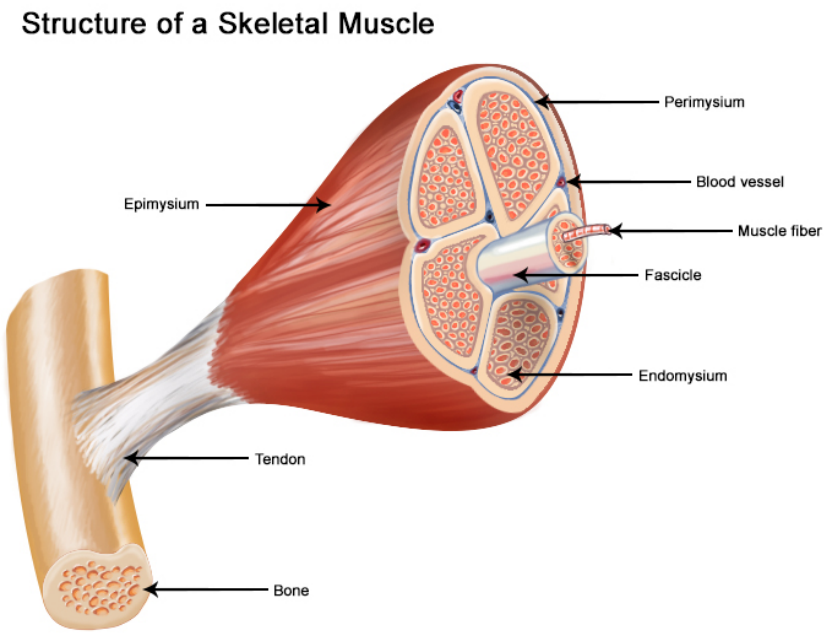


Figure 1.5: Structure of the skeletal muscle tissue. Figure taken from [5]

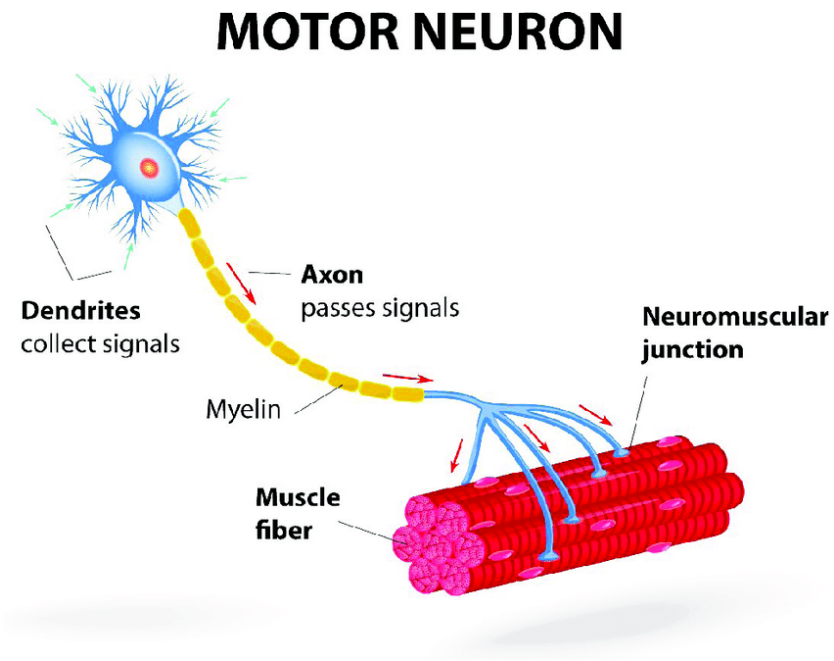


Figure 1.6: Elements of a motor unit. Figure taken from [6]

motor neurons, which stimulates the respective muscles fibers in order to create the contraction and the movement.

Therefore, there are different levels involved in a contraction, from the highest to the lowest:

- Central level: composed by the brain, for the creation and improvement of the stimulus of the movement.
- Propagation level: composed by the spinal cord and motor neurons, for the propagation of the stimulus.
- Actuation levels: composed by the muscles, for the actuation of the movement.

1.1.5 Motor impairments

Motor disabilities can be categorized into two types based on their origin: traumatic or pathological. Most traumatic motor impairments result from Spinal Cord Injuries (SCI), which occur when the spinal cord is completely or partially damaged. The level of the injury on the spinal cord determines whether the resulting condition is tetraplegia or paraplegia. Tetraplegia, or quadriplegia, is caused by high-level cervical nerve damage and results in paralysis of most of the body. On the other hand, paraplegia is caused by lower-level injury and affects only the lower body. In Figure 1.7 is it possible to see the vertebrae classification and the respective area of sensation.

Motor disabilities can be caused by various pathologies that affect different organs. Among these are neuromuscular diseases, which specifically target the peripheral nervous system (PNS), neuromuscular junction, or skeletal muscle. Examples of such conditions include Amyotrophic Lateral Sclerosis (ALS) and Spinal Muscular Atrophy (SMA), which both cause deterioration of the motor neurons.

In these scenarios, the motor cortex generates the appropriate signals, yet they fail to reach the affected muscles due to a disruption in the pathway. Therefore, restoring limb functionality necessitates employing a prostheses or an exoskeleton by bypassing the damaged area. This involves registering the signals from the motor cortex and converting them into prostheses commands, a technique known as Brain-Machine Interfaces.

1.2 Brain-Machine Interfaces (BMIs)

Brain-Machine Interface (BMI), also known as Brain-Computer Interface (BCI), can be defined a "system that measures CNS activity and converts it into artificial output that replaces, restores, enhances, supplements, or improves natural CNS output and thereby changes the ongoing interaction between the CNS and its external or internal environment."[20] Hence, a BMI is a direct link between the CNS

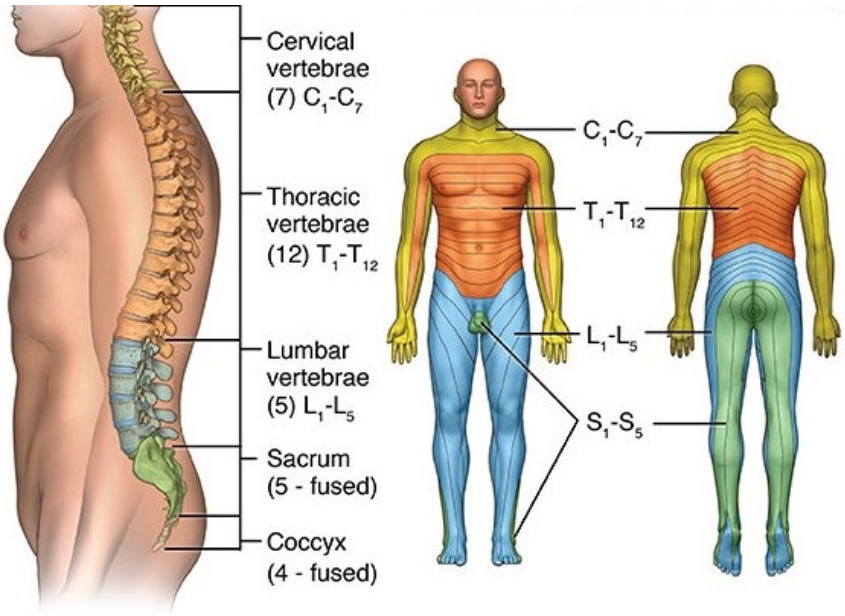


Figure 1.7: Classification of vertebrae and their areas of sensation. Figure taken from [7]

and external device, most commonly a computer or a prosthetic limb. As outlined in its definition, BMIs in medicine aim to achieve the following objectives:

- Replacing a natural output; for instance, restoring the ability to speak.
- Restoring a natural output; for instance, regaining control of a paralyzed limb or bladder function.
- Enhancing a natural CNS output; for instance, detecting and signalling a lack of attention during a continuous task.
- Supplementing a natural CNS output; for instance, controlling an additional limb.
- Improving a natural CNS output; for instance, stimulating muscles in stroke patients by combining BCI to Functional Electrical Stimulation (FES).

Brain-Machine Interfaces can be classified as follow:

- Non-invasive: this type of BMIs uses a non-invasive method of electrophysiological recordings, such as electroencephalogram (EEG), magnetoencephalogram (MEG) or functional magnetic resonance imaging (fMRI). EEG-based BMIs are the most common and typically used in two primary applications:

  - Basic communication, utilizing P300 evoked potential or steady-state visual evoked potentials. Neither require patient training and can be improved with AI, but these technologies are not particularly fast.
  - Providing users with feedback, particularly when integrated with Virtual Reality technology. This is often used for pain management. Using a non-invasive technique, this type of BMIs does not require a neurosurgical implantation. However, EEG signals have limited bandwidth.

- Invasive: This type of BMI utilizes invasive methods for electrophysiological recordings, such as electrocorticogram (ECoG) signals. Local field potentials (LFP) and single- or multi-unit signals can be detected with ECoG. This solution requires subdural electrodes, which require neurosurgical implantation. It is possible to record signals from a single or multiple recording sites. Most single-site BMIs place microelectrodes in the primary motor cortex (M1) or posterior parietal cortex (PP).

BMIs can also be categorized as unidirectional or bidirectional systems. Unidirectional BMIs enable one-way communication, either collecting data from the brain to trigger an action or delivering stimulation to the brain in response to specific events. Bidirectional BMIs, on the other hand, allow for two-way communication, enabling both data collection and stimulation capabilities.

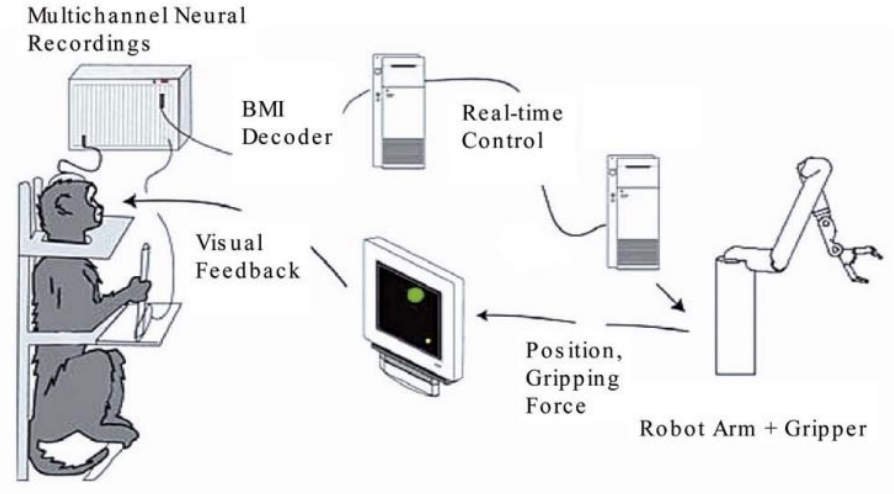


Figure 1.8: Example of BCI applied to a robotic arm. Figure taken from [8]

1.3 Transfer Learning Basics

Artificial Intelligence (AI) is becoming more and more important in our world thanks to its abilities and versatility. From image processing to text generation,

data analysis and prediction of future trends, AI can be trained and used to help in many different fields, such as economics, agriculture, smart houses, or health. However, one important limit of AI is its poor generalization ability, also known as low robustness. This can be a consequence of the lack of training data or the poor quality of the retrieved data. Hence, it is necessary to find new strategies to overcome this issue. In the last years, Transfer learning has emerged as a promising solution. Transfer Learning has not a single and univocal definition, but it can be described as a supervised learning technique that uses pre-trained model and re-train it on new data to solve new yet similar problems. For example, if a model has been trained to detect car images among a large set of images, through Transfer Learning techniques it is possible to speed up the process for the creation of a new truck recognition model. In the present work, Transfer Learning will be used to adapt the network to changes in input ECoG signals.

1.4 Contribution and context

1.4.1 B-CRATOS Project: wireless Brain-Connect interface TO machineS

The present work was carried out at the LINKS Foundation, a research center born from the collaboration between the Compagnia San Paolo and the Politecnico of Turin. The master thesis project is part of the B-CRATOS project, a European project whose objective is to develop an innovative wireless and bidirectional BMI system to stimulate various functions, mainly prostheses. B-CRATOS involves 7 different partners in 5 different countries and it is coordinated by University of Uppsala. B-CRATOS main objectives are:

- Proof-of-concept high-channel, high-speed, wireless brain implant capable of two-way communication without battery: This is crucial in order to sense high-resolution neural signals and precisely stimulating cortical targets. The biocompatible and hermetically-sealed implants are positioned in correspondence of the primary motor cortex (M1) and the somatosensory cortex (S1). The implanted electronics are responsible for the detection, amplification, and digitization of the neural activity and for the stimulation. Externally, a wearable module is used to provide the implants the power needed and high data rate wireless communication for transmitting the neural data stream.

- Fat-IBC: Develop a general-purpose, high-speed communications platform technology. This system is based on microwave propagation through the subdermal body fat.

- HPC based AI computing: Machine Learning and Deep Learning algorithms are used for classification and regression tasks for translating neural signals

into prosthesis commands.

- Biomechatronic prosthetic upper limb: B-CRATOS will use the 5-axis Mia robotic arm, which is a light, strong and fast prosthesis created from Prensilia s.r.l., a SME spin-off of Scuola Superiore Sant'Anna of Pisa.
- Artificial Skin: through a sensorized material that resemble human skin, it will be possible to detect a contact between the sensors and the object and a stimulus will be created. This will be transmitted to the implanted microelectrodes array that will stimulate the correct motor cortex area.

1.4.2 Contribution of this work

The master thesis project aims at finding the best solution to guarantee the correct usage of the BMI over the time by including Transfer Learning strategies within the AI algorithms task. Increasing the robustness of the algorithm over time, it will be possible to decrease the number of sessions of recording and to increase the acceptability and usability of the system by the patient.

Firstly, it was necessary to look for a public dataset that included neural and kinematic signals of a Non-Human Primate (NHP) while performing tasks that required arm movements. Then, multiple Neural Networks (NN) models have been trained on the pre-processed data in order to find the most performant one. At this point, several Transfer Learning strategies have been adopted to find the best one in terms of performances and number of sessions of re-training.

To summarise, the main contribution of this work are:

- Development of a decoding algorithm to perform regression of 2D velocity of finger movement based on MUA recordings from M1.
- Definition of a Fine-Tuning strategy to reduce the amount of labelled data needed to preserve the model accuracy over time.
- Evaluate whether the application of Self-Supervised Learning algorithms, which do not require any further recording sessions beyond the first one, can be a viable alternative solution to Supervised algorithms in terms of performance and robustness.

1.5 Acknowledgements

This work has been partially carried out on the MARCONI100 supercomputer provided by CINECA through the Italian Super-computing allocation scheme Iscra-C (project ID B-Cratos) and supported by H2020-FETOPEN B-CRATOS project

(No. 965044).

Chapter 2

Background research

2.1 Dataset search

In order to be representative of the scenario of B-Cratos project, and to allow us to extensively investigate the behaviour of decoding models through time, the dataset should have these characteristics:

- The dataset subject has to be a NHP.
- The dataset task has to include some type of hand or arm movement.
- The dataset had to include neural signals, in particular ECoGs, and kinetics signals relative to the hand/arm movement.
- No data pre-processing steps requiring offline processing must have been carried out on the neural data, or at least the raw data should be provided alongside the processed data. This is because the whole system have to work in real-time. An example of offline pre-processing is spike sorting.
- The microarray placement had to be in the motor cortex, even better if specifically placed in M1.
- The dataset had to include several recording sessions distribute over time, possibly reporting dates and time of recording.

It is important to emphasize that the choice of the task is not a crucial. In fact, it can be both a classification or a regression task, such as grasping of different types of objects or reaching random targets. However, it is necessary to have a large number of recording sessions carried out over a sufficiently long period of time. From a preliminary research, four public datasets were considered for further investigations: DRYAD [21], CRCNS [22], eBrains [10] and Zenodo [23]. Short names were assigned for convenience.

2.1.1 Dataset DRYAD

This dataset proposes three different combinations of tasks and acquired signals:

- Combination 1: the task consists of grasping of 35 different objects. On average, 10 trial per object were carried out per session. The signal was relative to the activity of individual neuron units (or single-unit activity). Monkey 1 and Monkey 2 were involved for this task, for a total of respectively 6 and 9 recording session. For each session, the depth of the microelectrodes was increased in order to register different neural populations.

- Combination 2: the task consists of grasping of 35 different objects. On average, 10 trial per object were carried out per session. The signal was reative to the activity of multiple neuron units (or multi-unit activity). Monkey 3 and Monkey 1 were involved for this task and one session was recorder for each monkey.

- Combination 3: the task consists of reaching 8 different positions (45° apart). The signal was relative to the activity of multiple neuron units (or multi-unit activity). Monkey 4 and Monkey 5 were involved for this task and one session was recorder for each monkey.

In all the three cases, the microarrays were placed on the motor cortex, specifically in the primary motor and somatosensory cortices (M1 and SCx). For both reaching and grasping tasks, neural responses were aligned to the start of the movement, resampled to 100 Hz (or 250 Hz for Monkey 3), and then smoothed by a convolution with a Gaussian (25 ms S.D.). Single-unit activities were then extracted by an offline software. Regarding the acquisition of kinematic information, 30 markers were placed on the monkey's hand and the coordinates in time of these markers were acquired with a sampling rate of 100 Hz with a 14-camera motion tracking system. Each joint angle and velocity were retrieved offline. In the dataset, the following variables were reported:

- SpikeTimes and SpikeCounts: these variables are relative to the neural activity. "SpikeTimes" it includes the specific time instant in which a spike (or peak of activity) appears, for each trial and for each area. "SpikeCounts" includes the number of spikes appeared in a specific time interval, for each trial and for each area.

- JointAngle, JointVelocity and JointNames: these variables are relative to joint information, such as angles, velocities and names.

- TimeBins: this is a variable indicating the rightward edges of bins during which each frame of kinematic data was recorded.

On one hand, the kinematic signals are very rich in information, making it possible to control many or few degrees of freedom of the prosthesis. On the other hand, there are no sessions that record the same neural activity over time. The only sessions registered on the same monkey while performing the same task are in the combination 1. However, these sessions registered different neural population and the neural signals were relative to single-unit activity. These type of neural signals requires offline pre-processing steps, which is not compatible with a real-time application. These aspects are crucial for the project. Consequently, it was not possible to work with this dataset.

2.1.2 Dataset CRCNS

In this case, the task involved reaching a target that appears on the computer screen, controlling a cursor with a robotic arm that moves in the x-y plane (or transversal plane), see Figure 2.1.

The dataset provided data on two monkeys, Monkey M (MT) and Monkey T (MT). For MM, only one session is provided, while for MT the number of registered sessions is three. MM and MT were both implanted with 100-electrode arrays (Blackrock Microsystems, Salt Lake City, UT) in the pre-motor cortex (PMd) and in M1. However, data from M1 were not collected for MT.

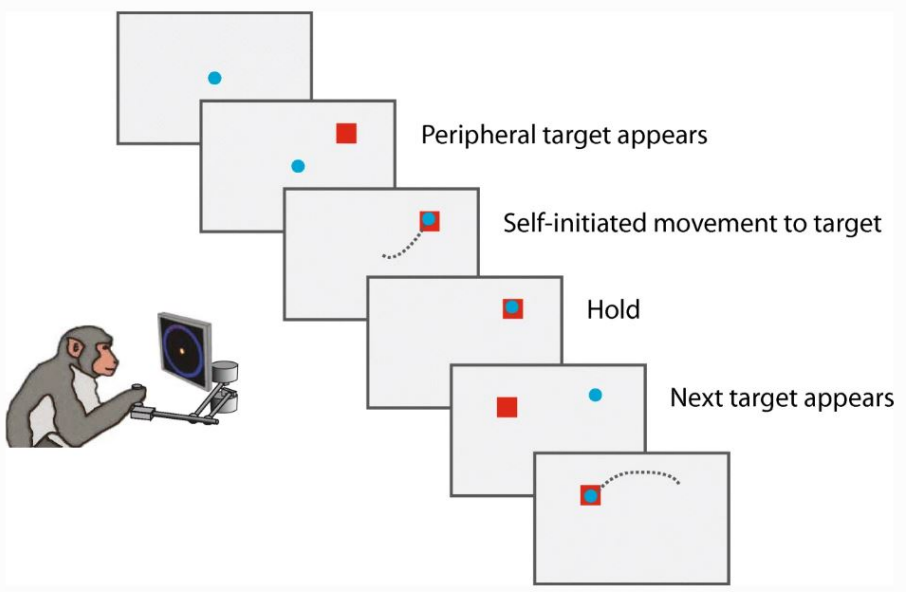


Figure 2.1: Task performed by the monkey in the CRCNS dataset. Figure taken from [9]

In both cases, spike-sorting was performed and units having high correlation with other units were excluded. In the end of the pre-processing, MM had 48 units left, while MT had 38 units. The dataset provided two version of the same data: one "raw," with the least-heavily-processed data, and one "processed." The first one includes all the electrophysiological, behavioural, and task data recorded for each trial. The second one included only a window of electrophysiological and behavioural data associated with a reaching movement.

The variables provided for each session are: kinematics data (with position, velocity, and acceleration of the cursor), neural data (specifically, spike counts) for both PMd and M1 and trial info. This dataset provides three registering sessions performed by the same monkey on the same task. However, no information was given on the time interval between sessions. Additionally, these sessions provided neural signals from the PMd and not from the M1. Data are also heavily pre-processed with offline techniques. For these reasons, this dataset was not chosen.

2.1.3 Dataset eBrains

In this dataset, the task performed by the monkeys L and N consisted of grasping an object in the shape of a square-based parallelepiped and pulling it towards itself (Figure 2.2). There were two types of grips:

- Precision grip, in which the monkey had to grip the parallelepiped by touching the upper and lower faces with thumb and forefinger.
- Side grip, in which the monkey had to grasp the side faces of the parallelepiped with thumb and forefinger.

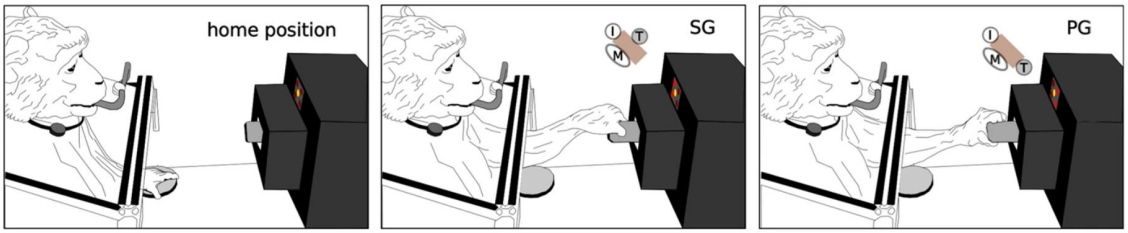


Figure 2.2: Task performed by the monkey in the eBrain dataset. Figure taken from [10]

Additionally, the force required to pull the object could be 1 or 2 Newtons. This led to four possible combinations of trials. The parallelepiped had 5 force sensors to retrieve the exerted force on itself and one position sensor to measure the displacement during the pulling phase. In this case, there were not kinematics

information, but instead signals relating to the force and pressure exerted on the object were provided.

For both monkeys, 96 neural signals from M1 were acquired, amplified, and filtered with a high-pass filter at 0.3 Hz and a low-pass filter at 7.5 kHz. These signals were lately resampled at 30 kHz and provided as "raw signals". Finally, all completed trials were identified and classified as one of four possible combinations. For each trial, it is reported whether it was successfully completed or not and all events concerning the execution of the task were detected.

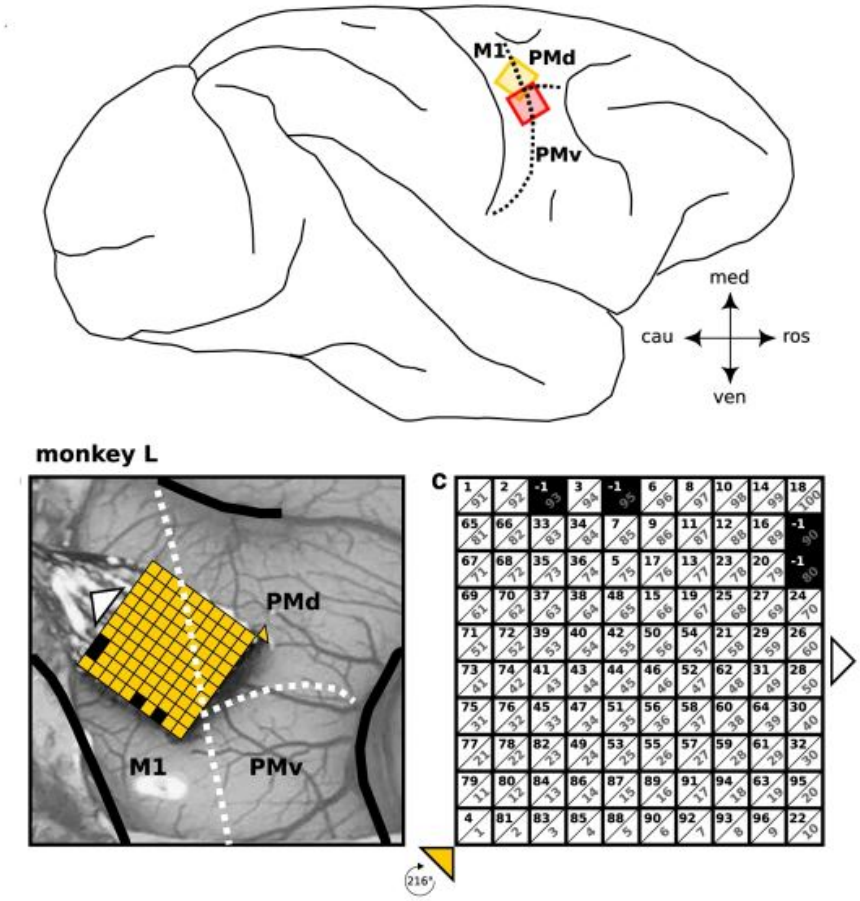


Figure 2.3: Example of implantation of 96-channel microelectrodes array in monkey L. Figure taken from [10]

Although the neural signals were optimal for the project purpose, since no offline pre-processing techniques were used on the data, there were only one registering session for each monkey. Thus, it was not possible to use this dataset for this work.

2.1.4 Dataset Zenodo

In this case, the task consisted of reaching a series of evenly distributed targets with the fingertip. The movements were concentrated in the transverse plane (so only two coordinates were acquired: x and y) in an 8x8 rectangular space.

The dataset provided data from two monkeys, Indy and Loco. For Indy, two 96-channels microelectrode arrays were implanted respectively in M1 and primary somatosensory cortex (S1) in the right hemisphere. 37 sessions were acquired within 10 months, but only in seven of them data were acquired from the S1 area. Also for Loco, two 96-channel microelectrode arrays were placed in M1 and S1. However, this time they were located in the left hemisphere. Loco underwent 10 recording sessions within one month, in which neural data were acquired from both M1 and S1 areas.

The neural signals were acquired with a sample rate of 24.4 Hz and they were filtered with a band-pass filter between 500 and 5000 Hz. To detect each spike, it was necessary to calculate the absolute value of the neural signal and extract the points at which the signal exceeded the threshold at 3.5/4 times the standard deviation. Offline spike sorting was then performed on neural data with a personalized software. Three to five units were extracted for each channel, each of which had a different spike waveform template. Each waveform template was characteristic for one specific unit. However, all spikes that exceeded the threshold but at the same time did not match any of the templates were collected in a unit called 'noisy'. Units with firing rate minor than 0.5 Hz were discarded.

Indy was trained to perform the task with the left arm, while Loco used the right arm. The position of the finger was acquired with a six-axis electromagnetic position sensor at a sample rate of 250 Hz. The three coordinates (x, y and z in mm) were always acquired. Additionally, sometime also the orientation of the sensor was retrieved. In total, the dataset provided for both monkeys:

- The channel name (for example, M1001, M1002 and so on).
- The cursor and the finger position: the cursor had only the x and y coordinates in millimetres (mm), while the finger had all the coordinates in centimetres (cm) and, occasionally, the orientation in degrees (°).
- The spike times for each channel and for each unit, including the "noisy" one.
- The time axis related to kinematic signals.

The dataset also provided raw neural and kinematics signals also for 32 out of 37 registering sessions.

Although spike sorting was performed, it is possible to retrieve the original signal by joining all the spikes from the different units for each channel. Overall, the dataset provides several recordings over time for each monkey. The task over the time is always performed in the same way and the neural signals are always obtained at the same position. This dataset matches most of the original requirements, hence it has been selected for this work. Specifically, it was decided to use data from the monkey Indy, due to the longer time spanned by recording sessions.

2.2 Literature and State-of-Art review

Once the dataset had been chosen, it was necessary to review the prior works involving this dataset, in order to propose innovative strategies. It was noticed that there are various works in the literature proposing AI algorithms for decoding neural signals into kinematic signals. However, it was necessary to find papers which evaluate the effectiveness of the algorithm over time.

For example, Li et al [11] work aimed at creating an accurate decoding system by separating task-relevant neural responses from task-irrelevant neural signals. In order to achieve this goal, a new framework called Distill-VAE (d-VAE) is introduced. d-VAE is an innovative variational-autoencoder (VAE) who combine probabilistic modelling techniques to AI algorithms. The main assumption is that task-irrelevant responses introduce noise into the input signals, greatly degrading decoding performance. Since there is not a ground-truth for the task-relevant signals, different criteria have to be introduced:

- The decoding performances achieved by using just task-relevant signals should surpass that of the raw signals. This solution should also have better decoding generalization than the raw signals.
- The task-irrelevant signals should contain minimal task information.
- The task-relevant signals should be sufficiently similar to the raw signals in order to preserve the neural properties.

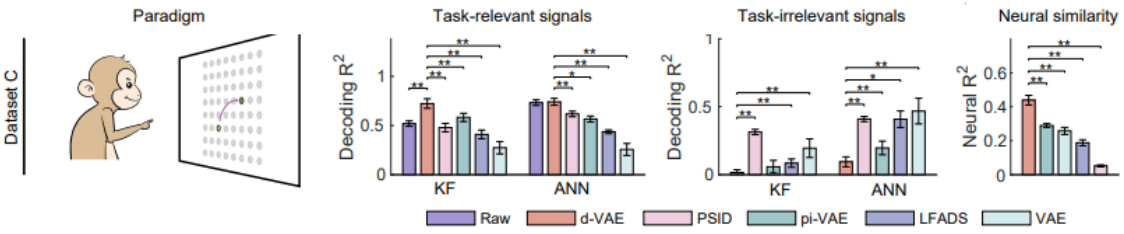


Figure 2.4: Results obtained in the article [11]. Figure taken from [11]

The results proved better decoding performance compared to using raw signals and using different methods, such as simple VAE, simple Neural Networks (ANN), Kalman Filter (KF) and another popular model called Latent Factor Analysis via Dynamical Systems (LFADS).

Li et al [12] also introduced a novel kernel regression framework. This approach is based on three main techniques:

- Siamese networks: these are two identical networks that shares the same weights. This technique requires a pair of samples as input and a measure of the similarity of the two input samples. Usually it is used for images, however it is particularly suitable for neural signals because it provides data augmentation and decreases the noise of input samples.

- Kinematics-to-neural loss (K2N): Instead of assessing the similarity between neural signals, it was decided to use the similarity between the kinematic signals relative to the input neural signals. This leads to a higher robustness of similarity measurements.

- Kernel regression: This method was compared to different well- known AI algorithms, such as long-short time memory (LSTM) neural network, simple NN, gated-recurrent neural network (GRNN) and KF. Although it does have the best performances with very short amount of data, it has lower performances than LSTM if a whole session is used to train the algorithm.

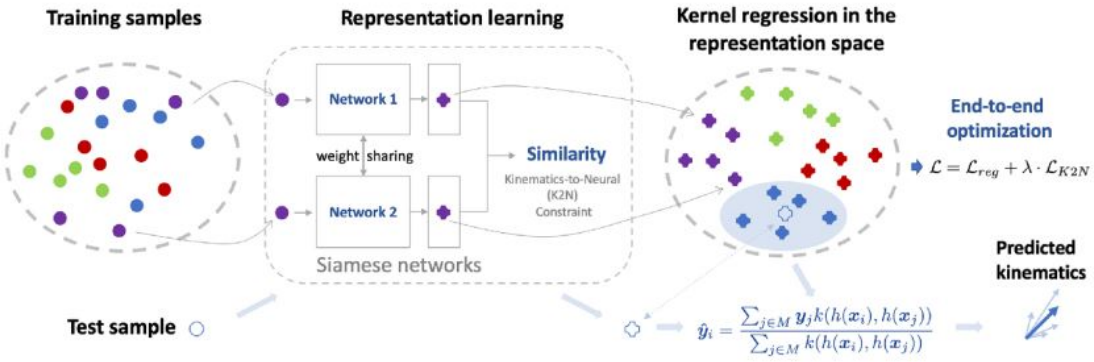


Figure 2.5: Framework schematization from article [12]. Figure taken from [12]

Finally, Reza Keshtkaran et al. [13] proposed another method based on an automatic system of model tuning and autoencoders called Auto-LFADS. This framework is by two main actors:

- LFADS: it is a Deep-Learning model composed by bidirectional Gated Recurrent Unit (GRU) neural networks, whose goal is to reconstruct the source neural signals. This particular model is composed by four NN: Initial Condition Encoder, Generator, Controller Input Encoder and Controller. To avoid the final model being a trivial identity function, a technique called Coordinated Dropout was introduced.

- Population Based Training (PBT): it is a framework that creates different LFADS models, called "workers," each with different parameters and trained in parallel. During the training, the workers with lower performances copies the hyperparameters of the best worker (exploit phase) and subsequently they change them slightly (explore phase), in order to focus on the better hyperparameters.

This method has been compared to a Gaussian process factor analysis (GPFA) and a LFADS random search.

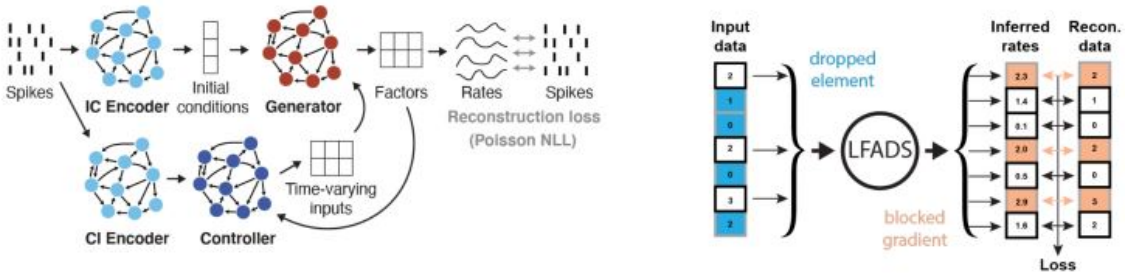


Figure 2.6: (Left) Framework scheme from article [13]. (Right) Scheme of the Coordinated Dropout. Figure taken from [13]

All three of the above-mentioned papers report results from only one of the 37 sessions. An example of paper reporting a comparison of algorithm performance over time was proposed by Ahmadi et al [14]. The aim of the publication is to find the combination of models, signals and time bin lengths that yields the best performance in reconstructing the velocity signal of the monkey's hand over time. In order to pursue the goal, multiple combination has been created and the following option have been introduced:

- NN Models: simple recurrent neural networks (RNN or SRNN), GRU, LSTM and quasi-recurrent neural networks (QRNN).

- ML Models: Wiener filter (WF), Wiener cascade filter (WCF), KF, Unscented Kalman filter (UKF).

- Signals: LFP, single-unit activity (SUA), MUA, Entire spiking activity (ESA).

- Time bin length: 20 ms (non-overlapping), 50 ms (non-overlapping) and 256 ms (overlapping of 252 ms).
- Training method:
  - Fixed Decoder: the model is trained on the first session and the trained weights are kept fixed for the rest of the sessions.
  - Retrained Decoder: training is done on a session-by-session basis by randomly initialising the weights each time.
  - Resumed decoder: as the Retrained decoder, training is done on a session-by-session basis. However, in this case the weights trained in the previous session are taken as initial weights for each training session. In this way, previous knowledge is transferred to the new model.

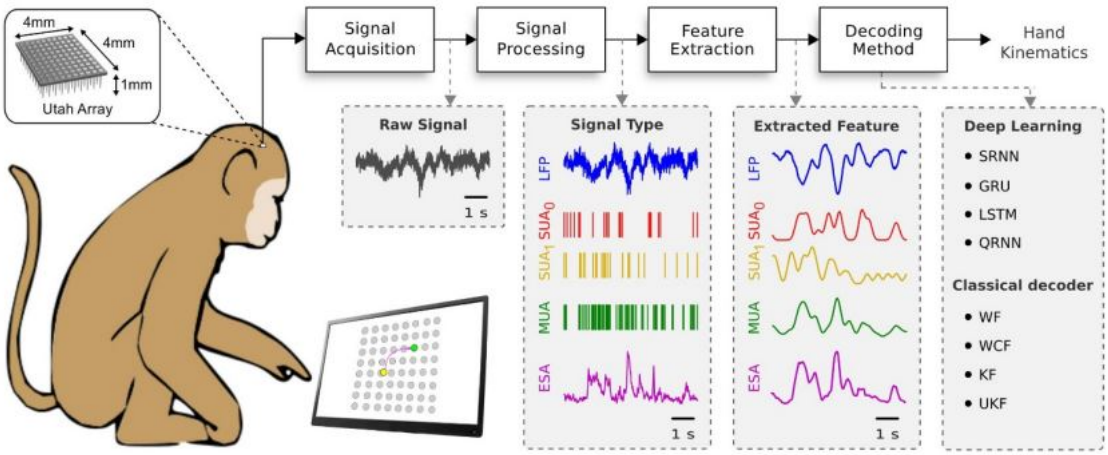


Figure 2.7: Framework scheme from article [14]. Figure taken from [14]

The resumed training represent an example of Transfer Learning applied in this context, hence it is particularly interesting for this work. After carrying out all possible tests, the combination with the best results uses ESA signals, QRNN model, 256 ms time bin and Retrained Decoder. However, further information can be obtained. For example, regarding the choice of AI model, all NN models return very similar performance (for the same signal, training method and time bin length) with higher performances than ML models. In term of signals, ESA outperformed MUA, SUA and LFP. However, the second-best performances were obtained by MUA signals.

It was therefore decided to take the best performance achieved by the paper as a gold-standard reference. The first step would be to understand how using

MUA signals only, a bin length compatible with real-time usage, and different networks compares in terms of performance to this reference model. Subsequently all the three training methods will be applied to obtain new performances and several Transfer Learning techniques will be introduced and compared to the gold standard. Some steps of the data pre-processing and the AI models are inspired by Elios Ghinato's [24] and Ilaria Gesmundo's [25] Master Thesis work, while some of the Transfer Learning techniques are inspired by Myriam Lubrano's work [26].

Chapter 3

Data analysis

3.1 Data description

To summarise, within the dataset we find the following signals:

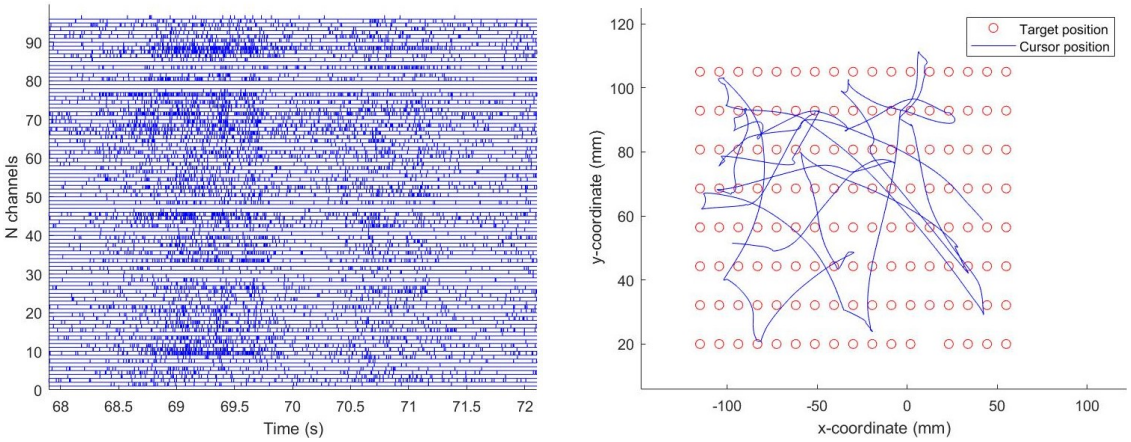
- Spike data: it is a N x m cell structure, with N equal to the number of channels and m is the number of units extracted for each channel. N can be either 96 (just M1 data) or 192 (both M1 and S1 data), while m can be either 3 or 5. For each channel and each unit, there is a vector showing the time stamps at which spikes were detected.

- Cursor position: as the finger position, it does indicate the position of the hand of the NHP while performing the task. The signal acquisition was done with a sampling frequency of 250 Hz. It has been decided to use the cursor position, since it has only the x-y coordinates in mm.

- Time: it is the time axis relative to finger position and cursor position.

Each session had a different duration and they are shown in the Table 3.1. Knowing the duration of the sessions will be crucial in choosing possible Transfer Learning strategies.

Before creating a decoding model, data must be pre-processed and transformed into the most suitable format. For the kinematics data, it is not useful to reconstruct the hand position signal. This is because it is reasonable to think that neural signals are not strongly correlated to the position (which is defined at less of a constant), but instead it is correlated to the movement itself, so to the velocity. Also because previous literature did that. Additionally, velocity as input signal for the prosthesis results in smoother and more stable control. For these reasons, it is more useful to retrieve the hand velocity from the hand position. Once the velocity has been predicted, it is still possible to trace the position of the hand.



(a) Spike times for each channel. (b) Example of cursor position on the grid.

Figure 3.1: Neural (a) and kinematics (b) signals from the session "`indy_20160407_02`". Figure created on MatLab.

Subsequently, the data must be discretised by creating time windows or 'time bins'. These will be necessary to create observation windows or sequences, in order to have a small portion of the signal and predict the future portion.

3.2 Data pre-processing

The first pre-processing step that had to be performed was to merge the different units for each channel. In this way, it was possible to easily switch from SUA signals to MUA signals. To do this, all the units in each channel were simply concatenated and ordered.

SUA

N° channels	Unit 1	Unit 2	Unit 3	Unit 4	Unit 5
	[23587x1 array]	[643x1 array]	[9213x1 array]	[253x1 array]	[6467x1 array]
	[143x1 array]	[]	[266x1 array]	[112x1 array]	[]
	…	…	…	…	…
	[1603x1 array]	[206x1 array]	[1256x1 array]	[]	[]
	[23587x1 array]	[643x1 array]	[9213x1 array]	[253x1 array]	[6467x1 array]

MUA

N° channels	Unit
	[40163x1 array]
	[521x1 array]
	…
	[3065x1 array]
	[40163x1 array]

Figure 3.2: Example of SUA structure and MUA structure.

Once the MUA have been created, the spike rate had to be retrieved by setting

the length of the time bin. In this way, it is possible to count the number of spikes falling in each time bin and thus derive the spike rates. It has been shown in previous works [14, 24, 25, 26] that normally the time bin length is between 20 to 50 ms, without overlapping. Time bins smaller than 20 ms could lead to noisy signals and therefore noisy predictions, while time bin much larger than 50 ms could not be compatible with a real-time application. For this work, it has been decided to use a time bin 20 ms long, so the results could be so that the results could be compared with those in the gold standard.

Figure 3.3: Conversion from Spike times to Spike counts.

Once the spike rate was retrieved, a simple discrete derivative was used to derive the hand velocity from hand position, for both x and y coordinates. In order to downsample the new kinetic signals to the new sample rate, each time bin value was assigned the average of all velocity samples within that time bin. The new dataset was therefore saved in a .csv file: it included all the 96 neural signals (one for each channel), the respective channel names, and 2 kinetic signals (one for each coordinate). A .csv file was created for every session.

Once all the data were converted, Training Set (TRS), Validation Set (VAL) and Test Set (TSS) were created. To do this, the initial 80% of the signal was allocated to the Construction Set (CSS) (consisting of TRS and VAL), while the final 20% was allocated to the TSS. It is important to emphasise that CSS and TSS were divided time-wise, with initial segments of the recording sessions used for training and validation, while later segments are used for testing, so that the testing and evaluation of the AI algorithm on TSS would be as realistic as possible, in terms of system application. The same procedure was also carried out for the division of TRS and VAL: the first 80% of the CSS was allocated to the TRS, while the last 20% of the CSS was assigned to the VAL. In this way, the algorithm would be trained on the first portion of the signal, parameter tuning would be performed on

Figure 3.4: Example of resampling velocity signal.

the intermediate portion, and validation and evaluation of the algorithm would be done on the final portion.

A clarification is necessary: in the present work, the TSS set has always been used as a signal to test the performance of the model on a signal that the model itself has never seen during the training. Hence, if the duration of TRS and VAL was shorter than what was used later in this work for model fine-tuning (e.g. 5 minutes), it was decided to use the totality of TRS and VAL for fine-tuning. This happened whenever fine-tuning was performed on the initial portion of a recording whose duration was around 6 minutes. In fact, in case of fine-tuning on a portion of the session, TRS and VAL had to be recreated. For example, in case of fine-tuning on the first 5 minutes of recordings, TRS and VAL respectively had to contain 80% and 20% of the first five minutes of the signal. However, if the signal was too short, the initial 5 minutes of signal included also a portion of the pre-existing TSS. In the end, in order to compare the results obtained between all methods, it was decided to train the model on the pre-existing CSS set, even if the total duration was less than 5 minutes, leaving the TSS signal for the performance analysis. Furthermore, when training is performed on the entire session, it is implied that the TSS set never pollutes the training phase.

Once the three sets were created, it was possible to extract length-fixed windows of the signals. This is crucial for the adaptation of the system to a real-time application: the real-time algorithm cannot have the entire signal at its disposal, precisely because it would go against the very definition of real time. The available signal consists of a fixed *window* N of bins, comprising the 'current' bin and all previous N-1 bins. Consequently, to train the algorithm properly, it is necessary to divide the sets into several partially overlapping sequences. Two more parameters must be decided: the number of bins for each sequence and the number of overlapping bins. It was decided to have 24 bins for each sequence and an overlapping number of bins of 22. In this way, except for an irrelevant initial transient of 24*0.02 ms = 480 ms, each sequence will have 2 new final bins that must be predicted. Hence,

for each set two three-dimensional (3D) matrices were created:

- *_data: This is the data matrix. It is a [S x b x c] matrix, where:
  - S = number of resulting sequences.
  - b = number of bins per sequence (24).
  - c = number of neural signals (96).

  These are training_data, validation_data and test_data.

- *_labels: This is the "label" data, necessary for a supervised learning. It is a [S x b x v] matrix, where:
  - S = number of resulting sequences.
  - b = number of bins per sequence (24).
  - v = number of variables (coordinates) to be predicted.

  These are training_labels, validation_labels and test_labels.

This procedure was applied to all the sets for every session. Finally, all sets were saved in a .npz file.

Figure 3.5: Division of each sequence in TRS, VAL e TSS. Data is intended as both neural and kinematics signals.

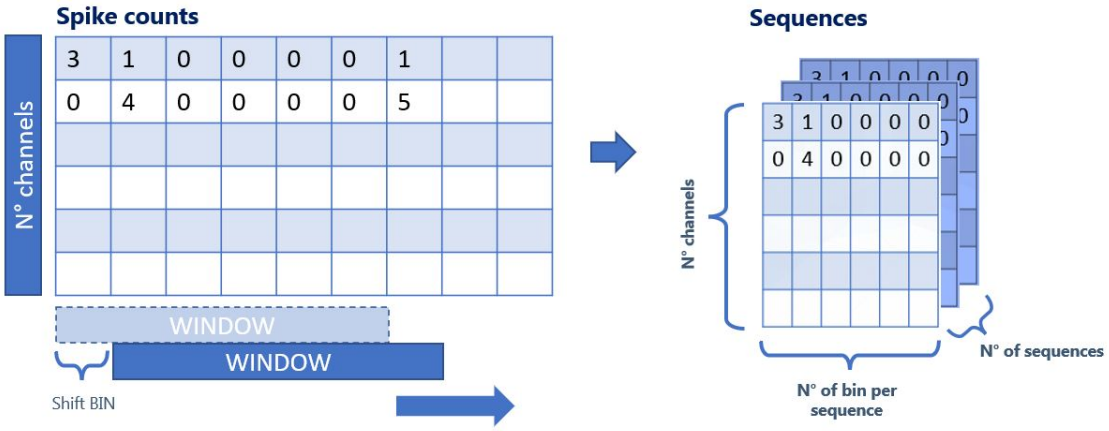


Figure 3.6: Procedure to retrieve sequences.

Session name	Duration (s)	Duration (min)
indy_20160407_02	817.76	13.62
indy_20160411_01	953.18	15.89
indy_20160411_02	879.96	14.67
indy_20160418_01	1357.58	22.63
indy_20160419_01	523.96	8.73
indy_20160420_01	1553.18	25.89
indy_20160426_01	1762.02	29.37
indy_20160622_01	2449.64	40.83
indy_20160624_03	499.96	8.33
indy_20160627_01	3362.92	56.05
indy_20160630_01	1463.18	24.39
indy_20160915_01	381.02	6.35
indy_20160916_01	451.80	7.53
indy_20160921_01	360.12	6.00
indy_20160927_04	389.34	6.49
indy_20160927_06	420.98	7.02
indy_20160930_02	460.76	7.68
indy_20160930_05	405.98	6.77
indy_20161005_06	373.98	6.23
indy_20161006_02	501.98	8.37
indy_20161007_02	491.36	8.19
indy_20161011_03	673.74	11.23
indy_20161013_03	517.42	8.62
indy_20161014_04	519.00	8.65
indy_20161017_02	495.80	8.26
indy_20161024_03	472.32	7.87
indy_20161025_04	504.02	8.40
indy_20161026_03	497.42	8.29
indy_20161027_03	578.92	9.65
indy_20161206_02	737.88	12.30
indy_20161207_02	445.04	7.42
indy_20161212_02	560.72	9.35
indy_20161220_02	576.30	9.60
indy_20170123_02	609.76	10.16
indy_20170124_01	589.98	9.83
indy_20170127_03	733.52	12.23
indy_20170131_02	15.94	13.60

Table 3.1: List of the sessions duration in seconds (center column) and minutes (right column).

Chapter 4

Decoding model selection and single-session results

4.1 Introduction to Artificial Intelligence and Deep Learning approaches

Before starting to create and train AI models, it is necessary to introduce some basic definitions, starting from the concept of artificial intelligence itself. Artificial intelligence (AI) is the theory and development of computer systems able to perform tasks normally requiring human intelligence. As it is possible to see, it is a very broad definition, covering many different applications in many different fields. However, not all the AI algorithms have the capability to learn new knowledge, contrary to what is often believed. The simplest architectures can include only rules handwritten by the creator of the algorithm.

In fact, learning is the fundamental feature of a subset of AI, called Machine Learning (ML). This AI system has the ability to learn a specific pattern from input data, which constitutes the very knowledge of the machine. To do so, it is necessary to extract multiple pieces of information for each data sample (for example, a patient), called features. The chosen set of feature will be used to represent each sample in an N-dimensional space (where N is the number of features). The more informative the feature set is, the easier it will be for the ML algorithm to learn and perform the task (e.g., classification) correctly. However, the choice of the right set of features is the main problem in this type of algorithm. This is due to the fact that, very often, the links between the different features are unknown and this is one of the main concerns of this field of research. On the other hand, once the right combination of features is found, it is much easier to interpret the steps taken by the algorithm and understand what combination and values of these features characterise the task itself.

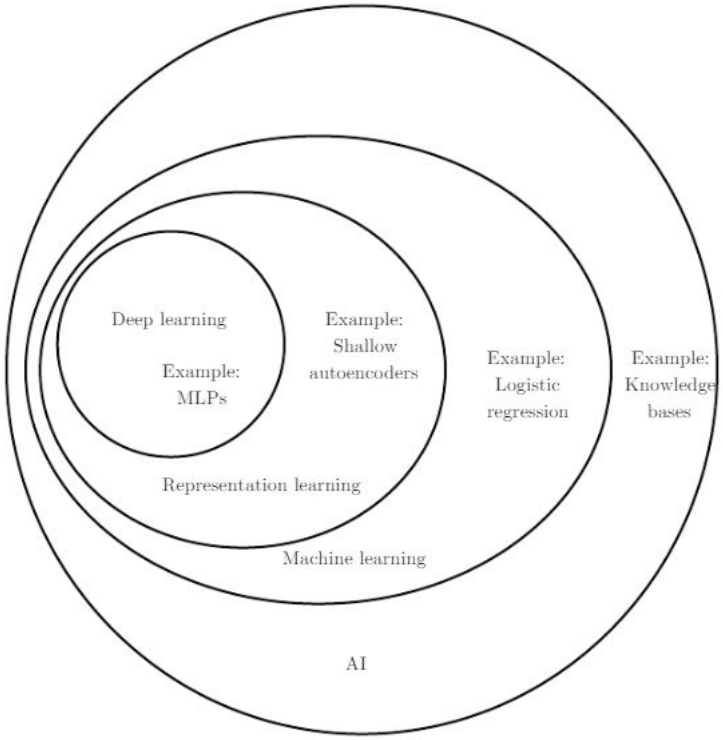


Figure 4.1: Venn Diagram of the Artificial Intelligence techniques. Figure taken from [15]

However, it is not always possible to characterize the task by "simple" features, such as age, gender, or weight. Most of the time, the task can be solved only by using abstract and high-level features, which can be very difficult to extract. To overcome this issue, a new subset of ML, known as Deep Learning (DL), has been introduced. The basic example of DL is a feedforward deep network called Multilayer Perceptron (MLP). The MLP is a Neural Network (NN) composed by three type of layers: the input layer, composed by one neuron (or node) per input data, one output layer, composed by one neuron per desired output (for example, one for each class) and one or more hidden layers. Each neuron is connected to all the neuron of the following layer (fully-connected network). The network is characterized by weights for each input of the neuron and an non-linear activation function that mimics the threshold potential of neurons. With the training, every weight is modified to obtain the correct result via backpropagation. On the one hand, with DL algorithms it is easier to obtain an algorithm that correctly performs the required task with good accuracy and it is not necessary to select or engineer features to give as input to the system, since the original entity itself is given as input (such as an image, a signal, etc...). On the contrary, since the algorithm extracts high-level features by learning from data, not only is it not easy to trace

and understand the extracted features, but it is also difficult to explain how an algorithm has extracted a certain class, signal, or image (see Figure 4.2). For example, the input for the NN an image depicting a tumour of a patient and it is necessary to understand whether it is benign or malignant. In the case of a positive result, it is not possible to understand the features that led to this conclusion. Since in the medical field the interpretability is required in many applications, it could be a problem.

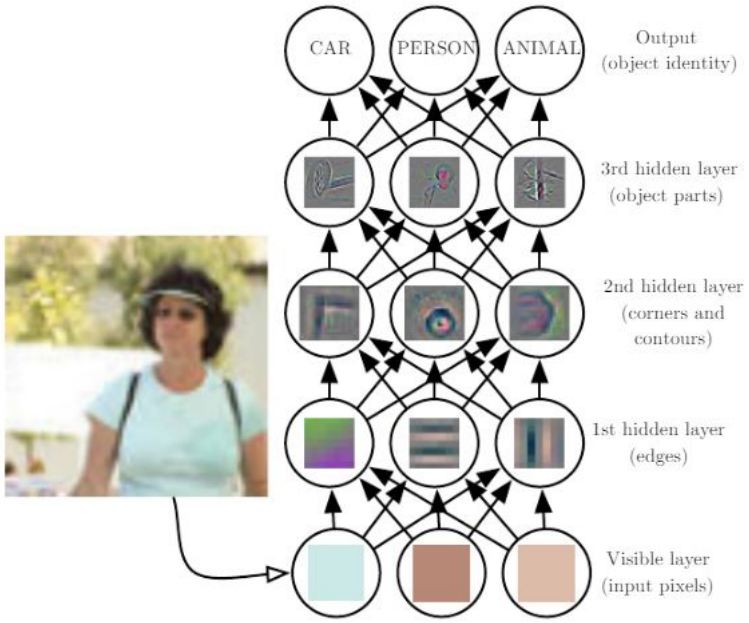


Figure 4.2: Illustration of a Deep Learning model. Figure taken from [15]

Until now, it has been discussed about training the ML algorithm so that, given an input, a result equal or similar to the actual signal is returned. If the network computes the difference between the two outputs (predicted and true) and learns from it, a supervised learning approach is used. This learning paradigm is the most used one, but it is not the only one: multiple variations have been introduced, some of which fall within the same paradigm, while others require new definitions, for example the unsupervised learning.

Supervised Learning, as previously said, is a ML approach that requires both the input (called *features*) and the output (called *label*). In this case, the goal of the ML algorithm is to learn the data pattern, mapping, or the function f(x) that allows to predict the outcome y* as close as possible to the true label y given the input x. This can be done, for example, through the backpropagation of the error in the MLP training. This approach has many pros and cons. For example,

supervised learning techniques are easier to implement and understand and they are more accurate. However, especially when using neural networks, a huge amount of data is required and this in not always possible. For example, patients are not always willing to undergo recording sessions or in other cases it is the facility itself that prevents it, due to, for example, overcrowding in the registration laboratories. So even if using Supervised Learning approaches is easier and more accurate, efforts are being made to adopt techniques that require fewer and fewer labels.

A totally different approach, known as Unsupervised Learning, does not need any output information at all. It is based on the division of data into groups called "clusters" or "neighbourhoods", based on the similarity of inputs between them. For example, the input images of an algorithm are pictures of different types on vehicles: the algorithm will learn to divide images by similarity and to differentiate the cars from motorbikes and trucks, even if there are not any labels. The algorithm will be able to identify N different entities. In this way, unsupervised learning can create a higher-level representation of the data. This approach requires no labels; however, it is not easy to use. For example, it is typically necessary to decide a priori the number of neighbourhoods. It is also hardly suitable for regression tasks and, without any label, it is not possible to identify a class (for example, from cluster 1 and cluster 2 to benign and malignant). So, most of the times the unsupervised learning approach is used as a precursor of the supervised learning technique. In this work, mainly supervised learning methods will be used.

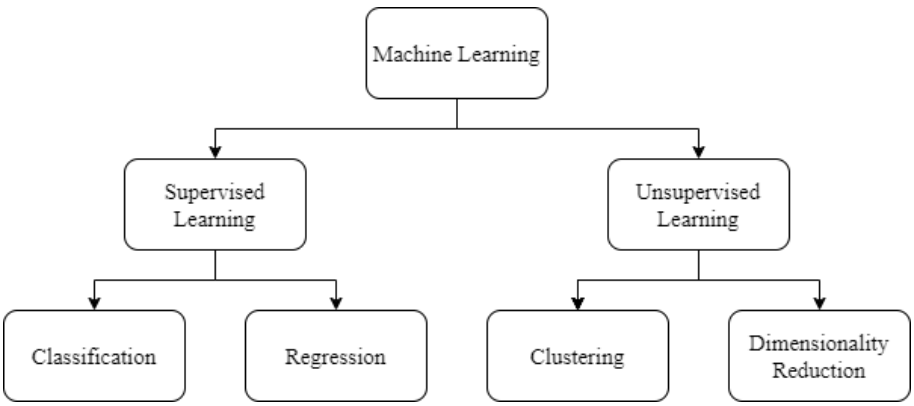


Figure 4.3: Division of training approaches and relative tasks.

In Supervised Learning, different tasks can be performed by applying AI algorithms. Mainly, the tasks are divided into two groups:

- Classification: the classification result is the assignment of an input to one of the N predefined groups or classes, where N is an integer and positive number. An example could be an algorithm that, given an image of an incoming tumor,

tells whether the tumor belongs to the benign or malignant class. In this case, there are two classes: benign and malignant.

- Regression: the result of a regression task is a continuous value. An example is the prediction of sales trends in a company or the reconstruction of a kinematic signal from neural signals. To perform this type of task, a sequence of samples is required and very often this is a time sequence.

As said in the section 2.1.4, the task carried out by the NHP is a continuous reach of random targets in the transversal plane. The goal of this work is to predict the hand movement based only on the neural signals. We can describe the learning task in this work as finding the correlation between a time series of neural signals and a pair of real-value variables. In this sense it is a supervised regression task.

As the input has specific characteristics (i.e., one coordinate is time), we know from literature that the most used and performant family of NN architecture are Recurrent Neural Networks (RNN), a type of NN specialized in processing sequential data or time series data. Many different variations exist, however in this work simple-RNN (sRNN), Long Short-Term Memory (LSTM) and Gated-Recurrent Unit (GRU) will be introduced.

4.1.1 Recurrent Neural Network (sRNN)

The output of unidirectional RNNs depends on prior samples in the same sequence. As other NN architectures, sRNN structure is composed by the input layer, a hidden layer with weights and activation functions, and the output layer. The main feature of RNNs is that they can search for correlations by analysing the input in a privileged direction with respect to others (i.e., time or space).

The two differences from common DL techniques are that weights are common within each layer (or inputs) and weights are updated by both the current input and the previous ones (as it is possible to see in Figure 4.4, by the arrow in green that connects the hidden layer to itself) with Backpropagation Through Time (BPTT). BPTT principles are the same of the simple backpropagation, however in this case the error between each predicted sample of the sequence and the counterpart of the actual sequence is calculated and then summed with the errors of previous samples.

Another variation of the sRNN is the Bidirectional RNN (BRNN). In this type of networks, samples are passed through the back propagation step twice, in both directions; the consequence is doubling the number of parameters, learning correlations in both time directions. Hence, the whole sequence is used to predict the current sample.

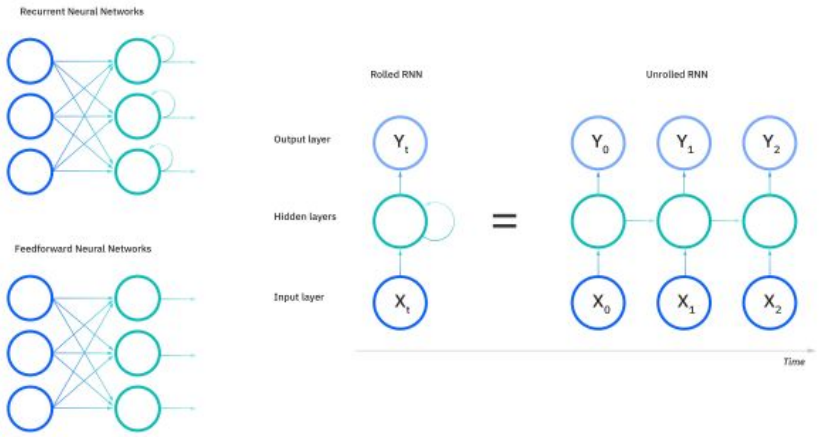


Figure 4.4: Architecture of a RNN compared to a simple MLP. Figure taken from [16]

The sharing of weights between the different layers and a sufficiently long sequence leads the RNN to a problem known as Vanishing/Exploding Gradient. This problem is caused by the magnitude of the gradient itself. If it is too small, the sums of the errors will further decrease the gradient, updating the weights approximately to zero. If it is too high, however, the weights will increase too much, until overflowing the precision of the type used to represent the gradient. The solution to this problem could be to decrease the number of samples in the sequence, however it is not always the correct choice as the network may not recognise the pattern in the sequence, thus losing usefulness.

4.1.2 Long Short-Term Memory (LSTM)

One solution to the vanishing/exploding gradient problem has been introduced by Hochreiter and Schmidhuber in 1997 [27] with the name Long Short-Term Memory neural network. The main difference from RNN is that LSTM use a long-term knowledge rather than just the previous prediction, avoiding the updating of weights to insignificant or huge numbers. Instead, the output prediction is a combination of the current long-term memory (known as cell state), the previous sample output (known as hidden state) and the current input sample. The structure of the LSTM is made of three parts called gates: the forget gate, the input gate, and the output gate (see Figure 4.5).

The forget gate f_t is the part of NN that decides how much the cell state can contribute, also having the previous hidden state h_{t-1} and the current input data

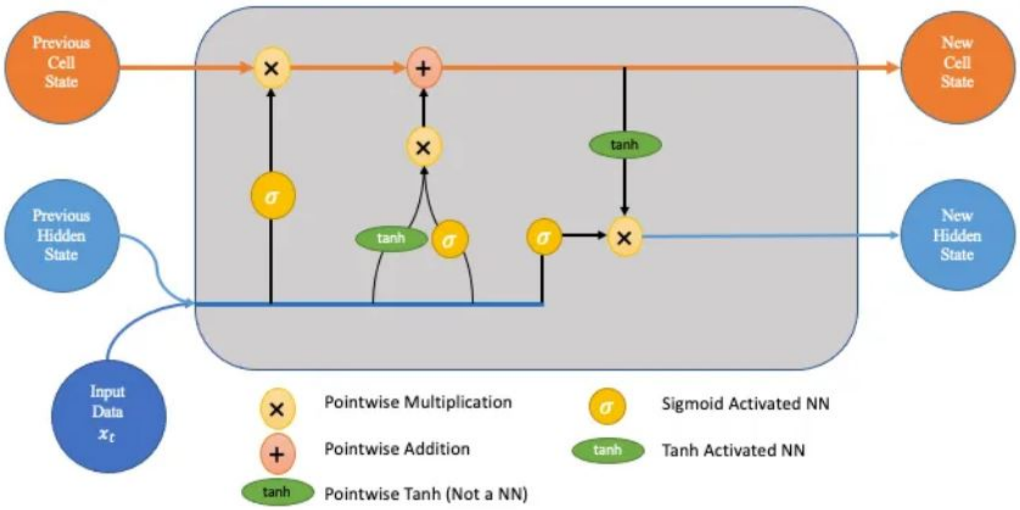


Figure 4.5: Scheme of a cell in LSTM layers. Figure taken from [17]

x_t. Both hidden state and input are weighted and summed, then fed to the sigmoid function σ (eq. 4.1). The more relevant is this combination, the closer to 1 will be the output of the sigmoid function. Otherwise, the result will be closer to zero. The result will be pointwise multiplied $\odot$ with the previous cell state.

$$f_t = \sigma(W_f x_t + U_f h_{t-1} + b_f) \tag{4.1}$$

The input gate i_t determines how much of the new information should be added to the long-term memory. As for the forget gate, both previous hidden state h_{t-1} and current input x_t are weighted and summed with two different couples of weights (W_i and W_c) and biases (b_i and b_c) and then respectively fed to a sigmoid function σ to retrieve the "Potential long-term memory" (eq. 4.2) and to an hyperbolic tangent function (tanh) to retrieve the "Potential Memory to remember" (eq. 4.3). Differently from a sigmoid function, the tanh function returns values between [-1, 1], so if the result is closer to -1, it means that it is necessary to reduce the impact of the long-term memory. Both components are then pointwise multiplied $\odot$ and then summed to the output of the multiplication between the forget gate and the previous cell state (eq. 4.5). The result will be the new long-term memory.

$$i_t = \sigma(W_i x_t + U_i h_{t-1} + b_i) \tag{4.2}$$

$$\tilde{c}_t = tanh(W_c x_t + U_c h_{t-1} + b_c) \tag{4.3}$$

The last element of the LSTM architecture is the output gate o_t, whose goal is to

determine the new hidden state. In this step, the previous hidden state h_{t-1} and the current input x_t are again weighted and summed, then fed to the sigmoid function σ (eq. 4.4). In the meantime, the new cell state is fed to the tanh function. Both results are pointwise multiplied $\odot$ to retrieve the new hidden state h_t (eq. 4.6). To retrieve the output from the new hidden state, it is necessary to apply a linear layer.

$$o_t = \sigma(W_o x_t + U_o h_{t-1} + b_o) \tag{4.4}$$

$$c_t = f_t \odot c_{t-1} + i_t \odot \tilde{c}_t \tag{4.5}$$

$$h_t = tanh(c_t) \odot \sigma(o_t) \tag{4.6}$$

These three gates compose the base cell of the LSTM network and it will be repeated for each input sample of the sequence. So, to predict a 24 bin-long sequence, the LSTM will have 24 cells. As for RNNs, there is also a bidirectional version for LSTMs called Bidirectional LSTM (BLSTM) in which the number of layers and weights doubles.

4.1.3 Gated Recurrent Unit (GRU)

Another way to avoid the vanishing/exploding gradient problem is to use another type of RNN called Gated Recurrent Unit (GRU). They were introduced by Cho et al in 2014 [28] and works similarly to LSTM. In fact, like LSTM also GRU's layers are composed by cells or gates, the update gate, and the reset gate.

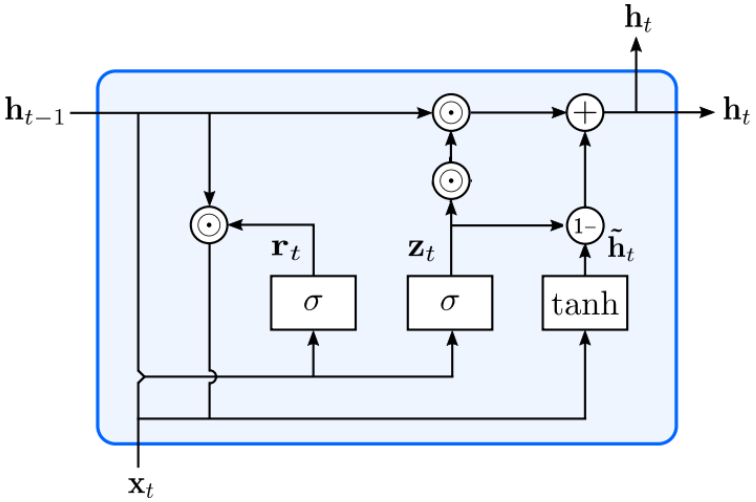


Figure 4.6: Scheme of a cell in GRU layers. Figure taken from [14]

The update gate z_t is very similar to the forget gate in LSTM: both current input x_t and previous hidden state h_{t-1} are weighted and summed, then fed to a sigmoid function σ that will return a value between [0,1] (eq. 4.7). As the forget gate, the update gate will determine how much of the past knowledge will affect the final result and will be maintained.

$$z_t = \sigma(W_z x_t + U_z h_{t-1}) \tag{4.7}$$

The reset gate r_t, as the name suggest, will decide how much of the past knowledge will be forgotten. To do so, the steps are the same of the update gate, but the weights are different (eq. 4.8).

$$r_t = \sigma(W_r x_t + U_r h_{t-1}) \tag{4.8}$$

Then, new memory has to be created and stored. This is done by weighting the previous hidden state h_t and multiplying it through Hadamard product $\odot$ with the result of the reset gate r_t. This will return how much of the past knowledge has to be removed. The result is then summed to the weighted current input and fed to a tanh function (eq. 4.9).

$$\tilde{h}_t = tanh(W x_t + r_t \odot U h_{t-1}) \tag{4.9}$$

The final step is the creation of the new hidden state h_t. This is the most complicated part of the GRU layer. Firstly, the previous hidden state is weighted and multiplied with the result of the update gate. This result will be summed to the result of the output of the tanh function multiplied by the value $(1 - zt)$, returning the new hidden state h_t (eq. 4.10). The factor (1-zt) indicates how much of the current information is useful to store in the new hidden state.

$$h_t = z_t \odot h_{t-1} + (1 - z_t) \odot \tilde{h}_t \tag{4.10}$$

Similar to the LSTM network, the GRU network also has its bidirectional variant (BGRU).

4.1.4 Hyperparameter optimization

Once the neural network model has been chosen, the hyperparameters must be set in order to obtain an effective and robust model. Hyperparameters are parameters that can be related both to the network structure (such as layer, dropout, etc.) and to the training algorithm (i.e., optimizer, learning rate, epochs, etc.). However, due to the large number of hyperparameters and the large range of values that each parameter can assume, the search for the optimal hyperparameters is a very long and

tedious process. There are, however, already-existing Hyperparameter Optimization frameworks which allow the search in hyperparameter space to be carried out automatically. The one used in the present work is KerasTuner [29] by TensorFlow. Between the different optimization algorithms provided, the BayesianOptimization option was used. This tuning algorithm uses information from the previous model to search the optimal hyperparameters in the area of the research space that yielded to better result.

The hyperparameters included in the optimization are the following:

- Number of units: it could be either 16, 32, 40, 64, 128 or 256.
- Number of layers: from 1 to 3.
- Dropout: from 0.4 to 0.9, with 0.1 steps.
- Kernel regularizer: between L1 (Lasso Regression), L2 (Ridge Regression) or both L1 and L2 (Ridge and Lasso Regression or Weight Decay).
- Recurrent regularizer: between L1, L2 or both L1 and L2.

For each model search 50 trials were deployed. Since initial weights are randomly initialized, three execution per trial were carried out in order to prevent an unfavourable initialization of weights from excluding a possible set of hyperparameters.

4.1.5 Evalueted metrics

Since the goal of this work is to predict the 2D velocity signal of the robotic arm, it is necessary to choose metrics that measure the similarity of two signals (the true one and the predicted one). The metrics chosen for the evaluation of network performance are the following:

- Root Mean Squared Error (RMSE) [30]:

$$\mathrm{RMSE}(y, \hat{y}) = \sqrt{\frac{\sum_{i=1}^{N}(y_i - \hat{y}_i)^2}{N}} \tag{4.11}$$

- Pearson Correlated Coefficient (CC) [31]:

$$CC = \frac{\sum_{i=1}^{n}(x_i - \bar{x})(y_i - \bar{y})}{\sqrt{\sum_{i=1}^{n}(x_i - \bar{x})^2}\sqrt{\sum_{i=1}^{n}(y_i - \bar{y})^2}} \tag{4.12}$$

These two metrics provide complementary information to each other. The RMSE gives an indication of the similarity between two signals, indicating the order of magnitude of the error. It is, however, greatly influenced by outliers (few errors

but large amplitude). The CC, on the other hand, is a measure of correlation and indicates how similar the signal trend is to the original signal trend. In contrast to the RMSE, it is not sensitive to scaling factors and is not greatly influenced by outliers. However, for this very reason, it is sufficient that there is, for example, a low amplitude peak at a very high peak to return a positive CC value. It is therefore necessary to consider both metrics to evaluate the performance of regression AI algorithms.

These metrics were retrieved for each kinematic signal (velocity x and velocity y) and then the mean value was retrieved.

4.2 Model selection results on a single session

In this section, results of training and optimization of hyperparameters on the first session "`indy_20160407_02`" will be discussed. These results will help choosing the most performant NN architecture to perform and analyze Transfer Learning tecnhiques on the remaining sessions.

For each NN, the sets of hyper-parameters found are shown in Table 4.1.

Best hyperparameter sets			
NN type	BRNN	BLSTM	BGRU
N° layers	3	3	3
N° units	256	128	128
Dropout	0.4	0.6	0.6
Kernel regularizer	L1_L2	L1	L1_L2
Recurrent regularizer	L1_L2	L1_L2	L1

Table 4.1: List of hyperparameters chosen for each NN from the optimization on session "`indy_20160407_02`".

For all the NN models, the performance achieved on the Test Set of session "`indy_20160407_02`" are shown in the Table 4.2.

Performance on TSS			
NN model	BRNN	BLSTM	BGRU
RMSE (mm/s)	34.49	30.50	30.71
CC (%)	77.78	83.46	82.92

Table 4.2: Performance achieved on the Test Set of session "`indy_20160407_02`" by each RNN model.

Since the BLSTM model returned the best performance, it was chosen to evaluate Transfer Learning strategies. In Figure 4.7, it is shown an example of a portion of the signal predicted by the BSTLM network compared with the true counterpart of the Test Set, from session "`indy_20160407_02`".

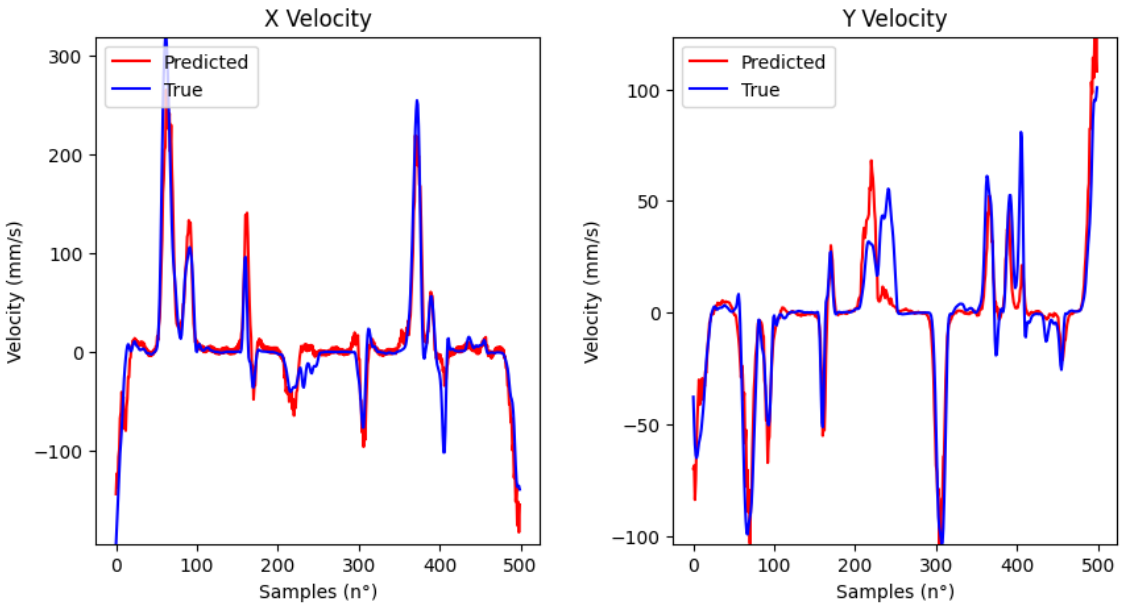


Figure 4.7: Comparison between a portion of the TSS predicted signal by BLSTM with the true counterpart.

In Table 4.2 it is possible to see the performance obtained on the first session. In this work, specific pre-processing steps were used to allow the signals and the model to be used in real time. Thus, in general, the length of the bins and the length of the sequences given as input to the model differ from the works mentioned above ([11], [12], [13], [14]). It is therefore not possible to make a direct comparison with the results obtained in the literature, but it can be seen that the results obtained in terms of RMSE and CC are in line with those of the works referred to.

For the performance figures regarding Training Set and Validation Set, refer to Appendix B.

Chapter 5

Robustness of the model over multiple sessions

It is time to see how the model performs over several sessions distributed over time and to implement different Transfer Learning strategies, in order to try to improve the robustness of the AI model. As previously said, Transfer Learning is a Supervised Learning technique that consists in the transfer of knowledge from one model to another that have similar tasks. In our case, the task is the same, but the input signals are always changing, so this technique is perfectly usable for this project.

Since the definition of Transfer Learning is very broad, there are a lot of different techniques in literature. For example, in the Master Thesis work of Lubrano [26], the fine tuning of the model trained originally on a single session was performed using the 20% of the following registration session, achieving promising results. However, it was not possible to carry out a study over the time, since the used dataset provided only two registration session.

In the present work, two categories of Transfer Learning techniques will be studied:

- Fully supervised Fine-Tuning of the model at regular time intervals, with the whole or initial portion of the session.

- Self-supervised learning, a ML borderline technique between Supervised Learning and Unsupervised Learning.

5.1 Fine-Tuning techniques

The first analysed technique is the fine-tuning over a regular time interval, using the totality or the first minutes of the session. However, the first step was to define the baseline performance of the model over the time. In order to do this, the model originally trained on the first session has been used to predict the output signals of each session and retrieve the performance. This attempt, called "Fixed Decoder", will represent the lower bound of performance that Transfer Learning techniques will have to overcome to be valid.

Subsequently, as in the work of Ahmadi et al. [14], we evaluate the most expensive approaches: a full re-training of the model for each session. Either by starting from scratch (randomly initialised weights) or by re-training the model from the previous session. This will represent the upper bound of the performance that we can achieve with the data at hand. The first one is called "Retrained Decoder" and it creates a new NN model with randomly initialized weights for each session to be trained on. Since there is no transfer of weights, which constitute network knowledge, this is not a TL technique.

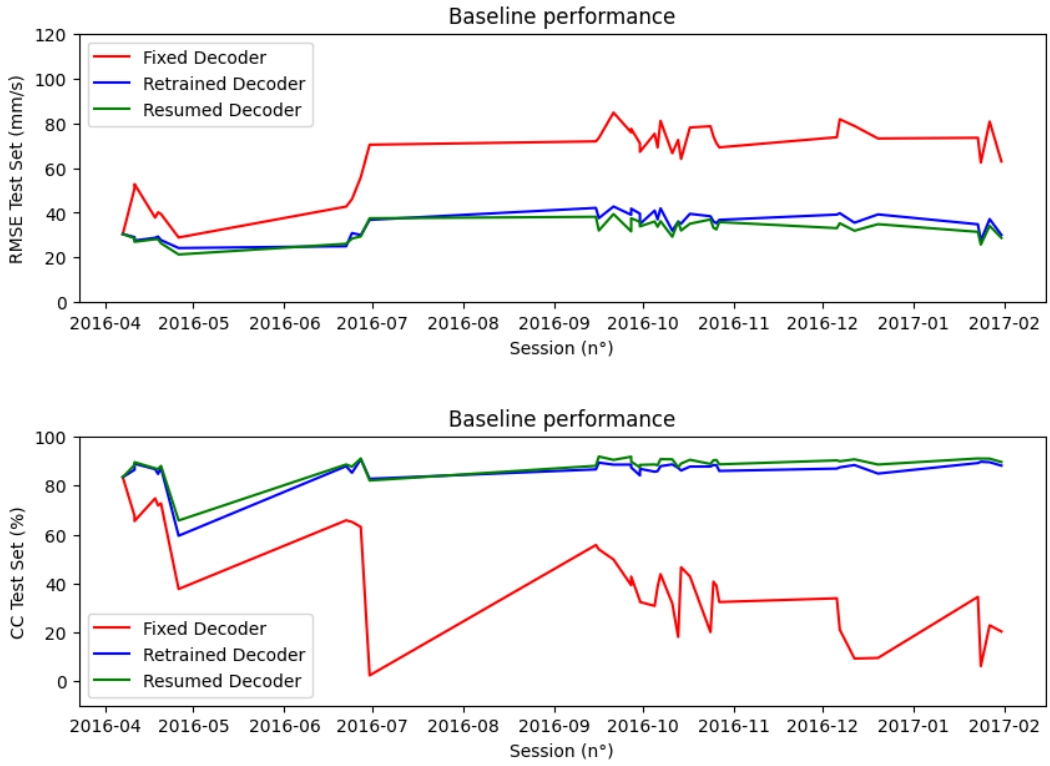


Figure 5.1: Baseline performance for Transfer Learning techniques.

The second techniques is called "Resumed Decoder" and it creates a new NN model for each session, but unlike "Retrained Decoder" in this case the weights trained on the previous session are assigned as initial weights. This is the first technique that somehow fall under the Transfer Learning definition used and it constitutes the gold standard of Transfer Learning on the dataset provided (Figure 5.1).

Once the baseline performance were defined, it was time to decide how frequently perform the fine-tuning on the sessions provided. Different time intervals were tried, both in terms of the months between fine-tuning sessions and the number of session minutes sufficient to achieve good performance. Ideally, fine-tuning techniques should request as few recording session as possible, in order to automatically adapt to the patient. In fact, the more sessions required and the longer the duration of the sessions, the more likely it is that the patient will reject the system or will be distracted during the acquisition, resulting in a noisy or distorted signal and thus in system failure. On the other hand, performing fine-tuning after too much time could lead to low accuracy, hence the inability to use the prosthesis as the model would make too many errors due to the high change in input signals. It is therefore necessary to find a compromise between the two needs.

5.1.1 Two months fine-tuning

The first fine-tuning technique require a fine-tuning every two months. In the present work, two different strategies were tested:

- Two-months fine-tuning of the entire session
- Two-months fine-tuning on the first 5 minutes of acquisition.

The two strategies were trained on the following sessions: "`indy_20160622_01`", "`indy_20160915_01`", "`indy_20161027_03`" and "`indy_20170123_02`" (refer to the Table 3.1 in Chapter 3).

As it is possible to seein Figure 5.2, the two strategies return similar performance. It is important to emphasise that the performances returned are very similar, except for the first fine-tuning session, where training over the whole session returns better performance than training over a 5-minute session. This is due to longer session duration of "`indy_20160622_01`": this session is 40-minutes long, while the others last between 6 and 10 minutes. Therefore, the difference of performance is higher because there is a lot of sample difference between the whole signal and 5 minutes of signal, unlike other fine-tuning sessions.

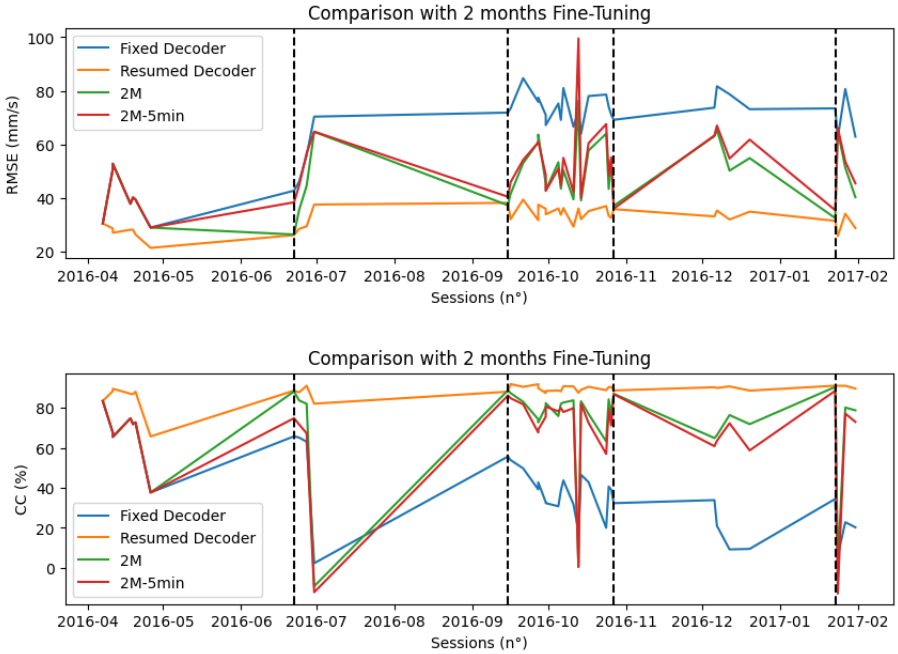


Figure 5.2: Comparison between two-months fine-tuning strategies in terms of RMSE (upper figure) and CC (lower figure). Each strategy is marked with a different colour, while the dashed vertical lines indicate the sessions in which fine-tuning was carried out. Both FT methods lay between the lower and upper bounds as expected.

5.1.2 Three months fine-tuning

The second fine-tuning technique require a fine-tuning every three months. In the present work, two different strategies were tested:

- Three-months fine-tuning of the entire session
- Three-month fine-tuning on the first 5 minutes of acquisition.

The two strategies were trained on the following sessions: "`indy_20160630_01`", "`indy_20161005_01`" and "`indy_20170123_02`" (refer to the Table 3.1 in Chapter 3).

Also in this case, the two strategies report similar performance, as shown in Figure 5.3. However, after the first session of fine-tuning it is possible to notice a significant deterioration in performance lasting until the next fine-tuning session, even worse than the performance achieved by the Fixed Decoder. This led to the assumption that not all available sessions were suitable for training, but on the contrary that there were noisy recording sessions. Hence, it was necessary to find a strategy to test whether a session would produce a robust trained model or if it is too noisy and would create a biased model.

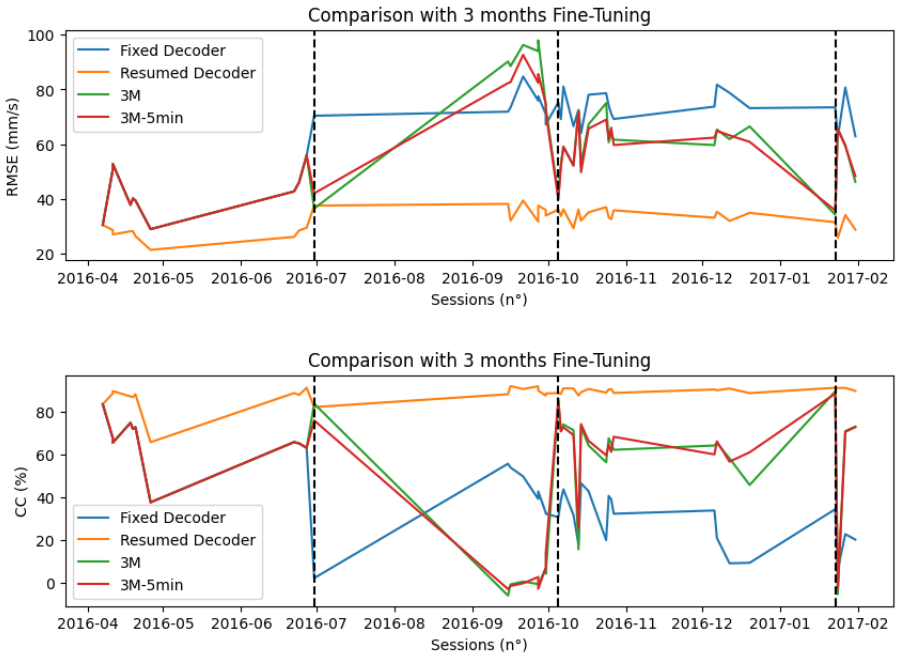


Figure 5.3: Comparison between three-months fine-tuning strategies in terms of RMSE (upper figure) and CC (lower figure). Each strategy is marked with a different colour, while the dashed vertical lines indicate the sessions in which fine-tuning was carried out.

Design of a strategy to identify noisy recording sessions

Due to the need to simulate real-time use, it is not possible to evaluate the performance on future sessions. However, it is possible to evaluate the performance of the algorithm on previous sessions to assess its robustness. This is because for the model the previous session is seen as a simple session without any temporal reference. What is expected is a moderate deterioration in performance, which in the case of the current project translates into an increase of a few tens of mm/s for RMSE and a few tens of percentage points for CC (see Figure 5.4). In case the deterioration is much greater than expected, the problem could be related to a larger than expected deviation of neural data of the current session with the previous one, hence it may be a signal of a noisy recording session. This strategy relies on an assumption of rather slow evolution of the signal across sessions, but this assumption is validated in this context. This strategy has been tested both on the three months fine-tuning and the two months fine-tuning approaches.

In Figure 5.5, it is possible to see the performance in terms of RSME and CC for both strategies on all the previous sessions. In this figure, the performance obtained on all the previous session are reported for completeness. However, it is important to remember that, when considering the real-life scenario, not all the

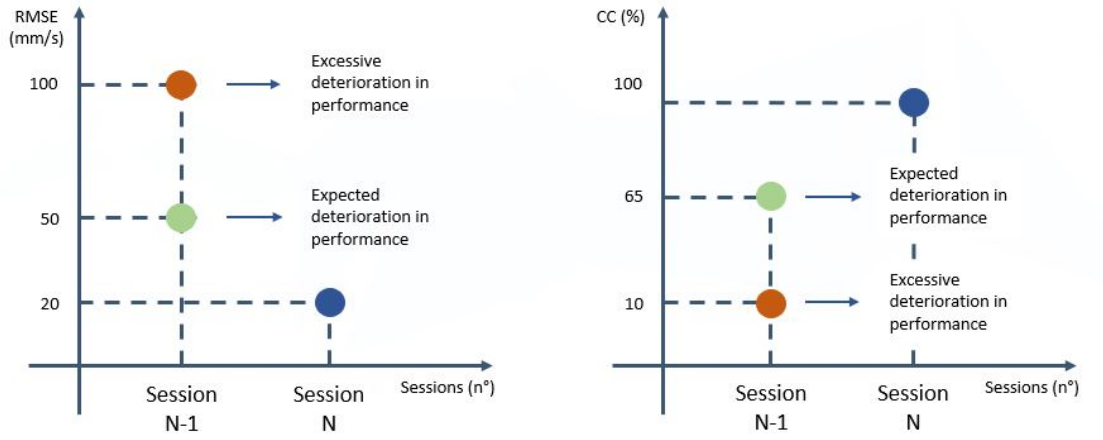


Figure 5.4: Example of application of the control strategy for detecting noisy sessions and ineffective training. RMSE (left) and CC (right) metrics are considered.

labels would be provided. In fact, only the labels of the sessions used for training and fine-tuning would be collected. The labels provided by the remaining sessions should be considered just to see if the model is robust or not, but in real life they would not be collected. In Figure 5.5, the inverted triangle indicates the current Fine-Tuning session, while the cross indicates the last Fine-Tuning session. For both cases, the cross indicates the session "`indy_20160407_02`". As previously said, the performance of the in between sessions are reported for completeness, but they should not be considered for evaluation purposes. As it is possible to see, the two-months fine-tuning approach reported a degradation of the performance of 30 mm/s on RMSE and 40% on CC. Even though it is not a negligible deterioration, it is a deterioration consistent with what was expected. While for the three-months fine-tuning strategy, RMSE has increased of 50 mm/s and CC has decreased to approximately 0%, so a deterioration of 80%. It is therefore evident that training on the "`indy_20160630_01`" session leads to a significant deterioration in performance. For this reason, this session will be excluded in the training phase, but not in the performance evaluation phase.

It was therefore necessary to repeat the three months strategy excluding the session "`indy_20160630_01`" from the training phase. The session "`indy_20160915_01`" was used for the fine-tuning, since it is the session just after the noisy one, and on the session "`indy_20161206_02`". The results are shown in Figure 5.6.

A further attempt was made to assess whether training using data from the current session combined with the previous session used for fine-tuning or training would lead to an improvement in the robustness of the algorithm. This strategy has been carried out using the whole sessions and it was compared to the three

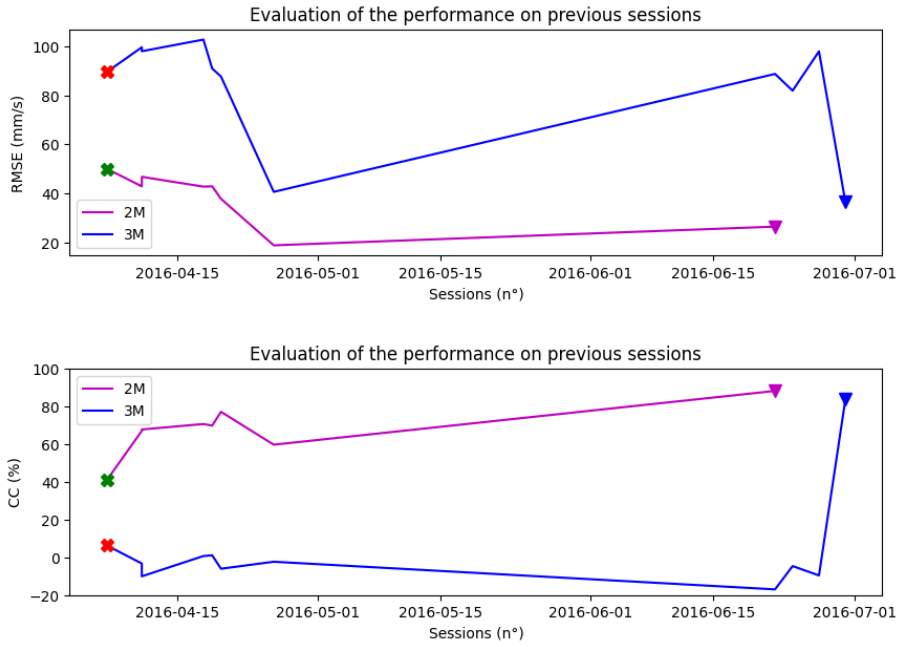


Figure 5.5: Comparison between two-months and three-months fine-tuning strategies in terms of RMSE (upper figure) and CC (lower figure) on previous sessions. The triangles indicate the current training session (one for each strategy), while the crosses indicate the previous training session. The red cross indicates an excessive degradation in performance, while the green cross indicates an expected degradation in performance.

months approach using the entire session for training. The results are reported in Figure 5.7.

Finally, the last Fine-Tuning strategy used provided one training session each week, with only a 3-minute session. This approach was introduced to assess whether more frequent training but with shorter labelled signals was a viable alternative to the strategies already reported. The results have been compared to the two-months and three-months (without the problematic session) approaches in Figure 5.8. As it is possible to see, this one-week strategy led to slightly better performance than the two-months fine-tuning approach, however the improvement is not high enough to justify the demand for such close and such a large number of recording sessions (13 recording sessions versus 4 with the dataset provided).

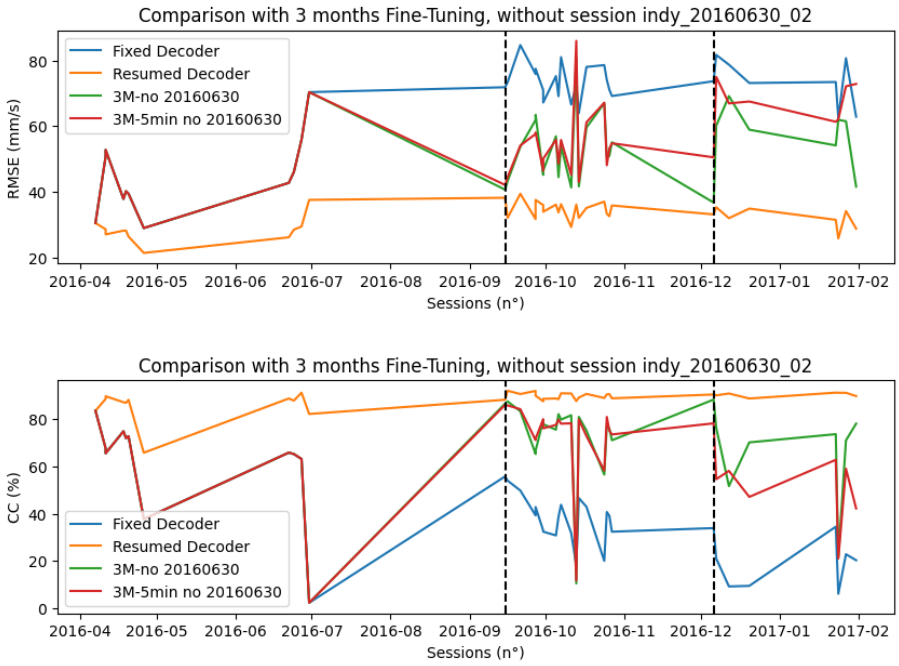


Figure 5.6: Comparison between three-months fine-tuning strategies in terms of RMSE (upper figure) and CC (lower figure).

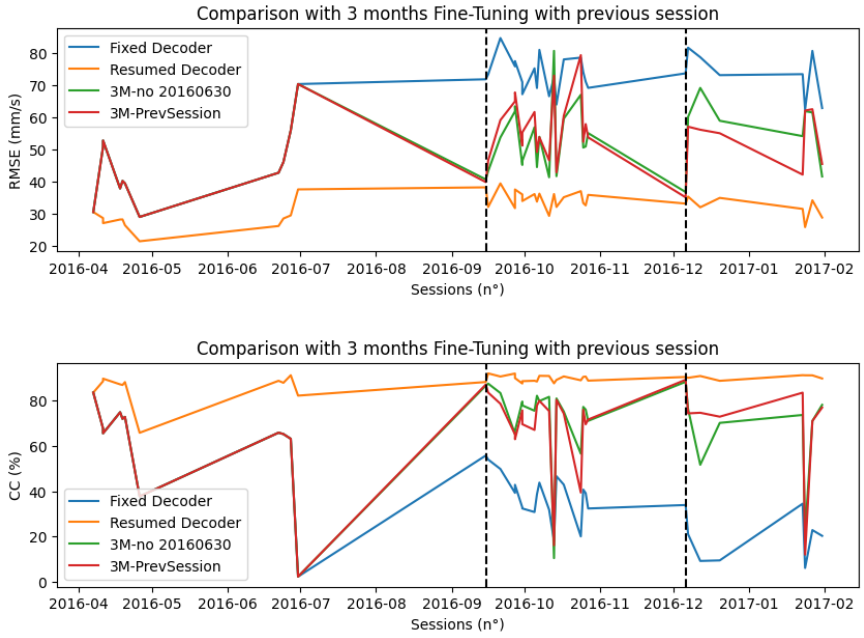


Figure 5.7: Comparison between three-months fine-tuning strategies in terms of RMSE (upper figure) and CC (lower figure), using data from previous session in training phase.

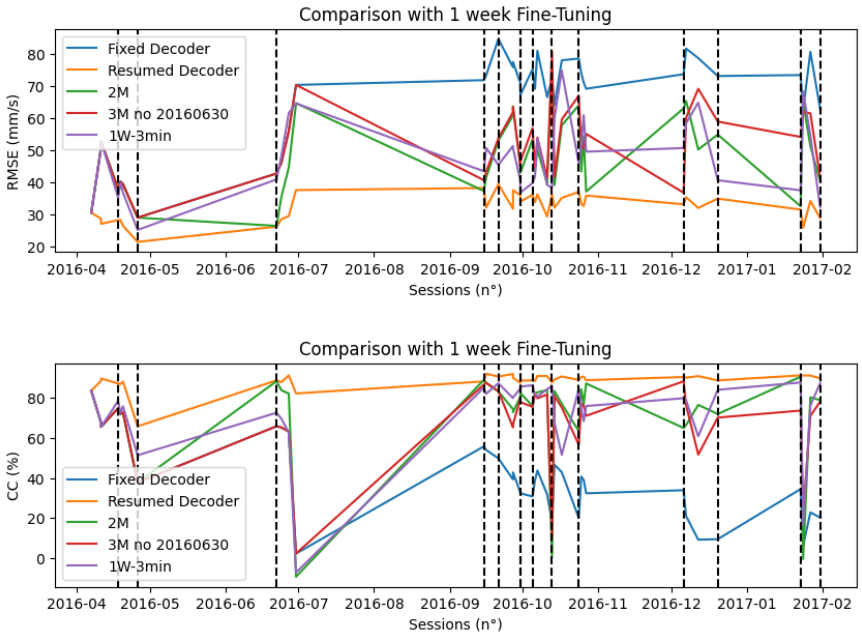


Figure 5.8: Comparison between 1 week fine-tuning strategies in terms of RMSE (upper figure) and CC (lower figure).

5.2 Self-Supervised Learning

Self-Supervised Learning (SSL) is a ML technique which lay between supervised and unsupervised learning. It is a paradigm created to decrease the over-dependence of AI models on labelled data. In this work, a simplified version of this approach was used: the model trained using SSL paradigm takes the input signals as a reference and generates the labels itself from which it will learn. As a result, the model is able to perform tasks of supervised learning (in this case, regression), but does not need the true labels like unsupervised learning. This approach can be really helpful for AI application in medical field, since it takes time and availability of doctors and patients to collect labels, as previously said. The idea behind the use of this method is that the model is able to adapt to the gradual changes in the neural signals and that it modifies the weights in such a way as to obtain signals similar to the real ones, but without having a reference to the real ones of the current session, but only to those of the first session. An example of application of SSL approach to a classification task is shown in Figure 5.9.

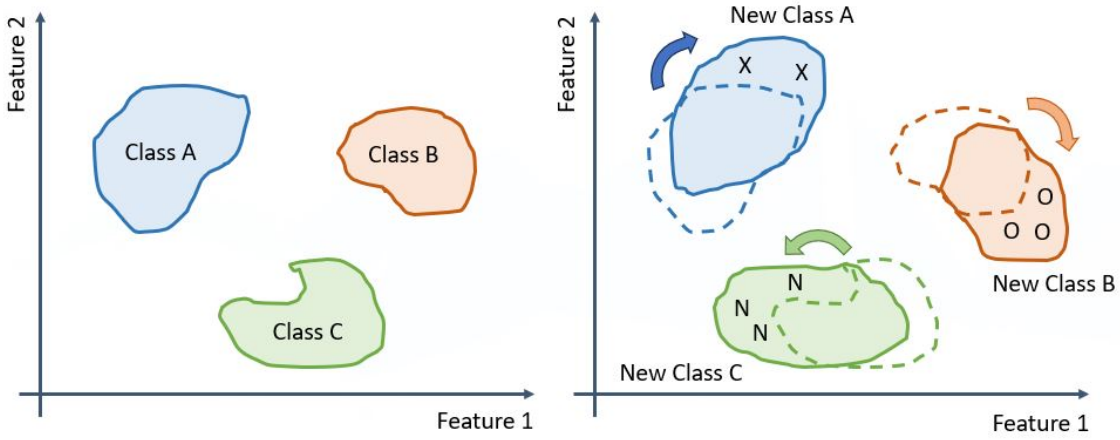


Figure 5.9: Idea between the combination of SSL techniques with the classification task. Left figure: initial situation with three distinguished classes in the 2D plan. Right figure: post-training situation with the new samples. Each sample has been classified as member of the closest class (prediction phase) and then the model has been trained with the new samples to update the knowledge (SSL training phase).

With this paradigm, two strategies have been explored:

- SSL on each session, using the entire TRS and VAL sets of the current session.
- SSL on each session, using the entire TRS and VAL sets of both the previous and the current sessions.

Operationally, for each recording session N, the model trained at session N-1 was taken, the velocity kinematic signals of session N were predicted and the model

was trained with the original neural signals and the predicted kinematic signals of session N.

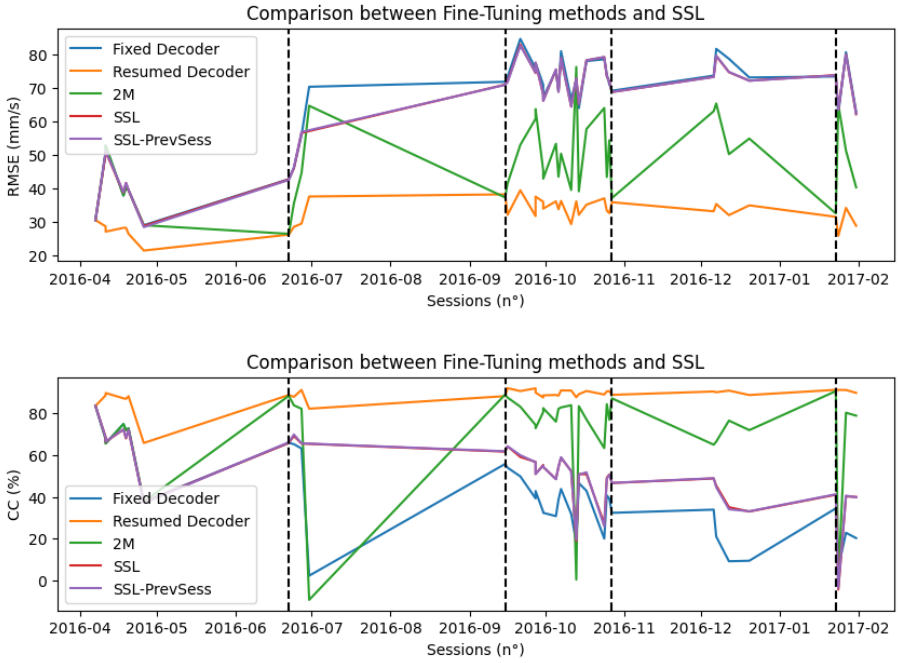


Figure 5.10: Comparison between SSL approach and 2-months fine-tuning in terms of RMSE (upper figure) and CC (lower figure).

As it is possible to see in Figure 5.10, the two techniques performed almost identically, differing only by a few hundredths of mm/s or percentage points. The results obtained are very similar to the performance obtained with the "Fixed Decoder" method and worse than the Fine-Tuning methods previously mentioned. This can be explained by remembering that SSL is an unsupervised learning method. The fact that it learns on the labels predicted by itself leads to not having a ground truth on which to calculate real errors and modify the weights with backpropagation appropriately. Moreover, the regression task itself leaves much less room for error than a classification task, as the output is not a value in a discrete number of classes but rather a set of real numbers. Consequently, in the case of the task of this work, the techniques used of SSL do not prove to be successful.

5.3 Final results

Finally, it is time to compare all the proposed Transfer Learning and retrieve the best one in the context of the current project. To do so, two boxplot graphs have been created and shown in Figure 5.11 and Figure 5.12. For each boxplot, the green

triangle represents the mean value, the orange line represents the median, the box extends from the lower (Q1) to the upper (Q3) quartile values and the whiskers extends to the values shown respectively in the equations 5.1 for the lower whisker and 5.2 for the upper whisker. The circles are the outliner values.

$$W_l = Q1 - 1.5 * (Q3 - Q1) \tag{5.1}$$

$$W_u = Q3 + 1.5 * (Q3 - Q1) \tag{5.2}$$

As it is possible to see, in terms of compromise between errors and range of time between the retraining sessions, the best Transfer Learning approaches proposed in the present work are the ones that perform a fine-tuning every two months, even though the 3-months fine-tuning provides comparable performance. It is possible to see the mean value and the standard deviation of the performance of each technique in the Appendix C.

It is important to underline that the one-week strategy return performance sligthly better than the two-months approach, however it is undeniable that requires much more commitment, data collection and patient disposition with respect to the two-month solutions.

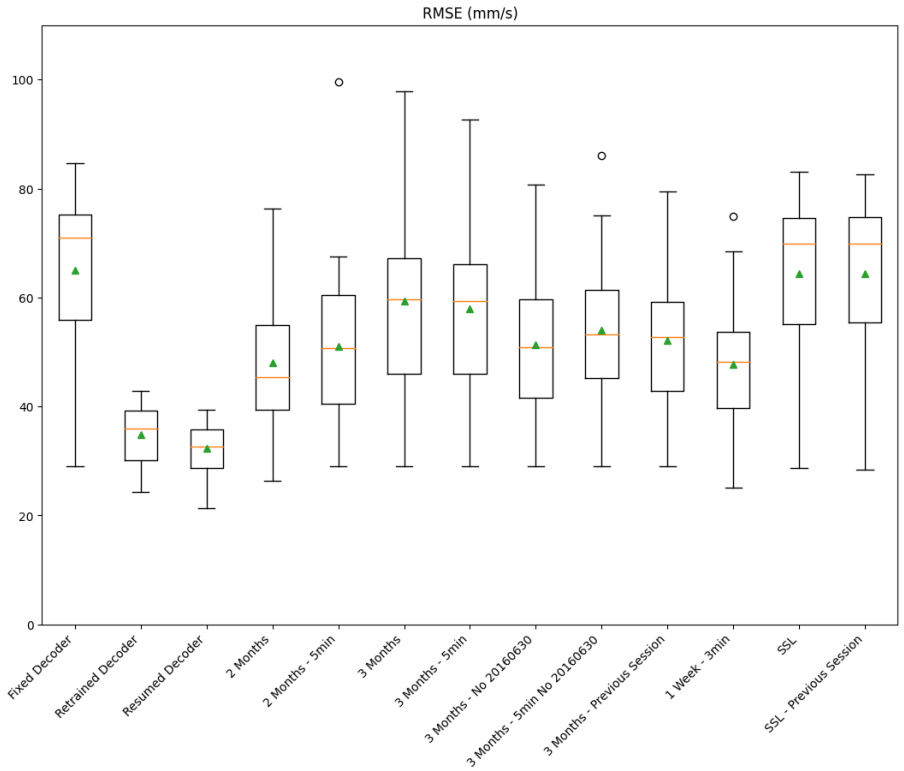


Figure 5.11: Comparison between Transfer Learning approaches in terms of RMSE.

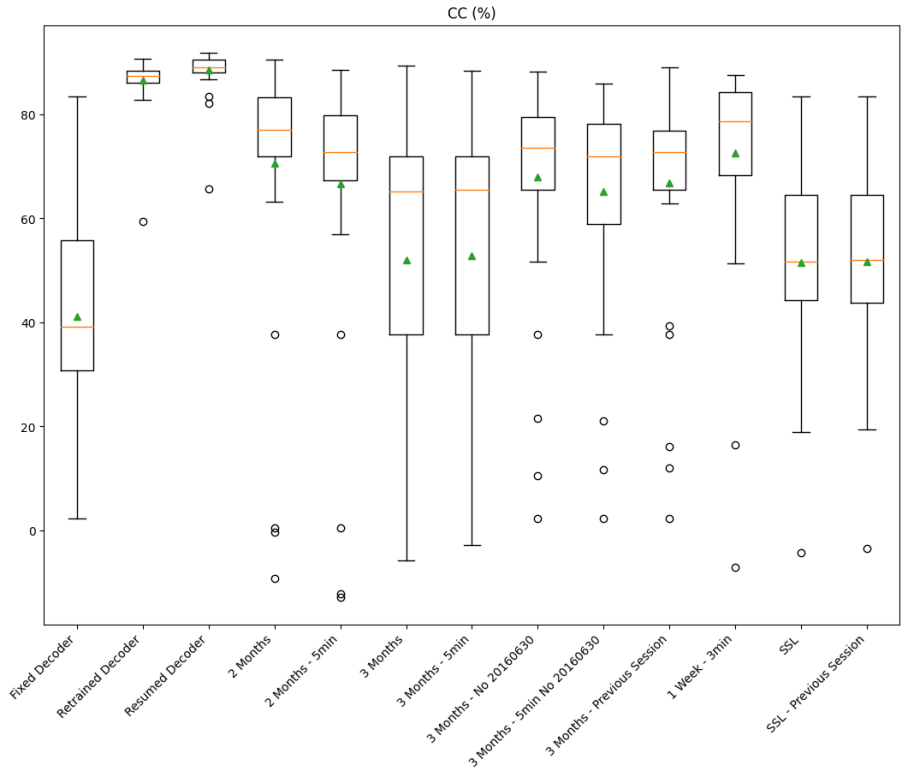


Figure 5.12: Comparison between Transfer Learning approaches in terms of CC.

Chapter 6

Conclusions

6.1 Result's analysis and future improvements

In this work, various transfer learning techniques were tested to maintain high decoding performance for decoding kinematic signals (x-y velocity) from neural signals (MUA) over a period of 10 months. In particular, the Fine-Tuning techniques returned better performances than the Self-Supervised Learning strategies, considering the totality of the registering sessions. The results obtained in this work are promising, as with fine-tuning every 2 to 3 months, it is possible to achieve prosthesis control with errors of approximately 48.04 $\pm$ 12.02 mm/s (refer to Appendix C), which is halfway between the lower bound (64.96 $\pm$ 15.38 mm/s) and the upper bound (32.26 $\pm$ 4.17). Although the results are promising and various strategies have been tested, further techniques must be explored to improve the robustness of the AI algorithm and reduce the number of recording sessions the patient has to attend. Ideally, the best solution would require a single recording session with labels, while fine-tuning should be done independently by the model in an unsupervised way.

One possible way to improve the robustness and at the same time the performance of the training itself is to discretise the output signal, i.e. the x and y velocities. The result would then be a step output signal, which can be post-processed to return a continuous signal by, for example, an interpolation operation to obtain a smooth movement. The purpose of discretization is to decrease the possible number of outputs, almost turning the regression task into a real-time classification task. From a functional point of view, it would be better to go for a suitable, non-uniform discretization. This is because a variation of 10 mm/s may be too small for high speeds (e.g. 200 or 250 mm/s), but may be too high for lower speeds and thus precision movements (e.g. 5 or 20 mm/s).

Once the discretization is completed, similar approaches to the ones proposed

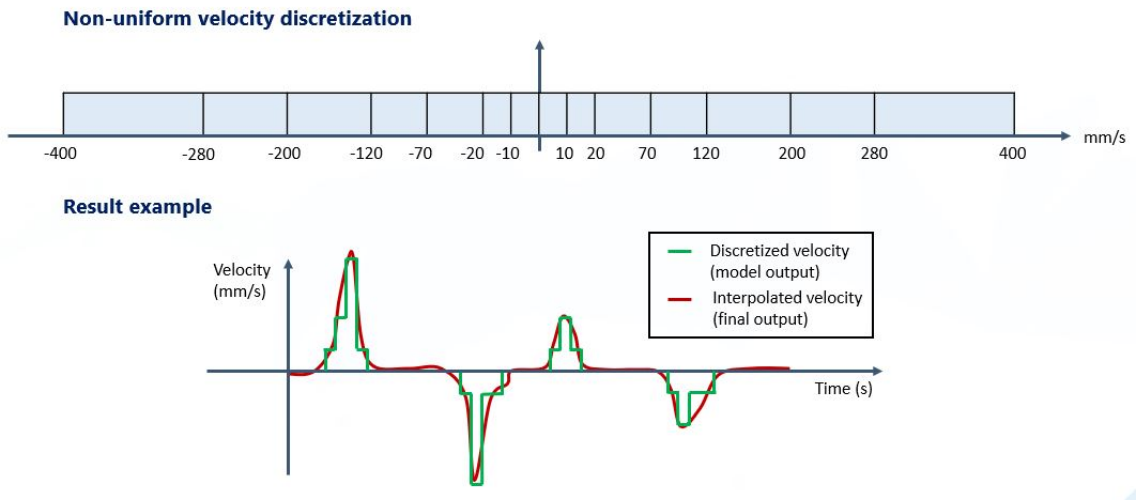


Figure 6.1: Example of velocity discretization (upper figure) and final output signal (lower figure). In green, the discretized signal returned by the model was highlighted. In red, the result of post-processing with interpolation.

in this work can be tested. For example, it could be interesting to see if the SSL strategies would return better performances. The idea behind the combination of SSL techniques to classification tasks is to gradually update the features that characterize a class. For example, let us consider a simple classification task in which the classes are characterized by a set of two features. In the simplest case, the classes (for example, three classes) are represented by a set of features that do not overlap each other. Once the model take the input and extracts the features of the input data, it has to decide in which class the sample belongs to, for example the nearest class. After predicting the new outputs, there should be new samples that defines the classes. By training the model using the new predicted samples, the model will learn how the class samples are evolving. In this way, the representation learned by the model will change at each sample and, gradually, the old knowledge will be replaced with the new knowledge, as shown in Figure 5.9 in Chapter 5.

This solution is not optimal when applied to a regression task, as the number of possible outputs belongs to the set of real numbers, which is much larger, even if a narrow range is considered (in this case, $\pm$ 400 mm/s). With the discretization and the definition of "N velocity ranges", there would be N classes described by M features and the model should be able to update the features of each class and return the correct range of velocity for each neural input signal time window.

Of course, SSL learning can be applied to any kind of classification task, such as grasping different types of objects, which has been analyzed in [25], [24] and [26]. However, due to the lack of multiple recording sessions in a wide time range, it was

not possible to try and validate this approach.

Given the unsupervised nature of the SSL approach, periodic fine-tuning may still be necessary to reset the model knowledge with new labels that are true and not predicted by the model itself. This combination of the SSL approach and Fine-Tuning could lead to increased robustness of the algorithm (due to the SSL approach) while maintaining high performance through periodic updating with true labels (due to Fine-Tuning) on a time interval much larger than 3 months.

This work is preparatory to the control of a two-degree-of-freedom robotic prosthesis by decoding the neural signals of a monkey, in particular the maintenance of high accuracy over time by increasing robustness through strategies that update the AI model to counteract the continuous changes in neural signals.

Robustness is a key factor in BCI applications, since the system has to guarantee the usability of the prosthesis despite neural changes. If the system is not robust enough, it would be necessary to conduct several recording sessions in a short range of time to update the model on the new neural signals and the respective kinematics signals. This, however, may entail a non-negligible commitment on the part of the patient and it may lead to the system rejection by the patient. Additionally, even if the patient was willing to endure several recording sessions, the chance of recording noisy signals (e.g. due to patient distraction, unwillingness to comply or incorrect positioning of the recording sensors) is greatly increased if the number of recording sessions is increased. This could eventually lead to low performance after the fine-tuning of the AI model.

Hence, it is necessary to find the best way to develop an effective and efficient system while requesting the lowest effort possible by the patient. To do so, new AI models can be explored, both supervised or unsupervised. If the model uses a supervised approach, it is necessary to have high robustness so as to require as few recording sessions as possible while maintaining high performance and smooth control of the prosthesis. If the model uses an unsupervised approach, inevitably the performance will be worse than a supervised algorithm. However, it must be assessed whether the difference in performance is compensated for by the fact that the model does not need to be updated periodically. A mixed approach can also be used, so that updates can be made in a much longer time frame than using supervised methods alone.

The results obtained in this work are promising and represent a starting point for further improvements. It was seen that the difference in performance is low between fine-tuning every week or every two months, which is a great result from the point of view of the trade-off between update interval and maintenance of high performance. With regard to the use of an unsupervised method, the results obtained were not up to the standard of other Transfer Learning methods. However, possible solutions and applications were provided to make the most of this training paradigm with

the data provided by the dataset.

The results will be used in the context of the B-CRATOS project to define appropriate strategies for fine-tuning the models to adapt them to the evolution of the subject's neural signals.

Appendix A

CINECA

MARCONI100 in an accelerated cluster based on IBM Power9 architecture and Volta NVIDIA GPUs (Tensor Core V100) acquired by Cineca within PPI4HPC European initiative. This machine allowed to perform the training and hyper-parameter optimization of BiRNN, BiLSTM and BiGRU in less than 6 hours, much less that it could have been possible with a normal laptop or workstation.

Node Performance		
Theoretical Peak Performance	CPU (nominal/peak freq.)	691/791 GFlops
	GPU	31.2 TFlops
	Total	220/300 GB/s
Memory Bandwidth (nominal/peak freq.)		220/300 GB/s

Appendix B

Results on the first session

Performance on TRS			
NN model	BRNN	BLSTM	BGRU
RMSE (mm/s)	21.23	12.93	17.97
CC (%)	90.97	96.59	93.51

Table B.1: Performance achieved on the Training Set of session "`indy_20160407_02`" by each RNN model.

Performance on VAL			
NN model	BRNN	BLSTM	BGRU
RMSE (mm/s)	30.1	26.88	29.05
CC (%)	84.56	85.52	87.65

Table B.2: Performance achieved on the Validation Set of session "`indy_20160407_02`" by each RNN model.

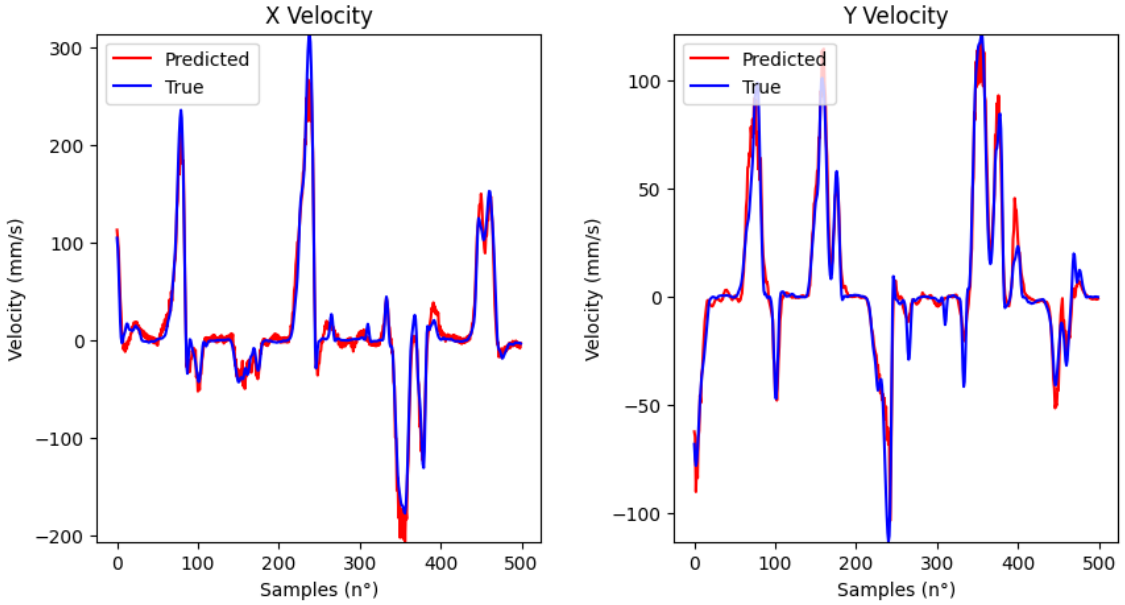


Figure B.1: Comparison between a portion of the TRS predicted signal by BLSTM with the true counterpart.

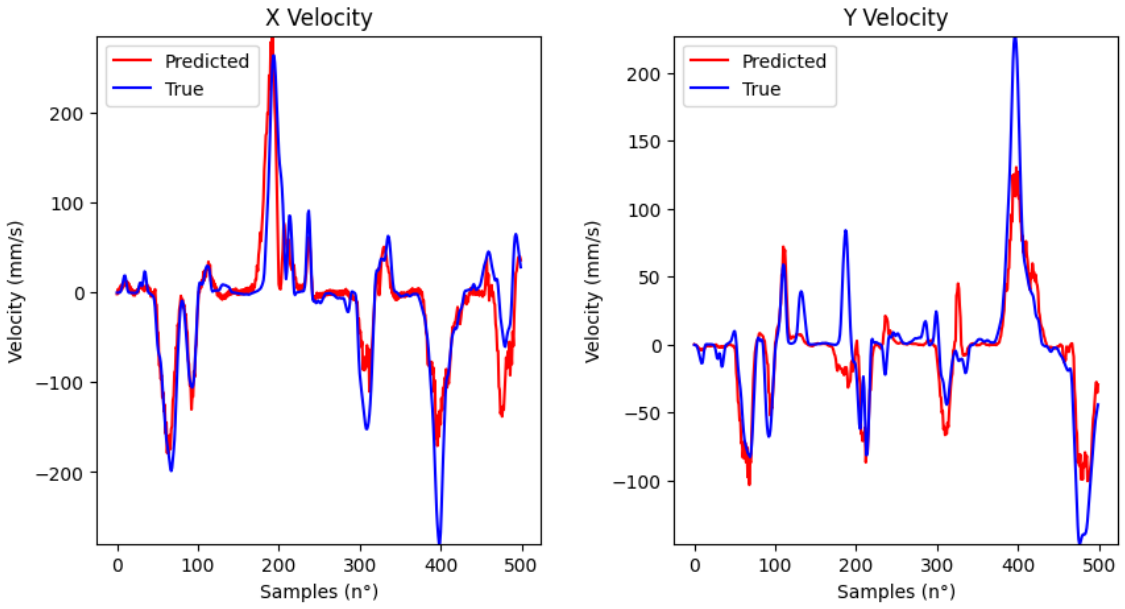


Figure B.2: Comparison between a portion of the VAL predicted signal by BLSTM with the true counterpart.

Appendix C

Summary table of the performance

Results including all the sessions		
TL technique	RMSE (mm/s)	CC (%)
Fixed Decoder	64.96 ± 15.38	41.06 ± 20.92
Retrained Decoder	34.89 ± 5.20	86.50 ± 4.90
Resumed Decoder	**32.26 ± 4.17**	**88.48 ± 4.36**
2 Months	**48.04 ± 12.02**	**70.63 ± 24.24**
2 Months - 5 min	50.99 ± 13.39	66.71 ± 24.62
3 Months	59.29 ± 18.46	51.98 ± 30.57
3 Months - 5 min	58.00 ± 16.06	52.77 ± 29.36
3 Months*	51.26 ± 11.59	67.91 ± 19.87
3 Months - 5 min*	53.95 ± 12.68	65.21 ± 19.95
3 Months - Previous Session	52.09 ± 11.59	66.84 ± 20.01
1 Week - 3 min	47.77 ± 11.31	72.64 ± 19.20
SSL	64.31 ± 15.15	51.59 ± 16.73
SSL - Previous Session	64.34 ± 15.13	51.61 ± 16.79

Table C.1: Performance achieved by all the proposed Transfer Learning techniques. In bold, the best technique is highlighted. The best Transfer Learning technique is underlined. The best technique, in terms of performance and interval between re-training, is underlined and in bold. The * symbol means that the session "`indy_20160630_01`" have been excluded from the training phase.

References

[1] "Nerve Tissue." https://training.seer.cancer.gov/anatomy/nervous/tissue.html.

[2] "Brain and Brain division." https://www.dana.org/article/neuroanatomy-the-basics/.

[3] "Homunculus: Somatosensory and Somatomotor Cortex." https://www.ebmconsult.com/articles/homunculus-sensory-motor-cortex.

[4] "Electroencephalogram (EEG)." https://www.health.harvard.edu/diseases-and-conditions/electroencephalogram-eeg-a-to-z.

[5] "Structure of Skeletal Muscle." https://training.seer.cancer.gov/anatomy/muscular/structure.html.

[6] A. C. Berlin C., *Production Ergonomics: Designing Work Systems to Support Optimal Human Performance*. CRC Press, 2006.

[7] "Spinal Cord Injury." https://www.paralysiscenter.org/spinal-cord-injury.

[8] M. Lebedev, "Brain-machine interfaces: an overview," *Translational Neuroscience*, vol. 5, pp. 99–110, 2014.

[9] P. N. Lawlor, M. G. Perich, L. E. Miller, and K. P. Kording, "Linear-nonlinear-time-warp-poisson models of neural activity," *Journal of Computational Neuroscience*, vol. 45, no. 3, pp. 173–191, 2018.

[10] T. Brochier, L. Zehl, Y. Hao, M. Duret, J. Sprenger, M. Denker, S. Grün, and A. Riehle, "Massively parallel multi-electrode recordings of macaque motor cortex during an instructed delayed reach-to-grasp task," *G-Node*, 2017.

[11] Y. Li, X. Zhu, Y. Qi, and Y. Wang, "Revealing unexpected complex encoding but simple decoding mechanisms in motor cortex via separating behaviorally relevant neural signals," *bioRxiv*, 2023.

[12] Y. Li, Y. Qi, Y. Wang, Y. Wang, K. Xu, and G. Pan, "Robust neural decoding by kernel regression with siamese representation learning," *Journal of Neural Engineering*, vol. 18, no. 5, 2021.

[13] M. R. Keshtkaran, A. R. Sedler, R. H. Chowdhury, R. Tandon, D. Basrai, S. L. Nguyen, H. Sohn, M. Jazayeri, L. E. Miller, and C. Pandarinath, "A large-scale neural network training framework for generalized estimation of single-trial population dynamics," *Nature Methods*, vol. 19, pp. 1572–1577, Dec. 2022.

[14] N. Ahmadi, T. G. Constandinou, and C.-S. Bouganis, "Robust and accurate decoding of hand kinematics from entire spiking activity using deep learning," *Journal of Neural Engineering*, vol. 18, no. 2, 2021.

[15] I. Goodfellow, Y. Bengio, and A. Courville, *Deep Learning*. MIT Press, 2016. `http://www.deeplearningbook.org`.

[16] "Recurrent Neural Network." `https://www.ibm.com/it-it/topics/recurrent-neural-networks`.

[17] "LSTM Networks." `https://towardsdatascience.com/lstm-networks-a-detailed-explanation-8fae6aefc7f9`.

[18] B. E. Penfield W., "Somatic motor and sensory representation in the cerebral cortex of man as studied by electrical stimulation," *Brain*, vol. 60, no. 4, pp. 389–443, 1937.

[19] E. Stark and M. Abeles, "Predicting movement from multiunit activity," *Journal of Neuroscience*, vol. 27, pp. 8387–8394, 8 2007.

[20] J. Wolpaw and E. Wolpaw, *Brain-Computer Interfaces: Principles and Practice*. Oxford University Press, 2012.

[21] A. K. Suresh, J. M. Goodman, E. V. Okorokova, M. Kaufman, N. G. Hatsopoulos, and S. J. Bensmaia, "Neural population dynamics in motor cortex are different for reach and grasp," *eLife*, vol. 9, p. e58848, nov 2020.

[22] M. G. Perich, P. N. Lawlor, K. P. Kording, and L. E. Miller, "Extracellular neural recordings from macaque primary and dorsal premotor motor cortex during a sequential reaching task," *CRCNS.org*, 2018.

[23] J. E. O'Doherty, M. M. B. Cardoso, J. G. Makin, and P. N. Sabes, "Nonhuman primate reaching with multichannel sensorimotor cortex electrophysiology," May 2017.

[24] E. Ghinato, *Robust Classification of a Neuromuscular Signal for Real-Time Control of a Prosthetic Hand.* PhD thesis, Polytechnic of Turin, Torino, Italy, Dec. 2022.

[25] I. Gesmundo, *Real-Time Classification of Neural Signal from Motor Cortex through Multiple Recording Sessions.* PhD thesis, Polytechnic of Turin, Torino, Italy, Mar. 2023.

[26] M. Lubrano, *Transfer Learning strategies for time robust neural decoding in a Brain-machine interface.* PhD thesis, Polytechnic of Turin, Torino, Italy, July 2023.

[27] S. Hochreiter and J. Schmidhuber, "Long short-term memory," *Neural Computation*, vol. 9, no. 8, pp. 1735–1780, 1997.

[28] K. Cho, B. van Merrienboer, D. Bahdanau, and Y. Bengio, "On the properties of neural machine translation: Encoder-decoder approaches," 2014.

[29] "Kerastuner." `https://keras.io/keras_tuner/`.

[30] "Sklearn metrics mse." `https://scikit-learn.org/stable/modules/generated/sklearn.metrics.mean_squared_error.html`.

[31] "Stats pearson." https://docs.scipy.org/doc/scipy/reference/generated/scipy.stats.pearsonr.html.