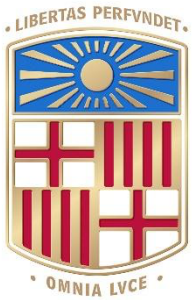

UNIVERSITAT DE BARCELONA

Final Degree Project

Biomedical Engineering Degree

Investigating the neural computations underlying the learning of a delay response task

Barcelona, 14th June 2021

Author: Leyre Azcárate Bescós

Director: Manuel Molano-Mazón

Co-Director: Albert Compte Braquets

Tutor: Roser Sala Llonch

Acknowledgements

I would like to thank all those who have helped me throughout the development of this project, without whom none of this would have been possible.

I wish to express my special thanks of gratitude to my director, Manuel Molano-Mazón, who has provided me with encouragement and patience throughout the duration of the project. I am extremely grateful to him for helping, teaching and supporting me during the development of the whole project.

I would also like to extend my gratitude to the co-director, Albert Compte, for providing an experienced point of view, valuable assistance and guidance. Many thanks to Tiffany Oña, the PhD of the group for her predisposition to help.

In addition, I would want to express my gratitude to my tutor Roser Sala, for her thoughtful comments, given feedback, and recommendations.

Finally, and on a more personal note, I would like to thank my family and friends for their unconditional support.

Thank you.

Abstract

The behaviour of experimental animals reflects their physical and cognitive state. Animal models are a fundamental tool and resource to study such states. When analysing behavioural studies, different learning patterns can be distinguished: a gradual improvement or a sudden understanding. The former is a progressive method used for developing a new behaviour by dividing it into several stages. In addition to gradual improvement, learning also occurs by abrupt understanding, also known as aha moment, which is defined as a moment of abrupt insight or discovery.
Lately, recent development of deep neural networks has had a remarkable impact on animal research. One strategy that has emerged as a promising tool for investigating the behaviour of animals performing a task is to study recurrent neural networks (RNNs) whose connection weights have been optimized to perform the same tasks as trained animals.
In this work we have created simulated networks that emulate the learning processes in animals. Specifically, we have trained Long Short-Term Memory (LSTM) networks, which are a special type of RNN, with a shaping protocol on a Delayed Response (DR) task, that is a typical approach for studying mice behaviour. For this purpose, we have used Reinforcement learning (RL), which concerns goal-oriented algorithms.
In order to analyse both mice and RNNs behaviour patterns, we have focused on the aha moment and compared their behaviours. We have complemented the study with an exploration of the effect of shaping in RNNs training.

Keywords

Cognitive task, computational model, aha moment, recurrent neural network, training, mice, learning.

Index

1 Introduction

Animal models are a fundamental tool and resource for scientists studying basic biological processes, diseases, pathogenesis, novel techniques and toxicologic research [1]. A usual experiment consists in training an animal to perform a task and recording its neuronal activity or only just analysing its behaviour while it is performing the task. The tasks can vary their complexity being able to be very specific, allowing the isolation of brain functions for the study of only one cognitive system.

In behavioural studies of learning, a difference between gradual and sudden performance improvements is frequently determined. Perceptual and motor skill learning is often characterized by gradual, incremental improvement. On the other hand, noticeable improvements in performance can occur suddenly, a phenomenon known as "insight". It is widely assumed that the neural changes that underlie insight are fundamentally different from the plasticity that gives rise to the more gradual forms of perceptual learning such as shaping. [2]

However, some tasks are too complex to be manageable by trial and error learning, and as a consequence, external guidance, called shaping, is decisive. The aim of **shaping** is to teach a new behaviour by breaking it into easily-achievable stages, which can be more or less depending on the training protocol. Through shaping, animals can be taught novel behaviours, guided by a task simplification.

On the other hand, **insight learning** is a category of cognitive learning in which the components of a problem are mentally rearranged or restructured in order to arrive at a sudden understanding of the problem and a solution, a concept developed to explain sophisticated behaviour that could not be the result of trial-and-error learning. The "**aha moment"** is related to this sudden comprehension of a problem or a situation that can bring to the solution. Although an insight may happen in the absence of any pre-existing interpretation, insights are frequently the consequence of the reorganization or restructuring of the elements of a situation or problem. It involves making a quick transition from not understanding the nature of a complex issue to understanding its deeper structure. [3]

In mice, the **delayed response (DR)** task is commonly used to examine its behaviour performing a task. It is a test in which the subject responds based on previously stored internal representations rather than information currently available in the environment. Typically, the organism receives a brief visual or auditory stimulus that is then removed, and after a few seconds, the organism is then presented with different response alternatives and is required to choose the one presented previously to obtain a reward. This task is a typical approach to study **working memory (WM),** which is a cognitive system that temporarily holds information and processes it. It allows to complete any task that requires information manipulation in a short amount of time. Moreover, it enables to keep track of the information needed right now and perform cognitive processes on it. [4], [5]

But, how can we make better sense of animal behaviour by using what we know about the brain? Scientists create **Artificial Neural Networks (ANNs)** to make computational models of the brain. These networks mimic the architecture of a nervous system by connecting elementary neuron-like units into networks in which they stimulate or inhibit each other's activity in much the same way neurons do. **Recurrent neural networks (RNNs)** are a class of ANNs that are often used as a tool to explain neurobiological phenomena, considering anatomical, electrophysiological and computational constraints. Recently, there has been large progress in utilizing trained RNNs both for computational tasks, and as explanations of neural phenomena. In contrast to real brains, artificial neural networks offer full access to the "neural circuit". In spite of existing several differences between both RNN and mice training, RNN can be used to better understand mice behaviour. [6]

1.1 Objectives

The main aim of this project is to create a RNNs able to replicate the behaviour of mice that have been trained to perform a DR task. The following specific objectives were set to achieve this major goal:

- Understanding how shaping helps mice learn the task
- Investigating if and how mice undergo insight learning
- Understanding the fundamental principles of Artificial Neural Networks and Reinforcement Learning
- Training Recurrent Neural Networks with the same shaping protocol used with mice
- Exploring the effect that shaping has on the Recurrent Neural Networks training
- Comparing network and mice behaviours when performing the same task and analysing both conduct patterns, extracting their similarities and differences

1.2 Methodology

The project was performed under the supervision of Manuel Molano and Albert Compte and consisted of the following stages:

- **Stage 1; Bibliographic research**: where several aspects of the study were examined. I focused on research which uses animal models that have been conditioned to perform

operational tasks. Following that, I explored deep learning techniques used to train animal models as well as RNN models trained to be as similar as possible to animal models. In addition, I supplemented the bibliographic studies with a visit to the CELLEX, a biomedical research centre located at the University of Barcelona's Faculty of Medicine, where the lab group trains mice to perform cognitive tasks. This visit provided a new perspective and helped me to reproduce the key features of the mice training. Moreover, the technical and economical feasibility, timeline and normative aspects of this study were assessed.

- **Stage 2; Mice Analysis**: I aimed at understanding mice behaviour by means of analysing several relevant parameters that were extracted from mice training. For that purpose, I focused on the aha moments that happen in a specific time of the training.

- **Stage 3; Networks developmental stage**: creation of an environment for network training. For that purpose, I used Python to simulate the DR task performed by mice. Though previous studies, such as M. Fradera's TFG, had already created an environment mimicking the task, we decided to design a new one from scratch with the purpose of creating a simpler and more efficient task. We arranged meetings with some group researchers for the task implementation to ensure that all features present in the real mice task were maintained in the networks' environment. These meetings were attended by the project's director and co-director Albert Compte, as well as Tiffany Oña, a PhD student in the group. Due to COVID-19, this project has been accomplished by daily meetings with the director and monthly meetings with the rest of the group. Furthermore, I was able to participate in some computational neuroscience and deep learning webinars organized by external [7] and internal lab community organizers to gain expertise in the field.

- **Stage 4; Networks Analysis**: we deeply studied the information acquired from networks training. Several parameters were compared in order to understand networks behaviour and to compare their behaviour from mice.

- **Stage 5; Editing stage:** it consisted in writing the present report and it was organized in the following way: First, I performed a literature review about Artificial Neural Networks and Reinforcement Learning algorithms, and I analysed the market of animal models. Then, in the detailed engineering section, I described all the work done during the developmental stage. Last, I performed the conclusion of the whole project. This stage also included the preparation of the oral presentation.

1.3 Limitations and scope

As this is a final degree project, time has been an important limitation. This project has been carried out from January 2021 until June 2021, for a total of six months.

Another large limitation has been the COVID-19 pandemic which has forced the project to be performed remotely. Meetings with the tutors were online and the interaction through emails and communication platforms such as Slack. Moreover, this has reduced the interaction with other students performing similar projects at the lab in IDIBAPS where the project could have taken place. Due to computational resources limitation, since I was using my laptop, we ran the networks in a computer in the lab. Then, though training was performed in a lab's computer, I had access to the information in order to analyse the results.

The scope of this project, considering the limitations mentioned previously, included:

- Analysis of mice behaviour through a shaping protocol
- Study of the fundamental principles of ANN and RL
- Training RNNs on a DR task
- Analysis of aha moments in both mice and RNNs
- Comparison between RNNs and mice behaviour when performing the same task.
- Comparison and analysis of RNNs' behaviour under shaping and no shaping
- Discussion of the results, errors detected and future work.

1.4 Location of the project

The project has been developed in collaboration with the Theoretical Neurobiology of Cortical Circuits research group at Institut d'Investigacions Biomèdiques August Pi i Sunyer (IDIBAPS). It has been performed at distance with the help of daily virtual meetings with Manuel Molano-Mazón, a postdoctoral researcher, and the co-direction of Albert Compte, the lab group's principal investigator, who assisted in the project's direction.
Roser Sala Llonch, Assistant Professor at the University of Barcelona's Department of Biomedicine, tutored and supervised this project.

2 Background

Whereas some behavioural characteristics are innate, many are learned from experience. Scientists define learning as a relatively permanent change in behaviour as a result of experience. An animal learns and is able to respond and adapt to a changing environment. If an environment changes, an animal's behaviours may no longer achieve results. The animal is forced to change its behaviour. It learns which responses get desired results and changes its behaviour accordingly.

In order to have a better understanding of the field of animal learning, some concepts such as shaping and insight learning will be defined in this section.

2.1 State-of-the-art

2.1.1 Behavioural shaping

Operant conditioning, also known as instrumental conditioning, is a method of learning attributed to Skinner [8], where the consequences of a response determine the probability of it being repeated. Through operant conditioning, a behaviour which is rewarded will likely be repeated, and a behaviour which is punished will occur less frequently. Skinner also introduced a new term called “reinforcement”, which means that a behaviour which is reinforced tends to be repeated and a behaviour which is not reinforced tends to be extinguished. Besides, he argued that the principles of operant conditioning can be used to produce extremely complex behaviour if rewards and punishments are delivered in such a way as to encourage moving an organism closer and closer to the desired behaviour each time.

Shaping, which is a variant of operant conditioning also developed by Skinner, is a powerful method in which novel behaviour (target behaviour) is created through successive reinforcement of behaviours, which become more and more similar to the target behaviour. Instead of waiting for a subject to perform a desirable behaviour, any behaviour that leads to it is rewarded. For example, Skinner found that, in order to teach a rat to push a lever, any movement in the direction of the lever had to be rewarded, until finally, the rat was trained to push a lever. Through shaping, animals can be taught impressive and novel behaviours. The capacity to form sameness and difference concepts was previously thought to be uniquely human. But now we know that even insects have this capacity [9].

Over the last decade, inspired by experiments on behaving primates, increasingly sophisticated procedures for performing robust perceptual behaviours have been developed for mice. In comparison to primates, in mice brain, cell-type-specific neurobiology is becoming routine. Transgenes can be directed at particular types of neurons. Then, these transgenes can be used to identify cell-types during recordings and to control circuit nodes during behaviour. As far as it is known, mice and other higher mammals have similar microcircuit organization of the brain [10].

2.1.2 Insight learning

When solving a problem, almost everybody has had the **"aha moment"**. After working on a difficult problem for some time, the solution appears in an unexpected flash.

Insight learning does not concern gradual shaping or trial and error. Instead, internal organizational processes occur that cause new behaviour. The term 'insight' refers to a clear and sudden understanding of how to solve a problem. Humans are used to undergoing these experiences, however, animals are also able to experience these insight moments. Psychological research on insight in animals begins with studies on chimpanzees by Köhler (1921), who agreed that insight learning is achieved through cognitive processes, rather than interactions with the outside world. An animal is said to show insight if an interval of poor performance with no clear tendency of improvement is followed by a sudden, abrupt and marked increase in performance.

In early 80s, learning was considered to be the result of reproductive thinking. This means that an organism reproduces a response to a given problem from previous experiences. Insight learning, on the other hand, does not directly depend on past experiences to solve a problem. Although previous experiences can aid in the process, an insight or innovative idea is needed to solve the problem. Despite a long history of study in the nature of insight experiences, there is still discussion over the specific mechanisms that distinguish problem solving that is accompanied by insight from problem solving that is not. A common distinction is that non-insightful solutions emerge from methodical, goal-directed, and strategic or analytic processes that are accessible to conscious awareness, while insight-based solutions do not. Nevertheless, evidence from several studies [11, 12] suggest that analytic processes are involved even in problems whose solution is followed by a sudden sense of insight, causing doubt on the usefulness of this distinction.

Most researchers [13, 14] believe that insight differs from trial-and-error or algorithmic problem-solving. The question of whether insight is a distinct type of problem-solving involving at least some distinct cognitive mechanisms, or whether it is simply an epiphenomenon based on the same cognitive mechanisms as non-insight solutions, continues to be debated.

2.1.3 Curriculum learning in RNN

Animals learn much better when the examples are presented in a meaningful order that explains gradually more concepts and step by step more complex ones than if notions are presented randomly.

As explained before in 2.1.1 section, by selecting which examples to present and in which order to present them to the learning system, one can guide training and significantly accelerate learning. Then, when talking about machine learning, one can think of the following question: can machine algorithms benefit from a similar training strategy? This idea of training a learning machine with a **curriculum** is similar to shaping: starting small, learning easier aspects of the task, and gradually increasing the difficulty level.

From a cognitive perspective, the question of guiding the learning of a recurrent neural network for learning a simple language and increasing its ability along the way was recently revisited, with evidence for faster convergence using a shaping technique [15]. Also making use of a shaping method, Hazy et al. [16] and O'Reilly et al. [17], suggested one of the most efficient and effective architectures in their prefrontal, basal ganglia, working memory (PBWM) model. They demonstrated their model with the 12-AX task, an abstract and hierarchical variant of the continuous performance working memory task that they created specifically for the purpose. The difficulty of this task emerges from its hierarchical organization, which concerns many subroutines. 12-AX task was also used in Krueger and Dayan [18], however, they studied shaping in a moderately complex hierarchical working memory task, demonstrating that it offers significant advantages for learning comparing with medium-term to long-term temporal demands. In this case, shaping evidenced benefits such as the speed, the better generalization over time and a higher flexibility in the face of task changes such as reversals. However, they also showed that shaping alone without the support of an allocation mechanism, can perform worse than no shaping.

Although shaping is widespread as a method of training subjects to perform complex behavioural tasks, its effects in computational models have not been extensively investigated. This may be due in part to the greater complexity of problems now being considered. In particular, recent work on learning programs with neural networks has relied on curricula to advance to longer or more complicated tasks. One reason for the slow adoption of curriculum learning is that its effectiveness is highly sensitive to the mode of progression through the tasks [19]. However, complete access to the activity and connectivity of the circuit, and the ability to manipulate them arbitrarily, has converted trained networks into a useful tool for biological circuits and a beneficial platform for theoretical investigation [20].

In spite of trying networks to behave as similar as possible to mice, there is a **stochastic factor** which is highly difficult to mimic. Belkaid et al. [18] have shown that animals are able to produce variable, unpredictable choices, especially when the reward delivery rule changes, is stochastic or is based on predictions about their decisions. Dominant theories of behaviour and particularly reinforcement learning (RL) rely on exploitation, which designates the process of repeating previously rewarded actions.

Task-trained artificial recurrent neural networks (RNNs) provide a computational modelling framework of increasing interest and application in computational systems, and cognitive neuroscience. RNNs can be trained, using deep-learning methods, to perform cognitive tasks used in animal and human experiments and can be studied to investigate potential neural representations and circuit mechanisms underlying cognitive computations and behaviour [21].

2.2 Previous work

This project is the continuation of two experiments conducted in the experimental neuroscience research groups at IDIBAPS in recent years.

2.2.1 Mice training in a Delayed Response task following a Shaping Protocol

The first study was conducted by Tiffany Oña, a PhD student of the IDIBAPS' Cortical Circuit Dynamics group, who trained mice to perform a delayed-response task using a shaping protocol. Information was introduced to mice as sound stimulus from both left and right sides during the DR task. Mice had to incorporate these stimuli and respond by licking one of the two available ports (left or right) after a variable delay to report which one was larger. Mice were rewarded with water if they got the answer correct. If the answer was wrong, the punishment was a 2-second timeout before the next hearing.

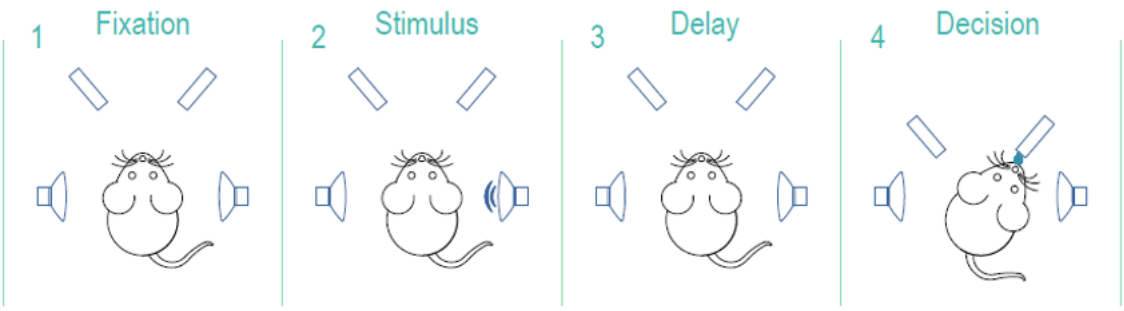

Figure 1: DR task training protocol is divided into 4 sections: 1. Fixation: the time it takes to start the trial; 2. Stimulus: the sound is played; 3. Delay: the time it takes for a decision to be made after the sound has been played; 4. Choice: the mouse can choose between the two ports by licking one of them.

The mice training was split into sessions by T. Oña. These sessions lasted approximately 300 trials, with the training sessions taking place on consecutive days. She divided the task into simpler sub-tasks using a shaping protocol because the whole task was too difficult for mice to complete all at once. T. Oña experimented with various training protocols before beginning the training to find the best shaping technique for mice learning. The final protocol included four stages, as well as an initial lick teaching stage to help the mice get used to it.

Nevertheless, T. Oña never trained mice in stage 4. Each stage had an objective that must be accomplished in order to move to the next one. However, the shaping protocol was created so gradual that the mice did not realize they were changing from one stage to the next one.

STAGE	DESCRIPTION	OBJECTIVE
Lick teaching	The mouse can obtain water from both lick-ports and it has to alternate at least every 3 licks.	Allow the animal to understand that the lick-port offers a reward and habituation to head-restriction.
Stage 1	Trial structure is introduced. Sound starts playing and after 300ms (ensure that they actually hear something), they can answer by licking one of the ports. The first answer is not penalized, so they can keep exploring until finding which port gives reward.	**Categorization learning**: animals learn to associate the sound origin with reward distribution. Animals naturally start understanding that there is a pattern and start showing asymmetric first lick, meaning that they tend to lick the corresponding port more than the incorrect one on the first try.
Stage 2	Similar to the previous stage but now the first lick counts. This means that the animal has to lick the correct port on the first try, otherwise it will have to wait until the next trial.	**Categorization learning:** animals continue to learn to associate the sound origin with reward distribution. Introduction of punishment (2s timeout) and only "one chance" tries to consolidate the pattern.
Stage 3	Slowly **start introducing the delay** component. In animals, there are some sessions needed to habituate to motor movement. Once they are used to it, trials are classified in three different difficulties corresponding to different delay lengths. Initially, the delay lengths are equivalent, but they increase nonuniformly if the animal keeps performing well until reaching a random distribution of trials of 0s, 1s and 3s duration.	**Timing learning**. Mice learn that, even though they know the correct answer, they have to wait until showing it.
Stage 4	Introduction of **ambiguity** in the sound. Instead of coming from either left or right, sound comes from both sides at different intensity volumes.	**Coherences** (sound difficulty). Animals are able to perform the task with ambiguity in the sound.

Table 1: Stages from shaping protocol developed by T. Oña

2.2.2 Training Recurrent Neural Networks with different protocols in Delayed Response Task

The second study is a Final Degree Project carried out by Marta Fradera Pruna under the supervision of Manuel Molano and Albert Compte (2020) in the Theoretical Neurobiology of Cortical Circuits research group. M. Fradera used Long Short-Term Memory networks to model the actions of mice performing a DR task. She used Reinforcement Learning (RL) to train the networks.

After confirming that the model can learn the task, she trained the networks using various shaping protocol variants, including training without shaping, with the aim of identifying the most relevant phases in the protocol, as well as those that may be ignored. She complemented the research by analysing the behaviour of the models after training.

This study aimed to see how the computational strategy formed by the networks varies significantly depending on whether they were trained with or without shaping.

3 Market Analysis

Task-trained artificial recurrent neural networks (RNNs) provide a computational modeling framework of increasing interest and application in computational systems and cognitive neuroscience. RNNs can be trained to perform cognitive tasks used in animal experiments and can be studied to investigate potential neural representations and circuit mechanisms underlying cognitive computations and behaviour [22].

Moreover, the models created in this project can be used either for testing hypotheses about neural circuits as well as for predicting animal behaviour in operant tasks, making it easier for the purpose of designing shaping protocols. Pharmaceutical companies, which regularly use animals for their experiments, will be specially interested in predicting animal behaviour for their studies, representing one of the main stakeholders.

The algorithm is principally aimed at **healthcare and biomedical facilities**, as well as **pharmaceutical companies** that conduct behavioural neuroscience research.

However, understanding aha moment principles and why they appear can provide interesting information to be used in different areas such as education or marketing. Comprehending why they appear can be a key tool in education in order to generate them and ease the learning process. Furthermore, other areas such as marketing are extremely interested in identifying the moment when new users first realize the value of their products.

3.1 Animal model

The animal model Market was valued at over USD 14.9 billion in 2019 and is projected to develop at a rate of over 7.5 percent between 2020 and 2026. The rising prevalence of chronic diseases, as well as the rise in drug side effects, have enhanced market growth.

Humanized mouse models are being developed in greater numbers as the market for personalized medicines grows. Furthermore, increased understanding of the use of animal models has resulted from government initiatives in various academic and research institutions.
Moreover, the current COVID-19 pandemic is an unprecedent public health threat, prompting an increase in vaccine and antiviral drug research and development. As a result, large numbers of animals are used to check the effectiveness and safety of vaccines and antiviral drugs prior to conducting human trials. COVID-19 epidemic is expected to have a major effect on the mice model market for vaccine and antiviral drug research and development. Mice models are the best small animal models for hepatitis B virus (HBV), hepatitis C virus (HCV), Zika virus, and cytomegalovirus (CMV), among other viruses.

Mice model market is especially expanding due to physiological similarities of mice to humans coupled with their ease of maintenance and breeding in the laboratory. Mice's genome is easily

modified and analysed, making them an excellent option for genetic testing studies in immunology and inflammation, oncology, central nervous system research, and diabetes research [23]. Because of the wide availability of transgenic mice that allow cell-type specific targeting, the mouse is a leading model system for mammalian circuit neuroscience [24]. Mice have recently become an advantageous animal model because of advances in both neuroimaging and genetic technologies, and because mice provide a large population of test subjects for behavioural screening [25].

Part of the most well-known names in the animal modelling industry are: Genoway SA, Eurofins Scientific SE, Crown Bioscience, Inc., Envigo CRS SA, and Transposagen Biopharmaceuticals, Inc. To expand their scope, these companies are engaging in acquisitions, mergers, and takeovers. Fortification strategies for brand new products are also being used by these players to improve their position in the global market. [26]

3.1.1 Restraints: Regulations and laws for the ethical use of animals in research

Research on basic animal biology and ecology is essential for increasing our understanding and for enhancing species conservation. However, it often causes the pain or death of many animals. [27]

Implementation of laws and regulations for animal protection and welfare has imposed restrictive practices and bans on the use of animals for different purposes. In the last five years, many countries have banned the use of animals in the cosmetic industry. [28]

However, studies agree that when deciding whether to use animals in experiments, we should only consider the importance of the purpose of the experiment, the quality of the research setup, including a consideration of which animals offer the best translation to the human situation, and the effects on the animals' interests. [29]

4 Conception Engineering

4.1 Project plan

As previously mentioned, our work is based on a previous study performed by M. Fradera which consisted of using RNNs to model actions of mice performing a delayed response task. In order to follow her project and obtain new results, we have trained RNNs on an analogous task to the one performed by mice. However, the code created to perform the task and to train the model was improved from M. Fradera because we have tried to simplify it and make a simpler but more efficient version.

First, we have explored mice training, then, we have designed the same task for networks analysing the same features. Concerning networks' training, in the first instance we have implemented the same shaping protocol as mice, and we have explored how different modifications in parameters can modify networks' learning. Following, we have not included the shaping protocol to examine networks behaviour in these conditions and become conscious of how much shaping influences.

In the case of mice, we have had access to data from real experiments, from where we have selected the more relevant information for our study. Regarding networks, the developed pipeline followed these steps: first, we have started with the task design in order to create a task for the RNNs similar to the one performed by mice. For that purpose, we have given some observations to the network that determine the action choices. As a consequence of this choice, the environment provides a (positive or negative) reward. Then, an analysis of the results obtained from the training of the networks has been performed. Finally, networks results have been compared with the ones previously obtained of real mice.

4.2. Project basis

Humans do not start thinking all over again every second. You comprehend each word as you read this essay based on your comprehension of previous words. You do not throw everything out and start over from the beginning. This issue is addressed by **recurrent neural networks.**

In order to deeply understand how recurrent neural networks work, it is essential to comprehend the principles on which RNN are based, beginning from the explanation of terms such as machine learning, deep learning and artificial neural networks. Therefore, in this section, the main concepts of machine learning are described.

Imagine you need to assemble a table and chair that you bought. How are you going to do it? Obviously, you will read the instruction manual that was given to you. You will create the entire set by following the instructions. If you do not have the instruction manual, you will have to figure out how to assemble the table and chairs on your own. These conditions are similar to **machine learning**, which uses advanced algorithms that learn a task without being explicitly designed to do

it. It usually does so by finding meaningful patterns in the dataset. Machine learning is divided into three categories: supervised, unsupervised, and reinforcement learning.

- Supervised learning: it uses labelled training data and a collection of training examples to deduce a feature. They are designed to learn by example. The name “supervised” learning originates from the idea that training this type of algorithm is like having a teacher supervising the whole process.

- Unsupervised learning: technique in which the model does not require supervision. It helps to find all kinds of unknown patterns of data. Clustering and association are types of unsupervised learning.

- Reinforcement learning: it is an area of machine learning that involves agents that should take certain actions from within an environment to maximize or attain some reward. As mentioned above, this was the methodology used for the study.

4.2.1 Artificial neural network

Artificial Neural Networks (ANNs) are machine learning models inspired by the architecture of brain circuits and formed by a set of units (neurons) that are connected via "synapses" with an arbitrary weight. These weights can be modified through training so the network can learn to do an arbitrary task. When networks have more than 2 layers we can talk about **deep learning**.

A neural network is a model that consists of a large number of neurons that are connected to one another. Each node represents an activation function, which is a type of output function. The weighting value for passing the connection signal is represented by the connection between any two nodes, which is equal to the memory of the artificial neural network. Network's performance varies depending on the network's connection method, weight value, and activation function [30]. For each unit, every input has an associated weight that defines its relative importance. When receiving the inputs, a basic unit (processing element) applies a linear transformation plus an activation function to compute the output. The linear transformation is computed through the following equation: $z = wx + b$**,** where x is a vector containing the inputs (x_1, x_2 …, x_n), w is a vector containing the corresponding weights for each input (w_1, w_2 …, w_n), and b is the bias vector. The neural network model is trained by optimizing the vector w using the input and output that we already have. Once z is computed, an activation function is applied (e.g. Rectified Linear Unit, sigmoid or tanh) to obtain the final output (y), as indicated in the following equation, where g is the activation function: $y = g(z)$

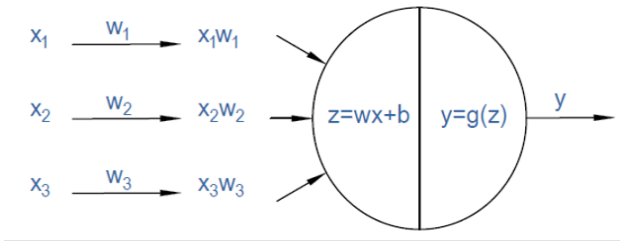

Figure 2: Neural network model. Every input ($x_1, x_2 \ldots, x_n$) is given with a different weight ($w_1, w_2 \ldots, x_n$) and bias (b) is the bias vector. With these parameters we can compute z. Then, in order to obtain the output (y), an activation function is applied (g in this case).

Neurons are arranged into layers and each layer contains a different number of neurons. The same inputs are given to all units in a layer, but with different weights. Each unit generates a result. The resulting collection of outputs is then used to feed another layer, and so on until the network reaches the final processing layer (or output layer), which generates the network's final output.

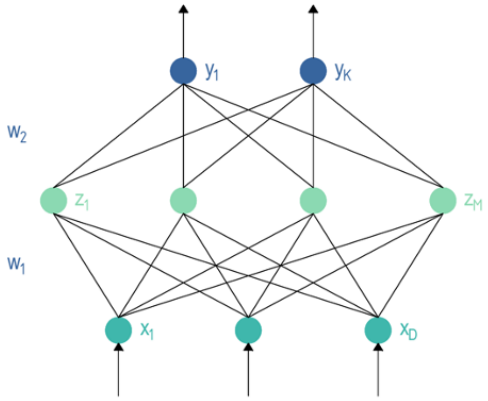

Figure 3: ANN architecture. Every input ($x_1, x_2 \ldots, x_D$) generates a result which will feed the next layer ($z_1, z_2 \ldots, z_M$). This value will depend on the weight applied ($w_1, w_2 \ldots, x_n$)

ANNs can present 2 main architectures: feedforward and recurrent. Regarding feedforward Neural Networks, the information is allowed to move in only one direction, from the input nodes through the hidden nodes and to the output nodes. However, **Recurrent neural networks (RNNs)** form a class of ANN models which are especially convenient to perform cognitive tasks which unfold across time, common in psychology and neuroscience, such as 'decision-making' or 'working-memory' tasks.

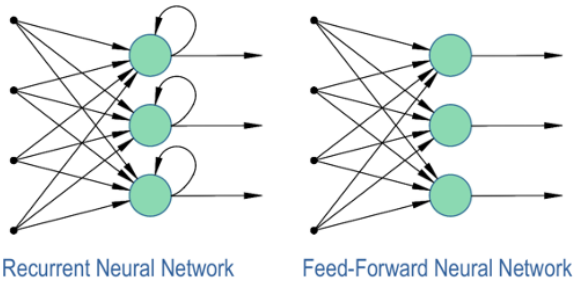

Figure 4: Main architectures for ANNs. RNNS can have information traveling in both directions by introducing loops in the network whereas in Feed-forward NNs there is no feedback, i.e., the output of any layer does not affect that same layer.

RNNs are made up of a network of loops that allow information to be predicted [31]. Previous information is easily accessible to recurrent neural networks. The activity of the RNN model is connected to the networks' activity at the previous time. Consequently, the activity of neurons in the network is affected not only by the current stimulus, but also by the current state of the network. The hidden state h_t at t determines the model performance. It is determined by the input x_t at t and the hidden state h_{t-1} at t - 1.

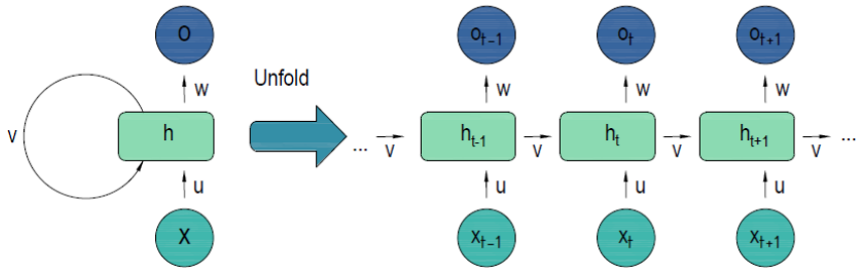

Figure 5: RNN working principle. The activity of neurons is connected to neurons' activity at previous time.

4.2.2 Long short-term memory networks (LSTM)

LSTMs are a special type of RNN, capable of learning long-term dependencies, i.e., those problems for which the desired output depends on inputs presented at times far in the past. All recurrent neural networks have the shape of a chain of repeating modules of neural networks. This repeating module in standard RNNs will have a very simple structure, such as a single tanh layer.

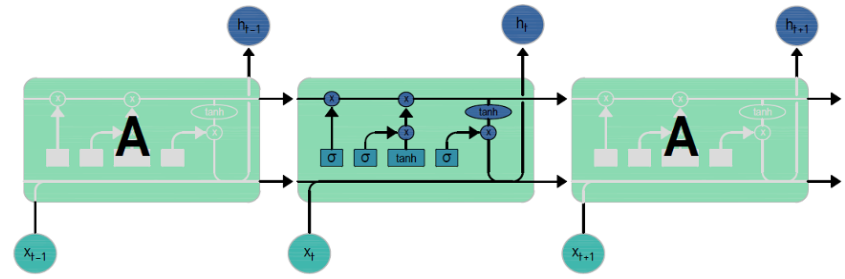

Figure 6: The repeating module of a standard RNN

However, in LSTMs, in spite of having this chain-like structure, the repeating module has a different structure. In this case, instead of having a single neural network layer, there are four interacting in a very special way.

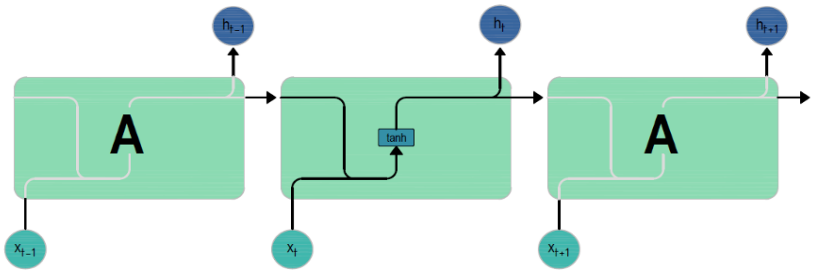

Figure 7: The repeating module in an LSTM which contains four main units: a cell, an input gate, an output gate and a forget gate. The cell remembers values over arbitrary time and the three gates regulate the flow of information into an out of the cell

The key of the LSTM model is the cell state (c_t), which only suffers some minimal interactions, and it is often unchanged. The LSTM is able to remove or add information to the cell state, meticulously regulated by structures called gates. This system employs various gates to dynamically learn when the network forgets historical data and when it needs to be updated with new data. These gates are a way to selectively allow information to pass through and are composed of a sigmoid neural net layer and a multiplication operation. The sigmoid layer can reach values ranging from 0 to 1, indicating how much of each component should be allowed to pass. A LSTM has three of these gates in order to control the cell state.

4.2.3 Networks training

RNNs are typically trained using supervised learning (SL) techniques, which consist of learning from a training set of labelled examples. Supervised learning is very effective and can be used in almost any situation. Nonetheless, a **dataset** is required, and it is insufficient for learning from interaction.

In SL each example has two parts: an input object and a desired output that specifies the appropriate action for the system to take in that situation (the label). Because the desired outputs for given inputs are known, the network can be trained to produce them using **gradients** computed using the backpropagation algorithm. RNNs use optimising strategies, such as gradient descent to minimize a **loss/cost function,** which evaluates how well an algorithm models a dataset. The loss function computes the error for a single training example whereas the cost function is the average of the loss functions for all the training samples. The reason for doing this is because a lower error between the actual and the predicted values means that the algorithm has learned the task. Thus, when calculating the error of the model during the optimization process, a loss function must be chosen. In one dimension it is quite easy to find the loss function but when increasing dimensions, it is not as simple, and we need to use gradient descent. **Gradient descent,** which is an algorithm to search for this minimum, is one of the most widely used optimization techniques, and it is the most frequent method for optimizing neural networks. The gradient of a function simply measures the change in the function caused by a small variation in its parameters. In machine learning, we use gradient descent to update the parameters of our model. Parameters refer to coefficients in Linear Regression and weights in neural networks. Essentially, the algorithm uses two pieces of information to reach the minima:

- The direction to go
- The size of the step to reach the destination (bigger or smaller)

The algorithm is able to make these decisions by means of derivatives, which are calculated as the slope of the graph at a particular point. Therefore, if we are able to compute this gradient, we will be able to compute the desired direction to reach the minima.

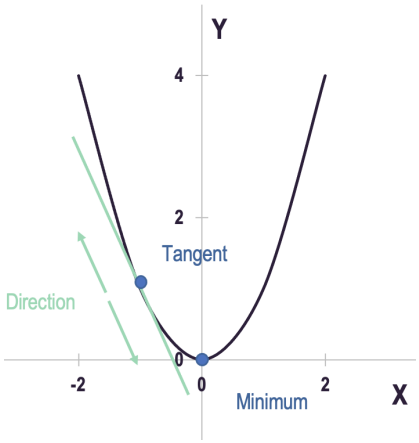

Figure 8: Gradient descent representation for an example loss function ($Y = X^2$). Gradient descent is an algorithm that looks for the minimum of the function by means of the direction and the size of the step to reach the destination. It measures the change in the function caused by a small variation in its parameters.

As said before, we use Reinforcement learning (RL) as the learning strategy for our networks. It is an area of machine learning concerned with how agents require taking actions in an environment in order to maximize the cumulative reward. It is one of three basic machine learning paradigms, next to supervised learning and unsupervised learning. Reinforcement learning comes into play when examples of desired behaviour are not available but where it is possible to score examples of behaviour according to some performance criterion. This method approximates in the best way how mice learn the task, by receiving reward every time they do a correct action. RL models have advanced our understanding of how animals learn and make decisions, and how the brain supports some aspects of learning. When using RL, networks only receive sparse feedback (a reward or a punishment, for correct and error choices, respectively), without being explicitly told the correct answer. Any RL task is defined by three things: observations, actions and rewards. Observations are a representation of the current world or environment of the task. Actions are something a RL agent can do to change the environment and rewards are what the agent receives for performing the right actions. The observations, actions and rewards are then transformed in a dataset. It differs from other computational approaches in that it emphasizes an agent's learning from direct interaction with its environment. The objective is to learn a “policy”, something which tells which action to take from each state in order to maximize the reward [32]. The reward is necessary to tell the system which state-action pairs are good or not. This reward can have a different value for the agent depending on the moment when it is received. The **discount** or gamma parameter (γ) is used to indicate how much the agent values future rewards, being γ=0, if the agent only cares about his first reward, or γ=1 if it values all rewards equally.

Backpropagation Through Time (BPTT) is an algorithm used to update the weights of certain types of RNNs. The backpropagation training algorithm aims to reduce the error of a neural network's outputs in comparison to any predicted output in response to corresponding inputs by modifying the weights of the network. However, one of the drawbacks of BPTT is that as the number of time steps increases, the computation also increases. As a result, the overall model would become noisier. Hence, **Truncated Backpropagation Through Time** method (TBPTT), which is a modified version of BPTT, is the approach chosen for our study. By using it, the sequence is

processed one time step at a time, and periodically the BPTT updates is performed for a fixed number of time steps, which is called **rollout** [33]. The rollout is the number of steps we use to update network weights, in other words, how much do we look back in time to learn.

Moreover, deep neural networks can be trained with RL where we convert rewards and observations into labels and samples and apply standard SL techniques to increase the probability of doing the actions that maximize reward and decrease the probability of actions that decrease the reward.

5 Detailed Engineering

In this project, we have developed a parallel analysis between both mice and networks behaviour with the aim of comparing them. First, with the purpose of understanding mice behaviour, we have performed a deep exploration of several parameters extracted from mice training. During this research, we have studied the aha moments that enable mice to learn to categorize. On the other hand, we have followed the same protocol to train RNNs as similar as possible to mice. With these results, we have studied the same features in mice and RNNs in order to compare them.
Then, with the purpose of determining how much shaping affect RNNs, we have performed an exhaustive analysis of parameters such as the punishment and the rollout. The following section will describe all the details necessary to understand the process. All the algorithms are written in the Python programming language and are public and available on GitHub [34, 35, 36].

5.1 Analysing mice behaviour following a Delayed Response task

The objective of this first research is to understand and analyse relevant parameters extracted from mice training in order to compare it with RNN. In order to analyse mice behaviour, two scripts have been used. They are located in different GitHub repositories ('CV-Learning' and 'ngym_shaping') under the same name of 'mice_behav_analysis.py' [34].

First, we have tried to understand some variables obtained from mice by means of representing different features, such as the accuracy over sessions for each subject. At this point, we have realized that we were losing information by looking at sessions, so we have changed our point of view and started looking at the **trials** (each session is equivalent to 400/500 trials).

At the end of each trial, performance takes a value of 1 if the answer is correct and 0 if is incorrect. In order to obtain a more visual value of the performance, and to properly observe the evolution of the network, we have performed a convolution to smooth the performance values. The vector with the performance values is convolved with a vector of values 1/*convolution_window* and length *convolution_window*, following the discrete convolution operation:

$$(a * v)[n] = \sum_{m=-\infty}^{\infty} a[m]v[n-m]$$

Moreover, the analysis of these plots indicates that the stage in which subjects stayed the longest time is 3. Then, we have analysed **motor and delay variables**, which are only active in stage 3. Since only in stage 3 the motor is activated and we could find delays, this was the stage that could offer us more information. Then, we decided to create an additional fourth stage in order to subdivide stage 3 into two. In this way, the fourth stage would be generated when two conditions were met: subject is at stage 3 and motor 6.0 is on. After doing this extra division of the stages, we realized that at some point mice's behaviour started to be strange, and T. Oña confirmed this was due to the fact that a lot of them suffered surgeries at some point of the training. Therefore, we

decided to consider these events (surgery, sickness, wounds) and remove all the information stored after they happened. In order to confirm our results, we decided to add more data and confirm our results with more mice. For this reason, more mice were trained and added to the experiment. In view of this, two batches of ten (from N19-N28) and six mice (from C17-C22) were added. As a result, we obtained a detailed study of mice behaviour.

In figure 9 we can see an example of a subject whose performance is represented in different colours depending on the stage it is. Represented in dashed vertical lines we can observe if the mouse has suffered any event, as in this case it has been submitted to a surgery and has been sick. Also in vertical lines we can observe the session change, due to the fact that there is one every 400/500 trials.

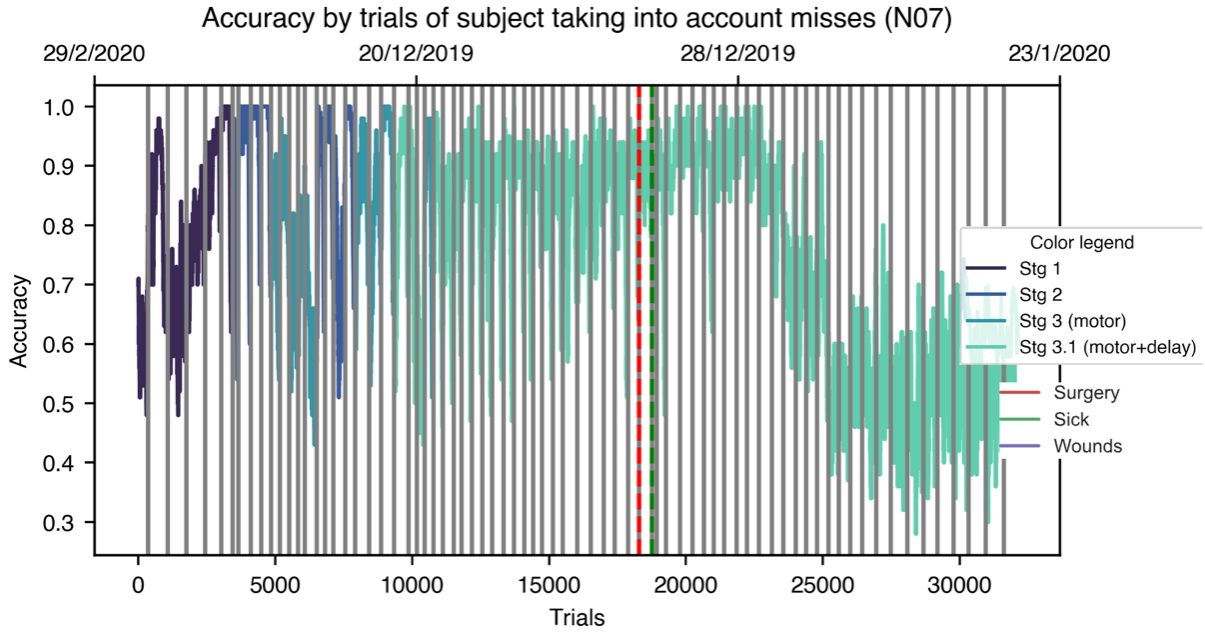

Figure 9: Accuracy by trials of subject considering misses and events. Gray lines correspond to session changes and coloured dashed lines to events (red: surgery; green: sick; blue: wounds).

After analysing the accuracy for different stages, we have decided to focus on **stage changes** in order to understand how much stage change affects the performance. Then, as in stage changes it is very difficult to perceive where mice learn, we have examined **stage 1**, where **categorization** takes place, which is the stage where mice learn that the right and left ports are in different positions.

With the purpose of defining when this categorization takes place, we have defined when mice have learned the assignment whereas when they are choosing the correct decision by chance. We have defined lower (Start of learning period) and upper (End of learning period) thresholds, so if the performance goes from being below the low threshold to above the upper threshold means that mice have learned.

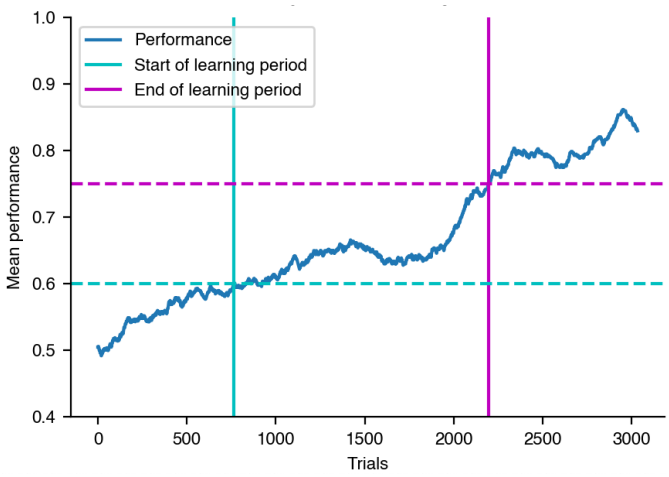

Figure 10: Lower and upper thresholds which determine if the subject has learnt. Lower and upper thresholds are set to 0.6 and 0.75 respectively. Moreover, the difference between them corresponds to the learning time needed, being able to classify the learning into slower (gradual) or faster (abrupt).

By means of exploring the time from one threshold to the other we can determine that **some mice learn faster than others**. Moreover, we can observe that learning does not always follow the same pattern so it can be more gradual or more abrupt. Then, we focused on these fast training (aha moments).

5.1.1 Aha moments

Aha moments are defined as the sudden comprehension from not understanding a problem to understanding its deeper structure. In our case, we can detect them by means of the lower and upper performance thresholds. If these thresholds are close enough, we will find an aha moment. The two requirements necessary to find an aha moment:

- A **sequence of 10 trials** (*window_aha*) with an average performance **above 0.75.**
- A **difference** between the previous and the after performance of 0.2.

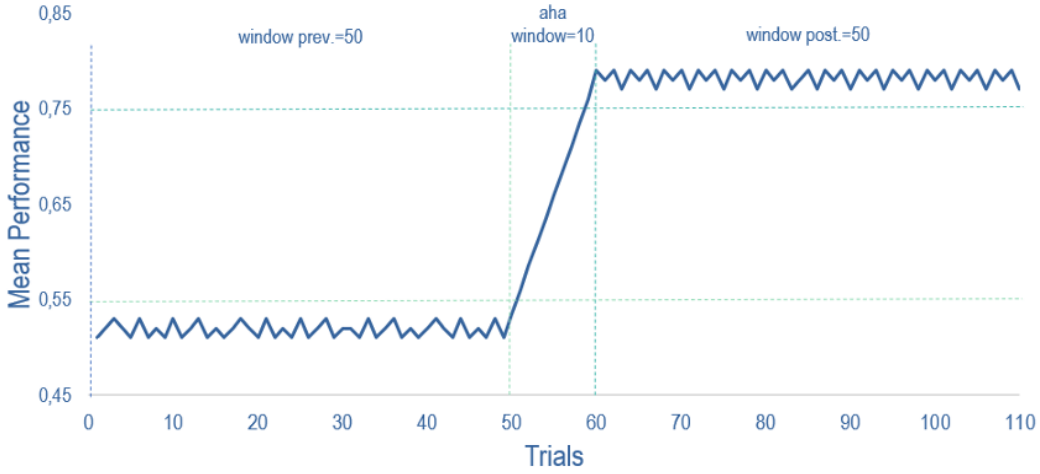

Figure 11: Aha moment representation for mice performance. To find an aha moment two conditions must be fulfilled: a sequence of 10 trials with an average performance above 0.75 and a mean performance difference between before and after the aha moment of 0.2.

The sudden increase of the performance is associated with a **streak of correct choices**. For mice, we have determined the aha moment as the change in performance from values with a performance difference above 0.2. Moreover, we have defined a variable for the window of the aha moment, so we were able to change it and see how this affected the number of aha moments detected. We have established this window to 10 trials.

The major question when studying aha moments is **why they appear**. There may exist some common pattern, such as repetitions, in all these aha experiences which is the origin of the learning. For that hypothesis, we have studied what happens before this moment. We have explored the **probabilities of choosing right or left**, and we have compared it to the probability of choosing the correct answer randomly by chance.

5.2 Task creation

As previously mentioned, neural networks' principal aim is to simulate as much as possible mice behaviour performing a delayed response task (DR). In order to achieve this goal, we have created the same task, which consists of combining two different stimuli and reporting which one is larger after a fluctuating delay. The task creation was performed on a script which is located in the 'ngym_shaping' Github repository under the name of 'DR_stage.py' [35].

The process of developing the task has consisted of mimicking as much as possible the training that Tiffany Oña uses for the mice, which is possible following a shaping protocol that helps mice to learn. First, we have defined the task of interest, and then, we have trained a recurrent neural network (RNN) to perform the task.

The environment provides our agent with some input observations at each time step. Then, the agent must choose between one of three possibilities (the same as mice): fixation, left or right (corresponding to left and right ports in mice task). After that, the environment provides positive or negative feedback to the agent after taking an action. This feedback helps the agent learn how to act in order to get the most positive reward, which is done by means of adapting the network's parameters in order to find the optimal input-output relation. Punishment was set to zero unless it was specified, so as positive (+1) and no negative reward (0) were given at the end of the trial depending on whether the agent's choice was the same as the network's ground truth.

As T. Oña did, we separated the task into trials. In our setup, each trial lasted 600 ms (plus a variable delay) divided into time-steps of 100 ms. More in detail, each trials always followed the same structure composed of four different intervals:

- **Fixation period:** during this time, animals must withhold their lick for the trial to start. If they keep licking, the mice will not pass to the next step. (200 ms)

- **Stimulus period:** the stimulus is presented to the agent (400 ms)

- **Delay period:** delay component is introduced to the task. After the stimulus, a waiting period is applied before letting the agent answer, so it must retain the information. The delay is variable. (0, 1000, 3000 ms)

- **Decision period**: the agent makes a choice (200 ms)

5.3 Training of RNNs

After defining the task of interest, the recurrent neural network (RNN) model is trained to perform the task. The process by which the network model updates its parameters in order to find the best input-output relationship is referred to as **network training**. Indeed, though several methods may be used to perform the training, the approach chosen for this project is reinforcement learning (RL). During the training phase in RL, the network communicates with the environment: the agent outputs an action at each time-step, and the environment responds with a reward and an observation. Moreover, as explained in section 4.2.2, we have trained the networks using the training algorithm Truncated Backpropagation through time.

Then, we must specify the model we would employ. The used RL algorithm and the desired policy network must be defined when creating the model. The chosen policy is LSTMPolicy, a policy object that uses an LSTM architecture to enforce actor-critic.

Finally, TensorFlow toolbox is necessary to train the generated model. Model training was run in the lab's computer on a script which is located in the 'ngym_shaping' Github repository under the name of 'example_neurogym_rl.py' [35]. Later, information was stored for further analysis.

5.3.1 Shaping

Task difficulty is managed by **four stages,** where the higher the stage, the higher the complexity. In consequence of the agent's performance, the stage increases. This means that when the performance stays above a threshold (in our case 0.75) during an established number of sessions, the stage increases. Nevertheless, if the performance does not accomplish the minimum of sessions above the threshold, the process is restarted.

- **Stage 0:** During the decision period, agents are rewarded by performing either a left or right action. The agent cannot repeat the same action more than three times in a row to ensure that all acts are linked to the reward. Otherwise, it will not be rewarded again until it switches sides. The main objective of this stage is to ensure that the agent understands that both actions (right and left) may provide a reward.

- **Stage 1:** Stimuli are introduced and during the decision period, agents can examine both actions until they find the one that gives reward without ending the trial. In this stage, agents receive reward on every trial in order to help them to find the optimal parameters to do

correct actions. The main objective of this stage is 'Categorization learning', which aims to ensure agents learn to associate each stimulus with the ground truth action (left or right).

- **Stage 2:** this stage is the same as the previous one, with the exception that in this stage, the agent must choose the correct choice in the first try. Alternatively, the agent receives a negative reward and the trial is finished. The main objective of this stage is to consolidate the knowledge from the previous stage (stage 1), so the principal aim continues being 'Categorization learning'. In the task implementation, this stage is the simplest one, and the other stages will be the same as stage 2 but with some modifications in each of the cases.

- **Stage 3:** delays are progressively introduced between stimulation and decision in order to recognize if the agent is retaining the information. We used three different delays (0, 100, 300 ms), which were incremented from 0 to 300 as the agent's performance increases in a stable way. However, they increase nonuniformly if the agent keeps performing well until reaching a random distribution of trials of 0, 100 and 300 ms duration. The main objective of this stage is 'timing learning', which indicates that, even though the agent knows the correct answer, it must wait until showing it and retain the information during a period.

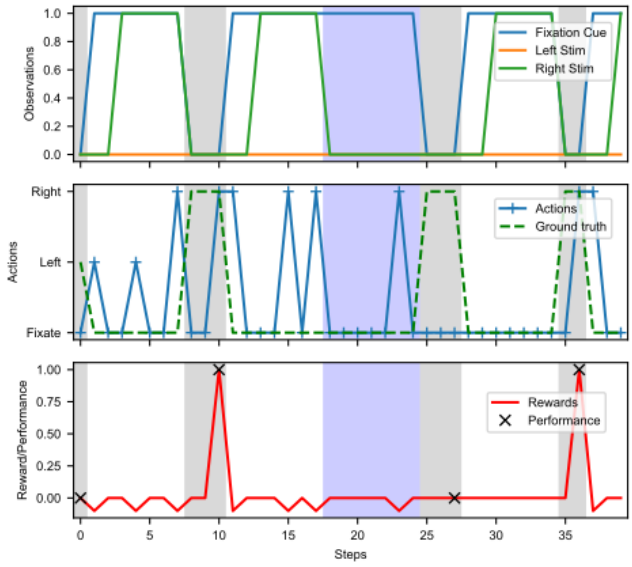

Figure 12: Stage 3. Time-steps are represented on the x axis. The purple area depicts the delay period whereas the grey area corresponds to the decision period. In the top panel: the environment provides observations (blue: fixation, orange: left stimulus, green: right stimulus). In the middle panel: the actions that the agent takes (blue), and the correct actions (dashed green). In the bottom panel: rewards (red) and performance at the end of the trial (black crosses)

- **Stage 4:** ambiguity and noise in the stimulus are introduced. The main objective of this stage is to assure that the agent is coherent, meaning that it can perform the task with ambiguity in the stimulus.

After designing the stages of the shaping protocol we have explored network training and how several parameters, such as the discount factor, affect the behaviour.

5.3.2 No shaping

Despite mice training has been performed under a shaping protocol, we have implemented a non-shaping protocol for networks in order to determine how much shaping affects learning a task. Besides the results obtained from comparing the training done with and without shaping, several parameters can be modified in order to explore training under different conditions. For this purpose, we have trained various instances in several training variants.

- The **rollout** study consists of analysing network performance for different rollout values. The rollout is the number of steps we use to update network weights. Then, a high rollout means that the task is more complex because more information needs to be processed. Each network has an associated value for the rollout used which can be extracted from the file name, for example, "*no_shaping_long_tr_one_agent_stg_4_nsteps_40"* indicates that we have used no shaping protocol with one agent, who is always in stage 4 and that uses a rollout of 40.

- **Punishment** increasement: different values of 0, -0.25, -0.5, -0.75 and -1 have been probed in order to observe if harder values of punishment make networks have a better or a worse performance. Then, for every network, punishment, and instance we could find a folder. For example the folder "*pun_-0.75_inst_2"* contains data from a punishment of -0.75 and the third instance (there are instances 0, 1 and 2). Inside this folder there is all the data stored so we can easily find this data stage and order inside the stage by the following pattern: "MeanPerf_bhvr_data" + stage + order inside the stage.

5.4 Analysis of RNNs

The last step consists of the analysis of all the information acquired under different protocols and different parameters. In order to analyse RNNs behaviour, we have created a script which can be found in the 'CV-Learning' GitHub repository under the name of 'analysis_rl.py' [35].

With the purpose of defining when the categorization takes place in networks, we have used the same method as in mice: defining a lower (Start of learning period) and an upper (End of learning period) thresholds, so if the performance goes from being below the low threshold to above the upper threshold means that networks have learned.

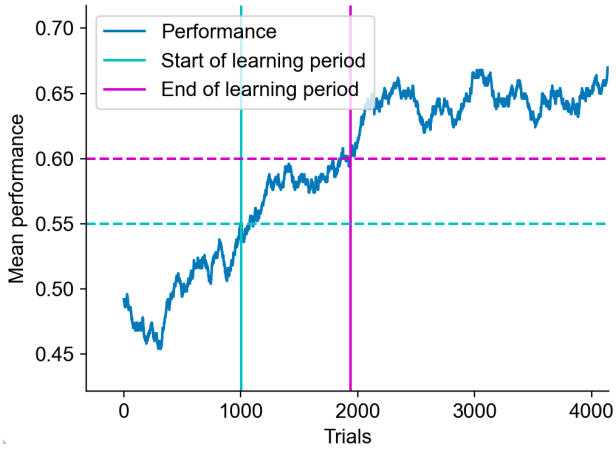

Figure 13: Lower and upper thresholds which determine if the network has learnt. Lower and upper thresholds are set to 0.55 and 0.6 respectively. Moreover, the difference between them corresponds to the learning time needed, being able to classify the learning into slower (gradual) or faster (abrupt).

By means of exploring the time from one threshold to the other we can determine that **some networks learn faster than others**. Then, we focused on these fast training (aha moments).

5.4.1 Aha moments

In the same way as in mice, we have explored aha moments. Depending on the protocol used, the parameters of aha moment may differ. In our case, using a shaping protocol, the objective of this aha moment is to reproduce when the networks have learnt to **categorize**, which is a piece of learning acquired at stage 1, so a requirement for networks to experiment an aha moment is to be in stage 1. Independently of the protocol used, there are two requirements necessary to find an aha moment in networks:

- A **sequence of 10 trials** (*window_aha*) with an average performance **above 0.65.** This means that for 10 trials, at least 7 should be correct.
- A **difference** between the previous and the after performance of 0.1.

For networks, we defined the aha moment as the difference in performance of 0.1 because they do not fluctuate as much as mice. In networks we have also defined a variable for the window of the aha moment of 10 trials. Despite the aim of defining an aha moment for networks as similar as possible to mice, almost all parameters need to be modified.

For networks we have also performed the probabilities of choosing right al left before the aha moments in order to check if there is any pattern.

6 Results and Discussion

6.1. Analysis of mice behaviour

Aiming to understand how the learning process in mice takes place we have performed several analyses that show the performance of the mice exposed to different training periods (stages) (Fig. 14). The performance of the mice shows a smooth learning without sharp changes at the moment in which the stage is changed (Fig. 15).

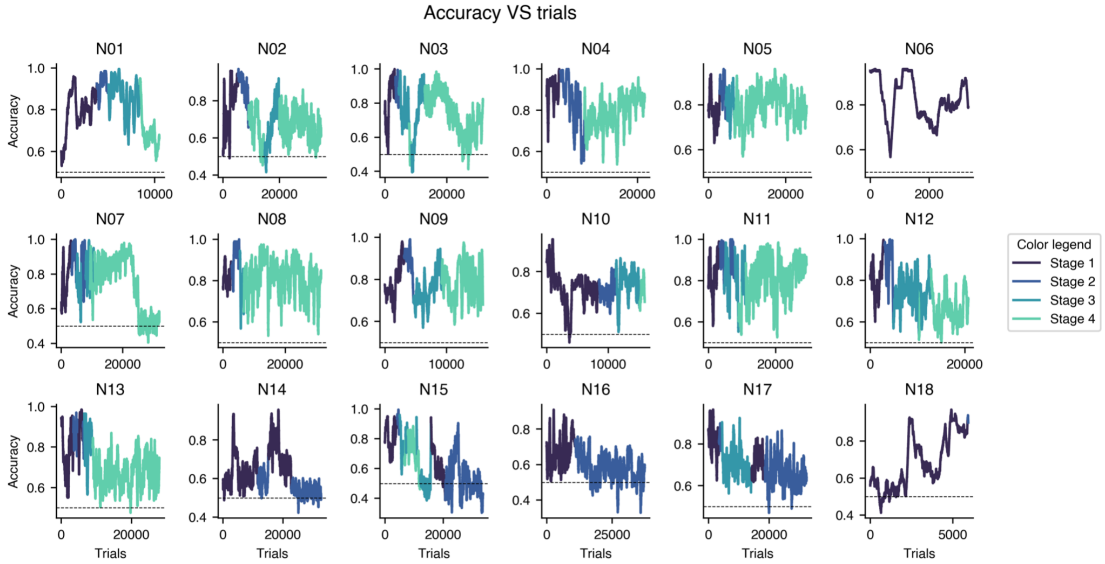

Figure 14: Accuracy VS trails with four stages. Each subject, and depending on their performance, reaches stages 1, 2 and 3 at different moments in time. Trials are represented in the x axes. Originally training was divided into 3 stages, but we added one more (stage 4) to explore the effect of the motor on the performance of the mice.

6.1.1 Performance at stage change

This gradual progression of the performance may be due to the shaping protocol used, which is able to divide the entire task into small steps which reduces as much as possible the impact that the increase in difficulty has on the mice performance. For instance, in stage 3, delays are increased gradually to help the animals learn to wait. A key moment to see if learning is being gradual are stage changes. In figure 15 we can observe the performance before and after the stage change, where it is shown that the learning is being fairly gradual, since although there is a change in environmental conditions, the subjects continue achieving a similar performance. The stage change that has the most impact on the performance of the animals is the one from 2 to 3. However, this could be due to other external factors that affect the behaviour of the animals (for instance, some mice underwent surgery once they reached a good performance in stage 3).

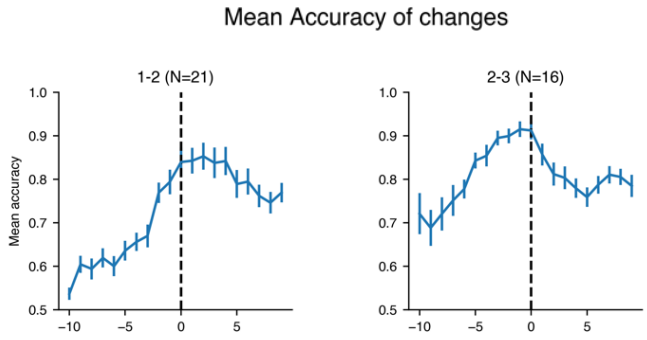

Figure 15: Mean accuracy at stage changes from 1-2 and 2-3. N shows the total number of times the change has been accomplished for all subjects (note that each subject can experiment one type of change several times). For both changes it is observable a decrease in accuracy associated with an increase in difficulty.

6.1.2 Learning to categorize the stimuli

In spite of its gradual learning, we hypothesized that mice must present sharp performance changes that reveal that different aspects of the task have been learnt. For instance, in stage 1, mice need to learn to categorize the stimuli, i.e., to associate each stimulus to its corresponding port. We first investigated how long it took for the animals to learn this association. For that we identified two different periods within stage 1: an initial period during which the animal performs at chance level and a later period during which it already displays a good performance (Fig. 16).

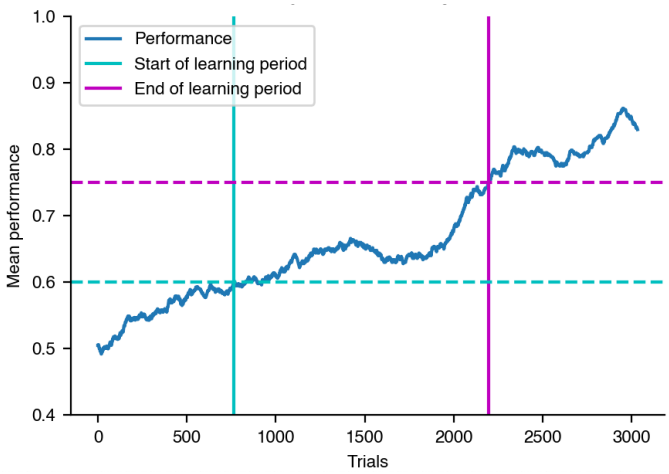

Figure 16: Dashed lines show the lower (cyan) and upper (magenta) thresholds used to determine if and when the subject has learnt. Lower and upper thresholds are set to 0.6 and 0.75, respectively. The learning time is defined as the time elapsed from the moment the animal starts learning (i.e. performance goes above lower threshold) and the moment at which it finishes learning (i.e. the performance goes above the upper threshold).

By means of exploring the time from one threshold to the other we can determine that **some mice learn faster than others** (Fig. 17). Moreover, we can observe that learning does not always follow the same pattern: it can be more gradual or more abrupt. For this reason, we have examined the time each subject needs to learn to categorize (Fig. 17). In this case, the time has been measured with the **number of trials needed to learn**. As it is shown in Figure 17 some subjects are able to learn the task in less time (<800 trials) than others (>1200).

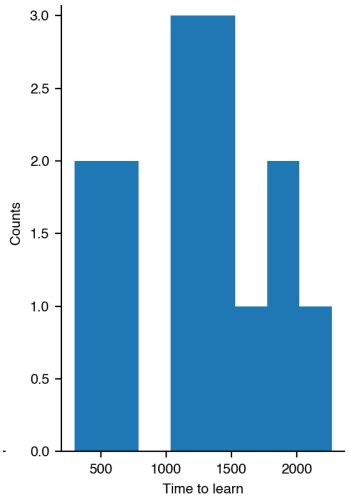

Figure 17: Number of trials needed to learn for the 21 subjects. 1276,93 ± 539,05 (mean ± SD)

6.1.3 Aha moments

A caveat of the above analysis is that learning times have been obtained by smoothing the mice performance with a large window (500 trials). This procedure allows better categorization of the periods at the expense of temporal resolution. This prevents us from seeing fast and abrupt changes. For this reason, we applied a different analysis aimed at discovering faster learning processes (aha moments) (Fig. 18). To detect the aha moments some requirements must be fulfilled as stated in section 5.3.2.

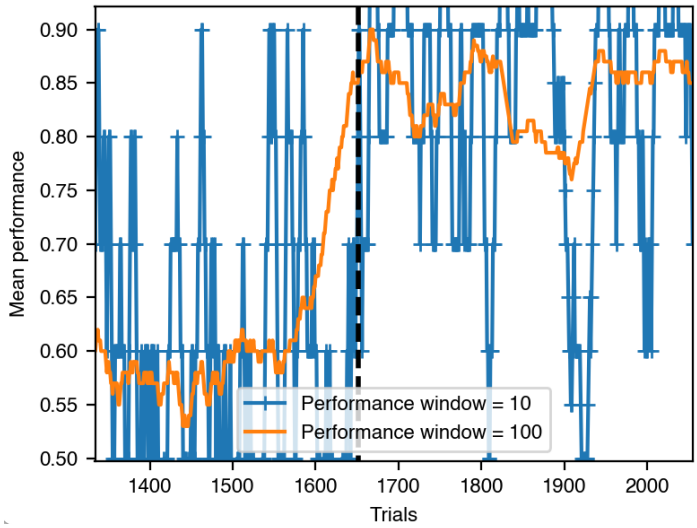

Figure 18: An aha moment example displayed by mice. Trials in x axis and mean performance in y axis. Performance is displayed in blue and orange with a convolution window of 10 and 1000 respectively. The vertical dashed black line indicates the aha moment.

Aha moments do not occur in all subjects. We have found that 88,8% of mice experience aha moments. On the other hand, subjects can undergo up to 4 aha moments in a single training although the average is 2 ± 1,23 (mean ± SD) (Fig. 19). Therefore, 60% of the mice undergo an aha moment more than once during the training.

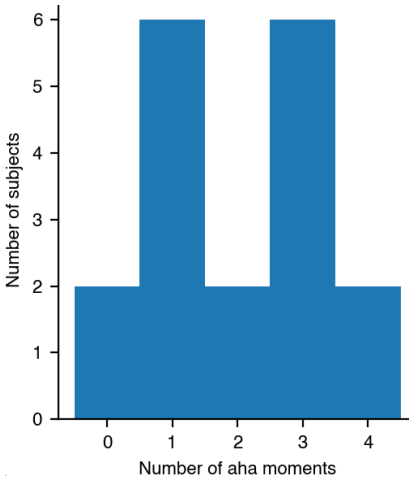

Figure 19: Count of aha moments by subject. The most usual is to do one or three aha moments. Only 2 subjects perform no aha moments.

6.1.4 What causes aha moments?

In order to understand the origin of the aha moments, we have explored whether there is any unbalance between the number of right and left rewards before and during the aha moments. We hypothesized that mice might learn to categorize the stimulus after experiencing a very stereotyped sequence of left/right trials. For this reason, we computed the number of right-side trials before and during the aha moments. However, we have not found any consistent deviation (Fig. 20, orange traces) from what it would be expected by chance (Fig. 20, blue traces). Further work should be done to explore the possibility that more intricate sequences trigger the aha moments.

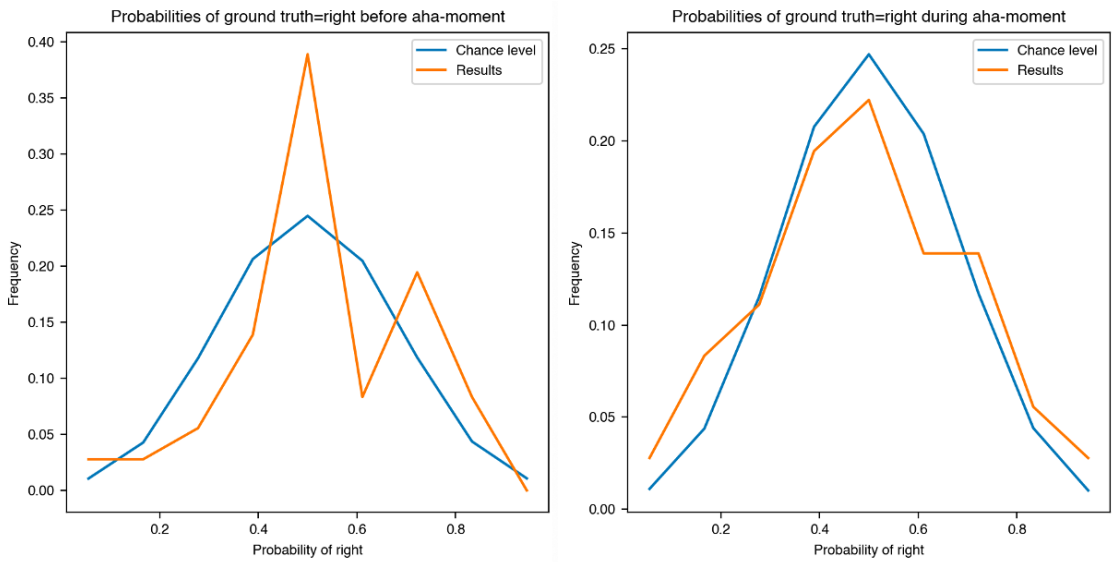

Figure 20: Histograms showing the probability of right-side trials before and during the aha moment VS what would be expected by chance. We did not find any consistent deviation from chance.

6.2. Analysis of networks learning (simulated data)

To better understand the behaviour observed in mice during the shaping training, we have trained RNNs using a simplified version of the shaping protocol (see point 5.3.1).

We found that RNNs were able to learn the categorization required at stage 1. However, networks did not achieve very high mean performances.

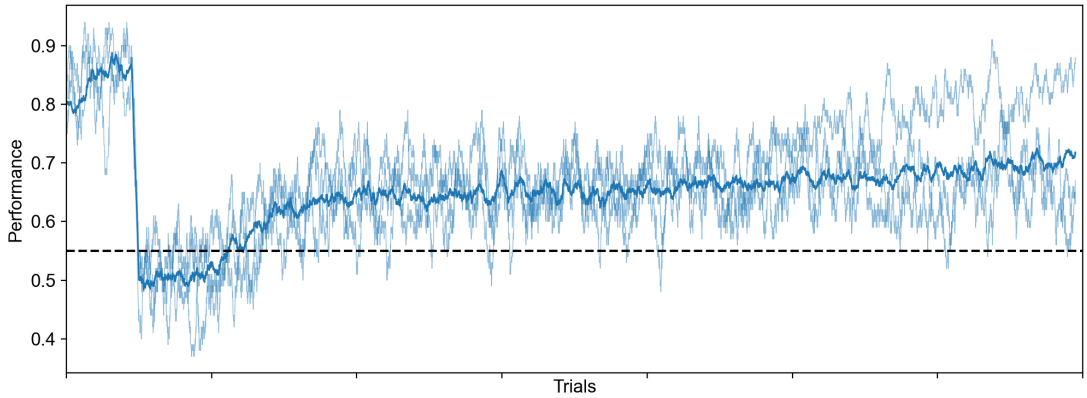

Figure 21: Performance of networks trained during the first 2 stages of shaping. The abrupt decrease in performance corresponds to the change from stage 0 to 1.

6.2.1 Learning to categorize the stimuli

As we did for mice, we investigated how fast RNNs learned to categorize the stimuli. As it is shown in Figure 22, networks seem to have several stages of learning. For the learning period in networks, we have established lower and upper thresholds of 0.55 and 0.6 respectively so all the networks have learnt.

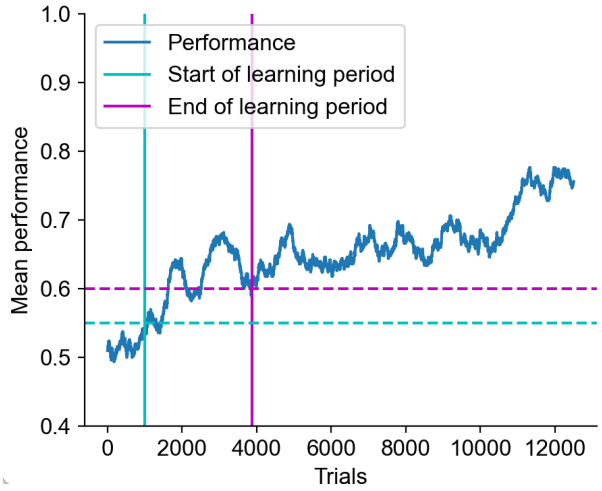

Figure 22: Dashed lines show the lower (cyan) and upper (magenta) thresholds used to determine if and when the subject has learnt. Lower and upper thresholds are set to 0.55 and 0.6, respectively. The learning time is defined as the time elapsed from the moment the animal starts learning (i.e. performance goes above lower threshold) and the moment at which it finishes learning (i.e. the performance goes above the upper threshold).

In the same way as in mice, we have concluded that **some networks learn faster than others** (Fig. 23).

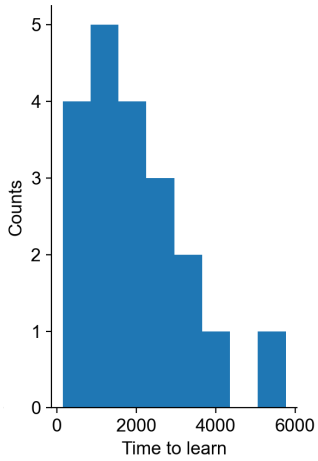

Figure 23: Number of trials needed to learn for the 20 networks. 1911,9 ± 1361,11 (mean ± SD)

6.2.2 Aha moments

In order to detect the aha moments for the RNNs, we adjusted some parameters in reference to those in mice. We considered that RNNs need a larger number of trials to learn. Moreover, networks mean performance grows more slowly but steadily than in mice. For this reason, the window used to compute the performance before and after the aha moment was made larger than the one used for mice (1000 instead of 100). Moreover, the difference between the before and after performance was set to 0.1 instead of 0.2.

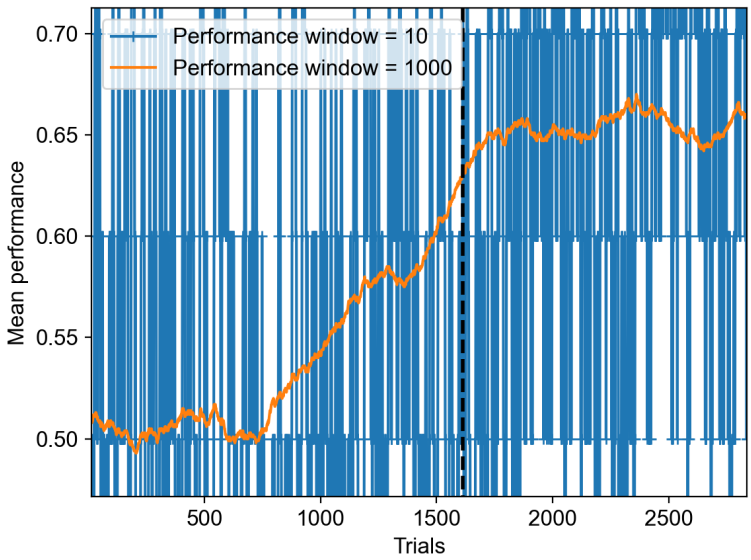

Figure 24: An aha moment displayed by an example network. Trials in x axis and mean performance in y axis. Performance is displayed in blue and orange with a convolution window of 10 and 1000 respectively. The vertical dashed black line indicates the aha moment.

6.2.3 What causes aha moments?

As we did for mice, in order to understand why aha moments appear, we have explored the sequence of trials before and during this moment. For this purpose, **probabilities of choosing right and left** were assessed in the same way as done in mice. As with mice, we did not find a clear difference between the distribution of probabilities of right-side trials before and during the aha moments and the one expected from a completely random process (i.e. a binomial distribution).

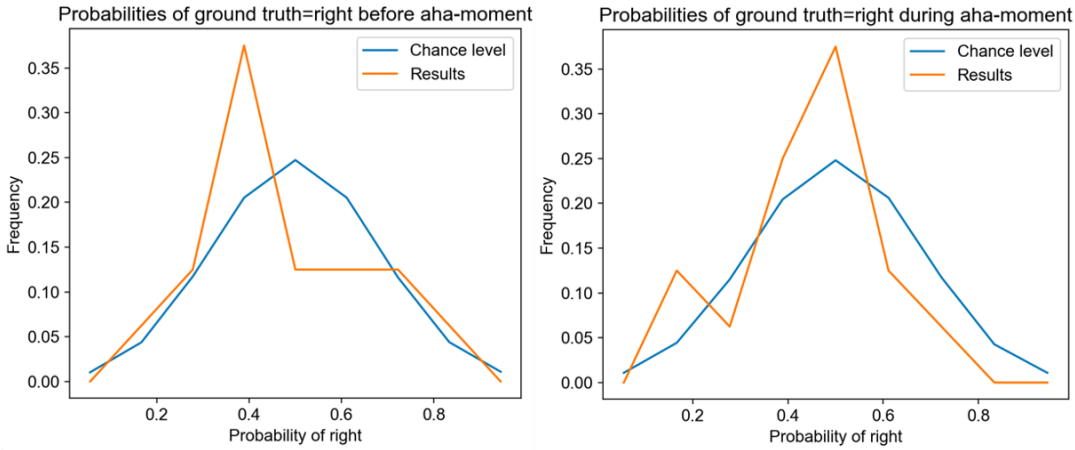

Figure 25: Histograms showing the probability of right-side trials before and during the aha moment VS what would be expected by chance. We did not find any consistent deviation from chance.

6.3 Comparison between mice and networks behaviour

The training implemented for the neural networks cannot be but a simplification of the real one. For instance, animals are penalized with a *timeout* of 2 seconds when they make a mistake, while RNNs are immediately presented with the next trial. Furthermore, the shaping protocol used in mice contains several details that we decided to exclude from the RNNs protocol for simplicity. For example, animals can be moved to an easier stage if they show a poor performance, while the training of RNNs can only increase in complexity.

Furthermore, the behaviour of the mice is affected by various phenomena that are not present in the RNNs: stress, urgency, satiation, tiredness, sickness... Consequently, the comparison between the behaviour of mice and networks can only be done at a qualitative level. When networks have learnt a task, if the environment does not change, they exhibit a constant performance, however, mice present a fluctuating performance. Despite they show slightly different behaviour, in both mice and RNNs we can find aha moments where subjects fastly understand the task. However, in order to find them, different requirements were for mice and networks. **When working with mice, the timescales are shorter,** so the window used to compare the performance before and after an aha moment is of 100 trials, whereas when dealing with networks we have used 1000 trials. Furthermore, we used a performance difference of 0.2 to detect the aha moments in mice, while that value was too high for the networks and needed to be set to 0.1. Moreover, as explained above,

mice behaviour is much more unstable, so when experiencing aha moments they present sharper performance changes than networks.

6.4 Comparison between shaping and no shaping in networks

Previous results obtained from M. Fradera's work stated a clear difference between the performance of networks under shaping and no-shaping. Nevertheless, we have found that this difference critically depended on two parameters used for the networks' training: the value for the **rollout** used in M. Fradera's project was much larger than the one we have used in our experiments (40 instead of 5). A large rollout implies that the task is more complex because more information needs to be processed. Further, the **discount factor (γ),** explained in section 4.2.3, plays a key role. In the original simulations, the γ was set to 0.99, meaning that the agent values rewards that are received both early and late almost equally. This likely affects the way RNNs learn the stage 1, in which they always receive a reward, with the only possible improvement of receiving it earlier. In our simulations, we set the γ to a value of 0.1 so networks will tend to avoid any delay in the reward.

As a consequence of simplifying the task and decreasing the rollout to a value of 5, we realized that, in contrast to M. Fradera findings, shaping does not provide any advantage over no-shaping. Indeed, no shaping demonstrated even better results, where networks achieved higher values for the performance.

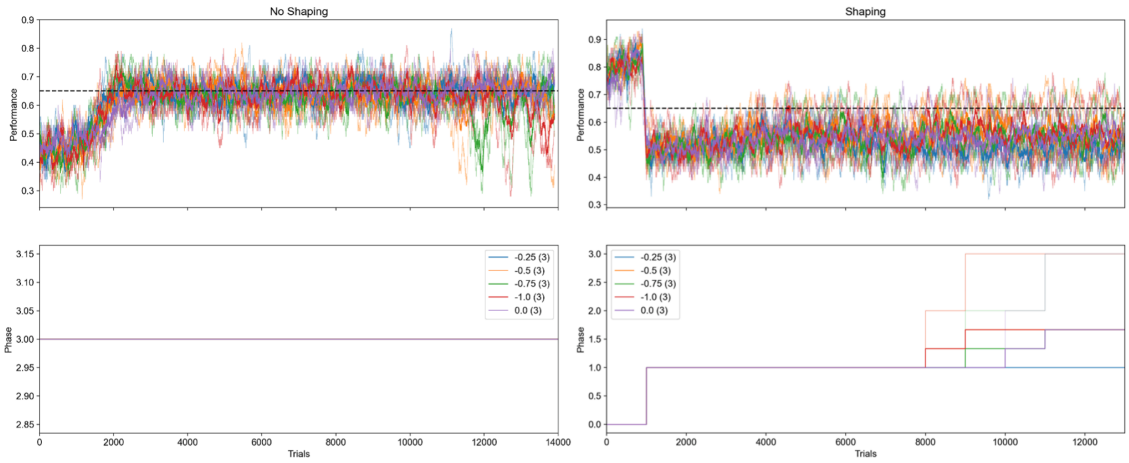

Figure 26: No shaping VS Shaping for rollout 40. No shaping accomplishes better performance, achieving a mean performance of around 0.65, whereas the shaping method gets stuck in stage 1 with a performance of 0.5 (i.e. chance level).

Networks using shaping were not able to learn. Nonetheless, making a simpler and more elementary task changing this "learning window" to 5 let the networks learn and gradually increase their performance.

6.4.1 Importance of rollout value

To confirm the above results, we have trained the networks under no-shaping protocols with different rollout values. We have found clear differences for no-shaping networks when trained with 5 and 40 rollouts. In order to understand how networks behave under different rollout values we decided to train the same networks only varying this value. Therefore, in figure 27 we can see that the higher the rollout, the less networks learn. This decrease of the learning appears in a gradual way as the number of steps we use to update network weights increases.

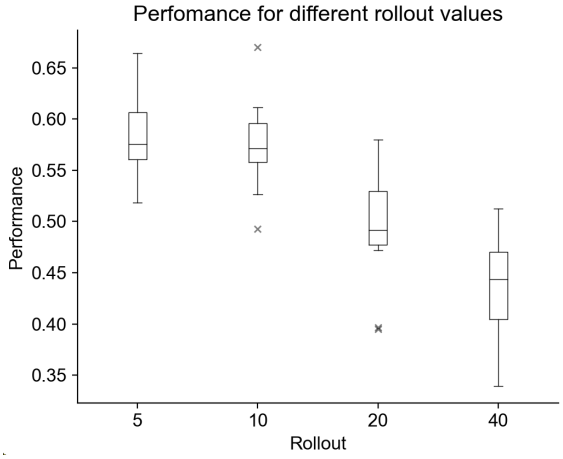

Figure 27: Gradual decrease of the performance as the rollout value increases. Rollout values from 5 to 40. Boxplot is used, where the box extends from the lower to upper quartile values of the data, with a line at the median. The whiskers extend from the box to show the range of the data.

6.4.2 Importance of punishment value

We have also explored how the amount of punishment affected the learning of the networks. In section 5.1 we set punishment 0, meaning that if the agent does not decide the correct answer, the environment will not provide any negative reward. Here, several punishment values have been trained in order to observe how the punishment affects performance and also what is the punishment that best represents the one given to mice.

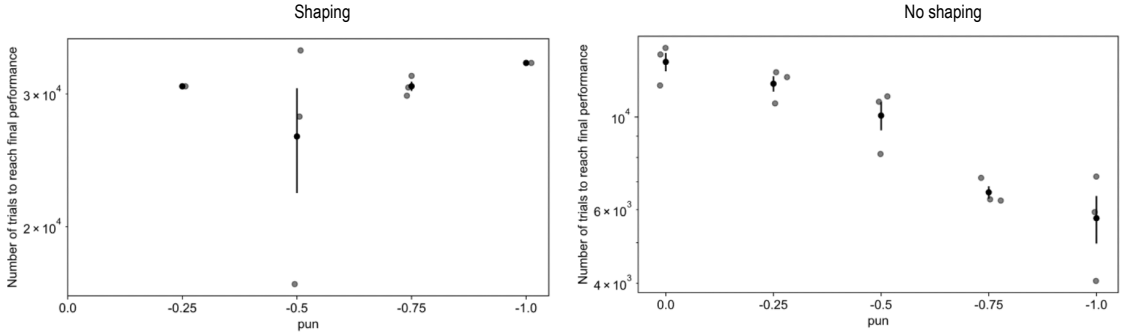

Figure 28: Punishment influence for achieving final performance in shaping and no shaping protocols. In the left figure (shaping), there is no visible pattern, whereas in the right (no shaping) the less punishment the faster the learning

For shaping, no clear results were found (Fig. 28, left). However, for no shaping, the more punishment, the faster networks arrive at the final performance (Fig. 28, right). Hence, punishment aids networks without shaping to learn. We can conclude that when using a not shaping protocol, the more punishment, the faster the learning.

7 Project Implementation Schedule

In the following section, I will describe in detail the tasks that were completed and its temporal organization in order to accomplish all the objectives of this project. I also performed a GANTT diagram shown in Table 2.

As previously said, the project duration goes from January to June 2021. However, some preliminary work was done before. Five Coursera Deep Learning courses were taken to get basic knowledge about the field. Moreover, some study of the previous work performed by Marta Fradera and Tiffany Oña was accomplished.

Some tasks such as writing the memory have been overlapped throughout the time of the fulfilment of other tasks, whereas others such as the training of the RNN requested the previous task creation to be finished before starting. Finally, the delivery date of the final degree project was the Monday 14th June of 2021 and the oral exposition the Tuesday 22nd June of 2021. The oral exposition was prepared the two previous weeks.

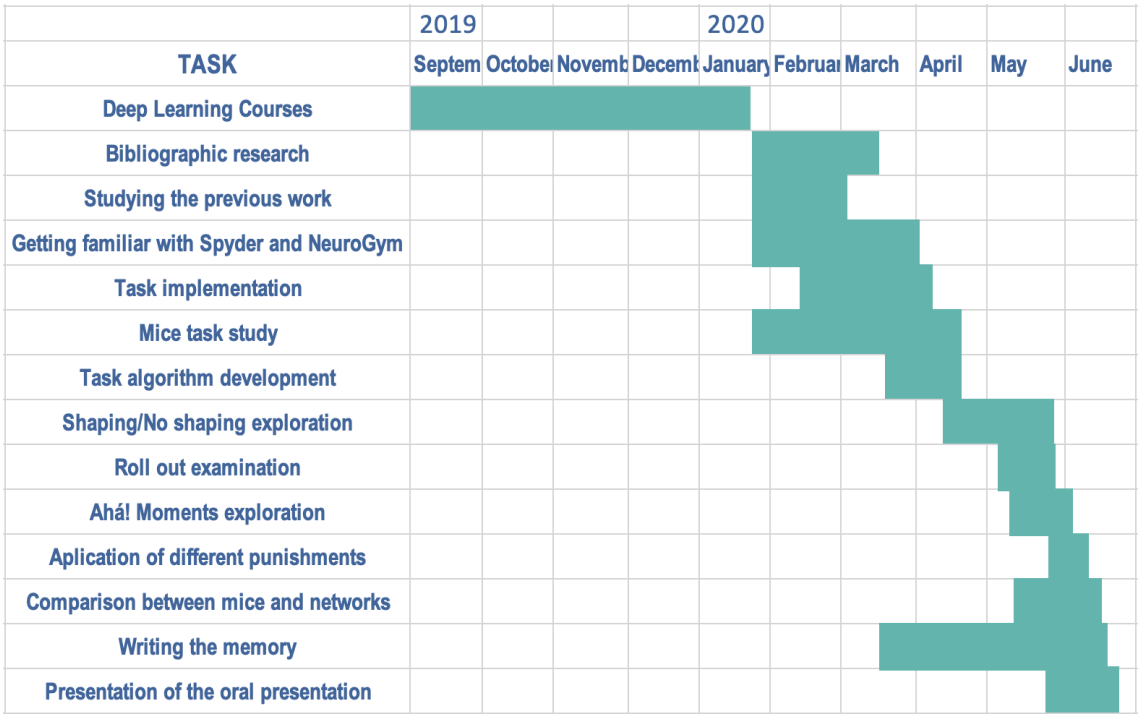

Table 2: Project implementation schedule. The accomplished tasks are organized in columns and the time dedicated to each one is represented by months

8 Technical Feasibility

In the following section, we performed the SWOT analysis. The internal factors (strengths and weaknesses) and the external factors (opportunities and threats) that have affected our study are going to be described.

8.1 Technical considerations

Infrastructure, software, specialist support, and equipment needed for the performance of the project are going to be discussed in this section.

To start with, it is important to consider that this project does not need any specific **infrastructure**. Therefore, due to the current situation of COVID-19 pandemic and the previously mentioned unnecessary presentiality of the project, we decided to accomplish it by remote working.

With respect to the **software**, all the needed applications, toolboxes and programs were freely available for users. In order to obtain free access to articles and papers, I used the SIRE button, which is a bookmarklet that allows the access of e-resources automatically through SIRE (E-Resources Access Service) at any point during browsing. The development of this study has been possible thanks to the NeuroGym toolbox [35], which is an open-source toolkit developed by the director of this project, Manuel Molano, together with Robert Yang, a postdoctoral researcher from the Center for Theoretical Neuroscience of Columbia University. Neurogym includes a series of popular neuroscience tasks designed to make neural network training easier. All training and analysis codes created during the execution of this project are available on GitHub [34, 35], which is a platform frequently used to host open-source projects and it is the biggest host of source code in the world [36].

Concerning **specialist support**, the evaluation of experts who have contributed in many ways has aided this research.

Regarding **equipment**, the used personal laptop is a 16-inch Apple MacBook Pro with the 2.6 GHz Intel Core i7 processor with two independent processor cores on a single chip (total 2 CPU) and 2 threads/core, 16 GB of RAM, and 512 GB storage. Though this laptop fits the necessary requirements to accomplish the writing of the project, the development of the scripts and the data analysis, the recommended specifications for the networks' training make this laptop not ideal for this task. Indeed, in the development and training of the task we faced some problems installing several toolboxes in my laptop. These problems were solved by running the same code at Google Collaboratory, which allows to run and program in Python in the browser without any kind of configuration and it gives free access to GPUs and allows easy content sharing. However, Google Collaboratory was not as efficient as we wanted and we decided to train networks from the lab's computer.

8.2 SWOT Analysis

This section consists in a SWOT analysis of the project, which includes internal (strengths, weaknesses) and external (opportunities, threats) aspects.

8.2.1 Strengths

- Opportunity to work in an interdisciplinary team of experts with similar experience in several projects
- Visit the biomedical research centre CELLEX to better understand and reproduce mice behaviour, as well as to be part of how different tasks are carried out in a laboratory
- Being able to train some part of the project via Google Collaborate from my personal laptop accelerated the work instead of using a computer at the laboratory or at the BSC
- Using Python as computer language, because its large and robust standard library makes it be preferred over other programming languages
- Possibility to perform work online, being able to work without problems due to COVID-19

8.2.2 Weaknesses

- Prior knowledge of deep learning and computational neuroscience was limited
- Incomplete knowledge of algorithms
- Only one shaping protocol performed in mice by Tiffany
- Small number of trained models to analyse
- Insufficient time to test hypotheses of the origin of aha moments
- Problems with installation of several packages of Python in the personal laptop

8.2.3 Opportunities

- Learn about deep learning techniques, RNNs and RL.
- Improve programming skills
- Improve teamwork
- Contribution to the Neurogym toolbox development
- Contribution to the general knowledge of data behaviour in the task

8.2.4 Threats

- Despite the growing impact of RNN modelling in neuroscience, the field is currently obstructed by the previous knowledge of deep learning platforms, such as TensorFlow or PyTorch, to train RNN models. This creates barriers for researchers to apply RNN modelling to their neuroscientific questions of interest.
- Existing RNNs lack basic biological features, limiting their ability to predict animal behaviour or neural circuit strategies.

9 Economical Feasibility

In this section the theoretical costs of the project are going to be described. Several costs such as licenses, software, hardware and staff are displayed. Then, the total cost needed to accomplish the project has been estimated from these expenses. In order to complete this section, a presentation of the financial resources has also been shown.

9.1 Expenses

There are no costs associated with fungible material or facilities for the construction of any system since this project primarily involves the execution of an algorithm. Potential costs, on the other hand, are derived from the necessary hardware for project execution, software licenses, licenses for bibliographic study, attended conferences, tutorials performed and proportional wages for technical workers.

The project's largest cost is related to professional personnel salaries, which are intended to cover engineering student and project supervisor fees. Junior biomedical engineers can expect to earn around 15€/hour, while experienced engineers can expect to earn around 20€/hour. The derived expense from engineering student rises to 5400€ due to the project's total length of 360 hours. The supervisor would be required to devote approximately 100 hours to the project, resulting in a total salary of 2000€. Nevertheless, since the student performs a non-remunerated work as part of the final degree project, this estimate is used to determine the project's expense.
Furthermore, the engineer must be adequately qualified prior to completing the project, which requires completion of several Coursera courses. These courses include an introductory machine learning course as well as a four-course deep learning curriculum. The cost of these courses is 43€ per course, for a total of 215 euros. On the other hand, Coursera courses have been performed using the auditing mode, which is a free option that provides access to all lectures and assignments but does not provide any grading or credential. Moreover, in order to achieve a deeper understanding of RNN before training them, it is pretty useful to attend some tutorials and lectures. For this reason, I decided to attend the Cosyne 2021 tutorial, which main topic was Recurrent Neural Networks for Neuroscience, because we thought it could be interesting and beneficial for the project. Therefore, the cost of this tutorial, which included one week plenty of lectures, programming exercises and speeches, was 20$.

Secondly, the hardware used during the development of this project was a computer, which needs some minimum requirements to fulfil deep learning tasks, such as an appropriate computer processor. Then, the estimated cost is 1.500€.

Thirdly, the software used was Python 4.1.5 in the Spyder environment, with the consequent usage of several Python toolboxes, such as TensorFlow, OpenAI Gym, and Stable Baselines. They are all freeware, which means they do not come with any additional costs. Moreover, AUTOCAD for the development of all the pictures and Excel for the Gant diagram.

On the other hand, in order to achieve a deep knowledge of the field, some bibliographic research was needed. Thus, I did some research on articles which required a paid subscription to access. However, this payment was avoided by employing the global subscription of the University of Barcelona.

Other costs include Internet connection, electrical power use and lighting. Even so, calculating the total amount of costs incurred is difficult.

To summarize, in Table 3 it is shown the cost of each element mentioned above, the total final cost was 8.916,79€. The expenses based on the type of cost are shown in Table 3, where it can be highlighted that the vast majority of the budget of this project was spent on human resources (83,18%). Then, hardware (16,82%) and others (0%) which include electrical power, Internet connection or lightning are negligible compared to the costs of human resources and hardware. Moreover, as all the software was free, it does not appear in Figure 29 either.

ITEM	COST
Biomedical Engineer student	5.400€
Student's training (Coursera + Cosyne)	0€ +16,79€
Project's supervisor	2000€
SOFTWARE	
Spyder	0€
TensorFlow	0€
OpenAI Gym	0€
Stable Baselines	0€
AutoCAD	0€
Excel	0€
HARDWARE	
Personal computer	1.500€
TOTAL COST	**8.916,79€**

Table 3: Expenses based on the type of cost

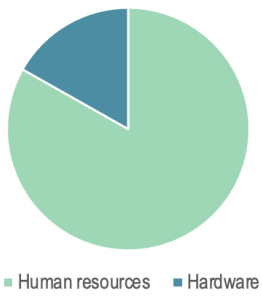

Figure 29: Cost per type of cost: human resources (83,18%), hardware (16,82%) and others (0%). Software is free.

9.2 Budget

The financing for this project comes from each of the participating institutions: The University of Barcelona, where the student is enrolled, and the IDIBAPS, where the project was carried out. As a result, the IDIBAPS Theoretical Neurobiology of Cortical Circuits research community has expensed the project's financial problems. The University of Barcelona has covered student job insurance due to an established academic partnership arrangement between the two institutions and the signed placement program.

Finally, we can conclude that due to the availability of free hardware and the fact that all the work done by the biomedical engineer was non remunerated, this project would be considered economically available.

10 Regulations and Legal Aspects

This project has been fully developed in Spain, hence the legal requirements that have been considered are based on the Spanish legislation.

First thing to consider is that the project is based on the previous study accomplished by Tiffany Oña, the PhD of the Cortical Circuit Dynamics group at IDIBAPS, where this final degree project also takes place. Considering the fact that the findings were obtained from mice rather than humans, presenting them to the current research does not require any knowledge consent.

Nevertheless, the mice studies were carried out in accordance with national and international ethical guidelines. The Animal Experimentation Ethics Committee (Comité d'Experimentació Animal – CEEA) of the University of Barcelona (UB) is in charge of the evaluation of experimentation and practices with experimental animals. It was created in 1998 to comply with Decree 214/1997 of 30 July of the Department of Agriculture, Livestock and Fisheries of the Generalitat de Catalunya, which regulates the use of animals for experimentation and other scientific purposes [38. Currently, it works under national rules regarding animal experiments for research contemplated in the RD 53/2013 [39] with modifications specified in the RD 1386/2018 [40].

Regarding software, as mentioned before, all programmes and applications employed were free, which means that they had open-source licenses, allowing all users to use, modify and share them. The most used program in the project has been Python. The Python Software Foundation License (PSFL) is a BSD-style, permissive free software license which is compatible with the GNU General Public License (GPL) [41]. The GNU is a set of commonly used free software licenses that give users the right to run, study, distribute, and change the software. Moreover, we also used Tensorflow [42], which is licensed under Apache License 2.0 [43], a free software license compatible with version 3 of the GNU General Public License. It is a permissive license whose main conditions require the preservation of copyright and license notices.
Furthermore, Github repositories were used everyday to share information due to the online conditions. Github is an Internet hosting company that specializes in software development and Git version control. It provides access control and several collaboration features such as bug tracking, feature requests, task management, continuous integration and wikis for every project. Furthermore, used GitHub repositories include Stable Baselines [44] and NGym [45], both under the MIT license [46].

In addition, the project's implemented DR task is part of the NeuroGym toolkit, which was created under an MIT license that allows commercial use, modification, distribution, and private use with the only condition of copyright and license notice preservation.

As a result, other researchers will have total access to the current work, inviting them to collaborate and allowing others to expand on it. For this reason, this work aims to contribute to scientific development by sharing all developed code as part of the open science platform.

11 Conclusions and future work

After analysing mice behaviour, we have seen that according to the shaping protocol used, the path followed by mice shows a smooth transition between different levels of complexity. However, **the speed at which mice learn to categorize the stimulus varies greatly among subjects**. In spite of this gradual learning, these small steps or little units of learning that are irreducible, called aha moments, must exist. Analysing them, we have found that **most mice (88,8%) undergo these experiences**, and the majority of them (60%) undergo several aha moments in a single training. Despite trying to understand the origin of the aha moments, the balance between right and left side trials, previous and during the aha moments, do not seem to be correlated to them.

Regarding RNNs trained with the same shaping protocol used with mice, **RNNs behavior seem to be qualitatively similar to that of mice but evolves at a much longer timescale**. Moreover, **mice behavior is much more unstable**, so when experiencing aha moments they present sharper performance changes than networks.

On the other hand, in order to examine the influence of shaping in networks, we have performed the same task without following a shaping protocol. As a result, we have found **that shaping only seems to be useful for some parameter configurations**. Therefore, when shaping is not used, the learning of the task is much slower when the networks need to process information about events happening very far in the past (rollout >> 1). Besides, **increasing the punishment when the RNNs make a mistake seems to help the learning of the full task**.

The methodology developed in the work, especially the aha moment identification, could be applied to other animal paradigms and be adapted to study how humans acquire specific knowledge. The shaping protocol that I have developed to train RNNs is an important step to create more natural protocols to help neural networks to smoothly learn complex tasks. In the future it will be interesting to study whether learning in **humans** follows dynamics similar to the ones I described for mice. One important difference between humans and mice is that the latter can acquire information verbally, which introduces a whole new level of complexity.

All in all, we can say that most of the initial study hypotheses have been validated. Even though, future work focused on the origin of the aha moments must be performed. Further research should concentrate on examining more elaborated patterns before and during the aha, exploring more parameters of the aha moment, and understanding why networks' performance is not perfect. Moreover, the focus of future research should also concentrate on adapting algorithm parameters, particularly in terms of regularization and exploration, in order to produce an artificial model that is more similar to biological brain circuits.

References

[1] Meyerholz, D.K., Beck, A.P. & Singh, B. Innovative use of animal models to advance scientific research. Cell Tissue Res 380, 205–206 (2020). https://doi.org/10.1007/s00441-020-03210-z

[2] Rubin, N., Nakayama, K., & Shapley, R. (1997). Abrupt learning and retinal size specificity in illusory-contour perception. Current Biology, 7(7), 461–467. https://doi.org/https://doi.org/10.1016/S0960-9822(06)00217-X

[3] Stuyck, H., Aben, B., Cleeremans, A., & Van den Bussche, E. (2021). The Aha! moment: Is insight a different form of problem solving? *Consciousness and Cognition*, *90*, 103055. https://doi.org/https://doi.org/10.1016/j.concog.2020.103055

[4] Shipstead, Z., Redick, T. S., & Engle, R. W. (2012). Is working memory training effective? Psychological Bulletin, 138(4), 628–654. https://doi.org/10.1037/a0027473

[5] B. Alan, "Working memory: looking back and looking forward," Nature Reviews Neuroscience, vol. 4, no. 10, p. 829, 2003, doi: 10.1038/nrn1201.

[6] Molano-Mazon, M., Duque, D., Yang, G. R., & de la Rocha, J. (2021). Pre-training RNNs on ecologically relevant tasks explains sub-optimal behavioral reset. *BioRxiv*, 2021.05.15.444287. https://doi.org/10.1101/2021.05.15.444287

[7] Cosyne 2021 tutorial. Topic: Recurrent Neural Networks for Neuroscience http://www.cosyne.org/c/index.php?title=Tutorial_2021

[8] Peterson GB. A day of great illumination: B. F. Skinner's discovery of shaping. *J Exp Anal Behav*. 2004;**82**:317–328.

[9] Lind, J., Ghirlanda, S., & Enquist, M. (2009). Insight learning or shaping? *Proceedings of the National Academy of Sciences of the United States of America*, *106*(28), E76–E77. https://doi.org/10.1073/pnas.0906120106

[10] Guo, Z. V, Hires, S. A., Li, N., O'Connor, D. H., Komiyama, T., Ophir, E., Huber, D., Bonardi, C., Morandell, K., Gutnisky, D., Peron, S., Xu, N., Cox, J., & Svoboda, K. (2014). Procedures for Behavioral Experiments in Head-Fixed Mice. *PLOS ONE*, *9*(2), e88678. https://doi.org/10.1371/journal.pone.0088678

[11] Chein, J. M., Weisberg, R. W., Streeter, N. L., & Kwok, S. (2010). Working memory and insight in the nine-dot problem. Memory & Cognition, 38, 883–892. doi:10.3758/MC.38.7.883

[12] Chronicle, E. P., MacGregor, J. N., & Ormerod, T. C. (2004). What makes an insight problem? The roles of heuristics, goal concep- tion, and solution recoding in knowledge-lean problems.

Journal of Experimental Psychology: Learning, Memory, and Cognition, 30, 14–27. doi:10.1037/0278-7393.30.1.14

[13]Weisberg, R. W., & Alba, J. W. (1981). An examination of the alleged role of "fixation" in the solution of several "insight" problems. Journal of Experimental Psychology: General, 110(2), 169–192. https://doi.org/10.1037/0096-3445.110.2.169

[14] Weisberg, R. (1986). A series of books in psychology.Creativity: Genius and other myths. W H Freeman/Times Books/ Henry Holt & Co.

[15] Bengio, Y., Louradour, J., Collobert, R., & Weston, J. (2009). Curriculum learning. In *Journal of the American Podiatry Association* (Vol. 60). https://doi.org/10.1145/1553374.1553380

[16] Hazy, T. E., Frank, M. J., & O'Reilly, R. C. (2007). Towards an executive without a homunculus: computational models of the prefrontal cortex/basal ganglia system. *Philosophical Transactions of the Royal Society B: Biological Sciences*, *362*(1485), 1601–1613. https://doi.org/10.1098/rstb.2007.2055

[17] O'Reilly, R. C., & Frank, M. J. (2006). Making Working Memory Work: A Computational Model of Learning in the Prefrontal Cortex and Basal Ganglia. *Neural Computation*, *18*(2), 283–328. https://doi.org/10.1162/089976606775093909

[18] Krueger, K. A., & Dayan, P. (2009). Flexible shaping: How learning in small steps helps. *Cognition*, *110*(3), 380–394. https://doi.org/https://doi.org/10.1016/j.cognition.2008.11.014

[19] Graves, A., Bellemare, M. G., Menick, J., Munos, R., & Kavukcuoglu, K. (2017). Automated Curriculum Learning for Neural Networks. In D. Precup & Y. W. Teh (Eds.), *Proceedings of the 34th International Conference on Machine Learning* (Vol. 70, pp. 1311–1320). PMLR. http://proceedings.mlr.press/v70/graves17a.html

[20] Song, H. F., Yang, G. R., & Wang, X.-J. (2016). Training Excitatory-Inhibitory Recurrent Neural Networks for Cognitive Tasks: A Simple and Flexible Framework. PLOS Computational Biology, 12(2), e1004792. https://doi.org/10.1371/journal.pcbi.1004792

[21] Belkaid, M., Bousseyrol, E., Durand-de Cuttoli, R., Dongelmans, M., Duranté, E. K., Ahmed Yahia, T., Didienne, S., Hanesse, B., Come, M., Mourot, A., Naudé, J., Sigaud, O., & Faure, P. (2020). Mice adaptively generate choice variability in a deterministic task. *Communications Biology*, *3*(1), 34. https://doi.org/10.1038/s42003-020-0759-x

[22] Ehrlich DB, Stone JT, Brandfonbrener D, Atanasov A, Murray JD. PsychRNN: An Accessible and Flexible Python Package for Training Recurrent Neural Network Models on Cognitive Tasks. eNeuro. 2021;8(1):ENEURO.0427-20.2020. Published 2021 Jan 15. doi:10.1523/ENEURO.0427-20.2020

[23] Sumant Ugalmugle, Rupali Swain: Animal Model Market size worth over $25 Bn by 2026. Published November 6, 2020

[24] Dymecki SM, Kim JC (2007) Molecular neuroanatomy's "Three Gs": a primer. Neuron 54: 17–34.

[25] Francis, N. A., & Kanold, P. O. (2017). Automated Operant Conditioning in the Mouse Home Cage. *Frontiers in Neural Circuits*, *11*, 10. https://doi.org/10.3389/fncir.2017.00010

[26] Animal Model Market Forecast, Trend Analysis & Competition Tracking - Global Market insights 2017 to 2026. Published Aug-2018

[27] Miriam A. Zemanova, More Training in Animal Ethics Needed for European Biologists, *BioScience*, Volume 67, Issue 3, March 2017, Pages 301–305, https://doi.org/10.1093/biosci/biw177

[28] Mice Model Market by Mice Type (Inbred, Knockout), Technology (CRISPR, TALEN, ZFN), Application (Oncology, Diabetes, Immunology), Service (Breeding, Cryopreservation, Genetic testing), Care Products (Cages, Bedding, Feed), Region - Global Forecast to 2025

[29] Bovenkerk, B., & Kaldewaij, F. (2015). The Use of Animal Models in Behavioural Neuroscience Research. In G. Lee, J. Illes, & F. Ohl (Eds.), Ethical Issues in Behavioral Neuroscience (pp. 17–46). Springer Berlin Heidelberg. https://doi.org/10.1007/7854_2014_329

[30] Gu, Quan, Lu, Na, and Liu, Lin. 'A Novel Recurrent Neural Network Algorithm with Long Short-term Memory Model for Futures Trading'. 1 Jan. 2019 : 4477 – 4484.

[31] De Mulder, W., Bethard, S., & Moens, M.-F. (2015). A survey on the application of recurrent neural networks to statistical language modeling. *Computer Speech & Language*, *30*(1), 61–98. https://doi.org/https://doi.org/10.1016/j.csl.2014.09.005

[32] Richard S Sutton and Andrew G Barto. Reinforcement learning: An introduction. MIT press, 2018.

[33] Tallec, C., & Ollivier, Y. (2017). *Unbiasing Truncated Backpropagation Through Time*. http://arxiv.org/abs/1705.08209

[34] Manuel Molano Leyre Azcárate. Cvlearning repository. https://github.com/manuelmolano/CV-Learning, 2021.

[35] Manuel Molano Leyre Azcárate. ngym_shaping repository. https://github.com/manuelmolano/ngym_shaping, 2021.

[36] Manuel Molano and Guangyu Robert Yang. Neurogym. https://github.com/gyyang/

neurogym, 2020.

[37] Animal Model Market Forecast, Trend Analysis & Competition Tracking - Global Market insights 2017 to 2026. Published Aug-2018

[38] Universitat de Barcelona. Comitè ètic d'experimentació animal. http://ub.edu/ceea/.

[39] Ministerio de la Presidencia. Real decreto 53/2013, de 1 de febrero, por el que se establecen las normas básicas aplicables para la protección de los animales utilizados en experimentación y otros fines científicos, incluyendo la docencia. Boletín Oficial del Estado, 34:11370–11421, 2013.

[40] Ministerio de la Presidencia. Real decreto 1386/2018, de 19 de noviembre, por el que se modifica el real decreto 53/2013, de 1 de febrero, por el que se establecen las normas básicas aplicables para la protección de los animales utilizados en experimentación y otros fines científicos, incluyendo la docencia. Boletín Oficial del Estado, 280:112804–112806, 2018.

[41] GNU Operating System. Gnu general public license. https://gnu.org/licenses/gpl-3.0.en.html.

[42] Martín Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Greg S. Corrado, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Ian Goodfellow, Andrew Harp, Geoffrey Irving, Michael Isard, Yangqing Jia, Rafal Jozefowicz, Lukasz Kaiser, Manjunath Kudlur, Josh Levenberg, Dandelion Mané, Rajat Monga, Sherry Moore, Derek Murray, Chris Olah, Mike Schuster, Jonathon Shlens, Benoit Steiner, Ilya Sutskever, Kunal Talwar, Paul Tucker, Vincent Vanhoucke, Vijay Vasudevan, Fernanda Viégas, Oriol Vinyals, Pete Warden, Martin Wattenberg, Martin Wicke, Yuan Yu, and Xiaoqiang Zheng. TensorFlow: Large-scale machine learning on heterogeneous systems, 2015. Software available from tensorflow.org.

[43] Apache. Apache license, version 2.0. https://www.apache.org/licenses/LICENSE-2.0.

[44] Ashley Hill, Antonin Raffin, Maximilian Ernestus, Adam Gleave, Anssi Kanervisto, Rene Traore, Prafulla Dhariwal, Christopher Hesse, Oleg Klimov, Alex Nichol, Matthias Plappert, Alec Radford, John Schulman, Szymon Sidor, and Yuhuai Wu. Stable baselines. https://github.com/hill-a/stable-baselines, 2018.

[45] Greg Brockman, Vicki Cheung, Ludwig Pettersson, Jonas Schneider, John Schulman, Jie Tang, and Wojciech Zaremba. Openai gym, 2016.

[46] GitHub. Mit license. https://choosealicense.com/licenses/mit/#.

Appendix:

Mice analysis:

Though the Python code developed is more extensive, the most representative parts are included below. However, the generated script can be found in the CV-Learning GitHub repository under the name of 'mice_behav_analysis.py'.

Manuel Molano Leyre Azcárate. Cvlearning repository. https://github.com/manuelmolano/CV-Learning, 2021.

```
### FUNCTIONS TO OBTAIN VARIABLES

def accuracy_trials_subj_stage4(df, subj, stg=None, conv_w=50):
    """
    Find accuracy values, number of sessions in each stage and color for each
    stage.

    Parameters
    ----------
    df : dataframe
        dataframe containing data.
    subj : str
        subject (each mouse).

    Returns
    -------
    For each mouse, it returns a list of the accuracies, a list of the
    sessions in each stage and a list with the colors of each stage.

    """
    # find accuracy values for each subject (hithistory takes the values  0/1)
    hit_raw = df.loc[df['subject_name'] == subj, 'wronglickhistory'].values
    # convolve it in order to get smooth values
    hit = np.convolve(hit_raw, np.ones((conv_w,))/conv_w, mode='valid')
    # find all the stages of the subject
    if stg is None:
        stg = df.loc[df['subject_name'] == subj, 'new_stage'].values

    # create the extremes (a 0 at the beggining and a 1 at the ending)
    stg_exp = np.insert(stg, 0, stg[0]-1)  # extended stages
    stg_exp = np.append(stg_exp, stg_exp[-1]+1)
    stg_diff = np.diff(stg_exp)  # change of stage
    stg_chng = np.where(stg_diff != 0)[0]  # index where stages change
    # We go over indexes where stage changes and plot chunks from ind_t-1
    # to ind_t
    hit_list = []  # list for accuracy
    xs_list = []  # list for the x axis
    stage_list = []  # list for the stages
    # iterate every stage to fill the lists
    for i_stg in range(1, len(stg_chng)):
        stage_list.append(stg_exp[stg_chng[i_stg-1]+1]-1)
```

```
        # HINT: xs will be larger than accs at i_stg == len(stg_cng). We need
        # this bc we will use xs in create_... fns to assign stages to trials.
        xs = range(stg_chng[i_stg-1], min(stg_chng[i_stg]+1, len(hit)+1))
        hits = hit[stg_chng[i_stg-1]:min(stg_chng[i_stg]+1, len(hit)+1)]
        hit_list.append(hits)
        xs_list.append(xs)
    return hit_list, xs_list, stage_list

def accuracy_at_stg_change_trials(df, subj_unq, prev_w=10, nxt_w=10,
conv_w=10):
    """
    The function returns the mean and standard deviation of the changes
    from a stage to another.

    Parameters
    ----------
    df : dataframe
        dataframe containing data.
    subj_unq : numpy.ndarray
        array of strings with the name of all the subjects
    prev_w: int
        previous window size (default value:10)
    nxt_w: int
        previous window size (default value:10)

    Returns
    -------
    Mean and standard deviation of each subject

    """
    mat_perfs = {}
    for i_s, sbj in enumerate(subj_unq):
        acc = df.loc[df['subject_name'] == sbj, 'hithistory'].values
        if conv_w > 0:
            accur_conv = np.convolve(acc, np.ones((conv_w,))/conv_w,
                                     mode='same')
        else:
            accur_conv = acc
        stg = df.loc[df['subject_name'] == sbj, 'new_stage'].values
        # create the extremes (a 0 at the beggining and a 1 at the ending)
        stg_diff = np.diff(stg)  # change of stage
        stg_chng = np.where(stg_diff != 0)[0]  # index where stages change
        # We go over indexes where stage changes and plot chunks from ind_t-1
        # to ind_t
        for i_stg in range(len(stg_chng)):
            # color = stg_exp[stg_chng[i_stg-1]+1]-1
            stg_prev = stg[stg_chng[i_stg]]  # get stage before the change
            stg_nxt = stg[stg_chng[i_stg]+1]  # get stage after the change
            assert stg_prev != stg_nxt, 'stages are supposed to be different'
            key = str(stg_prev)+'-'+str(stg_nxt)  # e.g. 1-2
            if key not in mat_perfs.keys():
                mat_perfs[key] = []
            # build chunk
            i_previo = max(0, stg_chng[i_stg]-prev_w)
            i_next = stg_chng[i_stg]+nxt_w
            chunk = -min(0, stg_chng[i_stg]-prev_w)*[np.nan] +\
                accur_conv[i_previo:i_next].tolist() +\
                max(0, i_next-len(acc))*[np.nan]
            # add chunk to the dictionary
```

```
            mat_perfs[key].append(chunk)
    # dictionary of the mean of the performance
    mat_mean_perfs = {}
    # dictionary of the standard deviation of the performace
    mat_std_perfs = {}
    # list of the number of samples of each change of stage
    number_samples = []
    for key in mat_perfs.keys():
        number_samples.append(len(mat_perfs[key]))  # save number of samples
        assert np.std([len(p) for p in mat_perfs[key]]) == 0
        mat_mean_perfs[key] = np.nanmean(np.array(mat_perfs[key]), axis=0)
        sqrt_n_smpls = np.sqrt(np.array(mat_perfs[key]).shape[0])
        mat_std_perfs[key] =\
            np.nanstd(np.array(mat_perfs[key]), axis=0)/sqrt_n_smpls
    return mat_mean_perfs, mat_std_perfs, number_samples


def create_motor_column(df, df_prms, subject):
    """
    Creation of an extra column for the motor variable in df_trials from the
    motor data found in df_params

    Parameters
    ----------
    df : dataframe
        data of trials
    df : dataframe
        data of sessions
    subject: str
        subject chosen

    Returns
    -------
    Dataframe with an extra column

    """
    # obtain values of the motor values in df_trials
    motor = df_prms.loc[df_prms['subject_name'] == subject, 'motor'].values
    # obtain values of the indexes and motor stages of the df_params
    _, indx_mstg, mot_stg = accuracy_sessions_subj(df=df_prms, subj=subject,
                                                   stg=motor)
    df_ss = df.loc[df.subject_name == subject, ['session']]
    sess_unq = np.unique(df_ss)
    trial_sess = np.zeros_like(df_ss).flatten()
    for ss in sess_unq:
        aux = [i_x for i_x, x in enumerate(indx_mstg) if ss in x][0]
        indx = np.where(df_ss == ss)[0]  # find where the sessions of both
        # datasets are the same
        trial_sess[indx] = mot_stg[aux]+1  # sessions are index+1
    df_trials_subject = df.loc[df.subject_name == subject]
    df_trials_subject['motor_stage'] = trial_sess
    return df_trials_subject

def find_events(df_tr, subj, event):
    """
    The function returns the day in which the subject had an event

    Parameters
    ----------
```

```
    df_tr : dataframe
        dataframe containing data.
    subj : string
        subject
    event: string
        mice event

    Returns
    -------
    Index in which this event happens

    """
    # take only 8 first digits of date, discarding exact time
    # alternatively, find index of events in dates
    index = np.where(df_tr.subject_name == subj)[0]
    dates = [x[:8] for x in df_tr['date'][index].values]
    index = np.where(np.array(dates) == md.events_dict[subj][event])[0]
    if len(index) > 0:
        index = index[0]
    else:
        index = -1
    return index

def learned_categories(sbj, df, index_event=None, color_ev='', verbose=True,
                       figsize=(8, 4), ax=None, plt_sess=True, stage=1):
    """
    The function plots accuracy over trials for every subject, showing
    the stages the mice are in different colors.
    Parameters
    ----------
    sbj : string
        Subject (each mouse)
    df : dataframe
        data
    color : list
        list of colors corresponding to the stage.
    Returns
    -------
    The plot of accuracy over trial for every subject.
    """
    def get_trial_info(df, subj):
        """
        Get relevant data trial by trial.

        Parameters
        ----------
        df : dataframe
            dataframe containing data.
        subj : str
            subject (each mouse).

        Returns
        -------
        For each mouse, it returns a list of the accuracies, the ground-truth
sides
        the sessions in each stage and the colors of each stage.

        """
```

```
        # find accuracy values for each subject (hithistory takes the values
0/1)
        hit = df.loc[df['subject_name'] == subj, 'wronglickhistory'].values
        # find all the stages of the subject
        stg = df.loc[df['subject_name'] == subj, 'new_stage'].values
        # find all the stages of the subject
        gt = df.loc[df['subject_name'] == subj, 'reward_side'].values
        # create the extremes (a 0 at the beggining and a 1 at the ending)
        stg_exp = np.insert(stg, 0, stg[0]-1)  # extended stages
        stg_exp = np.append(stg_exp, stg_exp[-1]+1)
        stg_diff = np.diff(stg_exp)  # change of stage
        stg_chng = np.where(stg_diff != 0)[0]  # index where stages change
        # We go over indexes where stage changes and plot chunks from ind_t-1
        # to ind_t
        hit_list = []  # list for accuracy
        xs_list = []  # list for the x axis
        stage_list = []  # list for the stages
        gt_list = []  # list for the stages
        # iterate every stage to fill the lists
        for i_stg in range(1, len(stg_chng)):
            stage_list.append(stg_exp[stg_chng[i_stg-1]+1]-1)
            # HINT: xs will be larger than accs at i_stg == len(stg_cng). We
need
            # this bc we will use xs in create_... fns to assign stages to
trials.
            xs = range(stg_chng[i_stg-1], min(stg_chng[i_stg]+1, len(hit)+1))
            hits = hit[stg_chng[i_stg-1]:min(stg_chng[i_stg]+1, len(hit)+1)]
            gts = gt[stg_chng[i_stg-1]:min(stg_chng[i_stg]+1, len(hit)+1)]
            hit_list.append(hits)
            xs_list.append(xs)
            gt_list.append(gts)
        return hit_list, xs_list, stage_list, gt_list

    hit_sbj, xs_sbj, color_sbj, gt_sbj = get_trial_info(df, subj=sbj)
    hit_sbj = hit_sbj[color_sbj == stage-1]
    # LEARNING TIME
    learn_data = {'learned': [], 'ev_not_l': [], 'ev_l': []}
    learn_data = arl.learned(perf=hit_sbj, learn_data=learn_data,
                             verbose=verbose)
    if verbose:
        name = sbj
        for k in learn_data:
            name += k+': '+str(learn_data[k])
        plt.title(name)
        f = plt.gcf()
        sv_fig(f=f, name=sbj+'_learned')

    # AHA-MOMENTS
    gt_sbj = gt_sbj[color_sbj == stage-1]
    stg_mat = np.ones_like(hit_sbj)
    aha_data = {'aha_mmts': [], 'prev_prfs': [], 'post_prfs': [],
                'gt_patterns': [], 'perf_patterns': [], 'prob_right': [],
                'prob_right_aha': []}
    aha_data = arl.get_ahas(stage=stg_mat, perf=hit_sbj, gt=gt_sbj,
                            aha_data=aha_data, verbose=verbose)
    if verbose:
        plt.title(sbj+'  number of ahas: '+str(len(aha_data['aha_mmts'])))
        f = plt.gcf()
        sv_fig(f=f, name=sbj+'_aha_mmnts')
```

```
    return learn_data, aha_data


### FUNCTIONS TO PLOT

def plot_accuracy_trials_coloured_stage4(sbj, df, index_event=None,
color_ev='',
                                         figsize=(8, 4), ax=None,
plt_sess=True):
    """
    The function plots accuracy over trials for every subject, showing
    the stages the mice are in different colors.
    Parameters
    ----------
    sbj : string
        Subject (each mouse)
    df : dataframe
        data
    color : list
        list of colors corresponding to the stage.
    Returns
    -------
    The plot of accuracy over trial for every subject.
    """
    save_fig = False
    if ax is None:
        f, ax = plt.subplots(figsize=figsize)
        save_fig = True
    hit_sbj, xs_sbj, color_sbj = accuracy_trials_subj_stage4(df, subj=sbj)
    for i_chnk, chnk in enumerate(hit_sbj):
        # iterate for every chunk, to paint the stages with different colors
        # HINT: see accuracy_sessions_... fn for an explanation fo why xs can
        # be larger than acc
        ax.plot(xs_sbj[i_chnk][:len(hit_sbj[i_chnk])], hit_sbj[i_chnk],
                color=COLORS[color_sbj[i_chnk]])
    ax.set_title("Accuracy by trials of subject taking into" +
                 " account misses (" + sbj+")")
    ax.set_xlabel('Trials')
    ax.set_ylabel('Accuracy')
    ax.legend(['Stage 1', 'Stage 2', 'Stage 3', 'Stage 4'],
              loc="center right",    # Position of legend
              borderaxespad=0.1,  # Small spacing around legend box
              title='Color legend')
    if save_fig:
        sv_fig(f=f, name='acc_acr_tr_subj_'+sbj)

def plot_final_acc_session_subj_stage4(subj_unq, df_trials, figsize=(8, 4),
                                       conv_w=200):
    """
    The function plots accuracy over session for all the subjects.

    Parameters
    ----------
    subj_unq : numpy.ndarray
        array of strings with the name of all the subjects

    Returns
    -------
    Plot of accuracy by session for every subject.
```

```
    """
    fig, ax = plt.subplots(nrows=3, ncols=6, figsize=figsize,
                           gridspec_kw={'wspace': 0.5, 'hspace': 0.5})
    # leave some space between two figures, wspace is the horizontal gap and
    # hspace is the vertical gap
    ax = ax.flatten()
    # plot a subplot for each subject
    for i_s, sbj in enumerate(subj_unq):
        hit_sbj, xs_sbj, color_sbj = accuracy_trials_subj_stage4(df=df_trials,
                                                                 subj=sbj,

conv_w=conv_w)
        plot_accuracy_trials_subj_stage4(hit=hit_sbj, xs=xs_sbj,
col=color_sbj,
                                         ax=ax[i_s], subj=sbj)
    fig.suptitle("Accuracy VS trials", fontsize="x-large")
    lines = [obj for obj in ax[0].properties()['children']  # all objs in
ax[0]
             if isinstance(obj, matplotlib.lines.Line2D)  # that are lines
             and obj.get_linestyle() != '--']  # that are not dashed
    fig.legend(lines, ['Stage 1', 'Stage 2', 'Stage 3',
                       'Stage 4'],
               loc="center right",   # Position of legend
               borderaxespad=0.1,  # Small spacing around legend box
               title='Color legend')
    sv_fig(fig, 'Accuracy VS trials with 3.1 stage')


def plot_means_std(means, std, list_samples, prev_w=10, nxt_w=10,
                   figsize=(6, 4)):
    """
    Plot mean and standard deviation from the accuracies of each state of each
    mouse.

    Parameters
    ----------
    means : dict
        dictionary containing all the stage changes (e.g. '1-2', '2-3'...) and
        the accuracies associated to each change.
    std : dict
        dictionary containing all the stage changes (e.g. '1-2', '2-3'...) and
        the standard deviations associated to each change.
    prev_w: int
        previous window size (default value:10)
    nxt_w: int
        previous window size (default value:10)

    Returns
    -------
    Plot of the mean and standard deviation of each stage for all the subjects

    """
    if len(means) == 5:
        fig, ax = plt.subplots(nrows=2, ncols=3, figsize=figsize,
                               gridspec_kw={'wspace': 0.5, 'hspace': 0.5})
    elif len(means) == 8:
        fig, ax = plt.subplots(nrows=2, ncols=4, figsize=figsize,
                               gridspec_kw={'wspace': 0.5, 'hspace': 0.5})
```

```
    ax = ax.flatten()
    fig.suptitle('Mean Accuracy of changes', fontsize='x-large')
    xs = np.arange(-prev_w, nxt_w)
    for i_k, (key, val) in enumerate(means.items()):
        ax[i_k].errorbar(xs, val, std[key], label=key)
        ax[i_k].set_ylim(0.5, 1)
        ax[i_k].set_title(key + ' (N='+str(list_samples[i_k])+')')
        ax[i_k].axvline(0, color='black', linestyle='--')
        # Hide the right and top spines
        ax[i_k].spines['right'].set_visible(False)
        ax[i_k].spines['top'].set_visible(False)
        # Only show ticks on the left and bottom spines
        ax[i_k].yaxis.set_ticks_position('left')
        ax[i_k].xaxis.set_ticks_position('bottom')
        if len(means) == 5:
            if i_k in [0, 3]:
                ax[i_k].set_ylabel('Mean accuracy')
            if i_k in [3, 4]:
                ax[i_k].set_xlabel('Sessions after stage change')
        elif len(means) == 8:
            if i_k in [0, 4]:
                ax[i_k].set_ylabel('Mean accuracy')
            if i_k in [4, 5, 6, 7]:
                ax[i_k].set_xlabel('Trials after stage change')
    if len(means) == 5:
        sv_fig(fig, 'Mean Accuracy of changes for 3 stages')
    elif len(means) == 8:
        sv_fig(fig, 'Mean Accuracy of changes for 4 stages')


def plot_trials_subj(df, subject, df_sbj_perf, ax=None, conv_w=200,
                     figsize=None):
    """
    Plots for each subject all hithistory variables (true/false),
    which describe the success of the trial.

    Parameters
    ----------
    df : dataframe
        data
    subject: str
        subject chosen
    df_sbj_perf: dataframe
        performance of each subject
    ax: None
        axes
    conv_w: int
        window used to smooth (convolving) the accuracy (default 200)

    Returns
    -------
    Plots a figure of the performance of the subject along trials

    """
    if ax is None:
        f, ax = plt.subplots(figsize=figsize)
    # plot the convolution of the performance of each subject
    ax.plot(np.convolve(df_sbj_perf, np.ones((conv_w,))/conv_w, mode='valid'))
    ax.set_title("Accuracy by trials of subject" + subject)
```

```
    ax.set_xlabel('Trials')
    ax.set_ylabel('Accuracy (Hit: True or False)')
    session = df.loc[df['subject_name'] == subject, 'session'].values
    # create the extremes (a 0 at the beggining and a 1 at the ending)
    ses_diff = np.diff(session)  # find change of stage
    ses_chng = np.where(ses_diff != 0)[0]  # find where is the previous change
    # plot a vertical line for every change os session
    for i in ses_chng:
        ax.axvline(i, color='black')
```

Networks analysis:

Though the Python code developed is more extensive, the most representative parts are included below. However, the generated scripts can be found in the CV-Learning GitHub repository under the names of 'analysis_rl.py' and 'example_neurogym_rl.py'.

Manuel Molano Leyre Azcárate. Cvlearning repository. https://github.com/manuelmolano/CV-Learning, 2021.

```
def data_extraction(folder, metrics, w_conv_perf=500, conv=[1, 0]):
    """ Extract data saved during training.
    metrics: dict containing the keys of the data to load extracted.
    conv: list of the indexes of the metrics to convolve."""
    # load all data from the same folder
    data = put_together_files(folder)
    data_flag = True
    if data:
        # extract each of the metrics
        for ind_k, k in enumerate(metrics.keys()):
            if k in data.keys():
                metric = data[k]
                if conv[ind_k]:
                    mean = np.convolve(metric,
                                       np.ones((w_conv_perf,))/w_conv_perf,
                                       mode='valid')
                else:
                    mean = metric
            else:
                mean = []
            metrics[k].append(mean)
    else:
        print('No data in: ', folder)
        data_flag = False
    return metrics, data_flag


def learned(perf, learn_data, verbose=True, **params):
    """Do a smoothing (np.convolve) with a very long window.
    Make a histogram with the values of the resulting factor to see if
    You get 2 "mountains": chance and learned performance.
    Establish a threshold from that histogram.
    Find all values below / above thresholds
    Measure the minimum distance between the periods.
```

```
    """
    def get_event(trace, frst_lst):
        dwn_idx = np.where(np.diff(trace) < 0)[0]
        up_idx = np.where(np.diff(trace) > 0)[0]
        ev = None
        if frst_lst == 'first' and len(dwn_idx) > 0:
            ev = dwn_idx[0] if (dwn_idx[0] < up_idx).all() else None
        elif frst_lst == 'last' and len(up_idx) > 0:
            ev = up_idx[-1] if (up_idx[-1] > dwn_idx).all() else None
        return ev
    learn_dic_def = {'w_perf': 500, 'perf_bef_aft': [.6, .75]}
    learn_dic_def.update(params)
    w_perf = learn_dic_def['w_perf']
    perf_bef_aft = learn_dic_def['perf_bef_aft']
    perf_conv = np.convolve(perf, np.ones((w_perf,))/w_perf, mode='valid')

    not_learned = 1*(perf_conv < perf_bef_aft[0])
    ev_not_l = get_event(trace=not_learned, frst_lst='first')
    learned = 1*(perf_conv > perf_bef_aft[1])
    ev_l = get_event(trace=learned, frst_lst='last')
    if verbose:
        f, ax = plt.subplots(1, 1, figsize=(5,4))
        ax.plot(perf_conv, label='Performance')
        ax.plot([ev_not_l, ev_not_l], [0, 1], 'c', label='Start of learning
period')
        ax.plot([ev_l, ev_l], [0, 1], 'm', label='End of learning period')
        ax.spines['right'].set_visible(False)
        ax.spines['top'].set_visible(False)
        ax.axhline(y=perf_bef_aft[0], color='c', linestyle='--')
        ax.axhline(y=perf_bef_aft[1], color='m', linestyle='--')
        ax.set_ylim(0.4,1)
        ax.set_xlabel('Trials')
        ax.set_ylabel('Mean performance')
        ax.legend(loc='upper left')
    learned = False if (ev_l is None or ev_not_l is None or ev_l <= ev_not_l)\
        else True
    learn_data['learned'].append(learned)
    learn_data['ev_not_l'].append(ev_not_l)
    learn_data['ev_l'].append(ev_l)
    return learn_data

def learning(folder, learn_data={}, verbose=True, conv=[1], **aha_dic):
    """ Extract data saved during training. metrics: dict containing
    the keys of the data to loaextractd.
    conv: list of the indexes of the metrics to convolve."""
    data = put_together_files(folder)  # load all data from the same folder
    data_flag = True
    if data:
        # extract each of the metrics
        if 'real_performance' in data.keys():
            perf = data['real_performance']
            stage = data['stage']
            perf = perf[stage == 1]
            learn_data = learned(perf=perf, learn_data=learn_data,
                                 **aha_dic)
    else:
        if verbose:
            print('No data in: ', folder)
        data_flag = False
```

```
    return learn_data, data_flag


def get_ahas(stage, perf, gt, aha_data, verbose=True, **aha_dic):
    """Find aha moments when all the requirements are fulfilled and
    plot them"""
    ahas_dic_def = {'w_ahas': 10, 'w_perf': 100,
                    'bef_aft_diff': 0.2, 'aha_th': 0.75, 'w_explore': 10}
    ahas_dic_def.update(aha_dic)
    prob_right = 0
    w_ahas = ahas_dic_def['w_ahas']
    w_perf = ahas_dic_def['w_perf']
    perf_th = ahas_dic_def['aha_th']
    bef_aft_diff = ahas_dic_def['bef_aft_diff']
    w_explore = ahas_dic_def['w_explore']
    no_shaping = len(np.unique(stage)) == 1 and 4 in stage
    if 1 in stage or no_shaping:
        indx = stage == 4 if no_shaping else stage == 1
        perf_stg_1 = perf[indx]
        gt = gt[indx]
        ahas = np.convolve(perf_stg_1, np.ones((w_ahas,))/w_ahas,
                           mode='valid')
        perf = np.convolve(perf_stg_1, np.ones((w_perf,))/w_perf,
                           mode='valid')
        if verbose:
            plt.figure(figsize=(4,3))
            # plt.title(folder)
            # plt.plot(perf_stg_1, '-+')
            plt.plot(ahas, '-+', label='Performance window = 10')
            plt.plot(perf, label='Performance window = 100')
            plt.legend()
            plt.xlabel('Trials')
            plt.ylabel('Mean performance')
            # plt.plot(np.convolve(perf_stg_1, np.ones((500,))/500,
            #                      mode='valid'))
        aha_indx = np.where(ahas > perf_th)[0]
        min_num_trs = 100
        aha_indx = aha_indx[aha_indx < len(perf_stg_1)-min_num_trs]
        aha_indx = aha_indx[aha_indx > min_num_trs]
        if len(aha_indx) > 0:
            prev_ai = -10e6
            for a_i in aha_indx:
                prev_perf = np.mean(perf_stg_1[a_i-w_perf:a_i])
                post_perf = np.mean(perf_stg_1[a_i+w_ahas:
                                               a_i+w_ahas+w_perf])
                aha_data['prev_prfs'].append(prev_perf)
                aha_data['post_prfs'].append(post_perf)
                # if verbose:
                #     plt.plot([a_i, a_i], [0, 1], '--m', lw=0.5)
                if prev_perf <= post_perf - bef_aft_diff and a_i >
prev_ai+w_perf:
                    prev_ai = a_i
                    aha_data['aha_mmts'].append(a_i)

                    if verbose:
                        plt.plot([a_i, a_i], [0, 1], '--k', lw=2)
                        print('AHA MOMENT')
                        print(gt[a_i-w_perf:a_i+w_ahas+w_perf])
                        print('**')
```

```
                    aha_data['gt_patterns'].append(gt[a_i-w_perf:
                                                      a_i+w_ahas+w_perf])
                    aha_data['perf_patterns'].append(perf_stg_1[a_i-w_perf:
a_i+w_ahas+w_perf])
                    # find probabilities of right before the aha window
                    right_number = np.sum(gt[a_i-w_explore:a_i] == 1)
                    prob_right = right_number/w_explore
                    aha_data['prob_right'].append(prob_right)
                    # find probabilities of right during the aha window
                    right_number = np.sum(gt[a_i:a_i+w_ahas] == 1)
                    prob_right = right_number/w_ahas
                    aha_data['prob_right_aha'].append(prob_right)

    return aha_data


def aha_moment(folder, aha_data={}, verbose=True, conv=[1], **aha_dic):
    """ Extract data saved during training. metrics: dict containing
    the keys of the data to loaextractd.
    conv: list of the indexes of the metrics to convolve."""
    data = put_together_files(folder)  # load all data from the same folder
    data_flag = True
    if data:
        # extract each of the metrics
        if 'real_performance' in data.keys():
            perf = data['real_performance']
            stage = data['stage']
            gt = data['gt']
            aha_data     =     get_ahas(stage=stage,     perf=perf,     gt=gt,
aha_data=aha_data,
                                **aha_dic)
    else:
        if verbose:
            print('No data in: ', folder)
        data_flag = False
    return aha_data, data_flag


def get_tag(tag, file):
    """Process name"""
    f_name = ntpath.basename(file)
    assert f_name.find(tag) != -1, 'Tag '+tag+' not found in '+f_name
    val = f_name[f_name.find(tag)+len(tag)+1:]
    val = val[:val.find('_')] if '_' in val else val
    return val

def perf_hist(metric, ax, index, trials_day=300):
    """Plot a normalized histogram of the number of days/sessions spent with
    the same metric vale (e.g. performance).
    trials_day: number of trials to include on a session/day."""
    metric = np.array(metric)
    index = np.array(index)
    unq_vals = np.unique(index)
    bins = np.linspace(0, 1, 20)
    for ind_val, val in enumerate(unq_vals):
        indx = index == val
        traces_temp = metric[indx]
        traces_temp = list(itertools.chain.from_iterable(traces_temp))
```

```
        hist_, plt_bins = np.histogram(traces_temp, bins=bins)
        hist_ = hist_/np.sum(hist_)
        plt_bins = plt_bins[:-1] + (plt_bins[1]-plt_bins[0])/2
        ax.plot(plt_bins, hist_, label=val, color=CLRS[ind_val])
    ax.legend()
    ax.set_xlabel('Performance')
    ax.set_ylabel('Days')


def plot_rew_across_training(metric, index, ax, n_traces=20,
                             selected_protocols=['-1.0', '-0.75', '-0.5',
                                                 '-0.25', '0.0']):
    """Plot traces across training, i.e. metric value per trial.
    """
    metric = np.array(metric)
    index = np.array(index)
    unq_vals = np.unique(index)
    for ind_val, val in enumerate(unq_vals):
        if val in selected_protocols:
            indx = index == val
            traces_temp = metric[indx][:n_traces]
            for trace in traces_temp:
                ax.plot(trace, color=CLRS[ind_val], alpha=0.5, lw=0.5)


def plt_means(metric, index, ax, limit_mean=True, limit_ax=True,
              selected_protocols=['-1.0', '-0.75', '-0.5', '-0.25', '0.0']):
    """Plot mean traces across training.
    """
    if limit_mean:
        min_dur = np.min([len(x) for x in metric])
        metric = [x[:min_dur] for x in metric]
    else:
        max_dur = np.max([len(x) for x in metric])
        metric = [np.concatenate((np.array(x),
                                  np.nan*np.ones((int(max_dur-len(x)),))))
                  for x in metric]

    metric = np.array(metric)
    index = np.array(index)
    unq_vals = np.unique(index)
    for ind_val, val in enumerate(unq_vals):
        if val in selected_protocols:
            indx = index == val
            traces_temp = metric[indx, :]
            if not (np.isnan(traces_temp)).all():
                ax.plot(np.nanmean(traces_temp, axis=0), color=CLRS[ind_val],
                        lw=1, label=val+' ('+str(np.sum(indx))+')')
    if limit_ax:
        assert limit_mean, 'limiting ax only works when mean is also limited'
        ax.set_xlim([0, min_dur])


def plt_perf_indicators(values, index_val, ax, f_props, ax_props, reached=None,
                        discard=[], plot_individual_values=True,
                        errorbars=True):
    """Plot final results, in this case, performance indicators
    """
    values = np.array(values)
```

```
    index_val = np.array(index_val)
    unq_vals = np.unique(index_val)
    if plot_individual_values:
        std_noise = get_noise(unq_vals)
    for ind_val, val in enumerate(unq_vals):
        # only for those thresholds different than full task
        if val not in discard:
            # only those traces with same value that have reached last phase
            if reached is not None:
                indx = np.logical_and(index_val == val, reached)
            else:
                indx = index_val == val
            values_temp = values[indx]
            n_vals = len(values_temp)
            if n_vals != 0:
                # plot number of trials
                f_props['markersize'] = 10
                if errorbars:
                    ax.errorbar([ALL_INDX[val]], np.nanmean(values_temp),
                                (np.nanstd(values_temp)/np.sqrt(n_vals)),
                                **f_props)
                else:
                    ax.plot(ALL_INDX[val], np.nanmean(values_temp), **f_props)
            if plot_individual_values:
                xs = np.random.normal(0, std_noise, ((np.sum(indx),))) +\
                    ALL_INDX[val]
                ax.plot(xs, values_temp, alpha=0.5, linestyle='None',
                        **f_props)
    ax.set_xlabel(ax_props['tag'])
    ax.set_ylabel(ax_props['ylabel'])
    ax.set_xticks(ax_props['ticks'])
    ax.set_xticklabels(ax_props['labels'])


def trials_per_stage(metric, ax, index):
    """Plot the mean number of trials spent on each of the stages."""
    bins = np.linspace(STAGES[0]-0.5, STAGES[-1]+.5, len(STAGES)+1)
    metric = np.array(metric)
    index = np.array(index)
    unq_vals = np.unique(index)
    # find all the trials spent on each stage
    for ind_val, val in enumerate(unq_vals):
        indx = index == val
        traces_temp = metric[indx]
        counts_mat = []
        n_traces = len(traces_temp)
        for ind_tr in range(n_traces):
            # plot the individual values
            counts = np.histogram(traces_temp[ind_tr], bins=bins)[0]
            indx = counts != 0
            noise = np.random.normal(0, 0.01, np.sum(indx))
            ax.plot(np.array(STAGES)[indx]+noise, counts[indx], '+',
                    color=CLRS[ind_val], alpha=0.5)
            counts_mat.append(counts)
        counts_mat = np.array(counts_mat)
        mean_counts = np.mean(counts_mat, axis=0)
        # (e.g. in protocol 0234, don't plot stage 1)
        # ax.errorbar(np.array(STAGES), mean_counts, std_counts, marker='+',
        #             color=CLRS[ind_val], label=val)
```

```
            # std_counts = np.std(counts_mat, axis=0)/np.sqrt(n_traces)
            indx = mean_counts != 0
            # plot the mean values
            ax.plot(np.array(STAGES)[indx], mean_counts[indx], marker='+',
                    linestyle='--', color=CLRS[ind_val], label=val)

    handles, labels = ax.get_legend_handles_labels()
    by_label = dict(zip(labels, handles))
    plt.yscale('log')
    ax.legend(by_label.values(), by_label.keys())
    ax.set_xlabel('Stage')
    ax.set_ylabel('Trials')

def plot_results(folder, setup='', setup_nm='', w_conv_perf=500, perf_th=0.6,
                 keys=['real_performance', 'stage'], limit_ax=True, final_ph=4,
                 ax_final=None, tag='th_stage', limit_tr=False, rerun=False,
                 f_final_prop={'color': (0, 0, 0), 'label': '', 'marker': '.'},
                 plt_ind_vals=True, plt_ind_traces=True, n_roll=5, name='',
                 x=0, ahas_dic={}, learn_dic={}):
    """This function uses the data generated during training to analyze it
    and generate figures showing the results in function of the different
    values used for the third level variable (i.e. differen threshold values
    or different shaping protocols).
    folder: folder where we store/load the data.
    algorithm: used algorithm for training.
    setup: value indicating the second level variable value, i.e. the used
    number of window or the used n_ch (number of channels) for training.
    setup_nm: indicates which second level exploration has been done
    (window/n_ch).
    w_conv_perf: dimension of the convolution window.
    keys: list of the names of the metrics to explore.
    limit_ax: limit axis when plottingg.
    final_ph: stage number that corresponds to the last stage of training.
    perf_th: threshold performance to separate the traces that have or have not
    learnt the task.
    ax_final: axes for plotting the final results.
    tag: name of the performed exploration ('th_stage' for different threshold
    values, and 'stages' for different shaping protocols').
    limit_tr: limit trace when plotting.
    rerun: regenerating the data obtained from the metrics during training.
    f_final_prop: plotting kwargs.
    plt_ind_vals: include the individual values (results for each trace) in the
    final plot.
    plt_ind_traces: plot traces across training.
    """
    # assert ('performance' in keys) and ('stage' in keys),\
    #     'performance and stage need to be included in the metrics (keys)'
    # PROCESS RAW DATA
    if not os.path.exists(folder+'/data'+'_'+setup_nm+'_'+setup +
                          '.npz') or rerun:
        print('Pre-processing raw data')
        files = glob.glob(folder+'*'+setup_nm+'_'+setup+'*')
        assert len(files) > 0, 'No files of the form: ' + folder + '*' +\
            setup_nm + '_'+setup+'_*'
        # files = sorted(files)
        val_index = []  # stores values for each instance
        metrics = {k: [] for k in keys}
        aha_data = {'aha_mmts': [], 'prev_prfs': [], 'post_prfs': [],
                    'gt_patterns': [], 'perf_patterns': [], 'prob_right': [],
```

```
                'prob_right_aha': []}
        learn_data = {'learned': [], 'ev_not_l': [], 'ev_l': []}
        keys = np.array(keys)
        for ind_f, file in enumerate(files):
            print(file)
            val = get_tag(tag, file)
            # get metrics
            metrics, flag = data_extraction(folder=file, metrics=metrics,
                                            w_conv_perf=w_conv_perf,
                                            conv=[1, 0])
            aha_data, flag = aha_moment(folder=file, aha_data=aha_data,
                                        **ahas_dic)
            learn_data, flag = learning(folder=file, learn_data=learn_data,
                                        **learn_dic)

            # store values
            if flag:
                val_index.append(val)
        val_index = np.array(val_index)
        # AHA-MOMENT
        aha_mmts = aha_data['aha_mmts']
        prev_prfs = aha_data['prev_prfs']
        post_prfs = aha_data['post_prfs']
        gt_patterns = aha_data['gt_patterns']
        perf_patterns = aha_data['perf_patterns']
        prob_right = aha_data['prob_right']
        prob_right_aha = aha_data['prob_right_aha']
        if len(aha_mmts) > 0:
            fig, ax1 = plt.subplots()
            colors = ['b', 'g']
            labels = ['prev_prfs', 'post_prfs']
            ax1.hist([prev_prfs,     post_prfs],     bins=10,     color=colors,
label=labels)
            ax1.legend()
            plt.tight_layout()
            plt.show()

        names = ['values_across_training_']  # 'mean_values_across_training_']
        ylabels = ['Performance', 'Phase', 'Number of steps',
                   'Session performance']
        ax_final_perfs = ax_final[1]
        metrics['real_performance']
        final_wind = 100
        final_perfs = [np.mean(p[-final_wind:])
                       for p in metrics['real_performance']]
        box_plot(data=final_perfs, ax=ax_final_perfs, x=x)
        num_sh = 100000
        bins = np.linspace(0, 1, 10)
        # probabilities of right
        w = 10
        r_m = np.random.rand(num_sh, w)
        r_m = np.sum(r_m > 0.5, axis=1)/w
        prob_R_chance, plt_bins = get_hist(r_m, bins=bins)
        f, ax = plt.subplots(1, 1)
        ax.plot(plt_bins, prob_R_chance)
        prob_R, plt_bins = get_hist(prob_right, bins=bins)
        ax.plot(plt_bins, prob_R)
        ax.legend(labels=('Ground truth', 'Right side trials'))
        ax.set_title('Probabilities of ground truth=right before aha-moment')
```

```
        # probabilities of right aha
        w = 10
        r_m = np.random.rand(num_sh, w)
        r_m = np.sum(r_m > 0.5, axis=1)/w
        prob_R_chance, plt_bins = get_hist(r_m, bins=bins)
        f, ax = plt.subplots(1, 1)
        ax.plot(plt_bins, prob_R_chance)
        prob_R, plt_bins = get_hist(prob_right_aha, bins=bins)
        ax.plot(plt_bins, prob_R)
        ax.legend(labels=('Ground truth', 'Right side trials'))
        ax.set_title('Probabilities of ground truth=right during aha-moment')

        # number of aha moments for each subj
        subj_length = [len(aha_mmts)]
        f, ax = plt.subplots(1, 1, figsize=(4.5, 5))
        ax.hist(subj_length, bins=np.arange(6)-0.5)
        ax.set_xlabel('Number of aha moments')
        ax.set_ylabel('Number of subjects')
        ax.spines['right'].set_visible(False)
        ax.spines['top'].set_visible(False)
        print('Mean/std number of aha-moments')
        print(np.mean(subj_length))
        print(np.std(subj_length))
        ax.set_title('Number of aha moments for each subject')
        f, ax = plt.subplots(1, 2)
        learned_mat = 1*np.array(learn_data['learned'])
        ax[0].hist(learned_mat)
        ax[0].set_title('Subjects that learn (1 learn, 0 not)')
        ax[0].spines['right'].set_visible(False)
        ax[0].spines['top'].set_visible(False)
        zip_ = zip(learn_data['ev_l'], learn_data['ev_not_l'])
        learning_time = [x-y for x, y in zip_ if x is not None and y is not
None]
        ax[1].hist(learning_time, 8)
        ax[1].spines['right'].set_visible(False)
        ax[1].spines['top'].set_visible(False)
        ax[1].set_xlabel('Time to learn')
        ax[1].set_ylabel('Counts')
        np.mean(learning_time)
        np.std(learning_time)
        min(learning_time)

        for ind in range(len(names)):
            f, ax = plt.subplots(sharex=True, nrows=len(keys), ncols=1,
                                 figsize=(12, 12))
            # plot means
            for ind_met, met in enumerate(keys):
                metric = metrics[met]
                if plt_ind_traces:
                    plot_rew_across_training(metric=metric, index=val_index,
                                             ax=ax[ind_met], n_traces=3)
                plt_means(metric=metric, index=val_index,
                          ax=ax[ind_met], limit_ax=limit_ax)
                ax[ind_met].set_ylabel(ylabels[ind_met])
            ax[0].set_title('Roll_out = ' + str(n_roll))
            ax[0].axhline(y=0.55, linestyle='--', color='k')
            ax[0].set_xlabel('Trials')
            ax[len(keys)-1].set_xlabel('Trials')
            ax[len(keys)-1].legend()
```

```
        f.savefig(folder+'/'+names[ind]+'_'+setup_nm+'_'+setup +
                  '_'+str(limit_tr)+'.png', dpi=200)
        # plt.close(f)

    # plot days under perf
    if 'curr_perf' in keys:
        f, ax = plt.subplots(nrows=1, ncols=1, figsize=(12, 12))
        metric = metrics['curr_perf']
        perf_hist(metric, ax=ax, index=val_index, trials_day=300)
        ax.set_title('Performance histogram ('+')')
        f.savefig(folder+'/perf_hist_'+'_'+setup_nm+'_'+setup +
                  '.svg', dpi=200)
        # plt.close(f)

    # plot trials per stage
    if 'stage' in keys:
        f, ax = plt.subplots(nrows=1, ncols=1, figsize=(8, 8))
        metric = metrics['stage']
        trials_per_stage(metric, ax=ax, index=val_index)
        ax.set_title('Average number of trials per stage ('+')')
        f.savefig(folder+'/trials_stage_'+'_'+setup_nm+'_' +
                  setup+'.svg', dpi=200)
        plt.close(f)

    # PROCESS TRACES AND SAVE DATA
    tr_to_perf = []  # stores trials to reach final performance
    reached_ph = []  # stores whether the final phase is reached
    reached_perf = []  # stores whether the pre-defined perf is reached
    exp_durations = []  # stores the total number of explored trials
    stability_mat = []  # stores the performance stability
    final_perf = []  # stores the average final performance
    tr_to_ph = []  # stores trials to reach final phase
    stps_to_perf = []  # stores steps to final performance
    stps_to_ph = []  # stores steps to final performance
    if limit_tr:
        min_dur = np.min([len(x) for x in metrics['stage']])
    else:
        min_dur = np.max([len(x) for x in metrics['stage']])

    for ind_f in range(len(metrics['stage'])):
        # store durations
        exp_durations.append(len(metrics['stage'][ind_f]))
        for k in metrics.keys():
            metrics[k][ind_f] = metrics[k][ind_f][:min_dur]
            if len(metrics[k][ind_f]) == 0:
                metrics[k][ind_f] = np.nan*np.ones((min_dur,))
        # phase analysis
        stage = metrics['stage'][ind_f]
        # number of trials until final phase
        tr_to_ph, reached = tr_to_final_ph(stage, tr_to_ph, w_conv_perf,
                                           final_ph)
        reached_ph.append(reached)
        # performance analysis
        perf = np.array(metrics['real_performance'][ind_f])
        # get final performance
        final_perf.append(perf[-1])
        # get trials to reach specified performance
        tt_ph = tr_to_ph[-1]
        tr_to_perf, reached = tr_to_reach_perf(perf=perf.copy(),
```

```
                                                    tr_to_ph=tt_ph,
                                                    reach_perf=perf_th,
                                                    tr_to_perf=tr_to_perf,
                                                    final_ph=final_ph)
            reached_perf.append(reached)
            # performance stability
            tt_prf = tr_to_perf[-1]
            stability_mat.append(compute_stability(perf=perf.copy(),
                                                    tr_ab_th=tt_prf))
        data = {'tr_to_perf': tr_to_perf, 'reached_ph': reached_ph,
                'reached_perf': reached_perf, 'exp_durations': exp_durations,
                'stability_mat': stability_mat, 'final_perf': final_perf,
                'tr_to_ph': tr_to_ph, 'stps_to_perf': stps_to_perf,
                'stps_to_ph': stps_to_ph, 'val_index': val_index}
        np.savez(folder+'/data'+'_'+setup_nm+'_'+setup+'.npz',
                 **data)
    # LOAD AND (POST)PROCESS DATA
    print('Loading data from: ', folder+'/data'+'_'+setup_nm +
          '_'+setup+'.npz')
    tmp = np.load(folder+'/data'+'_'+setup_nm+'_'+setup+'.npz',
                  allow_pickle=True)
    # the loaded file does not allow to modifying it
    data = {}
    for k in tmp.keys():
        data[k] = list(tmp[k])
    val_index = data['val_index']
    print('Plotting results')
    # define xticks
    ax_props = {'tag': tag}
    if tag == 'stages':
        ax_props['labels'] = list(PRTCLS_IND_MAP.keys())
        ax_props['ticks'] = list(PRTCLS_IND_MAP.values())
    elif tag == 'th_stage':
        ax_props['labels'] = list(THS_IND_MAP.keys())
        ax_props['ticks'] = list(THS_IND_MAP.values())
    elif tag == 'pun':
        ax_props['labels'] = list(PUN_IND_MAP.keys())
        ax_props['ticks'] = list(PUN_IND_MAP.values())

    # plot results
    ax1 = ax_final[0]
    ax3 = ax_final[2]
    # final figures
    # prop of instances reaching phase 4
    ax_props['ylabel'] = 'Proportion of instances reaching phase ' + \
        str(final_ph)
    plt_perf_indicators(values=data['reached_ph'], index_val=val_index,
                        ax=ax1[0], f_props=f_final_prop,
                        ax_props=ax_props, discard=['full', '4'],
                        errorbars=False, plot_individual_values=False)
    # trials to reach phase 4
    ax_props['ylabel'] = 'Number of trials to reach phase '+str(final_ph)
    plt_perf_indicators(values=data['tr_to_ph'],
                        f_props=f_final_prop, ax_props=ax_props,
                        index_val=val_index, ax=ax1[1],
                        reached=data['reached_ph'], discard=['full', '4'],
                        plot_individual_values=plt_ind_vals)
    handles, labels = ax1[0].get_legend_handles_labels()
    by_label = dict(zip(labels, handles))
```

```
ax1[0].legend(by_label.values(), by_label.keys())
ax1[1].set_yscale('log')

# plot final performance
ax_props['ylabel'] = 'Average performance'
plt_perf_indicators(values=data['final_perf'],
                    reached=data['reached_ph'],
                    f_props=f_final_prop, index_val=val_index,
                    ax=ax3[0, 0], ax_props=ax_props,
                    plot_individual_values=plt_ind_vals)
# prop of trials that reach final perf
ax_props['ylabel'] = 'Proportion of instances reaching final perf'
plt_perf_indicators(values=data['reached_perf'], index_val=val_index,
                    ax=ax3[0, 1], f_props=f_final_prop,
                    reached=data['reached_ph'], ax_props=ax_props,
                    errorbars=False, plot_individual_values=False)
# trials to reach final perf
ax_props['ylabel'] = 'Number of trials to reach final performance'
plt_perf_indicators(values=data['tr_to_perf'],
                    reached=data['reached_perf'],
                    index_val=val_index, ax=ax3[1, 0],
                    f_props=f_final_prop, ax_props=ax_props,
                    plot_individual_values=plt_ind_vals)
ax3[1, 0].set_yscale('log')
# plot stability
ax_props['ylabel'] = 'Stability'
plt_perf_indicators(values=data['stability_mat'], index_val=val_index,
                    ax=ax3[1, 1], f_props=f_final_prop,
                    ax_props=ax_props, reached=data['reached_perf'],
                    plot_individual_values=plt_ind_vals)
handles, labels = ax3[0, 0].get_legend_handles_labels()
by_label = dict(zip(labels, handles))
ax3[0, 0].legend(by_label.values(), by_label.keys())
```