UNIVERSITY OF OSLO

Data Analysis and Machine Learning
FYS-STK3155 / FYS4155

Assignment 2

Authors
Homa Priya Tarigopula
Mouhammad Abu Rasheed
Frederik Rogge

June 15, 2022

UiO : University of Oslo

Abstract

In this project, we developed an end-to-end pipeline starting with setting up the dataset, data visualization and defining the problem. Here, we designed a classification task on the calcium imaging data recorded from postrhinal cortex of mice, while the mouse was responding to visual cues in the presence of the reward. Simply defined, the task was to identify the category of the visual cue observed by the mouse based on the neural correlates of the calcium data. This task has important implications in neuroscience such as understanding learning and memory consolidation. Being able to classify the different stimuli from the neural correlates helps to understand if the response pattern is consistent across different presentations of the stimuli. Also, here we included data from different stages of learning defined based on the reward performance obtained, starting bad to better (i.e, 3 different sessions).

As a rule of thumb, we used 80-20 split on the datasets to create train and test sets. We used a five-fold cross validation approach and applied various classification techniques, such as support vector machines, boosting and artificial neural networks to analyse the data from all 3 sessions and two different representation schemes. We observe that all the classification techniques were able to model the data with near perfect accuracies(i.e. almost close to 100 %). This trend was observed across all sessions and data representation schemes. This implies that the postrhinal cortex has distinct neural correlates for each stimuli. This holds true also for the low performance case even when the stimulus reward associations are not yet formed. Also, postrhinal cortex has consistent representations for each stimuli across different presentations.

General information

The github repository consists of *'src/notebooks'* folder which has the jupyter notebooks corresponding to data visualization, and different classification methods. The file name of each notebook decribes the analysis contained in them. We have four notebooks in total which can be summarised as follows :

- 00_visualization.ipynb - contains the calcium trace visualizations with respect to class of the stimuli
- 01_svm.ipynb - contains classification training and evaluation with Support Vector Machines
- 02_boosting.ipynb - contains classification training and evaluation with Boosting
- 03_ann.ipynb -contains classification training and evaluation with Artificial Neural Networks

Each of these notebook are well documented with comments. The dataset used for this assignment is available on the following google drive link :
`http://https://drive.google.com/drive/folders/156aJUQdgv81rn7XbzpshAB23rPB38WCC?usp=sharing`

Acronyms

ANN	artificial neural network
CV	cross validation
GECI	genetic encoded calcium indicator
ML	machine learning
MSE	mean squared error
OLS	ordinary least squares
PV	parvalbumin
POR	postrhinal cortex
ROI	region of interest
SGD	stochatic gradient descent
SVD	singular value decompositon
SVM	support vector machine

Introduction

In this assignment, we will use calcium imaging data from the postrhinal cortex (POR) of mice. The POR is part of the visual association cortex and therefore involved in visual tasks. Calcium imaging data allows to simultaneously record neural activity from hundreds of neurons over long times. The goal of the experiments that the data result from is to study mechanisms of learning. Part of a widely used learning theory is that learned experience is *reactivated* during subsequent sleep or quiet waking in order to strengthen connections between neurons that fire together and therefore, consolidate memories [7]. The focus of these experiments are exactly these *reactivation* events that are well studied particularly in the hippocampus [7], but have also been shown to occur in the POR [14]. For this purpose, recordings were performed before, while and after mice learned a visual association task over a course of several sessions. In this task, they had to learn to discriminate three different grating orientations which were presented on a screen in front of them one at a time while they were head-fixed on a running wheel. The three different gratings result in three possible outcomes (reward, neutral, and aversive). The ultimate goal of these experiment to study the role of parvalbumin (PV) neurons, a type of inhibitory neurons that has shown to be important in synchronizing different oscillation pattern of brain activity in the brain that are involved in memory consolidation [11, 15]. This synchronization seems to be important for learning since first results show that mice with silenced PV neurons in the POR cannot learn the task even if trained for a long time. It will therefore be interesting to see what the effect of silencing these neurons on the reactivations is. For this purpose, two things have to be shown in advance

1. consistent response pattern to the different stimuli across different trials have to be shown in order to be able to identify these pattern during subsequent sleep,
2. the reactivations, so the reinstantiation of the pattern during sleep, has to be shown in control mice with intact PV neurons to have a baseline to compare the impaired mice with. There are different methods to do this, but no unique standard exists to this date.

In this assignment, we will focus on the first (might explore the second one) of these pre-eliminary tasks. To this end, we will use and compare different classifiers in classifying neural population activity. The possible outcomes will be the different stimuli, or non-stimuli classes. In particular, the questions that we want to address here are

- Can we train a classifier on neural activity pattern to infer the stimuli (or other states) that was presented based on unseen neural activity?
- Can we use the same classifier to identify these pattern during subsequent sleep, hence identifying reactivations?

Generally, we would expect the classification problem to be rather simple since we expect strong and (for some cells) selective responses in the POR to visual stimuli. In addition to that, we will only consider a small number of different classes. However, it will still be interesting to compare different classifiers, specially with respect to the next task in which the pattern have to be identified in subsequent sleep. The challenge here is that for this data there are no labels since in contrast to the training time, we do not present the mice with the visual stimuli, but instead just look for the reinstantation of the same pattern. A high and stable performance of the classifier is therefore crucial to yield reliable results. For this purpose, we will explore the use of different classifiers, data, and processing methods.

Background

This chapter deals with providing some background needed before diving into the classification problem. First, the basics of calcium imaging are explained in section 2.1. In section 2.2, the fundamentals of the learning theory including the reactivation phenomena are described. Section 2.3 explains the learning task in more detail.

2.1 Calcium Imaging Data

Calcium imaging refers to an imaging method in which the concentration of CA^2 in single neurons is measured. This can, for example, be done through two photon microscopes which is the technique used for the data in this work. Every action potential in neurons is followed by an influx of calcium ions. This effect is exploited by so-called calcium indicators. There exist two types of calcium indicators: chemical indicators and genetic encoded calcium indicators (GECIs). For the recordings in this work, a GECI from the GCaMP family was used. For this purpose, the neurons (globally or locally in a certain area or of a certain neuron type) are forced to produce a certain protein that can bind calcium ions. After binding calcium ions, it can be excited using light of a certain wavelength and would then in turn emit light through the fluorescence effect. This light can be measured by the two-photon microscope and used as a proxy for the neural activity [5].

The output of the microscope is a stack of 2d images recorded at a rate of $30.9Hz$. To extract the neurons and their activity from these images, an open-source tool called *suite2p* is used [12]. This tool performs the following steps

1. register images to reference image by calculating offsets using phase correlation (the offset can occur due to head movement of the mice during the experiment),
2. detect region of interests (ROIs),
3. classify ROIs as either cell or non-cell,
4. extract fluorescence signal of each of the cells which will also referred to as traces.

The resulting signal is the raw fluorescence that has an arbitrary unit. It can then further be processed and normalized since each cell might have different baseline activities, different impacts of noise from surrounding tissue, and individual dynamics. First, the raw traces are neuropil-corrected. The neuropil signal is the fluorescence from cells and tissue close to the cell of interest. It is therefore defined as a donut-like shape around the cell [12]. The traces are then corrected using the following formula

$$F_c = F - 0.7F_{neu}, \tag{1}$$

where F_c is the neuropil-corrected trace, F the raw trace and F_{neu} the neuropil signal. To make the traces of different cells more comparable, the so-called $\Delta F/F$ if commonly used. It is defined as

$$\Delta F/F = \frac{F_c - F_0}{F_0}, \tag{2}$$

where F_0 is the baseline activity which is calculated as the median of the corrected trace F_c in a sliding window of $300s$. As mentioned, calcium traces are transient signals that are only a proxy for the actual spikes (action potentials) of the cell. After a spike happened, the signal increases and then decreases again with a decay rate that depends the calcium indicator is is described by the parameter τ. To get the actual spike rate, a deconvolution algorithm can be used that would estimate the spike rate from the $\Delta F/F$ trace. In this case we used the *oasis* algorithm which is commonly used in the field [4]. We will use different types of signal and compare the results in in this work.

2.2 Learning Theory

The hippocampus has long been known to play a crucial role in the formation of new epsiodic memories. One famous example is that of Henry Gustav Molaison, a patient with epilepsy who lost the ability to form new memories after his hippocampus was impaired during a surgery [13]. Currently, the hippocampus is viewed as a kind of working memory that stores and represents information of an episode of experience. It is assumed that during subsequent sleep, these memories are being transferred to cortical areas into more efficient representations. Here, they are integrated into and connected to existing memories. This process is also referred to as *memory consolidation* [7]. During sleep, different types of oscillations in the neural activity have been identified. Two prominent examples are ripples (≈ 80Hz) and spindles ($\approx 12-15$Hz). Ripples occur mainly in the hippocampus and these fast oscillations are coupled with spindles which originate in the Thalamus but also reach cortical areas. It has been shown that this coupling is important for learning, and that PVs neurons play a crucial role in this coupling [11]. During this coupling, recent experiences are replayed/ reactivated which is thought to be another mechanism of learning through which the connections between neurons that fire together are strengthened [7]. This reactivations have been shown for hippocampal place cells [8], but also in cortical areas [14].

2.3 Visual Association Task

The experiment is based on a paper from 2017 [14]. Mice are head-fixed on a running wheel while being presented with visual stimuli on a screen in front of them. Each stimulus is presented for $3s$ and each of these presentations is referred to as one trial. The stimuli set consists of three different gratings. In front of the mice is a lick bout. Each of the stimuli is associated with a certain outcome. The reward cue will give a reward in form of a milkshake if the mouse licks. The second one, called neutral cue, will have no outcome and the third one, called the negative cue, will result in a punishment in form of quinine if the mouse licks after the stimulus was shown. After the stimulus onset, the mouse has a 2s response window to lick. The experiment of one mouse consists of several sessions and one session comprises all recordings from one day. Each session consists of 6 runs and each run is around 30 min long. The first run is a baseline run without any stimuli. Runs 2-4 are training runs during which the stimuli are shown in random order. After run 4, the mice are fed with a lot of milkshake and hence get tired. During run 5 and 6, no stimuli are shown and the idea is that during that time, the mice doze off. In this state, the mice will consolidate the memory and likely replay the experiences just made.

2.4 Dataset

The data set used in this study consists of cells recorded from the same mouse during different experimental sessions. The data is chunked according to the category of the presented visual stimuli. Thus each sample of data is a time series of all recorded cells and is labelled into 'positive', 'neutral', 'negative' or 'other' based on the visual cue. The category 'other' corresponds to the experimental instances during which no cue is presented. The data represented by each sample is either the normalized fluorescence(dff-denoised) or the firing frequency(de-convolved) of each cell over time. This continuous time series data is sampled at around 30.9Hz such that each sample consists of multiple discrete time points for each cell. This 2D feature matrix representing each sample is then converted to a 1D feature array by taking the maximum fluorescence value for each cell from the discretized time points.

Table 1: Overview of the dataset

Attribute	Statistics	Description
Mice	1	KKL366
Sessions	3	220124, 220125, 220126
Preprocessing methods	2	deconvolved, dff-denoised
Visual categories	4	positive, neutral, negative , other
Total Samples	2940	other: 2454, negative: 162, neutral: 162, positive: 162
Session-id	Number of samples	Number of cells recorded x Number of time points
220124	2940	620 x 30
220125	2940	551 x 30
220126	2940	553 x 30

Analysis

3.1 Calcium Traces

Figure 1 shows different signal of the same cell. On the top, the raw fluorescence is shown which has an arbitrary unit. The middle shows the denoised $\Delta F/F$ so the relative increase or decrease of the fluorescence with respect to the baseline. The bottom shows the deconvolved signal which represents the extracted spike rates from the trace. The colored rectangles show the reward (green), neutral (blue) and aversive (red) stimuli. It can be seen that this cell has a very clear selective response to the reward cue as the fluorescence always increases on presentation of this cue.

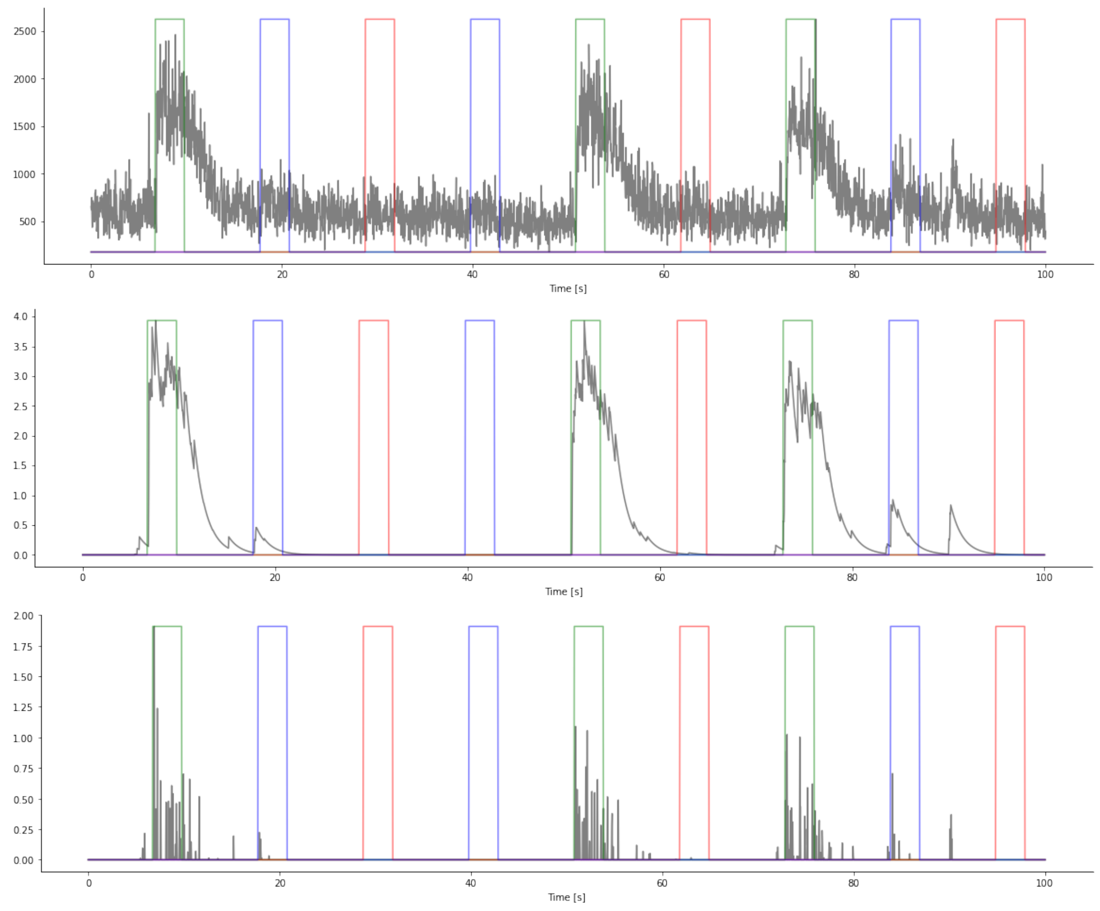

Figure 1: Traces from cell 2 of subject KKL366 in session 220124 during the first training run. From top to bottom: Raw fluorescence, denoised $\Delta F/F$, deconvolved trace. Colored rectangles represent the different stimuli. Reward stimulus shown in green, neutral stimulus shown in blue and aversive stimulus shown in red.

3.2 Behavioral Performance

To measure the performance of the mouse in the task, d prime is a common metric. It is the standardized difference between the hit rate (*How often did the mouse lick of the cases the positive stimulus was presented?*) and the false alarm rate (*How often did the mouse lick of the cases the negative or neutral stimulus was presented?*). A value close to 3 means an almost perfect performance while values close to zero mean a bad or random performance. The d́values for all the sessions used in this project can be seen in table 2. It shows that the performance is very low in the beginning (around 0 for session 220124) and gets better over time.

3.3 Support Vector Machines

As the first classifier, we will use support vector machines (SVMs). First, some background about SVMs is provided in section 3.3.1. The results are presented and discussed in section 3.3.2.

Table 2: Overview of the performance over sessions

Session	d prime
220124	-0.06
220125	0.89
220126	1.02

3.3.1 Background

The underlying idea of SVMs is to fit a separating hyperplane between the samples of different classes that maximizes the distance to the samples of each class. In our example, we have two different classes that we wish to separate, and that can take the values $y_i = \{-1, 1\}$. In the simple case, the separation is described by

$$x^T w + b = 0, \tag{3}$$

where w are the weights, b is a constant bias and x are all points that lie on the hyperplane. We want to find w such that for a sample x_i

$$\begin{aligned} w^T x_i - b &\geq 1, \text{ if } y_i = 1, \\ w^T x_i - b &\leq 1, \text{ if } y_i = -1, \end{aligned} \tag{4}$$

which means that the closest points to the hyperplane (the decision boundary) of each class have a distance of $\frac{1}{\|w\|}$ to it resulting in a width of $\frac{2}{\|w\|}$. More simple, this can be written as

$$y_i(w^T x_i - b) \geq 1. \tag{5}$$

At the same time, we also want to maximize the width between the distance to the two closest points of each class by minimizing the norm of w, $\|w\|$ in order to increase our confidence when classifying new samples. This can be formulated as an optimization problem:

$$\begin{aligned} &\text{Minimize } \|w\| \\ &\text{subject to } y_i(w^T x_i - b) \geq 1 \, \forall x_i \end{aligned} \tag{6}$$

To solve this, we define the Lagrangian function

$$\mathcal{L}(\lambda, b, w) = \frac{1}{2} w^T w - \sum_{i=1}^{n} \lambda_i [y_i(w^T x_i + b) - 1], \tag{7}$$

where λ is the Lagrangian multiplier. Using the derivates with respect to w and b, we yield

$$\mathcal{L} = \sum_i \lambda_i - \frac{1}{2} \sum_{ij}^{n} \lambda_i \lambda_j y_i y_j x_i^T x_j, \tag{8}$$

subject to the constraints $\lambda_i \geq 0$ and $\sum_i \lambda_i y_i = 0$. We can use this to solve for λ and then compute the parameters using

$$\begin{aligned} w &= \sum_i \lambda_i y_i x_i, \\ b &= \frac{1}{y_i} - w^T w. \end{aligned} \tag{9}$$

A sample can then be classified as one of the two classes using

$$y_i = \text{sign}(w^T x_i + b). \tag{10}$$

Instead of using a linear hyperplane, it is possible to use more complex ones that are able to separate the two classes better. For this, we define a so-called Kernel as

$$K(x_i, x_j) = z_i^T z_j = \phi(x_i)^T \phi(x_j), \tag{11}$$

where ϕ is some mapping function that we do not have to know explicitly if Mercer's theorem [10] is fulfilled. The objective then simply turns into

$$\mathcal{L} = \sum_i \lambda_i - \frac{1}{2}\sum_{ij}^{n} \lambda_i \lambda_j y_i y_j z_i^T z_j, \tag{12}$$

In this work, we will stick to the standard linear kernel. In addition to the two constraints mentioned before, it is possible to add a regularization similar to the ridge regression (the weighted norm of the parameters w) which is done by introducing a so-called slack variable [6].

3.3.2 Results & Discussion

The results of an SVM trained on the different datasets and sessions can be found in fig. 2. In all cases, the accuracy is 100% or almost 100%. This shows that the responses to the different stimuli are highly differentiated. Not only is a stimulus presentation differentiable from times of other activity, but the stimuli are also all uniquely encoded by different neurons. Even though this has already been shown in other works [14], it is interesting to note since the POR is a downstream visual area (in contrast to the primary visual cortex), and the performance of the mouse in the task is still very bad at this point (d prime of around 0). This shows that even though there is no learned association between stimulus and outcome, all stimuli are already uniquely encoded.

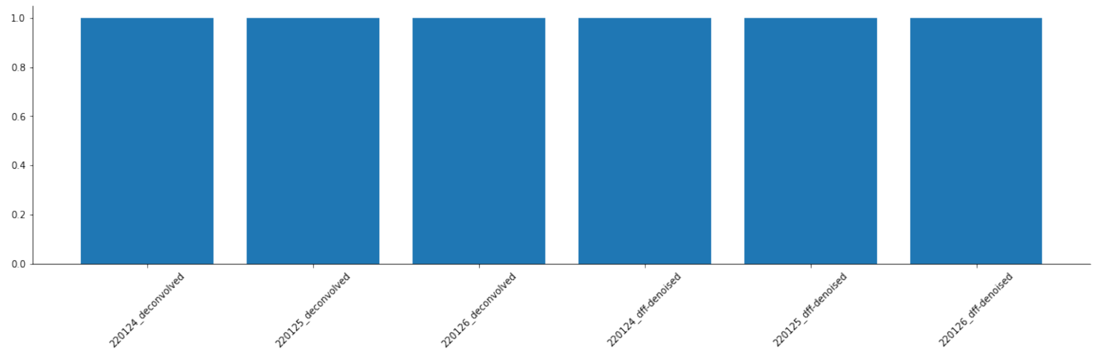

Figure 2: Mean accuracy (5-fold cross validation) of SVM on test set of different datasets.

3.4 Boosting

3.4.1 Background

Since the aim of any ML algorithm is to build a prediction model which is as much accurate as we can to predict some labels based on specific predetermined features, defining how these features shape the model was of highest priority and consumes most of the programmers' computation time. Starting from simple ML algorithms that are based on applying a one-step feature(s)-to-label(s) model, moving to the deep learning methods that can contain multi-step forwarding/convoluted model, it is the most challenging step to find a strict rule through which the given data would perfectly fit. However, it is not that difficult to determine several weak predictors based on some data that do not give high accuracy but it is better than random model. Based on these weak predictors, Boosting algorithm can sequentially check the mislabeled data points and give them higher weight for the next classifier, once the boosting algorithm combines these weak predictors, it will achieve a model with higher accuracy.
From its name, Boosting is a sequential ensemble learning algorithm that is being applied to enhance the accuracy of a given machine learning (ML) algorithm. It refers to a general and provably effective method of producing a very accurate prediction rule by combining rough and moderately inaccurate rules of thumb in a manner similar to that suggested above.
Boosting can be principally applied to any algorithm, for both regression and classification problems. Some popular types of Boosting algorithms are Ada-Boost, Gradient Boosting, and XGBoost. These methods are different in the scope of applications, the way the base model is chosen, and the way they iterate.
Since the given dataset in this project is basically a classification problem, our work will be focusing on implementing Ada-Boost algorithm which has been extensively studied and tested for this type of predictions.
The Ada-Boost algorithm was initially proposed by Freund and Schapire in 1995 [2]. Herein an explanation of this algorithm for a binary classification problem [3]:

1. Given a training dataset as $(x_1, y_1),\ldots,(x_i, y_i),\ldots,(x_m, y_m)$, where $x_i \in X$, $y_i \in Y = [-1, +1]$.
2. Ada-Boost calls a given weak or base learning algorithm repeatedly in a series of rounds: $t = 1, \ldots\ldots, T$.
3. Initialize equal weights for each training example at the first round $D_1(i) = 1/m$.
4. Train the base model using distribution D_1.
5. Get base (weak) classifier $h_t : X \to R$ such that the error ϵ_t is minimized; where ϵ_t is the error between $h_t(x_i)$ and y_i.
6. Choose $\alpha_t \in R$ that is a parameter to measure the importance to be assigned to h_t for the next distribution:
$$\alpha_t = \frac{1}{2} \ln\left(\frac{1-\epsilon_t}{\epsilon_t}\right) \tag{13}$$
7. The distribution D_t is then updated as follows:
$$D_{(t+1)}(i) = \frac{D_t(i) \times \exp\left(-\alpha_t y_i h_t(x_i)\right)}{Z_t} \tag{14}$$
Where Z_t is a normalization factor (chosen so that $\sum_{i=1}^{m} D_{(t+1)}(i) = 1$.
8. Repeat steps 5–7.
9. Output the final (combined) classifier $H(x)$ which is a weighted majority vote of the T base classifiers where α_t is the weight assigned to h_t:
$$H_x = \operatorname{sgn}\left(\sum_{t=1}^{T} \alpha_t h_t(x)\right) \tag{15}$$

Based on the previous algorithm, there were some controversial opinions whether Boosting would be a non-overfitting method which means that we do not need to stop the iterations at some point. However, it is currently known that even Boosting can leads to overfitting but it is considered to have slow overfitting behavior compared to other algorithms in several papers, especially in the classification case with AdaBoost algorithm [1].

3.4.2 Results & Discussion

Data-sets from all files were uploaded and split into training and testing data (80:20 ratio). The maximum value of the collected data was used as feature for labels prediction through the whole section. AdaBoost algorithm was implemented using python package (see the code). The two hyperparameters are number of estimators (n) and learning rate (r). These parameters were initially fixed at 500 and 1; respectively. Afterwards, a trade-off was studied for different values of n and r. For the sake of decreasing the computation cost, a step-wise increase of 10 in n and a logarithmic increase in r were set. Besides, the range for the former was limited to 100 based on preliminary accuracy data from optimization with fixed learning rate (r=1). This optimization was applied on all data-sets (deconvolved and dff-denoised) from the three collected sessions.
To sum up, the following steps were followed for the parameters optimization study:

1. The code was implemented using fixed hyperparameters, and values of 500 and 1 were chosen for n and r; respectively.
2. An optimization of number of estimators (n) with fixed $r = 1$ was done by implementing the AdaBoost code on a range of n values between 1 and 1000 ($stepsize = 10$) and calculating the accuracy percentage for each iteration.
3. A single AdaBoost implementation with a fixed hyperparameters at $n = 10000$ and $r = 1$ was employed to check if the accuracy score limit will stay stable after 1000.
4. The main optimization was done by running the AdaBoost algorithm on an array of different values of n and r as follows:
$n = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 20, 30, 40, 50, 60, 70, 80, 90]$
$r = [0.0001, 0.001, 0.01, 0.1, 1, 10]$
5. The accuracy score was the optimization parameter through this task.

The results for implementing AdaBoost algorithm with fixed hyperparameters (n=500, r=1) showed high accuracy score in the range (89.3 – 99.7%) before applying any optimization.
fig. 3 shows the optimization results of n between 1 and 1000 with a fixed learning rate at $r = 1$. Generally speaking, the model showed high accuracy for most datasets using low values of n. Except for session #25, naive deconvolved data-sets gave better accuracy than the dff-denoised ones, which suggest that the denoising process decreases the consistency in the data while eliminating the noise. Session #25 denoised data showed some fluctuations even at high estimators' numbers with comparatively low accuracy for the raw deconvolved data.

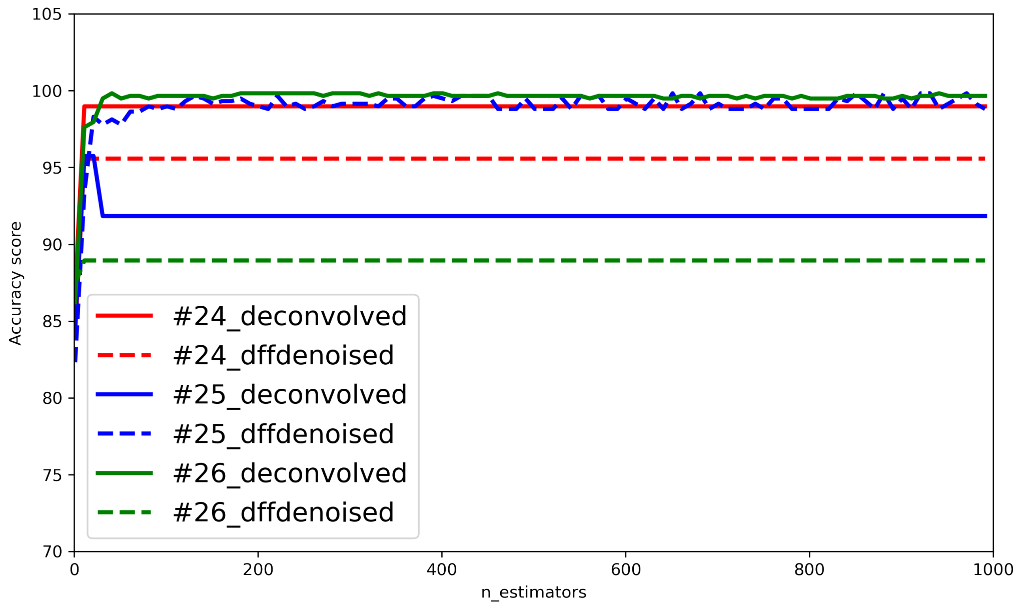

Figure 3: Accuracy scores of n-trade-off optimization for different data-sets.

Based on the previous results, we will first implement a one-time AdaBoost code at very high number of estimators ($n = 10000$) to check if the accuracy score will be improved. Also, the fluctuations in results from

some data sets suggest that the learning rate is relatively high. We will, hence, implement an n/r-trade-off to choose the most accurate model. Depending on the results from the first optimization, and in order to decrease the computation cost, this optimization will run in the range of $n \in [1, 100]$.
Out of all data-sets studied, only one showed a sign of little over-fitting at $n = 10000$. Some models produced very slightly increased accuracy at this limit. However, they consumed considerably longer computation time. This suggests the need to define our model in a lower range of n, possibly by manipulating the learning rate.

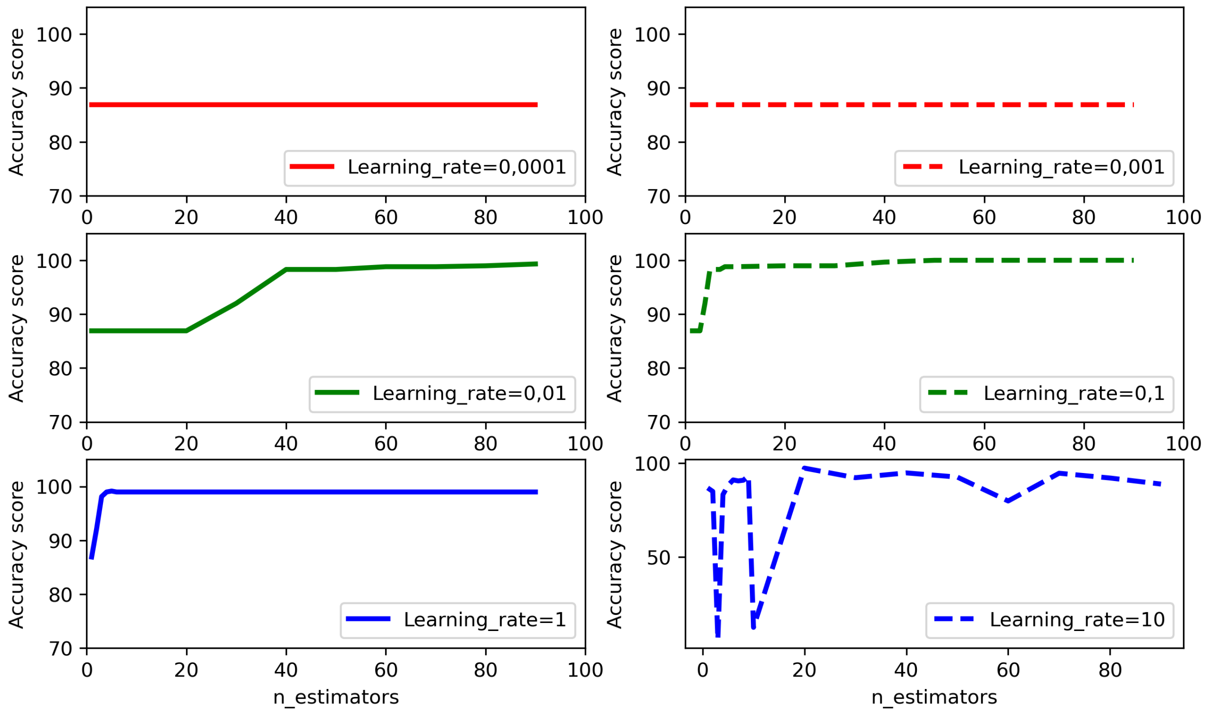

Figure 4: Accuracy scores of n/r-trade-off optimization for data-set #24-deconvolved.*Figures for other data/sets were omitted for clarity, but available at the git/hub code

fig. 4 reports the results from n/r-trade-off for the first data-set (#24 deconvolved). Firstly, lowering the learning rate to 0.1 had better effect on the accuracy score than increasing n. 100% accuracy score was retrieved in some models with substantially shorter computation time. When $r = 0.0001$ and $r = 0.001$ were used however, the models could not achieve any higher accuracy within the range of $n \in [1, 100]$. On the other hand, although the computation cost is lower for higher rates, the patterns were erratic when $r = 10$ was used. Some models even showed this pattern at $r = 1$ level. Roughly speaking, models with $r = 0.1$ and $n = 80$ were the best predictors for all models. Accuracy scores for these models are given in fig. 5.

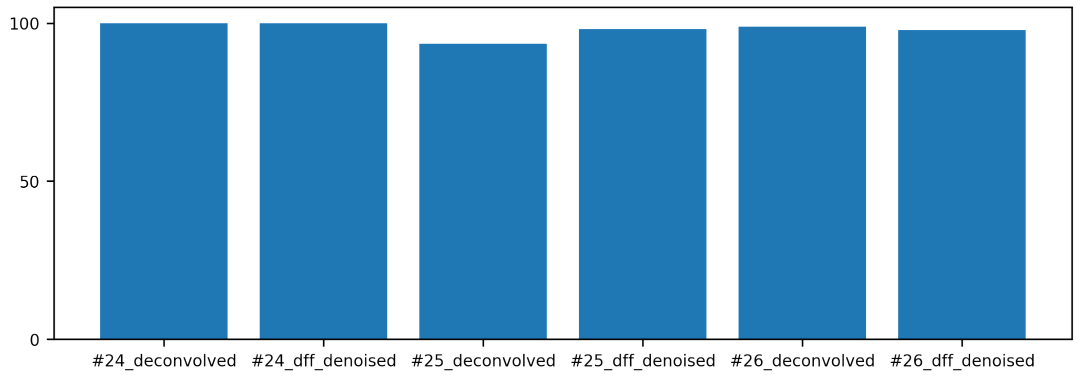

Figure 5: Accuracy scores of the optimized models at $n = 80, r = 0.1$.

3.5 Artificial Neural Networks

3.5.1 Background

Artificial Neural Networks, as the name suggests are a class of machine learning models inspired from brain. Their design mimics the biological system in which billions of neurons interact by propagating electrical signals to accomplish a task. Each neuron integrates the electrical signals it receives from upstream neurons (incoming connections) and sends out signals further to downstream neurons(outgoing connections or activations) when the accumulated signal exceeds a threshold. These neurons arrange themselves into layers, forming a hierarchy, based on the complexity of the signals they process. This was first mathematically modelled by [9] the following simple equation:

$$y = f(\sum_{i=1}^{n} w_i x_i) \tag{16}$$

where y represents the output activation of a single neuron, f a typical non-linear activation function which takes as input the weighted sum of the activations from other neurons namely $(x_1, x_2, x_3, ..., x_n)$

This is known as the McCulloch and Pitts model. The major difference between this model and the current day artificial networks is the McCulloch-Pitts model assumed the inputs $(x_1, x_2, x_3, ..., x_n)$ to be binary. However, the present day systems take any real valued input.

These networks are referred to as feed-forward neural networks. They generally consist of neurons arranged in one or more layers, with each node(neuron) in each layer integrating the inputs from the previous layers and evaluating the activation through a non-linearity. These activations are then passed on to layers further up. These networks consists of input, output and hidden layers. The input layer passes the input features or data to the hidden layers and the output layer returns the output which is a continuous value in case of regression or a class label in case of classification. There exist no lateral connections within each layer in these networks. In other words, neurons from one-layer always project to the next layer but never between themselves.These class of networks are also termed as Multi-Layer Perceptrons. These networks are most commonly used for supervised learning, where the network is trained to predict a certain value or a class based on the labelled/known ground truth data.

These networks are trained to minimize an objective which is the cost function or the loss function to learn the weights, **W** and biases, **b**. Here, **W** is a matrix of weights that act between every two consecutive layers of the network and **b** is a vector that gets summed to the weighted input . For eg., for a multi-layer perceptron with three layers- input: 10 nodes(features), hidden: 5 nodes and output: 2 nodes(classes); there exist two weight matrices - one between input and the hidden layer of size 10x5 and another between hidden and output layers of size 5x2 and two bias vectors each of size 5 and 2. These weights and biases determine the relative contribution of the different input features for classification or regression.

The generalized expression for the output from a Multi-layer Perceptron of l hidden layers can be defined as follows :

$$y_i^{l+1} = f^{l+1}\left(\sum_{j=1}^{N_l} w_{ij}^3 f^l\left(\sum_{k=1}^{N_{l-1}} w_{jk}^{l-1}\left(\dots f^1\left(\sum_{n=1}^{N_0} w_{mn}^1 x_n + b_m^1\right)\dots\right) + b_k^2\right) + b_1^3\right) \tag{17}$$

The most commonly used cost function or the loss function, in the case of regression, is the mean squared error loss and in the case of classification, is the cross entropy loss. The loss function is then optimized through back-propagation to update the weights through the layers. The back-propagation algorithm uses the chain rule to compute the gradients of the loss with respect to weights in each layer. These gradients are then used to adjust the weight matrices.

The back-propagation algorithm can thus be summarised as follows :

1. The partial derivative of the cost function $\mathcal{C}$ is computed with respect to the weights w_{ij}^l where l corresponds to layer and w_{ij} corresponds to the ij^{th} entry of the weight matrix. Chain rule is used to compute this derivative as follows:

$$\frac{\partial \mathcal{C}}{\partial w_{ij}^l} = \frac{\partial \mathcal{C}}{\partial z_i^l}\frac{\partial z_i^l}{\partial w_{ij}^l} \tag{18}$$

where

$$z_i^l = (\sum_{k=1}^{m} w_{ik}^l x_k^{l-1} + b_i^l) \tag{19}$$

2. The weights and the biases of the network are then updated as follows :

$$w_{ij}^l \leftarrow w_{ij}^l - \eta \frac{\partial \mathcal{C}}{\partial w_{ij}^l} \tag{20a}$$

$$b_i^l \leftarrow b_i^l - \eta \frac{\partial \mathcal{C}}{\partial b_i^l} \tag{20b}$$

where η corresponds to learning rate of the optimizer.

The design considerations while defining these networks include the number of nodes in each layer of the network, the number of layers in the network, the type of non-linearity used between the layers, choice of optimizer, learning rate, batch-size and epochs. All these are hyper-parameters which can be varied to improve the model performance.

3.5.2 Methods

Here, we designed a feed-forward neural network or a multi-layer perceptron with one hidden layer. The hidden layer consisted of 100 nodes and the output layer consisted of 4 nodes corresponding to each category. We used ReLU (Rectified Linear Unit) non-linearity on the hidden layer and dropout (with p=0.5). The dropout was introduced to have a regularizing effect on the weights. We used five fold cross validation on the data to evaluate the classification performance of the trained models. We trained one model for each session and each data preprocessing(dff-denoised vs deconvolved) type since the number of cells recorded differed between sessions. We used the Stochastic Gradient Descent(SGD) optimizer with a learning rate $lr = 0.003$ and batch size of the data loader was set to 16. The model was trained for 30 epochs.

3.5.3 Results & Discussion

We present in fig. 6 the results from training multi-layer perceptron across the five-folds for different sessions. The same results are also presented in table 3 summarizing the actual mean accuracies of each session. These results show that the simple neural network was sufficient to model the different sessions with near perfect accuracies(very close to 100%). This suggests that neuronal activity has unique coding for the different stimuli thus supporting perfect classification.

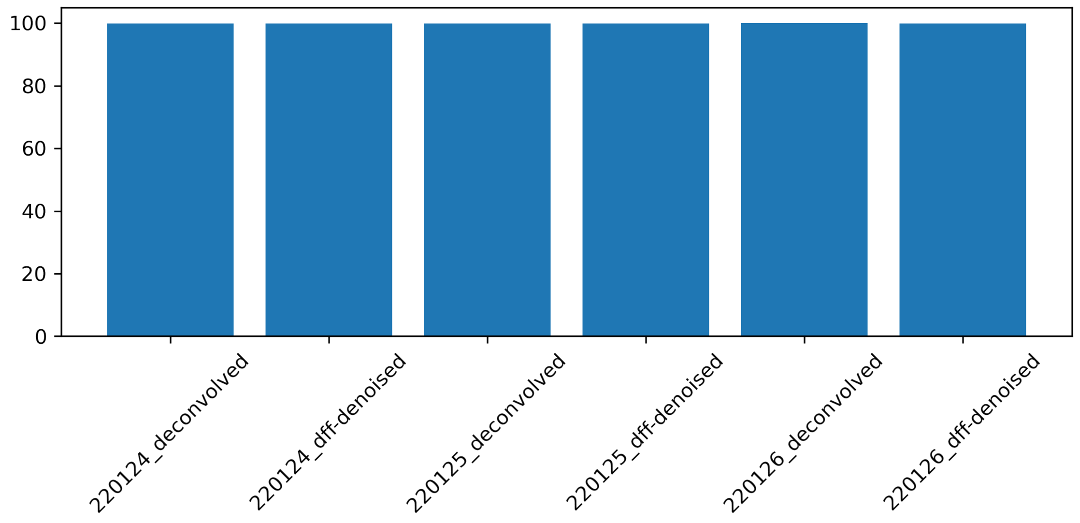

Figure 6: Mean accuracy of the ANN across the five test folds of the different sessions

Table 3: Mean accuracy across the five folds for different sessions and data preprocessing methods

Session ID	Data preprocessing	Mean accuracy(in %)
220124	deconvolved	99.92
220124	dff-denoised	99.89
220125	deconvolved	99.97
220125	dff-denoised	99.90
220126	deconvolved	100
220126	dff-denoised	99.92

Discussion

In this section, we briefly compare the different classification methods and summarise the main implications of these findings to neuroscience.

1. All the aforementioned models have almost 100% accuracy on the test set for the different sessions and data representations.
2. Since the size of the dataset for each of the sessions is not very large, there is no key difference in performance between the different techniques.
3. The results from all the above methods commonly emphasise that the neural representation from the postrhinal cortex has unique code for each stimulus category which support better classification.
4. Also, these neural correlates are consistent between different presentations of the same stimuli within the session.
5. The above finding holds true even for the low performance session in which the stimulus, outcome associations are not yet formed.
6. The difference in the frequency of different categories within a session doesn't affect classification performance.

References

[1] P. Buhlmann and T. Hothorn. Boosting algorithms: Regularization, prediction and model fitting. 22(4): 477–505. doi: 10.1214/07-STS242. URL http://rob.schapire.net/papers/FreundSc99.pdf.

[2] Y. Freund and R. E. Schapire. A decision-theoretic generalization of on-line learning and an application to boosting. 55(1):119–139, . doi: 10.1006/jcss.1997.1504. URL http://www.sciencedirect.com/science/article/pii/S002200009791504X?via%3Dihub.

[3] Y. Freund and R. E. Schapire. A short introduction to boosting. 14(5):771–780, . URL http://rob.schapire.net/papers/FreundSc99.pdf. [translated version by Naoki Abe].

[4] J. Friedrich, P. Zhou, and L. Paninski. Fast online deconvolution of calcium imaging data. 13(3):e1005423. ISSN 1553-7358. doi: 10.1371/journal.pcbi.1005423. URL https://dx.plos.org/10.1371/journal.pcbi.1005423.

[5] C. Grienberger and A. Konnerth. Imaging Calcium in Neurons. 73(5):862–885. ISSN 08966273. doi: 10.1016/j.neuron.2012.02.011. URL https://linkinghub.elsevier.com/retrieve/pii/S0896627312001729.

[6] T. Hastie, R. Tibshirani, and J. Friedman. *The elements of statistical learning: data mining, inference and prediction*. Springer, 2 edition, 2009. URL http://www-stat.stanford.edu/~tibs/ElemStatLearn/.

[7] J. G. Klinzing, N. Niethard, and J. Born. Mechanisms of systems memory consolidation during sleep. 22 (10):1598–1610. ISSN 1097-6256, 1546-1726. doi: 10.1038/s41593-019-0467-3. URL http://www.nature.com/articles/s41593-019-0467-3.

[8] K. Louie and M. A. Wilson. Temporally Structured Replay of Awake Hippocampal Ensemble Activity during Rapid Eye Movement Sleep. 29(1):145–156. ISSN 08966273. doi: 10.1016/S0896-6273(01)00186-6. URL https://linkinghub.elsevier.com/retrieve/pii/S0896627301001866.

[9] W. McCullock and W. Pitts. A logical calculus of ideas immanent in nervous activity. archive copy of 27 november 2007 on wayback machine. *Avtomaty [Automated Devices] Moscow, Inostr. Lit. publ*, pages 363–384, 1956.

[10] J. Mercer and A. R. Forsyth. Xvi. functions of positive and negative type, and their connection the theory of integral equations. *Philosophical Transactions of the Royal Society of London. Series A, Containing Papers of a Mathematical or Physical Character*, 209(441-458):415–446, 1909. doi: 10.1098/rsta.1909.0016. URL https://royalsocietypublishing.org/doi/abs/10.1098/rsta.1909.0016.

[11] N. Ognjanovski, S. Schaeffer, J. Wu, S. Mofakham, D. Maruyama, M. Zochowski, and S. J. Aton. Parvalbumin-expressing interneurons coordinate hippocampal network dynamics required for memory consolidation. 8(1):15039. ISSN 2041-1723. doi: 10.1038/ncomms15039. URL http://www.nature.com/articles/ncomms15039.

[12] M. Pachitariu, C. Stringer, M. Dipoppa, S. Schröder, L. F. Rossi, H. Dalgleish, M. Carandini, and K. D. Harris. Suite2p: Beyond 10,000 neurons with standard two-photon microscopy. URL http://biorxiv.org/lookup/doi/10.1101/061507.

[13] W. B. Scoviille and B. Milner. LOSS OF RECENT MEMORY AFTER BILATERAL HIPPOCAMPAL LESIONS. page 11.

[14] A. U. Sugden, J. D. Zaremba, L. A. Sugden, K. L. McGuire, A. Lutas, R. N. Ramesh, O. Alturkistani, K. K. Lensjø, C. R. Burgess, and M. L. Andermann. Cortical reactivations of recent sensory experiences predict bidirectional network changes during learning. 23(8):981–991. ISSN 1097-6256, 1546-1726. doi: 10.1038/s41593-020-0651-5. URL http://www.nature.com/articles/s41593-020-0651-5.

[15] F. Xia, B. A. Richards, M. M. Tran, S. A. Josselyn, K. Takehara-Nishiuchi, and P. W. Frankland. Parvalbumin-positive interneurons mediate neocortical-hippocampal interactions that are necessary for memory consolidation. 6:e27868. ISSN 2050-084X. doi: 10.7554/eLife.27868. URL https://elifesciences.org/articles/27868.