### IMPORTS FUNDAMENTAL LIBRARIES & MODULES NEEDED TO RUN THE CODE

import numpy as np

import matplotlib.pyplot as plt

%matplotlib inline

import seaborn as sns

import os

import subprocess

import time

import pandas

import pickle

import math

from sklearn.decomposition import PCA

from sklearn.preprocessing import StandardScaler

from sklearn.svm import SVC, SVR, LinearSVC

from sklearn.metrics import accuracy\_score, silhouette\_score, adjusted\_rand\_score, silhouette\_samples

from sklearn.cluster import AgglomerativeClustering, SpectralClustering, KMeans

from sklearn.model\_selection import KFold, LeaveOneOut, train\_test\_split

from sklearn.model\_selection import GridSearchCV

from sklearn.kernel\_ridge import KernelRidge

from sklearn import linear\_model

from sklearn.manifold import TSNE

import scipy.stats as stats

import scipy.io as sio

from scipy import interpolate

from sklearn.metrics import roc\_auc\_score as auROC

import statsmodels.api as sm

import statsmodels.formula.api as smf

from patsy import (ModelDesc, EvalEnvironment, Term, EvalFactor, LookupFactor, dmatrices, INTERCEPT)

from statsmodels.distributions.empirical\_distribution import ECDF

import matplotlib.cm as cm

import matplotlib.colors as colors

import matplotlib.colorbar as colorbar

import matplotlib.patches as mpatches

import sys

import re

import random

np.set\_printoptions(suppress=True)

### DEFINES FRAMERATE. INPUT BASED ON SCOPE IMAGING SPEED AND FRAME AVERAGING.

### MUST BE ADJUSTED FOR EACH LAB/EXPERIMENT

frameaveraging = 4

timebetweenframes = 33.33333

### THIS SHOULD NOT BE CHANGED

framerate = 1000/timebetweenframes/frameaveraging

### FUNCTIONS USED TO ALIGN BEHAVIORAL DATA AND IMAGING DATA. THESE LINES WILL NOT ACTUALLY RUN UNLESS THE

### FUNCTIONS ARE USED BELOW.

### EXTRACTS BEHAVIORAL DATA (.MAT) AND FLUORESCENT SIGNALS (.NPY) FOR ANALYSIS

def analyze\_single\_session(indir, window\_size, pre\_window\_size):

tempfiles = next(os.walk(indir))[2]

npyfiles = [f for f in tempfiles if os.path.splitext(f)[1]=='.npy' and 'raw\_averaged' in f]

matfiles = [f for f in tempfiles if os.path.splitext(f)[1]=='.mat']

if len(npyfiles) > 1:

npyfile = [f for f in tempfiles if os.path.splitext(f)[1]=='.npy' and 'raw\_averaged' in f and not 'part2' in f and not 'part3' in f]

npyfile = npyfile [0]

matfile = [f for f in tempfiles if os.path.splitext(f)[1]=='.mat' and not 'results' in f and not 'part2' in f and not 'part3' in f]

matfile = matfile [0]

else:

npyfile = npyfiles[0]

matfile = matfiles[0]

signals = np.squeeze(np.load(os.path.join(indir, npyfile)))

numrois = signals.shape[0]

behaviordata = sio.loadmat(os.path.join(indir, matfile))

eventlog = np.squeeze(behaviordata['eventlog'])

lastframe\_timestamp\_part1 = np.max(eventlog)

### DEFINES CERTAIN INPUTS IN EVENTLOG AS BEHAVIORAL STIMULI (LEVER PRESS, CUE, ETC)

activelever = eventlog[eventlog[:,0]==22,1]

activelevertimeout = eventlog[eventlog[:,0]==222,1]

cues = eventlog[eventlog[:,0]==7,1]

infusions = eventlog[eventlog[:,0]==4,1]

signals /= np.mean(signals, axis=1)[:, None]

signalsT = signals.T

frame\_timestamps = fix\_any\_dropped\_frames(eventlog[eventlog[:,0]==9,1],signals.shape[1])

frame\_timestamps = frame\_timestamps[::frameaveraging] ###incorporates averaging into timestamp array

### DISCARDS BEHAVIORAL EVENTS THAT WERE NOT FULLY MONITORED WITH IMAGING

if signals.shape[1] > frame\_timestamps.shape[0]:

signals = signals[:,:frame\_timestamps.shape[0]-1] ###cuts signals so it's not longer than the frame timestamps

final\_frame\_timestamp = frame\_timestamps[signals.shape[1]] #This is the timestamp of the final frame in milliseconds

activelever = activelever[activelever<(final\_frame\_timestamp-(window\_size/framerate\*1000))]

activelevertimeout = activelevertimeout[activelevertimeout<(final\_frame\_timestamp-(window\_size/framerate\*1000))]

### COMBINES ALL ACTIVE PRESSES ###

activeleverall = np.hstack((activelever, activelevertimeout))

### ALIGNS EVENTS (I.E. lEVER PRESSES, INFUSIONS) TO FRAMES

def calculate\_aligneddata\_forevent(data, frame\_after\_event):

framenumberfor\_eventofinterest = np.squeeze(framenumberforevent(frame\_after\_event, frame\_timestamps))

numtrials = framenumberfor\_eventofinterest.shape[0]

align = np.NAN\*np.zeros([numtrials,window\_size,numrois])

align\_to\_plot = np.NAN\*np.zeros([numtrials,window\_size,numrois])

temp = data

prevendindex = 0

for i in range(numtrials):

tempindex = framenumberfor\_eventofinterest[i]

tempindex = int(tempindex)

tempstartindex = np.amin([pre\_window\_size, tempindex]).astype(int)

startindex = np.amin([tempstartindex, tempindex-prevendindex]).astype(int)

tempendindex = np.amin([len(frame\_timestamps)-tempindex, post\_window\_size])

endindex = tempendindex.astype(int)

prevendindex = tempindex+endindex

align\_to\_plot[i,pre\_window\_size-startindex:pre\_window\_size+endindex,:] \

= temp[tempindex-startindex:tempindex+endindex,:]

align[i,pre\_window\_size-tempstartindex:pre\_window\_size+endindex,:] \

= temp[tempindex-tempstartindex:tempindex+endindex,:]

align\_to\_plot = align\_to\_plot[np.where(np.isfinite(align\_to\_plot[:,0,0]))[0],:,:]

return align, align\_to\_plot, framenumberfor\_eventofinterest

align\_activelever, align\_to\_plot\_activelever, framenumberfor\_frameafter\_activelever = \

calculate\_aligneddata\_forevent(signalsT, activeleverall) ###CHANGE LAST VARIABLE TO CHANGE EXTRACTED INFO

populationdata\_activelever = np.nanmean(align\_activelever, axis=0).T-1

return populationdata\_activelever, populationdata\_activelever, align\_activelever, align\_activelever, framerate

### IDENTIFIES THE FRAME NUMBER FOR BEHAVIORAL EVENTS

def framenumberforevent(event, frame\_timestamps):

framenumber = np.nan\*np.zeros(event.shape)

for ie, e in enumerate(event):

if np.isnan(e):

framenumber[ie] = np.nan

else:

temp = np.nonzero(frame\_timestamps<=e)[0]

if temp.shape[0]>0:

framenumber[ie] = np.nonzero(frame\_timestamps<=e)[0][-1]

else:

framenumber[ie] = 0

return framenumber

### ADDS FRAME TIMESTAMPS IF ANY TIMESTAMP WAS MISSED BY THE BEHAVIORAL COMPUTER. DOES SO BY INTERPOLATING TIMES

def fix\_any\_dropped\_frames(frame\_timestamps, numberofframes):

first\_frame = np.array([0])

last\_frame = np.array([int(np.max(frame\_timestamps)+(500\*timebetweenframes))])

frame\_index\_temp = np.concatenate((first\_frame,frame\_timestamps, last\_frame)) ###adds frame to timepoint '0' and an extra 500 frames at end

frames\_missed = [] ###creates empty list for us to add timestamps for missed frames

for i in range(len(frame\_index\_temp)-1): ###iterates through each collected frame

numframes\_missed = int(np.round((frame\_index\_temp[i+1]-frame\_index\_temp[i])\

/timebetweenframes)-1) ### number of missed frames per frame interval

if numframes\_missed > 0:

for j in range(numframes\_missed):

frame\_missed = np.array([frame\_index\_temp[i] + (int(timebetweenframes \* (j+1)))])

frames\_missed = np.concatenate((frames\_missed, frame\_missed))

corrected\_frame\_index = np.array(sorted(np.concatenate((frame\_index\_temp, frames\_missed))))

return corrected\_frame\_index

#### THIS SECTION IS FOR EXTRACTING ACTIVITY AROUND EACH LEVER PRESS####

### INSERT YOUR OWN DIRECTORY HERE

basedir = r"YOUR FOLDER DIRECTORY HERE"

###INSERT FOLDER TO ANALYZE

doibasedir = r"YOUR SUB-FOLDER DIRECTORY HERE FOR EACH DAY THAT YOU WANT TO ANALYZE"

#Insert file path for the day of interest (DOI)

print(doibasedir)

### ALL MICE OF INTEREST - THESE SHOULD EACH BE SUB-FOLDERS OF YOUR "doibasedir"

animals\_of\_interest = ['Sample\_Mouse1','Sample\_Mouse2','Sample\_Mouse3','Sample\_Mouse4','Sample\_Mouse5','Sample\_Mouse6']

### PICK THE FRAMES THAT YOU WANT TO BE PLOTTED

pre\_window\_size = int(10\*framerate) #How many frames per trial before origin to be plotted?

window\_size = int((pre\_window\_size\*2)+(3\*framerate)) #How many frames do you want to plot around the origin?

post\_window\_size = window\_size - pre\_window\_size

baselinefirstframe = 0

baselinelastframe = int(1\*framerate)

infusionframe = int(pre\_window\_size+(3\*framerate))

### FIGURE SETTINGS

fig, axs = plt.subplots(2, 1, figsize=(4, 6))

sns.set\_style('white')

cmax = .1

cmin = -cmax

ymax = .1

ymin = -ymax

### EXTRACTS AND PLOTS DATA FOR EACH ANIMAL OF INTEREST

populationdata\_active\_population = np.nan\*np.ones((1, window\_size)) # The second \_population refers to population across all animals and FOVs

populationdata\_inactive\_population = np.nan\*np.ones((1, window\_size))

for animal in animals\_of\_interest:

FOVs = next(os.walk(os.path.join(doibasedir, animal)))[1]

for fov in sorted(FOVs):

print(animal, fov)

temp1, temp2, temp3, temp4, framerate = analyze\_single\_session(os.path.join(doibasedir, animal, fov),

window\_size, pre\_window\_size)

populationdata\_active\_population = np.vstack((populationdata\_active\_population, temp1))

numneurons=populationdata\_active\_population.shape[0]

populationdata\_active\_population = populationdata\_active\_population[1:,:]

baseline = np.mean(populationdata\_active\_population[:,baselinefirstframe:baselinelastframe], axis=1)

pop\_active = populationdata\_active\_population - baseline[:,None]

numrois = pop\_active.shape[0]

tempresponse = np.mean(pop\_active[:,pre\_window\_size:pre\_window\_size\*2], axis=1)

sortresponse = np.argsort(tempresponse)[::-1]

ax = axs[0]

sns.heatmap(pop\_active[sortresponse,:], ax=ax, vmax = cmax, vmin = cmin, linewidth=0, cmap=plt.get\_cmap('PRGn\_r'),

cbar\_kws={'label': '$\Delta$F/F'})

ax.grid(False)

ax.set\_title('Responses')

ax.set\_ylabel('Sorted Cell number')

ax.set\_xlabel('Time from Active Press (s)')

ax.plot([pre\_window\_size, pre\_window\_size],

[0, numrois], '--k', linewidth=1.5)

ax.plot([infusionframe, infusionframe],

[0, numrois], '--k', linewidth=1.5)

ax = axs[1]

ax.plot(np.mean(pop\_active, axis = 0))

ax.plot([pre\_window\_size, pre\_window\_size],

[-.03,.03],'--k', linewidth=1.5)

ax.plot([infusionframe, infusionframe],

[-.03,.03], '--k', linewidth=1.5)

ax.plot([0, window\_size],

[0,0], '--k', linewidth=0.5)

fig.tight\_layout()