Tutorial of NeuroRA Version 1.0.7.30

Updated by 2020-04-11

This Tutorial of NeuroRA provides information on how to use the NeuroRA including its easy-to-use functions.

Before you read it, you only need to spend a little time learning the basic Python syntax and this toolkit is easy to understand. In addition, it would be better if you are familiar with Python, especially the matrix operations based on NumPy.

If there is anything wrong, difficult to understand or having any useful advice during reading it, you can contact me (zitonglu1996@gmail.com) and I will be happy and thankful to know about it.

Written by **Zitong Lu**

Institute of Cognitive Neuroscience

East China Normal University

This tutorial consists of these parts:

Content

Part 1: Introduction

NueorRA is a Python toolbox for multimode neural data representational analysis.

Overview

Representational Similarity Analysis (RSA) has become a popular and effective method to measure the representation of multivariable neural activity in different modes.

NeuroRA is a novel and easy-to-use toolbox based on Python, which can do some works about RSA among nearly all kinds of neural data, including behavioral, EEG, MEG, fNIRS, ECoG, electrophysiological and fMRI data.

Installation

- *pip install NeuroRA*

Required Dependencies

Numpy: a fundamental package for scientific computing
Matplotlib: a Python 2D plotting library
NiBabel: a package prividing read +/- write access to some common medical and neuroimaging file formats

Nilearn: a Python module for fast and easy statistical learning on NeuroImaging data.

MNE-Python: a Python software for exploring, visualizing, and analyzing human neurophysiological data.

Paper

Lu, Z., & Ku, Y. *NeuroRA: A Python toolbox of representational analysis from multi-modal neural data.* (bioRxiv: https://doi.org/10.1101/2020.03.25.008086)

Part 2: Data Conversion

Type of Neural Bata	Data Conversion Scheme

fMRI	Use *Nibabel* (https://nipy.org/nibabel/) to load fMRI data.
	`import nibabel as nib`
	
	`fmrifilename = "demo.nii"   # the fmri data file name with full address`
	`data = nib.load(fmrifilename).get_fdata()   # load fMRI data as ndarray`
	`……`

EEG/MEG	Use *MATLAB EEGLab* (http://sccn.ucsd.edu/eeglab/) to do preprocessing and obtain .mat files, and use *Scipy* (https://www.scipy.org) to load EEG data (.mat).
	`import scipy.io as sio`
	
	`filename = "demo.mat"   # the EEG/MEG data file name with full address`
	`data = sio.loadmat(filename)["data"]   # load EEG/MEG data as ndarray`
	`……`
	Or use *MNE* (https://mne-tools.github.io) to do preprocessing and return *ndarray*-type data.

fNIRS	For raw data from device, use *Numpy* (http://www.numpy.org) to load fNIRS data (.txt or .csv).
	`import numpy as np`
	
	`txtfilename = "demo.txt"   # the fNIRS data file name with full address`
	`csvfilename = "demo.csv"`
	`data = np.loadtxt(txtfilename)   # load fNIRS data as ndarray`
	`data = np.loadtxt(csvfilename, delimiter, usecols, unpack)`
	`……`

ECoG/sEEG	Use *Brainstorm* (https://neuroimage.usc.edu/brainstorm/) to do preprocessing and obtain .mat files, and use Scipy to load ECoG data (.mat).

Electrophysiology	Use *pyABF* (https://github.com/swharden/pyABF) to load electrophysiology data (.abf).
	`import pyabf`
	
	`abf = pyabf.ABF("demo.abf")   # the electrophysiology data file name with full address`
	`abf.setSweep(sweepNumber, channel)   # access sweep data`
	`data = abf.sweepY   # get sweep data with sweepY`
	`……`

Two functions, *NumPy.reshape()* & *NumPy.transpose()*, are recommended for further data transformation	

Part 3: Calculate the Neural Pattern Similarity

Module: ***nps_cal.py***

◆ nps(data, time_win=5, sub_opt=0)

A function for calculating the neural pattern similarity

Parameters:

data:	*array*
	The neural data. The shape of bhv_data must be [2, n_subs, n_trials, n_chlas, n_ts]. 2 represent 2 different conditions. n_subs, n_trials, n_chls, n_ts represent the number of subjects, the number of trials, the number of channels & the number of time-points.
time_win:	*int*
	(Only when time_opt = 1, time_win works) The time-window for each calculation.
sub_opt:	*int (0 / 1)*
	Calculate the NPS for each subject or not. 1 or 0.

Returns:

nps:	*array*
	If sub_opt=0, return nps (shape: [n_chls, n_ts/time_win]). If sub_opt=1, return nps (shape: [n_subs, n_chls, n_ts/time_win]).

◆ nps_fmri(fmri_data, time_win=5, sub_opt=0)

A function for calculating the neural pattern similarity of fMRI data

Parameters:

fmri_data:	*array*
	The fmri data. The shape of fmri_data must be [n_cons, n_subs, nx, ny, nz]. n_cons, n_subs, nx, ny, nz represent the number of conditions, the number of subjects, the size of the

fMRI data.

ksize: *array / list*

ksize=[kx, ky, kz] represents that the calculation unit contains k1*k2*k3 voxels.

strides: *array / list*

strides=[sx, sy, sz] represents the moving steps along the x, y, z.

Returns:

nps: *array*

NPS. The shape of nps is [n_x, n_y, n_z, 2].

◆ nps_fmri_roi(fmri_data, mask_data)

A function for calculating the neural pattern similarity of fMRI data for ROI

Parameters:

fmri_data: *array*

The fmri data. The shape of fmri_data must be [n_cons, n_subs, nx, ny, nz]. n_cons, n_subs, nx, ny, nz represent the number of conditions, the number of subjects, the size of the fMRI data.

mask_data: *array*

The fmri data. The shape of mask_data must be [nx, ny, nz]. nx, ny, nz represent the size of the fMRI data.

Returns:

nps: *array*

NPS. The shape of nps is [2].

Part 4: Calculate the RDM

Module: ***rdm_cal.py***

◆ bhvRDM(bhv_data, sub_opt=0, data_opt=1)

A function for calculating the RDM based on behavioral data

Parameters:

bhv_data: *array*

The behavioral data. If data_opt=0, the shape of bhv_data must be [n_cons, n_subs]. n_cons, n_subs represent the number of conditions & the number of subjects.

sub_opt: *int (0 / 1)*

Calculate the RDM for each subject or not. 1 or 0.

data_opt: *int (0 / 1)*

If data_opt=1, each subject's each trial has a value of data. If data_opt=0, each subject has a value of data, here ignore the effect of trials.

Returns:

rdm/rdms: *array*

If sub_opt=0, return only one rdm (shape: [n_cons, n_cons]). If sub_opt=1, return rdms (shape: [n_subs, n_cons, n_cons]).

◆ eegRDM(EEG_data, time_win=5, sub_opt=0, chl_opt=0, time_opt=0)

A function for calculating the RDM based on EEG/MEG/fNIRS data

Parameters:

EEG_data: *array*

The EEG/MEG/fNIRS data. The shape of EEG_data must be [n_cons, n_subs, n_trials, n_chls, n_ts]. n_cons, n_subs, n_trials, n_chls, n_ts represent the number of conditions, subjects, trials, channels, frequencies and time-points.

time_win: *int*

(Only when time_opt = 1, time_win works) The time-window for each calculation.

sub_opt: *int (0 / 1)*

Calculate the RDM for each subject or not. 1 or 0.

chl_opt: *int (0 / 1)*

Calculate the RDM for each channel or not. 1 or 0.

time_opt: *int (0 / 1)*

Calculate the RDM for each time-point or not. 1 or 0.

Returns:

rdm/rdms: *array*

If sub_opt=0 and chl_opt=0 and time_opt=0, return only one rdm (shape: [n_cons, n_cons]).

If sub_opt=0 and chl_opt=0 and time_opt=1, return rdms (shape: [n_ts/time_win, n_cons, n_cons]).

If sub_opt=0 and chl_opt=1 and time_opt=0, return rdms (shape: [n_chls, n_cons, n_cons]).

If sub_opt=1 and chl_opt=0 and time_opt=0, return rdms (shape: [n_subs, n_cons, n_cons]).

If sub_opt=0 and chl_opt=1 and time_opt=1, return rdms (shape: [n_chls, n_ts/time_win, n_cons, n_cons]).

If sub_opt=1 and chl_opt=0 and time_opt=1, return rdms (shape: [n_subs, n_ts/time_win, n_cons, n_cons]).

If sub_opt=1 and chl_opt=1 and time_opt=0, return rdms (shape: [n_subs, n_chls, n_cons, n_cons]).

If sub_opt=1 and chl_opt=1 and time_opt=1, return rdms (shape: [n_subs, n_chls, n_ts/time_win, n_cons, n_cons]).

◆ ecogRDM(ele_data, time_win=5, opt=”all”)

A function for calculating the RDM based on ECoG/electrophysiological data

Parameters:

ele_data: *array*

The ECoG/electrophysiological data. The shape of ele_data must be [n_cons, n_trials, n_chls, n_ts]. n_cons, n_trials, n_chls, n_ts represent the number of conditions, trials, channels, frequencies and time-points.

time_win: *int*

The time-window for each calculation.

opt: *string ("channels" or "time" or "all")*

Calculate the RDM for each channel or for each time-point or not. "channels" or "time" or "all".

Returns:

rdm/rdms: *array*

If opt="channels", return rdms (shape: [n_chls, n_cons, n_cons]).

If opt="time", return rdms (shape: [n_ts/time_win, n_cons, n_cons])

◆ fmriRDM(fmri_data, ksize=[3, 3, 3], strides=[1, 1, 1])

A function for calculating the RDM based on fMRI data

Parameters:

fmri_data: *array*

The fmri data. The shape of fmri_data must be [n_cons, n_subs, nx, ny, nz]. n_cons, n_subs, nx, ny, nz represent the number of conditions, the number of subjects, the size of the fMRI data.

ksize: *array / list*

ksize=[kx, ky, kz] represents that the calculation unit contains k1*k2*k3 voxels.

strides: *array / list*

strides=[sx, sy, sz] represents the moving steps along the x, y, z.

Returns:

rdms: *array*

Return rdms for each calculation unit. The shape of rdms is [n_x, n_y, n_z, n_cons, n_cons]. Here, n_x, n_y, n_z represent the number of calculation units along the x, y, z.

◆ fmriRDM_roi(fmri_data, mask_data)

A function for calculating the RDM based on fMRI data of a ROI

Parameters:

fmri_data: *array*

The fmri data. The shape of fmri_data must be [n_cons, n_subs, nx, ny, nz]. n_cons, n_subs, nx, ny, nz represent the number of conditions, the number of subjects, the size of the fMRI data.

mask_data: *array*

The mask data. The shape of mask_data must be [nx, ny, nz].

Returns:

rdm: *array*

Return the rdm for the ROI. The shape of rdm is [n_cons, n_cons]

Part 5: Calculate the Correlation Coefficient

Module: ***rdm_corr.py***

◆ rdm_correlation_spearman(RDM1, RDM2, rescale=False)

A function for calculating the Spearman correlation coefficient between two RDMs

Parameters:

RDM1:	*array*
	The shape of RDM1 must be [n_cons, n_cons].
RDM2:	*array*
	The shape of RDM2 must be [n_cons, n_cons].
rescale:	*Boolean (True or False)*
	Rescale the values in RDM or not. True or False.

Returns:

corr:	*array*
	The corr contains two values: correlation coefficient and p-value. The shape of corr is [2,].

◆ rdm_correlation_pearson(RDM1, RDM2, rescale=False)

A function for calculating the Pearson correlation coefficient between two RDMs

Parameters:

RDM1:	*array*
	The shape of RDM1 must be [n_cons, n_cons].
RDM2:	*array*
	The shape of RDM2 must be [n_cons, n_cons].
rescale:	*Boolean (True or False)*

Rescale the values in RDM or not. True or False.

Returns:

corr:	*array*
	The corr contains two values: correlation coefficient and p-value. The shape of corr is [2,].

◆ rdm_correlation_kendall(RDM1, RDM2, rescale=False)

A function for calculating the Kendalls tau correlation coefficient between two RDMs

Parameters:

RDM1:	*array*
	The shape of RDM1 must be [n_cons, n_cons].
RDM2:	*array*
	The shape of RDM2 must be [n_cons, n_cons].
rescale:	*Boolean (True or False)*
	Rescale the values in RDM or not. True or False.

Returns:

corr:	*array*
	The corr contains two values: correlation coefficient and p-value. The shape of corr is [2,].

◆ rdm_similarity(RDM1, RDM2, rescale=False)

A function for calculating the Cosine Similarity between two RDMs

Parameters:

RDM1:	*array*
	The shape of RDM1 must be [n_cons, n_cons].
RDM2:	*array*
	The shape of RDM2 must be [n_cons, n_cons].

Returns:

similarity: *float*

The Cosine Similarity.

◆ rdm_distance(RDM1, RDM2, rescale=False)

A function for calculating the Euclidean Distances between two RDMs

Parameters:

RDM1: *array*

The shape of RDM1 must be [n_cons, n_cons].

RDM2: *array*

The shape of RDM2 must be [n_cons, n_cons].

rescale: *Boolean (True or False)*

Rescale the values in RDM or not. True or False.

Returns:

dist: *float*

The Euclidean Distance.

◆ rdm_permutation (RDM1, RDM2, iter=1000)

A function for permutation test between two RDMs

Parameters:

RDM1: *array*

The shape of RDM1 must be [n_cons, n_cons].

RDM2: *array*

The shape of RDM2 must be [n_cons, n_cons].

iter: *int*

The number of iterations.

Returns:

p: *float*

The p-value.

Module: ***corr_cal.py***

◆ bhvANDeeg_corr(bhv_data, eeg_data, sub_opt=0, bhv_data_opt=1, chl_opt=0, time_opt=0, time_win=5, method="spearman", rescale=False)

A function for calculating the Similarity/Correlation Coefficient between behavioral data and EEG/MEG/fNIRS data

Parameters:

bhv_data: *array*

The behavioral data. If data_opt=0, the shape of bhv_data must be [n_cons, n_subs]. n_cons, n_subs represent the number of conditions & the number of subjects.

eeg_data: *array*

The EEG/MEG/fNIRS data. The shape of eeg_data must be [n_cons, n_subs, n_trials, n_chls, n_ts]. n_cons, n_subs, n_trials, n_chls, n_ts represent the number of conditions, subjects, trials, channels, frequencies and time-points.

sub_opt: *int (0 / 1)*

Calculate the RDM for each subject or not. 1 or 0.

bhv_data_opt: *int (0 / 1)*

If bhv_data_opt=1, each subject's each trial has a value of data. If bhv_data_opt=0, each subject has a value of data, here ignore the effect of trials.

time_win: *int*

The time-window for each calculation.

chl_opt: *int (0 / 1)*

Calculate the RDM for each channel or not. 1 or 0.

time_opt: *int (0 / 1)*

Calculate the RDM for each time-point or not. 1 or 0.

method: *string ("spearman" or "pearson" or "kendall" or "similarity"* *or "distance")*

They correspond to different methods of calculation. "spearman" or "pearson" or "kendall" or "similarity" or "distance".

rescale: *Boolean (True or False)*

Rescale the values in RDM or not. True or False.

Returns:

corr/corrs: *array*

The correlation coefficients corresponding to the RDMs.

◆ bhvANDecog_corr(bhv_data, ele_data, time_win=5, ecog_opt="allin", method="spearman" , rescale=False)

A function for calculating the Similarity/Correlation Coefficient between behavioral data and ECoG/electricophysiological data

Parameters:

bhv_data: *array*

The behavioral data. If data_opt=0, the shape of bhv_data must be [n_cons, n_subs]. n_cons, n_subs represent the number of conditions & the number of subjects.

ele_data: *array*

The ECoG/electrophysiological data. The shape of ele_data must be [n_cons, n_trials, n_chls, n_ts]. n_cons, n_trials, n_chls, n_ts represent the number of conditions, trials, channels, frequencies and time-points.

time_win: *int*

The time-window for each calculation.

opt: *string ("channels" or "time" or "all")*

Calculate the RDM for each channel or for each time-point or not. "channels" or "time" or "all".

method: *string ("spearman" or "pearson" or "kendall" or "similarity" or "distance")*

They correspond to different methods of calculation. "spearman" or "pearson" or "kendall" or "similarity" or "distance".

rescale: *Boolean (True or False)*

Rescale the values in RDM or not. True or False.

Returns:

corr/corrs: *array*

The correlation coefficients corresponding to the RDMs.

◆ bhvANDfmri_corr(bhv_data, fmri_data, bhv_data_opt=1, ksize=[3, 3, 3], strides=[1, 1, 1], method="spearman" , rescale=False)

A function for calculating the Similarity/Correlation Coefficient between behavioral data and fMRI data

Parameters:

bhv_data: *array*

The behavioral data. If data_opt=0, the shape of bhv_data must be [n_cons, n_subs]. n_cons, n_subs represent the number of conditions & the number of subjects.

fmri_data: *array*

The fmri data. The shape of fmri_data must be [n_cons, n_subs, nx, ny, nz]. n_cons, n_subs, nx, ny, nz represent the number of conditions, the number of subjects, the size of the fMRI data.

bhv_data_opt: *int (0 / 1)*

If bhv_data_opt=1, each subject's each trial has a value of data. If bhv_data_opt=0, each subject has a value of data, here ignore the effect of trials.

ksize: *array / list*

ksize=[kx, ky, kz] represents that the calculation unit contains k1*k2*k3 voxels.

strides: *array / list*

strides=[sx, sy, sz] represents the moving steps along the x, y, z.

method: *string ("spearman" or "pearson" or "kendall" or "similarity" or "distance")*

They correspond to different methods of calculation. "spearman" or "pearson" or "kendall" or "similarity" or "distance".

rescale: *Boolean (True or False)*

Rescale the values in RDM or not. True or False.

Returns:

corrs: *array*

The correlation coefficients corresponding to the RDMs. The shape of corrs is [n_x, n_y, n_z, 2].

◆ eegANDfmri_corr(eeg_data, fmri_data, chl_opt=0, ksize=[3, 3, 3], strides=[1, 1, 1], method="spearman" , rescale=False)

A function for calculating the Similarity/Correlation Coefficient between EEG/MEG/fNIRS data and fMRI data

Parameters:

eeg_data: *array*

The EEG/MEG/fNIRS data. The shape of eeg_data must be [n_cons, n_subs, n_trials, n_chls, n_ts]. n_cons, n_subs, n_trials, n_chls, n_ts represent the number of conditions, subjects, trials, channels, frequencies and time-points.

fmri_data: *array*

The fmri data. The shape of fmri_data must be [n_cons, n_subs, nx, ny, nz]. n_cons, n_subs, nx, ny, nz represent the number of conditions, the number of subjects, the size of the fMRI data.

chl_opt: *int (0 / 1)*

Calculate the RDM for each channel or not. 1 or 0.

ksize: *array / list*

ksize=[kx, ky, kz] represents that the calculation unit contains k1*k2*k3 voxels.

strides: *array / list*

strides=[sx, sy, sz] represents the moving steps along the x, y, z.

method: *string (“spearman” or “pearson” or “kendall” or “similarity” or “distance”)*

They correspond to different methods of calculation. “spearman” or “pearson” or “kendall” or “similarity” or “distance”.

rescale: *Boolean (True or False)*

Rescale the values in RDM or not. True or False.

Returns:

corrs: *array*

The correlation coefficients corresponding to the RDMs.

Module: ***corr_cal_by_rdm.py***

◆ eegrdms_corr(demo_rdm, EEG_rdms, method=”spearman” , rescale=False)

A function for calculating the Similarity/Correlation Coefficient between RDMs based on EEG/MEG/fNIRS data and a demo RDM

Parameters:

demo_rdm: *array*

The shape must be [n_cons, n_cons].

EEG_rdm: *array*

The shape must be [n_ts, n_cons, n_cons] or [n_chls, n_cons, n_cons].

method: *string (“spearman” or “pearson” or “kendall” or “similarity” or “distance”)*

They correspond to different methods of calculation. “spearman” or “pearson” or “kendall” or “similarity” or “distance”.

rescale: *Boolean (True or False)*

Rescale the values in RDM or not. True or False.

Returns:

corrs: *array*

The correlation coefficients corresponding to the RDMs.

◆ fmrirdms_corr(demo_rdm, fMRI_rdms, method=”spearman” , rescale=False)

A function for calculating the Similarity/Correlation Coefficient between RDMs based on fMRI data and a demo RDM

Parameters:

demo_rdm: *array*

The shape must be [n_cons, n_cons].

fmri_rdm: *array*

The shape must be [n_x, n_y, n_z, n_cons, n_cons].

method: *string (“spearman” or “pearson” or “kendall” or “similarity” or “distance”)*

They correspond to different methods of calculation. “spearman” or “pearson” or “kendall” or “similarity” or “distance”.

rescale: *Boolean (True or False)*

Rescale the values in RDM or not. True or False.

Returns:

corrs: *array*

The correlation coefficients corresponding to the RDMs. The shape of corrs is [n_x, n_y, n_z, 2].

Part 6: Save as a NIfTI file (for fMRI)

Module: ***corr_to_nii.py***

◆ corr_save_nii(corrs, filename, corr_mask=None, affine, size=[60, 60, 60], ksize=[3, 3, 3], strides=[1, 1, 1], p=1, r=0, similarity=0, distance=0, correct_method=None, correct_n=27, plotrlt=True, img_background=None)

A function for saving the correlation coefficients as a .nii file

Parameters:

corrs: *array*

corrs represent the correlation coefficients. Its shape must be [n_x, n_y, n_z, 2].

filename: *string*

The filename of the NIfTI file. Don't need a suffix.

corr_mask: *Niimg-like object or the filename*

A file for correcting the RSA result. It can just be one of your fMRI data file in your experiment.

affine: *array / list*

An affine array that tells you the position of the image array data in a reference space.

size: *array / list*

size=[x, y, z] represents that the size of the original data.

ksize: *array / list*

ksize=[kx, ky, kz] represents that the calculation unit contains k1*k2*k3 voxels.

strides: *array / list*

strides=[sx, sy, sz] represents the moving steps along the x, y, z.

p, r, similarity, distance: *float*

They represent the threshold value for calculation.

correct_method: *None / 'FWE' / 'FDR'*

The method for correction.

correct_n: *int*

The number of voxels used in correction.

plotrlt: *Boolean (True or False)*

Plot the RSA result or not.

img_background: *Niimg-like object or the filename*

The background image that the ROI/mask will be plotted on top of. If there is no special background requirement, set it as None.

Returns:

img_nii: *array*

The matrix form of the NIfTI file.

Part 7: Visualization for Results

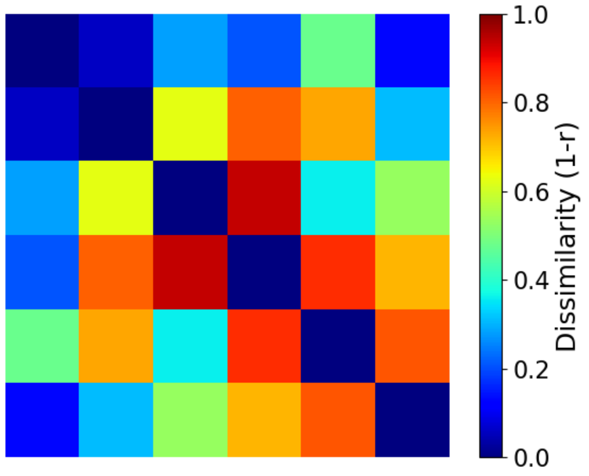
Module: ***rsa_plot.py***

◆ plot_rdm(rdm, rescale=False, conditions=None, con_fontsize=12, cmap=None)

A function for plotting the RDM

Parameters:

rdm:	*array / list*
	A representational dissimilarity matrix.
rescale:	*Boolean (True or False)*
	Rescale the values in RDM or not. True or False.
conditions:	*None / string list / string array*
	The labels of the conditions.
con_fontsize:	*int*
	Font size of the condition-labels.
cmap:	*Matplotlib colormap / string*
	The colormap of the figure.

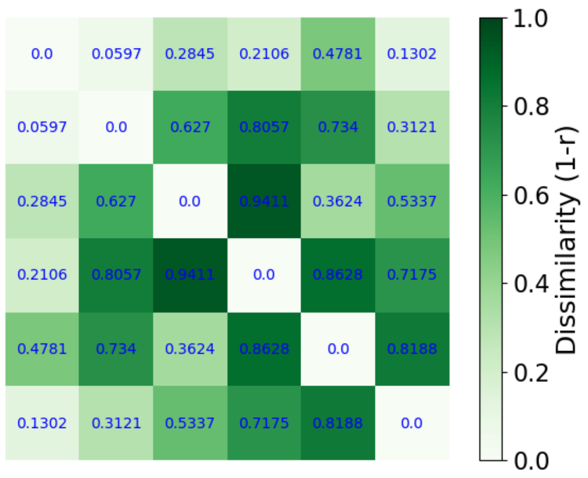


◆ plot_rdm_withvalue(rdm, fontsize=10, conditions=None, con_fontsize=12, cmap=None)

A function for plotting the RDM with visible values

Parameters:

rdm:	*array / list*
	A representational dissimilarity matrix.
fontsize:	*int / float*
	Font size of the visible values
conditions:	*None / string list / string array*
	The labels of the conditions.
con_fontsize:	*int*
	Font size of the condition-labels.
cmap:	*Matplotlib colormap / string*
	The colormap of the figure.



◆ plot_corrs_by_time(corrs, labels=None, time_unit=[0, 1])

A function for plotting the correlation coefficients by time sequence

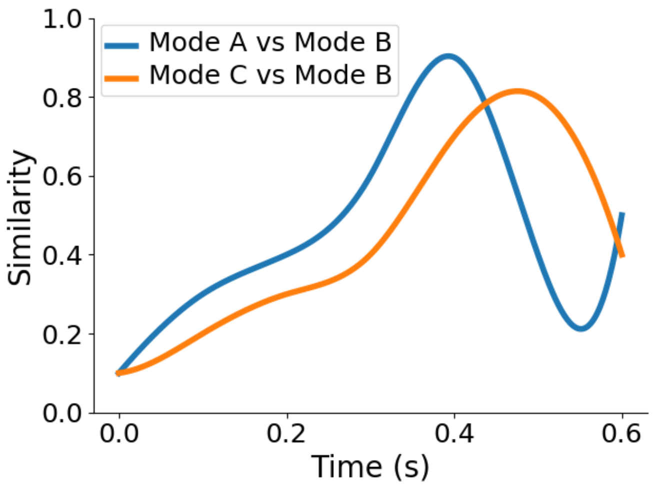
Parameters:

corrs:	*array*
	corrs represent the correlation coefficients point-by-point. Its shape must be [n_cons, ts, 2] or [n_cons, ts].
labels:	*None / string array / string list*

labels represent the names of conditions of RSA results.

time_unit: *array / list*

time_unit=[start_t, t_step]. Here, start_t represents the start time and t_step represents the time between two adjacent time-points.



◆ plot_corrs_hotmap(eegcorrs, chllabels=None, time_unit=[0, 0.1], lim=[0, 1], smooth=False, figsize=None, cmap=None)

A function for plotting the correlation coefficients by time sequence

Parameters:

eegcorrs: *array*

eegcorrs represent each channels' correlation coefficients point-by-point. Its shape must be [n_chls, ts, 2] or [n_chls, ts].

chllabels: *None / string array / string list*

labels represent the names of channels.

time_unit: *array / list*

time_unit=[start_t, t_step]. Here, start_t represents the start time and t_step represents the time between two adjacent time-points.

lim: *array / list*

lim=[lower, upper]. Here, min_r and max_r represent the upper limit and lower limit for plotting of the r-values.

smooth: *Boolean (True or False)*

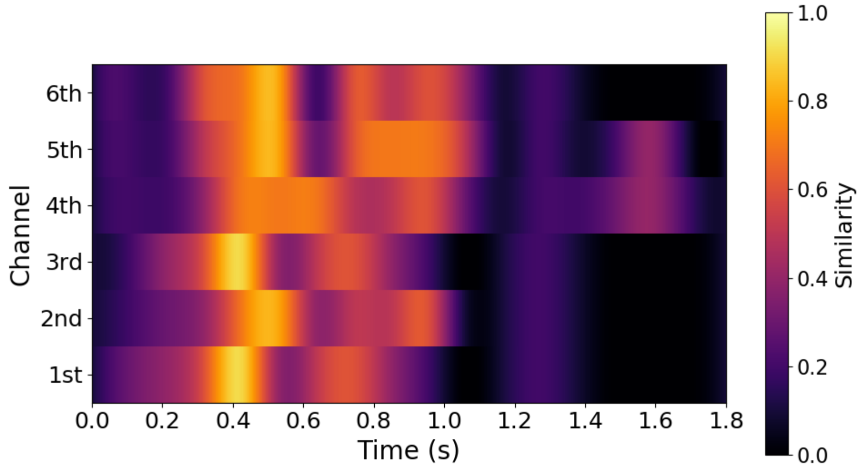
True or False represents smoothing the results or not.

figsize: *None / array / list*

The size of the Matplotlib figure. Such as figsize=[6.4, 4.8], if figsize=None, the size will be automatical.

cmap: *Matplotlib colormap / string*

The colormap of the figure.



- plot_nps_hotmap(similarities, chllabels=None, time_unit=[0, 0.1], lim=[0, 1], abs=False, smooth=False, figsize=None, cmap=None)

A function for plotting the correlation coefficients by time sequence

Parameters:

similarities: *array*

similarities represent correlation coefficients point-by-point. Its shape must be [n_chls, ts].

chllabels: *None / string array / string list*

labels represent the names of channels.

time_unit: *array / list*

time_unit=[start_t, t_step]. Here, start_t represents the start time and t_step represents the time between two adjacent time-points.

lim: *array / list*

lim=[lower, upper]. Here, min_r and max_r represent the upper limit and lower limit for plotting of the r-values.

abs: *Boolean (True or False)*

True or False represents changing the similarities into absolute values or not.

smooth: *Boolean (True or False)*

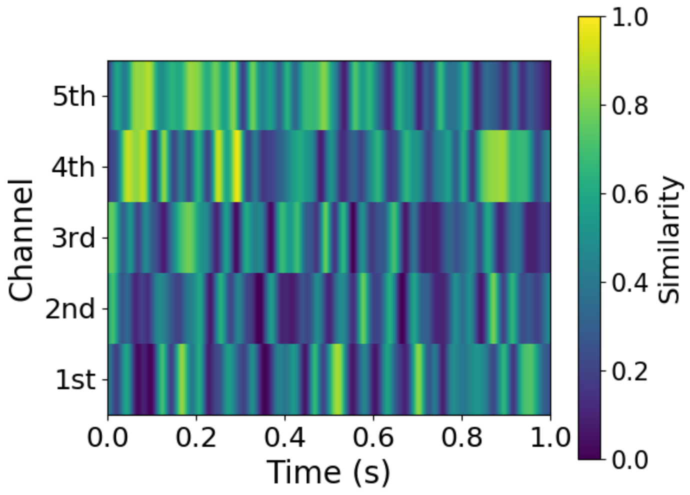
True or False represents smoothing the results or not.

figsize: *None / array / list*

The size of the Matplotlib figure. Such as figsize=[6.4, 4.8], if figsize=None, the size will be automatical.

cmap: *Matplotlib colormap / string*

The colormap of the figure.



◆ plot_brainrsa_region(img, threshold=None, background=get_bg_ch2())

A function for plotting the RSA-result regions by 3 cuts (frontal, axial, and lateral)

Parameters:

img: *Niimg-like object or the filename*

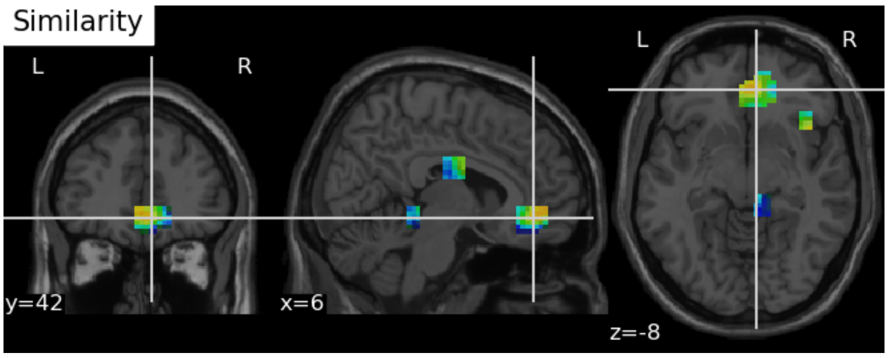
A 3-D image of the RSA result.

threshold: *None or int*

If it is an int, threshold is the number of voxels used in correction. Only the similarity clusters consisting more than threshold voxels will be visualized. If it is None, the threshold-correction won't work.

background: *Niimg-like object or the filename*

The background image that the ROI/mask will be plotted on top of.



◆ plot_brainrsa_montage (img, threshold=None, slice=[6, 6, 6], background=get_bg_ch2bet())

A function for plotting the RSA-result by different cuts

Parameters:

img: *Niimg-like object or the filename*

A 3-D image of the RSA result.

threshold: *None or int*

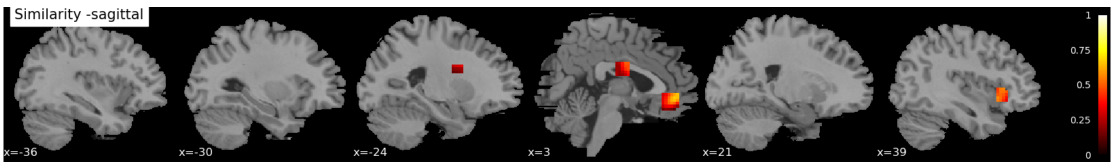
If it is an int, threshold is the number of voxels used in correction. Only the similarity clusters consisting more than threshold voxels will be visualized. If it is None, the threshold-correction won't work.

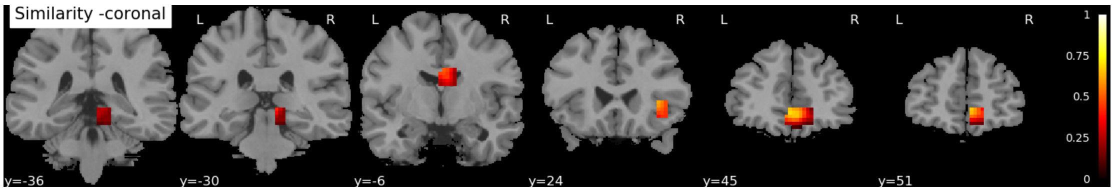
slice: *array*

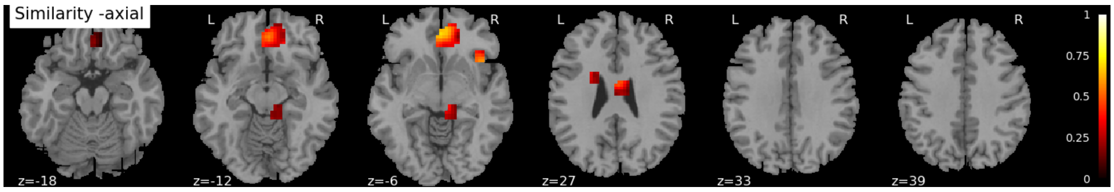
slice=[nx, ny, nz]. The nx, ny, nz is the number of cuts in the x, y, z directions.

background: *Niimg-like object or the filename*

The background image that the ROI/mask will be plotted on top of.





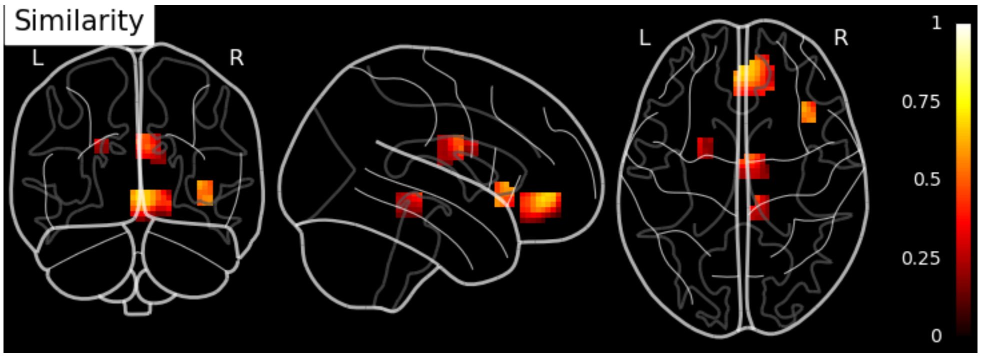


- plot_brainrsa_glass (img, threshold=None)

A function for plotting the 2-D projection of the RSA-result

Parameters:

img:	*Niimg-like object or the filename*
	A 3-D image of the RSA result.
threshold:	*None or int*
	If it is an int, threshold is the number of voxels used in correction. Only the similarity clusters consisting more than threshold voxels will be visualized. If it is None, the threshold-correction won't work.



- plot_brainrsa_surface (img, threshold=None)

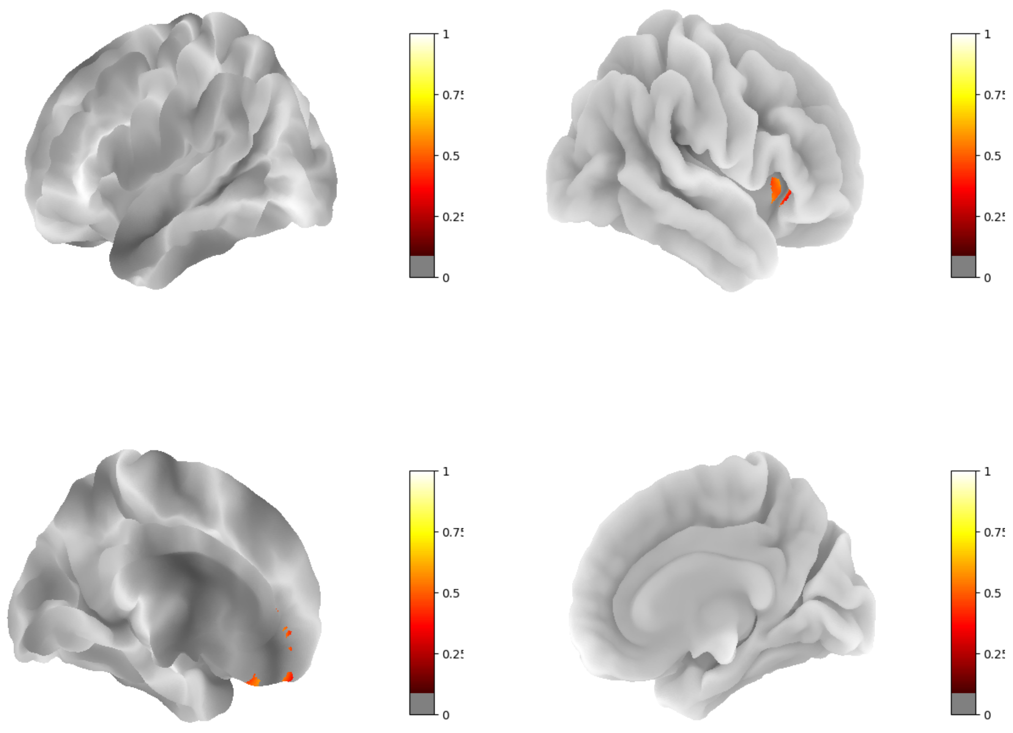
A function for plotting the RSA-result into a brain surface

Parameters:

img:	*Niimg-like object or the filename*
	A 3-D image of the RSA result.

threshold: *None or int*

If it is an int, threshold is the number of voxels used in correction. Only the similarity clusters consisting more than threshold voxels will be visualized. If it is None, the threshold-correction won't work.



◆ plot_brainrsa_rlts (img, threshold=None, slice=[6, 6, 6], background=None)

A function for plotting the RSA-result by a set of images

Parameters:

img: *Niimg-like object or the filename*

A 3-D image of the RSA result.

threshold: *None or int*

If it is an int, threshold is the number of voxels used in correction. Only the similarity clusters consisting more than threshold voxels will be visualized. If it is None, the threshold-correction won't work.

slice: *array*

slice=[nx, ny, nz]. The nx, ny, nz is the number of cuts in the x, y, z directions.

background:*Niimg-like object or the filename*

The background image that the ROI/mask will be plotted on top of. If there is no special background requirement, set it as None.

Part 8: Others

Module: ***stuff.py***

◆ limtozero(x)

A function for zeroing the value close to zero.

Parameters:

x:	*float*
	A value.

Returns:

0

◆ get_affine(file_name)

A function for getting the affine.

Parameters:

file_name:	*string*
	The file_name of a fMRI file.

Returns:

affine:	*array*
	An affine array that tells you the position of the image array data in a reference space.

◆ fwe_correct(p, size=[60, 60, 60], n=64)

A function for FWE correction.

Parameters:

p:	*array*
	A 3-D array of p-values, the number of p-value is the same as

the number of RSA calculation units in fMRI. Users can get p by corrs (Code: *p = corrs[:, :, :, 1]*).

size: *array*

size=[x, y, z] represents that the size of the original data.

n: *array*

The number of voxels used in correction.

Returns:

correctp: *array*

The FWE corrected p-values. A 3-D array of p-values, the number of p-value is the same as the number of RSA calculation units in fMRI.

◆ fdr_correct(p, size=[60, 60, 60], n=64)

A function for FDR correction.

Parameters:

p: *array*

A 3-D array of p-values, the number of p-value is the same as the number of RSA calculation units in fMRI. Users can get p by corrs (Code: *p = corrs[:, :, :, 1]*).

size: *array / list*

size=[x, y, z] represents that the size of the original data.

n: *int*

The number of voxels used in correction.

Returns:

correctp: *array*

The FDR corrected p-values. A 3-D array of p-values, the number of p-value is the same as the number of RSA calculation units in fMRI.

◆ correct_by_threshold(img, threshold)

A function for fMRI correction by threshold (the number of voxels).

Parameters:

img:	*array*
	A 3-D array of the RSA result.
threshold:	*n*
	The number of voxels used in correction. Only the similarity clusters consisting more than threshold voxels will be visualized.

Returns:

img:	*array*
	A 3-D array of the threshold-corrected RSA result.

◆ get_bg_ch2()

A function for getting the path of 'ch2.nii.gz'.

Returns:

path:	*string*
	The path of the file "ch2.nii.gz".

◆ get_bg_ch2bet()

A function for getting the path of 'ch2bet.nii.gz'.

Returns:

path:	*string*
	The path of the file "ch2bet.nii.gz".

◆ datamask(fmri_data, mask_data)

A function for filtering the data by a ROI mask.

Parameters:

fmri_data: *array*

The fmri data. The shape of fmri_data must be [nx, ny, nz]. nx, ny, nz represent the size of the fMRI data.

mask_data: *array*

The fmri data. The shape of mask_data must be [nx, ny, nz]. nx, ny, nz represent the size of the fMRI data.

Parameters:

newfmri_data: *array*

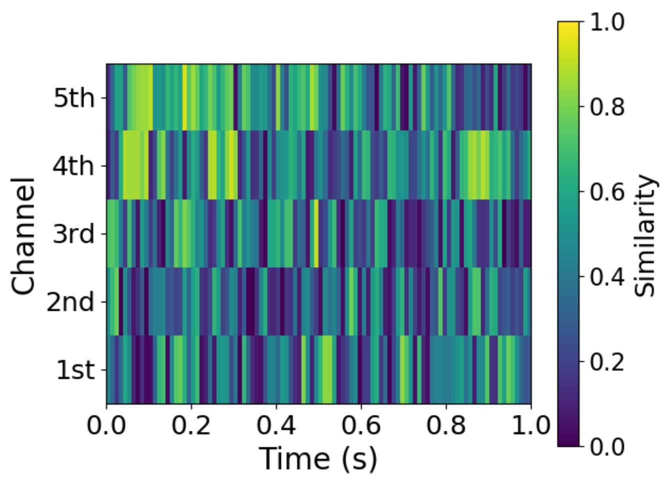
The new fmri data. The shape of fmri_data must be [nx, ny, nz]. nx, ny, nz represent the size of the fMRI data.

Part 9: Demo

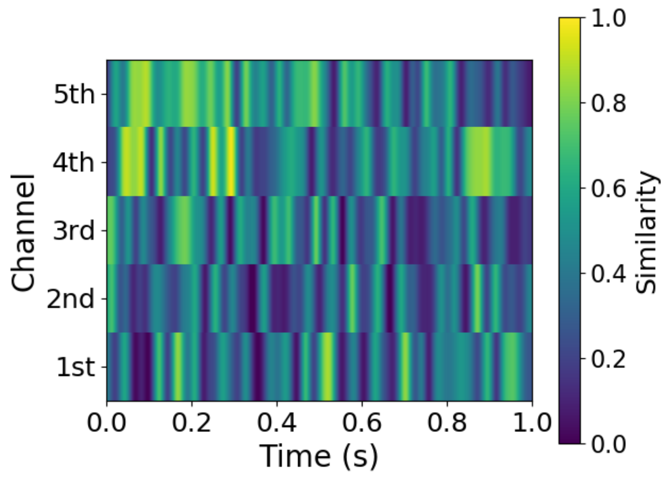
Here is a demo based on the publicly available visual-92-categories-task MEG datasets. (*Reference:* Cichy, R. M., Pantazis, D., & Oliva, A. "Resolving human object recognition in space and time." Nature neuroscience (2014): 17(3), 455-462.) MNE-Python has been used to load this dataset.

```
# -*- coding: utf-8 -*-

' a demo based on visual-92-categories-task MEG data '
# Users can learn how to use Neurora to do research based on EEG/MEG etc
data.

__author__ = 'Zitong Lu'

import numpy as np
import os.path as op
from pandas import read_csv
import mne
from mne.io import read_raw_fif
from mne.datasets import visual_92_categories
from neurora.nps_cal import nps
from neurora.rdm_cal import eegRDM
from neurora.rdm_corr import rdm_correlation_spearman
from neurora.corr_cal_by_rdm import rdms_corr
from neurora.rsa_plot import plot_rdm, plot_corrs_by_time,
plot_nps_hotmap, plot_corrs_hotmap

"""**********       Section 1: loading example data       **********"""
""" Here, we use MNE-Python toolbox for loading data and processing """
""" you can learn this process from MNE-Python (https://mne-
tools.github.io/stable/index.html) """

data_path = visual_92_categories.data_path()
fname = op.join(data_path, 'visual_stimuli.csv')
conds = read_csv(fname)
conditions = []
for c in conds.values:
    cond_tags = list(c[:2])
    cond_tags += [('not-' if i == 0 else '') + conds.columns[k]
```

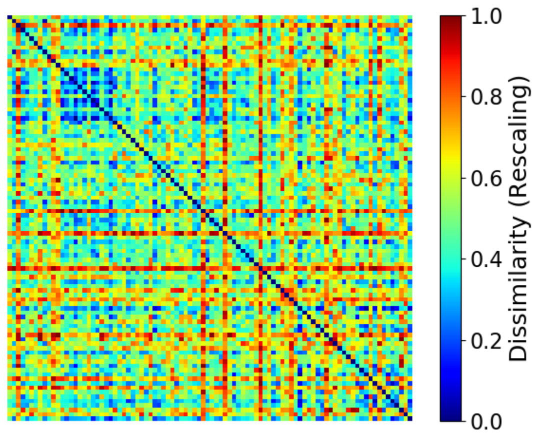
```
                   for k, i in enumerate(c[2:], 2)]
    conditions.append('/'.join(map(str, cond_tags)))
event_id = dict(zip(conditions, conds.trigger + 1))
print(event_id)
sub_id = [0, 1, 2]
megdata = np.zeros([3, 92, 306, 1101], dtype=np.float32)
subindex = 0
for id in sub_id:
    fname = op.join(data_path, 'sample_subject_'+str(id)+'_tsss_mc.fif')
    raw = read_raw_fif(fname)
    events = mne.find_events(raw, min_duration=.002)
    events = events[events[:, 2] <= 92]
    subdata = np.zeros([92, 306, 1101], dtype=np.float32)
    for i in range(92):
        epochs = mne.Epochs(raw, events=events, event_id=i + 1,
baseline=None,
                            tmin=-0.1, tmax=1, preload=True)
        data = epochs.average().data
        subdata[i] = data
    megdata[subindex] = subdata
    subindex = subindex + 1

# the shape of MEG data: megdata is [3, 92, 306, 1101]
# n_subs = 3, n_conditions = 92, n_channels = 306, n_timepoints = 1101
(-100ms to 1000ms)


"""**********       Section 2: Preprocessing       **********"""

# shape of megdata: [n_subs, n_cons, n_chls, n_ts] -> [n_cons, n_subs,
n_chls, n_ts]
megdata = np.transpose(megdata, (1, 0, 2, 3))

# shape of megdata: [n_cons, n_subs, n_chls, n_ts] -> [n_cons, n_subs,
n_trials, n_chls, n_ts]
# here data is averaged, so set n_trials = 1
megdata = np.reshape(megdata, [92, 3, 1, 306, 1101])


"""**********       Section 2: Calculating the neural pattern similarity
**********"""

# Get data under different condition
# Here we calculate the neural pattern similarity (NPS) between two
stimulus
# Seeing Humanface vs. Seeing Non-Humanface

# get data under "humanface" condtion
megdata_humanface = megdata[12:24]
# get data under "nonhumanface" condition
megdata_nonhumanface = megdata[36:48]

# Average the data
avg_megdata_humanface = np.average(megdata_humanface, axis=0)
avg_megdata_nonhumanface = np.average(megdata_nonhumanface, axis=0)

# Create NPS input data
# Here we extract the data from first 5 channels between 0ms and 1000ms
nps_data = np.zeros([2, 3, 1, 5, 1000]) # n_cons=2, n_subs=3, n_chls=5,
n_ts=1000
nps_data[0] = avg_megdata_humanface[:, :, :5, 100:1100] # the start time
of the data is -100ms
nps_data[1] = avg_megdata_nonhumanface[:, :, :5, 100:1100] # so 100:1200
corresponds 0ms-1000ms

# Calculate the NPS with a 10ms time-window
# (raw sampling requency is 1000Hz, so here
time_win=10ms/(1s/1000Hz)/1000=10)
nps = nps(nps_data, time_win=10)

# Plot the NPS results
plot_nps_hotmap(nps, time_unit=[0, 0.01], abs=True)
```



```
# Smooth the results and plot
plot_nps_hotmap(nps, time_unit=[0, 0.01], abs=True, smooth=True)
```

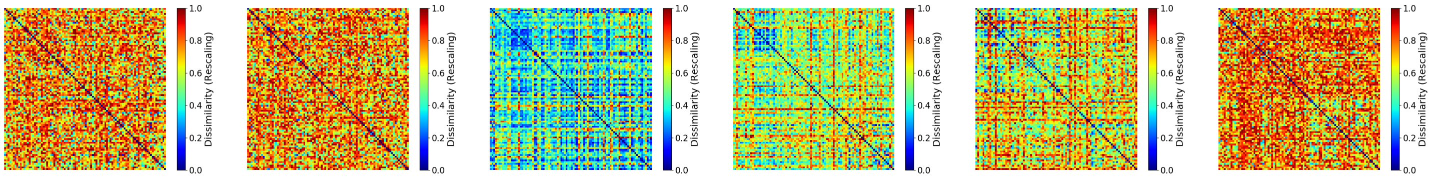


```
"""**********        Section 3: Calculating single RDM and Plotting
**********"""

# Calculate the RDM based on the data during 190ms-210ms
rdm = eegRDM(megdata[:, :, :, :, 290:310])

# Plot this RDM
plot_rdm(rdm, rescale=True)
```



```
"""**********        Section 4: Calculating RDMs and Plotting
**********"""

# Calculate the RDMs by a 10ms time-window
# (raw sampling requency is 1000Hz, so here
time_win=10ms/(1s/1000Hz)/1000=10)
```

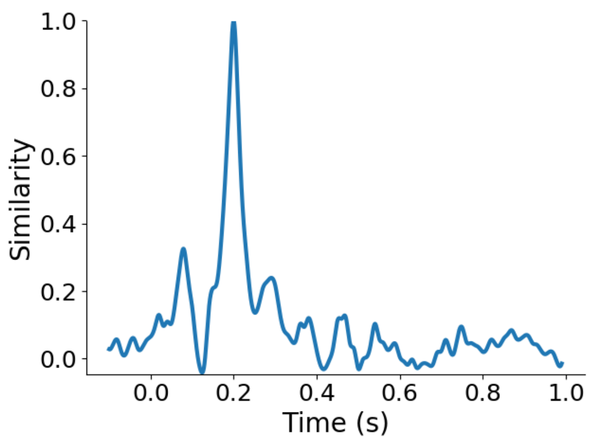
```
rdms = eegRDM(megdata, time_win=10, time_opt=1)

# Plot the RDM of 0ms, 50ms, 100ms, 150ms, 200ms
times = [0, 10, 20, 30, 40, 50]
for t in times:
    plot_rdm(rdms[t], rescale=True)
```



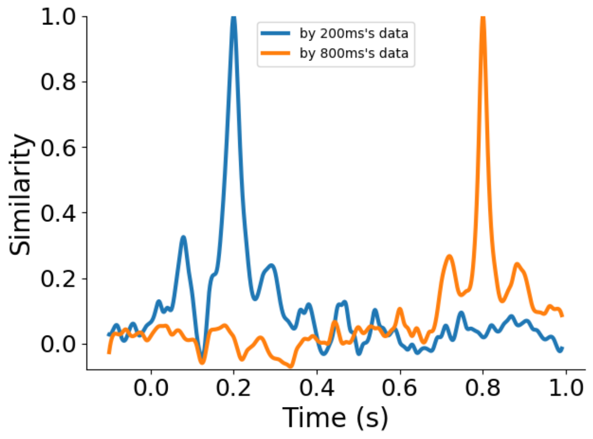
```
"""**********       Section 5: Calculating the Similarity between two
RDMs     **********"""

# RDM of 200ms
rdm_sample1 = rdms[30]
# RDM of 800ms
rdm_sample2 = rdms[90]

# calculate the correlation coefficient between these two RDMs
corr = rdm_correlation_spearman(rdm_sample1, rdm_sample2, rescale=True)
print(corr)
```

```
SpearmanrResult(correlation=0.02665680483550596, pvalue=0.08462337954774739)
```

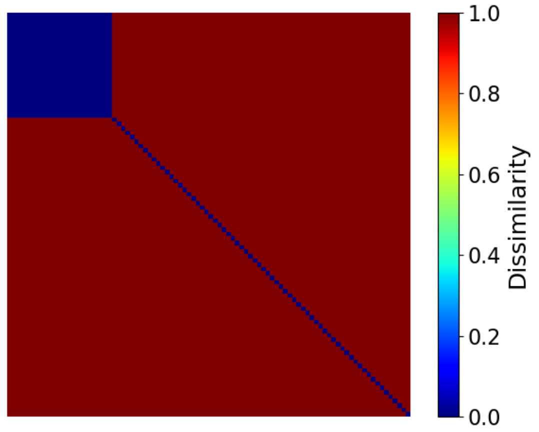
```
"""**********       Section 6: Calculating the Similarity and Plotting
**********"""

# Calculate the representational similarity between 200ms and all the
time points
corrs1 = rdms_corr(rdm_sample1, rdms)

# Plot the corrs1
corrs1 = np.reshape(corrs1, [1, 110, 2])
plot_corrs_by_time(corrs1, time_unit=[-0.1, 0.01])
```



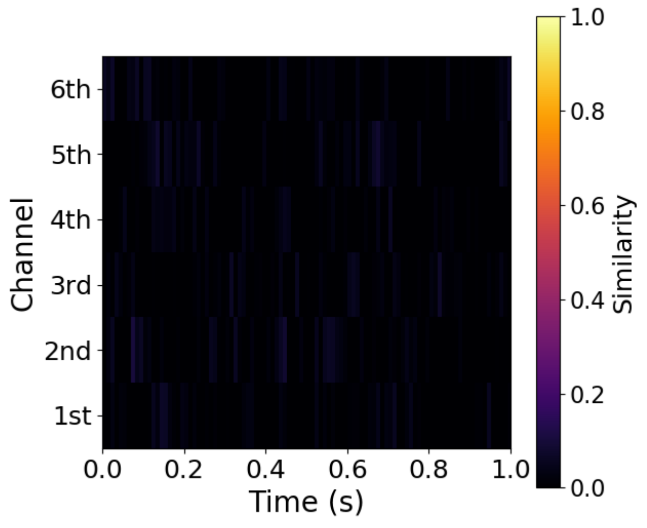
```
# Calculate and Plot multi-corrs
corrs2 = rdms_corr(rdm_sample2, rdms)
corrs = np.zeros([2, 110, 2])
corrs[0] = corrs1
corrs[1] = corrs2
labels = ["by 200ms's data", "by 800ms's data"]
plot_corrs_by_time(corrs, labels=labels, time_unit=[-0.1, 0.01])
```



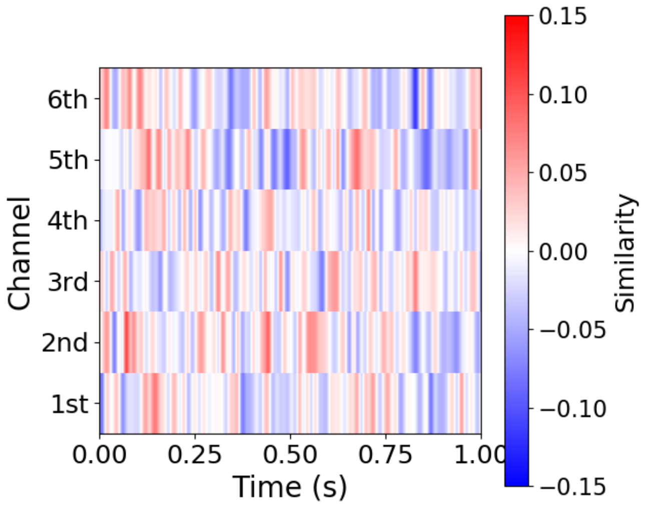
```
"""**********       Section 7: Calculating the RDMs for each channels
**********"""

# Calculate the RDMs for the first six channels by a 10ms time-window
between 0ms and 1000ms
rdms_chls = eegRDM(megdata[:, :, :, :6, 100:1100], chl_opt=1,
time_opt=1, time_win=10)

# Create a 'human-related' coding model RDM
model_rdm = np.ones([92, 92])
for i in range(92):
    for j in range(92):
        if (i < 24) and (j < 24):
            model_rdm[i, j] = 0
    model_rdm[i, i] = 0

# Plot this coding model RDM
plot_rdm(model_rdm)
```



```
# Calculate the representational similarity between the neural
activities and the coding model for each channel
corrs_chls = rdms_corr(model_rdm, rdms_chls)

# Plot the representational similarity results
plot_corrs_hotmap(corrs_chls, time_unit=[0, 0.01])
```



```
# Set more parameters and re-plot
plot_corrs_hotmap(corrs_chls, time_unit=[0, 0.01], lim=[-0.15, 0.15],
smooth=True, cmap='bwr')
```



6th
5th
4th
3rd
2nd
1st
Channel
0.00
0.25
0.50
0.75
1.00
Time (s)
0.15
0.10
0.05
0.00
−0.05
−0.10
−0.15
Similarity