ENGRAM

Release 0.0.2

Mar 25, 2020

Contents

Encoding Graphical Representations of Activated Memories (ENGRAM) is an open source Python package for developing cognitive neural prostheses.

CHAPTER 1

Getting Started

1.1 Installation

Get ENGRAM from pip:

```
pip install engram
```

1.2 Requirements

- **Python 3.7**
- Neo
- Tensorflow (**must install manually**)
- Numpy
- Scipy
- Visbrain

1.3 Key Features

- **Convert electrophysiology data from multiple brain regions into Engrams** using `engram.declarative`
- **Model multi-channel electrophysiology recordings** using multiple machine learning techniques (i.e. MIMO, CNN, RNN, etc) using `engram.procedural`
- **Visualize multi-input multi-output (MIMO) modeling** of electrophysiology recordings using `engram.episodic`
- Grow **artificial connections between functionally connected neurons**
- **Online data processing for OpenBCI headsets** using `engram.working`

CHAPTER 2

A Gentle Introduction to Neural Prostheses

Introduction This section will acquaint readers with concrete use cases of neural prostheses.

ENGRAM: The Tool This section will detail the background and features of ENGRAM.

How to Build a Memory Prosthesis This section is an extensive walkthrough of how to build a memory prosthesis: a specific application of cortical prostheses.

Conclusion This section will synthesize what we've learned into projections for the future of neural engineering.

Additional Resources This section gives additional resources for the curious to explore.

2.1 Introduction

Our minds function best unconsciously. Only when we become conscious, however, can we take ownership of their processes —— and control them.

> "Dementia is troubling because, at the same time as it erodes someone's memory, it also eats away at th[e] capacity to create shared meaning. If someone cannot remember not just where the milk bottle goes, but what a milk bottle is for, then the shared pre-suppositions on which communication, meaning, and identity depend become badly strained [Lea15].

Dr. Pauley is a dementia neurologist treating patients with anterograde amnesia (the inability to form new memories) as a result of Alzheimer's disease (AD), traumatic brain injury (TBI), and stroke.

2.1.1 Entry Vignette

To provide the reader with an inviting Introduction to the feel of the context in which the case takes place

Characters

The plan to provide narrative flow from the perspective of the user 1. **"Dr. Pauley": Dementia neurologist confronting concrete cases with new tools and unprecedented ethics.** 2. **Me: Addressed in asides to show my own growth** - Impetus: Neuromancer. To walk upon an electronic ground. - Ungeneralized Idea: Cognitive states (epi./phen.) are upheld by multiscale neural activity - Generalized Idea: Time and space change, but the ground retains memory. 3. **The CNE: Theodore Berger, Dong Song, Xiwei She** 4. **The Entrepreneurs:Bryan Johnson, Elon Musk** 5. **The Visionaries: Ed Boyden, Rajesh Rao** 6. **The Affected:Those with dementia** 7. **The Practiced:Epilepsy neurologists who already use RNS devices** 8. **The Concerned: FDA (regulators) and INS (ethicists)**

2.1.2 An Introduction

To familiarize the reader with the central features including rationale and research procedures

2.1.3 An Extensive Narrative Description

To of the case(s) and its context, which may involve historical or organizational information important for understanding the case

The Computational Basis of Memory Encoding Engrams are memory codes stored *someplace else* than the hippocampus.

2.1.4 Draw from Additional Data Sources

Integrate with the researcher's own interpretations of the issues and both confirming and disproving evidence are presented followed by the presentation of the overall case assertions

2.1.5 A Closing Vignette

As a way of cautoning the reader to the specific case context saying "I like to close on an experiential note, reminding the reader that this report is just one person's encounter with a complex case"

2.2 ENGRAM: The Tool

2.2.1 Definitions

Graphical Representations

Activated Memories

2.2.2 Origins and Early Visions

Theodore Berger

CNE From rats to primates to humans

What is a Cortical Prosthesis? The General Architecture Replacement parts for the brain must be (1) truly biomimetic, (2) network models, (3) bidirectional, and (4) adaptive, both to individual patients and their disease progression [BBB+01].

The core concepts & underlying technologies of our lab (ML/NC/CL-DBS)

Berger had the vision

Song had the math

You must outline the end-user

Ed Boyden

Neural Coprocessors

Rajesh Rao

BTBI

2.2.3 Core Features

Data Containers

ID: All data from a single individual - **Bin:** Binary data - **Cont:** Continuous data - **Events:** Event data

Signal Comparison Module

For use comparing (1) within individuals (i.e. between channels) or between multiple individuals - Rats vs humans signal quality

Mathematical Modeling Techniques

Minimal Dependencies - Classic Multi-Input Multi-Output (MIMO) Modeling - Classic Memory Decoding (An L1-regularized logistic regression model) - Closed Loop Hippocampal Prosthesis

2.2.4 Integration with Other Software Packages

Tensorflow - Deep MIMO and MD Models

Vispy/Visbrain - Novel visualization techniques

Brainflow - Online analysis of OpenBCI streams

ROOTS - Realistic neural growth between functionally connected sources

2.2.5 Ethical Considerations

Coming soon...

Note: Ethical concerns with neural prostheses should differ considerably from DBS, aDBS, and clDBS. This paper builds on existing models and literature on implantable neurological devices to distill unique ethical concerns associated with the design, development, and implementation of neural prostheses. In doing so, we hope that the resulting recommendations will be of use to guide this emerging field of neural engineering as it matures.

For instance, a recent review of the ethical issues related to neuroprosthetics, Walter Glannon questions whether a hippocampal prosthesis could be integrated into the brain's memory circuits to maintain important aspects of autobiographical memory, such as the interaction between emotional and episodic memory, selective meaning attribution, and place cell function (Glannon, 2016). In reference to case of neurodegenerative diseases such as Alzheimer's disease, Fabrice Jotterand has also pointed out that restoring psychological continuity (i.e. memory encoding) to patients would not repair the memories lost to neurodegeneration—-and that clinicians have an obligation to help restore the integrity of the patient's personal identity through a relational narrative with past events where memory had failed (Jotterand, 2019). As more generalizable conclusions are drawn about neural prostheses as a whole, however, a deeper understanding of the core technology behind these devices will be increasingly beneficial. Glannon: "A person with anterograde or retrograde amnesia for many years might have difficulty adjusting cognitively and emotionally to what could be a substantial change in the content of his mental states" (Glannon 2019, 164)].

In order to effectively design devices that intend to benefit disabled people, researchers must, as a matter of justice, begin to pay close attention to the actual needs and desires of their end-users (Goering & Klein, 2019). And what aspects of neural prostheses can UCD affect? [

Consider the following: 1. Identification of end users 2. Determination of timing and responsibility for end user engagement 3. Assessment of the significance of personal interactions with end users 4. Comparison of methods for obtaining end user views Principled considerations: 1. Specification of the values underlying BCI research (e.g., sophistication vs. accessibility) 2. Reflection on the ethical reasons to engage end user perspectives (Sullivan et al., 2018)] In order to be most effective, qualitative instruments should be used to account for potential phenomenological changes resulting from implanted devices, as well as patient preference information to inform later risk-benefit assessment (FDA, 2016; Gilbert et al., 2019).

In such cases, the role of scientists, clinicians, and engineers in risk assessment is to estimate the probability of a beneficial or adverse event based on data provided by sponsors or available in the published literature—-but patient input is what improves our estimates on the weight or importance of an event (Benz and Civillico, 2017).

2.3 How to Build a Memory Prosthesis

Coming soon...

2.4 Conclusion

2.4.1 A New Era of Open-Source Neuroscience

Coming soon...

2.4.2 Registries + Standardization: The Need for Speed

Coming soon...

2.5 Additional Resources

- CLARITY Technique (Karl Diesseroth)

Elephant (Electrophysiology Analysis Toolkit) is an emerging open-source, community centered library for the analysis of electrophysiological data in the Python programming language.

Neo is a Python package for working with electrophysiology data in Python, together with support for reading a wide range of neurophysiology file formats, including Spike2, NeuroExplorer, AlphaOmega, Axon, Blackrock, Plexon, Tdt, and support for writing to a subset of these formats plus non-proprietary formats including HDF5. [GGJ+14]

Neurotic is an app for Windows, macOS, and Linux that allows you to easily review and annotate your electrophysiology data and simultaneously captured video.

Ephyviewer is a Python library based on pyqtgraph for building custom viewers for electrophysiological signals, video, events, epochs, spike trains, data tables, and time-frequency representations of signals.

EEGLearn is a set of functions for supervised feature learning/classification of mental states from EEG based on "EEG images" idea. [BRYC16]

Wagner Lab is a memory lab at Stanford University that releases all of their code with extensive documentation and enough functionality to reproduce publication results. [GKM+18] [WFW17]

2.6 Glossary

2.6.1 E

Echphory

Engraphy

2.6.2 R

Redintegration

2.7 References

CHAPTER 3

API Reference Guide

3.1 engram.declarative package

3.1.1 Submodules

3.1.2 engram.declarative.engram module

This module defines *Engram*, the container for all offline analysis. It contains many Mneme objects that are:

- Labeled with their region of origin
- Organized into trial & channel subsections.

Each *Engram* has a unique event tag.

class engram.declarative.engram.**Engram**(*engram*, *id='User'*, *tag='Unspecified'*)
Bases: object

3.1.3 engram.declarative.id module

This module defines *ID*, the main container gathering all the data, whether discrete or continous, for a given recording session. It is the container for the Engram class.

class engram.declarative.id.**ID**(*name=None*, *extension=None*, *project=None*, *settings=None*, *load=False*)
Bases: object

Main container gathering all the data, whether discrete or continous, for a given recording session.

episode(*shader='engram'*)

load(*datadir='users'*)

loadEvents(*session=None*, *extension='.nex'*)

loadTrace(*method='name'*, *session=None*, *manual=None*, *regions=None*)

model(*method='channels'*, *model_type='CNN'*)

preprocess(*settings=None*)

save(*datadir='users'*)

3.1.4 engram.declarative.mneme module

This module defines *Mneme*, our smallest unit of memory.

It is the container for standardized features associated with an event.

class engram.declarative.mneme.**Mneme**(*id*, *raw=None*, *tag=None*, *settings=None*)
Bases: object

3.1.5 Module contents

:mod:'engram.declarative' provides classes for containing neurophysiology recordings and derivative features.

Classes from :mod:'engram.declarative' return nested data structures containing one or more classes from this module.

Classes:

class engram.declarative.**ID**(*name=None*, *extension=None*, *project=None*, *settings=None*, *load=False*)
Main container gathering all the data, whether discrete or continous, for a given recording session.

class engram.declarative.**Engram**(*engram*, *id='User'*, *tag='Unspecified'*)

class engram.declarative.**Mneme**(*id*, *raw=None*, *tag=None*, *settings=None*)

3.2 engram.procedural package

3.2.1 Submodules

3.2.2 engram.procedural.analyze module

3.2.3 engram.procedural.data module

engram.procedural.data.**events**(*feature*, *time*, *settings*, *prev_len*)

engram.procedural.data.**select**(*feature*, *time*, *settings*, *prev_len=None*)

engram.procedural.data.**trials**()

3.2.4 engram.procedural.events module

engram.procedural.events.**Neurogenesis**()

engram.procedural.events.**RAM**(*reader*)

engram.procedural.events.**select**(*name*, *reader*)

3.2.5 engram.procedural.features module

engram.procedural.features.**LFP**(*trace*, *settings*)

engram.procedural.features.**STFT**(*trace*, *settings*)

engram.procedural.features.**multiscale**(*trace*, *settings*)

engram.procedural.features.**normalize**(*data*, *settings*)

engram.procedural.features.**select**(*name*, *trace*, *settings*)

engram.procedural.features.**spikes**(*trace*, *settings*)

3.2.6 engram.procedural.filters module

engram.procedural.filters.**butter_bandpass**(*lowcut*, *highcut*, *fs*, *order=5*)

engram.procedural.filters.**butter_bandpass_filter**(*data*, *lowcut*, *highcut*, *fs*, *order=5*)

engram.procedural.filters.**butter_lowpass**(*cutoff*, *fs*, *order=5*)

engram.procedural.filters.**butter_lowpass_filter**(*data*, *cutoff*, *fs*, *order=5*)

engram.procedural.filters.**select**(*filter*, *data*, *min=0*, *max=None*, *fs=2000*, *order=5*)

3.2.7 engram.procedural.missingdata module

engram.procedural.missingdata.**interpolate_nans**(*y*)

Helper to handle indices and logical indices of NaNs.

Input:

- y, 1d numpy array with possible NaNs

Output:

- nans, logical indices of NaNs
- index, a function, with signature indices= index(logical_indices), to convert logical indices of NaNs to 'equivalent' indices

Example:

```
>>> # linear interpolation of NaNs
>>> nans, x= nan_helper(y)
>>> y[nans]= np.interp(x(nans), x(~nans), y[~nans])
```

3.2.8 engram.procedural.models module

engram.procedural.models.**cnn**(*shape*)

engram.procedural.models.**custom**(*shape*)

engram.procedural.models.**lstm**()

engram.procedural.models.**md**(*shape*)

engram.procedural.models.**mimo**(*shape*)

engram.procedural.models.**select**(*model*, *shape*)

3.2.9 engram.procedural.neo_handler module

engram.procedural.neo_handler.**unpackNeo**(*reader*)

3.2.10 engram.procedural.predict module

engram.procedural.predict.**predict**(*model=None*, *mneme=None*)

3.2.11 engram.procedural.train module

engram.procedural.train.**create_dataset**(*features=None*, *labels_for_categories=None*)
Load and parse dataset. Args:

filenames: list of image paths labels: numpy array of shape (BATCH_SIZE, N_LABELS) is_training: boolean to indicate training mode

engram.procedural.train.**get_data**(*features=None*, *labels_for_categories=None*)

engram.procedural.train.**train**(*model_type='CNN'*, *in_matrix=None*, *labels=None*)

3.2.12 Module contents

:mod:'engram.procedural' provides functions for processing Engram data structures and encoding them into models

3.3 engram.episodic package

3.3.1 Submodules

3.3.2 engram.episodic.classic module

engram.episodic.classic.**analog_signal**(*x=None*, *y=None*)

engram.episodic.classic.**spectrum**(*x=None*, *y=None*, *z=None*, *voltage_units='mV'*, *resIncrease=8*, *clims=(-5, 5)*)

3.3.3 engram.episodic.graphics module

3.3.4 engram.episodic.shaders module

class engram.episodic.shaders.**Galaxy**(*n=20000*)
Bases: object

Galaxy simulation using the density wave theory

eccentricity(*r*)

reset(*rad*, *radCore*, *deltaAng*, *ex1*, *ex2*, *sigma*, *velInner*, *velOuter*)

update(*timestep=100000*)
Update simulation

engram.episodic.shaders.**atom**()

engram.episodic.shaders.**bb_spectrum**(*wavelength*, *bbTemp=5000*)
Calculate, by Planck's radiation law, the emittance of a black body of temperature bbTemp at the given wavelength (in metres). */

engram.episodic.shaders.**boids**()
Demonstration of boids simulation. Boids is an artificial life program, developed by Craig Reynolds in 1986, which simulates the flocking behaviour of birds. Based on code from glumpy by Nicolas Rougier.

engram.episodic.shaders.**brain**()
3D brain mesh viewer.

engram.episodic.shaders.**constrain_rgb**(*r*, *g*, *b*)
If the requested RGB shade contains a negative weight for one of the primaries, it lies outside the colour gamut accessible from the given triple of primaries. Desaturate it by adding white, equal quantities of R, G, and B, enough to make RGB all positive. The function returns 1 if the components were modified, zero otherwise.

engram.episodic.shaders.**engram**(*regions*, *spikes*, *assignments*)

engram.episodic.shaders.**fireworks**()
Example demonstrating simulation of fireworks using point sprites. (adapted from the "OpenGL ES 2.0 Programming Guide")

This example demonstrates a series of explosions that last one second. The visualization during the explosion is highly optimized using a Vertex Buffer Object (VBO). After each explosion, vertex data for the next explosion are calculated, such that each explostion is unique.

engram.episodic.shaders.**fluid**()

engram.episodic.shaders.**fun**()

engram.episodic.shaders.**galaxy**()

engram.episodic.shaders.**gamma_correct**(*cs*, *c*)
Transform linear RGB values to nonlinear RGB values. Rec. 709 is ITU-R Recommendation BT. 709 (1990) ''Basic Parameter Values for the HDTV Standard for the Studio and for International Programme Exchange'', formerly CCIR Rec. 709. For details see

http://www.poynton.com/ColorFAQ.html http://www.poynton.com/GammaFAQ.html

engram.episodic.shaders.**gamma_correct_rgb**(*cs*, *r*, *g*, *b*)

engram.episodic.shaders.**graph**()
Plot clusters of data points and a graph of connections

engram.episodic.shaders.**inside_gamut**(*r*, *g*, *b*)
Test whether a requested colour is within the gamut achievable with the primaries of the current colour system. This amounts simply to testing whether all the primary weights are non-negative. */

engram.episodic.shaders.**interact**()
Test the fps capability of Vispy with meshdata primitive

engram.episodic.shaders.**norm_rgb**(*r*, *g*, *b*)
Normalise RGB components so the most intense (unless all are zero) has a value of 1.

engram.episodic.shaders.**oscilloscope**()
An oscilloscope, spectrum analyzer, and spectrogram.

This demo uses pyaudio to record data from the microphone. If pyaudio is not available, then a signal will be generated instead.

engram.episodic.shaders.**realtimesignals**()
Multiple real-time digital signals with GLSL-based clipping.

engram.episodic.shaders.**sandbox**()
A GLSL sandbox application based on the spinning cube. Requires PySide or PyQt5.

engram.episodic.shaders.**select**(*shader='atom'*, *regions=None*, *data=None*, *assignments=None*)

engram.episodic.shaders.**shadertoy**()

engram.episodic.shaders.**spectrogram**(*data*, *settings*)

engram.episodic.shaders.**spectrum_to_xyz**(*spec_intens*, *temp*)
Calculate the CIE X, Y, and Z coordinates corresponding to a light source with spectral distribution given by the function SPEC_INTENS, which is called with a series of wavelengths between 380 and 780 nm (the argument is expressed in meters), which returns emittance at that wavelength in arbitrary units. The chromaticity coordinates of the spectrum are returned in the x, y, and z arguments which respect the identity:

x + y + z = 1.

CIE colour matching functions xBar, yBar, and zBar for wavelengths from 380 through 780 nanometers, every 5 nanometers. For a wavelength lambda in this range:

```
cie_colour_match[(lambda - 380) / 5][0] = xBar
cie_colour_match[(lambda - 380) / 5][1] = yBar
cie_colour_match[(lambda - 380) / 5][2] = zBar
```

AH Note 2011: This next bit is kind of irrelevant on modern hardware. Unless you are desperate for speed. In which case don't use the Python version!

To save memory, this table can be declared as floats rather than doubles; (IEEE) float has enough significant bits to represent the values. It's declared as a double here to avoid warnings about "conversion between floating-point types" from certain persnickety compilers.

engram.episodic.shaders.**upvp_to_xy**(*up*, *vp*)

engram.episodic.shaders.**xy_toupvp**(*xc*, *yc*)

engram.episodic.shaders.**xyz_to_rgb**(*cs*, *xc*, *yc*, *zc*)
Given an additive tricolour system CS, defined by the CIE x and y chromaticities of its three primaries (z is derived trivially as 1-(x+y)), and a desired chromaticity (XC, YC, ZC) in CIE space, determine the contribution of each primary in a linear combination which sums to the desired chromaticity. If the requested chromaticity falls outside the Maxwell triangle (colour gamut) formed by the three primaries, one of the r, g, or b weights will be negative.

Caller can use constrain_rgb() to desaturate an outside-gamut colour to the closest representation within the available gamut and/or norm_rgb to normalise the RGB components so the largest nonzero component has value 1.

3.3.5 engram.episodic.terminal module

engram.episodic.terminal.**endProgress**()

engram.episodic.terminal.**progress**(*x*)

engram.episodic.terminal.**startProgress**(*title*)

3.3.6 Module contents

:mod:'engram.episodic' provides functions for visualizing Engrams

3.4 engram.working package

3.4.1 Submodules

3.4.2 engram.working.loggers module

3.4.3 engram.working.streams module

3.4.4 Module contents

3.5 Indices and tables

- genindex
- modindex
- search

CHAPTER 4

Contributing

Any feedback is gladly received and highly appreciated! ENGRAM is a community project, and all contributions are welcomed.

4.1 Developers' guide

These instructions are for developing on a Unix-like platform, e.g. Linux or macOS, with the bash shell. If you develop on Windows, please get in touch.

4.1.1 Mailing lists

There is not currently a platform for general discussion of ENGRAM development.

Discussion of issues specific to a particular ticket in the issue tracker should take place on the tracker.

4.1.2 Using the issue tracker

If you find a bug in ENGRAM, please create a new ticket on the issue tracker, setting the type to "defect". Choose a name that is as specific as possible to the problem you've found, and in the description give as much information as you think is necessary to recreate the problem. The best way to do this is to create the shortest possible Python script that demonstrates the problem, and attach the file to the ticket.

If you have an idea for an improvement to ENGRAM, create a ticket with type "enhancement". If you already have an implementation of the idea, create a patch (see below) and attach it to the ticket.

To keep track of changes to the code and to tickets, you can register for a GitHub account and then set to watch the repository at GitHub Repository (see https://help.github.com/en/articles/watching-and-unwatching-repositories).

4.1.3 Requirements

Coming Soon

We strongly recommend you develop within a virtual environment (from virtualenv, venv or conda). It is best to have at least one virtual environment with Python 2.7 and one with Python 3.x.

4.1.4 Getting the source code

We use the Git version control system. The best way to contribute is through GitHub. You will first need a GitHub account, and you should then fork the repository at GitHub Repository (see http://help.github.com/en/articles/fork-a-repo).

To get a local copy of the repository:

```
$ cd /some/directory
$ git clone git@github.com:<username>/engram.git
```

Now you need to make sure that the `engram` package is on your PYTHONPATH. You can do this either by installing ENGRAM:

```
$ cd engram
$ python setup.py install
$ python3 setup.py install
```

(if you do this, you will have to re-run `setup.py install` any time you make changes to the code) *or* by creating symbolic links from somewhere on your PYTHONPATH, for example:

```
$ ln -s engram/engram
$ export PYTHONPATH=/some/directory:${PYTHONPATH}
```

An alternate solution is to install Engram with the *develop* option, this avoids reinstalling when there are changes in the code:

```
$ sudo python setup.py develop
```

or using the "-e" option to pip:

```
$ pip install -e engram
```

To update to the latest version from the repository:

```
$ git pull
```

4.1.5 Running the test suite

Before you make any changes, run the test suite to make sure all the tests pass on your system:

```
$ cd engram/test
```

With Python 2.7 or 3.x:

```
$ python -m unittest discover
$ python3 -m unittest discover
```

If you have nose installed:

```
$ nosetests
```

At the end, if you see "OK", then all the tests passed (or were skipped because certain dependencies are not installed), otherwise it will report on tests that failed or produced errors.

To run tests from an individual file:

```
$ python test_id.py
$ python3 test_id.py
```

4.1.6 Writing tests

You should try to write automated tests for any new code that you add. If you have found a bug and want to fix it, first write a test that isolates the bug (and that therefore fails with the existing codebase). Then apply your fix and check that the test now passes.

To see how well the tests cover the code base, run:

```
$ nosetests --with-coverage --cover-package=engram --cover-erase
```

4.1.7 Working on the documentation

All modules, classes, functions, and methods (including private and subclassed builtin methods) should have docstrings. Please see PEP257 for a description of docstring conventions.

Module docstrings should explain briefly what functions or classes are present. Detailed descriptions can be left for the docstrings of the respective functions or classes. Private functions do not need to be explained here.

Class docstrings should include an explanation of the purpose of the class and, when applicable, how it relates to standard neuroscientific data. They should also include at least one example, which should be written so it can be run as-is from a clean newly-started Python interactive session (that means all imports should be included). Finally, they should include a list of all arguments, attributes, and properties, with explanations. Properties that return data calculated from other data should explain what calculation is done. A list of methods is not needed, since documentation will be generated from the method docstrings.

Method and function docstrings should include an explanation for what the method or function does. If this may not be clear, one or more examples may be included. Examples that are only a few lines do not need to include imports or setup, but more complicated examples should have them.

Examples can be tested easily using the iPython *%doctest_mode* magic. This will strip >>> and ... from the beginning of each line of the example, so the example can be copied and pasted as-is.

The documentation is written in reStructuredText, using the Sphinx documentation system. Any mention of another ENGRAM module, class, attribute, method, or function should be properly marked up so automatic links can be generated.

To build the documentation:

```
$ cd engram/doc
$ make html
```

Then open *some/directory/engram/doc/build/html/index.html* in your browser.

4.1.8 Committing your changes

Once you are happy with your changes, **run the test suite again to check that you have not introduced any new bugs**. It is also recommended to check your code with a code checking program. Then you can commit them to your local repository:

```
$ git commit -m 'informative commit message'
```

If this is your first commit to the project, please add your name and affiliation/employer to `doc/source/authors.rst`

You can then push your changes to your online repository on GitHub:

```
$ git push
```

Once you think your changes are ready to be included in the main ENGRAM repository, open a pull request on GitHub (see https://help.github.com/en/articles/about-pull-requests).

4.1.9 Python version

ENGRAM has only been tested using Python 3.7.

For future reference, Porting to Python 3 by Lennart Regebro is an excellent resource.

The most important thing to remember is to run tests with at least one version of Python 2 and at least one version of Python 3. There is generally no problem in having multiple versions of Python installed on your computer at once: e.g., on Ubuntu Python 2 is available as *python* and Python 3 as *python3*, while on Arch Linux Python 2 is *python2* and Python 3 *python*. See PEP394 for more on this. Using virtual environments makes this very straightforward.

4.1.10 Coding standards and style

All code should conform as much as possible to PEP 8, and should run with Python 2.7, and 3.5 or newer.

You can use the pycodestyle program to check the code for PEP 8 conformity.

Please do not use `from xyz import *`. This is slow, can lead to conflicts, and makes it difficult for code analysis software.

4.1.11 Making a release

Add a new version file, such as `/doc/releases/0.1.0.rst` for the release.

First check that the version string (in `engram/version.py`) is correct.

To build a source package:

```
$ python setup.py sdist
```

Tag the release in the Git repository and push it:

```
$ git tag <version>
$ git push --tags origin
$ git push --tags upstream
```

To upload the package to PyPI (currently Garrett Flynn has the necessary permissions to do this):

```
$ twine upload dist/engram-0.X.Y.tar.gz
```

4.2 Authors and Contributors

4.2.1 Garrett Flynn

University of Southern California

Important: Garrett's **senior thesis** project was to build out ENGRAM as an open source project. Feel free to reach out to him at garrett@garrettflynn.com with any questions.

If we've somehow missed you off the list we're very sorry - please let us know!

CHAPTER 5

Release Notes

5.1 engram 0.1.0

?? April 2020

5.1.1 Improvements

- Add continuous integration with Travis CI for automated testing
- Add some tests
- Migrate example data to GIN

5.1.2 Bug fixes

- Raise an exception if a Neo RawIO cannot be found for the data file

CHAPTER 6

Acknowledgements

ENGRAM was developed at Song Lab for the Restoring Active Memory (RAM) program, funded by the Defence Advanced Research Projects Agency (DARPA).

See our *Contributing* section to get to know all the wonderful people who've contributed directly to ENGRAM!

Bibliography

[BRYC16] Pouya Bashivan, Irina Rish, Mohammed Yeasin, and Noel Codella. Learning representations from EEG with deep recurrent-convolutional neural networks. In *4th International Conference on Learning Representations, ICLR 2016 - Conference Track Proceedings*. 2016. arXiv:1511.06448.

[BBB+01] T W Berger, M Baudry, R D Brinton, J S Liaw, V Z Marmarelis, and A Y Park. Brain-implantable biomimetic electronics as the next era in neural prosthetics. Proceedings of the. *Ieee*, 89(7):993–1012, 2001.

[GKM+18] G. Gagnon, S. Kumar, J. R. Maltais, A. N. Voineskos, B. H. Mulsant, and T. K. Rajji. Superior memory performance in healthy individuals with subclinical psychotic symptoms but without genetic load for schizophrenia. *Schizophrenia Research: Cognition*, 2018. doi:10.1016/j.scog.2018.06.001.

[GGJ+14] Samuel Garcia, Domenico Guarino, Florent Jaillet, Todd Jennings, Robert Pröpper, Philipp L. Rautenberg, Chris C. Rodgers, Andrey Sobolev, Thomas Wachtler, Pierre Yger, and Andrew P. Davison. Neo: An object model for handling electrophysiology data in multiple formats. *Frontiers in Neuroinformatics*, 2014. doi:10.3389/fninf.2014.00010.

[Lea15] Charles Leadbeater. The Disremembered. 2015. URL: https://aeon.co/essays/if-your-memory-fails-are-you-still-the-same-person.

[WFW17] Michael L. Waskom, Michael C. Frank, and Anthony D. Wagner. Adaptive Engagement of Cognitive Control in Context-Dependent Decision Making. *Cerebral cortex (New York, N.Y. : 1991)*, 2017. doi:10.1093/cercor/bhv333.

Python Module Index

e

Index