

UNIVERSITY OF COSTA RICA
ELECTRICAL ENGINEERING DEPARTMENT
GRADE THESIS

Local Field Potential (LFP) signals classifier's implementation using Neural
Networks

CODE DOCUMENTATION

JOSÉ PABLO ÁVILA LÓPEZ (B30724)

JOSE.AVILALOPEZ@UCR.AC.CR

PROFESSOR: ESTEBAN ORTIZ



JULY, 2021

Contents

1	LFP Classification	1
2	Namespace Index	3
2.1	Packages	3
3	File Index	5
3.1	File List	5
4	Namespace Documentation	7
4.1	DataExploration Namespace Reference	7
4.1.1	Function Documentation	7
4.1.1.1	dataExploration()	7
4.2	DatasetGenerator Namespace Reference	8
4.2.1	Function Documentation	8
4.2.1.1	data_set_generator()	8
4.3	Environment Namespace Reference	9
4.3.1	Function Documentation	10
4.3.1.1	finish_test()	10
4.3.1.2	init_environment()	11
4.3.1.3	log_versions()	11
4.3.1.4	print_box()	11
4.3.1.5	print_header()	12
4.3.1.6	print_parameters()	12
4.3.1.7	print_text()	12
4.3.1.8	step()	13
4.3.2	Variable Documentation	13
4.3.2.1	caller_file	13
4.3.2.2	CAPTURES_FOLDER	13
4.3.2.3	CURRENT_FOLDER	13
4.3.2.4	debug	13
4.3.2.5	LINE_LENGTH	14
4.3.2.6	LOGS_FOLDER	14

4.3.2.7	RESULTS_FOLDER	14
4.3.2.8	START_TIME	14
4.3.2.9	step_number	14
4.3.2.10	versions_log	14
4.3.2.11	VERSIONS_LOG_PATH	14
4.3.2.12	yomchi_log	14
4.3.2.13	YOMCHI_LOG_PATH	15
4.4	models Namespace Reference	15
4.4.1	Function Documentation	15
4.4.1.1	cnn()	15
4.4.1.2	compile_and_fit()	16
4.4.1.3	lstm()	16
4.4.1.4	mlp()	17
4.5	Predictor Namespace Reference	17
4.5.1	Function Documentation	17
4.5.1.1	predictor()	17
4.6	preprocessing Namespace Reference	18
4.6.1	Function Documentation	19
4.6.1.1	add_labels()	19
4.6.1.2	angles_expansion()	19
4.6.1.3	average_angles()	20
4.6.1.4	channels_to_windows()	20
4.6.1.5	clean_invalid_positional()	21
4.6.1.6	downsample_lfps()	21
4.6.1.7	interpolate_angles()	22
4.6.1.8	load_angles_data()	22
4.6.1.9	load_lfp_data()	23
4.6.1.10	ndarray_to_dataframe()	23
4.6.1.11	shortest_angle_interpolation()	23
4.6.1.12	vectorized_sai()	24
4.6.2	Variable Documentation	24
4.6.2.1	ANGLES	24
4.6.2.2	EC014_41_NUMBER_OF_CHANNELS	24
4.6.2.3	LFP	24
4.6.2.4	LFP_DATAMAX_SAMPLING_RATE	25
4.6.2.5	LFP_NEURALYNX_SAMPLING_RATE	25
4.6.2.6	PATH_TO_DATASETS	25
4.6.2.7	POSITION_DATA_SAMPLING_RATE	25
4.6.2.8	RAW_DATAMAX_SAMPLING_RATE	25
4.6.2.9	RAW_NEURALYNX_SAMPLING_RATE	25

4.7	visualization Namespace Reference	25
4.7.1	Function Documentation	26
4.7.1.1	store_figure()	26
4.7.2	Variable Documentation	26
4.7.2.1	FIG_FORMAT	26
5	File Documentation	27
5.1	/home/pabloav/Documents/Tesis_Lic/LFP-Classification/DataExploration/DataExploration.py File Reference	27
5.1.1	Detailed Description	27
5.2	DataExploration.py	27
5.3	/home/pabloav/Documents/Tesis_Lic/LFP-Classification/DatasetGenerator/DatasetGenerator.py File Reference	32
5.3.1	Detailed Description	32
5.4	DatasetGenerator.py	32
5.5	/home/pabloav/Documents/Tesis_Lic/LFP-Classification/Predictor/Predictor.py File Reference	35
5.5.1	Detailed Description	36
5.6	Predictor.py	36
5.7	/home/pabloav/Documents/Tesis_Lic/LFP-Classification/README.md File Reference	38
5.8	/home/pabloav/Documents/Tesis_Lic/LFP-Classification/README.md	38
5.9	/home/pabloav/Documents/Tesis_Lic/LFP-Classification/Yomchi/Environment.py File Reference	39
5.9.1	Detailed Description	40
5.10	Environment.py	41
5.11	/home/pabloav/Documents/Tesis_Lic/LFP-Classification/Yomchi/models.py File Reference	44
5.11.1	Detailed Description	44
5.12	models.py	45
5.13	/home/pabloav/Documents/Tesis_Lic/LFP-Classification/Yomchi/preprocessing.py File Reference	47
5.13.1	Detailed Description	48
5.14	preprocessing.py	48
5.15	/home/pabloav/Documents/Tesis_Lic/LFP-Classification/Yomchi/visualization.py File Reference	53
5.15.1	Detailed Description	54
5.16	visualization.py	54
Index		57

Chapter 1

LFP Classification

Classifier of Local Field Potential signals using Deep Neural Networks for head angular direction prediction

Prerequisites

The current project was developed using **Ubuntu 18.04.4 LTS**. The following are the required packages and python libraries. It is recommended to use the listed versions, nevertheless is worth to try the newest available version of them.

Packages:

1. Python (3.6)

Optional:

1. Latex
2. Doxygen
3. Doxypy
4. Make
5. Git
6. Graphviz

Python libraries:

Package	Version
pip3	20.1
numpy	1.18.4
matplotlib	3.2.1
tensorflow	2.2.0
keras	2.3.1
sklearn	0.0
pandas	1.0.3
scikit-learn	0.23.0
seaborn	0.10.1

Install

Check the installed Linux version using: `lsb_release -a`. Check the installed version of the package using: `<package> --<version>` Check the installed python libraries and their version using: `pip list`

Packages:

```
sudo apt-get install python3.6 texlive-latex-extra doxygen doxygen-gui make git graphviz
```

Pip:

This is necessary to install the rest of python libraries

```
curl https://bootstrap.pypa.io/get-pip.py -o get-pip.py
python3.6 get-pip.py
```

Python libraries:

If getting the latest available version:

```
pip3 install --upgrade <package>
```

To download an specific version:

```
pip3 install <package>==<version>
```

Usage

Author

Pablo Avila - [PabloAvLo](#)

License

This project is licensed under the MIT License - see the LICENSE.md file for details

Acknowledgments

- The present code is part of my grade thesis in Electrical Engineering at the University of Costa Rica.
- The main objective is to prove the hypothesis that based on Local Field Potential measurments from the Entorhinal Cortex of the Hippocampus of rats is possible to extract angular information of the head direction of the subject.
- The dataset used in this experiment is: [BuszakiLab HC-3 Dataset](#).
- In order to do so, machine learning algorithms will be applied to the data, such as Feedforward Neural Network, RNNs and CNNs.

Chapter 2

Namespace Index

2.1 Packages

Here are the packages with brief descriptions (if available):

DataExploration	7
DatasetGenerator	8
Environment	9
models	15
Predictor	17
preprocessing	18
visualization	25

Chapter 3

File Index

3.1 File List

Here is a list of all files with brief descriptions:

/home/pabloav/Documents/Tesis_Lic/LFP-Classification/DataExploration/ DataExploration.py	27
/home/pabloav/Documents/Tesis_Lic/LFP-Classification/DatasetGenerator/ DatasetGenerator.py	32
/home/pabloav/Documents/Tesis_Lic/LFP-Classification/Predictor/ Predictor.py	35
/home/pabloav/Documents/Tesis_Lic/LFP-Classification/Yomchi/ Environment.py	39
/home/pabloav/Documents/Tesis_Lic/LFP-Classification/Yomchi/ models.py	44
/home/pabloav/Documents/Tesis_Lic/LFP-Classification/Yomchi/ preprocessing.py	47
/home/pabloav/Documents/Tesis_Lic/LFP-Classification/Yomchi/ visualization.py	53

Chapter 4

Namespace Documentation

4.1 DataExploration Namespace Reference

Functions

- def [dataExploration](#) ()

4.1.1 Function Documentation

4.1.1.1 dataExploration()

```
def DataExploration.dataExploration ( )
```

Dataset Generator:

Setup

- Initialize [Environment](#).
- Initialize Tensorflow session.
- Set seed for Numpy and Tensorflow
- Specify run configuration parameters.
- Specify dataset generation parameters.

Procedure

- **Step 1:**
 - Import LFP data.
 - Plot channels 0 and 97.
- **Step 2:**
 - Import angles data.
 - Plot angles data.

- **Step 3:**
 - If the chosen synchronization method is to downsample the LFP data:
 - * Downsample LFP data to match Angles sampling rate.
 - * Plot channels 0 and 97 downsampled.
 - Else, the chosen synchronization method is to upsample the Angles data:
 - * Fill angles gaps to match the LFP sampling rate.
 - * Plot angles data in the new sampling rate.
- **Step 4:**
 - Interpolate angles data using a 'interpolation' approach.
 - Plot angles data after interpolation.
- **Step 5:**
 - Label data by concatenating LFPs and interpolated Angles in a single 2D-array.
 - Plot LFP and angles data.
 - Print an angles data window where the first LED is unsynchronized and then the second LED is lost too.
- **Step 6:**
 - Clean the labeled dataset from invalid values.
 - Print Bar plot for labeled angles.
- **Step 7:**
 - Count number of NaNs at the beginning and at the end of the data.
 - Print some NaN counts in different dataset stages.
- **Step 8:** Plotting clean LFP data from channels 0 and interpolated angles data.
- **Step 9:** Plotting Boxplot of all channels in two figures, from 0-49 and from 50-98.
- **Step 10:**
 - Plotting Boxplot of interpolated angles.
 - Plotting Boxplot of interpolated angles vs channel 0
- **Finish Test and Exit:** Save logs, captures and results.

Definition at line 94 of file [DataExploration.py](#).

4.2 DatasetGenerator Namespace Reference

Functions

- `def data_set_generator ()`

4.2.1 Function Documentation

4.2.1.1 `data_set_generator()`

```
def DatasetGenerator.data_set_generator ( )
```

Dataset Generator:**Setup**

- Initialize [Environment](#).
- Initialize Tensorflow session.
- Set seed for Numpy and Tensorflow
- Specify run configuration parameters.
- Specify dataset generation parameters.

Procedure

- **Step 1:** Import LFP data.
- **Step 2:** Import angles data.
- **Step 3:**
 - Downsample LFP data to match the Angles sampling rate or
 - Fill angles gaps to match the LFP sampling rate.
- **Step 4:** Label data by concatenating LFPs and Angles in a single 2D-array.
- **Step 5:** Clean the labeled dataframe from -1 values, which represent the wrongly acquired positional samples.
- **Step 6:** Interpolate angles data using an 'interpolation' approach.
- **Step 7:** Plotting Angles and the selected channel of the LFPs after cleaning and interpolation.
- **Step 8:**
 - Get preferred angle according to LFPs channel.
 - Plotting the sum of LFPs per angle from 0 to 359 degrees.
- **Step 9:** Get largest subset of data.
- **Step 10:** Split the subset into training, validation and testing set.
- **Step 11:** Save the dataset to a pickle file.
- **Step 12:** Convert data to windowed series.
- **Finish Test and Exit:** Save logs, captures, results and the generated dataset into a .pickle file.

Definition at line 65 of file [DatasetGenerator.py](#).

4.3 Environment Namespace Reference

Functions

- def [init_environment](#) (print_the_header=False, enable_debug=False)
Initialize the directories and files to log the results.
- def [print_text](#) (text)
Print a passed text and save it in the yomchi_log.txt.
- def [print_box](#) (text)
Print a passed text in a box and save it in the yomchi_log.txt.
- def [step](#) (description=None, new_step_number=0)

- *Print the current step number with a brief description.*
- `def print_parameters (params_dictionary)`
Print a list of parameters.
- `def log_versions ()`
Stores the environment versions including packages, libraries and OS.
- `def print_header ()`
Print a header in the console output and log files.
- `def finish_test (rename_results_folder=None)`
Safely finish the test run and log some final information.

Variables

- `START_TIME = datetime.datetime.now()`
- `CURRENT_FOLDER = os.getcwd()`
- `string RESULTS_FOLDER = CURRENT_FOLDER + "/Results-" + START_TIME.strftime("%Y-%m-%d_%H-%M") + "/"`
- `string LOGS_FOLDER = RESULTS_FOLDER + "Logs/"`
- `string CAPTURES_FOLDER = RESULTS_FOLDER + "Captures/"`
- `string VERSIONS_LOG_PATH = LOGS_FOLDER + "versions_log.txt"`
- `string YOMCHI_LOG_PATH = LOGS_FOLDER + "yomchi_log.txt"`
- `int LINE_LENGTH = 100`
- `int step_number = 1`
- `versions_log = None`
- `yomchi_log = None`
- `caller_file = None`
- `bool debug = False`

4.3.1 Function Documentation

4.3.1.1 finish_test()

```
def Environment.finish_test (
    rename_results_folder = None )
```

Safely finish the test run and log some final information.

Parameters

<code>rename_results_folder</code>	Optional name for the results folder.
------------------------------------	---------------------------------------

Returns

None

Definition at line 248 of file [Environment.py](#).

References [print_text\(\)](#).

4.3.1.2 `init_environment()`

```
def Environment.init_environment (
    print_the_header = False,
    enable_debug = False )
```

Initialize the directories and files to log the results.

Parameters

<i>print_the_header</i>	If True, print the header
<i>enable_debug</i>	If True, run some code to help debugging

Definition at line 49 of file [Environment.py](#).

References [log_versions\(\)](#), and [print_header\(\)](#).

4.3.1.3 `log_versions()`

```
def Environment.log_versions ( )
```

Stores the environment versions including packages, libraries and OS.

Returns

None

Definition at line 177 of file [Environment.py](#).

Referenced by [init_environment\(\)](#).

4.3.1.4 `print_box()`

```
def Environment.print_box (
    text )
```

Print a passed text in a box and save it in the yomchi_log.txt.

Parameters

<i>text</i>	Text to print
-------------	---------------

Returns

None

Definition at line 96 of file [Environment.py](#).

References [print_text\(\)](#).

Referenced by [step\(\)](#).

4.3.1.5 `print_header()`

```
def Environment.print_header ( )
```

Print a header in the console output and log files.

Returns

None

Definition at line 223 of file [Environment.py](#).

References [print_text\(\)](#).

Referenced by [init_environment\(\)](#).

4.3.1.6 `print_parameters()`

```
def Environment.print_parameters (
    params_dictionary )
```

Print a list of parameters.

Parameters

<i>params_dictionary</i>	Dictionary of parameters {name: value} to print.
--------------------------	--

Returns

None

Definition at line 149 of file [Environment.py](#).

References [print_text\(\)](#).

4.3.1.7 `print_text()`

```
def Environment.print_text (
    text )
```

Print a passed text and save it in the yomchi_log.txt.

Parameters

<i>text</i>	Text to print
-------------	---------------

Returns

None

Definition at line 83 of file [Environment.py](#).

Referenced by [finish_test\(\)](#), [print_box\(\)](#), [print_header\(\)](#), and [print_parameters\(\)](#).

4.3.1.8 step()

```
def Environment.step (
    description = None,
    new_step_number = 0 )
```

Print the current step number with a brief description.

Parameters

<i>description</i>	Description of the step.
<i>new_step_number</i>	If defined, reset the steps numeration to the passed value.

Returns

None

Definition at line 126 of file [Environment.py](#).

References [print_box\(\)](#).

4.3.2 Variable Documentation

4.3.2.1 caller_file

```
Environment.caller_file = None
```

Definition at line 39 of file [Environment.py](#).

4.3.2.2 CAPTURES_FOLDER

```
string Environment.CAPTURES_FOLDER = RESULTS_FOLDER + "Captures/"
```

Definition at line 30 of file [Environment.py](#).

4.3.2.3 CURRENT_FOLDER

```
Environment.CURRENT_FOLDER = os.getcwd()
```

Definition at line 27 of file [Environment.py](#).

4.3.2.4 debug

```
bool Environment.debug = False
```

Definition at line 40 of file [Environment.py](#).

4.3.2.5 LINE_LENGTH

```
int Environment.LINE_LENGTH = 100
```

Definition at line 33 of file [Environment.py](#).

4.3.2.6 LOGS_FOLDER

```
string Environment.LOGS_FOLDER = RESULTS_FOLDER + "Logs/"
```

Definition at line 29 of file Environment.py.

4.3.2.7 RESULTS_FOLDER

```
string Environment.RESULTS_FOLDER = CURRENT_FOLDER + "/Results-" + START_TIME.strftime("%Y-%m-%d↵
_%H-%M") + "/"
```

Definition at line 28 of file [Environment.py](#).

4.3.2.8 START_TIME

```
Environment.START_TIME = datetime.datetime.now()
```

Definition at line 26 of file [Environment.py](#).

4.3.2.9 step_number

```
int Environment.step_number = 1
```

Definition at line 36 of file [Environment.py](#).

4.3.2.10 versions_log

```
Environment.versions_log = None
```

Definition at line 37 of file Environment.py.

4.3.2.11 VERSIONS_LOG_PATH

```
string Environment.VERSIONS_LOG_PATH = LOGS_FOLDER + "versions_log.txt"
```

Definition at line 31 of file Environment.py.

4.3.2.12 yomchi_log

```
Environment.yomchi_log = None
```

Definition at line 38 of file [Environment.py](#).

4.3.2.13 YOMCHI_LOG_PATH

```
string Environment.YOMCHI_LOG_PATH = LOGS_FOLDER + "yomchi_log.txt"
```

Definition at line 32 of file [Environment.py](#).

4.4 models Namespace Reference

Functions

- def [mlp](#) (layers, units_per_layer, dropout=None)
Defines a classical neural network sometimes called: Multi-Layer Perceptron.
- def [cnn](#) (inputs, units_per_layer)
Creates a model of a Convolutional Neural Network with a Conv1D as the input layer, one dense as the only hidden layer and another dense with only 1 neuron as the output layer.
- def [lstm](#) (units_per_layer)
Creates a model of a Long Short-Term Memory Neural Network as the input layer, one dense with only 1 neuron as the output layer.
- def [compile_and_fit](#) (model, train, val, epochs=20, patience=2)
Fits the provided training data in the specified model and train's it.

4.4.1 Function Documentation

4.4.1.1 [cnn\(\)](#)

```
def models.cnn (
    inputs,
    units_per_layer )
```

Creates a model of a Convolutional Neural Network with a Conv1D as the input layer, one dense as the only hidden layer and another dense with only 1 neuron as the output layer.

The first two layer has ReLU activation and the last one uses a linear activation function.S

Parameters

<i>inputs</i>	Number of inputs of the network. It is used as the kernel size with the intention that the 1D convolutional layer outputs a single value throughout the specified number of filters.
<i>units_per_layer</i>	Specified the number of neurons of the hidden dense layer, which has to match with the number of filters that will output a result in the 1D convolutional input layer.

Returns

conv_model: The model of the CNN created for later usage as the predictor.

Definition at line 82 of file [models.py](#).

Referenced by [Predictor.predictor\(\)](#).

4.4.1.2 compile_and_fit()

```
def models.compile_and_fit (
    model,
    train,
    val,
    epochs = 20,
    patience = 2 )
```

Fits the provided training data in the specified model and train's it.

The validation data is used to compare the performance of the model against unknown data. MSE is used as the cost function to train the model and MAE as the performance evaluation metric.

Parameters

<i>model</i>	Model to train. It can be a MLP, CNN or LSTM, among others.
<i>train</i>	The dataset for training the model. Must have the shape: (batch, time, features)
<i>val</i>	The dataset for validating the model. Must have the shape: (batch, time, features)
<i>epochs</i>	Number of iteration over the entire set of data (all the batches).
<i>patience</i>	Number of epochs to wait for improvement in the metrics. If there is no notorious improvement in the performance of the validation set after the 'patience' epochs, the training will stop at this point.

Returns

history: The results of the training.

Definition at line 134 of file [models.py](#).

Referenced by [Predictor.predictor\(\)](#).

4.4.1.3 lstm()

```
def models.lstm (
    units_per_layer )
```

Creates a model of a Long Short-Term Memory Neural Network as the input layer, one dense with only 1 neuron as the output layer.

The activation functions of the LSTM layer are the regular ones for each of it's gates.

Parameters

<i>units_per_layer</i>	Number of inputs of the network.
------------------------	----------------------------------

Returns

lstm_model: The model of the LSTM created for later usage as the predictor.

Definition at line 105 of file [models.py](#).

Referenced by [Predictor.predictor\(\)](#).

4.4.1.4 mlp()

```
def models.mlp (
    layers,
    units_per_layer,
    dropout = None )
```

Defines a classical neural network sometimes called: Multi-Layer Perceptron.

It is Feedforward and fully-connected, with the specified parameters. The activation function of the hidden layers is ReLU, and the activation of the output is linear.

Parameters

<i>layers</i>	Number of hidden layers (besides the input and outputs ones).
<i>units_per_layer</i>	Number of neurons per layer. Applies to all layers except for the last one which has only 1: the predicted angle.
<i>dropout</i>	A value between 0 and 1 of neuron's results to discard of the training. If provided, two layers of this regularization method will be added to the model. One after the input layer and one before the output layer.

Returns

model: The model of the MLP created for later usage as the predictor.

Definition at line 32 of file [models.py](#).

Referenced by [Predictor.predictor\(\)](#).

4.5 Predictor Namespace Reference

Functions

- def [predictor](#) ()

4.5.1 Function Documentation

4.5.1.1 predictor()

```
def Predictor.predictor ( )
```

Predictor

Experiment Setup

- Initialize [Environment](#)
- Initialize Tensorflow session.
- Set seed for Numpy and Tensorflow
- Specify run configuration parameters.
- Specify run configuration parameters.

- Specify dataset generation parameters.
- Specify model's training parameters.

Procedure

- **Step 1:** Import training, validation and test datasets of LFPs as inputs and Angles as labels.
- **Step 2:** Convert data to windowed series.
- **Step 3:** Create the model with the specified parameters
- **Step 4:** Train the model with the training and validation data.
- **Step 5:** Plotting the original angular data vs the predictions for a determined number of batches
- **Finish Test and Exit:** Save logs, captures and results.

Definition at line 53 of file [Predictor.py](#).

References [models.cnn\(\)](#), [models.compile_and_fit\(\)](#), [models.lstm\(\)](#), and [models.mlp\(\)](#).

4.6 preprocessing Namespace Reference

Functions

- [def load_lfp_data](#) (file=[LFP\[771\]](#), channels=[EC014_41_NUMBER_OF_CHANNELS](#))
Loads the LFP signals from a .eeg file (LFP Data only $f < 625\text{Hz}$) or a .dat file (LFP Data + Spikes).
- [def load_angles_data](#) (file=[ANGLES\[771\]](#), degrees=True)
Loads the animal position data from a .whl file which contain 2 (x, y) pairs, one for each LED.
- [def downsample_lfps](#) (lfp_data, orig_rate, new_rate)
Downsample the LFP signal data after applying an anti-aliasing filter.
- [def angles_expansion](#) (angles_data, orig_rate, new_rate)
Fill angular data with 'NaN' values to match an expected sampling rate.
- [def shortest_angle_interpolation](#) (start, end, amount)
This interpolation method considers the 'start' and 'end' as angles in a circumference where the objective is to find the smallest arch between two angles.
- [def vectorized_sai](#) (angles_data)
Replace 'NaN' values between valid values (interpolation) in angular data with an interpolated value using the shortest angle interpolation.
- [def interpolate_angles](#) (angles_data, method="Shortest")
Replace 'NaN' values in angular data with an interpolated value using a given method.
- [def add_labels](#) (lfps, angles, round_labels, start=0, offset=30)
Add an additional column to the LFP signals matrix with the angular data used as the labels.
- [def clean_invalid_positional](#) (labeled_dataset, is_padded=True)
Clean the data rows which have '-1' values as labels (angles) from the the data and their LFPs associated in each channel.
- [def ndarray_to_dataframe](#) (dataset, rate)
Converts an n-D Numpy array to a Pandas Dataframe.
- [def channels_to_windows](#) (series, channel, window_size, batch_size, shuffle_buffer=None, offset=1)
Receives a numpy array containing the time series of LFP signals of n channels and returns the same data, separated in windows.
- [def average_angles](#) (angles, window_size)
Receives a numpy array containing the time series of the angles and returns the an set of windows with the average of the 'window_size' angles in each window.

Variables

- `PATH_TO_DATASETS` = `os.path.join(Env.CURRENT_FOLDER, "../Datasets/")`
- dictionary `LFP`
- dictionary `ANGLES`
- int `RAW_DATAMAX_SAMPLING_RATE` = 20000
- int `RAW_NEURALYNX_SAMPLING_RATE` = 32552
- int `LFP_DATAMAX_SAMPLING_RATE` = 1250
- int `LFP_NEURALYNX_SAMPLING_RATE` = 1252
- float `POSITION_DATA_SAMPLING_RATE` = 39.06
- int `EC014_41_NUMBER_OF_CHANNELS` = 99

4.6.1 Function Documentation

4.6.1.1 `add_labels()`

```
def preprocessing.add_labels (
    lfps,
    angles,
    round_labels,
    start = 0,
    offset = 30 )
```

Add an additional column to the LFP signals matrix with the angular data used as the labels.

Parameters

<i>lfps</i>	Matrix [n x numChannels] with the LFP signals used as the preliminary features of the data.
<i>angles</i>	Array with the angles data extracted from the positions used as the labels of the data.
<i>round_labels</i>	Boolean, if true the labels are rounded to angles multiples of 'offset' starting from 'start'
<i>start</i>	Angle in [0°, 360°] used as first label.
<i>offset</i>	Offset in [1°, 360°] between labels starting from 'start' angle.

Returns

`labeled_data`: Matrix with the labeled data [n x `lfps[numChannels]`, `angles`].

Definition at line 235 of file `preprocessing.py`.

4.6.1.2 `angles_expansion()`

```
def preprocessing.angles_expansion (
    angles_data,
    orig_rate,
    new_rate )
```

Fill angular data with 'NaN' values to match an expected sampling rate.

Usually the position data is acquired at a lower sampling rate than the LFP signals.

Assuming that the acquisition of data started and stopped at the same time, then no data has to be added after the last sample.

Parameters

<i>angles_data</i>	Array with the angles data extracted from the animal positions.
<i>orig_rate</i>	Sampling rate originally used to acquire the data.
<i>new_rate</i>	New sampling rate of the data. The gaps are filled with 'NaN'.

Returns

upsampled_data: Original data filled with 'NaN' to match the new sampling rate.

Definition at line 135 of file [preprocessing.py](#).

4.6.1.3 average_angles()

```
def preprocessing.average_angles (
    angles,
    window_size )
```

Receives a numpy array containing the time series of the angles and returns the an set of windows with the average of the 'window_size' angles in each window.

Each window is 1 element shifted from the previous window.

Parameters

<i>angles</i>	Numpy Array with the Angles data to use as the labels.
<i>window_size</i>	Size of the windows in which the data are being split.

Returns

average_angles: Averaged Angles data separated in windows.

Definition at line 416 of file [preprocessing.py](#).

4.6.1.4 channels_to_windows()

```
def preprocessing.channels_to_windows (
    series,
    channel,
    window_size,
    batch_size,
    shuffle_buffer = None,
    offset = 1 )
```

Receives a numpy array containing the time series of LFP signals of n channels and returns the same data, separated in windows.

Parameters

<i>series</i>	Numpy Array with the LFP data of the n channels.
<i>channel</i>	Channel to use.
<i>window_size</i>	Size of the windows in which the data are being split.
<i>batch_size</i>	Number of pairs data-labels to group as a batch

Parameters

<i>shuffle_buffer</i>	Number of windows to shuffle at the same time.
<i>offset</i>	Number of samples to shift between windows.

Returns

windowed_ds: LFP data of the selected channel separated in windows.

Definition at line 371 of file [preprocessing.py](#).

4.6.1.5 clean_invalid_positional()

```
def preprocessing.clean_invalid_positional (
    labeled_dataset,
    is_padded = True )
```

Clean the data rows which have '-1' values as labels (angles) from the the data and their LFPs associated in each channel.

Plus the following 31 rows with NaN as angle value in case of padded data.

The positional data taken from the LEDs placed in the rat have discontinuities where one or both LEDs are lost, making them invalid. Hence '-1' values are used instead to denote invalid position data and are meant to be removed from the data since they are not representative labels.

Parameters

<i>labeled_dataset</i>	Matrix [n x (numChannels + 1)] with the LFP signals used as the preliminary features of the data and the angles data extracted from the positions used as the labels of the data.
<i>is_padded</i>	If True, manage the input labeled dataset as a padded array of angles, or a downsampled LFP set otherwise.

Returns

clean_dataset: Input data without invalid positional values.

Definition at line 283 of file [preprocessing.py](#).

4.6.1.6 downsample_lfps()

```
def preprocessing.downsample_lfps (
    lfp_data,
    orig_rate,
    new_rate )
```

Downsample the LFP signal data after applying an anti-aliasing filter.

An order 8 Chebyshev type I filter is used. Usually the LFP signals are acquired at a higher sampling rate than the position data.

Note

: This method assumes that the reason of frequencies is 32 to compute the decimation.

Parameters

<i>lfp_data</i>	Matrix [n x numChannels] with the LFP signals
<i>orig_rate</i>	Sampling rate originally used to acquire the data.
<i>new_rate</i>	New sampling rate of the data.

Returns

resampled_data: Original data downsampled to the new rate.

Definition at line 108 of file [preprocessing.py](#).

4.6.1.7 interpolate_angles()

```
def preprocessing.interpolate_angles (
    angles_data,
    method = "Shortest" )
```

Replace 'NaN' values in angular data with an interpolated value using a given method.

Parameters

<i>angles_data</i>	Array with the angles data extracted from the animal positions.
<i>method</i>	Interpolation method to fill the gaps in the data. The optional methods available are the supported by pandas.DataFrame.interpolate function, which are: 'linear', 'quadratic', 'cubic', 'polynomial', among others.

Returns

interpolated_angles: Array with the angles data interpolated using the given method.

Definition at line 211 of file [preprocessing.py](#).

References [vectorized_sai\(\)](#).

4.6.1.8 load_angles_data()

```
def preprocessing.load_angles_data (
    file = ANGLES[771],
    degrees = True )
```

Loads the animal position data from a .whl file which contain 2 (x, y) pairs, one for each LED.

If any position value equals '-1' then it's replaced with 'NaN' instead.

Parameters

<i>file</i>	Path to the file containing the animal LED's position information
<i>degrees</i>	If this flag is set, then the angles are returned in degrees from [0, 360[, or radians otherwise.

Returns

angles: Array with the angles in radians extracted from the positions. The angles are given as float16 values calculated as $\arctan(y_2 - y_1 / x_2 - x_1)$. Unless the denominator is 0, in that case '0' is returned for that element.

Definition at line 71 of file [preprocessing.py](#).

4.6.1.9 load_lfp_data()

```
def preprocessing.load_lfp_data (
    file = LFP[771],
    channels = EC014_41_NUMBER_OF_CHANNELS )
```

Loads the LFP signals from a .eeg file (LFP Data only $f < 625\text{Hz}$) or a .dat file (LFP Data + Spikes).

Parameters

<i>file</i>	Path to the file containing the animal LFP data.
<i>channels</i>	Number of recorded channels in LFP signals file.

Returns

lfp: Array (n x channels) with the data. With the columns being the channels and the rows the a different time step.

Definition at line 48 of file [preprocessing.py](#).

4.6.1.10 ndarray_to_dataframe()

```
def preprocessing.ndarray_to_dataframe (
    dataset,
    rate )
```

Converts an n-D Numpy array to a Pandas Dataframe.

Parameters

<i>dataset</i>	Matrix [n x (numChannels +1)] with the LFP signals used as the preliminary features of the data and the angles data extracted from the positions used as the labels of the data.
<i>rate</i>	

Returns

dataframe: Pandas data frame with Channels 0-99 and Angles as columns names, and the timestamp as indexes calculated as $1/\text{rate} * 1e6$ to get the time step of the acquisition in microseconds.

Definition at line 345 of file [preprocessing.py](#).

4.6.1.11 shortest_angle_interpolation()

```
def preprocessing.shortest_angle_interpolation (
    start,
```

```

        end,
        amount )

```

This interpolation method considers the 'start' and 'end' as angles in a circumference where the objective is to find the smallest arch between two angles.

param start: Start angle with values in the range of [0 to 360[param end: Final angle with values in the range of [0 to 360[param amount: Value between [0, 1] which determines how close the interpolated angle will be placed from the Start angle (0) or from the Final angle (1), being 0.5 the middle. return interpolated_angle: Interpolated angle between 'start' and 'end'.

Definition at line 160 of file [preprocessing.py](#).

Referenced by [vectorized_sai\(\)](#).

4.6.1.12 vectorized_sai()

```

def preprocessing.vectorized_sai (
    angles_data )

```

Replace 'NaN' values between valid values (interpolation) in angular data with an interpolated value using the shortest angle interpolation.

param angles_data: Array with the angles data extracted from the animal positions. return interpolated_angles: Array with the angles data interpolated using the Shortest Angle Interpolation

Definition at line 173 of file [preprocessing.py](#).

References [shortest_angle_interpolation\(\)](#).

Referenced by [interpolate_angles\(\)](#).

4.6.2 Variable Documentation

4.6.2.1 ANGLES

```
dictionary preprocessing.ANGLES
```

Initial value:

```

00001 = {765: PATH_TO_DATASETS + "ec014.765.whl",
00002      771: PATH_TO_DATASETS + "ec014.771.whl"}

```

Definition at line 28 of file [preprocessing.py](#).

4.6.2.2 EC014_41_NUMBER_OF_CHANNELS

```
int preprocessing.EC014_41_NUMBER_OF_CHANNELS = 99
```

Definition at line 37 of file [preprocessing.py](#).

4.6.2.3 LFP

```
dictionary preprocessing.LFP
```

Initial value:

```
00001 = {765: PATH_TO_DATASETS + "ec014.765.eeg",  
00002      771: PATH_TO_DATASETS + "ec014.771.eeg"}
```

Definition at line 26 of file [preprocessing.py](#).

4.6.2.4 LFP_DATAMAX_SAMPLING_RATE

```
int preprocessing.LFP_DATAMAX_SAMPLING_RATE = 1250
```

Definition at line 34 of file [preprocessing.py](#).

4.6.2.5 LFP_NEURALYNX_SAMPLING_RATE

```
int preprocessing.LFP_NEURALYNX_SAMPLING_RATE = 1252
```

Definition at line 35 of file [preprocessing.py](#).

4.6.2.6 PATH_TO_DATASETS

```
preprocessing.PATH_TO_DATASETS = os.path.join(Env.CURRENT_FOLDER, "../Datasets/")
```

Definition at line 25 of file [preprocessing.py](#).

4.6.2.7 POSITION_DATA_SAMPLING_RATE

```
float preprocessing.POSITION_DATA_SAMPLING_RATE = 39.06
```

Definition at line 36 of file [preprocessing.py](#).

4.6.2.8 RAW_DATAMAX_SAMPLING_RATE

```
int preprocessing.RAW_DATAMAX_SAMPLING_RATE = 20000
```

Definition at line 32 of file [preprocessing.py](#).

4.6.2.9 RAW_NEURALYNX_SAMPLING_RATE

```
int preprocessing.RAW_NEURALYNX_SAMPLING_RATE = 32552
```

Definition at line 33 of file [preprocessing.py](#).

4.7 visualization Namespace Reference

Functions

- def `store_figure` (`fig_name`, `show=False`)
Stores a figure.

Variables

- string `FIG_FORMAT` = 'png'

4.7.1 Function Documentation

4.7.1.1 `store_figure()`

```
def visualization.store_figure (
    fig_name,
    show = False )
```

Stores a figure.

Parameters

<i>fig_name</i>	Name of the figure to store.
<i>show</i>	Displays the figure when ready. Warning: Stalls execution until closing it.

Returns

None

Definition at line 40 of file [visualization.py](#).

4.7.2 Variable Documentation

4.7.2.1 `FIG_FORMAT`

```
string visualization.FIG_FORMAT = 'png'
```

Definition at line 21 of file [visualization.py](#).

Chapter 5

File Documentation

5.1 /home/pabloav/Documents/Tesis_Lic/LFP-Classification/DataExploration/DataExploration.py File Reference

Namespaces

- [DataExploration](#)

Functions

- def [DataExploration.dataExploration](#) ()

5.1.1 Detailed Description

Author

Pablo Avila [B30724] jose.avilalopez@ucr.ac.cr

Copyright

MIT License

Date

July, 2021

Script for explore the input data by manipulating, printing and plotting the information in different ways.

Definition in file [DataExploration.py](#).

5.2 DataExploration.py

```
00001 # #####
00002 #           University of Costa Rica
00003 #           Electrical Engineering Department
00004 #           Grade Thesis
00005 # #####
00006
00007 """
00008 @file DataExploration.py
00009 @author Pablo Avila [B30724] jose.avilalopez@ucr.ac.cr
00010 @copyright MIT License
```

```

00011 @date July, 2021
00012 @details Script for explore the input data by manipulating, printing and plotting the information in
        different ways.
00013 """
00014
00015 import Yomchi.Environment as Env
00016 import Yomchi.preprocessing as data
00017 import Yomchi.visualization as ui
00018
00019 import matplotlib.pyplot as plt
00020 import numpy as np
00021 import seaborn as sns
00022 import pandas as pd
00023 import tensorflow as tf
00024
00025 def dataExploration():
00026     """
00027     @details
00028     <h1> Dataset Generator:</h1>
00029     <h2> Setup </h2>
00030     <ul>
00031         <li> Initialize Environment.
00032         <li> Initialize Tensorflow session.
00033         <li> Set seed for Numpy and Tensorflow
00034         <li> Specify run configuration parameters.
00035         <li> Specify dataset generation parameters.
00036     </ul>
00037     <h2> Procedure </h2>
00038     <ul>
00039         <li> <b>Step 1:</b>
00040             <ul>
00041                 <li> Import LFP data.
00042                 <li> Plot channels 0 and 97.
00043             </ul>
00044         <li> <b>Step 2:</b>
00045             <ul>
00046                 <li> Import angles data.
00047                 <li> Plot angles data.
00048             </ul>
00049         <li> <b>Step 3:</b>
00050             <ul>
00051                 <li> If the chosen synchronization method is to downsample the LFP data:
00052                     <ul>
00053                         <li> Downsample LFP data to match Angles sampling rate.
00054                         <li> Plot channels 0 and 97 downsampled.
00055                     </ul>
00056                 <li> Else, the chosen synchronization method is to upsample the Angles data:
00057                     <ul>
00058                         <li> Fill angles gaps to match the LFP sampling rate.
00059                         <li> Plot angles data in the new sampling rate.
00060                     </ul>
00061                 </ul>
00062         <li> <b>Step 4:</b>
00063             <ul>
00064                 <li> Interpolate angles data using a 'interpolation' approach.
00065                 <li> Plot angles data after interpolation.
00066             </ul>
00067         <li> <b>Step 5:</b>
00068             <ul>
00069                 <li> Label data by concatenating LFPs and interpolated Angles in a single 2D-array.
00070                 <li> Plot LFP and angles data.
00071                 <li> Print an angles data window where the first LED is unsynchronized and then the second LED
is lost too.
00072             </ul>
00073         <li> <b>Step 6:</b>
00074             <ul>
00075                 <li> Clean the labeled dataset from invalid values.
00076                 <li> Print Bar plot for labeled angles.
00077             </ul>
00078         <li> <b>Step 7:</b>
00079             <ul>
00080                 <li> Count number of NaNs at the beginning and at the end of the data.
00081                 <li> Print some NaN counts in different dataset stages.
00082             </ul>
00083         <li> <b>Step 8:</b> Plotting clean LFP data from channels 0 and interpolated angles data.
00084         <li> <b>Step 9:</b> Plotting Boxplot of all channels in two figures, from 0-49 and from 50-98.
00085         <li> <b>Step 10:</b>
00086             <ul>
00087                 <li> Plotting Boxplot of interpolated angles.
00088                 <li> Plotting Boxplot of interpolated angles vs channel 0
00089             </ul>
00090         <li> <b>Finish Test and Exit:</b> Save logs, captures and results.
00091     </ul>
00092     """
00093
00094     # ----- SETUP -----
00095     Env.init_environment(True, enable_debug=True)

```

```

00096
00097     tf.keras.backend.clear_session()
00098     tf.random.set_seed(51)
00099     np.random.seed(51)
00100
00101     # Run configuration parameters
00102     PLOT = True
00103
00104     # Session and methods Parameters
00105     session = 771 # or 765
00106
00107     interpolation = "Shortest" # "linear" "quadratic" "cubic" "nearest" "Shortest"
00108     sync_method = "Upsample Angles" # "Upsample Angles" "Downsample LFPs"
00109
00110     # Data Properties
00111     num_channels = data.EC014_41_NUMBER_OF_CHANNELS
00112     rate_used = data.POSITION_DATA_SAMPLING_RATE
00113     if sync_method == "Upsample Angles":
00114         rate_used = data.LFP_DATAMAX_SAMPLING_RATE
00115
00116     # Windowing properties
00117     window_size = 31
00118     batch_size = 31
00119     shuffle_buffer = 1000
00120     lfp_channel = 0
00121     round_angles = False
00122     base_angle = 0 # Unused if round_angles = False
00123     offset_between_angles = 30 # Unused if round_angles = False
00124
00125     extra = {"Round angles to get discrete label": round_angles}
00126     if round_angles:
00127         extra.update({"Angle labels starting from": str(base_angle) + "°",
00128                     "until 360° in steps of": str(offset_between_angles) + "°"})
00129
00130     parameters_dictionary = {"Recording Session Number": str(session),
00131                             "Interpolation Method": interpolation.title(),
00132                             "Synchronization Method": sync_method,
00133                             "Number of Recorded Channels": str(num_channels),
00134                             "Sampling Rate to Use": str(rate_used) + " Hz",
00135                             "Window Size": window_size,
00136                             "Batch size (# of windows)": batch_size,
00137                             "LFP Signal channel to use": lfp_channel,
00138                             "Shuffle buffer size": shuffle_buffer}
00139
00140     parameters_dictionary.update(extra)
00141     Env.print_parameters(parameters_dictionary)
00142
00143     # ----- STEP 1 -----
00144     Env.step(f"Importing LFP data from session: {session}")
00145     lfp_data = data.load_lfp_data(data.LFP[session])
00146
00147     if PLOT:
00148         Env.print_text("Plotting LFP data from channels 0 and 97 at " + str(data.LFP_DATAMAX_SAMPLING_RATE)
00149 + " Hz")
00150         figname = str(session) + "_LFP_C0_and_C97_" + str(data.LFP_DATAMAX_SAMPLING_RATE) + " Hz"
00151         plt.figure(figname)
00152         plt.subplot(211)
00153         plt.plot(lfp_data[:, 0], "xr")
00154         plt.title("Señal LFP. Sesión: " + str(session) + " del Canal 0 a "
00155 + str(data.LFP_DATAMAX_SAMPLING_RATE) + " Hz")
00156
00157         plt.subplot(212)
00158         plt.plot(lfp_data[:, 97], "xb")
00159         plt.title("Señal LFP. Sesión: " + str(session) + " del Canal 97 a "
00160 + str(data.LFP_DATAMAX_SAMPLING_RATE) + " Hz")
00161         ui.store_figure(figname, True)
00162
00163     # ----- STEP 2 -----
00164     Env.step(f"Importing angles data from session: {session}")
00165     angles_data = data.load_angles_data(data.ANGLES[session])
00166
00167     Env.print_text("Min angle: " + str(np.nanmin(angles_data)) + ", Max angle: " + str(np.nanmax(
00168 angles_data)))
00169
00170     if PLOT:
00171         Env.print_text("Plotting Angles data [°] at " + str(data.POSITION_DATA_SAMPLING_RATE) + " Hz")
00172         figname = str(session) + "_Angles_degrees_" + str(data.POSITION_DATA_SAMPLING_RATE) + " Hz"
00173         plt.figure(figname)
00174         plt.plot(angles_data[:, 0], "xr")
00175         plt.title("Información de ángulos [°]. Sesión: " + str(session) + " a "
00176 + str(data.POSITION_DATA_SAMPLING_RATE) + " Hz")
00177         ui.store_figure(figname, True)
00178
00179     # ----- STEP 3 -----
00180     if sync_method == "Downsample LFPs":
00181         Env.step("Downsample LFP data to match Angles sampling rate.")

```

```

00181
00182         lfp_data = data.downsample_lfps(lfp_data, data.LFP_DATAMAX_SAMPLING_RATE,
data.POSITION_DATA_SAMPLING_RATE)
00183
00184         if PLOT:
00185             Env.print_text("Plotting LFP downsampled data to " + str(rate_used) + "Hz from channels 0 and
97.")
00186             figname = str(session) + "_LFP_downsampled_0_and_97_" + str(rate_used) + "Hz"
00187             plt.figure(figname)
00188             plt.subplot(211)
00189             plt.plot(lfp_data[:, 0], "xr")
00190             plt.title("Señal LFP del Canal 0 a " + str(rate_used) + "Hz. Sesión: " + str(session))
00191
00192             plt.subplot(212)
00193             plt.plot(lfp_data[:, 97], "xb")
00194             plt.title("Señal LFP del Canal 97 a " + str(rate_used) + "Hz. Sesión: " + str(session))
00195             ui.store_figure(figname, True)
00196
00197         elif sync_method == "Upsample Angles":
00198             Env.step("Upsample Angles data to reach a higher sampling rate")
00199
00200             angles_data = data.angles_expansion(angles_data, data.POSITION_DATA_SAMPLING_RATE,
data.LFP_DATAMAX_SAMPLING_RATE)
00201
00202             if PLOT:
00203                 Env.print_text("Plotting Angles data expanded with 'NaN' [°]")
00204                 figname = str(session) + "Expanded_Angles_degrees_" + str(rate_used) + "Hz"
00205                 plt.figure(figname)
00206                 plt.plot(angles_data[:, "xb")
00207                 plt.title("Información de ángulos [°] expandida a " + str(rate_used) + "Hz. Sesión: " + str(
session))
00208                 ui.store_figure(figname, True)
00209
00210             # ----- STEP 4 -----
00211             Env.step("Interpolate angles data using a " + interpolation + " approach.", 4)
00212
00213             angles_data_interpolated = data.interpolate_angles(angles_data, interpolation)
00214
00215             if PLOT:
00216                 Env.print_text("Plotting Angles data after " + interpolation + " interpolation [°]")
00217                 figname = str(session) + "_Angles_" + interpolation + "_degrees_" + str(rate_used) + "Hz"
00218                 plt.figure(figname)
00219                 plt.plot(angles_data_interpolated[:, "xr")
00220                 plt.title("Información de ángulos [°]. Sesión: " + str(session) + ".\n Interpolada con: " +
interpolation
+ " a " + str(rate_used) + "Hz")
00221                 ui.store_figure(figname, True)
00222
00223             # ----- STEP 5 -----
00224             Env.step("Label data by concatenating LFPs and interpolated Angles in a single 2D-array.")
00225
00226             # IF round_angles: Rounding the angles to be discrete labels, starting from 'base_angle' until 360 on
steps of
00227             # 'offset_between_angles'. Else Not rounding the angles to be discrete labels
00228             labeled_data = data.add_labels(lfp_data, np.expand_dims(angles_data_interpolated, axis=1), round_angles
base_angle, offset_between_angles)
00229
00230             if PLOT:
00231                 Env.print_text("Plotting LFP data from channels 0 and interpolated angles data at " + str(rate_used
) + "Hz. [°]")
00232                 figname = str(session) + "_LFP_C0_and_angles_" + interpolation + "_" + str(rate_used) + "Hz"
00233                 plt.figure(figname)
00234                 plt.subplot(211)
00235                 plt.plot(labeled_data[:, -1, 0], "xr")
00236                 plt.title("Señal LFP del Canal 0. Muestreada a " + str(rate_used) + "Hz. Sesión: " + str(session))
00237
00238                 plt.subplot(212)
00239                 plt.plot(labeled_data[:, -1], "xb")
00240                 plt.title("Información de ángulos [°]. Sesión: " + str(session) + ".\n Interpolada con: " +
interpolation
+ " a " + str(rate_used) + "Hz")
00241                 ui.store_figure(figname, True)
00242
00243             if str(session) == "771" and sync_method == "Downsample LFPs":
00244                 Env.print_text("\nAngles Data from: 15560 to 15600 where at 15566 the first LED is lost and at
15583 both are lost")
00245                 Env.print_text(' '.join([str(elem) for elem in angles_data[15560:15600]]))
00246
00247                 Env.print_text("\nAngles Data Interpolated from: 15560 to 15600 where at 15566 the first LED is
lost and at 15583 "
"both are lost.")
00248                 Env.print_text(' '.join([str(elem) for elem in labeled_data[15560:15600, -1]]))
00249
00250             # ----- STEP 6 -----
00251             Env.step("Clean the labeled dataset from invalid values.")
00252
00253
00254
00255
00256

```

```

00257     clean_dataset = data.clean_invalid_positional(labeled_data, sync_method == "Upsample Angles")
00258
00259     if round_angles:
00260         labels = np.arange(base_angle, 360, offset_between_angles)
00261         percentages = []
00262         for u in range(base_angle, 360, offset_between_angles):
00263             percentages.append(round(np.sum(clean_dataset[:, -1] == u)*100/len(clean_dataset[:, -1])))
00264
00265         labels_percent = np.concatenate((np.expand_dims(labels, axis=1), np.expand_dims(percentages, axis=1
)), axis=1)
00266         dataframe_labels = pd.DataFrame(data=labels_percent, columns=["Angulos", "Porcentaje"])
00267
00268         if PLOT:
00269             Env.print_text("Plotting Barplot of Labels after " + interpolation + " interpolation")
00270             figname = str(session) + "_BarPlotAngles_" + interpolation + "_" + str(rate_used) + "Hz"
00271             plt.figure(figname)
00272             sns.barplot(x="Angulos", y="Porcentaje", data=dataframe_labels)
00273             plt.title("Gráfico de Barras de las etiquetas. Sesión: " + str(session) + ".\n Interpolada con:
" +
00274                     interpolation + " a " + str(rate_used) + "Hz")
00275             ui.store_figure(figname, True)
00276
00277         # ----- STEP 7 -----
00278         Env.step("Count and print number of NaNs in the Dataset.")
00279
00280         nans_begin = 0
00281         nans_end = 0
00282         length = len(angles_data_interpolated)
00283         for i in range(length):
00284             if ~np.isnan(angles_data_interpolated[i]):
00285                 nans_begin = i
00286                 break
00287         for i in range(length-1, 0, -1):
00288             if ~np.isnan(angles_data_interpolated[i]):
00289                 nans_end = i
00290                 break
00291         nans_end = length - (nans_end + 1)
00292
00293         Env.print_text("Number of NaNs in Angles Data without interpolation: " + str(np.count_nonzero(np.isnan(
angles_data))))
00294         Env.print_text("Number of NaNs in Labeled Dataset with interpolated Angles Data: "
+ str(np.count_nonzero(np.isnan(angles_data_interpolated))))
00295         Env.print_text("Number of NaNs at the beginning of the interpolated Angles Data: " + str(nans_begin))
00296         Env.print_text("Number of NaNs at the end of the interpolated Angles Data: " + str(nans_end))
00297         Env.print_text("Number of NaNs in Labeled and Clean Dataset with interpolated Angles Data: "
+ str(np.count_nonzero(np.isnan(clean_dataset[0][:, -1]))))
00298
00299         if PLOT:
00300             # ----- STEP 8 -----
00301             Env.step("Plotting clean LFP data from channels 0 and interpolated angles data at " + str(rate_used
) + "Hz. [°]")
00302
00303             figname = str(session) + "_LFP_C0_clean_and_angles_" + interpolation + "_" + str(rate_used) + "Hz"
00304             plt.figure(figname)
00305             plt.subplot(211)
00306             plt.plot(clean_dataset[0][:, 0], "xr")
00307             plt.title("Señal LFP del Canal 0 limpia. Muestreada a " + str(rate_used) + "Hz. Sesión: " + str(
session))
00308             plt.subplot(212)
00309             plt.plot(clean_dataset[0][:, -1], "xb")
00310             plt.title("Información de ángulos [°] limpia. Sesión: " + str(session) + ".\n Interpolada con: "
+ interpolation + " a " + str(rate_used) + "Hz")
00311             ui.store_figure(figname, True)
00312
00313             # ----- STEP 9 -----
00314             Env.step("Plotting Boxplot of all channels in two figures, from 0-49 and from 50-98.")
00315
00316             figname = str(session) + "_BoxPlot0-49_" + interpolation + "_" + str(rate_used) + "Hz"
00317             plt.figure(figname, figsize=[12, 16])
00318             sns.boxplot(data=clean_dataset[0][:, 0:50], orient="h")
00319             plt.ylabel("Canales")
00320             plt.xlabel("Voltaje")
00321             plt.title("Diagrama de caja de los canales 0-49.\nSesión: " + str(session) + ". Interpolada con: "
+ interpolation + " a " + str(rate_used) + "Hz")
00322             ui.store_figure(figname, True)
00323
00324             figname = str(session) + "_BoxPlot50-98_" + interpolation + "_" + str(rate_used) + "Hz"
00325             plt.figure(figname, figsize=[12, 16])
00326             sns.boxplot(data=clean_dataset[0][:, 50:99], orient="h")
00327             plt.ylabel("Canales")
00328             plt.xlabel("Voltaje")
00329             plt.title("Diagrama de caja de los canales 50-98.\nSesión: " + str(session) + ". Interpolada con: "
+ interpolation + " a " + str(rate_used) + "Hz")
00330             ui.store_figure(figname, True)
00331
00332             figname = str(session) + "_BoxPlotChannels_" + interpolation + "_" + str(rate_used) + "Hz"
00333             fig = plt.figure(figname)

```

```

00339
00340     # ----- STEP 10 -----
00341     Env.step("Plotting Boxplot of interpolated angles.")
00342
00343     figname = str(session) + "_BoxPlotAngles_" + interpolation + "_" + str(rate_used) + "Hz"
00344     plt.figure(figname)
00345     sns.boxplot(x=clean_dataset[0][:, -1])
00346     plt.ylabel("Muestras")
00347     plt.xlabel("Ángulos")
00348     plt.title("Diagrama de caja de los ángulos.\nSesión: " + str(session) + ". Interpolada con: "
00349             + interpolation + " a " + str(rate_used) + "Hz")
00350     ui.store_figure(figname, True)
00351
00352     # ----- FINISH TEST AND EXIT -----
00353     Env.finish_test()
00354     #Env.finish_test(str(session) + "_" + interpolation.title() + "_" + sync_method.replace(" ", ""))
00355
00356 # ----- Execute Dataset generation -----
00357 dataExploration()

```

5.3 /home/pabloav/Documents/Tesis_Lic/LFP-Classification/DatasetGenerator/DatasetGenerator.py File Reference

Namespaces

- [DatasetGenerator](#)

Functions

- def [DatasetGenerator.data_set_generator](#) ()

5.3.1 Detailed Description

Author

Pablo Avila [B30724] jose.avilalopez@ucr.ac.cr

Copyright

MIT License

Date

July, 2021

Properly loads the input data and labels and prepare a clean dataset to finally export it as a pickle file.

Definition in file [DatasetGenerator.py](#).

5.4 DatasetGenerator.py

```

00001 # #####
00002 #           University of Costa Rica
00003 #           Electrical Engineering Department
00004 #           Grade Thesis
00005 # #####
00006
00007 """
00008 @file DatasetGenerator.py
00009 @author Pablo Avila [B30724] jose.avilalopez@ucr.ac.cr
00010 @copyright MIT License
00011 @date July, 2021
00012 @details Properly loads the input data and labels and prepare a clean dataset to finally export it as a

```

```

        pickle file.
00013 """
00014
00015 from Yomchi import \
00016     Environment as Env, \
00017     preprocessing as data, \
00018     visualization as ui
00019
00020 import matplotlib.pyplot as plt
00021 import numpy as np
00022 import tensorflow as tf
00023 import pickle
00024
00025
00026 def data_set_generator():
00027     """
00028     @details
00029     <h1> Dataset Generator:</h1>
00030     <h2> Setup </h2>
00031     <ul>
00032         <li> Initialize Environment.
00033         <li> Initialize Tensorflow session.
00034         <li> Set seed for Numpy and Tensorflow
00035         <li> Specify run configuration parameters.
00036         <li> Specify dataset generation parameters.
00037     </ul>
00038     <h2> Procedure </h2>
00039     <ul>
00040         <li> <b>Step 1:</b> Import LFP data.
00041         <li> <b>Step 2:</b> Import angles data.
00042         <li> <b>Step 3:</b>
00043         <ul>
00044             <li> Downsample LFP data to match the Angles sampling rate or
00045             <li> Fill angles gaps to match the LFP sampling rate.
00046         </ul>
00047         <li> <b>Step 4:</b> Label data by concatenating LFPs and Angles in a single 2D-array.
00048         <li> <b>Step 5:</b> Clean the labeled dataframe from -1 values, which represent the wrongly
00049         acquired
00050         positional samples.
00051         <li> <b>Step 6:</b> Interpolate angles data using an 'interpolation' approach.
00052         <li> <b>Step 7:</b> Plotting Angles and the selected channel of the LFPs after cleaning and
00053         interpolation.
00054         <li> <b>Step 8:</b>
00055         <ul>
00056             <li> Get preferred angle according to LFPs channel.
00057             <li> Plotting the sum of LFPs per angle from 0 to 359 degrees.
00058         </ul>
00059         <li> <b>Step 9:</b> Get largest subset of data.
00060         <li> <b>Step 10:</b> Split the subset into training, validation and testing set.
00061         <li> <b>Step 11:</b> Save the dataset to a pickle file.
00062         <li> <b>Step 12:</b> Convert data to windowed series.
00063         <li> <b>Finish Test and Exit:</b> Save logs, captures, results and the generated dataset into a
00064         .pickle file.
00065     </ul>
00066     """
00067
00068     # ----- SETUP -----
00069     Env.init_environment(True, True)
00070
00071     tf.keras.backend.clear_session()
00072     tf.random.set_seed(51)
00073     np.random.seed(51)
00074
00075     # Dataset Generation Parameters
00076     # The recording session 771 has a 23.81% of invalid positions, while the session 765 has only 6.98%
00077     session = 771 # 771 or 765
00078     interpolation = "Shortest" # "linear" "quadratic" "cubic" "nearest" "Shortest"
00079     sync_method = "Downsample LFPs" # "Upsample Angles" "Downsample LFPs"
00080
00081     # Data Properties
00082     num_channels = data.EC014_41_NUMBER_OF_CHANNELS
00083     rate_used = data.POSITION_DATA_SAMPLING_RATE
00084     if sync_method == "Upsample Angles":
00085         rate_used = data.LFP_DATAMAX_SAMPLING_RATE
00086
00087     # Windowing properties
00088     window_size = 4 # Recommended between 100ms to 200ms.
00089     batch_size = 32
00090     shuffle_buffer = 1000
00091     lfp_channel = 70
00092     round_angles = False
00093     base_angle = 0 # Unused if round_angles = False
00094     offset_between_angles = 30 # Unused if round_angles = False
00095
00096     extra = {"Round angles to get discrete label": round_angles}
00097     if round_angles:
00098         extra.update({"Angle labels starting from": str(base_angle) + "°",

```

```

00096         "until 360° in steps of": str(offset_between_angles) + "°"))
00097
00098     o_pickle_file_name = f"S-{session}_C{lfp_channel}_I-{interpolation}_F-{rate_used}_W-" \
00099         f"{int(window_size * 1e3 / rate_used)}ms.pickle"
00100
00101     parameters_dictionary = {"Recording Session Number": str(session),
00102                             "Interpolation Method": interpolation.title(),
00103                             "Synchronization Method": sync_method,
00104                             "Number of Recorded Channels": str(num_channels),
00105                             "Sampling Rate to Use": str(rate_used) + " Hz",
00106                             "Window Size": window_size,
00107                             "Batch size (# of windows)": batch_size,
00108                             "LFP Signal channel to use": lfp_channel,
00109                             "Shuffle buffer size": shuffle_buffer)
00110
00111     parameters_dictionary.update(extra)
00112     Env.print_parameters(parameters_dictionary)
00113
00114     # ----- STEP 1 -----
00115     Env.step(f"Importing LFP data from session: {session}")
00116     lfp_data = data.load_lfp_data(data.LFP[session])
00117
00118     # ----- STEP 2 -----
00119     Env.step(f"Importing angles data from session: {session}")
00120     angles_data = data.load_angles_data(data.ANGLES[session])
00121
00122     # ----- STEP 3 -----
00123     if sync_method == "Downsample LFPs":
00124         Env.step("Downsample LFP data to match Angles sampling rate.")
00125         lfp_data = data.downsample_lfps(lfp_data, data.LFP_DATAMAX_SAMPLING_RATE,
data.POSITION_DATA_SAMPLING_RATE)
00126
00127     elif sync_method == "Upsample Angles":
00128         Env.step("Upsample Angles data to reach a higher sampling rate")
00129         angles_data = data.angles_expansion(angles_data, data.POSITION_DATA_SAMPLING_RATE,
data.LFP_DATAMAX_SAMPLING_RATE)
00130
00131     # ----- STEP 4 -----
00132     Env.step("Label data by concatenating LFPs and interpolated Angles in a single 2D-array.")
00133     # IF round_angles: Rounding the angles to be discrete labels, starting from 'base_angle' until 360 on
00134     steps of
00135     # 'offset_between_angles'. Else, the angles are not rounded to discrete labels.
00136     labeled_data = data.add_labels(lfp_data, np.expand_dims(angles_data, axis=1), round_angles,
00137                                   base_angle, offset_between_angles)
00138
00139     # ----- STEP 5 -----
00140     Env.step("Clean the labeled dataset from discontinuities in positional data (angles).")
00141
00142     clean_datasets = data.clean_invalid_positional(labeled_data)
00143
00144     # ----- STEP 6 -----
00145     Env.step(f"Interpolate angles data using a {interpolation} approach.")
00146
00147     # Get all LFP channels, excluding the angles.
00148     clean_interpolated_data = [s[:, :-1] for s in clean_datasets]
00149
00150     # Interpolate angles and add them to the dataset
00151     clean_interpolated_angles = [data.interpolate_angles(s[:, -1], interpolation) for s in clean_datasets]
00152
00153     for i in range(len(clean_interpolated_data)):
00154         clean_interpolated_data[i] = \
00155             np.concatenate((clean_interpolated_data[i], np.expand_dims(clean_interpolated_angles[i], axis=1
)), axis=1)
00156
00157     # ----- STEP 7 -----
00158     Env.step("Plotting Angles and one channel of the LFPs after cleaning and interpolation.")
00159
00160     Env.print_text(f"Plotting Angles data [°] at {rate_used}Hz")
00161     figname = f"{session}_Angles_degrees_{int(rate_used)}Hz"
00162     plt.figure(figname)
00163     plt.plot(clean_interpolated_data[4][:, -1], "xr")
00164     plt.title(f"Información de ángulos [°]. Sesión: {session} a {rate_used}Hz")
00165     ui.store_figure(figname, show=Env.debug)
00166
00167     Env.print_text(f"Plotting LFPs Channel {lfp_channel} at {rate_used}Hz")
00168
00169     figname = f"{session}_LFP_c{lfp_channel}_{int(rate_used)}Hz"
00170     plt.figure(figname)
00171     plt.plot(clean_interpolated_data[4][:, lfp_channel], "x-r")
00172     plt.title(f"LFPs del Canal {lfp_channel}. Sesión: {session} a {rate_used}Hz")
00173     ui.store_figure(figname, show=Env.debug)
00174
00175     # ----- STEP 8 -----
00176     Env.step("Get Preferred angle.")
00177
00178     angles = [0] * 361
00179

```



```

00180     for subset in clean_interpolated_data:
00181         for index in range(len(subset)):
00182             angles[int(round(subset[index, -1]))] += abs(subset[index, lfp_channel])
00183
00184     angles[0] += angles[360]
00185     angles = angles[:-1]
00186     preferred_angle = angles.index(max(angles))
00187
00188     Env.print_text(f" Preferred angle according to LFP Channel {lfp_channel}: {preferred_angle}°")
00189
00190     Env.print_text(f"Plotting Preferred angle according to LFP Channel {lfp_channel}")
00191     figname = f"{session}_preferred_angle_LFP_c{lfp_channel}_{int(rate_used)}Hz"
00192     plt.figure(figname)
00193     plt.plot(range(360), angles, "x-r")
00194     plt.title(f"Suma de LFPs del Canal {lfp_channel} por ángulo. Sesión: {session}.")
00195     plt.xlabel(f"Ángulo en grados")
00196     ui.store_figure(figname, show=Env.debug)
00197
00198     # ----- STEP 9 -----
00199     Env.step("Get largest subset of data.")
00200
00201     max_length = 0
00202     max_subset_index = 0
00203     for index, subset in enumerate(clean_interpolated_data):
00204         if max_length < len(subset):
00205             max_length = len(subset)
00206             max_subset_index = index
00207
00208     largest_subset = clean_interpolated_data[max_subset_index]
00209     Env.print_text(f"Shape of the largest subset of data : [{len(largest_subset)}, {len(largest_subset[0])}]")
00210
00211     # ----- STEP 10 -----
00212     Env.step()
00213
00214     n = len(largest_subset)
00215     train_array = largest_subset[0:int(n * 0.7), :]
00216     valid_array = largest_subset[int(n * 0.7):int(n * 0.9), :]
00217     test_array = largest_subset[int(n * 0.9):, :]
00218
00219     Env.print_text(f"Training data shape: [{len(train_array)}, {len(train_array[0])}]")
00220     Env.print_text(f"Validation data shape: [{len(valid_array)}, {len(valid_array[0])}]")
00221     Env.print_text(f"Test data shape: [{len(test_array)}, {len(test_array[0])}]")
00222
00223     # ----- STEP 11 -----
00224     Env.step(f"Save the dataset to pickle file: {o_pickle_file_name}.")
00225
00226     with open(f"{Env.RESULTS_FOLDER}/{o_pickle_file_name}", 'wb') as f:
00227         pickle.dump([train_array, valid_array, test_array], f)
00228
00229     # ----- STEP 12 -----
00230     Env.step("Convert data to windowed series.")
00231
00232     train_data = data.channels_to_windows(train_array, lfp_channel, window_size, batch_size, shuffle_buffer)
00233
00234     val_data = data.channels_to_windows(valid_array, lfp_channel, window_size, batch_size, shuffle_buffer)
00235     test_data = data.channels_to_windows(test_array, lfp_channel, window_size, batch_size, shuffle_buffer)
00236
00237     # The shape should be (batch, time, features) to be compatible with what tensorflow expects as default.
00238     for example_inputs, example_labels in train_data.take(1):
00239         Env.print_text(f'Inputs shape (batch, time, samples): {example_inputs.shape}')
00240         Env.print_text(f'Labels shape (batch, time, labels): {example_labels.shape}')
00241
00242     # ----- FINISH TEST AND EXIT -----
00243     Env.finish_test()
00244     #Env.finish_test(str(session) + "_" + interpolation.title() + "_" + sync_method.replace(" ", ""))
00245
00246     # ----- Execute Dataset generation -----
00247     data_set_generator()

```

5.5 /home/pabloav/Documents/Tesis_Lic/LFP-Classification/Predictor/Predictor.py File Reference

Namespaces

- [Predictor](#)

Functions

- def [Predictor.predictor\(\)](#)

5.5.1 Detailed Description

Author

Pablo Avila [B30724] jose.avilalopez@ucr.ac.cr

Copyright

MIT License

Date

July, 2021

The predictor loads a dataset of LFP with Angles as labels previously generated and feed it to a Neural Network in order to train it to predict new angles based on LFP data windows.

Definition in file [Predictor.py](#).

5.6 Predictor.py

```

00001 # #####
00002 #           University of Costa Rica
00003 #           Electrical Engineering Department
00004 #           Grade Thesis
00005 # #####
00006
00007 """
00008 @file Predictor.py
00009 @author Pablo Avila [B30724] jose.avilalopez@ucr.ac.cr
00010 @copyright MIT License
00011 @date July, 2021
00012 @details The predictor loads a dataset of LFP with Angles as labels previously generated and feed it to a
00013 Neural
00014 Network in order to train it to predict new angles based on LFP data windows.
00015 """
00016 from Yomchi import \
00017     Environment as Env, \
00018     preprocessing as data, \
00019     models, \
00020     visualization as ui
00021
00022 import numpy as np
00023 import tensorflow as tf
00024 import matplotlib.pyplot as plt
00025 import pickle
00026
00027
00028 def predictor():
00029     """
00030     @details
00031     <h1> Predictor </h1>
00032     <h2> Experiment Setup </h2>
00033     <ul>
00034         <li> Initialize Environment
00035         <li> Initialize Tensorflow session.
00036         <li> Set seed for Numpy and Tensorflow
00037         <li> Specify run configuration parameters.
00038         <li> Specify run configuration parameters.
00039         <li> Specify dataset generation parameters.
00040         <li> Specify model's training parameters.
00041     </ul>
00042     <h2> Procedure </h2>
00043     <ul>
00044         <li> <b>Step 1:</b> Import training, validation and test datasets of LFPs as inputs and Angles as
00045         labels.
00046         <li> <b>Step 2:</b> Convert data to windowed series.

```

```

00046         <li> <b>Step 3:</b> Create the model with the specified parameters
00047         <li> <b>Step 4:</b> Train the model with the training and validation data.
00048         <li> <b>Step 5:</b> Plotting the original angular data vs the predictions for a determined number
of batches
00049         <li> <b>Finish Test and Exit:</b> Save logs, captures and results.
00050     </ul>
00051     """
00052
00053     # ----- SETUP -----
00054     Env.init_environment(True, True)
00055
00056     tf.keras.backend.clear_session()
00057     tf.keras.backend.set_floatx('float64')
00058     tf.random.set_seed(51)
00059     np.random.seed(51)
00060
00061     # Session, data and methods Parameters
00062     session = 771 # 765 or 771
00063     interpolation = "Shortest" # or "linear", "quadratic", "cubic", "nearest", "Shortest"
00064     rate_used = data.POSITION_DATA_SAMPLING_RATE # or data.POSITION_DATA_SAMPLING_RATE
00065
00066     # Windowing properties
00067     window_size = 4 # Equals to 100ms at 1250Hz. Recommended between 100ms to 200ms
00068     batch_size = 32
00069     shuffle_buffer = 1000
00070     lfp_channel = 70
00071     batches_to_plot = 20
00072
00073     # Parameters
00074     units_per_layer = 32
00075     dropout = None
00076     # dropout = 0.60
00077     layers = 3
00078     epochs = 20
00079     model_name = "LSTM" # "LSTM" "MLP" or "CNN"
00080
00081     # Input pickle file.
00082     i_pickle_file_name = f"
S-{session}_C{lfp_channel}_I-{interpolation}_F-{rate_used}_W-{int(window_size*1e3/rate_used)}" \
f"ms.pickle"
00083
00084     parameters_dictionary = {"Recording Session Number": str(session),
00085                             "Interpolation Method": interpolation.title(),
00086                             "Sampling Rate to Use": str(rate_used) + "Hz",
00087                             "LFP Signal channel to use": lfp_channel,
00088                             "Shuffle buffer size": str(shuffle_buffer),
00089                             "Pickle file used": i_pickle_file_name,
00090                             "Batches to Plot": batches_to_plot,
00091                             "Batch size (# of windows)": batch_size,
00092                             "Window Size": window_size,
00093                             "Model Name": model_name,
00094                             "Layers" : str(layers),
00095                             "Epochs" : str(epochs),
00096                             "Units per Layer" : str(units_per_layer),
00097                             "Dropout Regularization %": str(dropout)
00098                             }
00099
00100     Env.print_parameters(parameters_dictionary)
00101
00102     # ----- STEP 1 -----
00103     Env.step("Import training, validation and test datasets of LFPs as inputs and Angles as labels.")
00104
00105     with open(f"{data.PATH_TO_DATASETS}Pickles/{i_pickle_file_name}", "rb") as f:
00106         train_array, valid_array, test_array = pickle.load(f)
00107
00108     # ----- STEP 2 -----
00109     Env.step("Convert data to windowed series.")
00110
00111     # The train data is shuffled so the network is trained with a variety of examples.
00112     train_data = data.channels_to_windows(train_array, lfp_channel, window_size, batch_size, shuffle_buffer
)
00113
00114     # The validation and testing data are not shuffled because the intention is to predict the series in
order.
00115     val_data = data.channels_to_windows(valid_array, lfp_channel, window_size, batch_size)
00116     test_data = data.channels_to_windows(test_array, lfp_channel, window_size, batch_size)
00117
00118     for example_inputs, example_labels in train_data.take(1):
00119         Env.print_text(f'Inputs shape (batch, time, channels): {example_inputs.shape}')
00120         Env.print_text(f'Labels shape (batch, time, labels): {example_labels.shape}')
00121
00122     # ----- STEP 3 -----
00123     Env.step("Create the model with the specified parameters ")
00124
00125     if model_name == "LSTM":
00126         model = models.lstm(units_per_layer)
00127     elif model_name == "CNN":
00128         model = models.cnn(window_size, units_per_layer)

```

```

00129     else:
00130         model = models.mlp(layers, units_per_layer, dropout)
00131
00132     for example_inputs, example_labels in train_data.take(1):
00133         Env.print_text(f'Input shape: {example_inputs.shape}')
00134         Env.print_text(f'Output shape: {model(example_inputs).shape}')
00135
00136     # ----- STEP 4 -----
00137     Env.step("Train the model with the training and validation data.")
00138
00139     history = models.compile_and_fit(model, train_data, val_data, epochs=epochs,
00140                                     patience=epochs)
00141
00142     Env.print_text(f"\nTraining metrics history:")
00143     Env.print_text(f"Loss: {history.history['loss']}")
00144     Env.print_text(f"Mean Absolute Error: {history.history['mean_absolute_error']}")
00145     Env.print_text(f"\nValidation metrics history:")
00146     Env.print_text(f"Loss: {history.history['val_loss']}")
00147     Env.print_text(f"Mean Absolute Error: {history.history['val_mean_absolute_error']}")
00148
00149     # ----- STEP 5 -----
00150     Env.step(f"Plotting the original angular data vs the predictions of {batches_to_plot} batches")
00151
00152     lfps = []
00153     real = []
00154     pred = []
00155     for inputs, label in test_data.take(batches_to_plot):
00156         # Getting original data
00157         real = np.append(real, label[:, 0, 0].numpy())
00158         lfps = np.append(lfps, inputs[:, 0, 0].numpy())
00159         # Getting predicted data
00160         pred = np.append(pred, model.predict(inputs))
00161
00162     # Plotting
00163     figname = f"predictions_{model_name}_S{session}"
00164     plt.figure(figname, figsize=(12, 8))
00165     plt.subplot(211)
00166     plt.title(f"Ángulos calculados contra originales")
00167     plt.ylabel('Ángulos')
00168     plt.scatter(range(len(real)), real, edgecolors='k', label='Originales', c='#2ca02c', s=16)
00169     plt.scatter(range(len(pred)), pred, marker='X', edgecolors='k', label='Predicciones', c='#ff7f0e', s=16)
00170
00171     plt.legend()
00172
00173     plt.subplot(212)
00174     plt.title(f'Canal de LFP: {lfp_channel}')
00175     #plt.plot(lfps, "-")
00176     plt.scatter(range(len(lfps)), lfps, edgecolors='k', label='LFPs', c='#0a7fdb', s=16)
00177     plt.legend()
00178     ui.store_figure(figname, show=Env.debug)
00179
00180     # ----- FINISH TEST AND EXIT -----
00181     Env.finish_test()
00182     # Env.finish_test(f"M-MLP_S-{session}_I-{interpolation}_F-{rate_used}")
00183
00184     # ----- Execute Dataset generation -----
00185     predictor()

```

5.7 /home/pabloav/Documents/Tesis_Lic/LFP-Classification/README.md File Reference

5.8 /home/pabloav/Documents/Tesis_Lic/LFP-Classification/README.md

```

00001 # LFP Classification
00002 Classifier of Local Field Potential signals using Deep Neural Networks for head angular direction
00003 prediction
00004
00005 ## Prerequisites
00006 The current project was developed using **Ubuntu 18.04.4 LTS**.
00007 The following are the required packages and python libraries.
00008 It is recommended to use the listed versions, nevertheless is worth to try the newest available
00009 version of them.
00010
00011 ### Packages:
00012 ```
00013 1. Python (3.6)
00014 ```
00015
00016 **Optional:**
00017 ```
00018 1. Latex

```

```

00017 2. Doxygen
00018 3. Doxypy
00019 4. Make
00020 5. Git
00021 6. Graphviz
00022 ```
00023
00024 ### Python libraries:
00025 Package | Version
00026 ----- | -----
00027 pip3 | 20.1
00028 numpy | 1.18.4
00029 matplotlib | 3.2.1
00030 tensorflow | 2.2.0
00031 keras | 2.3.1
00032 sklearn | 0.0
00033 pandas | 1.0.3
00034 scikit-learn | 0.23.0
00035 seaborn | 0.10.1
00036
00037
00038 ## Install
00039
00040 Check the installed Linux version using: `lsb_release -a`.
00041 Check the installed version of the package using: `<package> --<version>`
00042 Check the installed python libraries and their version using: `pip list`
00043
00044 ### Packages:
00045 ```
00046 sudo apt-get install python3.6 texlive-latex-extra doxygen doxygen-gui make git graphviz
00047 ```
00048
00049 ### Pip:
00050
00051 This is necessary to install the rest of python libraries
00052
00053 ```
00054 curl https://bootstrap.pypa.io/get-pip.py -o get-pip.py
00055 python3.6 get-pip.py
00056 ```
00057
00058 ### Python libraries:
00059
00060 If getting the latest available version:
00061 ```
00062 pip3 install --upgrade <package>
00063 ```
00064
00065 To download an specific version:
00066 ```
00067 pip3 install <package>==<version>
00068 ```
00069
00070 ## Usage
00071
00072
00073 ## Author
00074 **Pablo Avila** - [PabloAvLo] (https://github.com/PabloAvLo)
00075
00076 ## License
00077 This project is licensed under the MIT License - see the LICENSE.md file for details
00078
00079 ## Acknowledgments
00080
00081 * The present code is part of my grade thesis in Electrical Engineering at the University of Costa Rica.
00082
00083 * The main objective is to prove the hypothesis that based on Local Field Potential measurments from the Entorhinal Cortex of the Hippocampus of rats is possible to extract angular information of the head direction of the subject.
00084
00085 * The dataset used in this experiment is: [BuszakiLab HC-3 Dataset] (https://crcns.org/data-sets/hc/hc-3/about-hc-3/?searchterm=LFP).
00086
00087 * In order to do so, machine learning algorithms will be applied to the data, such as Feedforward Neural Network, RNNs and CNNs.

```

5.9 /home/pabloav/Documents/Tesis_Lic/LFP-Classification/Yomchi/Environment.py File Reference

Namespaces

- [Environment](#)

Functions

- def [Environment.init_environment](#) (print_the_header=False, enable_debug=False)
Initialize the directories and files to log the results.
- def [Environment.print_text](#) (text)
Print a passed text and save it in the yomchi_log.txt.
- def [Environment.print_box](#) (text)
Print a passed text in a box and save it in the yomchi_log.txt.
- def [Environment.step](#) (description=None, new_step_number=0)
Print the current step number with a brief description.
- def [Environment.print_parameters](#) (params_dictionary)
Print a list of parameters.
- def [Environment.log_versions](#) ()
Stores the environment versions including packages, libraries and OS.
- def [Environment.print_header](#) ()
Print a header in the console output and log files.
- def [Environment.finish_test](#) (rename_results_folder=None)
Safely finish the test run and log some final information.

Variables

- [Environment.START_TIME](#) = datetime.datetime.now()
- [Environment.CURRENT_FOLDER](#) = os.getcwd()
- string [Environment.RESULTS_FOLDER](#) = CURRENT_FOLDER + "/Results-" + START_TIME.strftime("%Y-%m-%d_%H-%M") + "/"
- string [Environment.LOGS_FOLDER](#) = RESULTS_FOLDER + "Logs/"
- string [Environment.CAPTURES_FOLDER](#) = RESULTS_FOLDER + "Captures/"
- string [Environment.VERSIONS_LOG_PATH](#) = LOGS_FOLDER + "versions_log.txt"
- string [Environment.YOMCHI_LOG_PATH](#) = LOGS_FOLDER + "yomchi_log.txt"
- int [Environment.LINE_LENGTH](#) = 100
- int [Environment.step_number](#) = 1
- [Environment.versions_log](#) = None
- [Environment.yomchi_log](#) = None
- [Environment.caller_file](#) = None
- bool [Environment.debug](#) = False

5.9.1 Detailed Description

Author

Pablo Avila [B30724] jose.avilalopez@ucr.ac.cr

Copyright

MIT License

Date

May, 2020

Python environment which integrates data pre-processing, ML models implementation, data visualization and metrics. This module contains a set of functions to successfully document every test run by creating Results folders with console logs, generated images, dependencies versions, etc.

Definition in file [Environment.py](#).

5.10 Environment.py

```

00001 # #####
00002 #           University of Costa Rica
00003 #           Electrical Engineering Department
00004 #           Grade Thesis
00005 # #####
00006
00007 """
00008 @file Environment.py
00009 @author Pablo Avila [B30724] jose.avilalopez@ucr.ac.cr
00010 @copyright MIT License
00011 @date May, 2020
00012 @details Python environment which integrates data pre-processing, ML models implementation, data
00013          visualization and
00014          metrics. This module contains a set of functions to successfully document every test run by creating
00015          Results folders
00016          with console logs, generated images, dependencies versions, etc.
00017 """
00018
00019 import os
00020 import sys
00021 import inspect
00022 import platform
00023 import datetime
00024 import numpy as np
00025
00026 # Module Constants
00027 START_TIME = datetime.datetime.now()
00028 CURRENT_FOLDER = os.getcwd()
00029 RESULTS_FOLDER = CURRENT_FOLDER + "/Results-" + START_TIME.strftime("%Y-%m-%d_%H-%M") + "/"
00030 LOGS_FOLDER = RESULTS_FOLDER + "Logs/"
00031 CAPTURES_FOLDER = RESULTS_FOLDER + "Captures/"
00032 VERSIONS_LOG_PATH = LOGS_FOLDER + "versions_log.txt"
00033 YOMCHI_LOG_PATH = LOGS_FOLDER + "yomchi_log.txt"
00034 LINE_LENGTH = 100
00035
00036 # Module Variables
00037 step_number = 1
00038 versions_log = None
00039 yomchi_log = None
00040 caller_file = None
00041 debug = False
00042
00043 def init_environment(print_the_header=False, enable_debug=False):
00044     """
00045     Initialize the directories and files to log the results.
00046     @param print_the_header: If True, print the header
00047     @param enable_debug: If True, run some code to help debugging
00048     """
00049     global versions_log
00050     global yomchi_log
00051     global caller_file
00052     global debug
00053
00054     debug = enable_debug
00055     os.makedirs(REULTS_FOLDER)
00056     os.makedirs(LOGS_FOLDER)
00057     os.makedirs(CAPTURES_FOLDER)
00058
00059     # First get the full filename (including path and file extension)
00060     caller_frame = inspect.stack()[1]
00061     caller_filename = caller_frame.filename
00062
00063     # Now get rid of the directory and extension
00064     caller_file = os.path.basename(caller_filename)
00065
00066     yomchi_log = open(YOMCHI_LOG_PATH, "w+")
00067     versions_log = open(VERSIONS_LOG_PATH, "w+")

```

```

00068
00069     if print_the_header:
00070         print_header()
00071
00072     log_versions()
00073
00074
00075 def print_text(text):
00076     """
00077     Print a passed text and save it in the yomchi_log.txt.
00078     @param text: Text to print
00079     @return None
00080     """
00081     global yomchi_log
00082
00083     print(text)
00084     yomchi_log.write(text + "\n")
00085
00086
00087 def print_box(text):
00088     """
00089     Print a passed text in a box and save it in the yomchi_log.txt.
00090     @param text: Text to print
00091     @return None
00092     """
00093     text_length = len(text)
00094     counter = 0
00095     for i in range(0, text_length):
00096         if text[i] == '\n':
00097             counter = 0
00098         else:
00099             counter += 1
00100
00101         if counter == LINE_LENGTH:
00102             if text[i] in ['a', 'e', 'i', 'o', 'u']:
00103                 text = text[:i - 1] + '-' + '\n' + text[i - 1:]
00104             else:
00105                 text = text[:i] + '\n' + text[i:]
00106             counter = 0
00107
00108     text_array = text.splitlines()
00109     print_text("\n" + "-" * (LINE_LENGTH + 2) + "|")
00110     for line in text_array:
00111         print_text("| " + line.ljust(LINE_LENGTH, ' ') + " |")
00112     print_text("|" + "-" * (LINE_LENGTH + 2) + "|")
00113
00114
00115 def step(description=None, new_step_number=0):
00116     """
00117     Print the current step number with a brief description.
00118     @param description: Description of the step.
00119     @param new_step_number: If defined, reset the steps numeration to the passed value.
00120     @return None
00121     """
00122     global step_number
00123
00124     larger_string = 7
00125     pad = round((LINE_LENGTH - larger_string) / 2)
00126
00127     if new_step_number != 0:
00128         step_number = new_step_number
00129
00130     text = " " * pad + "Step " + str(step_number)
00131     if description is not None:
00132         text = text + "\n" + description
00133
00134     print_box(text)
00135     step_number += 1
00136
00137
00138 def print_parameters(params_dictionary):
00139     """
00140     Print a list of parameters.
00141     @param params_dictionary: Dictionary of parameters {name: value} to print.
00142     @return None
00143     """
00144
00145     pad = round((LINE_LENGTH - 18) / 2)
00146     title = " " * pad + "*** Parameters ***"
00147
00148     names = list(params_dictionary.keys())
00149     values = list(params_dictionary.values())
00150     longest_name = max(names, key=len)
00151
00152     new_names = []
00153     for name in names:
00154         new_names.append(name + " " * (len(longest_name) - len(name)) + " : ")

```



```

00155
00156     new_dictionary = np.char.add(new_names, values)
00157
00158     print_text("\n|" + "-" * (LINE_LENGTH + 2) + "|")
00159     print_text("| " + title.ljust(LINE_LENGTH, ' ') + " |")
00160     print_text("| " + ''.ljust(LINE_LENGTH, ' ') + " |")
00161     for param in new_dictionary:
00162         print_text("| " + param.ljust(LINE_LENGTH, ' ') + " |")
00163     print_text("|" + "-" * (LINE_LENGTH + 2) + "|")
00164
00165
00166 def log_versions():
00167     """
00168     Stores the environment versions including packages, libraries and OS.
00169     @return None
00170     """
00171     global versions_log
00172     required_packages = ["Keras", "numpy", "pandas", "matplotlib", "scikit-learn", "seaborn",
00173                          "sklearn", "tensorflow", "pip"]
00174
00175     try:
00176         from pip._internal.operations import freeze
00177     except ImportError: # pip < 10.0
00178         from pip.operations import freeze
00179
00180     try:
00181         distro = platform.linux_distribution()
00182     except:
00183         distro = ["N/A", ""]
00184
00185     versions_log.write("\n\nOperation System: ")
00186     versions_log.write("\n Kernel: " + platform.system() + " " + platform.release())
00187     versions_log.write("\n Distribution: " + distro[0] + " " + distro[1])
00188
00189     versions_log.write("\n\nPython version: " + str(sys.version_info[0]) + "."
00190                       + str(sys.version_info[1]) + "." + str(sys.version_info[2]))
00191     versions_log.write("\n\nPackages versions:\n")
00192
00193     list = freeze.freeze()
00194
00195     versions_log.write(" Required Packages:")
00196     for package in list:
00197         for required in required_packages:
00198             if package.find(required) != -1:
00199                 index = package.find("==")
00200                 package = package.replace("==", " " * (25 - index) + "= ")
00201                 versions_log.write("\n      " + package)
00202
00203     list = freeze.freeze()
00204     versions_log.write("\n\n Complete List:")
00205     for package in list:
00206         index = package.find("==")
00207         package = package.replace("==", " " * (25 - index) + "= ")
00208         versions_log.write("\n      " + package)
00209
00210
00211 def print_header():
00212     """
00213     Print a header in the console output and log files.
00214     @return None
00215     """
00216     larger_string = 33
00217     pad = round((LINE_LENGTH - larger_string) / 2)
00218     header = "#" * LINE_LENGTH + "#" * 3 \
00219         + "\n#" + " " * pad + " " + " " * pad + "#" \
00220         + "\n#" + " " * pad + " University of Costa Rica " + " " * pad + "#" \
00221         + "\n#" + " " * pad + "Electrical Engineering Department" + " " * pad + "#" \
00222         + "\n#" + " " * pad + "Grade Thesis " + " " * pad + "#" \
00223         + "\n#" + " " * pad + "Pablo Avila [B30724] " + " " * pad + "#" \
00224         + "\n#" + " " * pad + "jose.avilalopez@ucr.ac.cr " + " " * pad + "#" \
00225         + "\n#" + " " * pad + " " + " " * pad + "#" \
00226         + "\n" + "#" * LINE_LENGTH + "#" * 3 \
00227         + "\n" + " " * pad + " *** START OF TEST *** " \
00228         + "\n# File : " + caller_file \
00229         + "\n# Date : " + START_TIME.strftime("%m-%d-%Y") \
00230         + "\n# Start Time: " + START_TIME.strftime("%H:%M:%S") + "\n"
00231     print_text(header)
00232     versions_log.write(header)
00233
00234 def finish_test(rename_results_folder=None):
00235     """
00236     Safely finish the test run and log some final information.
00237     @param rename_results_folder: Optional name for the results folder.
00238     @return None
00239     """
00240     global yomchi_log
00241     global versions_log

```

```

00242
00243     larger_string = 33
00244     pad = round((LINE_LENGTH - larger_string) / 2)
00245
00246     finish_time = datetime.datetime.now()
00247     elapsed_time = finish_time - START_TIME
00248
00249     days = elapsed_time.days
00250     hours, rem = divmod(elapsed_time.seconds, 3600)
00251     minutes, seconds = divmod(rem, 60)
00252
00253     footer = "\n" + "#" * LINE_LENGTH + "#" * 3 \
00254             + "\n" + " " * pad + "    *** END OF TEST ***  " \
00255             + "\n# Test Duration : " + str(days) + str(hours) + ":" + str(minutes) + ":" + str(seconds) \
00256             + "\n# Finish Time    : " + finish_time.strftime("%H:%M:%S") \
00257             + "\n" + "#" * LINE_LENGTH + "#" * 3
00258
00259     print_text(footer)
00260
00261     yomchi_log.close()
00262     versions_log.close()
00263
00264     if rename_results_folder is not None:
00265         os.rename(RESULTS_FOLDER, CURRENT_FOLDER + "/" + rename_results_folder + "/")
00266
00267     sys.exit()

```

5.11 /home/pabloav/Documents/Tesis_Lic/LFP-Classification/Yomchi/models.py File Reference

Namespaces

- [models](#)

Functions

- def [models.mlp](#) (layers, units_per_layer, dropout=None)
Defines a classical neural network sometimes called: Multi-Layer Perceptron.
- def [models.cnn](#) (inputs, units_per_layer)
Creates a model of a Convolutional Neural Network with a Conv1D as the input layer, one dense as the only hidden layer and another dense with only 1 neuron as the output layer.
- def [models.lstm](#) (units_per_layer)
Creates a model of a Long Short-Term Memory Neural Network as the input layer, one dense with only 1 neuron as the output layer.
- def [models.compile_and_fit](#) (model, train, val, epochs=20, patience=2)
Fits the provided training data in the specified model and train's it.

5.11.1 Detailed Description

Author

Pablo Avila [B30724] jose.avilalopez@ucr.ac.cr

Copyright

MIT License

Date

July, 2021

This module contains a set of functions to generate Machine Learning models such as Feed-forward Neural Networks, Long Short-Term Memory Neural Network (LSTM), Convolutional Neural Networks (CNN) and more.

Definition in file [models.py](#).

5.12 models.py

```

00001 # #####
00002 #             University of Costa Rica
00003 #             Electrical Engineering Department
00004 #             Grade Thesis
00005 # #####
00006
00007 """
00008 @file models.py
00009 @author Pablo Avila [B30724] jose.avilalopez@ucr.ac.cr
00010 @copyright MIT License
00011 @date July, 2021
00012 @details This module contains a set of functions to generate Machine Learning models such as Feed-forward
          Neural
00013 Networks, Long Short-Term Memory Neural Network (LSTM), Convolutional Neural Networks (CNN) and more.
00014 """
00015 import Yomchi.Environment as Env
00016 import tensorflow as tf
00017
00018
00019 def mlp(layers, units_per_layer, dropout=None):
00020     """
00021     Defines a classical neural network sometimes called: Multi-Layer Perceptron. It is Feedforward and
00022     fully-connected,
00023     with the specified parameters. The activation function of the hidden layers is ReLU, and the activation
00024     of the
00025     output is linear.
00026     @param layers: Number of hidden layers (besides the input and outputs ones).
00027     @param units_per_layer: Number of neurons per layer. Applies to all layers except for the last one
00028     which has only 1:
00029     the predicted angle.
00030     @param dropout: A value between 0 and 1 of neuron's results to discard of the training. If provided,
00031     two layers of
00032     this regularization method will be added to the model. One after the input layer and one before the
00033     output layer.
00034     @return model: The model of the MLP created for later usage as the predictor.
00035     """
00036
00037     Env.print_text(f"Creating a fully-connected feed-forward Neural Network model with {layers} layers,
00038     using "
00039                   f"{units_per_layer} units per layer. \n\nThe activation function in the hidden layers is '
00040                   ReLu' and "
00041                   f"the output layer has only 1 unit (the predicted angle).")
00042     model = tf.keras.Sequential()
00043
00044     # Shape: (time, features) => (time*features)
00045     model.add(tf.keras.layers.Flatten())
00046
00047     # First layer (need to specify the input size)
00048     model.add(tf.keras.layers.Dense(
00049         units=units_per_layer,
00050         activation=tf.nn.relu))
00051
00052     if dropout is not None:
00053         model.add(tf.keras.layers.Dropout(dropout))
00054
00055     # Other hidden layers
00056     for n in range(1, layers):
00057         model.add(tf.keras.layers.Dense(
00058             units=units_per_layer,
00059             activation=tf.nn.relu))
00060
00061     if dropout is not None:
00062         model.add(tf.keras.layers.Dropout(dropout))
00063
00064     # Output layer
00065     model.add(tf.keras.layers.Dense(
00066         units=1
00067     ))
00068
00069     # Add back the time dimension.

```

```

00063     # Shape: (outputs) => (1, outputs)
00064     model.add(tf.keras.layers.Reshape([1, -1]))
00065
00066     return model
00067
00068
00069 def cnn(inputs, units_per_layer):
00070     """
00071     Creates a model of a Convolutional Neural Network with a Conv1D as the input layer, one dense as the
00072     only hidden
00073     layer and another dense with only 1 neuron as the output layer. The first two layer has ReLU activation
00074     and the last
00075     one uses a linear activation function.S
00076     @param inputs: Number of inputs of the network. It is used as the kernel size with the intention that
00077     the 1D
00078     convolutional layer outputs a single value throughout the specified number of filters.
00079     @param units_per_layer: Specified the number of neurons of the hidden dense layer, which has to match
00080     with the
00081     number of filters that will output a result in the 1D convolutional input layer.
00082     @return conv_model: The model of the CNN created for later usage as the predictor.
00083     """
00084     Env.print_text(f"Creating a Convolutional Neural Network model with one 1D convolutional layer, using "
00085                    f"{units_per_layer} filters and units in the following dense layer. \nThe activation
00086                    function in "
00087                    f"the dense layer is 'ReLU' and the output dense layer has only 1 unit (the predicted
00088                    angle).")
00089
00090     conv_model = tf.keras.Sequential([
00091         tf.keras.layers.Conv1D(filters=units_per_layer,
00092                                kernel_size=(inputs,),
00093                                activation='relu'),
00094         tf.keras.layers.Dense(units=units_per_layer, activation='relu'),
00095         tf.keras.layers.Dense(units=1),
00096     ])
00097
00098     return conv_model
00099
00100 def lstm(units_per_layer):
00101     """
00102     Creates a model of a Long Short-Term Memory Neural Network as the input layer, one dense with only 1
00103     neuron as the
00104     output layer. The activation functions of the LSTM layer are the regular ones for each of it's gates.
00105     @param units_per_layer: Number of inputs of the network.
00106     @return lstm_model: The model of the LSTM created for later usage as the predictor.
00107     """
00108     Env.print_text(f"Creating a Long Short-term Memory (LSTM) Neural Network using {units_per_layer} units
00109     per layer."
00110                    f"\nThis network only outputs the final timestamp, giving the model time to warm up its
00111                    internal "
00112                    f"state before making a single prediction.")
00113
00114     lstm_model = tf.keras.models.Sequential([
00115         # Shape [batch, time, features] => [batch, time, lstm_units]
00116         tf.keras.layers.LSTM(units_per_layer, return_sequences=False),
00117         # Shape => [batch, time, features]
00118         tf.keras.layers.Dense(units=1)
00119     ])
00120
00121     return lstm_model
00122
00123
00124 def compile_and_fit(model, train, val, epochs=20, patience=2):
00125     """
00126     Fits the provided training data in the specified model and train's it. The validation data is used to
00127     compare the
00128     performance of the model against unknown data. MSE is used as the cost function to train the model and
00129     MAE as the
00130     performance evaluation metric.
00131     @param model: Model to train. It can be a MLP, CNN or LSTM, among others.
00132     @param train: The dataset for training the model. Must have the shape: (batch, time, features)
00133     @param val: The dataset for validating the model. Must have the shape: (batch, time, features)
00134     @param epochs: Number of iteration over the entire set of data (all the batches).
00135     @param patience: Number of epochs to wait for improvement in the metrics. If there is no notorious
00136     improvement in
00137     the performance of the validation set after the 'patience' epochs, the training will stop at this
00138     point.
00139     @return history: The results of the training.
00140     """
00141     Env.print_text(f"Compiling the input model {model.name} with {epochs} epochs. The "
00142                    f"loss function is the MSE and metric to evaluate improvement is the MAE.")
00143
00144     early_stopping = tf.keras.callbacks.EarlyStopping(monitor='val_loss',
00145                                                        patience=patience,
00146                                                        mode='min')

```

```

00137
00138     model.compile(loss=tf.losses.MeanSquaredError(), # tf.losses.mean_absolute_percentage_error()
00139                  metrics=[tf.metrics.MeanAbsoluteError()])
00140
00141     history = model.fit(train, epochs=epochs,
00142                       validation_data=val,
00143                       callbacks=[early_stopping])
00144
00145     return history

```

5.13 /home/pabloav/Documents/Tesis_Lic/LFP-Classification/Yomchi/preprocessing.py File Reference

Namespaces

- [preprocessing](#)

Functions

- def [preprocessing.load_lfp_data](#) (file=LFP[771], channels=EC014_41_NUMBER_OF_CHANNELS)
Loads the LFP signals from a .eeg file (LFP Data only $f < 625\text{Hz}$) or a .dat file (LFP Data + Spikes).
- def [preprocessing.load_angles_data](#) (file=ANGLES[771], degrees=True)
Loads the animal position data from a .whl file which contain 2 (x, y) pairs, one for each LED.
- def [preprocessing.downsample_lfps](#) (lfp_data, orig_rate, new_rate)
Downsample the LFP signal data after applying an anti-aliasing filter.
- def [preprocessing.angles_expansion](#) (angles_data, orig_rate, new_rate)
Fill angular data with 'NaN' values to match an expected sampling rate.
- def [preprocessing.shortest_angle_interpolation](#) (start, end, amount)
This interpolation method considers the 'start' and 'end' as angles in a circumference where the objective is to find the smallest arch between two angles.
- def [preprocessing.vectorized_sai](#) (angles_data)
Replace 'NaN' values between valid values (interpolation) in angular data with an interpolated value using the shortest angle interpolation.
- def [preprocessing.interpolate_angles](#) (angles_data, method="Shortest")
Replace 'NaN' values in angular data with an interpolated value using a given method.
- def [preprocessing.add_labels](#) (lfps, angles, round_labels, start=0, offset=30)
Add an additional column to the LFP signals matrix with the angular data used as the labels.
- def [preprocessing.clean_invalid_positional](#) (labeled_dataset, is_padded=True)
Clean the data rows which have '-1' values as labels (angles) from the the data and their LFPs associated in each channel.
- def [preprocessing.ndarray_to_dataframe](#) (dataset, rate)
Converts an n-D Numpy array to a Pandas Dataframe.
- def [preprocessing.channels_to_windows](#) (series, channel, window_size, batch_size, shuffle_buffer=None, offset=1)
Receives a numpy array containing the time series of LFP signals of n channels and returns the same data, separated in windows.
- def [preprocessing.average_angles](#) (angles, window_size)
Receives a numpy array containing the time series of the angles and returns the an set of windows with the average of the 'window_size' angles in each window.

Variables

- `preprocessing.PATH_TO_DATASETS` = `os.path.join(Env.CURRENT_FOLDER, "../Datasets/")`
- dictionary `preprocessing.LFP`
- dictionary `preprocessing.ANGLES`
- int `preprocessing.RAW_DATAMAX_SAMPLING_RATE` = 20000
- int `preprocessing.RAW_NEURALYNX_SAMPLING_RATE` = 32552
- int `preprocessing.LFP_DATAMAX_SAMPLING_RATE` = 1250
- int `preprocessing.LFP_NEURALYNX_SAMPLING_RATE` = 1252
- float `preprocessing.POSITION_DATA_SAMPLING_RATE` = 39.06
- int `preprocessing.EC014_41_NUMBER_OF_CHANNELS` = 99

5.13.1 Detailed Description

Author

Pablo Avila [B30724] `jose.avilalopez@ucr.ac.cr`

Copyright

MIT License

Date

July, 2021

This module contains a set of functions to import clean, parse and reshape input data.

Definition in file `preprocessing.py`.

5.14 preprocessing.py

```
00001 # #####
00002 #           University of Costa Rica
00003 #           Electrical Engineering Department
00004 #           Grade Thesis
00005 # #####
00006
00007 """
00008 @file preprocessing.py
00009 @author Pablo Avila [B30724] jose.avilalopez@ucr.ac.cr
00010 @copyright MIT License
00011 @date July, 2021
00012 @details This module contains a set of functions to import clean, parse and reshape input data.
00013 """
00014
00015 import os
00016 import pandas as pd
00017 import pickle
00018 import scipy.signal
00019 import numpy as np
00020 import tensorflow as tf
00021 import Yomchi.Environment as Env
00022
00023 # Dataset Paths
00024 PATH_TO_DATASETS = os.path.join(Env.CURRENT_FOLDER, "../Datasets/")
00025 LFP = {765: PATH_TO_DATASETS + "ec014.765.eeg",
00026        771: PATH_TO_DATASETS + "ec014.771.eeg"}
00027 ANGLES = {765: PATH_TO_DATASETS + "ec014.765.whl",
00028           771: PATH_TO_DATASETS + "ec014.771.whl"}
00029
00030 # Sampling Rates
00031 RAW_DATAMAX_SAMPLING_RATE = 20000
00032 RAW_NEURALYNX_SAMPLING_RATE = 32552
00033 LFP_DATAMAX_SAMPLING_RATE = 1250
00034 LFP_NEURALYNX_SAMPLING_RATE = 1252
00035 POSITION_DATA_SAMPLING_RATE = 39.06
00036 EC014_41_NUMBER_OF_CHANNELS = 99
```

```

00037
00038
00039 def load_lfp_data(file=LFP[771], channels=EC014_41_NUMBER_OF_CHANNELS):
00040     """
00041     Loads the LFP signals from a .eeg file (LFP Data only f < 625Hz) or a .dat file (LFP Data + Spikes).
00042     @param file: Path to the file containing the animal LFP data.
00043     @param channels: Number of recorded channels in LFP signals file.
00044     @return lfp: Array (n x channels) with the data. With the columns being the channels and the rows the
00045     a different time step.
00046     """
00047     Env.print_text("Loading LFP Data of " + str(channels) + " channels from: " + os.path.basename(file) + "
00048 .")
00049     signals = open(file, "rb")
00050     signalsArray = np.fromfile(file=signals, dtype=np.int16)
00051     lfp = np.reshape(signalsArray, (-1, channels))
00052     signals.close()
00053
00054     Env.print_text("LFP Data shape: " + str(np.shape(lfp)))
00055
00056     return lfp
00057
00058
00059 def load_angles_data(file=ANGLES[771], degrees=True):
00060     """
00061     Loads the animal position data from a .whl file which contain 2 (x, y) pairs, one for each LED. If any
00062     position
00063     value equals '-1' then it's replaced with 'NaN' instead.
00064     @param file: Path to the file containing the animal LED's position information
00065     @param degrees: If this flag is set, then the angles are returned in degrees from [0, 360[, or radians
00066     otherwise.
00067     @return angles: Array with the angles in radians extracted from the positions. The angles are given as
00068     float16 values calculated as arctan(y2 - y1 / x2 - x1). Unless the denominator is 0, in that case '0' is
00069     returned
00070     for that element.
00071     """
00072     Env.print_text("Loading the animal position data from: " + os.path.basename(file) + ".")
00073
00074     positions_file = open(file, "rb")
00075     positions = np.genfromtxt(fname=positions_file, dtype=np.float16, delimiter='\t')
00076     positions_file.close()
00077
00078     positions[positions == -1] = np.NaN
00079     #TODO: of x_2 - x_1 = 0, angles should be NaN.
00080
00081     angles = np.arctan2((np.subtract(positions[:, 3], positions[:, 1])),
00082                        (np.subtract(positions[:, 2], positions[:, 0])))
00083
00084     if degrees:
00085         angles = np.degrees(angles)
00086         angles += 180
00087
00088     angles[np.isnan(angles)] = -1
00089     invalid = np.count_nonzero(angles == -1)
00090
00091     Env.print_text("Head angle data shape: " + str(np.shape(angles)))
00092     Env.print_text(f"Head angle invalid data: {invalid:d} ({invalid/len(angles) * 100:.2f}%)")
00093
00094     return angles
00095
00096
00097 def downsample_lfps(lfp_data, orig_rate, new_rate):
00098     """
00099     Downsample the LFP signal data after applying an anti-aliasing filter.
00100     An order 8 Chebyshev type I filter is used. Usually the LFP signals are acquired at a higher sampling
00101     rate
00102     than the position data.
00103     @note: This method assumes that the reason of frequencies is 32 to compute the decimation.
00104     @param lfp_data: Matrix [n x numChannels] with the LFP signals
00105     @param orig_rate: Sampling rate originally used to acquire the data.
00106     @param new_rate: New sampling rate of the data.
00107     @return resampled_data: Original data downsampled to the new rate.
00108     """
00109     Env.print_text("Downsampling LFP data to match the new sampling rate: " + str(new_rate)
00110                    + "Hz. Original was: " + str(orig_rate) + "Hz.")
00111
00112     resampled_data = []
00113     for channel_i in np.transpose(lfp_data):
00114         channel_i = scipy.signal.decimate(channel_i, 8)
00115         channel_i = scipy.signal.decimate(channel_i, 4)
00116         resampled_data.append(channel_i)
00117
00118     resampled_data = np.transpose(resampled_data)
00119     Env.print_text("LFP data decimated shape: " + str(np.shape(resampled_data)))
00120
00121     return resampled_data

```

```

00119
00120 def angles_expansion(angles_data, orig_rate, new_rate):
00121     """
00122     Fill angular data with 'NaN' values to match an expected sampling rate. Usually the position data is
    acquired at a
00123     lower sampling rate than the LFP signals.
00124     @details Assuming that the acquisition of data started and stopped at the same time, then no data has
    to be added
00125     after the last sample.
00126     @param angles_data: Array with the angles data extracted from the animal positions.
00127     @param orig_rate: Sampling rate originally used to acquire the data.
00128     @param new_rate: New sampling rate of the data. The gaps are filled with 'NaN'.
00129     @return upsampled_data: Original data filled with 'NaN' to match the new sampling rate.
00130     """
00131     Env.print_text("Expanding angles data with 'NaN' values to match the new sampling rate: " + str(
new_rate) + "Hz. "
00132                     + "Original was: " + str(orig_rate) + "Hz.")
00133
00134     expansion_factor = round(new_rate/orig_rate)
00135     padding = np.full((len(angles_data), expansion_factor - 1), np.NaN)
00136     upsampled_data = np.concatenate((np.transpose(np.array([angles_data])), padding), axis=1)
00137     upsampled_data = upsampled_data[: -1, :]
00138     upsampled_data = upsampled_data.flatten()
00139
00140     Env.print_text("Angles data upsampled shape: " + str(np.shape(upsampled_data)))
00141
00142     return upsampled_data
00143
00144
00145 def shortest_angle_interpolation(start, end, amount):
00146     """
00147     This interpolation method considers the 'start' and 'end' as angles in a circumference where the
    objective is to
00148     find the smallest arch between two angles.
00149     param start: Start angle with values in the range of [0 to 360[
00150     param end: Final angle with values in the range of [0 to 360[
00151     param amount: Value between [0, 1] which determines how close the interpolated angle will be placed
    from the Start
00152     angle (0) or from the Final angle (1), being 0.5 the middle.
00153     return interpolated_angle: Interpolated angle between 'start' and 'end'.
00154     """
00155     shortest_angle = ((end-start) + 180) % 360 - 180
00156     interpolated_angle = (start + shortest_angle * amount) % 360
00157     return interpolated_angle
00158
00159
00160 def vectorized_sai(angles_data):
00161     """
00162     Replace 'NaN' values between valid values (interpolation) in angular data with an interpolated value
    using
00163     the shortest angle interpolation.
00164     param angles_data: Array with the angles data extracted from the animal positions.
00165     return interpolated_angles: Array with the angles data interpolated using the Shortest Angle
    Interpolation
00166     """
00167     start_angle = np.nan
00168     no_nans = 0
00169     first_nan_index = 0
00170     interpolated_angles = angles_data
00171
00172     for i in range(1, len(interpolated_angles)):
00173         # If a valid value followed by NaN: this is the first NaN, start counting
00174         if np.isnan(interpolated_angles[i]) and not np.isnan(interpolated_angles[i-1]):
00175             start_angle = interpolated_angles[i-1]
00176             no_nans = 1
00177             first_nan_index = i
00178
00179         # If a NaN followed by another NaN: Increment counter 1+.
00180         elif np.isnan(interpolated_angles[i]) and np.isnan(interpolated_angles[i - 1]):
00181             no_nans += 1
00182
00183         # If a NaN followed by a valid value: This is the last NaN, interpolate.
00184         elif not np.isnan(interpolated_angles[i]) and np.isnan(interpolated_angles[i - 1]):
00185             if no_nans > 0 and not np.isnan(start_angle):
00186                 amount = 0
00187                 end_angle = interpolated_angles[i]
00188                 for j in range(first_nan_index, first_nan_index + no_nans):
00189                     amount += 1/(no_nans + 1)
00190                     interpolated_angles[j] = shortest_angle_interpolation(
start_angle, end_angle, amount)
00191                 start_angle = np.nan
00192
00193     return interpolated_angles
00194
00195
00196 def interpolate_angles(angles_data, method="Shortest"):
00197     """

```



```

00198     Replace 'NaN' values in angular data with an interpolated value using a given method.
00199     @param angles_data: Array with the angles data extracted from the animal positions.
00200     @param method: Interpolation method to fill the gaps in the data. The optional methods available are
the supported
00201     by pandas.DataFrame.interpolate function, which are: 'linear', 'quadratic', 'cubic', 'polynomial',
among others.
00202     @return interpolated_angles: Array with the angles data interpolated using the given method.
00203     """
00204     Env.print_text("Interpolate angles data using " + method + " method.")
00205
00206     if method == "Shortest":
00207         interpolated_angles = vectorized_sai(angles_data)
00208
00209     else:
00210         angles_series = pd.Series(angles_data)
00211         interpolated_angles = angles_series.interpolate(method)
00212         interpolated_angles = interpolated_angles.to_numpy()
00213
00214     return interpolated_angles
00215
00216
00217 def add_labels(lfps, angles, round_labels, start=0, offset=30):
00218     """
00219     Add an additional column to the LFP signals matrix with the angular data used as the labels.
00220     @param lfps: Matrix [n x numChannels] with the LFP signals used as the preliminary features of the
data.
00221     @param angles: Array with the angles data extracted from the positions used as the labels of the data.
00222     @param round_labels: Boolean, if true the labels are rounded to angles multiples of 'offset' starting
from 'start'
00223     @param start: Angle in [0°, 360°[ used as first label.
00224     @param offset: Offset in [1°, 360°[ between labels starting from 'start' angle.
00225     @return labeled_data: Matrix with the labeled data [n x lfps[numChannels], angles].
00226     """
00227
00228     len_lfps = len(lfps)
00229     len_angles = len(angles)
00230
00231     if len_lfps > len_angles:
00232         lfps = lfps[0: len_angles]
00233     elif len_lfps < len_angles:
00234         angles = angles[0: len_lfps]
00235
00236     Env.print_text("Adding labels to the data by concatenating the [" + str(len(lfps)) + " x " + str(len(
lfps[0])) +
00237                    "]" LFP data matrix with the [" + str(len(angles)) + "]" Angles vector.")
00238     Env.print_text("Rounding Labels = " + str(round_labels))
00239
00240     if round_labels:
00241         if (0 <= start < 360) and (1 <= offset < 360):
00242             labels = np.arange(start, 360, offset)
00243
00244             for i in range(0, len(angles)):
00245                 if not np.isnan(angles[i]):
00246                     minval = np.inf
00247                     label = start
00248                     for tag in labels:
00249                         diff = abs(angles[i] - tag)
00250                         if diff < minval:
00251                             minval = diff
00252                             label = tag
00253                     angles[i] = label
00254
00255     labeled_data = np.concatenate((lfps, angles), axis=1)
00256
00257     return labeled_data
00258
00259
00260 def clean_invalid_positional(labeled_dataset, is_padded=True):
00261     """
00262     Clean the data rows which have '-1' values as labels (angles) from the the data and their LFPs
associated in each
00263     channel. Plus the following 31 rows with NaN as angle value in case of padded data.
00264     @details The positional data taken from the LEDs placed in the rat have discontinuities where one or
both LEDs
00265     are lost, making them invalid.
00266     Hence '-1' values are used instead to denote invalid position data and are meant to be removed from the
data since
00267     they are not representative labels.
00268     @param labeled_dataset: Matrix [n x (numChannels +1)] with the LFP signals used as the preliminary
features of the
00269     data and the angles data extracted from the positions used as the labels of the data.
00270     @param is_padded: If True, manage the input labeled dataset as a padded array of angles, or a
downsampled LFP set
00271     otherwise.
00272     @return clean_dataset: Input data without invalid positional values.
00273     """
00274     if is_padded:

```

```

00275     # Get the indexes of all invalid values (-1) plus the 31 following padded values (NaN).
00276     # and save that range as an element of 'invalid_indexes' array.
00277     invalid_indexes = [np.arange(i, i + 32).tolist() for i, v in enumerate(labeled_dataset) if v[-1] ==
-1]
00278     Env.print_text(f"Amount of invalid indexes: {len(invalid_indexes)}")
00279
00280     # Merge all sublists as a single consecutive array of invalid indexes.
00281     invalid_indexes = [item for sublist in invalid_indexes for item in sublist]
00282     Env.print_text(f"Amount of invalid indexes + associated expanded indexes: {len(invalid_indexes)}")
00283     Env.print_text(f"Number of valid samples: {len(labeled_dataset) - len(invalid_indexes)} (including
the 31 NaN "
00284                     f"values padded for the last real positional data.)")
00285     else:
00286     # Get the indexes of all invalid values (-1) and save that range as an element of 'invalid_indexes'
array.
00287     invalid_indexes = [i for i, v in enumerate(labeled_dataset) if v[-1] == -1]
00288     Env.print_text(f"Amount of invalid indexes: {len(invalid_indexes)}")
00289     Env.print_text(f"Number of valid samples: {len(labeled_dataset) - len(invalid_indexes)}")
00290
00291     # Get the indexes where the discontinuities start and end.
00292     discontinuities_starts = [invalid_indexes[0]]
00293     discontinuities_ends = []
00294     for i in range(1, len(invalid_indexes)):
00295         if invalid_indexes[i] - 1 != invalid_indexes[i-1]:
00296             discontinuities_starts.append(invalid_indexes[i])
00297             discontinuities_ends.append(invalid_indexes[i-1])
00298     discontinuities_ends.append(invalid_indexes[-1])
00299
00300     # Stores sub-arrays of valid indexes:
00301     clean_datasets = []
00302
00303     # From the first item to the start of the first discontinuity
00304     if discontinuities_starts[0] != 0:
00305         clean_datasets.append(labeled_dataset[0:discontinuities_starts[0], :])
00306
00307     # From the end+1 of the ith discontinuity to the start (not included) of the ith + 1 discontinuity
00308     for i in range(len(discontinuities_starts)-1):
00309         clean_datasets.append(labeled_dataset[discontinuities_ends[i]+1:discontinuities_starts[i+1], :])
00310
00311     # From the end of the last discontinuity to the last item.
00312     if discontinuities_ends[-1] != len(labeled_dataset)-1:
00313         clean_datasets.append(labeled_dataset[discontinuities_ends[-1] + 1:-1, :])
00314
00315     # Delete subsets with only 1 valid sample (i.e its length is <=32)
00316     # From the resulting subsets, delete the last 31 rows with NaN angles
00317     clean_datasets = [v[:-31, :] for v in clean_datasets if len(v) > 32]
00318
00319     Env.print_text(f"Total number of valid subsets: {len(clean_datasets)}")
00320     for s in range(0, len(clean_datasets)):
00321         Env.print_text(f"Total number of samples and channels + angles in subsets {s}:
{clean_datasets[s].shape}")
00322
00323     return clean_datasets
00324
00325
00326 def ndarray_to_dataframe(dataset, rate):
00327     """
00328     Converts an n-D Numpy array to a Pandas Dataframe
00329     @param dataset: Matrix [n x (numChannels +1)] with the LFP signals used as the preliminary features
00330     of the data and the angles data extracted from the positions used as the labels of the data.
00331     @param rate:
00332     @return dataframe: Pandas data frame with Channels 0-99 and Angles as columns names, and the timestamp
as indexes
00333     calculated as 1/rate * 1e6 to get the time step of the acquisition in microseconds.
00334     """
00335
00336     columns = []
00337     for i in range(0, 99):
00338         columns.append(f"Channel {i}")
00339     columns.append("Angles")
00340
00341     time_step = round((1/rate) * 1e6) # 1/f [us]: 1250Hz => 800us, 39.06Hz => 2560.164us
00342     indeces = np.arange(0, len(dataset) * time_step, time_step)
00343     dataframe = pd.DataFrame(data=dataset, columns=columns, index=indeces)
00344
00345     return dataframe
00346
00347
00348 def channels_to_windows(series, channel, window_size, batch_size, shuffle_buffer=None,
offset=1):
00349     """
00350     Receives a numpy array containing the time series of LFP signals of n channels and returns the same
data, separated
in windows.
00351     @param series: Numpy Array with the LFP data of the n channels.
00352     @param channel: Channel to use.
00353     @param window_size: Size of the windows in which the data are being split.

```

```

00355     @param batch_size: Number of pairs data-labels to group as a batch
00356     @param shuffle_buffer: Number of windows to shuffle at the same time.
00357     @param offset: Number of samples to shift between windows.
00358     @return windowed_ds: LFP data of the selected channel separated in windows.
00359     """
00360
00361     labels = np.expand_dims(series[window_size::, -1], 1)
00362     data = series[:, channel]
00363
00364     # Get the average angle for each window.
00365     #labels_ds = average_angles(labels, window_size)
00366
00367     labels_ds = tf.data.Dataset.from_tensor_slices(np.expand_dims(labels, 1))
00368
00369     # Creates a dataset from the input
00370     windowed_data = tf.data.Dataset.from_tensor_slices(np.expand_dims(data, 1))
00371
00372     # Split the data set in windows shifting each window by 1 and forcing them to the same size
00373     (window_size + 1)
00374     windowed_data = windowed_data.window(window_size, shift=offset, drop_remainder=True)
00375
00376     # Make each window a numpy array row.
00377     windowed_data = windowed_data.flat_map(lambda window: window.batch(window_size))
00378
00379     # Add labels to the data
00380     windowed_ds = tf.data.Dataset.zip((windowed_data, labels_ds))
00381
00382     if Env.debug:
00383         for x, y in windowed_ds.take(1):
00384             print("x = ", x.numpy())
00385             print("y = ", y.numpy())
00386
00387     if shuffle_buffer != None:
00388         # Shuffle the data in groups of shuffle_buffer to accelerate. Instead of shuffle it all at once.
00389         windowed_ds = windowed_ds.shuffle(shuffle_buffer)
00390
00391     # Batch the data into sets of 'batch_size'.
00392     windowed_ds = windowed_ds.batch(batch_size).prefetch(1)
00393
00394     return windowed_ds
00395
00396 def average_angles(angles, window_size):
00397     """
00398     Receives a numpy array containing the time series of the angles and returns the an set of windows with
00399     the average
00400     of the 'window_size' angles in each window. Each window is 1 element shifted from the previous window.
00401     @param angles: Numpy Array with the Angles data to use as the labels.
00402     @param window_size: Size of the windows in which the data are being split.
00403     @return average_angles: Averaged Angles data separated in windows.
00404     """
00405
00406     # Get an array of labels from the series
00407     windowed_angles = tf.data.Dataset.from_tensor_slices(angles)
00408
00409     # Split the data set in windows shifting each window by 1 and forcing them to the same size
00410     (window_size + 1)
00411     windowed_angles = windowed_angles.window(window_size, shift=1, drop_remainder=True)
00412
00413     # Make each window a numpy array row.
00414     windowed_angles = windowed_angles.flat_map(lambda window: window.batch(window_size))
00415     averaged_angles = windowed_angles.map(lambda window: tf.math.reduce_mean(window))
00416
00417     if Env.debug:
00418         print("Angles:")
00419         for window in windowed_angles.take(1):
00420             print(window.numpy())
00421         for window in averaged_angles.take(1):
00422             print(window.numpy())
00423
00424     return averaged_angles

```

5.15 /home/pabloav/Documents/Tesis_Lic/LFP-Classification/Yomchi/visualization.py File Reference

Namespaces

- [visualization](#)

Functions

- def `visualization.store_figure` (fig_name, show=False)
Stores a figure.

Variables

- string `visualization.FIG_FORMAT` = 'png'

5.15.1 Detailed Description

Author

Pablo Avila [B30724] `jose.avilalopez@ucr.ac.cr`

Copyright

MIT License

Date

July, 2021

This module contains a set of functions to plot and in general terms, visualize data.

Definition in file `visualization.py`.

5.16 visualization.py

```
00001 # #####
00002 #           University of Costa Rica
00003 #           Electrical Engineering Department
00004 #           Grade Thesis
00005 # #####
00006
00007 """
00008 @file visualization.py
00009 @author Pablo Avila [B30724] jose.avilalopez@ucr.ac.cr
00010 @copyright MIT License
00011 @date July, 2021
00012 @details This module contains a set of functions to plot and in general terms, visualize data.
00013 """
00014
00015 import matplotlib.pyplot as plt
00016 import Yomchi.Environment as Env
00017
00018 # Matplotlib figures parameters
00019 FIG_FORMAT = 'png'
00020 plt.rcParams['figure.dpi'] = 200
00021 plt.rcParams['figure.figsize'] = [12, 8]
00022 plt.rcParams['savefig.format'] = FIG_FORMAT
00023 plt.rcParams['lines.markersize'] = 1
00024 plt.rcParams['font.family'] = 'serif'
00025 plt.rcParams['font.size'] = 18
00026 plt.rcParams['figure.autolayout'] = True
00027 plt.rcParams['axes.titlepad'] = 7
00028 plt.rcParams['axes.grid'] = False
00029
00030
00031
00032 def store_figure(fig_name, show=False):
00033     """
00034     Stores a figure.
00035     @param fig_name: Name of the figure to store.
00036     @param show: Displays the figure when ready. Warning: Stalls execution until closing it.
00037     @return None
00038     """
00039
```

```
00040     plt.figure(fig_name)
00041     plt.savefig(fname=Env.CAPTURES_FOLDER + fig_name + "." + FIG_FORMAT)
00042     if show:
00043         plt.show()
00044     plt.close(fig_name)
```


Index

/home/pabloav/Documents/Tesis_Lic/LFP-Classification/↔ Environment, 13
DataExploration/DataExploration.py, 27 downsample_lfps
/home/pabloav/Documents/Tesis_Lic/LFP-Classification/↔ preprocessing, 21
DatasetGenerator/DatasetGenerator.py, 32
/home/pabloav/Documents/Tesis_Lic/LFP-Classification/↔ EC014_41_NUMBER_OF_CHANNELS
Predictor/Predictor.py, 35, 36 preprocessing, 24
/home/pabloav/Documents/Tesis_Lic/LFP-Classification/↔ Environment, 9
README.md, 38 CAPTURES_FOLDER, 13
/home/pabloav/Documents/Tesis_Lic/LFP-Classification/↔ CURRENT_FOLDER, 13
Yomchi/Environment.py, 39, 41 caller_file, 13
/home/pabloav/Documents/Tesis_Lic/LFP-Classification/↔ debug, 13
Yomchi/models.py, 44, 45 finish_test, 10
/home/pabloav/Documents/Tesis_Lic/LFP-Classification/↔ init_environment, 10
Yomchi/preprocessing.py, 47, 48 LINE_LENGTH, 13
/home/pabloav/Documents/Tesis_Lic/LFP-Classification/↔ LOGS_FOLDER, 14
Yomchi/visualization.py, 53, 54 log_versions, 11
print_box, 11
print_header, 11
print_parameters, 12
print_text, 12
RESULTS_FOLDER, 14
START_TIME, 14
step, 12
step_number, 14
VERSIONS_LOG_PATH, 14
versions_log, 14
YOMCHI_LOG_PATH, 15
yomchi_log, 14

ANGLES
preprocessing, 24
add_labels
preprocessing, 19
angles_expansion
preprocessing, 19
average_angles
preprocessing, 20

CAPTURES_FOLDER
Environment, 13
CURRENT_FOLDER
Environment, 13
caller_file
Environment, 13
channels_to_windows
preprocessing, 20
clean_invalid_positional
preprocessing, 21
cnn
models, 15
compile_and_fit
models, 15

data_set_generator
DatasetGenerator, 8
DataExploration, 7
dataExploration, 7
dataExploration
DataExploration, 7
DatasetGenerator, 8
data_set_generator, 8
debug

FIG_FORMAT
visualization, 26
finish_test
Environment, 10
init_environment
Environment, 10
interpolate_angles
preprocessing, 22
LFP_DATAMAX_SAMPLING_RATE
preprocessing, 25
LFP_NEURALYNX_SAMPLING_RATE
preprocessing, 25
LFP
preprocessing, 24
LINE_LENGTH
Environment, 13
LOGS_FOLDER
Environment, 14
load_angles_data

- preprocessing, [22](#)
- load_lfp_data
 - preprocessing, [23](#)
- log_versions
 - Environment, [11](#)
- lstm
 - models, [16](#)
- mlp
 - models, [16](#)
- models, [15](#)
 - cnn, [15](#)
 - compile_and_fit, [15](#)
 - lstm, [16](#)
 - mlp, [16](#)
- ndarray_to_dataframe
 - preprocessing, [23](#)
- PATH_TO_DATASETS
 - preprocessing, [25](#)
- POSITION_DATA_SAMPLING_RATE
 - preprocessing, [25](#)
- Predictor, [17](#)
 - predictor, [17](#)
- predictor
 - Predictor, [17](#)
- preprocessing, [18](#)
 - ANGLES, [24](#)
 - add_labels, [19](#)
 - angles_expansion, [19](#)
 - average_angles, [20](#)
 - channels_to_windows, [20](#)
 - clean_invalid_positional, [21](#)
 - downsample_lfps, [21](#)
 - EC014_41_NUMBER_OF_CHANNELS, [24](#)
 - interpolate_angles, [22](#)
 - LFP_DATAMAX_SAMPLING_RATE, [25](#)
 - LFP_NEURALYNX_SAMPLING_RATE, [25](#)
 - LFP, [24](#)
 - load_angles_data, [22](#)
 - load_lfp_data, [23](#)
 - ndarray_to_dataframe, [23](#)
 - PATH_TO_DATASETS, [25](#)
 - POSITION_DATA_SAMPLING_RATE, [25](#)
 - RAW_DATAMAX_SAMPLING_RATE, [25](#)
 - RAW_NEURALYNX_SAMPLING_RATE, [25](#)
 - shortest_angle_interpolation, [23](#)
 - vectorized_sai, [24](#)
- print_box
 - Environment, [11](#)
- print_header
 - Environment, [11](#)
- print_parameters
 - Environment, [12](#)
- print_text
 - Environment, [12](#)
- RAW_DATAMAX_SAMPLING_RATE
 - preprocessing, [25](#)
- RAW_NEURALYNX_SAMPLING_RATE
 - preprocessing, [25](#)
- RESULTS_FOLDER
 - Environment, [14](#)
- START_TIME
 - Environment, [14](#)
- shortest_angle_interpolation
 - preprocessing, [23](#)
- step
 - Environment, [12](#)
- step_number
 - Environment, [14](#)
- store_figure
 - visualization, [26](#)
- VERSIONS_LOG_PATH
 - Environment, [14](#)
- vectorized_sai
 - preprocessing, [24](#)
- versions_log
 - Environment, [14](#)
- visualization, [25](#)
 - FIG_FORMAT, [26](#)
 - store_figure, [26](#)
- YOMCHI_LOG_PATH
 - Environment, [15](#)
- yomchi_log
 - Environment, [14](#)