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

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):

Chapter 3

File Index

3.1 File List

Here is a list of all files with brief descriptions:

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

rename_results_folder	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

print_the_header	If True, print the header
enable_debug	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

text	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

params_dictionary	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

text	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

description	Description of the step.
new_step_number	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

inputs	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.
units_per_layer	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

model	Model to train. It can be a MLP, CNN or LSTM, among others.
train	The dataset for training the model. Must have the shape: (batch, time, features)
val	The dataset for validating the model. Must have the shape: (batch, time, features)
epochs	Number of iteration over the entire set of data (all the batches).
patience	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

units_per_layer	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

layers	Number of hidden layers (besides the input and outputs ones).
units_per_layer	Number of neurons per layer. Applies to all layers except for the last one which has only 1: the predicted angle.
dropout	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 $<$ 625Hz) 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

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

angles_data	Array with the angles data extracted from the animal positions.
orig_rate	Sampling rate originally used to acquire the data.
new_rate	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

angles	Numpy Array with the Angles data to use as the labels.
window_size	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

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

Parameters

shuffle_buffer	Number of windows to shuffle at the same time.
offset	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

labeled_dataset	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.
is_padded	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

lfp_data	Matrix [n x numChannels] with the LFP signals
orig_rate	Sampling rate originally used to acquire the data.
new_rate	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

angles_data	Array with the angles data extracted from the animal positions.
method	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

file	Path to the file containing the animal LED's position information
degrees	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(y2 - y1 /x2 - x1). 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 < 625Hz) or a .dat file (LFP Data + Spikes).

Parameters

file	Path to the file containing the animal LFP data.
channels	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

dataset	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.
rate	

Returns

dataframe: Pandas data frame with Channels 0-99 and Angles as columns names, and the timestamp as indexes calculated as 1/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

fig_name	Name of the figure to store.
show	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/Data↩ Exploration.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

```
# ########################################################################
#                   University of Costa Rica
#               Electrical Engineering Department
#                         Grade Thesis
# ########################################################################

"""
@file DataExploration.py
@author Pablo Avila [B30724] jose.avilalopez@ucr.ac.cr
@copyright MIT License
```

```
@date July, 2021
@details Script for explore the input data by manipulating, printing and plotting the information in different ways.
"""

import Yomchi.Environment as Env
import Yomchi.preprocessing as data
import Yomchi.visualization as ui

import matplotlib.pyplot as plt
import numpy as np
import seaborn as sns
import pandas as pd
import tensorflow as tf

def dataExploration():
    """
    @details
    <h1> Dataset Generator:</h1>
    <h2> Setup </h2>
    <ul>
        <li> Initialize Environment.
        <li> Initialize Tensorflow session.
        <li> Set seed for Numpy and Tensorflow
        <li> Specify run configuration parameters.
        <li> Specify dataset generation parameters.
    </ul>
    <h2> Procedure </h2>
    <ul>
    <li> <b>Step 1:</b>
        <ul>
        <li> Import LFP data.
        <li> Plot channels 0 and 97.
        </ul>
    <li> <b>Step 2:</b>
        <ul>
        <li> Import angles data.
        <li> Plot angles data.
        </ul>
    <li> <b>Step 3:</b>
        <ul>
        <li> If the chosen synchronization method is to downsample the LFP data:
            <ul>
                <li> Downsample LFP data to match Angles sampling rate.
                <li> Plot channels 0 and 97 downsampled.
            </ul>
        <li> Else, the chosen synchronization method is to upsample the Angles data:
            <ul>
                <li> Fill angles gaps to match the LFP sampling rate.
                <li> Plot angles data in the new sampling rate.
            </ul>
        </ul>
    <li> <b>Step 4:</b>
        <ul>
            <li> Interpolate angles data using a 'interpolation' approach.
            <li> Plot angles data after interpolation.
        </ul>
    <li> <b>Step 5:</b>
        <ul>
            <li> Label data by concatenating LFPs and interpolated Angles in a single 2D-array.
            <li> Plot LFP and angles data.
            <li> Print an angles data window where the first LED is unsynchronized and then the second LED is lost too.
        </ul>
    <li> <b>Step 6:</b>
        <ul>
            <li> Clean the labeled dataset from invalid values.
            <li> Print Bar plot for labeled angles.
        </ul>
    <li> <b>Step 7:</b>
        <ul>
            <li> Count number of NaNs at the beginning and at the end of the data.
            <li> Print some NaN counts in different dataset stages.
        </ul>
    <li> <b>Step 8:</b> Plotting clean LFP data from channels 0 and interpolated angles data.
    <li> <b>Step 9:</b> Plotting Boxplot of all channels in two figures, from 0-49 and from 50-98.
    <li> <b>Step 10:</b>
        <ul>
            <li> Plotting Boxplot of interpolated angles.
            <li> Plotting Boxplot of interpolated angles vs channel 0
        </ul>
    <li> <b>Finish Test and Exit:</b> Save logs, captures and results.
    </ul>
    """

    # ------- SETUP -------
    Env.init_environment(True, enable_debug=True)
```

```python

    tf.keras.backend.clear_session()
    tf.random.set_seed(51)
    np.random.seed(51)

    # Run configuration parameters
    PLOT = True

    # Session and methods Parameters
    session = 771  # or 765

    interpolation = "Shortest"  # "linear" "quadratic" "cubic" "nearest" "Shortest"
    sync_method = "Upsample Angles"  # "Upsample Angles" "Downsample LFPs"

    # Data Properties
    num_channels = data.EC014_41_NUMBER_OF_CHANNELS
    rate_used = data.POSITION_DATA_SAMPLING_RATE
    if sync_method == "Upsample Angles":
        rate_used = data.LFP_DATAMAX_SAMPLING_RATE

    # Windowing properties
    window_size = 31
    batch_size = 31
    shuffle_buffer = 1000
    lfp_channel = 0
    round_angles = False
    base_angle = 0  # Unused if round_angles = False
    offset_between_angles = 30  # Unused if round_angles = False

    extra = {"Round angles to get discrete label": round_angles}
    if round_angles:
        extra.update({"Angle labels starting from": str(base_angle) + "°",
                      "until 360° in steps of": str(offset_between_angles) + "°"})

    parameters_dictionary = {"Recording Session Number": str(session),
                             "Interpolation Method": interpolation.title(),
                             "Synchronization Method": sync_method,
                             "Number of Recorded Channels": str(num_channels),
                             "Sampling Rate to Use": str(rate_used) + " Hz",
                             "Window Size": window_size,
                             "Batch size (# of windows)": batch_size,
                             "LFP Signal channel to use": lfp_channel,
                             "Shuffle buffer size": shuffle_buffer}

    parameters_dictionary.update(extra)
    Env.print_parameters(parameters_dictionary)

    # ------- STEP 1 -------
    Env.step(f"Importing LFP data from session: {session}")
    lfp_data = data.load_lfp_data(data.LFP[session])

    if PLOT:
        Env.print_text("Plotting LFP data from channels 0 and 97 at " + str(data.LFP_DATAMAX_SAMPLING_RATE)
 + "Hz")
        figname = str(session) + "_LFP_C0_and_C97_" + str(data.LFP_DATAMAX_SAMPLING_RATE) + "Hz"
        plt.figure(figname)
        plt.subplot(211)
        plt.plot(lfp_data[:, 0], "xr")
        plt.title("Señal LFP. Sesión: " + str(session) + " del Canal 0 a "
                  + str(data.LFP_DATAMAX_SAMPLING_RATE) + "Hz")

        plt.subplot(212)
        plt.plot(lfp_data[:, 97], "xb")
        plt.title("Señal LFP. Sesión: " + str(session) + " del Canal 97 a "
                  + str(data.LFP_DATAMAX_SAMPLING_RATE) + "Hz")
        ui.store_figure(figname, True)

    # ------- STEP 2 -------
    Env.step(f"Importing angles data from session: {session}")

    angles_data = data.load_angles_data(data.ANGLES[session])

    Env.print_text("Min angle: " + str(np.nanmin(angles_data)) + ", Max angle: " + str(np.nanmax(
angles_data)))

    if PLOT:
        Env.print_text("Plotting Angles data [°] at " + str(data.POSITION_DATA_SAMPLING_RATE) + "Hz")
        figname = str(session) + "_Angles_degrees_" + str(data.POSITION_DATA_SAMPLING_RATE) + "Hz"
        plt.figure(figname)
        plt.plot(angles_data[:], "xr")
        plt.title("Información de ángulos [°]. Sesión: " + str(session) + " a "
                  + str(data.POSITION_DATA_SAMPLING_RATE) + "Hz")
        ui.store_figure(figname, True)

    # ------- STEP 3 -------
    if sync_method == "Downsample LFPs":
        Env.step("Downsample LFP data to match Angles sampling rate.")
```

```
        lfp_data = data.downsample_lfps(lfp_data, data.LFP_DATAMAX_SAMPLING_RATE,
data.POSITION_DATA_SAMPLING_RATE)

        if PLOT:
            Env.print_text("Plotting LFP downsampled data to " + str(rate_used) + "Hz from channels 0 and
97.")
            figname = str(session) + "_LFP_downsampled_0_and_97_" + str(rate_used) + "Hz"
            plt.figure(figname)
            plt.subplot(211)
            plt.plot(lfp_data[:, 0], "xr")
            plt.title("Señal LFP del Canal 0 a " + str(rate_used) + "Hz. Sesión: " + str(session))

            plt.subplot(212)
            plt.plot(lfp_data[:, 97], "xb")
            plt.title("Señal LFP del Canal 97 a " + str(rate_used) + "Hz. Sesión: " + str(session))
            ui.store_figure(figname, True)

    elif sync_method == "Upsample Angles":
        Env.step("Upsample Angles data to reach a higher sampling rate")

        angles_data = data.angles_expansion(angles_data, data.POSITION_DATA_SAMPLING_RATE,
data.LFP_DATAMAX_SAMPLING_RATE)

        if PLOT:
            Env.print_text("Plotting Angles data expanded with 'NaN' [°]")
            figname = str(session) + "Expanded_Angles_degrees_" + str(rate_used) + "Hz"
            plt.figure(figname)
            plt.plot(angles_data[:], "xb")
            plt.title("Información de ángulos [°] expandida a " + str(rate_used) + "Hz. Sesión: " + str(
session))
            ui.store_figure(figname, True)

    # ------- STEP 4 -------
    Env.step("Interpolate angles data using a " + interpolation + " approach.", 4)

    angles_data_interpolated = data.interpolate_angles(angles_data, interpolation)

    if PLOT:
        Env.print_text("Plotting Angles data after " + interpolation + " interpolation [°]")
        figname = str(session) + "_Angles_" + interpolation + "_degrees_" + str(rate_used) + "Hz"
        plt.figure(figname)
        plt.plot(angles_data_interpolated[:], "xr")
        plt.title("Información de ángulos [°]. Sesión: " + str(session) + ".\n Interpolada con: " +
interpolation
                  + " a " + str(rate_used) + "Hz")
        ui.store_figure(figname, True)

    # ------- STEP 5 -------
    Env.step("Label data by concatenating LFPs and interpolated Angles in a single 2D-array.")

    # IF round_angles: Rounding the angles to be descrete labels, starting from 'base_angle'until 360 on
 steps of
    # 'offset_between_angles'. Else Not rounding the angles to be discrete labels
    labeled_data = data.add_labels(lfp_data, np.expand_dims(angles_data_interpolated, axis=1), round_angles
,
                                   base_angle, offset_between_angles)

    if PLOT:
        Env.print_text("Plotting LFP data from channels 0 and interpolated angles data at " + str(rate_used
) + "Hz. [°]")
        figname = str(session) + "_LFP_C0_and_angles_" + interpolation + "_" + str(rate_used) + "Hz"
        plt.figure(figname)
        plt.subplot(211)
        plt.plot(labeled_data[:-1, 0], "xr")
        plt.title("Señal LFP del Canal 0. Muestreada a " + str(rate_used) + "Hz. Sesión: " + str(session))

        plt.subplot(212)
        plt.plot(labeled_data[:, -1], "xb")
        plt.title("Información de ángulos [°]. Sesión: " + str(session) + ".\n Interpolada con: " +
interpolation
                  + " a " + str(rate_used) + "Hz")
        ui.store_figure(figname, True)

    if str(session) == "771" and sync_method == "Downsample LFPs":
        Env.print_text("\nAngles Data from: 15560 to 15600 where at 15566 the first LED is lost and at
 15583 both are lost")
        Env.print_text(' '.join([str(elem) for elem in angles_data[15560:15600]]))

        Env.print_text("\nAngles Data Interpolated from: 15560 to 15600 where at 15566 the first LED is
 lost and at 15583 "
                       "both are lost.")
        Env.print_text(' '.join([str(elem) for elem in labeled_data[15560:15600, -1]]))

    # ------- STEP 6 -------
    Env.step("Clean the labeled dataset from invalid values.")

```

```python
    clean_dataset = data.clean_invalid_positional(labeled_data, sync_method == "Upsample Angles")

    if round_angles:
        labels = np.arange(base_angle, 360, offset_between_angles)
        percentages = []
        for u in range(base_angle, 360, offset_between_angles):
            percentages.append(round(np.sum(clean_dataset[:, -1] == u)*100/len(clean_dataset[:, -1])))

        labels_percent = np.concatenate((np.expand_dims(labels, axis=1), np.expand_dims(percentages, axis=1)), axis=1)
        dataframe_labels = pd.DataFrame(data=labels_percent, columns=["Angulos", "Porcentaje"])

        if PLOT:
            Env.print_text("Plotting Barplot of Labels after " + interpolation + " interpolation")
            figname = str(session) + "_BarPLotAngles_" + interpolation + "_" + str(rate_used) + "Hz"
            plt.figure(figname)
            sns.barplot(x="Angulos", y="Porcentaje", data=dataframe_labels)
            plt.title("Gráfico de Barras de las etiquetas. Sesión: " + str(session) + ".\n Interpolada con: " +
                      interpolation + " a " + str(rate_used) + "Hz")
            ui.store_figure(figname, True)

    # ------- STEP 7 -------
    Env.step("Count and print number of NaNs in the Dataset.")

    nans_begin = 0
    nans_end = 0
    length = len(angles_data_interpolated)
    for i in range(length):
        if ~np.isnan(angles_data_interpolated[i]):
            nans_begin = i
            break
    for i in range(length-1, 0, -1):
        if ~np.isnan(angles_data_interpolated[i]):
            nans_end = i
            break
    nans_end = length - (nans_end + 1)

    Env.print_text("Number of NaNs in Angles Data without interpolation: " + str(np.count_nonzero(np.isnan(angles_data))))
    Env.print_text("Number of NaNs in Labeled Dataset with interpolated Angles Data: "
                   + str(np.count_nonzero(np.isnan(angles_data_interpolated))))
    Env.print_text("Number of NaNs at the beginning of the interpolated Angles Data: " + str(nans_begin))
    Env.print_text("Number of NaNs at the end of the interpolated Angles Data: " + str(nans_end))
    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]))))

    if PLOT:
        # ------- STEP 8 -------
        Env.step("Plotting clean LFP data from channels 0 and interpolated angles data at " + str(rate_used) + "Hz. [°]")

        figname = str(session) + "_LFP_C0_clean_and_angles_" + interpolation + "_" + str(rate_used) + "Hz"
        plt.figure(figname)
        plt.subplot(211)
        plt.plot(clean_dataset[0][:, 0], "xr")
        plt.title("Señal LFP del Canal 0 limpia. Muestreada a " + str(rate_used) + "Hz. Sesión: " + str(session))
        plt.subplot(212)
        plt.plot(clean_dataset[0][:, -1], "xb")
        plt.title("Información de ángulos [°] limpia. Sesión: " + str(session) + ".\n Interpolada con: "
                  + interpolation + " a " + str(rate_used) + "Hz")
        ui.store_figure(figname, True)

        # ------- STEP 9 -------
        Env.step("Plotting Boxplot of all channels in two figures, from 0-49 and from 50-98.")

        figname = str(session) + "_BoxPLot0-49_" + interpolation + "_" + str(rate_used) + "Hz"
        plt.figure(figname, figsize=[12, 16])
        sns.boxplot(data=clean_dataset[0][:, 0:50], orient="h")
        plt.ylabel("Canales")
        plt.xlabel("Voltaje")
        plt.title("Diagrama de caja de los canales 0-49.\nSesión: " + str(session) + ". Interpolada con: "
                  + interpolation + " a " + str(rate_used) + "Hz")
        ui.store_figure(figname, True)

        figname = str(session) + "_BoxPLot50-98_" + interpolation + "_" + str(rate_used) + "Hz"
        plt.figure(figname, figsize=[12, 16])
        sns.boxplot(data=clean_dataset[0][:, 50:99], orient="h")
        plt.ylabel("Canales")
        plt.xlabel("Voltaje")
        plt.title("Diagrama de caja de los canales 50-98.\nSesión: " + str(session) + ". Interpolada con: "
                  + interpolation + " a " + str(rate_used) + "Hz")
        ui.store_figure(figname, True)

        figname = str(session) + "_BoxPLotChannels_" + interpolation + "_" + str(rate_used) + "Hz"
        fig = plt.figure(figname)
```

```

        # ------- STEP 10 -------
        Env.step("Plotting Boxplot of interpolated angles.")

        figname = str(session) + "_BoxPLotAngles_" + interpolation + "_" + str(rate_used) + "Hz"
        plt.figure(figname)
        sns.boxplot(x=clean_dataset[0][:, -1])
        plt.ylabel("Muestras")
        plt.xlabel("Ángulos")
        plt.title("Diagrama de caja de los ángulos.\nSesión: " + str(session) + ". Interpolada con: "
                  + interpolation + " a " + str(rate_used) + "Hz")
        ui.store_figure(figname, True)

    # ------- FINISH TEST AND EXIT -------
    Env.finish_test()
    #Env.finish_test(str(session) + "_" + interpolation.title() + "_" + sync_method.replace(" ", ""))

# ------- Execute Dataset generation -------
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

```
# ##########################################################################
#                    University of Costa Rica
#                Electrical Engineering Department
#                          Grade Thesis
# ##########################################################################

"""
@file DatasetGenerator.py
@author Pablo Avila [B30724] jose.avilalopez@ucr.ac.cr
@copyright MIT License
@date July, 2021
@details Properly loads the input data and labels and prepare a clean dataset to finally export it as a
```

```
pickle file.
"""

from Yomchi import \
    Environment as Env, \
    preprocessing as data, \
    visualization as ui

import matplotlib.pyplot as plt
import numpy as np
import tensorflow as tf
import pickle


def data_set_generator():
    """
    @details
    <h1> Dataset Generator:</h1>
    <h2> Setup </h2>
    <ul>
        <li> Initialize Environment.
        <li> Initialize Tensorflow session.
        <li> Set seed for Numpy and Tensorflow
        <li> Specify run configuration parameters.
        <li> Specify dataset generation parameters.
    </ul>
    <h2> Procedure </h2>
    <ul>
        <li> <b>Step 1:</b> Import LFP data.
        <li> <b>Step 2:</b> Import angles data.
        <li> <b>Step 3:</b>
        <ul>
            <li> Downsample LFP data to match the Angles sampling rate or
            <li> Fill angles gaps to match the LFP sampling rate.
        </ul>
        <li> <b>Step 4:</b> Label data by concatenating LFPs and Angles in a single 2D-array.
        <li> <b>Step 5:</b> Clean the labeled dataframe from -1 values, which represent the wrongly acquired
        positional samples.
        <li> <b>Step 6:</b> Interpolate angles data using an 'interpolation' approach.
        <li> <b>Step 7:</b> Plotting Angles and the selected channel of the LFPs after cleaning and interpolation.
        <li> <b>Step 8:</b>
        <ul>
            <li> Get preferred angle according to LFPs channel.
            <li> Plotting the sum of LFPs per angle from 0 to 359 degrees.
        </ul>
        <li> <b>Step 9:</b> Get largest subset of data.
        <li> <b>Step 10:</b> Split the subset into training, validation and testing set.
        <li> <b>Step 11:</b> Save the dataset to a pickle file.
        <li> <b>Step 12:</b> Convert data to windowed series.
        <li> <b>Finish Test and Exit: </b> Save logs, captures, results and the generated dataset into a .pickle file.
    </ul>
    """

    # ------- SETUP -------
    Env.init_environment(True, True)

    tf.keras.backend.clear_session()
    tf.random.set_seed(51)
    np.random.seed(51)

    # Dataset Generation Parameters
    # The recording session 771 has a 23.81% of invalid positions, while the session 765 has only 6.98%
    session = 771  # 771 or 765
    interpolation = "Shortest"  # "linear" "quadratic" "cubic" "nearest" "Shortest"
    sync_method = "Downsample LFPs"  # "Upsample Angles" "Downsample LFPs"

    # Data Properties
    num_channels = data.EC014_41_NUMBER_OF_CHANNELS
    rate_used = data.POSITION_DATA_SAMPLING_RATE
    if sync_method == "Upsample Angles":
        rate_used = data.LFP_DATAMAX_SAMPLING_RATE

    # Windowing properties
    window_size = 4  # Recommended between 100ms to 200ms.
    batch_size = 32
    shuffle_buffer = 1000
    lfp_channel = 70
    round_angles = False
    base_angle = 0  # Unused if round_angles = False
    offset_between_angles = 30  # Unused if round_angles = False

    extra = {"Round angles to get discrete label": round_angles}
    if round_angles:
        extra.update({"Angle labels starting from": str(base_angle) + "°",
```

```python
                        "until 360° in steps of": str(offset_between_angles) + "°"})

    o_pickle_file_name = f"S-{session}_C{lfp_channel}_I-{interpolation}_F-{rate_used}_W-" \
                         f"{int(window_size * 1e3 / rate_used)}ms.pickle"

    parameters_dictionary = {"Recording Session Number": str(session),
                             "Interpolation Method": interpolation.title(),
                             "Synchronization Method": sync_method,
                             "Number of Recorded Channels": str(num_channels),
                             "Sampling Rate to Use": str(rate_used) + " Hz",
                             "Window Size": window_size,
                             "Batch size (# of windows)": batch_size,
                             "LFP Signal channel to use": lfp_channel,
                             "Shuffle buffer size": shuffle_buffer}

    parameters_dictionary.update(extra)
    Env.print_parameters(parameters_dictionary)

    # ------- STEP 1 -------
    Env.step(f"Importing LFP data from session: {session}")
    lfp_data = data.load_lfp_data(data.LFP[session])

    # ------- STEP 2 -------
    Env.step(f"Importing angles data from session: {session}")
    angles_data = data.load_angles_data(data.ANGLES[session])

    # ------- STEP 3 -------
    if sync_method == "Downsample LFPs":
        Env.step("Downsample LFP data to match Angles sampling rate.")
        lfp_data = data.downsample_lfps(lfp_data, data.LFP_DATAMAX_SAMPLING_RATE, data.POSITION_DATA_SAMPLING_RATE)

    elif sync_method == "Upsample Angles":
        Env.step("Upsample Angles data to reach a higher sampling rate")
        angles_data = data.angles_expansion(angles_data, data.POSITION_DATA_SAMPLING_RATE,
                                            data.LFP_DATAMAX_SAMPLING_RATE)

    # ------- STEP 4 -------
    Env.step("Label data by concatenating LFPs and interpolated Angles in a single 2D-array.")
    # IF round_angles: Rounding the angles to be discrete labels, starting from 'base_angle' until 360 on steps of
    # 'offset_between_angles'. Else, the angles are not rounded to discrete labels.
    labeled_data = data.add_labels(lfp_data, np.expand_dims(angles_data, axis=1), round_angles,
                                   base_angle, offset_between_angles)

    # ------- STEP 5 -------
    Env.step("Clean the labeled dataset from discontinuities in positional data (angles).")

    clean_datasets = data.clean_invalid_positional(labeled_data)

    # ------- STEP 6 -------
    Env.step(f"Interpolate angles data using a {interpolation} approach.")

    # Get all LFP channels, excluding the angles.
    clean_interpolated_data = [s[:, :-1] for s in clean_datasets]

    # Interpolate angles and add them to the dataset
    clean_interpolated_angles = [data.interpolate_angles(s[:, -1], interpolation) for s in clean_datasets]

    for i in range(len(clean_interpolated_data)):
        clean_interpolated_data[i] = \
            np.concatenate((clean_interpolated_data[i], np.expand_dims(clean_interpolated_angles[i], axis=1)), axis=1)

    # ------- STEP 7 -------
    Env.step("Plotting Angles and one channel of the LFPs after cleaning and interpolation.")

    Env.print_text(f"Plotting Angles data [°] at {rate_used}Hz")
    figname = f"{session}_Angles_degrees_{int(rate_used)}Hz"
    plt.figure(figname)
    plt.plot(clean_interpolated_data[4][:, -1], "xr")
    plt.title(f"Información de ángulos [°]. Sesión: {session} a {rate_used}Hz")
    ui.store_figure(figname, show=Env.debug)

    Env.print_text(f"Plotting LFPs Channel {lfp_channel} at {rate_used}Hz")

    figname = f"{session}_LFP_c{lfp_channel}_{int(rate_used)}Hz"
    plt.figure(figname)
    plt.plot(clean_interpolated_data[4][:, lfp_channel], "x-r")
    plt.title(f"LFPs del Canal {lfp_channel}. Sesión: {session} a {rate_used}Hz")
    ui.store_figure(figname, show=Env.debug)

    # ------- STEP 8 -------
    Env.step("Get Preferred angle.")

    angles = [0] * 361

```

```
    for subset in clean_interpolated_data:
        for index in range(len(subset)):
            angles[int(round(subset[index, -1]))] += abs(subset[index, lfp_channel])

    angles[0] += angles[360]
    angles = angles[:-1]
    preferred_angle = angles.index(max(angles))

    Env.print_text(f" Preferred angle according to LFP Channel {lfp_channel}: {preferred_angle}°")

    Env.print_text(f"Plotting Preferred angle according to LFP Channel {lfp_channel}")
    figname = f"{session}_preferred_angle_LFP_c{lfp_channel}_{int(rate_used)}Hz"
    plt.figure(figname)
    plt.plot(range(360), angles, "x-r")
    plt.title(f"Suma de LFPs del Canal {lfp_channel} por ángulo. Sesión: {session}.")
    plt.xlabel(f"Ángulo en grados")
    ui.store_figure(figname, show=Env.debug)

    # ------- STEP 9 -------
    Env.step("Get largest subset of data.")

    max_length = 0
    max_subset_index = 0
    for index, subset in enumerate(clean_interpolated_data):
        if max_length < len(subset):
            max_length = len(subset)
            max_subset_index = index

    largest_subset = clean_interpolated_data[max_subset_index]
    Env.print_text(f"Shape of the largest subset of data : [{len(largest_subset)},{len(largest_subset[0])}]
")

    # ------- STEP 10 -------
    Env.step()

    n = len(largest_subset)
    train_array = largest_subset[0:int(n * 0.7), :]
    valid_array = largest_subset[int(n * 0.7):int(n * 0.9), :]
    test_array = largest_subset[int(n * 0.9):, :]

    Env.print_text(f"Training data shape: [{len(train_array)},{len(train_array[0])}]")
    Env.print_text(f"Validation data shape: [{len(valid_array)},{len(valid_array[0])}]")
    Env.print_text(f"Test data shape: [{len(test_array)},{len(test_array[0])}]")

    # ------- STEP 11 -------
    Env.step(f"Save the dataset to pickle file: {o_pickle_file_name}.")

    with open(f"{Env.RESULTS_FOLDER}/{o_pickle_file_name}", 'wb') as f:
        pickle.dump([train_array, valid_array, test_array], f)

    # ------- STEP 12 -------
    Env.step("Convert data to windowed series.")

    train_data = data.channels_to_windows(train_array, lfp_channel, window_size, batch_size, shuffle_buffer
)
    val_data = data.channels_to_windows(valid_array, lfp_channel, window_size, batch_size, shuffle_buffer)
    test_data = data.channels_to_windows(test_array, lfp_channel, window_size, batch_size, shuffle_buffer)

    # The shape should be (batch, time, features) to be compatible with what tensorflow expects as default.
    for example_inputs, example_labels in train_data.take(1):
        Env.print_text(f'Inputs shape (batch, time, samples): {example_inputs.shape}')
        Env.print_text(f'Labels shape (batch, time, labels): {example_labels.shape}')

    # ------- FINISH TEST AND EXIT -------
    Env.finish_test()
    #Env.finish_test(str(session) + "_" + interpolation.title() + "_" + sync_method.replace(" ", ""))


# ------- Execute Dataset generation -------
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

```
# ########################################################################
#                  University of Costa Rica
#              Electrical Engineering Department
#                        Grade Thesis
# ########################################################################

"""
@file Predictor.py
@author Pablo Avila [B30724] jose.avilalopez@ucr.ac.cr
@copyright MIT License
@date July, 2021
@details 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.
"""

from Yomchi import \
    Environment as Env, \
    preprocessing as data, \
    models, \
    visualization as ui

import numpy as np
import tensorflow as tf
import matplotlib.pyplot as plt
import pickle


def predictor():
    """
    @details
    <h1> Predictor </h1>
    <h2> Experiment Setup </h2>
    <ul>
        <li> Initialize Environment
        <li> Initialize Tensorflow session.
        <li> Set seed for Numpy and Tensorflow
        <li> Specify run configuration parameters.
        <li> Specify run configuration parameters.
        <li> Specify dataset generation parameters.
        <li> Specify model’s training parameters.
    </ul>
    <h2> Procedure </h2>
    <ul>
        <li> <b>Step 1:</b> Import training, validation and test datasets of LFPs as inputs and Angles as labels.
        <li> <b>Step 2:</b> Convert data to windowed series.
```

```
        <li> <b>Step 3:</b> Create the model with the specified parameters
        <li> <b>Step 4:</b> Train the model with the training and validation data.
        <li> <b>Step 5:</b> Plotting the original angular data vs the predictions for a determined number of batches
        <li> <b>Finish Test and Exit:</b> Save logs, captures and results.
    </ul>
    """

    # ------- SETUP -------
    Env.init_environment(True, True)

    tf.keras.backend.clear_session()
    tf.keras.backend.set_floatx('float64')
    tf.random.set_seed(51)
    np.random.seed(51)

    # Session, data and methods Parameters
    session = 771  # 765 or 771
    interpolation = "Shortest"  # or "linear", "quadratic", "cubic", "nearest", "Shortest"
    rate_used = data.POSITION_DATA_SAMPLING_RATE  # or data.POSITION_DATA_SAMPLING_RATE

    # Windowing properties
    window_size = 4 # Equals to 100ms at 1250Hz. Recommended between 100ms to 200ms
    batch_size = 32
    shuffle_buffer = 1000
    lfp_channel = 70
    batches_to_plot = 20

    # Parameters
    units_per_layer = 32
    dropout = None
    # dropout = 0.60
    layers = 3
    epochs = 20
    model_name = "LSTM" #  "LSTM" "MLP" or "CNN"

    # Input pickle file.
    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"
    parameters_dictionary = {"Recording Session Number": str(session),
                             "Interpolation Method": interpolation.title(),
                             "Sampling Rate to Use": str(rate_used) + "Hz",
                             "LFP Signal channel to use": lfp_channel,
                             "Shuffle buffer size": str(shuffle_buffer),
                             "Pickle file used": i_pickle_file_name,
                             "Batches to Plot": batches_to_plot,
                             "Batch size (# of windows)": batch_size,
                             "Window Size": window_size,
                             "Model Name": model_name,
                             "Layers" : str(layers),
                             "Epochs" : str(epochs),
                             "Units per Layer" : str(units_per_layer),
                             "Dropout Regularization %": str(dropout)
                             }

    Env.print_parameters(parameters_dictionary)

    # ------- STEP 1 -------
    Env.step("Import training, validation and test datasets of LFPs as inputs and Angles as labels.")

    with open(f"{data.PATH_TO_DATASETS}Pickles/{i_pickle_file_name}", "rb") as f:
        train_array, valid_array, test_array = pickle.load(f)

    # ------- STEP 2 -------
    Env.step("Convert data to windowed series.")

    # The train data is shuffled so the network is trained with a variety of examples.
    train_data = data.channels_to_windows(train_array, lfp_channel, window_size, batch_size, shuffle_buffer)

    # The validation and testing data are not shuffled because the intention is to predict the series in order.
    val_data = data.channels_to_windows(valid_array, lfp_channel, window_size, batch_size)
    test_data = data.channels_to_windows(test_array, lfp_channel, window_size, batch_size)

    for example_inputs, example_labels in train_data.take(1):
      Env.print_text(f'Inputs shape (batch, time, channels): {example_inputs.shape}')
      Env.print_text(f'Labels shape (batch, time, labels): {example_labels.shape}')

    # ------- STEP 3 -------
    Env.step("Create the model with the specified parameters ")

    if model_name == "LSTM":
        model = models.lstm(units_per_layer)
    elif model_name == "CNN":
        model = models.cnn(window_size, units_per_layer)
```

```
    else:
        model = models.mlp(layers, units_per_layer, dropout)

    for example_inputs, example_labels in train_data.take(1):
      Env.print_text(f'Input shape: {example_inputs.shape}')
      Env.print_text(f'Output shape: {model(example_inputs).shape}')

    # ------- STEP 4 -------
    Env.step("Train the model with the training and validation data.")

    history = models.compile_and_fit(model, train_data, val_data, epochs=epochs, patience=epochs)

    Env.print_text(f"\nTraining metrics history:")
    Env.print_text(f"Loss: {history.history['loss']}")
    Env.print_text(f"Mean Absolute Error: {history.history['mean_absolute_error']}")
    Env.print_text(f"\nValidation metrics history:")
    Env.print_text(f"Loss: {history.history['val_loss']}")
    Env.print_text(f"Mean Absolute Error: {history.history['val_mean_absolute_error']}")

    # ------- STEP 5 -------
    Env.step(f"Plotting the original angular data vs the predictions of {batches_to_plot} batches")

    lfps = []
    real = []
    pred = []
    for inputs, label in test_data.take(batches_to_plot):
        # Getting original data
        real = np.append(real, label[:, 0, 0].numpy())
        lfps = np.append(lfps, inputs[:, 0, 0].numpy())
        # Getting predicted data
        pred = np.append(pred, model.predict(inputs))

    # Plotting
    figname = f"predictions_{model_name}_S{session}"
    plt.figure(figname, figsize=(12, 8))
    plt.subplot(211)
    plt.title(f"Ángulos calculados contra originales")
    plt.ylabel('Ángulos')
    plt.scatter(range(len(real)), real, edgecolors='k', label='Originales', c='#2ca02c', s=16)
    plt.scatter(range(len(pred)), pred, marker='X', edgecolors='k', label='Predicciones', c='#ff7f0e', s=16)
    plt.legend()

    plt.subplot(212)
    plt.title(f'Canal de LFP: {lfp_channel}')
    #plt.plot(lfps, "-")
    plt.scatter(range(len(lfps)), lfps, edgecolors='k', label='LFPs', c='#0a7fdb', s=16)
    plt.legend()
    ui.store_figure(figname, show=Env.debug)

    # ------- FINISH TEST AND EXIT -------
    Env.finish_test()
    # Env.finish_test(f"M-MLP_S-{session}_I-{interpolation}_F-{rate_used}")


# ------- Execute Dataset generation -------
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

```
# 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](https://github.com/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](https://crcns.org/data-sets/hc/hc-3/about-hc-3/?searchterm=LFP).

* 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

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

```
# ##########################################################################
#                    University of Costa Rica
#                Electrical Engineering Department
#                          Grade Thesis
# ##########################################################################

"""
@file Environment.py
@author Pablo Avila [B30724] jose.avilalopez@ucr.ac.cr
@copyright MIT License
@date May, 2020
@details 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.
"""

import os
import sys
import inspect
import platform
import datetime
import numpy as np

# Module Constants
START_TIME = datetime.datetime.now()
CURRENT_FOLDER = os.getcwd()
RESULTS_FOLDER = CURRENT_FOLDER + "/Results-" + START_TIME.strftime("%Y-%m-%d_%H-%M") + "/"
LOGS_FOLDER = RESULTS_FOLDER + "Logs/"
CAPTURES_FOLDER = RESULTS_FOLDER + "Captures/"
VERSIONS_LOG_PATH = LOGS_FOLDER + "versions_log.txt"
YOMCHI_LOG_PATH = LOGS_FOLDER + "yomchi_log.txt"
LINE_LENGTH = 100

# Module Variables
step_number = 1
versions_log = None
yomchi_log = None
caller_file = None
debug = False


def init_environment(print_the_header=False, enable_debug=False):
    """
    Initialize the directories and files to log the results.
    @param print_the_header: If True, print the header
    @param enable_debug: If True, run some code to help debugging
    """
    global versions_log
    global yomchi_log
    global caller_file
    global debug

    debug = enable_debug
    os.makedirs(RESULTS_FOLDER)
    os.makedirs(LOGS_FOLDER)
    os.makedirs(CAPTURES_FOLDER)

    # First get the full filename (including path and file extension)
    caller_frame = inspect.stack()[1]
    caller_filename = caller_frame.filename

    # Now get rid of the directory and extension
    caller_file = os.path.basename(caller_filename)

    yomchi_log = open(YOMCHI_LOG_PATH, "w+")

    versions_log = open(VERSIONS_LOG_PATH, "w+")
```

```
    if print_the_header:
        print_header()

    log_versions()


def print_text(text):
    """
    Print a passed text and save it in the yomchi_log.txt.
    @param text: Text to print
    @return None
    """
    global yomchi_log

    print(text)
    yomchi_log.write(text + "\n")


def print_box(text):
    """
    Print a passed text in a box and save it in the yomchi_log.txt.
    @param text: Text to print
    @return None
    """
    text_length = len(text)
    counter = 0
    for i in range(0, text_length):
        if text[i] == ’\n’:
            counter = 0
        else:
            counter += 1

        if counter == LINE_LENGTH:
            if text[i] in [’a’, ’e’, ’i’, ’o’, ’u’]:
                text = text[:i - 1] + ’-’ + ’\n’ + text[i - 1:]
            else:
                text = text[:i] + ’\n’ + text[i:]
            counter = 0

    text_array = text.splitlines()
    print_text("\n|" + "-" * (LINE_LENGTH + 2) + "|")
    for line in text_array:
        print_text("| " + line.ljust(LINE_LENGTH, ’ ’) + " |")
    print_text("|" + "-" * (LINE_LENGTH + 2) + "|")


def step(description=None, new_step_number=0):
    """
    Print the current step number with a brief description.
    @param description: Description of the step.
    @param new_step_number: If defined, reset the steps numeration to the passed value.
    @return None
    """
    global step_number

    larger_string = 7
    pad = round((LINE_LENGTH - larger_string) / 2)

    if new_step_number != 0:
        step_number = new_step_number

    text = " " * pad + "Step " + str(step_number)
    if description is not None:
        text = text + "\n" + description

    print_box(text)
    step_number += 1


def print_parameters(params_dictionary):
    """
    Print a list of parameters.
    @param params_dictionary: Dictionary of parameters {name: value} to print.
    @return None
    """

    pad = round((LINE_LENGTH - 18) / 2)
    title = " " * pad + "*** Parameters ***"

    names = list(params_dictionary.keys())
    values = list(params_dictionary.values())
    longest_name = max(names, key=len)

    new_names = []
    for name in names:
        new_names.append(name + " " * (len(longest_name) - len(name)) + " : ")
```

```
    new_dictionary = np.char.add(new_names, values)

    print_text("\n|" + "-" * (LINE_LENGTH + 2) + "|")
    print_text("| " + title.ljust(LINE_LENGTH, ' ') + " |")
    print_text("| " + ''.ljust(LINE_LENGTH, ' ') + " |")
    for param in new_dictionary:
        print_text("| " + param.ljust(LINE_LENGTH, ' ') + " |")
    print_text("|" + "-" * (LINE_LENGTH + 2) + "|\n")


def log_versions():
    """
    Stores the environment versions including packages, libraries and OS.
    @return None
    """
    global versions_log
    required_packages = ["Keras", "numpy", "pandas", "matplotlib", "scikit-learn", "seaborn",
                         "sklearn", "tensorflow", "pip"]

    try:
        from pip._internal.operations import freeze
    except ImportError:  # pip < 10.0
        from pip.operations import freeze

    try:
        distro = platform.linux_distribution()
    except:
        distro = ["N/A", ""]

    versions_log.write("\n\nOperation System: ")
    versions_log.write("\n  Kernel: " + platform.system() + " " + platform.release())
    versions_log.write("\n  Distribution: " + distro[0] + " " + distro[1])

    versions_log.write("\n\nPython version: " + str(sys.version_info[0]) + "."
                       + str(sys.version_info[1]) + "." + str(sys.version_info[2]))
    versions_log.write("\n\nPackages versions:\n")

    list = freeze.freeze()

    versions_log.write("  Required Packages:")
    for package in list:
        for required in required_packages:
            if package.find(required) != -1:
                index = package.find("==")
                package = package.replace("==", " " * (25 - index) + "=  ")
                versions_log.write("\n    " + package)

    list = freeze.freeze()
    versions_log.write("\n\n  Complete List:")
    for package in list:
        index = package.find("==")
        package = package.replace("==", " " * (25 - index) + "=  ")
        versions_log.write("\n    " + package)


def print_header():
    """
    Print a header in the console output and log files.
    @return None
    """
    larger_string = 33
    pad = round((LINE_LENGTH - larger_string) / 2)
    header = "#" * LINE_LENGTH + "#" * 3 \
             + "\n#" + " " * pad + "                                 " + " " * pad + "#" \
             + "\n#" + " " * pad + "    University of Costa Rica     " + " " * pad + "#" \
             + "\n#" + " " * pad + "Electrical Engineering Department" + " " * pad + "#" \
             + "\n#" + " " * pad + "          Grade Thesis           " + " " * pad + "#" \
             + "\n#" + " " * pad + "      Pablo Avila [B30724]       " + " " * pad + "#" \
             + "\n#" + " " * pad + "    jose.avilalopez@ucr.ac.cr    " + " " * pad + "#" \
             + "\n#" + " " * pad + "                                 " + " " * pad + "#" \
             + "\n" + "#" * LINE_LENGTH + "#" * 3 \
             + "\n" + " " * pad + "    *** START OF TEST ***   "\
             + "\n# File       : " + caller_file \
             + "\n# Date       : " + START_TIME.strftime("%m-%d-%Y") \
             + "\n# Start Time: " + START_TIME.strftime("%H:%M:%S") + "\n"
    print_text(header)
    versions_log.write(header)

def finish_test(rename_results_folder=None):
    """
    Safely finish the test run and log some final information.
    @param rename_results_folder: Optional name for the results folder.
    @return None
    """
    global yomchi_log
    global versions_log
```

```
    larger_string = 33
    pad = round((LINE_LENGTH - larger_string) / 2)

    finish_time = datetime.datetime.now()
    elapsed_time = finish_time - START_TIME

    days = elapsed_time.days
    hours, rem = divmod(elapsed_time.seconds, 3600)
    minutes, seconds = divmod(rem, 60)

    footer = "\n" + "#" * LINE_LENGTH + "#" * 3 \
             + "\n" + " " * pad + "     *** END OF TEST ***  " \
             + "\n# Test Duration : " + str(days) + str(hours) + ":" + str(minutes) + ":" + str(seconds) \
             + "\n# Finish Time   : " + finish_time.strftime("%H:%M:%S") \
             + "\n" + "#" * LINE_LENGTH + "#" * 3

    print_text(footer)

    yomchi_log.close()
    versions_log.close()

    if rename_results_folder is not None:
        os.rename(RESULTS_FOLDER, CURRENT_FOLDER + "/" + rename_results_folder + "/")

    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

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

```
# ##########################################################################
#                       University of Costa Rica
#                  Electrical Engineering Department
#                              Grade Thesis
# ##########################################################################

"""
@file models.py
@author Pablo Avila [B30724] jose.avilalopez@ucr.ac.cr
@copyright MIT License
@date July, 2021
@details 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.
"""
import Yomchi.Environment as Env
import tensorflow as tf


def 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.
    @param layers: Number of hidden layers (besides the input and outputs ones).
    @param units_per_layer: Number of neurons per layer. Applies to all layers except for the last one which has only 1:
    the predicted angle.
    @param dropout: 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.
    @return model: The model of the MLP created for later usage as the predictor.
    """

    Env.print_text(f"Creating a fully-connected feed-forward Neural Network model with {layers} layers, using "
                   f"{units_per_layer} units per layer. \nThe activation function in the hidden layers is 'ReLu' and "
                   f"the output layer has only 1 unit (the predicted angle).")
    model = tf.keras.Sequential()

    # Shape: (time, features) => (time*features)
    model.add(tf.keras.layers.Flatten())

    # First layer (need to specify the input size)
    model.add(tf.keras.layers.Dense(
        units=units_per_layer,
        activation=tf.nn.relu))

    if dropout is not None:
        model.add(tf.keras.layers.Dropout(dropout))

    # Other hidden layers
    for n in range(1, layers):
        model.add(tf.keras.layers.Dense(
            units=units_per_layer,
            activation=tf.nn.relu))

    if dropout is not None:
        model.add(tf.keras.layers.Dropout(dropout))

    # Output layer
    model.add(tf.keras.layers.Dense(
        units=1
    ))

    # Add back the time dimension.
```

```python
    # Shape: (outputs) => (1, outputs)
    model.add(tf.keras.layers.Reshape([1, -1]))

    return model


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. The first two layer has ReLU activation and the last
    one uses a linear activation function.S
    @param inputs: 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.
    @param units_per_layer: 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.
    @return conv_model: The model of the CNN created for later usage as the predictor.
    """
    Env.print_text(f"Creating a Convolutional Neural Network model with one 1D convolutional layer, using "
                   f"{units_per_layer} filters and units in the following dense layer. \nThe activation function in "
                   f"the dense layer is 'ReLU' and the output dense layer has only 1 unit (the predicted angle).")

    conv_model = tf.keras.Sequential([
        tf.keras.layers.Conv1D(filters=units_per_layer,
                               kernel_size=(inputs,),
                               activation='relu'),
        tf.keras.layers.Dense(units=units_per_layer, activation='relu'),
        tf.keras.layers.Dense(units=1),
    ])

    return conv_model


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. The activation functions of the LSTM layer are the regular ones for each of it's gates.
    @param units_per_layer: Number of inputs of the network.
    @return lstm_model: The model of the LSTM created for later usage as the predictor.
    """

    Env.print_text(f"Creating a Long Short-term Memory (LSTM) Neural Network using {units_per_layer} units per layer."
                   f"\nThis network only outputs the final timestamp, giving the model time to warm up its internal  "
                   f"state before making a single prediction.")

    lstm_model = tf.keras.models.Sequential([
        # Shape [batch, time, features] => [batch, time, lstm_units]
        tf.keras.layers.LSTM(units_per_layer, return_sequences=False),
        # Shape => [batch, time, features]
        tf.keras.layers.Dense(units=1)
    ])

    return lstm_model


def 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.
    @param model: Model to train. It can be a MLP, CNN or LSTM, among others.
    @param train: The dataset for training the model. Must have the shape: (batch, time, features)
    @param val: The dataset for validating the model. Must have the shape: (batch, time, features)
    @param epochs: Number of iteration over the entire set of data (all the batches).
    @param patience: 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.
    @return history: The results of the training.
    """

    Env.print_text(f"Compiling the input model {model.name} with {epochs} epochs. The "
                   f"loss function is the MSE and metric to evaluate improvement is the MAE.")

    early_stopping = tf.keras.callbacks.EarlyStopping(monitor='val_loss',
                                                      patience=patience,
                                                      mode='min')
```

```

    model.compile(loss=tf.losses.MeanSquaredError(),  # tf.losses.mean_absolute_percentage_error()
                  metrics=[tf.metrics.MeanAbsoluteError()])

    history = model.fit(train, epochs=epochs,
                        validation_data=val,
                        callbacks=[early_stopping])

    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 $<$ 625Hz) 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

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

```
# ###########################################################################
#                    University of Costa Rica
#                Electrical Engineering Department
#                          Grade Thesis
# ###########################################################################

"""
@file preprocessing.py
@author Pablo Avila [B30724] jose.avilalopez@ucr.ac.cr
@copyright MIT License
@date July, 2021
@details This module contains a set of functions to import clean, parse and reshape input data.
"""

import os
import pandas as pd
import pickle
import scipy.signal
import numpy as np
import tensorflow as tf
import Yomchi.Environment as Env

# Dataset Paths
PATH_TO_DATASETS = os.path.join(Env.CURRENT_FOLDER, "../Datasets/")
LFP = {765: PATH_TO_DATASETS + "ec014.765.eeg",
       771: PATH_TO_DATASETS + "ec014.771.eeg"}
ANGLES = {765: PATH_TO_DATASETS + "ec014.765.whl",
          771: PATH_TO_DATASETS + "ec014.771.whl"}

# Sampling Rates
RAW_DATAMAX_SAMPLING_RATE   = 20000
RAW_NEURALYNX_SAMPLING_RATE = 32552
LFP_DATAMAX_SAMPLING_RATE   = 1250
LFP_NEURALYNX_SAMPLING_RATE = 1252
POSITION_DATA_SAMPLING_RATE = 39.06
EC014_41_NUMBER_OF_CHANNELS = 99
```

```python


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 < 625Hz) or a .dat file (LFP Data + Spikes).
    @param file: Path to the file containing the animal LFP data.
    @param channels: Number of recorded channels in LFP signals file.
    @return lfp: Array (n x channels) with the data. With the columns being the channels and the rows the
    a different time step.
    """
    Env.print_text("Loading LFP Data of " + str(channels) + " channels from: " + os.path.basename(file) + ".")

    signals = open(file, "rb")
    signalsArray = np.fromfile(file=signals, dtype=np.int16)
    lfp = np.reshape(signalsArray, (-1, channels))
    signals.close()

    Env.print_text("LFP Data shape: " + str(np.shape(lfp)))

    return lfp


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. If any position
    value equals '-1' then it's replaced with 'NaN' instead.
    @param file: Path to the file containing the animal LED's position information
    @param degrees: If this flag is set, then the angles are returned in degrees from [0, 360[, or radians otherwise.
    @return angles: Array with the angles in radians extracted from the positions. The angles are given as
    float16 values calculated as arctan(y2 - y1 /x2 - x1). Unless the denominator is 0, in that case '0' is returned
    for that element.
    """
    Env.print_text("Loading the animal position data from: " + os.path.basename(file) + ".")

    positions_file = open(file, "rb")
    positions = np.genfromtxt(fname=positions_file, dtype=np.float16, delimiter='\t')
    positions_file.close()

    positions[positions == -1] = np.NAN
    #TODO: of x_2 - x_1 = 0, angles should be NaN.

    angles = np.arctan2((np.subtract(positions[:, 3], positions[:, 1])),
                        (np.subtract(positions[:, 2], positions[:, 0])))

    if degrees:
        angles = np.degrees(angles)
        angles += 180

    angles[np.isnan(angles)] = -1
    invalid = np.count_nonzero(angles == -1)

    Env.print_text("Head angle data shape: " + str(np.shape(angles)))
    Env.print_text(f"Head angle invalid data: {invalid:d} ({invalid/len(angles) * 100:.2f}%)")

    return angles


def 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.
    @param lfp_data: Matrix [n x numChannels] with the LFP signals
    @param orig_rate: Sampling rate originally used to acquire the data.
    @param new_rate: New sampling rate of the data.
    @return resampled_data: Original data downsampled to the new rate.
    """
    Env.print_text("Downsampling LFP data to match the new sampling rate: " + str(new_rate)
                   + "Hz. Original was: " + str(orig_rate) + "Hz.")

    resampled_data = []
    for channel_i in np.transpose(lfp_data):
        channel_i = scipy.signal.decimate(channel_i, 8)
        channel_i = scipy.signal.decimate(channel_i, 4)
        resampled_data.append(channel_i)

    resampled_data = np.transpose(resampled_data)
    Env.print_text("LFP data decimated shape: " + str(np.shape(resampled_data)))

    return resampled_data

```

```python

def 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.
    @details Assuming that the acquisition of data started and stopped at the same time, then no data has to be added
    after the last sample.
    @param angles_data: Array with the angles data extracted from the animal positions.
    @param orig_rate: Sampling rate originally used to acquire the data.
    @param new_rate: New sampling rate of the data. The gaps are filled with 'NaN'.
    @return upsampled_data: Original data filled with 'NaN' to match the new sampling rate.
    """
    Env.print_text("Expanding angles data with 'NaN' values to match the new sampling rate: " + str(new_rate) + "Hz. "
                   + "Original was: " + str(orig_rate) + "Hz.")

    expansion_factor = round(new_rate/orig_rate)
    padding = np.full((len(angles_data), expansion_factor - 1), np.NaN)
    upsampled_data = np.concatenate((np.transpose(np.array([angles_data])), padding), axis=1)
    upsampled_data = upsampled_data[:-1, :]
    upsampled_data = upsampled_data.flatten()

    Env.print_text("Angles data upsampled shape: " + str(np.shape(upsampled_data)))

    return upsampled_data


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.
    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'.
    """
    shortest_angle = ((end-start) + 180) % 360 - 180
    interpolated_angle = (start + shortest_angle * amount) % 360
    return interpolated_angle


def 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
    """
    start_angle = np.nan
    no_nans = 0
    first_nan_index = 0
    interpolated_angles = angles_data

    for i in range(1, len(interpolated_angles)):
        # If a valid value followed by NaN: this is the first NaN, start counting
        if np.isnan(interpolated_angles[i]) and not np.isnan(interpolated_angles[i-1]):
            start_angle = interpolated_angles[i-1]
            no_nans = 1
            first_nan_index = i

        # If a NaN followed by another NaN: Increment counter 1+.
        elif np.isnan(interpolated_angles[i]) and np.isnan(interpolated_angles[i - 1]):
            no_nans += 1

        # If a NaN followed by a valid value: This is the last NaN, interpolate.
        elif not np.isnan(interpolated_angles[i]) and np.isnan(interpolated_angles[i - 1]):
            if no_nans > 0 and not np.isnan(start_angle):
                amount = 0
                end_angle = interpolated_angles[i]
                for j in range(first_nan_index, first_nan_index + no_nans):
                    amount += 1/(no_nans + 1)
                    interpolated_angles[j] = shortest_angle_interpolation(start_angle, end_angle, amount)
                start_angle = np.nan

    return interpolated_angles


def interpolate_angles(angles_data, method="Shortest"):
    """
```

```
    Replace 'NaN' values in angular data with an interpolated value using a given method.
    @param angles_data: Array with the angles data extracted from the animal positions.
    @param method: 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.
    @return interpolated_angles: Array with the angles data interpolated using the given method.
    """
    Env.print_text("Interpolate angles data using " + method + " method.")

    if method == "Shortest":
        interpolated_angles = vectorized_sai(angles_data)

    else:
        angles_series = pd.Series(angles_data)
        interpolated_angles = angles_series.interpolate(method)
        interpolated_angles = interpolated_angles.to_numpy()

    return interpolated_angles


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

    len_lfps = len(lfps)
    len_angles = len(angles)

    if len_lfps > len_angles:
        lfps = lfps[0: len_angles]
    elif len_lfps < len_angles:
        angles = angles[0: len_lfps]

    Env.print_text("Adding labels to the data by concatenating the [" + str(len(lfps)) + " x " + str(len(lfps[0])) +
                   "] LFP data matrix with the [" + str(len(angles)) + "] Angles vector.")
    Env.print_text("Rounding Labels = " + str(round_labels))

    if round_labels:
        if (0 <= start < 360) and (1 <= offset < 360):
            labels = np.arange(start, 360, offset)

        for i in range(0, len(angles)):
            if not np.isnan(angles[i]):
                minval = np.inf
                label = start
                for tag in labels:
                    diff = abs(angles[i] - tag)
                    if diff < minval:
                        minval = diff
                        label = tag
                angles[i] = label

    labeled_data = np.concatenate((lfps, angles), axis=1)

    return labeled_data


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. Plus the following 31 rows with NaN as angle value in case of padded data.
    @details 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.
    @param labeled_dataset: 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.
    @param is_padded: If True, manage the input labeled dataset as a padded array of angles, or a downsampled LFP set
     otherwise.
    @return clean_dataset: Input data without invalid positional values.
    """
    if is_padded:
```

```
        # Get the indexes of all invalid values (-1) plus the 31 following padded values (NaN).
        # and save that range as an element of 'invalid_indexes' array.
        invalid_indexes = [np.arange(i, i + 32).tolist() for i, v in enumerate(labeled_dataset) if v[-1] == -1]
        Env.print_text(f"Amount of invalid indexes: {len(invalid_indexes)}")

        # Merge all sublists as a single consecutive array of invalid indexes.
        invalid_indexes = [item for sublist in invalid_indexes for item in sublist]
        Env.print_text(f"Amount of invalid indexes + associated expanded indexes: {len(invalid_indexes)}")
        Env.print_text(f"Number of valid samples: {len(labeled_dataset) - len(invalid_indexes)} (including the 31 NaN "
                       f"values padded for the last real positional data.)")
    else:
        # Get the indexes of all invalid values (-1) and save that range as an element of 'invalid_indexes' array.
        invalid_indexes = [i for i, v in enumerate(labeled_dataset) if v[-1] == -1]
        Env.print_text(f"Amount of invalid indexes: {len(invalid_indexes)}")
        Env.print_text(f"Number of valid samples: {len(labeled_dataset) - len(invalid_indexes)}")

    # Get the indexes where the discontinuities start and end.
    discontinuities_starts = [invalid_indexes[0]]
    discontinuities_ends = []
    for i in range(1, len(invalid_indexes)):
        if invalid_indexes[i] - 1 != invalid_indexes[i-1]:
            discontinuities_starts.append(invalid_indexes[i])
            discontinuities_ends.append(invalid_indexes[i-1])
    discontinuities_ends.append(invalid_indexes[-1])

    # Stores sub-arrays of valid indexes:
    clean_datasets = []

    # From the first item to the start of the first discontinuity
    if discontinuities_starts[0] != 0:
        clean_datasets.append(labeled_dataset[0:discontinuities_starts[0], :])

    # From the end+1 of the ith discontinuity to the start (not included) of the ith + 1 discontinuity
    for i in range(len(discontinuities_starts)-1):
        clean_datasets.append(labeled_dataset[discontinuities_ends[i]+1:discontinuities_starts[i+1], :])

    # From the end of the last discontinuity to the last item.
    if discontinuities_ends[-1] != len(labeled_dataset)-1:
        clean_datasets.append(labeled_dataset[discontinuities_ends[-1] + 1:-1, :])

    # Delete subsets with only 1 valid sample (i.e its length is <=32)
    # From the resulting subsets, delete the last 31 rows with NaN angles
    clean_datasets = [v[:-31, :] for v in clean_datasets if len(v) > 32]

    Env.print_text(f"Total number of valid subsets: {len(clean_datasets)}")
    for s in range(0, len(clean_datasets)):
        Env.print_text(f"Total number of samples and channels + angles in subsets {s}: {clean_datasets[s].shape}")

    return clean_datasets


def ndarray_to_dataframe(dataset, rate):
    """
    Converts an n-D Numpy array to a Pandas Dataframe
    @param dataset: 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.
    @param rate:
    @return dataframe: Pandas data frame with Channels 0-99 and Angles as columns names, and the timestamp as indexes
    calculated as 1/rate * 1e6 to get the time step of the acquisition in microseconds.
    """

    columns = []
    for i in range(0, 99):
        columns.append(f"Channel {i}")
    columns.append("Angles")

    time_step = round((1/rate) * 1e6)   # 1/f [us]: 1250Hz => 800us, 39.06Hz => 2560.164us
    indeces = np.arange(0, len(dataset) * time_step, time_step)
    dataframe = pd.DataFrame(data=dataset, columns=columns, index=indeces)

    return 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.
    @param series: Numpy Array with the LFP data of the n channels.
    @param channel: Channel to use.
    @param window_size: Size of the windows in which the data are being split.
```

```
    @param batch_size: Number of pairs data-labels to group as a batch
    @param shuffle_buffer: Number of windows to shuffle at the same time.
    @param offset: Number of samples to shift between windows.
    @return windowed_ds: LFP data of the selected channel separated in windows.
    """

    labels = np.expand_dims(series[window_size::, -1], 1)
    data = series[:, channel]

    # Get the average angle for each window.
    #labels_ds = average_angles(labels, window_size)

    labels_ds = tf.data.Dataset.from_tensor_slices(np.expand_dims(labels, 1))

    # Creates a dataset from the input
    windowed_data = tf.data.Dataset.from_tensor_slices(np.expand_dims(data, 1))

    # Split the data set in windows shifting each window by 1 and forcing them to the same size (window_size + 1)
    windowed_data = windowed_data.window(window_size, shift=offset, drop_remainder=True)

    # Make each window a numpy array row.
    windowed_data = windowed_data.flat_map(lambda window: window.batch(window_size))

    # Add labels to the data
    windowed_ds = tf.data.Dataset.zip((windowed_data, labels_ds))

    if Env.debug:
        for x, y in windowed_ds.take(1):
            print("x = ", x.numpy())
            print("y = ", y.numpy())

    if shuffle_buffer != None:
        # Shuffle the data in groups of shuffle_buffer to accelerate. Instead of shuffle it all at once.
        windowed_ds = windowed_ds.shuffle(shuffle_buffer)

    # Batch the data into sets of 'batch_size'.
    windowed_ds = windowed_ds.batch(batch_size).prefetch(1)

    return windowed_ds


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. Each window is 1 element shifted from the previous window.
    @param angles: Numpy Array with the Angles data to use as the labels.
    @param window_size: Size of the windows in which the data are being split.
    @return average_angles: Averaged Angles data separated in windows.
    """

    # Get an array of labels from the series
    windowed_angles = tf.data.Dataset.from_tensor_slices(angles)

    # Split the data set in windows shifting each window by 1 and forcing them to the same size (window_size + 1)
    windowed_angles = windowed_angles.window(window_size, shift=1, drop_remainder=True)

    # Make each window a numpy array row.
    windowed_angles = windowed_angles.flat_map(lambda window: window.batch(window_size))
    averaged_angles = windowed_angles.map(lambda window: tf.math.reduce_mean(window))

    if Env.debug:
        print("Angles:")
        for window in windowed_angles.take(1):
            print(window.numpy())
        for window in averaged_angles.take(1):
            print(window.numpy())

    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

```
# ##########################################################################
#                    University of Costa Rica
#                Electrical Engineering Department
#                              Grade Thesis
# ##########################################################################

"""
@file visualization.py
@author Pablo Avila [B30724] jose.avilalopez@ucr.ac.cr
@copyright MIT License
@date July, 2021
@details This module contains a set of functions to plot and in general terms, visualize data.
"""

import matplotlib.pyplot as plt
import Yomchi.Environment as Env


# Matplotlib figures parameters
FIG_FORMAT = 'png'
plt.rcParams['figure.dpi'] = 200
plt.rcParams['figure.figsize'] = [12, 8]
plt.rcParams['savefig.format'] = FIG_FORMAT
plt.rcParams['lines.markersize'] = 1
plt.rcParams['font.family'] = 'serif'
plt.rcParams['font.size'] = 18
plt.rcParams['figure.autolayout'] = True
plt.rcParams['axes.titlepad'] = 7
plt.rcParams['axes.grid'] = False


def store_figure(fig_name, show=False):
    """
    Stores a figure.
    @param fig_name: Name of the figure to store.
    @param show: Displays the figure when ready. Warning: Stalls execution until closing it.
    @return None
    """

```

```
    plt.figure(fig_name)
    plt.savefig(fname=Env.CAPTURES_FOLDER + fig_name + "." + FIG_FORMAT)
    if show:
        plt.show()
    plt.close(fig_name)
```

Index