optimization

Release 0.1

Freitas et al.

Jan 27, 2022

CONTENTS

CHAPTER

ONE

TABLE OF CONTENTS

1.1 API Reference

The HOSA library is split into five modules:

- *hosa.optimization*—for implementing the HOSA.
- *hosa.models.cnn*—for implementing classification and regression models using Convolutional Neural Networks (CNN).
- *hosa.models.rnn*—for implementing classification and regression models using Recurrent Neural Network (RNNs).
- *hosa.callbacks*—for implementing early stopping callbacks for halting the model's training.
- *hosa.helpers*—for implementing helper functions for the package.

1.1.1 HOSA

This module implements the Heuristic Oriented Search Algorithm (HOSA).

hosa.BaseHOSA(x, y, model, n_outputs, ...[, ...])	Heuristic Oriented Search Algorithm (HOSA)
hosa.HOSACNN(x, y, model, n_outputs, ...[, ...])	Heuristic Oriented Search Algorithm (HOSA) for CNNs.
hosa.HOSARNN(x, y, model, n_outputs, ...[, ...])	Heuristic Oriented Search Algorithm (HOSA) for RNNs.

hosa.optimization.hosa.BaseHOSA

class hosa.optimization.hosa.**BaseHOSA**(*x*, *y*, *model*, *n_outputs*, *parameters*, *tr*, *apply_rsv=True*, *validation_size=0.25*, *n_splits=10*)

Bases: object

Heuristic Oriented Search Algorithm (HOSA)

This class implments the HOSA. Following a heuristic search, the algorithm finetunes the most relevant models' parameters. Thus, HOSA avoids testing every possible combination, and therefore, an exhaustive search.

> **Warning:** This class should not be used directly. Use derived classes instead, i.e., *HOSACNN* or *HOSARNN*.

Parameters

- **x** (*numpy.ndarray*) – Input data.
- **y** (*numpy.ndarray*) – Target values (class labels in classification, real numbers in regression).
- **model** (*object*) – Class of the object to be optimized. Available options are: *RNNClassification*, *RNNRegression*, *CNNClassification* and *CNNRegression*.
- **n_outputs** (*int*) – Number of class labels in classification, or the number of numerical values to predict in regression.
- **parameters** (*dict*) – Dictionary with parameters names (str) as keys and lists of parameter settings to try as values.
- **tr** (*float*) – Minimum threshold of improvement of the performance metric.
- **apply_rsv** (*bool*) – `True` if random sub-sampling validation should be used during the optimization procedure.
- **validation_size** (*float*) – Proportion of the dataset to include in the validation split on the random sub-sampling validation. **Ignored if ``apply_rsv = False``**.
- **n_splits** (*int*) – Number of splits used in the random sub-sampling validation. **Ignored if ``apply_rsv = False``**.

Methods

fit(**kwargs)	Optimize the model following the HOSA approach with all sets of parameters.
get_model()	Get the best model found.
get_params()	Get parameters for the best model found.
grid_search([n_kernels, ...])	Runs a grid search on the remaining moldel's parameters.
predict(x, **kwargs)	Predicts the target values using the input data in the best model found.
score(x, y, **kwargs)	Computes the performance metrics on the given input data and target values in the best model found.

hosa.optimization.hosa.BaseHOSA.fit

abstract BaseHOSA.**fit**(***kwargs*)

Optimize the model following the HOSA approach with all sets of parameters.

Parameters ****kwargs** – Extra arguments explicitly used for regression or classification models.

hosa.optimization.hosa.BaseHOSA.get_model

BaseHOSA.**get_model**()
Get the best model found.

Returns *tensorflow.keras.Sequential* – Returns the best TensorFlow model found.

hosa.optimization.hosa.BaseHOSA.get_params

BaseHOSA.**get_params**()
Get parameters for the best model found.

Returns *dict* – Parameter names mapped to their values.

hosa.optimization.hosa.BaseHOSA.grid_search

BaseHOSA.**grid_search**(*n_kernels=None*, *n_neurons_dense_layer=None*, *n_units=None*, *n_subs_layers=None*, *imbalance_correction=None*, ***kwargs*)
Runs a grid search on the remaining moldel's parameters.

Parameters

- **n_kernels** (*list or None*) – *i*-th element represents the number of output filters of the convolution layer in the *i*-th GofLayer. **Ignored in the case of optimizing an RNN**.
- **n_neurons_dense_layer** (*int or None*) – Number of neurons of the penultimate dense layer (i.e., before the output layer). **Ignored in the case of optimizing an CNN**.
- **n_units** (*int or None*) – Dimensionality of the output space, i.e., the dimensionality of the hidden state. **Ignored in the case of optimizing an CNN**.
- **n_subs_layers** (*int or None*) – **Ignored in the case of optimizing an CNN**.
- **imbalance_correction** (*None or bool*) – Whether to apply correction to class imbalances. **Only used for classification problems. Ignored for regression.**
- ****kwargs** – Extra arguments explicitly used for regression or classification models, including the additional arguments that are used in the TensorFlow's model `fit` function. See here.

Returns *tensorflow.keras.Sequential* – Returns the best TensorFlow model found.

hosa.optimization.hosa.BaseHOSA.predict

BaseHOSA.**predict**(*x*, ***kwargs*)
Predicts the target values using the input data in the best model found.

Parameters

- **x** (*numpy.ndarray*) – Input data.
- ****kwargs** – Extra arguments that are used in the TensorFlow's model `predict` function. See here.

Returns *numpy.ndarray* – Returns an array containing the estimates that were obtained on the best-fitted model found.

hosa.optimization.hosa.BaseHOSA.score

BaseHOSA.**score**(*x*, *y*, ***kwargs*)

Computes the performance metrics on the given input data and target values in the best model found.

Parameters

- **x** (*numpy.ndarray*) – Input data.
- **y** (*numpy.ndarray*) – Target values (class labels in classification, real numbers in regression).
- ****kwargs** – Only used for classification, in order to set the value of the parameter `imbalance_correction`.

Returns *tuple* – Returns a tuple containing the performance metric according to the type of model.

hosa.optimization.hosa.HOSACNN

class hosa.optimization.hosa.**HOSACNN**(*x*, *y*, *model*, *n_outputs*, *parameters*, *tr*, *apply_rsv=True*, *validation_size=0.25*, *n_splits=10*)

Bases: *hosa.optimization.hosa.BaseHOSA*

Heuristic Oriented Search Algorithm (HOSA) for CNNs.

This class implments the HOSA for optimizing CNNs. Following a heuristic search, the algorithm finetunes the most relevant models' parameters. Thus, HOSA avoids testing every possible combination, and therefore, an exhaustive search.

Parameters

- **x** (*numpy.ndarray*) – Input data.
- **y** (*numpy.ndarray*) – Target values (class labels in classification, real numbers in regression).
- **model** (*object*) – Class of the object to be optimized. Available options are: *RNNClassification*, *RNNRegression*, *CNNClassification* and *CNNRegression*.
- **n_outputs** (*int*) – Number of class labels in classification, or the number of numerical values to predict in regression.
- **parameters** (*dict*) – Dictionary with parameters names (str) as keys and lists of parameter settings to try as values.
- **tr** (*float*) – Minimum threshold of improvement of the performance metric.
- **apply_rsv** (*bool*) – `True` if random sub-sampling validation should be used during the optimization procedure.
- **validation_size** (*float*) – Proportion of the dataset to include in the validation split on the random sub-sampling validation. **Ignored if ``apply_rsv = False``**.
- **n_splits** (*int*) – Number of splits used in the random sub-sampling validation. **Ignored if ``apply_rsv = False``**.

Examples

```
import numpy as np
from sklearn.model_selection import train_test_split

from hosa.models.cnn import CNNRegression
from hosa.optimization.hosa import HOSACNN

# 1 - Load the dataset
dataset = np.loadtxt('...', delimiter=',')
x = dataset[:, :-1]
y = dataset[:, -1]
# 2 - Split the data in train and test dataset
x_train, X_test, y_train, y_test = train_test_split(x, y, test_size=.3,
shuffle=False)
# 3 - Set the parameters to optimize
param_grid_rnn = {
        'overlapping_type':          ['central', 'left'],
        'overlapping_epochs':        [1],
        'n_kernels_first_gol':       [16, 32],
        'activation_function_dense': ['relu'],
        'mults':                     [1, 2],
        'optimizer':                 ['adam'],
        'batch_size':                [32],
}
# 4 - Create a HOSA instance and find the best model
regr = HOSACNN(x_train, y_train, CNNRegression, 1, param_grid_rnn, 0.01,
apply_rsv=False)
regr.fit(max_gol_sizes=4, show_progress=True, verbose=1, shuffle=False)
score = regr.score(X_test, y_test)
```

Methods

fit(max_gol_sizes[, show_progress, ...])	Optimize the model following the HOSA approach with all sets of parameters.
get_model()	Get the best model found.
get_params()	Get parameters for the best model found.
grid_search([n_kernels, ...])	Runs a grid search on the remaining moldel's parameters.
predict(x, **kwargs)	Predicts the target values using the input data in the best model found.
score(x, y, **kwargs)	Computes the performance metrics on the given input data and target values in the best model found.

hosa.optimization.hosa.HOSACNN.fit

HOSACNN.**fit**(*max_gol_sizes*, *show_progress=True*, *imbalance_correction=None*, ***kwargs*)

Optimize the model following the HOSA approach with all sets of parameters.

Parameters

- **max_gol_sizes** (*int*) – Maximum number of GofLayers to add to the model.
- **show_progress** (*bool*) – *True* to show a progress bar; *False* otherwise.
- **imbalance_correction** (*None or bool*) – Whether to apply correction to class imbalances.
- **imbalance_correction** – Whether to apply correction to class imbalances. **Only used for classification problems. Ignored for regression.**
- ****kwargs** – Extra arguments explicitly used for regression or classification models, including the additional arguments that are used in the TensorFlow's model `fit` function. See here.

Returns *tuple* – Returns a tuple containing the object of the best model found and respective performance metric and optimized parameters.

hosa.optimization.hosa.HOSACNN.get_model

HOSACNN.**get_model**()

Get the best model found.

Returns *tensorflow.keras.Sequential* – Returns the best TensorFlow model found.

hosa.optimization.hosa.HOSACNN.get_params

HOSACNN.**get_params**()

Get parameters for the best model found.

Returns *dict* – Parameter names mapped to their values.

hosa.optimization.hosa.HOSACNN.grid_search

HOSACNN.**grid_search**(*n_kernels=None*, *n_neurons_dense_layer=None*, *n_units=None*, *n_subs_layers=None*, *imbalance_correction=None*, ***kwargs*)

Runs a grid search on the remaining moldel's parameters.

Parameters

- **n_kernels** (*list or None*) – *i*-th element represents the number of output filters of the convolution layer in the *i*-th GofLayer. **Ignored in the case of optimizing an RNN**.
- **n_neurons_dense_layer** (*int or None*) – Number of neurons of the penultimate dense layer (i.e., before the output layer). **Ignored in the case of optimizing an CNN**.
- **n_units** (*int or None*) – Dimensionality of the output space, i.e., the dimensionality of the hidden state. **Ignored in the case of optimizing an CNN**.
- **n_subs_layers** (*int or None*) – **Ignored in the case of optimizing an CNN**.

- **imbalance_correction** (*None or bool*) – Whether to apply correction to class imbalances. **Only used for classification problems. Ignored for regression.**
- ****kwargs** – Extra arguments explicitly used for regression or classification models, including the additional arguments that are used in the TensorFlow's model `fit` function. See here.

Returns *tensorflow.keras.Sequential* – Returns the best TensorFlow model found.

hosa.optimization.hosa.HOSACNN.predict

HOSACNN.**predict**(*x*, ***kwargs*)

Predicts the target values using the input data in the best model found.

Parameters

- **x** (*numpy.ndarray*) – Input data.
- ****kwargs** – Extra arguments that are used in the TensorFlow's model `predict` function. See here.

Returns *numpy.ndarray* – Returns an array containing the estimates that were obtained on the best-fitted model found.

hosa.optimization.hosa.HOSACNN.score

HOSACNN.**score**(*x*, *y*, ***kwargs*)

Computes the performance metrics on the given input data and target values in the best model found.

Parameters

- **x** (*numpy.ndarray*) – Input data.
- **y** (*numpy.ndarray*) – Target values (class labels in classification, real numbers in regression).
- ****kwargs** – Only used for classification, in order to set the value of the parameter `imbalance_correction`.

Returns *tuple* – Returns a tuple containing the performance metric according to the type of model.

hosa.optimization.hosa.HOSARNN

class hosa.optimization.hosa.**HOSARNN**(*x*, *y*, *model*, *n_outputs*, *parameters*, *tr*, *apply_rsv=True*, *validation_size=0.25*, *n_splits=10*)

Bases: *hosa.optimization.hosa.BaseHOSA*

Heuristic Oriented Search Algorithm (HOSA) for RNNs.

This class implments the HOSA for optimizing RNNs. Following a heuristic search, the algorithm finetunes the most relevant models' parameters. Thus, HOSA avoids testing every possible combination, and therefore, an exhaustive search.

Parameters

- **x** (*numpy.ndarray*) – Input data.

- **y** (*numpy.ndarray*) – Target values (class labels in classification, real numbers in regression).
- **model** (*object*) – Class of the object to be optimized. Available options are: *RNNClassification*, *RNNRegression*, *CNNClassification* and *CNNRegression*.
- **n_outputs** (*int*) – Number of class labels in classification, or the number of numerical values to predict in regression.
- **parameters** (*dict*) – Dictionary with parameters names (str) as keys and lists of parameter settings to try as values.
- **tr** (*float*) – Minimum threshold of improvement of the performance metric.
- **apply_rsv** (*bool*) – True if random sub-sampling validation should be used during the optimization procedure.
- **validation_size** (*float*) – Proportion of the dataset to include in the validation split on the random sub-sampling validation. **Ignored if ``apply_rsv = False``**.
- **n_splits** (*int*) – Number of splits used in the random sub-sampling validation. **Ignored if ``apply_rsv = False``**.

Examples

```
import numpy as np
from sklearn.model_selection import train_test_split

from hosa.models.rnn import RNNClassification
from hosa.optimization.hosa import HOSARNN

# 1 - Load the dataset
dataset = np.loadtxt('...', delimiter=',')
x = dataset[:, :-1]
y = dataset[:, -1]
# 2 - Split the data in train and test dataset
x_train, X_test, y_train, y_test = train_test_split(x, y, test_size=.1,
shuffle=False)
# 3 - Set the parameters to optimize
param_grid_rnn = {
        'overlapping_type':          ['central', 'left'],
        'model_type':                ['lstm', 'gru'],
        'overlapping_epochs':        [1],
        'timesteps':                 [1],
        'activation_function_dense': ['relu'],
        'n_units':                   [10, 12],
        'mults':                     [1, 2],
        'optimizer':                 ['adam'],
        'batch_size':                [32],
}
# 4 - Create a HOSA instance and find the best model
clf = HOSARNN(x_train, y_train, RNNClassification, 2, param_grid_rnn, 0.01,
validation_size=.05, apply_rsv=False)
clf.fit(max_n_subs_layers=4, show_progress=True, verbose=0, shuffle=False,
imbalance_correction=True)
score = clf.score(X_test, y_test)
```

Methods

fit(max_n_subs_layers[, show_progress, ...])	Optimize the model following the HOSA approach with all sets of parameters.
get_model()	Get the best model found.
get_params()	Get parameters for the best model found.
grid_search([n_kernels, ...])	Runs a grid search on the remaining moldel's parameters.
predict(x, **kwargs)	Predicts the target values using the input data in the best model found.
score(x, y, **kwargs)	Computes the performance metrics on the given input data and target values in the best model found.

hosa.optimization.hosa.HOSARNN.fit

HOSARNN.**fit**(*max_n_subs_layers*, *show_progress=True*, *imbalance_correction=None*, ***kwargs*)

Optimize the model following the HOSA approach with all sets of parameters.

Parameters

- **max_n_subs_layers** (*int*) – Maximum number of subsequent layers to add to the model.
- **show_progress** (*bool*) – *True* to show a progress bar; *False* otherwise.
- **imbalance_correction** (*None or bool*) – Whether to apply correction to class imbalances. **Only used for classification problems. Ignored for regression.**
- ****kwargs** – Extra arguments explicitly used for regression or classification models, including the additional arguments that are used in the TensorFlow's model `fit` function. See here.

Returns *tuple* – Returns a tuple containing the object of the best model found and respective performance metric and optimized parameters.

hosa.optimization.hosa.HOSARNN.get_model

HOSARNN.**get_model**()

Get the best model found.

Returns *tensorflow.keras.Sequential* – Returns the best TensorFlow model found.

hosa.optimization.hosa.HOSARNN.get_params

HOSARNN.**get_params**()

Get parameters for the best model found.

Returns *dict* – Parameter names mapped to their values.

hosa.optimization.hosa.HOSARNN.grid_search

HOSARNN.**grid_search**(*n_kernels=None*, *n_neurons_dense_layer=None*, *n_units=None*, *n_subs_layers=None*, *imbalance_correction=None*, ***kwargs*)

Runs a grid search on the remaining moldel's parameters.

Parameters

- **n_kernels** (*list or None*) – *i*-th element represents the number of output filters of the convolution layer in the *i*-th GofLayer. **Ignored in the case of optimizing an RNN**.
- **n_neurons_dense_layer** (*int or None*) – Number of neurons of the penultimate dense layer (i.e., before the output layer). **Ignored in the case of optimizing an CNN**.
- **n_units** (*int or None*) – Dimensionality of the output space, i.e., the dimensionality of the hidden state. **Ignored in the case of optimizing an CNN**.
- **n_subs_layers** (*int or None*) – **Ignored in the case of optimizing an CNN**.
- **imbalance_correction** (*None or bool*) – Whether to apply correction to class imbalances. **Only used for classification problems. Ignored for regression.**
- ****kwargs** – Extra arguments explicitly used for regression or classification models, including the additional arguments that are used in the TensorFlow's model `fit` function. See here.

Returns *tensorflow.keras.Sequential* – Returns the best TensorFlow model found.

hosa.optimization.hosa.HOSARNN.predict

HOSARNN.**predict**(*x*, ***kwargs*)

Predicts the target values using the input data in the best model found.

Parameters

- **x** (*numpy.ndarray*) – Input data.
- ****kwargs** – Extra arguments that are used in the TensorFlow's model `predict` function. See here.

Returns *numpy.ndarray* – Returns an array containing the estimates that were obtained on the best-fitted model found.

hosa.optimization.hosa.HOSARNN.score

HOSARNN.**score**(*x*, *y*, ***kwargs*)

Computes the performance metrics on the given input data and target values in the best model found.

Parameters

- **x** (*numpy.ndarray*) – Input data.
- **y** (*numpy.ndarray*) – Target values (class labels in classification, real numbers in regression).
- ****kwargs** – Only used for classification, in order to set the value of the parameter `imbalance_correction`.

Returns *tuple* – Returns a tuple containing the performance metric according to the type of model.

1.1.2 Models

CNN

This module implements classification and regression models using Convolutional Neural Networks (cnn).

cnn_models.BaseCNN(n_outputs, n_kernels[, ...])	Base class for Convolutional Neural Network (CNN) models for classification and regression.
cnn_models.CNNClassification(n_outputs, ...)	Convolutional Neural Network (CNN) classifier.
cnn_models.CNNRegression(n_outputs, n_kernels)	Convolutional Neural Network (CNN) regressor.

hosa.models.cnn.cnn_models.BaseCNN

class hosa.models.cnn.cnn_models.**BaseCNN**(*n_outputs*, *n_kernels*, *n_neurons_dense_layer=50*, *optimizer='adam'*, *cnn_dim=1*, *kernel_size=2*, *pool_size=2*, *strides_convolution=1*, *strides_pooling=2*, *padding='valid'*, *dropout_percentage=0.1*, *activation_function_gol='relu'*, *activation_function_dense='relu'*, *batch_size=1000*, *epochs=50*, *patience=5*, ***kwargs*)

Bases: object

Base class for Convolutional Neural Network (CNN) models for classification and regression.

Each CNN model comprises an input layer, a set of groups of layers (GofLayers [Men20])—where each group is composed of one convolution layer, followed by one pooling layer and a dropout layer—, a dense layer, and an output layer. The output layer is a dense layer with `n_outputs` units, and with the softmax activation function (for classification) or the linear activation function (for regression).

Warning: This class should not be used directly. Use derived classes instead, i.e., *CNNClassification* or *CNNRegression*.

Note: The parameters used in this library were adapted from the exact parameters of the TensorFlow library. Descriptions were thus modified accordingly to our approach. However, refer to the TensorFlow documentation for more details about each of those parameters.

Parameters

- **n_outputs** (*int*) – Number of class labels in classification, or the number of numerical values to predict in regression.
- **n_kernels** (*list*) – *i*-th element represents the number of output filters of the convolution layer in the *i*-th GofLayer.
- **n_neurons_dense_layer** (*int*) – Number of neuron units in the dense layer (i.e., the dense layer after the set of groups of layers).
- **optimizer** (*str*) – Name of the optimizer. See tensorflow.keras.optimizers.
- **cnn_dim** (*int*) – Number of dimensions applicable to *all* the convolution layers of the GofLayers.
- **kernel_size** (*int or tuple*) – Integer or tuple/list of integers, specifying the length (for `cnn_dim` = 1), the height and width (for `cnn_dim` = 2), or the depth, height and width (for

cnn_dim = 3) of the convolution window. It can also be a single integer to specify the same value for all spatial dimensions. This applies to *all* the convolution layers of the GofLayers.

- **pool_size** (*int or tuple*) – Size of the max-pooling window applicable to *all* the max-pooling layers of the GofLayers. For cnn_dim = 1, use a tuple with 1 integer; For cnn_dim = 2, a tuple with 2 integers and for cnn_dim = 3, a tuple with 3 integers. It can also be a single integer to specify the same value for all spatial dimensions.
- **strides_convolution** (*int or tuple*) – Integer or tuple/list of integers, specifying the strides of the convolution. For cnn_dim = 1, use a tuple with 1 integer; For cnn_dim = 2, a tuple with 2 integers and for cnn_dim = 3, a tuple with 3 integers. It can also be a single integer to specify the same value for all spatial dimensions. This applies to *all* the convolution layers of the GofLayers.
- **strides_pooling** (*int or tuple*) – Integer or tuple/list of integers, specifying the strides of the pooling. For cnn_dim = 1, use a tuple with 1 integer; For cnn_dim = 2, a tuple with 2 integers and for cnn_dim = 3, a tuple with 3 integers. It can also be a single integer to specify the same value for all spatial dimensions. This applies to *all* the pooling layers of the GofLayers.
- **padding** (*str*) – Available options are valid or same. valid means no padding. same results in padding evenly to the left/right or up/down of the input such that output has the same height/width dimension as the input. This applies to *all* the pooling layers of the GofLayers.
- **dropout_percentage** (*float*) – Fraction of the input units to drop. This applies to *all* the dropout layers of the GofLayers.
- **activation_function_gol** (*str or None*) – Activation function to use in the convolution layers of the GofLayers. If not specified, no activation is applied (i.e., uses the linear activation function). See tensorflow.keras.activations. This applies to *all* the convolution layers of the GofLayers.
- **activation_function_dense** (*str or None*) – Activation function to use on the dense layer (i.e., the dense layer after the set of groups of layers). If not specified, no activation is applied (i.e., uses the linear activation function). See tensorflow.keras.activations.
- **batch_size** (*int or None*) – Number of samples per batch of computation. If None, batch_size will default to 32.
- **epochs** (*int*) – Maximum number of epochs to train the model.
- **patience** (*int*) – Number of epochs with no improvement after which training will be stopped.
- ****kwargs** – *Ignored*. Extra arguments that are used for compatibility's sake.

Methods

add_gol(n_kernel, cnn_dim)	Adds a GofLayer to the estimator.
aux_fit(x, y, callback, validation_size[, ...])	Auxiliar function for classification and regression models compatibility.
compile()	Compiles the model for training.
fit(x, y, **kwargs)	Fits the model to data matrix x and target(s) y.
predict(x, **kwargs)	Predicts the target values using the input data in the trained model.
prepare(x, y)	Prepares the model by adding the layers to the estimator: input layer, GofLayers, flatten and dense layers.

continues on next page

Table 6 – continued from previous page

score(x, y, **kwargs)	Computes the performance metric(s) (e.g., accuracy for classification) on the given input data and target values.

hosa.models.cnn.cnn_models.BaseCNN.add_gol

BaseCNN.**add_gol**(*n_kernel*, *cnn_dim*)

Adds a GofLayer to the estimator.

Parameters

- **n_kernel** (*int*) – Number of output filters.
- **cnn_dim** (*int*) – Number of dimensions.

Raises ValueError – If cnn_dim is not valid.

hosa.models.cnn.cnn_models.BaseCNN.aux_fit

BaseCNN.**aux_fit**(*x*, *y*, *callback*, *validation_size*, *atol=0.0001*, *rtol=0.001*, *class_weights=None*, *imbalance_correction=None*, *shuffle=True*, ***kwargs*)

Auxiliar function for classification and regression models compatibility.

> **Warning:** This function is not meant to be called by itself. It is just an auxiliary function called by the child classes' fit function.

Parameters

- **x** (*numpy.ndarray*) – Input data.
- **y** (*numpy.ndarray*) – Target values (class labels in classification, real numbers in regression).
- **callback** (*object*) – Early stopping callback for halting the model's training.
- **validation_size** (*float or int*) – Proportion of the training dataset that will be used the validation split.
- **atol** (*float*) – Absolute tolerance used for early stopping based on the performance metric.
- **rtol** (*float*) – Relative tolerance used for early stopping based on the performance metric.
- **class_weights** (*None or dict*) – Dictionary mapping class indices (integers) to a weight (float) value, used for weighting the loss function (during training only). **Only used for classification problems. Ignored for regression.**
- **imbalance_correction** (*None or bool*) – Whether to apply correction to class imbalances. **Only used for classification problems. Ignored for regression.**
- **shuffle** (*bool*) – Whether to shuffle the data before splitting.
- ****kwargs** – Extra arguments used in the TensorFlow's model fit function. See here.

hosa.models.cnn.cnn_models.BaseCNN.compile

abstract BaseCNN.**compile**()
Compiles the model for training.

hosa.models.cnn.cnn_models.BaseCNN.fit

abstract BaseCNN.**fit**(*x*, *y*, ***kwargs*)
Fits the model to data matrix x and target(s) y.

Parameters

- **x** (*numpy.ndarray*) – Input data.
- **y** (*numpy.ndarray*) – Target values (class labels in classification, real numbers in regression).
- ****kwargs** – Extra arguments explicitly used for regression or classification models.

hosa.models.cnn.cnn_models.BaseCNN.predict

abstract BaseCNN.**predict**(*x*, ***kwargs*)
Predicts the target values using the input data in the trained model.

Parameters

- **x** (*numpy.ndarray*) – Input data.
- ****kwargs** – Extra arguments that are used in the TensorFlow's model `predict` function. See here.

hosa.models.cnn.cnn_models.BaseCNN.prepare

BaseCNN.**prepare**(*x*, *y*)
Prepares the model by adding the layers to the estimator: input layer, GofLayers, flatten and dense layers.

Parameters

- **x** (*numpy.ndarray*) – Input data.
- **y** (*numpy.ndarray*) – Target values (class labels in classification, real numbers in
- **regression).** –

hosa.models.cnn.cnn_models.BaseCNN.score

abstract BaseCNN.**score**(*x*, *y*, ***kwargs*)
Computes the performance metric(s) (e.g., accuracy for classification) on the given input data and target values.

Parameters

- **x** (*numpy.ndarray*) – Input data.
- **y** (*numpy.ndarray*) – Target values (class labels in classification, real numbers in regression).

- ****kwargs** – Extra arguments that are explicitly used for regression or classification
- **models.** –

hosa.models.cnn.cnn_models.CNNClassification

class hosa.models.cnn.cnn_models.**CNNClassification**(*n_outputs*, *n_kernels*, *n_neurons_dense_layer=50*, *optimizer='adam'*, *metrics=None*, *cnn_dim=1*, *kernel_size=2*, *pool_size=2*, *strides_convolution=1*, *strides_pooling=2*, *padding='valid'*, *dropout_percentage=0.1*, *activation_function_gol='relu'*, *activation_function_dense='relu'*, *batch_size=1000*, *epochs=50*, *patience=5*, ***kwargs*)

Bases: *hosa.models.cnn.cnn_models.BaseCNN*

Convolutional Neural Network (CNN) classifier.

The model comprises an input layer, a set of groups of layers (GofLayers [Men20])—where each group is composed of one convolution layer, followed by one pooling layer and a dropout layer—, a dense layer, and an output layer. The output layer is a dense layer with `n_outputs` units, with the softmax activation function.

Note: The parameters used in this library were adapted from the exact parameters of the TensorFlow library. Descriptions were thus modified accordingly to our approach. However, refer to the TensorFlow documentation for more details about each of those parameters.

Parameters

- **n_outputs** (*int*) – Number of class labels to predict.
- **n_kernels** (*list*) – *i*-th element represents the number of output filters of the convolution layer in the *i*-th GofLayer.
- **n_neurons_dense_layer** (*int*) – Number of neuron units in the dense layer (i.e., the dense layer after the set of groups of layers).
- **optimizer** (*str*) – Name of the optimizer. See tensorflow.keras.optimizers.
- **metrics** (*list*) – List of metrics to be evaluated by the model during training and testing. Each item of the list can be a string (name of a TensorFlow's built-in function), function, or a tf.keras.metrics.Metric instance. If `None`, `metrics` will default to `['accuracy']`.
- **cnn_dim** (*int*) – Number of dimensions applicable to *all* the convolution layers of the GofLayers.
- **kernel_size** (*int or tuple*) – Integer or tuple/list of integers, specifying the length (for `cnn_dim` = 1), the height and width (for `cnn_dim` = 2), or the depth, height and width (for `cnn_dim` = 3) of the convolution window. It can also be a single integer to specify the same value for all spatial dimensions. This applies to *all* the convolution layers of the GofLayers.
- **pool_size** (*int or tuple*) – Size of the max-pooling window applicable to *all* the max-pooling layers of the GofLayers. For `cnn_dim` = 1, use a tuple with 1 integer; For `cnn_dim` = 2, a tuple with 2 integers and for `cnn_dim` = 3, a tuple with 3 integers. It can also be a single integer to specify the same value for all spatial dimensions.

- **strides_convolution** (*int or tuple*) – Integer or tuple/list of integers, specifying the strides of the convolution. For `cnn_dim` = 1, use a tuple with 1 integer; For `cnn_dim` = 2, a tuple with 2 integers and for `cnn_dim` = 3, a tuple with 3 integers. It can also be a single integer to specify the same value for all spatial dimensions. This applies to *all* the convolution layers of the GofLayers.
- **strides_pooling** (*int or tuple*) – Integer or tuple/list of integers, specifying the strides of the pooling. For `cnn_dim` = 1, use a tuple with 1 integer; For `cnn_dim` = 2, a tuple with 2 integers and for `cnn_dim` = 3, a tuple with 3 integers. It can also be a single integer to specify the same value for all spatial dimensions. This applies to *all* the pooling layers of the GofLayers.
- **padding** (*str*) – Available options are `valid` or `same`. `valid` means no padding. `same` results in padding evenly to the left/right or up/down of the input such that output has the same height/width dimension as the input. This applies to *all* the pooling layers of the GofLayers.
- **dropout_percentage** (*float*) – Fraction of the input units to drop. This applies to *all* the dropout layers of the GofLayers.
- **activation_function_gol** (*str or None*) – Activation function to use in the convolution layers of the GofLayers. If not specified, no activation is applied (i.e., uses the linear activation function). See tensorflow.keras.activations. This applies to *all* the convolution layers of the GofLayers.
- **activation_function_dense** (*str or None*) – Activation function to use on the dense layer (i.e., the dense layer after the set of groups of layers). If not specified, no activation is applied (i.e., uses the linear activation function). See tensorflow.keras.activations.
- **batch_size** (*int or None*) – Number of samples per batch of computation. If `None`, `batch_size` will default to 32.
- **epochs** (*int*) – Maximum number of epochs to train the model.
- **patience** (*int*) – Number of epochs with no improvement after which training will be stopped.
- ****kwargs** – *Ignored*. Extra arguments that are used for compatibility's sake.

Examples

```
from tensorflow import keras

from hosa.models.cnn import CNNClassification

# 1 - Load and split the data
fashion_mnist = keras.datasets.fashion_mnist
(x_train, y_train), (X_test, y_test) = fashion_mnist.load_data()
# 2 - Normalize the images, and reshape the images for cnn input
x_train = x_train / 255.0
X_test = X_test / 255.0
x_train = x_train.reshape((-1, 28 * 28))
X_test = X_test.reshape((-1, 28 * 28))
# 3 - Create and fit the model
clf = CNNClassification(10, 10, [4, 3])
clf.prepare(x_train, y_train)
clf.compile()
clf.fit(x_train, y_train)
```

(continues on next page)

(continued from previous page)

```
# 4 - Calculate predictions
clf.predict(X_test)
# 5 - Compute the score
score = clf.score(X_test, y_test)
```

Methods

add_gol(n_kernel, cnn_dim)	Adds a GofLayer to the estimator.
aux_fit(x, y, callback, validation_size[, ...])	Auxiliar function for classification and regression models compatibility.
compile()	Compiles the model for training.
fit(x, y[, validation_size, shuffle, atol, ...])	Fits the model to data matrix x and target(s) y.
predict(x, **kwargs)	Predicts the target values using the input data in the trained model.
prepare(x, y)	Prepares the model by adding the layers to the estimator: input layer, GofLayers, flatten and dense layers.
score(x, y[, imbalance_correction])	Computes the performance metrics on the given input data and target values.

hosa.models.cnn.cnn_models.CNNClassification.add_gol

CNNClassification.**add_gol**(*n_kernel*, *cnn_dim*)

Adds a GofLayer to the estimator.

Parameters

- **n_kernel** (*int*) – Number of output filters.
- **cnn_dim** (*int*) – Number of dimensions.

Raises ValueError – If cnn_dim is not valid.

hosa.models.cnn.cnn_models.CNNClassification.aux_fit

CNNClassification.**aux_fit**(*x*, *y*, *callback*, *validation_size*, *atol=0.0001*, *rtol=0.001*, *class_weights=None*, *imbalance_correction=None*, *shuffle=True*, ***kwargs*)

Auxiliar function for classification and regression models compatibility.

> **Warning:** This function is not meant to be called by itself. It is just an auxiliary function called by the child classes' `fit` function.

Parameters

- **x** (*numpy.ndarray*) – Input data.
- **y** (*numpy.ndarray*) – Target values (class labels in classification, real numbers in regression).
- **callback** (*object*) – Early stopping callback for halting the model's training.

- **validation_size** (*float or int*) – Proportion of the training dataset that will be used the validation split.
- **atol** (*float*) – Absolute tolerance used for early stopping based on the performance metric.
- **rtol** (*float*) – Relative tolerance used for early stopping based on the performance metric.
- **class_weights** (*None or dict*) – Dictionary mapping class indices (integers) to a weight (float) value, used for weighting the loss function (during training only). **Only used for classification problems. Ignored for regression.**
- **imbalance_correction** (*None or bool*) – Whether to apply correction to class imbalances. **Only used for classification problems. Ignored for regression.**
- **shuffle** (*bool*) – Whether to shuffle the data before splitting.
- ****kwargs** – Extra arguments used in the TensorFlow's model `fit` function. See here.

hosa.models.cnn.cnn_models.CNNClassification.compile

CNNClassification.**compile**()

Compiles the model for training.

Returns *tensorflow.keras.Sequential* – Returns an untrained but compiled TensorFlow model.

hosa.models.cnn.cnn_models.CNNClassification.fit

CNNClassification.**fit**(*x*, *y*, *validation_size=0.33*, *shuffle=True*, *atol=0.0001*, *rtol=0.001*, *class_weights=None*, *imbalance_correction=False*, ***kwargs*)

Fits the model to data matrix x and target(s) y.

Parameters

- **x** (*numpy.ndarray*) – Input data.
- **y** (*numpy.ndarray*) – Target values (i.e., class labels).
- **validation_size** (*float or int*) – Proportion of the train dataset to include in the validation split.
- **shuffle** (*bool*) – Whether to shuffle the data before splitting.
- **atol** (*float*) – Absolute tolerance used for early stopping based on the performance metric.
- **rtol** (*float*) – Relative tolerance used for early stopping based on the performance metric.
- **class_weights** (*None or dict*) – Dictionary mapping class indices (integers) to a weight (float) value, used for weighting the loss function (during training only).
- **imbalance_correction** (*bool*) – *True* if correction for imbalance should be applied to the metrics; *False* otherwise.
- ****kwargs** – Extra arguments that are used in the TensorFlow's model `fit` function. See here.

Returns *tensorflow.keras.Sequential* – Returns a trained TensorFlow model.

hosa.models.cnn.cnn_models.CNNClassification.predict

CNNClassification.**predict**(*x*, ***kwargs*)

Predicts the target values using the input data in the trained model.

Parameters

- **x** (*numpy.ndarray*) – Input data.
- ****kwargs** – Extra arguments that are used in the TensorFlow's model `predict` function. See here.

Returns *tuple* – Returns a tuple containing the probability estimates and predicted classes.

hosa.models.cnn.cnn_models.CNNClassification.prepare

CNNClassification.**prepare**(*x*, *y*)

Prepares the model by adding the layers to the estimator: input layer, GofLayers, flatten and dense layers.

Parameters

- **x** (*numpy.ndarray*) – Input data.
- **y** (*numpy.ndarray*) – Target values (i.e., class labels).

hosa.models.cnn.cnn_models.CNNClassification.score

CNNClassification.**score**(*x*, *y*, *imbalance_correction=False*, ***kwargs*)

Computes the performance metrics on the given input data and target values.

Parameters

- **x** (*numpy.ndarray*) – Input data.
- **y** (*numpy.ndarray*) – Target values (i.e., class labels).
- **imbalance_correction** (*bool*) – *True* if correction for imbalance should be applied to the metrics; *False* otherwise.
- ****kwargs** – *Ignored*. Only included here for compatibility's sake.

Returns

tuple –

Returns a tuple containing the area under the ROC curve (AUC), accuracy, sensitivity, and sensitivity.

Note: This function can be used for both binary and multiclass classification.

---

hosa.models.cnn.cnn_models.CNNRegression

class hosa.models.cnn.cnn_models.**CNNRegression**(*n_outputs*, *n_kernels*, *n_neurons_dense_layer=50*, *optimizer='adam'*, *metrics=None*, *cnn_dim=1*, *kernel_size=2*, *pool_size=2*, *strides_convolution=1*, *strides_pooling=2*, *padding='valid'*, *dropout_percentage=0.1*, *activation_function_gol='relu'*, *activation_function_dense='relu'*, *batch_size=1000*, *epochs=50*, *patience=5*, ***kwargs*)

Bases: *hosa.models.cnn.cnn_models.BaseCNN*

Convolutional Neural Network (CNN) regressor.

The model comprises an input layer, a set of groups of layers (GofLayers [Men20])—where each group is composed of one convolution layer, followed by one pooling layer and a dropout layer—, a dense layer, and an output layer. The output layer is a dense layer with `n_outputs` units, with the linear activation function.

Note: The parameters used in this library were adapted from the exact parameters of the TensorFlow library. Descriptions were thus modified accordingly to our approach. However, refer to the TensorFlow documentation for more details about each of those parameters.

Parameters

- **n_outputs** (*int*) – Number of numerical values to predict in regression.
- **n_kernels** (*list*) – *i*-th element represents the number of output filters of the convolution layer in the *i*-th GofLayer.
- **n_neurons_dense_layer** (*int*) – Number of neuron units in the dense layer (i.e., the dense layer after the set of groups of layers).
- **optimizer** (*str*) – Name of the optimizer. See tensorflow.keras.optimizers.
- **metrics** (*list*) – List of metrics to be evaluated by the model during training and testing. Each item of the list can be a string (name of a TensorFlow's built-in function), function, or a tf.keras.metrics.Metric instance. If `None`, `metrics` will default to `['mean_squared_error']`.
- **cnn_dim** (*int*) – Number of dimensions applicable to *all* the convolution layers of the GofLayers.
- **kernel_size** (*int or tuple*) – Integer or tuple/list of integers, specifying the length (for `cnn_dim` = 1), the height and width (for `cnn_dim` = 2), or the depth, height and width (for `cnn_dim` = 3) of the convolution window. It can also be a single integer to specify the same value for all spatial dimensions. This applies to *all* the convolution layers of the GofLayers.
- **pool_size** (*int or tuple*) – Size of the max-pooling window applicable to *all* the max-pooling layers of the GofLayers. For `cnn_dim` = 1, use a tuple with 1 integer; For `cnn_dim` = 2, a tuple with 2 integers and for `cnn_dim` = 3, a tuple with 3 integers. It can also be a single integer to specify the same value for all spatial dimensions.
- **strides_convolution** (*int or tuple*) – Integer or tuple/list of integers, specifying the strides of the convolution. For `cnn_dim` = 1, use a tuple with 1 integer; For `cnn_dim` = 2, a tuple with 2 integers and for `cnn_dim` = 3, a tuple with 3 integers. It can also be a single integer to specify the same value for all spatial dimensions. This applies to *all* the convolution layers of the GofLayers.

- **strides_pooling** (*int or tuple*) – Integer or tuple/list of integers, specifying the strides of the pooling. For `cnn_dim` = 1, use a tuple with 1 integer; For `cnn_dim` = 2, a tuple with 2 integers and for `cnn_dim` = 3, a tuple with 3 integers. It can also be a single integer to specify the same value for all spatial dimensions. This applies to *all* the pooling layers of the GofLayers.
- **padding** (*str*) – Available options are `valid` or `same`. `valid` means no padding. `same` results in padding evenly to the left/right or up/down of the input such that output has the same height/width dimension as the input. This applies to *all* the pooling layers of the GofLayers.
- **dropout_percentage** (*float*) – Fraction of the input units to drop. This applies to *all* the dropout layers of the GofLayers.
- **activation_function_gol** (*str or None*) – Activation function to use in the convolution layers of the GofLayers. If not specified, no activation is applied (i.e., uses the linear activation function). See tensorflow.keras.activations. This applies to *all* the convolution layers of the GofLayers.
- **activation_function_dense** (*str or None*) – Activation function to use on the dense layer (i.e., the dense layer after the set of groups of layers). If not specified, no activation is applied (i.e., uses the linear activation function). See tensorflow.keras.activations.
- **batch_size** (*int or None*) – Number of samples per batch of computation. If `None`, `batch_size` will default to 32.
- **epochs** (*int*) – Maximum number of epochs to train the model.
- **patience** (*int*) – Number of epochs with no improvement after which training will be stopped.
- ****kwargs** – *Ignored*. Extra arguments that are used for compatibility's sake.

Examples

```
import pandas as pd

from hosa.models.cnn import CNNRegression
from hosa.aux import create_overlapping

# 1 - Download, load, and split the data
dataset = pd.read_csv(
'https://raw.githubusercontent.com/jbrownlee/Datasets/master/airline-passengers
.csv', header=0, index_col=0)
x = dataset.Passengers.to_numpy().reshape((len(dataset), 1))
y = dataset.Passengers.to_numpy()
x_train, y_train = x[:100], y[:100]
X_test, y_test = x[100:], y[100:]
# 2 - Prepare the data for cnn input
x_train, y_train = create_overlapping(x_train, y_train, CNNRegression, 'central', 4)
X_test, y_test = create_overlapping(X_test, y_test, CNNRegression, 'central', 4)
# 3 - Create and fit the model
clf = CNNRegression(1, 50, [5], epochs=500, patience=500)
clf.prepare(x_train, y_train)
clf.compile()
clf.fit(x_train, y_train)
# 4 - Calculate predictions
```

(continues on next page)

(continued from previous page)

```
clf.predict(X_test)
# 5 - Compute the score
score = clf.score(X_test, y_test)
```

Methods

add_gol(n_kernel, cnn_dim)	Adds a GofLayer to the estimator.
aux_fit(x, y, callback, validation_size[, ...])	Auxiliar function for classification and regression models compatibility.
compile()	Compiles the model for training.
fit(x, y[, validation_size, shuffle, atol, rtol])	Fits the model to data matrix x and target(s) y.
predict(x, **kwargs)	Predicts the target values using the input data in the trained model.
prepare(x, y)	Prepares the model by adding the layers to the estimator: input layer, GofLayers, flatten and dense layers.
score(x, y, **kwargs)	Computes the performance metrics on the given input data and target values.

hosa.models.cnn.cnn_models.CNNRegression.add_gol

CNNRegression.**add_gol**(*n_kernel*, *cnn_dim*)

Adds a GofLayer to the estimator.

Parameters

- **n_kernel** (*int*) – Number of output filters.
- **cnn_dim** (*int*) – Number of dimensions.

Raises ValueError – If cnn_dim is not valid.

hosa.models.cnn.cnn_models.CNNRegression.aux_fit

CNNRegression.**aux_fit**(*x*, *y*, *callback*, *validation_size*, *atol=0.0001*, *rtol=0.001*, *class_weights=None*, *imbalance_correction=None*, *shuffle=True*, ***kwargs*)

Auxiliar function for classification and regression models compatibility.

> **Warning:** This function is not meant to be called by itself. It is just an auxiliary function called by the child classes' fit function.

Parameters

- **x** (*numpy.ndarray*) – Input data.
- **y** (*numpy.ndarray*) – Target values (class labels in classification, real numbers in regression).
- **callback** (*object*) – Early stopping callback for halting the model's training.
- **validation_size** (*float or int*) – Proportion of the training dataset that will be used the validation split.

- **atol** (*float*) – Absolute tolerance used for early stopping based on the performance metric.
- **rtol** (*float*) – Relative tolerance used for early stopping based on the performance metric.
- **class_weights** (*None or dict*) – Dictionary mapping class indices (integers) to a weight (float) value, used for weighting the loss function (during training only). **Only used for classification problems. Ignored for regression.**
- **imbalance_correction** (*None or bool*) – Whether to apply correction to class imbalances. **Only used for classification problems. Ignored for regression.**
- **shuffle** (*bool*) – Whether to shuffle the data before splitting.
- ****kwargs** – Extra arguments used in the TensorFlow's model `fit` function. See here.

hosa.models.cnn.cnn_models.CNNRegression.compile

CNNRegression.**compile**()

Compiles the model for training.

Returns *tensorflow.keras.Sequential* – Returns an untrained but compiled TensorFlow model.

hosa.models.cnn.cnn_models.CNNRegression.fit

CNNRegression.**fit**(*x*, *y*, *validation_size=0.33*, *shuffle=True*, *atol=0.0001*, *rtol=0.001*, ***kwargs*)

Fits the model to data matrix x and target(s) y.

Parameters

- **x** (*numpy.ndarray*) – Input data.
- **y** (*numpy.ndarray*) – Target values (i.e., real numbers).
- **validation_size** (*float or int*) – Proportion of the train dataset to include in the validation split.
- **shuffle** (*bool*) – Whether to shuffle the data before splitting.
- **atol** (*float*) – Absolute tolerance used for early stopping based on the performance metric.
- **rtol** (*float*) – Relative tolerance used for early stopping based on the performance metric.
- ****kwargs** – Extra arguments that are used in the TensorFlow's model `fit` function. See here.

Returns *tensorflow.keras.Sequential* – Returns a trained TensorFlow model.

hosa.models.cnn.cnn_models.CNNRegression.predict

CNNRegression.**predict**(*x*, ***kwargs*)

Predicts the target values using the input data in the trained model.

Parameters

- **x** (*numpy.ndarray*) – Input data.
- ****kwargs** – Extra arguments that are used in the TensorFlow's model `predict` function. See here.

Returns *numpy.ndarray* – Returns an array containing the estimates.

hosa.models.cnn.cnn_models.CNNRegression.prepare

CNNRegression.**prepare**(*x*, *y*)

Prepares the model by adding the layers to the estimator: input layer, GofLayers, flatten and dense layers.

Parameters

- **x** (*numpy.ndarray*) – Input data.
- **y** (*numpy.ndarray*) – Target values (i.e., real numbers).

hosa.models.cnn.cnn_models.CNNRegression.score

CNNRegression.**score**(*x*, *y*, ***kwargs*)

Computes the performance metrics on the given input data and target values.

Parameters

- **x** (*numpy.ndarray*) – Input data.
- **y** (*numpy.ndarray*) – Target values (i.e., real numbers).
- ****kwargs** – *Ignored*. Only included here for compatibility's sake.

Returns

tuple –

Returns a tuple containing the mean squared error (MSE) and coefficient of determination (R^2).

RNN

This module implements classification and regression models using Recurrent Neural Networks (RNNs).

rnn_models.BaseRNN(n_outputs, ...[, ...])	Base class for Recurrent Neural Network (RNN) models for classification and regression.
rnn_models.RNNClassification(n_outputs, ...)	Recurrent Neural Network (RNN) model classifier.
rnn_models.RNNRegression(n_outputs, ...[, ...])	Recurrent Neural Network (RNN) model regressor.

hosa.models.rnn.rnn_models.BaseRNN

class hosa.models.rnn.rnn_models.**BaseRNN**(*n_outputs*, *n_neurons_dense_layer*, *n_units*, *n_subs_layers*, *is_bidirectional=False*, *model_type='lstm'*, *optimizer='adam'*, *dropout_percentage=0.1*, *activation_function_dense='relu'*, *kernel_initializer='normal'*, *batch_size=1000*, *epochs=50*, *patience=5*, ***kwargs*)

Bases: `object`

Base class for Recurrent Neural Network (RNN) models for classification and regression.

Each RNN model comprises an input layer (an RNN or a bidirectional RNN cell), `n_subs_layers` subsequent layers (similar to the input cell), a dropout layer, a dense layer, and an output layer. The output layer is a dense layer with `n_outputs` units, with the linear activation function.

Warning: This class should not be used directly. Use derived classes instead, i.e., *RNNClassification* or *RNNRegression*.

Note: The parameters used in this library were adapted from the exact parameters of the TensorFlow library. Descriptions were thus modified accordingly to our approach. However, refer to the TensorFlow documentation for more details about each of those parameters.

Parameters

- **n_outputs** (*int*) – Number of class labels in classification, or the number of numerical values to predict in regression.
- **n_neurons_dense_layer** (*int*) – Number of neurons units of the penultimate dense layer (i.e., before the output layer).
- **n_units** (*int*) – Dimensionality of the output space, i.e., the dimensionality of the hidden state.
- **n_subs_layers** (*int*) – Number of subsequent recurrent layers beteween the input and output layers.
- **is_bidirectional** (*bool*) – If `true`, then bidirectional layers will be used to build the RNN model.
- **model_type** (*str*) – Type of RNN model to be used. Available options are `lstm`, for a Long Short-Term Memory model, or `gru`, for a Gated Recurrent Unit model.
- **optimizer** (*str*) – Name of the optimizer. See tensorflow.keras.optimizers.
- **dropout_percentage** (*float*) – Fraction of the input units to drop.
- **activation_function_dense** (*str*) – Activation function to use on the penultimate dense layer. If not specified, no activation is applied (i.e., uses the linear activation function). See tensorflow.keras.activations.
- **kernel_initializer** (*str*) – Initializer for the kernel weights matrix, used for the linear transformation of the inputs.
- **batch_size** (*int or None*) – Number of samples per batch of computation. If `None`, `batch_size` will default to 32.
- **epochs** (*int*) – Maximum number of epochs to train the model.

- **patience** (*int*) – Number of epochs with no improvement after which training will be stopped.
- ****kwargs** – *Ignored*. Extra arguments that are used for compatibility's sake.

Methods

aux_fit(x, y, callback, validation_size[, ...])	Auxiliar function for classification and regression models compatibility.
compile()	Compiles the model for training.
fit(x, y, **kwargs)	Fits the model to data matrix x and target(s) y.
predict(x, **kwargs)	Predicts the target values using the input data in the trained model.
prepare(x, y)	Prepares the model by adding the layers to the estimator: input layer, `n_subs_layers` subsequent layers, a dropout layer, a dense layer, and an output layer.
score(x, y, **kwargs)	Computes the performance metric(s) (e.g., accuracy for classification) on the given input data and target values.

hosa.models.rnn.rnn_models.BaseRNN.aux_fit

BaseRNN.**aux_fit**(*x*, *y*, *callback*, *validation_size*, *rtol=0.001*, *atol=0.0001*, *class_weights=None*, *imbalance_correction=None*, *shuffle=True*, ***kwargs*)

Auxiliar function for classification and regression models compatibility.

> **Warning:** This function is not meant to be called by itself. It is just an auxiliary function called by the child classes' `fit` function.

Parameters

- **x** (*numpy.ndarray*) – Input data.
- **y** (*numpy.ndarray*) – Target values (class labels in classification, real numbers in regression).
- **callback** (*object*) – Early stopping callback for halting the model's training.
- **validation_size** (*float or int*) – Proportion of the training dataset that will be used the validation split.
- **atol** (*float*) – Absolute tolerance used for early stopping based on the performance metric.
- **rtol** (*float*) – Relative tolerance used for early stopping based on the performance metric.
- **class_weights** (*None or dict*) – Dictionary mapping class indices (integers) to a weight (float) value, used for weighting the loss function (during training only). **Only used for classification problems. Ignored for regression.**
- **imbalance_correction** (*None or bool*) – Whether to apply correction to class imbalances. **Only used for classification problems. Ignored for regression.**
- **shuffle** (*bool*) – Whether to shuffle the data before splitting.

- ****kwargs** – Extra arguments used in the TensorFlow's model `fit` function. See here.

hosa.models.rnn.rnn_models.BaseRNN.compile

abstract BaseRNN.**compile**()
Compiles the model for training.

hosa.models.rnn.rnn_models.BaseRNN.fit

abstract BaseRNN.**fit**(*x*, *y*, ***kwargs*)
Fits the model to data matrix x and target(s) y.

Parameters

- **x** (*numpy.ndarray*) – Input data.
- **y** (*numpy.ndarray*) – Target values (class labels in classification, real numbers in regression).
- ****kwargs** – Extra arguments explicitly used for regression or classification models.

hosa.models.rnn.rnn_models.BaseRNN.predict

abstract BaseRNN.**predict**(*x*, ***kwargs*)
Predicts the target values using the input data in the trained model.

Parameters

- **x** (*numpy.ndarray*) – Input data.
- ****kwargs** – Extra arguments that are used in the TensorFlow's model `predict` function. See here.

hosa.models.rnn.rnn_models.BaseRNN.prepare

BaseRNN.**prepare**(*x*, *y*)
Prepares the model by adding the layers to the estimator: input layer, `n_subs_layers` subsequent layers, a dropout layer, a dense layer, and an output layer.

Parameters

- **x** (*numpy.ndarray*) – Input data.
- **y** (*numpy.ndarray*) – Target values (class labels in classification, real numbers in regression).

hosa.models.rnn.rnn_models.BaseRNN.score

abstract BaseRNN.**score**(*x*, *y*, ***kwargs*)

Computes the performance metric(s) (e.g., accuracy for classification) on the given input data and target values.

Parameters

- **x** (*numpy.ndarray*) – Input data.
- **y** (*numpy.ndarray*) – Target values (class labels in classification, real numbers in regression).
- ****kwargs** – Extra arguments that are explicitly used for regression or classification models.

hosa.models.rnn.rnn_models.RNNClassification

class hosa.models.rnn.rnn_models.**RNNClassification**(*n_outputs*, *n_neurons_dense_layer*, *n_units*, *n_subs_layers*, *is_bidirectional=False*, *model_type='lstm'*, *optimizer='adam'*, *dropout_percentage=0.1*, *metrics=None*, *activation_function_dense='relu'*, *kernel_initializer='normal'*, *batch_size=1000*, *epochs=50*, *patience=5*, ***kwargs*)

Bases: *hosa.models.rnn.rnn_models.BaseRNN*

Recurrent Neural Network (RNN) model classifier.

The model comprises an input layer (an RNN or a bidirectional RNN cell), `n_subs_layers` subsequent layers (similar to the input cell), a dropout layer, a dense layer, and an output layer.

Parameters

- **n_outputs** (*int*) – Number of class labels to predict.
- **n_neurons_dense_layer** (*int*) – Number of neurons units of the penultimate dense layer (i.e., before the output layer).
- **n_units** (*int*) – Dimensionality of the output space, i.e., the dimensionality of the hidden state.
- **n_subs_layers** (*int*) – Number of subsequent layers beteween the input and output layers.
- **is_bidirectional** (*bool*) – If `true`, then bidirectional layers will be used to build the RNN model.
- **model_type** (*str*) – Type of RNN model to be used. Available options are `lstm`, for a Long Short-Term Memory model, or `gru`, for a Gated Recurrent Unit model.
- **optimizer** (*str*) – Name of the optimizer. See tensorflow.keras.optimizers.
- **dropout_percentage** (*float*) – Fraction of the input units to drop.
- **metrics** (*list*) – List of metrics to be evaluated by the model during training and testing. Each item of the list can be a string (name of a TensorFlow's built-in function), function, or a tf.keras.metrics.Metric instance. If `None`, `metrics` will default to `['accuracy']`.
- **activation_function_dense** (*str*) – Activation function to use on the penultimate dense layer. If not specified, no activation is applied (i.e., uses the linear activation function). See tensorflow.keras.activations.

- **kernel_initializer** (*str*) – Initializer for the kernel weights matrix, used for the linear transformation of the inputs.
- **batch_size** (*int or None*) – Number of samples per batch of computation. If None, batch_size will default to 32.
- **epochs** (*int*) – Maximum number of epochs to train the model.
- **patience** (*int*) – Number of epochs with no improvement after which training will be stopped.
- ****kwargs** – *Ignored*. Extra arguments that are used for compatibility's sake.

Examples

```
from keras.datasets import imdb
from keras_preprocessing.sequence import pad_sequences
from tensorflow import keras

from hosa.models.rnn import RNNClassification
from hosa.aux import create_overlapping

# 1 - Load and split the data
max_sequence_length = 50
fashion_mnist = keras.datasets.fashion_mnist
(x_train, y_train), (X_test, y_test) = imdb.load_data(num_words=50)
# 2 - Prepare the data for rnn input
x_train = pad_sequences(x_train, maxlen=max_sequence_length, value=0.0)
X_test = pad_sequences(X_test, maxlen=max_sequence_length, value=0.0)
x_train, y_train = create_overlapping(x_train, y_train, RNNClassification,
'central', 3, stride=1, timesteps=2)
X_test, y_test = create_overlapping(X_test, y_test, RNNClassification, 'central',
3, stride=1, timesteps=2)
# 3 - Create and fit the model
clf = RNNClassification(2, 10, is_bidirectional=True)
clf.prepare(x_train, y_train)
clf.compile()
clf.fit(x_train, y_train)
# 4 - Calculate predictions
clf.predict(X_test)
# 5 - Compute the score
score = clf.score(X_test, y_test)
```

Methods

aux_fit(x, y, callback, validation_size[, ...])	Auxiliar function for classification and regression models compatibility.
compile()	Compiles the model for training.
fit(x, y[, validation_size, shuffle, rtol, ...])	Fits the model to data matrix x and target(s) y.
predict(x, **kwargs)	Predicts the target values using the input data in the trained model.

continues on next page

Table 11 – continued from previous page

prepare(x, y)	Prepares the model by adding the layers to the estimator: input layer, `n_subs_layers` subsequent layers, a dropout layer, a dense layer, and an output layer.
score(x, y[, imbalance_correction])	Computes the performance metrics on the given input data and target values.

hosa.models.rnn.rnn_models.RNNClassification.aux_fit

RNNClassification.**aux_fit**(*x*, *y*, *callback*, *validation_size*, *rtol=0.001*, *atol=0.0001*, *class_weights=None*, *imbalance_correction=None*, *shuffle=True*, ***kwargs*)

Auxiliar function for classification and regression models compatibility.

> **Warning:** This function is not meant to be called by itself. It is just an auxiliary function called by the child classes' `fit` function.

Parameters

- **x** (*numpy.ndarray*) – Input data.
- **y** (*numpy.ndarray*) – Target values (class labels in classification, real numbers in regression).
- **callback** (*object*) – Early stopping callback for halting the model's training.
- **validation_size** (*float or int*) – Proportion of the training dataset that will be used the validation split.
- **atol** (*float*) – Absolute tolerance used for early stopping based on the performance metric.
- **rtol** (*float*) – Relative tolerance used for early stopping based on the performance metric.
- **class_weights** (*None or dict*) – Dictionary mapping class indices (integers) to a weight (float) value, used for weighting the loss function (during training only). **Only used for classification problems. Ignored for regression.**
- **imbalance_correction** (*None or bool*) – Whether to apply correction to class imbalances. **Only used for classification problems. Ignored for regression.**
- **shuffle** (*bool*) – Whether to shuffle the data before splitting.
- ****kwargs** – Extra arguments used in the TensorFlow's model `fit` function. See here.

hosa.models.rnn.rnn_models.RNNClassification.compile

RNNClassification.**compile**()

Compiles the model for training.

Returns *tensorflow.keras.Sequential* – Returns an untrained but compiled TensorFlow model.

hosa.models.rnn.rnn_models.RNNClassification.fit

RNNClassification.**fit**(*x*, *y*, *validation_size=0.33*, *shuffle=True*, *rtol=0.001*, *atol=0.0001*, *class_weights=None*, *imbalance_correction=False*, ***kwargs*)

Fits the model to data matrix x and target(s) y.

Parameters

- **x** (*numpy.ndarray*) – Input data.
- **y** (*numpy.ndarray*) – Target values (i.e., class labels).
- **class_weights** (*None or dict*) – Dictionary mapping class indices (integers) to a weight (float) value, used for weighting the loss function (during training only).
- **validation_size** (*float or int*) – Proportion of the train dataset to include in the validation split.
- **shuffle** (*bool*) – Whether to shuffle the data before splitting.
- **atol** (*float*) – Absolute tolerance used for early stopping based on the performance metric.
- **rtol** (*float*) – Relative tolerance used for early stopping based on the performance metric.
- **class_weights** – Dictionary mapping class indices (integers) to a weight (float) value, used for weighting the loss function (during training only). imbalance_correction (bool): *True* if correction for imbalance should be applied to the metrics; *False* otherwise.
- ****kwargs** – Extra arguments that are used in the TensorFlow's model `fit` function. See here.

Returns *tensorflow.keras.Sequential* – Returns a trained TensorFlow model.

hosa.models.rnn.rnn_models.RNNClassification.predict

RNNClassification.**predict**(*x*, ***kwargs*)

Predicts the target values using the input data in the trained model.

Parameters

- **x** (*numpy.ndarray*) – Input data.
- ****kwargs** – Extra arguments that are used in the TensorFlow's model `predict` function. See here.

Returns *tuple* – Returns a tuple containing the probability estimates and predicted classes.

hosa.models.rnn.rnn_models.RNNClassification.prepare

RNNClassification.**prepare**(*x*, *y*)

Prepares the model by adding the layers to the estimator: input layer, `n_subs_layers` subsequent layers, a dropout layer, a dense layer, and an output layer.

Parameters

- **x** (*numpy.ndarray*) – Input data.
- **y** (*numpy.ndarray*) – Target values (i.e., class labels).

hosa.models.rnn.rnn_models.RNNClassification.score

RNNClassification.**score**(*x*, *y*, *imbalance_correction=False*)

Computes the performance metrics on the given input data and target values.

Parameters

- **x** (*numpy.ndarray*) – Input data.
- **y** (*numpy.ndarray*) – Target values (i.e., class labels).
- **imbalance_correction** (*bool*) – *True* if correction for imbalance should be applied to the metrics; *False* otherwise.

Returns *tuple* – Returns a tuple containing the area under the ROC curve (AUC), accuracy, sensitivity, and sensitivity.

Note: This function can be used for both binary and multiclass classification.

---

hosa.models.rnn.rnn_models.RNNRegression

class hosa.models.rnn.rnn_models.**RNNRegression**(*n_outputs*, *n_neurons_dense_layer*, *n_units*, *n_subs_layers*, *is_bidirectional=False*, *model_type='lstm'*, *optimizer='adam'*, *dropout_percentage=0.1*, *metrics=None*, *activation_function_dense='relu'*, *kernel_initializer='normal'*, *batch_size=1000*, *epochs=50*, *patience=5*, ***kwargs*)

Bases: *hosa.models.rnn.rnn_models.BaseRNN*

Recurrent Neural Network (RNN) model regressor.

The model comprises an input layer (an RNN or a bidirectional RNN cell), `n_subs_layers` subsequent layers (similar to the input cell), a dropout layer, a dense layer, and an output layer.

Parameters

- **n_outputs** (*int*) – Number of numerical values to predict in regression.
- **n_neurons_dense_layer** (*int*) – Number of neurons units of the penultimate dense layer (i.e., before the output layer).
- **n_units** (*int*) – Dimensionality of the output space, i.e., the dimensionality of the hidden state.
- **n_subs_layers** (*int*) – Number of subsequent layers beteween the input and output layers.
- **is_bidirectional** (*bool*) – If `true`, then bidirectional layers will be used to build the RNN model.
- **model_type** (*str*) – Type of RNN model to be used. Available options are `lstm`, for a Long Short-Term Memory model, or `gru`, for a Gated Recurrent Unit model.
- **optimizer** (*str*) – Name of the optimizer. See tensorflow.keras.optimizers.
- **dropout_percentage** (*float*) – Fraction of the input units to drop.

- **metrics** (*list*) – List of metrics to be evaluated by the model during training and testing. Each item of the list can be a string (name of a TensorFlow's built-in function), function, or a tf.keras.metrics.Metric instance. If `None`, `metrics` will default to `['mean_squared_error']`.
- **activation_function_dense** (*str*) – Activation function to use on the penultimate dense layer. If not specified, no activation is applied (i.e., uses the linear activation function). See tensorflow.keras.activations.
- **kernel_initializer** (*str*) – Initializer for the kernel weights matrix, used for the linear transformation of the inputs.
- **batch_size** (*int or None*) – Number of samples per batch of computation. If `None`, `batch_size` will default to 32.
- **epochs** (*int*) – Maximum number of epochs to train the model.
- **patience** (*int*) – Number of epochs with no improvement after which training will be stopped.
- ****kwargs** – *Ignored*. Extra arguments that are used for compatibility's sake.

Examples

```
import pandas as pd

from hosa.models.rnn import RNNRegression
from hosa.aux import create_overlapping

# 1 - Download, load, and split the data
dataset = pd.read_csv(
'https://raw.githubusercontent.com/jbrownlee/Datasets/master/airline-passengers
.csv', header=0, index_col=0)
x = dataset.Passengers.to_numpy().reshape((len(dataset), 1))
y = dataset.Passengers.to_numpy()
x_train, y_train = x[:100], y[:100]
X_test, y_test = x[100:], y[100:]
# 2 - Prepare the data for cnn input
x_train, y_train = create_overlapping(x_train, y_train, RNNRegression, 'central',
10, timesteps=1)
X_test, y_test = create_overlapping(X_test, y_test, RNNRegression, 'central', 10,
timesteps=1)
# 3 - Create and fit the model
clf = RNNRegression(1, 200, epochs=500, patience=500)
clf.prepare(x_train, y_train)
clf.compile()
clf.fit(x_train, y_train)
# 4 - Calculate predictions
clf.predict(X_test)
# 5 - Compute the score
score = clf.score(X_test, y_test)
```

Methods

aux_fit(x, y, callback, validation_size[, ...])	Auxiliar function for classification and regression models compatibility.
compile()	Compiles the model for training.
fit(x, y[, validation_size, atol, rtol, shuffle])	Fits the model to data matrix x and target(s) y.
predict(x, **kwargs)	Predicts the target values using the input data in the trained model.
prepare(x, y)	Prepares the model by adding the layers to the estimator: input layer, `n_subs_layers` subsequent layers, a dropout layer, a dense layer, and an output layer.
score(x, y, **kwargs)	Computes the performance metrics on the given input data and target values.

hosa.models.rnn.rnn_models.RNNRegression.aux_fit

RNNRegression.**aux_fit**(*x*, *y*, *callback*, *validation_size*, *rtol=0.001*, *atol=0.0001*, *class_weights=None*, *imbalance_correction=None*, *shuffle=True*, ***kwargs*)

Auxiliar function for classification and regression models compatibility.

> **Warning:** This function is not meant to be called by itself. It is just an auxiliary function called by the child classes' `fit` function.

Parameters

- **x** (*numpy.ndarray*) – Input data.
- **y** (*numpy.ndarray*) – Target values (class labels in classification, real numbers in regression).
- **callback** (*object*) – Early stopping callback for halting the model's training.
- **validation_size** (*float or int*) – Proportion of the training dataset that will be used the validation split.
- **atol** (*float*) – Absolute tolerance used for early stopping based on the performance metric.
- **rtol** (*float*) – Relative tolerance used for early stopping based on the performance metric.
- **class_weights** (*None or dict*) – Dictionary mapping class indices (integers) to a weight (float) value, used for weighting the loss function (during training only). **Only used for classification problems. Ignored for regression.**
- **imbalance_correction** (*None or bool*) – Whether to apply correction to class imbalances. **Only used for classification problems. Ignored for regression.**
- **shuffle** (*bool*) – Whether to shuffle the data before splitting.
- ****kwargs** – Extra arguments used in the TensorFlow's model `fit` function. See here.

hosa.models.rnn.rnn_models.RNNRegression.compile

RNNRegression.**compile**()
Compiles the model for training.

Returns *tensorflow.keras.Sequential* – Returns an untrained but compiled TensorFlow model.

hosa.models.rnn.rnn_models.RNNRegression.fit

RNNRegression.**fit**(*x*, *y*, *validation_size=0.33*, *atol=0.0001*, *rtol=0.001*, *shuffle=True*, ***kwargs*)
Fits the model to data matrix x and target(s) y.

Parameters

- **x** (*numpy.ndarray*) – Input data.
- **y** (*numpy.ndarray*) – Target values (i.e., real numbers).
- **validation_size** (*float or int*) – Proportion of the train dataset to include in the validation split.
- **atol** (*float*) – Absolute tolerance used for early stopping based on the performance metric.
- **rtol** (*float*) – Relative tolerance used for early stopping based on the performance metric.
- **shuffle** (*bool*) – Whether to shuffle the data before splitting.
- ****kwargs** – Extra arguments that are used in the TensorFlow's model `fit` function. See here.

Returns *tensorflow.keras.Sequential* – Returns a trained TensorFlow model.

hosa.models.rnn.rnn_models.RNNRegression.predict

RNNRegression.**predict**(*x*, ***kwargs*)
Predicts the target values using the input data in the trained model.

Parameters

- **x** (*numpy.ndarray*) – Input data.
- ****kwargs** – Extra arguments that are used in the TensorFlow's model `predict` function. See here.

Returns *numpy.ndarray* – Returns an array containing the estimates.

hosa.models.rnn.rnn_models.RNNRegression.prepare

RNNRegression.**prepare**(*x*, *y*)
Prepares the model by adding the layers to the estimator: input layer, `n_subs_layers` subsequent layers, a dropout layer, a dense layer, and an output layer.

Parameters

- **x** (*numpy.ndarray*) – Input data.
- **y** (*numpy.ndarray*) – Target values (i.e., real numbers).

hosa.models.rnn.rnn_models.RNNRegression.score

RNNRegression.**score**(*x*, *y*, ***kwargs*)

Computes the performance metrics on the given input data and target values.

Parameters

- **x** (*numpy.ndarray*) – Input data.
- **y** (*numpy.ndarray*) – Target values (i.e., real numbers).
- ****kwargs** – *Ignored*. Only included here for compatibility's sake.

Returns *tuple* – Returns a tuple containing the mean squared error (MSE) and coefficient of determination (R^2).

1.1.3 Callbacks

This module implements early stopping callbacks for halting the model's training.

early_stopping.EarlyStoppingAtMinLoss(...[, ...])	This class implements the early stopping for avoiding overfitting the model.

hosa.callbacks.early_stopping.EarlyStoppingAtMinLoss

class hosa.callbacks.early_stopping.**EarlyStoppingAtMinLoss**(*class_model*, *patience*, *validation_data*, *imbalance_correction=False*, *rtol=0.001*, *atol=0.0001*)

Bases: keras.callbacks.Callback

This class implements the early stopping for avoiding overfitting the model. The training is stopped when the monitored metric has stopped improving.

Parameters

- **class_model** – Class of the object to be optimized. Available options are: *RNNClassification*, *RNNRegression*, *CNNClassification* and *CNNRegression*.
- **patience** (*int*) – Number of epochs with no improvement after which training will be stopped.
- **validation_data** (*numpy.ndarray*) – Input data extracted from the validation dataset (which was itself extracted from the training dataset).
- **imbalance_correction** (*bool*) – *True* if correction for imbalance should be applied to the metrics; *False* otherwise.
- **rtol** (*float*) – The relative tolerance parameter, as used in *numpy.isclose*. See numpy.isclose.
- **atol** (*float*) – The absolute tolerance parameter, as used in *numpy.isclose*. See numpy.isclose.

Methods

on_batch_begin(batch[, logs])	A backwards compatibility alias for *on_train_batch_begin*.
on_batch_end(batch[, logs])	A backwards compatibility alias for *on_train_batch_end*.
on_epoch_begin(epoch[, logs])	Called at the start of an epoch.
on_epoch_end(epoch[, logs])	Checks, based on the patience value, if the training should stop.
on_predict_batch_begin(batch[, logs])	Called at the beginning of a batch in *predict* methods.
on_predict_batch_end(batch[, logs])	Called at the end of a batch in *predict* methods.
on_predict_begin([logs])	Called at the beginning of prediction.
on_predict_end([logs])	Called at the end of prediction.
on_test_batch_begin(batch[, logs])	Called at the beginning of a batch in *evaluate* methods.
on_test_batch_end(batch[, logs])	Called at the end of a batch in *evaluate* methods.
on_test_begin([logs])	Called at the beginning of evaluation or validation.
on_test_end([logs])	Called at the end of evaluation or validation.
on_train_batch_begin(batch[, logs])	Called at the beginning of a training batch in *fit* methods.
on_train_batch_end(batch[, logs])	Called at the end of a training batch in *fit* methods.
on_train_begin([logs])	Called at the beginning of training to initialize the variables for early stopping.
on_train_end([logs])	This function is called when the training is finished, and it is used to set a flag for early stopping.
set_model(model)	
set_params(params)	

hosa.callbacks.early_stopping.EarlyStoppingAtMinLoss.on_batch_begin

EarlyStoppingAtMinLoss.**on_batch_begin**(*batch*, *logs=None*)

A backwards compatibility alias for *on_train_batch_begin*.

hosa.callbacks.early_stopping.EarlyStoppingAtMinLoss.on_batch_end

EarlyStoppingAtMinLoss.**on_batch_end**(*batch*, *logs=None*)

A backwards compatibility alias for *on_train_batch_end*.

hosa.callbacks.early_stopping.EarlyStoppingAtMinLoss.on_epoch_begin

EarlyStoppingAtMinLoss.**on_epoch_begin**(*epoch*, *logs=None*)

Called at the start of an epoch.

Subclasses should override for any actions to run. This function should only be called during TRAIN mode.

Parameters

- **epoch** – Integer, index of epoch.
- **logs** – Dict. Currently no data is passed to this argument for this method but that may change in the future.

hosa.callbacks.early_stopping.EarlyStoppingAtMinLoss.on_epoch_end

EarlyStoppingAtMinLoss.**on_epoch_end**(*epoch*, *logs=None*)

Checks, based on the patience value, if the training should stop. After stopping, it restores the model's weights from the epoch with the best value of the monitored quantity.

Parameters

- **epoch** (*int*) – Index of epoch.
- **logs** (*dict*) – Currently no data is passed to this argument for this method but that
- **future.** (*may change in the*) –

hosa.callbacks.early_stopping.EarlyStoppingAtMinLoss.on_predict_batch_begin

EarlyStoppingAtMinLoss.**on_predict_batch_begin**(*batch*, *logs=None*)

Called at the beginning of a batch in *predict* methods.

Subclasses should override for any actions to run.

Note that if the *steps_per_execution* argument to *compile* in *tf.keras.Model* is set to *N*, this method will only be called every *N* batches.

Parameters

- **batch** – Integer, index of batch within the current epoch.
- **logs** – Dict. Currently no data is passed to this argument for this method but that may change in the future.

hosa.callbacks.early_stopping.EarlyStoppingAtMinLoss.on_predict_batch_end

EarlyStoppingAtMinLoss.**on_predict_batch_end**(*batch*, *logs=None*)

Called at the end of a batch in *predict* methods.

Subclasses should override for any actions to run.

Note that if the *steps_per_execution* argument to *compile* in *tf.keras.Model* is set to *N*, this method will only be called every *N* batches.

Parameters

- **batch** – Integer, index of batch within the current epoch.
- **logs** – Dict. Aggregated metric results up until this batch.

hosa.callbacks.early_stopping.EarlyStoppingAtMinLoss.on_predict_begin

EarlyStoppingAtMinLoss.**on_predict_begin**(*logs=None*)
Called at the beginning of prediction.

Subclasses should override for any actions to run.

Parameters **logs** – Dict. Currently no data is passed to this argument for this method but that may change in the future.

hosa.callbacks.early_stopping.EarlyStoppingAtMinLoss.on_predict_end

EarlyStoppingAtMinLoss.**on_predict_end**(*logs=None*)
Called at the end of prediction.

Subclasses should override for any actions to run.

Parameters **logs** – Dict. Currently no data is passed to this argument for this method but that may change in the future.

hosa.callbacks.early_stopping.EarlyStoppingAtMinLoss.on_test_batch_begin

EarlyStoppingAtMinLoss.**on_test_batch_begin**(*batch*, *logs=None*)
Called at the beginning of a batch in *evaluate* methods.

Also called at the beginning of a validation batch in the *fit* methods, if validation data is provided.

Subclasses should override for any actions to run.

Note that if the *steps_per_execution* argument to *compile* in *tf.keras.Model* is set to *N*, this method will only be called every *N* batches.

Parameters

- **batch** – Integer, index of batch within the current epoch.
- **logs** – Dict. Currently no data is passed to this argument for this method but that may change in the future.

hosa.callbacks.early_stopping.EarlyStoppingAtMinLoss.on_test_batch_end

EarlyStoppingAtMinLoss.**on_test_batch_end**(*batch*, *logs=None*)
Called at the end of a batch in *evaluate* methods.

Also called at the end of a validation batch in the *fit* methods, if validation data is provided.

Subclasses should override for any actions to run.

Note that if the *steps_per_execution* argument to *compile* in *tf.keras.Model* is set to *N*, this method will only be called every *N* batches.

Parameters

- **batch** – Integer, index of batch within the current epoch.
- **logs** – Dict. Aggregated metric results up until this batch.

hosa.callbacks.early_stopping.EarlyStoppingAtMinLoss.on_test_begin

EarlyStoppingAtMinLoss.**on_test_begin**(*logs=None*)
Called at the beginning of evaluation or validation.

Subclasses should override for any actions to run.

Parameters **logs** – Dict. Currently no data is passed to this argument for this method but that may change in the future.

hosa.callbacks.early_stopping.EarlyStoppingAtMinLoss.on_test_end

EarlyStoppingAtMinLoss.**on_test_end**(*logs=None*)
Called at the end of evaluation or validation.

Subclasses should override for any actions to run.

Parameters **logs** – Dict. Currently the output of the last call to *on_test_batch_end()* is passed to this argument for this method but that may change in the future.

hosa.callbacks.early_stopping.EarlyStoppingAtMinLoss.on_train_batch_begin

EarlyStoppingAtMinLoss.**on_train_batch_begin**(*batch*, *logs=None*)
Called at the beginning of a training batch in *fit* methods.

Subclasses should override for any actions to run.

Note that if the *steps_per_execution* argument to *compile* in *tf.keras.Model* is set to *N*, this method will only be called every *N* batches.

Parameters

- **batch** – Integer, index of batch within the current epoch.
- **logs** – Dict. Currently no data is passed to this argument for this method but that may change in the future.

hosa.callbacks.early_stopping.EarlyStoppingAtMinLoss.on_train_batch_end

EarlyStoppingAtMinLoss.**on_train_batch_end**(*batch*, *logs=None*)
Called at the end of a training batch in *fit* methods.

Subclasses should override for any actions to run.

Note that if the *steps_per_execution* argument to *compile* in *tf.keras.Model* is set to *N*, this method will only be called every *N* batches.

Parameters

- **batch** – Integer, index of batch within the current epoch.
- **logs** – Dict. Aggregated metric results up until this batch.

hosa.callbacks.early_stopping.EarlyStoppingAtMinLoss.on_train_begin

EarlyStoppingAtMinLoss.**on_train_begin**(*logs=None*)

Called at the beginning of training to initialize the variables for early stopping.

Parameters

- **logs** (*dict*) – Currently no data is passed to this argument for this method but that
- **future.** (*may change in the*) –

hosa.callbacks.early_stopping.EarlyStoppingAtMinLoss.on_train_end

EarlyStoppingAtMinLoss.**on_train_end**(*logs=None*)

This function is called when the training is finished, and it is used to set a flag for early stopping.

Parameters

- **logs** (*dict*) – Currently no data is passed to this argument for this method but that
- **future.** (*may change in the*) –

hosa.callbacks.early_stopping.EarlyStoppingAtMinLoss.set_model

EarlyStoppingAtMinLoss.**set_model**(*model*)

hosa.callbacks.early_stopping.EarlyStoppingAtMinLoss.set_params

EarlyStoppingAtMinLoss.**set_params**(*params*)

1.1.4 Helpers

This module implements all the helper functions required to prepare data, access models, and prepare the parameter grid for HOSA.

functions.sliding_window(x, window_size)	Creates a sliding window view of *x* according to the window size specified.
functions.metrics_multiclass(y_true, ...[, ...])	Computes the performance metrics for classification problems.
functions.create_overlapping(x, y, model[, ...])	Depending on the model chosen, prepare the data with segmented windows according to the number of epochs and overlapping type.
functions.create_parameter_grid(param_grid)	This function generates an iterator that can be traversed through all the parameter value combinations.
functions.prepare_param_overlapping(...)	Prepares, considering the given specification, the parameters for creating the input and output overlapping.

hosa.helpers.functions.sliding_window

hosa.helpers.functions.**sliding_window**(*x*, *window_size*)
Creates a sliding window view of *x* according to the window size specified.

Note: This function is based on the NumPy's function *sliding_window_view*. See numpy.lib.stride_tricks.sliding_window_view.

Parameters

- **x** (*numpy.ndarray*) – Input data.
- **window_size** (*int*) – Size of the sliding window.

Returns *(numpy.ndarray)* – Returns a sliding window view of the array.

hosa.helpers.functions.metrics_multiclass

hosa.helpers.functions.**metrics_multiclass**(*y_true*, *y_probs*, *n_classes*, *imbalance_correction=False*)
Computes the performance metrics for classification problems. This function supports multiclass classification, being, in this case, the metrics given in terms of the average value, or weighed average if *imbalance_correction=True*.

Parameters

- **y_true** (*numpy.ndarray*) – Ground truth (correct) labels.
- **y_probs** (*numpy.ndarray*) – Probability estimates.
- **n_classes** (*int*) – Number of classes (or labels) of the classification problem.
- **imbalance_correction** (*bool*) – *True* if correction for imbalance should be applied to the
- **otherwise.** (*metrics; False*) –

Returns *(tuple)* – Returns a tuple with the metrics for AUC, accuracy, sensitivity, and specificity.

hosa.helpers.functions.create_overlapping

hosa.helpers.functions.**create_overlapping**(*x*, *y*, *model*, *n_overlapping_epochs=0*, *overlapping_type=None*, *n_stride=1*, *n_timesteps=None*)
Depending on the model chosen, prepare the data with segmented windows according to the number of epochs and overlapping type.

Parameters

- **x** (*numpy.ndarray*) – Input data.
- **y** (*numpy.ndarray or None*) – Target values (class labels in classification, real numbers in regression). If *None*, the parameter will be ingored.
- **model** (*object*) – Class of the object to be optimized. Available options are: *RNNClassification*, *RNNRegression*, *CNNClassification* and *CNNRegression*.
- **n_overlapping_epochs** (*int*) – Number of epochs to be overlapped (in other words, the overlap duration).
- **overlapping_type** (*str or None*) – Type of overlapping to perform on the data. Available

- **are** (*options*) – *central*, where the target value corresponds to the central epoch of the overlapping window; *left*, where the target value corresponds to the rightmost epoch of the overlapping window and *right*, where the target value corresponds to the leftmost epoch of the overlapping window. When *n_overlapping_epochs=0*, this parameter is ignored.
- **n_stride** (*int*) – Number of strides to apply to the data.
- **n_timesteps** (*int*) – Number of timesteps to apply to the data for recurrent models, in other words, the number of lagged observations to be used in the model. **Only used when `model=RNNClassification` or `model=RNNRegression`.**

Returns

tuple –

Returns a tuple with the input data (*x*) and target values (*y*)—or *None* if *y=None*—, both in segmented window view.

hosa.helpers.functions.create_parameter_grid

hosa.helpers.functions.**create_parameter_grid**(*param_grid*)

This function generates an iterator that can be traversed through all the parameter value combinations. The order of the generated parameter combinations is deterministic, being done according to the total number of values to try in each parameter in descending order.

Parameters

- **param_grid** (*dict*) – Dictionary with parameters names (*str*) as keys and lists of
- **values.** (*parameter settings to try as*) –

hosa.helpers.functions.prepare_param_overlapping

hosa.helpers.functions.**prepare_param_overlapping**(*specification*)

Prepares, considering the given specification, the parameters for creating the input and output overlapping.

Parameters **specification** (*dict*) – Parameter names mapped to their values.

Returns *tuple* – Returns a tuple containing the overlapping type, number of overlapping epochs, strides, and timesteps.

1.2 Installation

1.3 References

BIBLIOGRAPHY

[Men20] Mendonça, F.; Mostafa, S.; Morgado-Dias, F.; Juliá-Serdá, G.; Ravelo-García, A. A Method for Sleep Quality Analysis Based on CNN Ensemble With Implementation in a Portable Wireless Device. *IEEE Access* **2020**, 8, 158523–158537.

PYTHON MODULE INDEX

h

INDEX

H

M

O

P

R

S