DD-GAN-AE

Release 1.0.0

Aug 24, 2021

CONTENTS:

1 Models **1**
- 1.1 Adversarial Autoencoder . . . 1
- 1.2 Convolutional Autoencoder . . . 3
- 1.3 SVD Autoencoder . . . 4

2 Hyperparameter optimization **7**
- 2.1 Flow Past Cylinder . . . 7
- 2.2 Slug Flow . . . 7

3 Utilities **9**

4 Library of Architectures **11**
- 4.1 Convolutional Architectures . . . 11
- 4.2 Discriminator Architectures . . . 17
- 4.3 Mixed architectures for SVD Autoencoder . . . 18

Python Module Index **25**

Index **27**

CHAPTER

ONE

MODELS

This package contains three models that can readily be used with user defined or imported network architectures. These consist of: adversarial autoencoder, convolutional autoencoder, and SVD autoencoder.

1.1 Adversarial Autoencoder

Implementation of two classes with a slightly different version of the adversarial autoencoder model. The former corresponds to the original paper on adversarial autoencoders:

https://arxiv.org/abs/1511.05644

and the second is an adaptation with weighted losses inspired by:

https://arxiv.org/abs/2104.06297

class models.aae.**AAE**(*encoder*, *decoder*, *discriminator*, *optimizer*, *seed=None*)

Adversarial autoencoder class

compile(*input_shape*)

Compilation of models according to original paper on adversarial autoencoders

Parameters **input_shape** (*tuple*) – Shape of input data

train(*train_data*, *epochs*, *val_data=None*, *batch_size=128*, *val_batch_size=128*, *wandb_log=False*)

Training model according to original paper on adversarial autoencoders

Parameters

- **train_data** (*np.ndarray*) – Train dataset
- **epochs** (*int*) – Number of training epochs to execute
- **val_data** (*np.ndarray, optional*) – Validation dataset. Defaults to None.
- **batch_size** (*int, optional*) – Training batch size. Defaults to 128.
- **val_batch_size** (*int, optional*) – Validation batch size. Defaults to 128.
- **wandb_log** (*bool, optional*) – Whether to log results to wandb. Note function needs to be called in wandb.init() scope for this to work. Defaults to False.

validate(*val_dataset*, *val_batch_size=128*)

Validate model on previously unseen dataset.

Parameters

- **val_dataset** (*np.ndarray*) – Validation dataset
- **val_batch_size** (*int, optional*) – Validation batch size. Defaults to 128.

Returns Validation losses and accuracies

Return type tuple

class models.aae.**AAE_combined_loss**(*encoder*, *decoder*, *discriminator*, *optimizer*, *seed=None*)
Adversarial autoencoder with combined loss class

compile(*input_shape*)
Compilation of models where we use a training method that weights the losses of the discriminator and autoencoder and as such combines them into one loss and trains on them simultaneously.

Parameters **input_shape** (*tuple*) – Shape of input data

train(*train_data*, *epochs*, *val_data=None*, *batch_size=128*, *val_batch_size=128*, *wandb_log=False*, *n_discriminator=5*)
Training model with combined loss strategy

Parameters

- **train_data** (*np.ndarray*) – Train dataset
- **epochs** (*int*) – Number of training epochs to execute
- **val_data** (*np.ndarray, optional*) – Validation dataset. Defaults to None.
- **batch_size** (*int, optional*) – Training batch size. Defaults to 128.
- **val_batch_size** (*int, optional*) – Validation batch size. Defaults to 128.
- **wandb_log** (*bool, optional*) – Whether to log results to wandb. Note function needs to be called in wandb.init() scope for this to work. Defaults to False.

validate(*val_dataset*, *val_batch_size=128*)
Validate model on previously unseen dataset.

Parameters

- **val_dataset** (*np.array*) – Validation dataset
- **val_batch_size** (*int, optional*) – Validation batch size. Defaults to 128.

Returns Validation losses and accuracies

Return type tuple

models.aae.**plot_losses**(*d_loss*, *g_loss*, *liveloss*, *d_loss_val=None*, *g_loss_val=None*)
Convenience function to plot a set of losses. Can be used by adversarial type of networks

Parameters

- **d_loss** (*float*) – Discriminator loss value
- **g_loss** (*float*) – Generator loss value
- **livaloss** (*object*) – livelossplot class instance
- **d_loss_val** (*float, optional*) – Validation discriminator loss value. Defaults to None.
- **g_loss_val** (*float, optional*) – Validation generator loss value. Defaults to None.

models.aae.**print_losses**(*d_loss*, *g_loss*, *epoch*, *d_loss_val=None*, *g_loss_val=None*)
Convenience function to print a set of losses. Can be used by adversarial type of networks

Parameters

- **d_loss** (*float*) – Discriminator loss value
- **g_loss** (*float*) – Generator loss value

- **epoch** (*int*) – Current epoch
- **d_loss_val** (*float, optional*) – Validation discriminator loss value. Defaults to None.
- **g_loss_val** (*float, optional*) – Validation generator loss value. Defaults to None.

1.2 Convolutional Autoencoder

Convolutional autoencoder model.

class models.cae.**CAE**(*encoder*, *decoder*, *optimizer*, *seed=None*)

Convolutional autoencoder class

compile(*input_shape*, *pi_loss=False*)

Compile model

Parameters

- **input_shape** (*tuple*) – Shape of input data
- **pi_loss** (*bool, optional*) – Whether to use physics informed loss, note this is currently in experimental stage. Defaults to False.

predict(*data*)

Convenience function that gives class predict method that just does forward pass through model.

Parameters **data** (*np.ndarray*) – Input grids that are to be reconstructed by this model

Returns Reconstructed grids

Return type np.ndarray

train(*train_data*, *epochs*, *val_data=None*, *batch_size=128*, *val_batch_size=128*, *wandb_log=False*)

Training convolutional autoencoder model

Parameters

- **train_data** (*np.ndarray*) – Train dataset
- **epochs** (*int*) – Number of training epochs to execute
- **val_data** (*np.ndarray, optional*) – Validation dataset. Defaults to None.
- **batch_size** (*int, optional*) – Training batch size. Defaults to 128.
- **val_batch_size** (*int, optional*) – Validation batch size. Defaults to 128.
- **wandb_log** (*bool, optional*) – Whether to log results to wandb. Note function needs to be called in wandb.init() scope for this to work. Defaults to False.

train_generate(*data_file_base*, *val_data*, *epochs*, *regen_epochs*)

Train and every *regen_epochs* epochs generate a new training set from available vtu files.

Currently in development.

Parameters

- **data_file_base** (*string*) – Path to vtu files
- **val_data** (*np.array*) – Array to use as validation dataset
- **epochs** (*int*) – Number of total epochs to do
- **regen_epochs** (*int*) – Interval at which to regenerate a new dataset

validate(*val_dataset*)
Validate model on validation dataset.

Parameters

- **val_dataset** (*np.ndarray*) – Validation dataset
- **val_batch_size** (*int, optional*) – Validation batch size. Defaults to 128.

Returns Validation losses and accuracies

Return type tuple

1.3 SVD Autoencoder

SVD autoencoder model. Can be used with any of the dense (or 1D convolutional) encoder and decoder architectures in architectures directory.

class models.svdae.**SVDAE**(*encoder*, *decoder*, *optimizer*, *seed=None*)
SVD Autoencoder class

calc_pod(*snapshots*, *nPOD=-2*, *cumulative_tol=0.99*)
Calculate POD coefficients and basis functions

Parameters

- **snapshots** (*list of ndarrays*) – List of arrays with subgrid snapshots. shape: (n_grids, n_nodes, n_timelevels)
- **nPOD** (*int*) – number of pod coefficients to use. Set to -2 for dynamic setting where a tolerance determines the number of POD coefficients. Defaults to -2.
- **cumulative_tol** (*float*) – Tolerance value to use if this option is selected in *nPOD* parameter.

Returns POD coefficients per subgrid

Return type list of ndarrays

compile(*nPOD*, *weight_loss=False*)
Compile SVD autoencoder

Parameters

- **nPOD** (*int*) – Number of POD coefficients to use
- **weight_loss** (*bool, optional*) – Whether to weight losses by singular value magnitudes, note this feature is currently in experimental phase. Defaults to False.

predict(*data*)
Pass a collection of grids through the model

Parameters **data** (*np.ndarray*) – Dataset that is to be passed through the model

Returns Reconstructed dataset

Return type np.ndarray

predict_single(*snapshot*)
Pass single array through full model including POD and the autoencoder

Parameters **snapshot** (*np.ndarray*) – Grid to be predicted

Returns Grid reconstructed by SVD autoencoder

Return type np.ndarray

reconstruct_from_pod(*R*)
Convenience function to reconstruct a grid from bases and coefficients

Parameters

- **coeffs** (*np.ndarray*) – POD coefficients
- **R** (*np.ndarray*) – POD basis matrix

Returns Grid reconstructed through POD

Return type np.ndarray

train(*train_data*, *epochs*, *val_data=None*, *batch_size=128*, *val_batch_size=128*, *wandb_log=False*)
Training SVD autoencoder model

Parameters

- **train_data** (*np.ndarray*) – Train dataset
- **epochs** (*int*) – Number of training epochs to execute
- **val_data** (*np.ndarray, optional*) – Validation dataset. Defaults to None.
- **batch_size** (*int, optional*) – Training batch size. Defaults to 128.
- **val_batch_size** (*int, optional*) – Validation batch size. Defaults to 128.
- **wandb_log** (*bool, optional*) – Whether to log results to wandb. Note function needs to be called in wandb.init() scope for this to work. Defaults to False.

validate(*val_dataset*)
Validate model on validation dataset.

Parameters **val_dataset** (*np.ndarray*) – Validation dataset

Returns Validation losses and accuracies

Return type tuple

models.svdae.**plot_losses**(*loss*, *liveloss*, *loss_val=None*)
Convenience function to plot a set of losses. Can be used by SVD autoencoder.

Parameters

- **loss** (*float*) – loss value
- **livaloss** (*object*) – livelossplot class instance
- **loss_val** (*float, optional*) – Validation loss value. Defaults to None.

models.svdae.**print_losses**(*loss*, *epoch*, *loss_val=None*)
Convenience function to print a set of losses. can be used by SVD autoencoder.

Parameters

- **loss** (*float*) – Loss value
- **epoch** (*int*) – Current epoch

CHAPTER

TWO

HYPERPARAMETER OPTIMIZATION

This package contains some functionality for doing hyperparameter optimization with the Weights and Biases platform. Below is the documentation for the functions that handle this for the flow past cylinder and slug flow problems.

2.1 Flow Past Cylinder

Functions used for weights and biases hyperparameter optimization of autoencoders on FPC dataset.

wandb.train_wandb_fpc.**train_wandb_aae**(*config=None*)
Construct and subsequently train the model while reporting losses to weights and biases platform. Weights and biases also controls hyperparameters.

Parameters **config** (*dict, optional*) – Dictionary with hyperparameters, set by weights and biases. Defaults to None.

wandb.train_wandb_fpc.**train_wandb_cae**(*config=None*)
Construct and subsequently train the model while reporting losses to weights and biases platform. Weights and biases also controls hyperparameters.

Parameters **config** (*dict, optional*) – Dictionary with hyperparameters, set by weights and biases. Defaults to None.

wandb.train_wandb_fpc.**train_wandb_svdae**(*config=None*)
Construct and subsequently train the model while reporting losses to weights and biases platform. Weights and biases also controls hyperparameters.

Parameters **config** (*dict, optional*) – Dictionary with hyperparameters, set by weights and biases. Defaults to None.

2.2 Slug Flow

Functions used for weights and biases hyperparameter optimization of autoencoders on slug flow dataset

wandb.train_wandb_sf.**train_wandb_aae**(*config=None*)
Construct and subsequently train the model while reporting losses to weights and biases platform. Weights and biases also controls hyperparameters.

Parameters **config** (*dict, optional*) – Dictionary with hyperparameters, set by weights and biases. Defaults to None.

wandb.train_wandb_sf.**train_wandb_cae**(*config=None*)
Construct and subsequently train the model while reporting losses to weights and biases platform. Weights and biases also controls hyperparameters.

Parameters **config** (*dict, optional*) – Dictionary with hyperparameters, set by weights and biases. Defaults to None.

wandb.train_wandb_sf.**train_wandb_svdae**(*config=None*)

Construct and subsequently train the model while reporting losses to weights and biases platform. Weights and biases also controls hyperparameters.

Parameters **config** (*dict, optional*) – Dictionary with hyperparameters, set by weights and biases. Defaults to None.

CHAPTER

THREE

UTILITIES

This package also contains some utilities for printing, loss functions, etc...

General utilities for package

utils.**calc_pod**(*snapshots*, *nPOD=-2*, *cumulative_tol=0.99*, *R=None*)
Calculate POD coefficients and basis functions

Parameters **snapshots** (*list of ndarrays*) – List of arrays with subgrid snapshots. shape: (n_grids, n_nodes*n_scalar, n_timelevels)

Returns POD coefficients per subgrid

Return type list of ndarrays

class utils.**mse_PI**(*dx=None*, *dy=None*)
Mean squared error loss class.

class utils.**mse_weighted**
Custom weighted mean squared error loss

utils.**reconstruct_pod**(*coeffs*, *R*)
Reconstruct grid from POD coefficients and transormation matrix R.

Parameters

- **coeffs** (*np.array*) – POD coefficients
- **R** (*np.array*) – Transformation matrix R

Returns Reconstructed grid

Return type np.array

Collection of preprocessing utilities. Further preprocessing utilities can be found in DD-GAN submodule.

preprocessing.utils.**convert_2d**(*subgrid_snapshots*, *shape*, *timesteps*)
Utility to convert list of grids to list of 2d grids

Parameters

- **subgrid_snapshots** (*List*) – List of subgrids
- **shape** (*Tuple*) – Shape of 2d grid, e.g. (nFields, nx, ny)
- **timesteps** (*Int*) – Number of timesteps

Returns List of converted subgrids

Return type List

CHAPTER

FOUR

LIBRARY OF ARCHITECTURES

While the package is built in such a way that the user can easily use the architectures they designed. This package also includes a set of premade architectures. These are listed below.

4.1 Convolutional Architectures

Note that we have different architectures for the flow past cylinder and slug flow problems.

4.1.1 Two Dimensional (Flow Past Cylinder)

Collection of encoders and decoders that can readily be imported and used by the 2D adversarial and convolutional autoencoder and predictive models.

Note that these models are currently adjusted to a 55 by 42 input shape.

`architectures.cae.D2.cae.`**`build_agostini_encoder_decoder`**(*input_shape*, *latent_dim*, *initializer*, *info=False*)

This encoder-decoder pair currently works for 221 by 42 grids

Parameters

- **`input_shape`** (*`tuple`*) – Shape tuple of input grids
- **`latent_dim`** (*`int`*) – Number of latent variables
- **`initializer`** (*`tf.keras.initializers.Initializer`*) – Weights initializer
- **`info`** (*`bool, optional`*) – Whether to print info. Defaults to False.

Returns encoder, decoder pair

Return type tuple

`architectures.cae.D2.cae.`**`build_custom_conv_decoder`**(*latent_dim*, *initializer*, *info=False*)

Builds a 2D convolutional decoder

Parameters

- **`latent_dim`** (*`int`*) – Number of latent variables
- **`initializer`** (*`tf.keras.initializers.Initializer`*) – Weights initializer
- **`info`** (*`bool, optional`*) – Whether to print info. Defaults to False.

Returns encoder

Return type tf.keras.Model

architectures.cae.D2.cae.**build_custom_conv_encoder**(*input_shape*, *latent_dim*, *initializer*, *info=False*)

Builds a 2D convolutional encoder

Parameters

- **input_shape** (*tuple*) – Shape tuple of input grids
- **latent_dim** (*int*) – Number of latent variables
- **initializer** (*tf.keras.initializers.Initializer*) – Weights initializer
- **info** (*bool, optional*) – Whether to print info. Defaults to False.

Returns encoder

Return type tf.keras.Model

architectures.cae.D2.cae.**build_deeper_omata_encoder_decoder**(*input_shape*, *latent_dim*, *initializer*, *info=False*, *act='elu'*, *dense_act=None*)

This encoder-decoder pair currently works for 55 by 42 grids

Parameters

- **input_shape** (*tuple*) – Shape tuple of input grids
- **latent_dim** (*int*) – Number of latent variables
- **initializer** (*tf.keras.initializers.Initializer*) – Weights initializer
- **info** (*bool, optional*) – Whether to print info. Defaults to False.
- **act** (*str, optional*) – Activation function to use. Defaults to "elu".
- **dense_act** (*str, optional*) – Dense layer activation function to use. Defaults to None.

Returns encoder, decoder pair

Return type tuple

architectures.cae.D2.cae.**build_denser_omata_encoder_decoder**(*input_shape*, *latent_dim*, *initializer*, *info=False*, *act='elu'*, *dense_act=None*)

This encoder-decoder pair currently works for 55 by 42 grids

Parameters

- **input_shape** (*tuple*) – Shape tuple of input grids
- **latent_dim** (*int*) – Number of latent variables
- **initializer** (*tf.keras.initializers.Initializer*) – Weights initializer
- **info** (*bool, optional*) – Whether to print info. Defaults to False.
- **act** (*str, optional*) – Activation function to use. Defaults to "elu".
- **dense_act** (*str, optional*) – Dense layer activation function to use. Defaults to None.

Returns encoder, decoder pair

Return type tuple

architectures.cae.D2.cae.**build_densest_omata_encoder_decoder**(*input_shape*, *latent_dim*, *initializer*, *info=False*, *act='elu'*, *dense_act=None*)

This encoder-decoder pair currently works for 55 by 42 grids

Parameters

- **input_shape** (*tuple*) – Shape tuple of input grids
- **latent_dim** (*int*) – Number of latent variables
- **initializer** (*tf.keras.initializers.Initializer*) – Weights initializer
- **info** (*bool, optional*) – Whether to print info. Defaults to False.
- **act** (*str, optional*) – Activation function to use. Defaults to "elu".
- **dense_act** (*str, optional*) – Dense layer activation function to use. Defaults to None.

Returns encoder, decoder pair

Return type tuple

architectures.cae.D2.cae.**build_mnist_wide_omata_encoder_decoder**(*input_shape*, *latent_dim*, *initializer*, *info=False*)

This encoder-decoder pair currently works for 28 by 28 grids so can work on MNIST dataset as a test

Parameters

- **input_shape** (*tuple*) – Shape tuple of input grids
- **latent_dim** (*int*) – Number of latent variables
- **initializer** (*tf.keras.initializers.Initializer*) – Weights initializer
- **info** (*bool, optional*) – Whether to print info. Defaults to False.

Returns encoder, decoder pair

Return type tuple

architectures.cae.D2.cae.**build_omata_encoder_decoder**(*input_shape*, *latent_dim*, *initializer*, *info=False*, *act='elu'*, *dense_act=None*)

This encoder-decoder pair currently works for 55 by 42 grids

Parameters

- **input_shape** (*tuple*) – Shape tuple of input grids
- **latent_dim** (*int*) – Number of latent variables
- **initializer** (*tf.keras.initializers.Initializer*) – Weights initializer
- **info** (*bool, optional*) – Whether to print info. Defaults to False.
- **act** (*str, optional*) – Activation function to use. Defaults to "elu".
- **dense_act** (*str, optional*) – Dense layer activation function to use. Defaults to None.

Returns encoder, decoder pair

Return type tuple

architectures.cae.D2.cae.**build_wide_omata_encoder_decoder**(*input_shape*, *latent_dim*, *initializer*, *info=False*, *act='elu'*, *dense_act=None*)

This encoder-decoder pair currently works for 55 by 42 grids

Parameters

- **input_shape** (*tuple*) – Shape tuple of input grids
- **latent_dim** (*int*) – Number of latent variables
- **initializer** (*tf.keras.initializers.Initializer*) – Weights initializer
- **info** (*bool, optional*) – Whether to print info. Defaults to False.
- **act** (*str, optional*) – Activation function to use. Defaults to "elu".
- **dense_act** (*str, optional*) – Dense layer activation function to use. Defaults to None.

Returns encoder, decoder pair

Return type tuple

architectures.cae.D2.cae.**build_wider_omata_encoder_decoder**(*input_shape*, *latent_dim*, *initializer*, *info=False*, *act='elu'*, *dense_act=None*)

This encoder-decoder pair currently works for 55 by 42 grids

Parameters

- **input_shape** (*tuple*) – Shape tuple of input grids
- **latent_dim** (*int*) – Number of latent variables
- **initializer** (*tf.keras.initializers.Initializer*) – Weights initializer
- **info** (*bool, optional*) – Whether to print info. Defaults to False.
- **act** (*str, optional*) – Activation function to use. Defaults to "elu".
- **dense_act** (*str, optional*) – Dense layer activation function to use. Defaults to None.

Returns encoder, decoder pair

Return type tuple

4.1.2 Three Dimensional (Slug Flow)

Collection of encoders and decoders that can readily be imported and used by the 3D adversarial and convolutional autoencoder and predictive models.

Note that these models are currently adjusted to a 60 by 60 by 20 input shape.

architectures.cae.D3.cae.**build_deeper_omata_encoder_decoder**(*input_shape*, *latent_dim*, *initializer*, *info=False*, *act='elu'*, *dense_act=None*, *final_act='linear'*)

This encoder-decoder pair currently works for 60 by 20 by 20 grids

Parameters

- **input_shape** (*tuple*) – Shape tuple of input grids
- **latent_dim** (*int*) – Number of latent variables
- **initializer** (*tf.keras.initializers.Initializer*) – Weights initializer
- **info** (*bool, optional*) – Whether to print info. Defaults to False.
- **act** (*str, optional*) – Activation function to use. Defaults to "elu".
- **dense_act** (*str, optional*) – Dense layer activation function to use. Defaults to None.
- **final_act** (*str, optional*) – Dense layer activation function to use. Defaults to "linear".

Returns encoder, decoder pair

Return type tuple

architectures.cae.D3.cae.**build_denser_omata_encoder_decoder**(*input_shape*, *latent_dim*, *initializer*, *info=False*, *act='elu'*, *dense_act=None*, *final_act='linear'*)

This encoder-decoder pair currently works for 60 by 20 by 20 grids

Parameters

- **input_shape** (*tuple*) – Shape tuple of input grids
- **latent_dim** (*int*) – Number of latent variables
- **initializer** (*tf.keras.initializers.Initializer*) – Weights initializer
- **info** (*bool, optional*) – Whether to print info. Defaults to False.
- **act** (*str, optional*) – Activation function to use. Defaults to "elu".
- **dense_act** (*str, optional*) – Dense layer activation function to use. Defaults to None.
- **final_act** (*str, optional*) – Dense layer activation function to use. Defaults to "linear".

Returns encoder, decoder pair

Return type tuple

architectures.cae.D3.cae.**build_densest_omata_encoder_decoder**(*input_shape*, *latent_dim*, *initializer*, *info=False*, *act='elu'*, *dense_act=None*, *final_act='linear'*)

This encoder-decoder pair currently works for 60 by 20 by 20 grids

Parameters

- **input_shape** (*tuple*) – Shape tuple of input grids
- **latent_dim** (*int*) – Number of latent variables
- **initializer** (*tf.keras.initializers.Initializer*) – Weights initializer
- **info** (*bool, optional*) – Whether to print info. Defaults to False.
- **act** (*str, optional*) – Activation function to use. Defaults to "elu".
- **dense_act** (*str, optional*) – Dense layer activation function to use. Defaults to None.
- **final_act** (*str, optional*) – Dense layer activation function to use. Defaults to "linear".

Returns encoder, decoder pair

Return type tuple

architectures.cae.D3.cae.**build_densest_thinner_omata_encoder_decoder**(*input_shape*, *latent_dim*, *initializer*, *info=False*, *act='elu'*, *dense_act=None*, *final_act='linear'*)

This encoder-decoder pair currently works for 60 by 20 by 20 grids

Parameters

- **input_shape** (*tuple*) – Shape tuple of input grids
- **latent_dim** (*int*) – Number of latent variables
- **initializer** (*tf.keras.initializers.Initializer*) – Weights initializer
- **info** (*bool, optional*) – Whether to print info. Defaults to False.
- **act** (*str, optional*) – Activation function to use. Defaults to "elu".
- **dense_act** (*str, optional*) – Dense layer activation function to use. Defaults to None.
- **final_act** (*str, optional*) – Dense layer activation function to use. Defaults to "linear".

Returns encoder, decoder pair

Return type tuple

architectures.cae.D3.cae.**build_omata_encoder_decoder**(*input_shape*, *latent_dim*, *initializer*, *info=False*, *act='elu'*, *dense_act=None*, *final_act='linear'*)

This encoder-decoder pair currently works for 60 by 20 by 20 grids

Parameters

- **input_shape** (*tuple*) – Shape tuple of input grids
- **latent_dim** (*int*) – Number of latent variables
- **initializer** (*tf.keras.initializers.Initializer*) – Weights initializer
- **info** (*bool, optional*) – Whether to print info. Defaults to False.
- **act** (*str, optional*) – Activation function to use. Defaults to "elu".
- **dense_act** (*str, optional*) – Dense layer activation function to use. Defaults to None.
- **final_act** (*str, optional*) – Dense layer activation function to use. Defaults to "linear".

Returns encoder, decoder pair

Return type tuple

architectures.cae.D3.cae.**build_wide_omata_encoder_decoder**(*input_shape*, *latent_dim*, *initializer*, *info=False*, *act='elu'*, *dense_act=None*, *final_act='linear'*)

This encoder-decoder pair currently works for 60 by 20 by 20 grids

Parameters

- **input_shape** (*tuple*) – Shape tuple of input grids
- **latent_dim** (*int*) – Number of latent variables
- **initializer** (*tf.keras.initializers.Initializer*) – Weights initializer
- **info** (*bool, optional*) – Whether to print info. Defaults to False.
- **act** (*str, optional*) – Activation function to use. Defaults to "elu".
- **dense_act** (*str, optional*) – Dense layer activation function to use. Defaults to None.
- **final_act** (*str, optional*) – Dense layer activation function to use. Defaults to "linear".

Returns encoder, decoder pair

Return type tuple

4.2 Discriminator Architectures

Collection of discriminators that can readily be imported and used by the adversarial autoencoder and predictive models

architectures.discriminators.discriminators.**build_custom_discriminator**(*latent_dim*, *initializer*, *info=False*)

Build a discriminator

Parameters

- **latent_dim** (*int*) – Number of latent variables.
- **initializer** (*tf.keras.initializers.Initializer*) – Weights initializer
- **info** (*bool, optional*) – Whether to print model info. Defaults to False.

Returns discriminator

Return type tf.keras.Model

architectures.discriminators.discriminators.**build_custom_wider_discriminator**(*latent_dim*, *initializer*, *info=False*)

Build a discriminator

Parameters

- **latent_dim** (*int*) – Number of latent variables.
- **initializer** (*tf.keras.initializers.Initializer*) – Weights initializer
- **info** (*bool, optional*) – Whether to print model info. Defaults to False.

Returns discriminator

Return type tf.keras.Model

4.3 Mixed architectures for SVD Autoencoder

Collection of encoders and decoders that can readily be imported and used by the SVD autoencoder model.

architectures.svdae.svdae.**build_conv_encoder_decoder**(*input_dim*, *latent_dim*, *initializer*, *info=False*, *act='relu'*, *dense_act='relu'*, *dropout=0.6*, *final_act='linear'*)

Create a 1D convolutional encoder and decoder

Parameters

- **input_dim** (*tuple*) – Shape tuple of input grids
- **latent_dim** (*int*) – Number of latent variables
- **initializer** (*tf.keras.initializers.Initializer*) – Weights initializer
- **info** (*bool, optional*) – Whether to print info. Defaults to False.
- **act** (*str, optional*) – Activation function to use. Defaults to "elu".
- **dense_act** (*str, optional*) – Dense layer activation function to use. Defaults to None.
- **dropout** (*int, optional*) – Dropout factor to use. Defaults to 0.6.

- **final_act** (*str, optional*) – Dense layer activation function to use. Defaults to "linear".

Returns encoder, decoder pair

Return type tuple

architectures.svdae.svdae.**build_deeper_dense_decoder**(*input_dim*, *latent_dim*, *initializer*, *info=False*, *act='relu'*, *dropout=0.6*, *final_act='linear'*)

Builds a dense decoder

Parameters

- **input_shape** (*tuple*) – Shape tuple of input grids
- **latent_dim** (*int*) – Number of latent variables
- **initializer** (*tf.keras.initializers.Initializer*) – Weights initializer
- **info** (*bool, optional*) – Whether to print info. Defaults to False.
- **dropout** (*int, optional*) – Dropout factor to use. Defaults to 0.6.
- **final_act** (*str, optional*) – Dense layer activation function to use. Defaults to "linear".

Returns encoder

Return type tf.keras.Model

architectures.svdae.svdae.**build_deeper_dense_encoder**(*latent_dim*, *initializer*, *info=False*, *act='relu'*, *dropout=0.6*)

Builds a dense encoder

Parameters

- **input_shape** (*tuple*) – Shape tuple of input grids
- **latent_dim** (*int*) – Number of latent variables
- **initializer** (*tf.keras.initializers.Initializer*) – Weights initializer
- **info** (*bool, optional*) – Whether to print info. Defaults to False.
- **dropout** (*int, optional*) – Dropout factor to use. Defaults to 0.6.

Returns encoder

Return type tf.keras.Model

architectures.svdae.svdae.**build_dense_decoder**(*input_dim*, *latent_dim*, *initializer*, *info=False*, *act='relu'*, *dropout=0.6*, *final_act='linear'*)

Builds a dense decoder

Parameters

- **input_shape** (*tuple*) – Shape tuple of input grids
- **latent_dim** (*int*) – Number of latent variables
- **initializer** (*tf.keras.initializers.Initializer*) – Weights initializer
- **info** (*bool, optional*) – Whether to print info. Defaults to False.

- **dropout** (*int, optional*) – Dropout factor to use. Defaults to 0.6.
- **final_act** (*str, optional*) – Dense layer activation function to use. Defaults to "linear".

Returns encoder

Return type tf.keras.Model

architectures.svdae.svdae.**build_dense_encoder**(*latent_dim*, *initializer*, *info=False*, *act='relu'*, *dropout=0.6*)

Builds a dense encoder

Parameters

- **input_shape** (*tuple*) – Shape tuple of input grids
- **latent_dim** (*int*) – Number of latent variables
- **initializer** (*tf.keras.initializers.Initializer*) – Weights initializer
- **info** (*bool, optional*) – Whether to print info. Defaults to False.
- **dropout** (*int, optional*) – Dropout factor to use. Defaults to 0.6.

Returns encoder

Return type tf.keras.Model

architectures.svdae.svdae.**build_slimmer_dense_decoder**(*input_dim*, *latent_dim*, *initializer*, *info=False*, *act='relu'*, *dropout=0.6*, *final_act='linear'*)

Builds a dense decoder

Parameters

- **input_shape** (*tuple*) – Shape tuple of input grids
- **latent_dim** (*int*) – Number of latent variables
- **initializer** (*tf.keras.initializers.Initializer*) – Weights initializer
- **info** (*bool, optional*) – Whether to print info. Defaults to False.
- **dropout** (*int, optional*) – Dropout factor to use. Defaults to 0.6.
- **final_act** (*str, optional*) – Dense layer activation function to use. Defaults to "linear".

Returns encoder

Return type tf.keras.Model

architectures.svdae.svdae.**build_slimmer_dense_encoder**(*latent_dim*, *initializer*, *info=False*, *act='relu'*, *dropout=0.6*)

Builds a dense encoder

Parameters

- **input_shape** (*tuple*) – Shape tuple of input grids
- **latent_dim** (*int*) – Number of latent variables
- **initializer** (*tf.keras.initializers.Initializer*) – Weights initializer
- **info** (*bool, optional*) – Whether to print info. Defaults to False.

- **dropout** (*int, optional*) – Dropout factor to use. Defaults to 0.6.

Returns encoder

Return type tf.keras.Model

architectures.svdae.svdae.**build_slimmer_vinicius_encoder_decoder**(*input_dim*, *latent_dim*, *initializer*, *info=False*, *act='elu'*, *dense_act='elu'*, *dropout=0.6*, *reg=0.001*, *batchnorm=True*, *final_act='linear'*)

This encoder-decoder pair currently works for 60 by 20 by 20 grids

Parameters

- **input_shape** (*tuple*) – Shape tuple of input grids
- **latent_dim** (*int*) – Number of latent variables
- **initializer** (*tf.keras.initializers.Initializer*) – Weights initializer
- **info** (*bool, optional*) – Whether to print info. Defaults to False.
- **act** (*str, optional*) – Activation function to use. Defaults to "elu".
- **dense_act** (*str, optional*) – Dense layer activation function to use. Defaults to None.
- **dropout** (*float*) – Dropout factor to use in dense layers.
- **reg** (*float*) – Level of weights regularization to use.
- **batchnorm** (*bool, optional*) – Whether to use batch normalization layers. Defaults to True.
- **final_act** (*str, optional*) – Dense layer activation function to use. Defaults to "linear".

Returns encoder, decoder pair

Return type tuple

architectures.svdae.svdae.**build_smaller_vinicius_encoder_decoder**(*input_dim*, *latent_dim*, *initializer*, *info=False*, *act='elu'*, *dense_act='elu'*, *dropout=0.6*, *reg=0.001*, *batchnorm=True*, *final_act='linear'*)

This encoder-decoder pair currently works for 60 by 20 by 20 grids

Parameters

- **input_shape** (*tuple*) – Shape tuple of input grids
- **latent_dim** (*int*) – Number of latent variables
- **initializer** (*tf.keras.initializers.Initializer*) – Weights initializer
- **info** (*bool, optional*) – Whether to print info. Defaults to False.
- **act** (*str, optional*) – Activation function to use. Defaults to "elu".
- **dense_act** (*str, optional*) – Dense layer activation function to use. Defaults to None.
- **dropout** (*float*) – Dropout factor to use in dense layers.
- **reg** (*float*) – Level of weights regularization to use.
- **batchnorm** (*bool, optional*) – Whether to use batch normalization layers. Defaults to True.
- **final_act** (*str, optional*) – Dense layer activation function to use. Defaults to "linear".

Returns encoder, decoder pair

Return type tuple

architectures.svdae.svdae.**build_vinicius_encoder_decoder**(*input_dim*, *latent_dim*, *initializer*, *info=False*, *act='elu'*, *dense_act='elu'*, *dropout=0.6*, *reg=0.001*, *batchnorm=True*, *final_act='linear'*)

This encoder-decoder pair currently works for 60 by 20 by 20 grids

Parameters

- **input_shape** (*tuple*) – Shape tuple of input grids
- **latent_dim** (*int*) – Number of latent variables
- **initializer** (*tf.keras.initializers.Initializer*) – Weights initializer
- **info** (*bool, optional*) – Whether to print info. Defaults to False.
- **act** (*str, optional*) – Activation function to use. Defaults to "elu".
- **dense_act** (*str, optional*) – Dense layer activation function to use. Defaults to None.
- **dropout** (*float*) – Dropout factor to use in dense layers.
- **reg** (*float*) – Level of weights regularization to use.
- **batchnorm** (*bool, optional*) – Whether to use batch normalization layers. Defaults to True.
- **final_act** (*str, optional*) – Dense layer activation function to use. Defaults to "linear".

Returns encoder, decoder pair

Return type tuple

architectures.svdae.svdae.**build_wider_conv_encoder_decoder**(*input_dim*, *latent_dim*, *initializer*, *info=False*, *act='relu'*, *dense_act='relu'*, *dropout=0.6*, *final_act='linear'*)

Create a 1D convolutional encoder and decoder

Parameters

- **input_dim** (*tuple*) – Shape tuple of input grids
- **latent_dim** (*int*) – Number of latent variables
- **initializer** (*tf.keras.initializers.Initializer*) – Weights initializer
- **info** (*bool, optional*) – Whether to print info. Defaults to False.
- **act** (*str, optional*) – Activation function to use. Defaults to "elu".
- **dense_act** (*str, optional*) – Dense layer activation function to use. Defaults to None.
- **dropout** (*int, optional*) – Dropout factor to use. Defaults to 0.6.
- **final_act** (*str, optional*) – Dense layer activation function to use. Defaults to "linear".

Returns encoder, decoder pair

Return type tuple

architectures.svdae.svdae.**build_wider_dense_decoder**(*input_dim*, *latent_dim*, *initializer*, *info=False*, *act='relu'*, *dropout=0.6*, *final_act='linear'*)

Builds a dense decoder

Parameters

- **input_shape** (*tuple*) – Shape tuple of input grids
- **latent_dim** (*int*) – Number of latent variables
- **initializer** (*tf.keras.initializers.Initializer*) – Weights initializer
- **info** (*bool, optional*) – Whether to print info. Defaults to False.
- **dropout** (*int, optional*) – Dropout factor to use. Defaults to 0.6.
- **final_act** (*str, optional*) – Dense layer activation function to use. Defaults to "linear".

Returns encoder

Return type tf.keras.Model

architectures.svdae.svdae.**build_wider_dense_encoder**(*latent_dim*, *initializer*, *info=False*, *act='relu'*, *dropout=0.6*)

Builds a dense encoder

Parameters

- **input_shape** (*tuple*) – Shape tuple of input grids
- **latent_dim** (*int*) – Number of latent variables

- **initializer** (*tf.keras.initializers.Initializer*) – Weights initializer
- **info** (*bool, optional*) – Whether to print info. Defaults to False.
- **dropout** (*int, optional*) – Dropout factor to use. Defaults to 0.6.

Returns encoder

Return type tf.keras.Model

PYTHON MODULE INDEX

a

architectures.cae.D2.cae, 11
architectures.cae.D3.cae, 14
architectures.discriminators.discriminators, 17
architectures.svdae.svdae, 18

m

models.aae, 1
models.cae, 3
models.svdae, 4

p

preprocessing.utils, 9

u

utils, 9

w

wandb.train_wandb_fpc, 7
wandb.train_wandb_sf, 7

INDEX

A

AAE (*class in models.aae*), 1
AAE_combined_loss (*class in models.aae*), 2
architectures.cae.D2.cae (*module*), 11
architectures.cae.D3.cae (*module*), 14
architectures.discriminators.discriminators (*module*), 17
architectures.svdae.svdae (*module*), 18

B

build_agostini_encoder_decoder() (*in module architectures.cae.D2.cae*), 11
build_conv_encoder_decoder() (*in module architectures.svdae.svdae*), 18
build_custom_conv_decoder() (*in module architectures.cae.D2.cae*), 11
build_custom_conv_encoder() (*in module architectures.cae.D2.cae*), 11
build_custom_discriminator() (*in module architectures.discriminators.discriminators*), 17
build_custom_wider_discriminator() (*in module architectures.discriminators.discriminators*), 18
build_deeper_dense_decoder() (*in module architectures.svdae.svdae*), 19
build_deeper_dense_encoder() (*in module architectures.svdae.svdae*), 19
build_deeper_omata_encoder_decoder() (*in module architectures.cae.D2.cae*), 12
build_deeper_omata_encoder_decoder() (*in module architectures.cae.D3.cae*), 14
build_dense_decoder() (*in module architectures.svdae.svdae*), 19
build_dense_encoder() (*in module architectures.svdae.svdae*), 20
build_denser_omata_encoder_decoder() (*in module architectures.cae.D2.cae*), 12
build_denser_omata_encoder_decoder() (*in module architectures.cae.D3.cae*), 15
build_densest_omata_encoder_decoder() (*in module architectures.cae.D2.cae*), 13
build_densest_omata_encoder_decoder() (*in module architectures.cae.D3.cae*), 15
build_densest_thinner_omata_encoder_decoder() (*in module architectures.cae.D3.cae*), 16
build_mnist_wide_omata_encoder_decoder() (*in module architectures.cae.D2.cae*), 13
build_omata_encoder_decoder() (*in module architectures.cae.D2.cae*), 13
build_omata_encoder_decoder() (*in module architectures.cae.D3.cae*), 16
build_slimmer_dense_decoder() (*in module architectures.svdae.svdae*), 20
build_slimmer_dense_encoder() (*in module architectures.svdae.svdae*), 20
build_slimmer_vinicius_encoder_decoder() (*in module architectures.svdae.svdae*), 21
build_smaller_vinicius_encoder_decoder() (*in module architectures.svdae.svdae*), 21
build_vinicius_encoder_decoder() (*in module architectures.svdae.svdae*), 22
build_wide_omata_encoder_decoder() (*in module architectures.cae.D2.cae*), 14
build_wide_omata_encoder_decoder() (*in module architectures.cae.D3.cae*), 17
build_wider_conv_encoder_decoder() (*in module architectures.svdae.svdae*), 22
build_wider_dense_decoder() (*in module architectures.svdae.svdae*), 23
build_wider_dense_encoder() (*in module architectures.svdae.svdae*), 23
build_wider_omata_encoder_decoder() (*in module architectures.cae.D2.cae*), 14

C

CAE (*class in models.cae*), 3
calc_pod() (*in module utils*), 9
calc_pod() (*models.svdae.SVDAE method*), 4
compile() (*models.aae.AAE method*), 1
compile() (*models.aae.AAE_combined_loss method*), 2
compile() (*models.cae.CAE method*), 3
compile() (*models.svdae.SVDAE method*), 4

convert_2d() (*in module preprocessing.utils*), 9

M

models.aae (*module*), 1
models.cae (*module*), 3
models.svdae (*module*), 4
mse_PI (*class in utils*), 9
mse_weighted (*class in utils*), 9

P

plot_losses() (*in module models.aae*), 2
plot_losses() (*in module models.svdae*), 5
predict() (*models.cae.CAE method*), 3
predict() (*models.svdae.SVDAE method*), 4
predict_single() (*models.svdae.SVDAE method*), 4
preprocessing.utils (*module*), 9
print_losses() (*in module models.aae*), 2
print_losses() (*in module models.svdae*), 5

R

reconstruct_from_pod() (*models.svdae.SVDAE method*), 5
reconstruct_pod() (*in module utils*), 9

S

SVDAE (*class in models.svdae*), 4

T

train() (*models.aae.AAE method*), 1
train() (*models.aae.AAE_combined_loss method*), 2
train() (*models.cae.CAE method*), 3
train() (*models.svdae.SVDAE method*), 5
train_generate() (*models.cae.CAE method*), 3
train_wandb_aae() (*in module wandb.train_wandb_fpc*), 7
train_wandb_aae() (*in module wandb.train_wandb_sf*), 7
train_wandb_cae() (*in module wandb.train_wandb_fpc*), 7
train_wandb_cae() (*in module wandb.train_wandb_sf*), 7
train_wandb_svdae() (*in module wandb.train_wandb_fpc*), 7
train_wandb_svdae() (*in module wandb.train_wandb_sf*), 8

U

utils (*module*), 9

V

validate() (*models.aae.AAE method*), 1
validate() (*models.aae.AAE_combined_loss method*), 2
validate() (*models.cae.CAE method*), 3
validate() (*models.svdae.SVDAE method*), 5

W

wandb.train_wandb_fpc (*module*), 7
wandb.train_wandb_sf (*module*), 7