Wildfire

Group Rush

May 26, 2023

CONTENTS:

class `model.dataloader.`**`ConvolutionalDataset`**(*data*, *transform=None*)

class `model.dataloader.`**`LatentDataset`**(*data*, *transform=None*, *timesteps=10*)

class `model.dataloader.`**`VariationalDataset`**(*dataset*, *batch_size=100*, *sequence_length=2*)

Dataset for variational auto encoder.

Args: - dataset (ndarray): The input data as a NumPy array. - batch_size (int): The size of each batch (default: 100). - sequence_length (int): The length of the image sequence (default: 2).

Returns: - current_image (ndarray): The current image at index *idx*. - next_image (ndarray): The next image at index *idx + 1*.

`model.dataloader.`**`load_data`**(*dataset_path='train'*, *model=None*, *timesteps=10*, *batch_size=32*, *shuffle=False*, *normalise=False*)

Creates a torch.utils.data.Dataset based on the model provided.

Parameters

- **`dataset_path`** (*`str, optional`*) – _description_. Defaults to 'train'.
- **`model`** (*`_type_, optional`*) – _description_. Defaults to None.
- **`timesteps`** (*`int, optional`*) – _description_. Defaults to 10.
- **`batch_size`** (*`int, optional`*) – _description_. Defaults to 32.
- **`shuffle`** (*`bool, optional`*) – _description_. Defaults to False.
- **`normalise`** (*`bool, optional`*) – _description_. Defaults to False.

Raises

- **`ValueError`** – _description_
- **`FileNotFoundError`** – _description_
- **`ValueError`** – _description_
- **`NotImplementedError`** – _description_

Returns

description

Return type

type

`model.dataloader.`**`normalize_transform`**(*sample*)

Normalizes the sample to have a mean of 0.5 and standard deviation of 0.5

Parameters

`sample` (*`torch.Tensor`*) – The sample to normalize

Returns

The normalized sample

Return type

torch.Tensor

class `model.CVAE.`**`CustomDataset`**(*dataset*, *batch_size=100*, *sequence_length=2*)

Custom dataset class for working with image data.

Args: - dataset (ndarray): The input data as a NumPy array. - batch_size (int): The size of each batch (default: 100). - sequence_length (int): The length of the image sequence (default: 2).

Returns: - current_image (ndarray): The current image at index *idx*. - next_image (ndarray): The next image at index *idx + 1*.

`model.CVAE.`**`KalmanGain`**`(`*B*, *H*, *R*`)`

Compute the Kalman gain matrix.

Parameters

- **B** (*np.ndarray*) – Covariance matrix of the predicted state estimate with shape (n, n), where n is the state dimension.
- **H** (*np.ndarray*) – Observation matrix with shape (m, n), where m is the measurement dimension and n is the state dimension.
- **R** (*np.ndarray*) – Measurement noise covariance matrix with shape (m, m), where m is the measurement dimension.

Returns

Kalman gain matrix with shape (n, m), where n is the state dimension and m is the measurement dimension.

Return type

np.ndarray

`class model.CVAE.`**`VAE_Decoder`**

`forward``(`*z*`)`

Forward pass of the VAE Decoder.

Parameters

z (*torch.Tensor*) – Input tensor to the decoder.

Returns

Decoded output tensor.

Return type

torch.Tensor

`class model.CVAE.`**`VAE_Encoder`**

`forward``(`*x*`)`

Forward pass of the VAE Encoder.

Parameters

x (*torch.Tensor*) – Input tensor to the encoder.

Returns

Encoded output tensor.

Return type

torch.Tensor

`class model.CVAE.`**`VariationalAutoencoder`**`(`*dims_latent*`)`

`forward``(`*x*, *prev_x*`)`

Defines the computation performed at every call.

Should be overridden by all subclasses.

Note: Although the recipe for forward pass needs to be defined within this function, one should call the `Module` instance afterwards instead of this since the former takes care of running the registered hooks while the latter silently ignores them.

`model.CVAE.`**`covariance_matrix`**`(`*X*`)`

Compute the covariance matrix of the input data.

Parameters

`X` (*`np.ndarray`*) – Input data with shape (n, m), where n is the number of samples and m is the number of features.

Returns

Covariance matrix of the input data with shape (m, m).

Return type

np.ndarray

`model.CVAE.`**`data_assimilation`**`(`*B*, *H*, *R*, *model_compr*, *satellite_compr*`)`

Perform data assimilation using the Kalman filter.

Parameters

- **`B`** (*`np.ndarray`*) – Background covariance matrix with shape (n, n), where n is the state dimension.
- **`H`** (*`np.ndarray`*) – Observation matrix with shape (m, n), where m is the measurement dimension and n is the state dimension.
- **`R`** (*`np.ndarray`*) – Measurement noise covariance matrix with shape (m, m), where m is the measurement dimension.
- **`model_compr`** (*`np.ndarray`*) – Compressed model data with shape (n, t), where n is the state dimension and t is the number of time steps.
- **`satellite_compr`** (*`np.ndarray`*) – Compressed satellite data with shape (m, t), where m is the measurement dimension and t is the number of time steps.

Returns

Updated data array after assimilation with shape (n, t), where n is the state dimension and t is the number of time steps.

Return type

np.ndarray

`model.CVAE.`**`load_data`**`(`*path*`)`

Load data from a specified path.

Parameters: - path (str): The file path to the data file.

Returns: - data (ndarray): The loaded data as a NumPy array.

`model.CVAE.`**`load_model`**`(`*path*, *ModelClass*, *device*`)`

Load the model from the specified path and move it to the specified device.

Parameters

- **`path`** (*`str`*) – File path to load the model.
- **`ModelClass`** (*`nn.Module`*) – Model class to instantiate.
- **`device`** (*`str`*) – Device to move the model to (e.g., ‘cpu’, ‘cuda’).

Returns

Loaded model.

Return type
nn.Module

`model.CVAE.`**`make_dataloader`**`(`*MyDataset*, *batch_size*, *shuffle=True*`)`

Create a data loader for the given dataset.

Parameters: - dataset (Dataset): The dataset object to create the data loader from. - batch_size (int): The batch size for the data loader.

Returns: - dataloader (DataLoader): The created data loader.

`model.CVAE.`**`make_forecast`**`(`*input*, *model*`)`

Generate forecasts for multiple input data using the specified model.

Parameters

- **`input`** (*`list`*) – Instance of the CustomDataset class.
- **`model`** (*`nn.Module`*) – Trained VAE model.

Returns
Array of forecasted images.

Return type
np.ndarray

`model.CVAE.`**`make_single_forecast`**`(`*input*, *model*`)`

Generate a single forecast using the input data and the specified model.

Parameters

- **`input`** (*`tuple`*) – Single image as numpy array.
- **`model`** (*`nn.Module`*) – Trained VAE model.

Returns
Forecasted image.

Return type
np.ndarray

`model.CVAE.`**`make_single_image_dataset`**`(`*image*`)`

Convert a single image into a dataset suitable for model input.

Parameters
`image` (*`np.ndarray`*) – Single image array of shape (256, 256).

Returns
Dataset object containing the single image.

Return type
CustomDataset

`model.CVAE.`**`make_tensor`**`(`*data*, *device*`)`

Convert a NumPy array to a PyTorch tensor and move it to the specified device.

Parameters: - data_1D (ndarray): The input array to be converted to a tensor. - device (torch.device): The device to which the tensor should be moved.

Returns: - tensor (torch.Tensor): The converted tensor.

`model.CVAE.`**`mse`**`(`*y_obs*, *y_pred*`)`

Calculate the mean squared error (MSE) between observed values and predicted values.

Parameters

- **y_obs** (*array-like*) – Array of observed values.
- **y_pred** (*array-like*) – Array of predicted values.

Returns
Mean squared error (MSE) between y_obs and y_pred.

Return type
float

`model.CVAE.`**`reconstruct`**`(`*data_compr*, *model*`)`

Reconstruct the compressed data using the specified VAE model.

Parameters

- **data_compr** (*np.ndarray*) – Compressed data with shape (t, n), where t is the number of time steps and n is the feature dimension.
- **model** (VariationalAutoencoder) – VAE model used for reconstruction.

Returns
Reconstructed data with shape (t, n), where t is the number of time steps and n is the feature dimension.

Return type
np.ndarray

`model.CVAE.`**`save_model`**`(`*model*, *path*`)`

Save the state dictionary of the model to the specified path.

Parameters

- **model** (*nn.Module*) – Model to be saved.
- **path** (*str*) – File path to save the model.

Returns
None

`model.CVAE.`**`train`**`(`*autoencoder*, *data*, *device*, *epochs*, *kl_div_on=True*`)`

Train the autoencoder model using the provided data.

Parameters

- **autoencoder** (*nn.Module*) – Autoencoder model to be trained.
- **data** (*DataLoader*) – DataLoader containing the training data.
- **device** (*str*) – Device to be used for training (e.g., ‘cpu’, ‘cuda’).
- **epochs** (*int*) – Number of training epochs.
- **kl_div_on** (*bool, optional*) – Whether to include the KL divergence term in the loss. Default is True.

Returns
Trained autoencoder model and list of losses per epoch.

Return type
Tuple[nn.Module, list]

`model.CVAE.`**`update_prediction`**`(`*x*, *K*, *H*, *y*`)`

Update the prediction using the Kalman filter equations.

Parameters

- **x** (*np.ndarray*) – State estimate at the previous time step with shape (n,).
- **K** (*np.ndarray*) – Kalman gain matrix with shape (n, m), where n is the state dimension and m is the measurement dimension.
- **H** (*np.ndarray*) – Observation matrix with shape (m, n), where m is the measurement dimension and n is the state dimension.
- **y** (*np.ndarray*) – Measurement vector at the current time step with shape (m,).

Returns
Updated state estimate at the current time step with shape (n,).

Return type
np.ndarray

model.CVAE.**visualise_results**(*nDisplay*, *predictions*, *data*, *ts=None*, *seed=None*)

Visualize the results of the VAE predictions.

Parameters

- **nDisplay** (*int*) – Number of samples to display.
- **predictions** (*np.ndarray*) – Array of VAE predictions with shape (num_samples, 256, 256).
- **data** (*np.ndarray*) – Array of original data with shape (num_samples, 256, 256).
- **seed** (*int*) – Seed for random index selection.
- **ts** (*list*) – Specified index selection.

Returns
None

class model.models.**ConvolutionalAE1**(*latent_dim*)

decode(*x*)

Reconstructs the input image from a latent vector

Parameters
x (*torch.tensor*) – latent vector

Returns
reconstructed image

Return type
torch.tensor

describe()

Analysis of the model

encode(*x*)

Compresses the input image into a latent vector

Parameters
x (*torch.tensor*) – Input image

Returns
latent vector representation of the input image(s)

Return type
torch.tensor

`forward`(*x*)

Forward pass through the network

Parameters
`x` (*torch.tensor*) – input image

Returns
reconstructed image

Return type
torch.tensor

`print_model`()

Prints the model architecture

`class model.models.ConvolutionalAE2`(*latent_dim*)

`decode`(*x*)

Reconstructs the input image from a latent vector

Parameters
`x` (*torch.tensor*) – latent vector

Returns
reconstructed image

Return type
torch.tensor

`describe`()

Analysis of the model

`encode`(*x*)

Compresses the input image into a latent vector

Parameters
`x` (*torch.tensor*) – Input image

Returns
latent vector representation of the input image(s)

Return type
torch.tensor

`forward`(*x*)

Forward pass through the network

Parameters
`x` (*torch.tensor*) – input image

Returns
reconstructed image

Return type
torch.tensor

`print_model`()

Prints the model architecture

`class model.models.LSTM0`(*input_size*, *hidden_size*, *num_layers*, *output_size*)

forward(*x*)

Defines the computation performed at every call.

Should be overridden by all subclasses.

Note: Although the recipe for forward pass needs to be defined within this function, one should call the `Module` instance afterwards instead of this since the former takes care of running the registered hooks while the latter silently ignores them.

class model.models.**LSTM1**(*input_size*, *hidden_size*, *num_layers*, *output_size*)

forward(*x*)

Defines the computation performed at every call.

Should be overridden by all subclasses.

Note: Although the recipe for forward pass needs to be defined within this function, one should call the `Module` instance afterwards instead of this since the former takes care of running the registered hooks while the latter silently ignores them.

class model.models.**LinearAE**(*latent_dim*)

describe()

Analysis of the model

forward(*x*)

Forward pass through the network

Parameters

x (*torch.tensor*) – input image

Returns

reconstructed image

Return type

torch.tensor

print_model()

Prints the model architecture

class model.models.**VAE_Decoder**

forward(*z*)

Forward pass of the VAE Decoder.

Parameters

z (*torch.Tensor*) – Input tensor to the decoder.

Returns

Decoded output tensor.

Return type

torch.Tensor

class model.models.**VAE_Encoder**

`forward`(*x*)

Forward pass of the VAE Encoder.

Parameters

`x` (*torch.Tensor*) – Input tensor to the encoder.

Returns

Encoded output tensor.

Return type

torch.Tensor

`class model.models.VariationalAutoencoder`(*dims_latent*)

`describe`()

Analysis of the model

`forward`(*x*, *prev_x*)

Defines the computation performed at every call.

Should be overridden by all subclasses.

Note: Although the recipe for forward pass needs to be defined within this function, one should call the `Module` instance afterwards instead of this since the former takes care of running the registered hooks while the latter silently ignores them.

`print_model`()

Prints the model architecture

`model.models.load_encoder`(*model_name*, *latent_dim=64*, *device=device(type='cpu')*)

Returns the autoencoder model based on the model name

Parameters

- **model_name** (*str*) – name of the model to be used
- **latent_dim** (*int*) – latent dimension of the model
- **device** (*str*) – device to be used for training/inference

Returns

model to be use with weights loaded

Return type

model (nn.Module)

`model.models.load_loss_function`(*loss_name*)

Returns the loss function based on the loss name

Parameters

loss_name (*str*) – name of the loss function to be used

Returns

loss function to be used

Return type

loss_fn (nn.Module)

`model.models.`**`load_predictor`**`(`*model_name*, *model_encoder*, *input_size*, *output_size*, *hidden_size=128*, *num_layers=2*, *device=device(type='cpu')*`)`

Returns a lstm model based on the model name

Parameters

- **`model_name`** (*`str`*) – name of the model to be used
- **`input_size`** (*`int`*) – input size of the model
- **`hidden_size`** (*`int`*) – hidden size of the model
- **`num_layers`** (*`int`*) – number of layers in the model
- **`output_size`** (*`int`*) – output size of the model
- **`device`** (*`str`*) – device to be used for training/inference

Returns

model to be use with weights loaded

Return type

model (nn.Module)

`model.models.`**`perceptual_loss`**`(`*output_image*, *input_image*`)`

Calculates the perceptual loss between the output and input images

Parameters

- **`output_image`** (*`torch.tensor`*) – output image
- **`input_image`** (*`torch.tensor`*) – input image

Returns

perceptual loss

Return type

torch.tensor

`model.models.`**`train_model`**`(`*epochs*, *model*, *data_loader*, *learning_rate*, *loss_function*, *device*, *kl_div_on=False*`)`

Trains the model for the given number of epochs

Parameters

- **`epochs`** (*`int`*) – number of epochs to train the model
- **`model`** (*`nn.Module`*) – model to be trained
- **`data_loader`** (*`torch.utils.data.DataLoader`*) – data loader for the training data
- **`learning_rate`** (*`float`*) – learning rate for the optimizer
- **`loss_function`** (*`nn.Module`*) – loss function to be used
- **`device`** (*`str`*) – device to be used for training
- **`kl_div_on`** (*`bool`*) – whether to use KL divergence or not

Returns

list of losses for each epoch

Return type

losses (list)

`model.test_CVAE.`**`test_CustomDataset`**`()`

Check that the return length is the same as the array length and the return type is tensor model.test_CVAE.**test_KalmanGain**()

Check that the KalmanGain function produces the correct result

model.test_CVAE.**test_VariationalAutoencoder**()

Check that the class instansiation of the VAE

model.test_CVAE.**test_covariance_matrix**()

Test that covariance is being calculated correctly using known test case

model.test_CVAE.**test_data_assimiliation**()

Check that the data assimilation runs

model.test_CVAE.**test_load_data**()

Test that output is a numpy array

model.test_CVAE.**test_make_dataloader**()

Check that the return type is a torch dataloader

model.test_CVAE.**test_make_forecast**()

Check that the make forecast function runs and produces a numpy array

model.test_CVAE.**test_make_single_forecast**()

Check that the make single forecast function runs and produces a numpy array

model.test_CVAE.**test_make_single_image_dataset**()

Test that the correct custom dataset has been created for the input image

model.test_CVAE.**test_make_tensor**()

Check that the return type is a tensor

model.test_CVAE.**test_mse**()

Check that when both arrays are identical the MSE returns 0

model.test_CVAE.**test_reconstruct**()

Check that the reconstruction of compressed data runs

model.test_CVAE.**test_save_and_load_model**()

Check that the save and load model functions run

model.test_CVAE.**test_train**()

Check that the train function runs

model.test_CVAE.**test_update_prediction**()

Check that the update_prediction function returns the original array if the two arrays are equal

model.test_CVAE.**test_visualise_results**()

Check that the visualise results function runs

model.test_lstm.**test_load_data**()

Tests the load_data function

model.test_lstm.**test_load_encoder**()

Tests the load_encoder function

model.test_lstm.**test_load_loss_function**()

Tests the load_loss_function function

`model.test_lstm.`**`test_load_predictor`**`()`

Tests the load_lstm function

`model.test_lstm.`**`test_normalize_transform`**`()`

Tests the normalize_transform function

CHAPTER

ONE

INDICES AND TABLES

- genindex
- modindex
- search

PYTHON MODULE INDEX

INDEX