andrew

unknown

May 27, 2022

CONTENTS

CHAPTER

ONE

ANDREW'S MODULE AND FUNCTIONS

This package implements LSTM and Convolutional LSTM with PCA

See :*tools* folder for more information.

CHAPTER

TWO

DATAPROCESSING.PY

`tools.dataprocessing.`**`Load_data_to_test`**(*test_source*, *download_from*)

The function takes in the location of the download directory where we installed all the data and the test_source directs us to the location of the data. The function loads all of this data and put them together in a dataframe and returns it

Parameters

- **`test_source`** (*`string`*) –
- **`download_from`** (*`string`*) –

Returns

- *pandas dataframe that merges all these files*
- *together*

Examples

```
>>> test_source = 'some_invalid_paths'
>>> download_from = './'
>>> failed_load = Load_data_to_test(test_source, download_from)
>>> failed_load
"Can't find the dataset, please set the directory again!"
```

Notes

See https://nbviewer.org/github/radiantearth/mlhub-tutorials/blob/main/notebooks/NASA%20Tropical%20Storm%20Wind%20Speed%20Challenge/nasa-tropical-storm-wind-speed-challenge-getting-started.ipynb for further details.

`tools.dataprocessing.`**`Load_data_to_train`**(*train_source*, *train_labels*, *download_from*)

The function takes in the location of the download directory where we installed all the data and the train_source directs us to the location of the data and train_labels directs us to the wind_speed label. The function loads all of this data and put them together in a dataframe and returns it.

Parameters

- **`train_source`** (*`string`*) –
- **`train_labels`** (*`string`*) –
- **`download_from`** (*`string`*) –

Returns

- *pandas dataframe that merges all these files*
- *together*

Examples

```
>>> train_source = 'some_invalid_paths'
>>> train_labels = "some_invalid_paths"
>>> download_from = './'
>>> failed_load = Load_data_to_train(train_source, train_labels, download_from)
>>> failed_load
"Can't find the dataset, please set the directory again!"
```

Notes

See https://nbviewer.org/github/radiantearth/mlhub-tutorials/blob/main/notebooks/NASA%20Tropical%20Storm%20Wind%20Speed%20Challenge/nasa-tropical-storm-wind-speed-challenge-getting-started.ipynb for further details.

`tools.dataprocessing.`**`Loader_to_1D_array`**(*loader*, *dimension*)

given a data loader (ie. train, test, validation) and the dimension of 1D array return 3 arrays containing the images, the labels(wind speed) and the image_id

Parameters

- **`loader`** (*`torch.utils.data.dataloader.DataLoader`*) –
- **`dimension`** (*`float/int`*) –

Return type return three lists

`tools.dataprocessing.`**`SplitData`**(*x_full_id*, *y_full_id*, *train_size*, *val_size*, *test_size*, *cleaning_class*, *old_mean*, *old_std*)

Given the dataframe for training and labelling (wind speed), the sizes of training, testing, validation data and the class for cleaning the data with the original mean and standard deviation. The data is splitted into the full dataset, training dataset, validation and testing dataset as

Parameters

- **`x_full_id`** (*`pd.DataFrame`*) –
- **`y_full_id`** (*`pd.DataFrame`*) –
- **`train_size`** (*`float/int`*) –
- **`val_size`** (*`float/int`*) –
- **`test_size`** (*`float/int`*) –
- **`cleaning_class`** (*`float/int`*) –
- **`old_mean`** (*`float/int`*) –
- **`old_std`** (*`float/int`*) –

Return type return two torch tensors one which will be trained and the other for prediction `tools.dataprocessing.`**`create_inout_seq`**`(`*input_data*, *tw*`)`

given a list of input_data and the future prediction number, we return the torch tensors of training data that takes tw datapoints and the corresponding labels which takes the next datapoint which will be used for prediction.

Parameters

- **`input_data`** (*`list`*) –
- **`tw`** (*`float/int`*) –

Return type return two torch tensors one which will be trained and the other for prediction

Notes

see https://stackabuse.com/time-series-prediction-using-lstm-with-pytorch-in-python/ for further details

`tools.dataprocessing.`**`data_storm_id`**`(`*full_df*, *id*`)`

given a dataframe and a storm id as a string, return the dataframe with the corresponding ID

Parameters

- **`full_df`** (*`pd.DataFrame`*) –
- **`id`** (*`string`*) –

Return type return a pandas dataframe with corresponding ID

Examples

```
>>> df1 = pd.DataFrame({"A": [5, 2], "B": [4, 7], "Z": [9, 3]})
>>> failed_datastormid = data_storm_id(df1, 2)
>>> failed_datastormid
"Storm ID doesn't exist in dataframe"
```

`tools.dataprocessing.`**`manipdata`**`(`*train_df*, *test_df*`)`

The function takes in training and testing data as a dataframe. Then, combines training and testing data, add filename as a column with the suitable path depending on the image id and adds the number of images per storm as the final column. The whole dataframe is returned

Parameters

- **`train_df`** (*`pd.DataFrame`*) –
- **`test_df`** (*`pd.DataFrame`*) –

Returns

- *pandas dataframe that merges all these files*
- *together*

Examples

```
>>> df1 = pd.DataFrame({"A": [5, 2], "B": [4, 7], "Z": [9, 3]})
>>> df2 = pd.DataFrame({"A": [1, 3], "B": [1, 9], "Z": [29, 30]})
>>> failed_manip = manipdata(df1, df2)
>>> failed_manip
"Image ID doesn't exist in dataframe"
```

```
>>> df3 = pd.DataFrame({"Image ID": ["5", "2"], "B": [4, 7], "C": [9, 3]})
>>> df4 = pd.DataFrame({"Image ID": ["1", "3"], "B": [1, 9], "C": [29, 30]})
>>> failed_manip2 = manipdata(df3, df4)
>>> failed_manip2
"Storm ID doesn't exist in dataframe"
```

Notes

See https://nbviewer.org/github/radiantearth/mlhub-tutorials/blob/main/notebooks/NASA%20Tropical%20Storm%20Wind%20Speed%20Challenge/nasa-tropical-storm-wind-speed-challenge-getting-started.ipynb for further details.

CHAPTER

THREE

FC_LSTM.PY

`tools.fc_lstm.`**`evaluate`**(*model*, *data_loader*)

Evaluate the model on test data_loader

Parameters

- **`model`** (*`nn.Module`*) –
- **`dataLoader`** (*`torch.utils.data.DataLoader`*) –

Return type The expected resultant output of model and ground truth image

Examples

```
>>> import torch
>>> import torch.nn as nn
>>> import tools.fc_lstm as fclstm
>>> from torch.utils.data import TensorDataset, DataLoader
>>> train = torch.stack([torch.randn((1,128*128))]*10)
>>> test = torch.stack([torch.randn((128*128))]*10)
>>> data = TensorDataset(train,test)
>>> dataloader = DataLoader(data, batch_size=1,shuffle=True, num_workers=0)
>>> model = fclstm.LSTM(128*128,150,128*128)
>>> result_1,result_2 = fclstm.evaluate(model,dataloader)
>>> result_1.size != 0
True
>>> result_2.size != 0
True
```

`tools.fc_lstm.`**`set_seed`**(*seed*)

Set a random seed for training

Parameters **`seed`** (*`int`*) –

Return type None

Examples

```
>>> import tools.fc_lstm as fclstm
>>> fclstm.set_seed(42)
True
```

`tools.fc_lstm.`**`train_lstm_SSIM`**(*model*, *optimizer*, *criterion*, *data_loader*)

Train the model with SSIM loss function

Parameters

- **`model`** (*`nn.Module`*) –
- **`optimizer`** (*`torch.optim.*`*) –
- **`criterion`** (*`torch.nn.MSELoss()`*) –
- **`dataLoader`** (*`torch.utils.data.DataLoader`*) –

Return type The total training loss after training (float)

Examples

```
>>> import torch
>>> import torch.nn as nn
>>> import tools.fc_lstm as fclstm
>>> from torch.utils.data import TensorDataset, DataLoader
>>> from pytorch_msssim import ssim
>>> train = torch.stack([torch.randn((1,128*128))]*10)
>>> test = torch.stack([torch.randn((128*128))]*10)
>>> data = TensorDataset(train,test)
>>> dataloader = DataLoader(data, batch_size=1,shuffle=True, num_workers=0)
>>> model = fclstm.LSTM(128*128,150,128*128)
>>> optimizer = torch.optim.Adam(model.parameters(), lr=0.01)
>>> criterion = ssim
>>> result = fclstm.train_lstm_SSIM(model, optimizer, criterion, dataloader)
>>> result != 0
tensor(True)
```

`tools.fc_lstm.`**`train_lstm_mse`**(*model*, *optimizer*, *criterion*, *data_loader*)

Train the model with MSE loss function

Parameters

- **`model`** (*`nn.Module`*) –
- **`optimizer`** (*`torch.optim.*`*) –
- **`criterion`** (*`torch.nn.MSELoss()`*) –
- **`dataLoader`** (*`torch.utils.data.DataLoader`*) –

Return type The total training loss after training (float)

Examples

```
>>> import torch
>>> import torch.nn as nn
>>> import tools.fc_lstm as fclstm
>>> from torch.utils.data import TensorDataset, DataLoader
>>> train = torch.stack([torch.randn((1,128*128))]*10)
>>> test = torch.stack([torch.randn((128*128))]*10)
>>> data = TensorDataset(train,test)
>>> dataloader = DataLoader(data, batch_size=1,shuffle=True, num_workers=0)
>>> model = fclstm.LSTM(128*128,150,128*128)
>>> optimizer = torch.optim.Adam(model.parameters(), lr=0.01)
>>> criterion = nn.MSELoss()
>>> result = fclstm.train_lstm_mse(model, optimizer, criterion, dataloader)
>>> result != 0
tensor(True)
```

`tools.fc_lstm.`**`train_mse`**(*seed*, *epochs*, *batch_size*, *model*, *criterion*, *optimizer*, *train_loader*, *val_loader*, *test_loader*)

Train epochs with MSE loss function

Parameters

- **seed** (*random seed (int)*) –
- **epochs** (*int*) –
- **batch_size** (*int*) –
- **model** (*nn.Module*) –
- **criterion** (*nn.MSELoss()*) –
- **optimizer** (*torch.optim.**) –
- **train_loader** (*torch.utils.data.DataLoader*) –
- **val_loader** (*torch.utils.data.DataLoader*) –
- **test_loader** (*torch.utils.data.DataLoader*) –

Return type bool

Examples

```
>>> import torch
>>> import torch.nn as nn
>>> import tools.fc_lstm as fclstm
>>> from torch.utils.data import TensorDataset, DataLoader
>>> train = torch.stack([torch.randn((1,128*128))]*10)
>>> test = torch.stack([torch.randn((128*128))]*10)
>>> data = TensorDataset(train,test)
>>> dataloader = DataLoader(data, batch_size=1,shuffle=True, num_workers=0)
>>> seed = 42
>>> model = fclstm.LSTM(128*128,150,128*128)
>>> optimizer = torch.optim.Adam(model.parameters(), lr=0.01)
>>> epochs = 10
```

(continues on next page)

(continued from previous page)

```
>>> batch_size = 1
>>> criterion = nn.MSELoss()
>>> fclstm.train_mse(seed, epochs, batch_size, model, criterion, optimizer,
↪dataloader, dataloader, dataloader)
True
```

`tools.fc_lstm.`**`train_ssim`**(*seed*, *epochs*, *batch_size*, *model*, *criterion*, *optimizer*, *train_loader*, *val_loader*, *test_loader*)

Train epochs with SSIM loss function

Parameters

- **seed** (*random seed (int)*) –
- **epochs** (*int*) –
- **batch_size** (*int*) –
- **model** (*nn.Module*) –
- **criterion** (*pytorch_msssim.ssim*) –
- **optimizer** (*torch.optim.**) –
- **train_loader** (*torch.utils.data.DataLoader*) –
- **val_loader** (*torch.utils.data.DataLoader*) –
- **test_loader** (*torch.utils.data.DataLoader*) –

Return type bool

Examples

```
>>> import torch
>>> import tools.fc_lstm as fclstm
>>> from torch.utils.data import TensorDataset, DataLoader
>>> from pytorch_msssim import ssim
>>> train = torch.stack([torch.randn((1,128*128))]*10)
>>> test = torch.stack([torch.randn((128*128))]*10)
>>> data = TensorDataset(train,test)
>>> dataloader = DataLoader(data, batch_size=1,shuffle=True, num_workers=0)
>>> seed = 42
>>> model = fclstm.LSTM(128*128,150,128*128)
>>> optimizer = torch.optim.Adam(model.parameters(), lr=0.01)
>>> epochs = 10
>>> batch_size = 1
>>> criterion = ssim
>>> fclstm.train_ssim(seed, epochs, batch_size, model, criterion, optimizer,
↪dataloader, dataloader, dataloader)
True
```

`tools.fc_lstm.`**`validate_SSIM`**(*model*, *criterion*, *data_loader*)

Validate the model with SSIM loss function

Parameters

- **model** (*nn.Module*) –

- **criterion** (*torch.nn.MSELoss()*) –
- **dataLoader** (*torch.utils.data.DataLoader*) –

Return type The total validation loss (float)

Examples

```
>>> import torch
>>> import torch.nn as nn
>>> import tools.fc_lstm as fclstm
>>> from torch.utils.data import TensorDataset, DataLoader
>>> from pytorch_msssim import ssim
>>> train = torch.stack([torch.randn((1,128*128))]*10)
>>> test = torch.stack([torch.randn((128*128))]*10)
>>> data = TensorDataset(train,test)
>>> dataloader = DataLoader(data, batch_size=1,shuffle=True, num_workers=0)
>>> model = fclstm.LSTM(128*128,150,128*128)
>>> criterion = ssim
>>> result = fclstm.validate_SSIM(model, criterion, dataloader)
>>> result != 0
tensor(True)
```

`tools.fc_lstm.`**`validate_mse`**(*model*, *criterion*, *data_loader*)

Validate the trained model with MSE loss function

Parameters

- **model** (*nn.Module*) –
- **criterion** (*torch.nn.MSELoss()*) –
- **dataLoader** (*torch.utils.data.DataLoader*) –

Return type The total validation loss (float)

Examples

```
>>> import torch
>>> import torch.nn as nn
>>> import tools.fc_lstm as fclstm
>>> from torch.utils.data import TensorDataset, DataLoader
>>> train = torch.stack([torch.randn((1,128*128))]*10)
>>> test = torch.stack([torch.randn((128*128))]*10)
>>> data = TensorDataset(train,test)
>>> dataloader = DataLoader(data, batch_size=1,shuffle=True, num_workers=0)
>>> model = fclstm.LSTM(128*128,150,128*128)
>>> criterion = nn.MSELoss()
>>> result = fclstm.validate_mse(model, criterion, dataloader)
>>> result != 0
tensor(True)
```

CHAPTER FOUR

VISUALISATION.PY

`tools.visualisation.`**`show_batch`**(*dataset*, *nr=4*, *nc=4*)

given a dataset with type dict with three keys including "Image ID", "image", "label" number of rows and columns, The function splits out images with that number of rows and columns

Parameters

- **`dataset`** (*`dict`*) –
- **`nr`** (*`float/int`*) –
- **`nc`** (*`float/int`*) –

Return type None

Examples

```
>>> image = {"Image ID": 1, "image": [[[1,2],[3,4],[5,6]]], "label": 1}
>>> displaypic = show_batch(image)
>>> displaypic
```

CHAPTER

FIVE

METRIC.PY

tools.metric.**Generate_ssim_mse**(*y_pred*, *RealPath*)

given a dataset numpy array of the prediction image and the path to the Real Images, return two arrays containining the ssim and mse values whilst printing each values

Parameters

- **y_pred** (*numpy.array*) –
- **RealPath** (*string*) –

Return type Two arrays containing the SSIM and MSE values.

Examples

```
>>> import tools.metric as metrics
>>> wrong_path = "a_wrong_path"
>>> ssim_mse = metrics.Generate_ssim_mse([], wrong_path)
>>> ssim_mse
'Please select the right path'
```

CHAPTER

SIX

FC_LSTM_PCA.PY

`tools.fc_lstm_pca.`**`evaluate_pca`**(*model*, *data_loader*, *trained_pca*)

Evaluate the model on test data_loader for after PCA. This function will return two numpy arrays one for prediction and the other for the actual solution

Parameters

- **`model`** (*`nn.Module`*) –
- **`data_loader`** (*`torch.utils.data.DataLoader`*) –
- **`trained_pca`** (*`sklearn.decomposition.PCA`*) –

Return type Resultant numpy arrays (numpy.ndarray)

Examples

```
>>> import torch
>>> import torch.nn as nn
>>> import tools.fc_lstm as fclstm
>>> import tools.fc_lstm_pca as fclstm_pca
>>> from tools.dataprocessing import create_inout_seq_pca
>>> from torch.utils.data import TensorDataset, DataLoader
>>> from sklearn.decomposition import PCA
>>> pca_trained = PCA(n_components=93) # n_components = min(n_components,n_
↪features) - 1
>>> train = torch.stack([torch.randn((100*100))] * 100)
>>> test = torch.stack([torch.randn((100*100))] * 100)
>>> train = pca_trained.fit_transform(train)
>>> test = pca_trained.transform(test)
>>> train_data = torch.from_numpy(train).float()
>>> test_data = torch.from_numpy(test).float()
>>> test_ds,test_label = create_inout_seq_pca(test_data,1) # (99,1,93)
>>> dummy_data = TensorDataset(test_ds, test_label)
>>> dummy_loader = DataLoader(dummy_data, batch_size=1,shuffle=True, num_workers=0)
>>> model = fclstm.LSTM(93, 100, 93)
>>> y_1, y_2 = fclstm_pca.evaluate_pca(model, dummy_loader, pca_trained)
>>> y_1.size != 0
True
>>> y_2.size != 0
True
```

Return type Two numpy arrays one for output of model and the other for ground truth image

References

PYTHON MODULE INDEX