MultiBench

Release 1.0.0

-

Apr 15, 2023

GETTING STARTED

Welcome to the documentation for MultiBench/MultiZoo, a comprehensive evaluation and set of implementations of existing multimodal machine learning algorithms. It contains 20 methods spanning different methodological innovations in (1) data preprocessing, (2) fusion paradigms, (3) optimization objectives, and (4) training procedures.

CHAPTER

ONE

INSTALLATION

To install the repository, first clone the repository via Git, and then install the prerequisite packages through Conda (Linux/MacOS):

```
conda env create [-n ENVNAME] -f environment.yml
```

Windows user should check out the Windows Subsystem for Linux (WSL). From there, you should be able to try out the example scripts and the other code through running a Python kernel inside the repository folder.

CHAPTER

TWO

DOWNLOADING DATASETS

We support a variety of datasets in MultiBench out of the box, but downloading each of them requires separate steps to do so:

2.1 Affective Computing

2.1.1 MUStARD

To grab the MUStARD/sarcasm dataset, download the dataset from here.

To load it, you can then use the following snippet:

```
from datasets.affect.get_data import get_dataloader
traindata, validdata, test_robust = get_dataloader('/path/to/raw/data/file', data_type=
↪'sarcasm')
```

2.1.2 CMU-MOSI

To grab the CMU-MOSI dataset, download the dataset from here.

To load it, you can then use the following snippet:

```
from datasets.affect.get_data import get_dataloader
traindata, validdata, test_robust = get_dataloader('/path/to/raw/data/file', data_type=
↪'mosi')
```

2.1.3 UR-Funny

To grab the UR-Funny/humor dataset, download the dataset from here.

To load it, you can then use the following snippet:

```
from datasets.affect.get_data import get_dataloader
traindata, validdata, test_robust = get_dataloader('/path/to/raw/data/file', data_type=
↪'humor')
```

2.1.4 CMU-MOSEI

To grab the CMU-MOSEI dataset, download the dataset from here.

To load it, you can then use the following snippet:

```
from datasets.affect.get_data import get_dataloader
traindata, validdata, test_robust = get_dataloader('/path/to/raw/data/file', data_type=
↪'mosei')
```

2.2 Healthcare

2.2.1 MIMIC

Access to the MIMIC dataset is restricted, and as a result you will need to follow the instructions here to gain access.

Once you do so, email ylyu1@andrew.cmu.edu with proof of those credentials to gain access to the preprocessed *im.pk* file we use in our dataloaders.

To load it, you can then use the following snippet:

```
from datasets.mimic.get_data import get_dataloader
traindata, validdata, test_robust = get_dataloader(tasknum, inputed_path='/path/to/raw/
↪data/file')
```

2.3 Robotics

2.3.1 MuJoCo Push

To grab the MuJoCo Push dataset, you simply need to run any of the example experiments, like the following:

```
python examples/gentle_push/LF.py
```

This will download the dataset into the `datasets/gentle_push/cache` folder directly.

As this uses `gdown` to download the files directly, sometimes this process fails.

Should that happen, you can simply download the following files:

- Download this file and place it under `datasets/gentle_push/cache/1qmBCfsAGu8eew-CQFmV1svodl9VJa6fX-gentle_push_10.hdf5`.
- Download this file and place it under `datasets/gentle_push/cache/18dr1z0N__yFiP_DAKxy-Hs9Vy_AsaW6Q-gentle_push_300.hdf5`.
- Download this file and place it under `datasets/gentle_push/cache/1JTgmq1KPRK9HYi8BgvljKg5MPqT_N4cR-gentle_push_1000.hdf5`.

Then, you can follow this code block as an example to get the dataloaders:

```
from datasets.gentle_push.data_loader import PushTask
Task = PushTask
modalities = ['control']

# Parse args
parser = argparse.ArgumentParser()
Task.add_dataset_arguments(parser)
args = parser.parse_args()
dataset_args = Task.get_dataset_args(args)

fannypack.data.set_cache_path('datasets/gentle_push/cache')

train_loader, val_loader, test_loader = Task.get_dataloader(
    16, modalities, batch_size=32, drop_last=True)
```

2.3.2 Vision&Touch

To grab the Vision&Touch dataset, please run the `download_data.sh` file located under `dataset/robotics/download_data.sh`.

This dataset, by default, only has the training and validation dataset, which you can access through the following call:

```
from datasets.robotics.data_loader import get_data
trainloader, valloader = get_data(device, config, "path/to/data/folder")
```

By default, the task is the **Contact** task, but passing in `output='ee_yaw_next'` into `get_data` will allow you to access the **End Effector** task

---

2.4 Finance

All of the dataloaders, when created, will automatically download the stock data for you.

As an example, this can be done through the following code block:

```
from datasets.stocks.get_data import get_dataloader
# Here, the list of stocks is a list of strings of stock symbols in all CAPS.
train_loader, val_loader, test_loader = get_dataloader(stocks, stocks, [args.target_
↪stock])
```

For the purposes of the MultiBench paper, we used the following lists per dataset:

```
F&B (18): CAG CMG CPB DPZ DRI GIS HRL HSY K KHC LW MCD MDLZ MKC SBUX SJM TSN YUM
Health (63): ABT ABBV ABMD A ALXN ALGN ABC AMGN ANTM BAX BDX BIO BIIB BSX BMY CAH CTLT␣
↪CNC CERN CI COO CVS DHR DVA XRAY DXCM EW GILD HCA HSIC HOLX HUM IDXX ILMN INCY ISRG␣
↪IQV JNJ LH LLY MCK MDT MRK MTD PKI PRGO PFE DGX REGN RMD STE SYK TFX TMO UNH UHS VAR␣
↪VRTX VTRS WAT WST ZBH ZTS
Tech (100): AAPL ACN ADBE ADI ADP ADSK AKAM AMAT AMD ANET ANSS APH ATVI AVGO BR CDNS CDW␣
↪CHTR CMCSA CRM CSCO CTSH CTXS DIS DISCA DISCK DISH DXC EA ENPH FB FFIV FIS FISV FLIR␣
↪FLT FOX FOXA FTNT GLW GOOG GOOGL GPN HPE HPQ IBM INTC INTU IPG IPGP IT JKHY JNPR KEYS␣
↪KLAC LRCX LUMN LYV MA MCHP MPWR MSFT MSI MU MXIM NFLX NLOK NOW NTAP NVDA NWS NWSA NXPI␣
↪OMC ORCL PAYC PAYX PYPL QCOM QRVO SNPS STX SWKS T TEL TER TMUS TRMB TTWO TWTR TXN TYL␣
↪V VIAC VRSN VZ WDC WU XLNX ZBRA
```

2.5 HCI

2.5.1 ENRiCO

To grab the ENRiCO dataset, please use the `download_data.sh` shell script under the `datasets/enrico` dataset.

Then, to load the data in, you can use something like the following code-block:

As an example, this can be done through the following code block:

```
from datasets.enrico.get_data import get_dataloader

(train_loader, val_loader, test_loader), weights = get_dataloader("datasets/enrico/
↪dataset")
```

2.6 Multimedia

2.6.1 AV-MNIST

To grab the AV-MNIST dataset, please download the `avmnist.tar.gz` file from here .

Once that is done, untar it your preferred location and do the following to get the dataloaders:

```
from datasets.avmnist.get_data import get_dataloader

train_loader, val_loader, test_loader = get_dataloader("path/to/dataset")
```

2.6.2 MM-IMDb

To grab the MM-IMDb dataset, please download the `multimodal_imdb.hdf5` file from here .

If you plan on testing the model's robustness, you will **also** need to download the raw file from here.

Once that is done, untar it your preferred location and do something the following to get the dataloaders:

```
from datasets.imdb.get_data import get_dataloader

train_loader, val_loader, test_loader = get_dataloader("path/to/processed_data","path/to/
↪raw/data/folder",vgg=True, batch_size=128)
```

2.6.3 Kinetics400

To download either of the Kinetics datasets, run the appropriate script under `special/kinetics_*.py`.

Then pass the location of the data to the associated file to finish it.

2.6.4 Clotho

To download the Clotho dataset, clone the repository somewhere on your device and follow the given instructions to pre-process the data.

To get the dataloaders, you will also need to add the path to the above repo to the get_dataloaders function under *datasets/clotho/get_data.py*.

CHAPTER

THREE

CONTINUOUS INTEGRATION GUIDE

3.1 Repo Structure

MultiBench is separated into several sub-packages, each of which handle separate functionality regarding either the set of reference implementations or the actual benchmarking process accordingly:

- **datasets** - This package handles loading data, and creating the dataloaders for each multimodal dataset we look at. See the 'Downloading Datasets' guide for information on how to use this section of the package.
- **eval_scripts** - This package contains the implementations of any and all complexity measures we use to measure a MultiModal model's complexity, along with a few helper functions to get all metrics during training.
- **fusions** - This package contains implementations of multiple multimodal fusion methods. These take in either raw or processed modalities, and combine them ahead of a final "classification head" layer.
- **unimodals** - This package contains implementations of several unimodal processing methods, which take in raw modality information and return dense vector representations for the fusion methods to use.
- **objective_functions** - This package contains implementations of objective functions, which are used to train the associated models accordingly.
- **robustness** - This package contains implementations of several unimodal noising methods, which take in raw modality information and add some noise to them.
- **training_structures** - This package contains implementations of generic model training structures, which take in the above systems and train/test the model end to end.
- **utils** - Lastly, this package contains assorted extranneous functions that are useful all around the package.

In addition, the following folders are also in the repository, to provide additional functionality / testing to the repository:

- **deprecated** - This contains older code not currently used in the current version of MultiBench.
- **images** - This contains images for the README.
- **pretrained** - This folder contains pre-trained encoders for some papers/modalities.
- **private_test_scripts** - This folder contains additional but private test scripts for MultiBench.
- **tests** - This folder contains the tests for our code-coverage report.
- **sphinx** - This folder contains all of the build information and source files for our documentation.

3.2 Dataloading, Training, and Evaluation

While the tutorials provided are complete, in that they walk you through sample uses of MultiBench, here's a quick overview of how to run your experiments in MultiBench:

1. **Construct your dataloaders** - If your dataset of interest has not been studied in MultiBench before, you'll need to construct the associated dataloaders in the datasets package first. Keep in mind that, if you want to test robustness, you will need to add the associated robustness transformations to the associated test/validation dataloader.
2. **Decide on your encoders, fusion model, and classification head** - MultiBench separates models into the following three sections. You'll need to choose your model's structure ahead of any test:
    a. **Encoders** - These take in raw modalities and process them.
    b. **Fusion Model** - These take in the processed modalities and combine them.
    c. **Classification Head** - This takes in the processed modalities and predicts classification output.
3. **Pick the associated training structure for your model** - For most purposes, this will be the training loop under `training_structures/Supervised_Learning.py`, but for some other architectures you'll need to use the other training structures.
4. **Pick the metrics and loss function**

Once you've done this, you can plug the associated code into the training loop like the examples, and go from there.

3.3 Testing and Contributing

If you would like to contribute, we would love to have you on. If you have a proposed extension, feel free to make an issue so we can contact you and get the ball rolling.

To add a new dataset:

1. **Create a new folder in the datasets/ package**
2. **Write a python file with a get_dataloader function**
3. **Go to examples/ and write an example to test that your code works with a simple training script**

To add a new algorithm:

1. **Decide where in the four algorithm folders your code will go**
2. **Write documentation for that algorithm, following the documentation process of the other modules**
    a. This will include arguments, return data, and associated information.
    b. If you can link to the paper for this algorithm, that would also be appreciated.
3. **Write unit and integration tests for that algorithm under tests/**
    a. Example unit tests and integration tests can be found under `tests/`, such as `tests/test_fusion.py` and `tests/test_Supervised_Learning.py`.
    b. `tests/common.py` provides utility functions that allow deterministic tests of function correctness using pseudo-random inputs.
4. **Run the test build locally to make sure the new changes can be smoothly integrated to the GitHub Actions workflows using the following command**

```
python -m pytest -s --cov-report html --cov=utils --cov=unimodals/ --cov=robustness --
↪cov=fusions/ --cov=objective_functions tests/
```

For debugging, the command to run a single test file is

```
python -m pytest -s --cov-report html --cov=utils --cov=unimodals/ --cov=robustness --
↪cov=fusions/ --cov=objective_functions tests/tests_TESTNAME.py
```

We suggest running the entire test build at least once locally before pushing the changes

5. **Create a pull request and the authors will merge these changes into the main branch**

CHAPTER

FOUR

DATASETS AND DATALOADERS

4.1 datasets package

4.1.1 Subpackages

datasets.RTFM package

Submodules

datasets.RTFM.get_env module

Implements environment getter for RTFM.

datasets.RTFM.get_env.**create_env**(*env='rtfm:groups_simple_stationary-v0'*, *height=6*, *width=6*, *partially_observable=False*, *max_placement=1*, *featurizer=None*, *shuffle_wiki=False*, *time_penalty=-0.02*)

Create RTFM environment.

Parameters

- **env** (*str, optional*) – RTFM environment name.. Defaults to "rtfm:groups_simple_stationary-v0".
- **height** (*int, optional*) – Height of environment. Defaults to 6.
- **width** (*int, optional*) – Width of environment. Defaults to 6.
- **partially_observable** (*bool, optional*) – Whether to only give partial observations or priviledged observations. Defaults to False.
- **max_placement** (*int, optional*) – Max placement. Defaults to 1.
- **featurizer** (*_type_, optional*) – Function for featurizing inputs. Defaults to None.
- **shuffle_wiki** (*bool, optional*) – Whether to shuffle wiki. Defaults to False.
- **time_penalty** (*float, optional*) – Time penalty. Defaults to -0.02.

Returns
gym environment

Return type
env

Module contents

datasets.affect package

Submodules

datasets.affect.get_bert_embedding module

Implements BERT embedding extractors.

datasets.affect.get_bert_embedding.**bert_version_data**(*data*, *raw_path*, *keys*, *max_padding=50*, *bert_max_len=None*)

Get bert encoded data

Parameters

- **data** (*dict*) – Data dictionary
- **raw_path** (*str*) – Path to raw data
- **keys** (*dict*) – List of keys in raw text getter
- **max_padding** (*int, optional*) – Maximum padding to add to list. Defaults to 50.
- **bert_max_len** (*int, optional*) – Maximum length in BERT. Defaults to None.

Returns
Dictionary from modality to data.

Return type
dict

datasets.affect.get_bert_embedding.**corresponding_other_modality_ids**(*orig_text*, *tokenized_text*)

Align word ids to other modalities.

Since tokenizer splits the word into parts e.g. '##ing' or 'you're' -> 'you', '", 're' we should get the corresponding ids for other modalities' features applied to modalities which aligned to words

Parameters

- **orig_text** (*list*) – List of strings corresponding to the original text.
- **tokenized_text** (*list*) – List of lists of tokens.

Returns
List of ids.

Return type
list

datasets.affect.get_bert_embedding.**get_bert_features**(*all_text*, *contextual_embedding=False*, *batch_size=500*, *max_len=None*)

Get bert features from data.

Use pipline to extract all the features, (num_points, max_seq_length, feature_dim): np.ndarray

Parameters

- **all_text** (*list*) – Data to get BERT features from
- **contextual_embedding** (*bool, optional*) – If True output the last hidden state of bert. If False, output the embedding of words. Defaults to False.

- **batch_size** (*int, optional*) – Batch size. Defaults to 500.
- **max_len** (*int, optional*) – Maximum length of the dataset. Defaults to None.

Returns
BERT features of text.

Return type
np.array

datasets.affect.get_bert_embedding.**get_rawtext**(*path*, *data_kind*, *vids=None*)

"Get raw text from the datasets.

Parameters

- **path** (*str*) – Path to data
- **data_kind** (*str*) – Data Kind. Must be 'hdf5'.
- **vids** (*list, optional*) – List of video data as np.array. Defaults to None.

Returns
Text data list, video data list

Return type
tuple(list, list)

datasets.affect.get_bert_embedding.**max_seq_len**(*id_list*, *max_len=50*)

Fix dataset to max sequence length.

Cut the id lists with the max length, but didnt do padding here. Add the first one as [CLS] and the last one for [SEP].

Parameters

- **id_list** (*list*) – List of ids to manipulate
- **max_len** (*int, optional*) – Maximum sequence length. Defaults to 50.

Returns
List of tokens

Return type
list

datasets.affect.get_data module

Implements dataloaders for AFFECT data.

class datasets.affect.get_data.**Affectdataset**(**args: Any*, ***kwargs: Any*)

Bases: Dataset

Implements Affect data as a torch dataset.

__init__(*data: Dict*, *flatten_time_series: bool*, *aligned: bool = True*, *task: Optional[str] = None*, *max_pad=False*, *max_pad_num=50*, *data_type='mosi'*, *z_norm=False*) → None

Instantiate AffectDataset

Parameters

- **data** (*Dict*) – Data dictionary
- **flatten_time_series** (*bool*) – Whether to flatten time series or not

- **aligned** (*bool, optional*) – Whether to align data or not across modalities. Defaults to True.
- **task** (*str, optional*) – What task to load. Defaults to None.
- **max_pad** (*bool, optional*) – Whether to pad data to max_pad_num or not. Defaults to False.
- **max_pad_num** (*int, optional*) – Maximum padding number. Defaults to 50.
- **data_type** (*str, optional*) – What data to load. Defaults to 'mosi'.
- **z_norm** (*bool, optional*) – Whether to normalize data along the z-axis. Defaults to False.

datasets.affect.get_data.**drop_entry**(*dataset*)

Drop entries where there's no text in the data.

datasets.affect.get_data.**get_dataloader**(*filepath: str*, *batch_size: int = 32*, *max_seq_len=50*, *max_pad=False*, *train_shuffle: bool = True*, *num_workers: int = 2*, *flatten_time_series: bool = False*, *task=None*, *robust_test=False*, *data_type='mosi'*, *raw_path='/home/van/backup/pack/mosi/mosi.hdf5'*, *z_norm=False*) → torch.utils.data.DataLoader

Get dataloaders for affect data.

Parameters

- **filepath** (*str*) – Path to datafile
- **batch_size** (*int, optional*) – Batch size. Defaults to 32.
- **max_seq_len** (*int, optional*) – Maximum sequence length. Defaults to 50.
- **max_pad** (*bool, optional*) – Whether to pad data to max length or not. Defaults to False.
- **train_shuffle** (*bool, optional*) – Whether to shuffle training data or not. Defaults to True.
- **num_workers** (*int, optional*) – Number of workers. Defaults to 2.
- **flatten_time_series** (*bool, optional*) – Whether to flatten time series data or not. Defaults to False.
- **task** (*str, optional*) – Which task to load in. Defaults to None.
- **robust_test** (*bool, optional*) – Whether to apply robustness to data or not. Defaults to False.
- **data_type** (*str, optional*) – What data to load in. Defaults to 'mosi'.
- **raw_path** (*str, optional*) – Full path to data. Defaults to '/home/van/backup/pack/mosi/mosi.hdf5'.
- **z_norm** (*bool, optional*) – Whether to normalize data along the z dimension or not. Defaults to False.

Returns

tuple of train dataloader, validation dataloader, test dataloader

Return type

DataLoader

datasets.affect.get_data.**get_rawtext**(*path*, *data_kind*, *vids*)

Get raw text, video data from hdf5 file.

datasets.affect.get_data.**z_norm**(*dataset*, *max_seq_len=50*)

Normalize data in the dataset.

datasets.affect.get_raw_data module

Handle getting raw data from mosi

datasets.affect.get_raw_data.**detect_entry_fold**(*entry*, *folds*)

Detect entry fold.

Parameters

- **entry** (*str*) – Entry string
- **folds** (*int*) – Number of folds

Returns
Entry fold index

Return type
int

datasets.affect.get_raw_data.**get_audio_visual_text**(*csds*, *seq_len*, *text_data*, *vids*)

Get audio visual from text.

datasets.affect.get_raw_data.**get_rawtext**(*path*, *data_kind*, *vids*)

Get raw text modality.

Parameters

- **path** (*str*) – Path to h5 file
- **data_kind** (*str*) – String for data format. Should be 'hdf5'.
- **vids** (*list*) – List of video ids.

Returns
Tuple of text_data and video_data in lists.

Return type
tuple(list,list)

datasets.affect.get_raw_data.**get_word2id**(*text_data*, *vids*)

From text_data, vids get word2id lsit

Parameters

- **text_data** (*list*) – List of text data
- **vids** (*list*) – List of video data

Returns
List of word2id data

Return type
list

datasets.affect.get_raw_data.**get_word_embeddings**(*word2id*, *save=False*)

Given a word2id, get the associated glove embeddings (300 dimensional).

Parameters

- **word2id** (*list*) – list of word, index pairs
- **save** (*bool, optional*) – Whether to save data to the folder (unused). Defaults to False.

Returns
List of embedded words

Return type
list[np.array]

datasets.affect.get_raw_data.**glove_embeddings**(*text_data*, *vids*, *paddings=50*)

Get glove embeddings of text, video pairs.

Parameters

- **text_data** (*list*) – list of text data.
- **vids** (*list*) – list of video data
- **paddings** (*int, optional*) – Amount to left-pad data if it's less than some size. Defaults to 50.

Returns
Array of embedded data

Return type
np.array

datasets.affect.get_raw_data.**lpad**(*this_array*, *seq_len*)

Left pad array with seq_len 0s.

Parameters

- **this_array** (*np.array*) – Array to pad
- **seq_len** (*int*) – Number of 0s to pad.

Returns
Padded array

Return type
np.array

Module contents

datasets.avmnist package

Submodules

datasets.avmnist.get_data module

Implements dataloaders for the AVMNIST dataset.

Here, the data is assumed to be in a folder titled "avmnist".

datasets.avmnist.get_data.**get_dataloader**(*data_dir*, *batch_size=40*, *num_workers=8*, *train_shuffle=True*, *flatten_audio=False*, *flatten_image=False*, *unsqueeze_channel=True*, *generate_sample=False*, *normalize_image=True*, *normalize_audio=True*)

Get dataloaders for AVMNIST.

Parameters

- **data_dir** (*str*) – Directory of data.
- **batch_size** (*int, optional*) – Batch size. Defaults to 40.
- **num_workers** (*int, optional*) – Number of workers. Defaults to 8.
- **train_shuffle** (*bool, optional*) – Whether to shuffle training data or not. Defaults to True.
- **flatten_audio** (*bool, optional*) – Whether to flatten audio data or not. Defaults to False.
- **flatten_image** (*bool, optional*) – Whether to flatten image data or not. Defaults to False.
- **unsqueeze_channel** (*bool, optional*) – Whether to unsqueeze any channels or not. Defaults to True.
- **generate_sample** (*bool, optional*) – Whether to generate a sample and save it to file or not. Defaults to False.
- **normalize_image** (*bool, optional*) – Whether to normalize the images before returning. Defaults to True.
- **normalize_audio** (*bool, optional*) – Whether to normalize the audio before returning. Defaults to True.

Returns
Tuple of (training dataloader, validation dataloader, test dataloader)

Return type
tuple

Module contents

datasets.clotho package

Submodules

datasets.clotho.clotho_data_loader module

Implements the creation of the dataloader for CLOTHO.

datasets.clotho.clotho_data_loader.**get_clotho_loader**(*data_dir: Path*, *split: str*, *input_field_name: str*, *output_field_name: str*, *load_into_memory: bool*, *batch_size: int*, *nb_t_steps_pad: Union[AnyStr, Tuple[int, int]]*, *shuffle: Optional[bool] = True*, *drop_last: Optional[bool] = True*, *input_pad_at: Optional[str] = 'start'*, *output_pad_at: Optional[str] = 'end'*, *num_workers: Optional[int] = 1*) → DataLoader

Gets the clotho data loader.

Parameters

- **data_dir** (*pathlib.Path*) – Directory with data.
- **split** (*str*) – Split to use (i.e. 'development', 'evaluation')
- **input_field_name** (*str*) – Field name of the clotho data to be used as input data to the method.
- **output_field_name** (*str*) – Field name of the clotho data to be used as output data to the method.
- **load_into_memory** (*bool*) – Load all data into memory?
- **batch_size** (*int*) – Batch size to use.
- **nb_t_steps_pad** (*str|(int, int)*) – Number of time steps to pad/truncate to. Cab use 'max', 'min', or exact number e.g. (1024, 10).
- **shuffle** (*bool, optional*) – Shuffle examples? Defaults to True.
- **drop_last** (*bool, optional*) – Drop the last examples if not making a batch of *batch_size*? Defaults to True.
- **input_pad_at** (*str*) – Pad input at the start or at the end?
- **output_pad_at** (*str*) – Pad output at the start or at the end?
- **num_workers** (*int, optional*) – Amount of workers, defaults to 1.

Returns
Dataloader for Clotho data.

Return type
torch.utils.data.dataloader.DataLoader

datasets.clotho.clotho_dataset module

Implements the CLOTHO dataset as a torch dataset.

class datasets.clotho.clotho_dataset.**ClothoDataset**(*data_dir: Path*, *split: AnyStr*, *input_field_name: AnyStr*, *output_field_name: AnyStr*, *load_into_memory: bool*)

Bases: Dataset

Implements the CLOTHO dataset as a torch dataset.

__init__(*data_dir: Path*, *split: AnyStr*, *input_field_name: AnyStr*, *output_field_name: AnyStr*, *load_into_memory: bool*) → None

Initialization of a Clotho dataset object.

Parameters

- **data_dir** (*pathlib.Path*) – Directory with data.
- **split** (*str*) – Split to use (i.e. 'development', 'evaluation')
- **input_field_name** (*str*) – Field name of the clotho data to be used as input data to the method.

- **output_field_name** (*str*) – Field name of the clotho data to be used as output data to the method.
- **load_into_memory** (*bool*) – Load all data into memory?

datasets.clotho.collate_fn module

Implements the collation function for CLOTHO data.

datasets.clotho.collate_fn.**clotho_collate_fn**(*batch: MutableSequence[ndarray]*, *nb_t_steps: Union[AnyStr, Tuple[int, int]]*, *input_pad_at: str*, *output_pad_at: str*) → Tuple[Tensor, Tensor]

Pads data.

Parameters

- **batch** (*list[numpy.ndarray]*) – Batch data.
- **nb_t_steps** (*str / (int, int)*) – Number of time steps to pad/truncate to. Cab use 'max', 'min', or exact number e.g. (1024, 10).
- **input_pad_at** (*str*) – Pad input at the start or at the end?
- **output_pad_at** (*str*) – Pad output at the start or at the end?

Returns
Padded data.

Return type
torch.Tensor, torch.Tensor

datasets.clotho.get_data module

Implements dataloader creators for the CLOTHO dataset used in MultiBench.

datasets.clotho.get_data.**get_dataloaders**(*path_to_clotho*, *input_modal='features'*, *output_modal='words_ind'*, *num_workers=1*, *shuffle_train=True*, *batch_size=20*)

Get dataloaders for CLOTHO dataset.

Parameters

- **path_to_clotho** (*str*) – Path to clotho dataset
- **input_modal** (*str, optional*) – Input modality. Defaults to 'features'.
- **output_modal** (*str, optional*) – Output modality. Defaults to 'words_ind'.
- **num_workers** (*int, optional*) – Number of workers. Defaults to 1.
- **shuffle_train** (*bool, optional*) – Whether to shuffle training data or not. Defaults to True.
- **batch_size** (*int, optional*) – Batch size. Defaults to 20.

Returns
Tuple of (training dataloader, validation dataloader)

Return type
tuple

Module contents

Implements CLOTHO dataloaders.

datasets.clotho.**get_clotho_loader**(*data_dir: Path*, *split: str*, *input_field_name: str*, *output_field_name: str*, *load_into_memory: bool*, *batch_size: int*, *nb_t_steps_pad: Union[AnyStr, Tuple[int, int]]*, *shuffle: Optional[bool] = True*, *drop_last: Optional[bool] = True*, *input_pad_at: Optional[str] = 'start'*, *output_pad_at: Optional[str] = 'end'*, *num_workers: Optional[int] = 1*) → DataLoader

Gets the clotho data loader.

Parameters

- **data_dir** (*pathlib.Path*) – Directory with data.
- **split** (*str*) – Split to use (i.e. 'development', 'evaluation')
- **input_field_name** (*str*) – Field name of the clotho data to be used as input data to the method.
- **output_field_name** (*str*) – Field name of the clotho data to be used as output data to the method.
- **load_into_memory** (*bool*) – Load all data into memory?
- **batch_size** (*int*) – Batch size to use.
- **nb_t_steps_pad** (*str|(int, int)*) – Number of time steps to pad/truncate to. Cab use 'max', 'min', or exact number e.g. (1024, 10).
- **shuffle** (*bool, optional*) – Shuffle examples? Defaults to True.
- **drop_last** (*bool, optional*) – Drop the last examples if not making a batch of *batch_size*? Defaults to True.
- **input_pad_at** (*str*) – Pad input at the start or at the end?
- **output_pad_at** (*str*) – Pad output at the start or at the end?
- **num_workers** (*int, optional*) – Amount of workers, defaults to 1.

Returns
Dataloader for Clotho data.

Return type
torch.utils.data.dataloader.DataLoader

datasets.enrico package

Submodules

datasets.enrico.get_data module

Implements dataloaders for ENRICO dataset.

class datasets.enrico.get_data.**EnricoDataset**(*data_dir*, *mode='train'*, *noise_level=0*, *img_noise=False*, *wireframe_noise=False*, *img_dim_x=128*, *img_dim_y=256*, *random_seed=42*, *train_split=0.65*, *val_split=0.15*, *test_split=0.2*, *normalize_image=False*, *seq_len=64*)

Bases: Dataset

Implements torch dataset class for ENRICO dataset.

__init__(*data_dir*, *mode='train'*, *noise_level=0*, *img_noise=False*, *wireframe_noise=False*, *img_dim_x=128*, *img_dim_y=256*, *random_seed=42*, *train_split=0.65*, *val_split=0.15*, *test_split=0.2*, *normalize_image=False*, *seq_len=64*)

Instantiate ENRICO dataset.

Parameters

- **data_dir** (*str*) – Data directory.
- **mode** (*str, optional*) – What data to extract. Defaults to "train".
- **noise_level** (*int, optional*) – Noise level, as defined in robustness. Defaults to 0.
- **img_noise** (*bool, optional*) – Whether to apply noise to images or not. Defaults to False.
- **wireframe_noise** (*bool, optional*) – Whether to apply noise to wireframes or not. Defaults to False.
- **img_dim_x** (*int, optional*) – Image width. Defaults to 128.
- **img_dim_y** (*int, optional*) – Image height. Defaults to 256.
- **random_seed** (*int, optional*) – Seed to split dataset on and shuffle data on. Defaults to 42.
- **train_split** (*float, optional*) – Percentage of training data split. Defaults to 0.65.
- **val_split** (*float, optional*) – Percentage of validation data split. Defaults to 0.15.
- **test_split** (*float, optional*) – Percentage of test data split. Defaults to 0.2.
- **normalize_image** (*bool, optional*) – Whether to normalize image or not Defaults to False.
- **seq_len** (*int, optional*) – Length of sequence. Defaults to 64.

featurizeElement(*element*)

Convert element into tuple of (bounds, one-hot-label).

datasets.enrico.get_data.**get_dataloader**(*data_dir*, *batch_size=32*, *num_workers=0*, *train_shuffle=True*, *return_class_weights=True*)

Get dataloaders for this dataset.

Parameters

- **data_dir** (*str*) – Data directory.
- **batch_size** (*int, optional*) – Batch size. Defaults to 32.
- **num_workers** (*int, optional*) – Number of workers. Defaults to 0.
- **train_shuffle** (*bool, optional*) – Whether to shuffle training data or not. Defaults to True.
- **return_class_weights** (*bool, optional*) – Whether to return class weights or not. Defaults to True.

Returns

Tuple of ((train dataloader, validation dataloader, test dataloader), class_weights) if return_class_weights, otherwise just the dataloaders

Return type
tuple

Module contents

datasets.gentle_push package

Submodules

datasets.gentle_push.data_loader module

Implements dataloaders for Gentle Push tasks.

Sourced from https://github.com/brentyi/multimodalfilter/blob/master/crossmodal/tasks/_push.py

class datasets.gentle_push.data_loader.**PushTask**

Bases: object

Dataset definition and model registry for pushing task.

classmethod add_dataset_arguments(*parser: ArgumentParser*) → None

Add dataset options to an argument parser.

Parameters
parser (*argparse.ArgumentParser*) – Parser to add arguments to.

classmethod get_dataloader(*subsequence_length: int*, *modalities=None*, *batch_size=32*, *drop_last=True*, ***dataset_args*)

Get dataloaders for gentle-push dataset.

Parameters

- **subsequence_length** (*int*) – Length of subsequences for each sample
- **modalities** (*list, optional*) – List of strings defining which modalities to use. Defaults to None. Pick from ['gripper_pos', 'gripper_sensors', 'image', 'control'].
- **batch_size** (*int, optional*) – Batch size. Defaults to 32.
- **drop_last** (*bool, optional*) – Whether to drop last datasample or not. Defaults to True.

Returns
Tuple of training dataloader, validation dataloader, and test dataloader

Return type
tuple

classmethod get_dataset_args(*args: Namespace*) → Dict[str, Any]

Get a dataset_args dictionary from a parsed set of arguments.

Parameters
args (*argparse.Namespace*) – Parsed arguments.

classmethod get_eval_trajectories(***dataset_args*) → List[*TrajectoryNumpy*]

Get evaluation trajectories.

classmethod get_test_trajectories(*modalities*, ***dataset_args*)

Get test trajectories.

classmethod get_train_trajectories(**dataset_args*) → List[*TrajectoryNumpy*]

Get training trajectories.

class datasets.gentle_push.data_loader.**SubsequenceDataset**(*trajectories: List[TrajectoryNumpy]*, *subsequence_length: int*, *modalities=None*)

Bases: `Dataset`

A data preprocessor for producing training subsequences from a list of trajectories. Thin wrapper around *torchfilter.data.split_trajectories()*.

Parameters

- **trajectories** (*list*) – list of trajectories, where each is a tuple of *(states, observations, controls)*. Each tuple member should be either a numpy array or dict of numpy arrays with shape *(T, ...)*.
- **subsequence_length** (*int*) – # of timesteps per subsequence.

__init__(*trajectories: List[TrajectoryNumpy]*, *subsequence_length: int*, *modalities=None*)

Initialize SubsequenceDataset object.

Parameters

- **trajectories** (*List[TrajectoryNumpy]*) – List of trajectories
- **subsequence_length** (*int*) – Length to sample each subsequence
- **modalities** (*list, optional*) – List of strings of modalities to choose. Defaults to None. Choose from ['gripper_pos', 'gripper_sensors', 'image', 'control'].

class datasets.gentle_push.data_loader.**TrajectoryNumpy**(*states: Any*, *observations: Any*, *controls: Any*)

Bases: `tuple`

Named tuple containing states, observations, and controls represented in NumPy.

property controls

Alias for field number 2

property observations

Alias for field number 1

property states

Alias for field number 0

datasets.gentle_push.data_loader.**split_trajectories**(*trajectories: List[TrajectoryNumpy]*, *subsequence_length: int*, *modalities=None*)

Helper for splitting a list of trajectories into a list of overlapping subsequences. For each trajectory, assuming a subsequence length of 10, this function includes in its output overlapping subsequences corresponding to timesteps... ```

[0:10], [10:20], [20:30], ...

` as well as... `

[5:15], [15:25], [25:30], ...

``` :param trajectories: List of trajectories. :type trajectories: List[torchfilter.base.TrajectoryNumpy] :param subsequence_length: # of timesteps per subsequence. :type subsequence_length: int
```

Returns
List of subsequences.

Return type
List[torchfilter.base.TrajectoryNumpy]

Module contents

datasets.imdb package

Submodules

datasets.imdb.get_data module

Implements dataloaders for IMDB dataset.

class datasets.imdb.get_data.**IMDBDataset**(*file: h5py.File*, *start_ind: int*, *end_ind: int*, *vggfeature: bool = False*)

Bases: Dataset

Implements a torch Dataset class for the imdb dataset.

__init__(*file: h5py.File*, *start_ind: int*, *end_ind: int*, *vggfeature: bool = False*) → None

Initialize IMDBDataset object.

Parameters

- **file** (*h5py.File*) – h5py file of data
- **start_ind** (*int*) – Starting index for dataset
- **end_ind** (*int*) – Ending index for dataset
- **vggfeature** (*bool, optional*) – Whether to return pre-processed vgg_features or not. Defaults to False.

class datasets.imdb.get_data.**IMDBDataset_robust**(*dataset*, *start_ind: int*, *end_ind: int*)

Bases: Dataset

Implements a torch Dataset class for the imdb dataset that uses robustness measures as data augmentation.

__init__(*dataset*, *start_ind: int*, *end_ind: int*) → None

Initialize IMDBDataset_robust object.

Parameters

- **file** (*h5py.File*) – h5py file of data
- **start_ind** (*int*) – Starting index for dataset
- **end_ind** (*int*) – Ending index for dataset
- **vggfeature** (*bool, optional*) – Whether to return pre-processed vgg_features or not. Defaults to False.

datasets.imdb.get_data.**get_dataloader**(*path: str*, *test_path: str*, *num_workers: int = 8*, *train_shuffle: bool = True*, *batch_size: int = 40*, *vgg: bool = False*, *skip_process=False*, *no_robust=False*) → Tuple[Dict]

Get dataloaders for IMDB dataset.

Parameters

- **path** (*str*) – Path to training datafile.
- **test_path** (*str*) – Path to test datafile.
- **num_workers** (*int, optional*) – Number of workers to load data in. Defaults to 8.
- **train_shuffle** (*bool, optional*) – Whether to shuffle training data or not. Defaults to True.
- **batch_size** (*int, optional*) – Batch size of data. Defaults to 40.
- **vgg** (*bool, optional*) – Whether to return raw images or pre-processed vgg features. Defaults to False.
- **skip_process** (*bool, optional*) – Whether to pre-process data or not. Defaults to False.
- **no_robust** (*bool, optional*) – Whether to not use robustness measures as augmentation. Defaults to False.

Returns
Tuple of Training dataloader, Validation dataloader, Test Dataloader

Return type
Tuple[Dict]

datasets.imdb.vgg module

Implements VGG pre-processer for IMDB data.

class datasets.imdb.vgg.**VGGClassifier**(*model_path='vgg.tar'*, *synset_words='synset_words.txt'*)

Bases: object

Implements VGG classifier instance.

__init__(*model_path='vgg.tar'*, *synset_words='synset_words.txt'*)

Instantiate VGG classifier instance.

Parameters

- **model_path** (*str, optional*) – VGGNet weight file. Defaults to 'vgg.tar'.
- **synset_words** (*str, optional*) – Path to synset words. Defaults to 'synset_words.txt'.

classify(*image*, *top=1*)

Classify an image with the 1000 concepts of the ImageNet dataset.

Image
numpy image or image path.

Top
Number of top classes for this image.

Returns
list of strings with synsets predicted by the VGG model.

get_features(*image*)

Return the activations of the last hidden layer for a given image.

Image
numpy image or image path.

Returns
numpy vector with 4096 activations.

`resize_and_crop_image`(*output_box=[224, 224]*, *fit=True*)

Downsample the image.

Sourced from https://github.com/BVLC/caffe/blob/master/tools/extra/resize_and_crop_images.py

`class datasets.imdb.vgg.VGGNet`(**args: Any*, ***kwargs: Any*)

Bases: `FeedforwardSequence`

Implements VGG pre-processor.

`__init__`(***kwargs*)

Instantiate VGG pre-processor instance.

Module contents

datasets.mimic package

Submodules

datasets.mimic.get_data module

Implements dataloaders for generic MIMIC tasks.

`datasets.mimic.get_data.get_dataloader`(*task*, *batch_size=40*, *num_workers=1*, *train_shuffle=True*, *imputed_path='im.pk'*, *flatten_time_series=False*, *tabular_robust=True*, *timeseries_robust=True*)

Get dataloaders for MIMIC dataset.

Parameters

- **task** (*int*) – Integer between -1 and 19 inclusive, -1 means mortality task, 0-19 means icd9 task.
- **batch_size** (*int, optional*) – Batch size. Defaults to 40.
- **num_workers** (*int, optional*) – Number of workers to load data in. Defaults to 1.
- **train_shuffle** (*bool, optional*) – Whether to shuffle training data or not. Defaults to True.
- **imputed_path** (*str, optional*) – Datafile location. Defaults to 'im.pk'.
- **flatten_time_series** (*bool, optional*) – Whether to flatten time series data or not. Defaults to False.
- **tabular_robust** (*bool, optional*) – Whether to apply tabular robustness as dataset augmentation or not. Defaults to True.
- **timeseries_robust** (*bool, optional*) – Whether to apply timeseries robustness noises as dataset augmentation or not. Defaults to True.

Returns
Tuple of training dataloader, validation dataloader, and test dataloader

Return type
tuple

datasets.mimic.multitask module

Implements data loaders for the multitask formulation of MIMIC.

datasets.mimic.multitask.**get_dataloader**(*batch_size=40*, *num_workers=1*, *train_shuffle=True*, *imputed_path='im.pk'*, *flatten_time_series=False*)

Generate dataloader for multi-task setup, using only tasks -1 and 7.

Parameters

- **batch_size** (*int, optional*) – Batch size. Defaults to 40.
- **num_workers** (*int, optional*) – Number of workers to load data in. Defaults to 1.
- **train_shuffle** (*bool, optional*) – Whether to shuffle training data or not. Defaults to True.
- **imputed_path** (*str, optional*) – Datafile location. Defaults to 'im.pk'.
- **flatten_time_series** (*bool, optional*) – Whether to flatten time series data or not. Defaults to False.

Returns
Tuple of training dataloader, validation dataloader, and test dataloader.

Return type
tupe

Module contents

datasets.robotics package

Submodules

datasets.robotics.MultimodalManipulationDataset module

Implements dataset for MultiModal Manipulation Task.

class datasets.robotics.MultimodalManipulationDataset.**MultimodalManipulationDataset**(*filename_list*, *transform=None*, *episode_length=50*, *training_type='selfsupervised'*, *n_time_steps=1*, *action_dim=4*, *pairing_tolerance=0.06*, *filedirprefix=''*)

Bases: Dataset

Multimodal Manipulation dataset.

__init__(*filename_list*, *transform=None*, *episode_length=50*, *training_type='selfsupervised'*, *n_time_steps=1*, *action_dim=4*, *pairing_tolerance=0.06*, *filedirprefix=''*)

Initialize dataset.

Parameters

- **filename_list** (*str*) – List of files to get data from
- **transform** (*fn, optional*) – Optional function to transform data. Defaults to None.
- **episode_length** (*int, optional*) – Length of each episode. Defaults to 50.
- **training_type** (*str, optional*) – Type of training. Defaults to "selfsupervised".
- **n_time_steps** (*int, optional*) – Number of time steps. Defaults to 1.
- **action_dim** (*int, optional*) – Action dimension. Defaults to 4.
- **pairing_tolerance** (*float, optional*) – Pairing tolerance. Defaults to 0.06.
- **filedirprefix** (*str, optional*) – File directory prefix (unused). Defaults to "".

class datasets.robotics.MultimodalManipulationDataset.**MultimodalManipulationDataset_robust**(*filename_list*, *transform=None*, *episode_length=*
training_type='selfsu
n_time_steps=1
action_dim=4,
pairing_tolerance=0
filedirprefix='',
noise_level=0,
image_noise=Fals
force_noise=Fa
prop_noise=Fal

Bases: Dataset

Multimodal Manipulation dataset.

__init__(*filename_list*, *transform=None*, *episode_length=50*, *training_type='selfsupervised'*, *n_time_steps=1*, *action_dim=4*, *pairing_tolerance=0.06*, *filedirprefix=''*, *noise_level=0*, *image_noise=False*, *force_noise=False*, *prop_noise=False*)

Parameters

- **hdf5_file** (*handle*) – h5py handle of the hdf5 file with annotations.
- **transform** (*callable, optional*) – Optional transform to be applied on a sample.

datasets.robotics.MultimodalManipulationDataset_robust module

Implements MultimodalManipulationDataset with robustness transforms.

class datasets.robotics.MultimodalManipulationDataset_robust.**MultimodalManipulationDataset_robust**(*filenam*
trans-
form=
episod
train-
ing_typ
n_time
ac-
tion_di
pair-
ing_tol
filedirp
fix='',
*noise_*
im-
age_no
*force_*
prop_n

Bases: `Dataset`

Multimodal Manipulation dataset.

__init__(*filename_list*, *transform=None*, *episode_length=50*, *training_type='selfsupervised'*, *n_time_steps=1*, *action_dim=4*, *pairing_tolerance=0.06*, *filedirprefix=''*, *noise_level=0*, *image_noise=False*, *force_noise=False*, *prop_noise=False*)

Parameters

- **hdf5_file** (*handle*) – h5py handle of the hdf5 file with annotations.
- **transform** (*callable, optional*) – Optional transform to be applied on a sample.

datasets.robotics.ProcessForce module

Implements processforce, which truncates force readings to a window size.

class datasets.robotics.ProcessForce.**ProcessForce**(*window_size*, *key='force'*, *tanh=False*)

Bases: `object`

Truncate a time series of force readings with a window size. :param window_size: Length of the history window that is

used to truncate the force readings

__init__(*window_size*, *key='force'*, *tanh=False*)

Initialize ProcessForce object.

Parameters

- **window_size** (*int*) – Windows size
- **key** (*str, optional*) – Key where data is stored. Defaults to 'force'.

- **tanh** (*bool, optional*) – Whether to apply tanh to output or not. Defaults to False.

datasets.robotics.ToTensor module

Implements another utility for this dataset.

`class datasets.robotics.ToTensor.ToTensor(device=None)`

Bases: `object`

Convert ndarrays in sample to Tensors.

`__init__(device=None)`

Initialize ToTensor object.

datasets.robotics.get_data module

Implements dataloaders for robotics data.

`datasets.robotics.get_data.combine_modalitiesbuilder(unimodal, output)`

Create a function data combines modalities given the type of input.

Parameters

- **unimodal** (*str*) – Input type as a string. Can be 'force', 'proprio', 'image'. Defaults to using all modalities otherwise
- **output** (*int*) – Index of output modality.

`datasets.robotics.get_data.get_data(device, configs, filedirprefix='', unimodal=None, output='contact_next')`

Get dataloaders for robotics dataset.

Parameters

- **device** (*torch.utils.device*) – Device to load data to.
- **configs** (*dict*) – Configuration dictionary
- **filedirprefix** (*str, optional*) – File directory prefix path. Defaults to "".
- **unimodal** (*str, optional*) – Input modality as a string. Defaults to None. Can be 'force', 'proprio', 'image'. Defaults to using all modalities otherwise.
- **output** (*str, optional*) – Output format. Defaults to 'contact_next'.

Returns

description

Return type

type

datasets.robotics.utils module

Extraneous methods for robotics dataset work.

datasets.robotics.utils.**augment_val**(*val_filename_list*, *filename_list*)

Augment lists of filenames so that they match the current directory.

Module contents

Package that implements dataloaders for robotics data on Multibench.

datasets.stocks package

Submodules

datasets.stocks.get_data module

Implements dataloaders for the robotics datasets.

class datasets.stocks.get_data.**Grouping**(*n_groups*)

Bases: Module

Module to collate stock data.

__init__(*n_groups*)

Instantiate grouper. n_groups determines the number of groups.

forward(*x*)

Apply grouper to input data.

Parameters
x (*torch.Tensor*) – Input data.

Returns
List of outputs

Return type
list

training: bool

datasets.stocks.get_data.**get_dataloader**(*stocks*, *input_stocks*, *output_stocks*, *batch_size=16*, *train_shuffle=True*, *start_date=datetime.datetime(2000, 6, 1, 0, 0)*, *end_date=datetime.datetime(2021, 2, 28, 0, 0)*, *window_size=500*, *val_split=3200*, *test_split=3700*, *modality_first=True*, *cuda=True*)

Generate dataloader for stock data.

Parameters

- **stocks** (*list*) – List of strings of stocks to grab data for.
- **input_stocks** (*list*) – List of strings of input stocks
- **output_stocks** (*list*) – List of strings of output stocks
- **batch_size** (*int, optional*) – Batchsize. Defaults to 16.

- **train_shuffle** (*bool, optional*) – Whether to shuffle training dataloader or not. Defaults to True.
- **start_date** (*datetime, optional*) – Start-date to grab data from. Defaults to datetime.datetime(2000, 6, 1).
- **end_date** (*datetime, optional*) – End-date to grab data from. Defaults to datetime.datetime(2021, 2, 28).
- **window_size** (*int, optional*) – Window size. Defaults to 500.
- **val_split** (*int, optional*) – Number of samples in validation split. Defaults to 3200.
- **test_split** (*int, optional*) – Number of samples in test split. Defaults to 3700.
- **modality_first** (*bool, optional*) – Whether to make modality the first index or not. Defaults to True.
- **cuda** (*bool, optional*) – Whether to load data to cuda objects or not. Defaults to True.

Returns
Tuple of training data-loader, test data-loader, and validation data-loader.

Return type
tuple

Module contents

4.1.2 Module contents

CHAPTER

FIVE

GENERAL EVALUATION SCRIPTS

5.1 eval_scripts package

5.1.1 Submodules

5.1.2 eval_scripts.complexity module

eval_scripts.complexity.**all_in_one_test**(*testprocess*, *testmodules*)

eval_scripts.complexity.**all_in_one_train**(*trainprocess*, *trainmodules*)

eval_scripts.complexity.**getallparams**(*li*)

5.1.3 eval_scripts.performance module

eval_scripts.performance.**AUPRC**(*pts*)

eval_scripts.performance.**accuracy**(*truth*, *pred*)

eval_scripts.performance.**eval_affect**(*truths*, *results*, *exclude_zero=True*)

eval_scripts.performance.**f1_score**(*truth*, *pred*, *average*)

eval_scripts.performance.**ptsort**(*tu*)

5.1.4 eval_scripts.robustness module

eval_scripts.robustness.**effective_robustness**(*robustness_result*, *task*)

Compute the effective robustenss metric given the performance of the method on the task.

eval_scripts.robustness.**effective_robustness_helper**(*robustness_result*, *task*)

Helper function that computes the effective robustness metric as the performance difference compared to late fusion method.

Parameters

- **robustness_result** – Performance of the method on datasets applied with different level of noises.
- **task** – Name of the task on which the method is evaluated.

eval_scripts.robustness.**get_robustness_metric**(*robustness_result*, *task*, *metric*)

Compute robustness metric given specific method performance and the task.

Parameters

- **robustness_result** – Performance of the method on datasets applied with different level of noises.
- **task** – Name of the task on which the method is evaluated.
- **metric** – Type of robustness metric to be computed. ("effective" / "relative")

eval_scripts.robustness.**maxmin_normalize**(*result*, *task*)

Normalize the metric for robustness comparison across all methods.

Parameters

- **result** – Un-normalized robustness metrics of all methods on the given task.
- **task** – Name of the task.

eval_scripts.robustness.**relative_robustness**(*robustness_result*, *task*)

Compute the relative robustenss metric given the performance of the method on the task.

eval_scripts.robustness.**relative_robustness_helper**(*robustness_result*, *task*)

Helper function that computes the relative robustness metric as the area under the performance curve.

Parameters

robustness_result – Performance of the method on datasets applied with different level of noises.

eval_scripts.robustness.**single_plot**(*robustness_result*, *task*, *xlabel*, *ylabel*, *fig_name*, *method*)

Produce performance vs. robustness plot of a single method.

Parameters

- **robustness_result** – Performance of the method on dataset applied with different level of noises.
- **task** – Name of the task on which the method is evaluated.
- **xlabel** – Label of x-axis to be appeared in the plot.
- **ylabel** – Label of y-axis to be appeared in the plot.
- **fig_name** – Name of plot to be saved.
- **method** – Name of the method.

5.1.5 Module contents

CHAPTER

SIX

MULTIMODAL FUSION TECHNIQUES

6.1 fusions package

6.1.1 Subpackages

fusions.finance package

Submodules

fusions.finance.early_fusion module

fusions.finance.late_fusion module

Module contents

fusions.mult package

Subpackages

fusions.mult.modules package

Submodules

fusions.mult.modules.multihead_attention module

fusions.mult.modules.position_embedding module

fusions.mult.modules.transformer module

Module contents

Submodules

fusions.mult.mult module

Module contents

Implements the MultimodalTransformer Model. See https://github.com/yaohungt/Multimodal-Transformer for more.

fusions.mult.**LayerNorm**(*embedding_dim*)

Generate LayerNorm Layer with given parameters.

Parameters
embedding_dim (*int*) – Embedding dimension

Returns
Initialized LayerNorm Module

Return type
nn.Module

fusions.mult.**Linear**(*in_features*, *out_features*, *bias=True*)

Generate Linear Layer with given parameters and Xavier initialization.

Parameters

- **in_features** (*int*) – Number of input features
- **out_features** (*int*) – Number of output features
- **bias** (*bool, optional*) – Whether to include a bias term or not. Defaults to True.

Returns
Initialized Linear Module.

Return type
nn.Module

class fusions.mult.**MULTModel**(*n_modalities*, *n_features*, *hyp_params=<class 'fusions.mult.MULTModel.DefaultHyperParams'>*)

Bases: Module

Implements the MultimodalTransformer Model.

See https://github.com/yaohungt/Multimodal-Transformer for more.

class **DefaultHyperParams**

Bases: object

Set default hyperparameters for the model.

all_steps = False

attn_dropout = 0.1

```
attn_dropout_modalities = [0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0,
0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0,
0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0,
0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0,
0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0,
0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0,
0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0,
0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0,
0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0,
0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0,
0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0,
0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0,
0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0,
0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0,
0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0,
0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0,
0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0,
0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0,
0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0,
0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0,
0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0,
0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0,
0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0,
0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0,
0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0,
0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0,
0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0,
0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0,
0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0,
0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0,
0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0,
0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0,
0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0,
0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0,
0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0,
0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0,
0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0,
0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0,
0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0,
0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0,
0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0,
0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0,
0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0,
0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0,
0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0,
0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0,
0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0,
0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0,
0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0,
0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0,
0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0,
0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0,
0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0,
0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0,
0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0,
0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0,
0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0,
0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0,
0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0,
```

attn_mask = True

embed_dim = 9

embed_dropout = 0.25

layers = 3

num_heads = 3

out_dropout = 0.0

output_dim = 1

relu_dropout = 0.1

res_dropout = 0.1

__init__(*n_modalities*, *n_features*, *hyp_params=<class 'fusions.mult.MULTModel.DefaultHyperParams'>*)

Construct a MulT model.

forward(*x*)

Apply MultModel Module to Layer Input.

Parameters

x – layer input. Has size n_modalities * [batch_size, seq_len, n_features]

get_network(*mod1*, *mod2*, *mem*, *layers=-1*)

Create TransformerEncoder network from layer information.

training: bool

class fusions.mult.SinusoidalPositionalEmbedding(*embedding_dim*, *padding_idx=0*, *left_pad=0*)

Bases: Module

This module produces sinusoidal positional embeddings of any length.

Padding symbols are ignored, but it is necessary to specify whether padding is added on the left side (left_pad=True) or right side (left_pad=False).

__init__(*embedding_dim*, *padding_idx=0*, *left_pad=0*)

Instantiate SinusoidalPositionalEmbedding Module.

Parameters

- **embedding_dim** (*int*) – Embedding dimension
- **padding_idx** (*int, optional*) – Padding index. Defaults to 0.
- **left_pad** (*int, optional*) – Whether to pad from the left or not. Defaults to 0.

forward(*input*)

Apply PositionalEncodings to Input.

Input is expected to be of size [bsz x seqlen].

Parameters

input (*torch.Tensor*) – Layer input

Returns

Layer output

Return type
torch.Tensor

static get_embedding(*num_embeddings*, *embedding_dim*, *padding_idx=None*)

Build sinusoidal embeddings.

This matches the implementation in tensor2tensor, but differs slightly from the description in Section 3.5 of "Attention Is All You Need".

training: bool

class fusions.mult.**TransformerEncoder**(*embed_dim*, *num_heads*, *layers*, *attn_dropout=0.0*, *relu_dropout=0.0*, *res_dropout=0.0*, *embed_dropout=0.0*, *attn_mask=False*)

Bases: Module

Transformer encoder consisting of *args.encoder_layers* layers.

Each layer is a *TransformerEncoderLayer*.

Parameters

- **embed_tokens** (*torch.nn.Embedding*) – input embedding
- **num_heads** (*int*) – number of heads
- **layers** (*int*) – number of layers
- **attn_dropout** (*float*) – dropout applied on the attention weights
- **relu_dropout** (*float*) – dropout applied on the first layer of the residual block
- **res_dropout** (*float*) – dropout applied on the residual block
- **attn_mask** (*bool*) – whether to apply mask on the attention weights

__init__(*embed_dim*, *num_heads*, *layers*, *attn_dropout=0.0*, *relu_dropout=0.0*, *res_dropout=0.0*, *embed_dropout=0.0*, *attn_mask=False*)

Initialize Transformer Encoder.

Parameters

- **embed_dim** (*int*) – Embedding dimension
- **num_heads** (*int*) – Number of heads
- **layers** (*int*) – Number of layers
- **attn_dropout** (*float, optional*) – Probability of dropout in attention mechanism. Defaults to 0.0.
- **relu_dropout** (*float, optional*) – Probability of dropout after ReLU. Defaults to 0.0.
- **res_dropout** (*float, optional*) – Probability of dropout in residual layer. Defaults to 0.0.
- **embed_dropout** (*float, optional*) – Probability of dropout in embedding layer. Defaults to 0.0.
- **attn_mask** (*bool, optional*) – Whether to apply a mask to the attention or not. Defaults to False.

forward(*x_in*, *x_in_k=None*, *x_in_v=None*)

Apply Transformer Encoder to layer input.

Parameters

- **x_in** (*FloatTensor*) – embedded input of shape *(src_len, batch, embed_dim)*
- **x_in_k** (*FloatTensor*) – embedded input of shape *(src_len, batch, embed_dim)*
- **x_in_v** (*FloatTensor*) – embedded input of shape *(src_len, batch, embed_dim)*

Returns

- **encoder_out** (Tensor): the last encoder layer's output of shape *(src_len, batch, embed_dim)*
- **encoder_padding_mask** (ByteTensor): the positions of padding elements of shape *(batch, src_len)*

Return type
dict

training: bool

class fusions.mult.**TransformerEncoderLayer**(*embed_dim*, *num_heads=4*, *attn_dropout=0.1*, *relu_dropout=0.1*, *res_dropout=0.1*, *attn_mask=False*)

Bases: Module

Implements encoder layer block.

In the original paper each operation (multi-head attention or FFN) is postprocessed with: *dropout -> add residual -> layernorm*. In the tensor2tensor code they suggest that learning is more robust when preprocessing each layer with layernorm and postprocessing with: *dropout -> add residual*. We default to the approach in the paper, but the tensor2tensor approach can be enabled by setting *args.encoder_normalize_before* to True.

__init__(*embed_dim*, *num_heads=4*, *attn_dropout=0.1*, *relu_dropout=0.1*, *res_dropout=0.1*, *attn_mask=False*)

Instantiate TransformerEncoderLayer Module.

Parameters

- **embed_dim** (*int*) – Embedding dimension
- **num_heads** (*int, optional*) – Number of heads. Defaults to 4.
- **attn_dropout** (*float, optional*) – Dropout for attention mechanism. Defaults to 0.1.
- **relu_dropout** (*float, optional*) – Dropout after ReLU. Defaults to 0.1.
- **res_dropout** (*float, optional*) – Dropout after residual layer. Defaults to 0.1.
- **attn_mask** (*bool, optional*) – Whether to apply an attention mask or not. Defaults to False.

forward(*x*, *x_k=None*, *x_v=None*)

Apply TransformerEncoderLayer to Layer Input.

Parameters

- **x** (*Tensor*) – input to the layer of shape *(seq_len, batch, embed_dim)*
- **encoder_padding_mask** (*ByteTensor*) – binary ByteTensor of shape *(batch, src_len)* where padding elements are indicated by 1.
- **x_k** (*Tensor*) – same as x
- **x_v** (*Tensor*) – same as x

Returns
encoded output of shape *(batch, src_len, embed_dim)*

training: bool

fusions.mult.**buffered_future_mask**(*tensor*, *tensor2=None*)

Generate buffered future mask.

Parameters

- **tensor** (*torch.Tensor*) – Tensor to initialize mask from.
- **tensor2** (*torch.Tensor, optional*) – Tensor to initialize target mask from. Defaults to None.

Returns
Buffered future mask.

Return type
torch.Tensor

fusions.mult.**fill_with_neg_inf**(*t*)

FP16-compatible function that fills a tensor with -inf.

fusions.mult.**make_positions**(*tensor*, *padding_idx*, *left_pad*)

Replace non-padding symbols with their position numbers.

Position numbers begin at padding_idx+1. Padding symbols are ignored, but it is necessary to specify whether padding is added on the left side (left_pad=True) or right side (left_pad=False).

Parameters

- **tensor** (*torch.Tensor*) – Tensor to generate padding on.
- **padding_idx** (*int*) – Position numbers start at padding_idx + 1
- **left_pad** (*bool*) – Whether to pad from the left or from the right.

Returns
Padded output

Return type
torch.Tensor

fusions.robotics package

Submodules

fusions.robotics.models_utils module

fusions.robotics.sensor_fusion module

Module contents

6.1.2 Submodules

6.1.3 fusions.MCTN module

Implements MCTN for Fusions.

class fusions.MCTN.**Attention**(*hidden_size*)

Bases: Module

Implements Attention Mechanism for MCTN.

__init__(*hidden_size*)

Initialize Attention Mechanism for MCTN.

Parameters
hidden_size (*int*) – Hidden Size of Attention Layer.

forward(*hidden*, *encoder_outputs*)

Apply Attention to Input, with Hidden Layers.

Parameters

- **hidden** – Initial hidden state.
- **encoder_outputs** – Outputs of Encoder object.

Returns
Output of Attention layer

Return type
output

training: bool

class fusions.MCTN.**Decoder**(*hidden_size*, *output_size*, *n_layers=1*, *dropout=0.2*)

Bases: Module

Apply a gated GRU to decode the input vector.

Paper/Code Sourced From: https://arxiv.org/pdf/1812.07809.pdf.

__init__(*hidden_size*, *output_size*, *n_layers=1*, *dropout=0.2*)

Initialize Decoder Mechanism for MCTN.

Parameters

- **hidden_size** (*int*) – Size of hidden layer.
- **output_size** (*int*) – Size of output layer
- **n_layers** (*int, optional*) – Number of layers in encoder. Defaults to 1.
- **dropout** (*float, optional*) – Dropout percentage. Defaults to 0.2.

forward(*input*, *last_hidden*, *encoder_outputs*)

Apply Decoder Mechanism for MCTN.

Parameters

- **input** – Input to MCTN Mechanism.
- **last_hidden** – Last hidden layer input.
- **encoder_outputs** – Output of Encoder object.

Returns
Output of this module.

Return type
output

training: bool

class fusions.MCTN.**Encoder**(*input_size*, *hidden_size*, *n_layers=1*, *dropout=0.2*)

Bases: Module

Apply a gated GRU to encode the input vector.

Paper/Code Sourced From: https://arxiv.org/pdf/1812.07809.pdf.

__init__(*input_size*, *hidden_size*, *n_layers=1*, *dropout=0.2*)

Create Encoder.

Parameters

- **input_size** – Encoder input size
- **hidden_size** – Hidden state size for internal GRU
- **n_layers** – Number of layers in recurrent unit
- **dropout** – Dropout Probability

forward(*src*, *hidden=None*)

Apply Encoder to input.

Parameters

- **src** – Encoder Input
- **hidden** – Encoder Hidden State

training: bool

class fusions.MCTN.**L2_MCTN**(*seq2seq_0*, *seq2seq_1*, *regression_encoder*, *head*, *p=0.2*)

Bases: Module

Implements 2-Output MCTN.

__init__(*seq2seq_0*, *seq2seq_1*, *regression_encoder*, *head*, *p=0.2*)

Initialize L2_MCTN.

Parameters

- **seq2seq_0** (*nn.Module*) – Seq2Seq Module converting input to target 1.
- **seq2seq_1** (*nn.Module*) – Seq2Seq Module converting input to target 2.
- **regression_encoder** (*nn.Module*) – Encoder applied to joint embedding.
- **head** (*nn.Module*) – Head module.
- **p** (*float, optional*) – Dropout percentage. Defaults to 0.2.

forward(*src*, *trg0=None*, *trg1=None*)

Apply L2_MCTN to input.

Parameters

- **src** (*torch.Tensor*) – Input tensor.
- **trg0** (*torch.Tensor, optional*) – Target output for Seq2Seq Module 1 Teacher Forcing. Defaults to None.
- **trg1** (*torch.Tensor, optional*) – Target output for Seq2Seq Module 2 Teacher Forcing. Defaults to None.

Returns
Output for L2_MCTN instance.

Return type
torch.Tensor

`training: bool`

`class fusions.MCTN.MCTN(seq2seq, regression_encoder, head, p=0.2)`

Bases: `Module`

Implements MCTN.

`__init__(seq2seq, regression_encoder, head, p=0.2)`

Initialize MCTN object.

Parameters

- **seq2seq** (*nn.Module*) – Seq2Seq module for MCTN.
- **regression_encoder** (*nn.Module*) – Encoder module for MCTN.
- **head** (*nn.Module*) – Head for MCTN.
- **p** (*float, optional*) – Dropout probability. Defaults to 0.2.

`forward(src, trg=None)`

Apply Cyclic Joint Embedding in MCTN.

Parameters

- **src** (*torch.Tensor*) – Input Tensor
- **trg** (*torch.Tensor, optional*) – Output Tensor for Teacher-Forcing. Defaults to None.

Returns
Output after applying MCTN.

Return type
torch.Tensor

`training: bool`

`class fusions.MCTN.Seq2Seq(encoder, decoder)`

Bases: `Module`

Implements a Seq2Seq Layer for MCTN.

`__init__(encoder, decoder)`

Initialize Seq2Seq Module.

Parameters

- **encoder** (*nn.Module*) – Encoder for the Seq2Seq Layer.
- **decoder** (*nn.Module*) – Decoder for the Seq2Seq Layer.

`forward(src, trg, teacher_forcing_ratio=0.5)`

Apply Seq2Seq Module to Input.

Parameters

- **src** (*torch.Tensor*) – Seq2Seq Input
- **trg** (*torch.Tensor*) – Seq2Seq Output for Teacher Forcing.

- **teacher_forcing_ratio** (*float, optional*) – Teacher Forcing Ratio. Set to 0 when evaluating. Defaults to 0.5.

Returns
description

Return type
type

training: bool

6.1.4 fusions.MVAE module

Implements MVAE.

class fusions.MVAE.**ProductOfExperts**(*size*)

Bases: Module

Return parameters for product of independent experts.

See https://arxiv.org/pdf/1410.7827.pdf for equations.

__init__(*size*)

Initialize Product of Experts Object.

Parameters
size (*int*) – Size of Product of Experts Layer

forward(*mus*, *logvars*, *eps=1e-08*)

Apply Product of Experts Layer.

Parameters

- **mus** (*torch.Tensor*) – torch.Tensor of Mus
- **logvars** (*torch.Tensor*) – torch.Tensor of Logvars
- **eps** (*float, optional*) – Epsilon for log-exponent trick. Defaults to 1e-8.

Returns
Output of PoE layer.

Return type
torch.Tensor, torch.Tensor

training: bool

class fusions.MVAE.**ProductOfExperts_Zipped**(*size*)

Bases: Module

Return parameters for product of independent experts.

See https://arxiv.org/pdf/1410.7827.pdf for equations.

__init__(*size*)

Initialize Product of Experts Object.

Parameters
size (*int*) – Size of Product of Experts Layer

forward(*zipped*, *eps=1e-08*)

Apply Product of Experts Layer.

Parameters

- **mus** (*torch.Tensor*) – torch.Tensor of Mus
- **logvars** (*torch.Tensor*) – torch.Tensor of Logvars
- **eps** (*float, optional*) – Epsilon for log-exponent trick. Defaults to 1e-8.

Returns
Output of PoE layer.

Return type
torch.Tensor, torch.Tensor

training: **bool**

6.1.5 fusions.common_fusions module

Implements common fusion patterns.

class fusions.common_fusions.**Concat**

Bases: Module

Concatenation of input data on dimension 1.

__init__()

Initialize Concat Module.

forward(*modalities*)

Forward Pass of Concat.

Parameters
modalities – An iterable of modalities to combine

training: **bool**

class fusions.common_fusions.**ConcatEarly**

Bases: Module

Concatenation of input data on dimension 2.

__init__()

Initialize ConcatEarly Module.

forward(*modalities*)

Forward Pass of ConcatEarly.

Parameters
modalities – An iterable of modalities to combine

training: **bool**

class fusions.common_fusions.**ConcatWithLinear**(*input_dim*, *output_dim*, *concat_dim=1*)

Bases: Module

Concatenates input and applies a linear layer.

__init__(*input_dim*, *output_dim*, *concat_dim=1*)

Initialize ConcatWithLinear Module.

Parameters

- **input_dim** – The input dimension for the linear layer
- **output_dim** – The output dimension for the linear layer

Concat_dim
The concatentation dimension for the modalities.

forward(*modalities*)

Forward Pass of Stack.

Parameters
modalities – An iterable of modalities to combine

training: bool

class fusions.common_fusions.**EarlyFusionTransformer**(*n_features*)

Bases: Module

Implements a Transformer with Early Fusion.

__init__(*n_features*)

Initialize EarlyFusionTransformer Object.

Parameters
n_features (*int*) – Number of features in input.

embed_dim = 9

forward(*x*)

Apply EarlyFusion with a Transformer Encoder to input.

Parameters
x (*torch.Tensor*) – Input Tensor

Returns
Layer Output

Return type
torch.Tensor

training: bool

class fusions.common_fusions.**LateFusionTransformer**(*embed_dim=9*)

Bases: Module

Implements a Transformer with Late Fusion.

__init__(*embed_dim=9*)

Initialize LateFusionTransformer Layer.

Parameters
embed_dim (*int, optional*) – Size of embedding layer. Defaults to 9.

forward(*x*)

Apply LateFusionTransformer Layer to input.

Parameters
x (*torch.Tensor*) – Layer input

Returns
Layer output

Return type
torch.Tensor

`training: bool`

`class fusions.common_fusions.LowRankTensorFusion(input_dims, output_dim, rank, flatten=True)`

Bases: `Module`

Implementation of Low-Rank Tensor Fusion.

See https://github.com/Justin1904/Low-rank-Multimodal-Fusion for more information.

`__init__(input_dims, output_dim, rank, flatten=True)`

Initialize LowRankTensorFusion object.

Parameters

- **input_dims** – list or tuple of integers indicating input dimensions of the modalities
- **output_dim** – output dimension
- **rank** – a hyperparameter of LRTF. See link above for details
- **flatten** – Boolean to dictate if output should be flattened or not. Default: True

`forward(modalities)`

Forward Pass of Low-Rank TensorFusion.

Parameters
modalities – An iterable of modalities to combine.

`training: bool`

`class fusions.common_fusions.MultiplicativeInteractions2Modal(input_dims, output_dim, output, flatten=False, clip=None, grad_clip=None, flip=False)`

Bases: `Module`

Implements 2-way Modal Multiplicative Interactions.

`__init__(input_dims, output_dim, output, flatten=False, clip=None, grad_clip=None, flip=False)`

Parameters

- **input_dims** – list or tuple of 2 integers indicating input dimensions of the 2 modalities
- **output_dim** – output dimension
- **output** – type of MI, options from 'matrix3D','matrix','vector','scalar'
- **flatten** – whether we need to flatten the input modalities
- **clip** – clip parameter values, None if no clip
- **grad_clip** – clip grad values, None if no clip
- **flip** – whether to swap the two input modalities in forward function or not

`forward(modalities)`

Forward Pass of MultiplicativeInteractions2Modal.

Parameters
modalities – An iterable of modalities to combine.

training: bool

class fusions.common_fusions.**MultiplicativeInteractions3Modal**(*input_dims*, *output_dim*, *task=None*)

Bases: Module

Implements 3-Way Modal Multiplicative Interactions.

__init__(*input_dims*, *output_dim*, *task=None*)

Initialize MultiplicativeInteractions3Modal object.

Parameters

- **input_dims** – list or tuple of 3 integers indicating sizes of input
- **output_dim** – size of outputs
- **task** – Set to "affect" when working with social data.

forward(*modalities*)

Forward Pass of MultiplicativeInteractions3Modal.

Parameters
modalities – An iterable of modalities to combine.

training: bool

class fusions.common_fusions.**NLgate**(*thw_dim*, *c_dim*, *tf_dim*, *q_linear=None*, *k_linear=None*, *v_linear=None*)

Bases: Module

Implements of Non-Local Gate-based Fusion.

See section F4 of https://arxiv.org/pdf/1905.12681.pdf for details

__init__(*thw_dim*, *c_dim*, *tf_dim*, *q_linear=None*, *k_linear=None*, *v_linear=None*)

q_linear, k_linear, v_linear are none if no linear layer applied before q,k,v.

Otherwise, a tuple of (indim,outdim) is required for each of these 3 arguments.

Parameters

- **thw_dim** – See paper
- **c_dim** – See paper
- **tf_dim** – See paper
- **q_linear** – See paper
- **k_linear** – See paper
- **v_linear** – See paper

forward(*x*)

Apply Low-Rank TensorFusion to input.

Parameters
x – An iterable of modalities to combine.

training: bool

class fusions.common_fusions.**Stack**

Bases: Module

Stacks modalities together on dimension 1.

__init__()

Initialize Stack Module.

forward(*modalities*)

Forward Pass of Stack.

Parameters

modalities – An iterable of modalities to combine

training: bool

class fusions.common_fusions.**TensorFusion**

Bases: Module

Implementation of TensorFusion Networks.

See https://github.com/Justin1904/TensorFusionNetworks/blob/master/model.py for more and the original code.

__init__()

Instantiates TensorFusion Network Module.

forward(*modalities*)

Forward Pass of TensorFusion.

Parameters

modalities – An iterable of modalities to combine.

training: bool

6.1.6 fusions.searchable module

Implements fusion network structure for MFAS.

See https://github.com/slyviacassell/_MFAS/tree/master/models for hyperparameter details.

class fusions.searchable.**Searchable**(*layered_encoders*, *rep_size*, *classes*, *conf*, *sub_sizes*, *alphas=False*)

Bases: Module

Implements MFAS's Searchable fusion module.

__init__(*layered_encoders*, *rep_size*, *classes*, *conf*, *sub_sizes*, *alphas=False*)

Instantiate Searchable Module.

Parameters

- **layered_encoders** (*List*) – List of nn.Modules for each encoder
- **rep_size** (*int*) – Representation size from unimodals
- **classes** (*int*) – Number of classes
- **conf** (*Config*) – Config instance that generates the layer in question.
- **sub_sizes** (*int*) – Sub sizes
- **alphas** (*bool, optional*) – Whether to generate alphas. Defaults to False.

alphasgen()

Generate alpha-layers if stated to do so.

central_params()

Define parameters for central module.

fcs()

Create fullyconnected layers given config.

forward(*inputs*)

Apply Searchable Module to Layer Inputs.

Parameters
inputs (*torch.Tensor*) – List of input tensors

Returns
Layer Output

Return type
torch.Tensor

training: bool

fusions.searchable.**get_possible_layer_configurations**(*max_labels*)

Generate possible layer configurations.

Parameters
max_labels (*int*) – Maximum number of labels

Returns
List of Configuration Instances.

Return type
list

fusions.searchable.**train_sampled_models**(*sampled_configurations*, *searchable_type*, *dataloaders*, *use_weightsharing*, *device*, *unimodal_files*, *rep_size*, *classes*, *sub_sizes*, *batch_size*, *epochs*, *eta_max*, *eta_min*, *Ti*, *Tm*, *return_model=False*, *premodels=False*, *preaccuracies=False*, *train_only_central_params=True*, *state_dict={}*)

Train sampled configurations from MFAS.

Parameters

- **sampled_configurations** (*List[config]*) – List of configurations to train on.
- **searchable_type** (*rn*) – Function to create full model from configurations.
- **dataloaders** (*List[torch.util.data.DataLoaders]*) – List of dataloaders to train on.
- **use_weightsharing** (*bool*) – Whether to use weightsharing or not.
- **device** (*torch.device*) – Device to train on.
- **unimodal_files** (*List[path]*) – List of unimodal encoder paths to train on.
- **rep_size** (*int*) – Internal Representation Size
- **classes** (*int*) – Number of classes
- **sub_sizes** (*int*) – Sub sizes.
- **batch_size** (*int*) – Batch size to train on.

- **epochs** (*int*) – Number of epochs to train on.
- **eta_max** (*float*) – Minimum eta of LRCosineAnnealingScheduler
- **eta_min** (*float*) – Maximum eta of LRCosineAnnealingScheduler
- **Ti** (*float*) – Ti for LRCosineAnnealingScheduler
- **Tm** (*float*) – Tm for LRCosineAnnealingScheduler
- **return_model** (*bool, optional*) – Whether to return the trained module as nn.Module. Defaults to False.
- **premodels** (*bool, optional*) – Whether there are pre-trained unimodal models or not. Defaults to False.
- **preaccuracies** (*bool, optional*) – (Unused). Defaults to False.
- **train_only_central_params** (*bool, optional*) – Whether to train only central parameters or not. Defaults to True.
- **state_dict** (*_type_, optional*) – (unused). Defaults to dict().

Returns
List of model accuracies.

Return type
List

fusions.searchable.**train_track_acc**(*model*, *criteria*, *optimizer*, *scheduler*, *dataloaders*, *dataset_sizes*, *device=None*, *num_epochs=200*, *verbose=False*, *multitask=False*)

Get best accuracy for model when training on a set of dataloaders.

Parameters

- **model** (*nn.Module*) – Model to train on.
- **criteria** (*nn.Module*) – Loss function.
- **optimizer** (*nn.optim.Optimizer*) – Optimizer instance
- **scheduler** (*nn.optim.Scheduler*) – LRScheduler to use.
- **dataloaders** (*List*) – List of dataloaders to train on.
- **dataset_sizes** (*List*) – List of the sizes of the datasets
- **device** (*torch.device, optional*) – Device to train on. Defaults to None.
- **num_epochs** (*int, optional*) – Number of epochs to train on. Defaults to 200.
- **verbose** (*bool, optional*) – (Unused) Defaults to False.
- **multitask** (*bool, optional*) – Whether to train as a multitask setting. Defaults to False.

Returns
Best accuracy when training.

Return type
float

6.1.7 Module contents

CHAPTER

SEVEN

OBJECTIVE FUNCTIONS

7.1 objective_functions package

7.1.1 Submodules

7.1.2 objective_functions.cca module

Implements losses for CCA.

class objective_functions.cca.**CCALoss**(*outdim_size*, *use_all_singular_values*, *device*)

Bases: Module

Implements Loss for CCA.

__init__(*outdim_size*, *use_all_singular_values*, *device*)

Initialize CCALoss Object.

Parameters

- **outdim_size** (*int*) – Output Dimension for TopK
- **use_all_singular_values** (*bool*) – Whether to include all singular values in the loss.
- **device** (*torch.device*) – What device to place this module on. Must agree with model.

forward(*H1*, *H2*)

Apply the CCALoss as described in the paper to inputs H1 and H2.

Parameters

- **H1** (*torch.Tensor*) – Tensor corresponding to the first random variable in CCA.
- **H2** (*torch.Tensor*) – Tensor corresponding to the second random variable in CCA.

Returns

CCALoss for this pair.

Return type

torch.Tensor

training: bool

7.1.3 objective_functions.contrast module

Implement objectives for contrastive loss.

class objective_functions.contrast.**AliasMethod**(*probs*)

Bases: object

Initializes a generic method to sample from arbritrary discrete probability methods.

Sourced From https://hips.seas.harvard.edu/blog/2013/03/03/the-alias-method-efficient-sampling-with-many-discrete-outcomes/
Alternatively, look here for more details: http://cgi.cs.mcgill.ca/~enewel3/posts/alias-method/index.html.

__init__(*probs*)

Initialize AliasMethod object.

Parameters
probs (*list[int]*) – List of probabilities for each object. Can be greater than 1, but will be normalized.

cuda()

Generate CUDA version of self, for GPU-based sampling.

draw(*N*)

Draw N samples from multinomial dkstribution, based on given probability array.

Parameters
N – number of samples

Returns
samples

class objective_functions.contrast.**MultiSimilarityLoss**

Bases: Module

Implements MultiSimilarityLoss.

__init__()

Initialize MultiSimilarityLoss Module.

forward(*feats*, *labels*)

Apply MultiSimilarityLoss to Tensor Inputs.

Parameters

- **feats** (*torch.Tensor*) – Features
- **labels** (*torch.Tensor*) – Labels

Returns
Loss output.

Return type
torch.Tensor

training: bool

class objective_functions.contrast.**NCEAverage**(*inputSize*, *outputSize*, *K*, *T=0.07*, *momentum=0.5*, *use_softmax=False*)

Bases: Module

Implements NCEAverage Loss Function.

__init__(*inputSize*, *outputSize*, *K*, *T=0.07*, *momentum=0.5*, *use_softmax=False*)

Instantiate NCEAverage Loss Function.

Parameters

- **inputSize** (*int*) – Input Size
- **outputSize** (*int*) – Output Size
- **K** (*float*) – K Value. See paper for more.
- **T** (*float, optional*) – T Value. See paper for more. Defaults to 0.07.
- **momentum** (*float, optional*) – Momentum for NCEAverage Loss. Defaults to 0.5.
- **use_softmax** (*bool, optional*) – Whether to use softmax or not. Defaults to False.

forward(*l*, *ab*, *y*, *idx=None*)

Apply NCEAverage Module.

Parameters

- **l** (*torch.Tensor*) – Labels
- **ab** (*torch.Tensor*) – See paper for more.
- **y** (*torch.Tensor*) – True values.
- **idx** (*torch.Tensor, optional*) – See paper for more. Defaults to None.

Returns

description

Return type

type

training: bool

class objective_functions.contrast.**NCECriterion**(*n_data*)

Bases: Module

Implements NCECriterion Loss.

Eq. (12): L_{NCE}

__init__(*n_data*)

Instantiate NCECriterion Loss.

forward(*x*)

Apply NCECriterion to Tensor Input.

Parameters

x (*torch.Tensor*) – Tensor Input

Returns

Loss

Return type

torch.Tensor

training: bool

class objective_functions.contrast.**NCESoftmaxLoss**

Bases: Module

Implements Softmax cross-entropy loss (a.k.a., info-NCE loss in CPC paper).

__init__()

Instantiate NCESoftmaxLoss Module.

forward(*x*)

Apply NCESoftmaxLoss to Layer Input.

Parameters
x (*torch.Tensor*) – Layer Input

Returns
Layer Output

Return type
torch.Tensor

training: **bool**

7.1.4 objective_functions.objectives_for_supervised_learning module

Implements various objectives for supervised learning objectives.

objective_functions.objectives_for_supervised_learning.**CCA_objective**(*out_dim*, *cca_weight=0.001*, *criterion=CrossEntropyLoss()*)

Define loss function for CCA.

Parameters

- **out_dim** – output dimension
- **cca_weight** – weight of cca loss
- **criterion** – criterion for supervised loss

objective_functions.objectives_for_supervised_learning.**MFM_objective**(*ce_weight*, *modal_loss_funcs*, *recon_weights*, *input_to_float=True*, *criterion=CrossEntropyLoss()*)

Define objective for MFM.

Parameters

- **ce_weight** – the weight of simple supervised loss
- **model_loss_funcs** – list of functions that takes in reconstruction and input of each modality and compute reconstruction loss
- **recon_weights** – list of float values indicating the weight of reconstruction loss of each modality
- **criterion** – the loss function for supervised loss (default CrossEntropyLoss)

objective_functions.objectives_for_supervised_learning.**MVAE_objective**(*ce_weight*, *modal_loss_funcs*, *recon_weights*, *input_to_float=True*, *annealing=1.0*, *criterion=CrossEntropyLoss()*)

Define objective for MVAE.

Parameters

- **ce_weight** – the weight of simple supervised loss
- **model_loss_funcs** – list of functions that takes in reconstruction and input of each modality and compute reconstruction loss
- **recon_weights** – list of float values indicating the weight of reconstruction loss of each modality
- **input_to_float** – boolean deciding if we should convert input to float or not.
- **annealing** – the annealing factor, i.e. the weight of kl.
- **criterion** – the loss function for supervised loss (default CrossEntropyLoss)

objective_functions.objectives_for_supervised_learning.**RMFE_object**(*reg_weight=1e-10*, *criterion=BCEWithLogitsLoss()*, *is_packed=False*)

Define loss function for RMFE.

Parameters

- **model** – model used for inference
- **reg_weight** – weight of regularization term
- **criterion** – criterion for supervised loss
- **is_packed** – packed for LSTM or not

objective_functions.objectives_for_supervised_learning.**RefNet_objective**(*ref_weight*, *criterion=CrossEntropyLoss()*, *input_to_float=True*)

Define loss function for RefNet.

Parameters

- **ref_weight** – weight of refiner loss
- **criterion** – criterion for supervised loss
- **input_to_float** – whether to convert input to float or not

7.1.5 objective_functions.recon module

Implements various reconstruction losses for MIMIC MVAE.

objective_functions.recon.**elbo_loss**(*modal_loss_funcs*, *weights*, *annealing=1.0*)

Create wrapper function that computes the model ELBO (Evidence Lower Bound) loss.

objective_functions.recon.**nosigmloss1d**(*a*, *b*)

Get 1D sigmoid loss, WITHOUT applying the sigmoid function to the inputs beforehand.

Parameters

- **a** (*torch.Tensor*) – Predicted output
- **b** (*torch.Tensor*) – True output

Returns
Loss

Return type
torch.Tensor

objective_functions.recon.**recon_weighted_sum**(*modal_loss_funcs*, *weights*)

Create wrapper function that computes the weighted model reconstruction loss.

objective_functions.recon.**sigmloss1d**(*a*, *b*)

Get 1D sigmoid loss, applying the sigmoid function to the inputs beforehand.

Parameters

- **a** (*torch.Tensor*) – Predicted output
- **b** (*torch.Tensor*) – True output

Returns
Loss

Return type
torch.Tensor

objective_functions.recon.**sigmloss1dcentercrop**(*adim*, *bdim*)

Get 1D sigmoid loss, cropping the inputs so that they match in size.

Parameters

- **adim** (*int*) – Predicted output size
- **bdim** (*int*) – True output size. Assumed to have larger size than predicted.

Returns
Loss function, taking in a and b respectively.

Return type
fn

7.1.6 objective_functions.regularization module

Implements the paper: "Removing Bias in Multi-modal Classifiers: Regularization by Maximizing Functional Entropies" NeurIPS 2020.

class objective_functions.regularization.**Perturbation**

Bases: object

Utility class for tensor perturbation techniques.

classmethod get_expanded_logits(*logits: Tensor*, *n_samples: int*, *logits_flg: bool = True*) → Tensor

Perform Softmax and then expand the logits depends on the num_eval_samples :param logits_flg: whether the input is logits or softmax :param logits: tensor holds logits outputs from the model :param n_samples: times to duplicate :return:

classmethod perturb_tensor(*tens: Tensor*, *n_samples: int*, *perturbation: bool = True*) → Tensor

Flatting the tensor, expanding it, perturbing and reconstructing to the original shape.

Note, this function assumes that the batch is the first dimension.

Parameters

- **tens** – Tensor to manipulate.
- **n_samples** – times to perturb
- **perturbation** – False - only duplicating the tensor

Returns
tensor in the shape of [batch, samples * num_eval_samples]

class objective_functions.regularization.**RegParameters**(*lambda_: float = 1e-10*, *norm: float = 2.0*, *estimation: str = 'ent'*, *optim_method: str = 'max_ent'*, *n_samples: int = 10*, *grad: bool = True*)

Bases: object

This class controls all the regularization-related properties

__init__(*lambda_: float = 1e-10*, *norm: float = 2.0*, *estimation: str = 'ent'*, *optim_method: str = 'max_ent'*, *n_samples: int = 10*, *grad: bool = True*)

Initialize RegParameters Object.

Parameters

- **lambda** (*float, optional*) – Lambda value. Defaults to 1e-10.
- **norm** (*float, optional*) – Norm value. Defaults to 2.0.
- **estimation** (*str, optional*) – Regularization estimation. Defaults to 'ent'.
- **optim_method** (*str, optional*) – Optimization method. Defaults to 'max_ent'.
- **n_samples** (*int, optional*) – Number of samples . Defaults to 10.
- **grad** (*bool, optional*) – Whether to regularize gradient or not. Defaults to True.

class objective_functions.regularization.**Regularization**

Bases: object

Class that in charge of the regularization techniques

classmethod get_batch_norm(*grad: Tensor*, *loss: Optional[Tensor] = None*, *estimation: str = 'ent'*) → Tensor

Calculate the expectation of the batch gradient :param loss: :param estimation: :param grad: tensor holds the gradient batch :return: approximation of the required expectation

classmethod get_batch_statistics(*loss: Tensor*, *n_samples: int*, *estimation: str = 'ent'*) → Tensor

Calculate the expectation of the batch gradient :param n_samples: :param loss: :param estimation: :return: Influence expectation

classmethod get_regularization_term(*inf_scores: Tensor*, *norm: float = 2.0*, *optim_method: str = 'max_ent'*) → Tensor

Compute the regularization term given a batch of information scores :param inf_scores: tensor holding a batch of information scores :param norm: defines which norm to use (1 or 2) :param optim_method: Define optimization method (possible methods: "min_ent", "max_ent", "max_ent_minus",

"normalized")

Returns

class objective_functions.regularization.**RegularizationLoss**(*loss: Module*, *model: Module*, *delta: float = 1e-10*, *is_pack: bool = True*)

Bases: Module

Define the regularization loss.

__init__(*loss: Module*, *model: Module*, *delta: float = 1e-10*, *is_pack: bool = True*) → None

Initialize RegularizationLoss Object

Parameters

- **loss** (*torch.nn.Module*) – Loss from which to compare output of model with predicted output
- **model** (*torch.nn.Module*) – Model to apply regularization loss to.
- **delta** (*float, optional*) – Strength of regularization loss. Defaults to 1e-10.
- **is_pack** (*bool, optional*) – Whether samples are packaed or not.. Defaults to True.

forward(*logits*, *inputs*)

Apply RegularizationLoss to input.

Parameters

- **logits** (*torch.Tensor*) – Desired outputs of model
- **inputs** (*torch.Tensor*) – Model Input.

Returns
Regularization Loss for this sample.

Return type
torch.Tensor

training: bool

7.1.7 Module contents

CHAPTER

EIGHT

ROBUSTNESS

8.1 robustness package

8.1.1 Submodules

8.1.2 robustness.all_in_one module

Implements generic modules to apply robustness functions to generic training processses.

robustness.all_in_one.**general_test**(*testprocess*, *filename*, *robustdatasets*, *encoder=False*, *multi_measure=False*)

Test model on noisy data.

Parameters

- **testprocess** – Testing function.
- **filename** – Filename of the saved model.
- **robustdatasets** – A list of dataloaders of noisy test data.
- **encoder** – True if the model has an encoder. (default: False)
- **multi_measure** – True if multiple evaluation metrics are used. (default: False)

robustness.all_in_one.**general_train**(*trainprocess*, *algorithm*, *encoder=False*)

Train model on data.

Parameters

- **trainprocess** – Training function.
- **algorithm** – Algorithm name.
- **encoder** – True if the model has an encoder. (default: False)

Returns

Filename of the best saved model trained on training data.

robustness.all_in_one.**stocks_test**(*num_training*, *models*, *noise_range*, *testprocess*, *encoder=False*)

Test model on noisy stocks data.

Parameters

- **num_training** – >1 to enable multiple training to reduce model variance.
- **models** – Models loaded with pre-trained weights.
- **noisy_range** – A range of noise level, e.g. 0 (no noise) - 1 (completely noisy).

- **testprocess** – Testing function.
- **encoder** – True if the model has an encoder. (default: False)

robustness.all_in_one.**stocks_train**(*num_training*, *trainprocess*, *algorithm*, *encoder=False*)

Train model on stocks data.

Parameters

- **num_training** – >1 to enable multiple training to reduce model variance.
- **trainprocess** – Training function.
- **algorithm** – Algorithm name.
- **encoder** – True if the model has an encoder. (default: False)

Returns

A list of filenames of the best saved models trained on stocks data.

8.1.3 robustness.audio_robust module

Implements audio transformations.

robustness.audio_robust.**add_audio_noise**(*tests*, *noise_level=0.3*, *noises=None*)

Add various types of noise to audio data.

Parameters

- **noise_level** – Probability of randomly applying noise to each audio signal, and standard deviation for gaussian noise, and structured dropout probability.
- **noises** – list of noises to add. # TODO: Change this to use either a list of enums or if statements.

robustness.audio_robust.**additive_white_gaussian_noise**(*signal*, *noise_level*)

Add gaussian white noise to audio signal.

Parameters

- **signal** – Audio signal to permute.
- **noise_level** – standard deviation of the gaussian noise.

robustness.audio_robust.**audio_random_dropout**(*sig*, *p*)

Randomly drop the signal for a single time step.

Parameters

- **signal** – Audio signal to transform.
- **p** – Dropout probability.

robustness.audio_robust.**audio_structured_dropout**(*signal*, *p*, *step=10*)

Randomly drop signal for *step* time steps.

Parameters

- **signal** – Audio signal to permute.
- **p** – Dropout probability.
- **step** – Number of time steps to drop the signal.

8.1.4 robustness.tabular_robust module

Implements tabular data transformations.

robustness.tabular_robust.**add_tabular_noise**(*tests*, *noise_level=0.3*, *drop=True*, *swap=True*)

Add various types of noise to tabular data.

Parameters

- **noise_level** – Probability of randomly applying noise to each element.
- **drop** – Drop elements with probability *noise_level*
- **swap** – Swap elements with probability *noise_level*

robustness.tabular_robust.**drop_entry**(*data*, *p*)

Randomly drop elements in *data* with probability *p*

Parameters

- **data** – Data to drop elements from.
- **p** – Probability of dropping elements.

robustness.tabular_robust.**swap_entry**(*data*, *p*)

Randomly swap adjacent elements in *data* with probability *p*.

Parameters

- **data** – Data to swap elems.
- **p** – Probability of swapping elements.

8.1.5 robustness.text_robust module

Implements text transformations.

robustness.text_robust.**add_text_noise**(*tests*, *noise_level=0.3*, *swap=True*, *rand_mid=True*, *typo=True*, *sticky=True*, *omit=True*)

Add various types of noise to text data.

Parameters

- **noise_level** – Probability of randomly applying noise to a word. (default: 0.1)
- **swap** – Swap two adjacent letters. (default: True)
- **rand_mid** – Randomly permute the middle section of the word, except for the first and last letters. (default: True)
- **typo** – Simulate keyboard typos for the word. (default: True)
- **sticky** – Randomly repeat letters inside a word. (default: True)
- **omit** – Randomly omit some letters from a word (default: True)

robustness.text_robust.**omission**(*word*, *num_omit=1*)

Randomly omit num_omit number of letters of a word.

Parameters

- **word** – word to apply transformations to.
- **num_sticky** – Number of letters to randomly omit.

robustness.text_robust.**qwerty_typo**(*word*)

Randomly replace num_typo number of letters of a word to a one adjacent to it on qwerty keyboard.

Parameters
word – word to apply transformations to.:

robustness.text_robust.**random_mid**(*word*)

Randomly permute the middle chunk of a word (all letters except the first and last letter).

Parameters
word – word to apply transformations to.

robustness.text_robust.**sticky_keys**(*word*, *num_sticky=1*)

Randomly repeat letters of a word once.

Parameters

- **word** – word to apply transformations to.
- **num_sticky** – Number of letters to randomly repeat once.

robustness.text_robust.**swap_letter**(*word*)

Swap two random adjacent letters.

Parameters
word – word to apply transformations to.

8.1.6 robustness.timeseries_robust module

Implements timeseries transformations.

robustness.timeseries_robust.**add_timeseries_noise**(*tests*, *noise_level=0.3*, *gaussian_noise=True*, *rand_drop=True*, *struct_drop=True*)

Add various types of noise to timeseries data.

Parameters

- **noise_level** – Standard deviation of gaussian noise, and drop probability in random drop and structural drop
- **gauss_noise** – Add Gaussian noise to the time series (default: True)
- **rand_drop** – Add randomized dropout to the time series (default: True)
- **struct_drop** – Add randomized structural dropout to the time series (default: True)

robustness.timeseries_robust.**random_drop**(*data*, *p*)

Drop each time series entry independently with probability p.

Parameters

- **data** – Data to process.
- **p** – Probability to drop feature.

robustness.timeseries_robust.**structured_drop**(*data*, *p*)

Drop each time series entry independently with probability p, but drop all modalities if you drop an element.

Parameters

- **data** – Data to process.
- **p** – Probability to drop entire element of time series.

robustness.timeseries_robust.**white_noise**(*data*, *p*)

Add noise sampled from zero-mean Gaussian with standard deviation p at every time step.

Parameters

- **data** – Data to process.
- **p** – Standard deviation of added Gaussian noise.

8.1.7 robustness.visual_robust module

Implements visual transformations.

robustness.visual_robust.**WB**(*img*, *p*)

Randomly change the white-black balance / temperature of an image.

Parameters

- **img** – Input to add noise to.
- **p** – Probability of applying transformation.

robustness.visual_robust.**add_visual_noise**(*tests*, *noise_level=0.3*, *gray=True*, *contrast=True*, *inv=True*, *temp=True*, *color=True*, *s_and_p=True*, *gaus=True*, *rot=True*, *flip=True*, *crop=True*)

Add various types of noise to visual data.

Parameters

- **noise_level** – Probability of randomly applying noise to each audio signal, and standard deviation for gaussian noise, and structured dropout probability.
- **gray** – Boolean flag denoting if grayscale should be applied as a noise type.
- **contrast** – Boolean flag denoting if lowering the contrast should be applied as a noise type.
- **inv** – Boolean flag denoting if inverting the image should be applied as a noise type.
- **temp** – Boolean flag denoting if randomly changing the image's color balance should be applied as a noise type.
- **color** – Boolean flag denoting if randomly tinting the image should be applied as a noise type.
- **s_and_p** – Boolean flag denoting if applying salt and pepper noise should be applied as a noise type.
- **gaus** – Boolean flag denoting if applying Gaussian noise should be applied as a noise type.
- **rot** – Boolean flag denoting if randomly rotating the image should be applied as a noise type.
- **flip** – Boolean flag denoting if randomly flipping the image should be applied as a noise type.
- **crop** – Boolean flag denoting if randomly cropping the image should be applied as a noise type.

robustness.visual_robust.**colorize**(*img*, *p*)

Randomly tint the color of an image using an existing RGB channel.

Parameters

- **img** – Input to add noise to.

- **p** – Probability of applying transformation.

robustness.visual_robust.**gaussian**(*img*, *p*)

Randomly add salt-and-pepper noise to the image.

Parameters

- **img** – Input to add noise to.
- **p** – Probability of applying transformation.

robustness.visual_robust.**grayscale**(*img*, *p*)

Randomly make an image grayscale.

Parameters

- **img** – Input to add noise to.
- **p** – Probability of applying transformation.

robustness.visual_robust.**horizontal_flip**(*img*, *p*)

Randomly flip the image horizontally.

robustness.visual_robust.**inversion**(*img*, *p*)

Randomly invert an image.

Parameters

- **img** – Input to add noise to.
- **p** – Probability of applying transformation.

robustness.visual_robust.**low_contrast**(*img*, *p*)

Randomly reduce the contract of an image.

Parameters

- **img** – Input to add noise to.
- **p** – Probability of applying transformation.

robustness.visual_robust.**periodic**(*img*, *periodic_noise_filename='periodic_noise'*)

Randomly expose the image to periodic pattern/noise.

robustness.visual_robust.**random_crop**(*img*, *p*)

Randomly apply cropping changes.

robustness.visual_robust.**rotate**(*img*, *p*)

Randomly rotate the image by a random angle in [20, 40].

robustness.visual_robust.**salt_and_pepper**(*img*, *p*)

Randomly add salt-and-pepper noise to the image.

Parameters

- **img** – Input to add noise to.
- **p** – Probability of applying transformation.

8.1.8 Module contents

CHAPTER

NINE

TRAINING STRUCTURES

9.1 training_structures package

9.1.1 Submodules

9.1.2 training_structures.MCTN_Level2 module

Implements training pipeline for 2 Level MCTN.

training_structures.MCTN_Level2.**single_test**(*model*, *testdata*, *max_seq_len=20*)

Get accuracy for a single model and dataloader.

Parameters

- **model** (*nn.Module*) – MCTN2 Model
- **testdata** (*torch.utils.data.DataLoader*) – Test Dataloader
- **max_seq_len** (*int, optional*) – Maximum sequence length. Defaults to 20.

Returns

description

Return type

type

training_structures.MCTN_Level2.**test**(*model*, *test_dataloaders_all*, *dataset*, *method_name='My method'*, *is_packed=False*, *criterion=CrossEntropyLoss()*, *task='classification'*, *auprc=False*, *input_to_float=True*, *no_robust=True*)

Test MCTN_Level2 Module on a set of test dataloaders.

Parameters

- **model** (*nn.Module*) – MCTN2 Module
- **test_dataloaders_all** (*list*) – List of dataloaders
- **dataset** (*Dataset*) – Dataset Name
- **method_name** (*str, optional*) – Name of method. Defaults to 'My method'.
- **is_packed** (*bool, optional*) – (unused). Defaults to False.
- **criterion** (*_type_, optional*) – (unused). Defaults to nn.CrossEntropyLoss().
- **task** (*str, optional*) – (unused). Defaults to "classification".
- **auprc** (*bool, optional*) – (unused). Defaults to False.

- **input_to_float** (*bool, optional*) – (unused). Defaults to True.
- **no_robust** (*bool, optional*) – Whether to not apply robustness transformations or not. Defaults to True.

training_structures.MCTN_Level2.**train**(*traindata*, *validdata*, *encoder0*, *decoder0*, *encoder1*, *decoder1*, *reg_encoder*, *head*, *criterion_t0=MSELoss()*, *criterion_c=MSELoss()*, *criterion_t1=MSELoss()*, *criterion_r=L1Loss()*, *max_seq_len=20*, *mu_t0=0.01*, *mu_c=0.01*, *mu_t1=0.01*, *dropout_p=0.1*, *early_stop=False*, *patience_num=15*, *lr=0.0001*, *weight_decay=0.01*, *op_type=<class 'torch.optim.adamw.AdamW'>*, *epoch=100*, *model_save='best_mctn.pt'*, *testdata=None*)

Train a 2-level MCTN Instance

Parameters

- **traindata** (*torch.util.data.DataLoader*) – Training data loader
- **validdata** (*torch.util.data.DataLoader*) – Test data loader
- **encoder0** (*nn.Module*) – Encoder for first Seq2Seq Module
- **decoder0** (*nn.Module*) – Decoder for first SeqSeq Module
- **encoder1** (*nn.Module*) – Encoder for second Seq2Seq Module
- **decoder1** (*nn.Module*) – Decoder for second Seq2Seq Module
- **reg_encoder** (*nn.Module*) – Regularization encoder.
- **head** (*nn.Module*) – Actual classifier.
- **criterion_t0** (*nn.Module, optional*) – Loss function for t0. Defaults to nn.MSELoss().
- **criterion_c** (*nn.Module, optional*) – Loss function for c. Defaults to nn.MSELoss().
- **criterion_t1** (*nn.Module, optional*) – Loss function for t1. Defaults to nn.MSELoss().
- **criterion_r** (*nn.Module, optional*) – Loss function for r. Defaults to nn.L1Loss().
- **max_seq_len** (*int, optional*) – Maximum sequence length. Defaults to 20.
- **mu_t0** (*float, optional*) – mu_t0. Defaults to 0.01.
- **mu_c** (*float, optional*) – mu_c. Defaults to 0.01.
- **mu_t1** (*float, optional*) – mu_t. Defaults to 0.01.
- **dropout_p** (*float, optional*) – Dropout Probability. Defaults to 0.1.
- **early_stop** (*bool, optional*) – Whether to apply early stopping or not. Defaults to False.
- **patience_num** (*int, optional*) – Patience Number for early stopping. Defaults to 15.
- **lr** (*float, optional*) – Learning rate. Defaults to 1e-4.
- **weight_decay** (*float, optional*) – Weight decay coefficient. Defaults to 0.01.
- **op_type** (*torch.optim.Optimizer, optional*) – Optimizer instance. Defaults to torch.optim.AdamW.
- **epoch** (*int, optional*) – Number of epochs. Defaults to 100.

- **model_save** (*str, optional*) – Path to save best model. Defaults to 'best_mctn.pt'.
- **testdata** (*torch.utils.data.DataLoader, optional*) – Data Loader for test data. Defaults to None.

9.1.3 training_structures.Supervised_Learning module

Implements supervised learning training procedures.

class training_structures.Supervised_Learning.**MMDL**(*encoders*, *fusion*, *head*, *has_padding=False*)

Bases: Module

Implements MMDL classifier.

__init__(*encoders*, *fusion*, *head*, *has_padding=False*)

Instantiate MMDL Module

Parameters

- **encoders** (*List*) – List of nn.Module encoders, one per modality.
- **fusion** (*nn.Module*) – Fusion module
- **head** (*nn.Module*) – Classifier module
- **has_padding** (*bool, optional*) – Whether input has padding or not. Defaults to False.

forward(*inputs*)

Apply MMDL to Layer Input.

Parameters
inputs (*torch.Tensor*) – Layer Input

Returns
Layer Output

Return type
torch.Tensor

training: bool

training_structures.Supervised_Learning.**deal_with_objective**(*objective*, *pred*, *truth*, *args*)

Alter inputs depending on objective function, to deal with different objective arguments.

training_structures.Supervised_Learning.**single_test**(*model*, *test_dataloader*, *is_packed=False*, *criterion=CrossEntropyLoss()*, *task='classification'*, *auprc=False*, *input_to_float=True*)

Run single test for model.

Parameters

- **model** (*nn.Module*) – Model to test
- **test_dataloader** (*torch.utils.data.Dataloader*) – Test dataloader
- **is_packed** (*bool, optional*) – Whether the input data is packed or not. Defaults to False.
- **criterion** (*_type_, optional*) – Loss function. Defaults to nn.CrossEntropyLoss().
- **task** (*str, optional*) – Task to evaluate. Choose between "classification", "multiclass", "regression", "posneg-classification". Defaults to "classification".

- **auprc** (*bool, optional*) – Whether to get AUPRC scores or not. Defaults to False.
- **input_to_float** (*bool, optional*) – Whether to convert inputs to float before processing. Defaults to True.

training_structures.Supervised_Learning.**test**(*model*, *test_dataloaders_all*, *dataset='default'*, *method_name='My method'*, *is_packed=False*, *criterion=CrossEntropyLoss()*, *task='classification'*, *auprc=False*, *input_to_float=True*, *no_robust=False*)

Handle getting test results for a simple supervised training loop.

Parameters

- **model** – saved checkpoint filename from train
- **test_dataloaders_all** – test data
- **dataset** – the name of dataset, need to be set for testing effective robustness
- **criterion** – only needed for regression, put MSELoss there

training_structures.Supervised_Learning.**train**(*encoders*, *fusion*, *head*, *train_dataloader*, *valid_dataloader*, *total_epochs*, *additional_optimizing_modules=[]*, *is_packed=False*, *early_stop=False*, *task='classification'*, *optimtype=<class 'torch.optim.rmsprop.RMSprop'>*, *lr=0.001*, *weight_decay=0.0*, *objective=CrossEntropyLoss()*, *auprc=False*, *save='best.pt'*, *validtime=False*, *objective_args_dict=None*, *input_to_float=True*, *clip_val=8*, *track_complexity=True*)

Handle running a simple supervised training loop.

Parameters

- **encoders** – list of modules, unimodal encoders for each input modality in the order of the modality input data.
- **fusion** – fusion module, takes in outputs of encoders in a list and outputs fused representation
- **head** – classification or prediction head, takes in output of fusion module and outputs the classification or prediction results that will be sent to the objective function for loss calculation
- **total_epochs** – maximum number of epochs to train
- **additional_optimizing_modules** – list of modules, include all modules that you want to be optimized by the optimizer other than those in encoders, fusion, head (for example, decoders in MVAE)
- **is_packed** – whether the input modalities are packed in one list or not (default is False, which means we expect input of [tensor(20xmodal1_size),(20xmodal2_size),(20xlabel_size)] for batch size 20 and 2 input modalities)
- **early_stop** – whether to stop early if valid performance does not improve over 7 epochs
- **task** – type of task, currently support “classification”,”regression”,”multilabel”
- **optimtype** – type of optimizer to use
- **lr** – learning rate

- **weight_decay** – weight decay of optimizer
- **objective** – objective function, which is either one of CrossEntropyLoss, MSELoss or BCEWithLogitsLoss or a custom objective function that takes in three arguments: prediction, ground truth, and an argument dictionary.
- **auprc** – whether to compute auprc score or not
- **save** – the name of the saved file for the model with current best validation performance
- **validtime** – whether to show valid time in seconds or not
- **objective_args_dict** – the argument dictionary to be passed into objective function. If not None, at every batch the dict's "reps", "fused", "inputs", "training" fields will be updated to the batch's encoder outputs, fusion module output, input tensors, and boolean of whether this is training or validation, respectively.
- **input_to_float** – whether to convert input to float type or not
- **clip_val** – grad clipping limit
- **track_complexity** – whether to track training complexity or not

9.1.4 training_structures.architecture_search module

Implements training procedure for MFAS.

class training_structures.architecture_search.**ModelSearcher**(*train_data*, *valid_data*, *search_iter*, *num_samples*, *epoch_surrogate*, *temperature_init*, *temperature_final*, *temperature_decay*, *max_progression_levels*, *lr_surrogate*, *device='cuda'*)

Bases: object

Implements MFAS Procedure.

See https://github.com/slyviacassell/_MFAS/blob/master/models/searchable.py for more details.

__init__(*train_data*, *valid_data*, *search_iter*, *num_samples*, *epoch_surrogate*, *temperature_init*, *temperature_final*, *temperature_decay*, *max_progression_levels*, *lr_surrogate*, *device='cuda'*)

Initialize ModelSearcher Object.

Parameters

- **train_data** (*torch.utils.data.DataLoader*) – Training Data Dataloader
- **valid_data** (*torch.utils.data.DataLoader*) – Validation Data Dataloader
- **search_iter** (*int*) – Number of search iterations
- **num_samples** (*int*) – Number of samples
- **epoch_surrogate** (*int*) – Number of epochs per surrogate
- **temperature_init** (*float*) – Initial softmax temperature
- **temperature_final** (*float*) – Final softmax temperature
- **temperature_decay** (*float*) – Softmax temperature decay rate
- **max_progression_levels** (*int*) – Maximum number of progression levels.
- **lr_surrogate** (*float*) – Surrogate learning rate.

- **device** (*str, optional*) – Device to place computation on. Defaults to "cuda".

search(*surrogate*, *use_weightsharing*, *unimodal_files*, *rep_size*, *classes*, *sub_sizes*, *batch_size*, *epochs*, *max_labels*, *eta_max*, *eta_min*, *Ti*, *Tm*, *criterion=MSELoss()*)

Search for the best model using MFAS.

Parameters

- **surrogate** (*nn.Module*) – Surrogate cost function module.
- **use_weightsharing** (*bool*) – Whether to use weight sharing or not.
- **unimodal_files** (*list*) – List of unimodal encoders, pre-trained.
- **rep_size** (*int*) – Dimensionality of unimodal encoder output
- **classes** (*int*) – Number of classes
- **sub_sizes** (*int*) – Sub sizes
- **batch_size** (*int*) – Batch size
- **epochs** (*int*) – Number of epochs
- **max_labels** (*int*) – Maximum number of labels
- **eta_max** (*float*) – eta_max for LRCosineAnnealing Scheduler
- **eta_min** (*float*) – eta_min for LRCosineAnnealingScheduler
- **Ti** (*float*) – Ti for LRCosineAnnealingScheduler
- **Tm** (*float*) – Tm for LRCosineAnnealingScheduler
- **criterion** (*nn.Module, optional*) – Loss function. Defaults to torch.nn.MSELoss().

Returns
Surrogate function training data (i.e. model configs and their performances)

Return type
torch.Tensor

training_structures.architecture_search.**single_test**(*model*, *test_dataloader*, *auprc=False*)

Get accuracy for a single dataloader for MFAS.

Parameters

- **model** (*nn.Module*) – MFAS Model
- **test_dataloader** (*torch.utils.data.DataLoader*) – Test dataloader to sample from
- **auprc** (*bool, optional*) – Whether to get AUPRC scores or not. Defaults to False.

Returns
Dictionary of (metric, value) pairs.

Return type
dict

training_structures.architecture_search.**test**(*model*, *test_dataloaders_all*, *dataset*, *method_name='My method'*, *auprc=False*, *no_robust=False*)

Test MFAS Model.

Parameters

- **model** (*nn.Module*) – Module to test.
- **test_dataloaders_all** (*list*) – List of dataloaders

- **dataset** (*str*) – Name of dataset.
- **method_name** (*str, optional*) – Method name. Defaults to 'My method'.
- **auprc** (*bool, optional*) – Whether to output AUPRC scores or not. Defaults to False.
- **no_robust** (*bool, optional*) – Whether to not apply robustness transformations or not. Defaults to False.

training_structures.architecture_search.**train**(*unimodal_files*, *rep_size*, *classes*, *sub_sizes*, *train_data*, *valid_data*, *surrogate*, *max_labels*, *batch_size=32*, *epochs=3*, *search_iter=3*, *num_samples=15*, *epoch_surrogate=50*, *eta_max=0.001*, *eta_min=1e-06*, *Ti=1*, *Tm=2*, *temperature_init=10.0*, *temperature_final=0.2*, *temperature_decay=4.0*, *max_progression_levels=4*, *lr_surrogate=0.001*, *use_weightsharing=False*)

Train MFAS Model.

See https://github.com/slyviacassell/_MFAS/blob/master/models/searchable.py for more details.

Parameters

- **unimodal_files** (*list[dict]*) – Dictionary of names of files containing pretrained unimodal encoders
- **rep_size** (*int*) – Size of Representation
- **classes** (*int*) – Output Size
- **sub_sizes** (*list of tuples*) – The output size of each layer within the unimodal encoders
- **train_data** (*torch.utils.data.DataLoader*) – Training data loader
- **valid_data** (*torch.utils.data.DataLoader*) – Validation data loader
- **surrogate** (*nn.Module*) – Surrogate Instance
- **max_labels** (*tuple*) – Search space of input architecture
- **batch_size** (*int, optional*) – Batch size Defaults to 32.
- **epochs** (*int, optional*) – Epoch count. Defaults to 3.
- **search_iter** (*int, optional*) – Number of iterations to search with MFAS. Defaults to 3.
- **num_samples** (*int, optional*) – Sample number. Defaults to 15.
- **epoch_surrogate** (*int, optional*) – Surrogate epoch. Defaults to 50.
- **eta_max** (*float, optional*) – See MFAS github for more details. Defaults to 0.001.
- **eta_min** (*float, optional*) – See MFAS github for more details. Defaults to 0.000001.
- **Ti** (*int, optional*) – See MFAS github for more details. Defaults to 1.
- **Tm** (*int, optional*) – See MFAS github for more details. Defaults to 2.
- **temperature_init** (*float, optional*) – See MFAS github for more details. Defaults to 10.0.
- **temperature_final** (*float, optional*) – See MFAS github for more details. Defaults to 0.2.

- **temperature_decay** (*float, optional*) – See MFAS github for more details. Defaults to 4.0.
- **max_progression_levels** (*int, optional*) – See MFAS github for more details. Defaults to 4.
- **lr_surrogate** (*float, optional*) – Surrogate learning rate. Defaults to 0.001.
- **use_weightsharing** (*bool, optional*) – Use weight-sharing when training architectures for evaluation. Defaults to False.

Returns
description

Return type
type

9.1.5 training_structures.gradient_blend module

Implements training structures for gradient blending.

training_structures.gradient_blend.**calcAUPRC**(*pts*)

Calculate AUPRC score given true labels and predicted probabilities.

Parameters
pts (*list*) – List of (true, predicted prob) for each sample in batch.

Returns
AUPRC score

Return type
float

class training_structures.gradient_blend.**completeModule**(*encoders*, *fuse*, *head*)

Bases: Module

Implements and combines sub-modules into a full classifier.

__init__(*encoders*, *fuse*, *head*)

Instantiate completeModule instance.

Parameters

- **encoders** (*list*) – List of nn.Module encoders
- **fuse** (*nn.Module*) – Fusion module
- **head** (*nn.Module*) – Classifier module

forward(*x*)

Apply classifier to output.

Parameters
x (*list[torch.Tensor]*) – List of input tensors

Returns
Classifier output

Return type
torch.Tensor

training: **bool**

training_structures.gradient_blend.**gb_estimate**(*unimodal_models*, *multimodal_classification_head*, *fuse*, *unimodal_classification_heads*, *train_dataloader*, *gb_epoch*, *batch_size*, *v_dataloader*, *lr*, *weight_decay=0.0*, *optimtype=<class 'torch.optim.sgd.SGD'>*)

Compute estimate of gradient-blending score.

Parameters

- **unimodal_models** (*list*) – List of encoder modules
- **multimodal_classification_head** (*nn.Module*) – Classifier given fusion instance
- **fuse** (*nn.Module*) – Fusion module
- **unimodal_classification_heads** (*list*) – List of unimodal classifiers
- **train_dataloader** (*torch.utils.data.Dataloader*) – Training data loader
- **gb_epoch** (*int*) – Number of epochs for gradient-blending
- **batch_size** (*int*) – Batch size
- **v_dataloader** (*torch.utils.data.Dataloader*) – Validation dataloader
- **lr** (*float*) – Learning Rate
- **weight_decay** (*float, optional*) – Weight decay parameter. Defaults to 0.0.
- **optimtype** (*torch.optim.Optimizer, optional*) – Optimizer instance. Defaults to torch.optim.SGD.

Returns
Normalized weights between unimodal and multimodal models

Return type
float

training_structures.gradient_blend.**getloss**(*model*, *head*, *data*, *monum*, *batch_size*)

Get loss for model given classification head.

Parameters

- **model** (*nn.Module*) – Module to evaluate
- **head** (*nn.Module*) – Classification head.
- **data** (*torch.utils.data.Dataloader*) – Dataloader to evaluate on.
- **monum** (*int*) – Unimodal model index.
- **batch_size** (*int*) – (unused) Batch Size

Returns
Average loss per sample.

Return type
float

training_structures.gradient_blend.**getmloss**(*models*, *head*, *fuse*, *data*, *batch_size*)

Calculate multimodal loss.

Parameters

- **models** (*list*) – List of encoder models
- **head** (*nn.Module*) – Classifier module

- **fuse** (*nn.Module*) – Fusion module
- **data** (*torch.utils.data.Dataloader*) – Data loader to calculate loss on.
- **batch_size** (*int*) – Batch size of dataloader

Returns
Average loss

Return type
float

training_structures.gradient_blend.**multimodalcompute**(*models*, *train_x*)

Compute encoded representation for each modality in train_x using encoders in models.

Parameters

- **models** (*list*) – List of encoder instances
- **train_x** (*List*) – List of Input Tensors

Returns
List of encoded tensors

Return type
List

training_structures.gradient_blend.**multimodalcondense**(*models*, *fuse*, *train_x*)

Compute fusion encoded output.

Parameters

- **models** (*List*) – List of nn.Modules for each encoder
- **fuse** (*nn.Module*) – Fusion instance
- **train_x** (*List*) – List of Input Tensors

Returns
Fused output

Return type
torch.Tensor

training_structures.gradient_blend.**single_test**(*model*, *test_dataloader*, *auprc=False*, *classification=True*)

Run single test with model and test data loader.

Parameters

- **model** (*nn.Module*) – Model to evaluate.
- **test_dataloader** (*torch.utils.data.DataLoader*) – Test data loader
- **auprc** (*bool, optional*) – Whether to return AUPRC scores or not. Defaults to False.
- **classification** (*bool, optional*) – Whether to return classification accuracy or not. Defaults to True.

Returns
Dictionary of (metric, value) pairs

Return type
dict

training_structures.gradient_blend.**test**(*model*, *test_dataloaders_all*, *dataset*, *method_name='My method'*, *auprc=False*, *classification=True*, *no_robust=False*)

Test module, reporting results to stdout.

Parameters

- **model** (*nn.Module*) – Model to test
- **test_dataloaders_all** (*list[torch.utils.data.Dataloader]*) – List of data loaders to test on.
- **dataset** (*string*) – Dataset name
- **method_name** (*str, optional*) – Method name. Defaults to 'My method'.
- **auprc** (*bool, optional*) – Whether to use AUPRC scores or not. Defaults to False.
- **classification** (*bool, optional*) – Whether the task is classificaion or not. Defaults to True.
- **no_robust** (*bool, optional*) – Whether to not apply robustness variations to input. Defaults to False.

training_structures.gradient_blend.**train**(*unimodal_models*, *multimodal_classification_head*, *unimodal_classification_heads*, *fuse*, *train_dataloader*, *valid_dataloader*, *num_epoch*, *lr*, *gb_epoch=20*, *v_rate=0.08*, *weight_decay=0.0*, *optimtype=<class 'torch.optim.sgd.SGD'>*, *finetune_epoch=25*, *classification=True*, *AUPRC=False*, *savedir='best.pt'*, *track_complexity=True*)

Train model using gradient_blending.

Parameters

- **unimodal_models** (*list*) – List of modules, unimodal encoders for each input modality in the order of the modality input data.
- **multimodal_classification_head** (*nn.Module*) – Classification head that takes in fused output of unimodal models of all modalities
- **unimodal_classification_heads** (*list[nn.Module]*) – List of classification heads that each takes in output of one unimodal model (must be in the same modality order as unimodal_models)
- **fuse** (*nn.Module*) – Fusion module that takes in a list of outputs from unimodal_models and generate a fused representation
- **train_dataloader** (*torch.utils.data.DataLoader*) – Training data loader
- **valid_dataloader** (*torch.utils.data.DataLoader*) – Validation data loader
- **num_epoch** (*int*) – Number of epochs to train this model on.
- **lr** (*float*) – Learning rate.
- **gb_epoch** (*int, optional*) – Number of epochs between re-evaluation of weights of gradient blend. Defaults to 20.
- **v_rate** (*float, optional*) – Portion of training set used as validation for gradient blend weight estimation. Defaults to 0.08.
- **weight_decay** (*float, optional*) – Weight decay of optimizer. Defaults to 0.0.
- **optimtype** (*torch.optim.Optimizer, optional*) – Type of optimizer to use. Defaults to torch.optim.SGD.

- **finetune_epoch** (*int, optional*) – Number of epochs to finetune the classification head. Defaults to 25.
- **classification** (*bool, optional*) – Whether the task is a classification task. Defaults to True.
- **AUPRC** (*bool, optional*) – Whether to compute auprc score or not. Defaults to False.
- **savedir** (*str, optional*) – The name of the saved file for the model with current best validation performance. Defaults to 'best.pt'.
- **track_complexity** (*bool, optional*) – Whether to track complexity or not. Defaults to True.

training_structures.gradient_blend.**train_multimodal**(*models*, *head*, *fuse*, *optim*, *trains*, *valids*, *epoch*, *batch_size*)

Train multimodal gradient-blending model.

Parameters

- **models** (*list*) – List of nn.modules for the encoders
- **head** (*nn.Module*) – Classifier, post fusion layer
- **fuse** (*nn.Module*) – Fusion module
- **optim** (*torch.optim.Optimizer*) – Optimizer instance.
- **trains** (*torch.utils.data.Dataloader*) – Training data dataloader
- **valids** (*torch.utils.data.Dataloader*) – Validation data dataloader
- **epoch** (*int*) – Number of epochs to train on
- **batch_size** (*int*) – Batch size

Returns
metric

Return type
float

training_structures.gradient_blend.**train_unimodal**(*model*, *head*, *optim*, *trains*, *valids*, *monum*, *epoch*, *batch_size*)

Train unimodal gradient blending module.

Parameters

- **model** (*nn.Module*) – Unimodal encoder
- **head** (*nn.Module*) – Classifier instance
- **optim** (*torch.optim.Optimizer*) – Optimizer instance
- **trains** (*torch.utils.data.DataLoader*) – Training Dataloader Instance
- **valids** (*torch.utils.data.DataLoader*) – Validation DataLoader Instance
- **monum** (*int*) – Modality index
- **epoch** (*int*) – Number of epochs to train on
- **batch_size** (*int*) – Batch size of data loaders

Returns
Metric

Return type
float

9.1.6 training_structures.unimodal module

Implements training pipeline for unimodal comparison.

training_structures.unimodal.**single_test**(*encoder*, *head*, *test_dataloader*, *auprc=False*, *modalnum=0*, *task='classification'*, *criterion=None*)

Test unimodal model on one dataloader.

Parameters

- **encoder** (*nn.Module*) – Unimodal encoder module
- **head** (*nn.Module*) – Module which takes in encoded unimodal input and predicts output.
- **test_dataloader** (*torch.utils.data.DataLoader*) – Data Loader for test set.
- **auprc** (*bool, optional*) – Whether to output AUPRC or not. Defaults to False.
- **modalnum** (*int, optional*) – Index of modality to consider for the test with the given encoder. Defaults to 0.
- **task** (*str, optional*) – Type of task to try. Supports "classification", "regression", or "multilabel". Defaults to 'classification'.
- **criterion** (*nn.Module, optional*) – Loss module. Defaults to None.

Returns
Dictionary of (metric, value) relations.

Return type
dict

training_structures.unimodal.**test**(*encoder*, *head*, *test_dataloaders_all*, *dataset='default'*, *method_name='My method'*, *auprc=False*, *modalnum=0*, *task='classification'*, *criterion=None*, *no_robust=False*)

Test unimodal model on all provided dataloaders.

Parameters

- **encoder** (*nn.Module*) – Encoder module
- **head** (*nn.Module*) – Module which takes in encoded unimodal input and predicts output.
- **test_dataloaders_all** (*dict*) – Dictionary of noisetype, dataloader to test.
- **dataset** (*str, optional*) – Dataset to test on. Defaults to 'default'.
- **method_name** (*str, optional*) – Method name. Defaults to 'My method'.
- **auprc** (*bool, optional*) – Whether to output AUPRC scores or not. Defaults to False.
- **modalnum** (*int, optional*) – Index of modality to test on. Defaults to 0.
- **task** (*str, optional*) – Type of task to try. Supports "classification", "regression", or "multilabel". Defaults to 'classification'.
- **criterion** (*nn.Module, optional*) – Loss module. Defaults to None.
- **no_robust** (*bool, optional*) – Whether to not apply robustness methods or not. Defaults to False.

training_structures.unimodal.**train**(*encoder*, *head*, *train_dataloader*, *valid_dataloader*, *total_epochs*, *early_stop=False*, *optimtype=<class 'torch.optim.rmsprop.RMSprop'>*, *lr=0.001*, *weight_decay=0.0*, *criterion=CrossEntropyLoss()*, *auprc=False*, *save_encoder='encoder.pt'*, *save_head='head.pt'*, *modalnum=0*, *task='classification'*, *track_complexity=True*)

Train unimodal module.

Parameters

- **encoder** (*nn.Module*) – Unimodal encodder for the modality
- **head** (*nn.Module*) – Takes in the unimodal encoder output and produces the final prediction.
- **train_dataloader** (*torch.utils.data.DataLoader*) – Training data dataloader
- **valid_dataloader** (*torch.utils.data.DataLoader*) – Validation set dataloader
- **total_epochs** (*int*) – Total number of epochs
- **early_stop** (*bool, optional*) – Whether to apply early-stopping or not. Defaults to False.
- **optimtype** (*torch.optim.Optimizer, optional*) – Type of optimizer to use. Defaults to torch.optim.RMSprop.
- **lr** (*float, optional*) – Learning rate. Defaults to 0.001.
- **weight_decay** (*float, optional*) – Weight decay of optimizer. Defaults to 0.0.
- **criterion** (*nn.Module, optional*) – Loss module. Defaults to nn.CrossEntropyLoss().
- **auprc** (*bool, optional*) – Whether to compute AUPRC score or not. Defaults to False.
- **save_encoder** (*str, optional*) – Path of file to save model with best validation performance. Defaults to 'encoder.pt'.
- **save_head** (*str, optional*) – Path fo file to save head with best validation performance. Defaults to 'head.pt'.
- **modalnum** (*int, optional*) – Which modality to apply encoder to. Defaults to 0.
- **task** (*str, optional*) – Type of task to try. Supports "classification", "regression", or "multilabel". Defaults to 'classification'.
- **track_complexity** (*bool, optional*) – Whether to track the model's complexity or not. Defaults to True.

9.1.7 Module contents

CHAPTER

TEN

UNIMODAL ENCODERS

10.1 unimodals package

10.1.1 Subpackages

unimodals.gentle_push package

Submodules

unimodals.gentle_push.head module

Implements gentle_push head module.

Taken from https://github.com/brentyi/multimodalfilter/blob/master/crossmodal/push_models/lstm.py

class unimodals.gentle_push.head.**GentlePushLateLSTM**(*input_size*, *hidden_size*)

Bases: Module

Implements Gentle Push's Late LSTM model.

__init__(*input_size*, *hidden_size*)

Instantiate GentlePushLateLSTM Module.

Parameters

- **input_size** (*int*) – Input dimension
- **hidden_size** (*int*) – Hidden dimension

forward(*x*)

Apply GentlePushLateLSTM to Model Input.

Parameters
x (*torch.Tensor*) – Model Input

Returns
Model Output

Return type
torch.Tensor

training: **bool**

class unimodals.gentle_push.head.**Head**(*units: int = 64*)

Bases: Module

Implements Gentle Push's Head module.

__init__(*units: int = 64*)

Instantiates Head module.

Parameters
units (*int, optional*) – Number of layers in LSTM. Defaults to 64.

forward(*fused_features*)

Apply Head module to Layer Inputs.

Parameters
fused_features (*torch.Tensor*) – Layer Inputs

Returns
Layer Outputs

Return type
torch.Tensor

training: **bool**

unimodals.gentle_push.layers module

Implements sub-layers for gentle_push model.

Taken from https://github.com/brentyi/multimodalfilter/blob/master/crossmodal/push_models/layers.py.

unimodals.gentle_push.layers.**control_layers**(*units: int*) → Module

Create a control command encoder block.

Parameters
units (*int*) – # of hidden units in network layers.

Returns
Encoder block.

Return type
nn.Module

unimodals.gentle_push.layers.**observation_image_layers**(*units: int*, *spanning_avg_pool: bool = False*) → Module

Create an image encoder block.

Parameters
units (*int*) – # of hidden units in network layers.

Returns
Encoder block.

Return type
nn.Module

unimodals.gentle_push.layers.**observation_pos_layers**(*units: int*) → Module

Create an end effector position encoder block.

Parameters
units (*int*) – # of hidden units in network layers.

Returns
Encoder block.

Return type
nn.Module

unimodals.gentle_push.layers.**observation_sensors_layers**(*units: int*) → Module

Create an F/T sensor encoder block.

Parameters
units (*int*) – # of hidden units in network layers.

Returns
Encoder block.

Return type
nn.Module

Module contents

unimodals.robotics package

Submodules

unimodals.robotics.decoders module

Implements decoders for robotics tasks.

class unimodals.robotics.decoders.**ContactDecoder**(*z_dim*, *deterministic*, *head=1*)

Bases: Module

Decodes everything, given some input.

__init__(*z_dim*, *deterministic*, *head=1*)

Initialize ContactDecoder Module.

Parameters

- **z_dim** (*float*) – Z dimension size
- **deterministic** (*float*) – Whether input parameters are deterministic or sampled from some distribution given prior mu and var.
- **head** (*int*) – Output dimension of head.

forward(*input*)

Apply ContactDecoder Module to Layer Input.

Parameters
input (*torch.Tensor*) – Layer Input

Returns
Decoded Output

Return type
torch.Tensor

training: bool

class unimodals.robotics.decoders.**EeDeltaDecoder**(*z_dim*, *action_dim*, *initailize_weights=True*)

Bases: Module

Implements an EE Delta Decoder.

__init__(*z_dim*, *action_dim*, *initailize_weights=True*)

Initialize EeDeltaDecoder Module.

Parameters

- **z_dim** (*float*) – Z dimension size
- **alpha** (*float*) – Alpha to multiply proprio input by.
- **initialize_weights** (*bool, optional*) – Whether to initialize weights or not. Defaults to True.

forward(*mm_act_feat*)

Apply EeDeltaDecoder Module to EE Delta.

Parameters
mm_act_feat (*torch.Tensor*) – EE Delta

Returns
Decoded Output

Return type
torch.Tensor

training: **bool**

class unimodals.robotics.decoders.**OpticalFlowDecoder**(*z_dim*, *initailize_weights=True*)

Bases: Module

Implements optical flow and optical flow mask decoder.

__init__(*z_dim*, *initailize_weights=True*)

Initialize OpticalFlowDecoder Module.

Parameters

- **z_dim** (*float*) – Z dimension size
- **alpha** (*float*) – Alpha to multiply proprio input by.
- **initialize_weights** (*bool, optional*) – Whether to initialize weights or not. Defaults to True.

forward(*tiled_feat*, *img_out_convs*)

Predicts the optical flow and optical flow mask.

Parameters

- **tiled_feat** – action conditioned z (output of fusion + action network)
- **img_out_convs** – outputs of the image encoders (skip connections)

training: **bool**

unimodals.robotics.encoders module

Implements encoders for robotics tasks.

class unimodals.robotics.encoders.**ActionEncoder**(*action_dim*)

Bases: Module

Implements an action-encoder module.

__init__(*action_dim*)

Instantiate ActionEncoder module.

Parameters
action_dim (*int*) – Dimension of action.

forward(*action*)

Apply action encoder to action input.

Parameters
action (*torch.Tensor optional*) – Action input

Returns
Encoded output

Return type
torch.Tensor

training: **bool**

class unimodals.robotics.encoders.**DepthEncoder**(*z_dim*, *alpha*, *initialize_weights=True*)

Bases: Module

Implements a simplified depth-encoder module.

Sourced from Making Sense of Vision and Touch.

__init__(*z_dim*, *alpha*, *initialize_weights=True*)

Initialize DepthEncoder Module.

Parameters

- **z_dim** (*float*) – Z dimension size
- **alpha** (*float*) – Alpha to multiply input by.
- **initialize_weights** (*bool, optional*) – Whether to initialize weights or not. Defaults to True.

forward(*depth_in*)

Apply encoder to depth input.

Parameters
depth_in (*torch.Tensor*) – Depth input

Returns
Output of encoder, Output of encoder after each convolution.

Return type
tuple(torch.Tensor, torch.Tensor)

training: **bool**

class unimodals.robotics.encoders.**ForceEncoder**(*z_dim*, *alpha*, *initialize_weights=True*)

Bases: Module

Implements force encoder module.

Sourced from selfsupervised code.

__init__(*z_dim*, *alpha*, *initialize_weights=True*)

Initialize ForceEncoder Module.

Parameters

- **z_dim** (*float*) – Z dimension size
- **alpha** (*float*) – Alpha to multiply proprio input by.
- **initialize_weights** (*bool, optional*) – Whether to initialize weights or not. Defaults to True.

forward(*force*)

Apply ForceEncoder to Force Input.

Parameters
force (*torch.Tensor*) – Force Input

Returns
Encoded Output

Return type
torch.Tensor

training: **bool**

class unimodals.robotics.encoders.**ImageEncoder**(*z_dim*, *alpha*, *initialize_weights=True*)

Bases: Module

Implements image encoder module.

Sourced from Making Sense of Vision and Touch.

__init__(*z_dim*, *alpha*, *initialize_weights=True*)

Initialize ImageEncoder Module.

Parameters

- **z_dim** (*float*) – Z dimension size
- **alpha** (*float*) – Alpha to multiply input by.
- **initialize_weights** (*bool, optional*) – Whether to initialize weights or not. Defaults to True.

forward(*vis_in*)

Apply encoder to image input.

Parameters
vis_in (*torch.Tensor*) – Image input

Returns
Output of encoder, Output of encoder after each convolution.

Return type
tuple(torch.Tensor, torch.Tensor)

training: bool

class unimodals.robotics.encoders.**ProprioEncoder**(*z_dim*, *alpha*, *initialize_weights=True*)

Bases: Module

Implements image encoder module.

Sourced from selfsupervised code.

__init__(*z_dim*, *alpha*, *initialize_weights=True*)

Initialize ProprioEncoder Module.

Parameters

- **z_dim** (*float*) – Z dimension size
- **alpha** (*float*) – Alpha to multiply proprio input by.
- **initialize_weights** (*bool, optional*) – Whether to initialize weights or not. Defaults to True.

forward(*proprio*)

Apply ProprioEncoder to Proprio Input.

Parameters
proprio (*torch.Tensor*) – Proprio Input

Returns
Encoded Output

Return type
torch.Tensor

training: bool

unimodals.robotics.layers module

Neural network layers.

class unimodals.robotics.layers.**CausalConv1D**(*in_channels*, *out_channels*, *kernel_size*, *stride=1*, *dilation=1*, *bias=True*)

Bases: Conv1d

Implements a causal 1D convolution.

__init__(*in_channels*, *out_channels*, *kernel_size*, *stride=1*, *dilation=1*, *bias=True*)

Initialize CausalConv1D Object.

Parameters

- **in_channels** (*int*) – Input Dimension
- **out_channels** (*int*) – Output Dimension
- **kernel_size** (*int*) – Kernel Size
- **stride** (*int, optional*) – Stride Amount. Defaults to 1.
- **dilation** (*int, optional*) – Dilation Amount. Defaults to 1.
- **bias** (*bool, optional*) – Whether to add a bias after convolving. Defaults to True.

bias: Optional[Tensor]

dilation: Tuple[int, ...]

forward(*x*)

Apply CasualConv1D layer to layer input.

Parameters
x (*torch.Tensor*) – Layer Input

Returns
Layer Output

Return type
torch.Tensor

groups: int

in_channels: int

kernel_size: Tuple[int, ...]

out_channels: int

output_padding: Tuple[int, ...]

padding: Union[str, Tuple[int, ...]]

padding_mode: str

stride: Tuple[int, ...]

transposed: bool

weight: Tensor

class unimodals.robotics.layers.**Flatten**

Bases: Module

Implements .reshape(*,-1) for nn.Sequential usage.

__init__()

Initialize Flatten Object.

forward(*x*)

Apply Flatten to Input.

Parameters
x (*torch.Tensor*) – Layer Input

Returns
Layer Output

Return type
torch.Tensor

training: bool

class unimodals.robotics.layers.**ResidualBlock**(*channels*)

Bases: Module

Implements a simple residual block.

__init__(*channels*)

Initialize ResidualBlock object.

Parameters
channels (*int*) – Number of input and hidden channels in block.

forward(*x*)

Apply ResidualBlock to Layer Input.

Parameters
x (*torch.Tensor*) – Layer Input

Returns
Layer Output

Return type
torch.Tensor

training: **bool**

class unimodals.robotics.layers.**View**(*size*)

Bases: Module

Extends .view() for nn.Sequential usage.

__init__(*size*)

Initialize View Object.

Parameters
size (*int*) – Output Size

forward(*tensor*)

Apply View to Layer Input.

Parameters
tensor (*torch.Tensor*) – Layer Input

Returns
Layer Output

Return type
torch.Tensor

training: **bool**

unimodals.robotics.layers.**conv2d**(*in_channels*, *out_channels*, *kernel_size=3*, *stride=1*, *dilation=1*, *bias=True*)

Create nn.Module with *same* convolution with LeakyReLU, i.e. output shape equals input shape.

Parameters

- **in_planes** (*int*) – The number of input feature maps.
- **out_planes** (*int*) – The number of output feature maps.
- **kernel_size** (*int*) – The filter size.
- **dilation** (*int*) – The filter dilation factor.
- **stride** (*int*) – The filter stride.

unimodals.robotics.layers.**crop_like**(*input*, *target*)

Crop Tensor based on Another's Size.

Parameters

- **input** (*torch.Tensor*) – Input Tensor
- **target** (*torch.Tensor*) – Tensor to Compare and Crop To.

Returns
Cropped Tensor

Return type
torch.Tensor

unimodals.robotics.layers.**deconv**(*in_planes*, *out_planes*)

Create Deconvolution Layer.

Parameters

- **in_planes** (*int*) – Number of Input Channels.
- **out_planes** (*int*) – Number of Output Channels.

Returns
Deconvolution Object with Given Input/Output Channels.

Return type
nn.Module

unimodals.robotics.layers.**predict_flow**(*in_planes*)

Create Optical Flow Prediction Layer.

Parameters
in_planes (*int*) – Number of Input Channels.

Returns
Convolution Object with Given Input/Output Channels.

Return type
nn.Module

unimodals.robotics.models_utils module

Utility functions for robotics unimodals.

unimodals.robotics.models_utils.**filter_depth**(*depth_image*)

Get filter depth given a depth image.

Parameters
depth_image (*torch.Tensor*) – Depth image.

Returns
Output

Return type
torch.Tensor

unimodals.robotics.models_utils.**init_weights**(*modules*)

Weight initialization from original SensorFusion Code.

unimodals.robotics.models_utils.**rescaleImage**(*image*, *output_size=128*, *scale=0.00392156862745098*)

Rescale the image in a sample to a given size.

Parameters

output_size (*tuple or int*) – Desired output size. If tuple, output is matched to output_size. If int, smaller of image edges is matched to output_size keeping aspect ratio the same.

Module contents

10.1.2 Submodules

10.1.3 unimodals.MVAE module

Implements various encoders and decoders for MVAE.

class unimodals.MVAE.**DeLeNet**(*in_channels*, *arg_channels*, *additional_layers*, *latent*)

Bases: Module

Implements an image deconvolution decoder for MVAE.

__init__(*in_channels*, *arg_channels*, *additional_layers*, *latent*)

Instantiate DeLeNet Module.

Parameters

- **in_channels** (*int*) – Number of input channels
- **arg_channels** (*int*) – Number of arg channels
- **additional_layers** (*int*) – Number of additional layers.
- **latent** (*int*) – Latent dimension size

forward(*x*)

Apply DeLeNet to Layer Input.

Parameters
x (*torch.Tensor*) – Layer Input

Returns
Layer Output

Return type
torch.Tensor

training: bool

class unimodals.MVAE.**LeNetEncoder**(*in_channels*, *arg_channels*, *additional_layers*, *latent*, *twooutput=True*)

Bases: Module

Implements a LeNet Encoder for MVAE.

__init__(*in_channels*, *arg_channels*, *additional_layers*, *latent*, *twooutput=True*)

Instantiate LeNetEncoder Module

Parameters

- **in_channels** (*int*) – Input Dimensions
- **arg_channels** (*int*) – Arg channels dimension size

- **additional_layers** (*int*) – Number of additional layers
- **latent** (*int*) – Latent dimension size
- **twooutput** (*bool, optional*) – Whether to output twice the size of the latent. Defaults to True.

forward(*x*)

Apply LeNetEncoder to Layer Input.

Parameters
x (*torch.Tensor*) – Layer Input

Returns
Layer Output

Return type
torch.Tensor

training: **bool**

class unimodals.MVAE.**MLPEncoder**(*indim*, *hiddim*, *outdim*)

Bases: Module

Implements MLP Encoder for MVAE.

__init__(*indim*, *hiddim*, *outdim*)

Initialzies MLPEncoder Object.

Parameters

- **indim** (*int*) – Input Dimension
- **hiddim** (*int*) – Hidden Dimension
- **outdim** (*int*) – Output Dimension

forward(*x*)

Apply MLPEncoder to Input.

Parameters
x (*torch.Tensor*) – Layer Input

Returns
Layer Output

Return type
torch.Tensor

training: **bool**

class unimodals.MVAE.**TSDecoder**(*indim*, *outdim*, *finaldim*, *timestep*)

Bases: Module

Implements a time-series decoder for MVAE.

__init__(*indim*, *outdim*, *finaldim*, *timestep*)

Instantiate TSDecoder Module.

Parameters

- **indim** (*int*) – Input dimension
- **outdim** (*int*) – (unused) Output dimension

- **finaldim** (*int*) – Hidden dimension
- **timestep** (*int*) – Number of timesteps

forward(*x*)

Apply TSDecoder to layer input.

Parameters
x (*torch.Tensor*) – Layer Input

Returns
Layer Output

Return type
torch.Tensor

training: **bool**

class unimodals.MVAE.**TSEncoder**(*indim*, *outdim*, *finaldim*, *timestep*, *returnvar=True*, *batch_first=False*)

Bases: Module

Implements a time series encoder for MVAE.

__init__(*indim*, *outdim*, *finaldim*, *timestep*, *returnvar=True*, *batch_first=False*)

Instantiate TSEncoder Module.

Parameters

- **indim** (*int*) – Input Dimension of GRU
- **outdim** (*int*) – Output dimension of GRU
- **finaldim** (*int*) – Output dimension of TSEncoder
- **timestep** (*float*) – Number of timestamps
- **returnvar** (*bool, optional*) – Whether to return the output split with the first encoded portion and the next or not. Defaults to True.
- **batch_first** (*bool, optional*) – Whether the batching dimension is the first dimension of the input or not. Defaults to False.

forward(*x*)

Apply TS Encoder to Layer Input.

Parameters
x (*torch.Tensor*) – Layer Input

Returns
Layer Output

Return type
torch.Tensor

training: **bool**

10.1.4 unimodals.common_models module

Implements common unimodal encoders.

class unimodals.common_models.**Constant**(*out_dim*)

Bases: Module

Implements a module that returns a constant no matter the input.

__init__(*out_dim*)

Initialize Constant Module.

Parameters
out_dim (*int*) – Output Dimension.

forward(*x*)

Apply Constant to Layer Input.

Parameters
x (*torch.Tensor*) – Layer Input

Returns
Layer Output

Return type
torch.Tensor

training: **bool**

class unimodals.common_models.**DAN**(*indim*, *hiddim*, *dropout=False*, *dropoutp=0.25*, *nlayers=3*, *has_padding=False*)

Bases: Module

Deep Averaging Network: https://people.cs.umass.edu/~miyyer/pubs/2015_acl_dan.pdf Deep Sets: https://arxiv.org/abs/1703.06114

__init__(*indim*, *hiddim*, *dropout=False*, *dropoutp=0.25*, *nlayers=3*, *has_padding=False*)

Initialize DAN Object.

Parameters

- **indim** (*int*) – Input Dimension
- **hiddim** (*int*) – Hidden Dimension
- **dropout** (*bool, optional*) – Whether to apply dropout to layer output. Defaults to False.
- **dropoutp** (*float, optional*) – Dropout probability. Defaults to 0.25.
- **nlayers** (*int, optional*) – Number of layers. Defaults to 3.
- **has_padding** (*bool, optional*) – Whether the input has padding. Defaults to False.

forward(*x*)

Apply DAN to input.

Parameters
x (*torch.Tensor*) – Layer Input

Returns
Layer Output

Return type
torch.Tensor

training: bool

class unimodals.common_models.**GRU**(*indim*, *hiddim*, *dropout=False*, *dropoutp=0.1*, *flatten=False*, *has_padding=False*, *last_only=False*, *batch_first=True*)

Bases: Module

Implements Gated Recurrent Unit (GRU).

__init__(*indim*, *hiddim*, *dropout=False*, *dropoutp=0.1*, *flatten=False*, *has_padding=False*, *last_only=False*, *batch_first=True*)

Initialize GRU Module.

Parameters

- **indim** (*int*) – Input dimension
- **hiddim** (*int*) – Hidden dimension
- **dropout** (*bool, optional*) – Whether to apply dropout layer or not. Defaults to False.
- **dropoutp** (*float, optional*) – Dropout probability. Defaults to 0.1.
- **flatten** (*bool, optional*) – Whether to flatten output before returning. Defaults to False.
- **has_padding** (*bool, optional*) – Whether the input has padding or not. Defaults to False.
- **last_only** (*bool, optional*) – Whether to return only the last output of the GRU. Defaults to False.
- **batch_first** (*bool, optional*) – Whether to batch before applying or not. Defaults to True.

forward(*x*)

Apply GRU to input.

Parameters
x (*torch.Tensor*) – Layer Input

Returns
Layer Output

Return type
torch.Tensor

training: bool

class unimodals.common_models.**GRUWithLinear**(*indim*, *hiddim*, *outdim*, *dropout=False*, *dropoutp=0.1*, *flatten=False*, *has_padding=False*, *output_each_layer=False*, *batch_first=False*)

Bases: Module

Implements a GRU with Linear Post-Processing.

__init__(*indim*, *hiddim*, *outdim*, *dropout=False*, *dropoutp=0.1*, *flatten=False*, *has_padding=False*, *output_each_layer=False*, *batch_first=False*)

Initialize GRUWithLinear Module.

Parameters

- **indim** (*int*) – Input Dimension
- **hiddim** (*int*) – Hidden Dimension
- **outdim** (*int*) – Output Dimension
- **dropout** (*bool, optional*) – Whether to apply dropout or not. Defaults to False.
- **dropoutp** (*float, optional*) – Dropout probability. Defaults to 0.1.
- **flatten** (*bool, optional*) – Whether to flatten output before returning. Defaults to False.
- **has_padding** (*bool, optional*) – Whether input has padding. Defaults to False.
- **output_each_layer** (*bool, optional*) – Whether to return the output of every intermediate layer. Defaults to False.
- **batch_first** (*bool, optional*) – Whether to apply batching before GRU. Defaults to False.

forward(*x*)

Apply GRUWithLinear to Input.

Parameters
x (*torch.Tensor*) – Layer Input

Returns
Layer Output

Return type
torch.Tensor

training: bool

class unimodals.common_models.**GlobalPooling2D**

Bases: Module

Implements 2D Global Pooling.

__init__()

Initializes GlobalPooling2D Module.

forward(*x*)

Apply 2D Global Pooling to Layer Input.

Parameters
x (*torch.Tensor*) – Layer Input

Returns
Layer Output

Return type
torch.Tensor

training: bool

class unimodals.common_models.**Identity**

Bases: Module

Identity Module.

__init__()

Initialize Identity Module.

forward(*x*)

Apply Identity to Input.

Parameters
x (*torch.Tensor*) – Layer Input

Returns
Layer Output

Return type
torch.Tensor

training: **bool**

class unimodals.common_models.**LSTM**(*indim*, *hiddim*, *linear_layer_outdim=None*, *dropout=False*, *dropoutp=0.1*, *flatten=False*, *has_padding=False*)

Bases: Module

Extends nn.LSTM with dropout and other features.

__init__(*indim*, *hiddim*, *linear_layer_outdim=None*, *dropout=False*, *dropoutp=0.1*, *flatten=False*, *has_padding=False*)

Initialize LSTM Object.

Parameters

- **indim** (*int*) – Input Dimension
- **hiddim** (*int*) – Hidden Layer Dimension
- **linear_layer_outdim** (*int, optional*) – Linear Layer Output Dimension. Defaults to None.
- **dropout** (*bool, optional*) – Whether to apply dropout to layer output. Defaults to False.
- **dropoutp** (*float, optional*) – Dropout probability. Defaults to 0.1.
- **flatten** (*bool, optional*) – Whether to flatten out. Defaults to False.
- **has_padding** (*bool, optional*) – Whether input has padding. Defaults to False.

forward(*x*)

Apply LSTM to layer input.

Parameters
x (*torch.Tensor*) – Layer Input

Returns
Layer Output

Return type
torch.Tensor

training: **bool**

class unimodals.common_models.**LeNet**(*in_channels*, *args_channels*, *additional_layers*, *output_each_layer=False*, *linear=None*, *squeeze_output=True*)

Bases: Module

Implements LeNet.

Adapted from centralnet code https://github.com/slyviacassell/_MFAS/blob/master/models/central/avmnist.py.

__init__(*in_channels*, *args_channels*, *additional_layers*, *output_each_layer=False*, *linear=None*, *squeeze_output=True*)

Initialize LeNet.

Parameters

- **in_channels** (*int*) – Input channel number.
- **args_channels** (*int*) – Output channel number for block.
- **additional_layers** (*int*) – Number of additional blocks for LeNet.
- **output_each_layer** (*bool, optional*) – Whether to return the output of all layers. Defaults to False.
- **linear** (*tuple, optional*) – Tuple of (input_dim, output_dim) for optional linear layer post-processing. Defaults to None.
- **squeeze_output** (*bool, optional*) – Whether to squeeze output before returning. Defaults to True.

forward(*x*)

Apply LeNet to layer input.

Parameters
x (*torch.Tensor*) – Layer Input

Returns
Layer Output

Return type
torch.Tensor

training: bool

class unimodals.common_models.Linear(*indim*, *outdim*, *xavier_init=False*)

Bases: Module

Linear Layer with Xavier Initialization, and 0 Bias.

__init__(*indim*, *outdim*, *xavier_init=False*)

Initialize Linear Layer w/ Xavier Init.

Parameters

- **indim** (*int*) – Input Dimension
- **outdim** (*int*) – Output Dimension
- **xavier_init** (*bool, optional*) – Whether to apply Xavier Initialization to Layer. Defaults to False.

forward(*x*)

Apply Linear Layer to Input.

Parameters
x (*torch.Tensor*) – Input Tensor

Returns
Output Tensor

Return type
torch.Tensor

training: bool

class unimodals.common_models.**MLP**(*indim*, *hiddim*, *outdim*, *dropout=False*, *dropoutp=0.1*, *output_each_layer=False*)

Bases: Module

Two layered perceptron.

__init__(*indim*, *hiddim*, *outdim*, *dropout=False*, *dropoutp=0.1*, *output_each_layer=False*)

Initialize two-layered perceptron.

Parameters

- **indim** (*int*) – Input dimension
- **hiddim** (*int*) – Hidden layer dimension
- **outdim** (*int*) – Output layer dimension
- **dropout** (*bool, optional*) – Whether to apply dropout or not. Defaults to False.
- **dropoutp** (*float, optional*) – Dropout probability. Defaults to 0.1.
- **output_each_layer** (*bool, optional*) – Whether to return outputs of each layer as a list. Defaults to False.

forward(*x*)

Apply MLP to Input.

Parameters
x (*torch.Tensor*) – Layer Input

Returns
Layer Output

Return type
torch.Tensor

training: bool

class unimodals.common_models.**MaxOut_MLP**(*num_outputs*, *first_hidden=64*, *number_input_feats=300*, *second_hidden=None*, *linear_layer=True*)

Bases: Module

Implements Maxout w/ MLP.

__init__(*num_outputs*, *first_hidden=64*, *number_input_feats=300*, *second_hidden=None*, *linear_layer=True*)

Instantiate MaxOut_MLP Module.

Parameters

- **num_outputs** (*int*) – Output dimension
- **first_hidden** (*int, optional*) – First hidden layer dimension. Defaults to 64.
- **number_input_feats** (*int, optional*) – Input dimension. Defaults to 300.
- **second_hidden** (*_type_, optional*) – Second hidden layer dimension. Defaults to None.
- **linear_layer** (*bool, optional*) – Whether to include an output hidden layer or not. Defaults to True.

forward(*x*)

Apply module to layer input

Parameters
x (*torch.Tensor*) – Layer Input

Returns
Layer Output

Return type
torch.Tensor

training: **bool**

class unimodals.common_models.**Maxout**(*d*, *m*, *k*)

Bases: Module

Implements Maxout module.

__init__(*d*, *m*, *k*)

Initialize Maxout object.

Parameters

- **d** (*int*) – (Unused)
- **m** (*int*) – Number of features remeaining after Maxout.
- **k** (*int*) – Pool Size

forward(*inputs*)

Apply Maxout to inputs.

Parameters
inputs (*torch.Tensor*) – Layer Input

Returns
Layer Output

Return type
torch.Tensor

training: **bool**

class unimodals.common_models.**ResNetLSTMEnc**(*hiddim*, *dropout=False*, *dropoutp=0.1*)

Bases: Module

Implements an encoder which applies as ResNet first, and then an LSTM.

__init__(*hiddim*, *dropout=False*, *dropoutp=0.1*)

Instantiates ResNetLSTMEnc Module

Parameters

- **hiddim** (*int*) – Hidden dimension size of LSTM.
- **dropout** (*bool, optional*) – Whether to apply dropout or not.. Defaults to False.
- **dropoutp** (*float, optional*) – Dropout probability. Defaults to 0.1.

forward(*x*)

Apply ResNetLSTMEnc Module to Input

Parameters
x (*torch.Tensor*) – Layer Input

Returns
Layer Output

Return type
torch.Tensor

training: **bool**

class unimodals.common_models.**Reshape**(*shape*)

Bases: Module

Custom reshape module for easier Sequential usage.

__init__(*shape*)

Initialize Reshape Module.

Parameters
shape (*tuple*) – Tuple to reshape input to

forward(*x*)

Apply Reshape Module to Input.

Parameters
x (*torch.Tensor*) – Layer Input

Returns
Layer Output

Return type
torch.Tensor

training: **bool**

class unimodals.common_models.**Sequential**(**args*, ***kwargs*)

Bases: Sequential

Custom Sequential module for easier usage.

__init__(**args*, ***kwargs*)

Initialize Sequential Layer.

forward(**args*, ***kwargs*)

Apply args to Sequential Layer.

class unimodals.common_models.**Squeeze**(*dim=None*)

Bases: Module

Custom squeeze module for easier Sequential usage.

__init__(*dim=None*)

Initialize Squeeze Module.

Parameters
dim (*int, optional*) – Dimension to Squeeze on. Defaults to None.

forward(*x*)

Apply Squeeze Layer to Input.

Parameters
x (*torch.Tensor*) – Layer Input

Returns
Layer Output

Return type
torch.Tensor

`training: bool`

`class unimodals.common_models.Transformer(n_features, dim)`

Bases: `Module`

Extends nn.Transformer.

`__init__(n_features, dim)`

Initialize Transformer object.

Parameters

- **n_features** (*int*) – Number of features in the input.
- **dim** (*int*) – Dimension which to embed upon / Hidden dimension size.

`forward(x)`

Apply Transformer to Input.

Parameters
x (*torch.Tensor*) – Layer Input

Returns
Layer Output

Return type
torch.Tensor

`training: bool`

`class unimodals.common_models.Transpose(dim0, dim1)`

Bases: `Module`

Custom transpose module for easier Sequential usage.

`__init__(dim0, dim1)`

Initialize Transpose Module.

Parameters

- **dim0** (*int*) – Dimension 1 of Torch.Transpose
- **dim1** (*int*) – Dimension 2 of Torch.Transpose

`forward(x)`

Apply Transpose Module to Input.

Parameters
x (*torch.Tensor*) – Layer Input

Returns
Layer Output

Return type
torch.Tensor

`training: bool`

class unimodals.common_models.**TwoLayersLSTM**(*indim*, *hiddim*, *dropout=False*, *dropoutp=0.1*, *flatten=False*, *has_padding=False*, *LayNorm=True*, *isBidirectional=True*)

Bases: Module

Implements and Extends nn.LSTM for 2-layer LSTMs.

__init__(*indim*, *hiddim*, *dropout=False*, *dropoutp=0.1*, *flatten=False*, *has_padding=False*, *LayNorm=True*, *isBidirectional=True*)

Initialize TwoLayersLSTM Object.

Parameters

- **indim** (*int*) – Input dimension
- **hiddim** (*int*) – Hidden layer dimension
- **dropout** (*bool, optional*) – Whether to apply dropout to layer output. Defaults to False.
- **dropoutp** (*float, optional*) – Dropout probability. Defaults to 0.1.
- **flatten** (*bool, optional*) – Whether to flatten layer output before returning. Defaults to False.
- **has_padding** (*bool, optional*) – Whether input has padding or not. Defaults to False.
- **isBidirectional** (*bool, optional*) – Whether internal LSTMs are bidirectional. Defaults to True.

forward(*x*)

Apply TwoLayersLSTM to input.

Parameters
x (*torch.Tensor*) – Layer Input

Returns
Layer Output

Return type
torch.Tensor

training: bool

class unimodals.common_models.**VGG**(*num_outputs*)

Bases: Module

Extends tmodels.vgg19 module with Global Pooling, BatchNorm, and a Linear Output.

__init__(*num_outputs*)

Initialize VGG Object.

Parameters
num_outputs (*int*) – Output Dimension

forward(*x*)

Apply VGG Module to Input.

Parameters
x (*torch.Tensor*) – Input Tensor

Returns
Output Tensor

Return type
torch.Tensor

`training: bool`

class unimodals.common_models.**VGG11Pruned**(*hiddim*, *dropout=True*, *prune_factor=0.25*, *dropoutp=0.2*)

Bases: Module

Extends VGG11 and prunes layers to make it even smaller.

Slimmer version of vgg11 model with fewer layers in classifier.

__init__(*hiddim*, *dropout=True*, *prune_factor=0.25*, *dropoutp=0.2*)

Initialize VGG11Pruned Object.

Parameters

- **hiddim** (*int*) – Hidden Layer Dimension
- **dropout** (*bool, optional*) – Whether to apply dropout after ReLU. Defaults to True.
- **prune_factor** (*float, optional*) – Percentage of channels to prune. Defaults to 0.25.
- **dropoutp** (*float, optional*) – Dropout probability. Defaults to 0.2.

forward(*x*)

Apply VGG11Pruned to layer input.

Parameters
x (*torch.Tensor*) – Layer Input

Returns
Layer Output

Return type
torch.Tensor

`training: bool`

class unimodals.common_models.**VGG11Slim**(*hiddim*, *dropout=True*, *dropoutp=0.2*, *pretrained=True*, *freeze_features=True*)

Bases: Module

Extends VGG11 with a fewer layers in the classifier.

Slimmer version of vgg11 model with fewer layers in classifier.

__init__(*hiddim*, *dropout=True*, *dropoutp=0.2*, *pretrained=True*, *freeze_features=True*)

Initialize VGG11Slim Object.

Parameters

- **hiddim** (*int*) – Hidden dimension size
- **dropout** (*bool, optional*) – Whether to apply dropout to output of ReLU. Defaults to True.
- **dropoutp** (*float, optional*) – Dropout probability. Defaults to 0.2.
- **pretrained** (*bool, optional*) – Whether to instantiate VGG11 from Pretrained. Defaults to True.
- **freeze_features** (*bool, optional*) – Whether to keep VGG11 features frozen. Defaults to True.

forward(*x*)

Apply VGG11Slim to Layer Input.

Parameters
x (*torch.Tensor*) – Layer Input

Returns
Layer Output

Return type
torch.Tensor

training: **bool**

class unimodals.common_models.**VGG16**(*hiddim*, *pretrained=True*)

Bases: Module

Extends VGG16 for encoding.

__init__(*hiddim*, *pretrained=True*)

Initialize VGG16 Object.

Parameters

- **hiddim** (*int*) – Size of post-processing layer
- **pretrained** (*bool, optional*) – Whether to instantiate VGG16 from pretrained. Defaults to True.

forward(*x*)

Apply VGG16 to Input.

Parameters
x (*torch.Tensor*) – Layer Input

Returns
Layer Output

Return type
torch.Tensor

training: **bool**

class unimodals.common_models.**VGG16Pruned**(*hiddim*, *dropout=True*, *prune_factor=0.25*, *dropoutp=0.2*)

Bases: Module

Extends VGG16 and prunes layers to make it even smaller.

Slimmer version of vgg16 model with fewer layers in classifier.

__init__(*hiddim*, *dropout=True*, *prune_factor=0.25*, *dropoutp=0.2*)

Initialize VGG16Pruned Object.

Parameters

- **hiddim** (*int*) – Hidden Layer Dimension
- **dropout** (*bool, optional*) – Whether to apply dropout after ReLU. Defaults to True.
- **prune_factor** (*float, optional*) – Percentage of channels to prune. Defaults to 0.25.
- **dropoutp** (*float, optional*) – Dropout probability. Defaults to 0.2.

forward(*x*)

Apply VGG16Pruned to layer input.

Parameters
x (*torch.Tensor*) – Layer Input

Returns
Layer Output

Return type
torch.Tensor

training: **bool**

class unimodals.common_models.**VGG16Slim**(*hiddim*, *dropout=True*, *dropoutp=0.2*, *pretrained=True*)

Bases: Module

Extends VGG16 with a fewer layers in the classifier.

Slimmer version of vgg16 model with fewer layers in classifier.

__init__(*hiddim*, *dropout=True*, *dropoutp=0.2*, *pretrained=True*)

Initialize VGG16Slim object.

Parameters

- **hiddim** (*int*) – Hidden dimension size
- **dropout** (*bool, optional*) – Whether to apply dropout to ReLU output. Defaults to True.
- **dropoutp** (*float, optional*) – Dropout probability. Defaults to 0.2.
- **pretrained** (*bool, optional*) – Whether to initialize VGG16 from pretrained. Defaults to True.

forward(*x*)

Apply VGG16Slim to model input.

Parameters
x (*torch.Tensor*) – Layer Input

Returns
Layer Output

Return type
torch.Tensor

training: **bool**

10.1.5 unimodals.res3d module

Implements 3dResnet.

Copied from https://github.com/kenshohara/3D-ResNets-PyTorch/blob/master/models/resnet2p1d.py

class unimodals.res3d.**BasicBlock**(*in_planes*, *planes*, *stride=1*, *downsample=None*)

Bases: Module

Implements basic block of a resnet.

__init__(*in_planes*, *planes*, *stride=1*, *downsample=None*)

Instantiate BasicBlock Module.

Parameters

- **in_planes** (*int*) – Number of input channels
- **planes** (*int*) – Number of output channels
- **stride** (*int, optional*) – Convolution Stride. Defaults to 1.
- **downsample** (*nn.Module, optional*) – Optional Downsampling Layer. Defaults to None.

expansion = 1

forward(*x*)

Apply Block to Layer Input.

Parameters
x (*torch.Tensor*) – Layer Input

Returns
Layer Output

Return type
torch.Tensor

training: bool

class unimodals.res3d.**Bottleneck**(*in_planes*, *planes*, *stride=1*, *downsample=None*)

Bases: Module

Implements bottleneck block of a resnet.

__init__(*in_planes*, *planes*, *stride=1*, *downsample=None*)

Instantiate Bottleneck Module.

Parameters

- **in_planes** (*int*) – Number of input channels
- **planes** (*int*) – Number of output channels
- **stride** (*int, optional*) – Convolution Stride. Defaults to 1.
- **downsample** (*nn.Module, optional*) – Optional Downsampling Layer. Defaults to None.

expansion = 4

forward(*x*)

Apply Bottleneck to Layer Input.

Parameters
x (*torch.Tensor*) – Layer Input

Returns
Layer Output

Return type
torch.Tensor

training: bool

class unimodals.res3d.**ResNet**(*block*, *layers*, *block_inplanes*, *n_input_channels=3*, *conv1_t_size=7*, *conv1_t_stride=1*, *no_max_pool=False*, *shortcut_type='B'*, *widen_factor=1.0*, *n_classes=400*)

Bases: Module

Implements a ResNet from scratch.

__init__(*block*, *layers*, *block_inplanes*, *n_input_channels=3*, *conv1_t_size=7*, *conv1_t_stride=1*, *no_max_pool=False*, *shortcut_type='B'*, *widen_factor=1.0*, *n_classes=400*)

Instantiate 3DResNet

Parameters

- **block** (*nn.Module*) – Block definition
- **layers** (*list[int]*) – List of number of blocks for 3d resnet
- **block_inplanes** (*list[int]*) – In-channel count per block
- **n_input_channels** (*int, optional*) – Number of input channels. Defaults to 3.
- **conv1_t_size** (*int, optional*) – Convolution input kernel size. Defaults to 7.
- **conv1_t_stride** (*int, optional*) – Convolution input stride size. Defaults to 1.
- **no_max_pool** (*bool, optional*) – Whether to not apply max pooling or not. Defaults to False.
- **shortcut_type** (*str, optional*) – Whether to apply downsampling or not. Defaults to 'B'.
- **widen_factor** (*float, optional*) – Widen factor. Defaults to 1.0.
- **n_classes** (*int, optional*) – Number of classes in output. Defaults to 400.

forward(*x*)

Apply ResNet3D to Layer Input

Parameters
x (*torch.Tensor*) – Layer Input

Returns
Layer Output

Return type
torch.Tensor

training: **bool**

unimodals.res3d.**generate_model**(*model_depth*, ***kwargs*)

Generate model given different standard model depths.

10.1.6 Module contents

CHAPTER

ELEVEN

UTILITIES

11.1 utils package

11.1.1 Submodules

11.1.2 utils.AUPRC module

Implements utils for getting AUPRC score given input and output pairs.

utils.AUPRC.**AUPRC**(*pts*)

Get average precision given a list of (true, predicted) pairs.

Parameters
pts (*List*) – List of true, predicted pairs

Returns
Average Precision Score.

Return type
float

utils.AUPRC.**ptsort**(*tu*)

Return first element of input.

11.1.3 utils.aux_models module

Implements a variety of modules for path-based dropout.

@author: juanma

class utils.aux_models.**AlphaScalarMultiplication**(*size_alpha_x*, *size_alpha_y*)

Bases: Module

Multiplies output element-wise by a scalar.

__init__(*size_alpha_x*, *size_alpha_y*)

Initialize AlphaScalarMultiplication module.

Parameters

- **size_alpha_x** (*int*) – Feature size for x input.
- **size_alpha_y** (*int*) – Feature size for y input.

forward(*x*, *y*)

Apply Alpha Scalar Multiplication to both inputs, followed by a sigmoid.

Parameters

- **x** (*torch.Tensor*) – Layer input
- **y** (*torch.Tensor*) – Layer input

Returns
Layer output

Return type
torch.Tensor

training: bool

class utils.aux_models.**AlphaVectorMultiplication**(*size_alpha*)

Bases: Module

Multiplies output element-wise by a vector.

__init__(*size_alpha*)

Initialize AlphaVectorMultiplication module.

Parameters
size_alpha (*int*) – Size of alpha module.

forward(*x*)

Apply Alpha Vector Multiplication to input.

Parameters
x (*torch.Tensor*) – Layer input

Returns
Layer output

Return type
torch.Tensor

training: bool

class utils.aux_models.**AuxiliaryHead**(*num_classes*, *filters=96*)

Bases: Module

Implements Auxiliary Head Module.

__init__(*num_classes*, *filters=96*)

Instantiate AuxiliaryHead Module.

Parameters

- **num_classes** (*int*) – Number of classes in output
- **filters** (*int, optional*) – Number of hidden channels. Defaults to 96.

forward(*x*)

Apply AuxiliaryHead to Layer Input.

Parameters
x (*torch.Tensor*) – Layer input

Returns
Layer output

Return type
torch.Tensor

`training: bool`

`class utils.aux_models.Cell(operation_labels, configuration_indexes, connections, args)`

Bases: `Module`

Implements a convnet classifier, using CellBlock instances and path-based dropout.

Generally unused.

`__init__(operation_labels, configuration_indexes, connections, args)`

Instantiate Cell Module.

Parameters

- **operation_labels** (*list*) – List of operation labels
- **configuration_indexes** (*list*) – list of configuration indexes
- **connections** (*list*) – list of connections
- **args** (*list*) – list of args for Cell

`forward(x1, x2)`

Apply cell to layer input, and add residual connection.

Parameters

- **x1** (*torch.Tensor*) – Input tensor 1
- **x2** (*torch.Tensor*) – Input tensor 2

Returns
Output Tensor

Return type
torch.Tensor

`training: bool`

`class utils.aux_models.CellBlock(op1_type, op2_type, args)`

Bases: `Module`

Implements a block of convolution cells, with path-based dropout.

`__init__(op1_type, op2_type, args)`

Instatiates CellBlock Module.

Parameters

- **op1_type** (*int | str*) – First convolution type
- **op2_type** (*int | str*) – Second convolution type
- **args** (*obj*) – Arguments for path-dropping and input/output channel number.

`forward(x1, x2)`

Apply block to layer input, and add residual connection.

Parameters

- **x1** (*torch.Tensor*) – Input tensor 1
- **x2** (*torch.Tensor*) – Input tensor 2

Returns
Output Tensor

Return type
torch.Tensor

`training: bool`

`class utils.aux_models.ChannelPadding(pad)`

Bases: `Module`

Applies Channel Padding to input.

`__init__(pad)`

Initialize Tensor1DLateralPadding Module.

Parameters
pad (*int*) – Padding amount

`forward(inputs)`

Apply Channel Padding to input.

Parameters
inputs (*torch.Tensor*) – Layer input

Returns
Layer output

Return type
torch.Tensor

`training: bool`

`class utils.aux_models.ConvBranch(in_planes, out_planes, kernel_size, separable=False)`

Bases: `Module`

Implements a convolution computational path for path-based dropout.

`__init__(in_planes, out_planes, kernel_size, separable=False)`

Initialize ConvBranch Module.

Parameters

- **in_planes** (*int*) – Input channel count
- **out_planes** (*int*) – Output channel count
- **kernel_size** (*int*) – Kernel size
- **separable** (*bool, optional*) – Whether to use Separable convolutions or not. Defaults to False.

`forward(x)`

Apply ConvBranch to Layer Input.

Parameters
x (*torch.Tensor*) – Layer input

Returns
Layer output

Return type
torch.Tensor

training: bool

utils.aux_models.**CreateOp**(*conv_type*, *input_planes=64*, *output_planes=64*)

Given a type of convolution, and the input/output channels, instatiate a convolution module.

Parameters

- **conv_type** (*int | string*) – Type of convolution.
- **input_planes** (*int, optional*) – Input channel number. Defaults to 64.
- **output_planes** (*int, optional*) – Output channel number. Defaults to 64.

Raises

NotImplementedError – Convolution not implemented.

Returns

Convolution instance

Return type

nn.Module

class utils.aux_models.**DropPath**(*keep_prob=0.9*)

Bases: Module

Implements path-based dropout.

__init__(*keep_prob=0.9*)

Initialize DropPath module.

Parameters

keep_prob (*float, optional*) – Probability to keep this path in the training pass. Defaults to 0.9.

forward(*x*, *other_dropped=False*)

Apply path-dropping to layer input.

Parameters

- **x** (*torch.Tensor*) – Layer Input
- **other_dropped** (*bool, optional*) – Whether to always drop or not. Defaults to False.

Returns

Tuple of the tensor (zeros if dropped), and a boolean of if the tensor was dropped.

Return type

tuple(tensor, was_dropped)

training: bool

class utils.aux_models.**FactorizedReduction**(*in_planes*, *out_planes*, *stride=2*)

Bases: Module

Implements Factozied Reduction.

Reduce both spatial dimensions (width and height) by a factor of 2, and potentially to change the number of output filters https://github.com/melodyguan/enas/blob/master/src/cifar10/general_child.py#L129

__init__(*in_planes*, *out_planes*, *stride=2*)

Initialize FactorizedReduction Module.

Parameters

- **in_planes** (*int*) – Input channel count

- **out_planes** (*int*) – Output channel count
- **stride** (*int, optional*) – Stride of convolutions. Defaults to 2.

forward(*x*)

Apply factorized reduction to layer input.

Parameters
x (*torch.Tensor*) – Layer Input

Returns
Layer Output

Return type
torch.Tensor

training: **bool**

class utils.aux_models.**FixedCell**(*operation_labels*, *configuration_indexes*, *connections*, *args*)

Bases: Module

Implements cell with fixed connections and no path-based dropout.

Generally unused, and probably buggy.

__init__(*operation_labels*, *configuration_indexes*, *connections*, *args*)

Instantiate Cell Module.

Parameters

- **operation_labels** (*list*) – List of operation labels
- **configuration_indexes** (*list*) – list of configuration indexes
- **connections** (*list*) – list of connections
- **args** (*list*) – list of args for Cell

forward(*x1*, *x2*)

Apply cell to layer input, and add residual connection.

Parameters

- **x1** (*torch.Tensor*) – Input tensor 1
- **x2** (*torch.Tensor*) – Input tensor 2

Returns
Output Tensor

Return type
torch.Tensor

training: **bool**

class utils.aux_models.**GlobalPooling1D**

Bases: Module

Implements 1D Global Average Pooling.

__init__()

Initialize GlobalPooling1D module.

forward(*x*)

Apply 1D Global Average Pooling to input.

Parameters
inputs (*torch.Tensor*) – Layer input

Returns
Layer output

Return type
torch.Tensor

training: **bool**

class utils.aux_models.**GlobalPooling2D**

Bases: Module

Implements 2D Global Average Pooling.

__init__()

Initialize GlobalPooling2D module.

forward(*x*)

Apply 2D Global Average Pooling to input.

Parameters
inputs (*torch.Tensor*) – Layer input

Returns
Layer output

Return type
torch.Tensor

training: **bool**

class utils.aux_models.**Identity**

Bases: Module

Implements an Identity Module.

forward(*inputs*)

Apply Identity to Layer Input.

Parameters
inputs (*torch.Tensor*) – Layer input

Returns
Layer output

Return type
torch.Tensor

training: **bool**

class utils.aux_models.**IdentityModule**

Bases: Module

Implements an Identity Module.

forward(*inputs*)

Apply Identity to Layer Input.

Parameters
inputs (*torch.Tensor*) – Layer input

Returns
Layer output

Return type
torch.Tensor

training: bool

class utils.aux_models.**Maxout**(*d*, *m*, *k*)

Bases: Module

Implements Maxout module.

__init__(*d*, *m*, *k*)

Initialize Maxout object.

Parameters

- **d** (*int*) – Input dimension.
- **m** (*int*) – Number of features remeaining after Maxout.
- **k** (*int*) – Pool Size

forward(*inputs*)

Apply Maxout to input.

Parameters
inputs (*torch.Tensor*) – Layer input

Returns
Layer output

Return type
torch.Tensor

training: bool

class utils.aux_models.**PoolBranch**(*in_planes*, *out_planes*, *avg_or_max*)

Bases: Module

Implements max pooling operations with 1x1 convolutions to fix output size.

__init__(*in_planes*, *out_planes*, *avg_or_max*)

Initialize PoolBranch module.

Parameters

- **in_planes** (*int*) – Input channel count
- **out_planes** (*int*) – Output channel count
- **avg_or_max** (*str*) – Whether to use average pooling ('avg') or max pooling 'max'

Raises
ValueError – Unknown Pooling Type.

forward(*x*)

Apply PoolBranch to Layer Input.

Parameters
x (*torch.Tensor*) – Layer input

Returns
Layer output

Return type
torch.Tensor

training: **bool**

class utils.aux_models.**SeparableConv**(*in_planes*, *out_planes*, *kernel_size*, *bias=False*)

Bases: Module

Implements Separable Convolutions.

__init__(*in_planes*, *out_planes*, *kernel_size*, *bias=False*)

Initialize SeparableConv Module.

Parameters

- **in_planes** (*int*) – Number of input channels.
- **out_planes** (*int*) – Number of output channels.
- **kernel_size** (*int*) – Size of kernel
- **bias** (*bool, optional*) – Whether to add a bias to each convolution or not. Defaults to False.

forward(*x*)

Apply Separable Convolution to Layer Input.

Parameters
inputs (*torch.Tensor*) – Layer input

Returns
Layer output

Return type
torch.Tensor

training: **bool**

class utils.aux_models.**SeparableConvOld**(*in_planes*, *out_planes*, *kernel_size*, *bias=False*)

Bases: Module

(deprecated) Implements 1D Separable Convolutions.

__init__(*in_planes*, *out_planes*, *kernel_size*, *bias=False*)

Initialize SeparableConvOld Module.

Parameters

- **in_planes** (*int*) – Number of input channels.
- **out_planes** (*int*) – Number of output channels.
- **kernel_size** (*int*) – Size of kernel
- **bias** (*bool, optional*) – (unused) Whether to add a bias to each convolution or not. Defaults to False.

forward(*x*)

Apply 1D Separable Convolution to Layer Input.

Parameters
inputs (*torch.Tensor*) – Layer input

Returns
Layer output

Return type
torch.Tensor

training: **bool**

class utils.aux_models.**Tensor1DLateralPadding**(*pad*)

Bases: Module

Applies 1DLateral Padding to input.

__init__(*pad*)

Initialize Tensor1DLateralPadding Module.

Parameters
pad (*int*) – Padding amount

forward(*inputs*)

Apply Lateral Padding to input.

Parameters
inputs (*torch.Tensor*) – Layer input

Returns
Layer output

Return type
torch.Tensor

training: **bool**

class utils.aux_models.**WeightedCrossEntropyWithLogits**(*pos_weight*)

Bases: Module

Implements cross entropy weighted by given weights.

__init__(*pos_weight*)

Initialize WeightedCrossEntropyWithLogits module.

Parameters
pos_weight (*np.array*) – Weight for each position in batch.

forward(*logits*, *targets*)

Get WeightedCrossEntropy Loss.

Parameters

- **logits** (*torch.Tensor*) – Logit Tensor
- **targets** (*torch.Tensor*) – Target labels

Returns
Weighted cross entropy

Return type
torch.Tensor

`training: bool`

utils.aux_models.**random**() → x in the interval [0, 1).

11.1.4 utils.evaluation_metric module

Implements various evaluation metrics for accuracies and MOSI/MOSEI.

utils.evaluation_metric.**eval_mosei_senti**(*results*, *truths*, *exclude_zero=False*)

Print out MOSEI metrics given results and ground truth.

Parameters

- **results** (*list*) – List of predicted results
- **truths** (*list*) – List of ground truth
- **exclude_zero** (*bool, optional*) – Whether to include zero or not. Defaults to False.

utils.evaluation_metric.**eval_mosei_senti_return**(*results*, *truths*, *exclude_zero=False*)

Evaluate MOSEI and return metric list.

Parameters

- **results** (*np.array*) – List of predicated values.
- **truths** (*np.array*) – List of true values.
- **exclude_zero** (*bool, optional*) – Whether to exclute zero. Defaults to False.

Returns
Return statistics for MOSEI.

Return type
tuple(mae, corr, mult_a7, f_score, accuracy)

utils.evaluation_metric.**eval_mosi**(*results*, *truths*, *exclude_zero=False*)

Evaluate MOSI results given predictions and ground truth.

Same as MOSEI evaluation.

Parameters

- **results** (*list*) – List of predicted results
- **truths** (*list*) – List of ground truth
- **exclude_zero** (*bool, optional*) – Whether to include zero or not. Defaults to False.

utils.evaluation_metric.**multiclass_acc**(*preds*, *truths*)

Compute the multiclass accuracy w.r.t. groundtruth.

Parameters

- **preds** – Float array representing the predictions, dimension (N,)
- **truths** – Float/int array representing the groundtruth classes, dimension (N,)

Returns
Classification accuracy

utils.evaluation_metric.**weighted_accuracy**(*test_preds_emo*, *test_truth_emo*)

Compute multiclass accuracy weighted by class occurence.

Parameters

- **test_preds_emo** (*np.array*) – List of predicted labels
- **test_truth_emo** (*np.array*) – List of true labels.

Returns
Weighted classification accuracy.

Return type
float

11.1.5 utils.helper_modules module

Defines some helper nn.module instances.

class utils.helper_modules.**Sequential2**(*a*, *b*)

Bases: Module

Implements a simpler version of sequential that handles inputs with 2 arguments.

__init__(*a*, *b*)

Instatiate Sequential2 object.

Parameters

- **a** (*nn.Module*) – First module to sequence
- **b** (*nn.Module*) – Second module

forward(*x*)

Apply Sequential2 modules to layer input.

Parameters
x (*torch.Tensor*) – Layer Input

Returns
Layer Output

Return type
torch.Tensor

training: **bool**

11.1.6 utils.scheduler module

Implements learning rate schedulers used in training loops.

class utils.scheduler.**FixedScheduler**(*lr*)

Bases: object

Implements a fixed learning rate for each epoch.

__init__(*lr*)

Initialize a FixedScheduler Object.

Parameters
lr (*float*) – Learning rate

step()

Update learning rate for scheduler.

Returns
Updated learning rate

Return type
float

update_optimizer(*optimizer*)

Update optimizer instance with current learning rate.

Parameters
optimizer (*nn.optim.Optimizer*) – Optimizer instance to modify

class utils.scheduler.**LRCosineAnnealingScheduler**(*eta_max*, *eta_min*, *Ti*, *Tmultiplier*, *num_batches_per_epoch*)

Bases: object

Implements an LRCosineAnnealingScheduler for lr adjustment.

__init__(*eta_max*, *eta_min*, *Ti*, *Tmultiplier*, *num_batches_per_epoch*)

Initialize LRCosineAnnealingScheduler Object.

Parameters

- **eta_max** (*float*) – Max eta.
- **eta_min** (*float*) – Minimum eta.
- **Ti** (*float*) – Initial temperature
- **Tmultiplier** (*float*) – Temperature multiplier
- **num_batches_per_epoch** (*int*) – Number of batches per epoch.

step()

Apply scheduler one step, and adjust the learning rate accordingly.

Returns
Adjusted learning rate

Return type
float

update_optimizer(*optimizer*)

Apply current scheduler learning rate to optimizer.

Parameters
optimizer (*torch.optim.Optimizer*) – Torch optimizer instance.

11.1.7 utils.search_tools module

Created on Tue Sep 11 18:42:17 2018 @author: juanma

utils.search_tools.**compute_temperature**(*iteration*, *init*, *final*, *decay*)

Compute temperature for a given round of the MFAS procedure.

Parameters

- **iteration** (*int*) – Iteration index

- **init** (*float*) – Initial temperature
- **final** (*float*) – Final temperature
- **decay** (*float*) – Temperature decay rate

Returns
Temperature for this round of MFAS.

Return type
float

utils.search_tools.**merge_unfolded_with_sampled**(*previous_top_k_configurations*, *unfolded_configurations*, *layer*)

Given a list of top k configurations, and an unfolded single layer configuration, merge them together at the given layer index.

Parameters

- **previous_top_k_configurations** (*list*) – List of configurations of size (seq_len, 3)
- **unfolded_configurations** (*list*) – Configuration for a single layer (,3)
- **layer** (*int*) – Index of layer to add unfolded configuration to.

Raises
ValueError – If there are no top k configurations, layer should be 0.

Returns
Merged list of configurations.

Return type
list

utils.search_tools.**predict_accuracies_with_surrogate**(*configurations*, *surrogate*, *device*)

Use surrogate to predict the effectiveness of different input configurations.

Parameters

- **configurations** (*list[dict]*) – List of configurations to search from.
- **surrogate** (*nn.Module*) – Learned surrogate cost model for configuration space.
- **device** (*string*) – Device to place computation on.

Returns
Accuracy per configuration.

Return type
list[float]

utils.search_tools.**sample_k_configurations**(*configurations*, *accuracies_*, *k*, *temperature*)

Sample k configurations from list of configurations and accuracies, based on past performance.

Parameters

- **configurations** (*list*) – List of configurations.
- **accuracies** (*list*) – List of accuracies for each config.
- **k** (*int*) – Number of configurations to sample on.
- **temperature** (*float*) – Temperature for the sample distribution.

Returns
List of sampled configurations.

Return type
list

utils.search_tools.**sample_k_configurations_directly**(*k*, *max_progression_levels*, *get_possible_layer_configurations_fun*)

Sample k configurations given a set number of progression_levels.

Parameters

- **k** (*int*) – Number of configurations to sampl.e
- **max_progression_levels** (*int*) – Maximum number of sample configurations.
- **get_possible_layer_configurations_fun** (*fn*) – Function to get layer configurations given some index input.

Returns
List of sampled configurations.

Return type
list

utils.search_tools.**sample_k_configurations_uniform**(*configurations*, *k*)

Sample k configurations uniformly.

Parameters

- **configurations** (*list*) – List of configurations to sample from.
- **k** (*int*) – Number of configurations to sample.

Returns
List of sampled configurations.

Return type
list

utils.search_tools.**train_surrogate**(*surrogate*, *surrogate_dataloader*, *surrogate_optimizer*, *surrogate_criterion*, *ep*, *device*)

Train surrogate model using surrogate dataloader.

Parameters

- **surrogate** (*nn.Module*) – Surrogate cost model instance.
- **surrogate_dataloader** (*torch.utils.data.DataLoader*) – Data loader for surrogate instance, mapping configurations to accuracies
- **surrogate_optimizer** (*torch.optim.Optimizer*) – Optimizer for surrogate parameters
- **surrogate_criterion** (*nn.Module*) – Loss function for surrogate instance.
- **ep** (*int*) – Number of epochs.
- **device** (*string*) – Device to train on.

Returns
Loss of surrogate with current training data.

Return type
float

utils.search_tools.**update_surrogate_dataloader**(*surrogate_dataloader*, *configurations*, *accuracies*)

Update surrogate dataloader with new configurations.

Parameters

- **surrogate_dataloader** (SurrogateDataloader) – Data Loader for Surrogate Cost Model.
- **configurations** (*list*) – List of new configurations to add.
- **accuracies** (*list[float]*) – List of accuracies to train cost model with.

11.1.8 utils.surrogate module

Created on Thu Jul 26 12:10:35 2018 @author: juanma

class utils.surrogate.**SimpleRecurrentSurrogate**(*num_hidden=100*, *number_input_feats=3*, *size_ebedding=100*)

Bases: Module

Defines simple surrogate for MFAS using a single LSTM layer.

__init__(*num_hidden=100*, *number_input_feats=3*, *size_ebedding=100*)

Initialize SimpleRecurrentSurrogate Module.

Parameters

- **num_hidden** (*int, optional*) – Hidden dimension size of LSTM. Defaults to 100.
- **number_input_feats** (*int, optional*) – Input dimension size. Defaults to 3.
- **size_ebedding** (*int, optional*) – Hidden dimension size before LSTM portion. Defaults to 100.

eval_model(*sequence_of_operations_np*, *device*)

Apply SimpleRecurrentSurrogate to a list of operations (a configuration) to get accuracy predictions.

Parameters

- **sequence_of_operations_np** (*np.array[int]*) – List of operations for this configuration. Size len_seq x input_size.
- **device** (*torch.utils.data.device*) – Device to train on.

Returns
Array of predicted accuracies.

Return type
np.array

forward(*sequence_of_operations*)

Apply SimpleRecurrentSurrogate to list of configurations, to get accuracy predictions.

Parameters
sequence_of_operations (*list*) – List of configurations to predict accuracies.

Returns
Predicted accuracies.

Return type
nn.Tensor

training: **bool**

class utils.surrogate.**SurrogateDataloader**

Bases: object

Implements a data loader for the surrogate instance, predicting accuracies from configurations.

__init__()

Initialize SurrogateDataloader Instance.

add_datum(*datum_conf*, *datum_acc*)

Add data to surrogate data loader

Parameters

- **datum_conf** (*list*) – List of operations for a configuration.
- **datum_acc** (*list[float]*) – Accuracies for this configuration.

get_data(*to_torch=False*)

Get data for training

Parameters
to_torch (*bool, optional*) – Whether to turn output to torch tensors. Defaults to False.

Returns
Data for surrogate instance to train on.

Return type
list[np.array|torch.tensor]

get_k_best(*k*)

Get K best configurations, given all that has been sampled so far.

Parameters
k (*int*) – Number of top configurations to get.

Returns
Tuple of the list of configurations, their accuracies, and their position in the dataloader.

Return type
tuple(configs, accuracies, index)

utils.surrogate.**train_simple_surrogate**(*model*, *criterion*, *optimizer*, *data_tensors*, *num_epochs*, *device*)

Train simple surrogate for MFAS procedure.

Parameters

- **model** (*nn.Module*) – Model to train on.
- **criterion** (*nn.Module*) – Loss function to train on.
- **optimizer** (*nn.optim.Optimizer*) – Optimizer to apply.
- **data_tensors** (*torch.Tensor*) – Dataset to train on.
- **num_epochs** (*int*) – Number of epochs to train this surrogate on.
- **device** (*torch.device*) – Device to train on.

Returns
Loss of this surrogate.

Return type
float

11.1.9 Module contents

CHAPTER

TWELVE

INDICES AND TABLES

- genindex
- modindex
- search

PYTHON MODULE INDEX

INDEX

Symbols

A

B

C

D

E

F

G

N

S

T

U

V

W

Z