AIToolbox

Release 1.3.0

Marko Vidoni

Apr 17, 2021

COMPONENTS:

CHAPTER

ONE

TORCHTRAIN

aitoolbox.torchtrain is the main user-facing API of the AIToolbox package. It incorporates the PyTorch model training via the train loop engine as well as automatic experiment progress and performance tracking. The experiment results are either stored only locally or if desired also automatically synced to the selected cloud storage (AWS S3 or Google Cloud Storage).

1.1 Train Loop

`TrainLoop` and it's module *aitoolbox.torchtrain.train_loop.train_loop* is at the core of and probably most important component of the entire *AIToolbox* package.

Common to all available TrainLoops is the *PyTorch* model training loop engine which automatically handles the deep learning training process. As part of this it does the batch feeding of data into the model, calculating loss and updating parameters for a specified number of epochs.

`torchtrain` and by extension `TrainLoop` has been designed with the ease of use in mind. One of the main design principles was to keep as much training code as possible exactly the same as would be used in normally *PyTorch*. Consequently, the user can define the dataset, dataloader and models in exactly the same way as it would be done when training directly with core *PyTorch*. Having no need to modify the definitions of common *PyTorch* training components in order to use torchtrain makes it very user-friendly and allows the user to apply torchtrain directly to projects which initially weren't even coded with AIToolbox in mind.

To train the model, all the user has to do is provide the TrainLoop with the model, train / validation / test dataloaders, loss function and the optimizer. That's it.

Once the TrainLoop with all the necessary components has been created all that's left is to start training the model. Common to all the available TrainLoops is the *aitoolbox.torchtrain.train_loop.train_loop.TrainLoop.fit()* method which initiates the training process. The `.fit()` method will train the provided model on the given training dataset in the training dataloader for the specified number of epochs.

Note: In order to use the *aitoolbox.torchtrain.train_loop* the user has to define their models as a *aitoolbox.torchtrain.model.TTModel* which is a slightly modified AIToolbox specific variation of the core PyTorch `torch.nn.Module`. Please have a look at the *TTModel* section of the documentation in order to learn how to define your TTModels compatible with TrainLoop supported training.

1.1.1 TrainLoop Variations

aitoolbox.torchtrain.train_loop module consists of submodules *aitoolbox.torchtrain.train_loop.train_loop* and *aitoolbox.torchtrain.train_loop.train_loop_tracking* which implement four different `TrainLoop` variations:

- *aitoolbox.torchtrain.train_loop.train_loop.TrainLoop*
- *aitoolbox.torchtrain.train_loop.train_loop_tracking.TrainLoopCheckpoint*
- *aitoolbox.torchtrain.train_loop.train_loop_tracking.TrainLoopEndSave*
- *aitoolbox.torchtrain.train_loop.train_loop_tracking.TrainLoopCheckpointEndSave*

The above listed TrainLoop options can be distinguished based on the varying extent of the automatic experiment tracking they do on top of the core training loop functionality. The available TrainLoops follow this naming convention:

- name includes `Checkpoint` keyword: the TrainLoop will automatically save the model after each training epoch
- name includes `EndSave` keyword: the TrainLoop will automatically evaluate final model performance and save the final model at the end of the training

1.1.1.1 TrainLoop

The simplest TrainLoop version which only performs the model training and does no experiment tracking and performance evaluation.

The API can be found in: *aitoolbox.torchtrain.train_loop.train_loop.TrainLoop*.

Example of the `TrainLoop` used to train the model:

```
from aitoolbox.torchtrain.train_loop import *


model = CNNModel()  # TTModel based neural model
train_loader = DataLoader(...)
val_loader = DataLoader(...)
test_loader = None

optimizer = optim.Adam(model.parameters(), lr=0.001, betas=(0.9, 0.999))
criterion = nn.NLLLoss()

tl = TrainLoop(model,
               train_loader, val_loader, test_loader,
               optimizer, criterion)

model = tl.fit(num_epochs=10)
```

1.1.1.2 TrainLoopCheckpoint

Same training process as in TrainLoop with additional automatic model checkpointing (saving) after every epoch. Model saving can be done only to the local disk or also to the cloud storage such as AWS S3.

The API can be found in: *aitoolbox.torchtrain.train_loop.train_loop_tracking.TrainLoopCheckpoint*.

```
from aitoolbox.torchtrain.train_loop import *
from aitoolbox.experiment.result_package.basic_packages import␣
↪ClassificationResultPackage


hyperparams = {
    'lr': 0.001,
    'betas': (0.9, 0.999)
}

model = CNNModel()  # TTModel based neural model
train_loader = DataLoader(...)
val_loader = DataLoader(...)
test_loader = DataLoader(...)

optimizer = optim.Adam(model.parameters(), lr=hyperparams['lr'], betas=hyperparams[
↪'betas'])
criterion = nn.NLLLoss()

tl = TrainLoopCheckpoint(
    model,
    train_loader, val_loader, test_loader,
    optimizer, criterion,
    project_name='train_loop_examples', experiment_name='TrainLoopCheckpoint_example',
    local_model_result_folder_path='results_dir',
    hyperparams=hyperparams,
    cloud_save_mode='s3', bucket_name='cloud_results'  # bucket_name should be set to␣
↪the bucket on your S3
)

model = tl.fit(num_epochs=10)
```

1.1.1.3 TrainLoopEndSave

Same training process as in TrainLoop with additional automatic model checkpointing (saving) and model performance evaluation at the end of the training process. This way the TrainLoop ensures experiment tracking a the end of the training. Model and experiment results saving can be done only to the local disk or also to the cloud storage such as AWS S3.

The API can be found in: *aitoolbox.torchtrain.train_loop.train_loop_tracking.TrainLoopEndSave*.

For information about the `ResultPackage` used in this example, have a look at the *Result Package* section.

```
from aitoolbox.torchtrain.train_loop import *
from aitoolbox.experiment.result_package.basic_packages import␣
↪ClassificationResultPackage


```

(continues on next page)

(continued from previous page)

```
hyperparams = {
    'lr': 0.001,
    'betas': (0.9, 0.999)
}

model = CNNModel()  # TTModel based neural model
train_loader = DataLoader(...)
val_loader = DataLoader(...)
test_loader = DataLoader(...)

optimizer = optim.Adam(model.parameters(), lr=hyperparams['lr'], betas=hyperparams[
↪'betas'])
criterion = nn.NLLLoss()

tl = TrainLoopEndSave(
    model,
    train_loader, val_loader, test_loader,
    optimizer, criterion,
    project_name='train_loop_examples', experiment_name='TrainLoopEndSave_example',
    local_model_result_folder_path='results_dir',
    hyperparams=hyperparams,
    val_result_package=ClassificationResultPackage(),
    test_result_package=ClassificationResultPackage(),
    cloud_save_mode='s3', bucket_name='cloud_results'  # bucket_name should be set to␣
↪the bucket on your S3
)

model = tl.fit(num_epochs=10)
```

1.1.1.4 TrainLoopCheckpointEndSave

For the most complete experiment tracking it is recommended to use the this TrainLoop option. At its core it is the same training process as in TrainLoop with additional automatic model checkpointing (saving) after each epoch as well as automatic model checkpointing and model performance evaluation at the end of the training process. This way the TrainLoop ensures full experiment tracking with the maximum extent. Model and experiment results saving can be done only to the local disk or also to the cloud storage such as AWS S3.

The API can be found in: *aitoolbox.torchtrain.train_loop.train_loop_tracking.TrainLoopCheckpointEndSave*.

For information about the `ResultPackage` used in this example, have a look at the *Result Package* section.

For a full working example of the `TrainLoopCheckpointEndSave` training, check out this TrainLoopCheckpointEndSave example training script.

```
from aitoolbox.torchtrain.train_loop import *
from aitoolbox.experiment.result_package.basic_packages import␣
↪ClassificationResultPackage


hyperparams = {
    'lr': 0.001,
    'betas': (0.9, 0.999)
}

model = CNNModel()  # TTModel based neural model
```

(continues on next page)

(continued from previous page)

```
train_loader = DataLoader(...)
val_loader = DataLoader(...)
test_loader = DataLoader(...)

optimizer = optim.Adam(model.parameters(), lr=hyperparams['lr'], betas=hyperparams[
↪'betas'])
criterion = nn.NLLLoss()

tl = TrainLoopCheckpointEndSave(
    model,
    train_loader, val_loader, test_loader,
    optimizer, criterion,
    project_name='train_loop_examples', experiment_name='TrainLoopCheckpointEndSave_
↪example',
    local_model_result_folder_path='results_dir',
    hyperparams=hyperparams,
    val_result_package=ClassificationResultPackage(),
    test_result_package=ClassificationResultPackage(),
    cloud_save_mode='s3', bucket_name='cloud_results'  # bucket_name should be set to␣
↪the bucket on your S3
)

model = tl.fit(num_epochs=10)
```

1.2 TTModel

Torchtrain Model - TTModel for short

To take advantage of the TrainLoop abstraction the user has to define their model as a class which is a standard way in core *PyTorch* as well. The only difference is that for TrainLoop supported training the model class has to be inherited from the AIToolbox specific *aitoolbox.torchtrain.model.TTModel* base class instead of *PyTorch* `torch.nn.Module`.

`TTModel` itself inherits from the normally used `nn.Module` class thus our models still retain all the expected *PyTorch* enabled functionality. The reason for using the TTModel super class is that TrainLoop requires users to implement two additional methods which describe how each batch of data is fed into the model when calculating the loss in the training mode and when making the predictions in the evaluation mode.

In total the user has to implement the following three methods when building a new model inherited from `TTModel`:

- `aitoolbox.torchtrain.model.TTModel.forward()` (inherited from `torch.nn.Module.forward()`)
- *aitoolbox.torchtrain.model.TTModel.get_loss()*
- *aitoolbox.torchtrain.model.TTModel.get_predictions()*

The code below shows the general skeleton all the TTModels have to follow to enable them to be trained with the TrainLoop:

```
from aitoolbox.torchtrain.model import TTModel


class MyNeuralModel(TTModel):
    def __init__(self):
        # model layers, etc.
```

(continues on next page)

(continued from previous page)

```
    def forward(self, x_data_batch):
        # The same method as required in the base PyTorch nn.Module
        ...
        # return prediction

    def get_loss(self, batch_data, criterion, device):
        # Get loss during training stage, called from fit() in TrainLoop
        ...
        # return batch loss

    def get_loss_eval(self, batch_data, criterion, device):
        # Get loss during evaluation stage. Normally just calls get_loss()
        return self.get_loss(batch_data, criterion, device)

    def get_predictions(self, batch_data, device):
        # Get predictions during evaluation stage
        # + return any metadata potentially needed for evaluation
        ...
        # return predictions, true_targets, metadata
```

For a full working example of the `TTModel` based model definition, check out this model example script.

1.3 Callbacks

For advanced model training experiments the basic logic offered in available TrainLoops might not be enough. Additional needed logic can be injected into the training procedure by using *callbacks* and providing them as a parameter list to `aitoolbox.torchtrain.train_loop.TrainLoop.fit()` function found in all TrainLoops.

1.3.1 Available Callbacks

AIToolbox by default already offers a wide selection of different useful callbacks which can be used to augment the base training procedure. These out of the box callbacks can be found in *aitoolbox.torchtrain.callbacks* module. There are several general categories of available callbacks:

- *aitoolbox.torchtrain.callbacks.basic* - general training augmentation
- *aitoolbox.torchtrain.callbacks.performance_eval* - model performance evaluation
- *aitoolbox.torchtrain.callbacks.model_save* - local / cloud based model saving
- *aitoolbox.torchtrain.callbacks.gradient* - model gradient reporting
- *aitoolbox.torchtrain.callbacks.model_load* - existing model loading at train start
- *aitoolbox.torchtrain.callbacks.tensorboard* - tensorboard training tracking

Example of the several basic callbacks used to infuse additional logic into the model training process:

```
from aitoolbox.torchtrain.train_loop import *
from aitoolbox.torchtrain.callbacks.basic import EarlyStopping, TerminateOnNaN,␣
↪AllPredictionsSame


model = CNNModel()  # TTModel based neural model
```

(continues on next page)

(continued from previous page)

```
train_loader = DataLoader(...)
val_loader = DataLoader(...)
test_loader = DataLoader(...)

optimizer = optim.Adam(model.parameters(), lr=0.001, betas=(0.9, 0.999))
criterion = nn.NLLLoss()

callbacks = [
    EarlyStopping(patience=3),
    TerminateOnNaN(),
    AllPredictionsSame(value=0.)
]

tl = TrainLoop(model,
               train_loader, val_loader, test_loader,
               optimizer, criterion)

model = tl.fit(num_epochs=10, callbacks=callbacks)
```

For a full working example which shows the use of multiple callbacks of various types, check out this fully tracked training experiment example.

1.3.2 Implementing New Callbacks

However when some completely new functionality is desired which is not available out of the box in AIToolbox the user can also implement their own custom callbacks. These can then be used as any other callback to further extend the training loop process.

1.3.2.1 AbstractCallback

The new callback can be implemented as a new class which is inheriting from the base callback *aitoolbox.torchtrain.callbacks.abstract.AbstractCallback*. All that the user has to do is to override and implement the methods corresponding to positions in the TrainLoop training process at which the newly developed callback should be executed. If a certain callback method is left unimplemented and thus left to the default from the parent `AbstractCallback` the callback has no effect on the TrainLoop at the corresponding position in the training process.

Callback execution is currently supported at the following positions in the TrainLoop via the following methods:

- *aitoolbox.torchtrain.callbacks.abstract.AbstractCallback.on_train_begin()*
- *aitoolbox.torchtrain.callbacks.abstract.AbstractCallback.on_epoch_begin()*
- *aitoolbox.torchtrain.callbacks.abstract.AbstractCallback.on_batch_begin()*
- *aitoolbox.torchtrain.callbacks.abstract.AbstractCallback.on_after_gradient_update()*
- *aitoolbox.torchtrain.callbacks.abstract.AbstractCallback.on_after_optimizer_step()*
- *aitoolbox.torchtrain.callbacks.abstract.AbstractCallback.on_batch_end()*
- *aitoolbox.torchtrain.callbacks.abstract.AbstractCallback.on_epoch_end()*
- *aitoolbox.torchtrain.callbacks.abstract.AbstractCallback.on_train_end()*

- *aitoolbox.torchtrain.callbacks.abstract.AbstractCallback.on_train_loop_registration()*
- *aitoolbox.torchtrain.callbacks.abstract.AbstractCallback.on_multiprocess_start()*

1.3.2.2 train_loop_obj

The most usable and thus important aspect of every callback is its ability to communicate and modify the encapsulating running TrainLoop. Every callback has a special attribute `aitoolbox.torchtrain.callbacks.abstract.AbstractCallback.train_loop_obj` which at the start of the TrainLoop training process gets assigned the reference (pointer) to the encapsulating TrainLoop object. In AIToolbox the process is called *TrainLoop registration* and is automatically done under the hood by the TrainLoop by calling the *aitoolbox.torchtrain.callbacks.abstract.AbstractCallback.register_train_loop_object()*.

Via the `train_loop_obj` the callback can thus have a complete access to and control of every aspect of the TrainLoop. While maybe dangerous for inexperienced users, this extensive low level control is especially welcome for the advanced research use of AIToolbox. After the train loop object registration inside the callback the reference to the encapsulating TrainLoop can be simply accessed from any implemented callback method via `self.train_loop_obj`.

1.3.2.3 Custom Callback Example

Example of a newly developed callback and its use in the TrainLoop:

```
from aitoolbox.torchtrain.train_loop import *
from aitoolbox.torchtrain.callbacks.abstract import AbstractCallback
from aitoolbox.torchtrain.callbacks.basic import EarlyStopping, TerminateOnNaN,␣
↪AllPredictionsSame


class MyDemoTrainingReportCallback(AbstractCallback):
    def __init__(self):
        super().__init__('simple callback example')

    def on_train_begin(self):
        experiment_start_time = self.train_loop_obj.experiment_timestamp
        print(f'Starting the training! Experiment started at: {experiment_start_time}
↪')

    def on_epoch_begin(self):
        current_epoch = self.train_loop_obj.epoch
        print(f'Starting new epoch num {current_epoch}')

    def on_epoch_end(self):
        val_predictions = self.train_loop_obj.predict_on_validation_set()
        print('Model predictions:')
        print(val_predictions)

    def on_train_end(self):
        print(f'End of training! Stopped at epoch {self.train_loop_obj.epoch}')

        test_predictions = self.train_loop_obj.predict_on_test_set()
        print('Model predictions:')
        print(test_predictions)
```

(continues on next page)

(continued from previous page)

```
model = CNNModel()  # TTModel based neural model
train_loader = DataLoader(...)
val_loader = DataLoader(...)
test_loader = DataLoader(...)

optimizer = optim.Adam(model.parameters(), lr=0.001, betas=(0.9, 0.999))
criterion = nn.NLLLoss()

callbacks = [
    MyDemoTrainingReportCallback(),
    EarlyStopping(patience=3),
    TerminateOnNaN(),
    AllPredictionsSame(value=0.)
]

tl = TrainLoop(model,
               train_loader, val_loader, test_loader,
               optimizer, criterion)

model = tl.fit(num_epochs=10, callbacks=callbacks)
```

1.3.2.4 AbstractExperimentCallback

In case of the developed callback is aimed at experiment tracking where information about the created experiment details such as project name, experiment name and path of the local experiment folder would be needed there is available also available the *aitoolbox.torchtrain.callbacks.abstract.AbstractExperimentCallback*. `AbstractExperimentCallback` has all the same properties as basic `AbstractCallback` and is extended with the convenience method *aitoolbox.torchtrain.callbacks.abstract.AbstractExperimentCallback.try_infer_experiment_details()* which extracts the experiment details from the running `TrainLoop` and infuses our callback with this additional needed information.

For the example of the `try_infer_experiment_details()` use in practice check this implementation: *aitoolbox.torchtrain.callbacks.performance_eval.ModelTrainHistoryPlot.on_train_loop_registration()*.

1.3.2.5 DDP Multi-Processing Callbacks

When the callbacks are used during the DistributedDataParallel TrainLoop (more about this can be found in *Multi-GPU Training*), by default they are executed in each of the running processes. This behaviour can be desired, however in certain situations the opposite is required and the callback should only be executed in one lead process.

When developing such a callback which is intended to be executed only in one of the spawned processes the torchtrain callbacks framework enables this via the `device_idx_execution` parameter which is part of every callback inherited from the `AbstractCallback`. It tells the TrainLoop engine as part of which process and corresponding *GPU device id* the callback should be executed. For exmaple if the callback has `device_idx_execution` set to 0, this means that the callback will only be executed as part of the process which is running on the first GPU. When `device_idx_execution` is set to `None` which is the default, the callback is executed inside every running process.

Simple example callback that gets executed in only the process running on the first GPU:

```
from aitoolbox.torchtrain.callbacks.abstract import AbstractCallback


class DemoFirstGPUCallback(AbstractCallback):
    def __init__(self):
        super().__init__('first GPU callback example',
                         device_idx_execution=0)

    def on_train_begin(self):
        ..... Some logic .....
```

1.4 Schedulers

TrainLoop-based training supports the use of learning rate schedulers. The built-in common learning rate schedulers can be found in the *aitoolbox.torchtrain.schedulers* sub-package. Currently AIToolbox comes out of the bag with the following scheduler types:

- *aitoolbox.torchtrain.schedulers.basic* - basic scheduler components and general schedulers
- *aitoolbox.torchtrain.schedulers.warmup* - schedulers based on HuggingFace Transformers

Schedulers are given to any TrainLoop type the same way as callbacks via the callbacks list provided to `fit()`.

1.4.1 Implementing New Schedulers

Under the hood the schedulers are just AIToolbox *Callbacks*. Consequently when desired, new learning rate schedulers can easily be implemented by just inheriting them from the commonly used *aitoolbox.torchtrain.callbacks.abstract.AbstractCallback* base class and implementing the necessary learning rate scheduling logic.

1.5 Multi-Loss and Multi-Optimizer

TODO

1.5.1 Multi-Loss Training

1.5.2 Multi-Optimizer Training

1.6 Multi-GPU Training

All TrainLoop versions in addition to single GPU also support multi-GPU training to achieve even faster training. Following the core *PyTorch* setup, two multi-GPU training approaches are available:

- `DataParallel` done via *aitoolbox.torchtrain.parallel.TTDataParallel*
- `DistributedDataParallel` done via *aitoolbox.torchtrain.parallel.TTDistributedDataParallel*

1.6.1 DataParallel

To use `DataParallel`-like multiGPU training with TrainLoop just switch the TrainLoop's `gpu_mode` parameter to `'dp'`:

```
from aitoolbox.torchtrain.train_loop import *
from aitoolbox.torchtrain.parallel import TTDataParallel


model = CNNModel()  # TTModel based neural model

train_loader = DataLoader(...)
val_loader = DataLoader(...)
test_loader = DataLoader(...)

optimizer = optim.Adam(model.parameters(), lr=0.001, betas=(0.9, 0.999))
criterion = nn.NLLLoss()

tl = TrainLoop(model,
               train_loader, val_loader, test_loader,
               optimizer, criterion,
               gpu_mode='dp')

model = tl.fit(num_epochs=10)
```

Check out a full DataParallel training example.

1.6.2 DistributedDataParallel

Distributed training on multiple GPUs via `DistributedDataParallel` is enabled by the TrainLoop itself under the hood by wrapping the *TTModel*-based model into *`aitoolbox.torchtrain.parallel.TTDistributedDataParallel`*. TrainLoop also automatically spawns multiple processes and initializes them. Inside each spawned process the model and all other necessary training components are moved to the correct GPU belonging to a specific process. Lastly, TrainLoop also automatically adds the *PyTorch* `DistributedSampler` to each of the provided data loaders in order to ensure different data batches go to different GPUs and there is no overlap.

To enable distributed training via DistributedDataParallel, the user has to set the TrainLoop's `gpu_mode` parameter to `'ddp'`.

```
from aitoolbox.torchtrain.train_loop import *


model = CNNModel()  # TTModel based neural model

train_loader = DataLoader(...)
val_loader = DataLoader(...)
test_loader = DataLoader(...)

optimizer = optim.Adam(model.parameters(), lr=0.001, betas=(0.9, 0.999))
criterion = nn.NLLLoss()

tl = TrainLoop(
    model,
    train_loader, val_loader, test_loader,
    optimizer, criterion,
    gpu_mode='ddp'
```

(continues on next page)

(continued from previous page)

```
)

model = tl.fit(num_epochs=10,
               num_nodes=1, node_rank=0, num_gpus=torch.cuda.device_count())
```

Check out a full DistributedDataParallel training example.

1.7 Automatic Mixed Precision Training

All the TrainLoop versions also support training with Automatic Mixed Precision (*AMP*). In the past this required using the Nvidia apex extension but from *PyTorch 1.6* onwards AMP functionality is built into core PyTorch and no separate instalation is needed. Current version of AIToolbox already supports the use of built-in PyTorch AMP.

The user only has to set the TrainLoop parameter `use_amp` to `use_amp=True` in order to use the default AMP initialization and start training the model in the mixed precision mode. If the user wants to specify custom AMP `GradScaler` initialization parameters, these should be provided as a dict parameter `use_amp={'init_scale': 2.**16, 'growth_factor': 2.0, ...}` to the TrainLoop. All AMP initializations and training related steps are then handled automatically by the TrainLoop.

You can read more about different AMP details in the PyTorch AMP documentation.

1.7.1 Single-GPU mixed precision training

Example of single-GPU AMP setup:

```
from aitoolbox.torchtrain.train_loop import *


train_loader = DataLoader(...)
val_loader = DataLoader(...)
test_loader = DataLoader(...)

model = CNNModel()  # TTModel based neural model

optimizer = optim.Adam(model.parameters(), lr=0.001, betas=(0.9, 0.999))
criterion = nn.NLLLoss()

tl = TrainLoop(model,
               train_loader, val_loader, test_loader,
               optimizer, criterion,
               use_amp=True)

model = tl.fit(num_epochs=10)
```

Check out a full AMP single-GPU training example.

1.7.2 Multi-GPU DDP mixed precision training

When training in the multi-GPU setting, the setup is mostly the same as in the single-GPU. All the user has to do is set accordingly the `use_amp` parameter of the TrainLoop and to switch its `gpu_mode` parameter to `'ddp'`. Under the hood, TrainLoop will initialize the model and the optimizer for AMP and start training using DistributedDataParallel approach.

Example of multi-GPU AMP setup:

```
from aitoolbox.torchtrain.train_loop import *


train_loader = DataLoader(...)
val_loader = DataLoader(...)
test_loader = DataLoader(...)

model = CNNModel()  # TTModel based neural model

optimizer = optim.Adam(model.parameters(), lr=0.001, betas=(0.9, 0.999))
criterion = nn.NLLLoss()

tl = TrainLoop(model,
               train_loader, val_loader, test_loader,
               optimizer, criterion,
               gpu_mode='ddp',
               use_amp=True)

model = tl.fit(num_epochs=10,
               num_nodes=1, node_rank=0, num_gpus=torch.cuda.device_count())
```

Check out a full AMP multi-GPU DistributedDataParallel training example.

1.8 Advanced Topics

This section presents more advanced low level aspects of the AIToolbox which are normally hidden from the user when training models. However, they could be potentially interesting to the developers who want to develop their own new components and extend AIToolbox functionality to better fit their specific use-cases.

1.8.1 Message Passing Service

Most of the time different components in AIToolbox operate either in isolation or communicate over specified APIs. While this is useful practice for error prevention in some cases less structured form of communication between components might be desired in order to simplify research development. One such example is the communication between different callbacks the user might provide to the TrainLoop. To support the convenient and easy development of callbacks and their communication the TrainLoop provides the *message passing service* implemented in *aitoolbox.torchtrain.train_loop.components.message_passing*.

1.8.1.1 MessageService Details

aitoolbox.torchtrain.train_loop.components.message_passing.MessageService is running as part of the TrainLoop and is exposed inside every provided callback via the `self.message_service`. When we want to pass some information from one callback to another callback (e.g. path where some intermediary results were saved) the sender callback has to send it into the *MessageService* by calling *aitoolbox.torchtrain.train_loop.components.message_passing.MessageService.write_message()* (inside the callback implementation that would be `self.message_service.write_message()`). Messages can be considered as a key-value pair with added message lifecycle setting.

Depending on the **message lifecycle setting**, the messages can be kept in the message service until the end of training, end of epoch or until first read. As such the message service allows the asynchronous and independent operation of callbacks enabling the users to add or remove callbacks from the training process as they will without running into interdependency issues. The message lifecycle settings can be imported from the *aitoolbox.torchtrain.train_loop.components.message_passing*. Currently supported settings are:

- `KEEP_FOREVER`
- `UNTIL_END_OF_EPOCH`
- `UNTIL_READ`
- `OVERWRITE`

In addition to writing messages, the message service of course also supports the reading of the accumulated messages. This can be achieved in any TrainLoop component having access to the *MessageService* (callbacks included) by calling *aitoolbox.torchtrain.train_loop.components.message_passing.MessageService.read_messages()*. This method will return all the messages accumulated under the specified key.

In our earlier example of one callback writing a message with the path to the stored intermediary results, the second callbacks tasked with processing the results or maybe saving them to the cloud would read that message with the data path and execute it's logic on the data originally provided by the first callback.

1.8.1.2 Example of MessageService in action

An actual example of such message passing between different callbacks can be observed in the implementations of *aitoolbox.torchtrain.callbacks.performance_eval.ModelTrainHistoryPlot* which sends the message containing the results path and the *aitoolbox.torchtrain.callbacks.basic.EmailNotification* which reads that message and uses the sent results path.

1.8.2 Model Prediction Store

In order to save compute time and prevent repetitive re-computation leading to the same output, TrainLoop utilizes the *aitoolbox.torchtrain.train_loop.components.model_prediction_store.ModelPredictionStore* which is used for results caching.

Especially when using multiple callbacks all executing the same computation, such as making predictions on the validation set this can get quite time consuming. To speed up training process TrainLoop will calculate the prediction on particular dataset as part of the current epoch only once and then cache the predictions. If as part of the same epoch another calculation of predictions on the same data set is requested, the TrainLoop will retrieve the cached results instead of recomputing them again. Currently the `ModelPredictionStore` supports caching the model loss and model prediction caching on the train, validation and test data sets.

As part of the TrainLoop the model prediction store cache lifecycle ends at the end of the epoch. All the cached model outputs are removed at the end of the epoch and the new epoch where the weights of the model will change is started with the clean prediction cache.

To most users this caching is visible as part of the TrainLoop's loss calculation methods:

- *aitoolbox.torchtrain.train_loop.train_loop.TrainLoop.evaluate_loss_on_train_set()*
- *aitoolbox.torchtrain.train_loop.train_loop.TrainLoop.evaluate_loss_on_validation_set()*
- *aitoolbox.torchtrain.train_loop.train_loop.TrainLoop.evaluate_loss_on_test_set()*

and as part of the TrainLoop's model prediction calculation methods:

- *aitoolbox.torchtrain.train_loop.train_loop.TrainLoop.predict_on_train_set()*
- *aitoolbox.torchtrain.train_loop.train_loop.TrainLoop.predict_on_validation_set()*
- *aitoolbox.torchtrain.train_loop.train_loop.TrainLoop.predict_on_test_set()*

Important to note here, is that by default TrainLoop will try to save compute time and cache model outputs when possible instead of recomputing them. However, if for a particular use case the user wants to get fresh recomputed loss or model predictions then the `force_prediction` parameter in any of the model output computation methods listed above has to be switched to `True`. This will cause them to ignore the cached values and recompute them from scratch.

1.8.3 Model Wrap and Batch Feed Definition

The preferred way of defining the model compatible with the TrainLoop is to implement it as the **TTModel** as discussed in the *TTModel* section.

The legacy approach of defining the model for the TrainLoop which still comes in handy in certain specific use cases was to implement a normal PyTorch nn.Module and define a separate **batch feed definition** for this particular model. Batch feed definitions are objects which need to be inherited from *aitoolbox.torchtrain.data.batch_model_feed_defs.AbstractModelFeedDefinition* and the user has to implement its abstract methods.

It can be seen that the abstract methods requiring the implementation as part of the *AbstractModelFeedDefinition* are exactly the same as those which need to be implemented as part of the new TTModel definition discussed in further detail in the *TTModel* section. While using TTModel is better for readability and experiment tracking on the other hand in some rare use cases operating on the core PyTorch *nn.Module* model is required instead of using the TTModel extension. For such cases the *nn.Module + model feed definition* combination option has been kept in the AIToolbox.

Last step that needs to be done in order to train the nn.Module with it's feed definition as part of the TrainLoop is to wrap the model and the feed definition into the *aitoolbox.torchtrain.model.ModelWrap*. TrainLoop will automatically detect the use of separate feed definition instead of the TTModel and execute the training based on the contents of the provided `ModelWrap`.

1.8.3.1 Example of the training with the model feed definition

For the practical example how the nn.Module can be paired together with its model feed definition and wrapped into the ModelWrapp for the TrainLoop training have a look at the this example training script.

CHAPTER

TWO

EXPERIMENT

aitoolbox.experiment defines the experiment tracking and performance evaluation components. Because all implemented components are completely independent from the TrainLoop engine they can be used either on their own in a more manual mode or as part of the TrainLoop functionality available in *aitoolbox.torchtrain*. Due to the independence of the components, certain elements, for performance evaluation can even be utilized for evaluation of non-PyTorch models.

In general, *aitoolbox.experiment* helps the user with the following:

- **Structured and reusable performance evaluation logic definition**
  - *aitoolbox.experiment.result_package*
  - *aitoolbox.experiment.core_metrics*
- **Tracked training performance history primitive**
  - *aitoolbox.experiment.training_history*
- **High level experiment tracking API**
  - *aitoolbox.experiment.experiment_saver*
  - *aitoolbox.experiment.local_experiment_saver*
- **Low level experiment tracking primitives for model saving and performance results saving**
  - *aitoolbox.experiment.local_save*
- **Saved model re-loading low level primitives**
  - *aitoolbox.experiment.local_load.local_model_load*

2.1 Result Package

Result Package found in *aitoolbox.experiment.result_package* defines a set of evaluation metrics that are used for the performance evaluation of the model on a certain ML task. For example, in the simple classification task, the corresponding result package would include metrics such as *accuracy*, *F1 score*, *ROC-AUC* and *PR-AUC*. Result packages can thus be thought of as wrappers around a set of evaluation metrics commonly used for different ML tasks.

The same as for all other components of *aitoolbox.experiment* module, when it comes to the usage of result packages, they can be either used in a standalone manually executed fashion for any kind of ML experiment evaluation. On the other hand, result packages can also be used in unison with the TrainLoop model training engine from the *aitoolbox.torchtrain*. There, the result package assumes the role of the *evaluation recipe* for a certain ML task. By providing the result package to the TrainLoop the user informs it how to automatically evaluate the model performance during or at the end of the training process.

2.1.1 Using Result Packages

Result Package implementations can be found in the *aitoolbox.experiment.result_package*. AIToolbox already comes with result packages for various popular ML tasks included out of the box. These can be found in the *aitoolbox.experiment.result_package.basic_packages*.

2.1.1.1 Result Package with torchtrain TrainLoop

When using result packages as part of TrainLoop supported training there are two main use-cases: as part of the *aitoolbox.torchtrain.callbacks.performance_eval.ModelPerformanceEvaluation* callback which optionally performs the model performance evaluation during the training, e.g. after each epoch, and on the other hand as part of the *"EndSave"* TrainLoop which automatically evaluates model's performance based on the provided result package at the end of the training.

```
from aitoolbox.torchtrain.train_loop import *
from aitoolbox.experiment.result_package.basic_packages import␣
↪ClassificationResultPackage
from aitoolbox.torchtrain.callbacks.performance_eval import \
    ModelPerformanceEvaluation, ModelPerformancePrintReport


hyperparams = {
    'lr': 0.001,
    'betas': (0.9, 0.999)
}

model = CNNModel()  # TTModel based neural model

train_loader = DataLoader(...)
val_loader = DataLoader(...)
test_loader = DataLoader(...)

optimizer = optim.Adam(model.parameters(), lr=hyperparams['lr'], betas=hyperparams[
↪'betas'])
criterion = nn.NLLLoss()

callbacks = [ModelPerformanceEvaluation(ClassificationResultPackage(), hyperparams,
                                        on_train_data=True, on_val_data=True),
             ModelPerformancePrintReport(['train_Accuracy', 'val_Accuracy'])]

tl = TrainLoopCheckpointEndSave(
    model,
    train_loader, val_loader, test_loader,
    optimizer, criterion,
    project_name='train_loop_examples',
    experiment_name='result_package_with_trainloop_example',
    local_model_result_folder_path='results_dir',
    hyperparams=hyperparams,
    val_result_package=ClassificationResultPackage(),
    test_result_package=ClassificationResultPackage()
)

model = tl.fit(num_epochs=10)
```

2.1.1.2 Standalone Result Package Use

As mentioned above, result packages are completely independent from TrainLoop engine and can thus also be used for a standalone model performance evaluation, even when not dealing with PyTorch models

```
from aitoolbox.experiment.result_package.basic_packages import␣
↪BinaryClassificationResultPackage


y_true = ...  # ground truth labels
y_predicted = ...  # predicted by the model

result_pkg = BinaryClassificationResultPackage()
result_pkg.prepare_result_package(y_true, y_predicted)

# get the results dict with performance results of all the metrics in the result␣
↪package
performance_results = result_pkg.get_results()
```

2.1.2 Implementing New Result Packages

Although AIToolbox already provides result packages for certain ML tasks sometimes the user wants do define a novel or unsupported performance evaluation metrics to properly evaluate the ML task at hand. The creation of new result packages in AIToolbox is supported and can be done very easily.

The new result package can be implemented as a new class which is inheriting from the base abstract result package *aitoolbox.experiment.result_package.abstract_result_packages.AbstractResultPackage* and implements the abstract method *aitoolbox.experiment.result_package.abstract_result_packages.AbstractResultPackage.prepare_results_dict()*.

Inside the `prepare_results_dict()` the user needs to implement the logic to evaluate the performance on desired performance metrics forming the result package. In order to perform the evaluation the predicted and ground truth values are normally needed. These are inserted into the package at run time (via `prepare_result_package()`) and exposed inside the result package via: `self.y_true` and `self.y_predicted` attributes. Logic inside the which the user needs to define, `prepare_results_dict()` should access the values in *y_true* and *y_predicted*, pass them through the desired performance metrics computations and return the results in the dict form. Inside the returned dict, keys should represent the evaluated metric names and values the corresponding evaluated performance metric values.

The performance metric computation as part of the result package can be directly implemented inside the result package class in the `prepare_results_dict()` method. However, especially in the case of more complex performance metric logic in order to ensure better reusability of the implemented metrics as well as more readable and structured code of the developed result packages it is common practice in the AIToolbox to implement performance metrics as a separate specialized metric class. This way the result packages become a lightweight wrappers around the selected performance metrics while the actual performance metric logic and calculation is done as part of the metric object instead of being done in the encapsulating result package. To learn more about the AIToolbox performance metric use and implementations have a look at the *Result Metric* documentation section.

2.1.2.1 Example or Result Package using AIToolbox Result Metric

```
from aitoolbox.experiment.result_package.abstract_result_packages import␣
↪AbstractResultPackage
from aitoolbox.experiment.core_metrics.classification import \
    AccuracyMetric, ROCAUCMetric, PrecisionRecallCurveAUCMetric


class ExampleClassificationResultPackage(AbstractResultPackage):
    def __init__(self):
        AbstractResultPackage.__init__(self, pkg_name='ExampleClassificationResult')

    def prepare_results_dict(self):
        accuracy_result = AccuracyMetric(self.y_true, self.y_predicted)
        roc_auc_result = ROCAUCMetric(self.y_true, self.y_predicted)
        pr_auc_result = PrecisionRecallCurveAUCMetric(self.y_true, self.y_predicted)

        return accuracy_result + roc_auc_result + pr_auc_result
```

2.1.2.2 Example of Result Package with Direct Performance Metric Calculation

```
from aitoolbox.experiment.result_package.abstract_result_packages import␣
↪AbstractResultPackage
from sklearn.metrics import accuracy_score, roc_auc_score, precision_recall_curve, auc


class ExampleClassificationResultPackage(AbstractResultPackage):
    def __init__(self):
        AbstractResultPackage.__init__(self, pkg_name='ExampleClassificationResult')

    def prepare_results_dict(self):
        accuracy = accuracy_score(self.y_true, self.y_predicted)
        roc_auc = roc_auc_score(self.y_true, self.y_predicted)

        precision, recall, thresholds = precision_recall_curve(self.y_true, self.y_
↪predicted)
        pr_auc = auc(recall, precision)

        return {'accuracy': accuracy, 'roc_auc': roc_auc, 'pr_auc': pr_auc}
```

2.2 Result Metric

Result metric (*aitoolbox.experiment.core_metrics*) is an abstraction built around the calculation of the single performance metric. It helps keep the code base more reusable and better structured, especially when used as part of the encapsulating *Result Package*.

AIToolbox comes out of the box with implemented several commonly used performance evaluation metrics implemented as result metrics. These can be found in:

- *aitoolbox.experiment.core_metrics.classification*
- *aitoolbox.experiment.core_metrics.regression*

2.2.1 Use of Result Metrics inside Result Packages

As it is described in the *Implementing New Result Packages* section, result metrics come in handy when developing the result packages which are wrapping together multiple metrics needed to evaluate a certain ML task. To support this chaining together of multiple performance metrics, the result metric abstraction offers a convenient metric concatenation and result package dictionary creation via the + operator. To create the dictionary holding all the performance metric results the user can simply write: `metric_1 + metric_2 + metric_3 + ...`. This makes the use of the + operator very convenient because the produced results dictionary format exactly matches that which is required when developing an encapsulating result package.

Example of result metric concatenation:

```
from aitoolbox.experiment.core_metrics.classification import \
    AccuracyMetric, ROCAUCMetric, PrecisionRecallCurveAUCMetric

accuracy_result = AccuracyMetric(y_true, y_predicted)
roc_auc_result = ROCAUCMetric(y_true, y_predicted)
pr_auc_result = PrecisionRecallCurveAUCMetric(y_true, y_predicted)

results_dict =  accuracy_result + roc_auc_result + pr_auc_result

# results_dict will hold:
# {'Accuracy': 0.95, 'ROC_AUC': 0.88, 'PrecisionRecall_AUC': 0.67}
```

2.2.2 Implementing New Result Metrics

When the needed result metric is not available in the AIToolbox, the users can easily implement their own new metrics. The approach is very similar to that of the new result package development.

In order to implement a new result metric, the user has to create a new metric class which inherits from the base abstract result metric *aitoolbox.experiment.core_metrics.abstract_metric.AbstractBaseMetric* and implements the abstract method *aitoolbox.experiment.core_metrics.abstract_metric.AbstractBaseMetric.calculate_metric()*.

As part of the `calculate_metric()` the user has to implement the logic for the performance metric calculation and return the metric result from the method. Predicted values and ground truth values normally needed for the performance metric calculations are available inside the metric as object attributes and can thus be accessed as: `self.y_true` and `self.y_predicted` throughout the metric class, `calculate_metric()` included.

Example Result Metric implementation:

```
from sklearn.metrics import accuracy_score
from aitoolbox.experiment.core_metrics.abstract_metric import AbstractBaseMetric


class ExampleAccuracyMetric(AbstractBaseMetric):
    def __init__(self, y_true, y_predicted, positive_class_thresh=0.5):
        # All additional attributes should be defined before the AbstractBaseMetric.__
↪init__
        self.positive_class_thresh = positive_class_thresh
        AbstractBaseMetric.__init__(self, y_true, y_predicted, metric_name='Accuracy')

    def calculate_metric(self):
        if self.positive_class_thresh is not None:
            self.y_predicted = self.y_predicted >= self.positive_class_thresh

        return accuracy_score(self.y_true, self.y_predicted)
```

2.3 Experiment Saving

One of the main uses of *aitoolbox.experiment* package is tracking experiments and saving models and result as the training progresses. This way the executed is well documented and its parameters and details can easily be determined even a long time after the original training.

The components in this package basically help prevent training the model, then doing some other experiments and coming back to the original experiment one month later and having no clue what was actually the experiment setting, what was the performance and how exactly were the results produced.

2.3.1 Experiment Saver

AIToolbox experiment savers at their core handle the creation of the experiment folder into which all the experiment results and model are automatically saved in a structured way which helps the experiment traceability. Experiment savers represent the high-level experiment tracking API and generally fall into two main categories: cloud experiment savers and local-only experiment savers.

Local-only experiment savers in *aitoolbox.experiment.local_experiment_saver* are simpler and only save the the experiment results onto the local drive. Cloud-enabled experiment savers in *aitoolbox.experiment.experiment_saver* are an extension in sense that they in addition to tracking the experiment on the local drive also automatically take care of uploading all the produced results and models to the cloud storage. This is especially useful when shutting automatically down the GPU instance after the training is finished. By using cloud experiment saver, all the experiment results are safely and automatically persisted in the cloud storage even after all the locally produced results are deleted when the instance is terminated.

Cloud enabled experiment savers:

- *aitoolbox.experiment.experiment_saver.FullPyTorchExperimentS3Saver*
- *aitoolbox.experiment.experiment_saver.FullKerasExperimentS3Saver*
- *aitoolbox.experiment.experiment_saver.FullPyTorchExperimentGoogleStorageSaver*
- *aitoolbox.experiment.experiment_saver.FullKerasExperimentGoogleStorageSaver*

Local-only experiment savers:

- *aitoolbox.experiment.local_experiment_saver.FullPyTorchExperimentLocalSaver*
- *aitoolbox.experiment.local_experiment_saver.FullKerasExperimentLocalSaver*

A very convenient property all the experiment savers have is that they all implement the same user facing API which makes them ideal for easy use as part of the larger system. Due to the unified API different experiment saver types can be easily dynamically exchanged according to desired training scenarios without any need to modify the surrounding code. The core API function that is common to all the experiment savers used to initiate the experiment snapshot saving is *aitoolbox.experiment.experiment_saver.AbstractExperimentSaver.save_experiment()*.

2.3.2 Local Save

While experiment savers described above serve as the high-level experiment tracking API, the local model and results savers are the low-level components on top of which the experiment saver API is built. Most users will probably very often just use the experiment savers, however in certain use cases the use of more low level components could still be desired.

The local experiment data saving low-level components can be found in the *aitoolbox.experiment.local_save* subpackage. They handle all the experiment tracking tasks ranging from experiment folder structuring, neural model weights & optimizer state saving and all the way to tracked results packaging before finally saving to the local drive.

2.3.2.1 Local Model Save

Implementations of model saving logic to the local drive. Currently available model savers:

- *aitoolbox.experiment.local_save.local_model_save.PyTorchLocalModelSaver*
- *aitoolbox.experiment.local_save.local_model_save.KerasLocalModelSaver*

2.3.2.2 Local Results Save

Implementation of training results saving logic to the local drive available in *aitoolbox.experiment.local_save.local_results_save.LocalResultsSaver*. This class offers two main options to save produced experiment results:

- saving all results into the single (potentially large) file via *aitoolbox.experiment.local_save.local_results_save.LocalResultsSaver.save_experiment_results*
- saving results into the single multiple separate files via *aitoolbox.experiment.local_save.local_results_save.LocalResultsSaver.save_experiment_results_separate_files*

2.4 Training History

aitoolbox.experiment.training_history.TrainingHistory is the util class which helps the user with tracking multiple performance metrics results during the model training process.

Under the hood it is just a simple Python dict and it thus supports many of the standard dict operations and operators for easier use. However the `TrainingHistory` also offers a convenient API on top of the dict geared towards experiment performance tracking. It is mostly used under the hood of the torchtrain TrainLoops, but it can be easily also utilized as a standalone results tracker for any kind of ML experiments.

CHAPTER

THREE

CLOUD

aitoolbox.cloud implements the components for communication and management of different cloud based services. Most of the functionality is available both for AWS and Google Cloud.

At its core, the package implements data saving and data downloading from the cloud storage. On top of this there are high level APIs available for conveniently downloading datasets and saving/loading models from the cloud data storage such as AWS S3.

3.1 Saving to Cloud

One of the most important aspects of the *aitoolbox.cloud* package is saving of data to the cloud storage.

The data saving components for *AWS S3* are available in:

- *aitoolbox.cloud.AWS.model_save*
- *aitoolbox.cloud.AWS.results_save*

The data saving components for *Google Cloud Storage* are available in:

- *aitoolbox.cloud.GoogleCloud.model_save*
- *aitoolbox.cloud.GoogleCloud.results_save*

The implementations found here provide an easy to use API to upload the saved models and experiment tracking results to the cloud storage.

3.1.1 Model Saving

`model_save` modules provide an API to which the user provides the model they wish to save and the module will automatically first locally save the model in the easy to track folder structure and then upload it to the selected cloud storage. Cloud experiment folder structure mirrors that which is created on the local drive. Currently supported cloud model savers can save PyTorch and Keras models to *AWS S3* or *Google Cloud Storage*.

PyTorch cloud model savers:

- *aitoolbox.cloud.AWS.model_save.PyTorchS3ModelSaver*
- *aitoolbox.cloud.GoogleCloud.model_save.PyTorchGoogleStorageModelSaver*

Keras cloud model savers:

- *aitoolbox.cloud.AWS.model_save.KerasS3ModelSaver*
- *aitoolbox.cloud.GoogleCloud.model_save.KerasGoogleStorageModelSaver*

3.1.2 Results Saving

`results_save` modules enables the user to save performance results to cloud as part of the training experiment tracking. Similarly to the cloud model saving, the cloud results savers first save the training results locally and then automatically uploads them to the selected cloud storage. Currently, cloud training results saving is supported for *AWS S3* and *Google Cloud Storage*.

Cloud results savers:

- *aitoolbox.cloud.AWS.results_save.S3ResultsSaver*
- *aitoolbox.cloud.GoogleCloud.results_save.GoogleStorageResultsSaver*

3.2 Loading Models from Cloud

aitoolbox.cloud.AWS.model_load and *aitoolbox.cloud.GoogleCloud.model_load* modules provide logic to conveniently download the previously saved model checkpoint from the cloud storage and initialize local model weight. This in effect automatically jump-starts the local model from the saved checkpoint and makes it ready for inference use or further training.

Currently available cloud model loading is for the PyTorch and supports *AWS S3* and *Google Cloud Storage*:

- *aitoolbox.cloud.AWS.model_load.PyTorchS3ModelLoader*
- *aitoolbox.cloud.GoogleCloud.model_load.PyTorchGoogleStorageModelLoader*

To load the model from cloud, first, the user needs to initialize the model object and then give it to the cloud model loader. Model loader will download the saved model from the cloud and use its weights to initialize the provided local model. Furthermore the model loader can also initialize the optimizer and the AMP in case the local model is not going to be used just for inference but for further model training.

3.3 Data Access

aitoolbox.cloud.AWS.data_access and *aitoolbox.cloud.GoogleCloud.data_access* implement low-level APIs to download and upload data to the *AWS S3* and *Google Cloud Storage*.

For *AWS S3* data uploading and downloading use:

- *aitoolbox.cloud.AWS.data_access.BaseDataSaver*
- *aitoolbox.cloud.AWS.data_access.BaseDataLoader*

For *Google Cloud Storage* data uploading and downloading use:

- *aitoolbox.cloud.GoogleCloud.data_access.BaseGoogleStorageDataSaver*
- *aitoolbox.cloud.GoogleCloud.data_access.BaseGoogleStorageDataLoader*

3.4 AWS Simple Email Service

AWS has an email sending service (*Simple Email Service*) which can be used to programmatically send emails from your python code. Under the hood of *torchtrain* this is used to send training progress notifications. However, the email sending component can also be used independently.

Email sending component build on top of Simple Email Service is implemented in *aitoolbox.cloud.AWS.simple_email_service.SESSender*.

To send the email, the user has to provide source and target email address and then specify email's subject and body. In the case of body content, to achieve a more advanced formatting, the body text should be provided as the HTML document. Additionally, attachment files can be also sent in the email by proving the list of attachment file paths.

CHAPTER

FOUR

NLP

Implements a mix of NLP data preprocessing components and various NLP performance metrics wrapped into the Result Packages which can be then used as part of TrainLoop training. These components can be found as part of the following `nlp` subpackages:

- *aitoolbox.nlp.experiment_evaluation*
- *aitoolbox.nlp.core*
- *aitoolbox.nlp.dataset*

CHAPTER

FIVE

EXAMPLES

AIToolbox provides several example training scripts to further support better understanding of the package functionality in addition to this documentation. All the examples are stored on github in the /examples folder.

Especially interesting examples worth looking at are:

- Model training with full experiment tracking
- Simple TTModel definition example & training
- DataParallel and DistributedDataParallel TrainLoop training examples
- Apex AMP (mixed precision) TrainLoop training examples

CHAPTER

SIX

AITOOLBOX

6.1 Subpackages

6.1.1 torchtrain

6.1.1.1 Subpackages

6.1.1.1.1 callbacks

Submodules

abstract

class aitoolbox.torchtrain.callbacks.abstract.**AbstractCallback**(*callback_name*, *execution_order=0*, *device_idx_execution=None*)

Bases: `object`

Abstract callback class that all actual callback classes have to inherit from

In the inherited callback classes the callback methods should be overwritten and used to implement desired callback functionality at specific points of the train loop.

Parameters

- **callback_name** (*str*) – name of the callback
- **execution_order** (*int*) – order of the callback execution. If all the used callbacks have the orders set to 0, than the callbacks are executed in the order they were registered.
- **device_idx_execution** (*int or None*) – index of the (CUDA GPU) device DDP process inside which the callback should be executed

register_train_loop_object(*train_loop_obj*)

Introduce the reference to the encapsulating trainloop so that the callback has access to the low level functionality of the trainloop

The registration is normally handled by the callback handler found inside the train loops. The handler is responsible for all the callback orchestration of the callbacks inside the trainloops.

Parameters **train_loop_obj** (*aitoolbox.torchtrain.train_loop.TrainLoop*) – reference to the encapsulating trainloop

Returns return the reference to the callback after it is registered

Return type *AbstractCallback*

on_train_loop_registration()
Execute callback initialization / preparation after the train_loop_object becomes available

Returns None

on_epoch_begin()
Logic executed at the beginning of the epoch

Returns None

on_epoch_end()
Logic executed at the end of the epoch

Returns None

on_train_begin()
Logic executed at the beginning of the overall training

Returns None

on_train_end()
Logic executed at the end of the overall training

Returns None

on_batch_begin()
Logic executed before the batch is inserted into the model

Returns None

on_batch_end()
Logic executed after the batch is inserted into the model

Returns None

on_after_gradient_update(*optimizer_idx*)
Logic executed after the model gradients are updated

To ensure the execution of this callback enable the *self.train_loop_obj.grad_cb_used = True* option in the on_train_loop_registration(). Otherwise logic implemented here will not be executed by the TrainLoop.

Parameters **optimizer_idx** (*int*) – index of the current optimizer. Mostly useful when using multiple optimizers. When only a single optimizer is used this parameter can be ignored.

Returns None

on_after_optimizer_step()
Logic executed after the optimizer does a new step and updates the model weights

To ensure the execution of this callback enable the *self.train_loop_obj.grad_cb_used = True* option in the on_train_loop_registration(). Otherwise logic implemented here will not be executed by the TrainLoop.

Returns None

on_multiprocess_start()
Logic executed after a new multiprocessing process is spawned at the beginning of every child process

Returns None

class aitoolbox.torchtrain.callbacks.abstract.**AbstractExperimentCallback**(*callback_name*, *project_name=None*, *experiment_name=None*, *local_model_result_folder_path=None*, *cloud_save_mode=None*, *bucket_name=None*, *cloud_dir_prefix=None*, *execution_order=0*, *device_idx_execution=None*)

Bases: *aitoolbox.torchtrain.callbacks.abstract.AbstractCallback*

Extension of the AbstractCallback implementing the automatic experiment details inference from TrainLoop

This abstract callback is inherited from when the implemented callbacks intend to save results files into the experiment folder and also potentially upload them to AWS S3.

Parameters

- **callback_name** (*str*) – name of the callback
- **project_name** (*str or None*) – root name of the project
- **experiment_name** (*str or None*) – name of the particular experiment
- **local_model_result_folder_path** (*str or None*) – root local path where project folder will be created
- **cloud_save_mode** (*str or None*) – Storage destination selector. For AWS S3: 's3' / 'aws_s3' / 'aws' For Google Cloud Storage: 'gcs' / 'google_storage' / 'google storage' Everything else results just in local storage to disk
- **bucket_name** (*str*) – name of the bucket in the cloud storage
- **cloud_dir_prefix** (*str*) – path to the folder inside the bucket where the experiments are going to be saved
- **execution_order** (*int*) – order of the callback execution. If all the used callbacks have the orders set to 0, than the callbacks are executed in the order they were registered.
- **device_idx_execution** (*int or None*) – index of the (CUDA GPU) device DDP process inside which the callback should be executed

try_infer_experiment_details(*infer_cloud_details*)

Infer paths where to save experiment related files from the running TrainLoop.

This details inference function should only be called after the callback has already been registered in the TrainLoop, e.g. in the on_train_loop_registration().

General rule:

take details from the TrainLoop -> for this option where experiment details are inferred from TrainLoop
all of the cloud_save_mode, bucket_name and cloud_dir_prefix should be set to None

Based on *self.cloud_save_mode* the inference decision is made as follows:

- ['s3', 'aws_s3', 'aws'] -> AWS S3
- ['gcs', 'google_storage', 'google storage'] -> Google Cloud Storage
- 'local' or whatever value -> local only

Parameters **infer_cloud_details** (*bool*) – should infer only local project folder details or also cloud project destination

Raises **AttributeError** –

Returns None

basic

class aitoolbox.torchtrain.callbacks.basic.**ListRegisteredCallbacks**

Bases: *aitoolbox.torchtrain.callbacks.abstract.AbstractCallback*

List all the callbacks which are used in the current TrainLoop

on_train_begin()

Logic executed at the beginning of the overall training

Returns None

class aitoolbox.torchtrain.callbacks.basic.**EarlyStopping**(*monitor='val_loss'*, *min_delta=0.0*, *patience=0*)

Bases: *aitoolbox.torchtrain.callbacks.abstract.AbstractCallback*

Early stopping of the training if the performance stops improving

Parameters

- **monitor** (*str*) – performance measure that is tracked to decide if performance is improving during training
- **min_delta** (*float*) – by how much the performance has to improve to still keep training the model
- **patience** (*int*) – how many epochs the early stopper waits after the performance stopped improving

on_epoch_end()

Logic executed at the end of the epoch

Returns None

class aitoolbox.torchtrain.callbacks.basic.**ThresholdEarlyStopping**(*monitor*, *threshold*, *patience=0*)

Bases: *aitoolbox.torchtrain.callbacks.abstract.AbstractCallback*

Early stopping of the training if the performance doesn't reach the specified threshold

Parameters

- **monitor** (*str*) – performance measure that is tracked to decide if performance reached the desired threshold
- **threshold** (*float*) – performance threshold that needs to be exceeded in order to continue training

- **patience** (*int*) – how many epochs the early stopper waits for the tracked performance to reach the desired threshold

on_epoch_end()
Logic executed at the end of the epoch

Returns None

class aitoolbox.torchtrain.callbacks.basic.**TerminateOnNaN**(*monitor='loss'*)
Bases: *aitoolbox.torchtrain.callbacks.abstract.AbstractCallback*

Terminate training if NaNs are predicted, thus metrics are NaN

Parameters **monitor** (*str*) – performance measure that is tracked to decide if performance is improving during training

on_epoch_end()
Logic executed at the end of the epoch

Returns None

class aitoolbox.torchtrain.callbacks.basic.**AllPredictionsSame**(*value=0.0*, *stop_training=False*, *verbose=True*)
Bases: *aitoolbox.torchtrain.callbacks.abstract.AbstractCallback*

Checks if all the predicted values are the same

Useful for example when dealing with extremely unbalanced classes.

Parameters

- **value** (*float*) – all predictions are the same as this value
- **stop_training** (*bool*) – if all predictions match the specified value, should the training be (early) stopped
- **verbose** (*bool*) – output messages

on_epoch_end()
Logic executed at the end of the epoch

Returns None

class aitoolbox.torchtrain.callbacks.basic.**EmailNotification**(*sender_name*, *sender_email*, *recipient_email*, *project_name=None*, *experiment_name=None*, *aws_region='eu-west-1'*)
Bases: *aitoolbox.torchtrain.callbacks.abstract.AbstractCallback*

Notify user via email about the training progression

Parameters

- **sender_name** (*str*) – Name of the email sender
- **sender_email** (*str*) – Email of the email sender
- **recipient_email** (*str*) – Email where the email will be sent
- **project_name** (*str or None*) – root name of the project

- **experiment_name** (*str or None*) – name of the particular experiment
- **aws_region** (*str*) – AWS SES region

on_epoch_end()
Logic executed at the end of the epoch

Returns None

on_train_end()
Logic executed at the end of the overall training

Returns None

get_metric_list_html()
Generate performance metrics list HTML

Returns HTML doc

Return type str

get_hyperparams_html()
Generate hyperparameters list HTML

Returns HTML doc

Return type str

on_train_loop_registration()
Execute callback initialization / preparation after the train_loop_object becomes available

Returns None

class aitoolbox.torchtrain.callbacks.basic.**LogUpload**(*log_file_path='~/project/training.log'*, *fail_if_cloud_missing=True*, *project_name=None*, *experiment_name=None*, *local_model_result_folder_path=None*, *cloud_save_mode=None*, *bucket_name=None*, *cloud_dir_prefix=None*)
Bases: *aitoolbox.torchtrain.callbacks.abstract.AbstractExperimentCallback*

Upload logging file to the cloud storage

Uploading happens after each epoch and at the end of the training process.

Parameters

- **log_file_path** (*str*) – path to the local logging file
- **fail_if_cloud_missing** (*bool*) – should throw the exception if cloud saving is not available
- **project_name** (*str or None*) – root name of the project
- **experiment_name** (*str or None*) – name of the particular experiment
- **local_model_result_folder_path** (*str or None*) – root local path where project folder will be created
- **cloud_save_mode** (*str or None*) – Storage destination selector. For AWS S3: 's3' / 'aws_s3' / 'aws' For Google Cloud Storage: 'gcs' / 'google_storage' / 'google storage' Everything else results just in local storage to disk
- **bucket_name** (*str*) – name of the bucket in the cloud storage

- **cloud_dir_prefix** (*str*) – path to the folder inside the bucket where the experiments are going to be saved

on_train_loop_registration()
Execute callback initialization / preparation after the train_loop_object becomes available

Returns None

on_epoch_end()
Logic executed at the end of the epoch

Returns None

on_train_end()
Logic executed at the end of the overall training

Returns None

upload_log_file()

class aitoolbox.torchtrain.callbacks.basic.**DataSubsetTestRun**(*num_train_batches=1*, *num_val_batches=0*, *num_test_batches=0*)
Bases: *aitoolbox.torchtrain.callbacks.abstract.AbstractCallback*

Subset the provided data loaders to execute neural net only on a small dataset subset

This is especially useful when first developing the neural architectures and debugging them. Subsetting the full dataset helps with fast development iterations.

Parameters

- **num_train_batches** (*int*) – number of the training data batches that are kept in the training dataset
- **num_val_batches** (*int*) – number of the validation data batches that are kept in the validation dataset
- **num_test_batches** (*int*) – number of the test data batches that are kept in the test dataset

on_train_begin()
Logic executed at the beginning of the overall training

Returns None

static subset_data_loader(*data_loader*, *num_batches*)

on_train_loop_registration()
Execute callback initialization / preparation after the train_loop_object becomes available

Returns None

class aitoolbox.torchtrain.callbacks.basic.**FunctionOnTrainLoop**(*fn_to_execute*, *tl_registration=False*, *epoch_begin=False*, *epoch_end=False*, *train_begin=False*, *train_end=False*, *batch_begin=False*, *batch_end=False*, *after_gradient_update=False*, *after_optimizer_step=False*, *execution_order=0*, *device_idx_execution=None*)

Bases: *aitoolbox.torchtrain.callbacks.abstract.AbstractCallback*

Execute given function as a callback in the TrainLoop

Parameters

- **fn_to_execute** (*function*) – function logic to be executed at the desired point of the TrainLoop. The function should take a single input as an argument which is the reference to the encapsulating TrainLoop object (self.train_loop_obj).
- **tl_registration** (*bool*) – should execute on TrainLoop registration
- **epoch_begin** (*bool*) – should execute at the beginning of the epoch
- **epoch_end** (*bool*) – should execute at the end of the epoch
- **train_begin** (*bool*) – should execute at the beginning of the training
- **train_end** (*bool*) – should execute at the end of the training
- **batch_begin** (*bool*) – should execute at the beginning of the batch
- **batch_end** (*bool*) – should execute at the end of the batch
- **after_gradient_update** (*bool*) – should execute after the gradient update
- **after_optimizer_step** (*bool*) – should execute after the optimizer step
- **execution_order** (*int*) – order of the callback execution. If all the used callbacks have the orders set to 0, than the callbacks are executed in the order they were registered.

execute_callback()

on_train_loop_registration()

Execute callback initialization / preparation after the train_loop_object becomes available

Returns None

on_epoch_begin()

Logic executed at the beginning of the epoch

Returns None

on_epoch_end()

Logic executed at the end of the epoch

Returns None

on_train_begin()
Logic executed at the beginning of the overall training

Returns None

on_train_end()
Logic executed at the end of the overall training

Returns None

on_batch_begin()
Logic executed before the batch is inserted into the model

Returns None

on_batch_end()
Logic executed after the batch is inserted into the model

Returns None

on_after_gradient_update(*optimizer_idx*)
Logic executed after the model gradients are updated

To ensure the execution of this callback enable the *self.train_loop_obj.grad_cb_used* = *True* option in the on_train_loop_registration(). Otherwise logic implemented here will not be executed by the TrainLoop.

Parameters **optimizer_idx** (*int*) – index of the current optimizer. Mostly useful when using multiple optimizers. When only a single optimizer is used this parameter can be ignored.

Returns None

on_after_optimizer_step()
Logic executed after the optimizer does a new step and updates the model weights

To ensure the execution of this callback enable the *self.train_loop_obj.grad_cb_used* = *True* option in the on_train_loop_registration(). Otherwise logic implemented here will not be executed by the TrainLoop.

Returns None

ddp

class aitoolbox.torchtrain.callbacks.ddp.**DistributedSamplerSetEpoch**(*train_sampler*, *validation_sampler*, *test_sampler*)

Bases: *aitoolbox.torchtrain.callbacks.abstract.AbstractCallback*

Callback setting epoch index in the DistributedSamplers at the beginning of every epoch

Parameters

- **train_sampler** (*torch.utils.data.distributed.DistributedSampler or None*) – Distributed sampler for train loader
- **validation_sampler** (*torch.utils.data.distributed.DistributedSampler or None*) – Distributed sampler for validation loader
- **test_sampler** (*torch.utils.data.distributed.DistributedSampler or None*) – Distributed sampler for test loader

on_epoch_begin()
Logic executed at the beginning of the epoch

Returns None

class aitoolbox.torchtrain.callbacks.ddp.**InMultiProcessDataLoad**(*train_loader_build_fn=None*, *val_loader_build_fn=None*, *test_loader_build_fn=None*)

Bases: *aitoolbox.torchtrain.callbacks.abstract.AbstractCallback*

Multiprocess in-process data loading logic infuser

Parameters

- **train_loader_build_fn** (*callable or bool or None*) – function specifying the training data reading and train data loader preparation which should be returned from the function. If not provided, the original train data loader in TrainLoop will be kept.
- **val_loader_build_fn** (*callable or bool or None*) – function specifying the validation data reading and validation data loader preparation which should be returned from the function. If not provided, the original validation data loader in TrainLoop will be kept.
- **test_loader_build_fn** (*callable or bool or None*) – function specifying the test data reading and test data loader preparation which should be returned from the function. If not provided, the original test data loader in TrainLoop will be kept.

on_multiprocess_start()

Logic executed after a new multiprocessing process is spawned at the beginning of every child process

Returns None

build_train_dataloader()

build_val_dataloader()

build_test_dataloader()

gradient

class aitoolbox.torchtrain.callbacks.gradient.**GradientCallbackBase**(*callback_name*, *execution_order=0*)

Bases: *aitoolbox.torchtrain.callbacks.abstract.AbstractCallback*

Base abstract class for gradient related callbacks

It has not implemented logic except for the the turning enabling of the grad_cb_used inside TrainLoop as part of the on_train_loop_registration(). Consequently, this potentially repeated task in every gradient calculation callback doesn't need to be done for every implemented callback.

Parameters

- **callback_name** (*str*) – name of the callback
- **execution_order** (*int*) – order of the callback execution. If all the used callbacks have the orders set to 0, than the callbacks are executed in the order they were registered.

on_train_loop_registration()

Execute callback initialization / preparation after the train_loop_object becomes available

Returns None

class aitoolbox.torchtrain.callbacks.gradient.**GradValueClip**(*max_grad_value*)

Bases: *aitoolbox.torchtrain.callbacks.gradient.GradientCallbackBase*

Gradient value clipping

Parameters **max_grad_value** (*int or float*) – maximum allowed value of the gradients

on_after_gradient_update(*optimizer_idx*)
Logic executed after the model gradients are updated

To ensure the execution of this callback enable the *self.train_loop_obj.grad_cb_used = True* option in the on_train_loop_registration(). Otherwise logic implemented here will not be executed by the TrainLoop.

Parameters **optimizer_idx** (*int*) – index of the current optimizer. Mostly useful when using multiple optimizers. When only a single optimizer is used this parameter can be ignored.

Returns None

class aitoolbox.torchtrain.callbacks.gradient.**GradNormClip**(*max_grad_norm*, ***kwargs*)

Bases: *aitoolbox.torchtrain.callbacks.gradient.GradientCallbackBase*

Gradient norm clipping

Parameters

- **max_grad_norm** (*int or float*) – max norm of the gradients
- ****kwargs** – **torch.nn.utils.clip_grad_norm_** additional arguemnts

on_after_gradient_update(*optimizer_idx*)
Logic executed after the model gradients are updated

To ensure the execution of this callback enable the *self.train_loop_obj.grad_cb_used = True* option in the on_train_loop_registration(). Otherwise logic implemented here will not be executed by the TrainLoop.

Parameters **optimizer_idx** (*int*) – index of the current optimizer. Mostly useful when using multiple optimizers. When only a single optimizer is used this parameter can be ignored.

Returns None

class aitoolbox.torchtrain.callbacks.gradient.**GradientStatsPrint**(*model_layers_extract_def*, *on_every_grad_update=False*)

Bases: *aitoolbox.torchtrain.callbacks.abstract.AbstractCallback*

Model gradients statistics reporting

Parameters

- **model_layers_extract_def** (*lambda or function*) – lambda/function accepting model as the input and returning a list of all the layers in the model for which the gradient stats should be calculated
- **on_every_grad_update** (*bool*) – should the gradient stats be calculated on every gradient update, e.g. after every batch or only at the end of the epoch

on_train_loop_registration()
Execute callback initialization / preparation after the train_loop_object becomes available

Returns None

on_after_gradient_update(*optimizer_idx*)
Logic executed after the model gradients are updated

To ensure the execution of this callback enable the *self.train_loop_obj.grad_cb_used = True* option in the on_train_loop_registration(). Otherwise logic implemented here will not be executed by the TrainLoop.

Parameters **optimizer_idx** (*int*) – index of the current optimizer. Mostly useful when using multiple optimizers. When only a single optimizer is used this parameter can be ignored.

Returns None

on_epoch_end()
Logic executed at the end of the epoch

Returns None

gradients_report()

class aitoolbox.torchtrain.callbacks.gradient.**GradDistributionPlot**(*model_layers_extract_def*, *grad_plots_dir_name='grad_distri*, *file_format='png'*, *project_name=None*, *experiment_name=None*, *local_model_result_folder_path=No*, *cloud_save_mode=None*, *bucket_name=None*, *cloud_dir_prefix=None*)

Bases: *aitoolbox.torchtrain.callbacks.abstract.AbstractExperimentCallback*

Plot layers' gradient distributions after every epoch

Parameters

- **model_layers_extract_def** (*lambda or function*) – lambda/function accepting model as the input and returning a list of all the layers in the model for which the gradient stats should be calculated
- **grad_plots_dir_name** (*str*) – name of the folder where gradient distribution plots are saved after every epoch
- **file_format** (*str*) – output file format. Can be either 'png' for saving separate images or 'pdf' for combining all the plots into a single pdf file.
- **project_name** (*str or None*) – root name of the project
- **experiment_name** (*str or None*) – name of the particular experiment
- **local_model_result_folder_path** (*str or None*) – root local path where project folder will be created
- **cloud_save_mode** (*str or None*) – Storage destination selector. For AWS S3: 's3' / 'aws_s3' / 'aws' For Google Cloud Storage: 'gcs' / 'google_storage' / 'google storage' Everything else results just in local storage to disk
- **bucket_name** (*str*) – name of the bucket in the cloud storage
- **cloud_dir_prefix** (*str*) – path to the folder inside the bucket where the experiments are going to be saved

on_train_loop_registration()
Execute callback initialization / preparation after the train_loop_object becomes available

Returns None

on_epoch_end()
Logic executed at the end of the epoch

Returns None

gradient_plot()

save_to_cloud(*saved_plot_paths*)

create_plot_dirs()

prepare_results_saver()

model_load

class aitoolbox.torchtrain.callbacks.model_load.**ModelLoadContinueTraining**(*saved_experiment_times saved_model_dir='check epoch_num=None, ig- nore_saved_schedulers= ig- nore_missing_saved_sch used_data_parallel=Fal cus- tom_local_loader_class= project_name=None, ex- per- i- ment_name=None, lo- cal_model_result_folder cloud_save_mode=None bucket_name=None, cloud_dir_prefix=None, **kwargs*)

Bases: *aitoolbox.torchtrain.callbacks.abstract.AbstractExperimentCallback*

(Down)load previously trained and saved model and continue training from this snapshot instead from beginning

Parameters

- **saved_experiment_timestamp** (*str*) – timestamp of the saved model experiment
- **saved_model_dir** (*str*) – folder where saved model file is inside main experiment folder
- **epoch_num** (*int or None*) – if loading checkpoint model instead of final model this parameter indicates from which epoch of training the model will be loaded
- **ignore_saved_schedulers** (*bool*) – if exception should be raised in the case there are found scheduler snapshots in the checkpoint, but not schedulers are provided to this method
- **ignore_missing_saved_schedulers** (*bool*) – if exception should be raised in the case schedulers are provided to this method but no saved scheduler snapshots can be found in the checkpoint
- **used_data_parallel** (*bool*) – if the saved model was nn.DataParallel or normal model
- **custom_local_loader_class** (*AbstractLocalModelLoader class or None*) – provide a custom local PyTorch model loader definition in case the default one is not suitable for particular use case. For example, in the case of complex custom optimizer initialization.
- **project_name** (*str or None*) – root name of the project
- **experiment_name** (*str or None*) – name of the particular experiment

- **local_model_result_folder_path** (*str or None*) – root local path where project folder will be created
- **cloud_save_mode** (*str or None*) – Storage destination selector. For AWS S3: 's3' / 'aws_s3' / 'aws' For Google Cloud Storage: 'gcs' / 'google_storage' / 'google storage' Everything else results just in local storage to disk
- **bucket_name** (*str*) – name of the bucket in the cloud storage
- **cloud_dir_prefix** (*str*) – path to the folder inside the bucket where the experiments are going to be saved
- ****kwargs** – additional parameters for the local model loader load_model() function

on_train_loop_registration()
Execute callback initialization / preparation after the train_loop_object becomes available

Returns None

on_train_begin()
Logic executed at the beginning of the overall training

Returns None

init_model_loader()
Initialize model loader object based on provided arguments to the callback object

Returns None

model_save

class aitoolbox.torchtrain.callbacks.model_save.**ModelCheckpoint**(*project_name*, *experiment_name*, *local_model_result_folder_path*, *hyperparams*, *cloud_save_mode='s3'*, *bucket_name='model-result'*, *cloud_dir_prefix=''*, *rm_subopt_local_models=False*, *num_best_checkpoints_kept=2*)

Bases: *aitoolbox.torchtrain.callbacks.abstract.AbstractCallback*

Check-point save the model during training to disk or also to S3 / GCS cloud storage

Parameters

- **project_name** (*str*) – root name of the project
- **experiment_name** (*str*) – name of the particular experiment
- **local_model_result_folder_path** (*str*) – root local path where project folder will be created
- **hyperparams** (*dict*) – used hyper-parameters. When running the TrainLoop from jupyter notebook in order to ensure the python experiment file copying to the experiment folder, the user needs to manually specify the python file path as the value for the *experiment_file_path* key. If running the training directly from the terminal the path deduction is done automatically.

- **cloud_save_mode** (*str or None*) – Storage destination selector. For AWS S3: 's3' / 'aws_s3' / 'aws' For Google Cloud Storage: 'gcs' / 'google_storage' / 'google storage' Everything else results just in local storage to disk
- **bucket_name** (*str*) – name of the bucket in the cloud storage
- **cloud_dir_prefix** (*str*) – path to the folder inside the bucket where the experiments are going to be saved
- **rm_subopt_local_models** (*bool or str*) – if True, the deciding metric is set to 'loss'. Give string metric name to set it as a deciding metric for suboptimal model removal. If metric name consists of substring 'loss' the metric minimization is done otherwise metric maximization is done
- **num_best_checkpoints_kept** (*int*) – number of best performing models which are kept when removing suboptimal model checkpoints

on_epoch_end()
Logic executed at the end of the epoch

Returns None

on_train_loop_registration()
Execute callback initialization / preparation after the train_loop_object becomes available

Returns None

save_hyperparams()

class aitoolbox.torchtrain.callbacks.model_save.**ModelIterationCheckpoint**(*save_frequency*, *project_name*, *experiment_name*, *local_model_result_folder_path*, *hyperparams*, *cloud_save_mode='s3'*, *bucket_name='model-result'*, *cloud_dir_prefix=''*, *rm_subopt_local_models=False*, *num_best_checkpoints_kept=0*)

Bases: *aitoolbox.torchtrain.callbacks.model_save.ModelCheckpoint*

Check-point save the model during training to disk or also to S3 / GCS cloud storage

Parameters

- **save_frequency** (*int*) – frequency of saving the model checkpoint every specified number of training iterations
- **project_name** (*str*) – root name of the project
- **experiment_name** (*str*) – name of the particular experiment
- **local_model_result_folder_path** (*str*) – root local path where project folder will be created

- **hyperparams** (*dict*) – used hyper-parameters. When running the TrainLoop from jupyter notebook in order to ensure the python experiment file copying to the experiment folder, the user needs to manually specify the python file path as the value for the *experiment_file_path* key. If running the training directly from the terminal the path deduction is done automatically.
- **cloud_save_mode** (*str or None*) – Storage destination selector. For AWS S3: 's3' / 'aws_s3' / 'aws' For Google Cloud Storage: 'gcs' / 'google_storage' / 'google storage' Everything else results just in local storage to disk
- **bucket_name** (*str*) – name of the bucket in the cloud storage
- **cloud_dir_prefix** (*str*) – path to the folder inside the bucket where the experiments are going to be saved
- **rm_subopt_local_models** (*bool or str*) – if True, the deciding metric is set to 'loss'. Give string metric name to set it as a deciding metric for suboptimal model removal. If metric name consists of substring 'loss' the metric minimization is done otherwise metric maximization is done
- **num_best_checkpoints_kept** (*int*) – number of best performing models which are kept when removing suboptimal model checkpoints

on_batch_end()

Logic executed after the batch is inserted into the model

Returns None

class aitoolbox.torchtrain.callbacks.model_save.**ModelTrainEndSave**(*project_name*, *experiment_name*, *local_model_result_folder_path*, *hyperparams*, *val_result_package=None*, *test_result_package=None*, *cloud_save_mode='s3'*, *bucket_name='model-result'*, *cloud_dir_prefix=''*)

Bases: *aitoolbox.torchtrain.callbacks.abstract.AbstractCallback*

At the end of training execute model performance evaluation, build result package report and save it together with the final model to local disk and possibly to S3 / GCS cloud storage

Parameters

- **project_name** (*str*) – root name of the project
- **experiment_name** (*str*) – name of the particular experiment
- **local_model_result_folder_path** (*str*) – root local path where project folder will be created
- **hyperparams** (*dict*) – used hyper-parameters. When running the TrainLoop from jupyter notebook in order to ensure the python experiment file copying to the experiment folder, the user needs to manually specify the python file path as the value for the *experiment_file_path* key. If running the training directly from the terminal the path deduction is done automatically.

- **val_result_package** (aitoolbox.experiment.result_package.abstract_result_packages.AbstractResultPackage) –
- **test_result_package** (aitoolbox.experiment.result_package.abstract_result_packages.AbstractResultPackage) –
- **cloud_save_mode** (*str or None*) – Storage destination selector. For AWS S3: 's3' / 'aws_s3' / 'aws' For Google Cloud Storage: 'gcs' / 'google_storage' / 'google storage' Everything else results just in local storage to disk
- **bucket_name** (*str*) – name of the bucket in the cloud storage
- **cloud_dir_prefix** (*str*) – path to the folder inside the bucket where the experiments are going to be saved

on_train_end()
Logic executed at the end of the overall training

Returns None

on_train_loop_registration()
Execute callback initialization / preparation after the train_loop_object becomes available

Returns None

save_hyperparams()

check_result_packages()

performance_eval

class aitoolbox.torchtrain.callbacks.performance_eval.**ModelPerformanceEvaluation**(*result_packag*
args,
on_each_epoc
on_train_data
on_val_data=
eval_frequenc
if_available_c

Bases: *aitoolbox.torchtrain.callbacks.abstract.AbstractCallback*

Track performance metrics from result_package and store them into TrainLoop's history

This callback is different from those for model and experiment saving where performance evaluations are also calculated. Here we only want to calculate performance and store it in memory into TrainLoop's history dict.

It is a more lightweight, on the go performance tracking without the need for the full project folder structure construction.

Parameters

- **result_package** (aitoolbox.experiment.result_package.abstract_result_packages.AbstractResultPackage) –
- **args** (*dict*) –
- **on_each_epoch** (*bool*) – calculate performance results just at the end of training or at the end of each epoch
- **on_train_data** (*bool*) –
- **on_val_data** (*bool*) –

- **eval_frequency** (*int or None*) – evaluation is done every specified number of epochs. Useful when predictions are quite expensive and are slowing down the overall training
- **if_available_output_to_project_dir** (*bool*) – if using train loop version which builds project local folder structure for saving checkpoints or creation of end of training reports, by setting if_available_output_to_project_dir to True the potential additional metadata result outputs from the result_package will be saved in the folder inside the main project folder. In this case the result_package's output folder shouldn't be full path but just the folder name and the full folder path pointing inside the corresponding project folder will be automatically created. If such a functionality should to be prevented and manual full additional metadata results dump folder is needed potentially outside the project folder, than set this argument to False and specify a full folder path.

on_train_end()
Logic executed at the end of the overall training

Returns None

on_epoch_end()
Logic executed at the end of the epoch

Returns None

evaluate_model_performance(*prefix=''*)
Calculate performance based on the provided result packages

Parameters **prefix** (*str*) – additional prefix for metric names that will get saved into the training history

Returns None

store_evaluated_metrics_to_history(*prefix=''*)
Save the calculated performance results into the training history

The performance results are saved into the training history after they are calculated by the before called evaluate_model_performance() function.

Parameters **prefix** (*str*) – additional prefix for metric names that will get saved into the training history

Returns None

on_train_loop_registration()
Execute callback initialization / preparation after the train_loop_object becomes available

Returns None

class aitoolbox.torchtrain.callbacks.performance_eval.**ModelPerformancePrintReport**(*metrics*, *on_each_epo*, *re-*
port_frequen, *strict_metric*, *list_tracked_*

Bases: *aitoolbox.torchtrain.callbacks.abstract.AbstractCallback*

Print the model performance to the console

Best used in combination with the callback which actually calculates some performance evaluation metrics, such as ModelPerformanceEvaluation. Otherwise we are limited only to automatic loss calculation reporting.

When listing callbacks for the TrainLoop it is important to list the ModelPerformanceEvaluation before this ModelPerformancePrintReport. This ensures that the calculated results are present in the TrainLoop.train_history before there is an attempt to print them.

Parameters

- **metrics** (*list*) – list of string metric names which should be presented in the printed report
- **on_each_epoch** (*bool*) – present results just at the end of training or at the end of each epoch
- **report_frequency** (*int or None*) – evaluation is done every specified number of epochs. Useful when predictions are quite expensive and are slowing down the overall training
- **strict_metric_reporting** (*bool*) – if False ignore missing metric in the TrainLoop.train_history, if True, in case of missing metric throw and exception and thus interrupt the training loop
- **list_tracked_metrics** (*bool*) – should all tracked metrics names be listed

on_train_end()
Logic executed at the end of the overall training

Returns None

on_epoch_end()
Logic executed at the end of the epoch

Returns None

print_performance_report(*prefix=''*)
Print the model performance

Parameters **prefix** (*str*) – additional prefix for metric names that will get saved into the training history

Returns None

class aitoolbox.torchtrain.callbacks.performance_eval.**TrainHistoryFormatter**(*input_metric_getter*, *output_metric_setter*, *epoch_end=True*, *train_end=False*, *strict_metric_extract=*

Bases: *aitoolbox.torchtrain.callbacks.abstract.AbstractCallback*

Format stored training history results

Parameters

- **input_metric_getter** (*lambda*) – extract full history for the desired metric, not just the last history input. Return should be represented as a list.
- **output_metric_setter** (*lambda*) – take the extracted full history of a metric and convert it as desired. Return new / transformed metric name and transformed metric result.
- **epoch_end** (*bool*) – should the formatting be executed at the end of the epoch
- **train_end** (*bool*) – should the formatting be executed at the end of the training process
- **strict_metric_extract** (*bool*) – in case of (quality) problems should exception be raised on just the notification printed to console

on_epoch_end()
Logic executed at the end of the epoch

Returns None

on_train_end()
Logic executed at the end of the overall training

Returns None

format_history()

check_if_history_updated(*train_end_phase*)

class aitoolbox.torchtrain.callbacks.performance_eval.**MetricHistoryRename**(*input_metric_path*, *new_metric_name*, *epoch_end=True*, *train_end=False*, *strict_metric_extract=Tr*

Bases: *aitoolbox.torchtrain.callbacks.performance_eval.TrainHistoryFormatter*

Specific interface for TrainHistoryFormatter which renames the metric in the training history

Parameters

- **input_metric_path** (*str or lambda*) – if using lambda, extract full history for the desired metric, not just the last history input. Return should be represented as a list.
- **new_metric_name** (*str*) – the new metric name
- **epoch_end** (*bool*) – should the formatting be executed at the end of the epoch
- **train_end** (*bool*) – should the formatting be executed at the end of the training process
- **strict_metric_extract** (*bool*) – in case of (quality) problems should exception be raised on just the notification printed to console

class aitoolbox.torchtrain.callbacks.performance_eval.**ModelTrainHistoryBaseCB**(*callback_name*, *execution_order=0*, *epoch_end=True*, *train_end=False*, *file_format=''*, *project_name=Non* *experiment_name=None* *local_model_result_* *cloud_save_mode=* *bucket_name=Non* *cloud_dir_prefix=*

Bases: *aitoolbox.torchtrain.callbacks.abstract.AbstractExperimentCallback*

Base callback class to be inherited from when reporting train performance history

Parameters

- **callback_name** (*str*) – name of the callback

- **execution_order** (*int*) – order of the callback execution. If all the used callbacks have the orders set to 0, than the callbacks are executed in the order they were registered.
- **epoch_end** (*bool*) – should plot after every epoch
- **train_end** (*bool*) – should plot at the end of the training
- **file_format** (*str*) – output file format
- **project_name** (*str or None*) – root name of the project
- **experiment_name** (*str or None*) – name of the particular experiment
- **local_model_result_folder_path** (*str or None*) – root local path where project folder will be created
- **cloud_save_mode** (*str or None*) – Storage destination selector. For AWS S3: 's3' / 'aws_s3' / 'aws' For Google Cloud Storage: 'gcs' / 'google_storage' / 'google storage' Everything else results just in local storage to disk
- **bucket_name** (*str*) – name of the bucket in the cloud storage
- **cloud_dir_prefix** (*str*) – path to the folder inside the bucket where the experiments are going to be saved

prepare_results_saver()
Initialize the required results saver

Returns None

class aitoolbox.torchtrain.callbacks.performance_eval.**ModelTrainHistoryPlot**(*epoch_end=True*, *train_end=False*, *file_format='png'*, *project_name=None*, *experiment_name=None*, *local_model_result_folder_path=None*, *cloud_save_mode=None*, *bucket_name=None*, *cloud_dir_prefix=None*)

Bases: *aitoolbox.torchtrain.callbacks.performance_eval.ModelTrainHistoryBaseCB*

Plot the evaluated performance metric history

Parameters

- **epoch_end** (*bool*) – should plot after every epoch
- **train_end** (*bool*) – should plot at the end of the training
- **file_format** (*str*) – output file format. Can be either 'png' for saving separate images or 'pdf' for combining all the plots into a single pdf file.
- **project_name** (*str or None*) – root name of the project
- **experiment_name** (*str or None*) – name of the particular experiment
- **local_model_result_folder_path** (*str or None*) – root local path where project folder will be created

- **cloud_save_mode** (*str or None*) – Storage destination selector. For AWS S3: 's3' / 'aws_s3' / 'aws' For Google Cloud Storage: 'gcs' / 'google_storage' / 'google storage' Everything else results just in local storage to disk
- **bucket_name** (*str*) – name of the bucket in the cloud storage
- **cloud_dir_prefix** (*str*) – path to the folder inside the bucket where the experiments are going to be saved

on_train_loop_registration()
Execute callback initialization / preparation after the train_loop_object becomes available

Returns None

on_epoch_end()
Logic executed at the end of the epoch

Returns None

on_train_end()
Logic executed at the end of the overall training

Returns None

plot_current_train_history(*prefix=''*)
Plot current training history snapshot in the encapsulating TrainLoop

Parameters **prefix** (*str*) – plots folder name prefix

Returns None

class aitoolbox.torchtrain.callbacks.performance_eval.**ModelTrainHistoryFileWriter**(*epoch_end=*
train_end=F
file_format=
project_nam
ex-
per-
i-
ment_name=
lo-
cal_model_r
*cloud_save_*
bucket_name
cloud_dir_p

Bases: *aitoolbox.torchtrain.callbacks.performance_eval.ModelTrainHistoryBaseCB*

Write evaluated performance metric history to the text file

Parameters

- **epoch_end** (*bool*) – should plot after every epoch
- **train_end** (*bool*) – should plot at the end of the training
- **file_format** (*str*) – output file format. Can be either 'txt' human readable output or 'tsv' for a tabular format or 'csv' for comma separated format.
- **project_name** (*str or None*) – root name of the project
- **experiment_name** (*str or None*) – name of the particular experiment
- **local_model_result_folder_path** (*str or None*) – root local path where project folder will be created

- **cloud_save_mode** (*str or None*) – Storage destination selector. For AWS S3: 's3' / 'aws_s3' / 'aws' For Google Cloud Storage: 'gcs' / 'google_storage' / 'google storage' Everything else results just in local storage to disk
- **bucket_name** (*str*) – name of the bucket in the cloud storage
- **cloud_dir_prefix** (*str*) – path to the folder inside the bucket where the experiments are going to be saved

on_train_loop_registration()
Execute callback initialization / preparation after the train_loop_object becomes available

Returns None

on_epoch_end()
Logic executed at the end of the epoch

Returns None

on_train_end()
Logic executed at the end of the overall training

Returns None

write_current_train_history(*prefix=''*)
Write to text file the current training history snapshot in the encapsulating TrainLoop

Parameters **prefix** (*str*) – history text file name prefix

Returns None

tensorboard

class aitoolbox.torchtrain.callbacks.tensorboard.**TensorboardReporterBaseCB**(*callback_name*, *log_dir=None*, *is_project=True*, *project_name=None*, *experiment_name=None*, *local_model_result_folde*
cloud_save_mode=Non
bucket_name=None, *cloud_dir_prefix=None*
***kwargs*)

Bases: *aitoolbox.torchtrain.callbacks.abstract.AbstractExperimentCallback*

Base Tensorboard callback wrapping SummaryWriter

This base callback is intended to be inherited and extended with the more concrete callback geared towards a particular use-case. This callback only setups all the folders needed for local and cloud experiment tracking.

Parameters

- **callback_name** (*str*) – name of the callback
- **log_dir** (*str or None*) – save directory location

- **is_project** (*bool*) – set to `True` if the results should be saved into the TrainLoop-created project folder structure or to `False` if you want to save into a specific full path given in the log_dir parameter.
- **project_name** (*str or None*) – root name of the project
- **experiment_name** (*str or None*) – name of the particular experiment
- **local_model_result_folder_path** (*str or None*) – root local path where project folder will be created
- **cloud_save_mode** (*str or None*) – Storage destination selector. For AWS S3: 's3' / 'aws_s3' / 'aws' For Google Cloud Storage: 'gcs' / 'google_storage' / 'google storage' Everything else results just in local storage to disk
- **bucket_name** (*str*) – name of the bucket in the cloud storage
- **cloud_dir_prefix** (*str*) – path to the folder inside the bucket where the experiments are going to be saved
- ****kwargs** – additional arguments for `torch.utils.tensorboard.SummaryWriter` wrapped inside this callback

log_mid_train_loss()
Log the training loss at the batch iteration level

Logs current batch loss and the accumulated average loss.

Returns None

log_train_history_metrics(*metric_names*)
Log the train history metrics at the end of the epoch

Parameters **metric_names** (*list*) – list of train history tracked metrics to be logged

Returns None

on_train_end()
Logic executed at the end of the overall training

Returns None

on_train_loop_registration()
Execute callback initialization / preparation after the train_loop_object becomes available

Returns None

create_log_dir()

prepare_results_saver()

upload_to_cloud()
Upload sync the local version of tensorboard file to the cloud storage

Will only upload to cloud if this callback is used as part of the experiment tracking TrainLoop and the results are saved in the cloud experiment's folder.

Returns None

class aitoolbox.torchtrain.callbacks.tensorboard.**TensorboardTrainBatchLoss**(*batch_log_frequency=*, *log_dir=None*, *is_project=True*, *project_name=None*, *experiment_name=None*, *local_model_result_folde*, *cloud_save_mode=No*, *bucket_name=None*, *cloud_dir_prefix=None*, ***kwargs*)

Bases: *aitoolbox.torchtrain.callbacks.tensorboard.TensorboardReporterBaseCB*

Tensorboard training loss logger

Parameters

- **batch_log_frequency** (*int*) – frequency of logging
- **log_dir** (*str or None*) – save directory location
- **is_project** (*bool*) – set to `True` if the results should be saved into the TrainLoop-created project folder structure or to `False` if you want to save into a specific full path given in the log_dir parameter.
- **project_name** (*str or None*) – root name of the project
- **experiment_name** (*str or None*) – name of the particular experiment
- **local_model_result_folder_path** (*str or None*) – root local path where project folder will be created
- **cloud_save_mode** (*str or None*) – Storage destination selector. For AWS S3: 's3' / 'aws_s3' / 'aws' For Google Cloud Storage: 'gcs' / 'google_storage' / 'google storage' Everything else results just in local storage to disk
- **bucket_name** (*str*) – name of the bucket in the cloud storage
- **cloud_dir_prefix** (*str*) – path to the folder inside the bucket where the experiments are going to be saved
- ****kwargs** – additional arguments for `torch.utils.tensorboard.SummaryWriter` wrapped inside this callback

on_batch_end()

Logic executed after the batch is inserted into the model

Returns None

on_epoch_end()

Logic executed at the end of the epoch

Returns None

class aitoolbox.torchtrain.callbacks.tensorboard.**TensorboardTrainHistoryMetric**(*metric_names=N log_dir=None, is_project=True, project_name=N experiment_name=Non local_model_result cloud_save_mod bucket_name=No cloud_dir_prefix= **kwargs*)

Bases: *aitoolbox.torchtrain.callbacks.tensorboard.TensorboardReporterBaseCB*

Tensorboard training history values logger

At each end of epoch logs to tensorboard the last value in the training history stored for some tracked metric.

Parameters

- **metric_names** (*list or None*) – list of metric names tracked in the training history. If left to `None`, all the metrics in the training history will be logged.
- **log_dir** (*str or None*) – save directory location
- **is_project** (*bool*) – set to `True` if the results should be saved into the TrainLoop-created project folder structure or to `False` if you want to save into a specific full path given in the log_dir parameter.
- **project_name** (*str or None*) – root name of the project
- **experiment_name** (*str or None*) – name of the particular experiment
- **local_model_result_folder_path** (*str or None*) – root local path where project folder will be created
- **cloud_save_mode** (*str or None*) – Storage destination selector. For AWS S3: 's3' / 'aws_s3' / 'aws' For Google Cloud Storage: 'gcs' / 'google_storage' / 'google storage' Everything else results just in local storage to disk
- **bucket_name** (*str*) – name of the bucket in the cloud storage
- **cloud_dir_prefix** (*str*) – path to the folder inside the bucket where the experiments are going to be saved
- ****kwargs** – additional arguments for `torch.utils.tensorboard.SummaryWriter` wrapped inside this callback

on_epoch_end()

Logic executed at the end of the epoch

Returns None

class aitoolbox.torchtrain.callbacks.tensorboard.**TensorboardFullTracking**(*metric_names=None*, *batch_log_frequency=1*, *log_dir=None*, *is_project=True*, *project_name=None*, *experiment_name=None*, *local_model_result_folder_path=None*, *cloud_save_mode=None*, *bucket_name=None*, *cloud_dir_prefix=None*, ***kwargs*)

Bases: *aitoolbox.torchtrain.callbacks.tensorboard.TensorboardReporterBaseCB*

Full Tensorboard logger

At each end of epoch logs to tensorboard the last value in the training history stored for some tracked metric. Also logs the training loss at the batch iteration level.

Parameters

- **metric_names** (*list or None*) – list of metric names tracked in the training history. If left to `None`, all the metrics in the training history will be logged.
- **batch_log_frequency** (*int*) – frequency of logging
- **log_dir** (*str or None*) – save directory location
- **is_project** (*bool*) – set to `True` if the results should be saved into the TrainLoop-created project folder structure or to `False` if you want to save into a specific full path given in the log_dir parameter.
- **project_name** (*str or None*) – root name of the project
- **experiment_name** (*str or None*) – name of the particular experiment
- **local_model_result_folder_path** (*str or None*) – root local path where project folder will be created
- **cloud_save_mode** (*str or None*) – Storage destination selector. For AWS S3: 's3' / 'aws_s3' / 'aws' For Google Cloud Storage: 'gcs' / 'google_storage' / 'google storage' Everything else results just in local storage to disk
- **bucket_name** (*str*) – name of the bucket in the cloud storage
- **cloud_dir_prefix** (*str*) – path to the folder inside the bucket where the experiments are going to be saved
- ****kwargs** – additional arguments for `torch.utils.tensorboard.SummaryWriter` wrapped inside this callback

on_batch_end()

Logic executed after the batch is inserted into the model

Returns None

on_epoch_end()

Logic executed at the end of the epoch

Returns None

train_schedule

6.1.1.1.2 data

Submodules

batch_model_feed_defs

class aitoolbox.torchtrain.data.batch_model_feed_defs.**AbstractModelFeedDefinition**
Bases: abc.ABC

Model Feed Definition

The primary way of defining the model for TrainLoop training is to utilize: aitoolbox.torchtrain.model.TTModel

Use of the ModelFeedDefinition is the legacy way of defining the model. However in certain scenarios where the TTModel might prove to increase complexity, ModelFeedDefinition still is useful for augmenting the nn.Module with the logic to calculate loss and predictions.

abstract get_loss(*model*, *batch_data*, *criterion*, *device*)
Get loss during training stage

Called from fit() in TrainLoop

Executed during training stage where model weights are updated based on the loss returned from this function.

Parameters

- **model** (*nn.Module*) – neural network model
- **batch_data** – model input data batch
- **criterion** – loss criterion
- **device** – device on which the model is being trained

Returns PyTorch loss

get_loss_eval(*model*, *batch_data*, *criterion*, *device*)
Get loss during evaluation stage

Called from evaluate_model_loss() in TrainLoop.

The difference compared with get_loss() is that here the backprop weight update is not done. This function is executed in the evaluation stage not training.

For simple examples this function can just call the get_loss() and return its result.

Parameters

- **model** (*nn.Module*) – neural network model
- **batch_data** – model input data batch
- **criterion** – loss criterion
- **device** – device on which the model is being trained

Returns PyTorch loss

abstract get_predictions(*model*, *batch_data*, *device*)
Get predictions during evaluation stage

Parameters

- **model** (*nn.Module*) – neural network model
- **batch_data** – model input data batch
- **device** – device on which the model is being trained

Returns y_pred.cpu(), y_test.cpu(), metadata

Return type np.array, np.array, dict

dataset

class aitoolbox.torchtrain.data.dataset.**BasicDataset**(*data*)
Bases: torch.utils.data.dataset.Dataset

Basic PyTorch dataset where each row (first dimension) represents the example

Parameters **data** (*list or torch.Tensor*) – dataset

class aitoolbox.torchtrain.data.dataset.**ListDataset**(**data_lists*)
Bases: torch.utils.data.dataset.Dataset

Dataset wrapping lists

Each sample will be retrieved by indexing tensors along the first dimension. This is the list dataset version of PyTorch built-in TensorDataset.

Parameters ***data_lists** (*list*) – data lists that have the same size of the first dimension. Input is represented as a list of data lists.

Examples

```
list_dataset_1 = [...]
list_dataset_2 = [...]
list_dataset_3 = [...]
ListDataset(list_dataset_1, list_dataset_2, list_dataset_3)
```

6.1.1.1.3 schedulers

Submodules

basic

class aitoolbox.torchtrain.schedulers.basic.**AbstractScheduler**
Bases: object

Scheduler (callback) base class

All the scheduler callbacks should in addition to AbstractCallback also inherit from this base class. This class serves to indicate to torchtrain components which used callbacks are schedulers and which are just normal callbacks which have nothing to do with learning rate scheduling.

In addition to the above, this scheduler base class also implements the interface methods needed for saving and loading the scheduler state_dict's when checkpointing and reloading the scheduler.

When implementing the actual scheduler callback make sure to assign the created learning rate scheduler to the `self.scheduler` class member.

state_dict()

load_state_dict(*state_dict*)

class aitoolbox.torchtrain.schedulers.basic.**GeneralLRSchedulerCallback**(*scheduler_class*, *optimizer_idx=None*, ***kwargs*)

Bases: *aitoolbox.torchtrain.schedulers.basic.AbstractScheduler*, *aitoolbox.torchtrain.callbacks.abstract.AbstractCallback*

Learning rate scheduler base class

Parameters

- **scheduler_class** – PyTorch learning rate scheduler class
- **optimizer_idx** (*int or torch.optim.optimizer.Optimizer or None*) – index or the actual object reference of the paired optimizer when using multiple optimizers
- ****kwargs** – learning rate scheduler additional parameters

register_train_loop_object(*train_loop_obj*)

Modified register_train_loop_object method to support scheduler creation

Parameters **train_loop_obj** (*aitoolbox.torchtrain.train_loop.TrainLoop*) – reference to the encapsulating TrainLoop

Returns return the reference to the callback after it is registered

Return type *AbstractCallback*

on_epoch_end()

Logic executed at the end of the epoch

Returns None

class aitoolbox.torchtrain.schedulers.basic.**ReduceLROnPlateauScheduler**(***kwargs*)

Bases: *aitoolbox.torchtrain.schedulers.basic.GeneralLRSchedulerCallback*

Learning rate scheduler which reduces the rate if the loss performance stops improving

Parameters ****kwargs** – learning rate scheduler additional parameters

on_epoch_end()

Logic executed at the end of the epoch

Returns None

class aitoolbox.torchtrain.schedulers.basic.**ReduceLROnPlateauMetricScheduler**(*metric_name*, ***kwargs*)

Bases: *aitoolbox.torchtrain.schedulers.basic.GeneralLRSchedulerCallback*

Learning rate scheduler which reduces the rate if the performance of the selected metric stops improving

Needs to be used in combination with ModelPerformanceEvaluation to calculate the metric and fill it in the TrainLoop history.

Parameters

- **metric_name** (*str*) – monitored metric based on which the learning rate scheduler modifies the learning rate

- ****kwargs** – learning rate scheduler additional parameters

on_epoch_end()
Logic executed at the end of the epoch

Returns None

class aitoolbox.torchtrain.schedulers.basic.**LambdaLRScheduler**(*lr_lambda*, *execute_epoch_end=True*, *execute_batch_end=False*, ***kwargs*)

Bases: *aitoolbox.torchtrain.schedulers.basic.GeneralLRSchedulerCallback*

Sets the learning rate of each parameter group to the initial lr times a given function

When last_epoch=-1, sets initial lr as lr.

Parameters

- **lr_lambda** (*callable or list*) – A function or a list of functions which computes a multiplicative factor given an integer parameter epoch, or a list of such functions, one for each group in optimizer.param_groups.
- **execute_epoch_end** (*bool*) – should scheduler step be executed at the end of the epoch
- **execute_batch_end** (*bool*) – should scheduler step be executed at the end of each batch
- ****kwargs** – learning rate scheduler additional parameters

on_epoch_end()
Logic executed at the end of the epoch

Returns None

on_batch_end()
Logic executed after the batch is inserted into the model

Returns None

class aitoolbox.torchtrain.schedulers.basic.**StepLRScheduler**(*step_size*, ***kwargs*)

Bases: *aitoolbox.torchtrain.schedulers.basic.GeneralLRSchedulerCallback*

Sets the learning rate of each parameter group to the initial lr decayed by gamma every step_size epochs

When last_epoch=-1, sets initial lr as lr.

Parameters

- **step_size** (*int*) – period of learning rate decay
- ****kwargs** – learning rate scheduler additional parameters

class aitoolbox.torchtrain.schedulers.basic.**MultiStepLRScheduler**(*milestones_list*, ***kwargs*)

Bases: *aitoolbox.torchtrain.schedulers.basic.GeneralLRSchedulerCallback*

Set the learning rate of each parameter group to the initial lr decayed by gamma once the number of epoch reaches one of the milestones.

When last_epoch=-1, sets initial lr as lr.

Parameters

- **milestones_list** (*list*) – list of epoch indices. Must be increasing
- ****kwargs** – learning rate scheduler additional parameters

warmup

class aitoolbox.torchtrain.schedulers.warmup.**ConstantWithWarmupScheduler**(*num_warmup_steps*, *last_epoch=-1*, ***kwargs*)

Bases: *aitoolbox.torchtrain.schedulers.basic.LambdaLRScheduler*

Constant scheduler with the initial warmup

Schedule with a constant learning rate preceded by a warmup period during which the learning rate increases linearly between 0 and the initial lr set in the optimizer.

https://huggingface.co/transformers/main_classes/optimizer_schedules.html#transformers.get_constant_schedule_with_warmup

Parameters

- **num_warmup_steps** (*int*) – The number of steps for the warmup phase
- **last_epoch** (*int*) – The index of the last epoch when resuming training
- ****kwargs** – learning rate scheduler additional parameters

class aitoolbox.torchtrain.schedulers.warmup.**CosineWithWarmupScheduler**(*num_warmup_steps*, *num_training_steps*, *num_cycles=0.5*, *last_epoch=-1*, ***kwargs*)

Bases: *aitoolbox.torchtrain.schedulers.basic.LambdaLRScheduler*

Cosine decreasing scheduler with the initial warmup

Schedule with a learning rate that decreases following the values of the cosine function between the initial lr set in the optimizer to 0, after a warmup period during which it increases linearly between 0 and the initial lr set in the optimizer.

https://huggingface.co/transformers/main_classes/optimizer_schedules.html#transformers.get_cosine_schedule_with_warmup

Parameters

- **num_warmup_steps** (*int*) – The number of steps for the warmup phase
- **num_training_steps** (*int*) – The total number of training steps
- **num_cycles** (*float*) – The number of waves in the cosine schedule (the defaults is to just decrease from the max value to 0 following a half-cosine).
- **last_epoch** (*int*) – The index of the last epoch when resuming training
- ****kwargs** – learning rate scheduler additional parameters

class aitoolbox.torchtrain.schedulers.warmup.**HardRestartsCosineWithWarmupScheduler**(*num_warm*
num_traini
num_cycle
last_epoch

1,
***kwargs*)

Bases: *aitoolbox.torchtrain.schedulers.basic.LambdaLRScheduler*

Cosine scheduler with hard restarts and the initial warmup

Schedule with a learning rate that decreases following the values of the cosine function between the initial lr set in the optimizer to 0, with several hard restarts, after a warmup period during which it increases linearly between 0 and the initial lr set in the optimizer.

https://huggingface.co/transformers/main_classes/optimizer_schedules.html#transformers.get_cosine_with_hard_restarts_schedule_with_warmup

Parameters

- **num_warmup_steps** (*int*) – The number of steps for the warmup phase
- **num_training_steps** (*int*) – The total number of training steps
- **num_cycles** (*float*) – The number of waves in the cosine schedule (the defaults is to just decrease from the max value to 0 following a half-cosine).
- **last_epoch** (*int*) – The index of the last epoch when resuming training
- ****kwargs** – learning rate scheduler additional parameters

class aitoolbox.torchtrain.schedulers.warmup.**LinearWithWarmupScheduler**(*num_warmup_steps*, *num_training_steps*, *last_epoch=-1*, ***kwargs*)

Bases: *aitoolbox.torchtrain.schedulers.basic.LambdaLRScheduler*

Linearly decreasing scheduler with the initial warmup

Schedule with a learning rate that decreases linearly from the initial lr set in the optimizer to 0, after a warmup period during which it increases linearly from 0 to the initial lr set in the optimizer.

Especially useful in the context of BERT-like models. Implementation based on HuggingFace Transformers library's `get_linear_schedule_with_warmup()` method.

https://huggingface.co/transformers/main_classes/optimizer_schedules.html#transformers.get_linear_schedule_with_warmup

Parameters

- **num_warmup_steps** (*int*) – The number of steps for the warmup phase
- **num_training_steps** (*int*) – The total number of training steps
- **last_epoch** (*int*) – The index of the last epoch when resuming training
- ****kwargs** – learning rate scheduler additional parameters

6.1.1.1.4 train_loop

Subpackages

components

Submodules

callback_handler

class aitoolbox.torchtrain.train_loop.components.callback_handler.**BasicCallbacksHandler**(*trai*
Bases: object

Callback handler used for the callback orchestration inside the TrainLoop

Common use of this handler is to call different methods inside the TrainLoop at different stages of the training process. Thus execute desired callbacks' functionality at the desired point of the training process.

Parameters **train_loop_obj** (*aitoolbox.torchtrain.train_loop.TrainLoop*) – reference to the encapsulating TrainLoop

register_callbacks(*callbacks*, *cache_callbacks=False*)
Register TrainLoop object reference inside the listed callbacks when the TrainLoop is created

Normally, this is called from inside of the train loop by the TrainLoop itself. Basically train loop "registers" itself.

Parameters

- **callbacks** (*list or None*) – list of callbacks
- **cache_callbacks** (*bool*) – should provided callbacks be cached and not yet registered. First subsequent time this method is called without cache_callbacks enabled all the previously cached callbacks are added and also registered with the current list of callbacks.

Returns None

execute_epoch_begin()

execute_epoch_end()

execute_train_begin()

execute_train_end()

execute_batch_begin()

execute_batch_end()

execute_gradient_update(*optimizer_idx=0*)

execute_optimizer_step()

execute_multiprocess_start()

mp_filter_callbacks()

enforce_callbacks_quality(*callbacks*)

static print_callback_info(*callback_list*)

print_registered_callback_names()

__add__(*other*)

Parameters **other** (*list*) – callbacks list

Returns

Return type *BasicCallbacksHandler*

__iadd__(*other*)

Parameters **other** (*list*) – callbacks list

Returns

Return type *BasicCallbacksHandler*

__contains__(*item*)

Parameters **item** –

Returns

Return type bool

class aitoolbox.torchtrain.train_loop.components.callback_handler.**CallbacksHandler**(*train_loop_*

Bases: *aitoolbox.torchtrain.train_loop.components.callback_handler.BasicCallbacksHandler*

Callback handler used for the callback orchestration inside the TrainLoop

Compared to *BasicCallbacksHandler*, this handler will at certain TrainLoop stage only execute those callbacks which have implemented the functionality intended to be executed at this particular stage. Thus, *CallbacksHandler* doesn't unnecessarily execute callbacks at stages they are not implemented at.

Common use of this handler is to call different methods inside the TrainLoop at different stages of the training process. Thus execute desired callbacks' functionality at the desired point of the training process.

Parameters **train_loop_obj** (*aitoolbox.torchtrain.train_loop.TrainLoop*) – reference to the encapsulating TrainLoop

register_callbacks(*callbacks*, *cache_callbacks=False*)

Register TrainLoop object reference inside the listed callbacks when the TrainLoop is created

Normally, this is called from inside of the train loop by the TrainLoop itself. Basically train loop "registers" itself.

Parameters

- **callbacks** (*list or None*) – list of callbacks
- **cache_callbacks** (*bool*) – should provided callbacks be cached and not yet registered. First subsequent time this method is called without `cache_callbacks` enabled all the previously cached callbacks are added and also registered with the current list of callbacks.

Returns None

execute_epoch_begin()

execute_epoch_end()

execute_train_begin()

execute_train_end()

execute_batch_begin()

execute_batch_end()

execute_gradient_update(*optimizer_idx=0*)

execute_optimizer_step()

execute_multiprocess_start()

split_on_execution_position(*callbacks*, *register_train_loop=False*)

mp_filter_callbacks()

ddp_handler

class aitoolbox.torchtrain.train_loop.components.ddp_handler.**DDPHandler**(*train_loop_obj*)
Bases: object

Distributed Data Parallel process handler for the TrainLoop

Parameters **train_loop_obj** (*aitoolbox.torchtrain.train_loop.TrainLoop*) – reference to the encapsulating TrainLoop

add_distributed_samplers(*world_size*, *rank*)
Add Distributed Samplers needed for DDP to the normal single process DataLoader provided to the TrainLoop

Parameters

- **world_size** (*int*) – world size of for the distributed training
- **rank** (*int*) – rank of the current process

static build_loader_sampler(*data_loader*, *shuffle*, *world_size*, *rank*)
Replicate given data loader with added distributed sampler

Parameters

- **data_loader** (*DataLoader*) – original single process data loader without the distributed sampler
- **shuffle** (*bool*) – should the added sampler be returning examples in the shuffled order
- **world_size** (*int*) – world size of for the distributed training
- **rank** (*int*) – rank of the current process

Returns

new data loader with the sampler, reference to the distributed sampler included in the new data loader

Return type DataLoader, DistributedSampler

mp_sync(*data*, *concat_mp_data=True*)
Multiprocess data sync

Share input data between all the active processes so that every process has all the values from all the processes. This way we can achieve the same state of the data across all the parallel processes.

Parameters

- **data** (*torch.Tensor, list, float, int*) – data to be synchronized between processes. In case this is torch.Tensor, resulting output the device location will be preserved.
- **concat_mp_data** (*bool*) – should the returned list of collected tensors be concatenated into a single list of values

Returns list of *data* variable values synced across all the active processes

Return type torch.Tensor

mp_sync_dict_of_lists(*dict_list_data*)
Multiprocess dict of lists sync

Convenience wrapper around the *mp_sync()* for the specific case of dict of lists syncing.

Parameters **dict_list_data** (*dict*) – dict of lists to be synchronized across the processes

Returns synchronized dict of lists with combined values gathered from all the active processes

Return type dict

message_passing

class aitoolbox.torchtrain.train_loop.components.message_passing.**Message**(*key*, *value*, *msg_handling_settings*)

Bases: object

Wrapper object to represent the messages in the MessageService together with their handling settings

Parameters

- **key** (*str*) – message key
- **value** – message value
- **msg_handling_settings** (*str or list*) – selected message handling settings for this particular message

class aitoolbox.torchtrain.train_loop.components.message_passing.**MessageService**
Bases: object

Message Passing Service

Primarily intended for passing the messages in the TrainLoop, especially for communication or data sharing between different callbacks.

read_messages(*key*)
Read messages by key from the TrainLoop message service

Parameters **key** (*str*) – message key

Returns if message key present return content, otherwise return None

Return type list or None

write_message(*key*, *value*, *msg_handling_settings='until_end_of_epoch'*)
Write a new message to the message service

Parameters

- **key** (*str*) – message key
- **value** – message content
- **msg_handling_settings** (*str or list*) – setting how to handle the lifespan of the message. Can use one of the following message lifecycle handling settings which are variables imported from this script file and can be found defined at the beginning of the script:

  – KEEP_FOREVER

- UNTIL_END_OF_EPOCH
- UNTIL_READ
- OVERWRITE

Returns None

end_of_epoch_trigger()
Purging of the message service at the end of the epoch

Normally executed by the TrainLoop automatically after all the callbacks were executed at the end of every epoch

Returns None

static validate_msg_handling_settings(*msg_handling_settings*)

model_prediction_store

class aitoolbox.torchtrain.train_loop.components.model_prediction_store.**ModelPredictionSto**
Bases: object

Service for TrainLoop enabling the prediction caching

Prediction calculation can be costly and it can have severe performance implications if the same predictions would be calculated repeatedly. This store caches already made predictions in the current iteration of the TrainLoop which takes the cached values if they are available instead of recalculating.

Parameters **auto_purge** (*bool*) – should the prediction service cache be automatically purged at the end of each iteration

insert_train_predictions(*predictions*, *iteration_idx*, *force_prediction=False*)
Insert training dataset predictions into the cache

Parameters

- **predictions** (*tuple*) – model training dataset predictions
- **iteration_idx** (*int*) – current iteration index of the TrainLoop
- **force_prediction** (*bool*) – insert the predicted values even if they are available in the prediction cache. This causes the old cached predictions to be overwritten.

Returns None

insert_val_predictions(*predictions*, *iteration_idx*, *force_prediction=False*)
Insert validation dataset predictions into the cache

Parameters

- **predictions** (*tuple*) – model validation dataset predictions
- **iteration_idx** (*int*) – current iteration index of the TrainLoop
- **force_prediction** (*bool*) – insert the predicted values even if they are available in the prediction cache. This causes the old cached predictions to be overwritten.

Returns None

insert_test_predictions(*predictions*, *iteration_idx*, *force_prediction=False*)
Insert test dataset predictions into the cache

Parameters

- **predictions** (*tuple*) – model test dataset predictions
- **iteration_idx** (*int*) – current iteration index of the TrainLoop
- **force_prediction** (*bool*) – insert the predicted values even if they are available in the prediction cache. This causes the old cached predictions to be overwritten.

Returns None

get_train_predictions(*iteration_idx*)
Get training dataset predictions out of the cache

Parameters **iteration_idx** (*int*) – current iterating index of the TrainLoop

Returns cached model train dataset predictions

Return type tuple

get_val_predictions(*iteration_idx*)
Get validation dataset predictions out of the cache

Parameters **iteration_idx** (*int*) – current iteration index of the TrainLoop

Returns cached model validation dataset predictions

Return type tuple

get_test_predictions(*iteration_idx*)
Get test dataset predictions out of the cache

Parameters **iteration_idx** (*int*) – current iteration index of the TrainLoop

Returns cached model test dataset predictions

Return type tuple

has_train_predictions(*iteration_idx*)
Are there training dataset predictions in the cache

Parameters **iteration_idx** (*int*) – current iteration index of the TrainLoop

Returns if predictions are in the cache

Return type bool

has_val_predictions(*iteration_idx*)
Are there validation dataset predictions in the cache

Parameters **iteration_idx** (*int*) – current iteration index of the TrainLoop

Returns if predictions are in the cache

Return type bool

has_test_predictions(*iteration_idx*)
Are there test dataset predictions in the cache

Parameters **iteration_idx** (*int*) – current iteration index of the TrainLoop

Returns if predictions are in the cache

Return type bool

insert_train_loss(*loss*, *iteration_idx*, *force_prediction=False*)
Insert training dataset loss into the cache

Parameters

- **loss** (*float or dict*) – model train dataset loss

- **iteration_idx** (*int*) – current iteration index of the TrainLoop
- **force_prediction** (*bool*) – insert the loss value even if it is available in the loss cache. This causes the old cached loss value to be overwritten.

Returns None

insert_val_loss(*loss*, *iteration_idx*, *force_prediction=False*)
Insert validation dataset loss into the cache

Parameters

- **loss** (*float or dict*) – model validation dataset loss
- **iteration_idx** (*int*) – current iteration index of the TrainLoop
- **force_prediction** (*bool*) – insert the loss value even if it is available in the loss cache. This causes the old cached loss value to be overwritten.

Returns None

insert_test_loss(*loss*, *iteration_idx*, *force_prediction=False*)
Insert test dataset loss into the cache

Parameters

- **loss** (*float or dict*) – model test dataset loss
- **iteration_idx** (*int*) – current iteration index of the TrainLoop
- **force_prediction** (*bool*) – insert the loss value even if it is available in the loss cache. This causes the old cached loss value to be overwritten.

Returns None

get_train_loss(*iteration_idx*)
Get training dataset model loss out of the cache

Parameters **iteration_idx** (*int*) – current iteration index of the TrainLoop

Returns cached model train dataset loss

Return type float or dict

get_val_loss(*iteration_idx*)
Get validation dataset model loss out of the cache

Parameters **iteration_idx** (*int*) – current iteration index of the TrainLoop

Returns cached model validation dataset loss

Return type float or dict

get_test_loss(*iteration_idx*)
Get test dataset model loss out of the cache

Parameters **iteration_idx** (*int*) – current iteration index of the TrainLoop

Returns cached model test dataset loss

Return type float or dict

has_train_loss(*iteration_idx*)
Is there training dataset model loss in the cache

Parameters **iteration_idx** (*int*) – current iteration index of the TrainLoop

Returns if loss value is in the cache

Return type bool

has_val_loss(*iteration_idx*)
iteration index

has_test_loss(*iteration_idx*)
Is there test dataset model loss in the cache

Parameters **iteration_idx** (*int*) – current epoch of the TrainLoop

Returns if loss value is in the cache

Return type bool

_insert_data(*source_name*, *data*, *iteration_idx*, *force_prediction=False*)
Insert a general value into the prediction / loss cache

Parameters

- **source_name** (*str*) – data source name
- **data** (*tuple or float or dict*) – data to be cached
- **iteration_idx** (*int*) – current iteration index of the TrainLoop
- **force_prediction** (*bool*) – insert the data into the cache even if it is already available in the cache. This causes the old cached data under the same `source_name` to be overwritten.

Returns None

_get_data(*source_name*, *iteration_idx*)
Get data based on the source name from the cache

Parameters

- **source_name** (*str*) – data source name
- **iteration_idx** (*int*) – current iteration index of the TrainLoop

Returns cached data

Return type tuple or float or dict

_has_data(*source_name*, *iteration_idx*)
Check if data under the specified source name is currently available in the cache

Parameters

- **source_name** (*str*) – data source name
- **iteration_idx** (*int*) – current iteration index of the TrainLoop

Returns if the requested data is available in the cache

Return type bool

auto_purge(*iteration_idx*)
Automatically purge the current cache if the given iteration index had moved past the last cached iteration

Parameters **iteration_idx** (*int*) – current iteration index of the TrainLoop

Returns None

pred_collate_fns

aitoolbox.torchtrain.train_loop.components.pred_collate_fns.**append_predictions**(*y_batch*, *predictions*)

Parameters

- **y_batch** (*torch.Tensor*) – predictions for the new batch
- **predictions** (*list*) – accumulation list where all the batched predictions are appended

Returns predictions list with the new tensor appended

Return type list

aitoolbox.torchtrain.train_loop.components.pred_collate_fns.**append_concat_predictions**(*y_batch*, *predictions*)

Parameters

- **y_batch** (*torch.Tensor or list*) –
- **predictions** (*list*) – accumulation list where all the batched predictions are added

Returns predictions list with the new tensor appended

Return type list

aitoolbox.torchtrain.train_loop.components.pred_collate_fns.**torch_cat_transf**(*predictions*)

PyTorch concatenation of the given list of tensors

Parameters **predictions** (*list*) – expects a list of torch.Tensor

Returns concatenated tensor made up of provided smaller tensors

Return type torch.Tensor

aitoolbox.torchtrain.train_loop.components.pred_collate_fns.**keep_list_transf**(*predictions*)

Identity transformation of the predictions keeping them as they were

Parameters **predictions** (*list*) – list of predictions

Returns returns unaltered list of predictions

Return type list

Submodules

train_loop

class aitoolbox.torchtrain.train_loop.train_loop.**TrainLoop**(*model*, *train_loader*, *validation_loader*, *test_loader*, *optimizer*, *criterion*, *collate_batch_pred_fn=<function append_predictions>*, *pred_transform_fn=<function torch_cat_transf>*, *end_auto_eval=True*, *lazy_experiment_save=False*, *gpu_mode='single'*, *cuda_device_idx=None*, *use_amp=False*)

Bases: `object`

Core PyTorch TrainLoop supporting the model training and target prediction

Implements core training procedures: batch feeding into the network as part of (multi)epoch train loop, calculation of the loss & gradients. Apart from training related functionality the TrainLoop also implements the logic needed for prediction of target variables.

Parameters

- **model** (*TTModel or ModelWrap or TTDataParallel*) – neural network model
- **train_loader** (*torch.utils.data.DataLoader*) – data loader for train data set
- **validation_loader** (*torch.utils.data.DataLoader or None*) – data loader for validation data set
- **test_loader** (*torch.utils.data.DataLoader or None*) – data loader for test data set
- **optimizer** (*torch.optim.optimizer.Optimizer or MultiOptimizer*) – optimizer algorithm.
- **criterion** (*torch.nn.modules.loss._Loss or MultiLoss or None*) – criterion during the training procedure
- **collate_batch_pred_fn** (*callable*) – collate function transforming batch predictions as they come out from the model
- **pred_transform_fn** (*callable*) – function transforming all the produced predictions after all the batches have been run through the model
- **end_auto_eval** (*bool or int*) – used to optionally disable otherwise automatic end of epoch/training val/test loss calculations. This is useful when conducting very costly experiments to save on compute time. Specify either True/False boolean to always run or never run after each epoch or specify an int to execute only every specified number of epochs.
- **lazy_experiment_save** (*bool*) – when in lazy mode experiment tracking components will create the experiment folder only after some training results are available (possibly at the end of the first epoch) instead of at the beginning of training.
- **gpu_mode** (*str*) – GPU training mode selection. TrainLoop supports different GPU training modes by specifying one of the following:
  - `'single'`: single GPU training
  - `'dp'`: multi-GPU training via DataParallel

  - `'ddp'`: multi-GPU training via DistributedDataParallel
- **cuda_device_idx** (*int or None*) – CUDA device index used when training on multiple GPUs
- **use_amp** (*bool or dict*) – use 16-bit Automatic Mixed Precision (AMP)

  To switch to AMP mode either:

  - set this parameter to `True` to use default AMP `torch.cuda.amp.GradScaler` initialization params
  - provide custom AMP `torch.cuda.amp.GradScaler` initialization parameters as a dict as this parameter

fit (*num_epochs=0*, *num_iterations=0*, *callbacks=None*, *grad_accumulation=1*, ***kwargs*)
Train the model using the train loop

This is the general API method which starts the model training. By calling this method and depending on the selected training mode provided as the TrainLoop's `gpu_mode` parameter the training will start in one of the following training modes:

- Basic (CPU or single GPU) mode
- DataParallel mode
- DistributedDataParallel mode

Parameters

- **num_epochs** (*int*) – how many epochs the network will be trained
- **num_iterations** (*int*) – how many iterations (batches) the network will be trained. This enables more granular specification of the training length than the `num_epochs` parameter.
- **callbacks** (*list or None*) – callbacks that are executed during the training run
- **grad_accumulation** (*int*) – number of batches the gradients are accumulated before updating weights
- ****kwargs** – additional parameters for training methods:

  – `aitoolbox.torchtrain.train_loop.TrainLoop._train_dp()`

  – `aitoolbox.torchtrain.train_loop.TrainLoop._train_ddp()`

  These training methods are called by the TrainLoop depending on the specified setting of the TrainLoop's `gpu_mode` parameter.

Returns trained model

Return type *TTModel* or torch.nn.modules.Module or *TTDataParallel*

_train (*num_epochs*, *num_iterations*, *callbacks=None*, *grad_accumulation=1*)
Train the model using the train loop

Parameters

- **num_epochs** (*int*) – how many epochs the network will be trained
- **num_iterations** (*int*) – how many iterations (batches) the network will be trained. This enables more granular specification of the training length than the `num_epochs` parameter.
- **callbacks** (*list or None*) – callbacks that are executed during the training run

- **grad_accumulation** (*int*) – number of batches the gradients are accumulated before updating weights

Returns trained model

Return type *TTModel* or torch.nn.modules.Module or *TTDataParallel*

_calculate_batch_loss(*batch_data*)
Push batch data through the model and calculate the batch loss

Parameters **batch_data** – input data batch

Returns loss calculated on current batch

Return type loss

_backward_pass(*loss_batch*, *optimizer_idx*)
Execute backward pass from the current batch loss

Parameters

- **loss_batch** – loss calculated on current batch
- **optimizer_idx** (*int*) – index of the current optimizer. Mostly useful when using multiple optimizers. When only a single optimizer is used this parameter can be ignored.

Returns None

_optimizer_step(*optimizer_idx*)
Execute the optimizer step

Parameters **optimizer_idx** (*int*) – index of the current optimizer. Mostly useful when using multiple optimizers. When only a single optimizer is used this parameter can be ignored.

Returns None

_optimizer_zero_grad(*optimizer_idx*)
Execute optimizer zero grad

Parameters **optimizer_idx** (*int*) – index of the current optimizer. Mostly useful when using multiple optimizers. When only a single optimizer is used this parameter can be ignored.

Returns None

auto_execute_end_of_epoch()
Basic performance evaluation executed by default at the end of each epoch

Mainly evaluation of the loss functions which are always present as part of the training loop.

Returns None

auto_execute_end_of_training()
Basic performance evaluation executed by default at the end of the training process

Returns None

parse_loss(*loss_record*)
Helper function to process different possible loss formats

Primarily useful for parsing between single loss representation and the multi-loss representation.

Parameters **loss_record** (*list*) – list losses from each processed batch

Returns

in the case of single loss numpy array is returned, otherwise the dict of multiple losses is returned

Return type np.array or dict

_print_save_loss(*loss_parsed*, *loss_type_name*, *loss_print_description*)
Helper function which prints information about parsed loss and saves the loss results into the history

Parameters

- **loss_parsed** (*np.array or dict*) – parsed loss result either as a single value or as a dict of multiple losses
- **loss_type_name** (*str*) – type of the provided loss result
- **loss_print_description** (*str*) – presentation description text of the provided loss result

Returns None

evaluate_loss_on_train_set(*force_prediction=False*)
Run train dataset through the network without updating the weights and return the loss

Parameters **force_prediction** (*bool*) – recompute the loss even if it is available in the prediction cache. This causes the old cached value to be overwritten.

Returns loss, in the case of multi loss, the dict gets returned

Return type float or dict

evaluate_loss_on_validation_set(*force_prediction=False*)
Run validation dataset through the network without updating the weights and return the loss

Parameters **force_prediction** (*bool*) – recompute the loss even if it is available in the prediction cache. This causes the old cached value to be overwritten.

Returns loss, in the case of multi loss, the dict gets returned

Return type float or dict

evaluate_loss_on_test_set(*force_prediction=False*)
Run test dataset through the network without updating the weights and return the loss

Parameters **force_prediction** (*bool*) – recompute the loss even if it is available in the prediction cache. This causes the old cached value to be overwritten.

Returns loss, in the case of multi loss, the dict gets returned

Return type float or dict

evaluate_model_loss(*data_loader*)
Run given dataset through the network without updating the weights and return the loss

Parameters **data_loader** (*torch.utils.data.DataLoader*) – dataloader containing the data on which the loss is calculated

Returns loss, in the case of multi loss, the dict gets returned

Return type float or dict

predict_on_train_set(*force_prediction=False*)
Run train dataset through the network and return true target values, target predictions and metadata

Parameters **force_prediction** (*bool*) – recompute the output prediction even if it is available in the prediction cache. This causes the old cached predictions to be overwritten.

Returns y_pred, y_true, metadata

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

predict_on_validation_set(*force_prediction=False*)
Run validation dataset through the network and return true target values, target predictions and metadata

Parameters **force_prediction** (*bool*) – recompute the output prediction even if it is available in the prediction cache. This causes the old cached predictions to be overwritten.

Returns y_pred, y_true, metadata

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

predict_on_test_set(*force_prediction=False*)
Run test dataset through the network and return true target values, target predictions and metadata

Parameters **force_prediction** (*bool*) – recompute the output prediction even if it is available in the prediction cache. This causes the old cached predictions to be overwritten.

Returns y_pred, y_true, metadata

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

predict_with_model(*data_loader*)
Run given dataset through the network and return true target values, target predictions and metadata

Parameters **data_loader** (*torch.utils.data.DataLoader*) – dataloader containing the data on which the output predictions are calculated

Returns y_pred, y_true, metadata

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

insert_metric_result_into_history(*metric_name*, *metric_result*)
Insert a metric result into the train history

This is the main and preferred API function for metric insertion as part of the train loop.

Parameters

- **metric_name** (*str*) – name of the metric to be inserted
- **metric_result** (*float or dict*) – new result for the corresponding metric

get_schedulers()
Get the registered schedulers

Schedulers in TrainLoop training are implemented as callbacks under the hood.

Returns list of scheduler (callbacks)

Return type list

get_num_training_steps()
Get the number of actual training steps

Useful in case of gradient accumulation to learn the number of steps where the gradient is actually updated in between the accumulation steps.

Returns number of training steps / iterations

Return type int

_train_dp(*num_epochs*, *num_iterations*, *callbacks=None*, *grad_accumulation=1*, *dp_model_args=None*)
Train the model on multi-GPU with DataParallel auto wrapping

Parameters

- **num_epochs** (*int*) – how many epochs the network will be trained

- **num_iterations** (*int*) – how many iterations (batches) the network will be trained. This enables more granular specification of the training length than the num_epochs parameter.
- **callbacks** (*list or None*) – callbacks that are executed during the training run
- **grad_accumulation** (*int*) – number of batches the gradients are accumulated before updating weights
- **dp_model_args** (*dict or None*) – parameters for *aitoolbox.torchtrain.parallel.TTDataParallel* / nn.DataParallel DP model wrap.

Returns trained model

Return type *TTDataParallel* or nn.DataParallel

_train_ddp(*num_epochs*, *num_iterations*, *callbacks=None*, *grad_accumulation=1*, *ddp_model_args=None*, *in_process_data_load=None*, *num_nodes=1*, *node_rank=0*, *num_gpus=0*)

Train the model using the train loop in the Distributed Data Parallel setting

During the training, multiple processes will be spawned, one for each of the available GPUs.

Parameters

- **num_epochs** (*int*) – how many epochs the network will be trained
- **num_iterations** (*int*) – how many iterations (batches) the network will be trained. This enables more granular specification of the training length than the num_epochs parameter.
- **callbacks** (*list or None*) – callbacks that are executed during the training run
- **grad_accumulation** (*int*) – number of batches the gradients are accumulated before updating weights
- **ddp_model_args** (*dict or None*) – parameters for DistributedDataParallel model Available parameters for DistributedDataParallel:

  https://pytorch.org/docs/master/nn.html#torch.nn.parallel.DistributedDataParallel
- **in_process_data_load** (AbstractCallback *or list or None*) – in-process data loading logic implemented as a torchtrain callback. The logic should be placed inside the on_multiprocess_start() callback function. When using this data loading option bare in mind that loaded dataset will be replicated in memory for every spawned training process. This can in turn in cause extensive overall memory consumption.
- **num_nodes** (*int*) – number of nodes in the cluster
- **node_rank** (*int*) – rank of the current node
- **num_gpus** (*int*) – number of GPUs in the node

_spawn_fit(*gpu*, *ddp_args*, *num_epochs*, *num_iterations*, *callbacks*, *grad_accumulation*, *in_process_data_load*)

Helper function that prepares the TrainLoop state inside each of the spawned processes and initiates training

Parameters

- **gpu** (*int*) – provided by the mp.spawn(); index of the GPU allocated to the current process
- **ddp_args** (*dict*) – parameters dict needed for the distributed training setup
- **num_epochs** (*int*) – how many epochs the network will be trained

- **num_iterations** (*int*) – how many iterations (batches) the network will be trained. This enables more granular specification of the training length than the num_epochs parameter.
- **callbacks** (*list or None*) – callbacks that are executed during the training run
- **grad_accumulation** (*int*) – number of batches the gradients are accumulated before updating weights
- **in_process_data_load** (*list or None*) – in-process data loading logic implemented as a torchtrain callback. The logic should be placed inside the on_multiprocess_start() callback function. When using this data loading option bare in mind that loaded dataset will be replicated in memory for every spawned training process. This can in turn in cause extensive overall memory consumption.

__call__(*num_epochs=0*, *num_iterations=0*, *callbacks=None*, *grad_accumulation=1*, ***kwargs*)

Train the model using the train loop

This is a convenience function which calls the main TrainLoop model training method fit().

Parameters

- **num_epochs** (*int*) – how many epochs the network will be trained
- **num_iterations** (*int*) – how many iterations (batches) the network will be trained. This enables more granular specification of the training length than the num_epochs parameter.
- **callbacks** (*list*) – callbacks that are executed during the training run
- **grad_accumulation** (*int*) – number of batches the gradients are accumulated before updating weights
- ****kwargs** – additional parameters for _train_dp() and _train_ddp() methods.

Returns trained model

Return type *TTModel* or torch.nn.modules.Module or *TTDataParallel*

train_loop_tracking

class aitoolbox.torchtrain.train_loop.train_loop_tracking.**TrainLoopCheckpoint**(*model*, *train_loader*, *validation_loader*, *test_loader*, *optimizer*, *criterion*, *project_name*, *experiment_name*, *local_model_result_*, *hyperparams*, *cloud_save_mode=*, *bucket_name='mo*, *result'*, *cloud_dir_prefix=*, *source_dirs=()*, *rm_subopt_local_*, *num_best_checkpo*, *iteration_save_freq=0*, *collate_batch_pred_f*, *append_predictions>*, *pred_transform_fn*, *torch_cat_transf>*, *end_auto_eval=Tr*, *lazy_experiment_s*, *gpu_mode='single'*, *cuda_device_idx=*, *use_amp=False*)

Bases: *aitoolbox.torchtrain.train_loop.train_loop.TrainLoop*

TrainLoop with the automatic model check-pointing at the end of each epoch

Parameters

- **model** (*TTModel or ModelWrap or TTDataParallel*) – neural network model
- **train_loader** (*torch.utils.data.DataLoader*) – data loader for train data set

- **validation_loader** (*torch.utils.data.DataLoader or None*) – data loader for validation data set
- **test_loader** (*torch.utils.data.DataLoader or None*) – data loader for test data set
- **optimizer** (*torch.optim.optimizer.Optimizer or MultiOptimizer*) – optimizer algorithm.
- **criterion** (*torch.nn.modules.loss._Loss or MultiLoss or None*) – criterion during the training procedure
- **project_name** (*str*) – root name of the project
- **experiment_name** (*str*) – name of the particular experiment
- **local_model_result_folder_path** (*str*) – root local path where project folder will be created
- **hyperparams** (*dict*) – used hyper-parameters. When running the TrainLoop from jupyter notebook in order to ensure the python experiment file copying to the experiment folder, the user needs to manually specify the python file path as the value for the *experiment_file_path* key. If running the training directly from the terminal the path deduction is done automatically.
- **cloud_save_mode** (*str or None*) – Storage destination selector. For AWS S3: 's3' / 'aws_s3' / 'aws' For Google Cloud Storage: 'gcs' / 'google_storage' / 'google storage' Everything else results just in local storage to disk
- **bucket_name** (*str*) – name of the bucket in the cloud storage
- **cloud_dir_prefix** (*str*) – path to the folder inside the bucket where the experiments are going to be saved
- **source_dirs** (*list or tuple*) – paths to the local folders with the source code files used in experiment
- **rm_subopt_local_models** (*bool or str*) – if True, the deciding metric is set to 'loss'. Give string metric name to set it as a deciding metric for suboptimal model removal. If metric name consists of substring 'loss' the metric minimization is done otherwise metric maximization is done
- **num_best_checkpoints_kept** (*int*) – number of best performing models which are kept when removing suboptimal model checkpoints
- **iteration_save_freq** (*int*) – frequency of saving the model checkpoint every specified number of training iterations
- **collate_batch_pred_fn** (*callable*) – collate function transforming batch predictions as they come out from the model
- **pred_transform_fn** (*callable*) – function transforming all the produced predictions after all the batches have been run through the model
- **end_auto_eval** (*bool or int*) – used to optionally disable otherwise automatic end of epoch/training val/test loss calculations. This is useful when conducting very costly experiments to save on compute time. Specify either True/False boolean to always run or never run after each epoch or specify an int to execute only every specified number of epochs.
- **lazy_experiment_save** (*bool*) – when in lazy mode experiment tracking components will create the experiment folder only after some training results are available (possibly at the end of the first epoch) instead of at the beginning of training.

- **gpu_mode** (*str*) – GPU training mode selection. TrainLoop supports different GPU training modes by specifying one of the following:
  - `'single'`: single GPU training
  - `'dp'`: multi-GPU training via DataParallel
  - `'ddp'`: multi-GPU training via DistributedDataParallel
- **cuda_device_idx** (*int or None*) – CUDA device index used when training on multiple GPUs
- **use_amp** (*bool or dict*) – use 16-bit Automatic Mixed Precision (AMP)

  To switch to AMP mode either:
  - set this parameter to `True` to use default AMP `torch.cuda.amp.GradScaler` initialization params
  - provide custom AMP `torch.cuda.amp.GradScaler` initialization parameters as a dict as this parameter

class aitoolbox.torchtrain.train_loop.train_loop_tracking.**TrainLoopEndSave**(*model*, *train_loader*, *validation_loader*, *test_loader*, *optimizer*, *criterion*, *project_name*, *experiment_name*, *local_model_result_folde*
hyperparams, *val_result_package=N*
test_result_package=N
cloud_save_mode='s3'
bucket_name='model-result',
cloud_dir_prefix='',
source_dirs=(),
collate_batch_pred_fn=<
append_predictions>,
pred_transform_fn=<f
torch_cat_transf>,
end_auto_eval=True,
lazy_experiment_save=
gpu_mode='single',
cuda_device_idx=None
use_amp=False)

Bases: *aitoolbox.torchtrain.train_loop.train_loop.TrainLoop*

TrainLoop with the model performance evaluation and final model saving at the end of the training process

Parameters

- **model** (*TTModel or ModelWrap or TTDataParallel*) – neural network model
- **train_loader** (*torch.utils.data.DataLoader*) – data loader for train data set
- **validation_loader** (*torch.utils.data.DataLoader or None*) – data loader for validation data set
- **test_loader** (*torch.utils.data.DataLoader or None*) – data loader for test data set
- **optimizer** (*torch.optim.optimizer.Optimizer or MultiOptimizer*) –

optimizer algorithm.

- **criterion** (*torch.nn.modules.loss._Loss or MultiLoss or None*) – criterion during the training procedure
- **project_name** (*str*) – root name of the project
- **experiment_name** (*str*) – name of the particular experiment
- **local_model_result_folder_path** (*str*) – root local path where project folder will be created
- **hyperparams** (*dict*) – used hyper-parameters. When running the TrainLoop from jupyter notebook in order to ensure the python experiment file copying to the experiment folder, the user needs to manually specify the python file path as the value for the *experiment_file_path* key. If running the training directly from the terminal the path deduction is done automatically.
- **val_result_package** (*AbstractResultPackage or None*) – result package evaluated on validation data at the end of the training
- **test_result_package** (*AbstractResultPackage or None*) – result package evaluated on test data at the end of the training
- **cloud_save_mode** (*str or None*) – Storage destination selector. For AWS S3: 's3' / 'aws_s3' / 'aws' For Google Cloud Storage: 'gcs' / 'google_storage' / 'google storage' Everything else results just in local storage to disk
- **bucket_name** (*str*) – name of the bucket in the cloud storage
- **cloud_dir_prefix** (*str*) – path to the folder inside the bucket where the experiments are going to be saved
- **source_dirs** (*list or tuple*) – paths to the local folders with the source code files used in experiment
- **collate_batch_pred_fn** (*callable*) – collate function transforming batch predictions as they come out from the model
- **pred_transform_fn** (*callable*) – function transforming all the produced predictions after all the batches have been run through the model
- **end_auto_eval** (*bool or int*) – used to optionally disable otherwise automatic end of epoch/training val/test loss calculations. This is useful when conducting very costly experiments to save on compute time. Specify either True/False boolean to always run or never run after each epoch or specify an int to execute only every specified number of epochs.
- **lazy_experiment_save** (*bool*) – when in lazy mode experiment tracking components will create the experiment folder only after some training results are available (possibly at the end of the first epoch) instead of at the beginning of training.
- **gpu_mode** (*str*) – GPU training mode selection. TrainLoop supports different GPU training modes by specifying one of the following:
  - `'single'`: single GPU training
  - `'dp'`: multi-GPU training via DataParallel
  - `'ddp'`: multi-GPU training via DistributedDataParallel
- **cuda_device_idx** (*int or None*) – CUDA device index used when training on multiple GPUs

- **use_amp** (*bool or dict*) – use 16-bit Automatic Mixed Precision (AMP)

  To switch to AMP mode either:

  – set this parameter to `True` to use default AMP `torch.cuda.amp.GradScaler` initialization params

  – provide custom AMP `torch.cuda.amp.GradScaler` initialization parameters as a dict as this parameter

check_if_result_packages_possible()

class aitoolbox.torchtrain.train_loop.train_loop_tracking.**TrainLoopCheckpointEndSave**(*model*, *train_lo* *val-* *i-* *da-* *tion_loa* *test_loa* *op-* *ti-* *mizer*, *cri-* *te-* *rion*, *project_* *ex-* *per-* *i-* *ment_na* *lo-* *cal_mod* *hy-* *per-* *params*, *val_resu* *test_res* *cloud_s* *bucket_* *result'*, *cloud_d* *source_* *rm_subd* *num_be* *it-* *er-* *a-* *tion_sav* *col-* *late_bat* *ap-* *pend_pr* *pred_tra* *torch_c* *end_aut* *lazy_exp* *gpu_mo* *cuda_de* *use_amp*

Bases: *aitoolbox.torchtrain.train_loop.train_loop_tracking.TrainLoopEndSave*

TrainLoop both saving model check-pointing at the end of each epoch and model performance reporting and model saving at the end of the training process

Parameters

- **model** (*TTModel or ModelWrap or TTDataParallel*) – neural network model

- **train_loader** (*torch.utils.data.DataLoader*) – data loader for train data set
- **validation_loader** (*torch.utils.data.DataLoader or None*) – data loader for validation data set
- **test_loader** (*torch.utils.data.DataLoader or None*) – data loader for test data set
- **optimizer** (*torch.optim.optimizer.Optimizer or MultiOptimizer*) – optimizer algorithm.
- **criterion** (*torch.nn.modules.loss._Loss or MultiLoss or None*) – criterion during the training procedure
- **project_name** (*str*) – root name of the project
- **experiment_name** (*str*) – name of the particular experiment
- **local_model_result_folder_path** (*str*) – root local path where project folder will be created
- **hyperparams** (*dict*) – used hyper-parameters. When running the TrainLoop from jupyter notebook in order to ensure the python experiment file copying to the experiment folder, the user needs to manually specify the python file path as the value for the *experiment_file_path* key. If running the training directly from the terminal the path deduction is done automatically.
- **val_result_package** (*AbstractResultPackage or None*) – result package evaluated on validation data at the end of the training
- **test_result_package** (*AbstractResultPackage or None*) – result package evaluated on test data at the end of the training
- **cloud_save_mode** (*str or None*) – Storage destination selector. For AWS S3: 's3' / 'aws_s3' / 'aws' For Google Cloud Storage: 'gcs' / 'google_storage' / 'google storage' Everything else results just in local storage to disk
- **bucket_name** (*str*) – name of the bucket in the cloud storage
- **cloud_dir_prefix** (*str*) – path to the folder inside the bucket where the experiments are going to be saved
- **source_dirs** (*list or tuple*) – paths to the local folders with the source code files used in experiment
- **rm_subopt_local_models** (*bool or str*) – if True, the deciding metric is set to 'loss'. Give string metric name to set it as a deciding metric for suboptimal model removal. If metric name consists of substring 'loss' the metric minimization is done otherwise metric maximization is done
- **num_best_checkpoints_kept** (*int*) – number of best performing models which are kept when removing suboptimal model checkpoints
- **iteration_save_freq** (*int*) – frequency of saving the model checkpoint every specified number of training iterations
- **collate_batch_pred_fn** (*callable*) – collate function transforming batch predictions as they come out from the model
- **pred_transform_fn** (*callable*) – function transforming all the produced predictions after all the batches have been run through the model

- **end_auto_eval** (*bool or int*) – used to optionally disable otherwise automatic end of epoch/training val/test loss calculations. This is useful when conducting very costly experiments to save on compute time. Specify either True/False boolean to always run or never run after each epoch or specify an int to execute only every specified number of epochs.
- **lazy_experiment_save** (*bool*) – when in lazy mode experiment tracking components will create the experiment folder only after some training results are available (possibly at the end of the first epoch) instead of at the beginning of training.
- **gpu_mode** (*str*) – GPU training mode selection. TrainLoop supports different GPU training modes by specifying one of the following:
  - `'single'`: single GPU training
  - `'dp'`: multi-GPU training via DataParallel
  - `'ddp'`: multi-GPU training via DistributedDataParallel
- **cuda_device_idx** (*int or None*) – CUDA device index used when training on multiple GPUs
- **use_amp** (*bool or dict*) – use 16-bit Automatic Mixed Precision (AMP)

  To switch to AMP mode either:
  - set this parameter to `True` to use default AMP `torch.cuda.amp.GradScaler` initialization params
  - provide custom AMP `torch.cuda.amp.GradScaler` initialization parameters as a dict as this parameter

6.1.1.2 Submodules

6.1.1.2.1 model

class aitoolbox.torchtrain.model.**TTModel**

Bases: `torch.nn.modules.module.Module`, `abc.ABC`

TTModel is an extension of core PyTorch nn.Module

TT in TTModel –> TorchTrain Model

In addition to the common `forward()` method required by the base nn.Module, the user also needs to implement the additional AIToolbox specific `get_loss()` and `get_predictions()` methods.

`transfer_model_attributes` (list or tuple): additional TTModel attributes which need to be transferred to the TTDataParallel level to enable their use in the transferred/exposed class methods. When coding the model's __init__() method user should also fill in the string names of attributes that should be transferred in case the model is wrapped for DP/DDP.

Initializes internal Module state, shared by both nn.Module and ScriptModule.

abstract get_loss(*batch_data*, *criterion*, *device*)

Get loss during training stage

Called from fit() in TrainLoop

Executed during training stage where model weights are updated based on the loss returned from this function.

Parameters

- **batch_data** – model input data batch

- **criterion** – loss criterion
- **device** – device on which the model is being trained

Returns PyTorch loss

get_loss_eval(*batch_data*, *criterion*, *device*)
Get loss during evaluation stage

Called from evaluate_model_loss() in TrainLoop.

The difference compared with get_loss() is that here the backprop weight update is not done. This function is executed in the evaluation stage not training.

For simple examples this function can just call the get_loss() and return its result.

Parameters

- **batch_data** – model input data batch
- **criterion** – loss criterion
- **device** – device on which the model is being trained

Returns PyTorch loss

abstract get_predictions(*batch_data*, *device*)
Get predictions during evaluation stage

Parameters

- **batch_data** – model input data batch
- **device** – device on which the model is making the prediction

Returns y_pred.cpu(), y_test.cpu(), metadata

Return type np.array, np.array, dict

training: bool

class aitoolbox.torchtrain.model.**TTBasicModel**
Bases: *aitoolbox.torchtrain.model.TTModel*

Extension of the TTModel abstract class with already implemented simple loss and prediction calculation functions

The pre-implemented get_loss() and get_predictions() will take all the provided data sources from the data loader except the last one as an input to the model. The last data source from the data loader will be treated as the target variable. (*batch_input_data, targets = batch_data)

This base class is mainly meant to be used for simple models. TTBasicModel removes the need to constantly duplicate code in get_loss and get_predictions.

Initializes internal Module state, shared by both nn.Module and ScriptModule.

get_loss(*batch_data*, *criterion*, *device*)
Get loss during training stage

Called from fit() in TrainLoop

Executed during training stage where model weights are updated based on the loss returned from this function.

Parameters

- **batch_data** – model input data batch

- **criterion** – loss criterion
- **device** – device on which the model is being trained

Returns PyTorch loss

get_predictions(*batch_data*, *device*)
Get predictions during evaluation stage

Parameters

- **batch_data** – model input data batch
- **device** – device on which the model is making the prediction

Returns y_pred.cpu(), y_test.cpu(), metadata

Return type np.array, np.array, dict

training: bool

class aitoolbox.torchtrain.model.**TTBasicMultiGPUModel**
Bases: *aitoolbox.torchtrain.model.TTBasicModel*

Extension of the TTModel abstract class with already implemented simple loss and prediction calculation functions which support leveled utilization when training on multi-GPU.

The pre-implemented get_loss() and get_predictions() will take all the provided data sources from the data loader except the last one as an input to the model. The last data source from the data loader will be treated as the target variable. (*batch_input_data, targets = batch_data)

In the case of the get_loss() the inout into the model's forward() function will also provide *targets* and *criterion* arguments in order to enable calculation of the loss inside forward() function.

The forward() function should have the following parameter signature and should finish with:

def forward(*batch_input_data, targets=None, criterion=None): ... predictions calculation via the computational graph ...

if criterion is not None: return criterion(predictions, targets)

else: return predictions

This base class is mainly meant to be used for simple models. TTBasicModel removes the need to constantly duplicate code in get_loss and get_predictions.

Initializes internal Module state, shared by both nn.Module and ScriptModule.

get_loss(*batch_data*, *criterion*, *device*)
Get loss during training stage

Called from fit() in TrainLoop

Executed during training stage where model weights are updated based on the loss returned from this function.

Parameters

- **batch_data** – model input data batch
- **criterion** – loss criterion
- **device** – device on which the model is being trained

Returns PyTorch loss

training: bool

class aitoolbox.torchtrain.model.**MultiGPUModelWrap**(*model*)
Bases: *aitoolbox.torchtrain.model.TTBasicMultiGPUModel*

Model wrapper optimizing the model for multi-GPU training by moving the loss calculation to the GPUs

Parameters **model** (*nn.Module or TTModel*) – neural network model. The model should follow the basic PyTorch model definition where the forward() function returns predictions

forward(**input_data*, *targets=None*, *criterion=None*)
DP friendly forward abstraction on top of the wrapped model's usual forward() function

Parameters

- ***input_data** – whatever input data should be passed into the wrapped model's forward() function
- **targets** – target variables which the model is training to fit
- **criterion** – loss function

Returns

PyTorch loss or model output predictions. If loss function criterion is provided this function returns the calculated loss, otherwise the model output predictions are returned

training: bool

class aitoolbox.torchtrain.model.**ModelWrap**(*model*, *batch_model_feed_def*)
Bases: object

TrainLoop model wrapper combining PyTorch model and model feed definition

NOTE: especially useful in the case when you want to train on multi-GPU where TTModel abstract functions can't be used.

ModelWrap can be used as a replacement of TTModel when using the TrainLoop.

Parameters

- **model** (*nn.Module*) – neural network model
- **batch_model_feed_def** (*AbstractModelFeedDefinition or None*) – data prep definition for batched data. This definition prepares the data for each batch that gets than fed into the neural network.

6.1.1.2.2 model_predict

class aitoolbox.torchtrain.model_predict.**PyTorchModelPredictor**(*model*, *data_loader*, *callbacks=None*)
Bases: object

PyTorch model predictions based on provided dataloader

Parameters

- **model** (*aitoolbox.torchtrain.model.TTModel or aitoolbox.torchtrain.model.ModelWrap*) – neural network model
- **data_loader** (*torch.utils.data.DataLoader*) – dataloader based on which the model output predictions are made

model_predict()
Calculate model output predictons

Returns y_pred, y_true, metadata

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

model_get_loss(*loss_criterion*)
Calculate model's loss on the given dataloader and based on provided loss function

Parameters **loss_criterion** (*torch.nn.modules.loss._Loss*) – criterion criterion during the training procedure

Returns loss

Return type float

evaluate_model(*result_package*, *project_name*, *experiment_name*, *local_model_result_folder_path*, *cloud_save_mode='s3'*, *bucket_name='model-result'*, *cloud_dir_prefix=''*, *save_true_pred_labels=False*)
Evaluate model's performance with full experiment tracking

Parameters

- **result_package** (aitoolbox.experiment.result_package.abstract_result_packages.AbstractResultPackage) – result package defining the evaluation metrics on which the model is evaluated when predicting the values from the provided dataloader
- **project_name** (*str*) – root name of the project
- **experiment_name** (*str*) – name of the particular experiment
- **local_model_result_folder_path** (*str*) – root local path where project folder will be created
- **cloud_save_mode** (*str or None*) – Storage destination selector. For AWS S3: 's3' / 'aws_s3' / 'aws' For Google Cloud Storage: 'gcs' / 'google_storage' / 'google storage' Everything else results just in local storage to disk
- **bucket_name** (*str*) – name of the bucket in the cloud storage
- **cloud_dir_prefix** (*str*) – path to the folder inside the bucket where the experiments are going to be saved
- **save_true_pred_labels** (*bool*) – should ground truth labels also be saved

Returns None

evaluate_result_package(*result_package*, *return_result_package=True*)
Evaluate model's performance based on provided Result Package

Parameters

- **result_package** (aitoolbox.experiment.result_package.abstract_result_packages.AbstractResultPackage) –
- **return_result_package** (*bool*) – if True, the full calculated result package is returned, otherwise only the results dict is returned

Returns

calculated result package or results dict

Return type *aitoolbox.experiment.result_package.abstract_result_packages.AbstractResultPackage* or dict

execute_batch_end_callbacks()
Execute provided callbacks which are triggered at the end of the batch in train loop

Returns None

execute_epoch_end_callbacks()
Execute provided callbacks which are triggered at the end of the epoch in train loop

Returns None

evaluate_metric(*metric_class*, *return_metric=True*)
Evaluate a model with a single performance metric

Only for really simple cases where the output from the network can be directly used for metric calculation. For more advanced cases where the network output needs to be preprocessed before the metric evaluation, the use of the result package is preferred.

Parameters

- **metric_class** (aitoolbox.experiment.core_metrics.abstract_metric.AbstractBaseMetric) – metric class not the object
- **return_metric** (*bool*) – if True, the full performance metric object is returned, otherwise only metric result dict is returned

Returns

calculated performance metric or result dict

Return type *aitoolbox.experiment.core_metrics.abstract_metric.AbstractBaseMetric* or dict

evaluate_metric_list(*metrics_class_list*, *return_metric_list=True*)
Evaluate a model with a list of performance metrics

Parameters

- **metrics_class_list** (*list*) – list of metric classes not the objects
- **return_metric_list** (*bool*) – if True, the full performance metrics objects are returned, otherwise only metric results dict is returned

Returns list of calculated performance metrics or results dict

Return type list or dict

6.1.1.2.3 multi_loss_optim

class aitoolbox.torchtrain.multi_loss_optim.**MultiLoss**(*loss_dict*, *loss_optimizer_map=None*, *retain_graph_until_last=True*)
Bases: object

Multiple loss wrapper for TrainLoop based training

Parameters

- **loss_dict** (*dict*) – dict of loss objects which are used to calculate losses in the TrainLoop
- **loss_optimizer_map** (*dict or None*) – dict mapping the loss name to the corresponding optimizer's index in the MultiOptimizer. If this parameter is left to None the mapping is automatically created by assigning values from range(len(loss_dict)) as corresponding optimizer indices.

- **retain_graph_until_last** (*bool*) – when calling backward should `retain_graph` option be enabled for all but last loss tensor

backward(*optimizer_idx*, *iteration*, *amp_grad_scaler*)
Executes backward() for the specific loss based on provided optimizer_idx

Parameters

- **optimizer_idx** (*int*) – index of the current optimizer. Mostly useful when using multiple optimizers. When only a single optimizer is used this parameter can be ignored.
- **iteration** (*int*) – Current iteration index. Not used in the most simple setup but provided in case of more elaborate loss backward logic is devised.
- **amp_grad_scaler** (*torch.cuda.amp.GradScaler*) – AMP GradScaler. If scaler `enabled` parameter is set to False the loss is still passed to it but it gets returned unscaled so the behaviour is as it is in the case of non-AMP training.

Returns None

item()

class aitoolbox.torchtrain.multi_loss_optim.**MultiOptimizer**(*optimizer_list*)
Bases: `object`

Multiple optimizer wrapper for TrainLoop based training

Parameters **optimizer_list** (*list*) – list of optimizer objects which are used in the TrainLoop

step(*optimizer_idx*, *iteration*, *amp_grad_scaler*)
Execute step for optimizer at the specified index

Parameters

- **optimizer_idx** (*int*) – index of the current optimizer. Mostly useful when using multiple optimizers. When only a single optimizer is used this parameter can be ignored.
- **iteration** (*int*) – Current iteration index. Not used in the most simple setup but provided in case of more elaborate loss backward logic is devised.
- **amp_grad_scaler** (*torch.cuda.amp.GradScaler*) – AMP GradScaler. If scaler `enabled` parameter is set to False the optimizer have it's normal step() method called withoud applying the AMP mandated unscaling beforehand. In this respect the behaviour will be the same as in the non-AMP training.

Returns None

zero_grad(*optimizer_idx*, *iteration*)
Execute zero_grad for optimizer at the specified index

Parameters

- **optimizer_idx** (*int*) – index of the current optimizer. Mostly useful when using multiple optimizers. When only a single optimizer is used this parameter can be ignored.
- **iteration** (*int*) – Current iteration index. Not used in the most simple setup but provided in case of more elaborate loss backward logic is devised.

Returns None

state_dict()

load_state_dict(*state_dict_list*)

6.1.1.2.4 parallel

class aitoolbox.torchtrain.parallel.**TTParallelBase**(*module*, *default_model_methods=('get_loss', 'get_loss_eval', 'get_predictions')*)

Bases: object

torchtrain parallel base class used for transferring TTModel functions to the PyTorch Parallel wrappers level

Parameters

- **module** (aitoolbox.torchtrain.model.TTModel) – neural network model
- **default_model_methods** (*list or tuple*) – list of core methods which are present also in TTModel abstract class

get_loss(*batch_data*, *criterion*, *device*)

get_loss_eval(*batch_data*, *criterion*, *device*)

get_predictions(*batch_data*, *device*)

class aitoolbox.torchtrain.parallel.**TTDataParallel**(*module*, *default_model_methods=('get_loss', 'get_loss_eval', 'get_predictions')*, ***kwargs*)

Bases: torch.nn.parallel.data_parallel.DataParallel, *aitoolbox.torchtrain.parallel.TTParallelBase*

torchtrain enabled DataParallel

This DataParallel wrapper works in the same way as the original PyTorch nn.DataParallel. Furthermore it exposes TTModel batch data feeding definitions (additional abstract methods) to the TrainLoop while still enabling multi GPU training.

Parameters

- **module** (aitoolbox.torchtrain.model.TTModel) – neural network model
- **default_model_methods** (*list or tuple*) – list of core methods which are present also in TTModel abstract class
- ****kwargs** – additional parameters for underlying nn.DataParallel

training: bool

class aitoolbox.torchtrain.parallel.**TTDistributedDataParallel**(*module*, *default_model_methods=('get_loss', 'get_loss_eval', 'get_predictions')*, ***kwargs*)

Bases: torch.nn.parallel.distributed.DistributedDataParallel, *aitoolbox.torchtrain.parallel.TTParallelBase*

torchtrain enabled DistributedDataParallel

Parameters

- **module** (aitoolbox.torchtrain.model.TTModel) – neural network model
- **default_model_methods** (*list or tuple*) – list of core methods which are present also in TTModel abstract class
- ****kwargs** – additional parameters for underlying nn.parallel.DistributedDataParallel

training: bool

6.1.2 experiment

6.1.2.1 Subpackages

6.1.2.1.1 core_metrics

Submodules

abstract_metric

class aitoolbox.experiment.core_metrics.abstract_metric.**AbstractBaseMetric**(*y_true*, *y_predicted*, *metric_name*, *np_array=True*)

Bases: abc.ABC

Base metric with core metric functionality needed by all the derived actual performance metrics

Parameters

- **y_true** (*numpy.array or list or str*) – ground truth targets
- **y_predicted** (*numpy.array or list or str*) – predicted targets
- **metric_name** (*str*) – name of the calculated metric
- **np_array** (*bool*) – should the provided targets be converted to numpy array or left as they are

abstract calculate_metric()

Perform metric calculation and return it from this function

Returns return metric_result

Return type float or dict

get_metric()

Returns metric result

Returns return metric_result

Return type float or dict

get_metric_dict()

Creates and return metric result key-value dict

Returns metric dict

Return type dict

_get_metric_self_other_val(*other*)

Metric comparison prep util

Parameters **other** (aitoolbox.experiment.core_metrics.abstract_metric.AbstractBaseMetric *or float or int*) – other compared metric

Returns metric value

Return type float or int

__add__(*other*)
Concatenate two metrics

Parameters **other** (*AbstractBaseMetric or dict*) – new metric to be added

Returns combined metric dict

Return type dict

__radd__(*other*)
Append another metric

Parameters **other** (*AbstractBaseMetric or dict*) – new metric to be added

Returns combined metric dict

Return type dict

concat_metric(*other*)
Concatenate another metric to this one

Parameters **other** (*AbstractBaseMetric or dict*) – new metric to be added

Returns combined metric dict

Return type dict

classification

class aitoolbox.experiment.core_metrics.classification.**AccuracyMetric**(*y_true*, *y_predicted*, *positive_class_thresh=0.5*)
Bases: *aitoolbox.experiment.core_metrics.abstract_metric.AbstractBaseMetric*

Model prediction accuracy

Parameters

- **y_true** (*numpy.array or list*) – ground truth targets
- **y_predicted** (*numpy.array or list*) – predicted targets
- **positive_class_thresh** (*float or None*) – predicted probability positive class threshold. Set it to None when dealing with multi-class labels.

calculate_metric()
Perform metric calculation and return it from this function

Returns return metric_result

Return type float or dict

class aitoolbox.experiment.core_metrics.classification.**ROCAUCMetric**(*y_true*, *y_predicted*)
Bases: *aitoolbox.experiment.core_metrics.abstract_metric.AbstractBaseMetric*

Model prediction ROC-AUC

Parameters

- **y_true** (*numpy.array or list*) – ground truth targets
- **y_predicted** (*numpy.array or list*) – predicted targets

calculate_metric()
Perform metric calculation and return it from this function

Returns return metric_result

Return type float or dict

class aitoolbox.experiment.core_metrics.classification.**PrecisionRecallCurveAUCMetric**(*y_true*, *y_predic*

Bases: *aitoolbox.experiment.core_metrics.abstract_metric.AbstractBaseMetric*

Model prediction PR-AUC

Parameters

- **y_true** (*numpy.array or list*) – ground truth targets
- **y_predicted** (*numpy.array or list*) – predicted targets

calculate_metric()
Perform metric calculation and return it from this function

Returns return metric_result

Return type float or dict

class aitoolbox.experiment.core_metrics.classification.**F1ScoreMetric**(*y_true*, *y_predicted*, *positive_class_thresh=0.5*)

Bases: *aitoolbox.experiment.core_metrics.abstract_metric.AbstractBaseMetric*

Model prediction F1 score

Parameters

- **y_true** (*numpy.array or list*) – ground truth targets
- **y_predicted** (*numpy.array or list*) – predicted targets
- **positive_class_thresh** (*float*) – predicted probability positive class threshold

calculate_metric()
Perform metric calculation and return it from this function

Returns return metric_result

Return type float or dict

class aitoolbox.experiment.core_metrics.classification.**PrecisionMetric**(*y_true*, *y_predicted*, *positive_class_thresh=0.5*)

Bases: *aitoolbox.experiment.core_metrics.abstract_metric.AbstractBaseMetric*

Model prediction precision

Parameters

- **y_true** (*numpy.array or list*) – ground truth targets
- **y_predicted** (*numpy.array or list*) – predicted targets
- **positive_class_thresh** (*float*) – predicted probability positive class threshold

calculate_metric()
Perform metric calculation and return it from this function

Returns return metric_result

Return type float or dict

class aitoolbox.experiment.core_metrics.classification.**RecallMetric**(*y_true*, *y_predicted*, *positive_class_thresh=0.5*)

Bases: *aitoolbox.experiment.core_metrics.abstract_metric.AbstractBaseMetric*

Model prediction recall score

Parameters

- **y_true** (*numpy.array or list*) – ground truth targets
- **y_predicted** (*numpy.array or list*) – predicted targets
- **positive_class_thresh** (*float*) – predicted probability positive class threshold

calculate_metric()

Perform metric calculation and return it from this function

Returns return metric_result

Return type float or dict

regression

class aitoolbox.experiment.core_metrics.regression.**MeanSquaredErrorMetric**(*y_true*, *y_predicted*)

Bases: *aitoolbox.experiment.core_metrics.abstract_metric.AbstractBaseMetric*

Model prediction MSE

Parameters

- **y_true** (*numpy.array or list*) – ground truth targets
- **y_predicted** (*numpy.array or list*) – predicted targets

calculate_metric()

Perform metric calculation and return it from this function

Returns return metric_result

Return type float or dict

class aitoolbox.experiment.core_metrics.regression.**MeanAbsoluteErrorMetric**(*y_true*, *y_predicted*)

Bases: *aitoolbox.experiment.core_metrics.abstract_metric.AbstractBaseMetric*

Model prediction MAE

Parameters

- **y_true** (*numpy.array or list*) – ground truth targets
- **y_predicted** (*numpy.array or list*) – predicted targets

calculate_metric()

Perform metric calculation and return it from this function

Returns return metric_result

Return type float or dict

6.1.2.1.2 local_load

Submodules

local_model_load

class aitoolbox.experiment.local_load.local_model_load.**AbstractLocalModelLoader**
Bases: abc.ABC

abstract load_model(*project_name*, *experiment_name*, *experiment_timestamp*, *model_save_dir*, *epoch_num=None*, ***kwargs*)
Model loading method all the model loaders need to implement

Parameters

- **project_name** (*str*) – root name of the project
- **experiment_name** (*str*) – name of the particular experiment
- **experiment_timestamp** (*str*) – time stamp at the start of training
- **model_save_dir** (*str*) – name of the folder inside experiment folder where the model is saved
- **epoch_num** (*int or None*) – epoch number of the model checkpoint or none if loading final model
- ****kwargs** – additional parameters for specific framework model loader

Returns model

class aitoolbox.experiment.local_load.local_model_load.**PyTorchLocalModelLoader**(*local_model_resu*
Bases: *aitoolbox.experiment.local_load.local_model_load.AbstractLocalModelLoader*

PyTorch saved model loader and initializer

Parameters **local_model_result_folder_path** (*str*) – root local path where project folder will be created

load_model(*project_name*, *experiment_name*, *experiment_timestamp*, *model_save_dir='checkpoint_model'*, *epoch_num=None*, *map_location=None*)
Model loading interface compatible with the experiment folder structure maintained by the AIToolbox TrainLoop

Parameters

- **project_name** (*str*) – root name of the project
- **experiment_name** (*str*) – name of the particular experiment
- **experiment_timestamp** (*str*) – time stamp at the start of training
- **model_save_dir** (*str*) – name of the folder inside experiment folder where the model is saved
- **epoch_num** (*int or None*) – epoch number of the model checkpoint or none if loading final model
- **map_location** (*str or None*) –

Returns model

load_model_from_path(*model_path*, *map_location=None*)
General model loading when the AIToolbox TrainLoop experiment folder structure is not used

Parameters

- **model_path** (*str*) – full path to the model
- **map_location** (*str or None*) – a function, `torch.device`, string or a dict specifying how to remap storage locations

Returns model

check_if_model_loaded()

init_model(*model*, *used_data_parallel=False*)
Initialize provided PyTorch model with the loaded model weights

For this function to work, load_model() must be first called to read the model representation into memory.

Parameters

- **model** (*TTModel or nn.Module*) – PyTorch model
- **used_data_parallel** (*bool*) – if the saved model was nn.DataParallel or normal model

Returns PyTorch model

init_optimizer(*optimizer*, *device='cuda'*)
Initialize the optimizer based on saved model/optimizer checkpoint

Parameters

- **optimizer** – PyTorch optimizer
- **device** (*str*) – device id

Returns PyTorch optimizer

init_scheduler(*scheduler_callbacks_list*, *ignore_saved=False*, *ignore_missing_saved=False*)
Initialize the list of schedulers based on saved model/optimizer/scheduler checkpoint

Parameters

- **scheduler_callbacks_list** (*list*) – list of scheduler (callbacks)
- **ignore_saved** (*bool*) – if exception should be raised in the case there are found scheduler snapshots in the checkpoint, but not schedulers are provided to this method
- **ignore_missing_saved** (*bool*) – if exception should be raised in the case schedulers are provided to this method but no saved scheduler snapshots can be found in the checkpoint

Returns list of initialized scheduler (callbacks)

Return type list

init_amp(*amp_scaler*)
Initialize AMP GradScaler

Parameters **amp_scaler** (*torch.cuda.amp.GradScaler*) – AMP GradScaler

Returns initialized AMP GradScaler

Return type torch.cuda.amp.GradScaler

6.1.2.1.3 local_save

Submodules

folder_create

class aitoolbox.experiment.local_save.folder_create.**ExperimentFolder**
Bases: object

static create_base_folder(*project_name*, *experiment_name*, *experiment_timestamp*, *local_model_result_folder_path*)
Create local folder hierarchy for the experiment tracking

Parameters

- **project_name** (*str*) – root name of the project
- **experiment_name** (*str*) – name of the particular experiment
- **experiment_timestamp** (*str*) – time stamp at the start of training
- **local_model_result_folder_path** (*str*) – root local path where project folder will be created

Returns path to the created experiment base folder

Return type str

static get_base_folder_paths(*project_name*, *experiment_name*, *experiment_timestamp*, *local_model_result_folder_path*)
Generate local folder hierarchy paths for the experiment tracking

Does not actually create the folders, just generates the folder paths

Parameters

- **project_name** (*str*) – root name of the project
- **experiment_name** (*str*) – name of the particular experiment
- **experiment_timestamp** (*str*) – time stamp at the start of training
- **local_model_result_folder_path** (*str*) – root local path where project folder will be created

Returns path to the main project folder, path to the particular experiment folder

Return type str, str

local_model_save

class aitoolbox.experiment.local_save.local_model_save.**AbstractLocalModelSaver**
Bases: abc.ABC

abstract save_model(*model*, *project_name*, *experiment_name*, *experiment_timestamp=None*, *epoch=None*, *iteration_idx=None*, *protect_existing_folder=True*)
Model saving method which all the model savers have to implement to give an expected API to other components

Parameters

- **model** (*keras.Model or dict*) – model representation. If used with PyTorch it is a simple dict under the hood. In the case of Keras training this would be the keras Model.
- **project_name** (*str*) – root name of the project
- **experiment_name** (*str*) – name of the particular experiment
- **experiment_timestamp** (*str or None*) – time stamp at the start of training
- **epoch** (*int or None*) – in which epoch the model is being saved
- **iteration_idx** (*int or None*) – at which training iteration the model is being saved
- **protect_existing_folder** (*bool*) – can override potentially already existing folder or not

Returns model_name, model_local_path

Return type (str, str)

class aitoolbox.experiment.local_save.local_model_save.**BaseLocalModelSaver**(*local_model_result_fol*
check-
point_model=False)

Bases: object

Base functionality for all the local model savers

Parameters

- **local_model_result_folder_path** (*str*) – root local path where project folder will be created
- **checkpoint_model** (*bool*) – if the model is coming from the mid-training checkpoint

create_experiment_local_models_folder(*project_name*, *experiment_name*, *experiment_timestamp*)

Creates experiment local folder hierarchy and place the 'models' folder in it

Parameters

- **project_name** (*str*) – root name of the project
- **experiment_name** (*str*) – name of the particular experiment
- **experiment_timestamp** (*str*) – time stamp at the start of training

Returns path to the created models folder in the experiment base folder

Return type str

class aitoolbox.experiment.local_save.local_model_save.**PyTorchLocalModelSaver**(*local_model_resul*
check-
point_model=Fals

Bases: *aitoolbox.experiment.local_save.local_model_save.AbstractLocalModelSaver*, *aitoolbox.experiment.local_save.local_model_save.BaseLocalModelSaver*

PyTorch experiment local model saver

Parameters

- **local_model_result_folder_path** (*str*) – root local path where project folder will be created
- **checkpoint_model** (*bool*) – if the model is coming from the mid-training checkpoint

save_model(*model*, *project_name*, *experiment_name*, *experiment_timestamp=None*, *epoch=None*, *iteration_idx=None*, *protect_existing_folder=True*)
Save the PyTorch model representation dict to the local drive

Parameters

- **model** (*dict*) – PyTorch model represented as a dict of weights, optimizer state and other necessary info.
- **project_name** (*str*) – root name of the project
- **experiment_name** (*str*) – name of the particular experiment
- **experiment_timestamp** (*str or None*) – time stamp at the start of training
- **epoch** (*int or None*) – in which epoch the model is being saved
- **iteration_idx** (*int or None*) – at which training iteration the model is being saved
- **protect_existing_folder** (*bool*) – can override potentially already existing folder or not

Returns model_name, model_local_path

Return type (str, str)

static check_model_dict_contents(*model*)
Check if PyTorch model save dict contains all the necessary elements for the training state reconstruction

Parameters **model** (*dict*) – PyTorch model represented as a dict of weights, optimizer state and other necessary info.

Raises **ValueError** –

Returns None

class aitoolbox.experiment.local_save.local_model_save.**KerasLocalModelSaver**(*local_model_result_fo*
check-
point_model=False)

Bases: *aitoolbox.experiment.local_save.local_model_save.AbstractLocalModelSaver*, *aitoolbox.experiment.local_save.local_model_save.BaseLocalModelSaver*

Keras experiment local model saver

Parameters

- **local_model_result_folder_path** (*str*) – root local path where project folder will be created
- **checkpoint_model** (*bool*) – if the model is coming from the mid-training checkpoint

save_model(*model*, *project_name*, *experiment_name*, *experiment_timestamp=None*, *epoch=None*, *iteration_idx=None*, *protect_existing_folder=True*)
Save the Keras model to the local drive

Parameters

- **model** (*keras.Model*) – Keras model
- **project_name** (*str*) – root name of the project
- **experiment_name** (*str*) – name of the particular experiment
- **experiment_timestamp** (*str or None*) – time stamp at the start of training

- **epoch** (*int or None*) – in which epoch the model is being saved
- **iteration_idx** (*int or None*) – at which training iteration the model is being saved
- **protect_existing_folder** (*bool*) – can override potentially already existing folder or not

Returns model_name, model_local_path

Return type (str, str)

class aitoolbox.experiment.local_save.local_model_save.**LocalSubOptimalModelRemover**(*metric_nam*
*num_best_*

Bases: object

Removes the tracked saved models which become suboptimal when new models are trained in subsequent epochs

Useful when interested in saving the limited local disk space, especially when dealing with large model which take a lot of disk space.

Parameters

- **metric_name** (*str*) – one of the metric names that will be calculated and will appear in the train_history dict in the TrainLoop
- **num_best_kept** (*int*) – number of best performing models which are kept when removing suboptimal model checkpoints

decide_if_remove_suboptimal_model(*history*, *new_model_dump_paths*)

Make decision if suboptimal model should be removed due to the introduction of the new and better model

Parameters

- **history** (aitoolbox.experiment.training_history.TrainingHistory) – training performance history
- **new_model_dump_paths** (*list*) – new saved models paths which will begin to be tracked

Returns None

static rm_suboptimal_model(*rm_model_paths*)

Utility to remove the file

Parameters **rm_model_paths** (*list*) – list of string paths

Returns None

local_results_save

class aitoolbox.experiment.local_save.local_results_save.**AbstractLocalResultsSaver**

Bases: abc.ABC

abstract save_experiment_results(*result_package*, *training_history*, *project_name*, *experiment_name*, *experiment_timestamp=None*, *save_true_pred_labels=False*, *protect_existing_folder=True*)

Single file results saving method which all the result savers have to implement to give an expected API

Parameters

- **result_package** (aitoolbox.experiment.result_package.abstract_result_packages.AbstractResultPackage) –
- **training_history** (aitoolbox.experiment.training_history.TrainingHistory) –
- **project_name** (*str*) – root name of the project
- **experiment_name** (*str*) – name of the particular experiment
- **experiment_timestamp** (*str or None*) – time stamp at the start of training
- **save_true_pred_labels** (*bool*) – should ground truth labels also be saved
- **protect_existing_folder** (*bool*) – can override potentially already existing folder or not

Returns

list of list with this format: [[results_file_name, results_file_local_path], ... [,]] Each file should be a new list specifying the file name and its full path

The first file path should be pointing to the main experiment results file.

Return type list

abstract save_experiment_results_separate_files(*result_package*, *training_history*, *project_name*, *experiment_name*, *experiment_timestamp*, *save_true_pred_labels=False*, *protect_existing_folder=True*)

Separate file results saving method which all the result savers have to implement to give an expected API

Parameters

- **result_package** (aitoolbox.experiment.result_package.abstract_result_packages.AbstractResultPackage) –
- **training_history** (aitoolbox.experiment.training_history.TrainingHistory) –
- **project_name** (*str*) – root name of the project
- **experiment_name** (*str*) – name of the particular experiment
- **experiment_timestamp** (*str or None*) – time stamp at the start of training
- **save_true_pred_labels** (*bool*) – should ground truth labels also be saved
- **protect_existing_folder** (*bool*) – can override potentially already existing folder or not

Returns

list of list with this format: [[results_file_name, results_file_local_path], ... [,]] Each file should be a new list specifying the file name and its full path

The first file path should be pointing to the main experiment results file.

Return type list

class aitoolbox.experiment.local_save.local_results_save.**BaseLocalResultsSaver**(*local_model_resu*, *file_format='pick*

Bases: object

Base functionality for all the local results savers

Parameters

- **local_model_result_folder_path** (*str*) – root local path where project folder will be created
- **file_format** (*str*) – pickle or json

create_experiment_local_folder_structure(*project_name*, *experiment_name*, *experiment_timestamp*)

Creates experiment local results folder hierarchy

Parameters

- **project_name** (*str*) – root name of the project
- **experiment_name** (*str*) – name of the particular experiment
- **experiment_timestamp** (*str*) – time stamp at the start of training

Returns experiment folder path

Return type str

static create_experiment_local_results_folder(*project_name*, *experiment_name*, *experiment_timestamp*, *local_model_result_folder_path*)

Creates experiment local results folder hierarchy

Parameters

- **project_name** (*str*) – root name of the project
- **experiment_name** (*str*) – name of the particular experiment
- **experiment_timestamp** (*str*) – time stamp at the start of training
- **local_model_result_folder_path** (*str*) – root local path where project folder will be created

Returns experiment results folder path (inside the experiment folder)

Return type str

static get_experiment_local_results_folder_paths(*project_name*, *experiment_name*, *experiment_timestamp*, *local_model_result_folder_path*)

Generates experiment local results folder hierarchy paths

Parameters

- **project_name** (*str*) – root name of the project
- **experiment_name** (*str*) – name of the particular experiment
- **experiment_timestamp** (*str*) – time stamp at the start of training
- **local_model_result_folder_path** (*str*) – root local path where project folder will be created

Returns project_dir_path, experiment_dir_path, experiment_results_dir_path

Return type str, str, str

save_file(*result_dict*, *file_name_w_type*, *file_local_path_w_type*)

Saves dict to file in desired format

Parameters

- **result_dict** (*dict*) – results dict
- **file_name_w_type** (*str*) – filename without the file extension at the end
- **file_local_path_w_type** (*str*) – file path without the file extension at the end

Returns saved file name, saved file path

Return type str, str

class aitoolbox.experiment.local_save.local_results_save.**LocalResultsSaver**(*local_model_result_fol*
file_format='pickle')

Bases: *aitoolbox.experiment.local_save.local_results_save.AbstractLocalResultsSaver*, *aitoolbox.experiment.local_save.local_results_save.BaseLocalResultsSaver*

Local model training results saver to local drive

Parameters

- **local_model_result_folder_path** (*str*) – root local path where project folder will be created
- **file_format** (*str*) – file format of the results file

save_experiment_results(*result_package*, *training_history*, *project_name*, *experiment_name*, *experiment_timestamp=None*, *save_true_pred_labels=False*, *protect_existing_folder=True*)

Saves all the experiment results into single local file

Parameters

- **result_package** (*aitoolbox.experiment.result_package.abstract_result_packages.AbstractResultPackage*) –
- **training_history** (*aitoolbox.experiment.training_history.TrainingHistory*) –
- **project_name** (*str*) – root name of the project
- **experiment_name** (*str*) – name of the particular experiment
- **experiment_timestamp** (*str or None*) – time stamp at the start of training
- **save_true_pred_labels** (*bool*) – should ground truth labels also be saved
- **protect_existing_folder** (*bool*) – can override potentially already existing folder or not

Returns

list of list with this format: [[results_file_path_inside_results_dir, results_file_local_path], ... [,]]
Each file should be a new list specifying the file name and its full path.

The first file path should be pointing to the main experiment results file.

Return type list

save_experiment_results_separate_files(*result_package*, *training_history*, *project_name*, *experiment_name*, *experiment_timestamp=None*, *save_true_pred_labels=False*, *protect_existing_folder=True*)

Saves the experiment results into separate local files

Parameters

- **result_package** (aitoolbox.experiment.result_package.abstract_result_packages.AbstractResultPackage) –
- **training_history** (aitoolbox.experiment.training_history.TrainingHistory) –
- **project_name** (*str*) – root name of the project
- **experiment_name** (*str*) – name of the particular experiment
- **experiment_timestamp** (*str or None*) – time stamp at the start of training
- **save_true_pred_labels** (*bool*) – should ground truth labels also be saved
- **protect_existing_folder** (*bool*) – can override potentially already existing folder or not

Returns

list of list with this format: [[results_file_path_inside_results_dir, results_file_local_path], ... [,]]
Each file should be a new list specifying the file name and its full path.

The first file path should be pointing to the main experiment results file.

Return type list

6.1.2.1.4 result_package

Submodules

abstract_result_packages

class aitoolbox.experiment.result_package.abstract_result_packages.**AbstractResultPackage**(*pk*
st
np
***

Bases: abc.ABC

Base Result package used to derive specific result packages from

Functions which the user should potentially override in a specific result package:

- prepare_results_dict()
- list_additional_results_dump_paths()
- set_experiment_dir_path_for_additional_results()

Parameters

- **pkg_name** (*str or None*) – result package name used just for clarity
- **strict_content_check** (*bool*) – should just print warning or raise the error and crash
- **np_array** (*bool or str*) – how the inputs should be handled. Should the package try to automatically guess or you want to manually decide whether to leave the inputs as they are or convert them to np.array. Possible options: True, False, 'auto' Be slightly careful with 'auto' as it sometimes doesn't work so it is preferable to explicitly use True/False
- ****kwargs** (*dict*) – additional package_metadata for the result package

abstract prepare_results_dict()
Perform result package building

Mostly this consists of executing calculation of selected performance metrics and returning their result dicts. If you want to use multiple performance metrics you have to combine them in the single self.results_dict at the end by doing this:

return {**metric_dict_1, **metric_dict_2}

Returns calculated result dict

Return type dict

prepare_result_package(*y_true*, *y_predicted*, *hyperparameters=None*, ***kwargs*)
Prepares the result package by taking labels and running them through the specified metrics

This function is automatically called from the torchtrain callbacks to evaluate the provided callback. The main feature of this function is the call to the user-derived prepare_results_dict() function of the implemented result package where the metrics evaluation logic is implemented.

Parameters

- **y_true** (*numpy.array or list*) – ground truth targets
- **y_predicted** (*numpy.array or list*) – predicted targets
- **hyperparameters** (*dict or None*) – dictionary filled with the set hyperparameters
- ****kwargs** (*dict*) – additional results for the result package

Returns None

static auto_y_input_array_convert(*y_array*)
Try to automatically decide if array should be left as it is or convert to np.array

Not working in all the situations so relying on it at all times is not recommended. Especially for costly experiments rely rather on your own judgement and explicitly define if np.array conversion is needed.

TODO: make it smarter so 'auto' option can be used more often

Parameters **y_array** (*list*) –

Returns

Return type list or numpy.array

get_results()
Get calculated results dict

Returns results dict

Return type dict

get_hyperparameters()
Get hyperparameters in a dict form

Returns hyperparameters dict

Return type dict

get_additional_results_dump_paths()
Return paths to the additional results which are stored to local drive when the package is evaluated

For example if package plots attention heatmaps and saves pictures to disk, this function will return paths to these picture files. This is achieved via the call to the use implemented function list_additional_results_dump_paths().

Returns

list of lists of string paths if it is not None. Each element of the list should be list of: [[results_file_name, results_file_local_path], ... [,]]

Return type list or None

list_additional_results_dump_paths()

Specify the list of meta data files you also want to save & upload to s3 during the experiment saving procedure

By default there are no additional files that are saved as the return is None. If you want to save your specific additional files produced during the training procedure, then override this method specifying the file paths.

If you want to save a whole folder of files, use zip_additional_results_dump() function to zip it into a single file and save this zip instead.

The specified files are any additional data you would want to include into the experiment folder in addition to the model save files and performance evaluation report files. For example a zip of attention heatmap pictures in the machine translation projects.

Returns

list of lists of string paths if it is not None. Each element of the list should be list of: [[results_file_name, results_file_local_path], ... [,]]

Return type list or None

set_experiment_dir_path_for_additional_results(*project_name*, *experiment_name*, *experiment_timestamp*, *local_model_result_folder_path*)

Set experiment folder path after potential timestamps have already been generated.

Experiment folder setting for additional metadata results output is needed only in certain result packages, for example in QuestionAnswerResultPackage where the self.output_text_dir initially has only the name of the folder where the results text predictions for each example should be stored. This function when implemented reforms the folder name so that it becomes a full path placing the folder inside the experiment folder (for which the timestamp at the start of train loop is needed).

Another use of this function is in MachineTranslationResultPackage where the attention heatmap pictures are stored as additional metadata results.

As can be seen from the fact that the train loop mechanism is mentioned, this method's functionality is primarily used for PyTorch experiments.

Parameters

- **project_name** (*str*) – root name of the project
- **experiment_name** (*str*) – name of the particular experiment
- **experiment_timestamp** (*str*) – time stamp at the start of training
- **local_model_result_folder_path** (*str*) – root local path where project folder will be created

Returns None

qa_check_hyperparameters_dict()

Quality check the hyperparams dict

Returns None

qa_check_additional_results_dump_paths()

Quality check the additional results path

Returns None

warn_about_result_data_problem(*msg*)
Generic function for writing out warnings

Either just printing out the warning or throw the error exception.

Parameters **msg** (*str*) – warning message either printed or written in the raised error

Raises **ValueError** –

Returns None

static zip_additional_results_dump(*source_dir_path*, *zip_path*)
Utility function for zipping a folder into .zip archive

Parameters

- **source_dir_path** (*str*) – path to the folder that is going to be zipped
- **zip_path** (*str*) – specify the path of the zip file which will be created

Returns the full path to the produced zip file (with the .zip extension appended)

Return type str

__add__(*other*)
Concatenate result packages

Combines results from both result packages into a single one.

Parameters **other** (*aitoolbox.experiment.result_package.abstract_result_packages.AbstractResultPackage or dict*) – another result package to be concatenated

Returns merged result package

Return type aitoolbox.experiment.result_package.MultipleResultPackageWrapper

__radd__(*other*)
Concatenate result package

Parameters **other** (*aitoolbox.experiment.result_package.abstract_result_packages.AbstractResultPackage or dict*) – another result package to be concatenated

Returns merged result package

Return type aitoolbox.experiment.result_package.MultipleResultPackageWrapper

add_merge_multi_pkg_wrap(*other_object*)
Result package merge

Parameters **other_object** (*aitoolbox.experiment.result_package.abstract_result_packages.AbstractResultPackage or dict*) – another result package to be merged with the current package

Returns

merged result package

Return type *aitoolbox.experiment.result_package.abstract_result_packages.MultipleResultPackageWrapper*

static _create_other_object_pkg(*other_object*)
Util to deep copy and wrap results into the simple result package

Parameters **other_object** (aitoolbox.experiment.result_package.abstract_result_packages.AbstractResultPackage *or dict*) – results package or results dict

Returns

deep copy of results wrapped in the simple result package

Return type AbstractResultPackage | MultipleResultPackageWrapper

__iadd__(*other*)
Append result package

Parameters **other** (aitoolbox.experiment.result_package.abstract_result_packages.AbstractResultPackage *or dict*) – another result package to be appended to the current package

Returns merged result package

Return type *aitoolbox.experiment.result_package.abstract_result_packages.AbstractResultPackage*

add_merge_dicts(*other*)
Append result package to the current one

Parameters **other** (aitoolbox.experiment.result_package.abstract_result_packages.AbstractResultPackage *or dict*) – another result package to be appended to the current package

Returns merged result package

Return type *aitoolbox.experiment.result_package.abstract_result_packages.AbstractResultPackage*

_merge_dicts(*other_results_dict*)
Results dict merge util

Parameters **other_results_dict** (*dict*) – another results dict to be added to the results dict in the current result package

Returns merged result package

Return type *aitoolbox.experiment.result_package.abstract_result_packages.AbstractResultPackage*

warn_if_results_dict_not_defined()

class aitoolbox.experiment.result_package.abstract_result_packages.**PreCalculatedResultPacka**

Bases: *aitoolbox.experiment.result_package.abstract_result_packages.AbstractResultPackage*

Result package which doesn't have any evaluation logic but just accepts pre-calculated results dict

Parameters

- **results_dict** (*dict*) – pre-calculated results dict
- **strict_content_check** (*bool*) – should just print warning or raise the error and crash
- ****kwargs** (*dict*) – result package additional meta-data

prepare_results_dict()
Perform result package building

Mostly this consists of executing calculation of selected performance metrics and returning their result dicts. If you want to use multiple performance metrics you have to combine them in the single self.results_dict at the end by doing this:

return {**metric_dict_1, **metric_dict_2}

Returns calculated result dict

Return type dict

class aitoolbox.experiment.result_package.abstract_result_packages.**MultipleResultPackageWra**

Bases: *aitoolbox.experiment.result_package.abstract_result_packages.AbstractResultPackage*

Wrapper result package which combines multiple evaluated result packages into a single result package

Parameters

- **strict_content_check** (*bool*) – should just print warning or raise the error and crash
- ****kwargs** (*dict*) – result package additional meta-data

prepare_result_package(*result_packages*, *hyperparameters=None*, ***kwargs*)

Prepares the multiple result package by merging the results from both result packages

Parameters

- **result_packages** (*list*) – list of result packages where each of them is object inherited from aitoolbox.experiment.result_package.abstract_result_packages.AbstractResultPackage. If you want to add raw results in dict form, this dict first needs to be wrapped into aitoolbox.experiment.result_package.abstract_result_packages.PreCalculatedResultPackage to satisfy the result package object requirement.
- **hyperparameters** (*dict or None*) – hyperparameters dict
- ****kwargs** – result package additional meta-data

Returns None

prepare_results_dict()

Perform result package building

Mostly this consists of executing calculation of selected performance metrics and returning their result dicts. If you want to use multiple performance metrics you have to combine them in the single self.results_dict at the end by doing this:

return {**metric_dict_1, **metric_dict_2}

Returns calculated result dict

Return type dict

get_additional_results_dump_paths()

Return paths to the additional results which are stored to local drive when the package is evaluated

For example if package plots attention heatmaps and saves pictures to disk, this function will return paths to these picture files. This is achieved via the call to the use implemented function list_additional_results_dump_paths().

Returns

list of lists of string paths if it is not None. Each element of the list should be list of: [[results_file_name, results_file_local_path], ... [,]]

Return type list or None

__len__()
Get number of result packages inside the multi result package wrapper

Returns number of result packages inside this multi package wrapper

Return type int

add_merge_multi_pkg_wrap(*other_object*)
Result package merge

Parameters **other_object** (aitoolbox.experiment.result_package.abstract_result_packages.AbstractResultPackage *or dict*) – another result package to be merged with the current package

Returns

merged result package

Return type *aitoolbox.experiment.result_package.abstract_result_packages.MultipleResultPackageWrapper*

basic_packages

class aitoolbox.experiment.result_package.basic_packages.**GeneralResultPackage**(*metrics_list*, *strict_content_che*, ***kwargs*)

Bases: *aitoolbox.experiment.result_package.abstract_result_packages.AbstractResultPackage*

Result package executing given list of metrics

Parameters

- **metrics_list** (*list*) – List of objects which are inherited from aitoolbox.experiment.core_metrics.BaseMetric.AbstractBaseMetric
- **strict_content_check** (*bool*) – should just print warning or raise the error and crash
- ****kwargs** (*dict*) – additional package_metadata for the result package

prepare_results_dict()
Perform result package building

Mostly this consists of executing calculation of selected performance metrics and returning their result dicts. If you want to use multiple performance metrics you have to combine them in the single self.results_dict at the end by doing this:

return {**metric_dict_1, **metric_dict_2}

Returns calculated result dict

Return type dict

qa_check_metrics_list()

class aitoolbox.experiment.result_package.basic_packages.**BinaryClassificationResultPackage**

Bases: *aitoolbox.experiment.result_package.abstract_result_packages.AbstractResultPackage*

Binary classification task result package

Evaluates the following metrics: accuracy, ROC-AUC, PR-AUC and F1 score

Parameters

- **positive_class_thresh** (*float or None*) – predicted probability positive class threshold
- **strict_content_check** (*bool*) – should just print warning or raise the error and crash
- ****kwargs** (*dict*) – additional package_metadata for the result package

prepare_results_dict()

Perform result package building

Mostly this consists of executing calculation of selected performance metrics and returning their result dicts. If you want to use multiple performance metrics you have to combine them in the single self.results_dict at the end by doing this:

return {**metric_dict_1, **metric_dict_2}

Returns calculated result dict

Return type dict

class aitoolbox.experiment.result_package.basic_packages.**ClassificationResultPackage**(*strict_co*
***kwarg*

Bases: *aitoolbox.experiment.result_package.abstract_result_packages.AbstractResultPackage*

Multi-class classification result package

Evaluates the accuracy of the predictions. Without Precision-Recall metric which is available only for binary classification problems.

Parameters

- **strict_content_check** (*bool*) – should just print warning or raise the error and crash
- ****kwargs** (*dict*) – additional package_metadata for the result package

prepare_results_dict()

Perform result package building

Mostly this consists of executing calculation of selected performance metrics and returning their result dicts. If you want to use multiple performance metrics you have to combine them in the single self.results_dict at the end by doing this:

return {**metric_dict_1, **metric_dict_2}

Returns calculated result dict

Return type dict

class aitoolbox.experiment.result_package.basic_packages.**RegressionResultPackage**(*strict_content_check=False*, ***kwargs*)

Bases: *aitoolbox.experiment.result_package.abstract_result_packages.AbstractResultPackage*

Regression task result package

Evaluates MSE and MAE metrics.

Parameters

- **strict_content_check** (*bool*) – should just print warning or raise the error and crash
- ****kwargs** (*dict*) – additional package_metadata for the result package

prepare_results_dict()

Perform result package building

Mostly this consists of executing calculation of selected performance metrics and returning their result dicts. If you want to use multiple performance metrics you have to combine them in the single self.results_dict at the end by doing this:

return {**metric_dict_1, **metric_dict_2}

Returns calculated result dict

Return type dict

6.1.2.1.5 result_reporting

Submodules

hyperparam_reporter

class aitoolbox.experiment.result_reporting.hyperparam_reporter.**HyperParamSourceReporter**(*project_name*, *experiment_name*, *experiment_timestamp*, *local_model_result_folder_path*)

Bases: object

Writer of selected hyperparameters to human-readable text file on disk

Parameters

- **project_name** (*str*) – root name of the project
- **experiment_name** (*str*) – name of the particular experiment
- **experiment_timestamp** (*str*) – time stamp of the training start
- **local_model_result_folder_path** (*str*) – root local path where project folder will be created

save_hyperparams_to_text_file(*hyperparams*, *sort_names=False*)
Save hyperparameters dict into text file on disk

Parameters

- **hyperparams** (*dict*) – hyper-parameters listed in the dict
- **sort_names** (*bool*) – should presented hyper-param names be listed alphabetically

Returns path to the saved hyper-param text file

Return type str

copy_to_cloud_storage(*local_hyperparams_file_path*, *cloud_saver*, *file_name=None*)
Copy saved text local file into cloud storage

Parameters

- **local_hyperparams_file_path** (*str*) – path to hyperparams file stored on local disk. File to be uploaded to cloud
- **cloud_saver** (*BaseModelSaver or BaseResultsSaver or BaseModelGoogleStorageSaver or BaseResultsGoogleStorageSaver*) –
- **file_name** (*str or None*) – manually specify the file name to be saved to the cloud instead of taking the default from self.file_name

Returns path where the file was saved in the cloud storage

Return type str

save_experiment_python_file(*hyperparams*)
Saves the python experiment file to the project folder

Python experiment file is file in which the main training procedure is defined. File from which the TrainLoop is executed

Parameters **hyperparams** (*dict*) – hyper-parameters listed in the dict. In order for this function to work, the dict needs to include *experiment_file_path* key.

Returns path to the saved main python experiment file

Return type str

save_experiment_source_files(*hyperparams*)
Saves all the experiment source files into single source code zip

Parameters **hyperparams** (*dict*) – hyper-parameters listed in the dict. In order for this function to work, the dict needs to include *source_dirs_paths* key.

Returns path to the saved experiment source code zip

Return type str

report_generator

class aitoolbox.experiment.result_reporting.report_generator.**TrainingHistoryPlotter**(*experimen*
Bases: object

Plot the calculated performance metrics in the training history

Parameters **experiment_results_local_path** (*str*) – path to the main experiment results folder on the local drive

generate_report(*training_history*, *plots_folder_name='plots'*, *file_format='png'*)
Plot all the currently present performance result in the training history

Every plot shows the progression of a single performance metric over the epochs.

Parameters

- **training_history** (aitoolbox.experiment.training_history.TrainingHistory) – TrainLoop training history
- **plots_folder_name** (*str*) – local dir name where the plots should be saved
- **file_format** (*str*) – output file format. Can be either 'png' for saving separate images or 'pdf' for combining all the plots into a single pdf file.

Returns list of saved plot paths

Return type list

plot_png(*training_history*, *plots_local_folder_path*, *plots_folder_name*)

plot_pdf(*training_history*, *plots_local_folder_path*, *plots_file_name*)

static generate_plots(*training_history*)

static plot_performance_curve(*metric_name*, *result_history*)
Plot the performance of a selected calculated metric over the epochs

Parameters

- **metric_name** (*str or int*) – name of plotted metric
- **result_history** (*list or np.array*) – results history for the selected metric

Returns plot figure

Return type plt.figure

class aitoolbox.experiment.result_reporting.report_generator.**TrainingHistoryWriter**(*experiment*
Bases: object

Write the calculated performance metrics in the training history into human-readable text file

Parameters **experiment_results_local_path** (*str or None*) – path to the main experiment results folder on the local drive

generate_report(*training_history*, *epoch*, *file_name*, *results_folder_name=''*, *file_format='txt'*)
Write all the currently present performance result in the training history into the text file

Parameters

- **training_history** (aitoolbox.experiment.training_history.TrainingHistory) –
- **epoch** (*int*) – current epoch
- **file_name** (*str*) – output text file name

- **results_folder_name** (*str*) – results folder path where the report file will be located
- **file_format** (*str*) – output file format. Can be either 'txt' human readable output or 'tsv' for a tabular format or 'csv' for comma separated format.

Returns file name/path inside the experiment folder, local file_path

Return type str, str

static write_txt(*training_history*, *epoch*, *file_path*)

write_csv_tsv(*training_history*, *epoch*, *file_path*, *delimiter*)

class aitoolbox.experiment.result_reporting.report_generator.**GradientPlotter**(*experiment_grad_re*

Bases: object

Plot the gradient distributions for model's layers

Parameters **experiment_grad_results_local_path** (*str*) – path to the main experiment results folder on the local drive

generate_report(*model_layer_gradients*, *grad_plots_folder_name='grad_plots'*, *file_format='png'*)

Plot all the gradient distributions for the layers in the model

Parameters

- **model_layer_gradients** (*list*) – list of model's gradients
- **grad_plots_folder_name** (*str*) – name of the folder where gradient distribution plots will be saved
- **file_format** (*str*) – output file format. Can be either 'png' for saving separate images or 'pdf' for combining all the plots into a single pdf file.

Returns list of saved plot paths: [file_path_in_cloud_grad_results_dir, local_file_path]

Return type list

plot_png(*model_layer_gradients*, *grad_plots_local_folder_path*, *plots_folder_name*)

plot_pdf(*model_layer_gradients*, *plots_local_folder_path*, *plots_file_name*)

static generate_dist_plots(*model_layer_gradients*, *layer_names=None*)

static plot_gradient_distribution(*gradients*, *layer_name*)

Plot and save to file the distribution of the single layer's gradients

Parameters

- **gradients** (*list or np.array*) – a flattened list of gradients from a single layer
- **layer_name** (*str or int*) – name or index of the layer

Returns plot figure

Return type plt.figure

6.1.2.2 Submodules

6.1.2.2.1 experiment_saver

class aitoolbox.experiment.experiment_saver.**AbstractExperimentSaver**

Bases: abc.ABC

abstract save_experiment(*model*, *result_package*, *training_history*, *experiment_timestamp=None*, *save_true_pred_labels=False*, *separate_files=False*, *protect_existing_folder=True*)

Method which all the experiment savers need to implement which instructs how the experiment should be saved

Parameters

- **model** –
- **result_package** (*aitoolbox.ExperimentSave.result_package.AbstractResultPackage*) –
- **training_history** (aitoolbox.experiment.training_history.TrainingHistory) –
- **experiment_timestamp** (*str*) – time stamp of the training start
- **save_true_pred_labels** (*bool*) – should ground truth labels also be saved
- **separate_files** (*bool*) – should the results be saved in separate pickle files or should all of the results be batched together in a single results file
- **protect_existing_folder** (*bool*) – can override potentially already existing folder or not

Returns string paths where the experiment files were saved

Return type list

class aitoolbox.experiment.experiment_saver.**BaseFullExperimentSaver**(*model_saver*, *results_saver*, *project_name*, *experiment_name*)

Bases: *aitoolbox.experiment.experiment_saver.AbstractExperimentSaver*

Base full experiment saver functionality used by the underlying experiment saver derivations

Parameters

- **model_saver** (aitoolbox.cloud.AWS.model_save.AbstractModelSaver) – selected saver used for model saving
- **results_saver** (aitoolbox.cloud.AWS.results_save.AbstractResultsSaver) – selected saver used for results save
- **project_name** (*str*) – root name of the project
- **experiment_name** (*str*) – name of the particular experiment

save_experiment(*model*, *result_package*, *training_history*, *experiment_timestamp=None*, *save_true_pred_labels=False*, *separate_files=False*, *protect_existing_folder=True*)

Save the experiment snapshot formed out of the model and model's results

Parameters

- **model** (*dict or keras.Model*) – model representation. If used with PyTorch it is a simple dict under the hood. In the case of Keras training this would be the keras Model.
- **result_package** (aitoolbox.experiment.result_package.abstract_result_packages.AbstractResultPackage) –
- **training_history** (aitoolbox.experiment.training_history.TrainingHistory) –
- **experiment_timestamp** (*str*) – time stamp at the start of training
- **save_true_pred_labels** (*bool*) – should ground truth labels also be saved
- **separate_files** (*bool*) – should the results be saved in separate pickle files or should all of the results be batched together in a single results file
- **protect_existing_folder** (*bool*) – can override potentially already existing folder or not

Returns cloud_model_path, cloud_results_path

Return type (str, str)

class aitoolbox.experiment.experiment_saver.**BaseFullExperimentS3Saver**(*model_saver*, *project_name*, *experiment_name*, *bucket_name='model-result'*, *cloud_dir_prefix=''*, *local_model_result_folder_path*

Bases: *aitoolbox.experiment.experiment_saver.BaseFullExperimentSaver*

Base experiment saver implementing the S3 saving functionality

This is used by the underlying experiment S3 saver derivations

Parameters

- **model_saver** (aitoolbox.cloud.AWS.model_save.AbstractModelSaver) – selected cloud model saver implementing the saving logic for the desired cloud storage provider file saving
- **project_name** (*str*) – root name of the project
- **experiment_name** (*str*) – name of the particular experiment
- **bucket_name** (*str*) – name of the bucket in the cloud storage
- **cloud_dir_prefix** (*str*) – path to the folder inside the bucket where the experiments are going to be saved
- **local_model_result_folder_path** (*str*) – root local path where project folder will be created

class aitoolbox.experiment.experiment_saver.**FullPyTorchExperimentS3Saver**(*project_name*, *experiment_name*, *bucket_name='model-result'*, *cloud_dir_prefix=''*, *local_model_result_folder_p*

Bases: *aitoolbox.experiment.experiment_saver.BaseFullExperimentS3Saver*

S3 saver for PyTorch experiments

Parameters

- **project_name** (*str*) – root name of the project
- **experiment_name** (*str*) – name of the particular experiment
- **bucket_name** (*str*) – name of the bucket in the cloud storage
- **cloud_dir_prefix** (*str*) – path to the folder inside the bucket where the experiments are going to be saved
- **local_model_result_folder_path** (*str*) – root local path where project folder will be created

class aitoolbox.experiment.experiment_saver.**FullKerasExperimentS3Saver**(*project_name*, *experiment_name*, *bucket_name='model-result'*, *cloud_dir_prefix=''*, *local_model_result_folder_pat*

Bases: *aitoolbox.experiment.experiment_saver.BaseFullExperimentS3Saver*

S3 saver for Keras experiments

Parameters

- **project_name** (*str*) – root name of the project
- **experiment_name** (*str*) – name of the particular experiment
- **bucket_name** (*str*) – name of the bucket in the cloud storage
- **cloud_dir_prefix** (*str*) – path to the folder inside the bucket where the experiments are going to be saved
- **local_model_result_folder_path** (*str*) – root local path where project folder will be created

class aitoolbox.experiment.experiment_saver.**BaseFullExperimentGoogleStorageSaver**(*model_saver*, *project_name*, *experiment_name*, *bucket_name='result'*, *cloud_dir_prefix*, *local_model_result_folder_path*)

Bases: *aitoolbox.experiment.experiment_saver.BaseFullExperimentSaver*

Base experiment saver implementing the Google Storage saving functionality

This is used by the underlying experiment Google Storage saver derivations

Parameters

- **model_saver** (aitoolbox.cloud.AWS.model_save.AbstractModelSaver) – selected cloud model saver implementing the saving logic for the desired cloud storage provider file saving
- **project_name** (*str*) – root name of the project
- **experiment_name** (*str*) – name of the particular experiment
- **bucket_name** (*str*) – name of the bucket in the cloud storage
- **cloud_dir_prefix** (*str*) – path to the folder inside the bucket where the experiments are going to be saved
- **local_model_result_folder_path** (*str*) – root local path where project folder will be created

class aitoolbox.experiment.experiment_saver.**FullPyTorchExperimentGoogleStorageSaver**(*project_name*, *experiment_name*, *bucket_name='result'*, *cloud_dir_prefix*, *local_model_result_folder_path*)

Bases: *aitoolbox.experiment.experiment_saver.BaseFullExperimentGoogleStorageSaver*

Google Storage saver for PyTorch experiments

Parameters

- **project_name** (*str*) – root name of the project
- **experiment_name** (*str*) – name of the particular experiment
- **bucket_name** (*str*) – name of the bucket in the cloud storage
- **cloud_dir_prefix** (*str*) – path to the folder inside the bucket where the experiments are going to be saved
- **local_model_result_folder_path** (*str*) – root local path where project folder will be created

class aitoolbox.experiment.experiment_saver.**FullKerasExperimentGoogleStorageSaver**(*project_nam ex- per- i- ment_name, bucket_name result', cloud_dir_p lo- cal_model_r*

Bases: *aitoolbox.experiment.experiment_saver.BaseFullExperimentGoogleStorageSaver*

Google Storage saver for Keras experiments

Parameters

- **project_name** (*str*) – root name of the project
- **experiment_name** (*str*) – name of the particular experiment
- **bucket_name** (*str*) – name of the bucket in the cloud storage
- **cloud_dir_prefix** (*str*) – path to the folder inside the bucket where the experiments are going to be saved
- **local_model_result_folder_path** (*str*) – root local path where project folder will be created

6.1.2.2.2 local_experiment_saver

class aitoolbox.experiment.local_experiment_saver.**BaseFullExperimentLocalSaver**(*model_saver, project_name, ex- per- i- ment_name, lo- cal_model_resul*

Bases: *aitoolbox.experiment.experiment_saver.AbstractExperimentSaver*

Base functionality class common to all the full experiment local saver derivations

Parameters

- **model_saver** (aitoolbox.experiment.local_save.local_model_save.AbstractLocalModelSaver) – selected model saver implementing the saving logic for the desired framework
- **project_name** (*str*) – root name of the project
- **experiment_name** (*str*) – name of the particular experiment
- **local_model_result_folder_path** (*str*) – root local path where project folder will be created

save_experiment(*model*, *result_package*, *training_history*, *experiment_timestamp=None*, *save_true_pred_labels=False*, *separate_files=False*, *protect_existing_folder=True*)

Save the experiment with the provided model saver

Parameters

- **model** (*dict or keras.Model*) – model representation. If used with PyTorch it is a simple dict under the hood. In the case of Keras training this would be the keras Model.
- **result_package** (aitoolbox.experiment.result_package.abstract_result_packages.AbstractResultPackage) – selected result package which will be evaluated to produce the performance results
- **training_history** (aitoolbox.experiment.training_history.TrainingHistory) –
- **experiment_timestamp** (*str*) – time stamp at the start of training
- **save_true_pred_labels** (*bool*) – should ground truth labels also be saved
- **separate_files** (*bool*) – should the results be saved in separate pickle files or should all of the results be batched together in a single results file
- **protect_existing_folder** (*bool*) – can override potentially already existing folder or not

Returns local model and results paths

Return type list

class aitoolbox.experiment.local_experiment_saver.**FullPyTorchExperimentLocalSaver**(*project_nam ex- per- i- ment_name, lo- cal_model_r*

Bases: *aitoolbox.experiment.local_experiment_saver.BaseFullExperimentLocalSaver*

PyTorch local experiment saver

Parameters

- **project_name** (*str*) – root name of the project
- **experiment_name** (*str*) – name of the particular experiment
- **local_model_result_folder_path** (*str*) – root local path where project folder will be created

class aitoolbox.experiment.local_experiment_saver.**FullKerasExperimentLocalSaver**(*project_name, ex- per- i- ment_name, lo- cal_model_resu*

Bases: *aitoolbox.experiment.local_experiment_saver.BaseFullExperimentLocalSaver*

Keras local experiment saver

Parameters

- **project_name** (*str*) – root name of the project
- **experiment_name** (*str*) – name of the particular experiment
- **local_model_result_folder_path** (*str*) – root local path where project folder will be created

6.1.2.2.3 training_history

class aitoolbox.experiment.training_history.**TrainingHistory**(*has_validation=True*, *strict_content_check=False*)

Bases: object

Training history abstraction adding specific functionality to the simple dict

In many ways the object can be used with the same API as a normal python dict. However, for the need of tracking performance in the TrainLoop TrainingHistory offers additional functions handling the input, output and quality assurance of the stored results.

Parameters

- **has_validation** – if train history should by default include 'val_loss'. This is needed when train loops by default evaluate loss on validation set when such a set is available.
- **strict_content_check** (*bool*) – should just print warning or raise the error and crash in case of found (quality) problems

insert_single_result_into_history(*metric_name*, *metric_result*)

Insert a key-value formatted result into the training history

Parameters

- **metric_name** (*str*) – name of the metric to be stored.
- **metric_result** (*float or dict*) – metric performance result to be stored.

get_train_history()

Returns the whole train history dict in its original form without any transformations

Returns training history dict

Return type dict

get_train_history_dict(*flatten_dict=False*)

Returns QA-ed and optionally flattened training history dict

Parameters **flatten_dict** (*bool*) – should the returned training history dict be flattened. So no nested dicts of dicts. The keys of the nested dicts will we "_" concatenated and moved into the single level dict.

Returns training history dict

Return type dict

wrap_pre_prepared_history(*history*)

Wrap existing history dict into the TrainingHistory object

Parameters **history** (*dict*) – training history base dict

Returns self

Return type *TrainingHistory*

Examples

Expected history dict to be wrapped:

```
history = {
    'val_loss': [2.2513437271118164, 2.1482439041137695, 2.0187528133392334,␣
↪1.7953970432281494,
                 1.5492324829101562, 1.715561032295227, 1.631982684135437, 1.
↪3721977472305298,
                 1.039527416229248, 0.9796673059463501],
    'val_acc': [0.25999999046325684, 0.36000001430511475, 0.5, 0.
↪5400000214576721, 0.5400000214576721,
                0.5799999833106995,  0.46000000834465027, 0.699999988079071,␣
↪0.7599999904632568,
                0.7200000286102295],
    'loss': [2.3088033199310303, 2.2141530513763428, 2.113713264465332, 1.
↪912109375, 1.666761875152588,
             1.460097312927246, 1.6031768321990967, 1.534214973449707, 1.
↪1710081100463867,
             0.8969314098358154],
    'acc': [0.07999999821186066, 0.33000001311302185, 0.3100000023841858, 0.
↪5299999713897705,
            0.5799999833106995, 0.6200000047683716, 0.4300000071525574, 0.
↪5099999904632568,
            0.6700000166893005, 0.7599999904632568]
}
```

qa_check_history_records()
Quality check history

Returns None

warn_about_result_data_problem(*msg*)

keys()

items()

add_history_dict(*other*)
Add another training history dict to this training history

Parameters **other** (*dict*) – another training history dict

Returns None

6.1.3 cloud

6.1.3.1 Subpackages

6.1.3.1.1 AWS

Submodules

data_access

class aitoolbox.cloud.AWS.data_access.**BaseDataSaver**(*bucket_name='model-result'*)
Bases: object

Base class implementing S3 file saving logic

Parameters **bucket_name** (*str*) – S3 bucket into which the files will be saved

save_file(*local_file_path*, *cloud_file_path*)
Save / upload file on local drive to the AWS S3

Parameters

- **local_file_path** (*str*) – path to the file on the local drive
- **cloud_file_path** (*str*) – destination where the file will be saved on S3 inside the specified bucket

Returns None

save_folder(*local_folder_path*, *cloud_folder_path*)
Save / upload the contents of the local folder on the local drive to AWS S3

This function uploads the *contents inside* the provided local folder. If the encapsulating folder should also be created on the S3, specify the folder name at the end of the `cloud_folder_path`.

For example if:

```
local_folder_path = '~/bla/my_folder'
```

and we want to have the content of *my_folder* also placed into the folder *my_folder on S3* then append *my_folder* at the end of the cloud_folder_path:

```
cloud_folder_path = 'cloud_bla/my_folder'
```

Parameters

- **local_folder_path** (*str*) – local path to the folder which should be uploaded
- **cloud_folder_path** (*str*) – destination path on S3 where the folder and its content should be uploaded

Returns None

class aitoolbox.cloud.AWS.data_access.**BaseDataLoader**(*bucket_name='dataset-store'*, *local_base_data_folder_path='~/project/data'*)

Bases: `object`

Base class implementing S3 file downloading logic

Parameters

- **bucket_name** (*str*) – S3 bucket from which the files will be downloaded
- **local_base_data_folder_path** (*str*) – local main experiment saving folder

load_file(*cloud_file_path*, *local_file_path*)
Download the file AWS S3 to the local drive

Parameters

- **cloud_file_path** (*str*) – location where the file is saved on S3 inside the specified bucket
- **local_file_path** (*str*) – destination path where the file will be downloaded to the local drive

Returns None

exists_local_data_folder(*data_folder_name*, *protect_local_folder=True*)
Check if a specific folder exists in the base data folder

For example, Squad dataset folder inside /data folder, or pretrained_models folder inside /model_results folder

Parameters

- **data_folder_name** (*str*) –
- **protect_local_folder** (*bool*) –

Returns

Return type bool

preproc_dataset_available(*preproc_dataset_name*)

class aitoolbox.cloud.AWS.data_access.**AbstractDatasetFetcher**
Bases: abc.ABC

abstract fetch_dataset(*dataset_name=None*, *protect_local_folder=True*)

Parameters

- **dataset_name** (*str or None*) –
- **protect_local_folder** (*bool*) –

Returns None

class aitoolbox.cloud.AWS.data_access.**SQuAD2DatasetFetcher**(*bucket_name='dataset-store'*, *local_dataset_folder_path='~/project/data'*)
Bases: *aitoolbox.cloud.AWS.data_access.AbstractDatasetFetcher*, *aitoolbox.cloud.AWS.data_access.BaseDataLoader*

Parameters

- **bucket_name** (*str*) –
- **local_dataset_folder_path** (*str*) –

fetch_dataset(*dataset_name=None*, *protect_local_folder=True*)

Parameters

- **dataset_name** (*None*) – no effect here
- **protect_local_folder** (*bool*) –

Returns None

class aitoolbox.cloud.AWS.data_access.**QAngarooDatasetFetcher**(*bucket_name='dataset-store'*, *local_dataset_folder_path='~/project/data'*)
Bases: *aitoolbox.cloud.AWS.data_access.AbstractDatasetFetcher*, *aitoolbox.cloud.AWS.data_access.BaseDataLoader*

Parameters

- **bucket_name** (*str*) –
- **local_dataset_folder_path** (*str*) –

fetch_dataset(*dataset_name=None*, *protect_local_folder=True*)

Parameters

- **dataset_name** (*str or None*) – possible options: medhop, wikihop or None
- **protect_local_folder** (*bool*) –

Returns None

class aitoolbox.cloud.AWS.data_access.**CNNDailyMailDatasetFetcher**(*bucket_name='dataset-store'*, *local_dataset_folder_path='~/project/d*

Bases: *aitoolbox.cloud.AWS.data_access.AbstractDatasetFetcher*, *aitoolbox.cloud.AWS.data_access.BaseDataLoader*

Parameters

- **bucket_name** (*str*) –
- **local_dataset_folder_path** (*str*) –

fetch_dataset(*dataset_name=None*, *protect_local_folder=True*)

Parameters

- **dataset_name** (*None*) – no effect here
- **protect_local_folder** (*bool*) –

Returns None

fetch_preprocessed_dataset(*preprocess_name*, *protect_local_folder=True*)

Parameters

- **preprocess_name** (*str*) –
- **protect_local_folder** (*bool*) –

Returns None

class aitoolbox.cloud.AWS.data_access.**HotpotQADatasetFetcher**(*bucket_name='dataset-store'*, *local_dataset_folder_path='~/project/data'*)

Bases: *aitoolbox.cloud.AWS.data_access.AbstractDatasetFetcher*, *aitoolbox.cloud.AWS.data_access.BaseDataLoader*

https://hotpotqa.github.io/ https://arxiv.org/pdf/1809.09600.pdf

https://github.com/hotpotqa/hotpot

Parameters

- **bucket_name** (*str*) –
- **local_dataset_folder_path** (*str*) –

fetch_dataset(*dataset_name=None*, *protect_local_folder=True*)

Parameters

- **dataset_name** (*None*) – no effect here
- **protect_local_folder** (*bool*) –

Returns None

class aitoolbox.cloud.AWS.data_access.**TriviaQADatasetFetcher**(*bucket_name='dataset-store'*, *local_dataset_folder_path='~/project/data'*)

Bases: *aitoolbox.cloud.AWS.data_access.AbstractDatasetFetcher*, *aitoolbox.cloud.AWS.data_access.BaseDataLoader*

Parameters

- **bucket_name** (*str*) –
- **local_dataset_folder_path** (*str*) –

fetch_dataset(*dataset_name=None*, *protect_local_folder=True*)

Parameters

- **dataset_name** (*str or None*) – possible options: rc, unfiltered or None
- **protect_local_folder** (*bool*) –

Returns None

model_load

class aitoolbox.cloud.AWS.model_load.**BaseModelLoader**(*local_model_loader*, *local_model_result_folder_path='~/project/model_result'*, *bucket_name='model-result'*, *cloud_dir_prefix=''*)

Bases: *aitoolbox.cloud.AWS.data_access.BaseDataLoader*

Base saved model loading from S3 storage

Parameters

- **local_model_loader** (AbstractLocalModelLoader) – model loader implementing the loading of the saved model for the selected deep learning framework
- **local_model_result_folder_path** (*str*) – root local path where project folder will be created
- **bucket_name** (*str*) – name of the bucket in the cloud storage from which the model will be downloaded
- **cloud_dir_prefix** (*str*) – path to the folder inside the bucket where the experiments are going to be saved

load_model(*project_name*, *experiment_name*, *experiment_timestamp*, *model_save_dir='checkpoint_model'*, *epoch_num=None*, ***kwargs*)

Download and read/load the model

Parameters

- **project_name** (*str*) – root name of the project
- **experiment_name** (*str*) – name of the particular experiment
- **experiment_timestamp** (*str*) – time stamp at the start of training
- **model_save_dir** (*str*) – name of the folder inside experiment folder where the model is saved
- **epoch_num** (*int or None*) – epoch number of the model checkpoint or none if loading final model

- ****kwargs** – additional local_model_loader parameters

Returns model representation. (currently only returning dicts as only PyTorch model loading is supported)

Return type dict

class aitoolbox.cloud.AWS.model_load.**PyTorchS3ModelLoader**(*local_model_result_folder_path='~/project/mod*
bucket_name='model-result',
cloud_dir_prefix='')

Bases: *aitoolbox.cloud.AWS.model_load.BaseModelLoader*

PyTorch S3 model downloader & loader

Parameters

- **local_model_result_folder_path** (*str*) – root local path where project folder will be created
- **bucket_name** (*str*) – name of the bucket in the cloud storage from which the model will be downloaded
- **cloud_dir_prefix** (*str*) – path to the folder inside the bucket where the experiments are going to be saved

init_model(*model*, *used_data_parallel=False*)
Initialize provided PyTorch model with the loaded model weights

For this function to work, load_model() must be first called to read the model representation into memory.

Parameters

- **model** – PyTorch model
- **used_data_parallel** (*bool*) – if the saved model was nn.DataParallel or normal model

Returns initialized model

init_optimizer(*optimizer*, *device='cuda'*)
Initialize PyTorch optimizer

Parameters

- **optimizer** –
- **device** (*str*) –

Returns initialized optimizer

init_scheduler(*scheduler_callbacks_list*, *ignore_saved=False*, *ignore_missing_saved=False*)
Initialize the list of schedulers based on saved model/optimizer/scheduler checkpoint

Parameters

- **scheduler_callbacks_list** (*list*) – list of scheduler (callbacks)
- **ignore_saved** (*bool*) – if exception should be raised in the case there are found scheduler snapshots in the checkpoint, but not schedulers are provided to this method
- **ignore_missing_saved** (*bool*) – if exception should be raised in the case schedulers are provided to this method but no saved scheduler snapshots can be found in the checkpoint

Returns list of initialized scheduler (callbacks)

Return type list

init_amp(*amp_scaler*)
Initialize AMP GradScaler

Parameters **amp_scaler** (*torch.cuda.amp.GradScaler*) – AMP GradScaler

Returns initialized AMP GradScaler

Return type torch.cuda.amp.GradScaler

model_save

class aitoolbox.cloud.AWS.model_save.**AbstractModelSaver**
Bases: abc.ABC

abstract save_model(*model*, *project_name*, *experiment_name*, *experiment_timestamp=None*, *epoch=None*, *iteration_idx=None*, *protect_existing_folder=True*)

Parameters

- **model** –
- **project_name** (*str*) –
- **experiment_name** (*str*) –
- **experiment_timestamp** (*str or None*) –
- **epoch** (*int or None*) –
- **iteration_idx** (*int or None*) –
- **protect_existing_folder** (*bool*) –

Returns model_s3_path, experiment_timestamp, model_local_path

Return type (str, str, str)

class aitoolbox.cloud.AWS.model_save.**BaseModelSaver**(*bucket_name='model-result'*, *cloud_dir_prefix=''*, *checkpoint_model=False*)
Bases: *aitoolbox.cloud.AWS.data_access.BaseDataSaver*

Base model saving to AWS S3 functionality

Parameters

- **bucket_name** (*str*) – S3 bucket into which the files will be saved
- **cloud_dir_prefix** (*str*) – destination folder path inside selected bucket
- **checkpoint_model** (*bool*) – if the model that is going to be saved is final model or mid-training checkpoint

create_experiment_cloud_storage_folder_structure(*project_name*, *experiment_name*, *experiment_timestamp*)

Parameters

- **project_name** (*str*) – root name of the project
- **experiment_name** (*str*) – name of the particular experiment
- **experiment_timestamp** (*str*) – time stamp at the start of training

Returns experiment cloud path

Return type str

class aitoolbox.cloud.AWS.model_save.**PyTorchS3ModelSaver**(*bucket_name='model-result'*, *cloud_dir_prefix=''*, *local_model_result_folder_path='~/project/model_r*, *checkpoint_model=False*)

Bases: *aitoolbox.cloud.AWS.model_save.AbstractModelSaver*, *aitoolbox.cloud.AWS.model_save.BaseModelSaver*

PyTorch AWS S3 model saving

Parameters

- **bucket_name** (*str*) – name of the bucket in the S3 to which the models will be saved
- **cloud_dir_prefix** (*str*) – destination folder path inside selected bucket
- **local_model_result_folder_path** (*str*) – root local path where project folder will be created
- **checkpoint_model** (*bool*) – if the model being saved is checkpoint model or final end of training model

save_model(*model*, *project_name*, *experiment_name*, *experiment_timestamp=None*, *epoch=None*, *iteration_idx=None*, *protect_existing_folder=True*)

Save PyTorch model representation to AWS S3

Parameters

- **model** (*dict*) – PyTorch model representation dict
- **project_name** (*str*) – root name of the project
- **experiment_name** (*str*) – name of the particular experiment
- **experiment_timestamp** (*str or None*) – time stamp at the start of training
- **epoch** (*int or None*) – epoch number
- **iteration_idx** (*int or None*) – at which training iteration the model is being saved
- **protect_existing_folder** (*bool*) – can override potentially already existing folder or not

Returns model_s3_path, experiment_timestamp, model_local_path

Return type (str, str, str)

Examples

```
local_model_result_folder_path = '~/project/model_results'
m_saver = PyTorchLocalModelSaver(local_model_result_folder_path=local_model_
↪result_folder_path)
m_saver.save_model(model=model,
                   project_name='QA_QAngaroo',
                   experiment_name='FastQA_RNN_concat_model_GLOVE',
                   protect_existing_folder=False)
```

class aitoolbox.cloud.AWS.model_save.**KerasS3ModelSaver**(*bucket_name='model-result'*, *cloud_dir_prefix=''*, *local_model_result_folder_path='~/project/model_resu*
checkpoint_model=False)

Bases: *aitoolbox.cloud.AWS.model_save.AbstractModelSaver*, *aitoolbox.cloud.AWS.model_save.BaseModelSaver*

Keras AWS S3 model saving

Parameters

- **bucket_name** (*str*) – name of the bucket in the S3 to which the models will be saved
- **cloud_dir_prefix** (*str*) – destination folder path inside selected bucket
- **local_model_result_folder_path** (*str*) – root local path where project folder will be created
- **checkpoint_model** (*bool*) – if the model being saved is checkpoint model or final end of training model

save_model(*model*, *project_name*, *experiment_name*, *experiment_timestamp=None*, *epoch=None*, *iteration_idx=None*, *protect_existing_folder=True*)

Save Keras model to AWS S3

Parameters

- **model** (*keras.Model*) –
- **project_name** (*str*) – root name of the project
- **experiment_name** (*str*) – name of the particular experiment
- **experiment_timestamp** (*str or None*) – time stamp at the start of training
- **epoch** (*int or None*) – epoch number
- **iteration_idx** (*int or None*) – at which training iteration the model is being saved
- **protect_existing_folder** (*bool*) – can override potentially already existing folder or not

Returns model_s3_path, experiment_timestamp, model_local_path

Return type (str, str, str)

Examples

```
local_model_result_folder_path = '~/project/model_results'
m_saver = KerasS3ModelSaver(local_model_result_folder_path=local_model_result_
↪folder_path)
m_saver.save_model(model=model,
                   project_name='QA_QAngaroo',
                   experiment_name='FastQA_RNN_concat_model_GLOVE',
                   protect_existing_folder=False)
```

results_save

class aitoolbox.cloud.AWS.results_save.**AbstractResultsSaver**

Bases: abc.ABC

abstract save_experiment_results(*result_package*, *training_history*, *project_name*, *experiment_name*, *experiment_timestamp=None*, *save_true_pred_labels=False*, *separate_files=False*, *protect_existing_folder=True*)

Parameters

- **result_package** (aitoolbox.experiment.result_package.abstract_result_packages.AbstractResultPackage) –
- **training_history** (aitoolbox.experiment.training_history.TrainingHistory) –
- **project_name** (*str*) –
- **experiment_name** (*str*) –
- **experiment_timestamp** (*str*) –
- **save_true_pred_labels** (*bool*) –
- **separate_files** (*bool*) –
- **protect_existing_folder** (*bool*) –

Returns results_file_s3_path, experiment_timestamp

Return type (str, str)

class aitoolbox.cloud.AWS.results_save.**BaseResultsSaver**(*bucket_name='model-result'*, *cloud_dir_prefix=''*)

Bases: *aitoolbox.cloud.AWS.data_access.BaseDataSaver*

Base experiment results saving to AWS S3 functionality

Parameters

- **bucket_name** (*str*) – S3 bucket into which the files will be saved
- **cloud_dir_prefix** (*str*) – destination folder path inside selected bucket

create_experiment_cloud_storage_folder_structure(*project_name*, *experiment_name*, *experiment_timestamp*)

Parameters

- **project_name** (*str*) – root name of the project
- **experiment_name** (*str*) – name of the particular experiment
- **experiment_timestamp** (*str*) – time stamp at the start of training

Returns experiment cloud path

Return type str

class aitoolbox.cloud.AWS.results_save.**S3ResultsSaver**(*bucket_name='model-result'*, *cloud_dir_prefix=''*, *local_model_result_folder_path='~/project/model_resul*

Bases: *aitoolbox.cloud.AWS.results_save.AbstractResultsSaver*, *aitoolbox.cloud.AWS.results_save.BaseResultsSaver*

AWS S3 results saver

It first saves the results files to local drive and then uploads them to S3

Parameters

- **bucket_name** (*str*) – name of the bucket in the S3 to which the results files will be saved
- **cloud_dir_prefix** (*str*) – destination folder path inside selected bucket
- **local_model_result_folder_path** (*str*) – root local path where project folder will be created

save_experiment_results(*result_package*, *training_history*, *project_name*, *experiment_name*, *experiment_timestamp=None*, *save_true_pred_labels=False*, *separate_files=False*, *protect_existing_folder=True*)

Save produced experiment results recorded in the result package to the results file on local drive and upload them to S3

Parameters

- **result_package** (aitoolbox.experiment.result_package.abstract_result_packages.AbstractResultPackage) –
- **training_history** (aitoolbox.experiment.training_history.TrainingHistory) –
- **project_name** (*str*) – root name of the project
- **experiment_name** (*str*) – name of the particular experiment
- **experiment_timestamp** (*str or None*) – time stamp at the start of training
- **save_true_pred_labels** (*bool*) – should ground truth labels also be saved
- **separate_files** (*bool*) – should the results be saved in separate pickle files or should all of the results be batched together in a single results file
- **protect_existing_folder** (*bool*) – can override potentially already existing folder or not

Returns results file path on S3, experiment timestamp

Return type (str, str)

simple_email_service

class aitoolbox.cloud.AWS.simple_email_service.**SESSender**(*sender_name*, *sender_email*, *recipient_email*, *aws_region='eu-west-1'*)

Bases: object

AWS Simple Email Service sender

Used for sending email notifications about the progression of the training.

Parameters

- **sender_name** (*str*) – Name of the email sender
- **sender_email** (*str*) – Email of the email sender
- **recipient_email** (*str*) – Email where the email will be sent
- **aws_region** (*str*) – AWS SES region

send_email(*subject*, *body*, *attachment_file_paths=None*)

Send email text with optional attachments

Parameters

- **subject** (*str*) – email subject
- **body** (*str*) – HTML body of the email
- **attachment_file_paths** (*list or None*) – list of local paths pointing to the email attachment files

Returns None

6.1.3.1.2 GoogleCloud

Submodules

data_access

class aitoolbox.cloud.GoogleCloud.data_access.**BaseGoogleStorageDataSaver**(*bucket_name='model-result'*)

Bases: object

Parameters **bucket_name** (*str*) –

save_file(*local_file_path*, *cloud_file_path*)

Parameters

- **local_file_path** (*str*) –
- **cloud_file_path** (*str*) –

Returns None

class aitoolbox.cloud.GoogleCloud.data_access.**BaseGoogleStorageDataLoader**(*bucket_name='dataset-store'*, *local_dataset_folder_path*)

Bases: object

Parameters

- **bucket_name** (*str*) –
- **local_dataset_folder_path** (*str*) –

load_file(*cloud_file_path*, *local_file_path*)

Parameters

- **cloud_file_path** (*str*) –
- **local_file_path** (*str*) –

Returns None

model_load

class aitoolbox.cloud.GoogleCloud.model_load.**BaseModelGoogleStorageLoader**(*local_model_loader*, *local_model_result_folder*, *bucket_name='model-result'*, *cloud_dir_prefix=''*)

Bases: *aitoolbox.cloud.GoogleCloud.data_access.BaseGoogleStorageDataLoader*

Base saved model loading from Google Cloud Storage

Parameters

- **local_model_loader** (AbstractLocalModelLoader) – model loader implementing the loading of the saved model for the selected deep learning framework
- **local_model_result_folder_path** (*str*) – root local path where project folder will be created
- **bucket_name** (*str*) – name of the bucket in the cloud storage from which the model will be downloaded
- **cloud_dir_prefix** (*str*) – path to the folder inside the bucket where the experiments are going to be saved

class aitoolbox.cloud.GoogleCloud.model_load.**PyTorchGoogleStorageModelLoader**(*local_model_result_folder*, *bucket_name='model-result'*, *cloud_dir_prefix=''*)

Bases: *aitoolbox.cloud.GoogleCloud.model_load.BaseModelGoogleStorageLoader*, *aitoolbox.cloud.AWS.model_load.PyTorchS3ModelLoader*

PyTorch Google Cloud Storage model downloader & loader

Parameters

- **local_model_result_folder_path** (*str*) – root local path where project folder will be created
- **bucket_name** (*str*) – name of the bucket in the cloud storage from which the model will be downloaded
- **cloud_dir_prefix** (*str*) – path to the folder inside the bucket where the experiments are going to be saved

model_save

class aitoolbox.cloud.GoogleCloud.model_save.**BaseModelGoogleStorageSaver**(*bucket_name='model-result'*, *cloud_dir_prefix=''*, *checkpoint_model=False*)

Bases: *aitoolbox.cloud.GoogleCloud.data_access.BaseGoogleStorageDataSaver*

Base model saving to Google Cloud Storage functionality

Parameters

- **bucket_name** (*str*) – Google Cloud Storage bucket into which the files will be saved
- **cloud_dir_prefix** (*str*) – destination folder path inside selected bucket
- **checkpoint_model** (*bool*) – if the model that is going to be saved is final model or mid-training checkpoint

class aitoolbox.cloud.GoogleCloud.model_save.**PyTorchGoogleStorageModelSaver**(*bucket_name='model-result'*, *cloud_dir_prefix=''*, *local_model_result_fold*, *checkpoint_model=False*)

Bases: *aitoolbox.cloud.GoogleCloud.model_save.BaseModelGoogleStorageSaver*, *aitoolbox.cloud.AWS.model_save.PyTorchS3ModelSaver*

PyTorch Google Cloud Storage model saving

Parameters

- **bucket_name** (*str*) – name of the bucket in the Google Cloud Storage to which the models will be saved
- **cloud_dir_prefix** (*str*) – destination folder path inside selected bucket
- **local_model_result_folder_path** (*str*) – root local path where project folder will be created
- **checkpoint_model** (*bool*) – if the model being saved is checkpoint model or final end of training model

class aitoolbox.cloud.GoogleCloud.model_save.**KerasGoogleStorageModelSaver**(*bucket_name='model-result'*, *cloud_dir_prefix=''*, *local_model_result_folder*, *checkpoint_model=False*)

Bases: *aitoolbox.cloud.GoogleCloud.model_save.BaseModelGoogleStorageSaver*, *aitoolbox.cloud.AWS.model_save.KerasS3ModelSaver*

Keras Google Storage model saving

Parameters

- **bucket_name** (*str*) – name of the bucket in the Google Cloud Storage to which the models will be saved
- **cloud_dir_prefix** (*str*) – destination folder path inside selected bucket
- **local_model_result_folder_path** (*str*) – root local path where project folder will be created
- **checkpoint_model** (*bool*) – if the model being saved is checkpoint model or final end of training model

results_save

class aitoolbox.cloud.GoogleCloud.results_save.**BaseResultsGoogleStorageSaver**(*bucket_name='model-result'*, *cloud_dir_prefix=''*)

Bases: *aitoolbox.cloud.GoogleCloud.data_access.BaseGoogleStorageDataSaver*

Base experiment results saving to Google Cloud Storage functionality

Parameters

- **bucket_name** (*str*) – Google Cloud Storage bucket into which the files will be saved
- **cloud_dir_prefix** (*str*) – destination folder path inside selected bucket

class aitoolbox.cloud.GoogleCloud.results_save.**GoogleStorageResultsSaver**(*bucket_name='model-result'*, *cloud_dir_prefix=''*, *local_model_result_folder_*

Bases: *aitoolbox.cloud.GoogleCloud.results_save.BaseResultsGoogleStorageSaver*, *aitoolbox.cloud.AWS.results_save.S3ResultsSaver*

Google Cloud Storage results saver

It first saves the results files to local drive and then uploads them to GCS.

Parameters

- **bucket_name** (*str*) – name of the bucket in the Google Cloud Storage to which the results files will be saved
- **cloud_dir_prefix** (*str*) – destination folder path inside selected bucket
- **local_model_result_folder_path** (*str*) – root local path where project folder will be created

6.1.4 nlp

6.1.4.1 Subpackages

6.1.4.1.1 core

Submodules

core

aitoolbox.nlp.core.core.**unicode_to_ascii**(*text_string*)

Turn a Unicode string to plain ASCII

Taken from: http://stackoverflow.com/a/518232/2809427

Parameters **text_string** (*str*) –

Returns

Return type str

aitoolbox.nlp.core.core.**normalize_string**(*text_string*, *unicode_to_ascii_convert=True*)

Lowercase, trim, and remove non-letter characters

Parameters

- **text_string** (*str*) –
- **unicode_to_ascii_convert** (*bool*) –

Returns

Return type str

aitoolbox.nlp.core.core.**str2bool**(*w*)

Parameters **w** –

Returns

Return type bool

aitoolbox.nlp.core.core.**find_sub_list**(*sub_list*, *main_list*)

Find starting and ending position of a sublist in a longer list.

Parameters

- **sub_list** (*list*) – sublist
- **main_list** (*list*) – main longer list

Returns start and end index in the list l. Returns None if sublist is not found in the main list.

Return type (int, int)

vocabulary

class aitoolbox.nlp.core.vocabulary.**Vocabulary**(*name*, *document_level=False*)

Bases: object

Vocabulary used for storing the tokens and converting between the indices and the tokens

Parameters

- **name** (*str*) – name of the vocabulary / type of vocabulary. Needed just for tracking purposes
- **document_level** (*bool*) – If the vocabulary is on the sentence level or on the document level. Document consists of multiple sentences. This in effect means that we are adding additional tokens for start and the end of the doc.

add_sentence(*sentence_tokens*)

Add tokenized sentence to the vocabulary

Parameters **sentence_tokens** (*list*) – sentence tokens, e.g. list of words representing the sentence

Returns None

add_word(*word*)

Add the single word to the vocabulary

Parameters **word** (*str*) – single word string

Returns None

trim(*min_count*)

Remove words below a certain count threshold

Parameters **min_count** (*int*) –

Returns None

convert_sent2idx_sent(*sent_tokens*, *start_end_token=True*)
Convert the given tokenized string sentence into the indices

Parameters

- **sent_tokens** (*list*) –
- **start_end_token** (*bool*) – string tokens forming a sentence

Returns sentence tokens converted into the corresponding indices

Return type list

convert_idx_sent2sent(*idx_sent*, *rm_default_tokens=False*)
Convert from token indices back to the human-readable string tokens

Parameters

- **idx_sent** – index tokens forming the sentence
- **rm_default_tokens** (*bool*) – should the default tokens such as padding and start/end sentence tokens be removed from the result.

Returns sentence represented as a sequence of the string tokens

Return type list

6.1.4.1.2 dataset

Subpackages

CNNDailyMail

Submodules

CNNDailyMail

aitoolbox.nlp.dataset.CNNDailyMail.CNNDailyMail.**get_preproc_dataset_local_copy**(*local_dataset_fo*
pre-
pro-
cess_name='abis
pro-
tect_local_folder

Interface method for getting a local copy of CNN/DailyMail dataset

If a local copy is not found, dataset is automatically downloaded from S3.

Parameters

- **local_dataset_folder_path** (*str*) –
- **preprocess_name** (*str*) –
- **protect_local_folder** (*bool*) –

Returns None

HotpotQA

Submodules

HotpotQA

aitoolbox.nlp.dataset.HotpotQA.HotpotQA.**get_dataset_local_copy**(*local_dataset_folder_path*, *protect_local_folder=True*)

Interface method for getting a local copy of HotpotQA dataset

If a local copy is not found, dataset is automatically downloaded from S3.

Parameters

- **local_dataset_folder_path** (*str*) –
- **protect_local_folder** (*bool*) –

Returns None

QAngaroo

Submodules

QAngaroo

aitoolbox.nlp.dataset.QAngaroo.QAngaroo.**get_dataset_local_copy**(*local_dataset_folder_path*, *dataset_name=None*, *protect_local_folder=True*)

Interface method for getting a local copy of QAngaroo dataset

If a local copy is not found, dataset is automatically downloaded from S3.

Parameters

- **local_dataset_folder_path** (*str*) –
- **dataset_name** (*str or None*) – possible options: medhop, wikihop or None
- **protect_local_folder** (*bool*) –

Returns None

SQuAD2

Subpackages

deprecated

Submodules

manual_SQuAD2

aitoolbox.nlp.dataset.SQuAD2.deprecated.manual_SQuAD2.**get_dataset_local_copy**(*local_dataset_folde*
pro-
tect_local_folder=T

Interface method for getting a local copy of SQuAD2 dataset

If a local copy is not found, dataset is automatically downloaded from S3.

Parameters

- **local_dataset_folder_path** (*str*) –
- **protect_local_folder** (*bool*) –

Returns None

class aitoolbox.nlp.dataset.SQuAD2.deprecated.manual_SQuAD2.**SQuAD2DatasetPrepareResult**(*datas*
datas
*save_*

Bases: object

Parameters

- **dataset_name** –
- **dataset_type** –
- **vocab_memory_safeguard** –

store_data(*context_text_list*, *question_text_list*, *answer_text_list*, *orig_answer_start_end_tuple_list*, *answer_start_end_tuple_list*)

Parameters

- **context_text_list** –
- **question_text_list** –
- **answer_text_list** –
- **orig_answer_start_end_tuple_list** –
- **answer_start_end_tuple_list** –

Returns:

store_vocab(*vocab*)

Parameters **vocab** –

Returns:

store_max_context_questions_max_len(*max_ctx_qs_len*)

Parameters **max_ctx_qs_len** –

Returns:

class aitoolbox.nlp.dataset.SQuAD2.deprecated.manual_SQuAD2.**SQuAD2DataPreparation**(*train_path*, *dev_path*, *skip_is_impo*
skip_exampl

Bases: object

Parameters

- **train_path** –

- **dev_path** –
- **skip_is_impossible** –
- **skip_examples_w_span** –

process_data(*dump_folder_path=None*)

Parameters **dump_folder_path** –

Returns:

vectorize_data(*train_data=None*, *dev_data=None*, *vocab=None*, *dump_folder_path=None*)

Parameters

- **train_data** –
- **dev_data** –
- **vocab** –
- **dump_folder_path** –

Returns:

get_vectorized_data_prep_result(*data_prep_result*, *vocab*)

Parameters

- **data_prep_result** –
- **vocab** –

Returns:

build_dataset(*data_json*, *dataset_name*)

Parameters

- **data_json** –
- **dataset_name** –

Returns:

process_context_text(*context_text*, *is_train*)

Parameters

- **context_text** –
- **is_train** –

Returns:

process_question_text(*question_text*, *is_train*)

Parameters

- **question_text** –
- **is_train** –

Returns:

process_answer_text(*answer_text*, *is_train*)

Parameters

- **answer_text** –

- **is_train** –

Returns:

load_json_file(*file_path*)

Parameters **file_path** –

Returns:

load_prep_dumps(*dump_folder_path*)

Parameters **dump_folder_path** –

Returns:

load_vect_prep_dumps(*dump_folder_path*)

Parameters **dump_folder_path** –

Returns:

Submodules

SQuAD2DataReader

aitoolbox.nlp.dataset.SQuAD2.SQuAD2DataReader.**get_dataset_local_copy**(*local_dataset_folder_path*, *protect_local_folder=True*)

Interface method for getting a local copy of SQuAD2 dataset

If a local copy is not found, dataset is automatically downloaded from S3.

Parameters

- **local_dataset_folder_path** (*str*) –
- **protect_local_folder** (*bool*) –

Returns None

class aitoolbox.nlp.dataset.SQuAD2.SQuAD2DataReader.**SQuAD2ConcatContextDatasetReader**(*file_path*, *tokenizer=None*, *is_train=True*, *dev_mode_size=None*)

Bases: object

Parameters

- **file_path** (*str*) –
- **tokenizer** –
- **is_train** (*bool*) –
- **dev_mode_size** –

read()

Read SQuAD data. Tested and it works

Returns

Return type list, *aitoolbox.nlp.core.vocabulary.Vocabulary*

process_example(*paragraph_tokens*, *question_tokens*, *char_spans*, *answer_texts*)

Parameters

- **paragraph_tokens** –
- **question_tokens** –
- **char_spans** –
- **answer_texts** –

Returns:

tokenize_process_paragraph(*paragraph_text*)

Parameters **paragraph_text** –

Returns:

tokenize_process_question(*question_text*)

Parameters **question_text** –

Returns:

class aitoolbox.nlp.dataset.SQuAD2.SQuAD2DataReader.**GeneratorSQuAD2ConcatContextDatasetRead**

Bases: *aitoolbox.nlp.dataset.SQuAD2.SQuAD2DataReader.SQuAD2ConcatContextDatasetReader*

This implementation with the generator has not been tested yet.

Check especially in the read() if calling list(self.read_generator()) also in turn fills the self.vocab, thus you get returned a complete vocabulary

Parameters

- **file_path** –
- **tokenizer** –
- **is_train** –
- **dev_mode_size** –

read()

Returns

Return type list

read_generator()

Yields *list*

TriviaQA

Submodules

TriviaQA

aitoolbox.nlp.dataset.TriviaQA.TriviaQA.**get_dataset_local_copy**(*local_dataset_folder_path*, *dataset_name=None*, *protect_local_folder=True*)

Interface method for getting a local copy of TriviaQA dataset

If a local copy is not found, dataset is automatically downloaded from S3.

Parameters

- **local_dataset_folder_path** (*str*) –
- **dataset_name** (*str or None*) – possible options: rc, unfiltered or None
- **protect_local_folder** (*bool*) –

Returns None

Submodules

torch_collate_fns

aitoolbox.nlp.dataset.torch_collate_fns.**qa_concat_ctx_span_collate_fn**(*data*)

QA system collate function

Creates mini-batch tensors from the list of tuples (src_seq, trg_seq). We should build a custom collate_fn rather than using default collate_fn, because merging sequences (including padding) is not supported in default. Sequences are padded to the maximum length of mini-batch sequences (dynamic padding).

Parameters **data** – list of tuple (src_seq, trg_seq). - src_seq: torch tensor of shape (?); variable length. - trg_seq: torch tensor of shape (?); variable length.

Returns torch tensor of shape (batch_size, padded_length). src_lengths: list of length (batch_size); valid length for each padded source sequence. trg_seqs: torch tensor of shape (batch_size, padded_length). trg_lengths: list of length (batch_size); valid length for each padded target sequence.

Return type src_seqs

6.1.4.1.3 experiment_evaluation

Submodules

NLP_metrics

class aitoolbox.nlp.experiment_evaluation.NLP_metrics.**ROUGEMetric**(*y_true*, *y_predicted*, *target_actual_text=False*, *output_text_dir=None*, *output_text_cleaning_regex=('<.*?>', '[^a-zA-Z0-9.?!]+')*)

Bases: *aitoolbox.experiment.core_metrics.abstract_metric.AbstractBaseMetric*

ROGUE score calculation

From this package: https://github.com/pltrdy/rouge

Parameters

- **y_true** (*numpy.array or list*) –
- **y_predicted** (*numpy.array or list*) –
- **target_actual_text** (*bool*) –
- **output_text_dir** (*str*) –
- **output_text_cleaning_regex** (*list*) –

calculate_metric()

Perform metric calculation and return it from this function

Returns return metric_result

Return type float or dict

prepare_text()

static dump_answer_text_to_disk(*true_text*, *pred_text*, *output_text_dir*, *output_text_cleaning_regex*, *target_actual_text*)

Problems: Defined regex text cleaning to deal with Illegal division by zero https://ireneli.eu/2018/01/11/working-with-rouge-1-5-5-evaluation-metric-in-python/

Parameters

- **true_text** (*list*) –
- **pred_text** (*list*) –
- **output_text_dir** (*str*) –
- **output_text_cleaning_regex** (*list*) –
- **target_actual_text** (*bool*) –

Returns:

class aitoolbox.nlp.experiment_evaluation.NLP_metrics.**ROUGEPerlMetric**(*y_true*, *y_predicted*, *output_text_dir*, *output_text_cleaning_regex=('<.*'[^a-zA-Z0-9.?!]+')*, *target_actual_text=False*)

Bases: *aitoolbox.experiment.core_metrics.abstract_metric.AbstractBaseMetric*

ROGUE score calculation using the Perl implementation

Use this package: https://pypi.org/project/pyrouge/ https://github.com/bheinzerling/pyrouge

Problems: Defined regex text cleaning to deal with Illegal division by zero https://ireneli.eu/2018/01/11/working-with-rouge-1-5-5-evaluation-metric-in-python/

Parameters

- **y_true** (*numpy.array or list*) – gold standard summaries are 'model' summaries
- **y_predicted** (*numpy.array or list*) – your summaries are 'system' summaries
- **output_text_dir** (*str*) –
- **output_text_cleaning_regex** (*list*) –
- **target_actual_text** (*bool*) –

calculate_metric()

Perform metric calculation and return it from this function

Returns return metric_result

Return type float or dict

static dump_answer_text_to_disk(*true_text*, *pred_text*, *output_text_dir*, *output_text_cleaning_regex*, *target_actual_text*)

Problems: Defined regex text cleaning to deal with Illegal division by zero https://ireneli.eu/2018/01/11/working-with-rouge-1-5-5-evaluation-metric-in-python/

Parameters

- **true_text** (*list*) –
- **pred_text** (*list*) –
- **output_text_dir** (*str*) –
- **output_text_cleaning_regex** (*list*) –
- **target_actual_text** (*bool*) –

Returns:

static regex_clean_text(*text*, *cleaning_regex_list*)

Parameters

- **text** (*list*) –
- **cleaning_regex_list** (*list*) –

Returns

Return type list

class aitoolbox.nlp.experiment_evaluation.NLP_metrics.**ExactMatchTextMetric**(*y_true*, *y_predicted*, *target_actual_text=False*, *output_text_dir=None*)

Bases: *aitoolbox.experiment.core_metrics.abstract_metric.AbstractBaseMetric*

Calculate exact match of answered strings

Parameters

- **y_true** (*numpy.array or list*) –
- **y_predicted** (*numpy.array or list*) –
- **target_actual_text** (*bool*) –
- **output_text_dir** (*str*) –

calculate_metric()

Perform metric calculation and return it from this function

Returns return metric_result

Return type float or dict

static normalize_answer(*text_str*)

Convert to lowercase and remove punctuation, articles and extra whitespace.

All methods below this line are from the official SQuAD 2.0 eval script https://worksheets.codalab.org/rest/bundles/0x6b567e1cf2e041ec80d7098f031c5c9e/contents/blob/

Parameters **text_str** (*str*) –

Returns str

class aitoolbox.nlp.experiment_evaluation.NLP_metrics.**F1TextMetric**(*y_true*, *y_predicted*, *target_actual_text=False*, *output_text_dir=None*)

Bases: *aitoolbox.experiment.core_metrics.abstract_metric.AbstractBaseMetric*

Calculate F1 score of answered strings

Parameters

- **y_true** (*numpy.array or list*) –
- **y_predicted** (*numpy.array or list*) –
- **target_actual_text** (*bool*) –
- **output_text_dir** (*str*) –

calculate_metric()

Perform metric calculation and return it from this function

Returns return metric_result

Return type float or dict

static compute_f1(*a_gold*, *a_pred*)

static get_tokens(*s*)

class aitoolbox.nlp.experiment_evaluation.NLP_metrics.**BLEUSentenceScoreMetric**(*y_true*, *y_predicted*, *source_sents=Non* *out-* *put_text_dir=None*

Bases: *aitoolbox.experiment.core_metrics.abstract_metric.AbstractBaseMetric*

BLEU score calculation

NLTK provides the sentence_bleu() function for evaluating a candidate sentence against one or more reference sentences.

https://machinelearningmastery.com/calculate-bleu-score-for-text-python/

The reference sentences must be provided as a list of sentences where each reference is a list of tokens. The candidate sentence is provided as a list of tokens. For example:

reference = [[‘this’, ‘is’, ‘a’, ‘test’], [‘this’, ‘is’ ‘test’]] candidate = [‘this’, ‘is’, ‘a’, ‘test’] score = sentence_bleu(reference, candidate)

Parameters

- **y_true** (*list*) –
- **y_predicted** (*list*) –
- **source_sents** (*list or None*) –
- **output_text_dir** (*str or None*) –

calculate_metric()

Perform metric calculation and return it from this function

Returns return metric_result

Return type float or dict

static dump_translation_text_to_disk(*source_sents*, *pred_translations*, *true_translations*, *sentence_bleu_results*, *output_text_dir*)

Parameters

- **source_sents** (*list*) –
- **pred_translations** (*list*) –
- **true_translations** (*list*) –
- **sentence_bleu_results** (*list*) –
- **output_text_dir** (*str*) –

Returns:

static check_transl_sent_num_match(*sent_types*)

Parameters **sent_types** (*list*) – list of lists

Raises **ValueError** –

class aitoolbox.nlp.experiment_evaluation.NLP_metrics.**BLEUCorpusScoreMetric**(*y_true*, *y_predicted*, *source_sents=None*, *output_text_dir=None*)

Bases: *aitoolbox.experiment.core_metrics.abstract_metric.AbstractBaseMetric*

BLEU corpus score calculation

Function called corpus_bleu() for calculating the BLEU score for multiple sentences such as a paragraph or a document.

https://machinelearningmastery.com/calculate-bleu-score-for-text-python/

The references must be specified as a list of documents where each document is a list of references and each alternative reference is a list of tokens, e.g. a list of lists of lists of tokens. The candidate documents must be specified as a list where each document is a list of tokens, e.g. a list of lists of tokens.

references = [[['this', 'is', 'a', 'test'], ['this', 'is' 'test']]] candidates = [['this', 'is', 'a', 'test']] score = corpus_bleu(references, candidates)

Parameters

- **y_true** (*list*) –
- **y_predicted** (*list*) –
- **source_sents** (*list or None*) –
- **output_text_dir** (*str or None*) –

calculate_metric()

Perform metric calculation and return it from this function

Returns return metric_result

Return type float or dict

class aitoolbox.nlp.experiment_evaluation.NLP_metrics.**BLEUScoreStrTorchNLPMetric**(*y_true*, *y_predicted*, *lowercase=False*, *source_sents=*, *output_text_dir=*

Bases: *aitoolbox.experiment.core_metrics.abstract_metric.AbstractBaseMetric*

BLEU score calculation using the TorchNLP implementation

Example

hypotheses = [“The brown fox jumps over the dog ”, “The brown fox jumps over the dog 2 ”]

references = [“The quick brown fox jumps over the lazy dog ”, “The quick brown fox jumps over the lazy dog ”]

get_moses_multi_bleu(hypotheses, references, lowercase=True) 46.51

Parameters

- **y_true** (*list*) –
- **y_predicted** (*list*) –
- **lowercase** (*bool*) –
- **source_sents** (*list or None*) –
- **output_text_dir** (*str or None*) –

calculate_metric()
Perform metric calculation and return it from this function

Returns return metric_result

Return type float or dict

class aitoolbox.nlp.experiment_evaluation.NLP_metrics.**PerplexityMetric**(*batch_losses*)
Bases: *aitoolbox.experiment.core_metrics.abstract_metric.AbstractBaseMetric*

Perplexity metric used in MT

Parameters **batch_losses** (*numpy.array or list*) –

calculate_metric()
Perform metric calculation and return it from this function

Returns return metric_result

Return type float or dict

class aitoolbox.nlp.experiment_evaluation.NLP_metrics.**GLUEMetric**(*y_true*, *y_predicted*, *task_name*)
Bases: *aitoolbox.experiment.core_metrics.abstract_metric.AbstractBaseMetric*

GLUE evaluation metrics

Wrapper around HF Transformers `glue_compute_metrics()`

Parameters

- **y_true** –
- **y_predicted** –
- **task_name** (*str*) – name of the GLUE task

calculate_metric()
Perform metric calculation and return it from this function

Returns return metric_result

Return type float or dict

class aitoolbox.nlp.experiment_evaluation.NLP_metrics.**XNLIMetric**(*y_true*, *y_predicted*)

Bases: *aitoolbox.experiment.core_metrics.abstract_metric.AbstractBaseMetric*

XNLI evaluation metrics

Wrapper around HF Transformers `xnli_compute_metrics()`

Parameters

- **y_true** –
- **y_predicted** –

calculate_metric()

Perform metric calculation and return it from this function

Returns return metric_result

Return type float or dict

NLP_result_package

class aitoolbox.nlp.experiment_evaluation.NLP_result_package.**QuestionAnswerResultPackage**(*pa ta ge ou pu us fla te st ***

Bases: *aitoolbox.experiment.result_package.abstract_result_packages.AbstractResultPackage*

Question Answering task performance evaluation result package

Parameters

- **paragraph_text_tokens** (*list*) –
- **target_actual_text** (*list or None*) –
- **output_text_dir** (*str or None*) –
- **use_perl_rouge** (*bool*) –
- **flatten_result_dict** (*bool*) –
- **strict_content_check** (*bool*) –
- ****kwargs** (*dict*) –

prepare_results_dict()

Returns

Return type dict

set_experiment_dir_path_for_additional_results(*project_name*, *experiment_name*, *experiment_timestamp*, *local_model_result_folder_path*)

Set experiment folder path after potential timestamps have already been generated.

Experiment folder setting for additional metadata results output is needed only in certain result packages, for example in QuestionAnswerResultPackage where the self.output_text_dir initially has only the name of the folder where the results text predictions for each example should be stored. This function when implemented reforms the folder name so that it becomes a full path placing the folder inside the experiment folder (for which the timestamp at the start of train loop is needed).

Another use of this function is in MachineTranslationResultPackage where the attention heatmap pictures are stored as additional metadata results.

As can be seen from the fact that the train loop mechanism is mentioned, this method's functionality is primarily used for PyTorch experiments.

Parameters

- **project_name** (*str*) – root name of the project
- **experiment_name** (*str*) – name of the particular experiment
- **experiment_timestamp** (*str*) – time stamp at the start of training
- **local_model_result_folder_path** (*str*) – root local path where project folder will be created

Returns None

list_additional_results_dump_paths()

Specify the list of meta data files you also want to save & upload to s3 during the experiment saving procedure

By default there are no additional files that are saved as the return is None. If you want to save your specific additional files produced during the training procedure, then override this method specifying the file paths.

If you want to save a whole folder of files, use zip_additional_results_dump() function to zip it into a single file and save this zip instead.

The specified files are any additional data you would want to include into the experiment folder in addition to the model save files and performance evaluation report files. For example a zip of attention heatmap pictures in the machine translation projects.

Returns

list of lists of string paths if it is not None. Each element of the list should be list of: [[results_file_name, results_file_local_path], ... [,]]

Return type list or None

class aitoolbox.nlp.experiment_evaluation.NLP_result_package.**QuestionAnswerSpanClassificat**

Bases: *aitoolbox.experiment.result_package.abstract_result_packages.AbstractResultPackage*

Extractive Question Answering task performance evaluation result package

Evaluates the classification of the correct answer start and end points.

Parameters

- **strict_content_check** (*bool*) –
- ****kwargs** (*dict*) –

prepare_results_dict()

Available general data:

y_span_start_true (numpy.array or list): y_span_start_predicted (numpy.array or list): y_span_end_true (numpy.array or list): y_span_end_predicted (numpy.array or list): strict_content_check (bool): **kwargs (dict):

Returns

Return type dict

class aitoolbox.nlp.experiment_evaluation.NLP_result_package.**TextSummarizationResultPackage**

Bases: *aitoolbox.experiment.result_package.abstract_result_packages.AbstractResultPackage*

Text summarization task performance evaluation package

Parameters

- **strict_content_check** (*bool*) –
- ****kwargs** (*dict*) –

prepare_results_dict()

Returns

Return type dict

class aitoolbox.nlp.experiment_evaluation.NLP_result_package.**MachineTranslationResultPackag**

Bases: *aitoolbox.experiment.result_package.abstract_result_packages.AbstractResultPackage*

Machine Translation task performance evaluation package

Parameters

- **target_vocab** (aitoolbox.nlp.core.vocabulary.Vocabulary) –
- **source_vocab** (aitoolbox.nlp.core.vocabulary.Vocabulary *or None*) –
- **source_sents** (*list or None*) –
- **output_text_dir** (*str or None*) –
- **output_attn_heatmap_dir** (*str or None*) –
- **strict_content_check** (*bool*) –
- ****kwargs** (*dict*) –

prepare_results_dict()

Returns

result dict which is combination of different BLEU metric calculations and possibly saved attention heatmap plot files and perplexity

Return type dict

set_experiment_dir_path_for_additional_results(*project_name*, *experiment_name*, *experiment_timestamp*, *local_model_result_folder_path*)

Set experiment folder path after potential timestamps have already been generated.

Experiment folder setting for additional metadata results output is needed only in certain result packages, for example in QuestionAnswerResultPackage where the self.output_text_dir initially has only the name of the folder where the results text predictions for each example should be stored. This function when implemented reforms the folder name so that it becomes a full path placing the folder inside the experiment folder (for which the timestamp at the start of train loop is needed).

Another use of this function is in MachineTranslationResultPackage where the attention heatmap pictures are stored as additional metadata results.

As can be seen from the fact that the train loop mechanism is mentioned, this method's functionality is primarily used for PyTorch experiments.

Parameters

- **project_name** (*str*) – root name of the project
- **experiment_name** (*str*) – name of the particular experiment
- **experiment_timestamp** (*str*) – time stamp at the start of training
- **local_model_result_folder_path** (*str*) – root local path where project folder will be created

Returns None

list_additional_results_dump_paths()

Specify the list of meta data files you also want to save & upload to s3 during the experiment saving procedure

By default there are no additional files that are saved as the return is None. If you want to save your specific additional files produced during the training procedure, then override this method specifying the file paths.

If you want to save a whole folder of files, use zip_additional_results_dump() function to zip it into a single file and save this zip instead.

The specified files are any additional data you would want to include into the experiment folder in addition to the model save files and performance evaluation report files. For example a zip of attention heatmap pictures in the machine translation projects.

Returns

list of lists of string paths if it is not None. Each element of the list should be list of: [[results_file_name, results_file_local_path], ... [,]]

Return type list or None

class aitoolbox.nlp.experiment_evaluation.NLP_result_package.**GLUEResultPackage**(*task_name*)

Bases: *aitoolbox.experiment.result_package.abstract_result_packages.AbstractResultPackage*

GLUE task result package

Wrapper around HF Transformers `glue_compute_metrics()`

Parameters **task_name** (*str*) – name of the GLUE task

prepare_results_dict()

Perform result package building

Mostly this consists of executing calculation of selected performance metrics and returning their result dicts. If you want to use multiple performance metrics you have to combine them in the single self.results_dict at the end by doing this:

return {**metric_dict_1, **metric_dict_2}

Returns calculated result dict

Return type dict

class aitoolbox.nlp.experiment_evaluation.NLP_result_package.**XNLIResultPackage**

Bases: *aitoolbox.experiment.result_package.abstract_result_packages.AbstractResultPackage*

XNLI task result package

Wrapper around HF Transformers `xnli_compute_metrics()`

prepare_results_dict()

Perform result package building

Mostly this consists of executing calculation of selected performance metrics and returning their result dicts. If you want to use multiple performance metrics you have to combine them in the single self.results_dict at the end by doing this:

return {**metric_dict_1, **metric_dict_2}

Returns calculated result dict

Return type dict

attention_heatmap

class aitoolbox.nlp.experiment_evaluation.attention_heatmap.**AttentionHeatMap**(*attention_matrices*, *source_sentences*, *target_sentences*, *plot_save_dir*)

Bases: *aitoolbox.experiment.core_metrics.abstract_metric.AbstractBaseMetric*

Neural attention heatmap plotting

Parameters

- **attention_matrices** (*numpy.array or list*) – list of attention 2D matrices
- **source_sentences** (*list*) – list of corresponding source sentence text tokens
- **target_sentences** (*list*) – list of corresponding target sentence text tokens
- **plot_save_dir** (*str*) – folder path on local drive where the plots should be saved

calculate_metric()

Perform metric calculation and return it from this function

Returns return metric_result

Return type float or dict

static plot_sentence_attention(*attention_matrix*, *sentence_source*, *sentence_target*, *plot_file_path=None*)

Plot the provided attention matrix

Parameters

- **attention_matrix** (*np.array*) – 2D attention matrix
- **sentence_source** (*list*) – corresponding source sentence text tokens
- **sentence_target** (*list*) – corresponding target sentence text tokens
- **plot_file_path** (*str*) – local drive file path where to save the plotted attention matrix heatmap

Returns None

static prepare_folder_for_saving(*output_plot_dir*)
Create attention heatmaps local folder where the heatmaps will be saved

Parameters **output_plot_dir** (*str*) –

Returns path to the created folder

Return type str

6.1.5 utils

6.1.5.1 Submodules

6.1.5.1.1 dict_util

aitoolbox.utils.dict_util.**combine_prediction_metadata_batches**(*metadata_list*)

Combines a list of dicts with the same keys and lists as values into a single dict with concatenated lists for each corresponding key

Parameters **metadata_list** (*list*) – list of dicts with matching keys and lists for values

Returns combined single dict

Return type dict

aitoolbox.utils.dict_util.**flatten_dict**(*nested_dict*, *parent_key=''*, *sep='_'*)
Flatten the nested dict of dicts of ...

Parameters

- **nested_dict** (*dict*) – input dict
- **parent_key** (*str*) –
- **sep** (*str*) – separator when flattening the category

Returns flattened dict

Return type dict

aitoolbox.utils.dict_util.**combine_dict_elements**(*list_of_dicts*)
Combine into single list the elements with the same key across several dicts

Parameters **list_of_dicts** (*list*) – list of dicts with matching keys

Returns combined single dict

Return type dict

aitoolbox.utils.dict_util.**flatten_combine_dict**(*train_history*)
Flatten all dict of dicts and combine elements with the same key into a single list in the dict

Parameters **train_history** (*dict*) –

Returns

Return type dict

6.1.5.1.2 file_system

aitoolbox.utils.file_system.**create_folder_hierarchy**(*base_folder_path*, *folder_names*)
Create nested folder hierarchy

Parameters

- **base_folder_path** (*str*) – folder from which the created folder hierarchy will go into further depth
- **folder_names** (*list*) – names of folders to be created one inside the previous

Returns path to final folder in hierarchy, all the folder paths in the created hierarchy

Return type str, list

aitoolbox.utils.file_system.**zip_folder**(*source_dir_path*, *zip_path*)
Utility function for zipping a folder into .zip archive

Parameters

- **source_dir_path** (*str*) – path to the folder that is going to be zipped
- **zip_path** (*str*) – specify the path of the zip file which will be created

Returns the full path to the produced zip file (with the .zip extension appended)

Return type str

aitoolbox.utils.file_system.**unzip_file**(*file_path*, *target_dir_path*)
Util function for zip file unzipping

Parameters

- **file_path** (*str*) – path to the zip file
- **target_dir_path** (*str*) – destination where unzipped content is stored

6.1.5.1.3 util

aitoolbox.utils.util.**function_exists**(*object_to_check*, *fn_name*)
Check if function exists in the object

Parameters

- **object_to_check** – object to be searched for the existence of the function
- **fn_name** (*str*) – name of the function

Returns if function is present in the provided object

Return type bool

aitoolbox.utils.util.**copy_function**(*fn*)
Deep copy a function

Based on http://stackoverflow.com/a/6528148/190597 (Glenn Maynard)

Parameters **fn** (*callable*) – original function

Returns copy of the provided function

Return type callable

aitoolbox.utils.util.**is_empty_function**(*fn*)
Returns true if f is an empty function

Taken from StackOverflow: https://stackoverflow.com/a/58973125

Parameters **fn** – function to be evaluated if it is empty or not

Returns true if provided function is empty, otherwise false

Return type bool

aitoolbox.utils.util.**flatten_list_of_lists**(*nested_list*)
Flatten the nested list of lists

Parameters **nested_list** (*list*) – nested list of lists to be flattened

Returns flattened list

Return type list or None

AIToolbox is a framework which helps you train deep learning models in PyTorch and quickly iterate experiments. It hides the repetitive technicalities of training the neural nets and frees you to focus on interesting part of devising new models. In essence, it offers a keras-style train loop abstraction which can be used for higher level training process while still allowing the manual control on the lower level when desired.

In addition to orchestrating the model training loop the framework also helps you keep track of different experiments by automatically saving models in a structured traceable way and creating performance reports. These can be stored both locally or on AWS S3 (Google Cloud Storage in beta) which makes the library very useful when training on the GPU instance on AWS. Instance can be automatically shut down when training is finished and all the results are safely stored on S3.

CHAPTER

SEVEN

MAIN COMPONENTS

AIToolbox consists of three main user-facing components:

- `aitoolbox.torchtrain` - PyTorch train loop engine
- `aitoolbox.experiment` - experiment tracking
- `aitoolbox.cloud` - cloud operations for *AWS* and *Google Cloud*

All three AIToolbox components can be used independently when only some subset of functionality is desired in a project. However, the greatest benefit of AIToolbox comes when all components are used together in unison in order to ease the process of PyTorch model training and experiment tracking as much as possible. Most of this top-level API is exposed to the user via the functionality implemented in `aitoolbox.torchtrain`.

CHAPTER

EIGHT

INSTALLATION

To install the AIToolbox package execute:

```
pip install aitoolbox
```

If you want to install the most recent version from github repository, first clone the package repository and then install via the *pip* command:

```
git clone https://github.com/mv1388/aitoolbox.git

pip install ./aitoolbox
```

AIToolbox package can be also provided as a dependency in the `requirements.txt` file. This can be done by just specifying the `aitoolbox` dependency. On the other hand, to automatically download the current master branch from github include the following dependency specification in the requirements.txt:

```
git+https://github.com/mv1388/aitoolbox#egg=aitoolbox
```

CHAPTER

NINE

DOCUMENTATION SECTIONS:

To learn more about components available in AIToolbox have a look at the corresponding documentation sections:

- *torchtrain*
- *experiment*
- *cloud*
- *nlp*

PYTHON MODULE INDEX

a

INDEX

Symbols

A

B

C

D

E

F

G

H

I

K

L

M

N

O

P

Q

R

S

T

U

V

W

X

Z