pytorch-igniter

Release 0.1.0

Dec 30, 2020

Contents:

CHAPTER 1

pytorch-igniter

Simplify pytorch training using a configurable wrapper around pytorch-ignite

1.1 Installation

```
pip install pytorch-igniter
```

1.2 Demo

See pytorch-igniter-demo

1.3 Documentation

View latest documentation at ReadTheDocs

1.4 Continuous Integration

View continuous integration at TravisCI

1.5 PyPI

View releases on PyPI

1.6 GitHub

View source code on GitHub

GitHub tags are automatically released on ReadTheDocs, tested on TravisCI, and deployed to PyPI if successful.

CHAPTER 2

Training

`pytorch-igniter` constructs a training engine so you can focus on machine learning.

2.1 Features

- Only create a model and write functions that train and evaluate on a single batch. `pytorch-igniter` constructs the training engine that checkpoints your model while training, evaluating, and logging.
- Standardized and documented `argparse` command-line arguments like `--batch-size`, `--max-epochs`, and `--learning-rate`. Only write custom arguments that are unique to your script.
- Save model on `ctrl-C` or `kill`. Automatically resume model from latest checkpoint. Configurable checkpointing.
- Simplify defining metrics. Metrics can average or otherwise accumulate data and can be saved, printed, and more depending on configuration.
- Integrate with MLflow for tracking training runs, including hyperparameters and metrics.
- Integrate with AWS SageMaker using aws-sagemaker-remote for tracking training runs and executing training remotely on managed containers.

2.2 Basic Usage

```
import torch
import torch.nn as nn
import torch.optim as optim
from torch.utils.data import DataLoader

from torchvision.datasets import MNIST
import torchvision.transforms as transforms
```

(continues on next page)

(continued from previous page)

```
from pytorch_igniter import train, RunSpec
from pytorch_igniter.args import train_kwargs
from pytorch_igniter.main import igniter_main


def main(
    args
):

    # Create data loaders
    train_loader = DataLoader(...)
    eval_loader = DataLoader(...)

    # Create model, optimizer, and criteria
    model = nn.Sequential(...)
    optimizer = torch.optim.Adam(...)
    criteria = nn.CrossEntropyLoss(...)

    # Single step of training
    def train_step(engine, batch):
        # Do training
        model.train()
        model.zero_grad()
        inputs, labels = batch
        outputs = model(inputs)
        loss = criteria(input=outputs, target=labels)
        loss.backward()
        optimizer.step()
        return {
            "loss": loss
        }

    # Single step of evaluation
    def eval_step(engine, batch):
        # Do evaluation
        model.eval()
        inputs, labels = batch
        outputs = model(inputs)
        loss = criteria(input=outputs, target=labels)
        return {
            "loss": loss
        }

    # Metrics average the outputs of the step functions and are printed and saved to␣
↪logs
    metrics = {
        'loss': 'loss'
    }

    # Objects to save
    to_save = {
        "model": model,
        "optimizer": optimizer
    }

    train(
        to_save=to_save,
```

(continues on next page)

(continued from previous page)

```
        # Training setup
        train_spec=RunSpec(
            step=train_step,
            loader=train_loader,
            metrics=metrics
        ),
        # Evaluation setup
        eval_spec=RunSpec(
            step=eval_step,
            loader=eval_loader,
            metrics=metrics
        ),
        **train_kwargs(args),
        parameters=vars(args)
    )


if __name__ == "__main__":
    igniter_main(
        main=main,
        inputs={
            'data': 'data'
        },
        # ...
    )
```

2.3 Command-Line Arguments

Note that additional command-line arguments are generated for each item in `inputs` and `dependencies` function arguments.

```
usage: pytorch-igniter [-h] [--max-epochs N] [--n-saved N_SAVED]
                       [--save-event SAVE_EVENT] [--train-pbar [TRAIN_PBAR]]
                       [--train-print-event TRAIN_PRINT_EVENT]
                       [--train-log-event TRAIN_LOG_EVENT]
                       [--sagemaker-profile SAGEMAKER_PROFILE]
                       [--sagemaker-run [SAGEMAKER_RUN]]
                       [--sagemaker-wait [SAGEMAKER_WAIT]]
                       [--sagemaker-spot-instances [SAGEMAKER_SPOT_INSTANCES]]
                       [--sagemaker-script SAGEMAKER_SCRIPT]
                       [--sagemaker-source SAGEMAKER_SOURCE]
                       [--sagemaker-training-instance SAGEMAKER_TRAINING_INSTANCE]
                       [--sagemaker-training-image SAGEMAKER_TRAINING_IMAGE]
                       [--sagemaker-training-image-path SAGEMAKER_TRAINING_IMAGE_PATH]
                       [--sagemaker-training-image-accounts SAGEMAKER_TRAINING_IMAGE_
↪ACCOUNTS]
                       [--sagemaker-training-role SAGEMAKER_TRAINING_ROLE]
                       [--sagemaker-base-job-name SAGEMAKER_BASE_JOB_NAME]
                       [--sagemaker-job-name SAGEMAKER_JOB_NAME]
                       [--sagemaker-experiment-name SAGEMAKER_EXPERIMENT_NAME]
                       [--sagemaker-trial-name SAGEMAKER_TRIAL_NAME]
                       [--sagemaker-volume-size SAGEMAKER_VOLUME_SIZE]
                       [--sagemaker-max-run SAGEMAKER_MAX_RUN]
                       [--sagemaker-max-wait SAGEMAKER_MAX_WAIT]
```

(continues on next page)

(continued from previous page)

```
                        [--sagemaker-output-json SAGEMAKER_OUTPUT_JSON]
                        [--model-dir MODEL_DIR] [--output-dir OUTPUT_DIR]
                        [--checkpoint-dir CHECKPOINT_DIR]
                        [--sagemaker-checkpoint-s3 SAGEMAKER_CHECKPOINT_S3]
                        [--sagemaker-checkpoint-container SAGEMAKER_CHECKPOINT_
↪CONTAINER]
```

2.3.1 Named Arguments

--max-epochs number of epochs to train (default: 10)

Default: 10

--n-saved Number of checkpoints to keep (default: `10`)

Default: 10

--save-event save event

Default: "EPOCH_COMPLETED"

--train-pbar Enable train progress bar

Default: True

--train-print-event training print event

--train-log-event training log event

--model-dir Directory to save final model (default: output/model)

Default: "output/model"

--output-dir Directory for logs, images, or other output files (default: "output/output")

Default: "output/output"

2.3.2 SageMaker

SageMaker options

--sagemaker-profile AWS profile for SageMaker session (default: [default])

Default: "default"

--sagemaker-run Run training on SageMaker (yes/no default=False)

Default: False

--sagemaker-wait Wait for SageMaker training to complete and tail logs files (yes/no default=True)

Default: True

--sagemaker-spot-instances Use spot instances for training (yes/no default=False)

Default: False

--sagemaker-script Script to run on SageMaker. (default: [script.py])

Default: "script.py"

--sagemaker-source Source to upload to SageMaker. Must contain script. If blank, default to directory containing script. (default: [])

Default: “”

--sagemaker-training-instance Instance type for training

Default: “ml.m5.large”

--sagemaker-training-image Docker image for training

Default: “aws-sagemaker-remote-training:latest”

--sagemaker-training-image-path Path to dockerfile if image does not exist

Default: “/home/docs/checkouts/readthedocs.org/user_builds/pytorch-igniter/envs/latest/lib/python3.7/site-packages/aws_sagemaker_remote/ecr/training”

--sagemaker-training-image-accounts Accounts for docker build

Default: [‘763104351884’]

--sagemaker-training-role Docker image for training

Default: “aws-sagemaker-remote-training-role”

--sagemaker-base-job-name Base job name for tracking and organization on S3. A job name will be generated from the base job name unless a job name is specified.

Default: “training-job”

--sagemaker-job-name Job name for tracking. Use –base-job-name instead and a job name will be automatically generated with a timestamp.

Default: “”

--sagemaker-experiment-name Name of experiment in SageMaker tracking.

--sagemaker-trial-name Name of experiment trial in SageMaker tracking.

--sagemaker-volume-size Volume size in GB.

Default: 30

--sagemaker-max-run Maximum runtime in seconds.

Default: 43200

--sagemaker-max-wait Maximum time to wait for spot instances in seconds.

Default: 86400

--sagemaker-output-json Output job details to JSON file.

2.3.3 Checkpoints

Checkpointing options

--checkpoint-dir Local directory to store checkpoints for resuming training (default: “output/checkpoint”)

Default: “output/checkpoint”

--sagemaker-checkpoint-s3 Location to store checkpoints on S3 or “default” (default: “default”)

Default: “default”

--sagemaker-checkpoint-container Location to store checkpoints on container (default: "/opt/ml/checkpoints")

Default: "/opt/ml/checkpoints"

See `aws-sagemaker-remote` documentation for SageMaker option documentation.

CHAPTER 3

pytorch_igniter

3.1 pytorch_igniter package

3.1.1 Subpackages

pytorch_igniter.demo package

Submodules

pytorch_igniter.demo.mnist_model module

class pytorch_igniter.demo.mnist_model.**MnistModel**
Bases: torch.nn.modules.module.Module

forward(*input*)
Defines the computation performed at every call.

Should be overridden by all subclasses.

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

Module contents

pytorch_igniter.inference package

Submodules

pytorch_igniter.inference.audio_input_fn module

pytorch_igniter.inference.greyscale_image_input_fn module

pytorch_igniter.inference.greyscale_image_input_fn.**input_fn**(*request_body*, *request_content_type*)

Convert input for your model

pytorch_igniter.inference.image_input_fn module

pytorch_igniter.inference.image_input_fn.**input_fn**(*request_body*, *request_content_type*)

Convert input for your model

pytorch_igniter.inference.inference module

pytorch_igniter.inference.inference.**model_fn**(*model_dir*)

Load your model from model_dir

pytorch_igniter.inference.inference.**predict_fn**(*input_data*, *model*)

Run your model

pytorch_igniter.inference.json_output_fn module

class pytorch_igniter.inference.json_output_fn.**TensorEncoder**(*, *skipkeys=False*, *ensure_ascii=True*, *check_circular=True*, *allow_nan=True*, *sort_keys=False*, *indent=None*, *separators=None*, *default=None*)

Bases: `json.encoder.JSONEncoder`

default(*obj*)

Implement this method in a subclass such that it returns a serializable object for `o`, or calls the base implementation (to raise a `TypeError`).

For example, to support arbitrary iterators, you could implement default like this:

```
def default(self, o):
    try:
        iterable = iter(o)
    except TypeError:
        pass
    else:
        return list(iterable)
    # Let the base class default method raise the TypeError
    return JSONEncoder.default(self, o)
```

pytorch_igniter.inference.json_output_fn.**output_fn**(*prediction*, *content_type*)

Convert model output

Module contents

3.1.2 Submodules

3.1.3 pytorch_igniter.args module

pytorch_igniter.args.**add_eval_args**(*parser*, *args*)

pytorch_igniter.args.**add_mlflow_args**(*parser*)

pytorch_igniter.args.**add_model_args**(*parser*, *args*)

pytorch_igniter.args.**add_train_args**(*parser*, *args*)

pytorch_igniter.args.**add_tran_and_eval_args**(*parser*)

pytorch_igniter.args.**eval_args**(*parser: argparse.ArgumentParser*)

pytorch_igniter.args.**fix_device**(*args*)

pytorch_igniter.args.**mlflow_args**(*parser*, *mlflow_enable=True*, *mlflow_tracking_uri=None*, *mlflow_tracking_password=None*, *mlflow_tracking_username=None*, *mlflow_tracking_secret_name=None*, *mlflow_tracking_secret_region=None*, *mlflow_tracking_secret_profile=None*, *mlflow_experiment_name='default'*, *mlflow_run_name=None*)

pytorch_igniter.args.**model_args**(*parser*, *device=None*)

pytorch_igniter.args.**parser_for_docs**()

pytorch_igniter.args.**train_and_eval_args**(*parser: argparse.ArgumentParser*, *eval_event='EPOCH_COMPLETED'*)

pytorch_igniter.args.**train_args**(*parser*, *max_epochs=10*, *n_saved=10*, *save_event='EPOCH_COMPLETED'*)

pytorch_igniter.args.**train_kwargs**(*args*)

3.1.4 pytorch_igniter.commands module

class pytorch_igniter.commands.**EvalCommand**(*igniter_config: pytorch_igniter.config.IgniterConfig*, *script*, *cmd='eval'*, ***kwargs*)

Bases: aws_sagemaker_remote.training.main.TrainingCommand

argparse_callback(*parser*)

runner(*args*)

class pytorch_igniter.commands.**TrainAndEvalCommand**(*igniter_config: pytorch_igniter.config.IgniterConfig*, *script*, *cmd='train-and-eval'*, ***kwargs*)

Bases: aws_sagemaker_remote.training.main.TrainingCommand

argparse_callback(*parser*)

runner(*args*)

class pytorch_igniter.commands.**TrainCommand**(*igniter_config: pytorch_igniter.config.IgniterConfig*, *script*, *cmd='train'*, ***kwargs*)

Bases: aws_sagemaker_remote.training.main.TrainingCommand

argparse_callback(*parser*)

runner(*args*)

3.1.5 pytorch_igniter.config module

class pytorch_igniter.config.**IgniterConfig**(*model_args=None*, *train_args=None*, *eval_args=None*, *description=None*, *train_inputs=None*, *eval_inputs=None*, *make_model=None*, *make_evaluator=None*, *make_trainer=None*, *inference_spec=None*, ***igniter_args*)

Bases: object

3.1.6 pytorch_igniter.engine module

pytorch_igniter.engine.**build_engine**(*spec: pytorch_igniter.spec.RunSpec*, *output_dir=None*, *trainer=None*, *metric_cls=<class 'ignite.metrics.running_average.RunningAverage'>*, *tag=''*, *mlflow_logger=None*, *is_training=None*, *device=None*)

3.1.7 pytorch_igniter.evaluator module

pytorch_igniter.evaluator.**dummy_step**(*engine*, *batch*)

pytorch_igniter.evaluator.**evaluate**(*eval_spec: pytorch_igniter.spec.RunSpec*, *output_dir=None*, *model_dir=None*, *to_load=None*, *tag='eval'*, *mlflow_enable=True*, *mlflow_tracking_uri=None*, *trainer=None*)

3.1.8 pytorch_igniter.experiment_cli module

pytorch_igniter.experiment_cli.**experiment_cli**(*script*, *config: pytorch_igniter.config.IgniterConfig*, *extra_commands=None*, *description=None*, *dry_run=False*, ***kwargs*)

pytorch_igniter.experiment_cli.**experiment_cli_commands**(*script*, *config: pytorch_igniter.config.IgniterConfig*, *extra_commands=None*, ***kwargs*)

3.1.9 pytorch_igniter.hooks module

class pytorch_igniter.hooks.**ConcatenateOutputsHook**(*tracked_outputs*, *axis=-1*)

Bases: object

attach(*engine: ignite.engine.engine.Engine*, *start_event=<Events.ITERATION_COMPLETED: 'iteration_completed'>*, *iteration_event=<Events.ITERATION_COMPLETED: 'iteration_completed'>*)

clear(*engine*)

collect(*engine*)

concatenate()

3.1.10 pytorch_igniter.main module

pytorch_igniter.main.**igniter_main**(*main*, *training_args=None*, ***sagemaker_args*)

Run a training script

Parameters

- **main** (*function*) – Main function. Must have one argument `parser:argparse.Namespace`.
- **training_args** (*dict*) – Keyword arguments to *pytorch_igniter.args.train_args()*.
- **sagemaker_args** (*dict*) – Keyword arguments to `aws-sagemaker-remote.training.main.sagemaker_training_main`. See aws_sagemaker_remote.

3.1.11 pytorch_igniter.metrics module

class pytorch_igniter.metrics.**SafeAverage**(*output_transform: Callable = <function Average.<lambda>>*, *device: Union[str, torch.device, None] = None*)

Bases: `ignite.metrics.accumulation.Average`

compute() → Union[Any, torch.Tensor, numbers.Number]

Computes the metric based on it's accumulated state.

By default, this is called at the end of each epoch.

Returns

the actual quantity of interest. However, if a `Mapping` is returned, it will be (shallow) flattened into *engine.state.metrics* when `completed()` is called.

Return type Any

Raises `NotComputableError` – raised when the metric cannot be computed.

3.1.12 pytorch_igniter.mlflow_ctx module

class pytorch_igniter.mlflow_ctx.**NullContext**

Bases: `object`

pytorch_igniter.mlflow_ctx.**get_mlflow_logger**(*output_dir=None*, *checkpoint_dir=None*, *mlflow_enable=True*)

pytorch_igniter.mlflow_ctx.**mlflow_ctx**(*checkpoint_dir=None*, *output_dir=None*, *run_id=None*, *mlflow_enable=True*, *allow_new=True*, *experiment_name=None*, *run_name=None*, *parameters=None*, *is_sagemaker=None*, *sagemaker_job_name=None*)

3.1.13 pytorch_igniter.spec module

class pytorch_igniter.spec.**InferenceSpec**(*inferencer*, *requirements=None*, *dependencies=None*, *input_fn=None*, *output_fn=None*)

Bases: object

class pytorch_igniter.spec.**RunSpec**(*loader*, *step*, *metrics*, *max_epochs=1*, *epoch_length=None*, *callback=None*, *enable_pbar=True*, *pbar_metrics='all'*, *print_event='default'*, *print_metrics='all'*, *print_fmt='default'*, *print_metric_fmt=' | {name}: {value}'*, *log_event='default'*, *log_metrics='all'*, *plot_event='default'*, *plot_metrics='all'*, *enable_timer=True*)

Bases: object

classmethod eval_spec(*loader*, *model*, *criteria*, *preproc=None*, ***kwargs*)

set_defaults(*is_training=True*)

Fill in the default events for training or evaluation specs

classmethod train_spec(*loader*, *model*, *criteria*, *optimizer*, *preproc=None*, ***kwargs*)

3.1.14 pytorch_igniter.ssm module

pytorch_igniter.ssm.**get_secret**(*profile_name='default'*, *secret_name='mlflow-secret'*, *region_name=None*)

3.1.15 pytorch_igniter.trainer module

pytorch_igniter.trainer.**train**(*to_save*, *model*, *train_spec: pytorch_igniter.spec.RunSpec*, *eval_spec: pytorch_igniter.spec.RunSpec = None*, *eval_event=<Events.EPOCH_COMPLETED: 'epoch_completed'>*, *save_event=<Events.EPOCH_COMPLETED: 'epoch_completed'>*, *n_saved=10*, *mlflow_enable=True*, *mlflow_tracking_uri=None*, *mlflow_tracking_username=None*, *mlflow_tracking_password=None*, *mlflow_tracking_secret_name=None*, *mlflow_tracking_secret_profile=None*, *mlflow_tracking_secret_region=None*, *mlflow_experiment_name=None*, *mlflow_run_name=None*, *model_dir='output'*, *checkpoint_dir='output'*, *output_dir='output'*, *parameters=None*, *device=None*, *max_epochs=None*, *is_sagemaker=False*, *sagemaker_job_name=None*, *inference_spec=None*, *inference_args=None*, *eval_pbar=None*, *train_pbar=None*, *train_print_event=None*, *eval_print_event=None*, *eval_log_event=None*, *train_log_event=None*)

Train a model

3.1.16 pytorch_igniter.util module

class pytorch_igniter.util.**StateDictStorage**

Bases: `torch.nn.modules.module.Module`

load_state_dict(*state_dict*, *strict=True*)

Copies parameters and buffers from *state_dict* into this module and its descendants. If `strict` is `True`, then the keys of *state_dict* must exactly match the keys returned by this module's `state_dict()` function.

Parameters

- **state_dict** (*dict*) – a dict containing parameters and persistent buffers.
- **strict** (*bool, optional*) – whether to strictly enforce that the keys in *state_dict* match the keys returned by this module's `state_dict()` function. Default: `True`

Returns

- **missing_keys** is a list of str containing the missing keys
- **unexpected_keys** is a list of str containing the unexpected keys

Return type `NamedTuple` with `missing_keys` and `unexpected_keys` fields

state_dict()

Returns a dictionary containing a whole state of the module.

Both parameters and persistent buffers (e.g. running averages) are included. Keys are corresponding parameter and buffer names.

Returns a dictionary containing a whole state of the module

Return type dict

Example:

```
>>> module.state_dict().keys()
['bias', 'weight']
```

pytorch_igniter.util.**apply_to_tensors**(*tensors*, *fn*)

pytorch_igniter.util.**auto_metric**(*value*, *cls=<class 'ignite.metrics.running_average.RunningAverage'>*)

pytorch_igniter.util.**capture_signals**(*signals=None*, *callback=None*, *die=False*, ***kwargs*)

pytorch_igniter.util.**chain_callbacks**(*callbacks=None*, ***kwargs*)

pytorch_igniter.util.**create_plots**(*engine*, *logs_fname*, *plots_fname*, *metric_names='all'*)

pytorch_igniter.util.**find_last_checkpoint**(*output_dir*)

pytorch_igniter.util.**get_last_checkpoint**(*checkpoint_handler: ignite.handlers.checkpoint.ModelCheckpoint*)

pytorch_igniter.util.**get_mean_value**(*key*)

pytorch_igniter.util.**get_metrics**(*engine*, *metric_names='all'*)

pytorch_igniter.util.**get_value**(*key*)

pytorch_igniter.util.**handle_exception**(*engine*, *e*, *callback=None*, ***kwargs*)

pytorch_igniter.util.**image_saver**(*engine*, *output_path*, *fn*)

pytorch_igniter.util.**image_saver_callback**(*output_path*, *images*)
Add a callback to save images

pytorch_igniter.util.**kill_signals**()

pytorch_igniter.util.**load_from**(*model_dir*, *to_load*)

pytorch_igniter.util.**print_logs**(*engine*, *trainer=None*, *fmt='[{epoch}/{max_epochs}][{i}/{max_i}]'*, *metric_fmt=' | {name}: {value}'*, *metric_names='all'*)

pytorch_igniter.util.**save_logs**(*engine*, *fname*, *trainer=None*, *metric_names='all'*)

pytorch_igniter.util.**tensors_to_device**(*device*)

pytorch_igniter.util.**tensors_to_items**(*tensors*)

pytorch_igniter.util.**tensors_to_numpy**(*tensors*)

pytorch_igniter.util.**timer_metric**(*engine*, *name='timer'*)

3.1.17 Module contents

CHAPTER 4

Indices and tables

- genindex
- modindex
- search

Python Module Index

p

Index

H

I

K

L

M

N

O

P

R

S

T