aws-sagemaker-remote

Release 0.0.1

Jan 07, 2021

Contents

CHAPTER 1

aws-sagemaker-remote

Remotely run and track ML research using AWS SageMaker.

- Standardized command line flags
- Remotely run scripts with minimal changes
- Automatically manage AWS resources
- All code, inputs, outputs, arguments, and settings are tracked in one place
- Reproducible batch processing jobs to prepare datasets
- Reproducible training jobs that track hyperparameters and metrics

Track three types of objects in a standard way:

- Processing jobs consume file inputs and produce file outputs. Useful for data conversion, extraction, etc.
- Training jobs train models while tracking metrics and hyperparameters.
- Inference models provide predictions and can be deployed on endpoints. Can be automatically created from and linked to training jobs for tracking purposes or can deploy externally-created models.

1.1 Installation

1.1.1 Release

```
pip install aws-sagemaker-remote
```

1.1.2 Development

```
git clone https://github.com/bstriner/aws-sagemaker-remote
cd aws-sagemaker-remote
python setup.py develop
```

1.2 Documentation

View latest documentation at ReadTheDocs

1.3 Continuous Integration

View continuous integration at TravisCI

1.4 PyPI

View releases on PyPI

1.5 GitHub

View source code on GitHub

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

CHAPTER 2

Processing

Processing jobs accept a set of one or more input file paths and write to a set of one or more output file paths. Ideal for file conversion or other data preparation tasks.

- Running locally, standard command line arguments for inputs and outputs are used as usual
- Running remotely, data is uploaded and downloaded using S3 for tracking

2.1 Basic usage

Write a script with a `main` function that calls `sagemaker_processing_main`.

```
from aws_sagemaker_remote import sagemaker_processing_main

def main(args):
    # your code here
    pass

if __name__ == '__main__':
    sagemaker_processing_main(
        main=main,
        # ...
    )
```

Pass function argument `run=True` or command line argument `--sagemaker-run=True` to run script remotely on SageMaker.

- Many command-line arguments are automatically added. See *Command-Line Arguments*.
- Parameters to `sagemaker_processing_main` control what command-line arguments are automatically added and the default values. See *aws_sagemaker_remote.processing.main.sagemaker_processing_main()* and *aws_sagemaker_remote.processing.args.sagemaker_processing_args()*

2.2 Processing Job Tracking

Use the SageMaker console to view a list of all processing jobs. For each job, SageMaker tracks:

- Processing time
- Container used
- Link to CloudWatch logs
- Path on S3 for each of:
  - Script file
  - Each input channel
  - Each output channel
  - Requirements file (if used)
  - Configuration script (if used)
  - Supporting code (if used)

2.3 Configuration

Many command line options are added by this command.

Option `--sagemaker-run` controls local or remote execution.

- Set `--sagemaker-run` to a falsy value (no,false,0), the script will call your main function as usual and run locally.
- Set `--sagemaker-run` to a truthy value (yes,true,1), the script will upload itself and any requirements or inputs to S3, execute remotely on SageMaker, and save outputs to S3, logging results to the terminal.

Set `--sagemaker-wait` truthy to tail logs and wait for completion or falsy to complete when the job starts.

Defaults are set through code. Defaults can be overwritten on the command line. For example:

- Use the function argument `image` to set the default container image for your script
- Use the command line argument `--sagemaker-image` to override the container image on a particular run

See **functions** and **commands** (todo: links)

2.4 Environment Customization

The environment can be customized in multiple ways.

- Instance
  - Function argument `instance`
  - Command line argument `--sagemaker-instance`
  - Select instance type of machine running the container
- Image
  - Function argument `image`

  - Command line argument `--sagemaker-image`
  - Accepts URI of Docker container image on ECR to run
  - Build a custom Docker image for major customizations
- Configuration script
  - Function argument `configuration_script`
  - Command line argument `--sagemaker-configuration-script`
  - Accepts path to a text file. Will upload text file to S3 and run `source [file]`.
  - Bash script file for minor customization, e.g., `export MYVAR=value` or `yum install -y mypackage`
- Configuration command
  - Function argument `configuration_command`
  - Command line argument `--sagemaker-configuration-command`
  - Accepts a bash command to run.
  - Bash command for minor customization, e.g., `export MYVAR=value && yum install -y mypackage`
- Requirements file
  - Function argument `requirements`
  - Command line argument `--sagemaker-requirements`
  - Accepts path to a text file. Will upload text file to S3 and run `python -m pip install -r [file]`
  - Use for installing Python packages by listing one on each line. Standard `requirements.txt` file format [https://pip.pypa.io/en/stable/reference/pip_install/#requirements-file-format]
- Module uploads
  - Function argument `modules`
    - Dictionary of `[key]->[value]`
    - Each key will create command line argument `--key` that defaults to `value`
  - Each `value` is a directory containing a Python module that will be uploaded to S3, downloaded to SageMaker, and put on the PYTHONPATH
  - For example, if directory `mymodule` contains the files `__init__.py` and `myfile.py` and `myfile.py` contains `def myfunction():...`, pass `modules={'mymodule':'path/to/mymodule'}` to `sagemaker_processing_main` and then use `from mymodule.myfile import myfunction` in your script.
  - Use module uploads for supporting code that is not being installed from packages.

2.5 Additional arguments

Any arguments passed to your script locally on the command line are passed to your script remotely and tracked by SageMaker
Internally, `sagemaker_processing_main` uses `argparse`. To add additional command-line flags:

- Pass a list of kwargs dictionaries to `additional_arguments`

```
sagemaker_processing_main(
  #...
  additional_arguments = [
    {
      'dest': '--filter-width',
      'default':32,
      'help':'Filter width'
    },
    {
      'dest':'--filter-height',
      'default':32,
      'help':'Filter height'
    }
  ]
)
```

- Pass a callback to `argparse_callback`

```
from argparse import ArgumentParser
def argparse_callback(parser:ArgumentParser):
  parser.add_argument(
  '--filter-width',
  default=32,
  help='Filter width')
  parser.add_argument(
  '--filter-height',
  default=32,
  help='Filter height')
sagemaker_training_main(
  # ...
  argparse_callback=argparse_callback
)
```

Note: local command-line arguments are parsed, stored on SageMaker, then used to generate a command line for your script. - All flags are serialized into a string to string dictionary. - All flags must have a single non-empty argument. - Use CSV, JSON, or other methods to use string arguments instead of repeated arguments. - Explicity passing empty arguments on the command-line is not supported.

2.6 Command-Line Arguments

These command-line arguments were created using the following parameters. Command-line arguments are generated for each item in `inputs`, `outputs` and `dependencies`.

```
inputs={
    'input': '/path/to/input'
},
outputs={
    'output': ('/path/to/output', 'default')
},
dependencies={
    'my_module': '/path/to/my_module'
}
```

```
usage: aws-sagemaker-remote-processing [-h]
                                       [--sagemaker-profile SAGEMAKER_PROFILE]
                                       [--sagemaker-run [SAGEMAKER_RUN]]
                                       [--sagemaker-wait [SAGEMAKER_WAIT]]
                                       [--sagemaker-script SAGEMAKER_SCRIPT]
                                       [--sagemaker-python SAGEMAKER_PYTHON]
                                       [--sagemaker-job-name SAGEMAKER_JOB_NAME]
                                       [--sagemaker-base-job-name SAGEMAKER_BASE_JOB_
↪NAME]
                                       [--sagemaker-runtime-seconds SAGEMAKER_RUNTIME_
↪SECONDS]
                                       [--sagemaker-role SAGEMAKER_ROLE]
                                       [--sagemaker-requirements SAGEMAKER_
↪REQUIREMENTS]
                                       [--sagemaker-configuration-script SAGEMAKER_
↪CONFIGURATION_SCRIPT]
                                       [--sagemaker-configuration-command SAGEMAKER_
↪CONFIGURATION_COMMAND]
                                       [--sagemaker-image SAGEMAKER_IMAGE]
                                       [--sagemaker-image-path SAGEMAKER_IMAGE_PATH]
                                       [--sagemaker-image-accounts SAGEMAKER_IMAGE_
↪ACCOUNTS]
                                       [--sagemaker-instance SAGEMAKER_INSTANCE]
                                       [--sagemaker-volume-size SAGEMAKER_VOLUME_SIZE]
                                       [--sagemaker-output-json SAGEMAKER_OUTPUT_JSON]
                                       [--sagemaker-input-mount SAGEMAKER_INPUT_MOUNT]
                                       [--input INPUT]
                                       [--input-mode INPUT_MODE]
                                       [--sagemaker-output-mount SAGEMAKER_OUTPUT_
↪MOUNT]
                                       [--output OUTPUT]
                                       [--output-s3 OUTPUT_S3]
                                       [--output-mode OUTPUT_MODE]
                                       [--sagemaker-module-mount SAGEMAKER_MODULE_
↪MOUNT]
                                       [--my-module MY_MODULE]
```

2.6.1 SageMaker

SageMaker options

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

Default: "default"

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

Default: False

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

Default: True

--sagemaker-script Python script to execute (default: [script.py])

Default: "script.py"

--sagemaker-python Python executable to use in container (default: [python3])

Default: "python3"

--sagemaker-job-name Job name for SageMaker processing. If not provided, will be generated from base job name. Leave blank for most use-cases. (default: [])

Default: ""

--sagemaker-base-job-name Base job name for SageMaker processing .Job name will be generated from the base name and a timestamp (default: [processing-job])

Default: "processing-job"

--sagemaker-runtime-seconds SageMaker maximum runtime in seconds (default: [3600])

Default: 3600

--sagemaker-role AWS role for SageMaker execution (default: [aws-sagemaker-remote-processing-role])

Default: "aws-sagemaker-remote-processing-role"

--sagemaker-requirements Requirements file to install on SageMaker (default: [None])

--sagemaker-configuration-script Bash configuration script to source on SageMaker (default: [None])

--sagemaker-configuration-command Bash command to run on SageMaker for configuration (e.g., `pip install aws_sagemaker_remote && export MYVAR=MYVALUE`) (default: [None])

--sagemaker-image AWS ECR image URI of Docker image to run SageMaker processing (default: [aws-sagemaker-remote-processing:latest])

Default: "aws-sagemaker-remote-processing:latest"

--sagemaker-image-path Path to Dockerfile if image does not exist yet (default: [/home/docs/checkouts/readthedocs.org/user_builds/aws-sagemaker-remote/checkouts/stable/aws_sagemaker_remote/ecr/processing])

Default: "/home/docs/checkouts/readthedocs.org/user_builds/aws-sagemaker-remote/checkouts/stable/aws_sagemaker_remote/ecr/processing"

--sagemaker-image-accounts Accounts required to build Dockerfile (default: [763104351884])

Default: "763104351884"

--sagemaker-instance AWS SageMaker instance to run processing (default: [ml.t3.medium])

Default: "ml.t3.medium"

--sagemaker-volume-size AWS SageMaker volume size in GB (default: [30])

Default: 30

--sagemaker-output-json Write SageMaker training details to JSON file (default: [None])

2.6.2 Inputs

Input options

--sagemaker-input-mount Mount point for inputs. If running on SageMaker, inputs are mounted here. If running locally, S3 inputs are downloaded here. No effect on local inputs when running locally. (default: [/opt/ml/processing/input])

Default: "/opt/ml/processing/input"

--input Input [input]. Local path or path on S3. If running locally, local paths are used directly. If running locally, S3 paths are downloaded to [*–sagemaker-input-mount*/input]. If running on SageMaker, local paths are uploaded to S3 then S3 data is downloaded to [*–sagemaker-input-mount*/input]. If running on SageMaker, S3 paths are downloaded to [*–sagemaker-input-mount*/input]. (default: [/path/to/input])

Default: "/path/to/input"

--input-mode Input [input] mode. File or Pipe. (default: [File])

Default: "File"

2.6.3 Output

Output options

--sagemaker-output-mount Mount point for outputs. If running on SageMaker, outputs written here are uploaded to S3. If running locally, S3 outputs written here are uploaded to S3. No effect on local outputs when running locally. (default: [/opt/ml/processing/output])

Default: "/opt/ml/processing/output"

--output Output [output] local path. If running locally, set to a local path. (default: [/path/to/output])

Default: "/path/to/output"

--output-s3 Output [output] S3 URI. Upload results to this URI. Empty string automatically generates a URI. (default: [default])

Default: "default"

--output-mode Output [output] mode. Set to Continuous or EndOfJob. (default: [EndOfJob])

Default: "EndOfJob"

2.6.4 Modules

Module options

--sagemaker-module-mount Mount point for modules. If running on SageMaker, modules are mounted here and this directory is added to PYTHONPATH (default: [/opt/ml/processing/modules])

Default: "/opt/ml/processing/modules"

--my-module Directory of [my_module] module. If running on SageMaker, modules are uploaded and placed on PYTHONPATH. (default: [/path/to/my_module])

Default: "/path/to/my_module"

2.7 Example Code

The following example creates a processor with no inputs and one output named `output`.

- Running the file without arguments will run locally. The argument `--output` sets the output directory.

- Running the file with `--sagemaker-run=yes` will run on SageMaker. The argument `--output` is automatically set to a mountpoint on SageMaker and outputs are uploaded to S3. Use `--output-s3` to set the S3 output path, or leave it as `default` to automatically generate an appropriate path based on the job name.

The example code uploads `aws_sagemaker_remote` from the local filesystem using the `dependencies` argument. Alternatively:

- Add `aws_sagemaker_remote` to your Docker image.
- Create a `requirements.txt` file including `aws_sagemaker_remote`. Pass the path of the requirements file to the `requirements` function argument or the `--sagemaker-requirements` command-line argument.
- Create a bash script including `pip install aws-sagemaker-remote`. Pass the path of the script to the `configuration_script` function argument or the `--sagemaker-configuration-script` command-line argument.
- Pass `pip install aws-sagemaker-remote` to the `configuration_command` function argument or the `--sagemaker-configuration-command` command-line argument.

See mnist_processor.py.

```
import argparse
import os
import pprint
from torch import nn
from torch.utils import data
from torchvision.datasets import MNIST
import torchvision.transforms as transforms
from aws_sagemaker_remote.processing import sagemaker_processing_main
import aws_sagemaker_remote

def main(args):
    # Main function runs locally or remotely
    dataroot = args.output
    MNIST(
        root=dataroot, download=True, train=True,
        transform=transforms.ToTensor()
    )
    MNIST(
        root=dataroot, download=True, train=False,
        transform=transforms.ToTensor()
    )
    print("Downloaded MNIST")


if __name__ == '__main__':
    sagemaker_processing_main(
        script=__file__, # script path for remote execution
        main=main, # main function for local execution
        outputs={
            # Add the command line flag `output`
            # flag: default path
            'output': 'output/data'
        },
        dependencies={
            # Add a module to SageMaker
            # module name: module path
            'aws_sagemaker_remote': aws_sagemaker_remote
```

(continues on next page)

(continued from previous page)

```
        },
        configuration_command='pip3 install --upgrade sagemaker sagemaker-experiments
↪',
        # Name the job
        base_job_name='demo-mnist-processor'
    )
```

CHAPTER 3

Training

Processing jobs accept a set of one or more input file paths and write to a set of one or more output file paths. Ideal for file conversion or other data preparation tasks.

- Running locally, standard command line arguments for inputs and outputs are used as usual
- Running remotely, data is uploaded and downloaded using S3 for tracking

3.1 Basic usage

Write a script with a `main` function that calls `sagemaker_training_main`.

```
from aws_sagemaker_remote import sagemaker_training_main

def main(args):
    # your code here
    pass

if __name__ == '__main__':
    sagemaker_training_main(
        main=main,
        # ...
    )
```

Pass function argument `run=True` or command line argument `--sagemaker-run=True` to run script remotely on SageMaker.

- Many command-line arguments are automatically added. See *Command-Line Arguments*.
- Parameters to `sagemaker_processing_main` control what command-line arguments are automatically added and the default values. See *aws_sagemaker_remote.training.main.sagemaker_training_main()* and *aws_sagemaker_remote.training.args.sagemaker_training_args()*

3.2 Path Handling

3.2.1 Inputs

Configure inputs by passing an `inputs` dictionary argument to `sagemaker_training_main`. See `aws_sagemaker_remote.args.sagemaker_training_args()`

For example, if your dictionary contains the key `my_dataset`:

- The command line argument `--my-dataset` accepts local paths or S3 URLs
- Local paths are uploaded to S3
- Data downloaded from S3 to container
- Location of data on container pulled from environment
- Your main function is called with `args.my_dataset` set to EFS mount on container

3.2.2 Outputs

There are three output paths:

- `args.model_dir` and `--model-dir`: Directory to export trained inference model. Used when deploying model for inference. Save everything you need for inference but don't save optimizers to minimize inference deployment.
- `args.output_dir` and `--output-dir`: Directory for outputs (logs, images, etc.)
- `args.checkpoint_dir` and `--checkpoint-dir`: Directory for training checkpoints. Save model, optimizer, step count, anything else you need. This will be backed up and restored if training is interrupted.

Running locally, arguments are passed through. If running on SageMaker, arguments are automatically set to mount-points which are uploaded to S3.

3.3 Training Job Tracking

Use the SageMaker console to view a list of all training jobs. For each job, SageMaker tracks:

- Training time
- Container used
- Link to CloudWatch logs
- Path on S3 for each of:
  - Source code ZIP
  - Each input channel
  - Model output ZIP
  - Dependencies (if used)

3.4 Configuration

Many command line options are added by this command.

Option `--sagemaker-run` controls local or remote execution.

- Set `--sagemaker-run` to a falsy value (no,false,0), the script will call your main function as usual and run locally.
- Set `--sagemaker-run` to a truthy value (yes,true,1), the script will upload itself and any requirements or inputs to S3, execute remotely on SageMaker, and save outputs to S3, logging results to the terminal.

Set `--sagemaker-wait` truthy to tail logs and wait for completion or falsy to complete when the job starts.

Defaults are set through code. Defaults can be overwritten on the command line. For example:

- Use the function argument `image` to set the default container image for your script
- Use the command line argument `--sagemaker-image` to override the container image on a particular run

See `aws_sagemaker_remote.args.sagemaker_training_args()` and *Command-Line Arguments* for details.

3.5 Environment Customization

The environment can be customized in multiple ways.

- Instance
  - Function argument `training_instance`
  - Command line argument `--sagemaker-training-instance`
  - Select instance type of machine running the container
- Image
  - Function argument `training_image`
  - Command line argument `--sagemaker-training-image`
  - Accepts URI of Docker container image on ECR or DockerHub to run
  - Build a custom Docker image for major customizations
- Requirements file
  - Create a file named `requirements.txt` in your `source` directory
  - `source` directory defaults to the directory containing your script but can be overridden
  - Use for installing Python packages by listing one on each line. Standard `requirements.txt` file format [https://pip.pypa.io/en/stable/reference/pip_install/#requirements-file-format]
- Dependencies
  - Function argument `dependencies`
    - Dictionary of `[key]->[value]`
    - Each key will create command line argument `--key` that defaults to `value`
  - Each `value` is a directory containing a Python module that will be uploaded to S3, downloaded to SageMaker, and put on the PYTHONPATH

  - For example, if directory `mymodule` contains the files `__init__.py` and `myfile.py` and `myfile.py` contains `def myfunction():...`, pass `dependencies={'mymodule':'path/to/mymodule'}` to `sagemaker_processing_main` and then use `from mymodule.myfile import myfunction` in your script.
  - Use module uploads for supporting code that is not being installed from packages.

3.6 Spot Training

Save on training costs by using spot training. Rather than starting immediately, AWS runs training when excess processing is available in exchange for cost savings.

- `--sagemaker-spot-instances=yes` Use spot instances
- `--sagemaker-max-run` Maximum training runtime in seconds
- `--sagemaker-max-wait` Maximum time to wait in seconds, must be greater than the runtime.

3.7 Additional arguments

Any arguments passed to your script locally on the command line are passed to your script remotely and tracked by SageMaker. Internally, `sagemaker_processing_main` uses `argparse`. To add additional command-line flags:

- Pass a list of kwargs dictionaries to `additional_arguments`

```
sagemaker_training_main(
  #...
  additional_arguments = [
    {
      'dest': '--filter-width',
      'default':32,
      'help':'Filter width'
    },
    {
      'dest':'--filter-height',
      'default':32,
      'help':'Filter height'
    }
  ]
)
```

- Pass a callback to `argparse_callback`

```
from argparse import ArgumentParser
def argparse_callback(parser:ArgumentParser):
  parser.add_argument(
  '--filter-width',
  default=32,
  help='Filter width')
  parser.add_argument(
  '--filter-height',
  default=32,
  help='Filter height')
sagemaker_training_main(
```

(continues on next page)

(continued from previous page)

```
    # ...
    argparse_callback=argparse_callback
)
```

3.8 Command-Line Arguments

These command-line arguments were created using the following parameters. Command-line arguments are generated for each item in `inputs` and `dependencies`.

```
inputs={
    'input': 'path/to/input'
},
dependencies={
    'my_module': 'path/to/my_module'
}
```

```
usage: aws-sagemaker-remote-training [-h]
                                     [--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]
                                     [--sagemaker-output-json SAGEMAKER_OUTPUT_JSON]
                                     [--my-module MY_MODULE]
                                     [--model-dir MODEL_DIR]
                                     [--output-dir OUTPUT_DIR]
                                     [--checkpoint-dir CHECKPOINT_DIR]
                                     [--sagemaker-checkpoint-s3 SAGEMAKER_CHECKPOINT_
↪S3]
                                     [--sagemaker-checkpoint-container SAGEMAKER_
↪CHECKPOINT_CONTAINER]
                                     [--checkpoint-initial CHECKPOINT_INITIAL]
                                     [--input INPUT] [--input-mode INPUT_MODE]
```

(continues on next page)

(continued from previous page)

```
                                    [--input-repeat INPUT_REPEAT]
                                    [--input-shuffle [INPUT_SHUFFLE]]
```

3.8.1 Named Arguments

--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"

--input-shuffle Shuffle inputs

Default: False

3.8.2 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/aws-sagemaker-remote/checkouts/stable/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.

3.8.3 Dependencies

Dependencies to upload to SageMaker

--my-module Directory for dependency [my_module] (default: "path/to/my_module")

Default: "path/to/my_module"

3.8.4 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"

--checkpoint-initial Initial checkpoint

3.8.5 Inputs

Inputs (local or S3)

--input Input channel [input]. Set to local path and it will be uploaded to S3 and downloaded to SageMaker. Set to S3 path and it will be downloaded to SageMaker. (default: [path/to/input])

Default: "path/to/input"

--input-mode Input channel [input] mode. (default: [File])

Default: "File"

--input-repeat Repeat input

Default: 1

3.9 Example Code

The following example creates a trainer with one input named `input`.

- Running the file without arguments will run locally. The argument `--input` sets the input directory.
- Running the file with `--sagemaker-run=yes` will run on SageMaker. The argument `--input` is uploaded to S3, downloaded to SageMaker, and automatically set to a mountpoint.

The example code uploads `aws_sagemaker_remote` from the local filesystem using the `dependencies` argument. Alternatively:

- Add `aws_sagemaker_remote` to your Docker image.
- Create a `requirements.txt` file including `aws_sagemaker_remote`. Place the file in your `source` directory (default to the directory containing the file containing the main function)

See mnist_training.py.

```
import argparse
from aws_sagemaker_remote.training.main import sagemaker_training_main
import torch
from torch import nn
from torch.utils import data
from torchvision import datasets
import torchvision.transforms as transforms
import aws_sagemaker_remote
import os


class Model(nn.Module):
    def __init__(self):
        super(Model, self).__init__()
        self.model = nn.Sequential(
            nn.Conv2d(in_channels=1, out_channels=32, kernel_size=3, stride=2),
            nn.LeakyReLU(),
            nn.Conv2d(in_channels=32, out_channels=64,
                      kernel_size=3, stride=2),
            nn.LeakyReLU(),
            nn.Conv2d(in_channels=64, out_channels=128,
                      kernel_size=3, stride=1),
```

(continues on next page)

(continued from previous page)

```
            nn.LeakyReLU(),
            nn.Conv2d(in_channels=128, out_channels=10,
                      kernel_size=3, stride=1),
        )

    def forward(self, input):
        return torch.mean(self.model(input), dim=(2, 3))


def main(args):
    print("Training")
    batch_size = 32
    device = 'cuda' if torch.cuda.is_available() else 'cpu'
    dataset = data.DataLoader(
        datasets.MNIST(
            root=args.input, download=True, train=True,
            transform=transforms.ToTensor()
        ),
        batch_size=batch_size,
        shuffle=True, num_workers=2, drop_last=False)
    # Create model, optimizer, and criteria
    model = Model().to(device)
    optimizer = torch.optim.Adam(
        params=model.parameters(), lr=args.learning_rate)
    criteria = nn.CrossEntropyLoss()
    model.train()

    for i in range(args.epochs):
        for j, (pixels, labels) in enumerate(dataset):
            pixels, labels = pixels.to(device), labels.to(device)
            logits = model(pixels)
            loss = criteria(input=logits, target=labels)
            accuracy = torch.mean(
                torch.eq(torch.argmax(logits, dim=-1), labels).float())
            loss.backward()
            optimizer.step()
            if j % 100 == 0:
                print("epoch {}, step {}, loss {}, accuracy {}".format(
                    i, j,
                    loss.item(), accuracy.item()
                ))
    os.makedirs(args.model_dir, exist_ok=True)
    torch.save(
        model, os.path.join(args.model_dir, 'model.pt')
    )


def argparse_callback(parser):
    parser.add_argument(
        '--learning-rate',
        default=1e-3,
        type=float,
        help='Learning rate')
    parser.add_argument(
        '--epochs',
        default=5,
        type=int,
```

(continues on next page)

(continued from previous page)

```
        help='Epochs to train')


if __name__ == '__main__':
    sagemaker_training_main(
        script=__file__,
        main=main,
        inputs={
            'input': 'output/data'
        },
        dependencies={
            'aws_sagemaker_remote': aws_sagemaker_remote
        },
        argparse_callback=argparse_callback
    )
```

CHAPTER 4

aws_sagemaker_remote

4.1 aws_sagemaker_remote package

4.1.1 Subpackages

aws_sagemaker_remote.processing package

Submodules

aws_sagemaker_remote.processing.args module

aws_sagemaker_remote.processing.args.**is_sagemaker**()

aws_sagemaker_remote.processing.args.**sagemaker_processing_args**(*parser: argparse.ArgumentParser*, *script*, *run=False*, *wait=True*, *profile='default'*, *role='aws-sagemaker-remote-processing-role'*, *image='aws-sagemaker-remote-processing:latest'*, *image_path='/home/docs/checkouts/rea*
*sagemaker-remote/checkouts/stable/aws_sagemaker_*
image_accounts='763104351884', *instance='ml.t3.medium'*, *inputs=None*, *outputs=None*, *dependencies=None*, *input_mount='/opt/ml/processing/input'*, *output_mount='/opt/ml/processing/output'*, *module_mount='/opt/ml/processing/modules*
base_job_name='processing-job', *job_name=''*, *runtime_seconds=3600*, *volume_size=30*, *python='python3'*, *requirements=None*, *configuration_script=None*, *configuration_command=None*, *additional_arguments=None*, *argparse_callback=None*, *output_json=None*, *env=None*)

Configure `argparse.ArgumentParser` for processing scripts.

Parameters

- **parser** (*argparse.ArgumentParser*) – Parser to configure
- **script** (*str*) – Path to script file to execute. Set default for `--sagemaker-script`
- **run** (*bool, optional*) – Run on SageMaker. Set default for `--sagemaker-run`.
- **wait** (*bool, optional*) – Wait for SageMaker processing to complete. Set default for `--sagemaker-wait`.
- **profile** (*str, optional*) – AWS profile to use for session. Set default for `--sagemaker-profile`.
- **role** (*str, optional*) – AWS IAM role name to use for processing. Will be created if it does not exist. Set default for `--sagemaker-role`.
- **image** (*str, optional*) – URI of ECR Docker image to use for processing. Set default for `--sagemaker-image`.
- **image_path** (*str, optional*) – Path to build docker if image does not exist. Set default for `--sagemaker-image-path`.
- **image_accounts** (*str, optional*) – Accounts required to build docker image. Set default for `--sagemaker-image-accounts`.
- **instance** (*str, optional*) – Type of instance to use for processing (e.g., `ml.t3.medium`). Set default for `--sagemaker-instance`.
- **inputs** (*dict(str,str), optional*) – Dictionary of input argument keys to strings or *aws_sagemaker_remote.args.PathArgument*. Strings are converted to `PathArgument` with `local` set to your string. This should be sufficient for most use cases. For eack key and value, create an argument `--key` that defaults to value.
  - Running locally, input arguments are unmodified.
  - Running remotely, inputs are set to appropriate SageMaker mount points. Local inputs are uploaded automatically.

  For example:

```
import OPTIONAL, PathArgument from aws_sagemaker_remote.args
inputs = {
    "my_input_1": "path/to/data1", # implicit
    "my_input_2": PathArgument(local="path/to/data2"), # explicit
    "my_optional_input": OPTIONAL
}
```

  Your script will now have arguments `--my-input-1`, `--my-input-2`, and `--my-optional-input`.

- **outputs** (*dict(str, str)*) – Dictionary of output arguments keys to strings or *aws_sagemaker_remote.args.PathArgument*. Strings are converted to `PathArgument` with `local` set to your string. This should be sufficient for most use cases. For eack key and value, create an argument `--key` that defaults to value.

  For eack key:

  - Create an argument `--key` that defaults to `value.local`. This controls an output path.
  - Create an argument `--key-s3` that defaults to `value.remote`. This controls where output is stored on S3. * Set to `default` to automatically create an output path based on the job name * Set to an S3 URL to store output at a specific location on S3

- **dependencies** (*dict(str, str)*) – Dictionary of modules. For eack key and value, create an argument `--module-key` that defaults to value. This controls the path of a dependency of your code. The files at the given path will be uploaded to S3, downloaded to SageMaker, and put on PYTHONPATH.
- **input_mount** (*str, optional*) – Local path on SageMaker container where inputs are downloaded. Set default for `--sagemaker-input-mount`.
- **output_mount** (*str, optional*) – Local path on SageMaker container where outputs are written before upload. Set default for `--sagemaker-output-mount`.
- **module_mount** (*str, optional*) – Local path on SageMaker container where source code is downloaded. Mount point is put on PYTHONPATH. Set default for `--sagemaker-module-mount`.
- **base_job_name** (*str, optional*) – Job name will be generated from `base_job_name` and a timestamp if `job_name` is not provided. Set default for `--sagemaker-base-job-name`.
- **job_name** (*str, optional*) – Job name is used for tracking and organization. Generated from `base_job_name` if not provided. Use `base_job_name` and leave `job_name` blank for most use-cases. Set default for `--sagemaker-job-name`.
- **runtime_seconds** (*int, optional*) – Maximum in seconds before killing job. Set default for `--sagemaker-runtime-seconds`.
- **volume_size** (*int, optional*) – Volume size in GB. Set default for `--sagemaker-volume-size`.
- **python** (*str, optional*) – Pyton executable on container (default: `python3`). Set default for `--sagemaker-python`.
- **requirements** (*str, optional*) – Set path to requirements file to upload and install with `pip install -r`. Set default for `--sagemaker-requirements`.
- **configuration_script** (*str, optional*) – Set path to bash script to upload and source. Set default for `--sagemaker-configuration-script`.
- **configuration_command** (*str, optional*) – Set command to be run to configure container, e.g. `pip install mypackage && export MYVAR=MYVALUE`. Set default for `--sagemaker-configuration-command`.
- **additional_arguments** (*list, optional*) – List of tuple of positional args and keyword args for `argparse.ArgumentParser.add_argument`. Use to add additional arguments to the script.
- **argparse_callback** (*function, optional*) – Function accepting one argument `parser:argparse.ArgumentParser` that adds additional arguments. Use to add additional arguments to the script.
- **output_json** (*str, optional*) – Write SageMaker training details to this path. Set default for `--sagemaker-output-json`

aws_sagemaker_remote.processing.args.**sagemaker_processing_input_args**(*parser: argparse.ArgumentParser*, *inputs=None*, *input_mount='/opt/ml/processing/*

aws_sagemaker_remote.processing.args.**sagemaker_processing_module_args**(*parser: argparse.ArgumentParser*, *dependencies=None*, *module_mount='/opt/ml/processin*

aws_sagemaker_remote.processing.args.**sagemaker_processing_output_args**(*parser: argparse.ArgumentParser*, *outputs=None*, *output_mount='/opt/ml/processin*

aws_sagemaker_remote.processing.args.**sagemaker_processing_parser_for_docs**()

aws_sagemaker_remote.processing.config module

class aws_sagemaker_remote.processing.config.**SageMakerProcessingConfig**(*dependencies=None*, *inputs=None*, *outputs=None*, *env=None*)

Bases: object

aws_sagemaker_remote.processing.iam module

aws_sagemaker_remote.processing.iam.**ensure_processing_role**(*iam*, *role_name*)

aws_sagemaker_remote.processing.main module

class aws_sagemaker_remote.processing.main.**ProcessingCommand**(*script*, *main*, *help=None*, ***processing_args*)

Bases: aws_sagemaker_remote.commands.Command

configure(*parser: argparse.ArgumentParser*)

run(*args*)

aws_sagemaker_remote.processing.main.**sagemaker_processing_handle**(*args*, *config*, *main*)

aws_sagemaker_remote.processing.main.**sagemaker_processing_local_args**(*args*, *config: aws_sagemaker_remote.process*

aws_sagemaker_remote.processing.main.**sagemaker_processing_main**(*main*, *script=None*, *description=None*, ***processing_args*)

Entry point for processing.

Example

```
from aws_sagemaker_remote import sagemaker_training_main

def main(args):
    # your code here
    pass

if __name__ == '__main__':
    sagemaker_training_main(
        main=main,
        # ... additional configuration
    )
```

Parameters

- **main** (*function*) – Main function. Must accept a single argument `args:argparse.Namespace`.
- **script** (*str, optional*) – Path to script file to execute. Set to `__file__` for most use-cases. Empty or None defaults to file containing `main`.
- **description** (*str, optional*) – Script description for argparse
- ****processing_args** (*dict, optional*) – Keyword arguments to *aws_sagemaker_remote.processing.args.sagemaker_processing_args()*

aws_sagemaker_remote.processing.process module

aws_sagemaker_remote.processing.process.**ensure_eol**(*file*)

Ensure that file has Linux line endings. Convert if it doesn't.

aws_sagemaker_remote.processing.process.**make_arguments**(*args*, *config: aws_sagemaker_remote.processing.config.SageMake*

aws_sagemaker_remote.processing.process.**make_processing_input**(*mount*, *name*, *source*, *s3*, *mode=None*)

aws_sagemaker_remote.processing.process.**process**(*session: sagemaker.session.Session*, *role*, *script*, *inputs=None*, *outputs=None*, *dependencies=None*, *requirements=None*, *configuration_script=None*, *configuration_command=None*, *base_job_name='processing-job'*, *job_name=None*, *image='aws-sagemaker-remote-processing:latest'*, *image_path='/home/docs/checkouts/readthedocs.org/user_build*
sagemaker-remote/checkouts/stable/aws_sagemaker_remote/ecr/processing
image_accounts='763104351884', *instance='ml.t3.medium'*, *volume_size=30*, *runtime_seconds=3600*, *output_mount='/opt/ml/processing/output'*, *input_mount='/opt/ml/processing/input'*, *module_mount='/opt/ml/processing/modules'*, *python='python3'*, *wait=True*, *logs=True*, *arguments=None*, *tags=None*, *output_json=None*, *env=None*)

aws_sagemaker_remote.processing.process.**sagemaker_arguments**(*vargs*)

aws_sagemaker_remote.processing.process.**sagemaker_processing_run**(*args*, *config*)

Module contents

aws_sagemaker_remote.training package

Submodules

aws_sagemaker_remote.training.args module

aws_sagemaker_remote.training.args.**CHECKPOINT_LOCAL_PATH = '/opt/ml/checkpoints'**
args = {} output_dir = os.environ.get('SM_OUTPUT_DIR', None) if output_dir:

args['output_dir'] = output_dir

model_dir = os.environ.get('SM_MODEL_DIR', None) if model_dir:

args['model_dir'] = model_dir

for channel in config.inputs.keys(): env_key = 'SM_CHANNEL_{}'.format(channel.upper()) channel_dir = os.environ.get(env_key, None) if channel_dir:

args[channel] = channel_dir

return args

Type def sagemaker_env_args(config)

aws_sagemaker_remote.training.args.**is_sagemaker**()

aws_sagemaker_remote.training.args.**sagemaker_env_arg**()
Check for SM_TRAINING_ENV environment variable and return object if it exists

aws_sagemaker_remote.training.args.**sagemaker_env_args**(*args: argparse.Namespace*, *config: aws_sagemaker_remote.training.config.SageMakerTra*
Check for SM_TRAINING_ENV environment variable and use it to override arguments.

aws_sagemaker_remote.training.args.**sagemaker_training_args**(*parser: argparse.ArgumentParser, script, source='', base_job_name='training-job', job_name='', profile='default', run=False, wait=True, inputs=None, dependencies=None, additional_arguments=None, argparse_callback=None, model_dir='output/model', output_dir='output/output', checkpoint_dir='output/checkpoint', checkpoint_s3='default', checkpoint_container='/opt/ml/checkpoints', checkpoint_initial=None, training_image='aws-sagemaker-remote-training:latest', training_image_path='/home/docs/checkouts/r sagemaker-remote/checkouts/stable/aws_sagemaker_remo training_image_accounts=['763104351884'], training_instance='ml.m5.large', training_role='aws-sagemaker-remote-training-role', enable_sagemaker=True, experiment_name=None, trial_name=None, spot_instances=False, volume_size=30, max_run=43200, max_wait=86400, env=None, workers=2, output_json=None*)

Configure `argparse.ArgumentParser` for training scripts.

Parameters

- **parser** (*argparse.ArgumentParser*) – Parser to configure
- **script** (*str*) – Path to script file to execute. Set default for `--sagemaker-script`

- **source** (*str, optional*) – Path of source directory to upload. Must include `script` path. Defaults to directory containing `script` if not provided.
- **base_job_name** (*str, optional*) – Job name will be generated from `base_job_name` and a timestamp if `job_name` is not provided. Set default for `--sagemaker-base-job-name`.
- **job_name** (*str, optional*) – Job name is used for tracking and organization. Generated from `base_job_name` if not provided. Use `base_job_name` and leave `job_name` blank for most use-cases. Set default for `--sagemaker-job-name`.
- **profile** (*str, optional*) – AWS profile to use for session. Set default for `--sagemaker-profile`.
- **run** (*bool, optional*) – Run on SageMaker. Set default for `--sagemaker-run`.
- **wait** (*bool, optional*) – Wait for SageMaker processing to complete. Set default for `--sagemaker-wait`.
- **inputs** (*dict(str,str), optional*) – Dictionary of input arguments. For eack key and value, create an argument `--key` that defaults to value. * Running locally, input arguments are unmodified. * Running remotely, inputs are set to appropriate SageMaker mount points. Local inputs are uploaded automatically.
- **dependencies** (*dict(str, str)*) – Dictionary of modules. For eack key and value, create an argument `--module-key` that defaults to value. This controls the path of a dependency of your code. The files at the given path will be uploaded to S3, downloaded to SageMaker, and put on PYTHONPATH.
- **additional_arguments** (*list, optional*) – List of tuple of positional args and keyword args for `argparse.ArgumentParser.add_argument`. Use to add additional arguments to the script.
- **argparse_callback** (*function, optional*) – Function accepting one argument `parser:argparse.ArgumentParser` that adds additional arguments. Use to add additional arguments to the script.
- **model_dir** (*string, optional*) – Directory to save trained inference model. Set default for `--model-dir`.
- **output_dir** (*string, optional*) – Directory to save outputs (images, logs, etc.). Set default for `--output-dir`.
- **checkpoint_dir** (*string, optional*) – Directory to save checkpoints for saving and resuming training. Set default for `--checkpoint-dir`.
- **checkpoint_s3** (*string, optional*) – S3 storage for checkpoints for saving and resuming training or "default". Set default for `--sagemaker-checkpoint-s3`.
- **checkpoint_container** (*string, optional*) – Local directory for checkpoints when running remotely. Set default for `--sagemaker-checkpoint-container`.
- **training_image** (*str, optional*) – URI of ECR or DockerHub Docker image to use for training. Set default for `--sagemaker-training-image`.
- **training_instance** (*str, optional*) – Type of instance to use for training (e.g., `ml.t3.medium`). Set default for `--sagemaker-training-instance`.
- **training_role** (*str, optional*) – AWS IAM role name to use for training. Will be created if it does not exist. Set default for `--sagemaker-training-role`.
- **experiment_name** (*str, optional*) – Name of experiment. Required if `trial_name` is provided. Set default for `--sagemaker-experiment-name`.

- **trial_name** (*str, optional*) – Name of trial within experiment. Set default for `--sagemaker-trial-name`.
- **enable_sagemaker** (*bool, optional*) –
  - True: Include SageMaker command-line options.
  - False: Only include local command-line options
- **max_run** (*int, optional*) – Maximum training time in seconds.
- **max_wait** (*int, optional*) – Maximum time to wait for a spot instance in seconds.
- **workers** (*int, optional*) – Number of workers

aws_sagemaker_remote.training.args.**sagemaker_training_channel_args**(*parser: argparse.ArgumentParser, inputs*)

aws_sagemaker_remote.training.args.**sagemaker_training_checkpoint_args**(*parser: argparse.ArgumentParser, checkpoint_dir, checkpoint_initial=None, checkpoint_s3='default', checkpoint_container='/opt/ml/chec enable_sagemaker=True*)

aws_sagemaker_remote.training.args.**sagemaker_training_dependency_args**(*parser: argparse.ArgumentParser, dependencies*)

aws_sagemaker_remote.training.args.**sagemaker_training_model_args**(*parser: argparse.ArgumentParser, model_dir='model'*)

aws_sagemaker_remote.training.args.**sagemaker_training_output_args**(*parser: argparse.ArgumentParser, output_dir*)

aws_sagemaker_remote.training.args.**sagemaker_training_parser_for_docs**()

aws_sagemaker_remote.training.channels module

aws_sagemaker_remote.training.channels.**expand_folder_channels**(*channels, session*)

aws_sagemaker_remote.training.channels.**expand_list_channels**(*channels*)

aws_sagemaker_remote.training.channels.**expand_repeated_channels**(*channels*)

aws_sagemaker_remote.training.channels.**parse_channel_arguments**(*channels*, *session*)

aws_sagemaker_remote.training.channels.**process_channels**(*channels*, *args*, *session*, *prefix*)

aws_sagemaker_remote.training.channels.**read_channel_arguments**(*channels*, *args*)

aws_sagemaker_remote.training.channels.**remove_empty_channels**(*channels*)

aws_sagemaker_remote.training.channels.**set_suffixes**(*channels*, *session*, *hyperparameters*)

aws_sagemaker_remote.training.channels.**standardize_channel**(*channel*)

aws_sagemaker_remote.training.channels.**standardize_channels**(*channels*)

aws_sagemaker_remote.training.channels.**upload_local_channel**(*channel*, *session*, *s3_uri*)

aws_sagemaker_remote.training.channels.**upload_local_channels**(*channels*, *session*, *prefix*)

aws_sagemaker_remote.training.config module

class aws_sagemaker_remote.training.config.**SageMakerTrainingConfig**(*inputs=None*, *dependencies=None*, *env=None*)

Bases: object

aws_sagemaker_remote.training.experiment module

aws_sagemaker_remote.training.experiment.**ensure_experiment**(*client*, *experiment_name*)

aws_sagemaker_remote.training.iam module

aws_sagemaker_remote.training.iam.**ensure_training_role**(*iam*, *role_name*)

aws_sagemaker_remote.training.main module

class aws_sagemaker_remote.training.main.**TrainingCommand**(*main*, *script=None*, *help=None*, *metrics=None*, ***training_args*)

Bases: aws_sagemaker_remote.commands.Command

configure(*parser: argparse.ArgumentParser*)

run(*args*)

aws_sagemaker_remote.training.main.**sagemaker_training_handle**(*args*, *config*, *main*, *metrics=None*)

aws_sagemaker_remote.training.main.**sagemaker_training_main**(*main*, *script=None*, *script_fn=None*, *description=None*, *metrics=None*, ***training_args*)

Entry point for training.

Example

```
from aws_sagemaker_remote import sagemaker_processing_main

def main(args):
    # your code here
    pass

if __name__ == '__main__':
    sagemaker_processing_main(
        main=main,
        # ... additional configuration
    )
```

Parameters

- **main** (*function*) – Main function. Must accept a single argument `args` (`argparse.Namespace`)
- **script** (*str, optional*) – Path to script file to execute. Set to `__file__` for most use-cases. Empty or None defaults to file containing `main`. Object interpreted as file containing the object.
- **description** (*str, optional*) – Script description for argparse
- **metrics** (*dict, optional*) – Metrics to record. Dictionary of metric name (str) to RegEx that extracts metric (str). See SageMaker Training Metrics Docs
- ****training_args** (*dict, optional*) – Keyword arguments to *aws_sagemaker_remote.training.args.sagemaker_training_args()*

aws_sagemaker_remote.training.train module

aws_sagemaker_remote.training.train.**sagemaker_training_run**(*args*, *config: aws_sagemaker_remote.training.config.SageM*, *metrics=None*)

Module contents

4.1.2 Submodules

4.1.3 aws_sagemaker_remote.args module

Modes

- File
- Pipe

- ManifestFile
- AugmentedManifestFile

class aws_sagemaker_remote.args.**PathArgument**(*local=None*, *remote='default'*, *optional=False*, *mode=None*, *attributes=None*, *repeat=1*, *shuffle=False*, *repeated=False*)

Bases: object

copy(***kwargs*)

aws_sagemaker_remote.args.**argparse_to_variable**(*flag*)

aws_sagemaker_remote.args.**bool_argument**(*parser: argparse.ArgumentParser*, **args*, ***kwargs*)

aws_sagemaker_remote.args.**convert_path_argument**(*param*, *cls=<class 'aws_sagemaker_remote.args.PathArgument'>*)

aws_sagemaker_remote.args.**convert_path_arguments**(*params*, *cls=<class 'aws_sagemaker_remote.args.PathArgument'>*)

aws_sagemaker_remote.args.**get_local_path**(*path*)

aws_sagemaker_remote.args.**get_mode**(*mode*)

aws_sagemaker_remote.args.**get_record_wrapping**(*mode*)

aws_sagemaker_remote.args.**get_s3_data_type**(*mode*)

aws_sagemaker_remote.args.**sagemaker_profile_args**(*parser*, *profile='default'*)

aws_sagemaker_remote.args.**strtobool_type**(*v*)

aws_sagemaker_remote.args.**variable_to_argparse**(*variable*)

4.1.4 aws_sagemaker_remote.iam module

aws_sagemaker_remote.iam.**create_role**(*iam*, *role_name*, *description*, *policies*, *trust*)

aws_sagemaker_remote.iam.**ensure_role**(*iam*, *role_name*, *description*, *policies*, *trust*)

aws_sagemaker_remote.iam.**get_role_by_name**(*iam*, *role_name*)

aws_sagemaker_remote.iam.**is_boto_exception**(*e*, *code*)

4.1.5 aws_sagemaker_remote.session module

aws_sagemaker_remote.session.**sagemaker_session**(*profile_name=None*)

4.1.6 Module contents

CHAPTER 5

Indices and tables

- genindex
- modindex
- search

Python Module Index

a

Index

G

I

M

P

R

S

T

U

V