OpenFed Documentation

Release 0.0.0

FederalLab

Sep 25, 2021

GET STARTED

CHAPTER

ONE

OPENFED

NOTE: Current version is unstable, and we will release the first stable version very soon.

1.1 Introduction

OpenFed is a foundational library for federated learning research and supports many research projects as below:

- benchmark-lightly: FederalLab's simulation benchmark.
- openfed-cv: FederalLab's toolkit and benchmark for computer vision in federated learning. This toolkit is based on mmcv, and provides the federated learning for following tasks:
  - MMClassification: OpenMMLab image classification toolbox and benchmark.
  - MMDetection: OpenMMLab detection toolbox and benchmark.
  - MMDetection3D: OpenMMLab's next-generation platform for general 3D object detection.
  - MMSegmentation: OpenMMLab semantic segmentation toolbox and benchmark.
  - MMAction2: OpenMMLab's next-generation action understanding toolbox and benchmark.
  - MMTracking: OpenMMLab video perception toolbox and benchmark.
  - MMPose: OpenMMLab pose estimation toolbox and benchmark.
  - MMEditing: OpenMMLab image and video editing toolbox.
  - MMOCR: OpenMMLab text detection, recognition and understanding toolbox.
  - MMGeneration: OpenMMLab image and video generative models toolbox.
- openfed-finance: FederalLab's toolbox and benchmark for finance data analysis in federated learning.
- openfed-medical: FederalLab's toolbox and benchmark for medical data analysis in federated learning. It is based on MONAI.
- openfed-nlp: FederalLab's toolbox and benchmark for natural language processing in federated learning. It is based on transformers.
- openfed-rl: FederalLab's toolbox and benchmark for reinforcement learning in federated learning. It is based on stable-baselines3

In addition, we also provide a toolkit for better compatibility with following libraries, so that you can use OpenFed with those libraries without obstacles and more easily:

- pytorch-lightning: The lightweight PyTorch wrapper for high-performance AI research. Scale your models, not the boilerplate.

- mmcv: MMCV is a foundational library for computer vision research and supports many research projects.

1.2 Install

PyTorch >= 1.5.1, python>=3.6

Stable version: `pip install openfed`

Latest version: `pip install -e git+https://github.com/FederalLab/OpenFed.git`

1.3 Start Federated Learning In An Unprecedented Simple Way

```python
import argparse
import random

import torch
import torch.nn as nn
from torch.utils.data import DataLoader
from torchvision.datasets import MNIST
from torchvision.transforms import ToTensor

# >>> Import OpenFed
import openfed
# <<<

# >>> Define arguments
parser = argparse.ArgumentParser(description='Simulator')
parser.add_argument('--props', type=str, default='/tmp/aggregator.json')
args = parser.parse_args()
# <<<

# >>> Load Federated Group Properties
props = openfed.federated.FederatedProperties.load(args.props)[0]
# <<<

network = nn.Linear(784, 10)
loss_fn = nn.CrossEntropyLoss()

sgd = torch.optim.SGD(
    network.parameters(), lr=1.0 if props.aggregator else 0.1)

# >>> Convert torch optimizer to federated optimizer
fed_sgd = openfed.optim.FederatedOptimizer(sgd, props.role)
# <<<

# >>> Define maintainer to maintain communication among each nodes
maintainer = openfed.core.Maintainer(props, network.state_dict(keep_vars=True))
# <<<

# >>> Auto register the hook function to maintainer
with maintainer:
```

(continues on next page)

(continued from previous page)

```
    openfed.functional.device_alignment()
    if props.aggregator:
        openfed.functional.count_step(props.address.world_size - 1)
# <<<

# total rounds to simulation
rounds = 10
if maintainer.aggregator:
    # >>> API Loop as aggregator
    api = openfed.API(maintainer, fed_sgd, rounds,
                      openfed.functional.average_aggregation)
    api.run()
    # <<<
else:
    mnist = MNIST(r'/tmp/', True, ToTensor(), download=True)
    # >>> Convert to federated dataset
    fed_mnist = openfed.data.PartitionerDataset(
        mnist, total_parts=100, partitioner=openfed.data.IIDPartitioner())
    # <<<

    dataloader = DataLoader(
        fed_mnist, batch_size=10, shuffle=True, num_workers=0, drop_last=False)

    for outter in range(rounds):
        # >>> Download latest model from aggregator
        maintainer.step(upload=False)
        # <<<

        # Pick up a random federated dataset part
        part_id = random.randint(0, 9)
        fed_mnist.set_part_id(part_id)

        network.train()
        losses = []
        for data in dataloader:
            x, y = data
            output = network(x.view(-1, 784))
            loss = loss_fn(output, y)

            fed_sgd.zero_grad()
            loss.backward()
            fed_sgd.step()
            losses.append(loss.item())
        loss = sum(losses) / len(losses)

        # >>> Finish a round
        fed_sgd.round()
        # <<<

        # >>> Upload trained model and optimizer state
        maintainer.update_version()
        maintainer.package(fed_sgd)
```

(continues on next page)

(continued from previous page)

```
        maintainer.step(download=False)
        # <<<

        # Clear state dict
        fed_sgd.clear_state_dict()
```

Now, save the piece of code as `run.py`, and you can use the provided script to start a simulator by:

```
(openfed) python -m openfed.tools.simulator --nproc 6 run.py
100%|| 10/10 [00:01<00:00,  7.21it/s]
```

This command will launch 6 processes (1 for aggregator, 5 for collaborators).

1.4 Citation

If you find this project useful in your research, please consider cite:

```
@misc{OpenFed,
Author = {Chen Dengsheng},
Title = {OpenFed: An Open-Source Security and Privacy Guaranteed Federated Learning␣
↪Framework},
Year = {2021},
Eprint = {arXiv:2109.07852},
}
```

1.5 Contributing

We appreciate all contributions to improve OpenFed. Please refer to CONTRIBUTUNG.md for the contributing guideline.

1.6 License

OpenFed is released under the MIT License.

CHAPTER

TWO

INSTALLATION

```
pip install openfed
```

CHAPTER

THREE

BUILD OPENFED FROM SOURCE

3.1 Build on Linux or macOS

```
git clone https://github.com/FederalLab/OpenFed.git
cd OpenFed
pip install -e .
```

3.2 Build on Windows

Building OpenFed on Windows is a familiar with that on Linux.

3.3 Test

```
(openfed) ./pytest.sh
General test...
================= test session starts ==================
platform darwin -- Python 3.7.10, pytest-6.2.4, py-1.10.0, pluggy-0.13.1
rootdir: /Users/densechen/code/OpenFed
plugins: xdist-2.4.0, ordering-0.6, forked-1.3.0
collected 32 items / 11 deselected / 21 selected

tests/test_simulator.py .                          [  4%]
tests/test_data/test_partitioner.py .              [  9%]
tests/test_api.py .                                [ 14%]
tests/test_build.py .                              [ 19%]
tests/test_common/test_address.py ....             [ 38%]
tests/test_common/test_meta.py .                   [ 42%]
tests/test_data/test_partitioner.py ...            [ 57%]
tests/test_topo/test_topo.py ....                  [ 76%]
tests/test_utils/test_table.py ..                  [ 85%]
tests/test_utils/test_utils.py ...                 [100%]

========== 21 passed, 11 deselected in 1.43s ===========
Federated...
================= test session starts ==================
platform darwin -- Python 3.7.10, pytest-6.2.4, py-1.10.0, pluggy-0.13.1
```

(continues on next page)

(continued from previous page)

```
rootdir: /Users/densechen/code/OpenFed
plugins: xdist-2.4.0, ordering-0.6, forked-1.3.0
gw0 [3] / gw1 [3] / gw2 [3]
...                                                      [100%]
================== 3 passed in 3.80s ===================
Maintainer...
================= test session starts ==================
platform darwin -- Python 3.7.10, pytest-6.2.4, py-1.10.0, pluggy-0.13.1
rootdir: /Users/densechen/code/OpenFed
plugins: xdist-2.4.0, ordering-0.6, forked-1.3.0
gw0 [3] / gw1 [3] / gw2 [3]
...                                                      [100%]
================== 3 passed in 1.73s ===================
Simulator...
================= test session starts ==================
platform darwin -- Python 3.7.10, pytest-6.2.4, py-1.10.0, pluggy-0.13.1
rootdir: /Users/densechen/code/OpenFed
plugins: xdist-2.4.0, ordering-0.6, forked-1.3.0
gw0 [4] / gw1 [4] / gw2 [4]
....                                                     [100%]
================== 4 passed in 1.84s ===================
Paillier Crypt...
================= test session starts ==================
platform darwin -- Python 3.7.10, pytest-6.2.4, py-1.10.0, pluggy-0.13.1
rootdir: /Users/densechen/code/OpenFed
plugins: xdist-2.4.0, ordering-0.6, forked-1.3.0
gw0 [2] / gw1 [2]
100%|| 2/2 [00:03<00:00,  1.87s/it]
.                                                        [100%]
================== 2 passed in 5.28s ===================
(openfed)  densechen@C02DW0CQMD6R  ~/code/OpenFed   main ±
```

CHAPTER

FOUR

COMMON

4.1 Meta

`Meta` class is a special dictionary that used to convey messages between aggregator and collaborators. It contains two default attributions:

- `mode`: String in [`train`, `others`]. If `mode==train`, the collaborator will train the global model with personal privacy data, and upload the trained model to aggregator. The aggregator will automatically aggregate the received models. Otherwise, the collaborator will not update the global model. It will test the global on personal privacy data and return the results to aggregator.
- `version`: Int. The version number of received global model. In federated learning, we need to use this version tag to control the update behavior of aggregator. Sometimes, the aggregator receives the invalid version of model, which may be out of date. When this case occurs, aggregator will apply some tragedies to deal with it.

`Meta` class can used as a standard dictionary:

```
>>> import openfed
>>> meta = openfed.Meta()
>>> meta
<OpenFed> Meta
+-------+---------+
|  mode | version |
+-------+---------+
| train |    -1   |
+-------+---------+

>>> meta['timestamp'] = openfed.utils.time_string()
>>> meta
<OpenFed> Meta
+-------+---------+---------------------+
|  mode | version |      timestamp      |
+-------+---------+---------------------+
| train |    -1   | 2021-09-21 09:50:41 |
+-------+---------+---------------------+
```

`Meta` class can also be used as a class to access his attributions:

```
>>> import openfed
>>> meta = openfed.Meta()
>>> meta
<OpenFed> Meta
```

(continues on next page)

(continued from previous page)

```
+-------+---------+
|  mode | version |
+-------+---------+
| train |    -1   |
+-------+---------+

>>> meta.timestamp = openfed.utils.time_string()
>>> meta
<OpenFed> Meta
+-------+---------+----------------------+
|  mode | version |      timestamp       |
+-------+---------+----------------------+
| train |    -1   | 2021-09-21 09:52:47 |
+-------+---------+----------------------+
```

4.2 Address

`Address` class stores all the arguments needed to build a process group. It will automatically check the arguments you passed in. There are two kinds of address:

- `tcp_address`: TCP address will keep the communication via a tcp address.
- `file_address`: File address will keep the communication via a shared file.

We also provide an `empty_address`, which contains nothing information, to play as a placeholder.

Define a tcp address:

```
>>> import openfed
>>> tcp_address = openfed.Address('gloo', 'tcp://localhost:1994')
>>> tcp_address
<OpenFed> Address
+---------+---------------------+------------+------+
| backend |     init_method     | world_size | rank |
+---------+---------------------+------------+------+
|   gloo  | tcp://localhost:... |     2      |  -1  |
+---------+---------------------+------------+------+
```

Load the `default_tcp_address`:

```
>>> import openfed
>>> openfed.default_tcp_address
<OpenFed> Address
+---------+---------------------+------------+------+
| backend |     init_method     | world_size | rank |
+---------+---------------------+------------+------+
|   gloo  | tcp://localhost:... |     2      |  -1  |
+---------+---------------------+------------+------+
```

Define a file address:

```
>>> import openfed
>>> file_address = openfed.Address('gloo', 'file:///tmp/openfed.sharedfile')
>>> file_address
<OpenFed> Address
+---------+---------------------+------------+------+
| backend |     init_method     | world_size | rank |
+---------+---------------------+------------+------+
|   gloo  | file:///tmp/open... |     2      |  -1  |
+---------+---------------------+------------+------+
```

Load the `default_file_address`:

```
>>> import openfed
>>> openfed.default_file_address
<OpenFed> Address
+---------+---------------------+------------+------+
| backend |     init_method     | world_size | rank |
+---------+---------------------+------------+------+
|   gloo  | file:///tmp/open... |     2      |  -1  |
+---------+---------------------+------------+------+
```

Load the `empty_address`:

```
>>> import openfed
>>> openfed.empty_address
<OpenFed> Address
+---------+-------------+------------+------+
| backend | init_method | world_size | rank |
+---------+-------------+------------+------+
|   null  |     null    |     2      |  -1  |
+---------+-------------+------------+------+
```

You can refer to the API documentation for more details about each arguments.

CHAPTER

FIVE

CORE

5.1 Maintainer

:class:`Maintainer` bridges the connection between upper(federated algorithms) and lower(communication and topology) layers. It has the following properties:

- `pipe`: The currently target to communicate with. A maintainer will manage several pipes in the same time, and `pipe` will indicate what is the current target.
- `pipes`: A list of pipes to communicate with.
- `current_step`: It is used to indicate which step is running on.
- `fed_props`: Actually, a maintainer is corresponding to a specified federated group. We record the related federated group properties in this attributions.

You can use :class:`Maintainer` to conduct a flexible communication with other nodes more easily than :class:`Pipe`.

5.2 Examples

Aggregator:

```
    # build a topology first
    import openfed
    import openfed.topo as topo

    aggregator = topo.Node('aggregator', openfed.default_tcp_address)
    alpha = topo.Node('alpha', openfed.empty_address)
    beta = topo.Node('beta', openfed.empty_address)

    topology = topo.Topology()
    topology.add_node_list([aggregator, alpha, beta])
    topology.add_edge(alpha, aggregator)
    topology.add_edge(beta, aggregator)

    # analysis topology to get federated group props
    federated_group_props = topo.analysis(topology, aggregator)
    assert len(federated_group_props) == 1
    federated_group_prop = federated_group_props[0]

    # build network
```

(continues on next page)

(continued from previous page)

```
    import torch.nn as nn
    network = nn.Linear(10, 1)

    # build maintainer
    from openfed.core import Maintainer
    maintainer = Maintainer(federated_group_prop,
                            network.state_dict(keep_vars=True))

    with maintainer:
        openfed.functional.device_alignment()
        openfed.functional.count_step(2)

    maintainer.step()
```

Collaborator alpha:

```
    # build a topology first
    import openfed
    import openfed.topo as topo

    aggregator = topo.Node('aggregator', openfed.default_tcp_address)
    alpha = topo.Node('alpha', openfed.empty_address)
    beta = topo.Node('beta', openfed.empty_address)

    topology = topo.Topology()
    topology.add_node_list([aggregator, alpha, beta])
    topology.add_edge(alpha, aggregator)
    topology.add_edge(beta, aggregator)

    # analysis topology to get federated group props
    federated_group_props = topo.analysis(topology, alpha)
    assert len(federated_group_props) == 1
    federated_group_prop = federated_group_props[0]

    # build network
    import torch.nn as nn
    network = nn.Linear(10, 1)

    # build maintainer
    from openfed.core import Maintainer
    maintainer = Maintainer(federated_group_prop,
                            network.state_dict(keep_vars=True))

    with maintainer:
        openfed.functional.device_alignment()

    maintainer.step(upload=False)
    maintainer.package()
    maintainer.step(download=False)
```

Collaborator beta:

```
# build a topology first
import openfed
import openfed.topo as topo

aggregator = topo.Node('aggregator', openfed.default_tcp_address)
alpha = topo.Node('alpha', openfed.empty_address)
beta = topo.Node('beta', openfed.empty_address)

topology = topo.Topology()
topology.add_node_list([aggregator, alpha, beta])
topology.add_edge(alpha, aggregator)
topology.add_edge(beta, aggregator)

# analysis topology to get federated group props
federated_group_props = topo.analysis(topology, beta)
assert len(federated_group_props) == 1
federated_group_prop = federated_group_props[0]

# build network
import torch.nn as nn
network = nn.Linear(10, 1)

# build maintainer
from openfed.core import Maintainer
maintainer = Maintainer(federated_group_prop,
                        network.state_dict(keep_vars=True))

with maintainer:
    openfed.functional.device_alignment()

maintainer.step(upload=False)
maintainer.package()
maintainer.step(download=False)
```

CHAPTER

SIX

DATA

6.1 FederatedDataset

In order to load the simulated federated data in a uniform way, we provide :class:`FederatedDataset`. Compared with :class:`Dataset`, it has two extra attributes:

- `part_id`: Part id to load.
- `total_parts`: The total number of parts.

6.2 PartitionerDataset

:class:`PartitionerDataset` will divide a custom dataset according to the partitioner you selected. It is the most convenient method to generate a simulated federated dataset for testing.

For example, we can use the following piece of code to generate the Federated-MNIST:

```
>>> from openfed.data import IIDPartitioner, PartitionerDataset
>>> from torchvision.datasets import MNIST
>>> from torchvision.transforms import ToTensor
>>> dataset = PartitionerDataset(
    MNIST(r'/tmp/', True, ToTensor(), download=True), total_parts=10,␣
↪partitioner=IIDPartitioner())
Downloading http://yann.lecun.com/exdb/mnist/train-images-idx3-ubyte.gz
Downloading http://yann.lecun.com/exdb/mnist/train-images-idx3-ubyte.gz to /tmp/MNIST/
↪raw/train-images-idx3-ubyte.gz
9913344it [00:19, 502512.54it/s]
Extracting /tmp/MNIST/raw/train-images-idx3-ubyte.gz to /tmp/MNIST/raw

Downloading http://yann.lecun.com/exdb/mnist/train-labels-idx1-ubyte.gz
Downloading http://yann.lecun.com/exdb/mnist/train-labels-idx1-ubyte.gz to /tmp/MNIST/
↪raw/train-labels-idx1-ubyte.gz
29696it [00:00, 853940.49it/s]
Extracting /tmp/MNIST/raw/train-labels-idx1-ubyte.gz to /tmp/MNIST/raw

Downloading http://yann.lecun.com/exdb/mnist/t10k-images-idx3-ubyte.gz
Downloading http://yann.lecun.com/exdb/mnist/t10k-images-idx3-ubyte.gz to /tmp/MNIST/raw/
↪t10k-images-idx3-ubyte.gz
1649664it [00:04, 406894.94it/s]
Extracting /tmp/MNIST/raw/t10k-images-idx3-ubyte.gz to /tmp/MNIST/raw
```

(continues on next page)

(continued from previous page)

```
Downloading http://yann.lecun.com/exdb/mnist/t10k-labels-idx1-ubyte.gz
Downloading http://yann.lecun.com/exdb/mnist/t10k-labels-idx1-ubyte.gz to /tmp/MNIST/raw/
↪t10k-labels-idx1-ubyte.gz
5120it [00:00, 14221746.01it/s]
Extracting /tmp/MNIST/raw/t10k-labels-idx1-ubyte.gz to /tmp/MNIST/raw

Processing...
/Users/densechen/miniconda3/envs/openfed/lib/python3.7/site-packages/torchvision/
↪datasets/mnist.py:502: UserWarning: The given NumPy array is not writeable, and␣
↪PyTorch does not support non-writeable tensors. This means you can write to the␣
↪underlying (supposedly non-writeable) NumPy array using the tensor. You may want to␣
↪copy the array to protect its data or make it writeable before converting it to a␣
↪tensor. This type of warning will be suppressed for the rest of this program.␣
↪(Triggered internally at  /Users/distiller/project/conda/conda-bld/pytorch_
↪1616554799287/work/torch/csrc/utils/tensor_numpy.cpp:143.)
  return torch.from_numpy(parsed.astype(m[2], copy=False)).view(*s)
Done!
>>> from openfed.data.utils import samples_distribution
>>> samples_distribution(dataset, True)
+-------+---------+---------+------+
| Parts | Samples |   Mean  | Var  |
+-------+---------+---------+------+
|   10  |  59960  | 5996.00 | 0.00 |
+-------+---------+---------+------+
[5996, 5996, 5996, 5996, 5996, 5996, 5996, 5996, 5996, 5996]
```

6.3 Partitioner

:class:`Partitioner` can generate a non-iid distribution datasets easily. We provide three different ways, i.e., `PowerLawPartitioner`, `DirichletPartitioner`, `IIDPartitioner`.

CHAPTER

SEVEN

FEDERATED

7.1 Pipe

:class:`Pipe` maintains the communication operation between two nodes, including tensor data and info message. It uses a store to transfer info message and process group with `gloo` or `mpi` to transfer tensor data.

7.2 DistributedProperties

:class:`DistributedProperties` contains all distributed attributions of `torch.distributed.distributed_c10d`. Usually, you can use it with context environment.

```
with dist_props:
    ...
```

7.3 FederatedProperties

:class:`FederatedProperties` contains all federated attributions, such as address, role and nick name. It is usually generated via :func:`openfed.topo.analysis`.

7.4 Examples

Here, we try to communicate some information among `aggregator`, `collaborator_alpha` and `collaborator_beta`. You need to open three independent terminals to run the following three scripts.

Aggregator:

```
# build a topology first
import time

# transfer data
import torch

import openfed
import openfed.topo as topo

aggregator = topo.Node('aggregator', openfed.default_tcp_address)
```

(continues on next page)

(continued from previous page)

```
alpha = topo.Node('alpha', openfed.empty_address)
beta = topo.Node('beta', openfed.empty_address)

topology = topo.Topology()
topology.add_node_list([aggregator, alpha, beta])
topology.add_edge(alpha, aggregator)
topology.add_edge(beta, aggregator)

# analysis topology to get federated group props
federated_group_props = topo.analysis(topology, aggregator)
assert len(federated_group_props) == 1
federated_group_prop = federated_group_props[0]

# build pipe
pipes = openfed.federated.init_federated_group(federated_group_prop)

assert len(pipes) == 2
alpha_pipe, beta_pipe = pipes

# transfer message
alpha_pipe.direct_set('message_0', 'hello world from aggregator to alpha')
beta_pipe.direct_set('message_0', 'hello world from aggregator to beta')

print(alpha_pipe.direct_get('message_1'))
print(beta_pipe.direct_get('message_1'))

data = torch.tensor(-1)
with alpha_pipe.dist_props:
    time.sleep(0.5)
    # send data to alpha
    alpha_pipe.upload(data)

    time.sleep(0.5)
    # download data from alpha
    assert alpha_pipe.download() == 1

with beta_pipe.dist_props:
    time.sleep(0.5)
    # send data to beta
    beta_pipe.upload(data)

    time.sleep(0.5)
    # download data from beta
    assert beta_pipe.download() == 2

time.sleep(1)
```

Collaborator alpha:

```
# build a topology first
import time

```

(continues on next page)

(continued from previous page)

```
# transfer tensor
import torch

import openfed
import openfed.topo as topo

aggregator = topo.Node('aggregator', openfed.default_tcp_address)
alpha = topo.Node('alpha', openfed.empty_address)
beta = topo.Node('beta', openfed.empty_address)

topology = topo.Topology()
topology.add_node_list([aggregator, alpha, beta])
topology.add_edge(alpha, aggregator)
topology.add_edge(beta, aggregator)

# analysis topology to get federated group props
federated_group_props = topo.analysis(topology, alpha)
assert len(federated_group_props) == 1
federated_group_prop = federated_group_props[0]

# build pipe
pipes = openfed.federated.init_federated_group(federated_group_prop)

alpha_pipe = pipes[0]

# transfer message
print(alpha_pipe.direct_get('message_0'))

alpha_pipe.direct_set('message_1', 'hello world from alpha to aggregator')

data = torch.tensor(1)
with alpha_pipe.dist_props:
    # download data from aggregator
    assert alpha_pipe.download() == -1

    # upload data to aggregator
    alpha_pipe.upload(data)

time.sleep(1)
```

Collaborator beta:

```
# build a topology first
import time

# transfer data
import torch

import openfed
import openfed.topo as topo

aggregator = topo.Node('aggregator', openfed.default_tcp_address)
```

(continues on next page)

(continued from previous page)

```
alpha = topo.Node('alpha', openfed.empty_address)
beta = topo.Node('beta', openfed.empty_address)

topology = topo.Topology()
topology.add_node_list([aggregator, alpha, beta])
topology.add_edge(alpha, aggregator)
topology.add_edge(beta, aggregator)

# analysis topology to get federated group props
federated_group_props = topo.analysis(topology, beta)
assert len(federated_group_props) == 1
federated_group_prop = federated_group_props[0]

# build pipe
pipes = openfed.federated.init_federated_group(federated_group_prop)

beta_pipe = pipes[0]

# transfer message
print(beta_pipe.direct_get('message_0'))

beta_pipe.direct_set('message_1', 'hello world from beta to aggregator')

data = torch.tensor(2)
with beta_pipe.dist_props:
    # download data from aggregator
    assert beta_pipe.download() == -1

    # upload data to aggregator
    beta_pipe.upload(data)

time.sleep(1)
```

The output of aggregator:

```
(openfed)  python aggregator.py
hello world from alpha to aggregator
hello world from beta to aggregator
```

The output of collaborator alpha:

```
(openfed) python collaborator_alpha.py
hello world from aggregator to alpha
```

The output of collaborator beta:

```
(openfed) python collaborator_beta.py
hello world from aggregator to beta
```

CHAPTER

EIGHT

FUNCTIONAL

There are three mainly kinds of hooks, i.e., package hook, unpackage hook and step hook. All these hooks can be automatically register to a maintainer in `with maintainer` context. There is a `nice` value to control the order of the hooks to apply. A lower `nice` value means a higher priority.

8.1 Step

`Step` hook is mainly used for control aggregator operations. You can define a step hook and register it to a maintainer via :func:`register_step_hook`.

8.2 Package and Unpackage

`Package` and `Unpackage` hooks usually pair up with each other. This hook is used for pack data before upload and unpack data after download. You can define a package hook and register it to a maintainer via :func:`register_package_hook`. You can also define a unpackage hook and register it to a maintainer via :func:`register_unpackage_hook`.

CHAPTER

NINE

OPTIM

9.1 FederatedOptimizer

:class:`FederatedOptimizer` wrapper an :class:`torch.optim.Optimizer`, and provide some necessary functions for federated learning. The simplest way to generate an federated optimizer is to use this wrapper like:

```
sgd = optim.SGD(...)
# For aggregator
fed_sgd = FederatedOptimizer(sgd, role=openfed.aggregator)
# For collaborator
fed_sgd = FederatedOptimizer(sgd, role=openfed.collaborator)
```

FederatedOptimizer usually has different behaviors when it plays different roles. It has two special functions, namely :func:`acg_step` and :func:`round`.

- `acg_step`: If you want to calculate some statistic metric of dataset with the downloaded model, you can implement here. This function will be called before the training phase.
- `round`: If you need to calculate some statistic metric of dataset with the trained model, you can implement here. This function will be called after the training phase.

CHAPTER

TEN

TOOLS

10.1 TopoBuilder

`TopoBuilder` provides a common line for you to build a massive topology graph more easily. Then you can save it to disk and load it in your code.

The following example shows how to build a hierarchical topology graph:

```
(openfed) python -m openfed.tools.topo_builder
A script to build topology.
<OpenFed>: add_node
Nick Name
red
Does this node requires address? (Y/n)
n
<OpenFed> Node
nick name: red
<OpenFed> Address
+---------+-------------+------------+------+
| backend | init_method | world_size | rank |
+---------+-------------+------------+------+
|   null  |     null    |     2      |  -1  |
+---------+-------------+------------+------+


<OpenFed>: add_node
Nick Name
green
Does this node requires address? (Y/n)
n
<OpenFed> Node
nick name: green
<OpenFed> Address
+---------+-------------+------------+------+
| backend | init_method | world_size | rank |
+---------+-------------+------------+------+
|   null  |     null    |     2      |  -1  |
+---------+-------------+------------+------+


<OpenFed>: add_node
```

(continues on next page)

(continued from previous page)

```
Nick Name
purple
Does this node requires address? (Y/n)
n
<OpenFed> Node
nick name: purple
<OpenFed> Address
+---------+-------------+------------+------+
| backend | init_method | world_size | rank |
+---------+-------------+------------+------+
|   null  |     null    |     2      |  -1  |
+---------+-------------+------------+------+


<OpenFed>: add_node
Nick Name
blue
Does this node requires address? (Y/n)
y
Backend (gloo, mpi, nccl)
gloo
Init method i.e., tcp://localhost:1994, file:///tmp/openfed.sharedfile)
tcp://localhost:1994
<OpenFed> Node
nick name: blue
<OpenFed> Address
+---------+---------------------+------------+------+
| backend |     init_method     | world_size | rank |
+---------+---------------------+------------+------+
|   gloo  | tcp://lo...ost:1994 |     2      |  -1  |
+---------+---------------------+------------+------+


<OpenFed>: add_node
Nick Name
yellow
Does this node requires address? (Y/n)
y
Backend (gloo, mpi, nccl)
mpi
Init method i.e., tcp://localhost:1994, file:///tmp/openfed.sharedfile)
file://tmp/openfed.sharedfile
<OpenFed> Node
nick name: yellow
<OpenFed> Address
+---------+---------------------+------------+------+
| backend |     init_method     | world_size | rank |
+---------+---------------------+------------+------+
|   mpi   | file://t...aredfile |     2      |  -1  |
+---------+---------------------+------------+------+
```

(continues on next page)

(continued from previous page)

```
<OpenFed>: build_edge
Start node nick name
red
End node nick name
blue
<OpenFed> Edge
|red -> blue.

<OpenFed>: build_edge
Start node nick name
green
End node nick name
blue
<OpenFed> Edge
|green -> blue.

<OpenFed>: build_edge
Start node nick name
blue
End node nick name
yellow
<OpenFed> Edge
|blue -> yellow.

<OpenFed>: build_edge
Start node nick name
purple
End node nick name
yellow
<OpenFed> Edge
|purple -> yellow.

<OpenFed>: save
Filename:
topology
+--------+-----+-------+--------+------+--------+
| CO\AG  | red | green | purple | blue | yellow |
+--------+-----+-------+--------+------+--------+
|  red   |  .  |   .   |   .    |  ^   |   .    |
| green  |  .  |   .   |   .    |  ^   |   .    |
| purple |  .  |   .   |   .    |  .   |   ^    |
|  blue  |  .  |   .   |   .    |  .   |   ^    |
| yellow |  .  |   .   |   .    |  .   |   .    |
+--------+-----+-------+--------+------+--------+
<OpenFed>: analysis
Folder to save the analysis result:
props
Processing red
[{'role': 'openfed_collaborator', 'nick_name': 'red', 'address': {'backend': 'gloo',
↪'init_method': 'tcp://localhost:1994', 'world_size': 3, 'rank': 2}}]
Processing green
[{'role': 'openfed_collaborator', 'nick_name': 'green', 'address': {'backend': 'gloo',
↪'init_method': 'tcp://localhost:1994', 'world_size': 3, 'rank': 1}}]
```

(continues on next page)

(continued from previous page)

```
Processing purple
[{'role': 'openfed_collaborator', 'nick_name': 'purple', 'address': {'backend': 'mpi',
↪'init_method': 'file://tmp/openfed.sharedfile', 'world_size': 3, 'rank': 2}}]
Processing blue
[{'role': 'openfed_aggregator', 'nick_name': 'blue', 'address': {'backend': 'gloo',
↪'init_method': 'tcp://localhost:1994', 'world_size': 3, 'rank': 0}}, {'role': 'openfed_
↪collaborator', 'nick_name': 'blue', 'address': {'backend': 'mpi', 'init_method':
↪'file://tmp/openfed.sharedfile', 'world_size': 3, 'rank': 1}}]
Processing yellow
[{'role': 'openfed_aggregator', 'nick_name': 'yellow', 'address': {'backend': 'mpi',
↪'init_method': 'file://tmp/openfed.sharedfile', 'world_size': 3, 'rank': 0}}]
<OpenFed>: exit
```

10.2 Simulator

`Simulator`, which is similar with `torch.distributed.launch`, is a module that spawns up multiple federated training processes on each of the training nodes. It will build a centralized topology automatically. It is very useful while simulating massive nodes to do the federated learning experience.

Write a piece of code, named `run.py`:

```
import argparse

parser = argparse.ArgumentParser()

parser.add_argument('--props')

args = parser.parse_args()

print(args.props)
```

Usage:

```
(openfed) python -m openfed.tools.simulator --nproc 10 run.py
/tmp/aggregator.json
/tmp/collaborator-1.json
/tmp/collaborator-2.json
/tmp/collaborator-3.json
/tmp/collaborator-4.json
/tmp/collaborator-5.json
/tmp/collaborator-6.json
/tmp/collaborator-7.json
/tmp/collaborator-8.json
/tmp/collaborator-9.json
```

CHAPTER

ELEVEN

TOPO

11.1 Node

Each device is regarded as a `Node` with `nick_name` and `address`. The nick name is the identification for each node and needs to be unique. Any nodes could connect to others via the address. Only when two nodes have the same address and nick name, we will regard them as the some one.

For example, we can define two nodes:

```
>>> import openfed
>>> alpha = openfed.topo.Node('alpha node', openfed.default_tcp_address)
>>> beta = openfed.topo.Node('beta node', openfed.default_file_address)
>>> alpha
<OpenFed> Node
nick name: alpha node
<OpenFed> Address
+---------+---------------------+------------+------+
| backend |     init_method     | world_size | rank |
+---------+---------------------+------------+------+
|   gloo  | tcp://localhost:... |     2      |  -1  |
+---------+---------------------+------------+------+


>>> beta
<OpenFed> Node
nick name: beta node
<OpenFed> Address
+---------+---------------------+------------+------+
| backend |     init_method     | world_size | rank |
+---------+---------------------+------------+------+
|   gloo  | file:///tmp/open... |     2      |  -1  |
+---------+---------------------+------------+------+
```

11.2 Edge

The relation between two nodes is determined via `Edge`. An `Edge` with two attributions:

- `start`: The start node, namely the collaborator nodes.
- `end`: The end node, namely the aggregator nodes.

If you want to build a connection between alpha(collaborator) and beta(aggregator), you may need a piece of code like:

```
>>> edge = openfed.topo.Edge(alpha, beta)
>>> edge
<OpenFed> Edge
|alpha node -> beta node.
```

In `OpenFed`, all the connection relationship should be represented as a `Topology`.

11.3 Topology

In `OpenFed`, we use `Topology` to manage massive nodes and edges. Here, we try to build a very simple centralized topology between three nodes, alpha(aggregator), beta(collaborator), gamma(collaborator).

```
>>> import openfed
>>> # define node
>>> alpha = openfed.topo.Node('alpha node', openfed.default_tcp_address)
>>> # the address of collaborator can be ignored.
>>> beta = openfed.topo.Node('beta node', openfed.empty_address)
>>> gamma = openfed.topo.Node('gamma node', openfed.empty_address)
>>> # define an empty topology
>>> topology = openfed.topo.Topology()
>>> # add nodes to topology
>>> topology.add_node(alpha)
>>> topology.add_node(beta)
>>> topology.add_node(gamma)
>>> # add edge
>>> topology.add_edge(beta, alpha)
>>> topology.add_edge(gamma, alpha)
>>> topology
+------------+------------+-----------+------------+
|   CO\AG    | alpha node | beta node | gamma node |
+------------+------------+-----------+------------+
| alpha node |     .      |     .     |     .      |
| beta node  |     ^      |     .     |     .      |
| gamma node |     ^      |     .     |     .      |
+------------+------------+-----------+------------+
```

11.4 FederatedGroup

We will analysis `Topology` and build a `FederatedGroup` for each node. Whatever the topology is, we will divide it into many federated groups. In each group, the node can only be a `aggregator` or a `collaborator`. In different groups, the node can play different roles.

Federated groups of alpha node:

```
>>> federated_groups = openfed.topo.analysis(topology, alpha)
>>> federated_groups
[<OpenFed> FederatedProperties
+--------------------+------------+
|        role        | nick_name  |
+--------------------+------------+
| openfed_aggregator | alpha node |
+--------------------+------------+
<OpenFed> Address
+---------+---------------------+------------+------+
| backend |     init_method     | world_size | rank |
+---------+---------------------+------------+------+
|   gloo  | tcp://localhost:... |     3      |  0   |
+---------+---------------------+------------+------+

]
```

Federated groups of beta node:

```
>>> federated_groups = openfed.topo.analysis(topology, beta)
>>> federated_groups
[<OpenFed> FederatedProperties
+----------------------+-----------+
|         role         | nick_name |
+----------------------+-----------+
| openfed_collaborator | beta node |
+----------------------+-----------+
<OpenFed> Address
+---------+---------------------+------------+------+
| backend |     init_method     | world_size | rank |
+---------+---------------------+------------+------+
|   gloo  | tcp://localhost:... |     3      |  1   |
+---------+---------------------+------------+------+

]
```

Federated groups of gamma node:

```
>>> federated_groups = openfed.topo.analysis(topology, gamma)
>>> federated_groups
[<OpenFed> FederatedProperties
+----------------------+------------+
|         role         | nick_name  |
+----------------------+------------+
| openfed_collaborator | gamma node |
+----------------------+------------+
```

(continues on next page)

(continued from previous page)

```
<OpenFed> Address
+---------+---------------------+------------+------+
| backend |     init_method     | world_size | rank |
+---------+---------------------+------------+------+
|   gloo  | tcp://localhost:... |     3      |  2   |
+---------+---------------------+------------+------+

]
```

You can refer to `openfed.tools.topo_builder` for more details about how to build a complex topology.

CHAPTER

TWELVE

UTILS

This component provides some useful functions, such as `seed_everything`, `time_string`.

12.1 Format output as table

Sometimes, we can receive a better visualization via show some data in a table.

You can do this with :func:`tablist`:

```
>>> from openfed.utils import tablist
>>> head = ['a', 'b', 'c', 'd', 'e', 'f']
>>> data = [1, 2, 3, 4, 5, 6]
>>> print(tablist(head, data, 3))
+---+---+---+
| a | b | c |
+---+---+---+
| 1 | 2 | 3 |
+---+---+---+
+---+---+---+
| d | e | f |
+---+---+---+
| 4 | 5 | 6 |
+---+---+---+
>>> print(tablist(head, data, force_in_one_row=True))
+---+---+---+---+---+---+
| a | b | c | d | e | f |
+---+---+---+---+---+---+
| 1 | 2 | 3 | 4 | 5 | 6 |
+---+---+---+---+---+---+
>>> data = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12]
>>> print(tablist(head, data, force_in_one_row=True))
+---+---+---+----+----+----+
| a | b | c | d  | e  | f  |
+---+---+---+----+----+----+
| 1 | 2 | 3 | 4  | 5  | 6  |
| 7 | 8 | 9 | 10 | 11 | 12 |
+---+---+---+----+----+----+
```

CHAPTER

THIRTEEN

API

:class:`API` provides a simple wrapper of aggregator logistic. After define an :class:`API`, you can use it in the backend:

```
api.start()
api.join()
```

or run it on the main process:

```
api.run()
```

When it runs on backend, you have to acquire the :attr:`openfed.federated.openfed_lock` before start your main process distributed training. The :attr:`openfed.federated.openfed_lock` will lock the data-transfer operation at openfed, but has no influence on message-transfer. Since openfed share the same module with torch to build communication between two process, we have to use this lock to control the data transfer operation.

CHAPTER

FOURTEEN

PAILLIER CRYPTO

This script shows a simple demostration about `Paillier Crypto` algorithm on federated MNIST dataset.

14.1 Public and Private Key

```
import os
import torch

from openfed.functional import key_gen

if not os.path.isfile('/tmp/public.key') or not os.path.isfile('/tmp/private.key'):
    public_key, private_key = key_gen()
    public_key.save('/tmp/public.key')
    private_key.save('/tmp/private.key')
    print("Generate new public and private key.")
else:
    print("Found public and private key under '/tmp'")
```

```
Found public and private key under '/tmp'
```

14.2 Network and Loss

```
import torch.nn as nn

def build_network():
    network = nn.Linear(784, 10)
    loss_fn = nn.CrossEntropyLoss()
    return network, loss_fn
```

14.3 Optimizer

```
from openfed.federated import is_aggregator, is_collaborator
from openfed.optim import FederatedOptimizer
import torch.optim as optim

def build_optimizer(network, role):
    sgd = optim.SGD(network.parameters(), lr=1.0 if is_aggregator(role) else 0.1)
    fed_sgd = FederatedOptimizer(sgd, role=role)
    return fed_sgd
```

14.4 Topology

```
import openfed
import openfed.topo as topo

def build_topology():
    aggregator_node = topo.Node('aggregator', address=openfed.default_tcp_address)
    collaborator_node = topo.Node('collaborator', address=openfed.empty_address)

    topology = topo.Topology()
    topology.add_edge(collaborator_node, aggregator_node)

    return topology
```

14.5 Federated Group Properties

```
def build_props(topology, role):
    fed_props = topo.analysis(topology, 'aggregator' if is_aggregator(role) else
↪'collaborator')
    assert len(fed_props) == 1
    fed_prop = fed_props[0]

    return fed_prop
```

14.6 Maintainer

```
from openfed.core import Maintainer

def build_maintainer(fed_prop, state_dict, role, part_per_round):
    maintainer = Maintainer(fed_prop, state_dict)

    with maintainer:
        openfed.functional.device_alignment()
        if is_aggregator(role):
```

(continues on next page)

(continued from previous page)

```
            openfed.functional.count_step(part_per_round)
        else:
            public_key = openfed.functional.PublicKey.load('/tmp/public.key')
            openfed.functional.paillier_package(public_key)
    return maintainer
```

14.7 Dataset

```
from torchvision.datasets import MNIST
from torchvision.transforms import ToTensor
from torch.utils.data import DataLoader
from openfed.data import IIDPartitioner, PartitionerDataset

def build_dataset():
    # Dataset
    mnist = MNIST(r'/tmp/', True, ToTensor(), download=True)
    fed_mnist = PartitionerDataset(mnist, total_parts=10, partitioner=IIDPartitioner())

    # Dataloader
    dataloader = DataLoader(fed_mnist, batch_size=10, shuffle=True, num_workers=0, drop_
↪last=False)

    return dataloader
```

14.8 API

```
def build_API(maintainer, fed_sgd, rounds):
    private_key = openfed.functional.PrivateKey.load('/tmp/private.key')
    api = openfed.API(maintainer,
                fed_sgd,
                rounds,
                agg_func=openfed.functional.paillier_aggregation,
                agg_func_kwargs=dict(private_key=private_key),
               )
    return api
```

14.9 Step

```
import random
import time

def step(mt, dataloader, network, loss_fn, fed_optim, rounds, part_per_round):
    version = 0
    outter_losses = []
    for outter in range(rounds):
```

(continues on next page)

(continued from previous page)

```
        outter_loss = []
        for inner in range(part_per_round):
            mt.update_version(version)
            mt.step(upload=False)

            part_id = random.randint(0, 9)
            dataloader.dataset.set_part_id(part_id)

            network.train()
            losses = []
            tic = time.time()
            for data in dataloader:
                x, y = data
                output = network(x.view(-1, 784))
                loss = loss_fn(output, y)

                fed_optim.zero_grad()
                loss.backward()
                fed_optim.step()
                losses.append(loss.item())
            toc = time.time()
            loss = sum(losses) / len(losses)
            outter_loss.append(loss)
            duration = toc - tic

            fed_optim.round()

            mt.update_version(version + 1)
            mt.package(fed_optim)
            mt.step(download=False)
            fed_optim.clear_state_dict()
        version += 1
        outter_losses.append(sum(outter_loss) / len(outter_loss))
    torch.save(outter_losses, '/tmp/outter_losses')
```

14.10 Main Function

```
def main_function(role, rounds, part_per_round):
    # Network
    network, loss_fn = build_network()

    if is_aggregator(role):
        print(network)
        print(loss_fn)

    # Optimizer
    fed_sgd = build_optimizer(network, role)

    if is_aggregator(role):
        print(fed_sgd)
```

(continues on next page)

(continued from previous page)

```
    # Topology
    topology = build_topology()

    if is_aggregator(role):
        print(topology)

    # Federated Group Properties
    fed_prop = build_props(topology, role)

    print(fed_prop)

    # Maintainer
    maintainer = build_maintainer(fed_prop, network.state_dict(keep_vars=True), role,␣
↪part_per_round)

    if is_aggregator(role):
        api = build_API(maintainer, fed_sgd, rounds)
        api.start()
    else:
        dataloader = build_dataset()
        step(maintainer, dataloader, network, loss_fn, fed_sgd, rounds, part_per_round)
```

14.11 Enable colorize output

```
from openfed.utils.utils import FMT

FMT.color = True
```

14.12 Run

```
from multiprocessing import Process
import openfed
rounds = 3
part_per_round = 5

aggregator_pc = Process(target=main_function, args=(openfed.federated.aggregator, rounds,
↪ part_per_round))
collaborator = Process(target=main_function, args=(openfed.federated.collaborator,␣
↪rounds, part_per_round))

aggregator_pc.start()
collaborator.start()

aggregator_pc.join()
collaborator.join()
```

```
Linear(in_features=784, out_features=10, bias=True)
CrossEntropyLoss()
[0;34m<OpenFed>[0m [0;35mFederatedProperties[0m
+----------------------+--------------+
|         role         |  nick_name   |
+----------------------+--------------+
| openfed_collaborator | collaborator |
+----------------------+--------------+
[0;34m<OpenFed>[0m [0;35mAddress[0m
+---------+---------------------+------------+------+
| backend |     init_method     | world_size | rank |
+---------+---------------------+------------+------+
|   gloo  | tcp://lo...ost:1994 |     2      |  1   |
+---------+---------------------+------------+------+


[0;34m<OpenFed>[0m [0;35mFederatedOptimizer[0m
SGD (
Parameter Group 0
    dampening: 0
    lr: 1.0
    momentum: 0
    nesterov: False
    weight_decay: 0
)

+--------------+--------------+------------+
|    CO\AG     | collaborator | aggregator |
+--------------+--------------+------------+
| collaborator |      .       |     ^      |
|  aggregator  |      .       |     .      |
+--------------+--------------+------------+
[0;34m<OpenFed>[0m [0;35mFederatedProperties[0m
+--------------------+------------+
|        role        | nick_name  |
+--------------------+------------+
| openfed_aggregator | aggregator |
+--------------------+------------+
[0;34m<OpenFed>[0m [0;35mAddress[0m
+---------+---------------------+------------+------+
| backend |     init_method     | world_size | rank |
+---------+---------------------+------------+------+
|   gloo  | tcp://lo...ost:1994 |     2      |  0   |
+---------+---------------------+------------+------+




100%|| 3/3 [00:12<00:00,  4.10s/it]
```

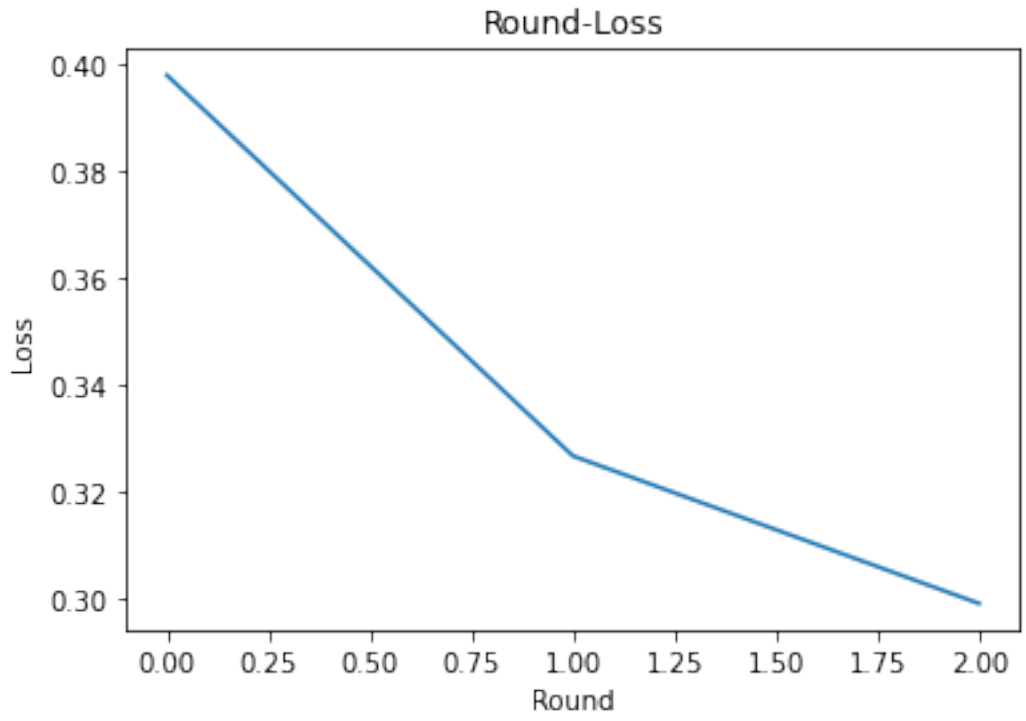
14.13 Result

```
%matplotlib inline

import matplotlib.pyplot as plt

outter_losses = torch.load('/tmp/outter_losses')

plt.figure()
plt.plot(outter_losses)
plt.title('Round-Loss')
plt.xlabel('Round')
plt.ylabel('Loss')
plt.show()
```



CHAPTER

FIFTEEN

SIMULATOR

This script provides a most simplest way to do federated learning with simultor.

15.1 Script

```
import argparse
import random
import torch
import torch.nn as nn
from torch.utils.data import DataLoader
from torchvision.datasets import MNIST
from torchvision.transforms import ToTensor

import openfed
from openfed.data import IIDPartitioner, PartitionerDataset

parser = argparse.ArgumentParser(description='Simulator')
parser.add_argument('--props', type=str, default='/tmp/aggregator.json')
args = parser.parse_args()

props = openfed.federated.FederatedProperties.load(args.props)
assert len(props) == 1
props = props[0]

network = nn.Linear(784, 10)
loss_fn = nn.CrossEntropyLoss()

sgd = torch.optim.SGD(
    network.parameters(), lr=1.0 if props.aggregator else 0.1)
fed_sgd = openfed.optim.FederatedOptimizer(sgd, props.role)

maintainer = openfed.core.Maintainer(props, network.state_dict(keep_vars=True))

with maintainer:
    openfed.functional.device_alignment()
    if props.aggregator:
        openfed.functional.count_step(props.address.world_size - 1)

rounds = 10
```

(continues on next page)

(continued from previous page)

```
if maintainer.aggregator:
    api = openfed.API(maintainer, fed_sgd, rounds,
                      openfed.functional.average_aggregation)
    api.run()
else:
    mnist = MNIST(r'/tmp/', True, ToTensor(), download=True)
    fed_mnist = PartitionerDataset(
        mnist, total_parts=100, partitioner=IIDPartitioner())

    dataloader = DataLoader(
        fed_mnist, batch_size=10, shuffle=True, num_workers=0, drop_last=False)

    version = 0
    for outter in range(rounds):
        maintainer.update_version(version)
        maintainer.step(upload=False)

        part_id = random.randint(0, 9)
        fed_mnist.set_part_id(part_id)

        network.train()
        losses = []
        for data in dataloader:
            x, y = data
            output = network(x.view(-1, 784))
            loss = loss_fn(output, y)

            fed_sgd.zero_grad()
            loss.backward()
            fed_sgd.step()
            losses.append(loss.item())
        loss = sum(losses) / len(losses)

        fed_sgd.round()

        maintainer.update_version(version + 1)
        maintainer.package(fed_sgd)
        maintainer.step(download=False)
        fed_sgd.clear_state_dict()
        version += 1
```

Copy and save these piece of code as `examples/run.py`.

15.2 Run

```
# Launch 6 process (1 for aggregator, 5 for collaborator) to do simulation.
!python -m openfed.tools.simulator --nproc 6 run.py
```

```
[W ProcessGroupGloo.cpp:559] Warning: Unable to resolve hostname to a (local) address.␣
↪Using the loopback address as fallback. Manually set the network interface to bind to␣
↪with GLOO_SOCKET_IFNAME. (function operator())
[W ProcessGroupGloo.cpp:559] Warning: Unable to resolve hostname to a (local) address.␣
↪Using the loopback address as fallback. Manually set the network interface to bind to␣
↪with GLOO_SOCKET_IFNAME. (function operator())
[W ProcessGroupGloo.cpp:559] Warning: Unable to resolve hostname to a (local) address.␣
↪Using the loopback address as fallback. Manually set the network interface to bind to␣
↪with GLOO_SOCKET_IFNAME. (function operator())
[W ProcessGroupGloo.cpp:559] Warning: Unable to resolve hostname to a (local) address.␣
↪Using the loopback address as fallback. Manually set the network interface to bind to␣
↪with GLOO_SOCKET_IFNAME. (function operator())
[W ProcessGroupGloo.cpp:559] Warning: Unable to resolve hostname to a (local) address.␣
↪Using the loopback address as fallback. Manually set the network interface to bind to␣
↪with GLOO_SOCKET_IFNAME. (function operator())
[W ProcessGroupGloo.cpp:559] Warning: Unable to resolve hostname to a (local) address.␣
↪Using the loopback address as fallback. Manually set the network interface to bind to␣
↪with GLOO_SOCKET_IFNAME. (function operator())
100%|| 10/10 [00:01<00:00,  5.90it/s]
```

```

```

CHAPTER

SIXTEEN

V0.0.0

In order to flexibly support more federated algorithms and projects, like `scaffold`, `mmcv`, the directory of `openfed` might be refactored.

`v0.0.0`'s directory was organized as follows.

```
openfed
├── __init__.py
├── api.py
├── common
│   ├── __init__.py
│   ├── address.py
│   └── meta.py
├── core
│   ├── __init__.py
│   ├── const.py
│   ├── functional.py
│   └── maintainer.py
├── data
│   ├── __init__.py
│   ├── datasets.py
│   ├── partitioner.py
│   └── utils.py
├── federated
│   ├── __init__.py
│   ├── const.py
│   ├── exceptions.py
│   ├── functional.py
│   ├── pipe.py
│   └── props.py
├── functional
│   ├── __init__.py
│   ├── agg.py
│   ├── const.py
│   ├── hooks.py
│   ├── paillier.py
│   ├── reduce.py
│   └── step.py
├── optim
│   ├── __init__.py
│   ├── elastic.py
│   ├── fed_optim.py
```

(continues on next page)

(continued from previous page)

```
│   ├── prox.py
│   └── scaffold.py
├── tools
│   ├── __init__.py
│   ├── simulator.py
│   └── topo_builder.py
├── topo
│   ├── __init__.py
│   ├── functional.py
│   └── topo.py
├── utils
│   ├── __init__.py
│   ├── table.py
│   └── utils.py
└── version.py
```

CHAPTER

SEVENTEEN

FREQUENTLY ASKED QUESTIONS

We list some common troubles faced by many users and their corresponding solutions here. Feel free to enrich the list if you find any frequent issues and have ways to help others to solve them.

CHAPTER

EIGHTEEN

MIT LICENSE

CHAPTER

NINETEEN

OPENFED CONTRIBUTOR LICENSE AGREEMENT

In order to clarify the intellectual property license granted with Contributions from any person or entity, FederalLab must have a Contributor License Agreement ("CLA") on file that has been signed by each Contributor, indicating agreement to the license terms below. This license is for your protection as a Contributor as well as the protection of FederalLab; it does not change your rights to use your own Contributions for any other purpose. You accept and agree to the following terms and conditions for Your present and future Contributions submitted to FederalLab. Except for the license granted herein to FederalLab and recipients of software distributed by FederalLab, You reserve all right, title, and interest in and to Your Contributions.

1. Definitions. "You" (or "Your") shall mean the copyright owner or legal entity authorized by the copyright owner that is making this Agreement with FederalLab. For legal entities, the entity making a Contribution and all other entities that control, are controlled by, or are under common control with that entity are considered to be a single Contributor. For the purposes of this definition, "control" means (i) the power, direct or indirect, to cause the direction or management of such entity, whether by contract or otherwise, or (ii) ownership of fifty percent (50%) or more of the outstanding shares, or (iii) beneficial ownership of such entity. "Contribution" shall mean any original work of authorship, including any modifications or additions to an existing work, that is intentionally submitted by You to FederalLab for inclusion in, or documentation of, any of the products owned or managed by FederalLab (the "Work"). For the purposes of this definition, "submitted" means any form of electronic, verbal, or written communication sent to FederalLab or its representatives, including but not limited to communication on electronic mailing lists, source code control systems, and issue tracking systems that are managed by, or on behalf of, FederalLab for the purpose of discussing and improving the Work, but excluding communication that is conspicuously marked or otherwise designated in writing by You as "Not a Contribution."
2. Grant of Copyright License. Subject to the terms and conditions of this Agreement, You hereby grant to FederalLab and to recipients of software distributed by FederalLab a perpetual, worldwide, non-exclusive, no-charge, royalty-free, irrevocable copyright license to reproduce, prepare derivative works of, publicly display, publicly perform, sublicense, and distribute Your Contributions and such derivative works.
3. Grant of Patent License. Subject to the terms and conditions of this Agreement, You hereby grant to FederalLab and to recipients of software distributed by FederalLab a perpetual, worldwide, non-exclusive, no-charge, royalty-free, irrevocable (except as stated in this section) patent license to make, have made, use, offer to sell, sell, import, and otherwise transfer the Work, where such license applies only to those patent claims licensable by You that are necessarily infringed by Your Contribution(s) alone or by combination of Your Contribution(s) with the Work to which such Contribution(s) was submitted. If any entity institutes patent litigation against You or any other entity (including a cross-claim or counterclaim in a lawsuit) alleging that your Contribution, or the Work to which you have contributed, constitutes direct or contributory patent infringement, then any patent licenses granted to that entity under this Agreement for that Contribution or Work shall terminate as of the date such litigation is filed.
4. You represent that you are legally entitled to grant the above license. If your employer(s) has rights to intellectual property that you create that includes your Contributions, you represent that you have received permission to make Contributions on behalf of that employer, that your employer has waived such rights for your Contributions to FederalLab, or that your employer has executed a separate Corporate CLA with FederalLab.
5. You represent that each of Your Contributions is Your original creation (see section 7 for submissions on behalf

of others). You represent that Your Contribution submissions include complete details of any third-party license or other restriction (including, but not limited to, related patents and trademarks) of which you are personally aware and which are associated with any part of Your Contributions.

6. You are not expected to provide support for Your Contributions, except to the extent You desire to provide support. You may provide support for free, for a fee, or not at all. Unless required by applicable law or agreed to in writing, You provide Your Contributions on an "AS IS" BASIS, WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied, including, without limitation, any warranties or conditions of TITLE, NON-INFRINGEMENT, MERCHANTABILITY, or FITNESS FOR A PARTICULAR PURPOSE.

7. Should You wish to submit work that is not Your original creation, You may submit it to FederalLab separately from any Contribution, identifying the complete details of its source and of any license or other restriction (including, but not limited to, related patents, trademarks, and license agreements) of which you are personally aware, and conspicuously marking the work as "Submitted on behalf of a third-party: [named here]".

8. You agree to notify FederalLab of any facts or circumstances of which you become aware that would make these representations inaccurate in any respect.

CHAPTER

TWENTY

PULL REQUEST (PR)

20.1 What is PR

PR is the abbreviation of `Pull Request`. Here's the definition of PR in the official document of Github.

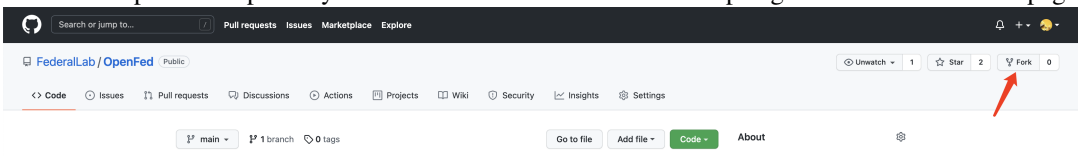
> Pull requests let you tell others about changes you've pushed to a branch in a repository on GitHub. Once a pull request is opened, you can discuss and review the potential changes with collaborators and add follow-up commits before your changes are merged into the base branch.

20.2 Basic Workflow

1. Get the most recent codebase
2. Checkout a new branch from the master branch
3. Commit your changes
4. Push your changes and create a PR
5. Discuss and review your code
6. Merge your branch to the master branch

20.3 Procedures in detail

1. Get the most recent codebase
   - When you work on your first PR
     - Fork the OpenFed repository: click the **fork** button at the top right corner of Github page



     - Clone forked repository to local

```
git clone git@github.com:XXX/OpenFed.git
```

     - Add source repository to upstream

```
git remote add upstream git@github.com:FederalLab/OpenFed.git
```

- After your first PR
  - Checkout master branch of the local repository and pull the latest master branch of the source repository

```
git checkout master
git pull upstream master
```

2. Checkout a new branch from the master branch

```
git checkout -b branchname
```

Tip: To make commit history clear, we strongly recommend you checkout the master branch before create a new branch.

1. Commit your changes

```
# coding
git add [files]
git commit -m 'messages'
```

2. Push your changes to the forked repository and create a PR
   - Push the branch to your forked remote repository

```
git push origin branchname
```

   - Revise PR message template to describe your motivation and modifications made in this PR. You can also link the related issue to the PR manually in the PR message (For more information, checkout the official guidance).
3. Discuss and review your code
   - After creating a pull request, you can ask a specific person to review the changes you've proposed.
   - Modify your codes according to reviewers' suggestions and then push your changes
4. Merge your branch to the master branch and delete the branch

```
git branch -d branchname # delete local branch
git push origin --delete branchname # delete remote branch
```

20.4 PR Specs

1. Use pre-commit hook to avoid issues of code style
2. One short-time branch should be matched with only one PR
3. Accomplish a detailed change in one PR. Avoid large PR
   - Bad: Support FedAvg
   - Acceptable: Add a new aggregate method of FedAvg
   - Good: Add a new aggragate function which enable the average operation.

4. Provide clear and significant commit message
5. Provide clear and meaningful PR description
    - Task name should be clarified in title. The general format is: [Prefix] Short description of the PR (Suffix)
    - Prefix: add new feature [Feature], fix bug [Fix], related to documents [Docs], in developing [WIP] (which will not be reviewed temporarily)
    - Introduce main changes, results and influences on other modules in short description
    - Associate related issues and pull requests with a milestone

CHAPTER

TWENTYONE

COMMON

CHAPTER

TWENTYTWO

CORE

CHAPTER

TWENTYTHREE

DATA

CHAPTER

TWENTYFOUR

FEDERATED

CHAPTER

TWENTYFIVE

FUNCTIONAL

CHAPTER

TWENTYSIX

OPTIM

CHAPTER

TWENTYSEVEN

TOOLS

CHAPTER

TWENTYEIGHT

TOPO

CHAPTER

TWENTYNINE

UTILS

CHAPTER

THIRTY

API

CHAPTER

THIRTYONE

HOW TO UPDATE THE DOCUMENTATION

We use sphinx to generate the documentation for this project. The documentation project has been initialized properly and we basically just need to update the actual content.

Install dependencies: `pip install -r ../requirements/docs.txt`.

If we ever change the code structure since last compilation, we may need to regenerate the docstring index:

```
sphinx-apidoc -f -o . ../openfed
sphinx-apidoc -f -o . ../examples
```

The command detects the code structure under `../openfed` and generates a series of `*.rst` files, such as `openfed.api.rst`. However, the docstring would not be compiled until we execute `make html` later.

We can also update the hand-crafted documents, including `intro.rst` and `tutorial.rst`. The `openfed.rst` is the entry file. We don't need to modify it unless we want to add more hand-crafted pages or adjust the order in the Contents page.

After completing revision on the .rst files, we would compile the documentation source code:

```
make clean
make html
```

The Makefile supports many targets. We choose html because we can easily host the documentation on a remote server:

```
cd _build/html
python -m http.server
```

CHAPTER

THIRTYTWO

INDICES AND TABLES

- genindex
- search