Federated Learning with flower

Release 0.1

hlg-mos-ml-private

Oct 25, 2021

CONTENTS:

CHAPTER

ONE

API REFERENCE

This page contains auto-generated API reference documentation[1].

1.1 har

1.1.1 Submodules

har.har

Inital model for HAR dataset.

This module defines the neural model and data loaders

Attributes:

DATA_ROOT [str] Default data directory

device [str] The device for training `cuda` or `cpu`

DATA_URL [str] Default data url for download

model [object] Model instance

Notes:

This is module contains some HACKs, and it can be / should be improved. For instance importing and exporting layers could use a `JSON` object. For this test we stick to python `dict`.

Module Contents

Classes

NeuralNetwork	Define the Neural Network Model.

[1] Created with sphinx-autoapi

Functions

parse_module(module: collections.OrderedDict) → Tuple[str, str]	Parse layer description.
load_data(test_path: str = None, train_path: str = None) → Tuple[torch.utils.data.DataLoader, torch.utils.data.DataLoader]	Load datasets.
train(model, dataloader, epochs, device) → None	Train model.
test(model, dataloader, device) → Tuple[float, float]	Test the model.

Attributes

DATA_ROOT
device
DATA_URL
model

har.har.**DATA_ROOT = ../../OUTPUT/**

har.har.**device**

har.har.**DATA_URL = https://archive.ics.uci.edu/ml/machine-learning-databases/00240/UCI%20HAR%20Dataset.zip**

har.har.**parse_module**(*module: collections.OrderedDict*) → Tuple[str, str]

Parse layer description.

PyTorch shows a layer description like this: 'Linear(in_features=561, out_features=512, bias=True)' This module parse this string to get the name of the layer (Linear) and the named arguments for this layer with correct types (in_features=561, out_features=512, bias=True)

module Dictionary with layer or activation description

layer_act Name of the layer or activation

kwrds keyword arguments dictionary, useful to recreate the layer

This is a HACK, and it can be / should be improved.

class har.har.**NeuralNetwork**(*model_dict: collections.OrderedDict = None*)

Bases: `torch.nn.Module`

Define the Neural Network Model.

The model is defined/loaded here.

set_nn(*self*, *model_dict: collections.OrderedDict = None*) → None

Set the network architecture from model_dict or defaults.

model_dict Dictionary with modules parameters

forward(*self*, *x: torch.Tensor*) → torch.Tensor
Do a forward pass.

x tensor

tensor logits

to_dict(*self*) → Dict
Return a dictionary with network layers.

stack_odict Ordered dictionary with layer configuration

This is part of a HACK, and it can be / should be improved.

from_dict(*self*, *model_dict: collections.OrderedDict*) → None
Set network layers from dictionary.

model_dict Ordered dictionary with layer configuration

This is part of a HACK, and it can be / should be improved.

to_string(*self*) → str
Return a string with a dictionary with network layers.

str String eith Ordered dictionary with layer configuration

This is part of a HACK, and it can be / should be improved. We could redifine the `__repr__` instead.

from_string(*self*, *model_str: str*) → None
Set network layers from a string of python dictionary.

model_str Ordered dictionary with layer configuration

This is part of a HACK, and it can be / should be improved.

har.har.**model**

har.har.**load_data**(*test_path: str = None*, *train_path: str = None*) → Tuple[torch.utils.data.DataLoader, torch.utils.data.DataLoader]
Load datasets.

test_path Path for test dataset

train_path Path for train dataset

tuple train DataLoader, test DataLoader

har.har.**train**(*model*, *dataloader*, *epochs*, *device*) → None
Train model.

model the model

dataloader Data Loader

epochs number of epochs

device CPU or GPU

har.har.**test**(*model*, *dataloader*, *device*) → Tuple[float, float]
Test the model.

model pytorch model

dataloader test DataLoader

device CPU or GPU

tuple test loss, accuracy

1.2 harserver

har-server.py script runs a Federated Learning server using flower.

The script can be run in command line:

1.2.1 Examples:

$./har-server.py - s 0.0.0.0:8080

or $python har-server.py -s [::]:8080

using environment variable HAR_SERVER $HAR_SERVER=[::]:8080 python har-server.py -m 2 -M 2 -r3

1.2.2 Attributes:

Model_dict [str] String that contains the `OrderedDict` with the layers.

1.2.3 Notes:

This is module contains some HACKs, and it can be / should be improved.

1.2.4 Module Contents

Functions

run_server(servername: str, min_fit_clients: int, min_available_clients: int, number_of_rounds: int, training_set: str = None, test_set: str = None, debug: bool = False) → None	Run a Federated Learning server using flower.
fit_config(rnd: int) → Dict[str, str]	Set config dictionary.

Attributes

Config

harserver.**Config**

harserver.**run_server**(*servername: str*, *min_fit_clients: int*, *min_available_clients: int*, *number_of_rounds: int*, *training_set: str = None*, *test_set: str = None*, *debug: bool = False*) → None

Run a Federated Learning server using flower.

Parameters

- **servername** – The FQDN or IP address with port, ex. mydomain.com:8080

- **min_fit_clients** – Minimum number of clients to be sampled for the next round of training
- **min_available_clients** – Minimum number of clients that need to be connected to the server before a training round
- **number_of_rounds** – Number of training rounds
- **training_set** – training dataset path
- **test_set** – test dataset path
- **debug** – Flag for trigger some debug strings

harserver.**fit_config**(*rnd: int*) → Dict[str, str]
Set config dictionary.

rnd number of rounds

config we should specify server/strategy configuration here

1.3 harclient

har-client.py script runs a Federated Learning client using flower.

The script can be run in command line:

1.3.1 Examples:

$./har-client.py -s localhost:8080 -T../../OUTPUT/2-ONS/train/2_ALL_train.csv -t../../OUTPUT/2-ONS/test/2_ALL_test.csv

or

$python har-client.py -s localhost:8080 -T../../OUTPUT/2-ONS/train/2_ALL_train.csv -t../../OUTPUT/2-ONS/test/2_ALL_test.csv

using environment variable HAR_SERVER

$HAR_SERVER=[::]:8080 python har-client.py -T../../OUTPUT/2-ONS/train/2_ALL_train.csv -t../../OUTPUT/2-ONS/test/2_ALL_test.csv

1.3.2 Attributes:

DEVICE [str] The device for training cuda or cpu

1.3.3 Module Contents

Classes

HARClient	Flower client implementing for HAR data using PyTorch.

Functions

main(servername: str, training_set: str, test_set: str, debug: bool) → None	Run a Federated Learning client using flower.
set_plot(number_of_rounds, title)	Set simple plot for accuracy vs round.

Attributes

USE_FEDBN
DEVICE

harclient.**USE_FEDBN :bool = False**

harclient.**DEVICE :str**

class harclient.**HARClient**(*model: object*, *trainloader: torch.utils.data.DataLoader*, *testloader: torch.utils.data.DataLoader*, *debug: bool = False*, *test_set_name: str = None*)

Bases: flwr.client.NumPyClient

Flower client implementing for HAR data using PyTorch.

Client implementation.

get_parameters(*self*) → List[numpy.ndarray]
Get parameters.

set_parameters(*self*, *parameters: List[numpy.ndarray]*) → None
Set parameters.

fit(*self*, *parameters: List[numpy.ndarray]*, *config: Dict[str, str]*) → Tuple[List[numpy.ndarray], int]
Set model parameters, train model, return updated model parameters.

parameters model parameters as a list of NumPy ndarrays, excluding parameters of BN layers when using FedBN

config complete

tuple updated parameters, size of train dataset, None

evaluate(*self*, *parameters: List[numpy.ndarray]*, *config: Dict[str, str]*) → Tuple[int, float, float]
Set model parameters, evaluate model on local test dataset, and return result.

parameters model parameters as a list of NumPy ndarrays, excluding parameters of BN layers when using FedBN

config complete.

tuple loss, size, and accuracy

harclient.**main**(*servername: str*, *training_set: str*, *test_set: str*, *debug: bool*) → None
Run a Federated Learning client using flower.

Parameters

- **servername** – The FQDN or IP address with port, ex. mydomain.com:8080

- **training_set** – training dataset path
- **test_set** – test dataset path
- **debug** – Flag for trigger some debug strings

harclient.**set_plot**(*number_of_rounds*, *title*)
Set simple plot for accuracy vs round.

1.4 server

1.4.1 Module Contents

server.**servername** =

1.5 simplephe

Simple Paillier Homomorphic Encrypted Aggregation.

Implement secure aggregation strategy based on flower implementation https://github.com/adap/flower/blob/main/src/py/flwr/server/strategy/fedavg.py

1.5.1 Module Contents

Classes

EncArray	Define an array for encryption/decryption.
KeyGenerator	Implement key generator for Paillier cryptosystem.
SimplePaillierAvg	Implement secure aggregation strategy.

Functions

encrypt_iterable(public_key, x: PlainIterable)	Encrypt an iterable of floats or integers.
decrypt_iterable(private_key, x: EncryptedIterable) → PlainIterable	Encrypt an iterable of floats or integers.
serialize_encrypted(enc: EncryptedNumberType, exponent: int) → str	Serialize an encrypted number.
load_encrypted_number(enc: int, exponent: int, public_key: PublicKeyType)	Load an encrypted number object.

Attributes

EncryptedNumberType
PublicKeyType
PrivateKeyType
PlainIterable
EncryptedIterable

simplephe.**EncryptedNumberType**

simplephe.**PublicKeyType**

simplephe.**PrivateKeyType**

simplephe.**PlainIterable**

simplephe.**EncryptedIterable**

simplephe.**encrypt_iterable**(*public_key*, *x:* PlainIterable)
Encrypt an iterable of floats or integers.

Parameters

- **public_key** – The public key object
- **x** – Iterable of floats

Returns

Return type map of EncryptedNumber

simplephe.**decrypt_iterable**(*private_key*, *x:* EncryptedIterable) → *PlainIterable*
Encrypt an iterable of floats or integers.

Parameters

- **private_key** – The private key object
- **x** – Iterable of encrypted objects

Returns

Return type list of decrypted numbers

simplephe.**serialize_encrypted**(*enc:* EncryptedNumberType, *exponent: int*) → str
Serialize an encrypted number.

Parameters

- **private_key** – The private key
- **x** – Iterable of encrypted objects

Returns

Return type list of decrypted numbers

simplephe.**load_encrypted_number**(*enc: int*, *exponent: int*, *public_key:* PublicKeyType)
Load an encrypted number object.

Parameters

- **enc** – encrypted serialized number as integer
- **exponent** – The exponent (precision)
- **public_key** – The public key

Returns

Return type list of decrypted numbers

class simplephe.**EncArray**(*shape*, *dtype=float*, *buffer=None*, *offset=0*, *strides=None*, *order=None*)

Bases: numpy.ndarray

Define an array for encryption/decryption.

This subclasses numpy ndarray, that allow us to leverage all the calculation machinery of numpy on cleartext and attempts to extend for addition and multiplication in case of encrypted numbers where applicable.

__array_finalize__(*self*, *obj*)

Add a custom attribute to this subclass EncArray.

__add__(*self*, *other*)

Override addition to allow encrypted objects.

encrypt(*self*, *public_key*) → *EncArray*

Encrypt this array.

decrypt(*self*, *private_key*) → *EncArray*

Decrypt this array.

serialize(*self*, *exp: int = - 32*) → List

Serialize EncArray of ciphertext objects.

Returns List with serialized ciphertext objects as strings

Return type List

serialize_ndarray(*self*, *exp: int = - 32*) → numpy.ndarray

Serialize EncArray of ciphertext objects.

Returns ndarray with serialized ciphertext objects as strings

Return type numpy.ndarray

classmethod deserialize(*cls*, *enc: Iterable[str]*, *public_key: phe.PaillierPublicKey*, *exp=- 32*) → *EncArray*

Deserialize an iterable of strings and return EncArray.

Parameters

- **enc** – Iterable of strings, serialized ciphertexts
- **public_key** – The public key used to get the ciphertexts

Returns Array with ciphertext objects

Return type *EncArray*

classmethod deserialize_ndarray(*cls*, *enc: numpy.ndarray*, *public_key: phe.PaillierPublicKey*, *exp=- 32*) → *EncArray*

Deserialize a numpy ndarray of strings and return EncArray.

Parameters

- **enc** – ndarray with serialized ciphertext objects as strings

- **public_key** – The public key used to get the ciphertexts

Returns Array with ciphertext objects

Return type *EncArray*

class simplephe.**KeyGenerator**(*key_length: int = 1024*)
Bases: object

Implement key generator for Paillier cryptosystem.

classmethod load_pkd(*cls*) → *KeyGenerator*
Load keys from pickle, do not generate them.

classmethod load(*cls*) → *KeyGenerator*
Load keys from JWK (JSON) files. Do not generate new keys.

classmethod load_public_pkd(*cls*, *filename: str = 'public.key'*) → *KeyGenerator*
Load public key only from pickle. Do not generate new keys.

classmethod load_public(*cls*, *filename: str = 'public_key.json'*) → *KeyGenerator*
Load public key from JWK (JSON) file. Do not generate new keys.

kid_metadata(*self*, *kind: str*) → str
Generate kid metadata string.

public_key_todict(*self*)
Export public key to python dict.

public_key_fromdict(*self*, *public_key*)
Import public key from python dict.

private_key_todict(*self*)
Export private key to python dict.

save_public_key_pkd(*self*, *filename: str = 'public.key'*) → None
Save the public key using pickle.

Parameters **filename** – The name for public key file

save_public_key(*self*, *filename: str = 'public_key.json'*) → None
Save the public key in a JWK format, JSON file.

Parameters **filename** – The name for public key json file

save_private_key_pkd(*self*, *filename: str = 'private.key'*) → None
Save the private key using pickle.

Parameters **filename** – The name for private key

save_private_key(*self*, *filename: str = 'private_key.json'*) → None
Save the private key in a JWK format, JSON file.

Parameters **filename** – The name for private key json file

save_keys_pkd(*self*) → None
Save the public and private keys using pickle.

save_keys(*self*) → None
Save the public and private keys in JWK format, JSON files.

load_public_key_pkd(*self*, *filename: str = 'public.key'*)
Load the public key using pickle.

Parameters **filename** – The name for public key file

load_public_key(*self*, *filename: str = 'public_key.json'*) → None
Load the public key from JSON file, JWK format.

Parameters **filename** – The name for public key json file

load_private_key_pkd(*self*, *filename: str = 'private.key'*)
Load the private key using pickle.

Parameters **filename** – The name for private key file

Returns

Return type key

load_private_key(*self*, *filename: str = 'private_key.json'*) → None
Load the private key from JSON file, JWK format.

Parameters **filename** – The name for private key json file

load_keys_pkd(*self*)
Load the public and private keys using pickle.

load_keys(*self*)
Load the public and private keys from JWK, JSON file.

class simplephe.**SimplePaillierAvg**(*fraction_fit: float = 0.1*, *fraction_eval: float = 0.1*, *min_fit_clients: int = 2*, *min_eval_clients: int = 2*, *min_available_clients: int = 2*, *eval_fn: Optional[Callable[[flwr.common.Weights], Optional[Tuple[float, Dict[str, flwr.common.Scalar]]]]] = None*, *on_fit_config_fn: Optional[Callable[[int], Dict[str, flwr.common.Scalar]]] = None*, *on_evaluate_config_fn: Optional[Callable[[int], Dict[str, flwr.common.Scalar]]] = None*, *accept_failures: bool = True*, *initial_parameters: Optional[flwr.common.Parameters] = None*)
Bases: flwr.server.strategy.fedavg.FedAvg

Implement secure aggregation strategy.

Extending the FedAvg class...

aggregate_fit(*self*, *rnd: int*, *results: List[Tuple[flwr.server.client_proxy.ClientProxy, flwr.common.FitRes]]*, *failures: List[BaseException]*) → Tuple[Optional[flwr.common.Parameters], Dict[str, flwr.common.Scalar]]
Aggregate fit results using weighted average.

evaluate(*self*, *parameters: flwr.common.Parameters*) → Optional[Tuple[float, Dict[str, flwr.common.Scalar]]]
Evaluate model parameters using an evaluation function.

CHAPTER

TWO

INDICES AND TABLES

- genindex
- modindex
- search

PYTHON MODULE INDEX

h

s

INDEX

N

P

R

S

T

U