

Decentralized Learning Made Easy With DecentralizePy

Akash Dhasade

Anne-Marie Kermarrec

Rafael Pires

Rishi Sharma

Milos Vujasinovic

Scalable Computing Systems Laboratory EPFL

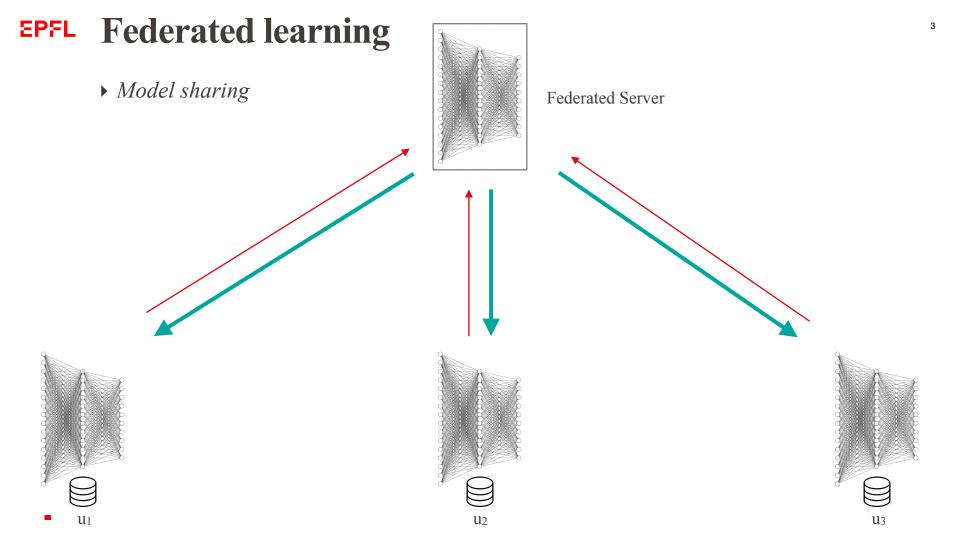
EPFL DecentralizePy

A framework for designing and studying decentralized learning systems.

Rapid development

Scalability

2



EPFL So many frameworks...

FedScale

A scalable and extensible federated learning engine and benchmark

GET STARTE

FedScale is a scalable and extensible open-source federated learning (FL) engine. It provides high-level APIs to implement FL algorithms, deploy and evaluate them at scale across diverse hardware and software backends. FedScale also includes the largest FL benchmark that contains FL tasks ranging from image classification and object detection to language modeling and speech recognition. Moreover, it includes datasets to faithfully emulate FL runtime environments where FL solutions will realistically be deployed.

We are actively developing FedScale, and welcome contributions fror community. Join our slack to keep up to date.

What's new? Flower Next Pilot Program >

[©] Flower

Flower A Friendly Federated Learning Framework

A unified approach to federated learning, analytics, and evaluation. Federate any workload, any ML framework, and any programming language.

Take the tutorial

to learn federated learning

Training	Tuning	Testing	Analytics
Model Manager			Selector
Client Manager	FedScale		Orchestrator
Data Catalog			Aggregator

TensorFlow Federated: Machine Learning on Decentralized Data

TensorFlow Federated (TFF) is an open-source framework for machine learning and other computations on decentralized data. TFF has been developed to facilitate open research and experimentation with Federated Learning (FL) *C_a* an approach to machine learning where a shared global model is trained across many participating clients that keep their training data locally. For example, FL has been used to train prediction models for mobile keyboards: *C_a* which uploading ensures they topic data to servers.

TFF enables developers to simulate the included federated learning algorithms on their models and data, as well as to experiment with novel algorithms. Researchers will find starting points and complete examples for many kinds of research. The building blocks provided by TFF can also be used to implement nonlearning computations, such as federated analytics. TFFs interfaces are organized in two main layers:

Federated Learning (FL) API

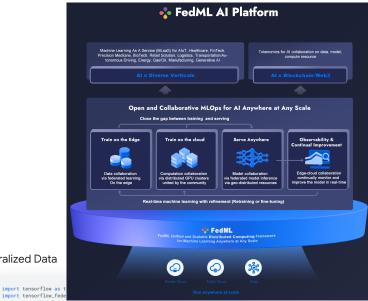
This layer offers a set of high-level interfaces that allow developers to apply the included implementations of federated training and evaluation to their existing TensorFlow models.

Federated Core (FC) API

At the core of the system is a set of lower-level interfaces for concisely expressing novel federated algorithms by combining TensorFlow with distributed communication operators within a strongly-typed functional programming environment. This layer also serves as the foundation upon which we've built Federated Learning.

TFF enables developers to declaratively express federated computations, so they could be deployed to diverse runtime environments. Included with TFF is a performant multi-machine simulation runtime for experiments. Please visit the turbrials and try to uty ourself!

For questions and support, find us at the tensorflow-federated tag on StackOverflow.



Load simulation data.

source, _ = tff.simulation.datasets.emnist.load_data()
def client_data(n):

Pick a subset of client devices to participate in training. train_data = [client_data(n) for n in range(3)]

Wrap a Keras model for use with TFF. def model_fn():

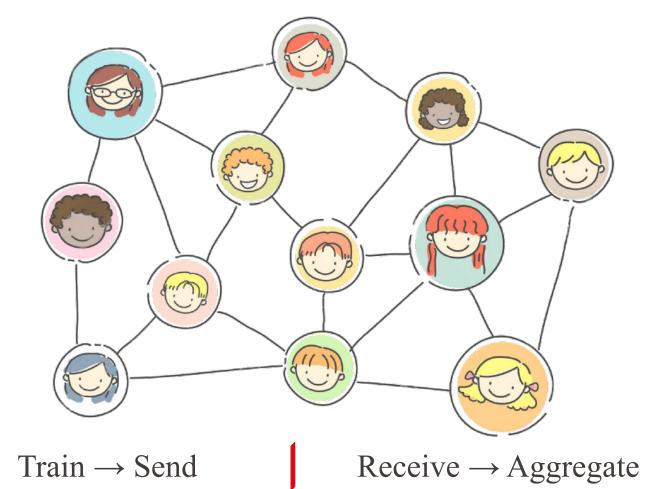
1)

return tff.learning.models.from_keras_model(
 model,
 input_spec=train_data[0].element_spec,
 loss=tf.keras.losses.SparseGategorialCrossentropy(),
 metrias=[tf.keras.metrics.SparseGategorialAccuracy()])

Simulate a few rounds of training with the selected client devices.

trainer = tff.learning.algorithms.build_weighted_fed_avg(model_fn, client_optimizer_fn=lambda: tf.keras.optimizers.SGD(0.1)) state = trainer.initialize() for _ in range(5):

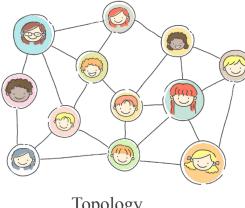
EPFL Decentralized learning



EPFL Decentralized learning

Simulations + No Re-usability!

EPFL Building decentralized learning systems



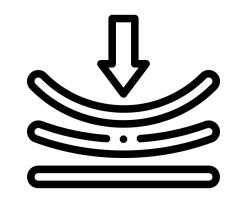
Topology



Roles

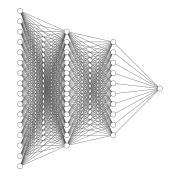


Communication



7

Compression

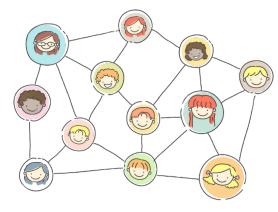




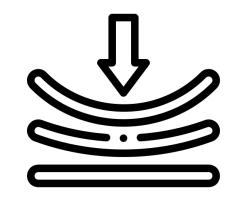
Datasets

Models

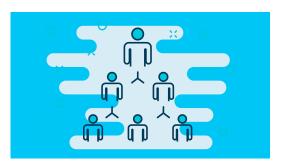
EPFL DecentralizePy Modules

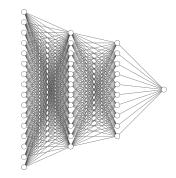






(Flexibility)







EPFL As easy as ABC ...

1 from decentralizepy.node.Node import Node 2 3 class DLNode(Node): def run(self, iterations, training, dataset, 4 sharing, graph, communication): for round in range(iterations): 5 training.train(dataset) 6 msg = sharing.get_message() 7 neighbors = graph.get_neighbors() 8 communication.send(neighbors, msg) 9 rcv = communication.receive_from_all() 10 sharing.average(rcv) 11 dataset.test() 12

 DecentralizePy already contains reference implementations of wellknown algorithms.

EPFL Development Phase: Single machine

1 from decentralizepy.node.Node import Node 1 from decentralizepy.node.Node import Node 3 class DLNode(Node): 3 class DLNode(Node): 4 def run(self, iterations, training, dataset, 4 def run(self, iterations, training, dataset, sharing, graph, communication): sharing, graph, communication): for round in range(iterations): for round in range(iterations): training.train(dataset) training.train(dataset) 6 6 msg = sharing.get_message() 7 msg = sharing.get_message() neighbors = graph.get_neighbors() neighbors = graph.get_neighbors() 8 communication.send(neighbors, msg) 0 communication.send(neighbors, msg) rcv = communication.receive from all() 10 rcv = communication.receive from all() sharing.average(rcv) 11 sharing.average(rcv) 11 dataset.test() 12 dataset.test() 12 1 from decentralizepy.node.Node import Node 1 from decentralizepy.node.Node import Node 2 3 class DLNode(Node): 3 class DLNode(Node): 4 def run(self, iterations, training, dataset, 4 def run(self, iterations, training, dataset, sharing, graph, communication): sharing, graph, communication): for round in range(iterations): for round in range(iterations): 5 6 training.train(dataset) training.train(dataset) msg = sharing.get_message() msg = sharing.get_message() neighbors = graph.get_neighbors() neighbors = graph.get_neighbors() communication.send(neighbors, msg) communication.send(neighbors, msg) rcv = communication.receive_from_all() rcv = communication.receive_from_all() 10 sharing.average(rcv) sharing.average(rcv) 11 11 dataset.test() dataset.test() 12 12

EPFL Real-world deployment



	from decentralizepy.node.Node import Node
	class OLNode(Node):
4	def run(self, iterations, training, dataset,
	sharing, graph, communication):
5	for round in range(iterations):
6	training.train(dataset)
2	<pre>nsg = sharing.get.message()</pre>
	reighbors = graph.get.neighbors()
	comunication.send(neighbors, msg)
1.0	rev = comunication.receive_from_all()
11	sharing, average(rcv)
12	dataset.test()



	om decentralizepy.node.Node import Node
2	
) cl	ass OLNode(Node):
4	def run(self, iterations, training, dataset,
	sharing, graph, communication):
5	for round in range(iterations):
6	training.train(dataset)
2	<pre>nsg = sharing.get.message()</pre>
5	reighbors = graph.get.neighbors()
	communication, send(neighbors, msg)
18	rev = comunication.receive_from_all(
11	sharing, average(rcv)
12	(ataset.test()









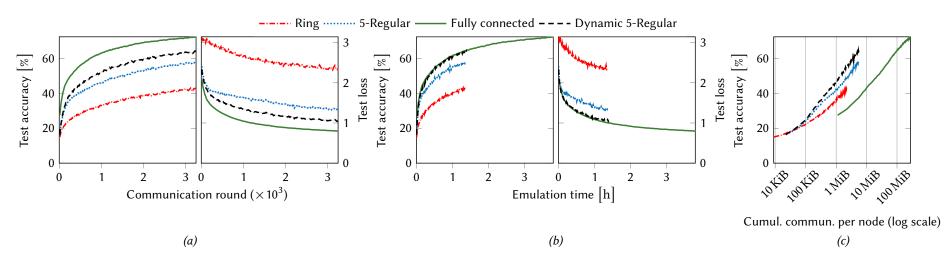
We use DecentralizePy as a **catalyst** for DL research in our lab.

EPFL Experimental Setup

◆ CIFAR-10 (Non-IID) with GN-LeNet

- ◆ 256 and 1024 DL nodes
- ✦ Emulation on 16 machines
- ◆ D-PSGD with Metropolis Hastings

(256-nodes)



Information spreads faster through the network with dynamic topologies.

EPFL Topologies

1 from decentralizepy.node.Node import Node

```
2
3 class DLNode(Node):
      def run(self, iterations, training, dataset,
4
           sharing, graph, communication):
          for round in range(iterations):
5
              training.train(dataset)
6
             msg = sharing.get_message()
7
             neighbors = graph.get_neighbors()
8
              communication.send(neighbors, msg)
9
             rcv = communication.receive_from_all()
10
              sharing.average(rcv)
11
             dataset.test()
12
```

EPFL Communication Compression

(256-nodes)Random sampling 10% —— Choco 10% —— Full sharing A designed and a second second designed and Test accuracy $\left[\%
ight]$ 40 20 0 10 MiB 100 MiB 1 GiB Cumulative communication per node (log scale)

The loss of information due to compression dramatically affects the convergence in non-IID settings at scale.

EPFL Communication Compression

```
1 from decentralizepy.node.Node import Node
2
3 class DLNode(Node):
      def run(self, iterations, training, dataset,
4
           sharing, graph, communication):
          for round in range(iterations):
5
              training.train(dataset)
6
             msg = sharing.get_message()
7
              neighbors = graph.get_neighbors()
8
              communication.send(neighbors, msg)
9
              rcv = communication.receive_from_all()
10
              sharing.average(rcv)
11
              dataset.test()
12
```

EPFL DecentralizePy is rapidly evolving

♦ Open source

- ✦ Already being used for a number of projects
- ✦ Adding new algorithms
- Realistic network emulations
- ✦ Peer-sampling and availability traces



https://github.com/sacs-epfl/decentralizepy

EPFL DecentralizePy is rapidly evolving

sacs-epfl/decentralizepy Public

Code 💿 Issues 🟠 Pull requests 🕞 Actions 🖽 Projects 🛄 Wiki 🕕 Security 🗠 Insights 🕸 Settings

rishi-s8 Add script to generate graph	8ae8221 4 hours a	go 🕲 192 commits
eval	Add dataset download Update tutorial	20 hours ago
src/decentralizepy	Add script to generate graph	4 hours ago
tutorial	Add script to generate graph	4 hours ago
.gitignore	Add dataset download Update tutorial	20 hours ago
isort.cfg	Initial Commit	2 years ago
LICENSE	Add license	3 months ago
README.rst	Add script to generate graph	4 hours ago
download_dataset.py	Add dataset download Update tutorial	20 hours ago
generate_graph.py	Add script to generate graph	4 hours ago
install_nMachines.sh	6 machine, move to eval	2 years ago
) pyproject.toml	Initial Commit	2 years ago
requirements.txt	Initial Commit	2 years ago
) setup.cfg	Add peer sampler, refactor everything	10 months ago
) setup.py	Modify Data and Dataset, add barebone Node, structure config.ini	2 years ago
split_into_files.py	Reddit	last year
README.rst		ı
decentralizepy		EPFL







aspects of distributed learning (communication efficiency, privacy, data heterogeneity etc.).

Python 88.2% Shell 11.8%

Please try DecentralizePy if you are working with DL and help us improve the framework.



Go Decentralized!



https://github.com/sacs-epfl/decentralizepy