Academia Sinica
Research Center for Information Technology Innovation
Research Assistant : Lee, Yi-Ting

Final Presentation

Content

MPU6050 Accelerometer Gesture Recognition

- MPU6050 & DA14580
- Realtime data visualization
- Gesture recognition with basic ML

Federated Learning Concepts

- OpenMined
- Pysyft & Duet

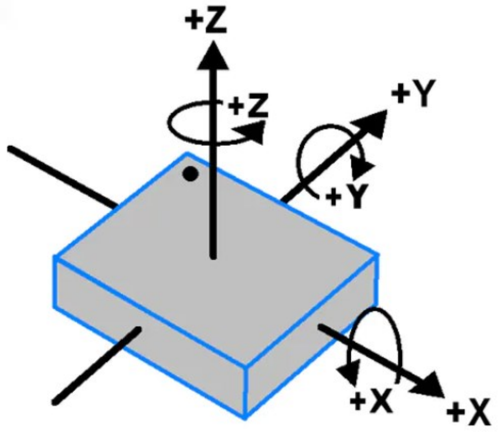
Federated Learning in PyTorch & Pysyft Experiment Results

- FL PyTorch training with Custom Dataset
- PyTorch vs Tensorflow
- IID vs NonIID data
- Differential Privacy (Opacus)

MPU6050 Accelerometer Gesture Recognition

MPU6050

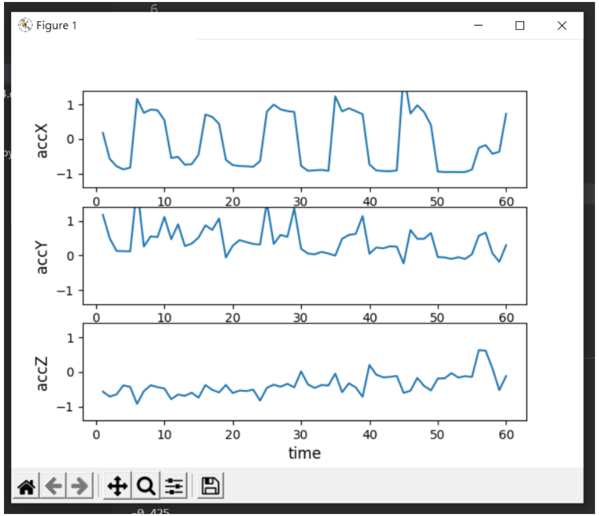
- 6-axis motion tracking device
  - Accelerometer x, y, z axis
  - Gyroscope x, y, z, axis
- designed for the low power, low cost, and high performance requirements of smartphones, tablets and wearable sensors

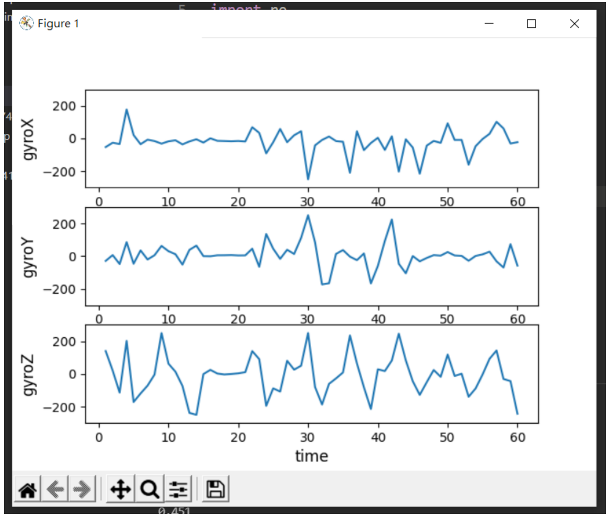


MPU-6050
Orientation & Polarity of Rotation

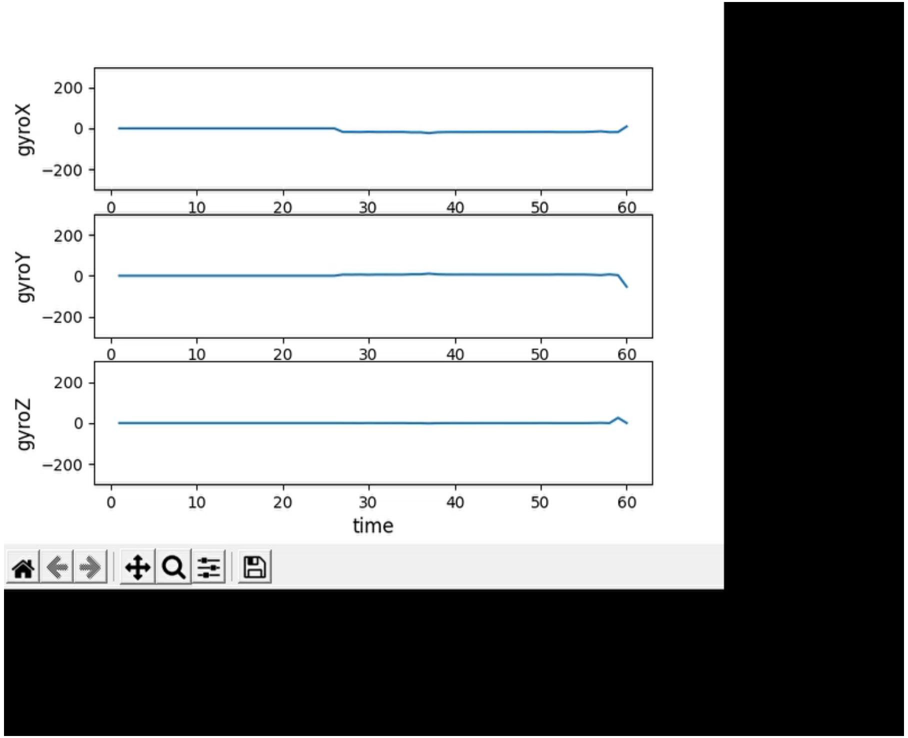
Realtime Data Visualization

- Read serial port with Python
- Plot real time graphs with Matplotlib



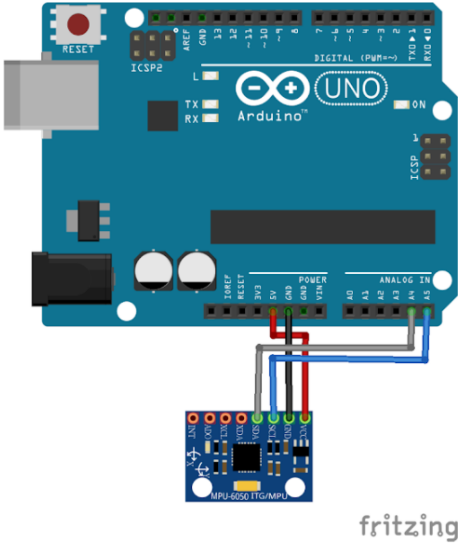


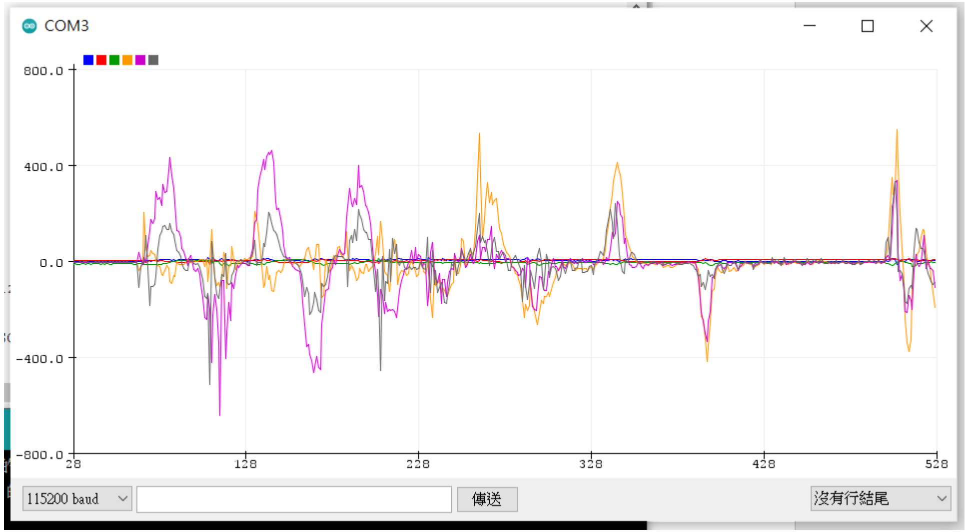
Realtime Data Visualization



Realtime Data Visualization

▷ Realtime visualization can also achieved by Arduino



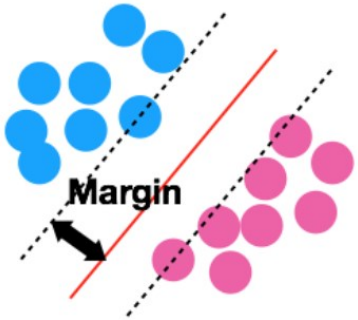


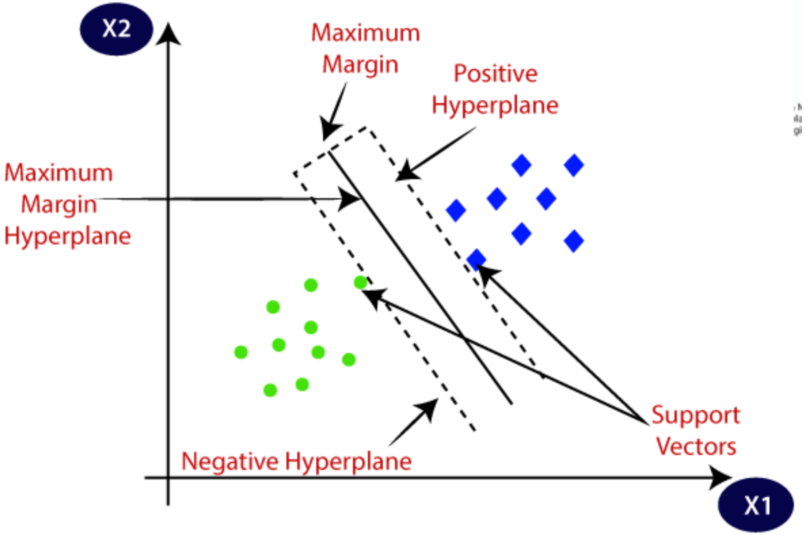
Gesture recognition with basic ML

- Self collected data (LtoR, RtoL)
- 150 samples / data
- Dataset too small -> basic ML method
- Support Vector Machine (SVM)

Gesture recognition with basic ML

- Principle Components Analysis (PCA)
  - widely used technique for dimensionality reduction of the data
  - makes the large data simpler, easy to explore and visualize
- Support Vector Machine (SVM

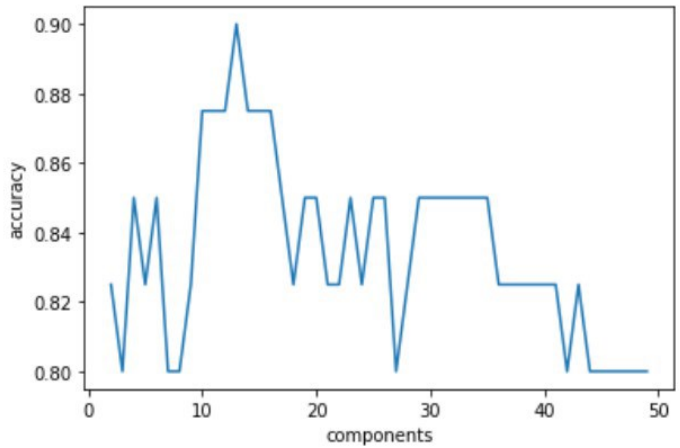




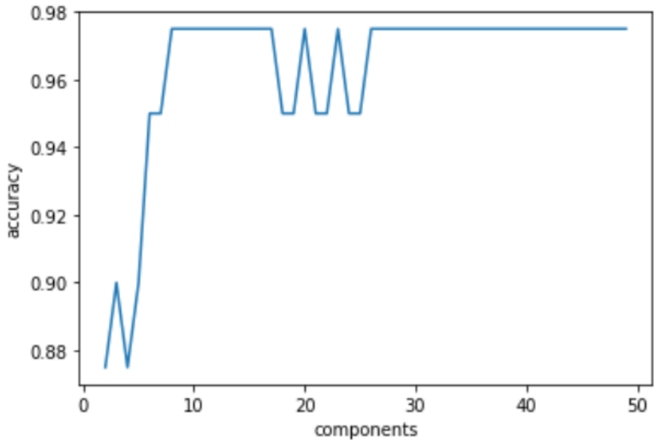
Gesture recognition with basic ML
Training result

▷ Only use accelerometer data achieves highest accuracy

All data (accel x, y, z & gyro x, y, z)



Accel x, y, z



Federated Learning Concepts

Federated Learning Concept

- model training on **central** server
- training data **decentralized**
- privacy-preserving
  - raw data stored locally and without transferred or exchanged

FL training process

- Client selection -> Broadcast -> Client computation -> Aggregation -> Model Update
- **Privacy** and **communication efficiency** are first-order concerns

Federated Learning Concept

FL tools & frameworks

- TensorFlow Federated
- OpenMined (Pysyft)

OpenMined

- Open-source community
- People and organizations can host private datasets, allowing data scientists to train or query on data they **"cannot see"**
- **Data owners retain complete control**: data is never copied, moved, or shared

Goal

- To make the world more privacy-preserving by lowering the barrier-to-entry to private AI technologies.

Pysyft

- a Python library for secure and private Deep Learning
- compute over information you do not own on machines you **do not have total control over**

Duet

- **Peer-to-peer** tool within Pysyft
- Research-friendly API
- Data owner can privately expose their data
- Allows you to get started using Pysyft
- Data between owner and scientists are sent by pointers

Data Owner

```
import syft as sy
```

Part 1: Launch a Duet Server

```
duet = sy.launch_duet(loopback=True)
```

Duet

```
🎤 🎸 ♪♪♪ Starting Duet ♫♫♫ 🎻 🎹

♫♫♫ > DISCLAIMER: Duet is an experimental feature currently in beta.
♫♫♫ > Use at your own risk.

    > ❤️ Love Duet? Please consider supporting our community!
    > https://github.com/sponsors/OpenMined

♫♫♫ > Punching through firewall to OpenGrid Network Node at:
♫♫♫ > http://ec2-18-216-8-163.us-east-2.compute.amazonaws.com:5000
♫♫♫ >
♫♫♫ > ...waiting for response from OpenGrid Network...
♫♫♫ > DONE!

♫♫♫ > STEP 1: Send the following code to your Duet Partner!

import syft as sy
duet = sy.join_duet(loopback=True)

♫♫♫ > Connecting...

♫♫♫ > CONNECTED!
```

Data Scientist

```
import syft as sy
```

Part 1: Join the Duet Server the Data Owner connected to

```
duet = sy.join_duet(loopback=True)
```

Duet

```
🎤 🎸 ♪♪♪ Joining Duet ♫♫♫ 🎻 🎹

♫♫♫ > DISCLAIMER: Duet is an experimental feature currently in beta.
♫♫♫ > Use at your own risk.

    > ❤️ Love Duet? Please consider supporting our community!
    > https://github.com/sponsors/OpenMined

♫♫♫ > Punching through firewall to OpenGrid Network Node at:
♫♫♫ > http://ec2-18-216-8-163.us-east-2.compute.amazonaws.com:5000
♫♫♫ >
♫♫♫ > ...waiting for response from OpenGrid Network...
♫♫♫ > DONE!

♫♫♫ > CONNECTED!
```

Data Owner – send data (pointer)

```
# Finally the data owner UPLOADS THE DATA to the Duet server and makes it searchable
# by data scientists. NOTE: The data is still on the Data Owners machine and cannot be
# viewed or retrieved by any Data Scientists without permission.
age_data_pointer = age_data.send(duet, pointable=True)
```

```
# Once uploaded, the data owner can see the object stored in the tensor
duet.store
```

```
[<syft.proxy.torch.TensorPointer object at 0x00000179E1F12730>]
```

```
# To see it in a human-readable format, data owner can also pretty-print the tensor information
duet.store.pandas
```

	ID	Tags	Description	object_type
0	<UID: e80e293ea16e46fba8ffdd30b03beec9>	[ages]	This is a list of ages of 6 people.	<class 'torch.Tensor'>

Data Scientist – get data (pointer)

```
# The data scientist can check the list of searchable data in Data Owner's duet store
duet.store.pandas
```

	ID	Tags	Description	object_type
0	<UID: e80e293ea16e46fba8ffdd30b03beec9>	[ages]	This is a list of ages of 6 people.	<class 'torch.Tensor'>

```
# Data Scientist likes the age data. (S)He needs a pointer to it.

data_ptr = duet.store[0]
# data_ptr = duet.store['ages']

# data_ptr is a reference to the age dataset remotely available on data owner's server
print(data_ptr)
```

```
<syft.proxy.torch.TensorPointer object at 0x0000022C87E24490>
```

Data Scientist

Perform basic analysis on the data – request from data owner

```
average_age = data_ptr.float().mean()
```

```
try:
    average_age.get()
except Exception as e:
    print(e)
```

```
average_age.request(
    reason="I am a data scientist and I need to know the average age for my analysis."
)
```

```
duet.requests.pandas
```

Data Owner

Response to requests coming from Data Scientist

```
# Oh there's a new request!
duet.requests.pandas
```

	Requested Object's tags	Reason	Request ID	Requested Object's ID	Requested Object's type
0	[ages, float, mean]	I am a data scientist and I need to know the a...	<UID: 3dab5ba34ffb455aab58308886eefb21>	<UID: 260dfb86bb0847d8aba5239ec1171ccb>	

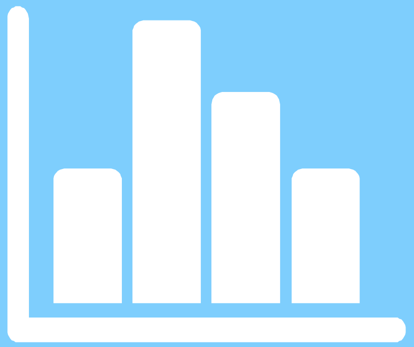
```
# Let's check what it says.
duet.requests[0].request_description
```

```
'I am a data scientist and I need to know the average age for my analysis.'
```

```
# The request looks reasonable. Should be accepted :)
duet.requests[0].accept()
```

Add request handlers

```
# You can automatically accept or deny requests, which is great for testing.
# We have more advanced handlers coming soon.

duet.requests.add_handler(action="accept")
```



Federated Learning in Pytorch & Pysyft

Experiment Results

FL in Pytorch with Custom Dataset

- hook
  A reference to the TorchHook object which is used to **modify PyTorch with PySyft's functionality**
- sy.VirtualWorker() simulate clients in FL

```python
hook = sy.TorchHook(torch)
clients = []

for i in range(args.clients):
    clients.append({'hook': sy.VirtualWorker(hook, id="client{}".format(i+1))})
```

```
client====== {'hook': <VirtualWorker id:client3 #objects:0>, 'trainset': <torch.utils.data.dataloader.Data
Loader object at 0x7f7c0ea24c90>, 'testset': <torch.utils.data.dataloader.DataLoader object at 0x7f7c0e933
050>, 'samples': 0.3333333333333333, 'model': Net(
  (conv1): Conv2d(1, 32, kernel_size=(3, 3), stride=(1, 1))
  (conv2): Conv2d(32, 64, kernel_size=(3, 3), stride=(1, 1))
  (fc1): Linear(in_features=12544, out_features=128, bias=True)
  (fc2): Linear(in_features=128, out_features=3, bias=True)
), 'optim': SGD (
Parameter Group 0
    dampening: 0
    lr: 0.01
    momentum: 0.8
    nesterov: False
    weight_decay: 0
), 'criterion': CrossEntropyLoss()}
```

FL in Pytorch with Custom Dataset

- Original dataset – MNIST in torchvision.datasets

  MNIST

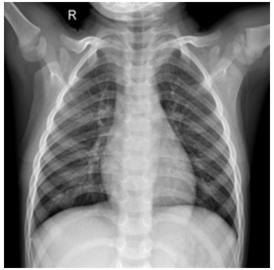
  ```
  CLASS torchvision.datasets.MNIST(root: str, train: bool = True, transform: Union[Callable, NoneType] = None, target_transform: Union[Callable, NoneType] = None, download: bool = False) → None
  ```

- Built-in image transform
- Get data in tensor form that can be trained immediately

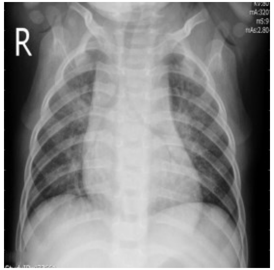
FL in Pytorch with Custom Dataset

- Custom dataset – Covid-19 dataset from Kaggle

COVID



NORMAL



Viral Pneumonia

- Has to follow PyTorch Dataset format
  - contain __init__(), __len__(), __getitem__() function
- Create Rescale() and ToTensor() function
  - adjust size of input image
  - transfer data to torch.Tensor type for the following training process

FL in Pytorch with Custom Dataset

- Create train.csv & test.csv
- Image path & label

jupyter train.csv 2021年5月5日

File Edit View Language

```
/home/citi302/Desktop/Codefolder/FL_DP_covid/FinalCovid19Dataset/train/1/NORMAL (975).png,1
/home/citi302/Desktop/Codefolder/FL_DP_covid/FinalCovid19Dataset/train/1/NORMAL (406).png,1
/home/citi302/Desktop/Codefolder/FL_DP_covid/FinalCovid19Dataset/train/1/NORMAL (1084).png,1
/home/citi302/Desktop/Codefolder/FL_DP_covid/FinalCovid19Dataset/train/1/NORMAL (194).png,1
```

FL in Pytorch with Custom Dataset

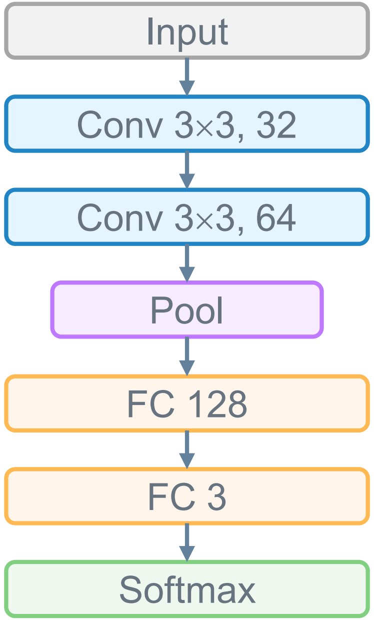
- Set hyperparameters & variables
  - Epochs, clients, rounds
  - Learning rate, batch size
  - client data(IID/NonIID)

- Start training federated learning models with PyTorch! ☺

```
class Arguments():
    def __init__(self):
        self.images = 3012
        self.clients = 10
        self.rounds = 1001
        self.epochs = 1
        self.local_batches = 20
        self.lr = 0.01
        self.dropout1 = 0.25
        self.dropout2 = 0.5
        self.C = 0.66
        self.drop_rate = 0.1
        self.torch_seed = 0
        self.log_interval = 10
        self.iid = 'noniid'
        self.split_size = int(self.images / self.clients)
        self.samples = self.split_size / self.images
        self.use_cuda = True
        self.save_model = False
        self.save_model_interval = 500
        self.clip = 1
        self.del_runs = False
        self.acc_csv = True
        self.acc_file = '0517_10clients_noniid1.csv'
        # number of classes per client on non iid case
        self.noniid_classnum = 1
        # data transform
        self.transform = transforms.Compose([Rescale(32), ToTensor()])
        # number of classes
        self.c_num = 3
```

Model settings

CNN model architecture




```python
class Net(nn.Module):
    def __init__(self):
        super(Net, self).__init__()
        self.conv1 = nn.Conv2d(in_channels = 3,
                               out_channels = 32,
                               kernel_size = 3,
                               stride = 1)
        self.conv2 = nn.Conv2d(in_channels = 32,
                               out_channels = 64,
                               kernel_size = 3,
                               stride = 1)
        self.fc1 = nn.Linear(14*14*64, 128)
        self.fc2 = nn.Linear(128, 3)

    def forward(self, x):
        x = F.relu(self.conv1(x))
        x = F.relu(self.conv2(x))
        x = F.max_pool2d(x, 2, 2)
        x = F.dropout(x, p=args.dropout1)
        x = x.view(-1, 14*14*64)
        x = F.relu(self.fc1(x))
        x = F.dropout(x, p=args.dropout2)
        x = self.fc2(x)
        return F.softmax(x)
```

How to reduce training time?

Original

- __getitem__() reads data from image repeatedly
- Loading image to array data takes much time
- Average training time: 7h 40m

```python
class CovidDataset(Dataset):
    def __init__(self, csv_path, transform=None):
        self.data_info = pd.read_csv(csv_path, header=None)
        self.transform = transform

    def __len__(self):
        return len(self.data_info)

    def __getitem__(self, idx):
        if torch.is_tensor(idx):
            idx = idx.tolist()

        img_name = self.data_info.iloc[idx, 0]
        image = cv2.imread(img_name, cv2.IMREAD_GRAYSCALE)
        label = self.data_info.iloc[idx, 1]
        label = np.array([label])
        sample = {'image': image, 'label': label}

        if self.transform:
            sample = self.transform(sample)

        return sample

    def get_labels(self):
        labels = []
        for i in range(len(self.data_info)):
            labels.append(self.data_info.iloc[i, 1])

        return labels
```

How to reduce training time?

Read from .npy file

- Convert images to array and store as .npy files first
- Data loading from .npy files are much more faster
- Average training time: 20m 10s

7h 40m ➜ 20m 10s

21x reduction

```
for i in range(data_info.shape[0]):
    # read image by cv2
    img = cv2.imread(data_info[0][i])
    # resize image to (32, 32, 3)
    img = cv2.resize(img, (32,32))
    # transpose image dimensions (3, 32, 32) for model training
    img = img.transpose((2, 0, 1))
    np.save('./FinalCovid19Dataset_npy/train/'+ str(data_info[1][i]) + '/' + str(i) + '.npy', img)
```

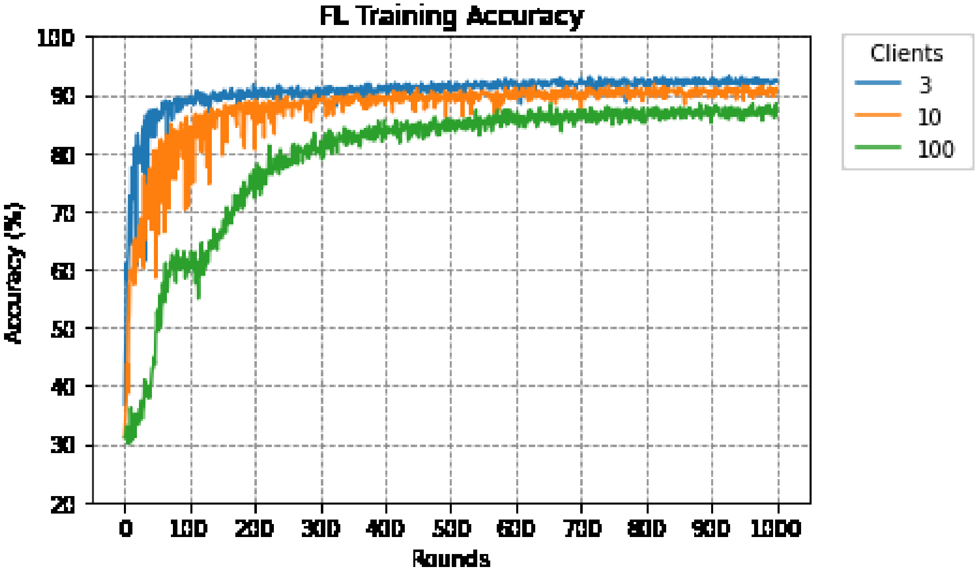
```
def npy_loader(path):
    sample = torch.from_numpy(np.load(path))
    return sample

def load_dataset(num_users, iidtype, transform, c_num, noniid_c = 0):
    data_path = "./FinalCovid19Dataset_npy/train"
    train_dataset = datasets.DatasetFolder(
        root=data_path,
        loader=npy_loader,
        extensions=tuple(['.npy']),
        transform=transform
    )
    print(train_dataset.classes)
    train_group = None
    if iidtype == 'iid':
        train_group = covidIID(train_dataset, num_users)

    elif iidtype == 'noniid':
        train_group = covidNonIID(train_dataset, num_users, c_num, noniid_c)

    else:
        train_group = covidNonIIDUnequal(train_dataset, num_users)

    return train_dataset, train_group
```

Experiment results

Training Time
Pytorch vs Tensorflow

- Running time (ran on the same computer):
  PyTorch (17m59s) ≈ Tensorflow (18m 5s)
- Differences between two approaches
  - API function calculation
  - Image processing difference (e.g. TF initially store images as array, PyTorch load .npy file)

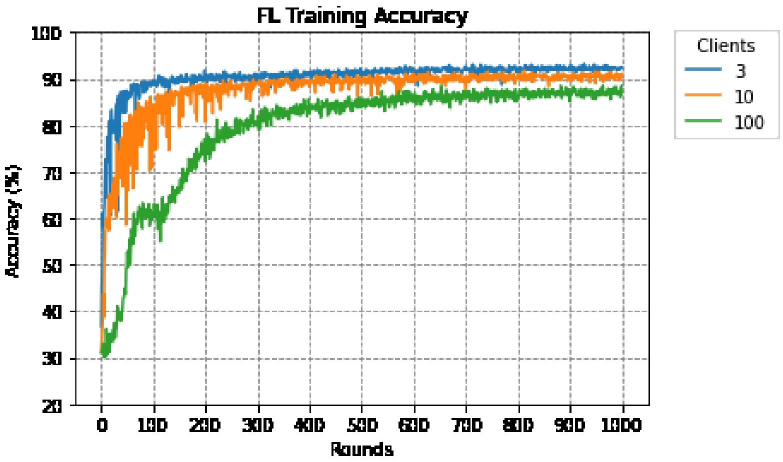
Number of clients

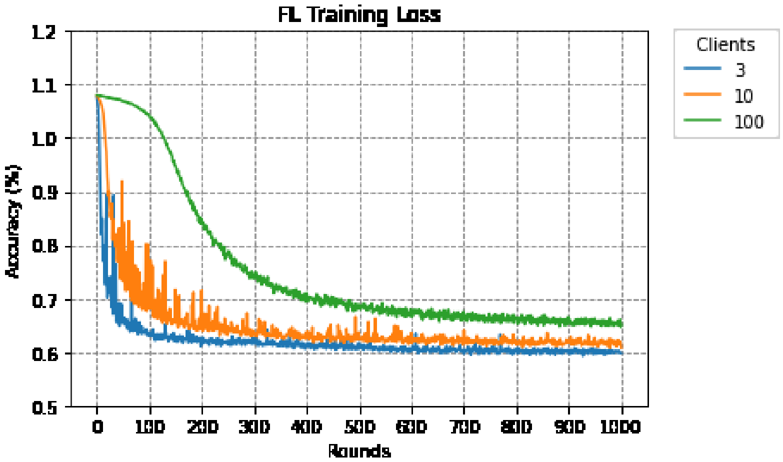


Properties

– IID Data
➜ Client's data **distributed evenly**
– Batch size = 20
– Local epoch =1
➜ every client trained once/round
– Client fraction = 0.66
➜ fraction of total clients that will join the training per round

Rounds	3 clients	10 clients	100 clients
	64.41%		
100	88.71%	79.15%	61.09%
	90.83%		
1000	92.43%	91.10%	87.12%



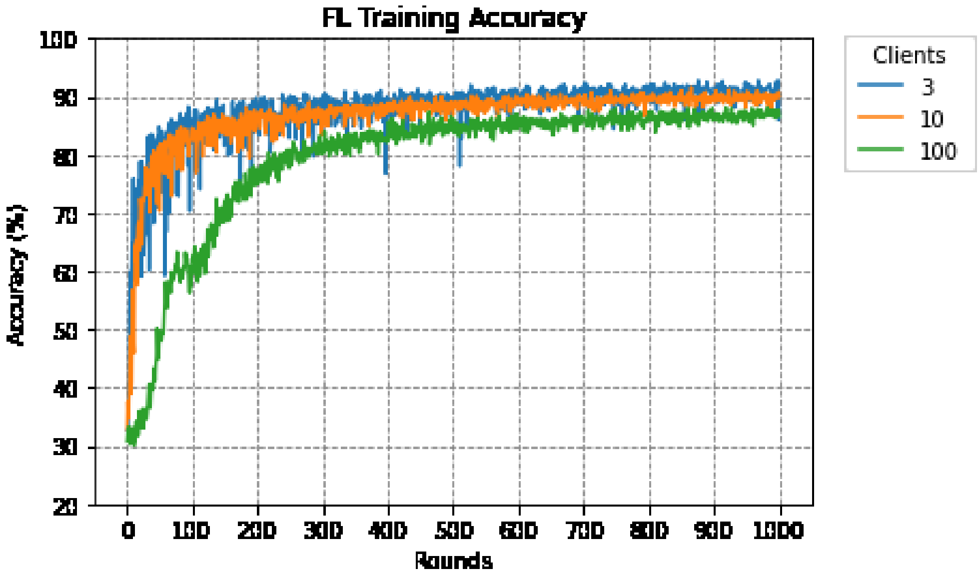


Result

− the more the clients, the slower the model achieve high accuracy
− the more the clients, the slower the loss drops
− the more the clients, the lower the accuracy (5% difference)

Rounds	3 clients	10 clients	100 clients
	64.41%		
100	88.71%	79.15%	61.09%
	90.83%		
1000	92.43%	91.10%	87.12%

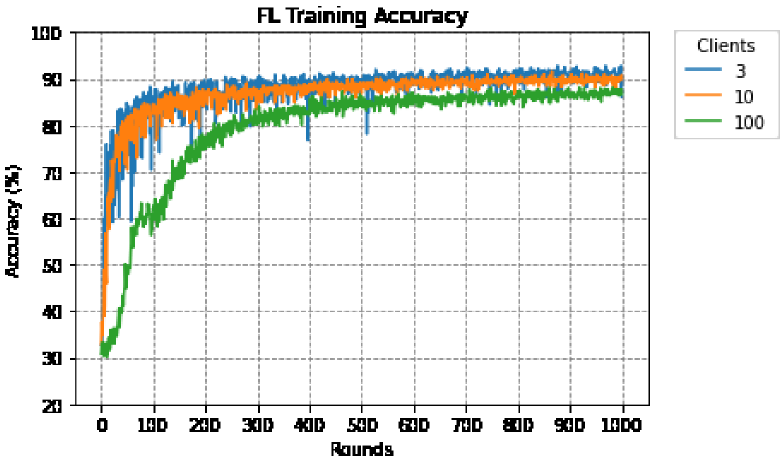
Non IID data (2)

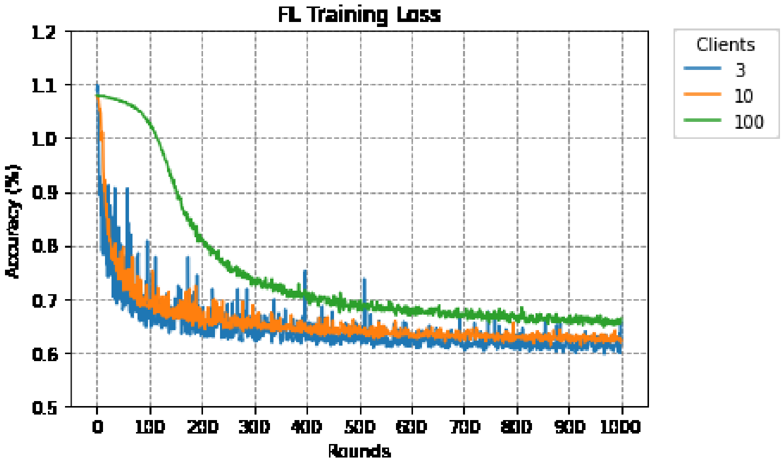


Properties

- Non IID Data
  ➜ Client data contains **two classes**
- Batch size = 20
- Local epoch =1
- Client fraction = 0.66

Rounds	3 clients	10 clients	100 clients
	75.96%		
100	83.93%	84.59%	59.90%
	87.25%		
1000	91.10%	90.04%	86.59%



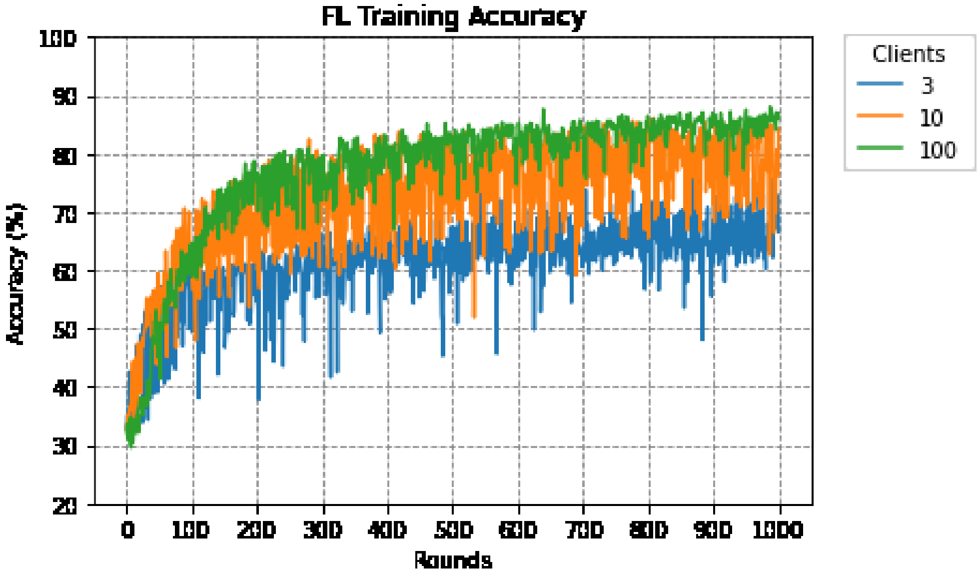


Result

– Fewer clients, training process **more unstable** at the early stage
– the more the clients, the **slower the loss drops**
– the more the clients, the lower the accuracy (5% difference)

Rounds	3 clients	10 clients	100 clients
	75.96%		
100	83.93%	84.59%	59.90%
	87.25%		
1000	91.10%	90.04%	86.59%

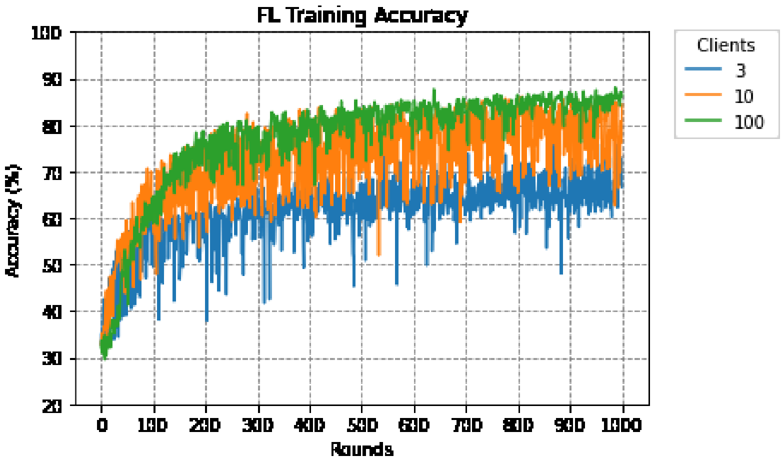
Non IID data (1)

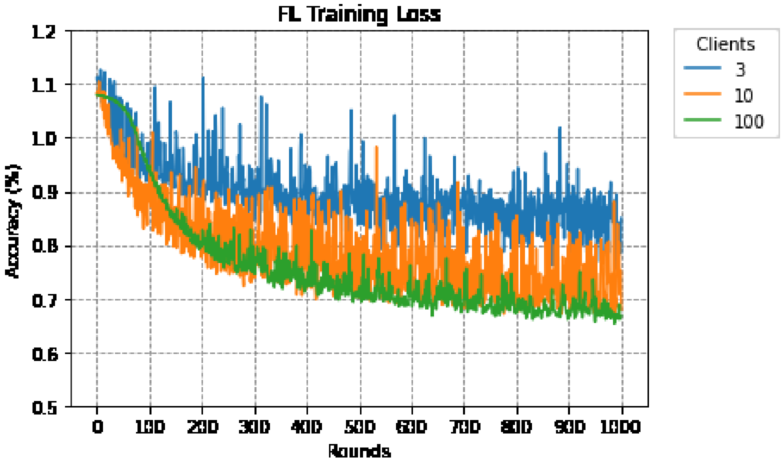


Properties

- Non IID Data
  ➜ Client data contains only **one class**
- Batch size = 20
- Local epoch =1
- Client fraction = 0.66

Rounds	3 clients	10 clients	**100 clients**
			30.54%
100	49.40%	58.43%	58.83%
			80.21%
1000	66.53%	84.33%	86.32%





Result

– Fewer clients, training process **more unstable**

– the **fewer** the clients, the lower the accuracy (20% difference)

➜ fewer clients, training data **more unbalanced** (eg. 3 clients, frac = 0.66, each round only two classes of data will be trained)

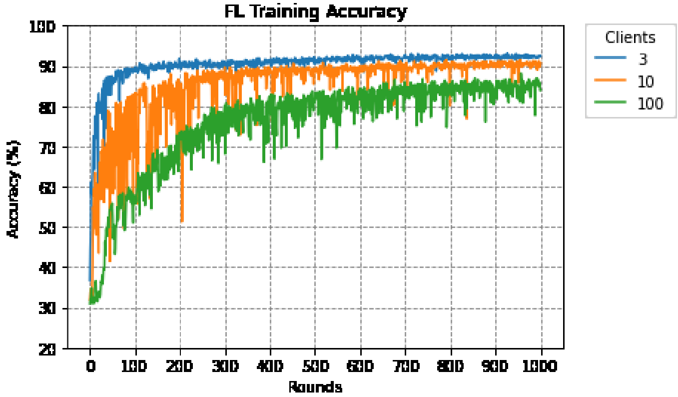
Rounds	3 clients	10 clients	100 clients
			30.54%
100	49.40%	58.43%	58.83%
			80.21%
1000	66.53%	84.33%	86.32%

Number of clients
Pytorch vs Tensorflow

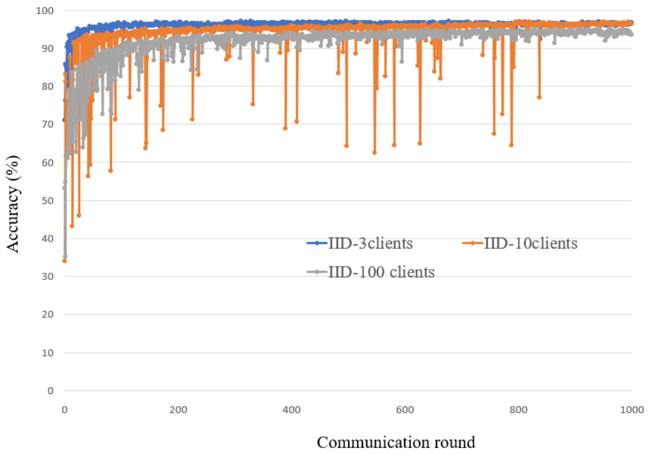
	PyTorch			Tensorflow		
	3 clients			3 clients		
400	**90.84%**	88.58%	79.81%	**96.28%**	95.33%	93.43%
	91.10%			**96.54%**		
800	**92.03%**	90.04%	84.33%	**96.81%**	96.80%	94.12%
	92.43%			**96.81%**		

- **2 clients** per round
- PyTorch < Tensorflow accuracy
  - 4-9% difference between two approaches

PyTorch



Tensorflow

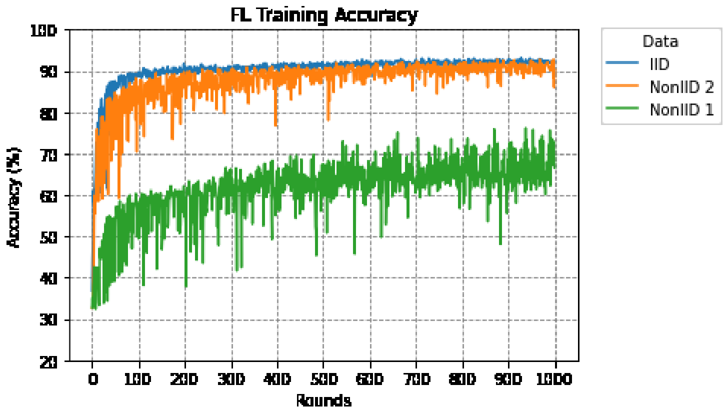


Non IID data (3 clients)
PyTorch vs Tensorflow

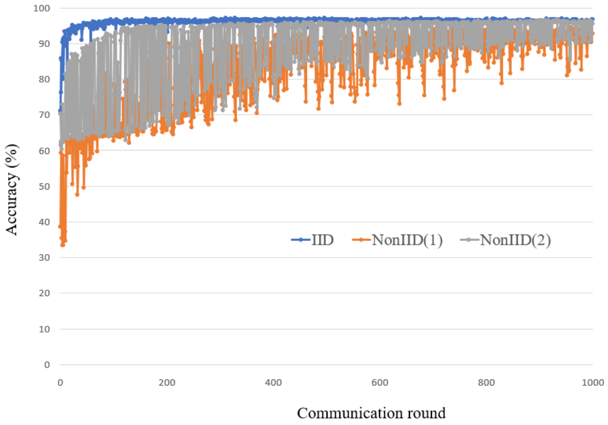
Round	IID		Non IID 2		Non IID 1	
	Torch	TF	Torch	TF	Torch	TF
400	90.84%	96.28%	87.25%	80.80%	65.87%	78.75%
800	92.03%	96.81%	91.10%	96.00%	69.06%	91.90%

- PyTorch < Tensorflow accuracy (4-25% difference)
  - Way of distributing non IID data may vary

PyTorch



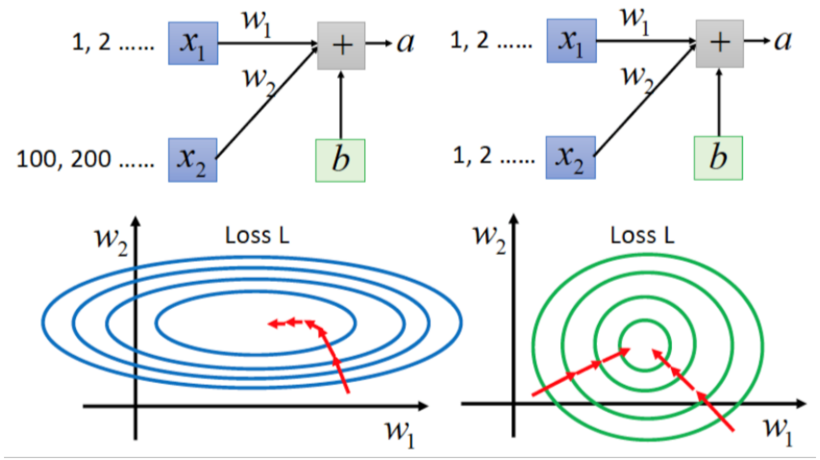
Tensorflow

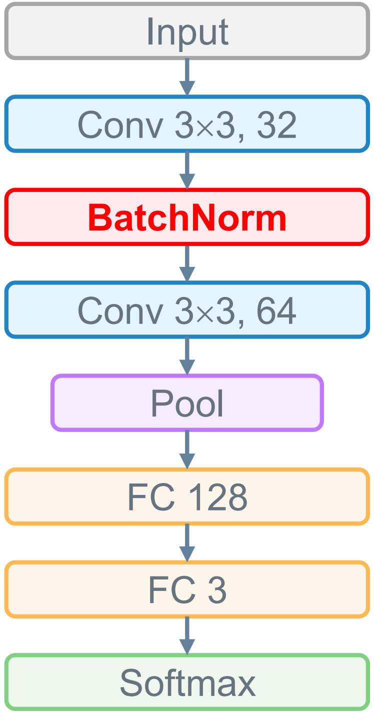


How to improve accuracy?

Batch normalization

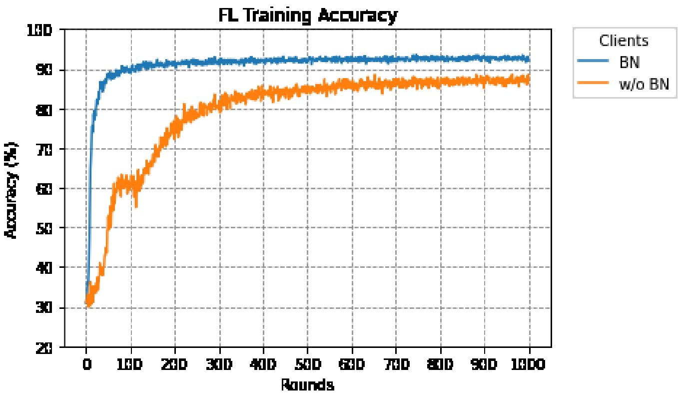
- **Stabilize model training →** normalize the layer inputs by re-centering and re-scaling
- **Feature scaling →**Make different features have the same scaling





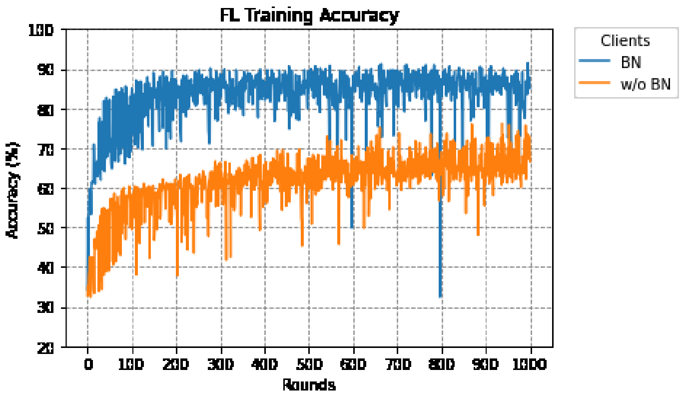
Batch Normalization

IID Data 100 clients



Rnd	3 clients	100 clients	
			BN
10	64.41%	36.25%	**64.94%**
			88.85%
400	90.83%	83.40%	**91.77%**
			92.83%

NonIID Data 1 3 clients



Rnd	100 clients	3 clients	
			BN
10	30.54%	42.36%	**61.49%**
			85.13%
400	80.21%	65.87%	**84.99%**
			85.13%

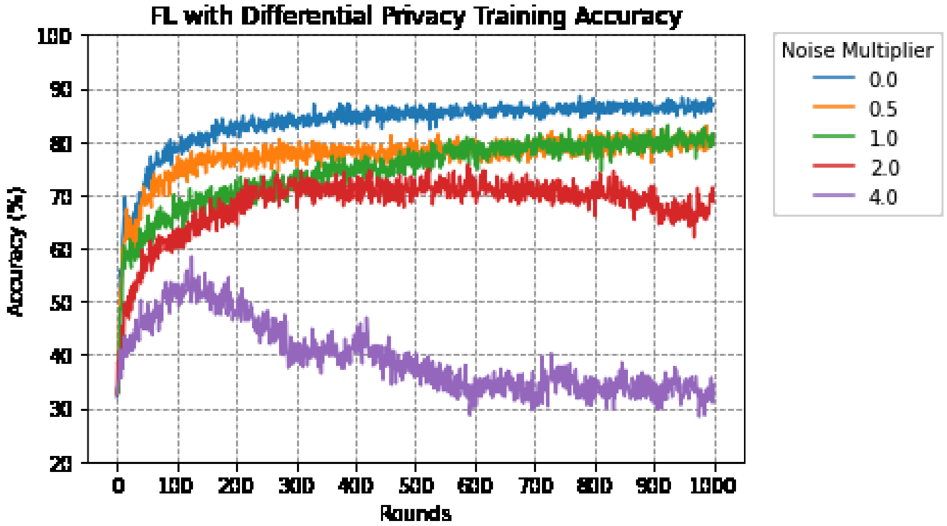
Differential Privacy

Opacus

- Opacus PrivacyEngine()
- Change noise multiplier to add noise

```
for client in clients:
    torch.manual_seed(args.torch_seed)
    client['model'] = Net().to(device)
    client['optim'] = optim.SGD(client['model'].parameters(), lr=args.lr)
    client['criterion'] = nn.CrossEntropyLoss(reduction='mean')
    client['pengine'] = PrivacyEngine(
                            client['model'],
                            batch_size=args.local_batches,
                            sample_size=len(client['trainset']),
                            sample_rate=args.C,
                            alphas=[2,3,4,5,6,7,8,10,11,12,14,16,20,24,28,32,64,256],
                            noise_multiplier=0.5,
                            max_grad_norm=1.0
                        )
    client['pengine'].attach(client['optim'])
```

Differential Privacy

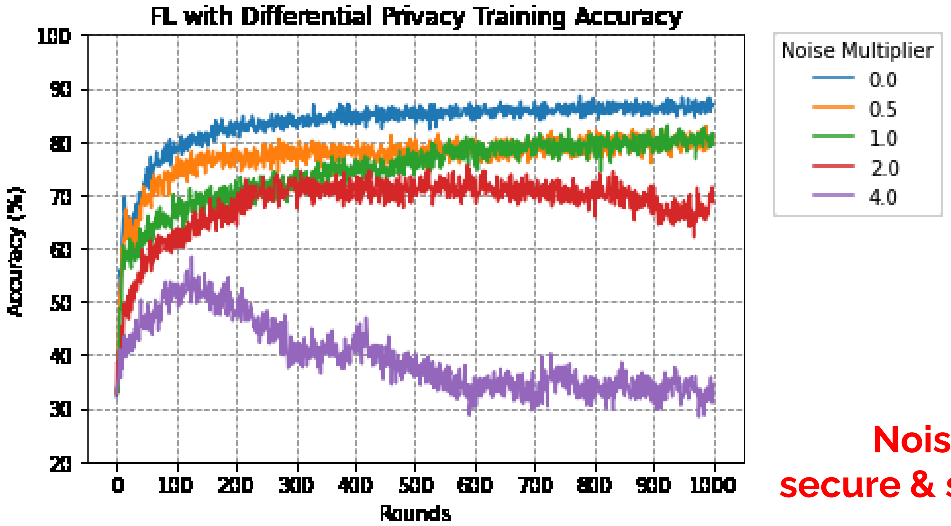


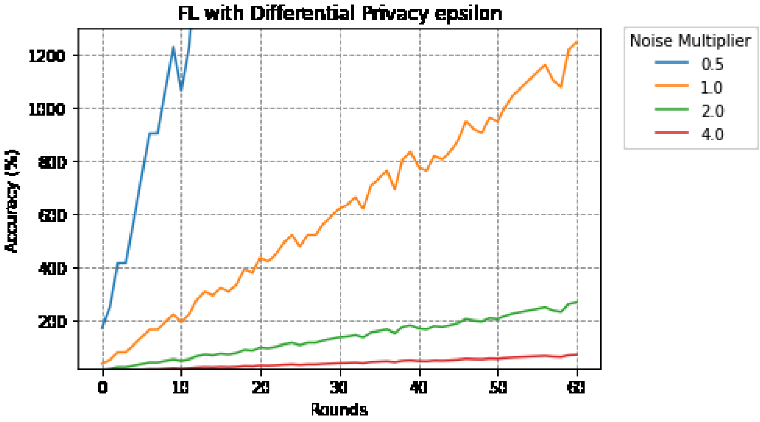
Properties

- IID Data
- Batch size = 20
- Number of clients = 3
- Client fraction = 0.66
- Max gradient norm: 1.0

Noise	Accuracy	Epsilon (60rnds)
0.5	80.34%	7091.6
1.0	80.61%	1248.8
2.0	68.66%	268.0

Differential Privacy





Noise	Accuracy	Epsilon (60rnds)
0.5	80.34%	7091.6
1.0	80.61%	1248.8
2.0	68.66%	268.0

Noise = 1.0
secure & stable

Secure ↑ Accuracy ↓

Overall summary

PyTorch vs Tensorflow

- Tensorflow

  → powerful functions with lots of parameters to customize

  → hard to understand how the function has calculated

- PyTorch

  → good way to learn the FL concept by implementing step by step

  → takes time to develop

Thanks!

Any questions?