How Does Google Train GBoard?

Introduction to Federated Learning

Google Keyboard

Next Word Prediction

Out-of-Vocabulary Word Discovery

Emoji Suggestions

Prerequisites

Basic Understanding of ML

Intermediate grasp over basic ML concepts and terminology

Desktop with Python IDE & A Python Environment

To extract the most out of this tutorial, it is recommended that you follow the coding exercises

Curiosity

Stay hungry, stay foolish - Steve Jobs

About Me

3rd Year Computer Science Undergrad

Interned at Acronis Researching Federated Learning

Interested in All Things AI/ML!

01

What is Traditional Machine Learning?

- All the data is stored in one storage unit or a cluster of storage units in close proximity
- All the data is accessible to the parties in control of the storage cluster
- Each machine learning (ML) model is trained with a full view of the entire dataset

- Data doesn't remain private
- All data must be aggregated in one place

Drawbacks?

- Healthcare
- Fintech
- Insurance
- Personal texts & images

02

The Solution? Collaborative Learning

also known as Federated Learning

Federated learning is decentralised machine learning wherein an algorithm is **trained across multiple edge devices holding local data samples, without exchanging them**

This means that that each edge device trains a model locally on its local samples

All these models are then aggregated again and again over several rounds of local training to then create a combined model

As a result, a robust ML model can be trained **without any party ever having to reveal its data**

Types of Federated Learning

Horizontal Federated Learning

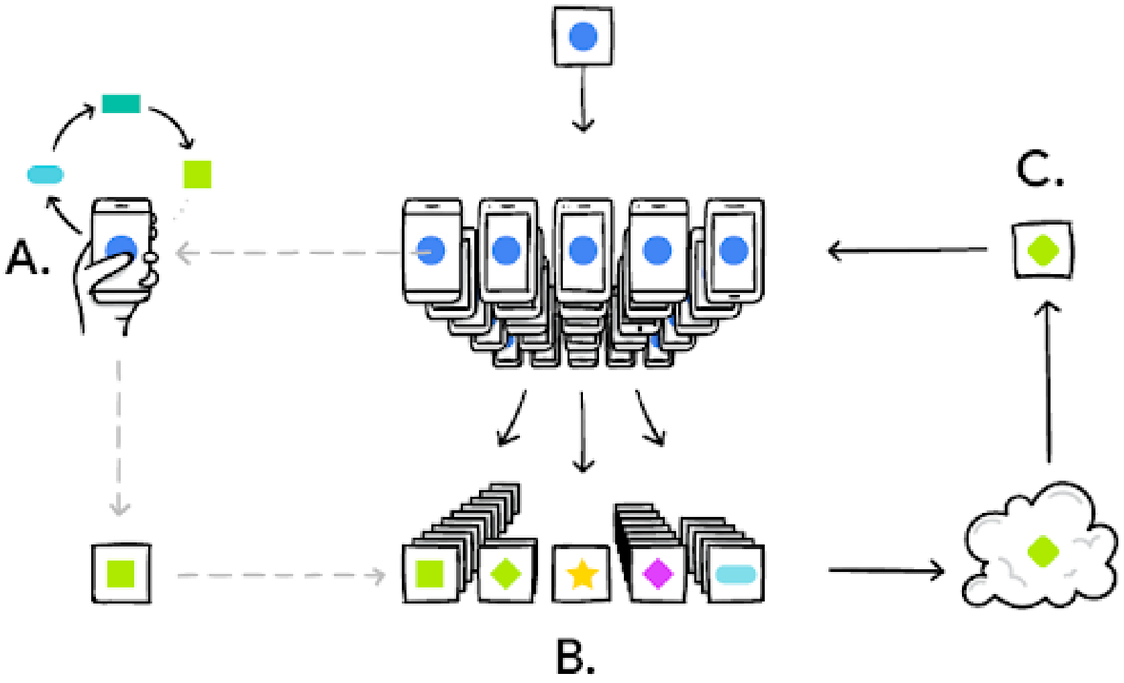
Datasets held locally by clients have the same features but different samples

Vertical Federated Learning

Datasets held locally by clients have different features but the same samples

Federated Transfer Learning

Vertical FL utilised with a pre-trained model that is trained on a similar dataset for solving a different problem



Source: Google

How Exactly Does It Work?

A: Your phone builds a model personal to you

B: Everyone's phone also does the same for them

C: All these personalised models are aggregated to form a **consensus change** to the shared model

Back to A: Rinse & Repeat

What Happens On The Edge Device? (Client-side)

Receive Global Model

Each edge device receives the global model (randomly initialised) from the server

Update Model (Local Update)

The global model is updated by the edge device by performing Stochastic Gradient Descent (SGD) using its local samples

Return Updated Model

Finally, the edge device sends the updated model back to the server

Note: There might be several rounds of local updates (instead of just one) before the updated model is returned (**Why?**)

What Happens At The Server? (Server-side)

Initialise Global Model

The server randomly initialises a global model

Send Global Model

The server then picks a random subset of available clients to send the global model to

Aggregate Models (Global Update)

Finally, the server aggregates all the updated models sent back by the clients to update the global model

Note: The several different strategies used for aggregating the local updates (updated models) heavily influence the performance of the final model

Chronology

Initialise Global Model → Send Global Model → Receive Global Model

Update Model (Local Update) → Return Updated Model → Aggregate Models (Global Update)

This cycle repeats for a predefined number of rounds

03

Federated Strategies

How **exactly** does the server combine the updated models sent back by the clients?

intuitively, we can think of the local updates as one of two things

Updated Model

The client returns an **updated version of the model**
Therefore, the server receives a bunch of new (updated) models

Update to Model

The client returns how much the model should be updated by to reduce loss
Therefore, the server receives a bunch of values each indicating how **much** the model should change by

For this tutorial, we use the **2nd** definition. That is, each client calculates (and returns) the **average gradient on its local data at the current model**
Second definition is easier to implement. **Why?**

Terminology

w_t model weight at the beginning of round **t**

n_k number of local samples at client **k**

n total number of samples

η fixed learning rate (1 by default)

Federated Average

Client-side

$$g_k = \nabla F_k(w_t)$$

average gradient on its local data with the current model

Server-side

$$\nabla f(w_t) = \sum_{k=1}^{K} n_k/n * g_k$$

weighted average of all local updates (local average gradients)

$$w_{t+1} = w_t - \eta \nabla f(w_t)$$

updated global model

Math Exercise

- In a federation of 3 clients, given an array of local gradient updates [13, 14.7, 15.92], an array of local sample sizes [250, 220, 175], and an initial model weight of 80.05, calculate the **updated model weight** after one round of federated learning. Assume all clients are chosen in every round and eta = 1 (in practice, a very small subset of clients are chosen in any given round).

- What if you **lower the value of eta**?

Federated Average with Momentum

What is Momentum?

In FedAvg, the update weight of the model in **round t+1** depended solely on the gradients calculated in **round t**

However, with momentum, the updated weight of the model in **round t+1** depends not only on the gradients calculated in **round t** but also **round t-1**, **round t-2**, ..., **round 1**
The older the round, the lesser its impact; for example, by round 100, the impact of the gradients calculated in round 10 will be negligible, but still existent.

Federated Average with Momentum

Client-side

$$g_k = \nabla F_k(w_t)$$ ⟶ same as FedAvg

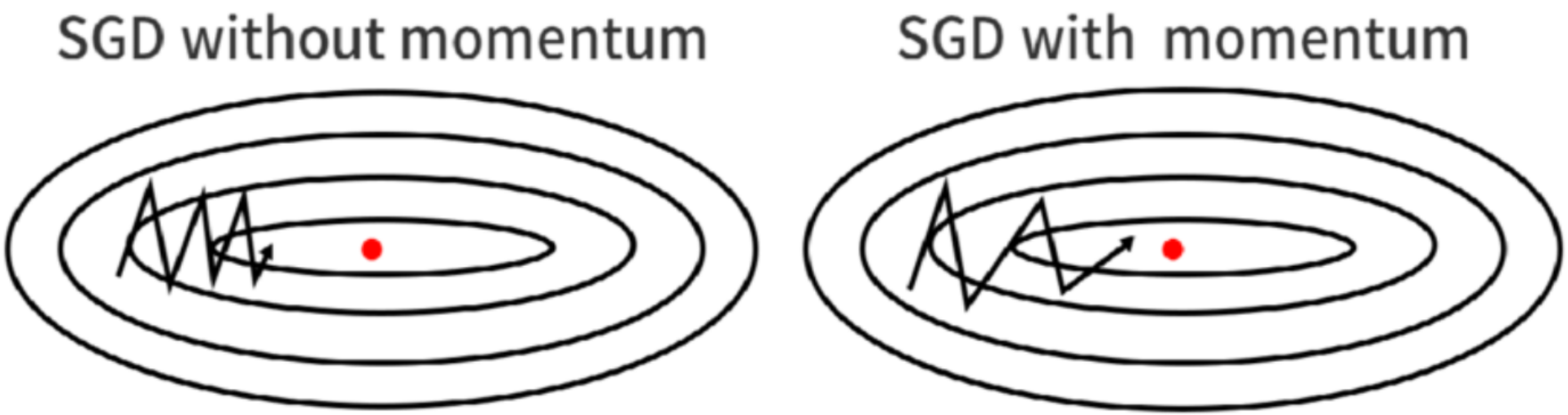
Server-side

$$\nabla f(w_t) = \sum_{k=1}^{K} n_k/n * g_k$$ ⟶ same as FedAvg

$$v_{t+1} = \beta v_t + \eta \sum_{k=1}^{K} (n_k/n) * g_k$$

$$w_{t+1} = w_t - v_{t+1}$$

weighted average of all local weights discounted by a discounted factor of all previous weighted averages



Source: Medium

Thinking Exercise

- Which strategy (out of FedAvg & FedAvgM) do you think fluctuates more? Why?

- What if you **lower the value of beta**?

Other Strategies

FedAdam

FedYogi

FedAdagrad

QFedAvg

FedDist

FedMa

FedPer

FaultTolerant FedAvg

Caveats

Communication

The biggest bottleneck is FL is the communication between the client and the server

This is why there are often multiple rounds of SGD before local updates are sent back to the server—to minimise the back-and-forth and converge faster

Applicability

- Focused on non-convex neural network objectives
- However, is applicable to any finite-sum objective of the form $\min_{w \in R^d} f(w)$ where $f(w) = (1/n) * \sum_{i=1}^{n} f_i(w)$

Speedup

- Increase parallelism by adding more clients
- Has an upper limit

FL in Action

Let's now code a simple federated learning algorithm for a logistic regression model on a simple tabular dataset

We will compare the performance of traditional machine learning with both FedAvg and FedAvgM

04

Federated Learning Frameworks

What's Popular?

- Tensorflow Federated
- **Flower**
- Paddle Federated Learning
- PySyft
- OpenFL
- LEAF
- and many more

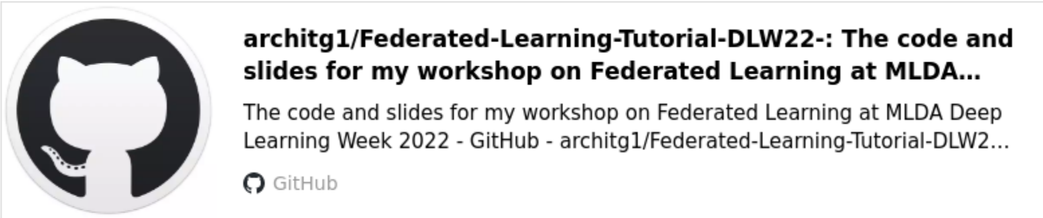
Why Flower?

- Compatible with most ML frameworks
- Allows deployment on edge devices such as phones
- Platform independent
- Easy to use

What Strategies Does it Support

- 7 most popular strategies
- Allows building and testing novel strategies

Code



https://colab.research.google.com/gist/architg1/ecea3a3e21c0d6a97cae939e9c39da51/dlw-fl.ipynb

https://drive.google.com/drive/folders/13vLnzXGHDIZFngg8pLskA5u494fqHz1L?usp=sharing

Installation

Python Env
conda install python=3.9.13

Pandas
pip install pandas

Terminal (Mac)
pip uninstall grpcio
conda install grpcio

Flower
pip install flwr

Sklearn
pip install -U scikit-learn

wsl –install in an administrator command prompt (Windows)

Traditional ML

In [28]:
```
from sklearn.linear_model import LogisticRegression
```

In [29]:
```
model = LogisticRegression(
    penalty="l2",
    max_iter=1,  # local epoch
    warm_start=True,  # prevent refreshing weights when fitting
)
```

In [30]:
```
model.fit(x_train, y_train)
```

Client-side

```python
    model = LogisticRegression(
        penalty="l2",
        max_iter=1,  # local epoch
        warm_start=True,  # prevent refreshing weights when fitting
    )

    # Setting initial parameters
    utils.set_initial_params(model)


    class Client(fl.client.NumPyClient):
        def get_parameters(self, config):  # type: ignore
            return utils.get_model_parameters(model)

        def fit(self, parameters, config):  # type: ignore
            utils.set_model_params(model, parameters)
            with warnings.catch_warnings():
                warnings.simplefilter("ignore")
                model.fit(x_train, y_train)
            print(f"Training finished for round {config['server_round']}")
            return utils.get_model_parameters(model), len(x_train), {}

        def evaluate(self, parameters, config):  # type: ignore
            utils.set_model_params(model, parameters)
            loss = log_loss(y_test, model.predict_proba(x_test))
            accuracy = model.score(x_test, y_test)
            return loss, len(x_test), {"accuracy": accuracy}


    fl.client.start_numpy_client(server_address="0.0.0.0:8080", client=Client())
```

Server-side

```python
    # The `evaluate` function will be called after every round
    def evaluate(server_round, parameters: fl.common.NDArrays, config):
        # Update model with the latest parameters
        utils.set_model_params(model, parameters)
        loss = log_loss(y_test, model.predict_proba(x_test))
        accuracy = model.score(x_test, y_test)
        return loss, {"accuracy": accuracy}

    return evaluate


# Start Flower server for three rounds of federated learning
if __name__ == "__main__":
    model = LogisticRegression()
    utils.set_initial_params(model)

    # implementing 7 strategies
    fed_avg_strategy = fl.server.strategy.FedAvg(
        min_available_clients=2,
        evaluate_fn=get_evaluate_fn(model),
        on_fit_config_fn=fit_round,
    )

    fed_avg_m_strategy = fl.server.strategy.FedAvgM(
        min_available_clients=2,
        evaluate_fn=get_evaluate_fn(model),
        on_fit_config_fn=fit_round,
        server_learning_rate=1,  # AssertionError: When using server-side optimization, model needs to be initialized.
        server_momentum=0.00     # AssertionError: When using server-side optimization, model needs to be initialized.
    )

    fl.server.start_server(server_address="0.0.0.0:8080",
                           strategy=fed_avg_strategy,
                           config=fl.server.ServerConfig(num_rounds=3)
    )
```

What Can Go Wrong?

Hypothesis Conflicts

There is a hypothesis conflict between two classification models if there exists an input **x** such that the models output different classifications

Different Local Loss Surfaces

Each local model might reach to some local minima on their own surface, however, aggregated model might not be close to any local minima on the loss surface defined by the union of local datasets

How Do You Fix This?

Server-side Training

A small, balanced **training dataset on server-side** to fine-tune the aggregated results

Run SGD with Common Parameters

Since each local model is initialized to the same parameters, and each agent uses the same projection, we believe this might **prevent local models from diverging too much from each other**

05

Threats to Federated Learning

What Next?

Read the papers behind FedAvg, FedAvgM, and FedOpt

Implement a federated strategy for KNN or Random Forest using Flower

Implement Federated Learning from scratch in Python

References & Resources

https://towardsdatascience.com/introduction-to-federated-learning-and-challenges-ea7e02f260ca
https://ai.googleblog.com/2017/04/federated-learning-collaborative.html
https://medium.com/bitgrit-data-science-publication/federated-learning-decentralized-ml-8709acfa9fa2
https://flower.dev/docs/quickstart-scikitlearn.html
https://flower.dev/docs/apiref-flwr.html#api-reference-flwr
https://arxiv.org/abs/2010.15582
https://arxiv.org/abs/2201.08135
https://towardsdatascience.com/ai-differential-privacy-and-federated-learning-523146d46b85
https://ai.googleblog.com/2022/02/federated-learning-with-formal.html
https://github.com/architg1/Federated-Learning-Tutorial-DLW22-