Federated Learning _ Flower (Simulation with TensorFlow/Keras)

در این گزارش، ما یک سیستم یادگیری فدرال را با ۱۰۰ مشتری شبیه سازی می کنیم. مشتریان از TensorFlow/Kera برای تعریف آموزش و ارزیابی مدل استفاده خواهند کرد. این گزارش شامل دو بخش است:

۱. بخش اول : شبیه سازی Federated Learning با استفاده از داده های MNIST و یک مدل شبکه MLP
۲. بخش دوم : شبیه سازی Federated Learning با استفاده از داده های CIFAR10 و یک مدل شبکه Pretrain

بخش اول : داده های MNIST و یک مدل شبکه MLP

بیایید با نصب Flower (منتشر شده به عنوان flwr در PyPI) با اضافه کردن شبیه سازی، شروع کنیم:

```
!pip install -q flwr["simulation"] tensorflow
!pip install -q flwr_datasets["vision"]
```

اجازه دهید Matplotlib را نیز نصب کنیم تا بتوانیم پس از تکمیل شبیه‌سازی، چند نمودار بسازیم

```
!pip install matplotlib
```

بعد، وابستگی های مورد نیاز را وارد می کنیم. مهم ترین واردات گل (flwr) و TensorFlow است:

```
from typing import Dict, List, Tuple
import tensorflow as tf
import flwr as fl
from flwr.common import Metrics
from flwr.simulation.ray_transport.utils import enable_tf_gpu_growth
from datasets import Dataset
from flwr_datasets import FederatedDataset


VERBOSE = ۰
NUM_CLIENTS = ۱۰۰
```

بیایید با تعریف مدلی که می خواهیم فدرال کنیم شروع کنیم. از آنجایی که ما با MNIST کار خواهیم کرد، استفاده از یک مدل MLP. البته می توانید این مدل را سفارشی کنید.

```
def get_model():
    """Constructs a simple model architecture suitable for MNIST."""
    model = tf.keras.models.Sequential(
        [
            tf.keras.layers.Flatten(input_shape=(۲۸, ۲۸)),
            tf.keras.layers.Dense(۱۲۸, activation="relu"),
            tf.keras.layers.Dropout(۰,۲),
            tf.keras.layers.Dense(۱۰, activation="softmax"),
        ]
    )
    model.compile("adam", "sparse_categorical_crossentropy", metrics=["accuracy"])
    return model
```

با وجود این موضوع، بیایید به قسمت های جالب توجه کنیم. سیستم های یادگیری فدرال شامل یک سرور و چندین مشتری است. در Flower، ما با پیاده‌سازی زیر کلاس‌های flwr.client.Client یا flwr.client.NumPyClient کلاینت‌ها را ایجاد می‌کنیم. ما در این آموزش از NumPyClient استفاده می‌کنیم زیرا پیاده‌سازی آن آسان‌تر است و نیاز به نوشتن کد های کمتری دارد.

برای پیاده‌سازی کلاینت در Flower، یک زیر کلاس از flwr.client.NumPyClient ایجاد می‌کنیم و سه تابع get_parameters، fit و evaluate را پیاده‌سازی می‌کنیم:

- get_parameters: این تابع پارامترهای مدل محلی فعلی را برمی گرداند
- fit: این تابع پارامترهای مدل را از سرور دریافت می کند و پارامترهای مدل را بر روی داده های محلی آموزش می دهد و در نهایت پارامترهای مدل (به روز شده) را به سرور برگرمی گرداند
- evaluate: این تابع پارامترهای مدل از سروردریافت می کند و بر روی داده های محلی ارزیابی انجام میدهد و نتیجه ارزیابی به سرور بر میگرداند.

اشاره کردیم که مشتریان ما از TensorFlow/Keras برای آموزش و ارزیابی مدل استفاده خواهند کرد. مدل‌های Keras روش‌هایی را ارائه می‌کنند که پیاده‌سازی را ساده می‌کنند: ما می‌توانیم مدل محلی را با پارامترهای سرور که از طریق model.set_weights گرفته شده ، به‌روزرسانی کنیم، می‌توانیم مدل را از طریق fit/evaluate آموزش-ارزیابی کنیم، و می‌توانیم پارامترهای مدل به‌روزرسانی شده را از طریق model.get_weights دریافت کنیم.

بیایید یک پیاده سازی ساده از یک کلاینت را ببینیم:

```
class FlowerClient(fl.client.NumPyClient):
    def __init__(self, trainset, valset) -> None:
```

```
        # Create model
        self.model = get_model()
        self.trainset = trainset
        self.valset = valset

    def get_parameters(self, config):
        return self.model.get_weights()

    def fit(self, parameters, config):
        self.model.set_weights(parameters)
        self.model.fit(self.trainset, epochs=1, verbose=VERBOSE)
        return self.model.get_weights(), len(self.trainset), {}

    def evaluate(self, parameters, config):
        self.model.set_weights(parameters)
        loss, acc = self.model.evaluate(self.valset, verbose=VERBOSE)
        return loss, len(self.valset), {"accuracy": acc}
```

کلاس ما FlowerClient نحوه انجام آموزش/ارزیابی محلی را تعریف می کند و به Flower اجازه می دهد تا آموزش/ارزیابی محلی را از طریق fit و evaluate فراخوانی کند. هر نمونه از FlowerClient یک مشتری واحد را در سیستم آموزشی فدرال نشان می دهد. سیستم‌های یادگیری فدرال چندین مشتری دارند (در غیر این صورت، چیز زیادی برای فدرال وجود ندارد؟)، بنابراین هر مشتری با نمونه‌ای از FlowerClient نشان داده می‌شود. اگر به عنوان مثال، سه مشتری در حجم کاری خود داشته باشیم، سه نمونه از FlowerClient خواهیم داشت. هنگامی که سرور یک کلاینت خاص را برای آموزش انتخاب می کند (و FlowerClient.evaluate را برای ارزیابی انتخاب می کند) Flower در نمونه مربوطه FlowerClient.fit را فراخوانی می کند.

در اینجا می خواهیم یک سیستم یادگیری فدرال را با ۱۰۰ مشتری روی یک ماشین شبیه سازی کنیم. این بدان معناست که سرور و همه ۱۰۰ مشتری روی یک ماشین زندگی می کنند و منابعی مانند CPU، GPU و حافظه را به اشتراک می گذارند. داشتن ۱۰۰ مشتری به معنای داشتن ۱۰۰ نمونه از FlowerClient در حافظه است. انجام این کار روی یک ماشین می‌تواند به سرعت منابع حافظه موجود را تمام کند، حتی اگر تنها زیرمجموعه‌ای از این کلاینت‌ها در یک دور واحد از یادگیری فدرال شرکت کنند.

Flower، قابلیت‌های شبیه‌سازی خاصی را ارائه می‌کند که نمونه‌های FlowerClient را تنها زمانی ایجاد می‌کند که واقعاً برای آموزش یا ارزیابی لازم باشد. برای فعال کردن این قابلیت Flower برای ایجاد کلاینت در صورت لزوم، باید تابعی به نام client_fn را پیاده سازی کنیم که یک نمونه FlowerClient در صورت تقاضا ایجاد می کند. Flower هر زمان که به یک نمونه از یک مشتری خاص برای فراخوانی مناسب یا ارزیابی نیاز داشته باشد، client_fn را فرا می‌خواند (این نمونه‌ها معمولاً پس از استفاده کنار گذاشته می‌شوند). مشتریان با شناسه مشتری یا cid شناسایی می شوند. از cid می توان برای مثال برای بارگذاری پارتیشن های مختلف داده محلی برای هر مشتری استفاده کرد.

اکنون سه تابع کمکی را برای این مثال تعریف می کنیم (توجه داشته باشید که دو تابع آخر کاملا اختیاری هستند):

- get_client_fn: تابعی است که تابع دیگری را برمی گرداند. Client_fn برگشتی توسط VirtualClientEngine Flower اجرا می شود هر بار که یک کلاینت مجازی جدید (یعنی کلاینتی که در یک فرآیند پایتون شبیه سازی شده است) نیاز به ایجاد شدن داشته باشد. هر بار که استراتژی از آنها نمونه برداری می کند یعنی fit() (یعنی آموزش مدل جهانی بر روی داده های محلی یک مشتری خاص) یا ارزیابی () (یعنی ارزیابی مدل جهانی در مجموعه اعتبار سنجی یک مشتری معین) را انجام دهد.
- weighted_average: این یک تابع اختیاری که در استراتژی استفاده می شود. پس از یک دور ارزیابی (یعنی زمانی که در یک کلاینت تابع ارزیابی اجرا می شود) و معیارهای ارزیابی مشتریان تجمع می کد. در این مثال، ما از این تابع برای محاسبه میانگین وزنی دقت کلاینت‌هایی که ارزیابی شده اند استفاده شده است.
- get_evaluate_fn: این هم تابعی است که تابع دیگری را برمی گرداند. تابع برگردانده شده توسط استراتژی در پایان یک دور آموزش و پس از بدست آوردن یک مدل جهانی جدید پس از تجمیع اجرا می شود. این یک استدلال اختیاری برای استراتژی های گل است. در این مثال، ما از کل مجموعه تست MNIST برای انجام این ارزیابی سمت سرور استفاده می کنیم.

```
def get_client_fn(dataset: FederatedDataset):


    def client_fn(cid: str) -> fl.client.Client:
        # Extract partition for client with id = cid
        client_dataset = dataset.load_partition(int(cid), "train")
        # Now let's split it into train (٩٠%) and validation (١٠%)
        client_dataset_splits = client_dataset.train_test_split(test_size=٠,١)
        trainset = client_dataset_splits["train"].to_tf_dataset(
            columns="img", label_cols="label", batch_size=٣٢
        )
        valset = client_dataset_splits["test"].to_tf_dataset(
            columns="img", label_cols="label", batch_size=٦٤
        )
        # Create and return client
        return FlowerClient(trainset, valset).to_client()
    return client_fn


def weighted_average(metrics: List[Tuple[int, Metrics]]) -> Metrics:
    # Multiply accuracy of each client by number of examples used
    accuracies = [num_examples * m["accuracy"] for num_examples, m in metrics]
    examples = [num_examples for num_examples, _ in metrics]
    # Aggregate and return custom metric (weighted average)
    return {"accuracy": sum(accuracies) / sum(examples)}
```

```
def get_evaluate_fn(testset: Dataset):
    # The `evaluate` function will be called after every round by the strategy
    def evaluate(
        server_round: int,
        parameters: fl.common.NDArrays,
        config: Dict[str, fl.common.Scalar],
    ):
        model = get_model()  # Construct the model
        model.set_weights(parameters)  # Update model with the latest parameters
        loss, accuracy = model.evaluate(testset, verbose=VERBOSE)
        return loss, {"accuracy": accuracy}
    return evaluate
```

ما اکنون FlowerClient را داریم که آموزش و ارزیابی سمت مشتری را تعریف می کند، و client_fn، که به Flower اجازه می دهد هر زمان که نیاز به فراخوانی مناسب یا ارزیابی روی یک مشتری خاص داشت، نمونه های FlowerClient ایجاد کند. آخرین مرحله، شروع شبیه سازی واقعی با استفاده از flwr.simulation.start_simulation است.

تابع start_simulation تعدادی آرگومان را می پذیرد، از جمله client_fn که برای ایجاد نمونه های FlowerClient استفاده می شود، تعداد کلاینت ها برای شبیه سازی num_clients، تعداد دورهای num_rounds و استراتژی. این استراتژی رویکرد/الگوریتم یادگیری فدرال را در بر می گیرد، به عنوان مثال، میانگین گیری فدرال (FedAvg).

Flower دارای تعدادی استراتژی داخلی است، اما ما همچنین می‌توانیم از پیاده‌سازی استراتژی خود برای سفارشی کردن تقریباً تمام جنبه‌های رویکرد یادگیری فدرال استفاده کنیم. برای این مثال، ما از پیاده سازی داخلی FedAvg استفاده می کنیم و آن را با استفاده از چند پارامتر اساسی سفارشی می کنیم. آخرین مرحله فراخوانی واقعی به start_simulation است .

ما می‌توانیم از Flower Datasets برای به دست آوردن یک مجموعه داده پارتیشن بندی شده یا پارتیشن بندی که از قبل پارتیشن بندی نشده است استفاده کنیم. بیایید MNIST را انتخاب کنیم.

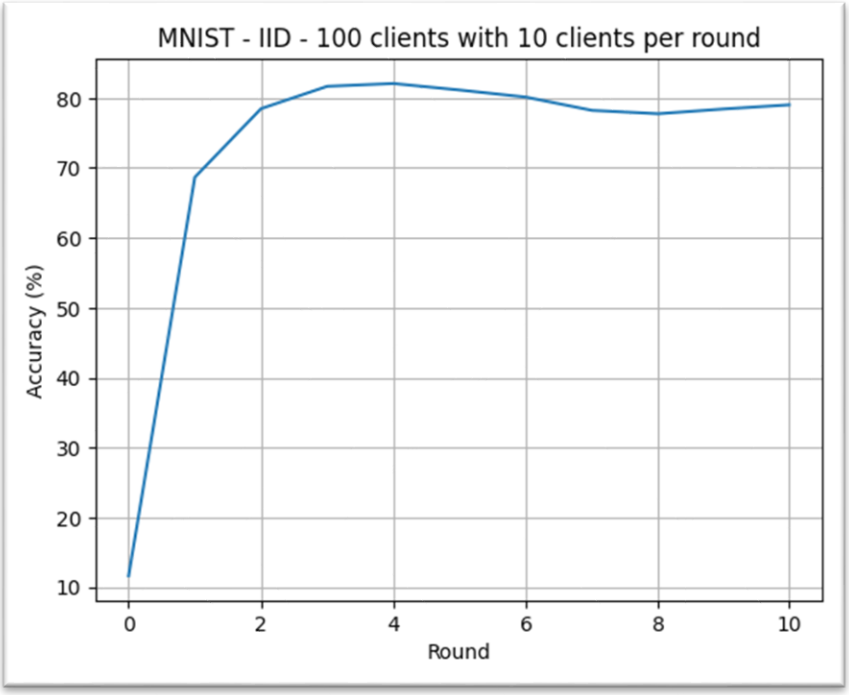
```
# Enable GPU growth in your main process
enable_tf_gpu_growth()
# Download MNIST dataset and partition it
mnist_fds = FederatedDataset(dataset="mnist", partitioners={"train": NUM_CLIENTS})
# Get the whole test set for centralised evaluation
centralized_testset = mnist_fds.load_full("test").to_tf_dataset(
    columns="image", label_cols="label", batch_size=٦٤
)
# Create FedAvg strategy
```

```
strategy = fl.server.strategy.FedAvg(
    fraction_fit=۰,۱,  # Sample ۱۰% of available clients for training
    fraction_evaluate=۰,۰۵,  # Sample ۵% of available clients for evaluation
    min_fit_clients=۱۰,  # Never sample less than ۱۰ clients for training
    min_evaluate_clients=۵,  # Never sample less than ۵ clients for evaluation
    min_available_clients=int(
        NUM_CLIENTS * ۰,۷۵
    ),  # Wait until at least ۷۵ clients are available
    evaluate_metrics_aggregation_fn=weighted_average,  # aggregates federated metrics
    evaluate_fn=get_evaluate_fn(centralized_testset),  # global evaluation function
)
# With a dictionary, you tell Flower's VirtualClientEngine that each
# client needs exclusive access to these many resources in order to run
client_resources = {"num_cpus": ۱, "num_gpus": ۰,۰}
# Start simulation
history = fl.simulation.start_simulation(
    client_fn=get_client_fn(mnist_fds),
    num_clients=NUM_CLIENTS,
    config=fl.server.ServerConfig(num_rounds=۱۰),
    strategy=strategy,
    client_resources=client_resources,
    actor_kwargs={
        "on_actor_init_fn": enable_tf_gpu_growth  # Enable GPU growth upon actor init.
    },
)
```

سپس می توانید از History بازگردانده شده برای ذخیره نتایج در دیسک یا انجام برخی تجسمات (یا البته هر دو) استفاده کنیم. در زیر می توانید ببینید که چگونه می توانید دقت متمرکز به دست آمده در پایان هر دور (از جمله در همان ابتدای آزمایش) را برای مدل جهانی رسم کنید. این تابع value_fn() است که به گزارش های استراتژی ارسال کردیم.

```
import matplotlib.pyplot as plt
print(f"{history.metrics_centralized = }")
global_accuracy_centralised = history.metrics_centralized["accuracy"]
round = [data[۰] for data in global_accuracy_centralised]
```

```
acc = [١٠٠.٠ * data[١] for data in global_accuracy_centralised]
plt.plot(round, acc)
plt.grid()
plt.ylabel("Accuracy (%)")
plt.xlabel("Round")
plt.title("MNIST - IID - ١٠٠ clients with ١٠ clients per round")
```



بخش دوم : داده های CIFAR10 و یک مدل شبکه Pretrain

در این بخش ما از داده های Cifar10 استفاده می کنیم. دیتاست CIFAR-10 دربرگیرنده- ۶۰۰۰۰ تصویر رنگی ۳۲*۳۲ در ده طبقه گوناگون است و برای آموزش و آزمایش انواع مدل تشخیص اشیاء استفاده می شود. ما این مجموعه داده را انخاب کردیم تا از یک شبکه Pretrain در کلاینت ها استفاده کنیم.

این بخش مشابه بخش اول است فقط در بعضی از قسمت ها تغییراتی با توجه به ساختار داده و استفاده از یک مدل Pretrain با توجه به محدودیت های سخت افزاری انجام داده ایم.

تعداد مشتری ها در این بخش ۱۰ تا در نظر گرفتیم.

```
NUM_CLIENTS = ١٠
```

همچنین در این بخش مدل درنظر گرفته شده در کلاینت ها رو یک شبکه Pretrain می باشد. ما از مدل VGG۱۶ که با وزن های دیتاست ImageNet آموزش دیده است ،استفاده کردیم. لایه های کانولیشن Freeze کردیم و قسمت مربوط به FullyConn حذف و با توجه به داده های Cifar10 شبکه جدید با ۱۰ کلاس خروجی ایجاد کردیم.

```
import tensorflow as tf
from tensorflow.keras import layers, models


def get_model():
    """Constructs a simple model architecture suitable for cifar۱۰."""
# Define VGG۱۶ model using the pre-trained weights on ImageNet
    from tensorflow.keras.applications import VGG۱۶
    base_model = VGG۱۶(input_shape=(۳۲, ۳۲, ۳), include_top=False, weights='imagenet')
# Freeze convolutional layers
    for layer in base_model.layers:
        layer.trainable = False


# Create a new model on top
    model = models.Sequential()
    model.add(base_model)
    model.add(layers.Flatten())
    model.add(layers.Dense(۵۱۲, activation=('relu')))
    model.add(layers.Dropout(۰,۲))
    model.add(layers.Dense(۲۵۶, activation=('relu')))
    model.add(layers.Dropout(۰,۲))
    model.add(layers.Dense(۱۰, activation=('softmax')))
# Compile the model
            model.compile(optimizer='adam',     loss='sparse_categorical_crossentropy',
metrics=['accuracy'])
    return model
```

همچنین در قسمت استراتژی ، با توجه به کاهش تعداد کلاینت ها ، تغییراتی دادیم:

```
# Create FedAvg strategy
strategy = fl.server.strategy.FedAvg(
    fraction_fit=۱,  # Sample ۱۰۰٪ of available clients for training
```

```
    fraction_evaluate=٠,٥,  # Sample ٥٠٪ of available clients for evaluation
    min_fit_clients=١٠,  # Never sample less than ١٠ clients for training
    min_evaluate_clients=٥,  # Never sample less than ٥ clients for evaluation
    min_available_clients=int(
        NUM_CLIENTS * ٠,٧٠
    ),  # Wait until at least ٧ clients are available
    evaluate_metrics_aggregation_fn=weighted_average,  # aggregates federated metrics
    evaluate_fn=get_evaluate_fn(centralized_testset),  # global evaluation function
)
```