Topik : 2.1 Federated Learning for Image Data

Objective : Memahami cara melatih model klasifikasi gambar secara federated

Task : Jalankan Tutorial FL dengan dataset EMNIST

Source :
https://www.tensorflow.org/federated/tutorials/federated_learning_for_image_classification?hl=id

Kita menggunakan pelatihan EMNIST klasik untuk memperkenalkan Federasi Learning (FL) lapisan API dari TFF. Tff.learning merupakan satu set antarmuka Tingkat yang lebih tinggi yang dapat digunakan untuk melakukan jenis umum dari tugas – tugas belajar federasi, seperti pelatihan gabungan , terhadap model yang disediakan pengguna yang diimplementasikan di TensorFlow.

Federated Learning API , ditujukan terutama untuk pengguna yang ingin menyambungkan model TensorFlow mereka sendiri ke TFF, memperlakukan yang terakhir Sebagian besar sebagai kotak hitam.

Preparing the input Data

Federated learning memerlukan Kumpulan data gabungan yaitu Kumpulan data dari banyak pengguna. Data federasi biasanya non iid, yang menimbulkan serangkaian tantangan yang unik.

Disini kita menggunakan repositori TFF dengan beberapa dataset, termasuk versi federated dari MINST yang berisi versi dataset asli NIST yang telah di proses ulang menggunakan leaf sehingga data tersebut diindeks berdasarkan penulis angka – angka tersebut.

Berikut cara membuat nya :

```
7   emnist_train, emnist_test = tff.simulation.datasets.emnist.load_data()
8
```

```
2025-08-30 09:14:50.363988: W tensorflow/compiler/tf2tensorrt/utils/py_utils.cc:
Downloading emnist_all.sqlite.lzma: 100%|| 170507172/170507172 [00:56<
2025-08-30 09:16:44.637124: I tensorflow/compiler/xla/stream_executor/cuda/cuda_
```

Dataset dikembalikan oleh load_data() adalah contoh dari tff.simulation.ClienData , sebuah antarmuka yang memungkinkan kita untuk menghitung set pengguna, untuk membangun sebuah tf.data.Dataset yang mewakili data pengguna tertentu, dan untuk query struktur elemen individu. Identitas klien tidak digunakan oleh kerangka pembelajaran federasi yang merupakan satu – satunya tujuan mereka Adalah untuk memungkinkan kita memilih subset data untuk melakukan simulasi.

```
print(len(emnist_train.client_ids))
print(emnist_train.element_type_structure)
```

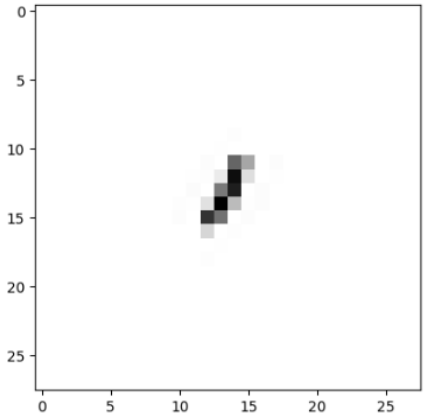
```
3383
OrderedDict([('label', TensorSpec(shape=(), dtype=tf.int32, name=None)), ('pixels', TensorSpec(shape=(28, 28), dtype=tf.float32, name=None))])
```

```
example_dataset = emnist_train.create_tf_dataset_for_client(
    emnist_train.client_ids[0]
)
example_element = next(iter(example_dataset))
print("label : ", example_element['label'].numpy())
```

```
OrderedDict([('label', TensorSpec(shape=(), dtype=tf.int32, name=None)), ('pixels', TensorSpec(shap
label :  1
(venv) ezranahumury@DESKTOP-8003BIM:/mnt/c/KP/MATERI/2.1 Federated Learning for Image Data$
```

Plt.show nya diubah menjadi plt.savefig. karena kita sedang menjalankan script python di terminal CLI dalam WSL/Linux. bukan di lingkungan yang punya GUI/X-server (misalnya Jupyter Notebook atau VS Code interactive).

```
plt.imshow(example_element['pixels'].numpy(), cmap='gray', aspect='equal')
plt.grid(False)
plt.savefig("sample.png")
```



Exploring heterogeneity in federated data

Data federasi biasanya non iid, pengguna biasanya memiliki distribusi data yang berbeda tergantung pada pola penggunaan. Beberapa klien mungkin memilih jumlah contoh pelatihan lebih sedikit dari perangkat, mengalami kekurangan data secara local, sementara klien lain mungkin memilih lebih dari cukup contoh pelatihan. Mari kita jelajahi konsep heterogenitas data yang khas pada sistem federasi menggunakan dataset EMNIST yang tersedia.

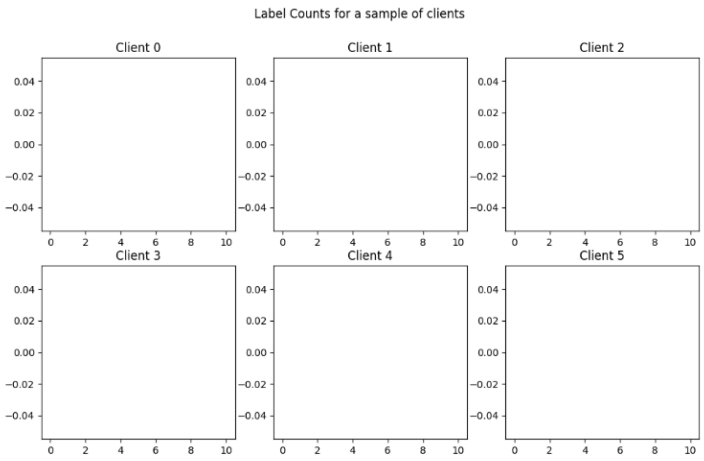
Pertama, mari kita ambil sample data dari satu clien untuk mendapatkan Gambaran tentang contoh – contoh pada satu perangkat simulasi. Karena dataset yang digunakan diindeks berdasarkann penulis unik, data dari satu klien mewakili tulisan tangan satu orang untuk sampel angka 0 – 9, mensimulasikan pola penggunaan unik satu pengguna.

```
#Example MNIST digits for one client
figure = plt.figure(figsize=(20,4))
j = 0

for example in example_dataset.take(40):
    plt.subplot(4, 10, j+1)
    plt.imshow(example_element['pixels'].numpy(), cmap='gray', aspect='equal')
    plt.axis('off')
    j += 1
    plt.savefig("Example.png")
```

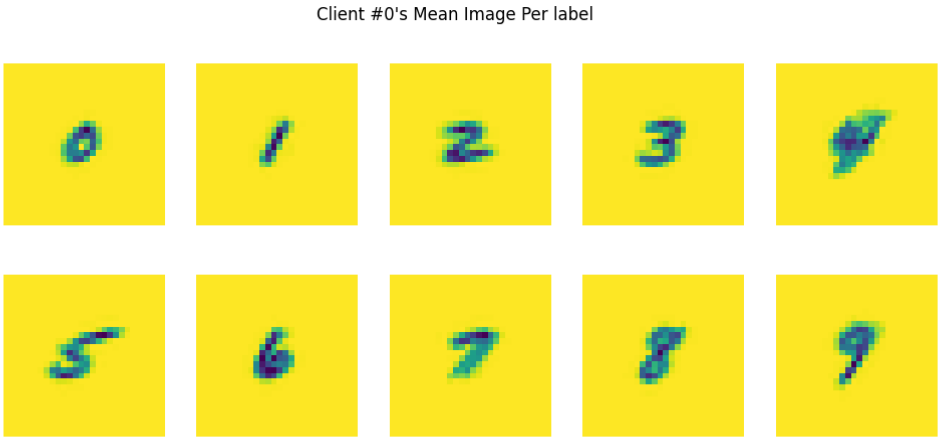
Sekarang, mari kita visualisasikan jumlah contoh pada setiap klien untuk setiap label digit MNIST. Dalam lingkungan federasi, jumlah contoh pada setiap klien dapat sedikit berbeda, tergantung pada perilaku pengguna.

```
#Number of examples per layer for a sample of clients
f = plt.figure(figsize=(12,7))
f.suptitle('Label Counts for a sample of clients')
for i in range(6):
    client_dataset = emnist_train.create_tf_dataset_for_client(
        emnist_train.client_ids[1]
    )
    plot_data = collections.defaultdict(list)
    for example in client_dataset:
        #Append counts individually per label to make plots
        #more colorful instead of one color per plot
        label = example['label'].numpy()
        plot_data['label'].append(label)
    plt.subplot(2,3, i+1)
    plt.title('Client {}'.format(i))
    for j in range(10):
        plt.hist(
            plot_data[j],
            density=False,
            bins=[0,1,2,3,4,5,6,7,8,9,10]
            )
plt.savefig("Client.png")
```

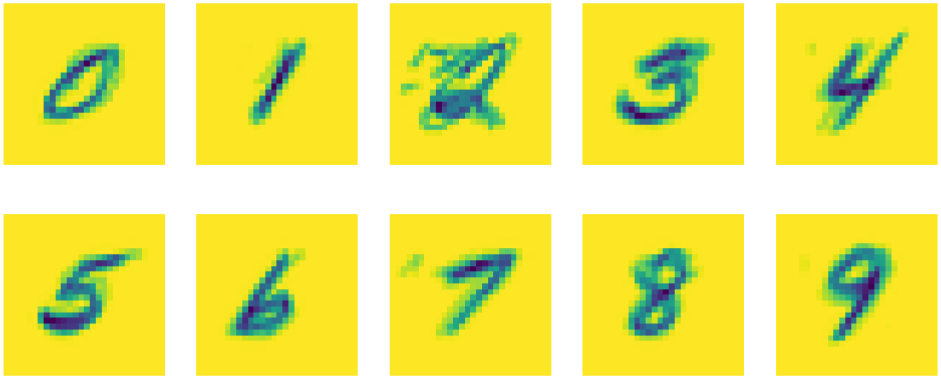


Sekarang mari kita visualisasikan gambar rata – rata per klien untuk setiap label MNIST, kode ini akan menghasilkan rata – rata setiap nilai piksel untuk semua contoh pengguna untuk satu label. Kita akan melihat bahwa gambar rata – rata satu klien untuk satu digit akan terlihat berbeda dari gambar rata – rata klien lain untuk angka yang sama, karena gaya tulisan tangan yang unik dari setiap orang. Kita dapat merenungkan tentang bagaimana setiap putaran pelatihan lokal akan mendorong model ke arah yang berbeda pada setiap klien, karena kita belajar dari data unik pengguna itu sendiri di putaran lokal tersebut

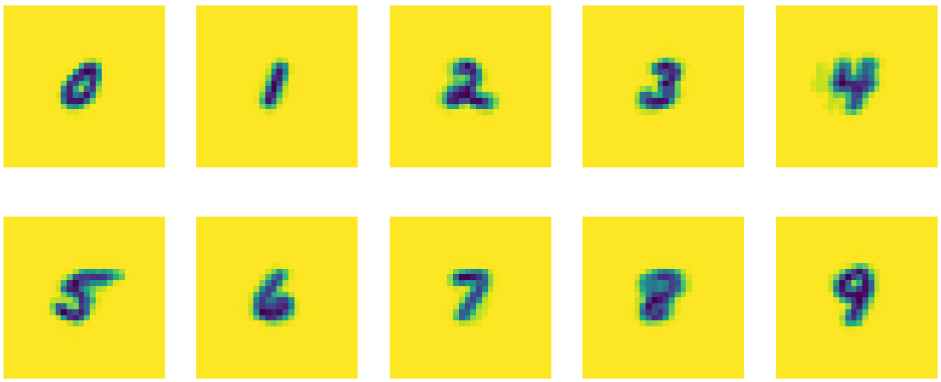
```python
for i in range (5):
    client_dataset = emnist_train.create_tf_dataset_for_client(
        emnist_train.client_ids[i]
    )
    plot_data = collections.defaultdict(list)
    for example in client_dataset :
        plot_data[example['label'].numpy()].append(example['pixels'].numpy())
    f = plt.figure(figsize=(12, 5))
    f.suptitle(f"Client #{i}'s Mean Image Per label")
    for j in range(10):
        mean_img = np.mean(plot_data[j],0)
        plt.subplot(2,5, j+1)
        plt.imshow(mean_img.reshape((28,28)))
        plt.axis('off')
    plt.savefig(f"Client_{i}.png")
    plt.close()
```



Client #2's Mean Image Per label



Client #3's Mean Image Per label



Client #4's Mean Image Per label

Data pengguna dapat bersifat berisik dan diberi label secara tidak akurat. Misalnya, jika kita melihat data Klien #2 di atas, kita dapat melihat bahwa untuk label 2, kemungkinan ada beberapa contoh yang diberi label salah, yang menyebabkan gambar rata-rata menjadi lebih berisik.

Prepocessing the input data

Karena data sudah berupa tf.data.Dataset, PreProcessing dapat dilakukan menggunakan transformasi dataset. Disini, kami meratakan gambar 28x28 menjadi array yang berelemen 784, mengacak contoh – contoh individu, mengorganisirnya ke dalam batch, dan mengganti nama fitur dari pixel dan label ke x dan y untuk digunakan dengan keras. Kami juga menambahkan repeat atas dataset untuk menjalankan beberapa epoch.

```
NUM_CLIENTS = 10
NUM_EPOCHS = 5
BATCH_SIZE = 20
SHUFFLE_BUFFER = 100
PREFETCH_BUFFER = 10

def preprocess(dataset):
    def batch_format_fn(element):
        return collections.OrderedDict(
            x = tf.reshape(element['pixels'], [-1, 784]),
            y = tf.reshape(element['label'], [-1,1])
        )
    return dataset.repeat(NUM_EPOCHS).shuffle(SHUFFLE_BUFFER, seed=1).batch(BATCH_SIZE).map(batch_format_fn).prefetch(PREFETCH_BUFFER)
preprocessed_example_dataset = preprocess(example_dataset)
sample_batch = tf.nest.map_structure(lambda x: x.numpy(), next(iter(preprocessed_example_dataset)))
print(sample_batch)
```

```
Skipping registering GPU devices...
OrderedDict([('x', array([[1., 1., 1., ..., 1., 1., 1.],
       [1., 1., 1., ..., 1., 1., 1.],
       [1., 1., 1., ..., 1., 1., 1.],
       ...,
       [1., 1., 1., ..., 1., 1., 1.],
       [1., 1., 1., ..., 1., 1., 1.],
       [1., 1., 1., ..., 1., 1., 1.]], dtype=float32)), ('y', array([[2],
       [1],
       [5],
       [7],
       [1],
       [7],
       [7],
       [1],
       [4],
       [7],
       [4],
       [2],
       [2],
       [5],
       [4],
       [1],
       [1],
       [0],
       [0],
       [9]], dtype=int32))])
(venv) ezranahumury@DESKTOP-8003BIM:/mnt/c/KP/MATERI/2.1 Federated Learning for Image Data$
```

Salah satu cara untuk memasukkan data federated ke TFF dalam simulasi Adalah dengan menggunakan daftar python, Dimana setiap elemen daftar tersebut menyimpan data pengguna individu. Baik sebagai daftar maupun sebagai tf.data.Dataset. karena kita sudah memiliki antarmuka yang menyediakan opsi terakhir.

Berikut Adalah fungsi bantu sederhana yang akan membangun daftar dataset dari Kumpulan pengguna yang diberikan sebagai input untuk putaran pelatihan dan evaluasi :

```
def make_federated_data(client_data, client_ids) :
    return [
        preprocess(client_data.create_tf_dataset_for_client(x))
        for x in client_ids
    ]

```

Dalam scenario pelatihan federasi yang tipikal, kita berurusan dengan populasi perangkat pengguna yang sangat besar, Dimana hanya Sebagian kecil dari perangkat tersebut yang mungkin tersedia untuk pelatihan pada suatu waktu tertentu. Hal ini terjadi, misalnya

Ketika perangkat klien Adalah ponsel pintar yang hanya berpartisipasi dalam pelatihan saat terhubung ke sumber daya Listrik, terhubung ke jaringan yang diukur, dan dalam keaadaan idle.

Secara umum, saat menjalankan simulasi, kita akan mengambil sample subset acak dari klien yang akan terlibat dalam setiap putaran pelatihan, umumnya berbeda di setiap putaran. Mencapai konvergensi dalam system dengan subset klien yang diambil secara acak di setiap putaran dapat memakan waktu yang cukup lama, dan tidak praktis untuk menjalankan putaran.

Kita akan melakukan pengambilan sample dari Kumpulan client sekali saja , dan menggunakan Kumpulan yang sama di seluruh putaran untuk mempercepat konvergensi (secara sengaja melakukan overfitting pada data dari sedikit pengguna).

```
118  sample_clients = emnist_train.client_ids[0:NUM_CLIENTS]
119  federated_train_data = make_federated_data(emnist_train, sample_clients)
120  print('Number of client datasets: {l}'.format(l=len(federated_train_data)))
121  print('First dataset: {d}'.format(d=federated_train_data[0]))
122
```

```
Number of client datasets: 10
First dataset: <_PrefetchDataset element_spec=OrderedDict([('x', TensorSpec(shape=(None, 784), dtype=tf.float32, name=None)), ('y', TensorSpec(shape=(None, 1), dtype=tf.int32, name=None))])>
(venv) ezranahumury@DESKTOP-8003BIM:/mnt/c/KP/MATERI/2.1 Federated Learning for Image Data$
```

Creating a model with Keras

Berikut Adalah contoh sederhana yang membangun model Keras :

```
127
128  def create_keras_model():
129      return tf.keras.models.Sequential([
130          tf.keras.layers.InputLayer(input_shape=(784,)),
131          tf.keras.layers.Dense(10, kernel_initializer='zeros'),
132          tf.keras.layers.Softmax(),
133      ])
```

Untuk membungkus model apapun dengan TFF, model tersebut harus dibungkus dalam instance antarmuka tff,learning.models.VariableModel, yang menyediakan metode untuk menandai proses forward pass model, property metadata dan sebagainya yang mirip dengan keras. tetapi juga memperkenalkan elemen tambahan, seperti cara mengontrol proses perhitungan metrik federasi.

```
138  def model_fn():
139
140      # We _must_ create a new model here, and _not_ capture it from an external
141      # scope. TFF will call this within different graph contexts.
142      keras_model = create_keras_model()
143      return tff.learning.models.from_keras_model(
144          keras_model,
145          input_spec = preprocessed_example_dataset.element_spec,
146          loss  = tf.keras.losses.SparseCategoricalCrossentropy(),
147          metrics = [tf.keras.metrics.SparseCategoricalAccuracy()]
148      )
```

Training the model on federated data

Catatan penting tentang algoritma Federated Averaging yaitu terdapat 2 optimizer :

1. _clientoptimizer (hanya digunakan untuk menghitung pembaruan model local di setiap klien)
2. _serveroptimizer (menerapkan pembaruan rata – rata ke model global di server)

Secara khusus, ini berarti pilihan optimizer dan laju pembelajaran yang digunakan mungkin perlu berbeeda dari yang kita gunakan untuk melatih model pada dataset i.i.d standar.

```
155 training_process = tff.learning.algorithms.build_weighted_fed_avg(
156     model_fn,
157     client_optimizer_fn=tff.learning.optimizers.build_sgdm(learning_rate=0.02),
158     server_optimizer_fn=tff.learning.optimizers.build_sgdm(learning_rate=1.0))
159
```

TFF telah membangun sepasang *perhitungan federasi* dan dikemas ke dalam sebuah tff.templates.IterativeProcess Dimana perhitungan ini tersedia sebagai sepasang sifat initialize dan next.

Singkatnya, *perhitungan federasi* adalah program di bahasa internal TFF yang dapat mengekspresikan berbagai algoritma federasi. Dalam hal ini, dua perhitungan yang dihasilkan dan dikemas ke dalam iterative_process menerapkan Federasi Averaging.

Tujuan TFF untuk mendefinisikan perhitungan dengan cara yang mereka dapat dieksekusi dalam pengaturan pembelajaran federasi nyata, tetapi saat ini hanya runtime simulasi eksekusi lokal yang diterapkan. Untuk menjalankan komputasi dalam simulator, Anda cukup memanggilnya seperti fungsi Python. Lingkungan interpretasi default ini tidak dirancang untuk kinerja tinggi, tetapi cukup untuk tutorial ini; kami berharap dapat menyediakan runtime simulasi berkinerja lebih tinggi untuk memfasilitasi penelitian skala besar di rilis mendatang.

Mari kita mulai dengan initialize perhitungan. Seperti halnya untuk semua komputasi gabungan, Anda dapat menganggapnya sebagai fungsi. Komputasi tidak memerlukan argumen, dan mengembalikan satu hasil - representasi status proses Rata-Rata Federasi di server. Meskipun kami tidak ingin menyelami detail TFF, mungkin bermanfaat untuk melihat seperti apa keadaan ini. Anda dapat memvisualisasikannya sebagai berikut.

```
160 print(training_process.initialize.type_signature)
```

```
( -> <global_model_weights=<trainable=<float32[784,10],float32[10]>,non_trainable=<>>,distributor=<>,client_work=<learning_rate=float32>,aggregator=<value_sum_process=<>,weight_sum_process=<>>,finalize
r=<learning_rate=float32>>@SERVER)
(venv) ezranahumury@DESKTOP-B003BIM:/mnt/c/KP/MATERI/2.1 Federated Learning for Image Data$
```

Tipe tanda tangan di atas mungkin pada awalnya tampak samar sedikit, Anda dapat mengenali bahwa server negara terdiri dari model (model parameter awal untuk MNIST yang akan didistribusikan ke semua perangkat), dan optimizer_state (informasi tambahan dikelola oleh server, seperti jumlah putaran yang digunakan untuk jadwal hyperparameter, dll.).

Mari kita memanggil initialize perhitungan untuk membangun server negara.

```
162 train_state = training_process.initialize()
```

Kedua dari pasangan perhitungan federasi, next , merupakan satu putaran Federasi Averaging, yang terdiri dari mendorong negara Server (termasuk parameter model) kepada klien, pada perangkat pelatihan data lokal mereka, mengumpulkan dan update Model averaging , dan menghasilkan model baru yang diperbarui di server.

Secara konseptual, Anda bisa memikirkan next sebagai memiliki tanda tangan jenis fungsional yang terlihat sebagai berikut.

```
SERVER_STATE, FEDERATED_DATA -> SERVER_STATE, TRAINING_METRICS
```

Salah satu harus berpikir tentang next() tidak sebagai fungsi yang berjalan di server, melainkan menjadi representasi fungsional deklaratif dari seluruh perhitungan desentralisasi - beberapa input yang disediakan oleh server (SERVER_STATE), tetapi masing-masing peserta perangkat menyumbangkan set data lokalnya sendiri.

Mari kita jalankan satu putaran pelatihan dan visualisasikan hasilnya. Kami dapat menggunakan data gabungan yang telah kami buat di atas untuk sampel pengguna.

```
164  result = training_process.next(train_state, federated_train_data)
165  train_state = result.state
166  train_metrics = result.metrics
167  print('round  1, metrics={}'.format(train_metrics))
168
```

```
round  1, metrics=OrderedDict([('distributor', ()), ('client_work', OrderedDict([('train', OrderedDict([('sparse_categorical_accuracy', 0.12345679), ('loss', 3.119374), ('num_examples', 4860), ('num_batches', 248)]))])), ('aggregator', OrderedDict([('mean_value', ()), ('mean_weight', ())])), ('finalizer', OrderedDict([('update_non_finite', 0)]))])
(venv) ezranahumury@DESKTOP-8003BIM:/mnt/c/KP/MATERI/2.1 Federated Learning for Image Data$
```

Jalankan beberapa putaran lagi. Seperti disebutkan sebelumnya, biasanya pada titik ini anda akan memilih subset data simulasi anda dari sample pengguna baru yang dipilih secara acak untuk setiap putaran untuk mensimulasikan penerapan realistis Dimana pengguna terus datang dan pergi.

```
170  NUM_ROUNDS = 11
171  for round_num in range(2, NUM_ROUNDS):
172    result = training_process.next(train_state, federated_train_data)
173    train_state = result.state
174    train_metrics = result.metrics
175    print('round {:2d}, metrics={}'.format(round_num, train_metrics))
176
```

```
round  2, metrics=OrderedDict([('distributor', ()), ('client_work', OrderedDict([('train', OrderedDict([('sparse_categorical_accuracy', 0.14012346), ('loss', 2.9851403), ('num_examples', 4860), ('num_batches', 248)]))])), ('aggregator', OrderedDict([('mean_value', ()), ('mean_weight', ())])), ('finalizer', OrderedDict([('update_non_finite', 0)]))])
round  3, metrics=OrderedDict([('distributor', ()), ('client_work', OrderedDict([('train', OrderedDict([('sparse_categorical_accuracy', 0.1590535), ('loss', 2.861713), ('num_examples', 4860), ('num_batches', 248)]))])), ('aggregator', OrderedDict([('mean_value', ()), ('mean_weight', ())])), ('finalizer', OrderedDict([('update_non_finite', 0)]))])
round  4, metrics=OrderedDict([('distributor', ()), ('client_work', OrderedDict([('train', OrderedDict([('sparse_categorical_accuracy', 0.17860082), ('loss', 2.740137), ('num_examples', 4860), ('num_batches', 248)]))])), ('aggregator', OrderedDict([('mean_value', ()), ('mean_weight', ())])), ('finalizer', OrderedDict([('update_non_finite', 0)]))])
round  5, metrics=OrderedDict([('distributor', ()), ('client_work', OrderedDict([('train', OrderedDict([('sparse_categorical_accuracy', 0.20102881), ('loss', 2.6186543), ('num_examples', 4860), ('num_batches', 248)]))])), ('aggregator', OrderedDict([('mean_value', ()), ('mean_weight', ())])), ('finalizer', OrderedDict([('update_non_finite', 0)]))])
round  6, metrics=OrderedDict([('distributor', ()), ('client_work', OrderedDict([('train', OrderedDict([('sparse_categorical_accuracy', 0.22345679), ('loss', 2.5006154), ('num_examples', 4860), ('num_batches', 248)]))])), ('aggregator', OrderedDict([('mean_value', ()), ('mean_weight', ())])), ('finalizer', OrderedDict([('update_non_finite', 0)]))])
round  7, metrics=OrderedDict([('distributor', ()), ('client_work', OrderedDict([('train', OrderedDict([('sparse_categorical_accuracy', 0.24794239), ('loss', 2.3858361), ('num_examples', 4860), ('num_batches', 248)]))])), ('aggregator', OrderedDict([('mean_value', ()), ('mean_weight', ())])), ('finalizer', OrderedDict([('update_non_finite', 0)]))])
round  8, metrics=OrderedDict([('distributor', ()), ('client_work', OrderedDict([('train', OrderedDict([('sparse_categorical_accuracy', 0.27160493), ('loss', 2.2757034), ('num_examples', 4860), ('num_batches', 248)]))])), ('aggregator', OrderedDict([('mean_value', ()), ('mean_weight', ())])), ('finalizer', OrderedDict([('update_non_finite', 0)]))])
round  9, metrics=OrderedDict([('distributor', ()), ('client_work', OrderedDict([('train', OrderedDict([('sparse_categorical_accuracy', 0.2958848), ('loss', 2.17098), ('num_examples', 4860), ('num_batches', 248)]))])), ('aggregator', OrderedDict([('mean_value', ()), ('mean_weight', ())])), ('finalizer', OrderedDict([('update_non_finite', 0)]))])
round 10, metrics=OrderedDict([('distributor', ()), ('client_work', OrderedDict([('train', OrderedDict([('sparse_categorical_accuracy', 0.3251029), ('loss', 2.0727072), ('num_examples', 4860), ('num_batches', 248)]))])), ('aggregator', OrderedDict([('mean_value', ()), ('mean_weight', ())])), ('finalizer', OrderedDict([('update_non_finite', 0)]))])
(venv) ezranahumury@DESKTOP-8003BIM:/mnt/c/KP/MATERI/2.1 Federated Learning for Image Data$
```

Kehilangan pelatihan menurun setelah setiap putaran pelatihan gabungan, menunjukkan model konvergen. Ada beberapa keberatan penting dengan metrik pelatihan ini, bagaimanapun, lihat bagian *Evaluasi* nanti dalam tutorial ini.

Displaying model metrics in TensorBoard

Visualisasikan metrik dari komputasi gabungan menggunakan TensorBoard, dimulai dengan membuat direktori dan penulis ringkasan yang sesuai untuk menulis metrics :

```
181  logdir = "/tmp/logs/scalars/training"
182  summary_writer = tf.summary.create_file_writer(logdir)
183  train_state = training_process.initialize()
```

Plot metrics scalar yang relevan dengan penulis ringkasan yang sama :

```
185  with summary_writer.as_default():
186      for round_num in range(1, NUM_ROUNDS):
187          result = training_process.next(train_state, federated_train_data)
188          train_state = result.state
189          train_metrics = result.metrics
190          for name, value in train_metrics['client_work']['train'].items():
191              tf.summary.scalar(name, value, step=round_num)
```

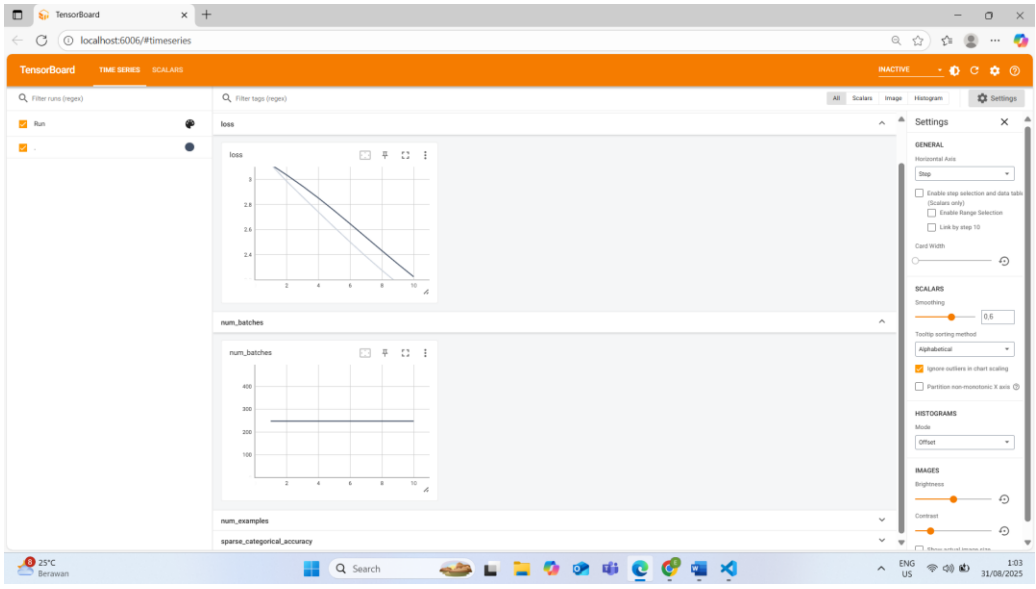
Mulai TensorBoard dengan direktori root log yang ditentukan di atas. Diperlukan beberapa detik untuk memuat data.

```
Skipping registering GPU devices...
(venv) ezranahumury@DESKTOP-8003BIM:/mnt/c/KP/MATERI/2.1 Federated Learning for Image Data$ ls -R /tmp/logs/scalars/training
/tmp/logs/scalars/training:
events.out.tfevents.1756576165.DESKTOP-8003BIM.27769.0.v2
events.out.tfevents.1756576323.DESKTOP-8003BIM.31100.0.v2
events.out.tfevents.1756576640.DESKTOP-8003BIM.34434.0.v2
(venv) ezranahumury@DESKTOP-8003BIM:/mnt/c/KP/MATERI/2.1 Federated Learning for Image Data$
```

```
events.out.tfevents.1756576640.DESKTOP-8003BIM.34434.0.v2
(venv) ezranahumury@DESKTOP-8003BIM:/mnt/c/KP/MATERI/2.1 Federated Learning for Image Data$ tensorboard --logdir=/tmp/logs/scalars/training --port=6006
2025-08-31 01:01:59.332753: I tensorflow/tsl/cuda/cudart_stub.cc:28] Could not find cuda drivers on your machine, GPU will not be used.
2025-08-31 01:01:59.941850: E tensorflow/compiler/xla/stream_executor/cuda/cuda_dnn.cc:9342] Unable to register cuDNN factory: Attempting to register factory for plugin cuDNN when one has already been registered
2025-08-31 01:01:59.943336: E tensorflow/compiler/xla/stream_executor/cuda/cuda_fft.cc:609] Unable to register cuFFT factory: Attempting to register factory for plugin cuFFT when one has already been registered
2025-08-31 01:01:59.946451: E tensorflow/compiler/xla/stream_executor/cuda/cuda_blas.cc:1518] Unable to register cuBLAS factory: Attempting to register factory for plugin cuBLAS when one has already been registered
2025-08-31 01:02:00.314253: I tensorflow/tsl/cuda/cudart_stub.cc:28] Could not find cuda drivers on your machine, GPU will not be used.
2025-08-31 01:02:00.317396: I tensorflow/core/platform/cpu_feature_guard.cc:182] This TensorFlow binary is optimized to use available CPU instructions in performance-critical operations.
To enable the following instructions: AVX2 FMA, in other operations, rebuild TensorFlow with the appropriate compiler flags.
2025-08-31 01:02:10.983237: W tensorflow/compiler/tf2tensorrt/utils/py_utils.cc:38] TF-TRT Warning: Could not find TensorRT
2025-08-31 01:02:36.113458: I tensorflow/compiler/xla/stream_executor/cuda/cuda_gpu_executor.cc:880] could not open file to read NUMA node: /sys/bus/pci/devices/0000:01:00.0/numa_node
Your kernel may have been built without NUMA support.
2025-08-31 01:02:36.115833: W tensorflow/core/common_runtime/gpu/gpu_device.cc:2211] Cannot dlopen some GPU libraries. Please make sure the missing libraries mentioned above are installed properly if you would like to use GPU. Follow the guide at https://www.tensorflow.org/install/gpu for how to download and setup the required libraries for your platform.
Skipping registering GPU devices...

NOTE: Using experimental fast data loading logic. To disable, pass
    "--load_fast=false" and report issues on GitHub. More details:
    https://github.com/tensorflow/tensorboard/issues/4784

Serving TensorBoard on localhost; to expose to the network, use a proxy or pass --bind_all
TensorBoard 2.14.1 at http://localhost:6006/ (Press CTRL+C to quit)
```



Customizing the model implementation

Keras Adalah antarmuka model Tingkat tinggi yang direkomendasikan untuk TensorFlow, dan kami menyarankan untuk menggunakan model Keras (melalui tff.learning.models.from_keras_model) dalam TFF sebanyak mungkin.

Namun, tff.learning menyediakan antarmuka model Tingkat rendah, tff.learning.models.VariableModel, yang mengekspos fungsionalitas minimal yang diperlukan untuk menggunakan model dalam pembelajaran federasi. Mengimplementasikan antarmuka ini secara langsung (mungkin masih menggunakan blok bangunan seperti tf.keras.layers) memungkinkan kustomisasi maksimal tanpa mengubah internal algoritma pembelajaran federasi.

1. Defining model Variables, Forward pass, and metrics

   Langkah pertama Adalah mengidentifikasi variable Tensorflow yang akan kita gunakan. Untuk membuat kode berikut lebih mudah dibaca, kita definisikan struktur data untuk mewakili seluruh set. Struktur ini akan mencakup variable seperti bobot dan bias yang akan kita latih, serta variable yang akan menyimpan berbagai statistic kumpulatif dan penghitung yang akan kita perbarui selama pelatihan, seperti loss_sum, accuracy_sum, dan num_examples.

```
199  #Defining model Variables, Forward pass, and metrics
200  MnistVariables = collections.namedtuple(
201      "MnistVariables", 'weights bias num_examples loss_sum accuracy_sum'
202  )
```

   Berikut Adalah metode yang membuat variable. Demi kesederhanaan, kami mewakili semua statistic sebagai tf.float32, seperti yang akan menghilangkan kebutuhan untuk konversi tipe pada tahap berikutnya. Pembungkus initializer variable sebagai lambdas Adalah persyaratan yang diberlakukan oleh variable sumber daya.

```
205  def create_mnist_variables() :
206      return MnistVariables(
207          weights=tf.Variable(
208              lambda: tf.zeros(dtype=tf.float32, shape=(784, 10)),
209              name='weights',
210              trainable=True
211          ),
212          bias=tf.Variables(
213              lambda: tf.zeros(dtype=tf.float32, shape=(10)),
214              name-'bias',
215              trainable=True
216          ),
217          num_examples=tf.Variable(0.0, name='num_examples', trainable=False),
218          loss_sum=tf.Variable(0.0, name="loss_sum", trainable=False),
219          accuracy_sum=tf.Variable(0.0, name="accuracy_sum", trainable=False)
220      )
```

   Dengan variable untuk parameter model dan statistic kumulatif, sekarang kita dapat mendefinisikan metode forward pass yang menghitung kerugian, memancarkan prediksi dan memperbarui statistic kumulatif untuk satu Kumpulan data input sebagai berikut :

```
def predict_on_batch(variables, x) :
    return tf.nn.softmax(tf.matmul(x, variables.weights) + variables.bias)

def mnist_forward_pass(variables, batch) :
    y = predict_on_batch(variables, batch['x'])
    predictions = tf.cast(tf.argmax(y, 1), tf.int32)

    flat_labels = tf.reshape(batch['y'], [-1])
    loss = -tf.reduce_mean(
        tf.reduce_sum(tf.one_hot(flat_labels, 10) * tf.math.log(y), axis=[1])
    )
    accuracy = tf.reduce_mean(
        tf.cast(tf.equal(predictions, flat_labels), tf.float32)
    )

    num_examples = tf.cast(tf.size(batch['y']), tf.float32)
    variables.num_examples.assign_add(num_examples)
    variables.loss_sum.assign_add(loss * num_examples)
    variables.accuracy_sum.assign_add(accuracy * num_examples)

    return loss, predictions
```

Selanjutnya, kami mendefinisikan fungsi yang mengembalikan sekumpulan metrics local, sekali lagi menggunakan Tensorflow. Ini Adalah nilai (selain pembaruan model, yang ditangani secara otomatis) yang memenuhi syarat untuk digabungkan ke server dalam proses pembelajaran atau evaluasi gabungan.

Kita hanya mengembalikan rata – rata loss dan accuracy, serta num_examples, yang kita harus benar berat kontribusi dari pengguna yang berbeda Ketika menghitung agregat federasi.

```
def get_local_unfinalized_metrics(variables):
  return collections.OrderedDict(
      num_examples=[variables.num_examples],
      loss=[variables.loss_sum, variables.num_examples],
      accuracy=[variables.accuracy_sum, variables.num_examples],
  )
```

Kita perlu menentukan bagaimana agregat metrics local yang dipancarkan oleh masing – masing perangkat melalui perangkat get_local_mnist_metrics. Ini Adalah satu – satunya bagian dari kode yang tidak ditulis dalam Tensorflow itu Adalah perhitungan federasi yang dinyatakan dalam TFF.

```
def get_metric_finalizers():
  return collections.OrderedDict(
      num_examples=tf.function(func=lambda x: x[0]),
      loss=tf.function(func=lambda x: x[0] / x[1]),
      accuracy=tf.function(func=lambda x: x[0] / x[1]),
  )
```

Input metrics argumen berkorespondensi dengan OrderedDict dikembalikan oleh get_local_mnist_metrics di atas, tetapi kritis nilai-nilai tidak lagi tf.Tensors - mereka adalah "kotak" sebagai tff.Value s, untuk membuatnya jelas Anda tidak lagi dapat memanipulasi mereka menggunakan TensorFlow, tetapi hanya menggunakan operator federasi TFF seperti tff.federated_mean dan tff.federated_sum .

2. Constructing an instance of tff.learning.model

Membuat representasi model untuk digunakan dengan TFF serupa dengan yang dibuat dengan membiarkan TFF menyerap dengan model keras.

```
import collections
from collections.abc import Callable


class MnistModel(tff.learning.models.VariableModel):

  def __init__(self):
    self._variables = create_mnist_variables()

  @property
  def trainable_variables(self):
    return [self._variables.weights, self._variables.bias]

  @property
  def non_trainable_variables(self):
    return []

  @property
  def local_variables(self):
    return [
        self._variables.num_examples,
        self._variables.loss_sum,
        self._variables.accuracy_sum,
    ]

  @property
  def input_spec(self):
    return collections.OrderedDict(
        x=tf.TensorSpec([None, 784], tf.float32),
        y=tf.TensorSpec([None, 1], tf.int32),
    )

  @tf.function
  def predict_on_batch(self, x, training=True):
    del training
    return predict_on_batch(self._variables, x)

  @tf.function
  def forward_pass(self, batch, training=True):
    del training
    loss, predictions = mnist_forward_pass(self._variables, batch)
    num_examples = tf.shape(batch['x'])[0]
    return tff.learning.models.BatchOutput(
        loss=loss, predictions=predictions, num_examples=num_examples
    )

  @tf.function
  def report_local_unfinalized_metrics(
      self,
  ) -> collections.OrderedDict[str, list[tf.Tensor]]:
    """Creates an `OrderedDict` of metric names to unfinalized values."""
    return get_local_unfinalized_metrics(self._variables)

  def metric_finalizers(
      self,
  ) -> collections.OrderedDict[str, Callable[[list[tf.Tensor]], tf.Tensor]]:
    """Creates an `OrderedDict` of metric names to finalizers."""
    return get_metric_finalizers()

  @tf.function
  def reset_metrics(self):
    """Resets metrics variables to initial value."""
    for var in self.local_variables:
      var.assign(tf.zeros_like(var))
```

Metode abstrak dan properti yang didefinisikan oleh tff.learning.Model berkorespondensi dengan potongan kode di bagian sebelumnya yang memperkenalkan variabel dan mendefinisikan kerugian dan statistik.

Berikut beberapa poin yang perlu ditekankan:

- Semua variable yang akan digunakan oleh model harus didefinisikan sebagai variable tensorflow, karena TFF tidak menggunakan Python saat Runtime.
- Model harus mendeskripsikan jenis data yang dapat diterimanya (input_spec), karena secara umum TFF Adalah lingkungan yang kuat tipenya dan ingin menentukan tanda tipe untuk semua komponen.
- Membungkus semua logika Tensorflow (forward pass, perhitungan metrics, dll) sebagai tf.function, karena ini membantu memastikan tensorflow dapat diserialisasi dan menghilangkan kebutuhan akan dependensi control eksplisit.

3. Simulating federating training with the new model

Mengganti konstruktur kelas model baru, dan gunakan dua perhitungan federasi dalam proses iterative yang telah dibuat untuk mengulang putaran pelatihan

```
#Simulating federating training with the new model
training_process = tff.learning.algorithms.build_weighted_fed_avg(
    MnistModel,
     client_optimizer_fn=tff.learning.optimizers.build_sgdm(learning_rate=0.02),
     server_optimizer_fn=tff.learning.optimizers.build_sgdm(learning_rate=1.0))


train_state = training_process.initialize()
result = training_process.next(train_state, federated_train_data)
train_state = result.state
metrics = result.metrics
print('round  1, metrics={}'.format(metrics))
```

```
round  1, metrics=OrderedDict([('distributor', ()), ('client_work', OrderedDict([('train', OrderedDict([('num_examples', 4860.0), ('loss', 3.1193738), ('accuracy', 0.12345679)]))])), ('aggregator', Ord
eredDict([('mean_value', ()), ('mean_weight', ())])), ('finalizer', OrderedDict([('update_non_finite', 0)]))])
(venv) ezranahumury@DESKTOP-8003BIM:/mnt/c/KP/MATERI/2.1 Federated Learning for Image Data$
```

```
for round_num in range(2,11):
    result = training_process.next(train_state, federated_train_data)
    train_state = result.state
    metrics = result.metrics
    print('round {:2d}, metrics={}'.format(round_num, metrics))
```

```
round  2, metrics=OrderedDict([('distributor', ()), ('client_work', OrderedDict([('train', OrderedDict([('num_examples', 4860.0), ('loss', 2.9851398), ('accuracy', 0.14012346)]))])), ('aggregator', Ord
eredDict([('mean_value', ()), ('mean_weight', ())])), ('finalizer', OrderedDict([('update_non_finite', 0)]))])
round  3, metrics=OrderedDict([('distributor', ()), ('client_work', OrderedDict([('train', OrderedDict([('num_examples', 4860.0), ('loss', 2.8617127), ('accuracy', 0.1590535)]))])), ('aggregator', Orde
redDict([('mean_value', ()), ('mean_weight', ())])), ('finalizer', OrderedDict([('update_non_finite', 0)]))])
round  4, metrics=OrderedDict([('distributor', ()), ('client_work', OrderedDict([('train', OrderedDict([('num_examples', 4860.0), ('loss', 2.740137), ('accuracy', 0.17860082)]))])), ('aggregator', Orde
redDict([('mean_value', ()), ('mean_weight', ())])), ('finalizer', OrderedDict([('update_non_finite', 0)]))])
round  5, metrics=OrderedDict([('distributor', ()), ('client_work', OrderedDict([('train', OrderedDict([('num_examples', 4860.0), ('loss', 2.618655), ('accuracy', 0.20102881)]))])), ('aggregator', Orde
redDict([('mean_value', ()), ('mean_weight', ())])), ('finalizer', OrderedDict([('update_non_finite', 0)]))])
round  6, metrics=OrderedDict([('distributor', ()), ('client_work', OrderedDict([('train', OrderedDict([('num_examples', 4860.0), ('loss', 2.5006156), ('accuracy', 0.22345679)]))])), ('aggregator', Ord
eredDict([('mean_value', ()), ('mean_weight', ())])), ('finalizer', OrderedDict([('update_non_finite', 0)]))])
round  7, metrics=OrderedDict([('distributor', ()), ('client_work', OrderedDict([('train', OrderedDict([('num_examples', 4860.0), ('loss', 2.3858361), ('accuracy', 0.24794239)]))])), ('aggregator', Ord
eredDict([('mean_value', ()), ('mean_weight', ())])), ('finalizer', OrderedDict([('update_non_finite', 0)]))])
round  8, metrics=OrderedDict([('distributor', ()), ('client_work', OrderedDict([('train', OrderedDict([('num_examples', 4860.0), ('loss', 2.2757037), ('accuracy', 0.27160493)]))])), ('aggregator', Ord
eredDict([('mean_value', ()), ('mean_weight', ())])), ('finalizer', OrderedDict([('update_non_finite', 0)]))])
round  9, metrics=OrderedDict([('distributor', ()), ('client_work', OrderedDict([('train', OrderedDict([('num_examples', 4860.0), ('loss', 2.1709795), ('accuracy', 0.2958848)]))])), ('aggregator', Orde
redDict([('mean_value', ()), ('mean_weight', ())])), ('finalizer', OrderedDict([('update_non_finite', 0)]))])
round 10, metrics=OrderedDict([('distributor', ()), ('client_work', OrderedDict([('train', OrderedDict([('num_examples', 4860.0), ('loss', 2.0727072), ('accuracy', 0.3251029)]))])), ('aggregator', Orde
redDict([('mean_value', ()), ('mean_weight', ())])), ('finalizer', OrderedDict([('update_non_finite', 0)]))])
(venv) ezranahumury@DESKTOP-8003BIM:/mnt/c/KP/MATERI/2.1 Federated Learning for Image Data$
```

Evaluation

Untuk melakukan evaluasi data federasi, kita akan membuat perhitungan federasi lain yang dirancang untuk tujuan ini menggunakan tff.learning.build_federated_evaluation fungsi, dan lewat dalam model konstruktor sebagai argument. Evaluasi tidak melakukan penurunan gradien, dan tidak perlu membuat pengoptimal.

```
358  evalution_process = tff.learning.algorithms.build_fed_eval(MnistModel)
359
```

Memeriksa tipe signature abstrak dari fungsi evaluasi :

```
361  print(evalution_process.next.type_signature.formatted_representation())
```

```
2025-08-31 14:55:04.271251: W tensorflow/core/common_runtime/gpu/gpu_device.cc:2211] Cannot dlopen some GPU libraries
Skipping registering GPU devices...
(<
  state=<
    global_model_weights=<
      trainable=<
        float32[784,10],
        float32[10]
      >,
      non_trainable=<>
    >,
    distributor=<>,
    client_work=<
      <>,
      <
        num_examples=<
          float32
        >,
        loss=<
          float32,
          float32
        >,
        accuracy=<
          float32,
          float32
        >
      >
    >,
    aggregator=<
      value_sum_process=<>,
      weight_sum_process=<>
    >,
    finalizer=<>
  >@SERVER,
  client_data={<
    x=float32[?,784],
    y=int32[?,1]
  >*}@CLIENTS
> -> <
  state=<
    global_model_weights=<
      trainable=<
        float32[784,10],
        float32[10]
      >,
      non_trainable=<>
    >,
    distributor=<>,
    client_work=<
      <>,
      <
        num_examples=<
          float32
        >,
        loss=<
          float32,
          float32
        >,
        accuracy=<
          float32,
          float32
        >
      >
    >,
```

```
  >@SERVER,
  metrics=<
    distributor=<>,
    client_work=<
      eval=<
        current_round_metrics=<
          num_examples=float32,
          loss=float32,
          accuracy=float32
        >,
        total_rounds_metrics=<
          num_examples=float32,
          loss=float32,
          accuracy=float32
        >
      >
    >,
    aggregator=<
      mean_value=<>,
      mean_weight=<>
    >,
    finalizer=<>
  >@SERVER
>)
(venv) ezranahumury@DESKTOP-8003BIM:/mnt/c/KP/MATERI/2.1 Federated Learning for Image Data$
```

Mirip dengan tff.templates.IterativeProcess.next tetapi dengan dua perbandingan penting :

1. Kami tidak mengembalikan status server, karena evaluasi tidak mengubah model atau aspek status lainnya.
2. Evaluasi hanya membutuhkan model, dan tidak memerlukan bagian lain dari status server yang mungkin terkait dengan pelatihan, seperti variable pengoptimalan.

```
SERVER_MODEL, FEDERATED_DATA -> TRAINING_METRICS
```

Mari kita lakukan evaluasi pada status terakhir yang kita capai selama pelatihan. Dalam rangka untuk mengekstrak model dilatih terbaru dari negara server, Anda cukup mengakses .model anggota, sebagai berikut.

```
361  evalution_state = evalution_process.initialize()
362  model_weights = training_process.get_model_weights(train_state)
363  evalution_state = evalution_process.set_model_weights(evalution_state, model_weights)
```

Dengan keadaan evaluasi yang berisi bobot model yang akan dievaluasi, kita dapat menghitung metrics evaluasi menggunakan dataset evaluasi dengan memanggil metode sama seperti saat pelatihan.

Ini akan mengembalikan instance tff.learning.templates.LearningProcessOutput

```
365
366  evaluation_output = evalution_process.next(
367      evalution_state, federated_train_data
368  )
```

Secara konvensional, metrics pelatihan yang dilaporkan oleh proses pelatihan iterative umumnya mencerminkan kinerja model pada awal putaran pelatihan, sehingga metrics evaluasi selalu satu Langkah di depan.

```
369
370  print(str(evaluation_output.metrics))
```

```
Skipping registering GPU devices...
OrderedDict([('distributor', ()), ('client_work', OrderedDict([('eval', OrderedDict([('current_round_metrics', OrderedDict([('num_examples', 4860.0), ('loss', 1.6654207), ('accuracy', 0.3621399)])), ('
total_rounds_metrics', OrderedDict([('num_examples', 4860.0), ('loss', 1.6654207), ('accuracy', 0.3621399)]))]))])), ('aggregator', OrderedDict([('mean_value', ()), ('mean_weight', ())])), ('finalizer'
, ())])
(venv) ezranahumury@DESKTOP-8003BIM:/mnt/c/KP/MATERI/2.1 Federated Learning for Image Data$
```

Mari kita kompilasi sampel uji data federasi dan jalankan kembali evaluasi pada data uji. Data tersebut akan berasal dari sampel pengguna nyata yang sama, tetapi dari kumpulan data yang dipisahkan secara terpisah.

```
372  federated_test_data = make_federated_data(emnist_test, sample_clients)
373  print(len(federated_test_data), federated_test_data[0])
```

```
Skipping registering GPU devices...
10 <_PrefetchDataset element_spec=OrderedDict([('x', TensorSpec(shape=(None, 784), dtype=tf.float32, name=None)), ('y', TensorSpec(shape=(None, 1), dtype=tf.int32, name=None))])>
(venv) ezranahumury@DESKTOP-8003BIM:/mnt/c/KP/MATERI/2.1 Federated Learning for Image Data$
```

```
375  evaluation_output = evalution_process.next(evalution_state, federated_test_data)
376  print(str(evaluation_output.metrics))
377
```

,

```
OrderedDict([('distributor', ()), ('client_work', OrderedDict([('eval', OrderedDict([('current_round_metrics', OrderedDict([('num_examples', 580.0), ('loss', 1.7750841), ('accura
cy', 0.33620688)])), ('total_rounds_metrics', OrderedDict([('num_examples', 580.0), ('loss', 1.7750841), ('accuracy', 0.33620688)]))]))])), ('aggregator', OrderedDict([('mean_val
ue', ()), ('mean_weight', ())])), ('finalizer', ())])
(venv) ezranahumury@DESKTOP-8003BIM:/mnt/c/KP/MATERI/2.1 Federated Learning for Image Data$
```