Topik : 2.2. Federated Averaging Algorithm

Objective : Memahami tff.learning.algorithms.build_fed_avg()

Task : Ubah jumlah client dan epochs → analisis hasil

Source : https://arxiv.org/pdf/1602.05629

The FederatedAveraging Algorithm

SGD dapat diterapkan secara naif pada masalah optimasi terfederasi, di mana satu perhitungan gradien batch (misalnya pada klien yang dipilih secara acak) dilakukan per putaran komunikasi. Pendekatan ini efisien secara komputasi, tetapi membutuhkan putaran pelatihan yang sangat banyak untuk menghasilkan model yang baik (misalnya, bahkan dengan menggunakan pendekatan lanjutan seperti normalisasi batch, Ioffe dan Szegedy [21] melatih MNIST selama 50.000 langkah pada minibatch berukuran 60). Kami mempertimbangkan baseline ini dalam eksperimen CIFAR-10 kami.

Dalam pengaturan federasi, ada sedikit biaya dalam waktu jam dinding untuk melibatkan lebih banyak klien, dan jadi untuk baseline kami, kami menggunakan SGD sinkron batch besar; eksperimen oleh Chen et al. menunjukkan pendekatan ini adalah yang tercanggih dalam pengaturan pusat data, di mana ia mengungguli pendekatan asinkron. Untuk menerapkan pendekatan ini dalam pengaturan federasi, kami memilih fraksi C klien pada setiap putaran, dan menghitung gradien kerugian atas semua data yang dimiliki oleh klien ini. Dengan demikian, C mengontrol ukuran batch global, dengan C 1 sesuai dengan penurunan gradien batch penuh (non-stokastik). Kami menyebut algoritma baseline ini sebagai FederatedSGD (atau FedSGD).

Implementasi tipikal dari **FedSGD** dengan $C = 1$ dan laju pembelajaran tetap ∩ , membuat setiap k menghitung $g_k = \nabla F_k(w_t),$ yaitu gradien rata – rata pada data lokalnya terhadap model saat ini $w_t.$. server pusat kemudian menggabungkan gradien ini dan menerapkan pembaruan.

$$w_{t+1} \leftarrow w_t - \eta \sum_{k=1}^{K} \frac{n_k}{n} g_k,$$

karena $\sum_{k=1}^{K} \frac{n_k}{n} g_k = \nabla f(w_t).$

Pembaruan yang ekuivalen dapat dituliskan sebagai berikut : untuk setiap k,

$$w_{t+1}^k \leftarrow w_t - \eta g_k,$$

Dan kemudian

$$w_{t+1} \leftarrow \sum_{k=1}^{K} \frac{n_k}{n} w_{t+1}^k.$$

Artinya, setiap klien secara local melakukan satu Langkah penurunan gradien pada model saat ini menggunakan data lokalnya, lalu server mengambil rata – rata berbobot dari model yang dihasilkan.

Setelah algoritma ditulis dengan cara ini, kita dapat menambahkan lebih banyak komputasi ke setiap klien dengan mengulangi pembaruan local.

$$w^k \leftarrow w^k - \eta \nabla F_k(w^k)$$

Beberapa kali sebelum Langkah averaging. Kami menyebut pendekatan ini sebagai **FederatedAveraging (FedAvg)**.

Jumlah komputasi dikendalikan oleh tiga parameter kunci :

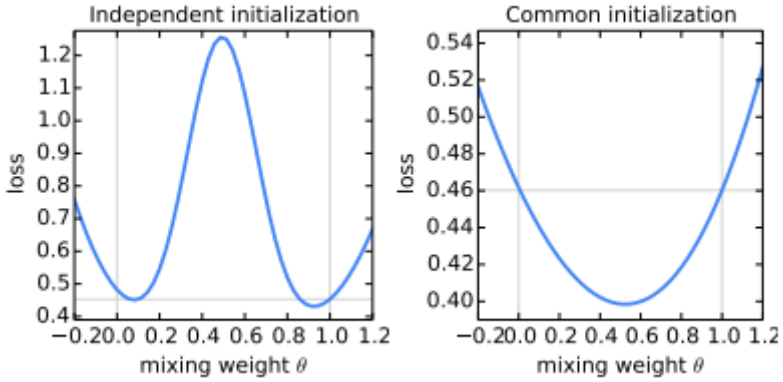
- **C,** Fraksi klien yang ikut serta dalam setiap putaran
- **E,** Jumlah epoch yang dilakukan setiap klien pada dataset lokalnya dalam setiap putaran.
- **B,** ukuran minibatch local yang digunakan untuk pembaruan klien.

Kami menuliskan $B = \infty$ untuk menunjukkan bahwa seluruh dataset local diperlakukan sebagai satu minibatch. Jadi, pada salah satu ujung dari keluarga algoritma, kita bisa mengambil $B = \infty$ dan $E = 1$ yang sesuai dengan **FedSGD.**

Untuk klien dengan n_k contoh local, jumlah pembaruan local per putaran diberikan oleh :

$$u_k = \frac{E\, n_k}{B}.$$

Untuk tujuan non-konveks umum, rata-rata model dalam ruang parameter dapat menghasilkan model yang sangat buruk.



gambar 1

Loss pada seluruh set pelatihan MNIST untuk model yang dihasilkan dengan merata-ratakan parameter dari dua model www dan w′w'w′ menggunakan.

$$\theta w + (1-\theta)w'$$

untuk 50 nilai θ yang terdistribusi merata dalam interval $\theta \in [-0.2, 1.2]$.

KETERANGAN Gambar:

*Model w dan w' dilatih menggunakan **SGD** pada dataset kecil yang berbeda. Untuk plot disebelah kiri, w dan w' diinisialisasikan menggunakan random seed yang berbeda; sedangkan untuk plot di sebelah kanan, digunakan Seed yang sama. Perhatikan bahwa skala sumbu -y berbeda.*

Garis horizontal menunjukkan loss terbaik yang dicapai oleh w atau w' ((yang cukup mirip, sesuai dengan garis vertikal pada $\theta = 0$ dan $\theta = 1$*). Dengan inisialisasi Bersama (shared initialization), merata – ratakan model menghasilkan penurunan signifikan pada loss di keseluruhan set pelatihan. (jauh lebih baik disbanding loss dari salah satu model induk)*

Mengikuti pendekatan dari Goodfellow et al, perilaku buruk ini Ketika kita merata – ratakan dua model pengenalan digit MNIST yang dilatih dari kondisi inisialisasi berbeda (gambar 1, kiri). Untuk gambar ini, model induk w dan w' masing – masing dilatih pada sampel IID non-overlapping sebanyak 600 contoh dari set pelatihan MNIST.

Pelatihan menggunakan **SGD** dengan laju pembelajaran tetap 0.1 selama 240 pembaruan pada minibatch berukuran 50 atau $E = 20$ kali *pass* pada dataset berukuran 600. Hal ini kira – kira sama dengan jumlah pelatihan Dimana model mulai *overfit* pada dataset lokalnya.

Penelitian terbaru menunjukkan bahwa dalam praktiknya, permukaan *loss* dari jaringan saraf berparameter besar cukup *well-behaved* dan khususnya jauh lebih sedikit cenderung terjebak pada *local minima* yang buruk daripada yang sebelumnya dan diperkirakan [11,17,9]. Dan memang , Ketika kita memulai dua model **dari inisialisasi acak yang sama** dan kemudian melatih setiap model secara independent pada subsetdata yang berbeda , kita menemukan bahwa rata – rata parameter secara naif bekerja sangat baik, rata – rata dua model :

$$\frac{1}{2}w + \frac{1}{2}w',$$

Mencapai *loss* yang jauh lebih rendah pada seluruh set pelatigan MNIST dibandingkan dengan model terbaik yang diperoleh dengan melatih salah satu dari subset data kecil tersebut secara independent.

Algorithm 1 `FederatedAveraging`. The K clients are indexed by k; B is the local minibatch size, E is the number of local epochs, and η is the learning rate.

Server executes:
initialize w_0
for each round $t = 1, 2, \dots$ **do**
  $m \leftarrow \max(C \cdot K, 1)$
  $S_t \leftarrow$ (random set of m clients)
  for each client $k \in S_t$ **in parallel do**
    $w_{t+1}^k \leftarrow \text{ClientUpdate}(k, w_t)$
  $m_t \leftarrow \sum_{k \in S_t} n_k$
  $w_{t+1} \leftarrow \sum_{k \in S_t} \frac{n_k}{m_t} w_{t+1}^k$ *// Erratum*[4]

ClientUpdate(k, w): *// Run on client k*
$\mathcal{B} \leftarrow$ (split $\mathcal{P}_k$ into batches of size B)
for each local epoch i from 1 to E **do**
  for batch $b \in \mathcal{B}$ **do**
    $w \leftarrow w - \eta \nabla \ell(w; b)$
return w to server

Source : https://www.tensorflow.org/federated/api_docs/python/tff/learning/algorithms

Library yang menyediakan implementasi federated Learning Algorithms

`build_fed_eval(...)`	Membangun proses pembelajaran yang melakukan evaluasi federated.
`build_fed_kmeans(...)`	Membangun proses pembelajaran untuk *federated k-means clustering*
`build_fed_recon(...)`	Membangun *IterativeProcess* untuk optimisasi menggunakan FedRecon
`build_fed_recon_eval(...)`	Membanngun sebuah **tff.Computation** untuk mengevaluasi model rekonstruksi.
`build_fed_sgd(...)`	Membangun proses pembelajaran yang melakukan *federated* SGD
`build_mime_lite_with_optimizer_schedule(...)`	Membangun proses pembelajaran untuk *Mime Lite* dengan penjadwalan optimizer.
`build_personalization_eval_computation(...)`	Membangun perhitungan TFF untuk mengevaluasi strategi personalisasi.
`build_unweighted_fed_avg(...)`	Membangun proses pembelajaran yang melakukan *federated averaging*
`build_unweighted_fed_prox(...)`	Membangun proses pembelajaran yang menjalankan algoritma *FedProx*
`build_unweighted_mime_lite(...)`	Membangun proses pembelajaran yang menjalankan *Mime Lite*
`build_weighted_fed_avg(...)`	Membangun proses pembelajaran yang melakukan *federated averaging*
`build_weighted_fed_avg_with_optimizer_schedule(...)`	Membangun proses pembelajaran untuk *FedAvg* dengan penjadwalan optimizer klien.
`build_weighted_fed_prox(...)`	Membangun proses pembelajaran yang menjalankan algoritma *FedProx.*
`build_weighted_mime_lite(...)`	Membangun proses

- NUM_CLIENTS → berapa klien **ikut per ronde (C)**.
- NUM_EPOCHS di preprocess(...).repeat(NUM_EPOCHS) → **berapa epoch lokal (E)** yang dijalankan tiap klien sebelum averaging.

Task :

Menggunakan client = 10 dan Epoch nya 5

```
95    NUM_CLIENTS = 10
96    NUM_EPOCHS = 5
97    BATCH_SIZE = 20
98    SHUFFLE_BUFFER = 100
99    PREFETCH_BUFFER = 10
100
```

```
round  1, metrics=OrderedDict([('distributor', ()), ('client_work', OrderedDict([('train', OrderedDict([('sparse_categorical_accuracy', 0.12345679), ('loss', 3.119374), ('num_examples', 4860), ('num_ba
tches', 248)])])), ('aggregator', OrderedDict([('mean_value', ()), ('mean_weight', ())])), ('finalizer', OrderedDict([('update_non_finite', 0)])])
round  2, metrics=OrderedDict([('distributor', ()), ('client_work', OrderedDict([('train', OrderedDict([('sparse_categorical_accuracy', 0.14012346), ('loss', 2.9851403), ('num_examples', 4860), ('num_b
atches', 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_bat
ches', 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_ba
tches', 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_b
atches', 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_b
atches', 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_b
atches', 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_b
atches', 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_batc
hes', 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_ba
tches', 248)])])), ('aggregator', OrderedDict([('mean_value', ()), ('mean_weight', ())])), ('finalizer', OrderedDict([('update_non_finite', 0)])])
(venv) ezranahumury@DESKTOP-8003BIM:/mnt/c/KP/MATERI/2.2 Federated Averaging Algorithm$
```

Menggunakan clientnya 20 dan Epoch nya 3

```
95    NUM_CLIENTS = 20
96    NUM_EPOCHS = 3
97    BATCH_SIZE = 20
98    SHUFFLE_BUFFER = 100
99    PREFETCH_BUFFER = 10
```

```
Skipping registering GPU devices...
round  1, metrics=OrderedDict([('distributor', ()), ('client_work', OrderedDict([('train', OrderedDict([('sparse_categorical_accuracy', 0.116658), ('loss', 3.0941992), ('num_examples', 5769), ('num_bat
ches', 297)])])), ('aggregator', OrderedDict([('mean_value', ()), ('mean_weight', ())])), ('finalizer', OrderedDict([('update_non_finite', 0)])])
round  2, metrics=OrderedDict([('distributor', ()), ('client_work', OrderedDict([('train', OrderedDict([('sparse_categorical_accuracy', 0.11891142), ('loss', 3.0247633), ('num_examples', 5769), ('num_b
atches', 297)])])), ('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', 0.13225862), ('loss', 2.9488199), ('num_examples', 5769), ('num_b
atches', 297)])])), ('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', 0.14439245), ('loss', 2.8751888), ('num_examples', 5769), ('num_b
atches', 297)])])), ('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', 0.15531288), ('loss', 2.8029232), ('num_examples', 5769), ('num_b
atches', 297)])])), ('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', 0.16814005), ('loss', 2.732064), ('num_examples', 5769), ('num_ba
tches', 297)])])), ('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', 0.183394), ('loss', 2.662589), ('num_examples', 5769), ('num_batc
hes', 297)])])), ('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', 0.19639452), ('loss', 2.5944479), ('num_examples', 5769), ('num_b
atches', 297)])])), ('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', 0.21459524), ('loss', 2.5276048), ('num_examples', 5769), ('num_b
atches', 297)])])), ('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', 0.22915584), ('loss', 2.4621482), ('num_examples', 5769), ('num_b
atches', 297)])])), ('aggregator', OrderedDict([('mean_value', ()), ('mean_weight', ())])), ('finalizer', OrderedDict([('update_non_finite', 0)])])
(venv) ezranahumury@DESKTOP-8003BIM:/mnt/c/KP/MATERI/2.2 Federated Averaging Algorithm$
```

Menggunakan clientnya 20 dan Epoch nya 10

```
95  NUM_CLIENTS = 20
96  NUM_EPOCHS = 10
97  BATCH_SIZE = 20
98  SHUFFLE_BUFFER = 100
99  PREFETCH_BUFFER = 10
100
```

```
round  1, metrics=OrderedDict([('distributor', ()), ('client_work', OrderedDict([('train', OrderedDict([('sparse_categorical_accuracy', 0.1376495), ('loss', 3.0042956), ('num_examples', 19230), ('num_batches', 968)])])), ('aggregator', OrderedDict([('mean_value', ()), ('mean_weight', ())])), ('finalizer', OrderedDict([('update_non_finite', 0)])])
round  2, metrics=OrderedDict([('distributor', ()), ('client_work', OrderedDict([('train', OrderedDict([('sparse_categorical_accuracy', 0.16952679), ('loss', 2.7603402), ('num_examples', 19230), ('num_batches', 968)])])), ('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', 0.21617265), ('loss', 2.5182953), ('num_examples', 19230), ('num_batches', 968)])])), ('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', 0.2652106), ('loss', 2.3060207), ('num_examples', 19230), ('num_batches', 968)])])), ('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', 0.3151362), ('loss', 2.1169574), ('num_examples', 19230), ('num_batches', 968)])])), ('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', 0.36448258), ('loss', 1.9487166), ('num_examples', 19230), ('num_batches', 968)])])), ('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', 0.41138846), ('loss', 1.8021601), ('num_examples', 19230), ('num_batches', 968)])])), ('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', 0.45439416), ('loss', 1.6739341), ('num_examples', 19230), ('num_batches', 968)])])), ('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', 0.49646387), ('loss', 1.560842), ('num_examples', 19230), ('num_batches', 968)])])), ('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', 0.53385335), ('loss', 1.4617819), ('num_examples', 19230), ('num_batches', 968)])])), ('aggregator', OrderedDict([('mean_value', ()), ('mean_weight', ())])), ('finalizer', OrderedDict([('update_non_finite', 0)])])
(venv) ezranahumury@DESKTOP-8003BIM:/mnt/c/KP/MATERI/2.2 Federated Averaging Algorithm$
```

Menggunakan clientnya 20 dan Epoch nya 15

```
95  NUM_CLIENTS = 20
96  NUM_EPOCHS = 15
97  BATCH_SIZE = 20
98  SHUFFLE_BUFFER = 100
99  PREFETCH_BUFFER = 10
100
```

```
Skipping registering GPU devices...
round  1, metrics=OrderedDict([('distributor', ()), ('client_work', OrderedDict([('train', OrderedDict([('sparse_categorical_accuracy', 0.16037442), ('loss', 2.8771412), ('num_examples', 28845), ('num_batches', 1449)])])), ('aggregator', OrderedDict([('mean_value', ()), ('mean_weight', ())])), ('finalizer', OrderedDict([('update_non_finite', 0)])])
round  2, metrics=OrderedDict([('distributor', ()), ('client_work', OrderedDict([('train', OrderedDict([('sparse_categorical_accuracy', 0.21951811), ('loss', 2.522267), ('num_examples', 28845), ('num_batches', 1449)])])), ('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', 0.29596117), ('loss', 2.2035096), ('num_examples', 28845), ('num_batches', 1449)])])), ('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', 0.3659213), ('loss', 1.9442384), ('num_examples', 28845), ('num_batches', 1449)])])), ('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', 0.43716416), ('loss', 1.7289083), ('num_examples', 28845), ('num_batches', 1449)])])), ('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', 0.5008147), ('loss', 1.5534811), ('num_examples', 28845), ('num_batches', 1449)])])), ('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', 0.5555556), ('loss', 1.4097036), ('num_examples', 28845), ('num_batches', 1449)])])), ('aggregator', OrderedDict([('mean_value', ()), ('mean_weight', ())])), ('finalizer', OrderedDict([('update_non_finite', 0)])])
mples', 28845), ('num_batches', 1449)])])), ('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', 0.43716416), ('loss', 1.7289083), ('num_examples', 28845), ('num_batches', 1449)])])), ('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', 0.5008147), ('loss', 1.5534811), ('num_examples', 28845), ('num_batches', 1449)])])), ('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', 0.5555556), ('loss', 1.4097036), ('num_examples', 28845), ('num_batches', 1449)])])), ('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', 0.43716416), ('loss', 1.7289083), ('num_examples', 28845), ('num_batches', 1449)])])), ('aggregator', OrderedDict([('mean_value', ()), ('mean_weight', ())])), ('finalizer', OrderedDict([('update_non_finite', 0)])])
mples', 28845), ('num_batches', 1449)])])), ('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', 0.43716416), ('loss', 1.7289083), ('num_exmples', 28845), ('num_batches', 1449)])])), ('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', 0.43716416), ('loss', 1.7289083), ('num_examples', 28845), ('num_batches', 1449)])])), ('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', 0.5008147), ('loss', 1.5534811), ('num_examples', 28845), ('num_batches', 1449)])])), ('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', 0.5555556), ('loss', 1.4097036), ('num_examples', 28845), ('num_batches', 1449)])])), ('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', 0.60364014), ('loss', 1.2918327), ('num_examples', 28845), ('num_batches', 1449)])])), ('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', 0.64291906), ('loss', 1.1942939), ('num_examples', 28845), ('num_batches', 1449)])])), ('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', 0.67502165), ('loss', 1.1130261), ('num_examples', 28845), ('num_batches', 1449)])])), ('aggregator', OrderedDict([('mean_value', ()), ('mean_weight', ())])), ('finalizer', OrderedDict([('update_non_finite', 0)])])
(venv) ezranahumury@DESKTOP-8003BIM:/mnt/c/KP/MATERI/2.2 Federated Averaging Algorithm$
```

RANGKUMAN TASK :

Menggunakan clientnya 10 dan Epoch nya 5

```
95  NUM_CLIENTS = 10
96  NUM_EPOCHS = 5
97  BATCH_SIZE = 20
98  SHUFFLE_BUFFER = 100
99  PREFETCH_BUFFER = 10
100
```

```
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)])])
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/2.2 Federated Averaging Algorithm$
```

Menggunakan clientnya 20 dan Epoch nya 15

```
95  NUM_CLIENTS = 20
96  NUM_EPOCHS = 15
97  BATCH_SIZE = 20
98  SHUFFLE_BUFFER = 100
99  PREFETCH_BUFFER = 10
100
```

```
Skipping registering GPU devices...
round  1, metrics=OrderedDict([('distributor', ()), ('client_work', OrderedDict([('train', OrderedDict([('sparse_categorical_accuracy', 0.16037442), ('loss', 2.8771412), ('num_examples', 28845), ('num_batches', 1449)])])), ('aggregator', OrderedDict([('mean_value', ()), ('mean_weight', ())])), ('finalizer', OrderedDict([('update_non_finite', 0)])])
round  2, metrics=OrderedDict([('distributor', ()), ('client_work', OrderedDict([('train', OrderedDict([('sparse_categorical_accuracy', 0.21951811), ('loss', 2.522267), ('num_examples', 28845), ('num_batches', 1449)])])), ('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', 0.29596117), ('loss', 2.2035096), ('num_examples', 28845), ('num_batches', 1449)])])), ('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', 0.3659213), ('loss', 1.9442384), ('num_examples', 28845), ('num_batches', 1449)])])), ('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', 0.43716416), ('loss', 1.7289083), ('num_examples', 28845), ('num_batches', 1449)])])), ('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', 0.5008147), ('loss', 1.5534811), ('num_examples', 28845), ('num_batches', 1449)])])), ('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', 0.5555556), ('loss', 1.4097036), ('num_examples', 28845), ('num_batches', 1449)])])), ('aggregator', OrderedDict([('mean_value', ()), ('mean_weight', ())])), ('finalizer', OrderedDict([('update_non_finite', 0)])])
mples', 28845), ('num_batches', 1449)])])), ('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', 0.43716416), ('loss', 1.7289083), ('num_examples', 28845), ('num_batches', 1449)])])), ('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', 0.5008147), ('loss', 1.5534811), ('num_examples', 28845), ('num_batches', 1449)])])), ('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', 0.5555556), ('loss', 1.4097036), ('num_examples', 28845), ('num_batches', 1449)])])), ('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', 0.43716416), ('loss', 1.7289083), ('num_examples', 28845), ('num_batches', 1449)])])), ('aggregator', OrderedDict([('mean_value', ()), ('mean_weight', ())])), ('finalizer', OrderedDict([('update_non_finite', 0)])])
mples', 28845), ('num_batches', 1449)])])), ('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', 0.43716416), ('loss', 1.7289083), ('num_exmples', 28845), ('num_batches', 1449)])])), ('aggregator', OrderedDict([('mean_value', ()), ('mean_weight', ())])), ('finalizer', OrderedDict([('update_non_finite', 0)])])
mples', 28845), ('num_batches', 1449)])])), ('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', 0.43716416), ('loss', 1.7289083), ('num_examples', 28845), ('num_batches', 1449)])])), ('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', 0.5008147), ('loss', 1.5534811), ('num_examples', 28845), ('num_batches', 1449)])])), ('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', 0.5555556), ('loss', 1.4097036), ('num_examples', 28845), ('num_batches', 1449)])])), ('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', 0.60364014), ('loss', 1.2918327), ('num_examples', 28845), ('num_batches', 1449)])])), ('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', 0.64291906), ('loss', 1.1942939), ('num_examples', 28845), ('num_batches', 1449)])])), ('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', 0.67502165), ('loss', 1.1130261), ('num_examples', 28845), ('num_batches', 1449)])])), ('aggregator', OrderedDict([('mean_value', ()), ('mean_weight', ())])), ('finalizer', OrderedDict([('update_non_finite', 0)])])
(venv) ezranahumury@DESKTOP-8003BIM:/mnt/c/KP/MATERI/2.2 Federated Averaging Algorithm$
```

- **Run A**: 10 klien, 5 epoch/klien/round → akurasi naik pelan, loss turun tapi masih tinggi.
- **Run B**: 20 klien, 15 epoch/klien/round → akurasi naik jauh lebih cepat, loss turun lebih dalam dan stabil.

Run B tampak lebih bagus dikarenakan :

1. **Lebih banyak data per round**
   20 klien ⇒ agregasi gradien lebih mendekati populasi global ⇒ varians update lebih kecil ⇒ konvergensi lebih cepat.
2. **Lebih banyak langkah lokal per round**
   15 epoch vs 5 epoch ⇒ update lokal jauh lebih besar per round.
   Secara kasar, kamu mengalikan **komputasi per round ≈ 2× (klien) × 3× (epoch) = 6×**.