

Topik : 3.2. Membuat Model Custom (LogReg / NN)  
Objective : Desain model klasifikasi biner layak subsidi  
Task : Buat model di TensorFlow, wrap ke tff.learning.from\_keras\_model

Source : [https://www.tensorflow.org/federated/tutorials/custom\\_federated\\_algorithms\\_2](https://www.tensorflow.org/federated/tutorials/custom_federated_algorithms_2)

## Custom Federated Algorithms, Part 2: Implementing Federated Averaging

### Implementing Federated Averaging

Menggunakan contoh MNIST

#### 1. Preparing federated data sets

Mensimulasikan scenario Dimana kita memiliki data dari 10 pengguna, dan masing – masing pengguna memberikan pengetahuan tentang cara mengenali digit yang berbeda. Pertama, mari kita muat data MNIST standar :

```
7 mnist_train, mnist_test = tf.keras.datasets.mnist.load_data()
8 [(x.dtype, x.shape) for x in mnist_train]
```

Downloading data from https://storage.googleapis.com/tensorflow/tf-keras-datasets/mnist.npz  
11490434/11490434 [=====] - 3s 0us/step

```
[(dtype('uint8'), (60000, 28, 28)), (dtype('uint8'), (60000,))]
```

Data tersebut disajikan dalam bentuk array numpy, satu berisi gambar dan yang lain berisi label digit. Keduanya dengan dimensi pertama yang mencakup contoh – contoh individu. Mari kita tulis fungsi bantu yang memformat data tersebut agar compatible dengan cara kita memasukkan urutan federasi ke dalam perhitungan TFF, yaitu sebagai daftar dari daftar – daftar luar mencakup pengguna (digit), sedangkan daftar dalam mencakup batch data dalam urutan setiap klien.

Seperti biasa, kita akan mengstruktur setiap batch sebagai pasangan tensor bernama x dan y, masing-masing dengan dimensi batch terdepan. Selagi melakukannya, kita juga akan meratakan setiap gambar menjadi vektor berelemen 784 dan menskalakan piksel di dalamnya ke rentang 0..1, sehingga kita tidak perlu membebani logika model dengan konversi data.]

```
10 NUM_EXAMPLE_PER_USER = 1000
11 BATCH_SIZE = 100
12
13 def get_data_for_digit(source, digit):
14     output_sequence = []
15     all_samples = [i for i, d in enumerate(source[1]) if d == digit]
16     for i in range(0, min(len(all_samples), NUM_EXAMPLE_PER_USER), BATCH_SIZE):
17         batch_sample = all_samples[i:i + BATCH_SIZE]
18         output_sequence.append({
19             'x':
20                 np.array([source[0][i].flatten() / 255.0 for i in batch_sample], dtype=np.float32),
21             'y':
22                 np.array([source[1][i] for i in batch_sample], dtype=np.int32)
23         })
24     return output_sequence
25
26 federated_train_data = [get_data_for_digit(mnist_train, d) for d in range(10)]
27 federated_test_data = [get_data_for_digit(mnist_test, d) for d in range(10)]
```

```
29 print(federated_train_data[5][-1]['y'])
```

```
[5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5  
5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5  
5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5]
```

```
(venv) ezranahumury@DESKTOP-8003BIM:/mnt/c/KP/MATERI/3.2. Membuat Model Custom$
```

```
32 from matplotlib import pyplot as plt
33 plt.imshow(federated_train_data[5][:-1][:-1].reshape(28,28), cmap='gray')
34 plt.grid(False)
35 plt.savefig('hasil.jpg')
```



logika TensorFlow dapat dikembangkan dan diuji menggunakan praktik terbaik dan alat TF (seperti mode eager), sebelum menserialisasi komputasi untuk TFF (misalnya, dengan memanggil `tff.tensorflow.computation` dengan fungsi Python sebagai argumen).

### 3. Defining a loss function

Mendefinisikan fungsi kerugian yang dapat digunakan untuk pelatihan. Pertama, mari kita definisikan jenis\_input sebagai tuple Bernama TFF. Karena ukuran batch data dapat bervariasi, kita menetapkan dimensi batch menjadi None untuk menunjukkan bahwa ukuran dimensi ini tidak diketahui.

```
38 #Defining a loss function
39 BATCH_SPEC = collections.OrderedDict(
40     x=tf.TensorSpec(shape=[None,784], dtype=tf.float32),
41     y=tf.TensorSpec(shape=[None], dtype=tf.int32)
42 )
43
44 BATCH_TYPE = tff.types.StructType([
45     ('x', tff.types.TensorType(np.float32, [None, 784])),
46     ('y', tff.types.TensorType(np.int32, [None]))
47 ])
48
49 print(str(BATCH_TYPE))
```

```
<x=float32[?,784],y=int32[?]>
```

Simbol BATCH\_TYPE yang didefinisikan di atas mewakili spesifikasi tipe TFF abstrak. Penting untuk membedakan tipe TFF abstrak ini dari tipe representasi Python konkret, misalnya kontainer seperti dict atau collections.namedtuple yang mungkin digunakan untuk mewakili tipe TFF dalam tubuh fungsi Python.

Berbeda dengan python, TFF memiliki konstruktor tipe abstrak tunggal federated\_language.StructType untuk kontainer serupa tuple, dengan elemen yang dapat diberi nama secara individual atau dibiarkan tanpa nama. Tipe ini juga digunakan untuk memodelkan parameter formal perhitungan, karena perhitungan TFF secara formal hanya dapat mendeklarasikan satu parameter dan satu hasil.

Mendefinisikan tipe FF untuk parameter model, lagi – lagi sebagai tuple Bernama TFF dari bobot dan bias :

```
52 MODEL_SPEC = collections.OrderedDict(
53     weights=tf.TensorSpec(shape=[784,10], dtype=tf.float32),
54     bias=tf.TensorSpec(shape=[10], dtype=tf.float32)
55 )
56
57 MODEL_TYPE = tff.types.StructType([
58     ('weights', tff.types.TensorType(np.float32, [784,10])),
59     ('bias', tff.types.TensorType(np.float32, [10]))
60 ])
61 print(MODEL_TYPE)
```

```
<weights=float32[784,10],bias=float32[10]>
```

Mendefinisikan kerugian untuk suatu model pada satu batch. Perhatikan penggunaan decorator @tf.function didalam decorator @tff.tensorflow.computation. hal ini memungkinkan kita untuk menulis kode TF dengan semantic python meskipun berada dalam konteks tf.Graph yang dibuat oleh decorator tff.tensorflow.computation.

```

69 @tf.function
70 def forward_pass(model, batch):
71     predicted_y = tf.nn.softmax(
72         tf.matmul(batch['x'], model['weights']) + model['bias']
73     )
74
75     return -tf.reduce_mean(
76         tf.reduce_sum(
77             tf.one_hot(batch['y'], 10) * tf.math.log(predicted_y), axis=[1]
78         )
79     )
80
81 @tff.tensorflow.computation(MODEL_TYPE, BATCH_TYPE)
82 def batch_loss(model, batch):
83     return forward_pass(model, batch)

```

Seperti yang diharapkan, fungsi `batch_loss` mengembalikan nilai `float32` untuk kerugian (loss) berdasarkan model dan satu batch data. Perhatikan bahwa `MODEL_TYPE` dan `BATCH_TYPE` telah digabungkan menjadi tuple berukuran 2 sebagai parameter formal; Anda dapat mengenali tipe `batch_loss` sebagai `(<MODEL_TYPE, BATCH_TYPE> -> float32)`.

```

85 print(str(batch_loss.type_signature))

```

```

Skipping registering GPU devices...
(<model=<weights=float32[784,10],bias=float32[10]>,batch=<x=float32[?,784],y=int32[?]>> -> float32)
(venv) ezranahumury@DESKTOP-8003BIM:/mnt/c/KP/MATERI/3.2. Membuat Model Custom$

```

Sebagai Langkah pengecekan, mari kita bangun model awal yang diisi dengan nol dan hitung kerugian (loss) atas batch data yang telah divisualisasikan.

```

88 initial_model = collections.OrderedDict(
89     weights=np.zeros([784,10], dtype=np.float32),
90     bias=np.zeros([10], dtype=np.float32)
91 )
92 sample_batch = federated_train_data[5][-1]
93 print(batch_loss(initial_model,sample_batch))

```

```

Skipping registering GPU devices...
2.3025851
(venv) ezranahumury@DESKTOP-8003BIM:/mnt/c/KP/MATERI/3.2. Membuat Model Custom$

```

Memasukkan perhitungan TFF dengan model awal yang didefinisikan sebagai Dictionary (dict), meskipun tubuh fungsi python yang mendefinisikannya mengonsumsi parameter model sebagai `model['weight']` dan `model['bias']`. Argument panggilan `batch_loss` tidak langsung diteruskan ke tubuh fungsi tersebut.

Tubuh fungsi Python `batch_loss` telah ditelusuri dan diserialisasi di sel di atas tempat ia didefinisikan. TFF bertindak sebagai pemanggil `batch_loss` pada saat definisi komputasi, dan sebagai target panggilan saat `batch_loss` dipanggil. Dalam kedua peran tersebut, TFF bertindak sebagai jembatan antara sistem tipe abstrak TFF dan tipe representasi Python. Pada saat pemanggilan, TFF akan menerima sebagian besar tipe kontainer Python standar (dict, list, tuple, collections.namedtuple, dll.) sebagai representasi konkret dari tuple abstrak TFF. Selain itu, meskipun seperti yang disebutkan di atas, komputasi TFF secara formal hanya menerima satu parameter, Anda dapat menggunakan sintaks panggilan Python yang familiar dengan argumen posisional dan/atau kata kunci jika tipe parameter adalah tuple

#### 4. Gradient descent on a single batch

Mendefinisikan suatu perhitungan yang menggunakan fungsi kerugian ini untuk melakukan satu Langkah penurunan gradien. Perhatikan bahwa dalam mendefinisikan fungsi ini, kita menggunakan `batch_loss` sebagai subkomponen. Kita dapat memanggil perhitungan yang dibangun dengan `tff.tensorflow.computation` di dalam tubuh perhitungan lain, meskipun biasanya hal ini tidak diperlukan.

```
97 #Gradien descent on a single batch
98 from tensorflow_federated.python.common_libs import structure
99
100 @tff.tensorflow.computation(MODEL_TYPE, BATCH_TYPE, tf.float32)
101 def batch_train(initial_model, batch, learning_rate):
102     model_vars = collections.OrderedDict([
103         (name, tf.Variable(name=name, initial_value=value))
104         for name, value in structure.to_elements(initial_model)
105     ])
106     optimizer = tf.keras.optimizers.SGD(learning_rate)
107
108     @tf.function
109     def _train_on_batch(model_vars, batch):
110         with tf.GradientTape() as tape:
111             loss = forward_pass(model_vars, batch)
112             grads = tape.gradient(loss, model_vars)
113             optimizer.apply_gradients(
114                 zip(tf.nest.flatten(grads), tf.nest.flatten(model_vars))
115             )
116         return model_vars
117
118     return _train_on_batch(model_vars, batch)
119
120 print(str(batch_train.type_signature))
121
```

Skipping registering GPU devices...  
(<initial\_model=<weights=float32[784,10],bias=float32[10]>,batch=<x=float32[?,784],y=int32[?]>,learning\_rate=float32> -> <weights=float32[784,10],bias=float32[10]>)  
(venv) ezranahumury@DESKTOP-8003BIM:/mnt/c/KP/MATERI/3.2. Membuat Model Custom\$

Ketika kita memanggil fungsi python yang didekorasi dengan `tff.tensorflow.computation` didalam tubuh fungsi lain yang serupa, logika perhitungan TFF bagian dalam akses tertanam (secara esensial, di inline) ke dalam logika perhitungan bagian luar.

jika Anda menulis kedua komputasi tersebut, kemungkinan lebih baik membuat fungsi dalam (`batch_loss` dalam hal ini) sebagai fungsi Python biasa atau `tf.function` daripada `tff.tensorflow.computation`. Namun, di sini kami menunjukkan bahwa memanggil satu `tff.tensorflow.computation` di dalam yang lain pada dasarnya berfungsi seperti yang diharapkan. Hal ini mungkin diperlukan jika, misalnya, Anda tidak memiliki kode Python yang mendefinisikan `batch_loss`, tetapi hanya representasi TFF yang diserialisasikan.

Menerapkan fungsi beberapa kali pada model awal untuk melihat apakah kerugian berkurang.

```
123 model = initial_model
124 losses = []
125 for _ in range(5):
126     model = batch_train(model, sample_batch, 0.1)
127     losses.append(batch_loss(model, sample_batch))
128
129 print(losses)
```

Skipping registering GPU devices...  
[0.19690025, 0.13176318, 0.101132266, 0.08273812, 0.07030139]  
(venv) ezranahumury@DESKTOP-8003BIM:/mnt/c/KP/MATERI/3.2. Membuat Model Custom\$

## 5. Gradient descent on a sequence of local data

Karena `batch_train` tampaknya berfungsi, maka fungsi pelatihan serupa Bernama `local_train` yang mengolah seluruh urutan batch dari satu pengguna, bukan hanya satu batch. Perhitungan baru ini sekarang perlu menggunakan `federated_language.SequenceType(BATCH_TYPE)`.

```
132 #Gradient descent on a sequence of local data
133 LOCAL_DATA_TYPE = tff.SequenceType(BATCH_TYPE)
134
135 @tff.federated_computation(MODEL_TYPE, np.float32, LOCAL_DATA_TYPE)
136 def local_train(initial_model, learning_rate, all_batches):
137
138     # Reduction function to apply to each batch.
139     @tff.federated_computation((MODEL_TYPE, np.float32), BATCH_TYPE)
140     def batch_fn(model_with_lr, batch):
141         model, lr = model_with_lr
142         return batch_train(model, batch, lr), lr
143
144     trained_model, _ = tff.sequence_reduce(
145         all_batches, (initial_model, learning_rate), batch_fn
146     )
147     return trained_model
148 print(str(local_train.type_signature))
```

```
Skipping registering GPU devices...
<initial_model=<weights=float32[784,10],bias=float32[10]>,learning_rate=float32,all_batches=<x=float32[?,784],y=int32[?]>*> -> <weights=float32[784,10],bias=float32[10]>>
(venv) ezranahumury@DESKTOP-8003B1M: /mnt/c/KP/MATERI/3.2. Membuat Model Custom$
```

Pertama, meskipun kita dapat mengimplementasikan logika ini sepenuhnya di TensorFlow dengan mengandalkan `tf.data.Dataset.reduce` untuk memproses urutan data serupa dengan yang telah kita lakukan sebelumnya, kali ini kita memilih untuk mengekspresikan logika tersebut dalam bahasa pemersatu (glue language) sebagai `federated_language.federated_computation`. Kita menggunakan operator `federated_language.sequence_reduce` untuk melakukan reduksi.

Operator `federated_language.sequence_reduce` digunakan serupa dengan `tf.data.Dataset.reduce`. Anda dapat memikirkannya sebagai hal yang pada dasarnya sama dengan `tf.data.Dataset.reduce`, tetapi untuk digunakan di dalam komputasi federasi, yang seperti yang mungkin Anda ingat, tidak boleh mengandung kode TensorFlow. Ini adalah operator templat dengan parameter formal berupa tuple 3-elemen yang terdiri dari urutan elemen bertipe `T`, keadaan awal reduksi (kami akan merujuknya secara abstrak sebagai `u0`) bertipe `U`, dan operator reduksi bertipe `(U,T) -> U` yang mengubah keadaan reduksi dengan memproses satu elemen. Hasilnya adalah keadaan akhir reduksi setelah memproses semua elemen secara berurutan. Dalam contoh kita, keadaan reduksi adalah model yang dilatih pada prefiks data, dan elemen-elemennya adalah batch data.

Kedua, perhatikan bahwa kami kembali menggunakan satu komputasi (`batch_train`) sebagai komponen dalam komputasi lain (`local_train`), tetapi tidak secara langsung. Kami tidak dapat menggunakannya sebagai operator reduksi karena komputasi ini memerlukan parameter tambahan—laju pembelajaran. Untuk mengatasi hal ini, kami mendefinisikan perhitungan federasi tertanam `batch_fn` yang mengikat parameter `learning_rate` dari `local_train` dalam tubuhnya. Diperbolehkan bagi perhitungan anak yang didefinisikan dengan cara ini untuk menangkap parameter

formal dari orang tuanya selama perhitungan anak tidak dipanggil di luar tubuh orang tuanya. Anda dapat memikirkan pola ini sebagai setara dengan `functools.partial` di Python.

Implikasi praktis dari menangkap `learning_rate` dengan cara ini, tentu saja, adalah bahwa nilai `learning_rate` yang sama digunakan di seluruh batch.

Mencoba fungsi pelatihan local yang baru didefinisikan pada seluruh urutan data dari pengguna yang sama yang menyumbangkan batch sampel (digit 5)

```
150 locally_trained_model = local_train(initial_model, 0.1, federated_train_data[5])
151 print(locally_trained_model)
152
```

```
Skipping registering GPU devices...
OrderedDict([('weights', array([[0., 0., 0., ..., 0., 0., 0.],
 [0., 0., 0., ..., 0., 0., 0.],
 [0., 0., 0., ..., 0., 0., 0.],
 ...,
 [0., 0., 0., ..., 0., 0., 0.],
 [0., 0., 0., ..., 0., 0., 0.],
 [0., 0., 0., ..., 0., 0., 0.]]), dtype=float32)), ('bias', array([-0.01939448, -0.01939448, -0.01939448, -0.01939448, -0.01939448,
 0.17455032, -0.01939448, -0.01939448, -0.01939448, -0.01939448],
 dtype=float32))])
(env) ezranahumury@DESKTOP-8003BIM:/mnt/c/KP/MATERI/3.2. Membuat Model Custom$
```

## 6. Local Evaluation

Salah satu cara untuk menerapkan evaluasi local dengan menjumlahkan kerugian di seluruh batch data (kita juga bisa menghitung rata – ratanya)

```
156 #Local Evaluation
157 @tff.federated_computation(MODEL_TYPE, LOCAL_DATA_TYPE)
158 def local_eval(model, all_batches):
159
160     @tff.tensorflow.computation((MODEL_TYPE, np.float32), BATCH_TYPE)
161     def accumulate_evaluation(model_and_accumulator, batch):
162         model, accumulator = model_and_accumulator
163         return model, accumulator + batch_loss(model, batch)
164
165     _, total_loss=tff.sequence_reduce(
166         all_batches, (model,0.0), accumulate_evaluation
167     )
168     return total_loss
169 print(str(local_eval.type_signature))
170
```

```
Skipping registering GPU devices...
(<model=<weights=float32[784,10],bias=float32[10]>,all_batches=<x=float32[?,784],y=int32[?]>*> -> float32)
(env) ezranahumury@DESKTOP-8003BIM:/mnt/c/KP/MATERI/3.2. Membuat Model Custom$
```

Pertama, kita telah menggunakan dua operator federated baru untuk memproses urutan: `tff.sequence_map` yang menerima fungsi pemetaan  $T \rightarrow U$  dan urutan  $T$ , lalu menghasilkan urutan  $U$  yang diperoleh dengan menerapkan fungsi pemetaan secara titik demi titik, dan `tff.sequence_sum` yang hanya menjumlahkan semua elemen. Di sini, kita memetakan setiap batch data ke nilai kerugian, lalu menjumlahkan nilai kerugian yang dihasilkan untuk menghitung kerugian total.

Perlu dicatat bahwa kita bisa saja menggunakan `tff.sequence_reduce`, tetapi ini bukan pilihan terbaik - proses reduksi, secara definisi, bersifat sequential, sedangkan pemetaan dan penjumlahan dapat dihitung secara paralel. Ketika diberi pilihan, sebaiknya menggunakan operator yang tidak membatasi pilihan implementasi, sehingga ketika perhitungan TFF dikompilasi di masa depan untuk diimplementasikan di lingkungan spesifik, kita dapat memanfaatkan sepenuhnya semua peluang untuk eksekusi yang lebih cepat, lebih skalabel, dan lebih efisien sumber daya.



Kedua, perhatikan bahwa sama seperti di `local_train`, fungsi komponen yang kita butuhkan (`batch_loss`) memerlukan lebih banyak parameter daripada yang diharapkan oleh operator `federated` (`tff.sequence_map`), jadi kita kembali mendefinisikan fungsi parsial, kali ini secara inline dengan membungkus `lambda` langsung sebagai `tff.federated_computation`. Menggunakan wrapper secara inline dengan fungsi sebagai argumen adalah cara yang direkomendasikan untuk menggunakan `tff.tensorflow.computation` guna menyematkan logika TensorFlow dalam TFF.

```
172 print("Initial_model loss = ", local_eval(initial_model,federated_train_data[5]))
173 print("locally_trained_model loss = ", local_eval(locally_trained_model, federated_train_data[5]))
```

```
Skipping registering GPU devices...
Initial_model loss = 23.025854
Locally_trained_model loss = 0.43484676
(venv) ezranahumury@DESKTOP-8003BIM:/mnt/c/KP/MATERI/3.2. Membuat Model Custom$
```

Memang, kerugiannya berkurang. Tapi apa yang terjadi jika kita mengevaluasinya menggunakan data pengguna lain ?

```
175 print("Initial_model loss = ", local_eval(initial_model,federated_train_data[0]))
176 print("locally_trained_model loss = ", local_eval(locally_trained_model, federated_train_data[0]))
```

```
Skipping registering GPU devices...
Initial_model loss = 23.025854
Locally_trained_model loss = 74.50075
(venv) ezranahumury@DESKTOP-8003BIM:/mnt/c/KP/MATERI/3.2. Membuat Model Custom$
```

Seperti yang diharapkan, situasinya semakin memburuk. Model tersebut dilatih untuk mengenali angka 5 dan belum pernah melihat angka 0.

## 7. Federated Evaluation

Sepasang definisi tipe TFF untuk model yang berasal dari server dan data yang tetap berada di klien.

```
181 SERVER_MODEL_TYPE = tff.FederatedType(MODEL_TYPE, tff.SERVER)
182 CLIENT_DATA_TYPE = tff.FederatedType(LOCAL_DATA_TYPE, tff.CLIENTS)
```

Mengimplementasikan evaluasi federasi dalam TFF dapat dilakukan dengan satu baris kode yang kita distribusikan modelnya ke klien, membiarkan setiap klien menjalankan evaluasi lokal pada data bagian lokalnya, dan kemudian menghitung rata – rata kerugian.

```
184 @tff.federated_computation(SERVER_MODEL_TYPE, CLIENT_DATA_TYPE)
185 def federated_eval(model, data):
186     return tff.federated_mean(
187         tff.federated_map(local_eval, [tff.federated_broadcast(model), data])
188     )
```

Pertama, mari kita uraikan bagian **“setiap klien menjalankan evaluasi lokal pada bagiannya masing-masing dari data”**. Seperti yang mungkin Anda ingat dari bagian sebelumnya, `local_eval` memiliki *type signature* dalam bentuk `(<MODEL_TYPE, LOCAL_DATA_TYPE> -> float32)`.



Operator federated **tff.federated\_map** adalah sebuah template yang menerima parameter berupa *2-tuple* yang terdiri dari fungsi pemetaan bertipe  $T \rightarrow U$  dan sebuah nilai federated bertipe  $\{T\}@CLIENTS$  (yaitu, dengan setiap anggota memiliki tipe yang sama seperti parameter dari fungsi pemetaan), lalu mengembalikan hasil bertipe  $\{U\}@CLIENTS$ .

Karena kita memberikan `local_eval` sebagai fungsi pemetaan yang akan diterapkan pada masing-masing klien, argumen kedua seharusnya bertipe federated  $\{<MODEL\_TYPE, LOCAL\_DATA\_TYPE>\}@CLIENTS$ , yaitu dalam istilah bagian sebelumnya, sebuah *federated tuple*. Setiap klien seharusnya memiliki satu set lengkap argumen untuk `local_eval` sebagai anggota. Namun, yang kita berikan justru berupa *Python list* dengan 2 elemen.

mirip dengan *implicit cast* yang mungkin pernah Anda temui di tempat lain, misalnya ketika Anda memberikan sebuah `int` ke fungsi yang menerima `float`. **Implicit casting** di TFF saat ini masih jarang digunakan, tetapi rencananya akan lebih diperluas agar bisa meminimalkan kode boilerplate.

*Implicit cast* yang diterapkan di sini adalah kesetaraan antara federated tuple dengan bentuk  $\{<X,Y>\}@Z$  dan tuple dari federated values  $\{<X>\}@Z, \{<Y>\}@Z$ . Secara formal, kedua *type signature* ini berbeda, tetapi dari sudut pandang programmer, setiap perangkat dalam himpunan  $Z$  sama-sama memegang dua unit data  $X$  dan  $Y$ . Apa yang terjadi di sini mirip dengan fungsi **zip** di Python, dan memang TFF menyediakan operator **tff.federated\_zip** untuk melakukan konversi seperti ini secara eksplisit. Ketika `tff.federated_map` menemukan tuple sebagai argumen kedua, ia secara otomatis memanggil `tff.federated_zip` untuk Anda.

Mengenali ekspresi `tff.federated_broadcast(model)` sebagai sebuah nilai dengan tipe TFF  $\{MODEL\_TYPE\}@CLIENTS$ , dan data sebagai sebuah nilai dengan tipe TFF  $\{LOCAL\_DATA\_TYPE\}@CLIENTS$  (atau singkatnya `CLIENT_DATA_TYPE`). Keduanya kemudian digabungkan melalui *implicit* `tff.federated_zip` untuk membentuk argumen kedua dari `tff.federated_map`.

Operator `tff.federated_broadcast`, hanya mentransfer data dari server ke para klien. Mari kita lihat bagaimana pelatihan local memengaruhi rata – rata loss dalam system.

```
191 print("Initial_model loss = ", federated_eval(initial_model,federated_train_data))
192 print("locally_trained_model loss = ", federated_eval(locally_trained_model, federated_train_data))
```

  

```
Skipping registering GPU devices...
Initial_model loss = 23.025852
locally_trained_model loss = 54.43263
(venv) ezranahumury@DESKTOP-8003BIM:/mnt/c/KP/MATERI/3.2. Membuat Model Custom$
```

Kerugian telah meningkat, . Untuk meningkatkan model bagi semua pengguna, kita perlu melatihnya menggunakan data dari semua orang.

## 8. Federated Training

Cara termudah untuk menerapkan pelatihan federasi Adalah dengan melatih model secara local, lalu menggabungkan hasilnya. Metode ini menggunakan blok bangunan atau pola yang sama.

```
195 # Federated Training
196 SERVER_FLOAT_TYPE = tff.FederatedType(np.float32, tff.SERVER)
197
198 @tff.federated_computation(SERVER_MODEL_TYPE, SERVER_FLOAT_TYPE, CLIENT_DATA_TYPE)
199 def federated_train(model, learning_rate, data):
200     return tff.federated_mean(
201         tff.federated_map(local_train, [
202             tff.federated_broadcast(model),
203             tff.federated_broadcast(learning_rate), data
204         ])
205     )
```

Implementasi Federated Averaging yang lengkap yaitu yang disediakan oleh `tff.learning`, dari pada mengumpulkan rata – rata model, kita lebih memilih untuk mengumpulkan rata – rata delta model karena beberapa alasan, misalnya kemampuan untuk membatasi norma pembaruan, untuk kompresi dan sebagainya.

Pelatihan berfungsi dengan menjalankan putaran pelatihan dan membandingkan rata – rata kerugian sebelum dan sesudah.

```
207 model = initial_model
208 learning_rate = 0.1
209 for round in range(5):
210     model = federated_train(model, learning_rate, federated_train_data)
211     learning_rate = learning_rate * 0.9
212     loss = federated_eval(model, federated_train_data)
213     print('round {}, loss={}'.format(round, loss))
```

```
you would like to use GPU. Follow the guide at https://www.tensorflow.org/install/gpu
Skipping registering GPU devices...
round 0, loss=21.60552406311035
round 1, loss=20.365678787231445
round 2, loss=19.27480125427246
round 3, loss=18.31110954284668
round 4, loss=17.45725440979004
(venv) ezranahumury@DESKTOP-8003BIM:/mnt/c/KP/MATERI/3.2. Membuat Model Custom$
```

Mari kita jalankan data uji untuk memastikan bahwa model kita dapat melakukan generalisasi dengan baik .

```
216 print(
217     'initial_model test loss =',
218     federated_eval(initial_model, federated_test_data),
219 )
220 print('trained_model test loss =', federated_eval(model, federated_test_data))
```

```
Skipping registering GPU devices...
initial_model test loss = 22.795593
trained_model test loss = 20.101158
(venv) ezranahumury@DESKTOP-8003BIM:/mnt/c/KP/MATERI/3.2. Membuat Model Custom$
```

## Task :

Buat model di TensorFlow, wrap ke `tff.learning.from_keras_model`

### 1. Model Di Tensorflow : terdapat 2 opsi

#### a. Logistic regression

- Input → langsung masuk ke layer `Dense(1, sigmoid)`.
- Fungsinya: menghitung probabilitas kelas 1 (layak subsidi).

```
61 # ===== Logistic Regression Model =====
62 def create_logreg_model(input_dim):
63     return tf.keras.Sequential([
64         tf.keras.layers.Input(shape=(input_dim,)),
65         tf.keras.layers.Dense(1, activation='sigmoid')
66     ])
```

#### b. Neural Network

- Ada hidden layer ReLU untuk menambah kemampuan belajar.

```
58 # ===== Neural Network Model =====
59 def create_nn_model(input_dim):
60     return tf.keras.Sequential([
61         tf.keras.layers.Input(shape=(input_dim,)),
62         tf.keras.layers.Dense(16, activation='relu'),
63         tf.keras.layers.Dense(8, activation='relu'),
64         tf.keras.layers.Dense(1, activation='sigmoid')
65     ])
66
```

### 2. Wrap Ke TFF

Supaya model bisa dipakai dalam **Federated Learning**, kita bungkus dengan `tff.learning.models.from_keras_model`.

```
67 # ===== Wrap ke TFF =====
68 def model_fn():
69     input_dim = input_spec[0].shape[-1]
70
71     # pilih model
72     keras_model = create_nn_model(input_dim) # NN kecil
73     # keras_model = create_logreg_model(input_dim) # Logistic Regression
74
75     return tff.learning.models.from_keras_model(
76         keras_model=keras_model,
77         input_spec=input_spec,
78         loss=tf.keras.losses.BinaryCrossentropy(),
79         metrics=[tf.keras.metrics.BinaryAccuracy(),
80                 tf.keras.metrics.AUC()]
81     )
82
```

- keras\_model → model biner yang dibuat dengan Tensorflow (LogReg/NN)
- input\_spec → memberitahu TFF bentuk (x,y) dari dataset (jumlah fitur, tipe data)
- loss → pakai BinaryCrossentropy karena klasifikasi 0/1.
- Metrics → evaluasi akurasi (BinaryAccuracy) dan kualitas pemisahan kelas (AUC).

Model dibuat menggunakan Keras seperti biasa, namun karena TensorFlow Federated (TFF) tidak bisa langsung memakai model Keras, maka model tersebut harus dibungkus dengan `tff.learning.models.from_keras_model`; pada tahap pembungkusan ini, TFF diberi informasi mengenai bentuk data masukan melalui `input_spec`, fungsi loss yang digunakan untuk menghitung error (misalnya BinaryCrossentropy untuk klasifikasi biner), serta metrik evaluasi (seperti BinaryAccuracy atau AUC) yang digunakan untuk memantau performa model selama proses federated learning.

OUTPUT :

Menggunakan Federated Averaging :

```

83 # ===== Federated Averaging =====
84 iterative_process = tff.learning.algorithms.build_weighted_fed_avg(
85     model_fn,
86     client_optimizer_fn=tff.learning.optimizers.build_adam(learning_rate=0.01),
87     server_optimizer_fn=tff.learning.optimizers.build_sgdm(learning_rate=1.0)
88 )
89
90 state = iterative_process.initialize()
91
92 # ===== Training Loop =====
93 NUM_ROUNDS = 10
94 for round_num in range(1, NUM_ROUNDS + 1):
95     result = iterative_process.next(state, federated_train_data)
96     state = result.state
97
98     # ambil metrik dari client_work → train
99     train_metrics = result.metrics['client_work']['train']
100     loss = float(train_metrics['loss'])
101     acc = float(train_metrics['binary_accuracy'])
102
103     print(f"Round {round_num} - Loss: {loss:.4f}, Accuracy: {acc:.4f}")
104

```

```

Round 1 - Loss: 0.6725, Accuracy: 0.6060
Round 2 - Loss: 0.6504, Accuracy: 0.6360
Round 3 - Loss: 0.6419, Accuracy: 0.6560
Round 4 - Loss: 0.6460, Accuracy: 0.6420
Round 5 - Loss: 0.6395, Accuracy: 0.6540
Round 6 - Loss: 0.6252, Accuracy: 0.6580
Round 7 - Loss: 0.6195, Accuracy: 0.6700
Round 8 - Loss: 0.6097, Accuracy: 0.6900
Round 9 - Loss: 0.6120, Accuracy: 0.6800
Round 10 - Loss: 0.6071, Accuracy: 0.6840
(venv) ezranahumury@DESKTOP-8003BIM:/mnt/c/KP/MATERI/3.2. Membuat Model Custom$
0 5

```