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>

**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 :

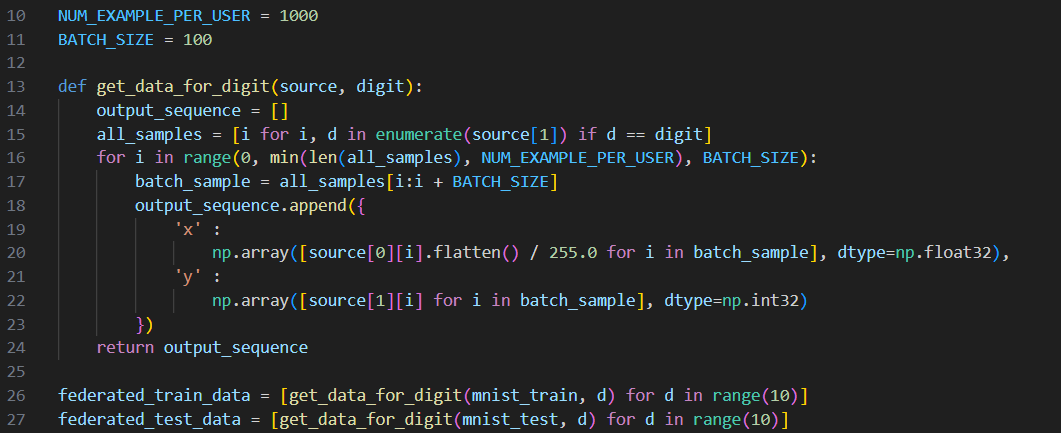






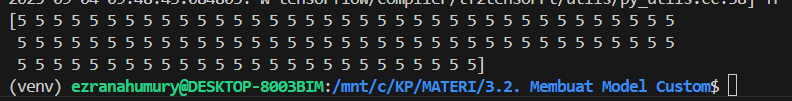
Data tersebut disajikan dalam bentuk array numpy, satu berisi gambar dan yang lain berisi label digit. Keduanya dengan dimensi pertam yang mencakup contoh – contoh individu. Mari kita tulis fungsi bantu yang memformat data tersebut agar compatible dengan car akita kita memasukkan urutan federasi ke dalam 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.]

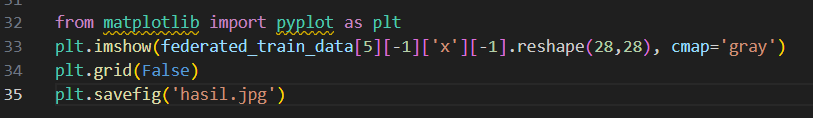


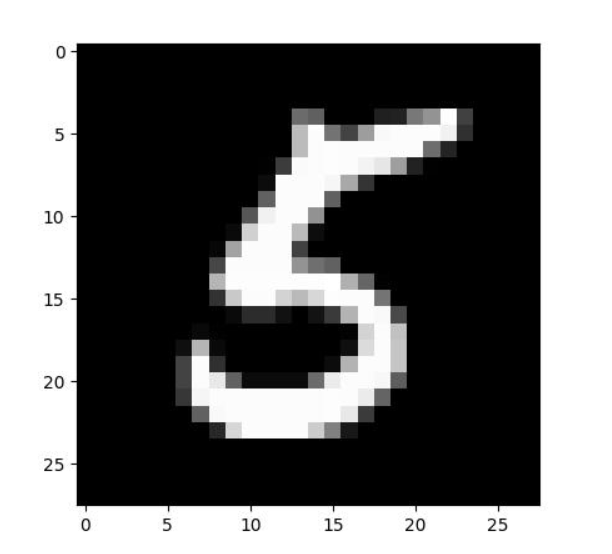
Sebagai pemeriksaan cepat, mari kita lihat tensor Y pada batch data terakhir yang disumbangkan oleh klien kelima ( yang sesuai dengan angka 5)





Hanya untuk memastikam mari kita juga lihat gambar yang sesuai dengan element terakhir dari batch tersebut.





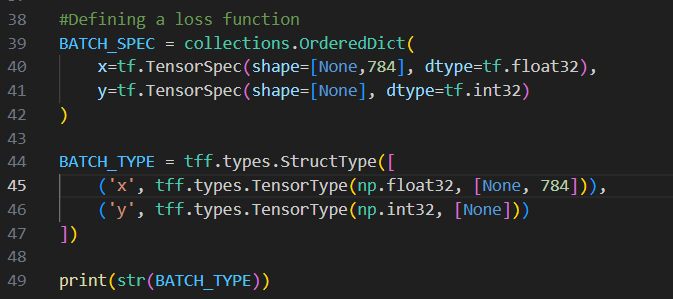
1. On Combining Tensorflow and TFF

Menambahkan dekorasi tff.tensorflow.computation pada fungsi yang memperkenalkan logika tensorflow. Debugging TensorFlow sudah menjadi tantangan, dan debugging TensorFlow setelah sepenuhnya diserialisasi dan kemudian diimpor kembali secara otomatis kehilangan beberapa metadata dan membatasi interaktivitas, sehingga membuat debugging menjadi lebih menantang.

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).

1. Defining a loss function

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

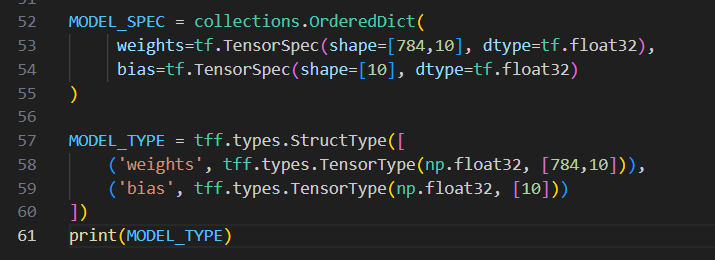




Simbol BATCH\_TYPE yang didefinisikan di atas mewakili spesifikasi tipe TFF abstrak. Penting untuk membedakan tipe TFF abstrak ini dari tipe representasi Python konkret, misalnya kontainer seperti dict atau collections.namedtuple yang mungkin digunakan untuk mewakili tipe TFF dalam tubuh fungsi Python.

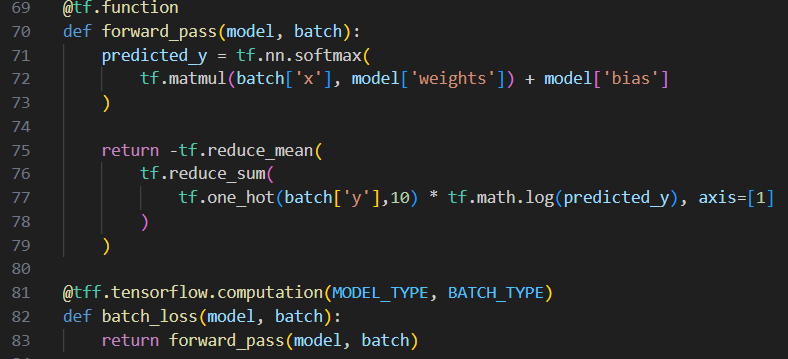
Berbeda dengan python, TFF memiliki konstruktor tipe abstrak tunggal federated\_language.StructType untuk kontainer serupa tuple, dengan elemen yang dapat diberi nama secara individual atau dibiarkan tanpa nama. Tipe ini juga digunakan untuk memodelkan parameter formal perhitungan, karena perhitungan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





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.

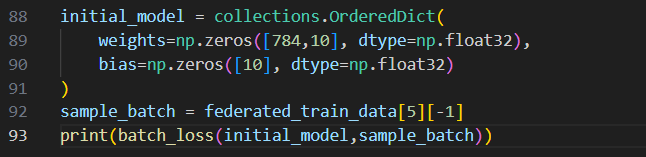


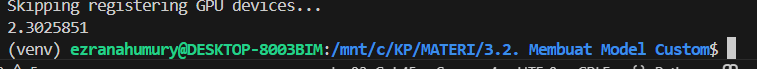
Seperti yang diharapkan, fungsi batch\_loss mengembalikan nilai float32 untuk kerugian (loss) berdasarkan model dan satu batch data. Perhatikan bahwa MODEL\_TYPE dan BATCH\_TYPE telah digabungkan menjadi tuple berukuran 2 sebagai parameter formal; Anda dapat mengenali tipe batch\_loss sebagai (<MODEL\_TYPE,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





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



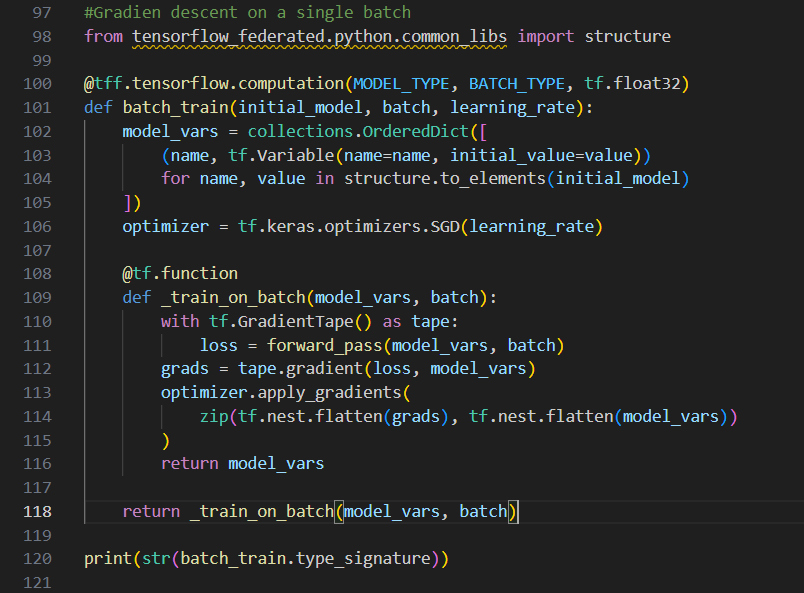


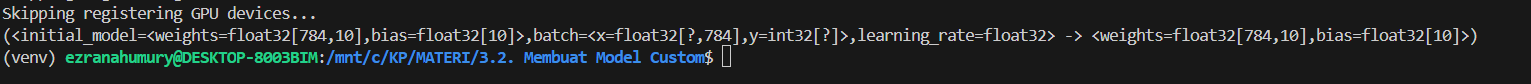
Memasukkan perhitungan TFF dengan model awal yang didefinisikan sebagai Dictionary (dict), meskipun tubuh fungsi python yang mendefinisikannya mengonsumsi parameter model sebagai model[‘weight’] dan model[‘bias’]. Argument panggilan batch\_loss tidak langsung diteruskan ke tubuh fungsi tersebut.

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

1. Gradient descent on a single batch

Mendefinisikan suatu perhitungana 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.

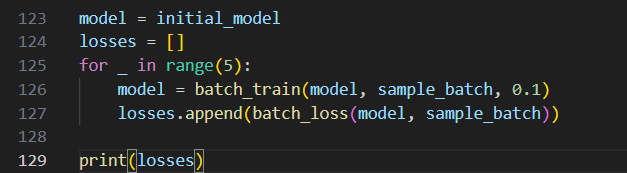


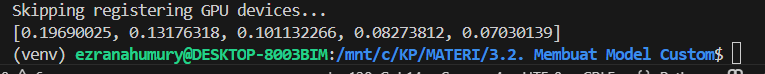


Ketika kita memanggil fungsi python yang didekorasi dengan tff.tensorflow.comutation didalam tubuh fungsi lain yang serupa, logika perhitungan TFF bagian dalam akses tertanam (secara esensial, di inline) ke dalam logika perhitungan bagian luar.

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.

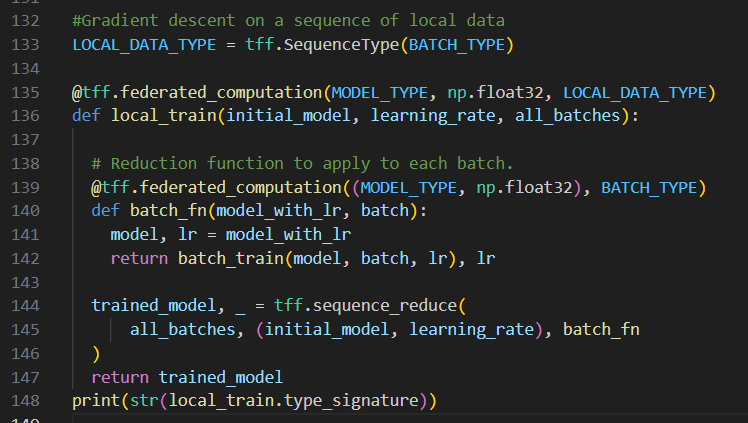
Menerapkan fungsi beberapa kali pada model awal untuk melihat apakah kerugian berkurang.

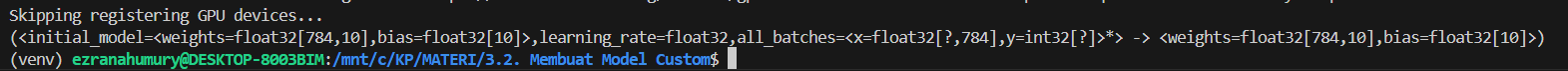




1. Gradient descent on a sequence of local data

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) .





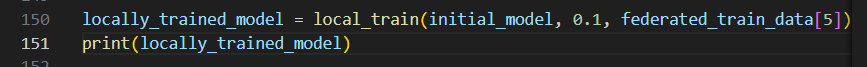
Pertama, meskipun kita dapat mengimplementasikan logika ini sepenuhnya di TensorFlow dengan mengandalkan tf.data.Dataset.reduce untuk memproses urutan data serupa dengan yang telah kita lakukan sebelumnya, kali ini kita memilih untuk mengekspresikan logika tersebut dalam bahasa pemersatu (glue language)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

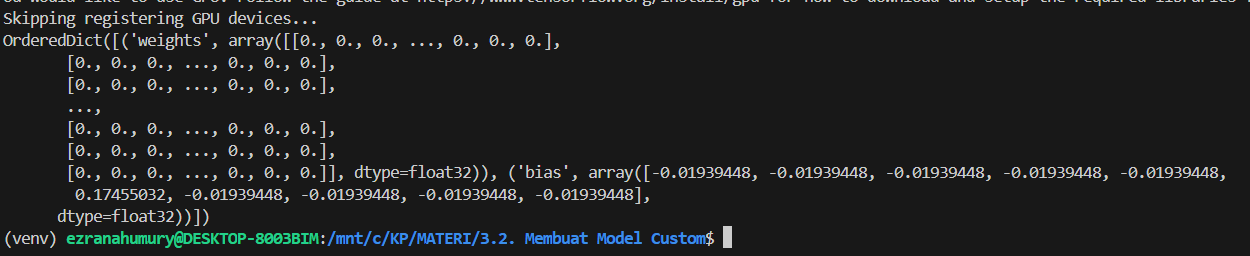
Operator federated\_language.sequence\_reduce digunakan serupa dengan tf.data.Dataset.reduce. Anda dapat memikirkannya sebagai hal yang pada dasarnya sama dengan tf.data.Dataset.reduce, tetapi untuk digunakan di dalam komputasi federasi, yang seperti yang mungkin Anda ingat, tidak boleh mengandung kode 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 nol) 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.

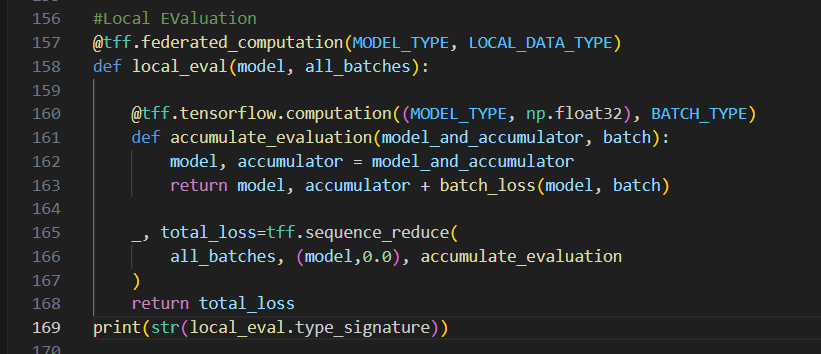
Mencoba fungsi pelatihan local yang baru didefinisikan pada seluruh urutan data dari pengguna yang sama yang menyumbangkan batch sampel (digit 5)

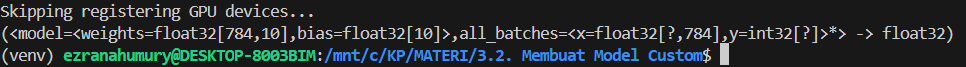




1. Local Evaluation

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



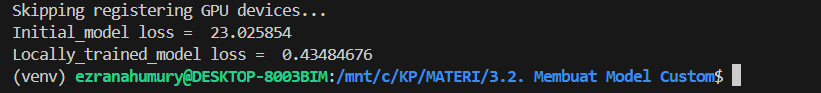


Pertama, kita telah menggunakan dua operator federated baru untuk memproses urutan: tff.sequence\_map yang menerima fungsi pemetaan T->U dan urutan T, lalu menghasilkan urutan U yang diperoleh dengan menerapkan fungsi pemetaan secara titik demi titik, dan tff.sequence\_sum yang hanya menjumlahkan semua 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.

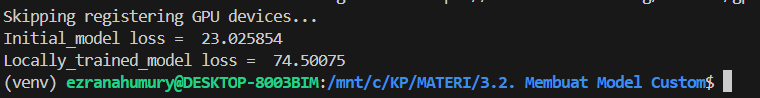
Kedua, perhatikan bahwa sama seperti di local\_train, fungsi komponen yang kita butuhkan (batch\_loss) memerlukan lebih banyak parameter daripada yang diharapkan oleh operator federated (tff.sequence\_map), jadi kita kembali mendefinisikan fungsi parsial, kali ini secara inline dengan membungkus lambda 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.





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





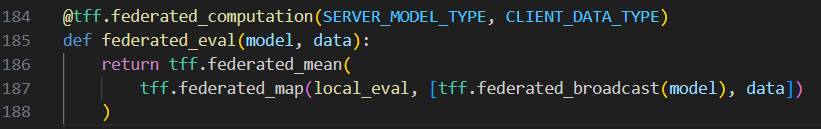
Seperti yang diharapkan, situasinya semakin memburuk. Model tersebut dilatih untuk mengenali angka 5 dan belum pernah melihat angka 0.

1. Federated Evaluation

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



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



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->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.

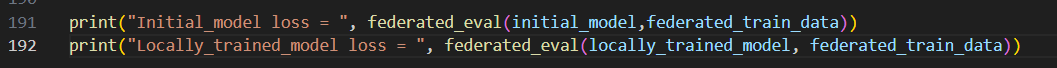
Karena kita memberikan local\_eval sebagai fungsi pemetaan yang akan diterapkan pada masing-masing klien, argumen kedua seharusnya bertipe federated {<MODEL\_TYPE, LOCAL\_DATA\_TYPE>}@CLIENTS, yaitu dalam istilah bagian sebelumnya, sebuah *federated tuple*. Setiap klien seharusnya memiliki satu set lengkap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

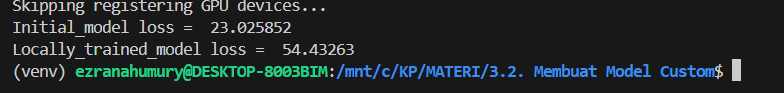
mirip dengan *implicit cast* yang mungkin pernah Anda temui di tempat lain, misalnya ketika Anda memberikan sebuah int ke fungsi yang menerima float. **Implicit casting** di TFF saat ini masih jarang digunakan, tetapi rencananya akan lebih diperluas agar bisa meminimalkan kode boilerplate.

*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.

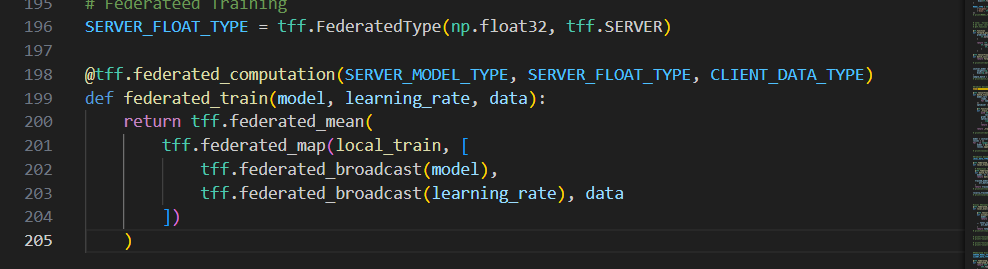




Kerugian telah meningkat, . Untuk meningkatkan model bagi semua pengguna, kita perlu melatihnya menggunakan data dari semua orang.

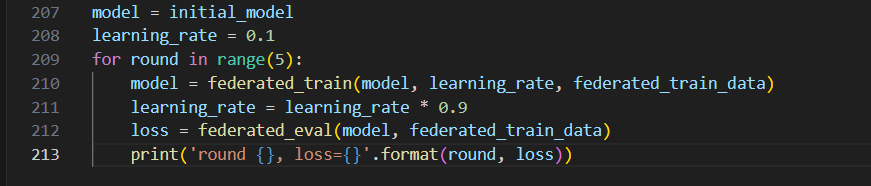
1. Federated Training

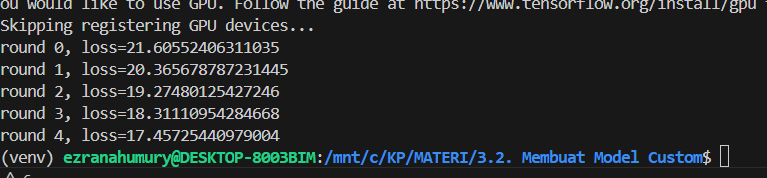
Cara termudah untuk menerapkan pelatihan federasi Adalah dengan melatih model secara local, lalu menggabungkan hasilnya. Metode ini menggunakan blok bangunan atau pola yang sama.



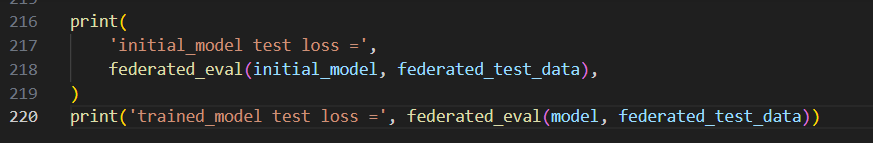
Implementasi Federated Averaging yang lengkap yaitu yang disediakan oleh tff.learning, dari pada mengumpulkan rata – rata model, kita lebih memilih untuk mengumpulkan rata – rata delta model karena beberapa alasan, misalnya kemampuan untuk membatasi norma pembaruan, untuk kompresi dan sebagainya.

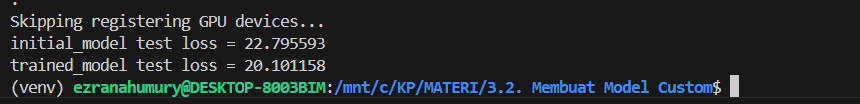
Pelatihan berfungsi dengan menjalankan putaran pelatihan dan membandingkan rata – rata kerugian sebelum dan sesudah.





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



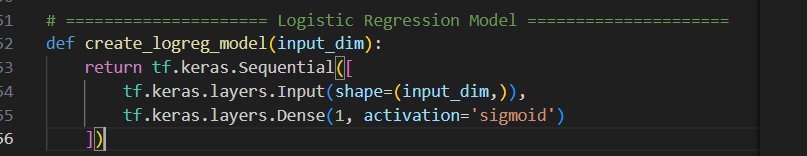


**Task :**

Buat model di TensorFlow, wrap ke tff.learning.from\_keras\_model

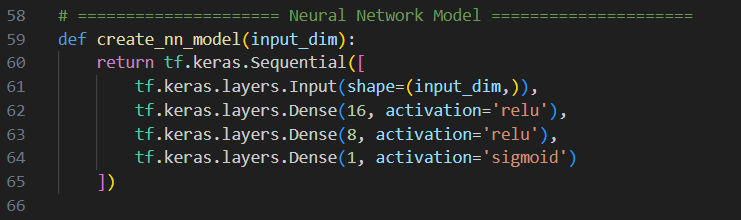
1. Model Di Tensorflow : terdapat 2 opsi
2. Logistic regression

* Input 🡪 langsung masuk ke layer Dense(1, sigmoid).
* Fungsinya: menghitung probabilitas kelas 1 (layak subsidi).



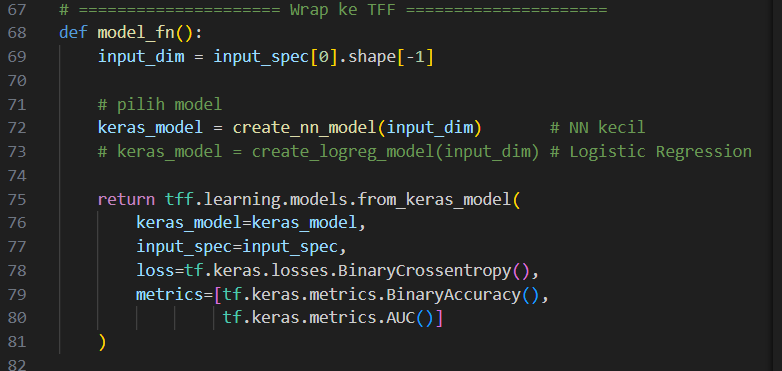
1. Neural Network

* Ada hidden layer ReLU untuk menambah kemampuan belajar.



1. Wrap Ke TFF

Supaya model bisa dipakai dalam **Federated Learning**, kita bungkus dengan tff.learning.models.from\_keras\_model.



* 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 :

