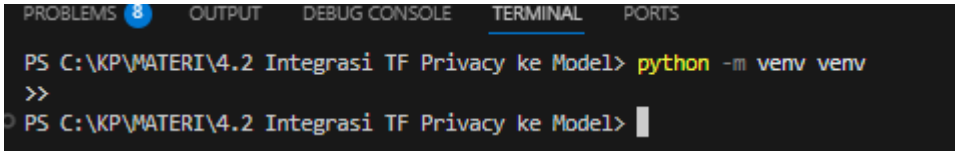
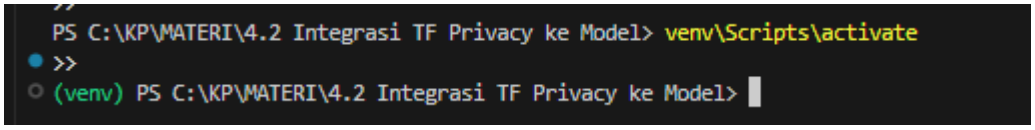
Topik : 4.2. Integrasi TF Privacy ke Model

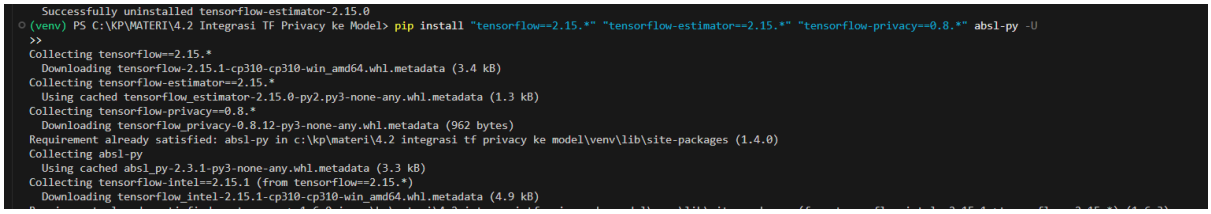
Objective : Implementasikan optimizer dengan DP-SGD

Task : Tambahkan dp_keras_optimizer ke model FL

Source : https://github.com/tensorflow/privacy/blob/master/tutorials/mnist_dpsgd_tutorial.py







1. Import dan Konfigurasi Awal

```python
import time

from absl import app
from absl import flags
from absl import logging
import tensorflow as tf
tf.compat.v1.disable_eager_execution()
from tensorflow_estimator import estimator as tf_estimator

from tensorflow_privacy.privacy.analysis import compute_dp_sgd_privacy_lib
from tensorflow_privacy.privacy.optimizers import dp_optimizer
import mnist_dpsgd_tutorial_common as common
```

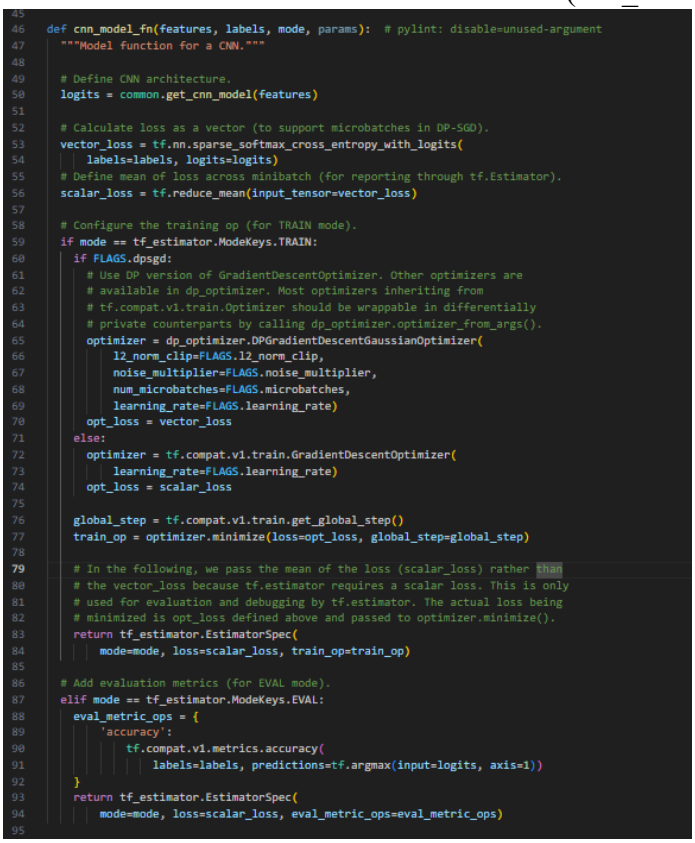
Mengaktifkan graph mode, menyiapkan estimator, DP-SGD, dan modul common yang berisi get_cnn_model() & make_input_fn()

2. Hyperparameter via absl.flags

```python
flags.DEFINE_boolean(
    'dpsgd', True, 'If True, train with DP-SGD. If False, '
    'train with vanilla SGD.')
flags.DEFINE_float('learning_rate', .15, 'Learning rate for training')
flags.DEFINE_float('noise_multiplier', 1.1,
                   'Ratio of the standard deviation to the clipping norm')
flags.DEFINE_float('l2_norm_clip', 1.0, 'Clipping norm')
flags.DEFINE_integer('batch_size', 256, 'Batch size')
flags.DEFINE_integer('epochs', 30, 'Number of epochs')
flags.DEFINE_integer(
    'microbatches', 256, 'Number of microbatches '
    '(must evenly divide batch_size)')
flags.DEFINE_string('model_dir', None, 'Model directory')

FLAGS = flags.FLAGS
```

semua parameter pelatihan & privasi dikontrol dari sini.

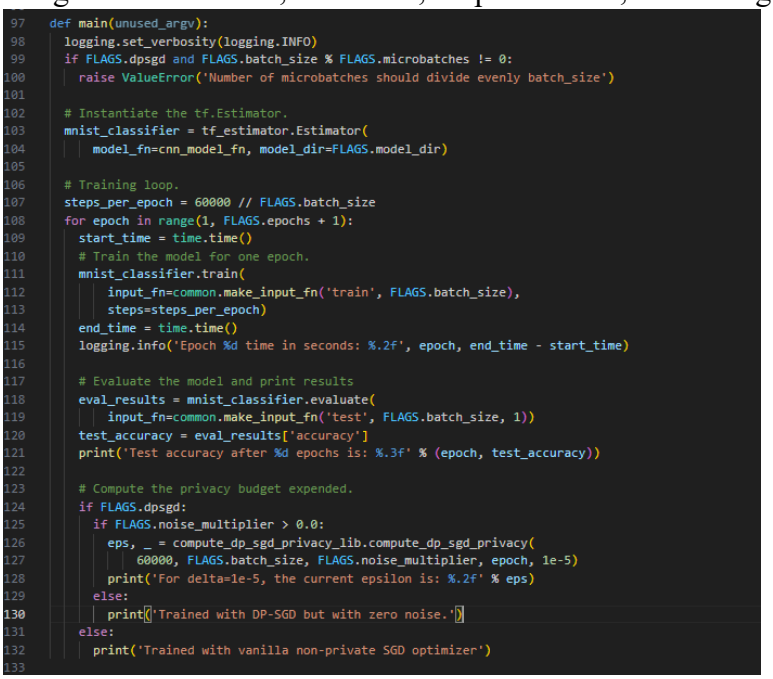
3. Model function untuk melakukan estimator (cnn_model_fn)



```python
def cnn_model_fn(features, labels, mode, params):  # pylint: disable=unused-argument
  """Model function for a CNN."""

  # Define CNN architecture.
  logits = common.get_cnn_model(features)

  # Calculate loss as a vector (to support microbatches in DP-SGD).
  vector_loss = tf.nn.sparse_softmax_cross_entropy_with_logits(
      labels=labels, logits=logits)
  # Define mean of loss across minibatch (for reporting through tf.Estimator).
  scalar_loss = tf.reduce_mean(input_tensor=vector_loss)

  # Configure the training op (for TRAIN mode).
  if mode == tf_estimator.ModeKeys.TRAIN:
    if FLAGS.dpsgd:
      # Use DP version of GradientDescentOptimizer. Other optimizers are
      # available in dp_optimizer. Most optimizers inheriting from
      # tf.compat.v1.train.Optimizer should be wrappable in differentially
      # private counterparts by calling dp_optimizer.optimizer_from_args().
      optimizer = dp_optimizer.DPGradientDescentGaussianOptimizer(
          l2_norm_clip=FLAGS.l2_norm_clip,
          noise_multiplier=FLAGS.noise_multiplier,
          num_microbatches=FLAGS.microbatches,
          learning_rate=FLAGS.learning_rate)
      opt_loss = vector_loss
    else:
      optimizer = tf.compat.v1.train.GradientDescentOptimizer(
          learning_rate=FLAGS.learning_rate)
      opt_loss = scalar_loss

    global_step = tf.compat.v1.train.get_global_step()
    train_op = optimizer.minimize(loss=opt_loss, global_step=global_step)

    # In the following, we pass the mean of the loss (scalar_loss) rather than
    # the vector_loss because tf.estimator requires a scalar loss. This is only
    # used for evaluation and debugging by tf.estimator. The actual loss being
    # minimized is opt_loss defined above and passed to optimizer.minimize().
    return tf_estimator.EstimatorSpec(
        mode=mode, loss=scalar_loss, train_op=train_op)

  # Add evaluation metrics (for EVAL mode).
  elif mode == tf_estimator.ModeKeys.EVAL:
    eval_metric_ops = {
        'accuracy':
            tf.compat.v1.metrics.accuracy(
                labels=labels, predictions=tf.argmax(input=logits, axis=1))
    }
    return tf_estimator.EstimatorSpec(
        mode=mode, loss=scalar_loss, eval_metric_ops=eval_metric_ops)
```

- **DP-SGD**: clipping L2 + Gaussian noise → wajib opt_loss = vector_loss.
- **Non-DP**: opt_loss = scalar_loss.
- Metrik yang dilaporkan saat evaluasi: **akurasi**.

4. Fungsi main: validasi, Estimator, loop train/eval, dan hitung ε



```python
def main(unused_argv):
  logging.set_verbosity(logging.INFO)
  if FLAGS.dpsgd and FLAGS.batch_size % FLAGS.microbatches != 0:
    raise ValueError('Number of microbatches should divide evenly batch_size')

  # Instantiate the tf.Estimator.
  mnist_classifier = tf_estimator.Estimator(
      model_fn=cnn_model_fn, model_dir=FLAGS.model_dir)

  # Training loop.
  steps_per_epoch = 60000 // FLAGS.batch_size
  for epoch in range(1, FLAGS.epochs + 1):
    start_time = time.time()
    # Train the model for one epoch.
    mnist_classifier.train(
        input_fn=common.make_input_fn('train', FLAGS.batch_size),
        steps=steps_per_epoch)
    end_time = time.time()
    logging.info('Epoch %d time in seconds: %.2f', epoch, end_time - start_time)

    # Evaluate the model and print results
    eval_results = mnist_classifier.evaluate(
        input_fn=common.make_input_fn('test', FLAGS.batch_size, 1))
    test_accuracy = eval_results['accuracy']
    print('Test accuracy after %d epochs is: %.3f' % (epoch, test_accuracy))

    # Compute the privacy budget expended.
    if FLAGS.dpsgd:
      if FLAGS.noise_multiplier > 0.0:
        eps, _ = compute_dp_sgd_privacy_lib.compute_dp_sgd_privacy(
            60000, FLAGS.batch_size, FLAGS.noise_multiplier, epoch, 1e-5)
        print('For delta=1e-5, the current epsilon is: %.2f' % eps)
      else:
        print('Trained with DP-SGD but with zero noise.')
    else:
      print('Trained with vanilla non-private SGD optimizer')
```

- **Validasi**: batch_size harus habis dibagi microbatches.
- **Train/Eval** per-epoch.
- **Epsilon (ε)** dihitung kumulatif per-epoch → trade-off akurasi vs privasi.

5. Entry point

```
134
135   if __name__ == '__main__':
136     app.run(main)
137
```

menjalankan main() dan mengurai flag dari command line.

Ringkasan :

- **Estimator** memanggil cnn_model_fn untuk **TRAIN** (pilih DP-SGD vs SGD) dan **EVAL** (akurasi).
- **DP-SGD**: pakai **loss per-contoh** (vector_loss), **clipping L2** (l2_norm_clip), **noise Gaussian** (noise_multiplier), dan **microbatches**.
- Setiap epoch: **latih → uji → cetak akurasi**, dan kalau DP aktif **hitung ε** (semakin kecil = privasi lebih kuat).

Source : https://www.tensorflow.org/responsible_ai/privacy/tutorials/classification_privacy

Implement Differential Privacy with Tensorflow Privacy

Privacy differensial (DP) Adalah sebuah kerangka kerja untuk mengukur jaminan privasi yang diberikan oleh sebuah algoritma. Melalui lensa privasi differensial , kita dapat merancang algoritma machine learning yang dapat melatih model secara bertanggung jawab pada data pribadi. Pembelajaran dengan privasi diferensial memberikan jaminan privasi yang terukur, membantu mengurangi risiko terbukanya data sensitif dalam pembelajaran mesin. Secara intuitif, model yang dilatih dengan privasi diferensial tidak boleh dipengaruhi oleh satu contoh pelatihan tunggal, atau kumpulan kecil contoh pelatihan, dalam himpunan datanya. Hal ini membantu mengurangi risiko terbukanya data sensitif dalam ML.

Ide dasar dari pendeketan ini, yang disebut *differentially private stochastic gradient descent (DP-SGD)*, Adalah melakukan modifikasi gradien yang akan digunakan dalam *stochastic gradient descent (SGD)*, yang merupakan inti dari hampir semua algoritma deep learning. Model yang dilatih dengan DP-SGD memberikan jaminan privasi diferensial yang terbukti untuk data input mereka.

2 modifikasi yang dilakukan pada algoritma SGD standar :

1. Sensitivitas setiap gradien perlu dibatasi

   Dengan kata lain, kita perlu membatasi sejauh mana setiap titik data pelatihan yang diambil dalam sebuah minibatch dapat memengaruhi perhitungan gradien dan pembaruan yang diterapkan pada parameter model. Hal ini dilakukan dengan *clipping* (memotong) setiap gradient yang dihitung pada setiap titik pelatihan.

2. Noise acak

   Diambil sampelnya dan ditambahkan pada gradient yang sudah dipotong, untuk membuatnya secara statistic mustahil mengetahui apakah suatu titik data tertentu termasuk atau tidak dalam dataset pelatihan, dengan cara membandingkan pembaruan yang dilakukan oleh SGD Ketika beroperasi dengan atau tanpa titik data tersebut dalam dataset pelatihan.

Tutorial ini menggunakan tf.keras untuk melatih *convolutional neural network (CNN)* dalam mengenali digit tulisan tangan dengan optimizer DP-SGD yang disediakan oleh Pustaka Tensorflow Privacy. Tensorflow Privacy menyediakan kode untuk membungkus optimizer Tensorflow yang sudah ada untuk membuat varian yang mengimplementasikan DP-SGD.

1. SetUp

```
############################## SETUP ##############################
import tensorflow as tf


import numpy as np
tf.get_logger().setLevel('ERROR')

import tensorflow_privacy
from tensorflow_privacy.privacy.analysis import compute_dp_sgd_privacy
```

```
Successfully installed tensorflow-probability-0.19.0
(.venv) PS C:\KP\MATERI\4.2 Integrasi TF Privacy ke Model> pip uninstall -y tensorflow tensorflow-intel tensorflow-estimator tensorflow-privacy tensorflow-probability
>> pip install tensorflow==2.15.0 tensorflow-estimator==2.15.0 tensorflow-privacy==0.9.0 tensorflow-probability==0.22.1
>>
Found existing installation: tensorflow 2.12.0
```

2. Load and pre-process the dataset

   Muat dataset MNIST dan siapkan data untuk pelatihan.

```
############################## Load and pre-process the dataset ##############################
train, test = tf.keras.datasets.mnist.load_data()
train_data, train_label = train
test_data , test_label = test

train_data = np.array(train_data, dtype=np.float32) / 255
test_data = np.array(test_data, dtype=np.float32) / 255

train_data = train_data.reshape(train_data.shape[0], 28, 28, 1)
test_data = test_data.reshape(test_data.shape[0], 28, 28, 1)

train_label = np.array(train_label, dtype=np.int32)
test_label = np.array(test_label, dtype=np.int32)

train_label = tf.keras.utils.to_categorical(train_label, num_classes=10)
test_label = tf.keras.utils.to_categorical(test_label, num_classes=10)

assert train_data.min() == 0
assert train_data.max() == 1
assert test_data.min() == 0
assert test_data.max() == 1
```

```
(.venv) PS C:\KP\MATERI\4.2 Integrasi TF Privacy ke Model> python classification_privacy.py
2025-09-10 09:10:35.255575: I tensorflow/core/util/port.cc:113] oneDNN custom operations are on. You
_ONEDNN_OPTS=0`.
WARNING:tensorflow:From C:\KP\MATERI\4.2 Integrasi TF Privacy ke Model\.venv\Lib\site-packages\keras\

Downloading data from https://storage.googleapis.com/tensorflow/tf-keras-datasets/mnist.npz
11490434/11490434 [==============================] - 4s 0us/step
(.venv) PS C:\KP\MATERI\4.2 Integrasi TF Privacy ke Model>
```

3. Define the hyperparameters
   Tetapkan nilai hiperparameter model pembelajaran.

```
37  ##########################
38  epochs = 3
39  batch_size = 250
```

DP-SGD memiliki tiga hyperparameter khusus privasi dan satu hyperparameter yang sudah ada yang harus anda atur :

1. l2_norm_clip (float)
   Nilai maksimum norma Euclidean (L2) dari setiap gradient yang diterapkan untuk memperbarui parameter model. Hyperparameter ini digunakan untuk membatasi sensitivitas optimizer terhadap titik data pelatihan individu.

2. noise_multiplier (float)
   Jumlah noise yang diambil sampelnya dan ditambahkan ke gradien selama pelatihan. Secara umum, semakin banyak noise maka semakin menghasilkan privasi yang lebih baik (meskipun tidak selalu dengan mengorbankan utilitas yang lebih rendah)

3. microbatches (int)
   Setiap batch data dibagi ke dalam unit – unit lebih kecil yang disebut microbatch. Secara default, setiap microbatch sebaiknya hanya berisi satu contoh pelatihan. Hal ini memungkinan kita untuk melakukan clipping gradien pada basis per-contoh, bukan setelah rata – rata di seluruh minibatch. Dengan demikian, efek (nefatif) clipping pada sinyal yang ditemukan di gradien berkurang dan biasanya memaksimalkan utilitas. Namun, overhead komputasi dapat dikurangi dengan memperbesar ukuran microbatch sehingga mencakup lebih dari satu contoh pelatihan. Rata – rata gradien di seluruh contoh pelatihan ini kemudian di-clipping. Jumlah total contoh yang digunakan dalam satu batch (satu Langkah *gradient descent)* tetap sama. Jumlah microbatch harus membagi ukuran batch secara merata.

4. learning_rate (float)
   Hyperparameter ini sudah ada dalam vanilla SGD. Semakin tinggi nilai learning rate, semakin besar pengaruh setiap pembaruan. Jika pembaruan sangat bising (misalnya saat noise aditif besar dibandingkan ambang clipping), learning rate yang rendah dapat membantu prosedur pelatihan untuk konvergen.

Rangkuman :

- l2_norm_clip → membatasi gradien biar tidak ada satu data mendominasi.
- noise_multiplier → mengatur seberapa besar noise ditambahkan demi privasi.
- microbatches → membagi batch jadi unit kecil supaya clipping lebih akurat.
- learning_rate → mengatur kecepatan update (harus seimbang dengan noise & clipping).

Gunakan nilai hiperparameter di bawah ini untuk mendapatkan model yang cukup akurat (akurasi pengujian 95%):

```
41  l2_norm_clip = 1.5
42  noise_multiplier = 1.3
43  num_microbatches = 250
44  learning_rate = 0.25
45
46  if batch_size % num_microbatches != 0 :
47      raise ValueError ('Batch size should be an integer multiple of the number of microbatches')
48
```

4. Build the model

Tentukan jaringan saraf konvolusional sebagai model pembelajaran.

```
53  ############################## Build the model ##############################
54  model = tf.keras.Sequential([
55      tf.keras.layers.Conv2D(16, 8, strides=2, padding='same', activation='relu', input_shape=(28,28,1)),
56      tf.keras.layers.MaxPool2D(2,1),
57      tf.keras.layers.Conv2D(32,4, strides=2, padding='valid', activation='relu'),
58      tf.keras.layers.MaxPool2D(2,1),
59      tf.keras.layers.Flatten(),
60      tf.keras.layers.Dense(32, activation='relu'),
61      tf.keras.layers.Dense(10)
62  ])
```

Tentukan pengoptimal dan fungsi kerugian untuk learning model. Hitung kerugian sebagai vector kerugian per-contoh, alih – alih sebagai rata – rata pada minibatch untuk mendukung manipulasi gradien pada setiap titik pelatihan.

```
63
64  optimizer = tensorflow_privacy.DPKerasSGDOptimizer(
65      l2_norm_clip=l2_norm_clip,
66      noise_multiplier=noise_multiplier,
67      num_microbatches=num_microbatches,
68      learning_rate=learning_rate
69  )
70  loss = tf.keras.losses.CategoricalCrossentropy(
71      from_logits = True, reduction=tf.losses.Reduction.NONE
72  )
```

5. Train the model

```
74
75
76  ############################## Train the model ##############################
77  model.compile(optimizer=optimizer, loss=loss, metrics=['accuracy'])
78  model.fit(train_data,
79            train_label,
80            epochs=epochs,
81            validation_data=(test_data, test_label),
82            batch_size=batch_size)
```

```
ags.
Epoch 1/3
240/240 [==============================] - 1115s 5s/step - loss: 0.9647 - accuracy: 0.7020 - val_loss: 0.4016 - val_accuracy: 0.8855
Epoch 2/3
240/240 [==============================] - 633s 3s/step - loss: 0.3895 - accuracy: 0.8970 - val_loss: 0.3299 - val_accuracy: 0.9219
Epoch 3/3
240/240 [==============================] - 610s 3s/step - loss: 0.3546 - accuracy: 0.9188 - val_loss: 0.3235 - val_accuracy: 0.9296
Traceback (most recent call last):
```

6. Measure the differential privacy guarantee

Melakukan analisis privasi untuk mengukur jaminan DP yang dicapai oleh algoritma pelatihan. Mengetahui Tingkat DP yang dicapai memungkinkan perbandingan objektif dari dua proses pelatihan untuk menentukan mana yang lebih menjaga privasi.

Analisis privasi mengukur seberapa jauh seorang lawan potensial dapat meningkatkan tebakan mereka tentang property titik data individu dengan mengamati hasil dari prosedur pelatihan (misalnya, pembaruan model dan parameter).

Jaminan ini kadang disebut sebagai anggaran privasi (privacy budget). Anggaran privasi yang lebih rendah membatasi lebih ketat kemampuan lawan untuk meningkatkan tebakan mereka. Hal ini memastikan jaminan privasi yang lebih kuat. Secara intuitif, ini karena lebih sulit bagi satu titik data pelatihan untuk memengaruhi hasil pembelajaran : misalnya, informasi dalam titik data pelatihan tidak bisa dihafal oleh algoritma ML, dan privasi individu yang menyumbangkan titik data tersebut ke dataset tetap terjaga.

Dalam tutorial ini, analisis privasi dilakukan dengan kerangka kerja *Renyi Differential Privacy (*RDP), yang merupakan relaksasi dari pure DP yang sangat cocok untuk DP-SGD.

Dua metrics yang digunakan untuk mengekspresikan jaminan DP dari sebuah algoritma ML :

1. **Delta (δ)**
   Membatasi probabilitas jaminan privasi tidak berlaku. Aturannya Adalah nilainya lebih kecil dari kebalikan ukuran dataset pelatihan. Dalam tutorial ini, δ diset ke 10^-5 karena dataset MNIST memiliki 60.000 titik data pelatihan.

2. **Epsilon (ε)**
   Ini Adalah privacy budget. Mengukur kekuatan jaminan privasi dengan membatasi seberapa besar probabilitas keluaran model tertentu dapat bervariasi dengan menyertakan (atau menghapus) satu titik data pelatihan. Nilai ε yang lebih kecil berarti jaminan privasi lebih baik. Namun, nilai ε hanyalah batas atas; nilai yang besar masih bisa berarti privasi yang cukup baik dalam praktik.

Tensorflow privacy menyediakan sebuah alat, **compute_dp_sgd_privacy**, untuk menghitung nilai ε dengan nilai δ yang tetap dan hyperparameter berikut dari proses pelatihan:

1. Jumlah total titik data dalam data pelatihan, **n**
2. **Batch_size**
3. **noise_multiplier**
4. jumlah **epochs** pelatihan

```
87
88  ############################## Measure the differential privacy guarantee ################
89  from tensorflow_privacy.privacy.analysis import compute_dp_sgd_privacy_lib
90
91  epsilon, best_alpha = compute_dp_sgd_privacy_lib.compute_dp_sgd_privacy(
92      n=train_data.shape[0],
93      batch_size=batch_size,
94      noise_multiplier=noise_multiplier,
95      epochs=epochs,
96      delta=1e-5
97  )
98
99  print(f"DP-SGD with ε = {epsilon:.2f}, δ = 1e-5. For α = {best_alpha}.")
100
```

```
(.venv) PS C:\KP\MATERI\4.2 Integrasi TF Privacy ke Model>
(.venv) PS C:\KP\MATERI\4.2 Integrasi TF Privacy ke Model> python classification_privacy.py
2025-09-10 11:33:43.345782: I tensorflow/core/util/port.cc:113] oneDNN custom operations are
e to floating-point round-off errors from different computation orders. To turn them off, se
WARNING:tensorflow:From C:\KP\MATERI\4.2 Integrasi TF Privacy ke Model\.venv\Lib\site-packag
oftmax_cross_entropy is deprecated. Please use tf.compat.v1.losses.sparse_softmax_cross_entr

2025-09-10 11:33:54.319346: I tensorflow/core/platform/cpu_feature_guard.cc:182] This Tensor
ns in performance-critical operations.
To enable the following instructions: SSE SSE2 SSE3 SSE4.1 SSE4.2 AVX2 FMA, in other operati
ags.
WARNING:absl:`compute_dp_sgd_privacy` is deprecated. It does not account for doubling of sen
ling, which is rarely used in practice. Please use `compute_dp_sgd_privacy_statement`, which
ute epsilon under different assumptions than those in `compute_dp_sgd_privacy_statement`, ca
DP-SGD with ε = 0.56, δ = 1e-5. For α = 18.0.
(.venv) PS C:\KP\MATERI\4.2 Integrasi TF Privacy ke Model>
```

Alat tersebut melaporkan bahwa untuk hiperparameter yang dipilih di atas, model yang dilatih memiliki nilai sebesar 1,18.

Task : Tambahkan dp_keras_optimizer ke model FL

1. **Import & hyperparameter DP klien**

```
10  print("TensorFlow:", tf.__version__)
11  try:
12      from tensorflow_privacy.privacy.optimizers.dp_optimizer_keras import DPKerasSGDOptimizer
13      from tensorflow_privacy.privacy.analysis import compute_dp_sgd_privacy_lib
14      TF_PRIVACY_OK = True
15      print("TF-Privacy: OK (dp_keras_optimizer tersedia)")
```

```
TensorFlow: 2.14.1
TF-Privacy: OK (dp_keras_optimizer tersedia)
```

```
DP_CLIENT_LR            = 0.05
DP_L2_NORM_CLIP_CLIENT  = 1.5
NOISE_WARMUP            = 0.001   # kecil supaya naik di awal
NOISE_MAIN              = 0.1
```

2. **Dipasang ke model klien saat compile**

```
# DP optimizer
opt = DPKerasSGDOptimizer(
    l2_norm_clip=DP_L2_NORM_CLIP_CLIENT,
    noise_multiplier=client_noise,
    num_microbatches=DP_NUM_MICROBATCHES,   # = BATCH_SIZE
    learning_rate=DP_CLIENT_LR,
    momentum=0.9
)
loss = tf.keras.losses.BinaryCrossentropy(from_logits=False, reduction=tf.losses.Reduction.NONE)
local_model.compile(optimizer=opt, loss=loss, metrics=["accuracy"])
```

3. **Dataset klien disiapkan agar kompatibel dengan DP**

```
ds = ds.batch(BATCH_SIZE, drop_remainder=True).repeat(local_epochs).prefetch(tf.data.AUTOTUNE)
return ds
```

4. **Setelah training lokal selesai, delta diklip & di-agregasi → inilah bagian FL-nya:**

```
# Delta
w_local = local_model.get_weights()
delta = [wl - wg for wl, wg in zip(w_local, global_weights)]

# Clip user-level sebelum agregasi
delta_clipped, _ = clip_by_l2_norm(delta, SERVER_CLIP)
```

Hasil Training :

```
Round 01 | client_loss=0.4943 | global_loss=0.5345 | global_acc=0.7449 | ma_acc=0.7449 | client_noise=0.001
Round 02 | client_loss=0.4152 | global_loss=0.4404 | global_acc=0.7449 | ma_acc=0.7449 | client_noise=0.001
Round 03 | client_loss=0.3370 | global_loss=0.3723 | global_acc=0.7818 | ma_acc=0.7560 | client_noise=0.001
Round 04 | client_loss=0.2747 | global_loss=0.2796 | global_acc=0.8756 | ma_acc=0.7918 | client_noise=0.001
Round 05 | client_loss=0.2238 | global_loss=0.2243 | global_acc=0.9191 | ma_acc=0.8300 | client_noise=0.001
Round 06 | client_loss=0.1630 | global_loss=0.1881 | global_acc=0.9387 | ma_acc=0.8626 | client_noise=0.1
Round 07 | client_loss=0.1403 | global_loss=0.1756 | global_acc=0.9407 | ma_acc=0.8860 | client_noise=0.1
Round 08 | client_loss=0.1272 | global_loss=0.1682 | global_acc=0.9342 | ma_acc=0.9005 | client_noise=0.1
Round 09 | client_loss=0.1240 | global_loss=0.1645 | global_acc=0.9362 | ma_acc=0.9112 | client_noise=0.1
Round 10 | client_loss=0.1151 | global_loss=0.1629 | global_acc=0.9376 | ma_acc=0.9191 | client_noise=0.1
Round 11 | client_loss=0.1021 | global_loss=0.1589 | global_acc=0.9371 | ma_acc=0.9245 | client_noise=0.1
Round 12 | client_loss=0.0996 | global_loss=0.1506 | global_acc=0.9402 | ma_acc=0.9292 | client_noise=0.1
Round 13 | client_loss=0.0953 | global_loss=0.1472 | global_acc=0.9400 | ma_acc=0.9325 | client_noise=0.1
Round 14 | client_loss=0.0948 | global_loss=0.1369 | global_acc=0.9424 | ma_acc=0.9355 | client_noise=0.1
Round 15 | client_loss=0.0860 | global_loss=0.1282 | global_acc=0.9458 | ma_acc=0.9386 | client_noise=0.1

Training selesai (FedAvg manual + DP-SGD di klien).
WARNING:absl:`compute_dp_sgd_privacy` is deprecated. It does not account for doubling of sensitivity with microba
_dp_sgd_privacy_statement`, which provides appropriate context for the guarantee. To compute epsilon under differ
raries directly.
Perkiraan DP-SGD klien (fase utama): ε ≈ 3361.26, δ = 1e-5 (α = 1.25)
(venv) ezranahumury@DESKTOP-8003BIM:/mnt/c/KP/MATERI/4.2 Integrasi TF Privacy ke Model/task$
```

Measure the differential privacy

```python
# ========================================================
# 7)  Estimasi epsilon untuk fase utama
# ========================================================
try:
    n = max(client_sizes)
    eps, best_alpha = compute_dp_sgd_privacy_lib.compute_dp_sgd_privacy(
        n=n,
        batch_size=BATCH_SIZE,
        noise_multiplier=NOISE_MAIN,
        epochs=LOCAL_EPOCHS_MAIN,
        delta=1e-5
    )
    print(f"Perkiraan DP-SGD klien (fase utama): ε ≈ {eps:.2f}, δ = 1e-5 (α = {best_alpha})")
except Exception as e:
    print("Lewati perhitungan epsilon:", e)

```

```
raries directly.
Perkiraan DP-SGD klien (fase utama): ε ≈ 3361.26, δ = 1e-5 (α = 1.25)
(venv) ezranahumury@DESKTOP-8003BIM:/mnt/c/KP/MATERI/4.2 Integrasi TF Privacy ke Model/task$
```

Estimasi ε, δ dengan compute_dp_sgd_privacy_lib.compute_dp_sgd_privacy(...) per klien.