privacy_dpsgd

August 5, 2025

1 Implement Differential Privacy with TensorFlow Privacy

1.1 Learning Objectives

- Learn how to wrap existing optimizers (e.g., SGD, Adam) into their differentially private counterparts using TensorFlow Privacy
- Understand hyperparameters introduced by differentially private machine learning
- Measure the privacy guarantee provided using analysis tools included in TensorFlow Privacy

1.2 Overview

Differential privacy (DP) is a framework for measuring the privacy guarantees provided by an algorithm. Through the lens of differential privacy, you can design machine learning algorithms that responsibly train models on private data. Learning with differential privacy provides measurable guarantees of privacy, helping to mitigate the risk of exposing sensitive training data in machine learning. Intuitively, a model trained with differential privacy should not be affected by any single training example, or small set of training examples, in its data set. This helps mitigate the risk of exposing sensitive training data in ML.

The basic idea of this approach, called differentially private stochastic gradient descent (DP-SGD), is to modify the gradients used in stochastic gradient descent (SGD), which lies at the core of almost all deep learning algorithms. Models trained with DP-SGD provide provable differential privacy guarantees for their input data. There are two modifications made to the vanilla SGD algorithm:

1. First, the sensitivity of each gradient needs to be bounded. In other words, you need to limit how much each individual training point sampled in a minibatch can influence gradient computations and the resulting updates applied to model parameters. This can be done by *clipping* each gradient computed on each training point.
2. *Random noise* is sampled and added to the clipped gradients to make it statistically impossible to know whether or not a particular data point was included in the training dataset by comparing the updates SGD applies when it operates with or without this particular data point in the training dataset.

This tutorial uses tf.keras to train a convolutional neural network (CNN) to recognize handwritten digits with the DP-SGD optimizer provided by the TensorFlow Privacy library. TensorFlow Privacy provides code that wraps an existing TensorFlow optimizer to create a variant that implements DP-SGD.

1.3 Setup

[ ]:
```
!pip install --user --no-deps tensorflow-privacy==0.8.12 dp_accounting==0.4.3
```

Begin by importing the necessary libraries:

[2]:
```
import os
import warnings

os.environ["TF_CPP_MIN_LOG_LEVEL"] = "2"
warnings.filterwarnings("ignore")
```

[3]:
```
import numpy as np
import tensorflow as tf

tf.get_logger().setLevel("ERROR")
```

```
2024-03-11 18:23:24.342772: E
external/local_xla/xla/stream_executor/cuda/cuda_dnn.cc:9261] Unable to register
cuDNN factory: Attempting to register factory for plugin cuDNN when one has
already been registered
2024-03-11 18:23:24.342907: E
external/local_xla/xla/stream_executor/cuda/cuda_fft.cc:607] Unable to register
cuFFT factory: Attempting to register factory for plugin cuFFT when one has
already been registered
2024-03-11 18:23:24.349126: E
external/local_xla/xla/stream_executor/cuda/cuda_blas.cc:1515] Unable to
register cuBLAS factory: Attempting to register factory for plugin cuBLAS when
one has already been registered
```

Import TensorFlow Privacy.

[4]:
```
import tensorflow_privacy
from tensorflow_privacy.privacy.analysis import compute_dp_sgd_privacy
```

1.4 Load and pre-process the dataset

Load the MNIST dataset and prepare the data for training.

[5]:
```
train, test = tf.keras.datasets.mnist.load_data()
train_data, train_labels = train
test_data, test_labels = 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_labels = np.array(train_labels, dtype=np.int32)
test_labels = np.array(test_labels, dtype=np.int32)

train_labels = tf.keras.utils.to_categorical(train_labels, num_classes=10)
test_labels = tf.keras.utils.to_categorical(test_labels, num_classes=10)

assert train_data.min() == 0.0
assert train_data.max() == 1.0
assert test_data.min() == 0.0
assert test_data.max() == 1.0
```

1.5 Define the hyperparameters

Set learning model hyperparamter values.

DP-SGD has three general hyperamater and three privacy-specific hyperparameters that you must tune:

General hyperparameters

1. `epochs` (int) - This refers to the one entire passing of training data through the algorithm. Larger epoch increase the privacy risks since the model is trained on a same data point for multiple times.
2. `batch_size` (int) - Batch size affects different aspects of DP-SGD training. For instance, increasing the batch size could reduce the amount of noise added during training under the same privacy guarantee, which reduces the training variance.
3. `learning_rate` (float) - This hyperparameter already exists in vanilla SGD. The higher the learning rate, the more each update matters. If the updates are noisy (such as when the additive noise is large compared to the clipping threshold), a low learning rate may help the training procedure converge.

Privacy-specific hyperparameters 1. `l2_norm_clip` (float) - The maximum Euclidean (L2) norm of each gradient that is applied to update model parameters. This hyperparameter is used to bound the optimizer's sensitivity to individual training points. 2. `noise_multiplier` (float) - Ratio of the standard deviation to the clipping norm (The amount of noise sampled and added to gradients during training). Generally, more noise results in better privacy (often, but not necessarily, at the expense of lower utility). 3. `microbatches` (int) - Each batch of data is split in smaller units called microbatches. By default, each microbatch should contain a single training example. This allows us to clip gradients on a per-example basis rather than after they have been averaged across the minibatch. This in turn decreases the (negative) effect of clipping on signal found in the gradient and typically maximizes utility. However, computational overhead can be reduced by increasing the size of microbatches to include more than one training examples. The average gradient across these multiple training examples is then clipped. The total number of examples consumed in a batch, i.e., one step of gradient descent, remains the same. The number of microbatches should evenly divide the batch size.

Use the hyperparameter values below to obtain a reasonably accurate model (95% test accuracy):

[6]:
```
epochs = 1
batch_size = 32
```

```
learning_rate = 0.25

l2_norm_clip = 1.0
noise_multiplier = 0.5
num_microbatches = 32  # Same as the batch size (i.e. no microbatch)

if batch_size % num_microbatches != 0:
    raise ValueError(
        "Batch noise_multipliere should be an integer multiple of the number of␣
 ↪microbatches"
    )
```

1.6 Build the model

Define a convolutional neural network as the learning model.

[7]:
```
model = tf.keras.Sequential(
    [
        tf.keras.layers.Conv2D(
            16,
            8,
            strides=2,
            padding="same",
            activation="relu",
            input_shape=(28, 28, 1),
        ),
        tf.keras.layers.MaxPool2D(2, 1),
        tf.keras.layers.Conv2D(
            32, 4, strides=2, padding="valid", activation="relu"
        ),
        tf.keras.layers.MaxPool2D(2, 1),
        tf.keras.layers.Flatten(),
        tf.keras.layers.Dense(32, activation="relu"),
        tf.keras.layers.Dense(10, activation="softmax"),
    ]
)
```

Define the optimizer and loss function for the learning model. Compute the loss as a vector of losses per-example rather than as the mean over a minibatch to support gradient manipulation over each training point.

[8]:
```
optimizer = tensorflow_privacy.DPKerasSGDOptimizer(
    l2_norm_clip=l2_norm_clip,
    noise_multiplier=noise_multiplier,
    num_microbatches=num_microbatches,
    learning_rate=learning_rate,
)
```

```
loss = tf.keras.losses.CategoricalCrossentropy(
    reduction=tf.losses.Reduction.NONE
)
```

1.7 Train the model

[9]:
```
model.compile(optimizer=optimizer, loss=loss, metrics=["accuracy"])

model.fit(
    train_data,
    train_labels,
    epochs=epochs,
    validation_data=(test_data, test_labels),
    batch_size=batch_size,
)
```

```
1875/1875 [==============================] - 168s 89ms/step - loss: 0.7247 -
accuracy: 0.8371 - val_loss: 0.8313 - val_accuracy: 0.8877
```

[9]:
```
<keras.src.callbacks.History at 0x7f52b431b040>
```

1.8 Measure the differential privacy guarantee

Perform a privacy analysis to measure the DP guarantee achieved by a training algorithm. Knowing the level of DP achieved enables the objective comparison of two training runs to determine which of the two is more privacy-preserving. At a high level, the privacy analysis measures how much a potential adversary can improve their guess about properties of any individual training point by observing the outcome of the training procedure (e.g., model updates and parameters).

This guarantee is sometimes referred to as the **privacy budget**. A lower privacy budget bounds more tightly an adversary's ability to improve their guess. This ensures a stronger privacy guarantee. Intuitively, this is because it is harder for a single training point to affect the outcome of learning: for instance, the information contained in the training point cannot be memorized by the ML algorithm and the privacy of the individual who contributed this training point to the dataset is preserved.

In this tutorial, the privacy analysis is performed in the framework of Rényi Differential Privacy (RDP), which is a relaxation of pure DP based on this paper that is particularly well suited for DP-SGD.

Two metrics are used to express the DP guarantee of an ML algorithm:

1. Delta (δ) - Bounds the probability of the privacy guarantee not holding. A rule of thumb is to set it to be less than the inverse of the size of the training dataset. In this tutorial, it is set to 10^{-5} as the MNIST dataset has 60,000 training points.
2. Epsilon (ϵ) - This is the privacy budget. It measures the strength of the privacy guarantee (or maximum tolerance for revealing information on input data) by bounding how much the probability of a particular model output can vary by including (or excluding) a single training point. A smaller value for ϵ implies a better privacy guarantee. However, the ϵ value is only an upper bound and a large value could still mean good privacy in practice.

For more detail about the mathematical definition of (ϵ, δ)-differential privacy, see the original DP-SGD paper.

Tensorflow Privacy provides a tool, `compute_dp_sgd_privacy`, to compute the value of ϵ given a fixed value of δ and the following hyperparameters from the training process:

1. The total number of points in the training data, `n`.
2. The `batch_size`.
3. The `noise_multiplier`.
4. The number of `epochs` of training.

[10]:
```
dpsgd_statement = compute_dp_sgd_privacy.compute_dp_sgd_privacy_statement(
    number_of_examples=train_data.shape[0],
    batch_size=batch_size,
    noise_multiplier=noise_multiplier,
    used_microbatching=False,
    num_epochs=epochs,
    delta=1e-5,
)

print(dpsgd_statement)
```

```
DP-SGD performed over 60000 examples with 32 examples per iteration, noise
multiplier 0.5 for 1 epochs without microbatching, and no bound on number of
examples per user.

This privacy guarantee protects the release of all model checkpoints in addition
to the final model.

Example-level DP with add-or-remove-one adjacency at delta = 1e-05 computed with
RDP accounting:
    Epsilon with each example occurring once per epoch:        10.726
    Epsilon assuming Poisson sampling (*):                      3.800

No user-level privacy guarantee is possible without a bound on the number of
examples per user.

(*) Poisson sampling is not usually done in training pipelines, but assuming
that the data was randomly shuffled, it is believed the actual epsilon should be
closer to this value than the conservative assumption of an arbitrary data
order.
```

The tool reports ϵ value for the hyperparameters chosen above, including $\delta = 10^{-5}$.

the specific language governing permissions and limitations under the License

[ ]: