TensorFlow Everywhere

TensorFlow & Deep Learning Singapore

Martin Andrews

martin@reddragon.ai

About Me

- Machine Intelligence / Startups / Finance
  - NYC to Singapore in 2013
- 2014 = 'fun' :
  - Machine Learning, Deep Learning, NLP
  - Robots, drones
- Since 2015 = 'serious' :: NLP + Deep Learning
  - GDE ML
  - TF&DL co-organiser
  - Research
  - Red Dragon AI

Outline

A few quick topics:

- `tf.experimental.numpy`
- TensorFlow Profiler
- Research with TensorFlow : Privacy and NeRF-in-the-Wild

Our next online events for Singapore :

- Courses in partnership with SGinnovate (with funding…)
- Beginner-friendly code & idea walk-throughs
- Deep Learning : The Cool Stuff

tf.experimental.numpy

Accelerate NumPy using TensorFlow

NumPy works with TensorFlow

You can now:

- Run a subset of NumPy on CPU / GPU / TPU
- Differentiate through NumPy code
- Combine NumPy code with TensorFlow APIs (`tf.linalg`, `tf.signal`, `tf.data`, `tf.keras`, `tf.distribute`)
- Compile NumPy code using `` `tf.function` `` and vectorize it using `` `tf.vectorized_map` ``

Visit tensorflow.org/guide/tf_numpy to learn more

```
# Available in TensorFlow Nightly
# `pip install tf-nightly`
import tensorflow.experimental.numpy as tnp

# Write NumPy Code, accelerated by TensorFlow on GPUs
x = tnp.random.randn(100, 100).clip(-2, 2)
print(x.data.device)
```

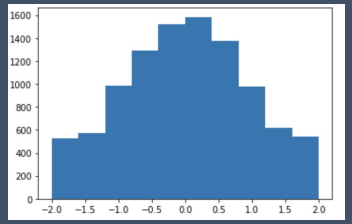
```
/job:localhost/replica:0/task:0/device:GPU:0
```

```
# Note that TensorFlow ND Arrays can be passed to APIs expecting NumPy arrays.
# This works since ND Array class implements `__array__` interface defined by NumPy.
# The code below demonstrates matplotlib plotting on ND Array.
```

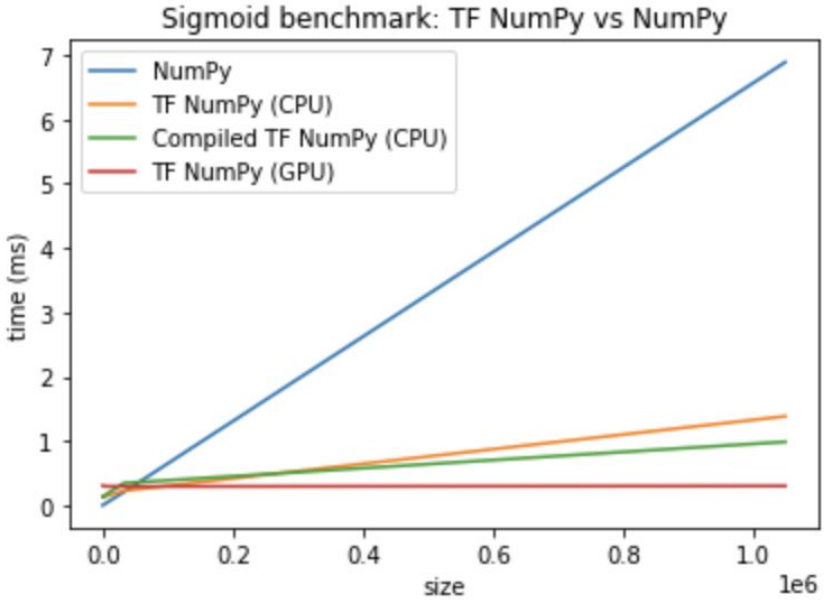
```
# Available in TensorFlow Nightly
# `pip install tf-nightly`
import tensorflow.experimental.numpy as tnp

# Write NumPy Code, accelerated by TensorFlow on GPUs
x = tnp.random.randn(100, 100).clip(-2, 2)
print(x.data.device)
```

```
/job:localhost/replica:0/task:0/device:GPU:0
```

```
# Note that TensorFlow ND Arrays can be passed to APIs expecting NumPy arrays.
# This works since ND Array class implements `__array__` interface defined by NumPy.
# The code below demonstrates matplotlib plotting on ND Array.

import matplotlib.pyplot as plt
plt.hist(x.ravel())
```



Performance example



TensorFlow runtime provides highly optimized kernels for different devices. However, NumPy has a lower dispatch latency (~1us). TensorFlow thus runs faster for workloads not dominated by dispatch latency.

Learn more in the guide: tensorflow.org/guide/tf_numpy

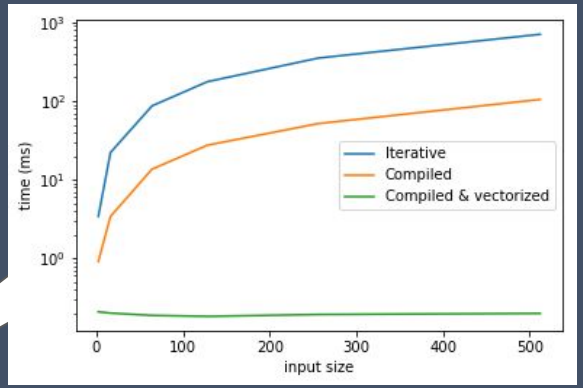
```
# Use NumPy code in input pipelines
dataset = tf.data.Dataset.from_tensor_slices(tnp.random.randn(1000, 1024)).map(
    lambda z: z.clip(-1, 1)).batch(100)
```

```
# Use NumPy code in input pipelines
dataset = tf.data.Dataset.from_tensor_slices(tnp.random.randn(1000, 1024)).map(
    lambda z: z.clip(-1, 1)).batch(100)

# Compute gradients through NumPy code
def grad(x, wt):
  with tf.GradientTape() as tape:
    tape.watch(wt)
    output = tnp.dot(x, wt)
    output = 1 / (1 + tnp.exp(-output))
  return tape.gradient(tnp.sum(output), wt)  # Also see tape.batch_jacobian
```

```
# Use NumPy code in input pipelines
dataset = tf.data.Dataset.from_tensor_slices(tnp.random.randn(1000, 1024)).map(
    lambda z: z.clip(-1, 1)).batch(100)

# Compute gradients through NumPy code
def grad(x, wt):
  with tf.GradientTape() as tape:
    tape.watch(wt)
    output = tnp.dot(x, wt)
    output = 1 / (1 + tnp.exp(-output))
  return tape.gradient(tnp.sum(output), wt)  # Also see tape.batch_jacobian

wt = tnp.random.randn(1024, 1024)

# Write code with python control flow
for inputs in dataset:
  gradients = grad(inputs, wt)
```

```
# Use NumPy code in input pipelines
dataset = tf.data.Dataset.from_tensor_slices(tnp.random.randn(1000, 1024)).map(
    lambda z: z.clip(-1, 1)).batch(100)

# Compute gradients through NumPy code
def grad(x, wt):
  with tf.GradientTape() as tape:
    tape.watch(wt)
    output = tnp.dot(x, wt)
    output = tf.math.sigmoid(output)  # Interleave with TensorFlow APIs
  return tape.gradient(tnp.sum(output), wt)

def per_example_grad(x, wt):
  return tf.map_fn(lambda y: grad(y, wt), x)  # Interleave with TensorFlow APIs

wt = tnp.random.randn(1024, 1024)
# Write idiomatic code, with python control flow
for inputs in dataset:
  per_example_gradients = per_example_grad(inputs, wt)
```

```python
# Use NumPy code in input pipelines
dataset = tf.data.Dataset.from_tensor_slices(tnp.random.randn(1000, 1024)).map(
    lambda z: z.clip(-1, 1)).batch(100)

# Compute gradients through NumPy code
def grad(x, wt):
  with tf.GradientTape() as tape:
    tape.watch(wt)
    output = tnp.dot(x, wt)
    output = tf.math.sigmoid(output)
  return tape.gradient(tnp.sum(output), wt)
```




```python
# Speedup NumPy code with compilation and vectorization
@tf.function  # Compilation
def per_example_grad(x, wt):
  return tf.vectorized_map(lambda y: grad(y, wt), x)  # Auto-vectorization

wt = tnp.random.randn(1024, 1024)
# Write idiomatic code, with python control flow
for inputs in dataset:
  per_example_gradients = per_example_grad(inputs, wt)
```

TensorFlow NumPy: Future Directions

- In-place mutation of ND arrays (`x[0, :] = x[0, :]*2`)
- Support for more ops
- Fast operation dispatch using TFRT
- Lower-level APIs for distribution
- Key NumPy library support (e.g. Trax, scikit-learn)

Learn more at tensorflow.org/guide/tf_numpy

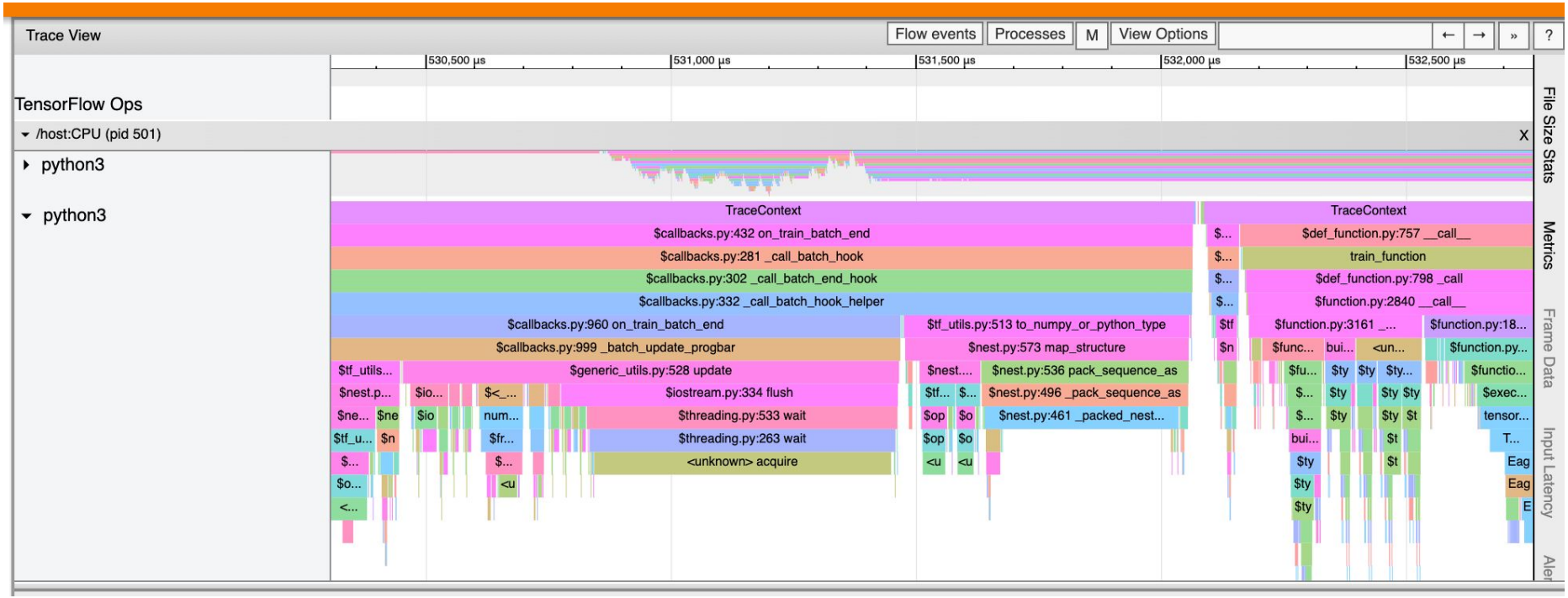
TensorFlow Profiler

TensorFlow Profiler

- Quick DEMO

Learn more at github.com/tensorflow/profiler

Python Tracer



Memory Profile Tool

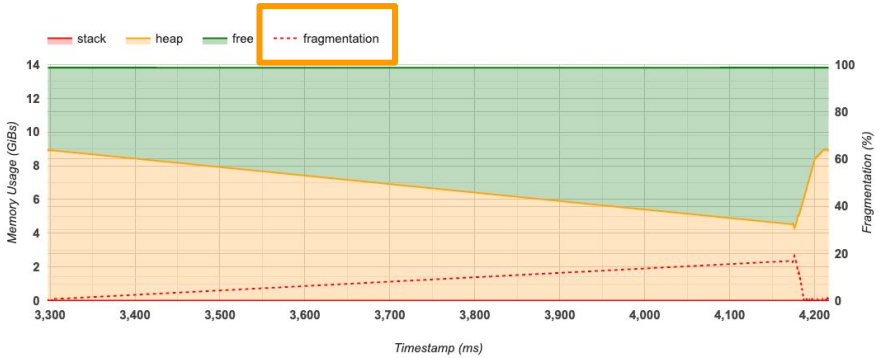
Memory Profile Summary

Memory ID *show memory profile for selected device*	GPU_...
#Allocation	880
#Deallocation	120
Memory Capacity	13.82 GiBs
Peak Heap Usage *high water mark in lifetime*	9.14 GiBs
Peak Memory Usage *stack + heap, within profiling window*	8.98 GiBs

- Timestamp: 4213.4 ms
- Stack Reservation: 0.00 GiBs
- Heap Allocation: 8.98 GiBs
- Free Memory: 4.84 GiBs
- Fragmentation: 0.19%

Memory Timeline Graph

Tips: Zoom in: left click and drag; Zoom out: right click; Metadata: click on heap data point.



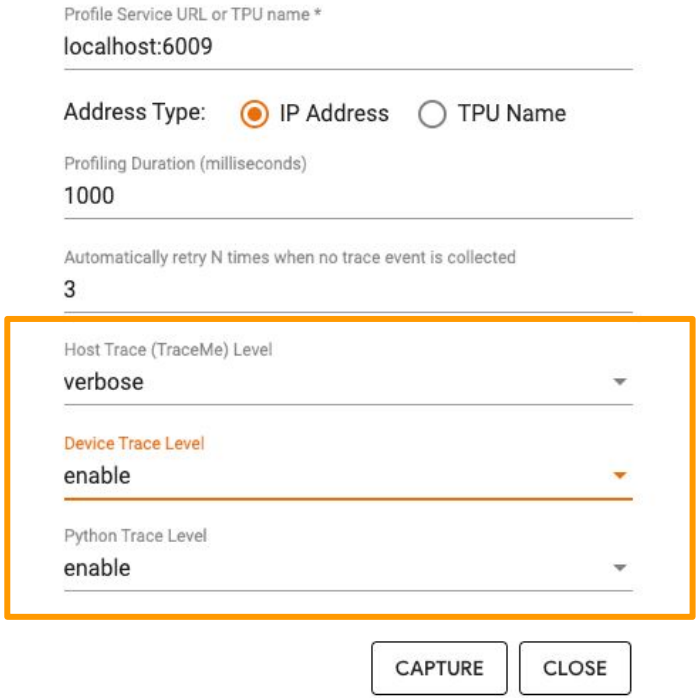
Memory Breakdown Table

Operation

Note: Showing active memory allocations at peak usage within the profiling window. To avoid sluggishness, only the allocations with size over 1MiB are shown in the table below.

Op Name	Allocation Size (GiBs)	Requested Size (GiBs)	Occurrences	Region type	Data type	Shape
preallocated/unknown	3.926	3.926	1	persist/dynamic	INVALID	unknown
gradient_tape/bert_span_labeler/transformer_encoder/transformer/layer_6/self_attention/masked_softmax_6/mul_1	0.035	0.018	1	output	half	[4,16,384,384]
gradient_tape/bert_span_labeler/transformer_encoder/transformer/layer_0/self_attention/masked_softmax/mul_1	0.023	0.018	1	output	half	[4,16,384,384]
gradient_tape/bert_span_labeler/transformer_encoder/transformer/layer_1/self_attention/masked_softmax_1/mul_1	0.023	0.018	1	output	half	[4,16,384,384]
gradient_tape/bert_span_labeler/transformer_encoder/transformer/layer_10/self_attention/masked_softmax_10/mul_1	0.023	0.018	1	output	half	[4,16,384,384]
gradient_tape/bert_span_labeler/transformer_encoder/transformer/layer_11/self_attention/masked_softmax_11/mul_1	0.023	0.018	1	output	half	[4,16,384,384]
gradient_tape/bert_span_labeler/transformer_encoder/transformer/layer_12/self_attention/masked_softmax_12/mul_1	0.023	0.018	1	output	half	[4,16,384,384]
gradient_tape/bert_span_labeler/transformer_encoder/transformer/layer_13/self_attention/masked_softmax_13/mul_1	0.023	0.018	1	output	half	[4,16,384,384]
gradient_tape/bert_span_labeler/transformer_encoder/transformer/layer_14/self_attention/masked_softmax_14/mul_1	0.023	0.018	1	output	half	[4,16,384,384]

More Trace Levels



- Host Trace Level
  - Control the amount of host trace collected
- Device Trace Level
  - Enable or disable device trace collection
- Python Trace Level
  - Enable or disable python tracer

Cutting-edge Research

in TensorFlow

Two topics

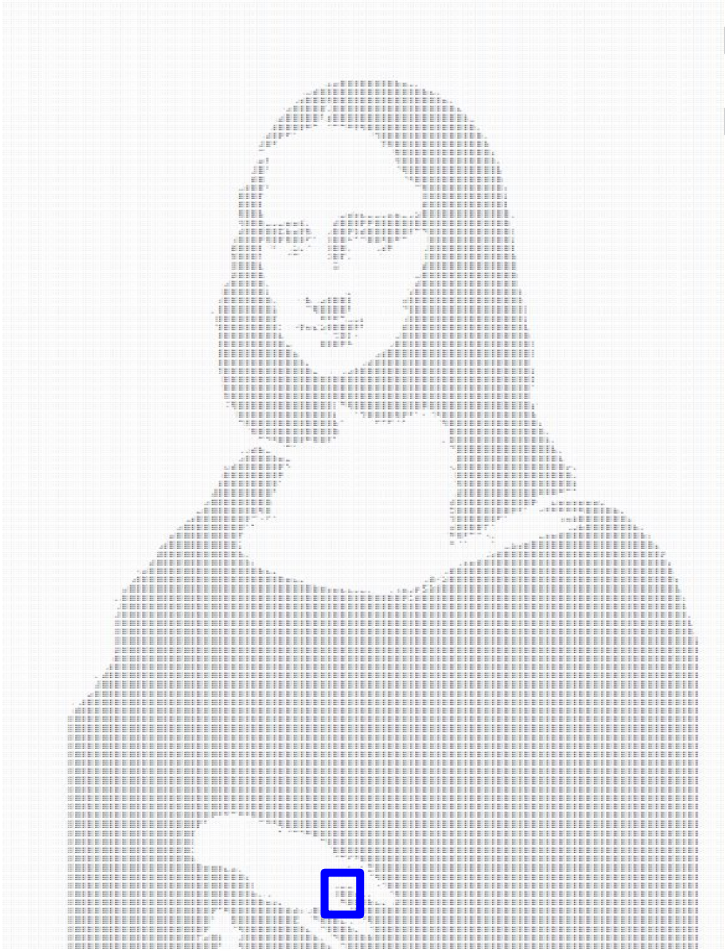
- Measuring privacy guarantees with TensorFlow Privacy

- Reconstructing famous landmarks from vacation photos with NeRF-W

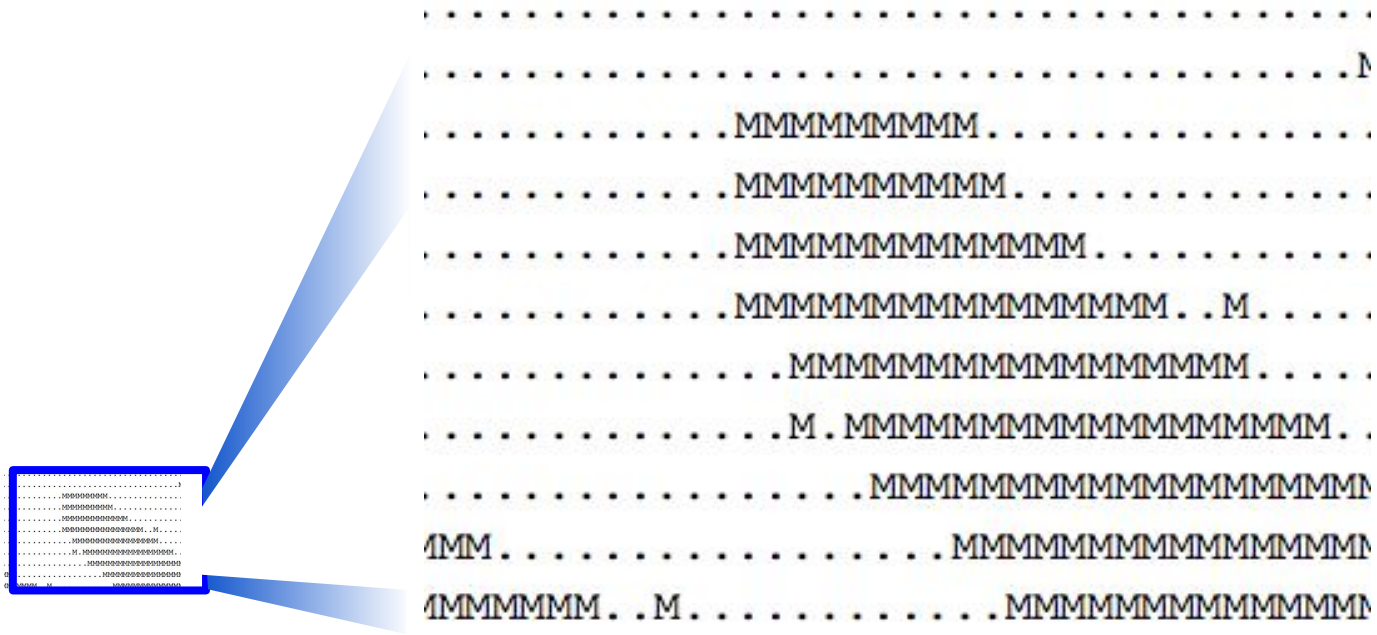
TensorFlow Privacy

What's in your model?

What does it mean to learn with privacy?

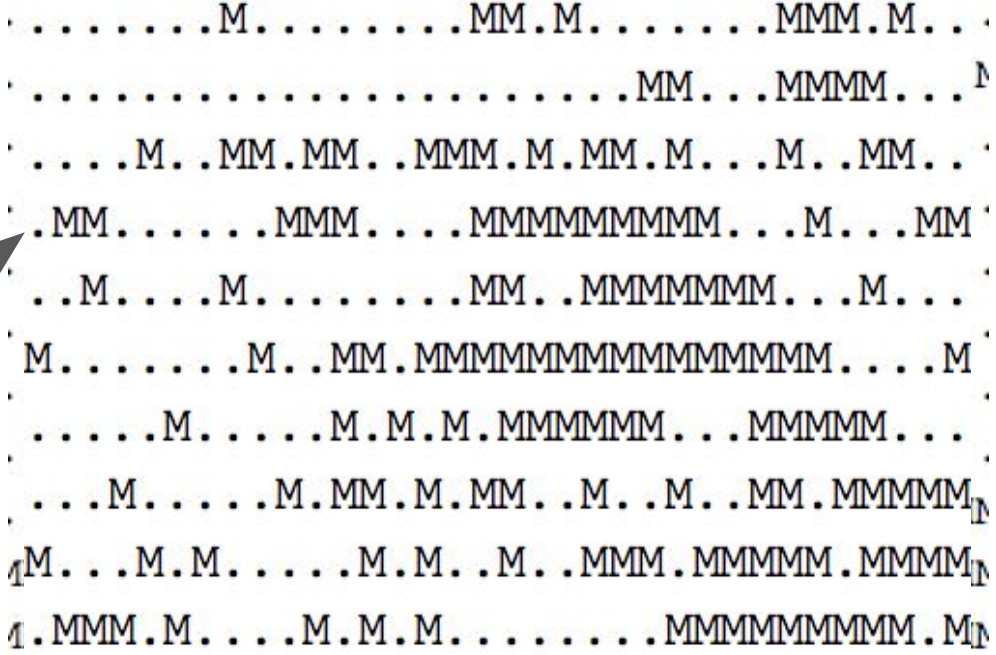


Training data consists of individual examples



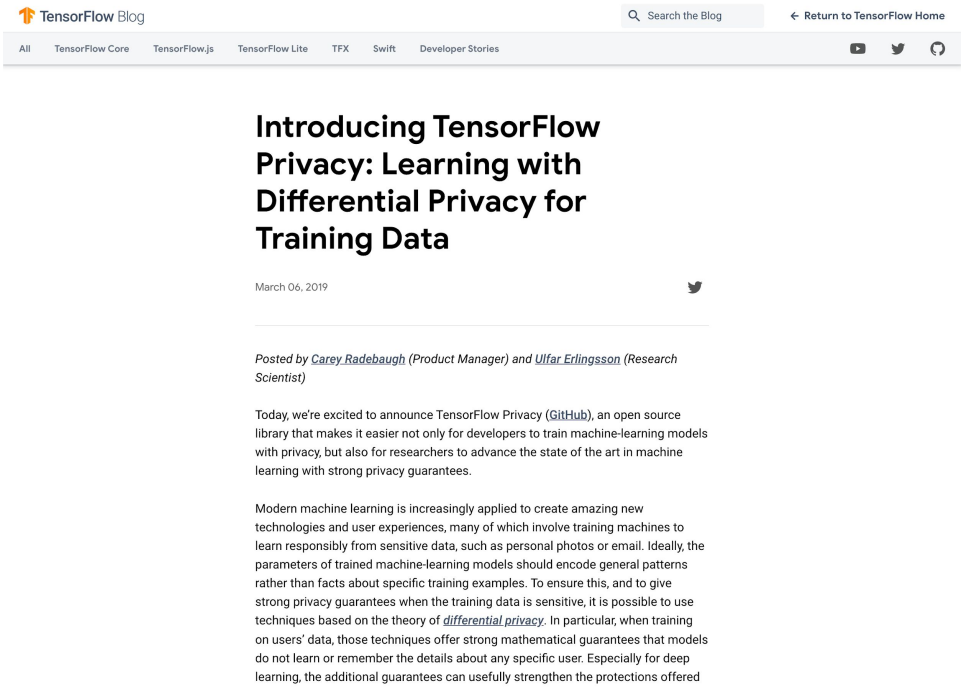
Let's add noise to the examples

Each bit is flipped with probability 25%



Can still learn:
Big picture remains

TensorFlow Privacy



TensorFlow Blog

Search the Blog

← Return to TensorFlow Home

All TensorFlow Core TensorFlow.js TensorFlow Lite TFX Swift Developer Stories

Introducing TensorFlow Privacy: Learning with Differential Privacy for Training Data

March 06, 2019

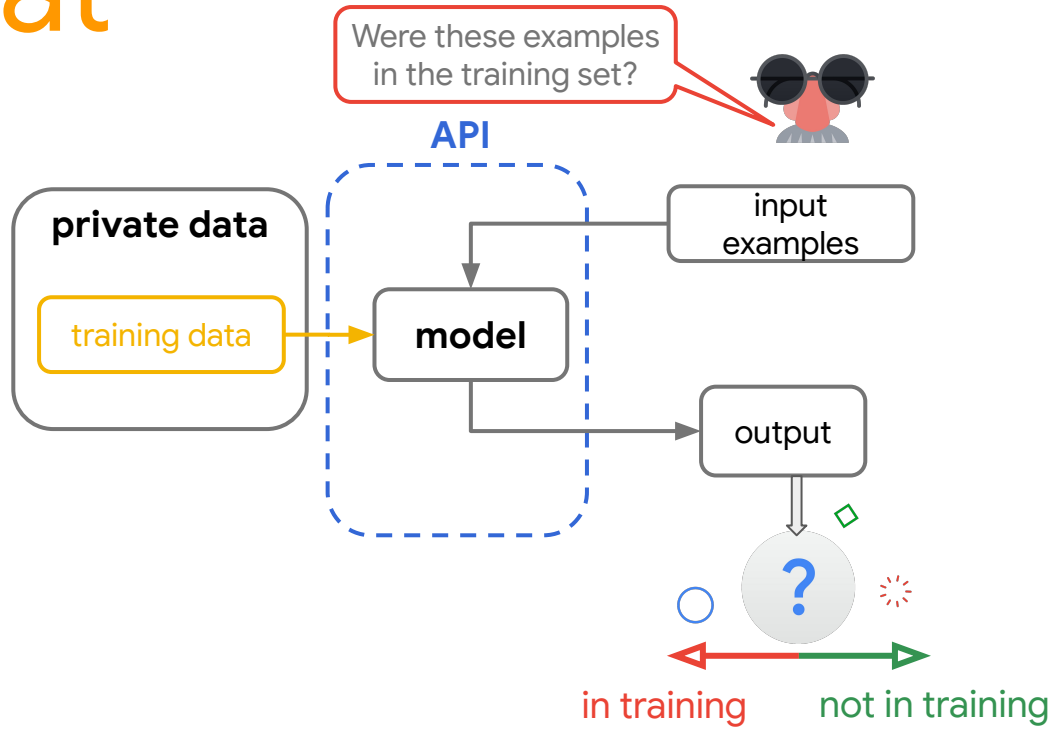
Posted by Carey Radebaugh (Product Manager) and Ulfar Erlingsson (Research Scientist)

Today, we're excited to announce TensorFlow Privacy (GitHub), an open source library that makes it easier not only for developers to train machine-learning models with privacy, but also for researchers to advance the state of the art in machine learning with strong privacy guarantees.

Modern machine learning is increasingly applied to create amazing new technologies and user experiences, many of which involve training machines to learn responsibly from sensitive data, such as personal photos or email. Ideally, the parameters of trained machine-learning models should encode general patterns rather than facts about specific training examples. To ensure this, and to give strong privacy guarantees when the training data is sensitive, it is possible to use techniques based on the theory of *differential privacy*. In particular, when training on users' data, those techniques offer strong mathematical guarantees that models do not learn or remember the details about any specific user. Especially for deep learning, the additional guarantees can usefully strengthen the protections offered

Our library now measures model memorization!

Measures the likelihood that your model leaks data



Regular SGD training

```
vector_loss = tf.nn.sparse_softmax_cross_entropy_with_logits(
    labels=labels, logits=logits)
scalar_loss = tf.reduce_mean(vector_loss)

optimizer = GradientDescentOptimizer(
    learning_rate=0.25)

train_op = optimizer.minimize(global_step=global_step,
    loss=scalar_loss)
```

tensorflow.privacy training

```
vector_loss = tf.nn.sparse_softmax_cross_entropy_with_logits(
    labels=labels, logits=logits)


optimizer = dp_optimizer.DPGradientDescentGaussianOptimizer(
    learning_rate=0.25,
    l2_norm_clip=1.5, noise_multiplier=1.3)
train_op = optimizer.minimize(global_step=global_step,
    loss=vector_loss)
```

tensorflow.privacy training

```
vector_loss = tf.nn.sparse_softmax_cross_entropy_with_logits(
    labels=labels, logits=logits)


optimizer = dp_optimizer.DPGradientDescentGaussianOptimizer(
    learning_rate=0.25,
    l2_norm_clip=1.5, noise_multiplier=1.3)
train_op = optimizer.minimize(global_step=global_step,
    loss=vector_loss)
```

Bounds impact of any one example

Level of noise added relative to an example

Can set both parameters to just above 1.0, in practice

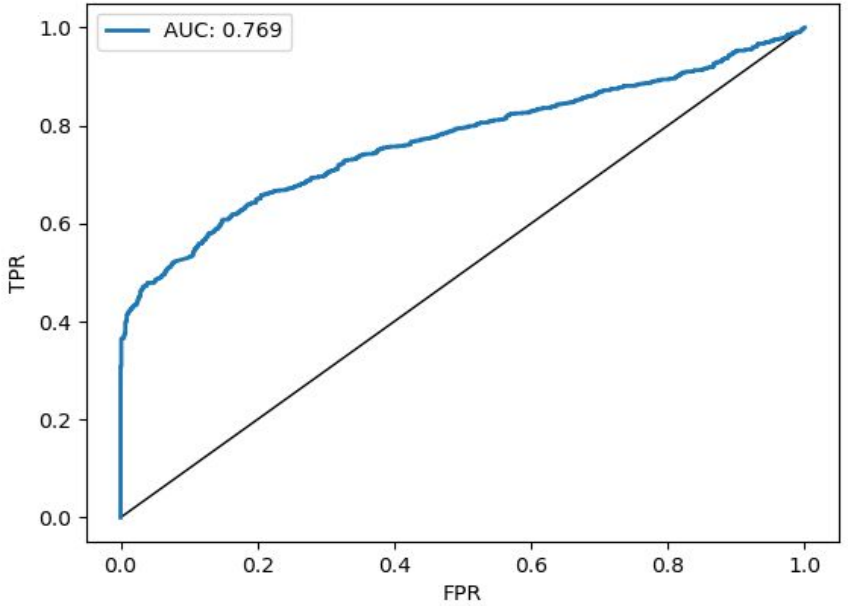
Tensorflow.privacy evaluation

```
attack_results = mia.run_attacks(
    AttackInputData(
        labels_train=training_labels,
        labels_test=test_labels,
        logits_train=training_pred,
        logits_test=test_pred,
        loss_train=crossentropy(training_labels, training_pred),
        loss_test=crossentropy(test_labels, test_pred)),
    SlicingSpec(entire_dataset=True, by_class=True),
    attack_types=(AttackType.THRESHOLD_ATTACK,
AttackType.LOGISTIC_REGRESSION),
    privacy_report_metadata=None)
```

AUC indicates attack's likelihood

↑AUC = ↓Privacy

↓AUC = ↑Privacy



What's next?

- New privacy attacks
- More partnerships with academic institutions like Princeton. If you're interested reach out to tf-privacy@google.com

Check it out!

github.com/tensorflow/privacy

NeRF in the Wild

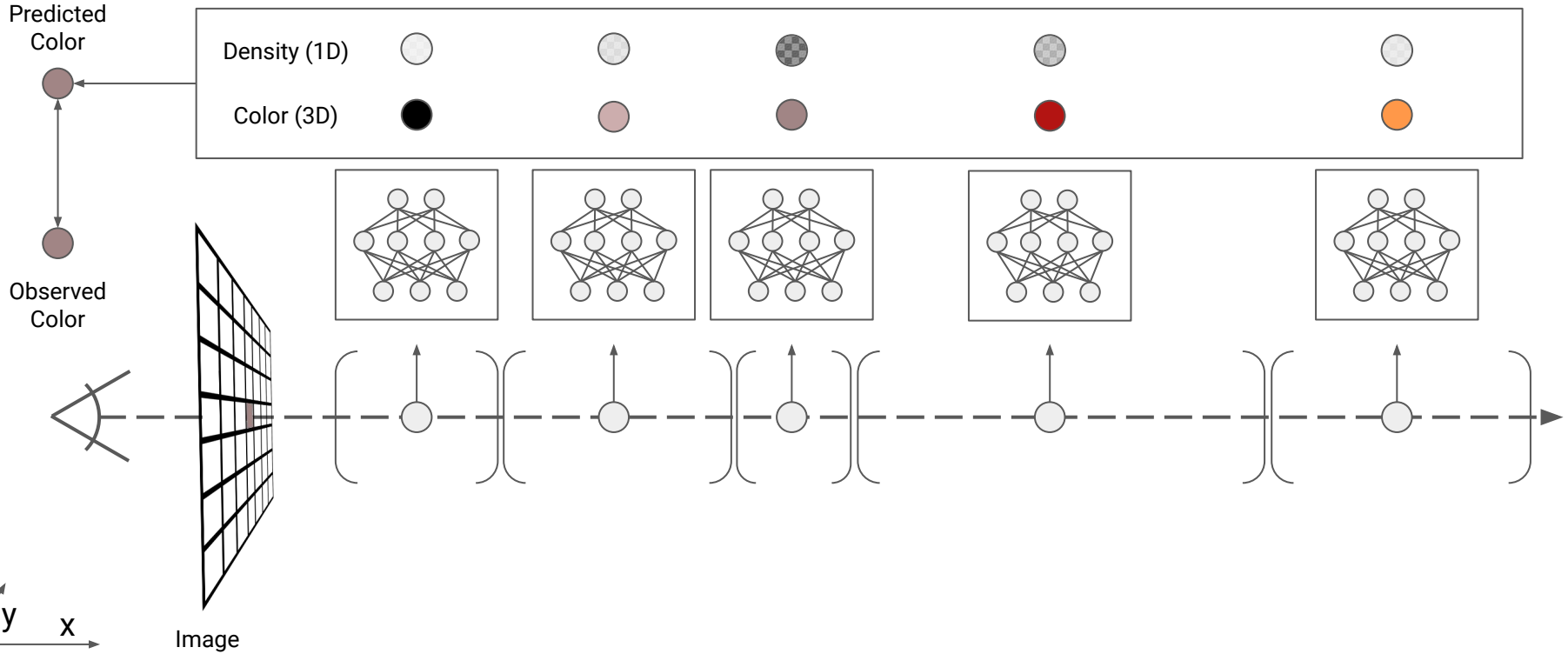
Neural Radiance Fields for Unconstrained Photo Collections

Problem

Internet Photography

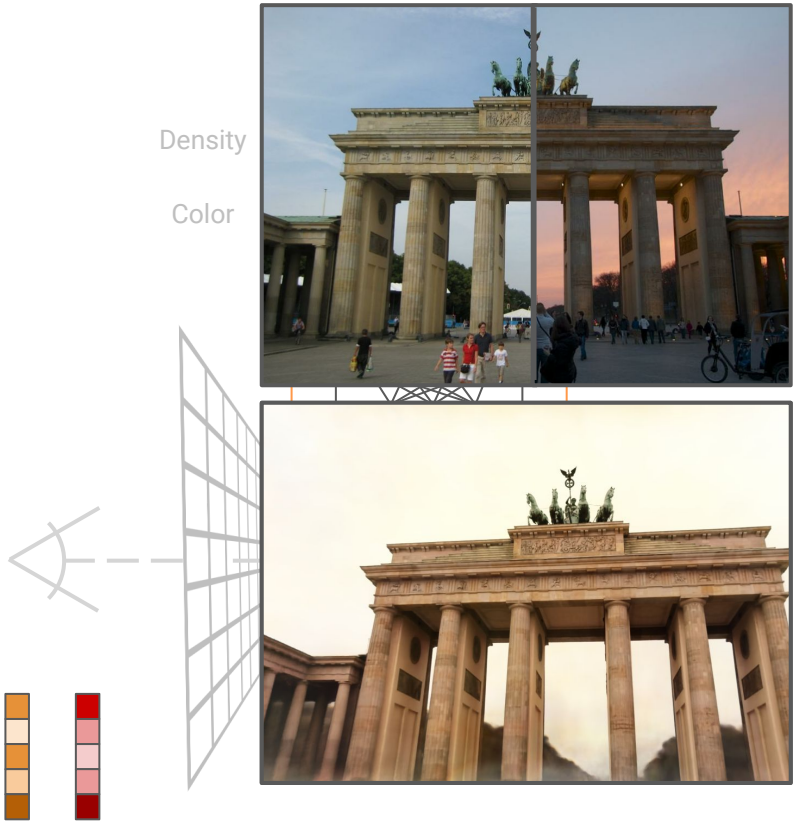
3D Reconstruction

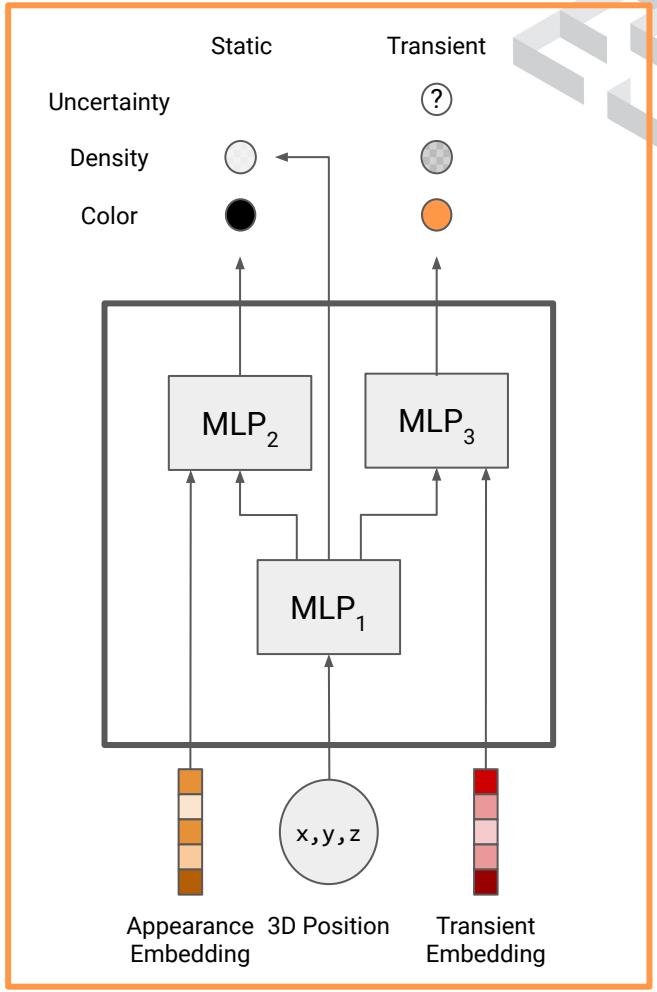
Method

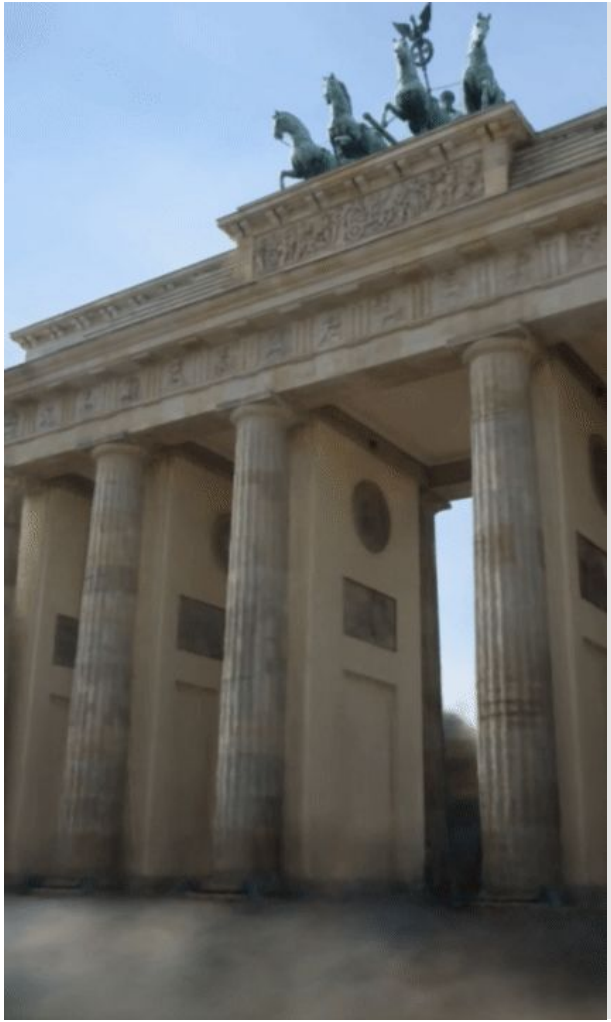

Predicted Color
Density (1D)
Color (3D)
Observed Color
z
y
x
Image

```python
import tensorflow as tf


def render_pixel(boundaries, color, density):
    assert len(boundaries) - 1 == len(color) == len(density)
    n = len(boundaries) - 1
    predicted_color = tf.zeros([3])  # RGB color
    accumulated_opacity = 1.0
    for i in range(n):
        length = boundaries[i+1] - boundaries[i]
        opacity = 1. - tf.math.exp(-1. * tf.nn.relu(density[i]) * length)
        predicted_color += opacity * color[i] * accumulated_opacity
        accumulated_opacity *= (1. - opacity)
    return predicted_color
```

Method


Density
Color


Static
Transient
Uncertainty
Density
Color
MLP2
MLP3
MLP1
x,y,z
Appearance Embedding
3D Position
Transient Embedding

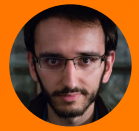


TensorFlow

Website	https://nerf-w.github.io
Paper	https://arxiv.org/abs/2008.02268
YouTube	https://youtu.be/yPKIxoN2Vf0
Code	https://github.com/bmild/nerf

Ricardo Martin-Brualla*
Google Research, Seattle

Noha Radwan*
Google Research, Berlin



Mehdi S. M. Sajjadi*
Google Research, Berlin

Jonathan T. Barron
Google Research, San Francisco

Alexey Dosovitskiy
Google Research, Berlin

Daniel Duckworth
Google Research, Berlin

Wrap-up

What’s next in TF&DL …

Training

JumpStart Workshop : Red Dragon AI (through SGInnovate)

- First part of Full Developer Course
  - Dates : February 22, 23 (live online)
  - 2 week-days + online content
  - Play with real models & Pick-a-Project
  - Certificates, etc
- Cost is heavily subsidised for SC/PR!
- https://www.sginnovate.com/talent-development

Next Events...

The Show Must Go On!

- https://www.meetup.com/TensorFlow-and-Deep-Learning-Singapore/
- Trying out different formats
  - Beginner-friendly code & idea walk-throughs
    - Expected : 2 March 2021
  - Latest-Research roundups (i.e. Deep Learning focus)
    - Expected : 16 March 2021

Thank You!

Let us know what else you'd like to see...

- https://www.meetup.com/TensorFlow-and-Deep-Learning-Singapore/
  - As always : We welcome Lightning Talk suggestions!