TensorFlow

What's New on TensorFlow 2.4?

Advanced features!

Carlos Timoteo
Data Scientist
GDE in ML
chmstimoteo@gmail.com

New features

Model building

- tf.experimental.numpy
- Keras Preprocessing layers
- TF Recommenders library

Performance and debugging

- New features in tf.data: Service and Snapshot
- Improvements in the TensorFlow Profiler

TensorFlow

tf.experimental.numpy

Accelerate NumPy using TensorFlow

NumPy works with TensorFlow

You can now:

- Run a **subset** of full NumPy spec 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

```python
# 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)
```

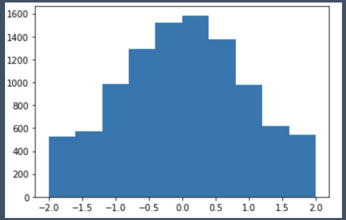
```
/job:localhost/replica:0/task:0/device:GPU:0
```

```python
# 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.
```

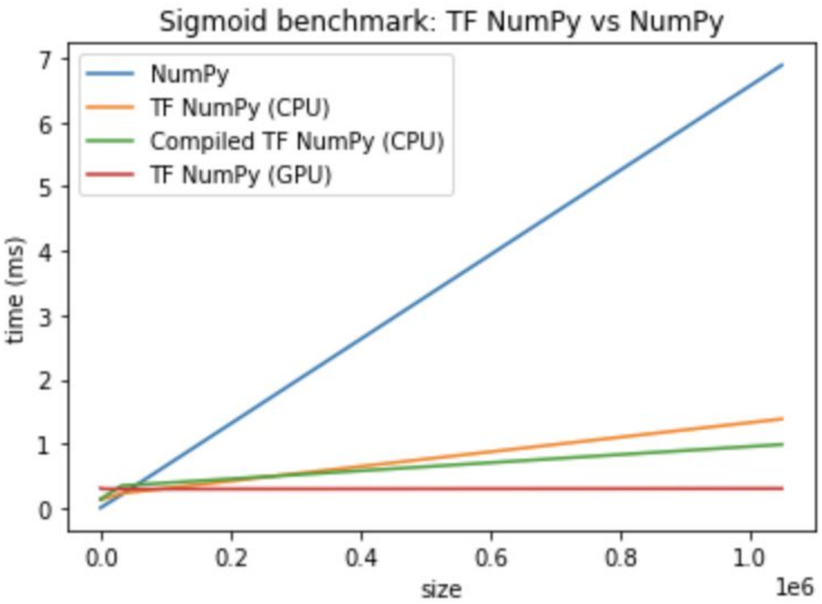
```python
# Available in TensorFlow Nightly
# `pip install tf-nightly`
import tensorflow.experimental.numpy as tnp

# 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
```

```python
# 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

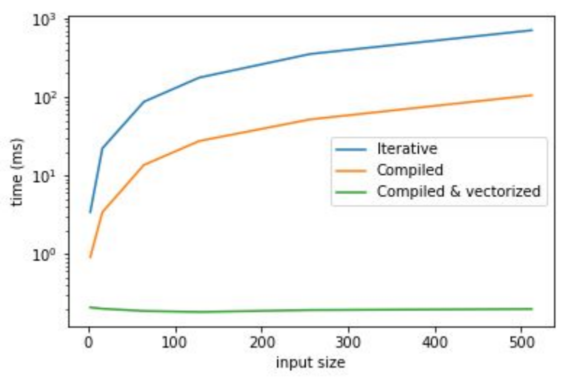
```python
# Use NumPy code in input pipelines
dataset = tf.data.Dataset.from_tensor_slices(tnp.random.randn(1000, 1024)).map(
    lambda z: z.clip(-1, 1)).batch(100)
```

```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 = 1 / 1 + tnp.exp(-output)
  return tape.gradient(tnp.sum(output), wt)  # Also see tape.batch_jacobian
```

```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 = 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)
```

```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)  # 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
```

```python
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
- 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

Keras preprocessing layers

Build end-to-end models

Background

Writing preprocessing logic is time consuming

- When building a model to classifies text, you need to write preprocessing logic for standardization, tokenization, and vectorization.

- To deploy that model, you need to ensure that text is preprocessed in exactly the same way.

- This can result in code duplication for complex logic, that's difficult to maintain.

Keras preprocessing layers

A **user-friendly** way to include preprocessing logic as layers inside your model. Enables you to:

- Save models that take raw images, text, or structured data as input
- Deploy models without needing to re-implement preprocessing logic server-side, i.e. preventing training-serving skew
- Easily run image data augmentation on accelerators

```python
train_texts = tf.data.Dataset.from_tensor_slices(["foo", "bar", "baz"])
max_features = 5000  # Maximum vocab size.
sequence_length = 4  # Sequence length to pad the outputs to.
embedding_dims = 2

# Create a text preprocessing layer
vectorize_layer = TextVectorization(
    max_tokens=max_features,
    output_mode='int',
    output_sequence_length=sequence_length)
```

```python
# Create a text preprocessing layer
vectorize_layer = TextVectorization(
    max_tokens=max_features,
    output_mode='int', # Outputs integer indices, one integer index per split token
    output_sequence_length=sequence_length)

# Adapt it your vocabulary
# Now that the vocab layer has been created, call `adapt` on the text-only
# dataset to create the vocabulary. You don't have to batch, but for large
# datasets this means we're not keeping spare copies of the dataset.
vectorize_layer.adapt(train_texts)


vectorize_layer.get_vocabulary()
# ['', '[UNK]', 'foo', 'bar', 'baz']
```

```python
# Create a text preprocessing layer
vectorize_layer = TextVectorization(
    max_tokens=max_features,
    output_mode='int',
    output_sequence_length=sequence_length)

# Adapt it your vocabulary
vectorize_layer.adapt(train_texts)

# Include it inside your model
model = tf.keras.Sequential([
  tf.keras.Input(shape=(1,), dtype=tf.string),
  vectorize_layer,
  layers.Embedding(max_features + 1, 32),
  layers.Dropout(0.2),
  layers.GlobalAveragePooling1D(),
  layers.Dropout(0.2),
  layers.Dense(1)])
```

```python
# Create a text preprocessing layer
vectorize_layer = TextVectorization(
    max_tokens=max_features,
    output_mode='int',
    output_sequence_length=sequence_length)

# Adapt it your vocabulary
vectorize_layer.adapt(train_texts)

# Include it inside your model
model = tf.keras.Sequential([
  tf.keras.Input(shape=(1,), dtype=tf.string),
  vectorize_layer,
  layers.Embedding(max_features + 1, 32),
  layers.Dropout(0.2),
  layers.GlobalAveragePooling1D(),
  layers.Dropout(0.2),
  layers.Dense(1)])

# Save a model that accepts raw strings as input
model.save('path/to/location')
```

```python
# Include it inside your model
model = tf.keras.Sequential([
  tf.keras.Input(shape=(1,), dtype=tf.string),
  vectorize_layer,

# Create a data augmentation stage with horizontal flipping, rotations, zooms
data_augmentation = tf.keras.Sequential([RandomFlip("horizontal"),
                                         RandomRotation(0.1), RandomZoom(0.1),])


# Create a model that includes the augmentation stage
inputs = tf.keras.Input(shape=input_shape)
x = data_augmentation(inputs)
# Rescale image values to [0, 1]
x = Rescaling(1.0/255)(x)
# Add the rest of the model
outputs = tf.keras.applications.ResNet50(weights=None,
                                         input_shape=(32, 32, 3), classes=10)(x)

model = tf.keras.Model(inputs, outputs)
```

Out of the box support for common data types

Text

- Standardize, tokenize, and vectorize

Images

- Resize, normalize, and run data augmentation on the GPU

Structured data

- Support for numeric and categorical attributes. One hot encode, hash, discretize, bucketize, category crossing and more.

API doc: tensorflow.org/api_docs/python/tf/keras/layers/experimental/preprocessing

Learn more

Developer guide and complete examples

- keras.io/guides/preprocessing_layers/
- tensorflow.org/tutorials/images/classification,
- tensorflow.org/tutorials/keras/text_classification
- keras.io/examples/structured_data/structured_data_classification_from_scratch/

TensorFlow

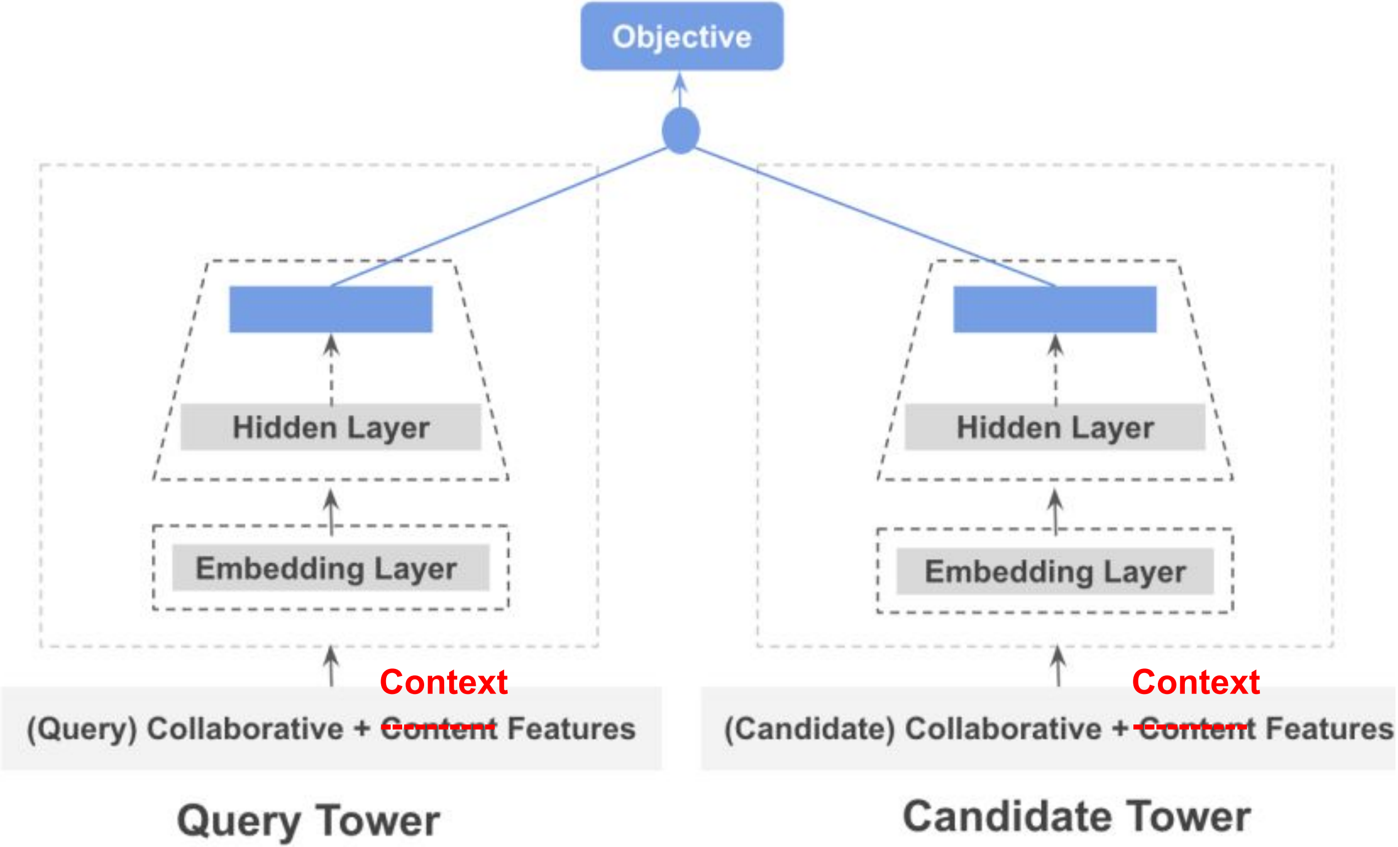
TensorFlow Recommenders

Build recommender systems with TensorFlow

TensorFlow Recommenders

- A library of components to build flexible nomination and ranking models.
- Built on TensorFlow 2.0.
- Seamlessly integrates with the TensorFlow ecosystem for training and serving
- Enables you to easily build deep recommendation models that have many advantages over traditional matrix factorization approaches.

Two-Tower Recommender Example



```python
import tensorflow_datasets as tfds
import tensorflow_recommenders as tfrs

# Data on ratings.
ratings = tfds.load("movielens/100k-ratings", split="train")
# Movie features.
movies = tfds.load("movielens/100k-movies", split="train")
```

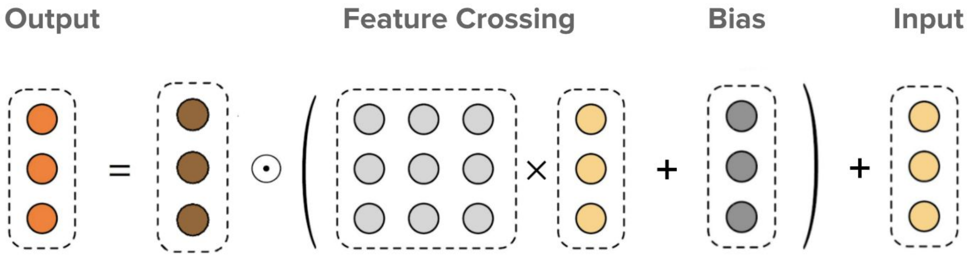
```python
import tensorflow_datasets as tfds
import tensorflow_recommenders as tfrs

# Data on ratings.
ratings = tfds.load("movielens/100k-ratings", split="train")
# Movie features.
movies = tfds.load("movielens/100k-movies", split="train")

# The user and movie models can be arbitrary Keras models.
user_model = tf.keras.Sequential([
    tf.keras.layers.Embedding(1000, 32),
    tf.keras.layers.Dense(64, activation=”relu”)
])

# tfrs.layers.embedding.TPUEmbedding(feature_config=feature_config,
#  optimizer='sgd') accelerates embedding lookups for large tables with TPU.
# Spoiler! Supported on TF 2.5.
movie_model = tf.keras.layers.Embedding(1700, 64)
```

```
model = MovielensModel(
  user_model=user_model,
  movie_model=movie_model,
  # The retrieval task will optimize for retrieving the best movies.
  task=tfrs.tasks.Retrieval(
    # Model retrieval accuracy measured across all recommendable movies.
    metrics=tfrs.metrics.FactorizedTopK(
      candidates=movies.batch(128).map(movie_model)
    ))
  )
```

```python
# Train the model.
model.compile(optimizer=tf.keras.optimizers.Adagrad(0.5))
model.fit(ratings.batch(4096), epochs=3)

# Evaluate retrieval performance.
model.evaluate(ratings.batch(4096))
```

Deep & Cross Network (DCNv2)

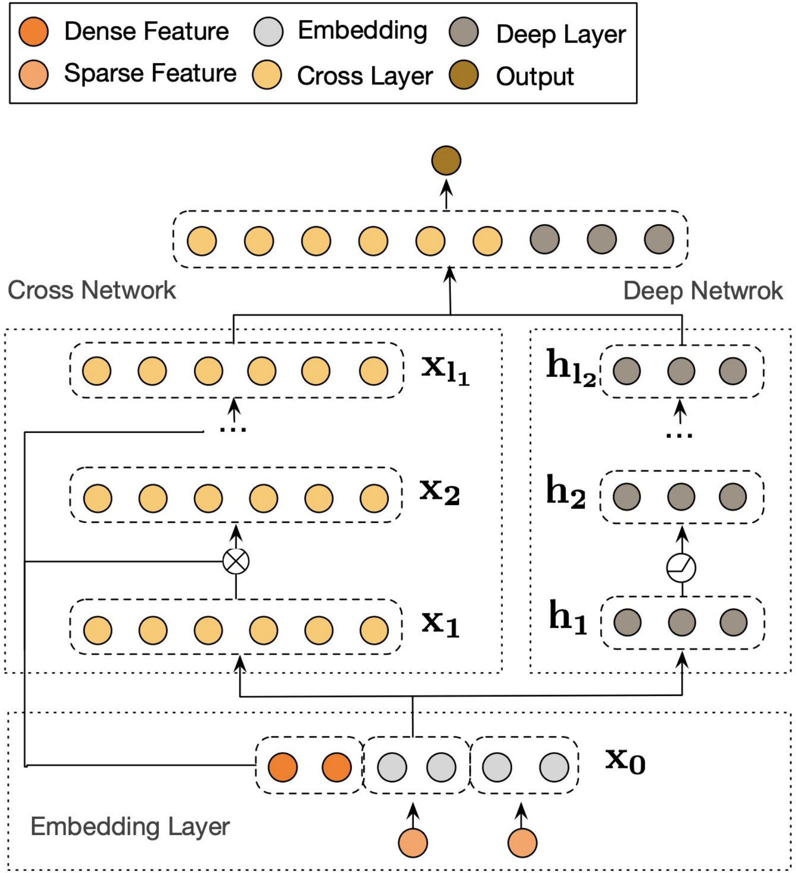
Deep & Cross Network (DCNv2)

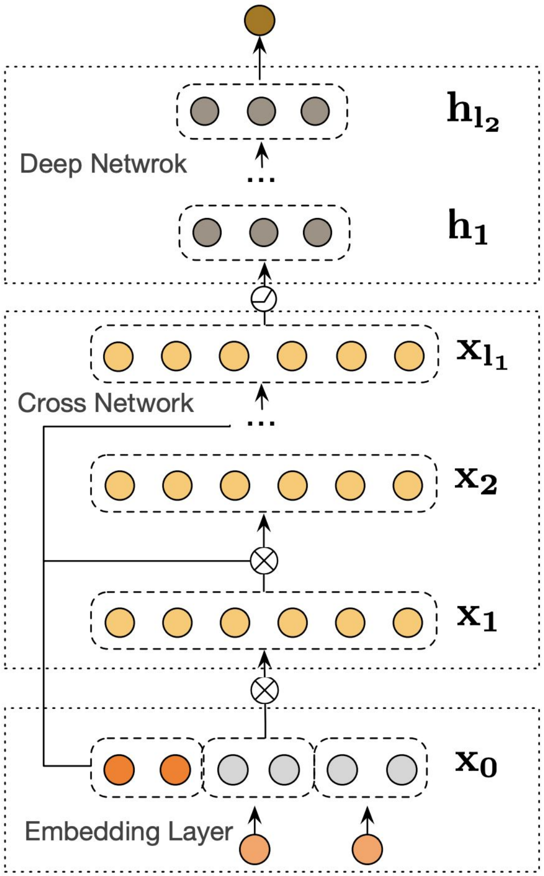


$$x_{i+1} = x_0 \odot (W \times x_i + b) + x_i$$

$$y = f(x_1, x_2, x_3) = 0.1x_1 + 0.4x_2 + 0.7x_3 + 0.1x_1x_2 + 3.1x_2x_3 + 0.1x_3^2$$

Deep & Cross Network (DCNv2)





Learn more about TensorFlow Recommenders

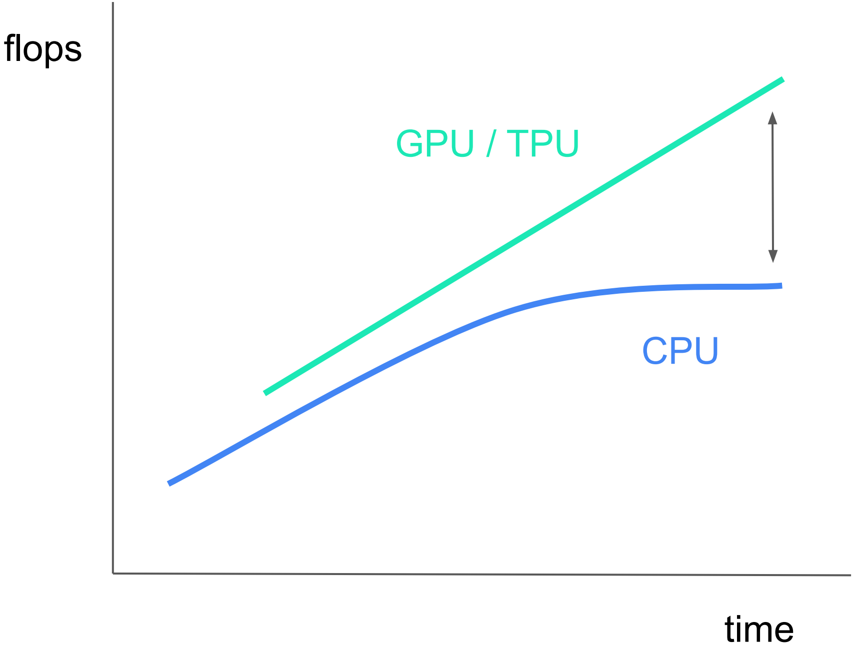
Visit tensorflow.org/recommenders for tutorials and guides

What's new in tf.data?

Snapshot and service

Data processing bottleneck

- Accelerator performance (GPU, TPU) is increasing faster than CPU
- Model complexity is not growing as fast, which means step time is decreasing and more data per step is expected
- Host CPU is unable to keep up with increase in demand



Easy ways to improve performance

Speed up your input pipelines with one line of code

- tf.data.Dataset.prefetch - overlaps upstream computation (e.g. CPU data processing) with downstream computation (e.g. GPU/TPU training)
- tf.data.Dataset.cache - automatically caches your dataset in-memory, or to a file

Learn more with these guides

- Optimize pipeline performance: https://www.tensorflow.org/guide/data_performance
- Using the TF Profiler: https://www.tensorflow.org/guide/data_performance_analysis

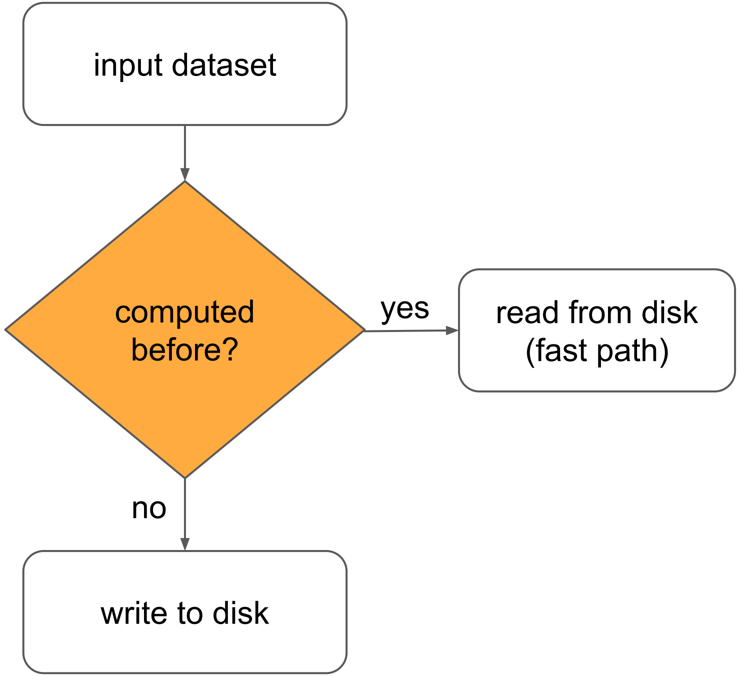
New in TF 2.3

New features

- tf.data.snapshot can help you avoid repetitive data processing
- tf.data.service can help you horizontally scale data processing
- New API for efficient saving and loading of arbitrary tf.data datasets

tf.data snapshot

Reuse computation



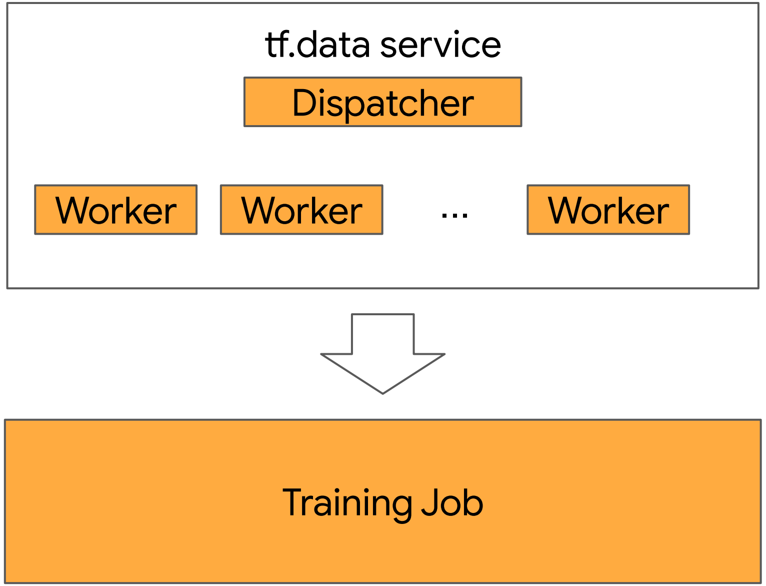
```python
import tensorflow as tf

def preprocess(record):
  ...

# Use parallel interleave to optimize I/O when reading from remote BLOB storage
dataset = tf.data.TFRecordDataset(".../*.tfrecord")
dataset = dataset.map(preprocess, num_parallel_calls=tf.data.AUTOTUNE)

dataset = dataset.shuffle(buffer_size=1024)
dataset = dataset.batch(batch_size=32)
dataset = dataset.prefetch(tf.data.AUTOTUNE)

model = tf.keras.Model(...)
model.fit(dataset)
```

```
import tensorflow as tf

def preprocess(record):
  ...


dataset = tf.data.TFRecordDataset(".../*.tfrecord")
dataset = dataset.map(preprocess, num_parallel_calls=tf.data.AUTOTUNE)
dataset = dataset.snapshot("/path/to/snapshot_dir")
dataset = dataset.shuffle(buffer_size=1024)
dataset = dataset.batch(batch_size=32)
dataset = dataset.prefetch(tf.data.AUTOTUNE)

model = tf.keras.Model(...)
model.fit(dataset)
```

snapshot transformation

tf.data service

Scale horizontally



```python
import tensorflow as tf

def preprocess(record):
    ...


dataset = tf.data.TFRecordDataset(".../*.tfrecord")
dataset = dataset.shuffle(buffer_size=1024)
dataset = dataset.map(preprocess, num_parallel_calls=tf.data.AUTOTUNE)
dataset = dataset.batch(batch_size=32)

dataset = dataset.prefetch(tf.data.AUTOTUNE)

model = tf.keras.Model(...)
model.fit(dataset)
```

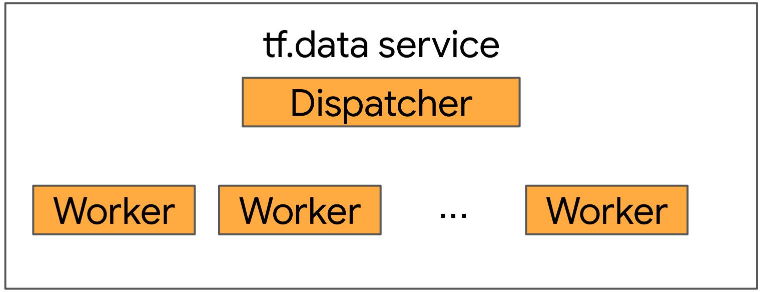
```python
import tensorflow as tf

def preprocess(record):
  ...


dataset = tf.data.TFRecordDataset(".../*.tfrecord")
dataset = dataset.shuffle(buffer_size=1024)
dataset = dataset.map(preprocess, num_parallel_calls=tf.data.AUTOTUNE)
dataset = dataset.batch(batch_size=32)
dataset = dataset.distribute(“<master_address>”)
dataset = dataset.prefetch(tf.data.AUTOTUNE)

model = tf.keras.Model(...)
model.fit(dataset)
```

distribute transformation

Deploying tf.data service

tensorflow.org has API docs for DispatchServer and WorkerServer.

tensorflow/ecosystem/data_service provides a full example of running tf.data service on Google Kubernetes Engine.



```python
import tensorflow as tf

def preprocess(record):
  ...


dataset = tf.data.TFRecordDataset(".../*.tfrecord")
dataset = dataset.shuffle(buffer_size=1024)
dataset = dataset.map(preprocess, num_parallel_calls=tf.data.AUTOTUNE)
dataset = dataset.batch(batch_size=32, num_parallel_calls=tf.data.AUTOTUNE)
dataset = dataset.distribute("<master_address>")
dataset = dataset.prefetch(tf.data.AUTOTUNE)

model = tf.keras.Model(...)
model.fit(dataset)
```

Spoiler! Comes in TF 2.5

Key takeaways

- Training is increasingly bottlenecked on data preprocessing
- Prefetch and cache speed up your pipeline with one line of code
- Snapshot allows reuse of data preprocessing
- Service enables distributed data preprocessing

Learn more

Developer guides

- https://www.tensorflow.org/guide/data_performance
- https://www.tensorflow.org/guide/data_performance_analysis
- **NEW!** https://www.tensorflow.org/guide/data_performance_analysis

TensorFlow

What's new in tf.distribute?

ParameterServerStrategy is now Experimental
MultiWorkerMirroredStrategy is not Experimental anymore

New in TF 2.4

New features

- Introduces experimental support for asynchronous training of models with ParameterServerStrategy
- MultiWorkerMirroredStrategy is now part of the stable API
- Use tf.data API and tf.distribute API to work with tf.distribute.DistributedDataset.

TensorFlow Profiler

Memory Profile Tool

Memory Profile Summary

Memory ID GPU_...
show memory profile for selected device

#Allocation 880

#Deallocation 120

Memory Capacity 13.82 GiBs

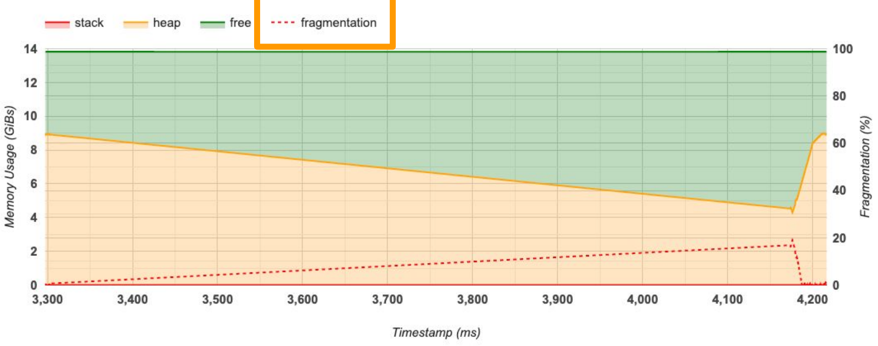
Peak Heap Usage 9.14 GiBs
high water mark in lifetime

Peak Memory Usage 8.98 GiBs
stack + heap, within profiling window

- 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.



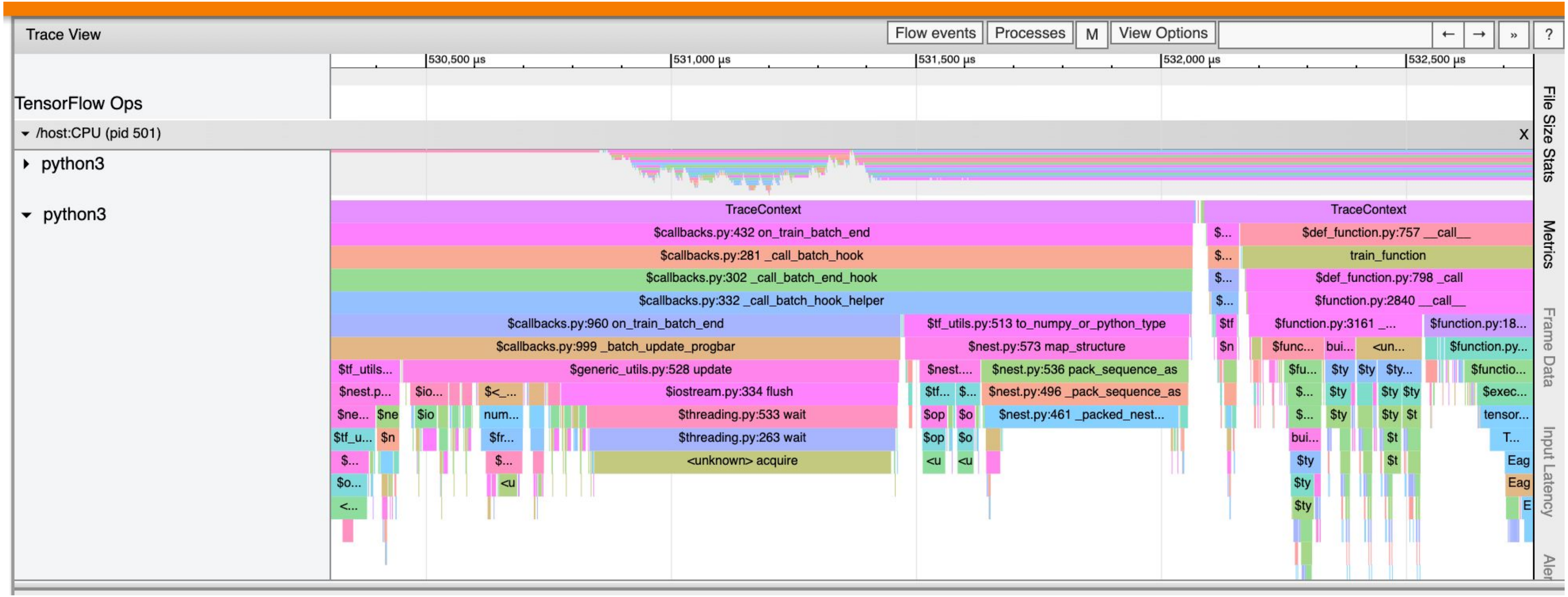
Memory Breakdown Table

Operation

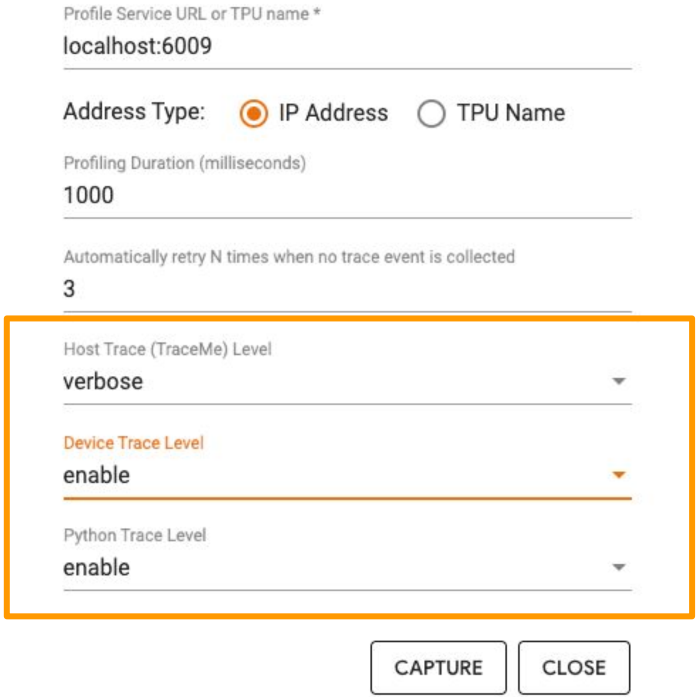
Note: Showing active memory allocations at peak usage within the profiling window. To avoid sluggishness, only the allocations with size over 1MiB are shown in the table below.

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]

Python Tracer



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

And that's a wrap

In review

TensorFlow has a **bunch** of new features to help you build new models and get the performance you need.

- tf.experimental.**numpy**
- **Keras preprocessing** layers
- TF **Recommenders** library
- New features in **tf.data: Service and Snapshot**
- Improvements in the **TensorFlow Profiler**

Thank you!

Learn more

- tensorflow.org/tutorials
- tensorflow.org/guide

Stay up to date

- blog.tensorflow.org
- youtube.com/tensorflow
- twitter.com/tensorflow