Achieving optimal speed on TF 2

Zebroid Meeting 2020-07-21

Andres Potapczynski

Topics for today

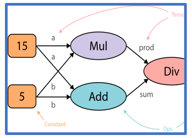
Relevant questions

A What are the major changes from TF1 to TF2?

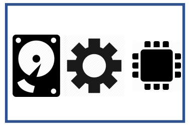
B How does TF2 compare against TF1 and PyTorch?

C What are the most important TF APIs for performance?

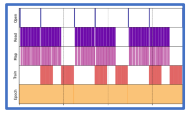
APIs / Tools analyzed



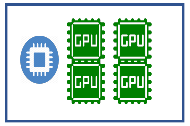
Computational Graph (tf.function)



Input Pipeline (tf.data)



Performance profiler (tf.profiler)



Distributed training (tf.distribute)

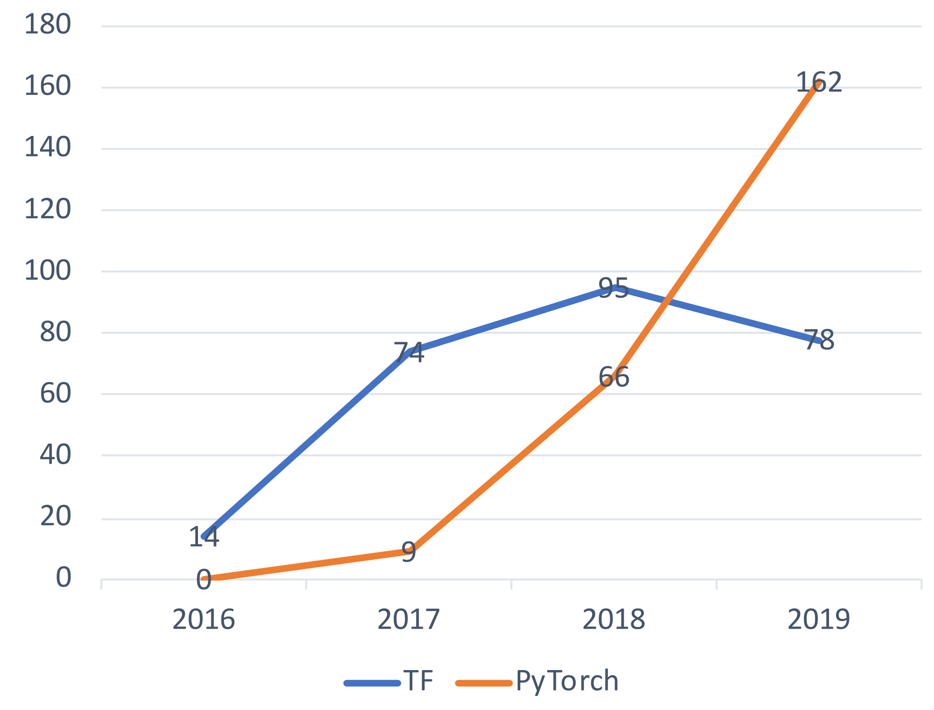
Preamble

Focus

A Similar to PyTorch, TF 2 is easier to debug and more pythonic

PyTorch has been gaining popularity in the community

of NeurIPs papers with implementations on a given framework*



From (TF 1)**

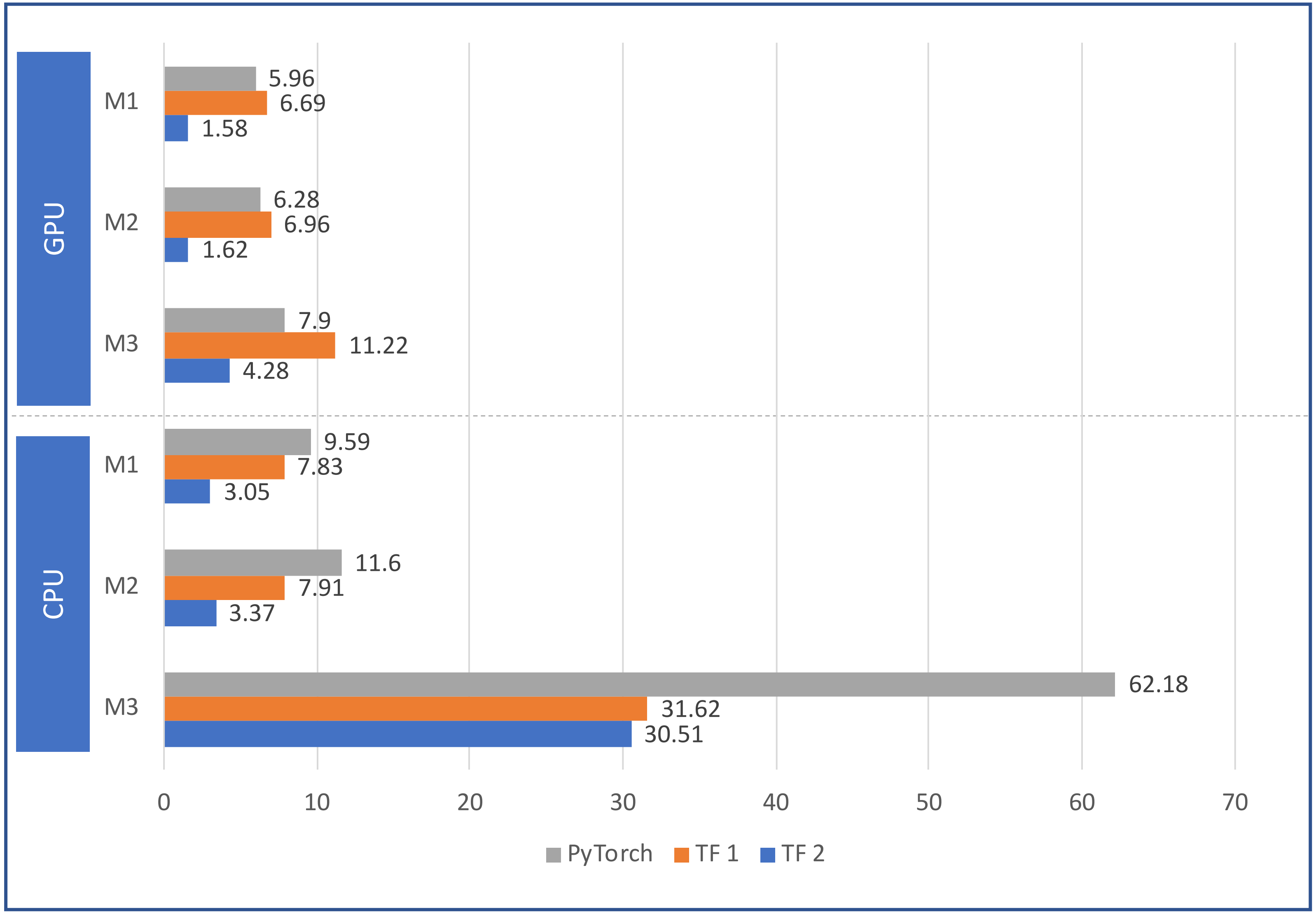
- Using graphs by default
- Defining nodes on a graph
- Constructing a graph with a fixed type in mind
- Using control dependencies to manage the execution
- Initializing variables manually
- Writing non-intuitive control flow statements and functions

To (TF 2)**

- Running code on eager execution (enables per step debugging)
- Selecting operations to run under tf.function
- Having functions that ”adapt” to different types
- Writing code imperatively to control order of execution
- Letting the variables be automatically initialized
- Using python syntax to define control flows (and letting Autograph do the translation)

* http://horace.io/pytorch-vs-tensorflow/, ** tf.function and Autograph (TF Dev Summit 2019) talk

B In principle, TF 2 should not be slower than TF 1 (VAEs)



Models on MNIST

M1 VAE (dense linear architecture)

- Encoder: Dense(784) - Dense(400) – Dense(64)
- Decoder: Dense(64) – Dense(400) – Dense(784)

M2 VAE (dense nonlinear architecture)

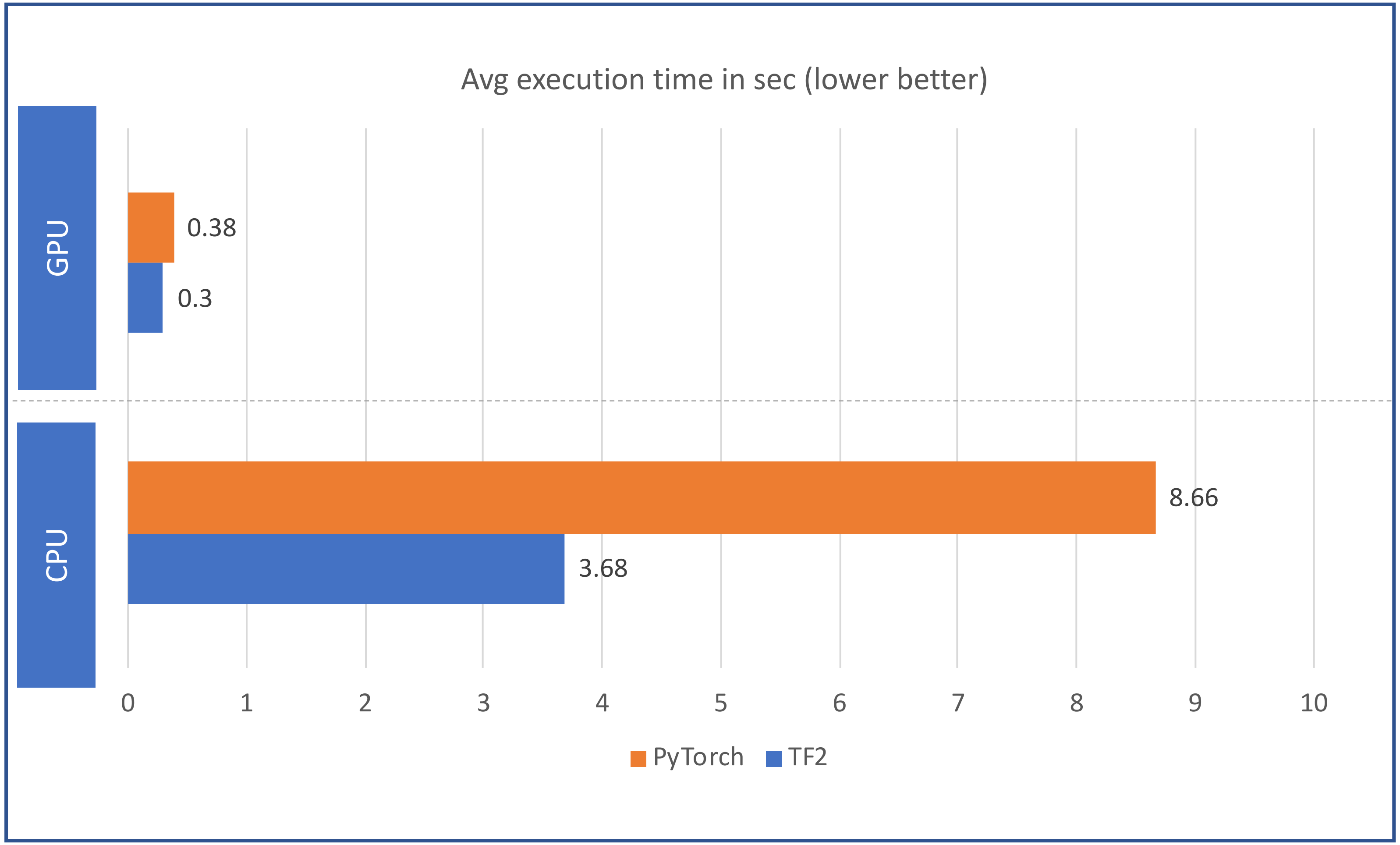
- Encoder: Dense(784) - Dense(400) – Dense (240) – Dense(64)
- Decoder: Dense(64) – Dense(240) – Dense(400) – Dense(784)

M3 VAE (convolutional architecture)

- Encoder: Conv2D(32, 3, 2) - Conv2D(64, 3, 2) – Dense(64).
- Decoder: Dense(1568) - Conv2DT(64, 3, 2) - Conv2DT(32, 3, 2) - Conv2DT(1, 3, 1)

All models ran with a batch size of 128 and latent dimension of 64. All GPUs runs were done in the custom Google Cloud images optimized for each framework

B And in some cases faster than PyTorch



Models

Generator

- 4 Conv2DTranspose layers with Batch Normalization and ReLU activations
- 1 Conv2DTranpose layer with Tanh at the end

Discriminator

- 4 Conv2D layers with Batch Normalization and LeakyReLU activations
- 1 Conv2D layer with Sigmoid at the end

C TF 2 encapsulates several APIs for different tasks

NOT EXHAUSTIVE

High-level DL APIs	tf.keras, tf.estimator
Low-level DL APIs	tf.nn, tf.losses, tf.metrics, tf.optimizers, tf.train, tf.initializers
Autodiff	tf.GradientTape, tf.gradients
I/O and Preprocessing	tf.data, tf.feature_column, tf.audio, tf.image, tf.io, tf.queue
Visualization	tf.summary, tf.profile
Deployment and optimization	tf.distribute, tf.saved_model, tf.autograph, tf.lite, tf.quantization, tf.tpu
Special data structures	tf.lookup, tf.ragged, tf.nest, tf.sparse
Mathematics	tf.math, tf.linalg, tf.signal, tf.random, tf.bitwise

Based on: Geron's *Hands-on Machine Learning with Scikit-Learn, Keras & TensorFlow (2019)*

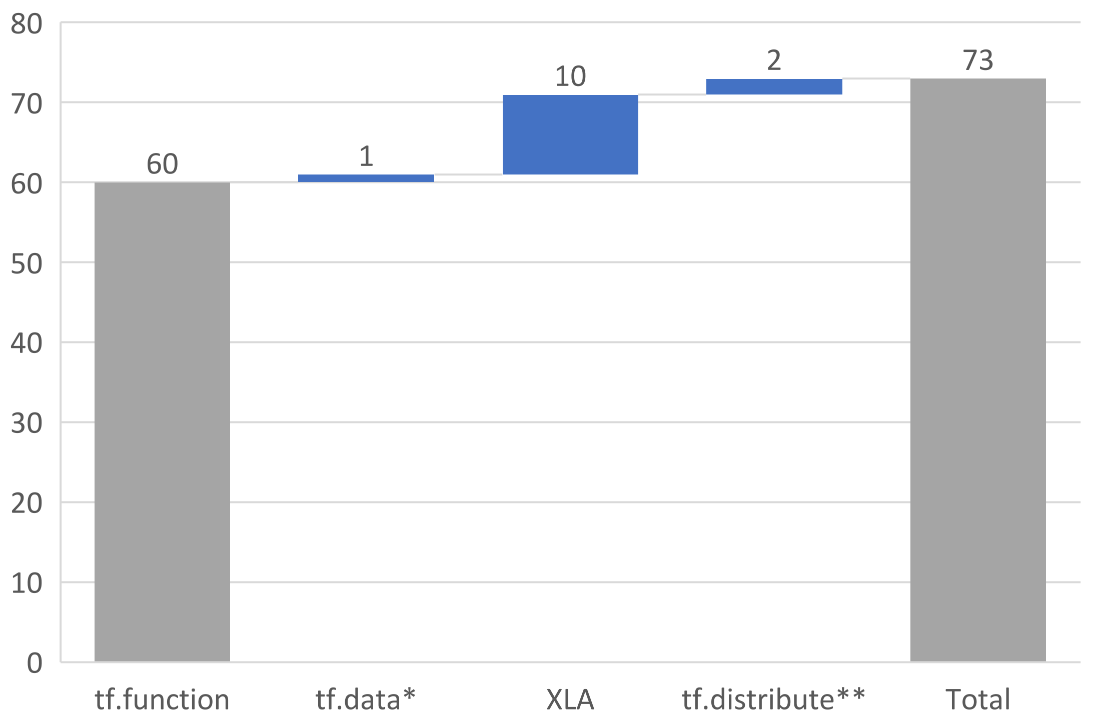
C We focus on the APIs that drive performance

Covered

Activity	API / Tool
• Ensure code is graph executable (optimize graph with XLA)	• tf.function
• Preprocess data efficiently	• tf.data
• Profile bottlenecks	• tf.profiler
• Distribute across GPUs	• tf.distribute
• Accelerate computation by reducing precision	• tf.keras.mixed_precision / tf.keras.quantize
• Shrink your models	• tf.keras.prune

Even small models can benefit from all the optimizations

M3 VAE on MNIST, % reduction in execution time over baseline



* Data optimization consists of caching and prefetching, ** Even for small models, tf.distribute yields some improvement despite its overhead. Distributed across 2 Tesla P4 GPUs

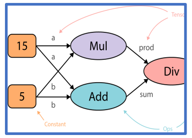
Topics for today

Next

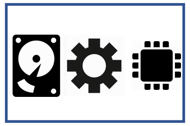
Relevant questions

- **A** What are the major changes from TF1 to TF2?
- **B** How does TF2 compare against TF1 and PyTorch?
- **C** What are the most important TF APIs for performance?

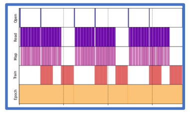
APIs / Tools analyzed



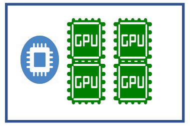
Computational Graph (tf.function)



Input Pipeline (tf.data)



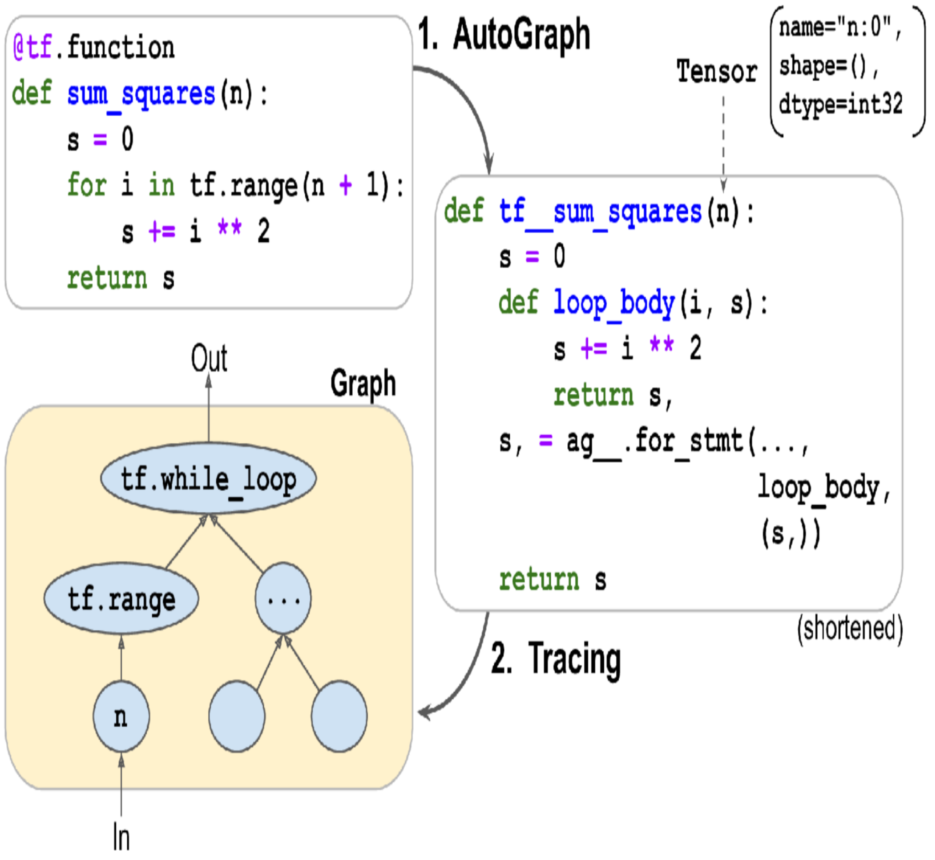
Performance profiler (tf.profiler)



Distributed training (tf.distribute)

tf.function requires some rules to compile properly

Translation of python code to graph by decoration



Common mistakes

- Writing all the code and then trying to decorate it
- Calling external libraries (like NumPy)
- Multiple function retracing from feeding python objects
- Returning non tensor objects
- Creating variables on every call
- Using the tensor's shape value (specially batch size)
- Putting decorators on all functions

Lack of examples and documentation might be hampering the adoption of TF 2!

Based on: Geron's *Hands-on Machine Learning with Scikit-Learn, Keras & TensorFlow (2019)*, https://stackoverflow.com/search?q=tf.function

Topics for today

Next

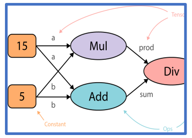
Relevant questions

A What are the major changes from TF1 to TF2?

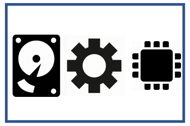
B How does TF2 compare against TF1 and PyTorch?

C What are the most important TF APIs for performance?

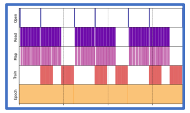
APIs / Tools analyzed



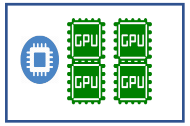
Computational Graph (tf.function)



Input Pipeline (tf.data)



Performance profiler (tf.profiler)



Distributed training (tf.distribute)

tf.data simplifies the input pipeline

Standard code example

Extract

```
files = tf.data.Dataset.list_files(file_pattern)
ds = tf.data.TFRecordDataset(files)
```

Transform

```
ds = ds.shuffle(buffer_size)
ds = ds.repeat(epochs)
ds = ds.map(pre_fn, num_parallel_calls=tf.data.experimental.AUTOTUNE)
ds = ds.batch(batch_size)
ds = ds.cache()
ds = ds.prefetch(tf.data.experimental.AUTOTUNE)
```

Load

```
iterator = ds.make_one_shot_iterator()
x = iterator.get_next()
```

* tf.data: Fast, flexible and easy-to-use input pipelines (https://www.youtube.com/watch?v=uIcqeP7MFH0&t=230s), Inside TensorFlow: tf.data (https://www.youtube.com/watch?v=kVEOCfBy9uY&list=WL&index=2&t=938s)

Topics for today

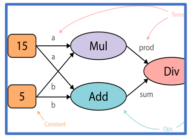
Relevant questions

A What are the major changes from TF1 to TF2?

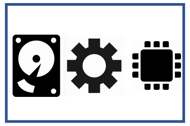
B How does TF2 compare against TF1 and PyTorch?

C What are the most important TF APIs for performance?

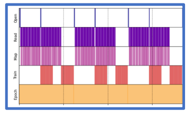
APIs / Tools analyzed



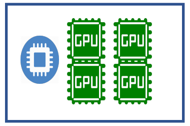
Computational Graph (tf.function)



Input Pipeline (tf.data)



Performance profiler (tf.profiler)



Distributed training (tf.distribute)

Next

tf.profiler helps your model execute faster

NOT EXHAUSTIVE

EXAMPLE

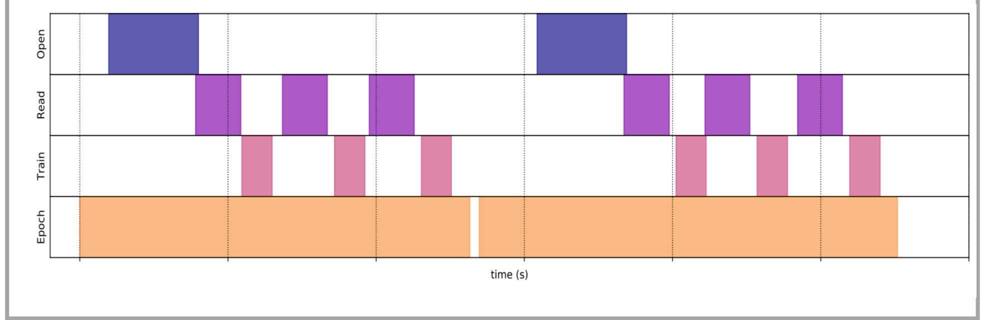
Benefits (tools)

- Shows input bottlenecks (input pipeline analyzer)
- Guides how to speed up parts of training by exhibiting the cost of their ops (TF Stats)
- Breaks down ops per device and exhibits if they are waiting for input

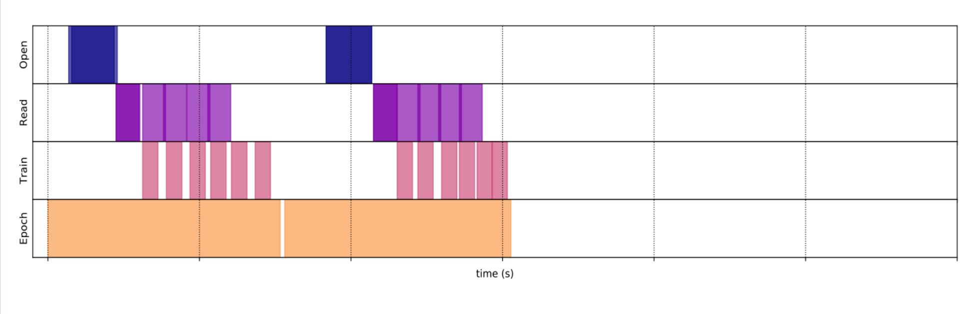
Best Practices*

- Use parallel calls on map, prefetch and cache input
- Utilize the devices more by increasing batch size
- Calculate metrics every few steps and reduce callbacks
- Reduce precision fp16 and make dimension divisible by 8
- Send data to multiple devices in parallel
- Use tf.name_scope to identify most costly ops in python construct

Inadequate trace



Optimized trace



Based on: https://www.tensorflow.org/tensorboard/tensorboard_profiling_keras, * https://www.tensorflow.org/guide/profiler, https://www.tensorflow.org/guide/data_performance_analysis

Topics for today

Next

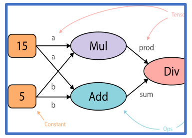
Relevant questions

A What are the major changes from TF1 to TF2?

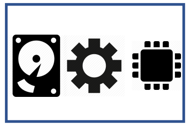
B How does TF2 compare against TF1 and PyTorch?

C What are the most important TF APIs for performance?

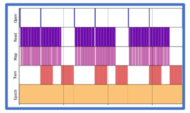
APIs / Tools analyzed



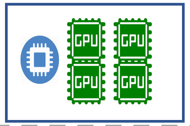
Computational Graph (tf.function)



Input Pipeline (tf.data)

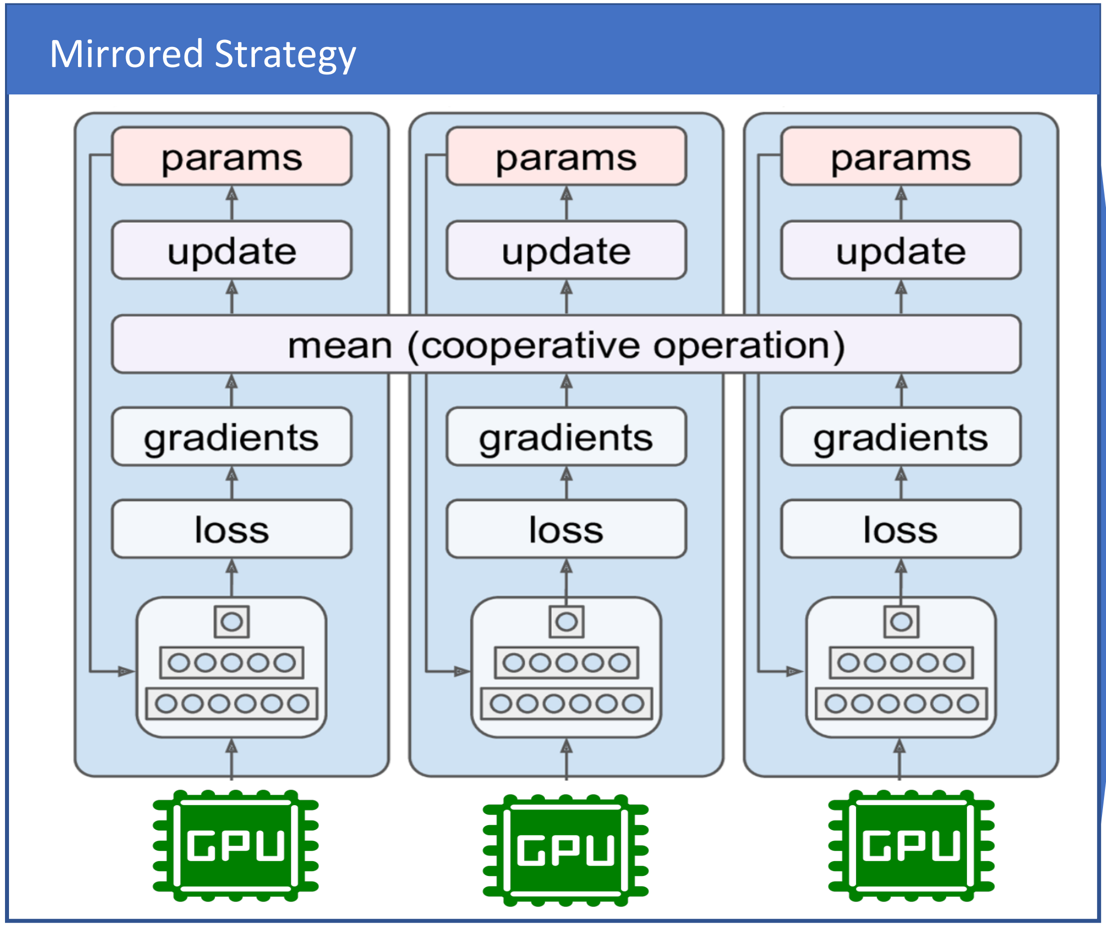


Performance profiler (tf.profiler)



Distributed training (tf.distribute)

tf.distribute is easy to use since TF 2 is “strategy aware”



Code changes

- Change data pipeline by distributing it
- Create model and optimizer under the strategy scope
- Define loss on a per example basis, create distributed train step function and aggregate loss and gradients using global batch size