Sneak peek behind TensorFlow Python API

Accelerator integrator perspective

Mateusz Nowak

30.05.2022

codeeurope

Who am I

AI Software Engineering Manager

Yes, I still write code ;)

Located in Gdańsk, Poland

habana®
An Intel Company

All views expressed herein are those of my own and do not represent the opinions of any entity whatsoever with which I have been, am now, or will be affiliated.

Table of contents

TensorFlow overview

TensorFlow framework

TensorFlow framework

TensorFlow

Open-source deep learning framework

TensorFlow framework

Open-source deep learning framework

Python based API for models

TensorFlow framework

TensorFlow

- Open-source deep learning framework
- Python based API for models
- Supports multiple devices like:
  - CPU
  - GPU
  - Habana Gaudi (a.k.a. HPU, using Habana TF plugin)

TensorFlow framework

TensorFlow

- Open-source deep learning framework
- Python based API for models
- Supports multiple devices like:
  - CPU
  - GPU
  - Habana Gaudi (a.k.a. HPU, using Habana TF plugin)
- Supports distributed trainings

TensorFlow framework

Open-source deep learning framework

Python based API for models

Supports multiple devices like:

- CPU
- GPU
- Habana Gaudi (a.k.a. HPU, using Habana TF plugin)

Supports distributed trainings

Implemented in C++

TensorFlow development resources

`https://github.com/tensorflow/tensorflow`

`https://www.tensorflow.org/api_docs/`

`https://www.tensorflow.org/guide`

Sample model

```python
num_classes = 5

model = Sequential([
  layers.experimental.preprocessing.Rescaling(1./255, input_shape=(img_height, img_width, 3)),
  layers.Conv2D(16, 3, padding='same', activation='relu')
])

model.compile(optimizer='adam',
              loss=tf.keras.losses.SparseCategoricalCrossentropy(from_logits=True),
              metrics=['accuracy'])
```

Modes of execution

Modes of execution

Pure eager

tf.function graph eager

Legacy graph mode (TF1.x)

Eager tf.function graph mode

Eager tf.function graph mode

Achieved by marking function with tf.function annotation

Eager tf.function graph mode

Achieved by marking function with tf.function annotation

Executes function at once as graph

Eager tf.function graph mode

Achieved by marking function with tf.function annotation

Executes function at once as graph

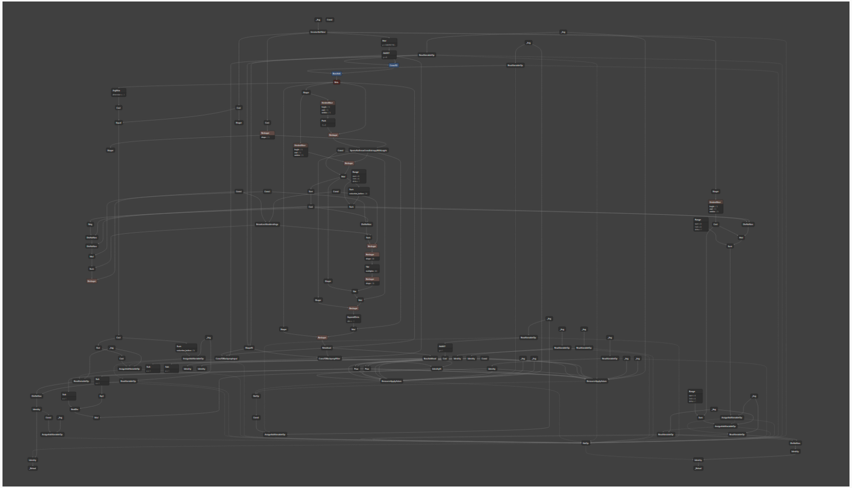
Before execution optimization passes are run on the graph

Pure eager mode

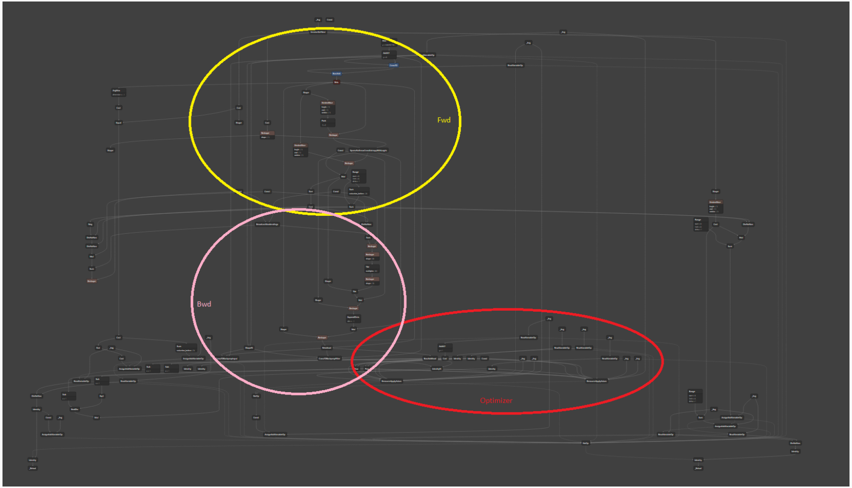
- Operators are executed one by one
- Up to TF2.9: No optimization passes performed
- Post TF2.9 or TF2.9 with TF_RUN_EAGER_OP_AS_FUNCTION flag set
  - Each operator is encapsulated in tf.function
  - All optimization passess are being run on such operator
  - TensorFlow developers effort to unify execution

TensorFlow model graph

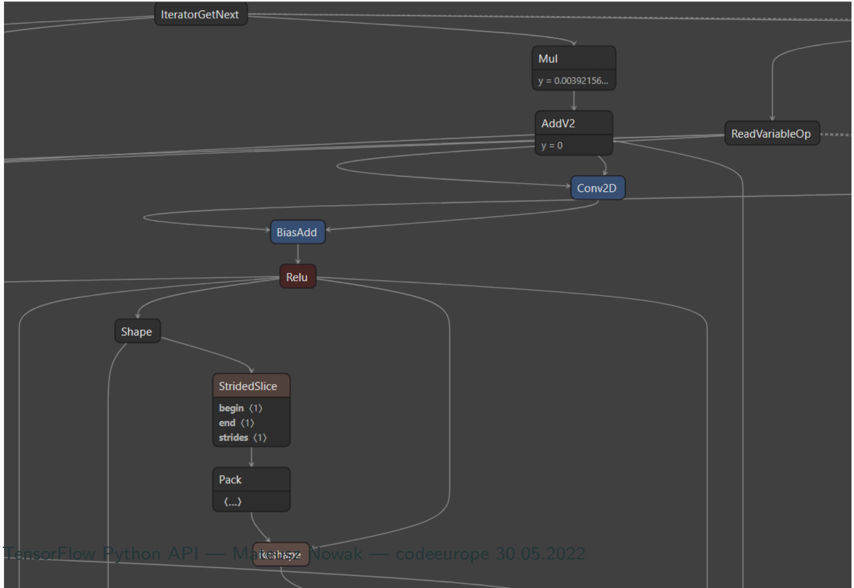
Sample model graph



Sample model graph - phases



Sample model graph - zoom in



TensorFlow operators

TensorFlow operators

Each operator represents arithmetical, logical or control operation (or other)

TensorFlow operators

Each operator represents arithmetical, logical or control operation (or other)

Each operator can have multiple variants based on attributes

TensorFlow operators

Each operator represents arithmetical, logical or control operation (or other)

Each operator can have multiple variants based on attributes

- E.g., for different data types, data layouts, modes, etc.

TensorFlow operators

Each operator represents arithmetical, logical or control operation (or other)

Each operator can have multiple variants based on attributes

- E.g., for different data types, data layouts, modes, etc.

Each operator can be implemented with one or more kernels

TensorFlow operators

Each operator represents arithmetical, logical or control operation (or other)

Each operator can have multiple variants based on attributes

- E.g., for different data types, data layouts, modes, etc.

Each operator can be implemented with one or more kernels

Each kernel can cover one or more variant

TensorFlow operators

- Each operator represents arithmetical, logical or control operation (or other)
- Each operator can have multiple variants based on attributes
  - E.g., for different data types, data layouts, modes, etc.
- Each operator can be implemented with one or more kernels
- Each kernel can cover one or more variant
- Each device supply its own kernels

TensorFlow operators

- Each operator represents arithmetical, logical or control operation (or other)
- Each operator can have multiple variants based on attributes
  - E.g., for different data types, data layouts, modes, etc.
- Each operator can be implemented with one or more kernels
- Each kernel can cover one or more variant
- Each device supply its own kernels
  - Exception: DEFAULT_DEVICE kernels

Registering TensorFlow operator

Registering TensorFlow operator

Using REGISTER_OP macro

Registering TensorFlow operator

Using REGISTER_OP macro

With builder pattern, list operator name, attributes, inputs, outputs, and shape inference function

Registering TensorFlow operator

Using REGISTER_OP macro

With builder pattern, list operator name, attributes, inputs, outputs, and shape inference function

Translates to static variable, initialized at process startup, which builds OpDef, and registers it in runtime registry

Registering TensorFlow operator

Using REGISTER_OP macro

With builder pattern, list operator name, attributes, inputs, outputs, and shape inference function

Translates to static variable, initialized at process startup, which builds OpDef, and registers it in runtime registry

This opens up possibility of altering operator registrations in runtime

Registering TensorFlow operator

Using REGISTER_OP macro

With builder pattern, list operator name, attributes, inputs, outputs, and shape inference function

Translates to static variable, initialized at process startup, which builds OpDef, and registers it in runtime registry

This opens up possibility of altering operator registrations in runtime

Note: This is low level operator, high level operators calling low level ones can be separately defined on python level

Operator registration example

```
REGISTER_OP("MaxPoolGradV2")
    .Attr(GetPaddingAttrString())
    .Attr(GetConvnetDataFormatAttrString())
    .Input("orig_input: T")
    .Input("orig_output: T")
    .Input("grad: T")
    .Input("ksize: int32")
    .Input("strides: int32")
    .Output("output: T")
    .Attr("T: realnumbertype = DT_FLOAT")
    .SetShapeFn(shape_inference::MaxPoolGradShape);
```

Registering kernels

Registering kernels

Using REGISTER_KERNEL_BUILDER macro

Registering kernels

Using REGISTER_KERNEL_BUILDER macro

With builder pattern list operator name, device the implementation is targeting, constraints, and implementation class

Registering kernels

Using REGISTER_KERNEL_BUILDER macro

With builder pattern list operator name, device the implementation is targeting, constraints, and implementation class

Translates to static variable, initialized at process startup, which builds KernelDef, and registers it in runtime registry

Registering kernels

Using REGISTER_KERNEL_BUILDER macro

With builder pattern list operator name, device the implementation is targeting, constraints, and implementation class

Translates to static variable, initialized at process startup, which builds KernelDef, and registers it in runtime registry

This opens up possibility of altering kernel registrations in runtime

Kernel registration example

```
template <class Device, class T>
class MaxPoolingGradOp : public OpKernel

REGISTER_KERNEL_BUILDER(Name("MaxPoolGradV2")
                            .Device(DEVICE_CPU)
                            .HostMemory("ksize")
                            .HostMemory("strides")
                            .TypeConstraint<float>("T"),
                        MaxPoolingGradOp<DCPU, float>);
```

Tensors and Variables

Tensor

Tensor

Represents a multidimensional array of elements

Tensor

Represents a multidimensional array of elements

Data structure consisting of:

Tensor

Represents a multidimensional array of elements

Data structure consisting of:

- Metadata (data type, shape), stored on host

Tensor

Represents a multidimensional array of elements

Data structure consisting of:

- Metadata (data type, shape), stored on host
- Reference counted data buffer, stored on host or device

Tensor

Represents a multidimensional array of elements

Data structure consisting of:

- Metadata (data type, shape), stored on host
- Reference counted data buffer, stored on host or device
- Reference count

Tensor

Represents a multidimensional array of elements

Data structure consisting of:

- Metadata (data type, shape), stored on host
- Reference counted data buffer, stored on host or device
- Reference count

Underlaying data buffer can be shared between tensors

Tensor

- Represents a multidimensional array of elements
- Data structure consisting of:
  - Metadata (data type, shape), stored on host
  - Reference counted data buffer, stored on host or device
  - Reference count
- Underlaying data buffer can be shared between tensors
  - Different tensors can use different slices of buffer

Variables

In python created with tf.Variable

Variables

In python created with tf.Variable

Represent shared, persistent state

Variables

In python created with tf.Variable

Represent shared, persistent state

Two variables will not share the same memory

Variables

In python created with tf.Variable

Represent shared, persistent state

Two variables will not share the same memory

Preserved when saving and loading models

Variables

In python created with tf.Variable

Represent shared, persistent state

Two variables will not share the same memory

Preserved when saving and loading models

Trainable model weights are stored as variables

Variables

In python created with tf.Variable

Represent shared, persistent state

Two variables will not share the same memory

Preserved when saving and loading models

Trainable model weights are stored as variables

Co-location rules apply

Variables

In python created with tf.Variable

Represent shared, persistent state

Two variables will not share the same memory

Preserved when saving and loading models

Trainable model weights are stored as variables

Co-location rules apply

Resource variables

Resource variables

TF2 implementation of Variables

Resource variables

TF2 implementation of Variables

Has strict guarantees on R/W order and value changes visibility

Resource variables

TF2 implementation of Variables

Has strict guarantees on R/W order and value changes visibility

In practice, following is happening:

Resource variables

TF2 implementation of Variables

Has strict guarantees on R/W order and value changes visibility

In practice, following is happening:

- More control edges are added to graph to ensure ordering

Resource variables

TF2 implementation of Variables

Has strict guarantees on R/W order and value changes visibility

In practice, following is happening:

- More control edges are added to graph to ensure ordering
- Additional tensor copies are being done, to ensure changes won't leak

Resource variables

TF2 implementation of Variables

Has strict guarantees on R/W order and value changes visibility

In practice, following is happening:

- More control edges are added to graph to ensure ordering
- Additional tensor copies are being done, to ensure changes won't leak
- Resource Manager locks concurrent accesses to variable tensor

Optimization passes

Optimization passes

Optimization passes

Transformations of graph

Optimization passes

Transformations of graph

Various targets

Optimization passes

- Transformations of graph
- Various targets
  - Pattern matching and fusing operators

Optimization passes

- Transformations of graph
- Various targets
  - Pattern matching and fusing operators
  - Optimizing graph - removing dead nodes, reducing control flows, etc.

Optimization passes

- Transformations of graph
- Various targets
  - Pattern matching and fusing operators
  - Optimizing graph - removing dead nodes, reducing control flows, etc.
  - Reassigning betweeen devices

Optimization passes

- Transformations of graph
- Various targets
  - Pattern matching and fusing operators
  - Optimizing graph - removing dead nodes, reducing control flows, etc.
  - Reassigning betweeen devices
  - Device specific optimizations, e.g. MKL

Optimization pass phases

Optimization pass phases

Grappler

Optimization pass phases

Grappler

PRE_PLACEMENT - after cost model assignment, before placement

Optimization pass phases

Grappler

PRE_PLACEMENT - after cost model assignment, before placement

POST_PLACEMENT - after placement

Optimization pass phases

Grappler

PRE_PLACEMENT - after cost model assignment, before placement

POST_PLACEMENT - after placement

POST_REWRITE_FOR_EXEC - after re-write using feed/fetch endpoints

Optimization pass phases

Grappler

PRE_PLACEMENT - after cost model assignment, before placement

POST_PLACEMENT - after placement

POST_REWRITE_FOR_EXEC - after re-write using feed/fetch endpoints

POST_PARTITIONING - after partitioning

Optimization pass phases

Grappler

PRE_PLACEMENT - after cost model assignment, before placement

POST_PLACEMENT - after placement

POST_REWRITE_FOR_EXEC - after re-write using feed/fetch endpoints

POST_PARTITIONING - after partitioning

Each pass has priority in group

Registering custom optimization pass

Registering custom optimization pass

Runtime registry of optimization passess

Registering custom optimization pass

Runtime registry of optimization passess

```
class LowerFunctionalOpsPass : public GraphOptimizationPass
```

Registering custom optimization pass

Runtime registry of optimization passess

class LowerFunctionalOpsPass : **public** GraphOptimizationPass

Status Run(**const** GraphOptimizationPassOptions& options) override

Registering custom optimization pass

Runtime registry of optimization passess

class LowerFunctionalOpsPass : **public** GraphOptimizationPass

Status Run(**const** GraphOptimizationPassOptions& options) override

REGISTER_OPTIMIZATION(OptimizationPassRegistry::PRE_PLACEMENT, 10, LowerFunctionalOpsPass)

TensorFlow devices

Device abstraction

Device abstraction

Has unique name

Device abstraction

Has unique name

Has context

Device abstraction

Has unique name

Has context

Serves related host and device memory allocators

Device abstraction

Has unique name

Has context

Serves related host and device memory allocators

Manager tensor creation and moving data between host and inside the device

Device abstraction

Has unique name

Has context

Serves related host and device memory allocators

Manager tensor creation and moving data between host and inside the device

Executes kernel and ensures execution queues synchronization

Device types

Device types

ThreadPoolDevice

Device types

ThreadPoolDevice

GpuDevice

Device types

ThreadPoolDevice

GpuDevice

PluggableDevice

Registering new device

Registering new device

Implement device using one of 3 base interfaces

Registering new device

Implement device using one of 3 base interfaces

Implement DeviceFactory for device discovery and instantiation

Registering new device

Implement device using one of 3 base interfaces

Implement DeviceFactory for device discovery and instantiation

Register factory in runtime registry

Registering new device

Implement device using one of 3 base interfaces

Implement DeviceFactory for device discovery and instantiation

Register factory in runtime registry

REGISTER_LOCAL_DEVICE_FACTORY("CPU", ThreadPoolDeviceFactory, 60)

TensorFlow allocators

Types of memory

Types of memory

Device memory

Types of memory

Device memory

Host memory

Types of memory

Device memory

Host memory

Caveat: CPU has both device and host memory

Selecting memory type

Selecting memory type

Using AllocatorAttributes structure on tensor allocation

Selecting memory type

Using AllocatorAttributes structure on tensor allocation

Set set_on_host(true) to select host memory

Selecting memory type

Using AllocatorAttributes structure on tensor allocation

Set set_on_host(true) to select host memory

Set set_gpu_compatible(true) to make host memory be pinned, for fast GPU DMAs

Allocator interface

Allocator interface

High level allocator

Allocator interface

High level allocator

Important methods:

```
void* AllocateRaw( size_t alignment, size_t num_bytes)
void* AllocateRaw( size_t alignment, size_t num_bytes, const
 AllocationAttributes & allocation_attr )
void DeallocateRaw(void* ptr)
```

BFC Allocator

BFC Allocator

Implementing best-fit with coalescing algorithm

BFC Allocator

Implementing best-fit with coalescing algorithm

Assumes that whole memory is owned and in a pool

BFC Allocator

Implementing best-fit with coalescing algorithm

Assumes that whole memory is owned and in a pool

Splits available memory into buckets of various sizes (dynamically split & merged to prevent fragmentation)

BFC Allocator

Implementing best-fit with coalescing algorithm

Assumes that whole memory is owned and in a pool

Splits available memory into buckets of various sizes (dynamically split & merged to prevent fragmentation)

Sensitive to small allocations - smallest bucket is quite large

SubAllocator interface

SubAllocator interface

Low level allocator

SubAllocator interface

Low level allocator

De-facto allocates buffers, e.g., calls malloc & free

Adding device allocator

Adding device allocator

By simply implementing dedicated Allocator and SubAllocator

Adding device allocator

By simply implementing dedicated Allocator and SubAllocator

Simply served by GetAllocator method of device

Adding custom CPU allocator

Adding custom CPU allocator

Implementing dedicated Allocator and SubAllocator

Adding custom CPU allocator

Implementing dedicated Allocator and SubAllocator

Implementing AllocatorFactory

Adding custom CPU allocator

Implementing dedicated Allocator and SubAllocator

Implementing AllocatorFactory

Registering to runtime registry

REGISTER_MEM_ALLOCATOR("MklCPUAllocator", 200, MklCPUAllocatorFactory)

Adding custom CPU allocator

Implementing dedicated Allocator and SubAllocator

Implementing AllocatorFactory

Registering to runtime registry

REGISTER_MEM_ALLOCATOR("MklCPUAllocator", 200, MklCPUAllocatorFactory)

Highest priority allocator is being used

Adding custom CPU allocator

Implementing dedicated Allocator and SubAllocator

Implementing AllocatorFactory

Registering to runtime registry

REGISTER_MEM_ALLOCATOR("MklCPUAllocator", 200, MklCPUAllocatorFactory)

Highest priority allocator is being used

Caveat: Custom CPU allocator has to be registered before any call to TF APIs that may allocate

Adding custom CPU allocator

Implementing dedicated Allocator and SubAllocator

Implementing AllocatorFactory

Registering to runtime registry

REGISTER_MEM_ALLOCATOR("MklCPUAllocator", 200, MklCPUAllocatorFactory)

Highest priority allocator is being used

Caveat: Custom CPU allocator has to be registered before any call to TF APIs that may allocate

Need to be aware of static initialization fiasco to correctly handle this

Caveat: Unexpected alloaction sources

Caveat: Unexpected alloaction sources

Usually memory buffers are from TF allocators defined for device

Caveat: Unexpected alloaction sources

Usually memory buffers are from TF allocators defined for device

But! user can provide manually allocated buffers

Caveat: Unexpected alloaction sources

Usually memory buffers are from TF allocators defined for device

But! user can provide manually allocated buffers

Example: Anytime user provides tensor created in numpy

Caveat: Unexpected alloaction sources

Usually memory buffers are from TF allocators defined for device

But! user can provide manually allocated buffers

Example: Anytime user provides tensor created in numpy

Cannot assume, that all buffers will come from given host allocator

Thanks for your attention!

Mateusz Nowak

https://www.linkedin.com/in/mateusz-nowak-qq/

https://github.com/noqqaqq

@noqqaqq

Feedback is appreciated ☺