TensorFlow 2.0 规划信息汇总

颜发才

facai.yan@gmail.com
facaiy.com

计算平台事业部

2018 年 10 月 25 日

图: 我们不生产水，我们只做大自然的搬运工[1]

[1]图片来源：快资讯

目录

易用性

- 用户层面
- tf.Variable
- tf.function
- tf.print

用户代码示例

```python
data = np.random.random((1000, 32))
labels = np.random.random((1000, 10))

# Instantiates a toy dataset instance:
dataset = tf.data.Dataset.from_tensor_slices((data, labels))
dataset = dataset.batch(32)
dataset = dataset.repeat()

# Create a trivial model
model = keras.Sequential([
    keras.layers.Dense(10, input_shape=(32,)),
    keras.layers.Dense(10, activation='softmax')
])
model.compile(optimizer='rmsprop',
              loss='categorical_crossentropy',
              metrics=['accuracy'])

# Don't forget to specify `steps_per_epoch` when calling `fit` on a dataset.
model.fit(dataset, epochs=10, steps_per_epoch=30)

# Save entire model to a HDF5 file
model.save('my_model.h5')

# Recreate the exact same model, including weights and optimizer.
model = keras.models.load_model('my_model.h5')
```

用户层面

- eager execution
- tf.data (tensorflow/datasets)
- tf.keras (tensorflow/models)
- estimator(model_fn + Head) and feature column (tensorflow/estimator)
- 多语言化
  - tensorflow/docs
  - core/api

RefVariable 的读写顺序问题

```python
a = tf.Variable(1.0, use_resource=True)
a.initializer.run()

assign = a.assign(2.0)

with tf.control_dependencies([assign]):
  b = a.read_value()

with tf.control_dependencies([b]):
  other_assign = a.assign(3.0)

with tf.control_dependencies([other_assign]):
  # Will print 2.0 because the value was read before other_assign ran. If
  # `a` was a tf.Variable instead, 2.0 or 3.0 could be printed.
  tf.Print(b, [b]).eval()
```

tf.Variable

The API for Variables will then change in the following ways for TF 2.0:[2]

- RefVariable → ResourceVariable
- clean global scopes, and collections
  - remove variable_scope → name_scope
    graph.variable_scope_stack → module-global weak dict
- tf.assign* will be removed
- get_variable → tf.Variable + scoped factory functions

[2]RFC: Variables in TensorFlow 2.0

可能的常见用法

```
# 1. don't care
a = tf.Variable(**kwargs)

# 2.1 official subclass
b_1 = ResourceVariable(**kwargs)

def custom_creator(next_creator, **kwargs):
  return ResourceVariable(**kwargs)

with tf.variable_creator_scope(custom_creator):
  b_2 = tf.Variable(**kwargs)
assert b_1.eval() == b_2.eval()

# 2.2 chain
def custom_creator(next_creator, **kwargs):
  vars = [next_creator(**your_kwargs) for _ in range(3)]
  return PartitialedVariable(variable_list=vars, **kwargs)

# 3. custom subclass
class MyVariable(Variable):
  pass
```

```python
def my_creator(next, **kwargs):
  return next(**kwargs)

def other_creator(next, **kwargs):
  return next(**kwargs)

def default_creator(next, **kwargs):
  if v1:
    return RefVariable(**kwargs)
  else:
    return ResourceVariable(**kwargs)

creator_stack =[my_creator,
                other_creator,
                # ......
                # ......
                default_creator]

# equal to
def my_getter(**kwargs):
  return my_creator(
      other_creator(
          default_creator(None, **kwargs),
          **kwargs),
      **kwargs)

my_variable = my_getter(**kwargs)
```

tf.Variable and scoped factory function

```
class Graph(object):
  @tf_contextlib.contextmanager
  def _variable_creator_scope(self, creator):
    old = list(self._variable_creator_stack)
    self._thread_local._variable_creator_stack.append(creator)
    try:
      yield
    finally:
      self._thread_local._variable_creator_stack = old

def _make_getter(captured_creator, previous_getter):
  return lambda **kwargs: captured_creator(previous_getter, **kwargs)

# 封装到 variable_scope.variable_creator_scope:
with (ops.get_default_graph()
          ._variable_creator_scope(custom_creator)):
  # 封装进 tf.Variable:
  previous_getter = lambda **kwargs: default_variable_creator(None, **kwargs)
  for creator in ops.get_default_graph()._variable_creator_stack:
    previous_getter = _make_getter(creator, previous_getter),
  return previous_getter(**kwargs)

# tf 2.0:
with tf.variable_creator_scope(custom_creator):
  a = tf.Variable(**kwargs)
```

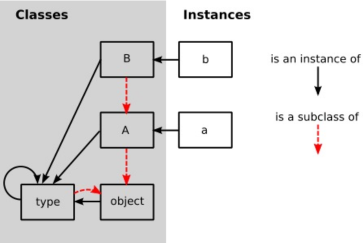
```python
def creator_func(next_creator, **kwargs):
  pass

def getter_func(**kwargs):
  pass

my_getter = (
    lambda **k2:
        my_creator(
            (lambda **k1:
                other_creator(
                    (lambda **k0: default_creator(None, **k0)),
            **k1)),
        **k2)),

my_variable = my_getter(**kwargs)
```

$$
\begin{aligned}
\text{funcs} &= [f_0(g, x), f_1(g, x), \cdots, f_n(g, x)] \\
g_0(x) &= f_0(\_, x) \\
g_n(x) &= f_n(g_{n-1}, x) \qquad \text{For } n = 1, 2, \cdots, n
\end{aligned}
$$



图：Python 对象关系[3]

```python
class type(object):
  def __call__(cls, *args, **kwargs):
    obj = cls.__new__()
    obj.__init__(*args, **kwargs)
    return obj

class A(object, metaclass=type):
  pass

A = type('A', (), {})

class B(A):
  pass

B = type('B', (A,), {})

a = A()
a = super(A, cls).__call__()
```

[3]用 Python 实现一个最简单的对象模型

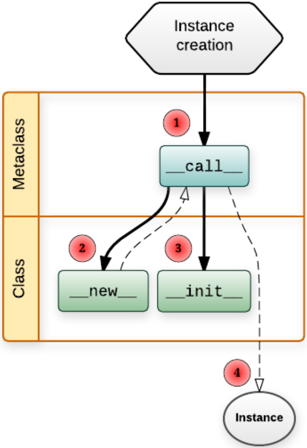


图: The diagram of how instances are constructed.[4]

[4] Understanding Python metaclasses

code snippet of tf 2.0 Variable

```python
class VariableMetaclass(type):

  def _variable_v1_call(cls, **kwargs):
    pass

  def _variable_v2_call(cls, **kwargs):
    pass

  def __call__(cls, *args, **kwargs):
    if cls is VariableV1:
      return cls._variable_v1_call(*args, **kwargs)
    elif cls is Variable:
      return cls._variable_v2_call(*args, **kwargs)
    else:
      return super(VariableMetaclass, cls).__call__(*args, **kwargs)

@tf_export("Variable", v1=[])
class Variable(six.with_metaclass(VariableMetaclass,
                                  checkpointable.CheckpointableBase)):
  def __init__(self, **kwargs):
    raise NotImplementedError
```

source: tensorflow/python/ops/variables.py
commit: 4a5693e732b80a593bca7bf94ddd5df9e5d78cc0

tf.function 示例

```python
import tensorflow as tf

@tf.function
def compute_z0(x, y):
  return tf.add(x, y)

@tf.function
def compute_z1(x):
  return compute_z0(x, tf.square(x))

z0 = compute_z0(2., 3.)
# 5.
z1 = compute_z1(2.)
# 6.
```

tf.function

make TensorFlow be more "Pythonic" in 2.0.[5]

- graph + session → function
- 状态一致: python object 与 tf runtime
- easy to export: GraphDef + Checkpoint and / or SaveModel
- enable eager execution by default
- 兼容 1.x 代码: tf.compat.v1.wrap_function

主要问题：现有图优化技术可能受影响？

[5] TensorFlow 2.0: Functions, not Sessions

For W, b, and c, the lifetime of the Python objects and the runtime state are tied together.

```python
W = tf.Variable(
    tf.glorot_uniform_initializer()((10, 10)))
b = tf.Variable(tf.zeros(10))
c = tf.Variable(0)

@tf.function
def f(x):
  c.assign_add(1)
  return tf.matmul(x, W) + b

print(f(make_input_value())
assert int(c) == 1
```

- state are only created the first time the function f is called.
- variable referenced by the function still exists when called.

Automatically insert control dependencies to ensure stateful operations follow graph construction order.[6]

```
a = tf.Variable(1.0)
b = tf.Variable(1.0)

@tf.function
def f():
  a.assign(2.0)
  b.assign(3.0)
  return a + b

print(f())
```

Note: avoid only observable differences from program order.

[6]AutomaticControlDependencies

Trace Caches

Every time functioin is invoked in the Python program, a trace_cache_key is computed.[7]

```
@tf.function
def f(x):
  return tf.square(x)

f(tf.constant(1, dtype=tf.int32))
f(tf.constant(1.0, dtype=tf.float32))
f(2.0)  # use tf.constant instead.
f(3.0)

# 1. Input Signatures:
@tf.function(input_signature=((tf.float32, [None]))
def f(x):
  return tf.add(x, 1.)
# 2. GC + weak reference.
# 3. warning if ratio of calls is too greater.
```

[7] PolymorphicFunction._maybe_define_function

潜在的用法

member function of a class

```python
class ScalarModel(object):

  def __init__(self):
    self.v = tf.Variable(0)

  @tf.function
  def increment(self, amount):
    self.v.assign_add(amount)
```

示例一：[8]

```python
class Dense(Layer):
  """Just your regular densely-connected NN layer."""

  def build(self, input_shape):
    self.kernel = self.add_weight(
        'kernel',
        shape=[input_shape[-1].value, self.units],
        initializer=self.kernel_initializer,
        regularizer=self.kernel_regularizer,
        constraint=self.kernel_constraint,
        dtype=self.dtype,
        trainable=True)
    self.built = True

  def call(self, inputs):
    outputs = gen_math_ops.mat_mul(inputs, self.kernel)
    if self.use_bias:
      outputs = nn.bias_add(outputs, self.bias)
    if self.activation is not None:
      return self.activation(outputs)
    return outputs
```

[8] tensorflow/python/keras/layers/core.py

示例二：[9]

```python
class Model(Network):
  """Model groups layers into an object with training and inference features."""

  def _make_train_function(self):
    # ... ...
    self.train_function = K.function(
        inputs, [self.total_loss] + self.metrics_tensors,
        updates=updates,
        name='train_function',
        **self._function_kwargs)

  def _make_test_function(self):
     # ... ...
     self.test_function = K.function(
         inputs, [self.total_loss] + self.metrics_tensors,
         updates=self.state_updates + self.metrics_updates,
         name='test_function',
         **self._function_kwargs)

  def _make_predict_function(self):
    # ... ...
    pass
```

[9] tensorflow/python/keras/engine/training.py

示例三：[10]

```python
class Estimator(object):
  """Estimator class to train and evaluate TensorFlow models."""

  def _train_model_default(self, input_fn, hooks, saving_listeners):
    pass

  def _train_model_distributed(self, input_fn, hooks, saving_listeners)
    pass

  def _call_model_fn_eval(self, input_fn, config):
    pass

  def _call_model_fn_eval_distributed(self, input_fn, config):
    pass

  def predict(self, **kwargs):
    pass

  def _add_meta_graph_for_mode(self, **kwargs):
    pass
```

[10] tensorflow_estimator/python/estimator/estimator.py

tf.print

similar to the standard python print API.[11]

- tf.Print → tf.print, tf.strings.format
  - For python 2: from __future__ import print_function[12],[13]
- identity op → control dependencies
- controllable logging levels
  - stdout/stderr，与 notebook 不兼容
  - device: cpu:0 by default?
- supports for nested data structures

[11] RFC: New tf.print

[12] Moving to require Python 3

[13] Cheat Sheet: Writing Python 2-3 compatible code

eager mode

```
tf.enable_eager_execution()
tensor = tf.range(10)
tf.print(tensor, output_stream=sys.stderr)
# (This prints "[0 1 2 ... 7 8 9]" to sys.stderr)
```

graph mode

```
with sess.as_default():
  tensor = tf.range(10)
  print_op = tf.print(tensor, output_stream=sys.stdout)
  # For tf 1.0: return an identity op:
  # doubled_tensor = print_op * 2
  # For tf 2.0:
  with tf.control_dependencies([print_op]):
    doubled_tensor = tensor * 2
  sess.run(doubled_tensor)
  # (This prints "[0 1 2 ... 7 8 9]" to sys.stdout)
```

用户代码模块化

- collections
- Optimizer
- RNN

collections

we have situations where we might build multiple models in a graph, and functions cause further issues because functions are graphs of their own.[14]

收集汇总 用户自行收集和追踪

- queue runner → tf.data
- variable → 利用 variable creator 在创建时追踪
- update op → 在 model_fn 里更新，或者用 keras 的 model.updates

序列化 SaveModel，后续会有专门 API 支持

维持状态 SharedEmbeddingColumns，使用全局变量替代

[14]RFC: Deprecate Collections

VariableTracker example

```python
class VariableTracker(object):
  def __init__(self):
    self.variables = []

  def variable_tracker(self, next_creator, **kwargs):
    v = next_creator(**kwargs)
    self.variables.append(v)
    return v

with tf.variable_creator_scope(tracker.variable_tracker):
  # ...
  a = tf.Variable(0)
  # ...
assert tracker.variables == [a]
```

Optimizer unification

- extending the TensorFlow Optimizer API[15]
  - based on the existing tf.contrib.optimizer_v2 optimizers
  - serializable: *_config, *_weights
  - modifiable hyperparameters: optimizer.learning_rate = 0.2
  - gradient clipping: get_gradients, *_updates
- disable reusing a single optimizer instance across multiple graphs.
- use_locking argument is removed: internal implementation details.
- should not require positional arguments.

[15] RFC: Optimizer unification in TensorFlow 2.0

The set of new optimizers would be (same signatures, same objects, no wrappers):

1. SGD (both GradientDescentOptimizer and MomentumOptimizer)
2. Adadelta
3. Adagrad
4. Adam
5. FTRL (not yet in Keras)
6. RMSProp
7. Adamax (not yet in TF)
8. Nadam (not yet in TF)

Unify RNN interface

Unify the final API that is similar to existing Keras API, and port functionalities from TF RNN to Keras.[16]

- gate order: IFCO vs ICFO
- tf.contrib.rnn: 只迁移少部份 RNN Cell
- NVidia CuDNN

[16] RFC: Unify RNN interface

清理老旧设计

- namespaces
- tf.contrib

namespaces

structure namespaces in a clear way for easier discoverability and usability.[17]

- tf_export decorator
- additional namespaces
  - tf.losses $\rightarrow$ tf.keras.losses
  - tf.metrics $\rightarrow$ tf.keras.metrics
  - tf.layers $\rightarrow$ tf.keras.layers
- deprecated namespaces
  - tf.logging $\rightarrow$ Python logging module
  - tf.manip: keep them in root instead.

[17] RFC: TensorFlow API symbols and namespaces

tf.contrib

sunset the present tf.contrib, and replace its important functions with more maintainable alternatives.[18]

- moving to core: symbols should be prefixed with experimental.
- moving to a seperate repository
    - tensorflow/addons: layer, metric, loss, optimizer, op or kernel
    - tensorflow/IO
    - tensorflow/network
    - tensorflow/scientific
- deleting

[18] RFC: Sunset tf.contrib

小结

小结

易用性	eager, tf.data, tf.keras
模块化	tf.keras
一致性	统一、去重、移除

谢谢！

TensorFlow 2.0 规划信息汇总

颜发才

facai.yan@gmail.com
facaiy.com

计算平台事业部

2018 年 10 月 25 日