# coding=utf-8

# Copyright 2018 Google LLC & Hwalsuk Lee.

#

# Licensed under the Apache License, Version 2.0 (the "License");

# you may not use this file except in compliance with the License.

# You may obtain a copy of the License at

#

# http://www.apache.org/licenses/LICENSE-2.0

#

# Unless required by applicable law or agreed to in writing, software

# distributed under the License is distributed on an "AS IS" BASIS,

# WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.

# See the License for the specific language governing permissions and

# limitations under the License.

"""Tests for deterministic TensorFlow operations."""

from \_\_future\_\_ import absolute\_import

from \_\_future\_\_ import division

from \_\_future\_\_ import print\_function

import os

from absl import flags

from absl import logging

from absl.testing import parameterized

from compare\_gan.tpu import tpu\_random

import numpy as np

from six.moves import range

import tensorflow as tf

FLAGS = flags.FLAGS

class TpuRandomTest(parameterized.TestCase, tf.test.TestCase):

def \_run\_graph\_op\_in\_estimator(self, create\_op\_fn, model\_dir, use\_tpu,

training\_steps=4):

"""Helper function to test an operation within a Estimator.

Args:

create\_op\_fn: Function that will be called from within the model\_fn.

The returned op will be run as part of the training step.

model\_dir: Directory for saving checkpoints.

use\_tpu: Whether to use TPU.

training\_steps: Number of trainings steps.

"""

def input\_fn(params):

features = {"x": np.ones((8, 3), dtype=np.float32)}

labels = np.ones((8, 1), dtype=np.float32)

dataset = tf.data.Dataset.from\_tensor\_slices((features, labels))

# Add a feature for the random offset of operations in tpu\_random.py.

dataset = tpu\_random.add\_random\_offset\_to\_features(dataset)

return dataset.repeat().batch(params["batch\_size"], drop\_remainder=True)

def model\_fn(features, labels, mode, params):

# Set the random offset tensor for operations in tpu\_random.py.

tpu\_random.set\_random\_offset\_from\_features(features)

test\_op = create\_op\_fn()

predictions = tf.layers.dense(features["x"], 1)

loss = tf.losses.mean\_squared\_error(labels, predictions)

optimizer = tf.train.GradientDescentOptimizer(0.01)

if params["use\_tpu"]:

optimizer = tf.contrib.tpu.CrossShardOptimizer(optimizer)

with tf.control\_dependencies([test\_op]):

train\_op = optimizer.minimize(

loss, global\_step=tf.train.get\_or\_create\_global\_step())

return tf.contrib.tpu.TPUEstimatorSpec(

mode=mode,

loss=loss,

train\_op=train\_op)

if tf.gfile.Exists(model\_dir):

tf.gfile.DeleteRecursively(model\_dir)

run\_config = tf.contrib.tpu.RunConfig(

model\_dir=model\_dir,

save\_checkpoints\_steps=1,

tpu\_config=tf.contrib.tpu.TPUConfig(iterations\_per\_loop=1))

estimator = tf.contrib.tpu.TPUEstimator(

config=run\_config,

use\_tpu=use\_tpu,

model\_fn=model\_fn,

train\_batch\_size=2)

estimator.train(input\_fn, steps=training\_steps)

@parameterized.parameters(

{"use\_tpu": False},

{"use\_tpu": True},

)

def testIsDeterministic(self, use\_tpu):

def create\_op\_fn():

z = tf.get\_variable("z", (3,), tf.float32)

random\_z = tpu\_random.uniform((3,), name="random\_z")

if use\_tpu:

random\_z = tf.contrib.tpu.cross\_replica\_sum(random\_z)

return tf.assign(z, random\_z).op

model\_dir\_1 = os.path.join(FLAGS.test\_tmpdir, "1")

self.\_run\_graph\_op\_in\_estimator(create\_op\_fn, model\_dir\_1, use\_tpu=use\_tpu)

model\_dir\_2 = os.path.join(FLAGS.test\_tmpdir, "2")

self.\_run\_graph\_op\_in\_estimator(create\_op\_fn, model\_dir\_2, use\_tpu=use\_tpu)

for step in range(1, 5):

self.assertTrue(tf.gfile.Exists(

os.path.join(model\_dir\_1, "model.ckpt-{}.index".format(step))))

ckpt\_1 = tf.train.load\_checkpoint(

os.path.join(model\_dir\_1, "model.ckpt-{}".format(step)))

ckpt\_2 = tf.train.load\_checkpoint(

os.path.join(model\_dir\_2, "model.ckpt-{}".format(step)))

z\_1 = ckpt\_1.get\_tensor("z")

z\_2 = ckpt\_2.get\_tensor("z")

logging.info("step=%d, z\_1=%s, z\_2=%s", step, z\_1, z\_2)

# Both runs are the same.

self.assertAllClose(z\_1, z\_2)

@parameterized.parameters(

{"use\_tpu": False},

{"use\_tpu": True},

)

def testIsDifferentAcrossSteps(self, use\_tpu):

def create\_op\_fn():

z = tf.get\_variable("z", (3,), tf.float32)

random\_z = tpu\_random.uniform((3,), name="random\_z")

if use\_tpu:

random\_z = tf.contrib.tpu.cross\_replica\_sum(random\_z)

return tf.assign(z, random\_z).op

model\_dir = os.path.join(FLAGS.test\_tmpdir, "1")

self.\_run\_graph\_op\_in\_estimator(create\_op\_fn, model\_dir, use\_tpu=use\_tpu)

previous\_z = None

for step in range(1, 5):

self.assertTrue(tf.gfile.Exists(

os.path.join(model\_dir, "model.ckpt-{}.index".format(step))))

ckpt = tf.train.load\_checkpoint(

os.path.join(model\_dir, "model.ckpt-{}".format(step)))

z = ckpt.get\_tensor("z")

logging.info("step=%d, z=%s", step, z)

# Different to previous run.

if previous\_z is not None:

self.assertNotAllClose(previous\_z, z)

previous\_z = z

def testIsDifferentAcrossCores(self):

def create\_op\_fn():

z\_sum = tf.get\_variable("z\_sum", (3,), tf.float32)

z\_first\_core = tf.get\_variable("z\_first\_core", (3,), tf.float32)

random\_z = tpu\_random.uniform((3,), name="random\_z")

random\_z\_sum = tf.contrib.tpu.cross\_replica\_sum(random\_z)

return tf.group(tf.assign(z\_sum, random\_z\_sum).op,

tf.assign(z\_first\_core, random\_z))

model\_dir = os.path.join(FLAGS.test\_tmpdir, "1")

self.\_run\_graph\_op\_in\_estimator(create\_op\_fn, model\_dir, use\_tpu=True)

for step in range(1, 5):

self.assertTrue(tf.gfile.Exists(

os.path.join(model\_dir, "model.ckpt-{}.index".format(step))))

ckpt = tf.train.load\_checkpoint(

os.path.join(model\_dir, "model.ckpt-{}".format(step)))

z\_sum = ckpt.get\_tensor("z\_sum")

z\_first\_core = ckpt.get\_tensor("z\_first\_core")

logging.info("step=%d, z\_sum=%s, z\_first\_core=%s",

step, z\_sum, z\_first\_core)

# Sum is not the first core times 2.

self.assertNotAllClose(z\_sum, 2 \* z\_first\_core)

if \_\_name\_\_ == "\_\_main\_\_":

tf.test.main()