ConcreteFunctionTPU

November 24, 2020

[1]:
```
# mre 2020-11-24

import numpy as np
import scipy as sp
import matplotlib.pyplot as plt

import os
os.environ["CUDA_VISIBLE_DEVICES"] = "-1"

import tensorflow as tf
```

1 Helper for interpreter

[2]:
```
# run interpreter on random input
def test(interpreter):
    interpreter.allocate_tensors()

    input_details  = interpreter.get_input_details()
    output_details = interpreter.get_output_details()

    input_shape = input_details[0]['shape']
    input_data  = np.array(np.random.random_sample(input_shape), dtype=np.
 ↪float32)
    interpreter.set_tensor(input_details[0]['index'], input_data)

    interpreter.invoke()

    output_data = interpreter.get_tensor(output_details[0]['index'])
    return output_data


# run interpreter on real dataset data
def run(interpreter, data):
    interpreter.allocate_tensors()

    input_index  = interpreter.get_input_details()[0]["index"]
```

```
    output_index = interpreter.get_output_details()[0]["index"]

    y = []
    for i, x in enumerate(data):
        tx = tf.constant(x, shape=(1,1))

        interpreter.set_tensor(input_index, tx)
        interpreter.invoke()
        output = interpreter.tensor(output_index)

        y.append(output()[0][0])

    return np.array(y)
```

2 Helper uint8

[3]:
```
# convert float to uint8
def toUint8(xx):
    x = np.array(xx)
    c = x.mean()
    r = x.max() - x.min()
    return np.uint8(255 * (x - x.min()) / (x.max() - x.min()))


# run uint8-Model with IO-conversion
def runUint8(interpreter, x):
    """
    x -> uint8-input -> interpreter -> uint8-output -> float
    """
    x8 = toUint8(x)

    y8 = np.array(run(interpreter, x8))

    s, c =interpreter.get_output_details()[0]['quantization']
    y = s * (y8 - c)

    return y
```

3 Helper for TPU execution

[4]:
```
import platform
import tflite_runtime.interpreter as tflite

EDGETPU_SHARED_LIB = {
  'Linux': 'libedgetpu.so.1',
```

```
  'Darwin': 'libedgetpu.1.dylib',
  'Windows': 'edgetpu.dll'
}[platform.system()]


def make_interpreter(model_file):
  model_file, *device = model_file.split('@')
  return tflite.Interpreter(
      model_path=model_file,
      experimental_delegates=[
          tflite.load_delegate(EDGETPU_SHARED_LIB,
                               {'device': device[0]} if device else {})
      ])
```

4 Definition of (concrete) function and data

[5]:
```
@tf.function
def cf(a):
  return 2*a + 3

cf(tf.ones([2, 2]))
```

[5]:
```
<tf.Tensor: shape=(2, 2), dtype=float32, numpy=
array([[5., 5.],
       [5., 5.]], dtype=float32)>
```

[6]:
```
nx = 10

#data = [np.array([x], dtype=np.float32) for x in np.arange(nx) + 1]

np.random.seed(17)
data = [np.array([10*x], dtype=np.float32) for x in np.random.randn(nx)]
data
```

[6]:
```
[array([2.7626588], dtype=float32),
 array([-18.54628], dtype=float32),
 array([6.2390113], dtype=float32),
 array([11.453113], dtype=float32),
 array([10.371904], dtype=float32),
 array([18.86639], dtype=float32),
 array([-1.1169829], dtype=float32),
 array([-3.6210134], dtype=float32),
 array([1.4867505], dtype=float32),
 array([-4.3778315], dtype=float32)]
```

5 Conversion to TFLite and execution on CPU

5.1 Convert concrete function to TFLite

[7]:
```
converter = tf.lite.TFLiteConverter.from_concrete_functions([cf.
↪get_concrete_function(tf.ones([1,1]))])

model_lite = converter.convert()
```

5.2 Execution with random input

[8]:
```
interpreter = tf.lite.Interpreter(model_content=model_lite)

test(interpreter)
```

[8]:
```
array([[4.1035028]], dtype=float32)
```

5.3 Execution on data

[9]:
```
run(interpreter, data)
```

[9]:
```
array([  8.525318 , -34.09256  ,  15.478023 ,  25.906225 ,  23.743809 ,
        40.73278  ,   0.7660341,  -4.242027 ,   5.973501 ,  -5.755663 ],
      dtype=float32)
```

6 Conversion to TFLite-uint8, execution on CPU and TPU

6.1 Representative data for quantization

[10]:
```
def representative_data_gen():
    for x in data:
        yield [tf.cast(x, tf.float32)]

list(representative_data_gen())
```

[10]:
```
[[<tf.Tensor: shape=(1,), dtype=float32, numpy=array([2.7626588],
dtype=float32)>],
 [<tf.Tensor: shape=(1,), dtype=float32, numpy=array([-18.54628],
dtype=float32)>],
 [<tf.Tensor: shape=(1,), dtype=float32, numpy=array([6.2390113],
dtype=float32)>],
 [<tf.Tensor: shape=(1,), dtype=float32, numpy=array([11.453113],
dtype=float32)>],
 [<tf.Tensor: shape=(1,), dtype=float32, numpy=array([10.371904],
dtype=float32)>],
 [<tf.Tensor: shape=(1,), dtype=float32, numpy=array([18.86639],
dtype=float32)>],
```

```
 [<tf.Tensor: shape=(1,), dtype=float32, numpy=array([-1.1169829],
dtype=float32)>],
 [<tf.Tensor: shape=(1,), dtype=float32, numpy=array([-3.6210134],
dtype=float32)>],
 [<tf.Tensor: shape=(1,), dtype=float32, numpy=array([1.4867505],
dtype=float32)>],
 [<tf.Tensor: shape=(1,), dtype=float32, numpy=array([-4.3778315],
dtype=float32)>]]
```

6.2 Convert concrete function to TFLite-uint8

[11]:
```
converter = tf.lite.TFLiteConverter.from_concrete_functions([cf.
 ↪get_concrete_function(tf.ones([1,1]))])

converter.experimental_new_converter = True
converter.target_spec.supported_ops  = [tf.lite.OpsSet.TFLITE_BUILTINS_INT8]

converter.inference_input_type  = tf.uint8
converter.inference_output_type = tf.uint8

converter.representative_dataset = representative_data_gen

model_int_lite = converter.convert()
```

6.3 Run uint8-model on CPU with uint8-IO

[12]:
```
interpreter = tf.lite.Interpreter(model_content=model_int_lite)

run(interpreter, toUint8(data))
```

[12]:
```
array([145,   0, 168, 204, 197, 254, 118, 101, 136,  96], dtype=uint8)
```

6.4 Run uint8-model on CPU with float-IO

[13]:
```
runUint8(interpreter, data)
```

[13]:
```
array([ 8.50954866, 41.08057976, 15.25850105, 25.8220787 , 23.76804972,
       40.49371433,  0.58686543, 70.71728373,  5.86865425, 69.25012016])
```

7 EdgeTPU

7.1 Write uint8-model to file and compile it for TPU

[14]:
```
with open('model_int.tflite', 'wb') as f:
    f.write(model_int_lite)
```

[15]:
```
! edgetpu_compiler model_int.tflite
```

```
Edge TPU Compiler version 15.0.340273435

Model compiled successfully in 12 ms.

Input model: model_int.tflite
Input size: 1.41KiB
Output model: model_int_edgetpu.tflite
Output size: 32.49KiB
On-chip memory used for caching model parameters: 512.00B
On-chip memory remaining for caching model parameters: 8.10MiB
Off-chip memory used for streaming uncached model parameters: 0.00B
Number of Edge TPU subgraphs: 1
Total number of operations: 4
Operation log: model_int_edgetpu.log
See the operation log file for individual operation details.
```

[16]:
```
! ls -l model_int*.tflite
```

```
-rw-rw-r-- 1 mre mre 33272 Nov 24 18:49 model_int_edgetpu.tflite
-rw-rw-r-- 1 mre mre  1440 Nov 24 18:49 model_int.tflite
```

7.2 Run uint8-model on TPU with uint8-IO

[17]:
```
interpreter = make_interpreter("model_int_edgetpu.tflite")

run(interpreter, toUint8(data))
```

[17]:
```
array([145,   0, 168, 204, 197, 254, 118, 101, 136,  96], dtype=uint8)
```

7.3 Run uint8-model on CPU with float-IO

[18]:
```
runUint8(interpreter, data)
```

[18]:
```
array([ 8.50954866, 41.08057976, 15.25850105, 25.8220787 , 23.76804972,
       40.49371433,  0.58686543, 70.71728373,  5.86865425, 69.25012016])
```