Assignment 3: TENSORFLOW – COMPUTATIONAL GRAPH
Submit file PDF: assign3_lớp.nhóm_tênho.pdf
For example: assign2_02clc.03_quetd.pdf
assign2_02cq.03_quetd.pdf

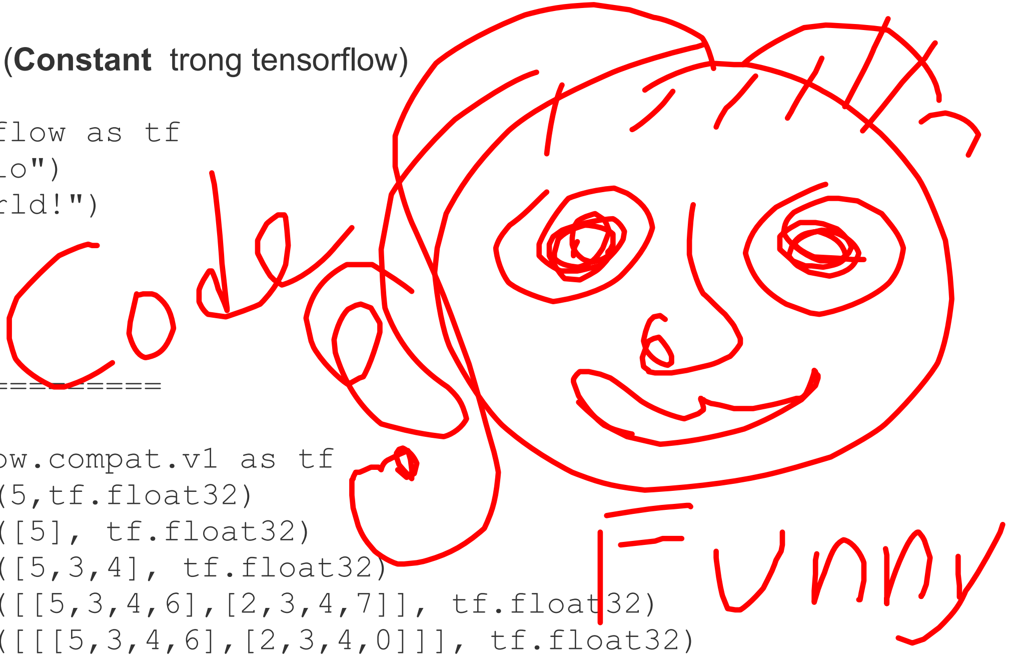
- Submit before MONDAY (08/09) 11:30PM for both CQ & CLC classes

==

When running these examples, students must deal with errors. This is a way to study!

3.1. Running and explain (**Constant** trong tensorflow)

```
3.1.1  import tensorflow as tf
h = tf.constant("Hello")
w = tf.constant(" World!")
hw = h + w
with tf.Session() as
ans = sess.run(hw)
print (ans)
     ========================
3.1.2
     import tensorflow.compat.v1 as tf
     x = tf.constant(5,tf.float32)
     y = tf.constant([5], tf.float32)
     z = tf.constant([5,3,4], tf.float32)
     t = tf.constant([[5,3,4,6],[2,3,4,7]], tf.float32)
     u = tf.constant([[[5,3,4,6],[2,3,4,0]]], tf.float32)
     v = tf.constant([[[5,3,4,6],[2,3,4,0]],
                      [[5,3,4,6],[2,3,4,0]],
                      [[5,3,4,6],[2,3,4,0]]
                     ], tf.float32)
     print(v)
```



3.2. Running and explain (Variable in tensorflow)

```
3.2.1

     import tensorflow.compat.v1 as tf
     tf.compat.v1.disable_eager_execution()

     x1 = tf.Variable(5.3, tf.float32)
     x2 = tf.Variable(4.3, tf.float32)
     x  = tf.multiply(x1,x2)

     init = tf.global_variables_initializer()
     with tf.Session() as sess:
          sess.run(init)
          t = sess.run(x)
          print(t)
     ==================================
```

```
3.2.2 import tensorflow.compat.v1 as tf

     tf.compat.v1.disable_eager_execution()

     x1 = tf.Variable([[5.3,4.5,6.0],
                       [4.3,4.3,7.0]
                      ], tf.float32)
     x2 = tf.Variable([[4.3,4.3,7.0],
                       [5.3,4.5,6.0]
                      ], tf.float32)
     x  = tf.multiply(x1,x2)

     init = tf.global_variables_initializer()
     with tf.Session() as sess:
         sess.run(init)
         t = sess.run(x)
         print(t)
```

```
3.2.3 import tensorflow.compat.v1 as tf


     # creating nodes in computation graph
     node = tf.Variable(tf.zeros([2,2]))

     # running computation graph
     with tf.Session() as sess:

         # initialize all global variables
         sess.run(tf.global_variables_initializer())

         # evaluating node
         print("Tensor value before addition:\n",sess.run(node))

         # elementwise addition to tensor
         node = node.assign(node + tf.ones([2,2]))

         # evaluate node again
         print("Tensor value after addition:\n", sess.run(node))
         sess.close()
```

3.3. Runnng and explain (Placeholder)

```
3.3.1
```

```
import tensorflow.compat.v1 as tf
tf.compat.v1.disable_eager_execution()

x = tf.placeholder(tf.float32,None)
y = tf.add(x,x)

with tf.Session() as sess:
    x_data= 5
    result = sess.run(y,feed_dict={x:x_data})
    print(result)
```

3.3.2

```
import tensorflow.compat.v1 as tf
tf.compat.v1.disable_eager_execution()

x = tf.placeholder(tf.float32,[None,3])
y = tf.add(x,x)

with tf.Session() as sess:
    x_data= [[1.5, 2.0, 3.3]]
    result = sess.run(y,feed_dict={x:x_data})
    print(result)
```

3.3.3

```
import tensorflow.compat.v1 as tf
tf.compat.v1.disable_eager_execution()

x = tf.placeholder(tf.float32,[None,None,3])
y = tf.add(x,x)

with tf.Session() as sess:
    x_data= [[[1,2,3]]]
    result = sess.run(y,feed_dict={x:x_data})
    print(result)
```

3.3.4

```
import tensorflow.compat.v1 as tf
tf.compat.v1.disable_eager_execution()

x = tf.placeholder(tf.float32,[None,4,3])
y = tf.add(x,x)

with tf.Session() as sess:
    x_data= [[[1,2,3],
              [2,3,4],
              [2,3,5],
              [0,1,2]
            ]]
    result = sess.run(y,feed_dict={x:x_data})
    print(result)
```

3.3.5

```
import tensorflow.compat.v1 as tf
tf.compat.v1.disable_eager_execution()

x = tf.placeholder(tf.float32,[2,4,3])
y = tf.add(x,x)

with tf.Session() as sess:
    x_data= [[[1,2,3],
              [2,3,4],
              [2,3,5],
              [0,1,2]
            ],
            [[1,2,3],
              [2,3,4],
              [2,3,5],
              [0,1,2]
            ]]
    result = sess.run(y,feed_dict={x:x_data})
    print(result)
```

3.3.6

```
import tensorflow.compat.v1 as tf
tf.compat.v1.disable_eager_execution()

x = tf.placeholder(tf.float32,[2,4,3])
y = tf.placeholder(tf.float32,[2,4,3])

z = tf.add(x,y)
u = tf.multiply(x,y)

with tf.Session() as sess:
    x_data= [[[1,2,3],
              [2,3,4],
              [2,3,5],
              [0,1,2]
            ],
            [[1,2,3],
              [2,3,4],
              [2,3,5],
              [0,1,2]
            ]]
    y_data= [[[1,2,3],
              [2,3,4],
         [2,3,5],
              [0,1,2]
            ],
            [[1,2,3],
              [2,3,4],
              [2,3,5],
              [0,1,2]
            ]]
    result1 = sess.run(z,feed_dict={x:x_data, y:y_data})
    result2 = sess.run(u,feed_dict={x:x_data, y:y_data})
    print("result1 =", result1)
    print("result2 =", result2)
```

3.4. Operation. Run and explain

3.4.1

```
     import tensorflow.compat.v1 as tf
     tf.compat.v1.disable_eager_execution()

     x1 = tf.constant(5.3, tf.float32)
     x2 = tf.constant(1.5, tf.float32)
     w1 = tf.Variable(0.7, tf.float32)
     w2 = tf.Variable(0.5, tf.float32)
     u = tf.multiply(x1,w1)
     v = tf.multiply(x2,w2)
     z = tf.add(u,v)
     result = tf.sigmoid(z)
     init = tf.global_variables_initializer()

     with tf.Session() as sess:
         sess.run(init)
         print(sess.run(result))
```

3.4.2

```
import numpy as np
import matplotlib.pyplot as plt

number_of_points = 500
x_point = []
y_point = []
a = 0.22
b = 0.78

for i in range(number_of_points):
    x = np.random.normal(0.0,0.5)
    y = a*x + b +np.random.normal(0.0,0.1)
    x_point.append([x])
    y_point.append([y])
```

```
plt.plot(x_point,y_point, 'o', label = 'Input Data')
plt.legend()
plt.show()
```

3.4.3

```
import tensorflow.compat.v1 as tf
tf.compat.v1.disable_eager_execution()

x1 = tf.placeholder(tf.float32,[None,3])
x2 = tf.placeholder(tf.float32,[None,3])

w1 = tf.Variable([0.5,0.4,0.7],tf.float32)
w2 = tf.Variable([0.8,0.5,0.6], tf.float32)

u1 = tf.multiply(w1,x1)
u2 = tf.multiply(w2,x2)
v = tf.add(u1,u2)
z = tf.sigmoid(v)

init = tf.global_variables_initializer()

with tf.Session() as sess:
    x1_data= [[1,2,3]]
    x2_data= [[1,2,3]]
    sess.run(init)
    result = sess.run(z,feed_dict={x1:x1_data, x2:x2_data})
    print(result)
```

3.4.4

```
import tensorflow as tf
import numpy as np

matrix1 = np.array([(2,2,2),(2,2,2),(2,2,2)],dtype = 'int32')
matrix2 = np.array([(1,1,1),(1,1,1),(1,1,1)],dtype = 'int32')

print (matrix1)
print (matrix2)

matrix1 = tf.constant(matrix1)
matrix2 = tf.constant(matrix2)
matrix_product = tf.matmul(matrix1, matrix2)
matrix_sum = tf.add(matrix1,matrix2)
matrix_3 = np.array([(2,7,2),(1,4,2),(9,0,2)],dtype = 'float32')
print (matrix_3)

matrix_det = tf.matrix_determinant(matrix_3)
with tf.Session() as sess:
   result1 = sess.run(matrix_product)
   result2 = sess.run(matrix_sum)
   result3 = sess.run(matrix_det)

print (result1)
print (result2)
print (result3)
```

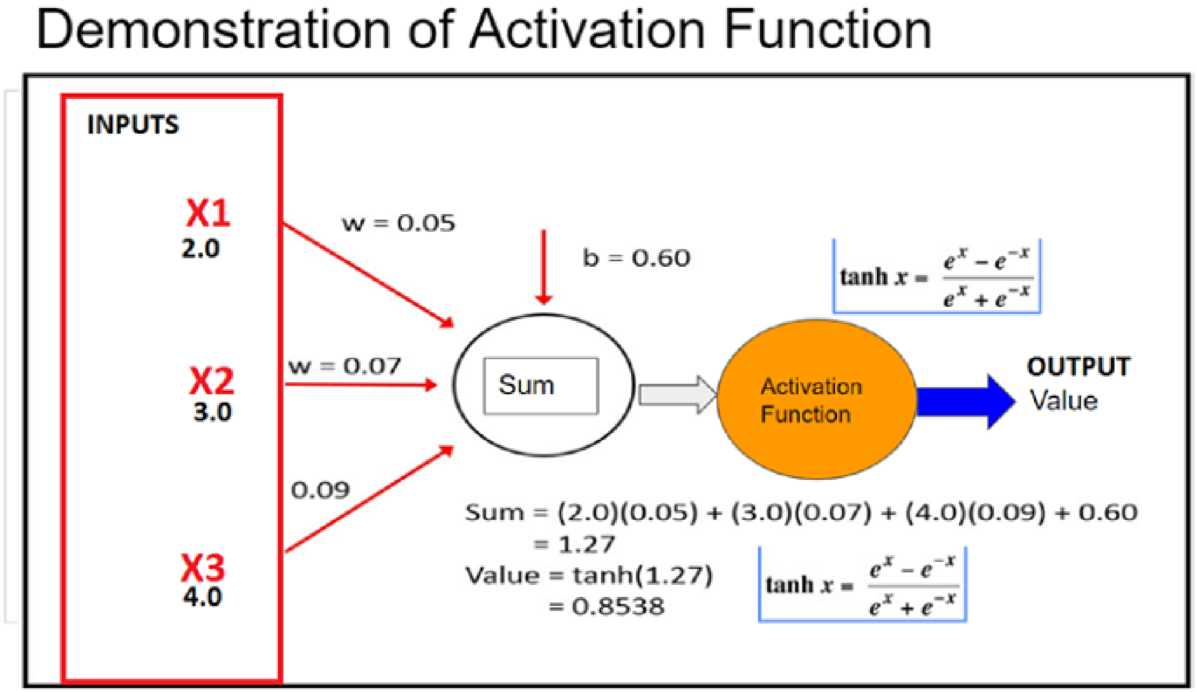
3.5. Running and explain **Linear Regression model** using TensorFlow Core API.

```
# importing the dependencies
import tensorflow.compat.v1 as tf
import numpy as np
import matplotlib.pyplot as plt

# Model Parameters
learning_rate = 0.01
training_epochs = 2000
display_step = 200

# Training Data
train_X =
np.asarray([3.3,4.4,5.5,6.71,6.93,4.168,9.779,6.182,7.59,2.167,

7.042,10.791,5.313,7.997,5.654,9.27,3.1])
train_y =
np.asarray([1.7,2.76,2.09,3.19,1.694,1.573,3.366,2.596,2.53,1.22
1,
                         2.827,3.465,1.65,2.904,2.42,2.94,1.3])
n_samples = train_X.shape[0]

# Test Data
test_X = np.asarray([6.83, 4.668, 8.9, 7.91, 5.7, 8.7, 3.1,
2.1])

# Set placeholders for feature and target vectors
X = tf.placeholder(tf.float32)
y = tf.placeholder(tf.float32)

# Set model weights and bias)
test_y = np.asarray([1.84, 2.273, 3.2, 2.831, 2.92, 3.24, 1.35,
1.03])
W = tf.Variable(np.random.randn(), name="weight")
b = tf.Variable(np.random.randn(), name="bias")

# Construct a linear model
linear_model = W*X + b

# Mean squared error
```

```
cost = tf.reduce_sum(tf.square(linear_model - y)) /
(2*n_samples)

# Gradient descent
optimizer =
tf.train.GradientDescentOptimizer(learning_rate).minimize(cost)

# Initializing the variables
init = tf.global_variables_initializer()

# Launch the graph
with tf.Session() as sess:
    # Load initialized variables in current session
    sess.run(init)
 # Fit all training data
    for epoch in range(training_epochs):

        # perform gradient descent step
        sess.run(optimizer, feed_dict={X: train_X, y: train_y})

        # Display logs per epoch step
        if (epoch+1) % display_step == 0:
            c = sess.run(cost, feed_dict={X: train_X, y:
train_y})
            print("Epoch:{0:6} \t Cost:{1:10.4} \t W:{2:6.4} \t
b:{3:6.4}".
                  format(epoch+1, c, sess.run(W), sess.run(b)))

    # Print final parameter values
    print("Optimization Finished!")
    training_cost = sess.run(cost, feed_dict={X: train_X, y:
train_y})
    print("Final training cost:", training_cost, "W:",
sess.run(W), "b:",
          sess.run(b), '\n')

    # Graphic display
    plt.plot(train_X, train_y, 'ro', label='Original data')
    plt.plot(train_X, sess.run(W) * train_X + sess.run(b),
label='Fitted line')
    plt.legend()
    plt.show()

    # Testing the model
    testing_cost = sess.run(tf.reduce_sum(tf.square(linear_model
- y)) / (2 * test_X.shape[0]),
                            feed_dict={X: test_X, y: test_y})
    print("Final testing cost:", testing_cost)
    print("Absolute mean square loss difference:",
abs(training_cost - testing_cost))

    # Display fitted line on test data
    plt.plot(test_X, test_y, 'bo', label='Testing data')
    plt.plot(train_X, sess.run(W) * train_X + sess.run(b),
label='Fitted line')
    plt.legend()
    plt.show()
```

3.6. Model Neuron Network

a. View Fig1.1 in the diagram. Code and explain



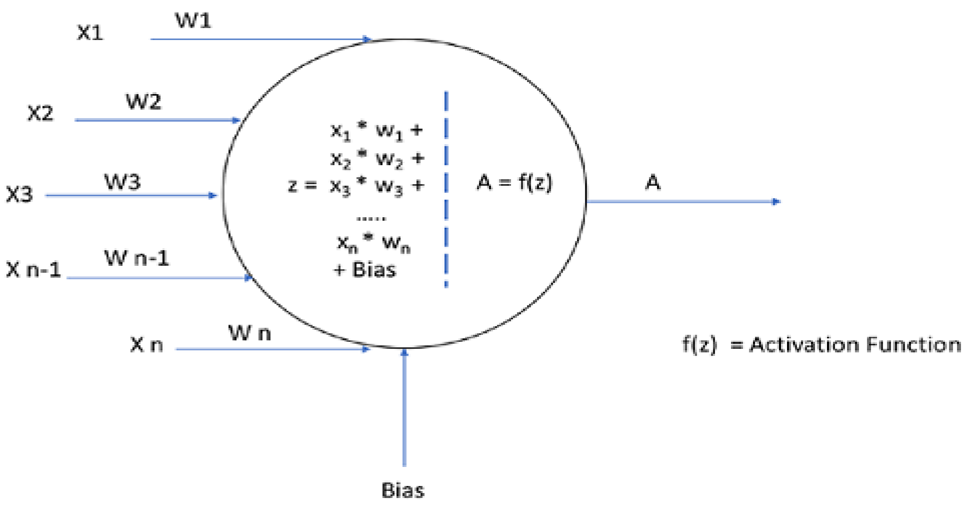
Demonstration of Activation Function
INPUTS
X1
2.0
X2
3.0
X3
4.0
w = 0.05
w = 0.07
0.09
b = 0.60
Sum
Activation Function
$\tanh x = \frac{e^x - e^{-x}}{e^x + e^{-x}}$
OUTPUT
Value
Sum = (2.0)(0.05) + (3.0)(0.07) + (4.0)(0.09) + 0.60
= 1.27
Value = tanh(1.27)
= 0.8538
$\tanh x = \frac{e^x - e^{-x}}{e^x + e^{-x}}$

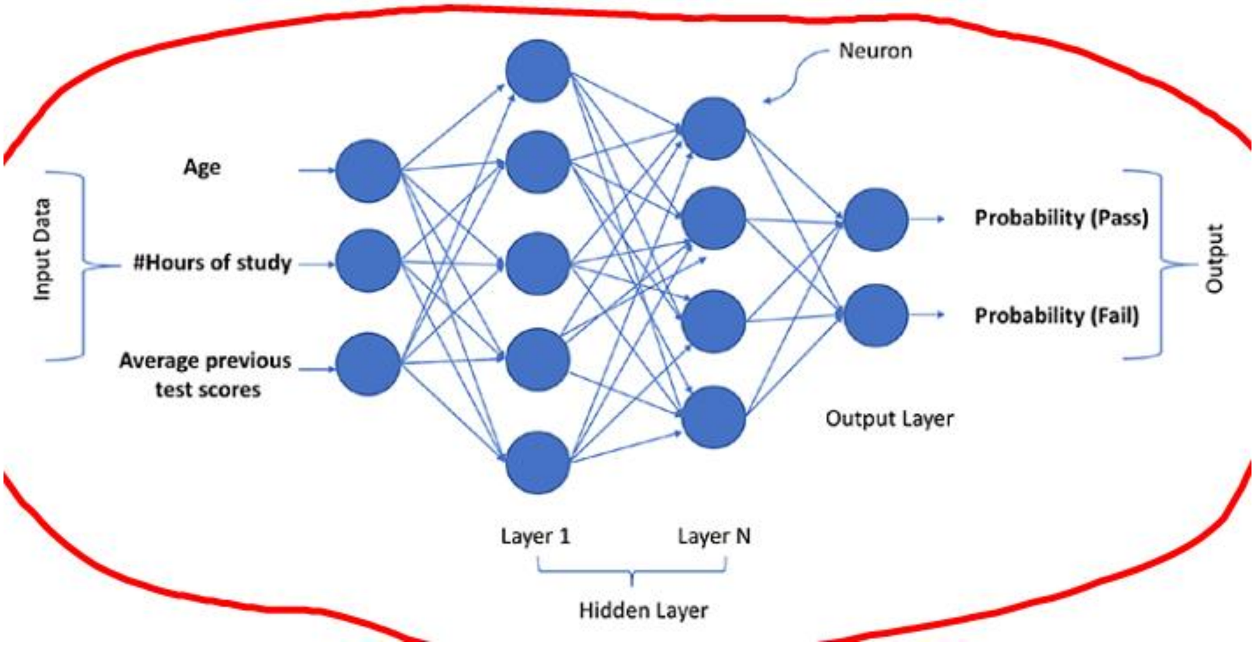
Figure 1-1. *An activation function*

b. Design a simple neuron network that s given in the digram. Student replaces tanh various sigmoid, relu… Visualize all versions and explain**.** Copy code and images of running program

3.7 Desing code with tensorflow for the network given in the following Figure. Copy code and running image. Explain code.

A Single Neuron


X1
W1
X2
W2
X3
W3
X n-1
W n-1
X n
W n
z = x1 * w1 + x2 * w2 + x3 * w3 + ... xn * wn + Bias
A = f(z)
A
f(z) = Activation Function
Bias



==========================

3.8 Refer to textbook 2 and Explain with simple examples: model, layer, batch, epoch, structure, loss, optimization, gradient. Explain Figures 2.21, 2.22, 2.23, 2.24, 2.25. Explain your understand in section 3.6

3.9 Running code processing MNIST with keras and WITHOUT keras (pure tensorflow). Refer Chapter 2 and 4

3.10 Present your understand of Chap 4. Running code with datasets MNIST and IMDB. Explain.

TensorFlow operator	Shortcut	Description
`tf.add()`	`a + b`	Adds a and b, element-wise.
`tf.multiply()`	`a * b`	Multiplies a and b, element-wise.
`tf.subtract()`	`a - b`	Subtracts a from b, element-wise.
`tf.divide()`	`a / b`	Computes Python-style division of a by b.
`tf.pow()`	`a ** b`	Returns the result of raising each element in a to its corresponding element b, element-wise.
`tf.mod()`	`a % b`	Returns the element-wise modulo.
`tf.logical_and()`	`a & b`	Returns the truth table of `a & b`, element-wise. `dtype` must be `tf.bool`.
`tf.greater()`	`a > b`	Returns the truth table of `a > b`, element-wise.
`tf.greater_equal()`	`a >= b`	Returns the truth table of `a >= b`, element-wise.
`tf.less_equal()`	`a <= b`	Returns the truth table of `a <= b`, element-wise.
`tf.less()`	`a < b`	Returns the truth table of `a < b`, element-wise.
`tf.negative()`	`-a`	Returns the negative value of each element in `a`.
`tf.logical_not()`	`~a`	Returns the logical NOT of each element in `a`. Only compatible with Tensor objects with `dtype` of `tf.bool`.
`tf.abs()`	`abs(a)`	Returns the absolute value of each element in `a`.
`tf.logical_or()`	`a | b`	Returns the truth table of `a | b`, element-wise. `dtype` must be `tf.bool`.

Data type	Python type	Description
`DT_FLOAT`	`tf.float32`	32-bit floating point.
`DT_DOUBLE`	`tf.float64`	64-bit floating point.
`DT_INT8`	`tf.int8`	8-bit signed integer.
`DT_INT16`	`tf.int16`	16-bit signed integer.
`DT_INT32`	`tf.int32`	32-bit signed integer.
`DT_INT64`	`tf.int64`	64-bit signed integer.
`DT_UINT8`	`tf.uint8`	8-bit unsigned integer.
`DT_UINT16`	`tf.uint16`	16-bit unsigned integer.
`DT_STRING`	`tf.string`	Variable-length byte array. Each element of a Tensor is a byte array.
`DT_BOOL`	`tf.bool`	Boolean.
`DT_COMPLEX64`	`tf.complex64`	Complex number made of two 32-bit floating points: real and imaginary parts.
`DT_COMPLEX128`	`tf.complex128`	Complex number made of two 64-bit floating points: real and imaginary parts.
`DT_QINT8`	`tf.qint8`	8-bit signed integer used in quantized ops.
`DT_QINT32`	`tf.qint32`	32-bit signed integer used in quantized ops.
`DT_QUINT8`	`tf.quint8`	8-bit unsigned integer used in quantized ops.