TensorFlow 2.0

Dogyeom Park and Seungchul Lee

Industrial AI Lab.

Training Neural Networks: Deep Learning Libraries

- TensorFlow
  - Google
  - Platform: Linux, Mac OS, Windows
  - Interface: Python, C/C++, Java, Go, R

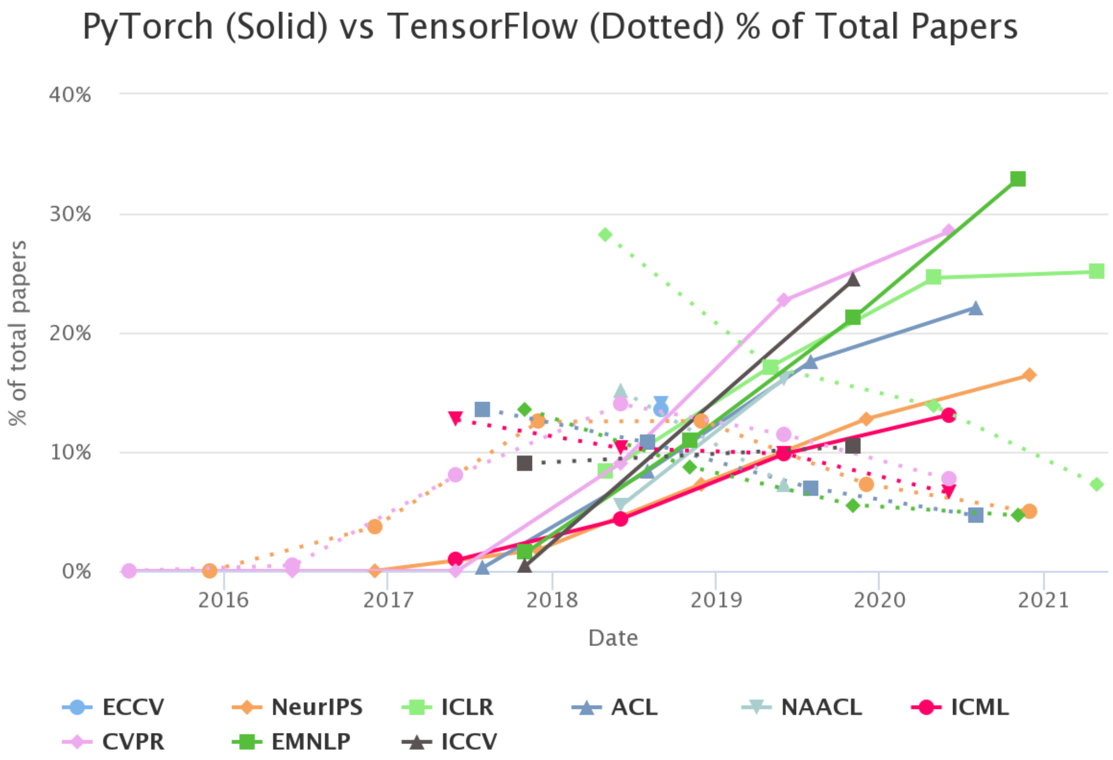
- Keras
  - Francois Chollet

- PyTorch
  - Facebook

TensorFlow

- Developed by Google and it is one of the most popular machine learning libraries involving large number of mathematical operations
- It is a framework to perform computation very efficiently, and it can tap into the GPU for higher speed

- TensorFlow 2.0
  - Keras is now combined with TensorFlow 2.0!



Keras

- import tensorflow as tf
- from tensorflow import keras

- TensorFlow 1.0과 TensorFlow 2.0은 서로 호환되지 않음
  - import tensorflow.compat.v1 as tf
  - tf.disable_v2_behavior()

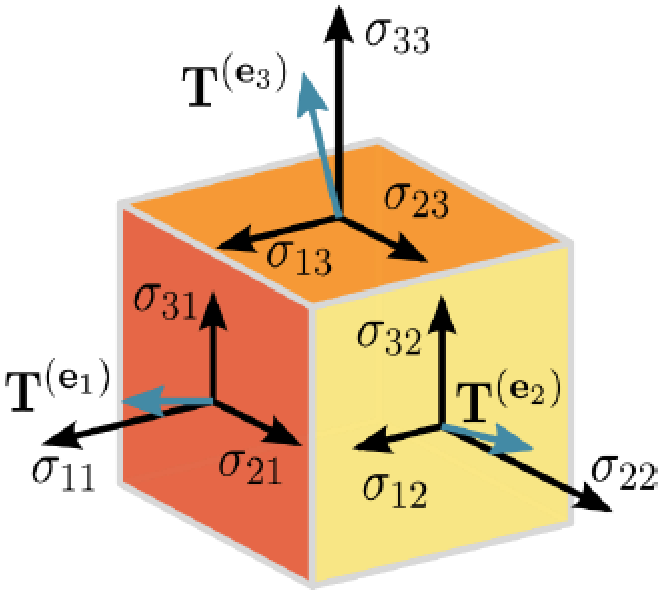
Why Do We Use TensorFlow?

- TensorFlow is only a tool for implementing deep learning easily
  - Easy to create matrix
  - Easy to multiply matrices
  - Easy to stack multiple layers, which is essential in deep learning
  - Parallel computing
    - Operations are conducted in parallel by CPU, GPU, TPU

- Why do we need matrix?
  - When creating a deep neural network, large number of ‘w’s needs to be stored and then used for ‘w1x1 + w2x2 + …’. TensorFlow automatically stores them as matrices and allows for fast matrix multiplication

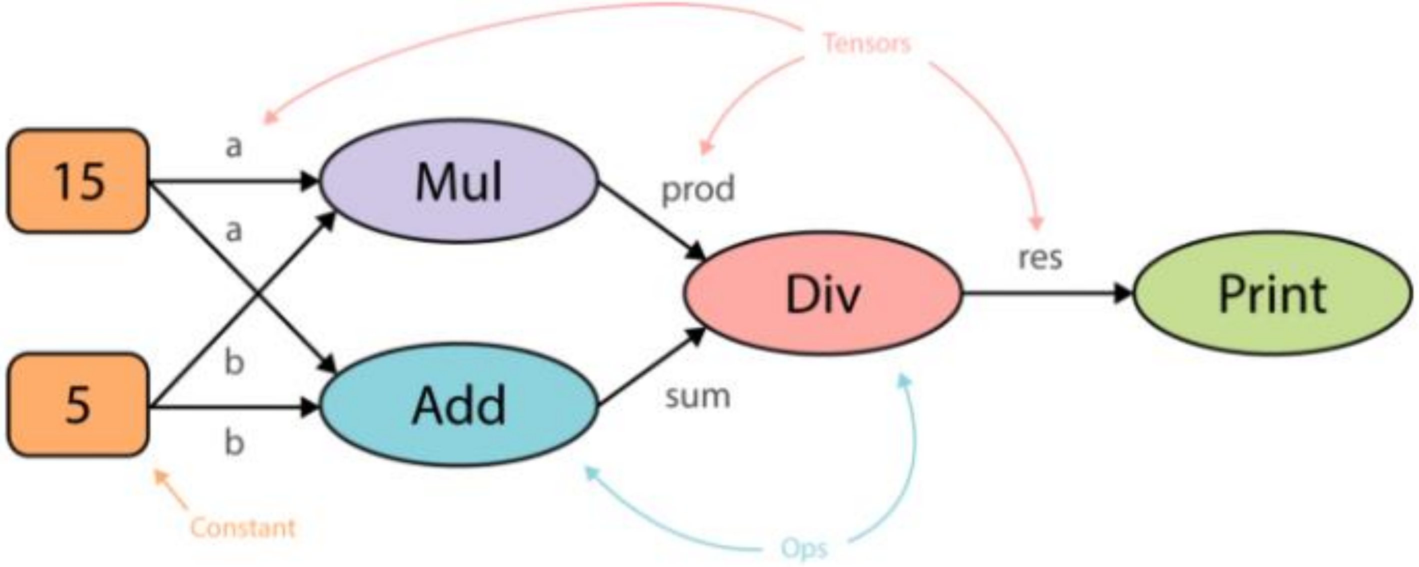
What is Tensor?

- TensorFlow gets its name from tensors, which are arrays of arbitrary dimensionality
- The ‘flow’ part of the name refers to computation flowing through a graph



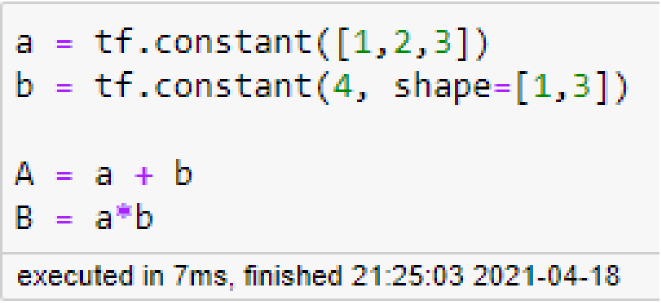
- 0-d tensor: scalar
- 1-d tensor: vector
- 2-d tensor: matrix

Computational Graph



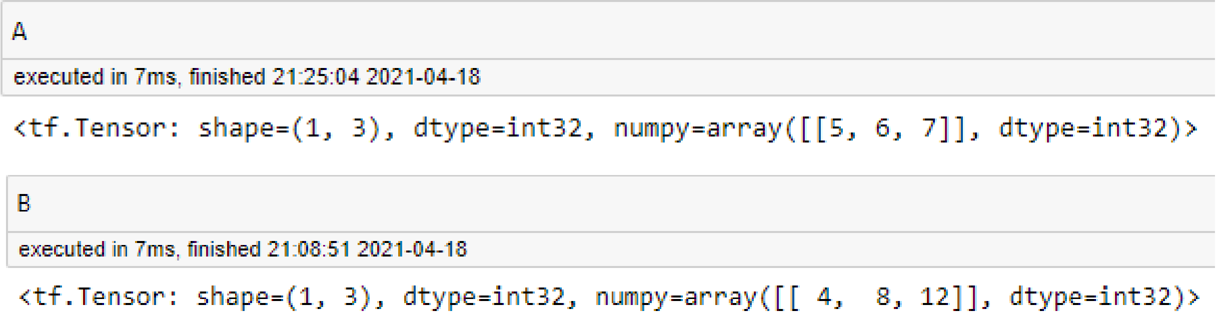
Basic Code

- tf.constant
  - Creates a constant tensor specified by value, shape, and type



```
a = tf.constant([1,2,3])
b = tf.constant(4, shape=[1,3])

A = a + b
B = a*b
```

executed in 7ms, finished 21:25:03 2021-04-18



```
A
```

executed in 7ms, finished 21:25:04 2021-04-18

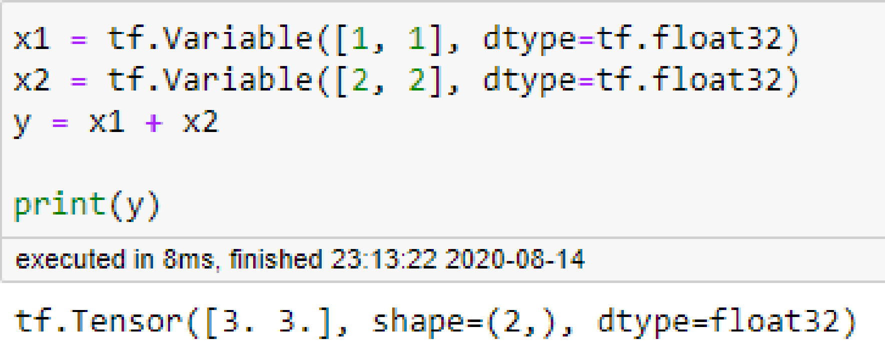
```
<tf.Tensor: shape=(1, 3), dtype=int32, numpy=array([[5, 6, 7]], dtype=int32)>
```

```
B
```

executed in 7ms, finished 21:08:51 2021-04-18

```
<tf.Tensor: shape=(1, 3), dtype=int32, numpy=array([[ 4,  8, 12]], dtype=int32)>
```

- tf.Variable
  - Regarded as the decision variable in optimization. We should initialize variables to use it



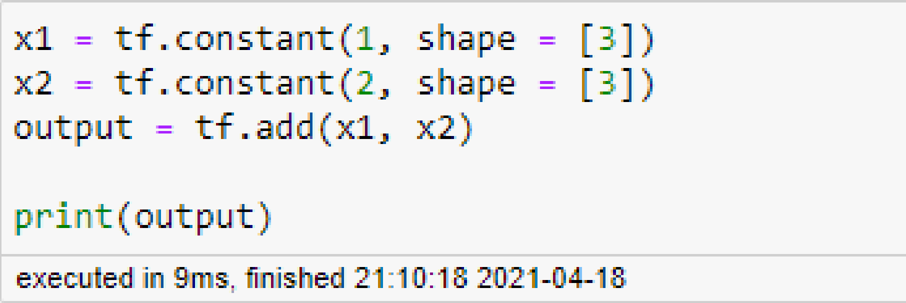
```
x1 = tf.Variable([1, 1], dtype=tf.float32)
x2 = tf.Variable([2, 2], dtype=tf.float32)
y = x1 + x2

print(y)
```

executed in 8ms, finished 23:13:22 2020-08-14

```
tf.Tensor([3. 3.], shape=(2,), dtype=float32)
```

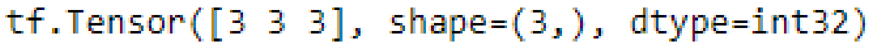
Basic Computation

- Adding matrices
  - tf.add



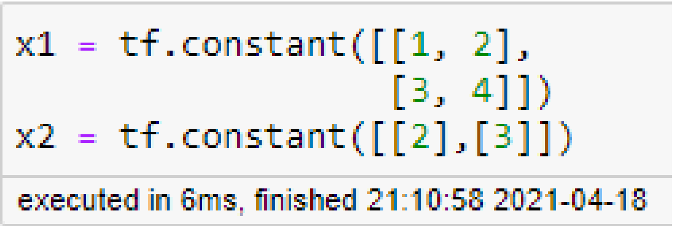
```
x1 = tf.constant(1, shape = [3])
x2 = tf.constant(2, shape = [3])
output = tf.add(x1, x2)

print(output)
```

executed in 9ms, finished 21:10:18 2021-04-18



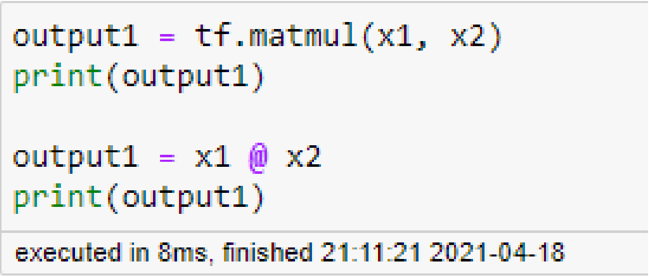
```
tf.Tensor([3 3 3], shape=(3,), dtype=int32)
```

- Multiplying matrices



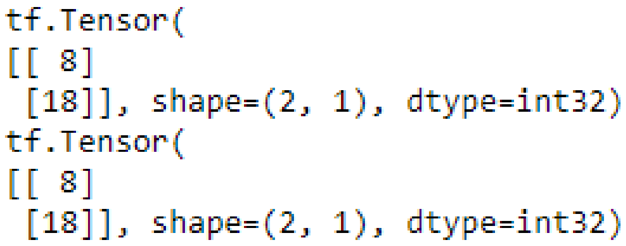
```
x1 = tf.constant([[1, 2],
                  [3, 4]])
x2 = tf.constant([[2],[3]])
```

executed in 6ms, finished 21:10:58 2021-04-18



```
output1 = tf.matmul(x1, x2)
print(output1)

output1 = x1 @ x2
print(output1)
```

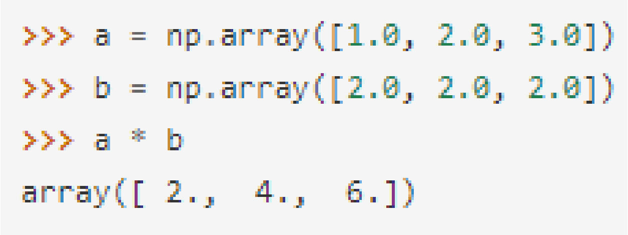
executed in 8ms, finished 21:11:21 2021-04-18



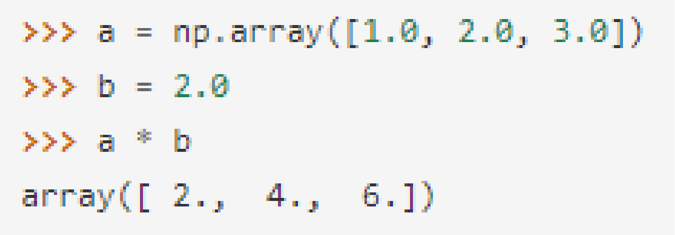
```
tf.Tensor(
[[ 8]
 [18]], shape=(2, 1), dtype=int32)
tf.Tensor(
[[ 8]
 [18]], shape=(2, 1), dtype=int32)
```

Basic Computation

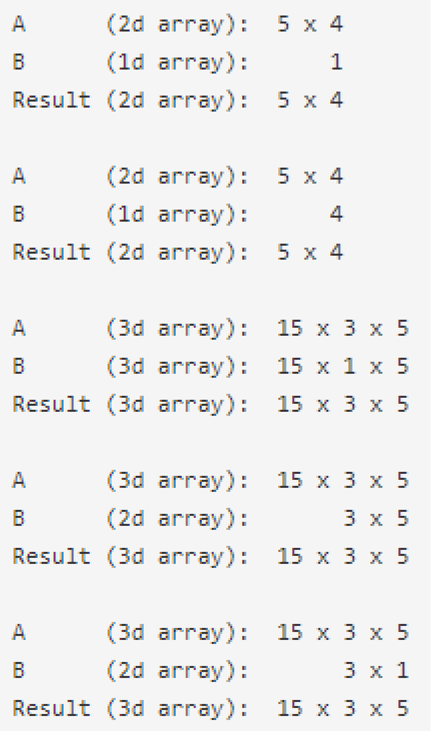
- Multiplying matrices
  - Broadcasting (same as numpy)
    - Smaller array is ‘broadcast’ across the larger array so that they have compatible shape



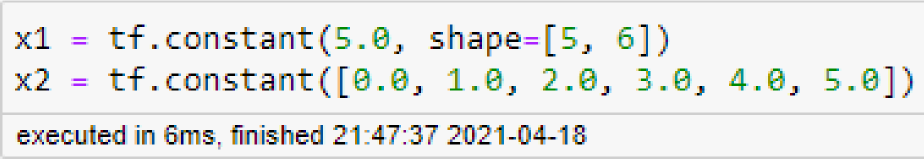
```
>>> a = np.array([1.0, 2.0, 3.0])
>>> b = np.array([2.0, 2.0, 2.0])
>>> a * b
array([ 2.,  4.,  6.])
```



```
>>> a = np.array([1.0, 2.0, 3.0])
>>> b = 2.0
>>> a * b
array([ 2.,  4.,  6.])
```

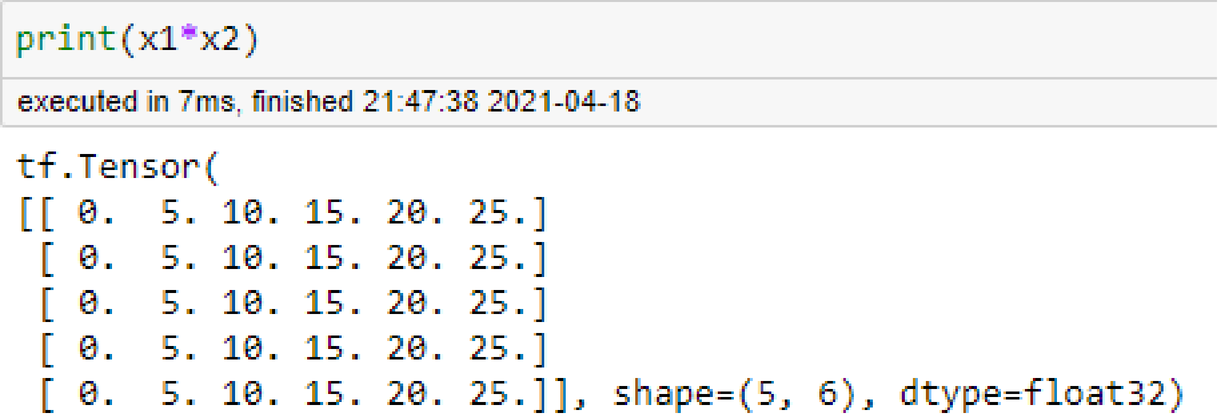


```
A      (2d array):  5 x 4
B      (1d array):      1
Result (2d array):  5 x 4

A      (2d array):  5 x 4
B      (1d array):      4
Result (2d array):  5 x 4

A      (3d array):  15 x 3 x 5
B      (3d array):  15 x 1 x 5
Result (3d array):  15 x 3 x 5

A      (3d array):  15 x 3 x 5
B      (2d array):       3 x 5
Result (3d array):  15 x 3 x 5

A      (3d array):  15 x 3 x 5
B      (2d array):       3 x 1
Result (3d array):  15 x 3 x 5
```



```
x1 = tf.constant(5.0, shape=[5, 6])
x2 = tf.constant([0.0, 1.0, 2.0, 3.0, 4.0, 5.0])
```

executed in 6ms, finished 21:47:37 2021-04-18



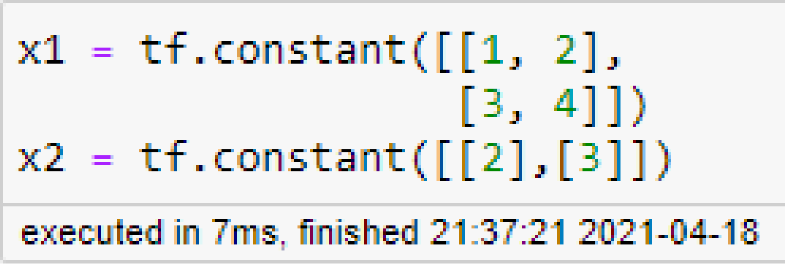
```
print(x1*x2)
```

executed in 7ms, finished 21:47:38 2021-04-18

```
tf.Tensor(
[[ 0.   5. 10. 15. 20. 25.]
 [ 0.   5. 10. 15. 20. 25.]
 [ 0.   5. 10. 15. 20. 25.]
 [ 0.   5. 10. 15. 20. 25.]
 [ 0.   5. 10. 15. 20. 25.]], shape=(5, 6), dtype=float32)
```

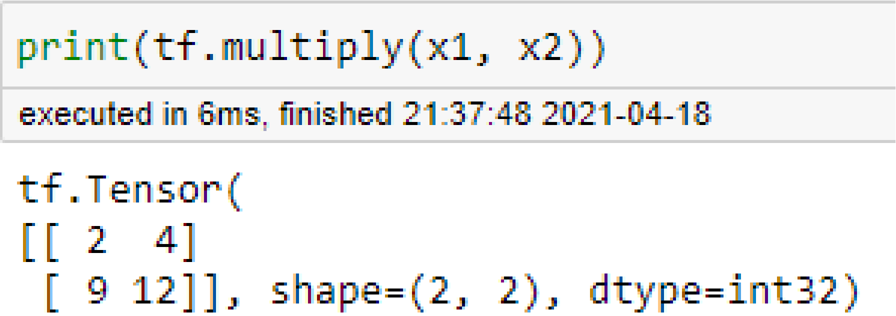
Basic Computation

- Multiplying matrices
  - Broadcasting (same as numpy)



```
x1 = tf.constant([[1, 2],
                  [3, 4]])
x2 = tf.constant([[2],[3]])
```

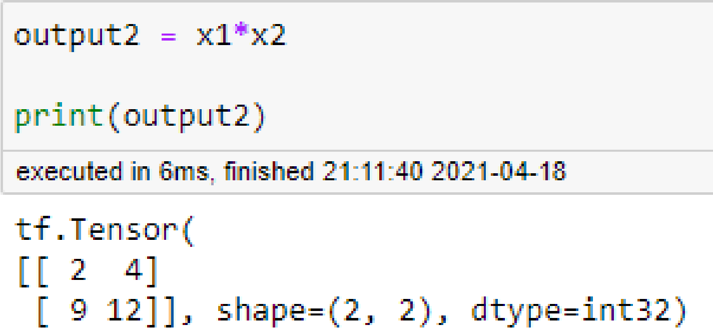
executed in 7ms, finished 21:37:21 2021-04-18



```
print(tf.multiply(x1, x2))
```

executed in 6ms, finished 21:37:48 2021-04-18

```
tf.Tensor(
[[ 2  4]
 [ 9 12]], shape=(2, 2), dtype=int32)
```



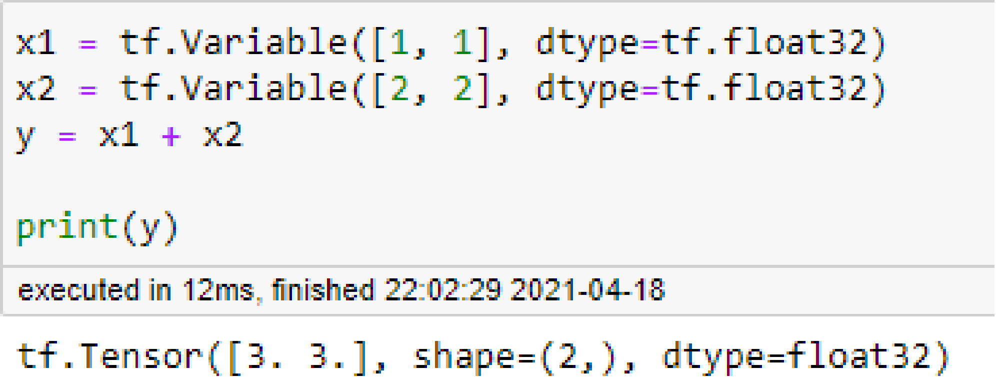
```
output2 = x1*x2

print(output2)
```

executed in 6ms, finished 21:11:40 2021-04-18

```
tf.Tensor(
[[ 2  4]
 [ 9 12]], shape=(2, 2), dtype=int32)
```

Variable

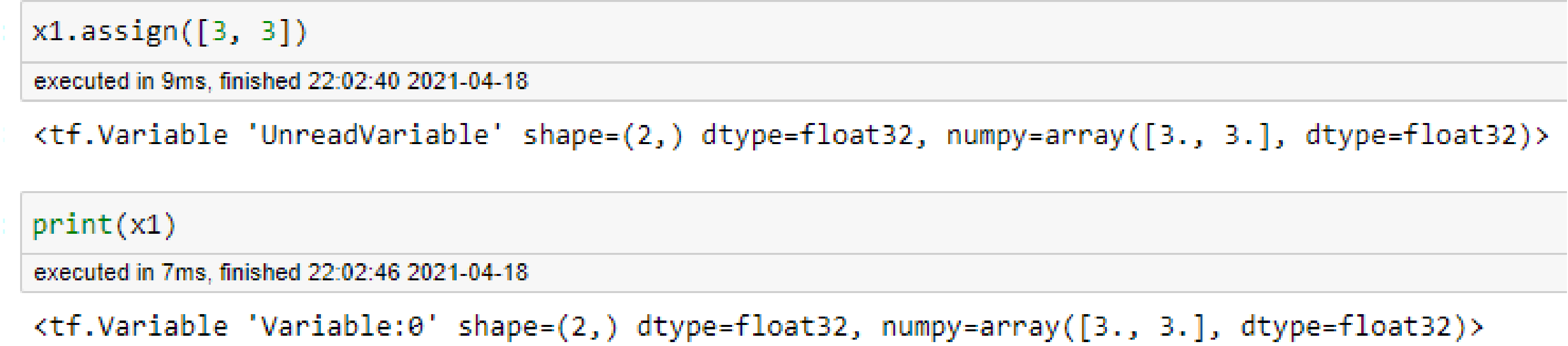
- tf.Variable
  - Weight



```
x1 = tf.Variable([1, 1], dtype=tf.float32)
x2 = tf.Variable([2, 2], dtype=tf.float32)
y = x1 + x2

print(y)
```

executed in 12ms, finished 22:02:29 2021-04-18

```
tf.Tensor([3. 3.], shape=(2,), dtype=float32)
```



```
x1.assign([3, 3])
```

executed in 9ms, finished 22:02:40 2021-04-18

```
<tf.Variable 'UnreadVariable' shape=(2,) dtype=float32, numpy=array([3., 3.], dtype=float32)>
```

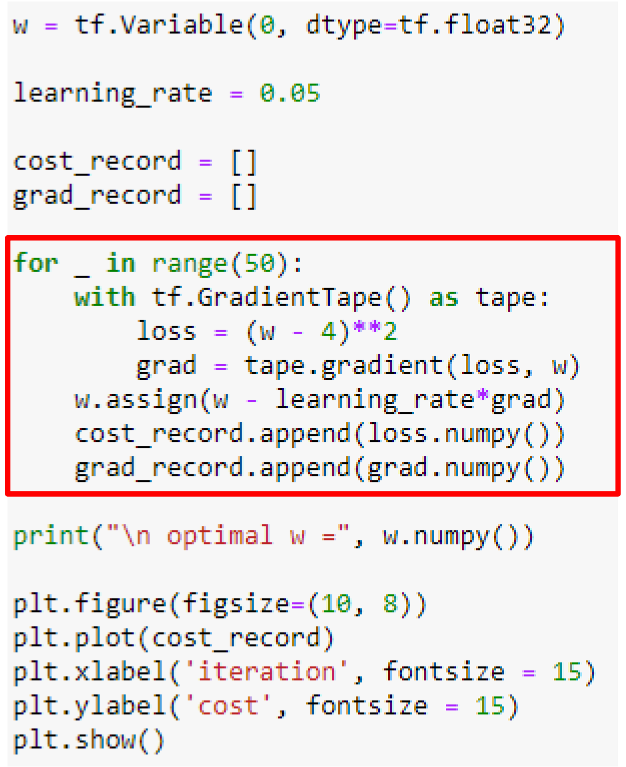
```
print(x1)
```

executed in 7ms, finished 22:02:46 2021-04-18

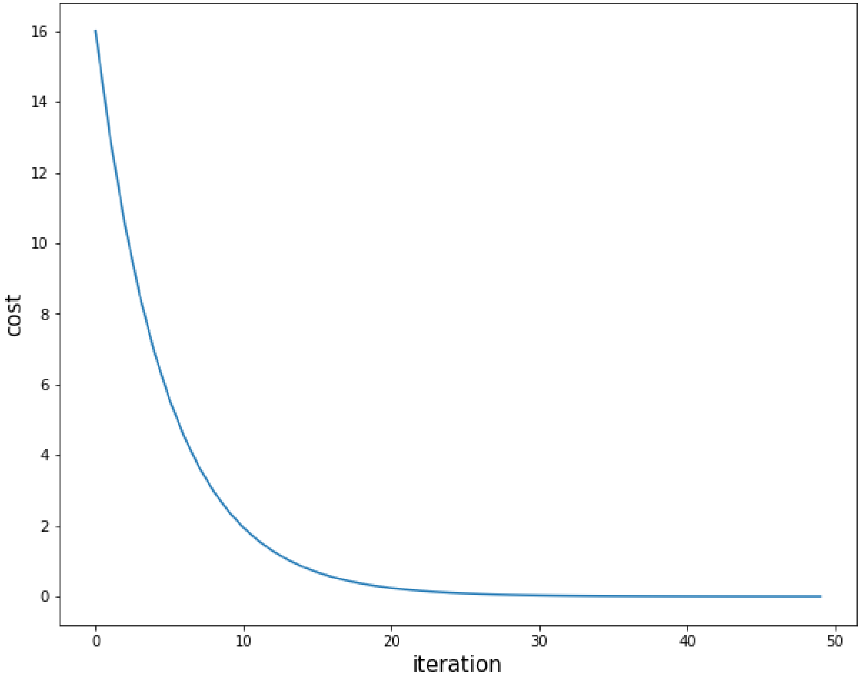
```
<tf.Variable 'Variable:0' shape=(2,) dtype=float32, numpy=array([3., 3.], dtype=float32)>
```

TensorFlow 2.0 as Optimization Solver

$$\min_{\omega} \ (\omega - 4)^2$$



```
w = tf.Variable(0, dtype=tf.float32)

learning_rate = 0.05

cost_record = []
grad_record = []

for _ in range(50):
    with tf.GradientTape() as tape:
        loss = (w - 4)**2
        grad = tape.gradient(loss, w)
    w.assign(w - learning_rate*grad)
    cost_record.append(loss.numpy())
    grad_record.append(grad.numpy())

print("\n optimal w =", w.numpy())

plt.figure(figsize=(10, 8))
plt.plot(cost_record)
plt.xlabel('iteration', fontsize = 15)
plt.ylabel('cost', fontsize = 15)
plt.show()
```



Keras

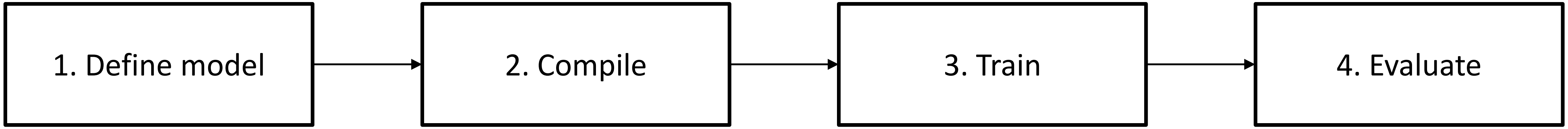
- Even simpler and more intuitive

- Supports all TensorFlow functions
  - import tensorflow as tf
  - from tensorflow import keras

- Sequential model
  - Stack multiple layers sequentially

Keras

- Framework



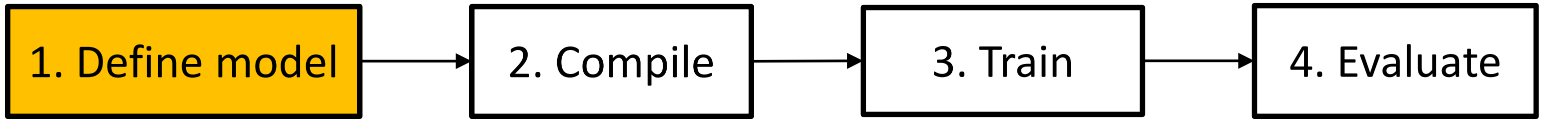
1. Define model

2. Compile

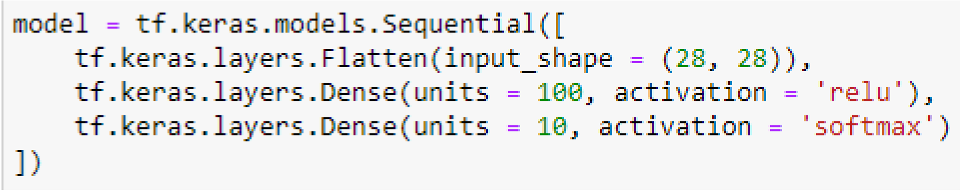
3. Train

4. Evaluate

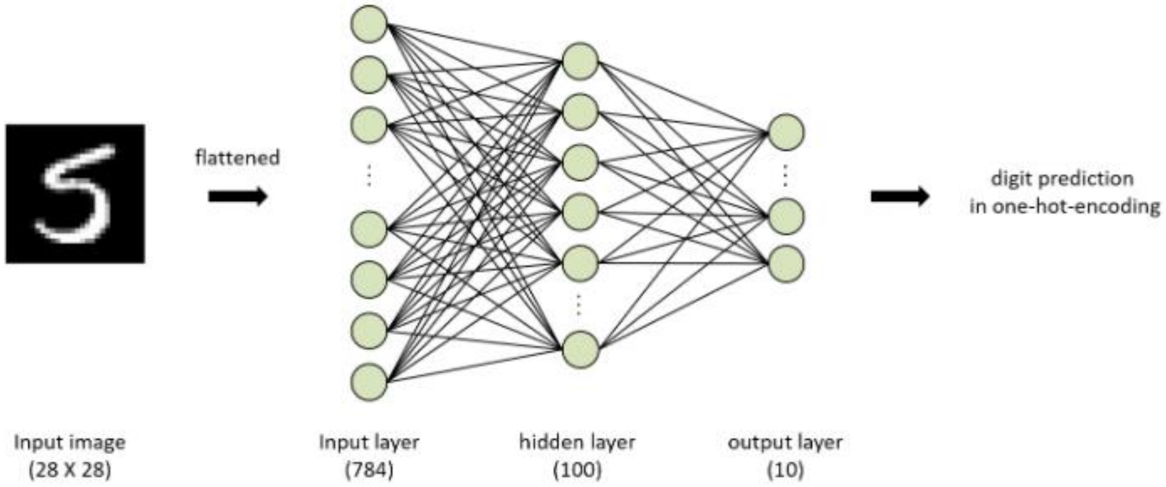
Keras


1. Define model
2. Compile
3. Train
4. Evaluate

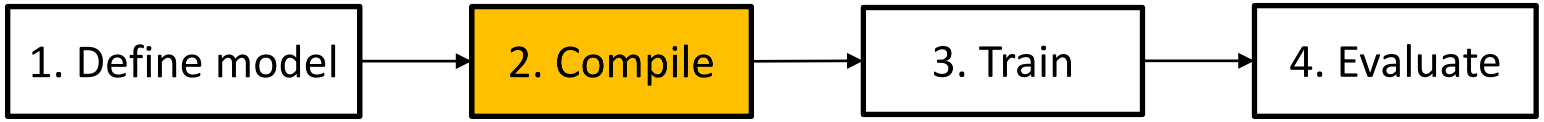
- Define model



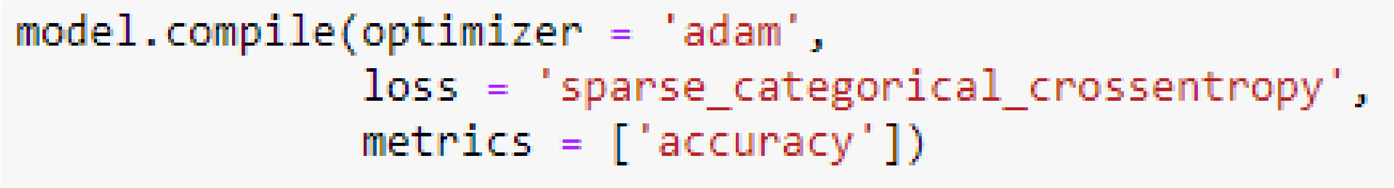
```
model = tf.keras.models.Sequential([
    tf.keras.layers.Flatten(input_shape = (28, 28)),
    tf.keras.layers.Dense(units = 100, activation = 'relu'),
    tf.keras.layers.Dense(units = 10, activation = 'softmax')
])
```



Keras

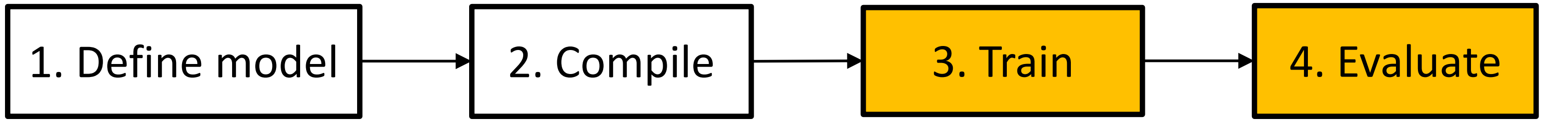


- Compile

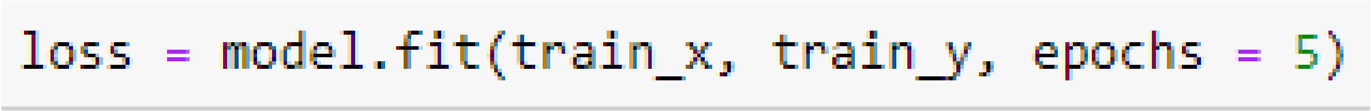


```
model.compile(optimizer = 'adam',
              loss = 'sparse_categorical_crossentropy',
              metrics = ['accuracy'])
```

Keras

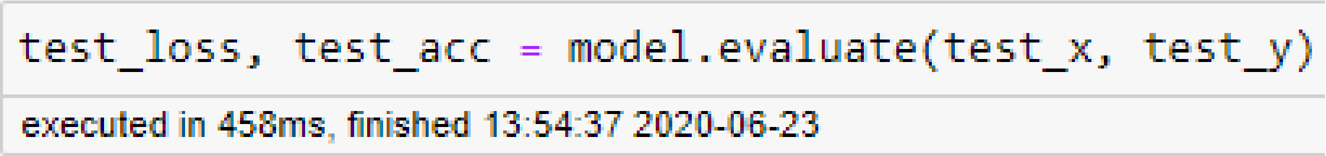


- Train



```
loss = model.fit(train_x, train_y, epochs = 5)
```

- Evaluate



```
test_loss, test_acc = model.evaluate(test_x, test_y)
executed in 458ms, finished 13:54:37 2020-06-23
```