THỰC HÀNH 2 : CƠ BẢN TENSORFLOW

Sau bài thực hành này, sinh viên có khả năng:

- Cài đặt được thư viện tensorflow trên môi trường Google Colab
- Xử lý được các đối tượng tính toán cơ bản trên tensorflow
- Tạo được các tensor đa chiều

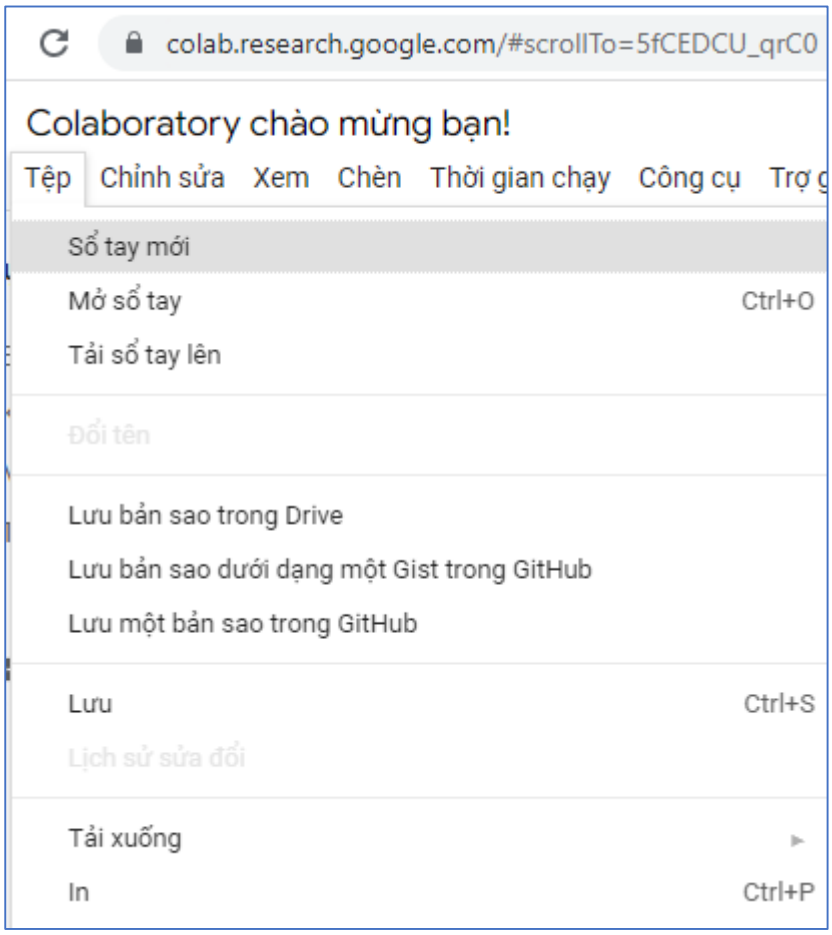
1. GIỚI THIỆU TENSORFLOW

- Tensor là một vector hay ma trận n-chiều biểu diễn cho tất cả các kiểu dữ liệu.
- Tất cả giá trị trong một tensor có cùng kiểu dữ liệu còn gọi là **shape.**
- **Shape** của dữ liệu là chiều của ma trận hoặc mảng (array).
- **Tensorflow là 1 framework xử lý các tensor.**
- Tất cả các tính toán đều được thực hiện trên **graph**
- **Graph** là tập các phép toán được thực hiện liên tục.
- Mỗi phép toán được gọi là 1 **opnode và có liên kết với nhau**

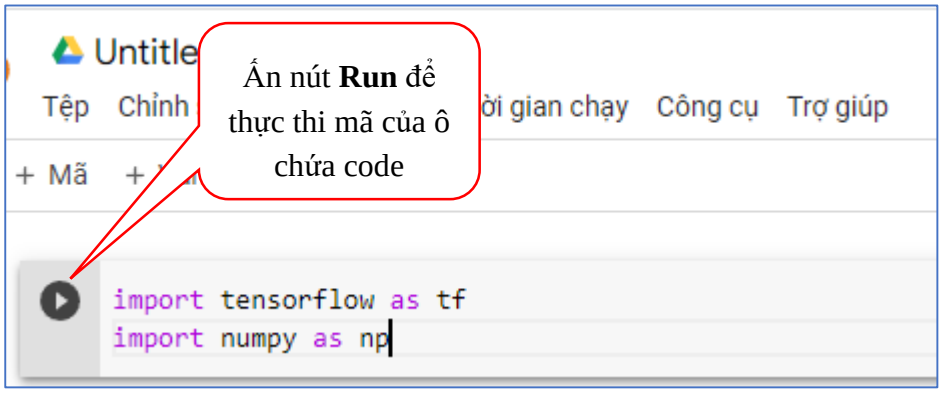
2. MÔI TRƯỜNG LẬP TRÌNH

Ta sử dụng môi trường Google Colab để thực hành. Yêu cầu sinh viên đăn nhập bằng tài khoản gmail và truy cập địa chỉ sau https://colab.research.google.com/

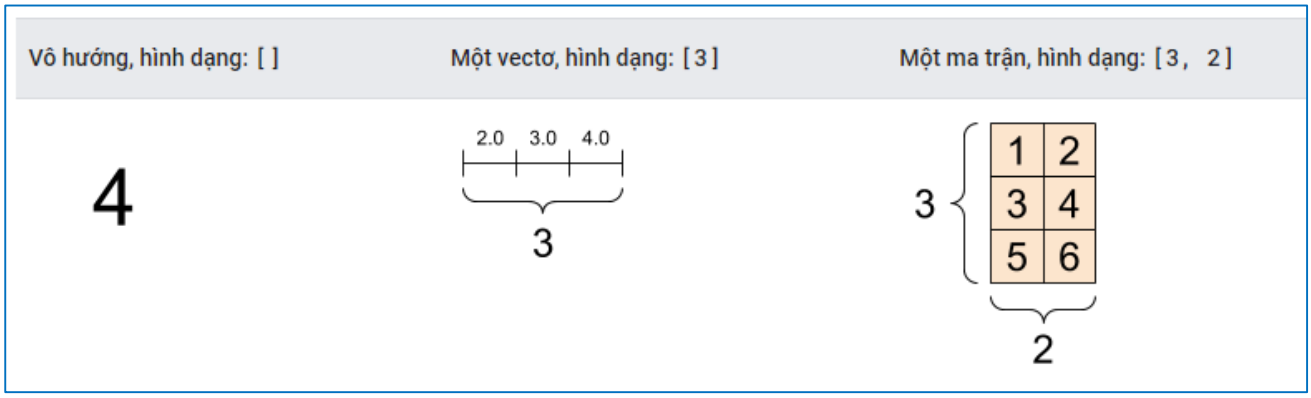
Sau đó chọn **Tệp → Sổ tay mới** và đặt tên là **tensorflow_basic.ipynb**



Chèn thư viện **tensorflow** và **numpy**



3. BIỂU DIỄN MỘT TENSOR



3.1. Tensor 0-chiều

Tensor 0-chiều gọi là scalar chỉ chứa 1 giá trị và không có trục.

Ví dụ: tạo một tensor scalar có giá trị là 4

```
# This will be an int32 tensor by default;
rank_0_tensor = tf.constant(4)
print(rank_0_tensor)
```

```
tf.Tensor(4, shape=(), dtype=int32)
```

Mặc định tạo một tensor có kiểu int32

Ví dụ: tạo một tensor scalar với kiểu dữ liệu là int16, float32 và string

```
# This will be an int32 tensor by default;
rank_0_tensor = tf.constant(4)
print(rank_0_tensor)
r1 = tf.constant(9, tf.int16)
print(r1)
r1_float = tf.constant(1.12, tf.float32)
print(r1_float)
r1_string = tf.constant('Hello Vanlang University', tf.string)
print(r1_string)
```

```
tf.Tensor(4, shape=(), dtype=int32)
tf.Tensor(9, shape=(), dtype=int16)
tf.Tensor(1.12, shape=(), dtype=float32)
tf.Tensor(b'Hello Vanlang University', shape=(), dtype=string)
```

3.2. Tensor 1-chiều

Tensor 1-chiều còn gọi là vector có 1 trục. Tensor 1-chiều giống như một danh sách

Ví dụ: tạo tensor 1-chiều gồm 3 phần tử

```
# We make this a float tensor.
rank_1_tensor = tf.constant([2.0, 3.0, 4.0])
print(rank_1_tensor)
```

```
tf.Tensor([2. 3. 4.], shape=(3,), dtype=float32)
```

3.3. Tensor 2-chiều (ma trận)

Là một ma trận hoặc có 2 trục

```
rank_2_tensor = tf.constant([[1, 2],
                             [3, 4],
                             [5, 6]], dtype=tf.float16)
print(rank_2_tensor)
```

```
tf.Tensor(
[[1. 2.]
 [3. 4.]
 [5. 6.]], shape=(3, 2), dtype=float16)
```

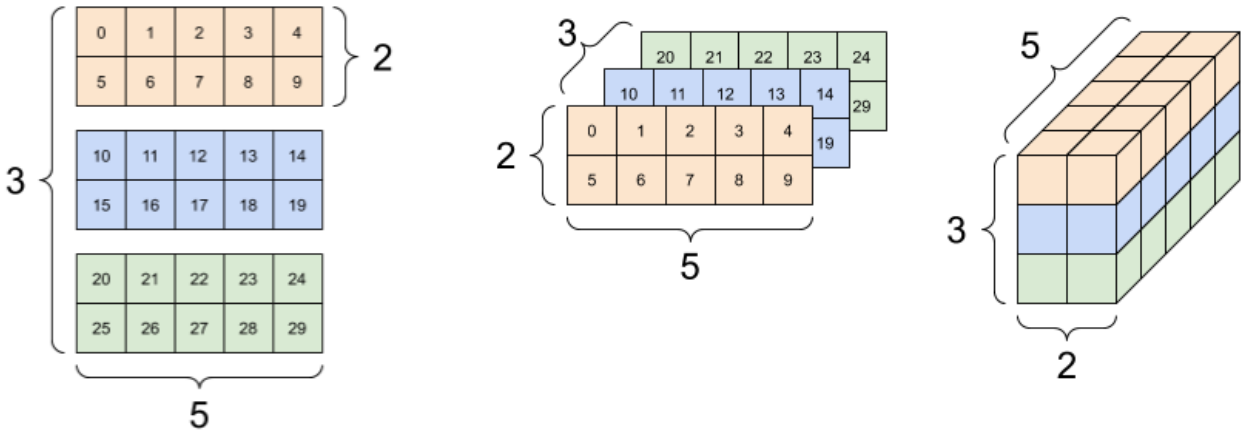
3.4. Tensor n-chiều (nhiều trục)

Ví dụ: tạo tensor 3-chiều

```
rank_3_tensor = tf.constant([
  [[0, 1, 2, 3, 4],
   [5, 6, 7, 8, 9]],
  [[10, 11, 12, 13, 14],
   [15, 16, 17, 18, 19]],
  [[20, 21, 22, 23, 24],
   [25, 26, 27, 28, 29]],])

print(rank_3_tensor)
```

```
tf.Tensor(
[[[ 0  1  2  3  4]
  [ 5  6  7  8  9]]

 [[10 11 12 13 14]
  [15 16 17 18 19]]

 [[20 21 22 23 24]
  [25 26 27 28 29]]], shape=(3, 2, 5), dtype=int32)
```

Tensor 3-chiều ví dụ trên có hình dạng [3, 2, 5]



3.5. Thuộc tính của tensor

Một tensor gồm có 3 thuộc tính:

- Label (name)
- Dimension (shape)
- Data type (dtype)

```
rank_3_tensor = tf.constant([
  [[0, 1, 2, 3, 4],
   [5, 6, 7, 8, 9]],
  [[10, 11, 12, 13, 14],
   [15, 16, 17, 18, 19]],
  [[20, 21, 22, 23, 24],
   [25, 26, 27, 28, 29]],])

print(rank_3_tensor)
print('shape:', rank_3_tensor.shape)
print('type of tensor:',rank_3_tensor.dtype)
```

```
tf.Tensor(
[[[ 0  1  2  3  4]
  [ 5  6  7  8  9]]

 [[10 11 12 13 14]
  [15 16 17 18 19]]

 [[20 21 22 23 24]
  [25 26 27 28 29]]], shape=(3, 2, 5), dtype=int32)
shape: (3, 2, 5)
type of tensor: <dtype: 'int32'>
```

3.6. Tạo tensor tự động gán giá trị 0 ban đầu

```
t_zero = tf.zeros(10)
print(t_zero)
```

```
tf.Tensor([0. 0. 0. 0. 0. 0. 0. 0. 0. 0.], shape=(10,), dtype=float32)
```

```
t_2_zero = tf.zeros([3, 3])
print(t_2_zero)
```

```
tf.Tensor(
[[0. 0. 0.]
 [0. 0. 0.]
 [0. 0. 0.]], shape=(3, 3), dtype=float32)
```

```
t_3_zero = tf.zeros([3, 3, 5])
print(t_3_zero)
```

```
tf.Tensor(
[[[0. 0. 0. 0. 0.]
  [0. 0. 0. 0. 0.]
  [0. 0. 0. 0. 0.]]

 [[0. 0. 0. 0. 0.]
  [0. 0. 0. 0. 0.]
  [0. 0. 0. 0. 0.]]

 [[0. 0. 0. 0. 0.]
  [0. 0. 0. 0. 0.]
  [0. 0. 0. 0. 0.]]], shape=(3, 3, 5), dtype=float32)
```

3.7. Tạo tensor tự động gán giá trị 1 ban đầu

```
t_one = tf.ones(10)
print(t_one)
```

```
tf.Tensor([1. 1. 1. 1. 1. 1. 1. 1. 1. 1.], shape=(10,), dtype=float32)
```

```
t_2_one = tf.ones([3,3])
print(t_2_one)
```

```
tf.Tensor(
[[1. 1. 1.]
 [1. 1. 1.]
 [1. 1. 1.]], shape=(3, 3), dtype=float32)
```

```
t_3_one = tf.ones([3,3, 5])
print(t_3_one)
```

```
tf.Tensor(
[[[1. 1. 1. 1. 1.]
  [1. 1. 1. 1. 1.]
  [1. 1. 1. 1. 1.]]

 [[1. 1. 1. 1. 1.]
  [1. 1. 1. 1. 1.]
  [1. 1. 1. 1. 1.]]

 [[1. 1. 1. 1. 1.]
  [1. 1. 1. 1. 1.]
  [1. 1. 1. 1. 1.]]], shape=(3, 3, 5), dtype=float32)
```

3.8. Tạo tensor tự động với giá trị bất kỳ

Tạo tensor 1-chiều có 3 phần tử có giá trị 10

```
t_1 = tf.fill(3, 10)
print(t_1)
```

```
tf.Tensor([10 10 10], shape=(3,), dtype=int32)
```

```
t_2 = tf.fill([3, 3], 9)
print(t_2)
```

```
tf.Tensor(
[[9 9 9]
 [9 9 9]
 [9 9 9]], shape=(3, 3), dtype=int32)
```

3.9. Tạo tensor 1-chiều với hàm linspace

Hàm linspace tạo một tensor với số phần tử mong muốn trong một phạm vi nhất định

```
lin_tensor = tf.linspace(5.0, 20.0, 5)
print(lin_tensor)
```

```
tf.Tensor([ 5.    8.75 12.5  16.25 20.  ], shape=(5,), dtype=float32)
```

Code trên tạo 5 phần tử có giá trị trong khoảng [5.0, 20.0]

3.10. Tạo tensor 1-chiều với hàm range

Hàm range tạo một tensor với số phần tử không được khai báo. Số phần tử sẽ được hàm tính bằng cách cộng thêm giá trị delta.

```
range_tensor = tf.range(3.0, 7.0, delta=0.5)
print(range_tensor)
```

```
tf.Tensor([3.  3.5 4.  4.5 5.  5.5 6.  6.5], shape=(8,), dtype=float32)
```

Code trên tạo một tensor trong khoảng [3.0, 7.0), mỗi giá trị mới cộng thêm delta=0.5

```
range_tensor = tf.range(3.0, delta=0.5)
print(range_tensor)
```

```
tf.Tensor([0.  0.5 1.  1.5 2.  2.5], shape=(6,), dtype=float32)
```

Code trên tạo 1 tensor trong khoảng [0.0, 3.0) nếu ta không khai báo giá trị ban đầu thì mặc định sẽ bắt đầu là 0.0

3.11. Tạo tensor với giá trị ngẫu nhiên

Function	Description
`random_normal(shape, mean=0.0, stddev=1.0, dtype=tf.float32, seed=None, name=None)`	Creates a tensor with normally distributed values
`truncated_normal(shape, mean=0.0, stddev=1.0, dtype=tf.float32, seed=None, name=None)`	Creates a tensor with normally distributed values excluding those lying outside two standard deviations
`random_uniform(shape, minval=0, maxval=None, dtype=tf.float32, seed=None, name=None)`	Creates a tensor with uniformly distributed values between the minimum and maximum values
`random_shuffle(tensor, seed=None, name=None)`	Shuffles a tensor along its first dimension
`set_random_seed(seed)`	Set the seed value for all random number generation in the graph

Ví dụ tạo tensor ngẫu nhiên với hàm random.normal

```
range_tensor = tf.random.normal([5])
print(range_tensor)
```

```
tf.Tensor([ 1.3137248   0.28413358 -0.07439218  0.99079883  0.44873315], shape=(5,), dtype=float32)
```

```
range_tensor = tf.random.normal([3,3])
print(range_tensor)
```

```
tf.Tensor(
[[-0.80970925  0.52634764  0.18630104]
 [ 1.3200511   0.48812193 -0.35287604]
 [ 0.9208391  -1.3945693  -0.16214153]], shape=(3, 3), dtype=float32)
```

Ví dụ tạo tensor ngẫu nhiên với hàm random.truncated_normal

```
range_tensor = tf.random.truncated_normal([5])
print(range_tensor)
```

```
tf.Tensor([-1.3134787   1.1004283  -0.6853268   0.37969747  1.6887165 ], shape=(5,), dtype=float32)
```

```
range_tensor = tf.random.truncated_normal([3, 2])
print(range_tensor)
```

```
tf.Tensor(
[[-0.10306646  0.3766136 ]
 [-1.9410796  -0.6633329 ]
 [-0.857614   -1.3217859 ]], shape=(3, 2), dtype=float32)
```

```
range_tensor = tf.random.truncated_normal([3, 2, 2])
print(range_tensor)
```

```
tf.Tensor(
[[[ 0.25656125 -0.5445741 ]
  [ 0.7954485   0.40173045]]

 [[-1.0182971   0.564984  ]
  [-0.9286383   0.6447141 ]]

 [[ 0.32566854 -0.45738077]
  [ 0.8643322   0.88202286]]], shape=(3, 2, 2), dtype=float32)
```

Ví dụ tạo tensor ngẫu nhiên với hàm random.uniform

```
range_tensor = tf.random.uniform([5], minval=10, maxval=20)
print(range_tensor)
```

```
tf.Tensor([19.96089  12.445324 13.524061 13.214278 14.372077], shape=(5,), dtype=float32)
```

```
range_tensor = tf.random.uniform([3, 2], minval=10, maxval=20)
print(range_tensor)
```

```
tf.Tensor(
[[14.65098  17.727661]
 [14.493259 17.61549 ]
 [19.62026  19.448536]], shape=(3, 2), dtype=float32)
```

```
range_tensor = tf.random.uniform([3, 2, 5], minval=10, maxval=20)
print(range_tensor)
```

```
tf.Tensor(
[[[16.092644 18.105427 15.17811  19.463688 18.692924]
  [10.929277 14.294512 15.002382 11.948755 10.381508]]

 [[13.17425  13.773919 18.290325 11.512618 10.911771]
  [14.662882 19.173141 19.555218 10.228304 17.350395]]

 [[13.936299 10.167504 13.19025  10.809617 15.963331]
  [17.84211  18.782251 10.543228 13.92933  12.074038]]], shape=(3, 2, 5), dtype=float32)
```

Ví dụ tạo tensor ngẫu nhiên với hàm **random.uniform** và dùng **shuffle** để xáo trộn giá trị trong tensor

```
range_tensor = tf.random.uniform([5], minval=10, maxval=20)
print(range_tensor)
print(tf.random.shuffle(range_tensor))
```

```
tf.Tensor([16.98472  13.894254 15.739035 11.069143 13.56448 ], shape=(5,), dtype=float32)
tf.Tensor([13.894254 13.56448  16.98472  15.739035 11.069143], shape=(5,), dtype=float32)
```

3.12. Chuyển đổi kiểu dữ liệu

Để chuyển từ kiểu dữ liệu này sang kiểu khác ta dùng hàm tf.cast

```
t_float = tf.constant(3.9, tf.float32)
print(t_float,'---', t_float.dtype)
t_int = tf.cast(t_float, dtype=tf.int32)
print(t_int, '---', t_int.dtype)
```

```
tf.Tensor(3.9, shape=(), dtype=float32) --- <dtype: 'float32'>
tf.Tensor(3, shape=(), dtype=int32) --- <dtype: 'int32'>
```

3.13. Các phép toán trong tensorflow

3.13.1.Tính căn bậc 2 của một tensor scalar

```
x = tf.constant(2, dtype=tf.float32)
print(tf.sqrt(x))
```

```
tf.Tensor(1.4142135, shape=(), dtype=float32)
```

3.13.2.Cộng 2 tensor

```
a = tf.constant([[1, 2]], tf.int32)
b = tf.constant([[3, 4]], tf.int32)
c = tf.add(a, b)
d = a + b
print(c)
print(d)
```

```
tf.Tensor([[4 6]], shape=(1, 2), dtype=int32)
tf.Tensor([[4 6]], shape=(1, 2), dtype=int32)
```

3.13.3.Nhân 2 tensor

```
a = tf.constant([[1, 2]], tf.int32)
b = tf.constant([[3, 4]], tf.int32)
c = tf.multiply(a, b)
d = a * b
print(c)
print(d)
```

```
tf.Tensor([[3 8]], shape=(1, 2), dtype=int32)
tf.Tensor([[3 8]], shape=(1, 2), dtype=int32)
```

Function	Description
`add(x, y, name=None)`	Adds two tensors
`subtract(x, y, name=None)`	Subtracts two tensors
`multiply(x, y, name=None)`	Multiplies two tensors
`divide(x, y, name=None)`	Divides the elements of two tensors
`div(x, y, name=None)`	Divides the elements of two tensors
`add_n(inputs, name=None)`	Adds multiple tensors
`scalar_mul(scalar, x)`	Scales a tensor by a scalar value
`mod(x, y, name=None)`	Performs the modulo operation
`abs(x, name=None)`	Computes the absolute value
`negative(x, name=None)`	Negates the tensor's elements
`sign(x, name=None)`	Extracts the signs of the tensor's element
`reciprocal(x, name=None)`	Computes the reciprocals

```
a = tf.constant([6. , 6. , 6.])
b = tf.constant([4. , 4. , 4.])
c = tf.constant([3. , 3. , 3.])
d = tf.add(a, b)
e = tf.add_n([a, b, c])
f = tf.subtract(a, b)
g = tf.multiply(a, b)
h = tf.divide(a, b)
i = tf.scalar_mul(3, a)
j = tf.math.reciprocal(a)
k = tf.math.mod(a, b)
l = tf.negative(a)

print(i)
print(l)
```

Sinh viên nhập code trên và xem kết quả

3.14. Các phép toán làm tròn và so sánh

Function	Description
`round(x, name=None)`	Rounds to the nearest integer, rounding up if there are two nearest integers
`rint(x, name=None)`	Rounds to the nearest integer, rounding to the nearest even integer if there are two nearest integers
`ceil(x, name=None)`	Returns the smallest integer greater than the value
`floor(x, name=None)`	Returns the greatest integer less than the value
`maximum(x, y, name=None)`	Returns a tensor containing the larger element of each input tensor
`minimum(x, y, name=None)`	Returns a tensor containing the smaller element of each input tensor
`argmax(x, axis=None, name=None, dimension=None)`	Returns the index of the greatest element in the tensor
`argmin(x, axis=None, name=None, dimension=None)`	Returns the index of the smallest element in the tensor

```
t = tf.constant([-6.5, -3.5, 3.5, 6.5])
r1 = tf.round(t)
r2 = tf.math.rint(t)
r3 = tf.math.ceil(t)
r4 = tf.math.floor(t)
print(r4)
```

```
tf.Tensor([-7. -4.  3.  6.], shape=(4,), dtype=float32)
```

3.15. Các phép toán trên vector và ma trận

Function	Description
`tensordot(a, b, axes, name=None)`	Returns the sum of products for the elements in the given axes
`cross(a, b, name=None)`	Returns the element-wise cross product
`diag(diagonal, name=None)`	Returns a matrix with the given diagonal values, other values set to zero
`trace(x, name=None)`	Returns the sum of the diagonal elements
`transpose(x, perm=None, name='transpose')`	Switches rows and columns
`eye(num_rows, num_columns=None, batch_shape=None, dtype=tf.float32, name=None)`	Creates an identity matrix with the given shape and data type
`matmul(a, b, transpose_a=False, transpose_b=False, adjoint_a=False, adjoint_b=False, a_is_sparse=False, b_is_sparse=False, name=None)`	Returns the product of the two input matrices
`norm(tensor, ord='euclidean', axis=None, keep_dims=False, name=None)`	Returns the norm of the given axis of the input tensor with the specified order
`matrix_solve(A, b, adjoint=None, name=None)`	Returns the tensor *x*, such that $Ax = b$, where *A* is a matrix, and *b* is a vector

Function	Description
`qr(input, full_matrices=None, name=None)`	Returns the eigenvectors and eigenvalues of the given matrix or matrices
`svd(tensor, full_matrices=False, compute_uv=True, name=None)`	Factors the matrix into a unitary matrix, a diagonal matrix, and the conjugate transpose of the unitary matrix
`einsum(equation, *inputs)`	Executes a custom mathematical operation

Bài tập: sinh viên tìm hiểu các hàm trên qua ví dụ sau

```
t1 = tf.constant([4., 3., 2.])
t2 = tf.constant([3., 2., 1.])
t3 = tf.constant([[1, 2, 3],
                  [4, 5, 6]])
t4 = tf.constant([[7, 8],
                  [9, 10],
                  [11, 12]])
dot_x = tf.tensordot(t1, t2, 0)
dot_y = tf.tensordot(t1, t2, 1)
print(t1)
print(t2)
print(t3)
print(t4)
cross = tf.linalg.cross(t1, t2)
mm = tf.matmul(t3, t4)
d = tf.linalg.diag(t3)
trace = tf.linalg.trace(t3)
trans = tf.transpose(t3)
eye = tf.eye(5, 5)
print(dot_x)
print(d)
print(cross)
print(trace)
print(trans)
print(eye)
```

3.16. Biến trong tensorflow

Để khai báo một biến trong tensorflow, ta thực hiện như sau

```
toan = tf.Variable(8.5, dtype=tf.float32)
print(toan)
diem = tf.Variable([5, 9])
print(diem)
print('Shape', diem.shape)
print('Dtype', diem.dtype)
print('numpy:', diem.numpy())
```

```
<tf.Variable 'Variable:0' shape=() dtype=float32, numpy=8.5>
<tf.Variable 'Variable:0' shape=(2,) dtype=int32, numpy=array([5, 9], dtype=int32)>
Shape (2,)
Dtype <dtype: 'int32'>
numpy: [5 9]
```

Phép gán giá trị cho tensor

```
toan = tf.Variable(8.5, dtype=tf.float32)
print(toan)
diem = tf.Variable([5, 9])
print(diem)
diem.assign([8, 10])
print(diem)
```

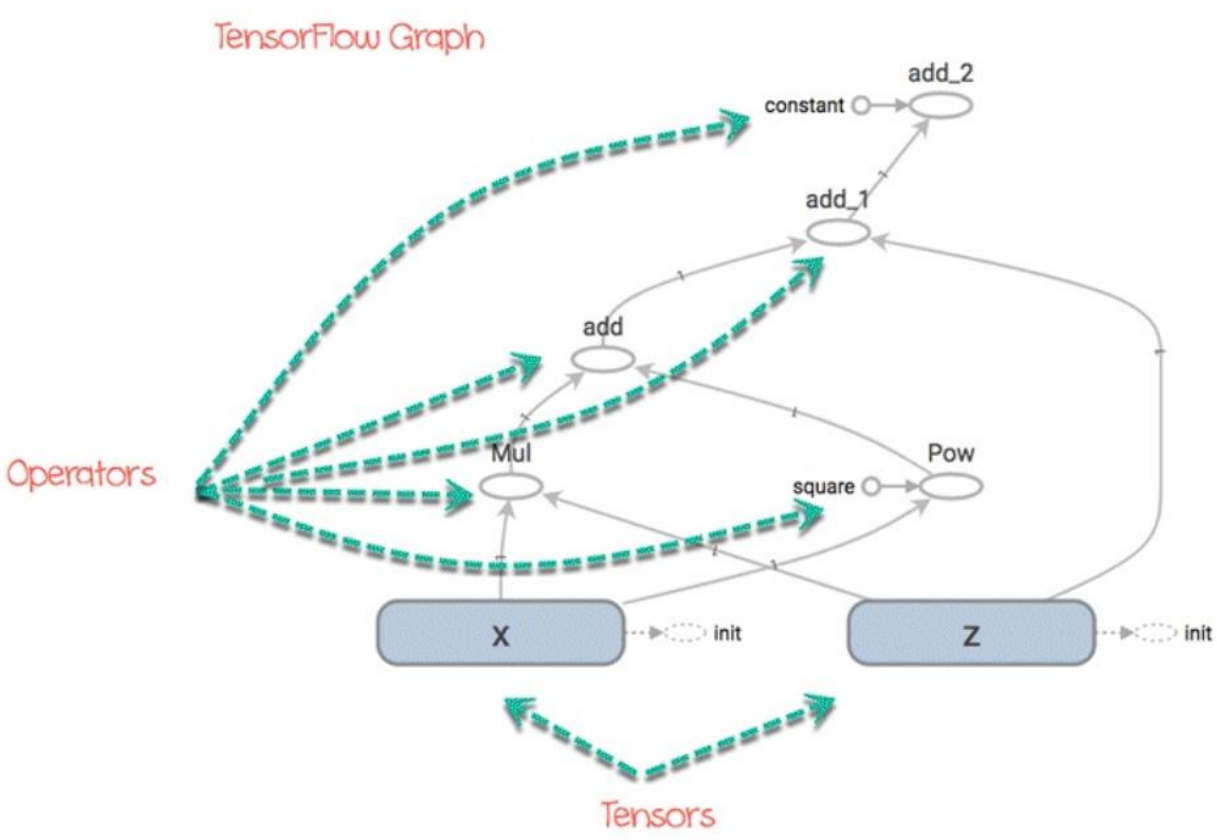
```
<tf.Variable 'Variable:0' shape=() dtype=float32, numpy=8.5>
<tf.Variable 'Variable:0' shape=(2,) dtype=int32, numpy=array([5, 9], dtype=int32)>
<tf.Variable 'Variable:0' shape=(2,) dtype=int32, numpy=array([ 8, 10], dtype=int32)>
```

```
toan = tf.Variable(8.5, dtype=tf.float32)
print(toan)
diem = tf.Variable([5, 9])
print(diem)
diem.assign([8, 10])
diem.assign_add([1, 1])
print(diem)
diem.assign_sub([5, 5])
print(diem)
```

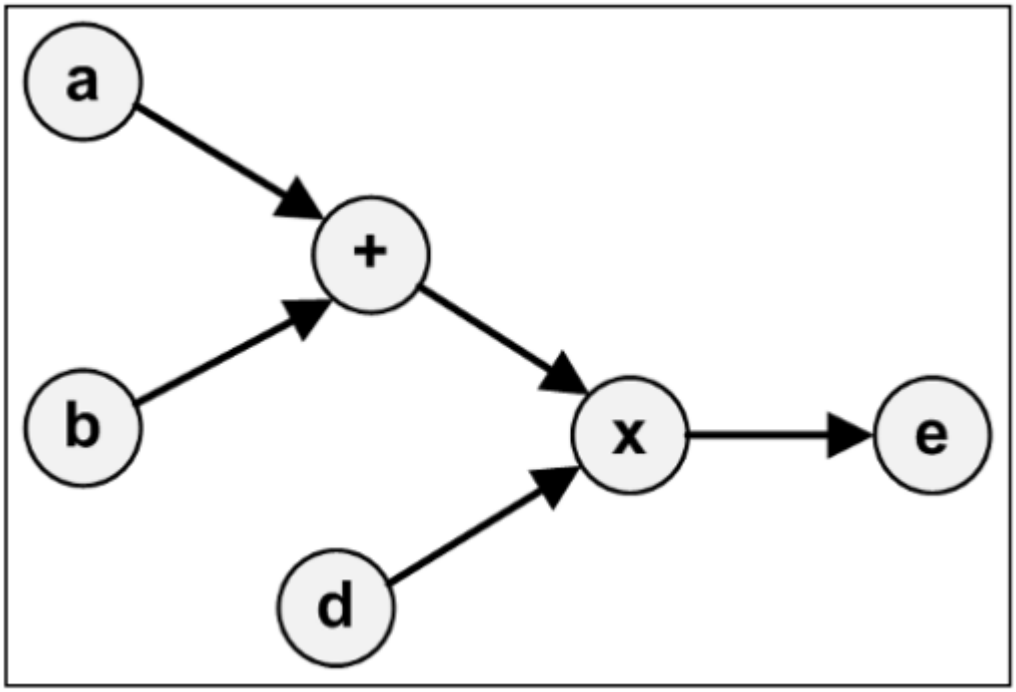
3.17. Môi trường tính toán Graph trong tensorflow

Graph cho phép thực thi tính toán linh hoạt ngoài môi trường Python. **Graph execution** cho phép các tính toán tensor theo cơ chế graph.

Graph là cấu trúc dữ liệu chứa các các đối tượng **tf.Operation** biểu diễn cho một đơn vị tính toán; và **tf.Tensor** biểu diễn cho một đơn vị dữ liệu truyền qua lại giữa các **tf.Operation**.




```
c = tf.add(a, b)
e = tf.multiply(c, d)
```



Graph có thể được sử dụng trong môi trường không cần bộ thông dịch Python như mobile, thiết bị nhúng, …

Graph giúp tensorflow chạy nhanh, tính toán song song và trên nhiều thiết bị

```
# Define a Python function.
def a_regular_function(x, y, b):
  x = tf.matmul(x, y)
  x = x + b
  return x

# `a_function_that_uses_a_graph` is a TensorFlow `Function`.
a_function_that_uses_a_graph = tf.function(a_regular_function)

# Make some tensors.
x1 = tf.constant([[1.0, 2.0]])
y1 = tf.constant([[2.0], [3.0]])
b1 = tf.constant(4.0)

orig_value = a_regular_function(x1, y1, b1).numpy()
# Call a `Function` like a Python function.
tf_function_value = a_function_that_uses_a_graph(x1, y1, b1).numpy()

print(orig_value)
print(tf_function_value)
```

```
[[12.]]
[[12.]]
```

Đoạn mã trên, ta định nghĩa 1 hàm Python thuần túy tên a_regular_function và dùng hàm tf.function để đưa hàm a_regular_function vào graph của tensorflow.

Cách gọi hàm trên graph tương tự như gọi hàm trong Python

3.17.1.Biến một hàm Python thuần túy sang graph

Ta có thể dùng decorator để biến một hàm Python thành graph

```
def inner_function(x, y, b):
  x = tf.matmul(x, y)
  x = x + b
  return x

# Use the decorator to make `outer_function` a `Function`.
@tf.function
def outer_function(x):
  y = tf.constant([[2.0], [3.0]])
  b = tf.constant(4.0)

  return inner_function(x, y, b)

# Note that the callable will create a graph that
# includes `inner_function` as well as `outer_function`.
rs = outer_function(tf.constant([[1.0, 2.0]])).numpy()
print(rs)
```

```
[[12.]]
```

3.18. Modules, Layers và Models

Một model là :

- Một hàm tính toán trên tensors
- Một vài biến được cập nhật trong quá trình huấn luyện

Hầu hết các model có cấu tạo từ layers. Layers là những hàm có cấu trúc toán học có thể được tái sử dụng và có biến huấn luyện. Các layers được cài đặt trên lớp cơ sở tf.Module

Ví dụ tạo lớp SimpleModule xử lý tensor scalar (Layer)

```
class SimpleModule(tf.Module):
  def __init__(self, name=None):
    super().__init__(name=name)
    self.a_variable = tf.Variable(5.0, name="train_me")
    self.non_trainable_variable = tf.Variable(5.0, trainable=False, name="do_not_train_me")
  def __call__(self, x):
    return self.a_variable * x + self.non_trainable_variable

simple_module = SimpleModule(name="simple")

rs = simple_module(tf.constant(5.0))
print(rs)
print('Train variables', simple_module.trainable_variables)
print('All variables', simple_module.variables)
```

```
tf.Tensor(30.0, shape=(), dtype=float32)
Train variables (<tf.Variable 'train_me:0' shape=() dtype=float32, numpy=5.0>,)
All variables (<tf.Variable 'train_me:0' shape=() dtype=float32, numpy=5.0>, <tf.Variable 'do_not_train_me:0' shape=() dtype=float32, numpy=5.0>)
```

Ví dụ tạo model với nhiều tầng (layer).

```
class Dense(tf.Module):
  def __init__(self, in_features, out_features, name=None):
    super().__init__(name=name)
    self.w = tf.Variable(
      tf.random.normal([in_features, out_features]), name='w')
    self.b = tf.Variable(tf.zeros([out_features]), name='b')
  def __call__(self, x):
    y = tf.matmul(x, self.w) + self.b
    return tf.nn.relu(y)

class SequentialModule(tf.Module):
  def __init__(self, name=None):
    super().__init__(name=name)

    self.dense_1 = Dense(in_features=3, out_features=3)
    self.dense_2 = Dense(in_features=3, out_features=2)

  def __call__(self, x):
    x = self.dense_1(x)
    return self.dense_2(x)

# You have made a model!
my_model = SequentialModule(name="the_model")

# Call it, with random results
print("Model results:", my_model(tf.constant([[2.0, 2.0, 2.0]])))
```

```
Model results: tf.Tensor([[2.1907609 0.       ]], shape=(1, 2), dtype=float32)
```

Lớp Dense là một layer có hàm toán học Linear trong hàm __call__

Lớp SequentialModule là một model vì có chứa 2 layers Dense

Hàm relu có nhiệm vụ hạn chế việc model trả về kết quả tuyến tính và được định nghĩa

$$f(x) = max(0, x)$$

3.19. Lưu weight

Chúng ta có thể lưu tf.Module ở dạng **checkpoint** lẫn **SavedModel**

3.19.1. Lưu checkpoint

Checkpoint là các weight (là giá trị của tập biến trong model hoặc submodel)

```
chkp_path = "my_checkpoint"
checkpoint = tf.train.Checkpoint(model=my_model)
checkpoint.write(chkp_path)
```

```
'my_checkpoint'
```

Trong đoạn mã trên, tensorflow sẽ tạo 2 file

- my_checkpoint.data-00000-of-00001: chứa giá trị của các biến và đường dẫn chứa thuộc tính biến
- my_checkpoint.index: lưu trữ đánh số checkpoints.

3.19.2. Phục hồi checkpoint

Để nạp lại model đã lưu bằng checkpoint, ta dùng hàm restore

```
new_model = SequentialModule()
new_checkpoint = tf.train.Checkpoint(model=new_model)
new_checkpoint.restore("my_checkpoint")

# Should be the same result as above
print(new_model(tf.constant([[2.0, 2.0, 2.0]])))
```

```
tf.Tensor([[2.1907609 0.       ]], shape=(1, 2), dtype=float32)
```

4. BÀI TẬP

1. Tạo một tensor có kích thước [2, 3] với các phần tử được gán 0
2. Cho X = [[1, 2, 3], [4, 5, 6]], hãy tạo tensor có kích thước và dtype như X với các phần tử được gán 0
3. Tạo một tensor có kích thước [3, 2] với các phần tử được gán 5
4. Tạo một tensor 1-chiều với 50 phần tử chẵn trong khoảng [5, 10]
5. Tạo một tensor 1-chiều có giá trị như [10, 12, 14, …, 100]
6. Tạo một tensor kích thước [3, 2] có giá trị ngẫu nhiên theo phân phối normal với mean=0, standard deviation = 2
7. Tạo một tensor kích thước [3, 2] có giá trị ngẫu nhiên theo phân phối uniform trong phạm vi [0, 2]
8. Cho tensor X = [[1, 2], [3, 4], [5, 6]]. Hãy shuffle X theo chiều thứ nhất (sử dụng random_shuffle)
9. Hãy tạo ngẫu nhiên tensor kích thước (10, 10, 3). Cắt thành tensor có kích thước (5, 5, 3) (sử dụng random_crop)
10. Tạo ngẫu nhiên hai tensor 1-chiều x, y có 5 phần tử. Trả về x + y nếu x < y, và x – y nếu x > y
11. Tạo ngẫu nhiên 2 tensor kích thước x = (3, 2) và giá trị trong khoảng [5, 10], y = (2, 3) và giá trị [15, 20] tương ứng. Hãy giữ giá trị đường chéo của x và thực hiện x + y.

12. Viết hàm tạo 2 tensor ngẫu nhiên có kích thước x = (3, 2, 2), y = (3, 2, 2). Viết hàm tính cộng, trừ, nhân, chia sau đó chuyển các hàm này vào trong graph.
13. Xây dựng model để tính tổng hai tensor x, y
14. Xây dựng model để tính hiệu và tích hai tensor x, y
15. Xây dựng model để tính chu vi hình chữ nhật P = (a +b) * 2
16. Xây dựng model để tính diện tích toàn phần hình hộp chữ nhật

    $S_{xung\ quanh}$ = (a + b) * c * 2

    $S_{toàn\ phần}$ = (a * b) * 2 + $S_{xung\ quanh}$
17. Xây dựng model chứng minh hai cạnh bên hình thang song song theo điều kiện

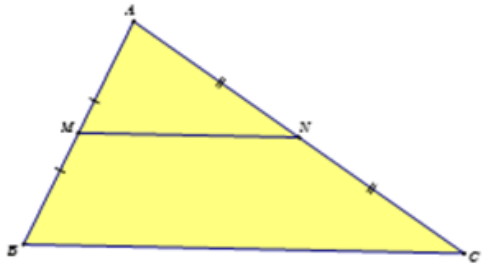
$$AD\ //\ BC \Leftrightarrow \begin{cases} AD = BC \\ AB = CD \end{cases}$$

18. Xây dựng model chứng minh hai cạnh đáy hình thang bằng nhau theo điều kiện

$$AB = CD \Leftrightarrow \begin{cases} AD = BC \\ AD\ //\ BC \end{cases}$$

19. Xây dựng model chứng minh NA = NC theo mô tả sau với AM, MB, AN, NC là đầu vào

+) **Đường trung bình của tam giác:** là đoạn thẳng nối trung điểm hai cạnh của tam giác.



- Tam giác ABC: $\begin{cases} AM = MB \\ AN = NC \end{cases}$ thì MN là đường trung bình của tam giác ABC

- MN là đường trung bình của tam giác ABC $\begin{cases} MN\ //\ BC \\ MN = \dfrac{BC}{2} \end{cases}$

- $\begin{cases} MN = MB \\ MN\ //\ BC \end{cases} \Rightarrow NA = NC$