Modul Praktikum

Kecerdasan Buatan

7

Daftar Isi

TensorFlow Basic

Introduction to TensorFlow

TensorFlow merupakan library open-source yang digunakan untuk Machine Learning dan Kecerdasan Buatan. TensorFlow bisa digunakan untuk berbagai macam kebutuhan tetapi TensorFlow lebih terfokus kepada training dan inference dalam neural network.

Di dalam TensorFlow, Tensor diklasifikasikan menjadi 2 yaitu Constant Tensor dan Variable Tensor

- **Constant tensor** yang telah ditentukan memiliki nilai dan dimensi yang tidak dapat diubah, dan **variabel tensor** yang telah didefinisikan memiliki nilai yang dapat diubah tetapi dimensinya tidak dapat diubah
- Dalam Neural Network, **variabel tensor** umumnya digunakan sebagai matriks untuk menyimpan bobot dan informasi lainnya, dan merupakan jenis data yang dapat dilatih. Sedangkan, **constant tensor** dapat digunakan untuk menyimpan hyperparameter atau data terstruktur lainnya.

How to Create a Tensor

1. **Constant Tensor**

   Methods yang biasa digunakan dalam pembuatan sebuah Constant Tensor:
   - `tf.constant()`: Untuk membuat constant tensor
   - `tf.zeros()`, `tf.zeros_likes()`, `tf.ones()`, dan `tf.ones_like()`: untuk membuat all-zero atau all-one constant tensor
   - `tf.fill()`: Membuat sebuah tensor dengan nilai user-defined
   - `tf.random`: membuat sebuah tensor dengan
   - Membuat sebuah list objek dengan library NumPy, kemudian mengonversi list objek ke dalam tensor menggunakan `tf.using_to_tensor`

a. tf.constant()

tf.constant(value, dtype=none, shape=None, name='Const'):

- value: Nilai constant atau list
- dtype: jenis elemen yang akan ada di dalam tensor
- shape: Dimensi tensor yang akan dihasilkan
- name: Penamaan untuk tensor

Code:

```
#Pemanggilan Library TensorFlow
import tensorflow as tf

#Penggunaan tf.constant()
const_a = tf.constant([[1, 2, 3, 4]],shape=[2,2], dtype=tf.float32)

#Pengecekan nilai constant
print(const_a)
```

Output:

```
tf.Tensor(
[[1. 2.]
 [3. 4.]], shape=(2, 2), dtype=float32)
```

b. tf.zeros()

tf.zeros(shape, dtype=tf.float32, name=None):

- shape: sebuah list dalam bentuk integer, sebuah list dalam bentuk integer, atau sebuah tensor 1 dimensi dengan tipe int32.t
- dtype: jenis elemen yang akan ada di dalam tensor
- name: penamaan untuk tensor

Code:

```
#Penggunaan tf.zeros()
zeros_b = tf.zeros(shape=[2, 3], dtype=tf.int32)

#Pengecekan nilai constant
print(zeros_b)
```

Output:

```
tf.Tensor(
[[0 0 0]
 [0 0 0]], shape=(2, 3), dtype=int32)
```

c. tf.fill()

tf.fill(dims, value, name=None):

- dims: Urutan angka non-negatif 1 dimensi. Merepresentasikan bentuk output dari tf.tensor. Masukan harus bertipe int32 atau int64
- value: sebuah nilai yang mengembalikan isi dari tf.tensor
- name: penamaan untuk tensor

Code:

```
#penggunaan tf.fill()
fill_d = tf.fill([3,3], 8)

#Pengecekan nilai constant
fill_d.numpy()
```

Output:

```
array([[8, 8, 8],
       [8, 8, 8],
       [8, 8, 8]])
```

d. tf.random()

tf.random.normal(shape, mean=0.0, stddev=1.0, dtype=tf.float32, seed=None, name=None)

- shape:
- mean:
- stddev:
- dtype:
- seed:
- name:

Code:

```
#penggunaan tf.random.normal()
random_e = tf.random.normal([5,5],mean=0,stddev=1.0, seed = 1)

#Pengecekan nilai constant
random_e.numpy()
```

Output:

```
array([[-0.8113182 ,  1.4845988 ,  0.06532937, -2.4427042 ,  0.0992484 ],
       [ 0.5912243 ,  0.59282297, -2.1229296 , -0.72289723, -0.05627038],
       [ 0.6435448 , -0.26432407,  1.8566332 ,  0.5678417 , -0.3828359 ],
       [-1.4853433 ,  1.2617711 , -0.02530608, -0.2646297 ,  1.5328138 ],
       [-1.7429771 , -0.43789294, -0.56601   ,  0.32066926,  1.132831  ]],
      dtype=float32)
```

e. tf.using_to_tensor

tf.convert_to_tensor(value, dtype=None, dtype_hint=None, name=None)

Code:

```
#Pembuatan list
list_f = [1,2,3,4,5,6]

#Penggunaan tf.convert_to_tensor
tensor_f = tf.convert_to_tensor(list_f, dtype=tf.float32)

#Pengecekan nilai constant
print(tensor_f)
```

Output:

```
tf.Tensor([1. 2. 3. 4. 5. 6.], shape=(6,), dtype=float32)
```

2. Variable Tensor

- Variabel dioperasikan dengan menggunakan class tf.Variable
- Nilai pada tf.Variable dapat dirubah dengan menjalakan operasi aritmatika pada tf.Variable

Code:

```
#Penggunaan tf.Variable
var_1 = tf.Variable(tf.ones([2,3]))

#Output dari tf.Variable
print(var_1)
```

Output:

```
<tf.Variable 'Variable:0' shape=(2, 3) dtype=float32, numpy=
array([[1., 1., 1.],
       [1., 1., 1.]], dtype=float32)>
```

Code:

```
#Read the variable value.
print("Value of the variable var_1:",var_1.read_value())

#Assign a variable value.
var_value_1=[[1,2,3],[4,5,6]]
var_1.assign(var_value_1)
print("Value of the variable var_1 after the assignment:",var_1.read_value())
```

Output:

```
Value of the variable var_1: tf.Tensor(
[[1. 1. 1.]
 [1. 1. 1.]], shape=(2, 3), dtype=float32)
Value of the variable var_1 after the assignment: tf.Tensor(
[[1. 2. 3.]
 [4. 5. 6.]], shape=(2, 3), dtype=float32)
```

Tensor Slicing and Indexing

1. **Slicing**

   Tensor slicing include:

   - [start:end]: Mengekstrak irisan data dari posisi start ke posisi end tensor
   - [start:end:step]: or [::step]: Mengekstrak irisan data pada interval dari posisi start ke posisi end tensor
   - [::-1]: Slices data dari elemen terakhir
   - '...': Mengindikasi slice data dengan panjang berapa pun

   Code:

```
print("Create a 4-dimensional tensor. The tensor contains four images. The size of each image is 100 x 100 x 3.")
tensor_h = tf.random.normal([4,100,100,3])
print(tensor_h)

print("\n Extract the first image.")
print(tensor_h[0,:,:,:])

print("\n Extract one slice at an interval of two images.")
print(tensor_h[::2,...])

print("\n Slice data from the last element.")
print(tensor_h[::-1])
```

Output:

```
Create a 4-dimensional tensor. The tensor contains four images. The size of
each image is 100 x 100 x 3.
tf.Tensor(
[[[[ 3.32316995e-01  4.14429307e-01 -2.05124187e+00]
   [-3.12114328e-01  7.69425333e-01 -1.40278757e+00]
   [ 2.28206053e-01  1.50574994e+00  1.21064000e-01]
   ...
   [-1.30165255e+00 -1.86255783e-01 -6.45645380e-01]
   [-3.47768635e-01 -2.51498222e+00  7.32671559e-01]
   [-1.55445015e+00  1.00313373e-01  1.18665881e-01]]

  [[ 1.45937145e-01  8.76392365e-01 -1.82573783e+00]
   [ 1.95287395e+00 -7.89778888e-01  1.88492239e+00]
   [-5.16299307e-01 -2.08932564e-01  1.04721367e+00]
   ...

Extract the first image.
tf.Tensor(
[[[-1.93250000e+00 -1.38260201e-01 -5.65310955e-01]
  [ 9.99143183e-01 -5.31990409e-01 -7.56538749e-01]
  [-1.64496124e+00  1.10704052e+00  1.23902750e+00]
  ...
  [-3.25841643e-02 -5.77418745e-01  1.25851738e+00]
  [-5.61836183e-01 -5.48085809e-01  1.17263925e+00]
  [ 1.52576625e-01  1.55385053e+00  1.32141292e+00]]

 [[-2.19975185e+00  4.89854999e-02  6.80724442e-01]
  [-1.00667369e+00 -8.89646530e-01  7.03344345e-01]

Extract one slice at an interval of two images.
tf.Tensor(
[[[[-1.93250000e+00 -1.38260201e-01 -5.65310955e-01]
   [ 9.99143183e-01 -5.31990409e-01 -7.56538749e-01]
   [-1.64496124e+00  1.10704052e+00  1.23902750e+00]
   ...
   [-3.25841643e-02 -5.77418745e-01  1.25851738e+00]
   [-5.61836183e-01 -5.48085809e-01  1.17263925e+00]
   [ 1.52576625e-01  1.55385053e+00  1.32141292e+00]]

  [[-2.19975185e+00  4.89854999e-02  6.80724442e-01]
   [-1.00667369e+00 -8.89646530e-01  7.03344345e-01]
   [ 2.11391234e+00  1.03057337e+00 -4.87599075e-01]
   ...
```

```
Slice data from the last element.
tf.Tensor(
[[[[ 5.06397665e-01  5.61371446e-01 -9.93135050e-02]
   [ 5.89880049e-01 -8.75897147e-03 -3.66167694e-01]
   [-1.52844393e+00 -1.38444200e-01 -1.83972156e+00]
   ...
   [ 9.03405905e-01  7.76836991e-01  1.02732360e+00]
   [ 6.32334054e-02  9.26207781e-01 -8.02184999e-01]
   [-5.81016123e-01  1.36589205e+00  3.27501059e-01]]

  [[ 4.65376258e-01  3.95817906e-01 -1.94681525e-01]
   [ 8.72957408e-01 -3.12797695e-01 -3.45325321e-01]
   [ 1.83032703e+00  4.35686707e-01  2.40513086e+00]
   ...
   [-6.72167897e-01  4.11164761e-01  1.14053562e-01]
```

2. Indexing

Format standar untuk indeing adalah tensor[d1][d2][d3]. Jika indeks data yang akan diekstrasi tidak berurutan maka tf.gather dan tf.gather_nd biasa digunakan untuk ekstraksi data di TensorFlow

- Untuk mengekstrak data dari dimensi tertentu menggunakan: Tf.gather(params, indices, axis=None):
  - params: Input tensor
  - indices: index dari data yang ingin di ekstrak
  - axis: dimensi dari data yang akan di ekstrak

  Code:

```
#Obtain the pixel in the position [20,40] in the second channel of the first image.
tensor_h[0][19][39][1]

#Extract the first, second, and fourth images from tensor_h ([4,100,100,3]).
indices = [0,1,3]
tf.gather(tensor_h,axis=0,indices=indices)
```

Output:

```
<tf.Tensor: shape=(3, 100, 100, 3), dtype=float32, numpy=
array([[[[-1.93250000e+00, -1.38260201e-01, -5.65310955e-01],
         [ 9.99143183e-01, -5.31990409e-01, -7.56538749e-01],
         [-1.64496124e+00,  1.10704052e+00,  1.23902750e+00],
         ...,
```

- tf.gather_nd bisa mengekstrak data dari multiple dimensi
  tf.gather_nd(params, indices, batch_dims=0, name=None):
  - params: Tensor yang digunakan untuk mengumpulkan nilai
  - indices: Tensor dengan type int32 atau int64
  - batch_dims: Sebuah integer atau scalar 'Tensor'. Jumlah dimensi batch

  Code:

```
#Obtain the pixel in the position [20,40] in the second channel of the first image.
tensor_h[0][19][39][1]

#Extract the pixel in [1,1] from the first dimension of the first image and the pixel in [2,2] from the first dimension of the second image in tensot_h ([4,100,100,3]).
indices = [[0,1,1,0],[1,2,2,0]]
tf.gather_nd(tensor_h,indices=indices)
```

Output:

```
<tf.Tensor: shape=(2,), dtype=float32, numpy=array([-1.0066737 ,  0.03599121],
dtype=float32)>
```

Tensor Dimension Modification

1. **Dimension Display**

   Untuk menampilkan dimensi dari Tensor kita menggunakan .shape dan .get_shape() mengembalikan TensorShape dan tf.shape(x) mengembalikan Tensor object.

   Code:

```
const_d_1 = tf.constant([[1, 2, 3, 4]],shape=[2,2], dtype=tf.float32)
#Three common methods for displaying a dimension:
print(const_d_1.shape)
print(const_d_1.get_shape())
print(tf.shape(const_d_1))#The output is a tensor. The value of the tensor indicates the size of the tensor dimension to be displayed.
```

   Output:

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

2. **Dimension Reshaping**

   tf.reshape(tensor,shape,name=None):

   - tensor: Input tensor
   - shape: Dimensi Tensor yang di reshaped

   Code:

```
reshape_1 = tf.constant([[1,2,3],[4,5,6]])
print(reshape_1)
tf.reshape(reshape_1, (3,2))
```

   Output:

```
tf.Tensor(
[[1 2 3]
 [4 5 6]], shape=(2, 3), dtype=int32)

<tf.Tensor: shape=(3, 2), dtype=int32, numpy=
array([[1, 2],
       [3, 4],
       [5, 6]])>
```

3. Dimension Expansion

tf.expand_dims(input,axis,name=None):

- Input: Input tensor
- axis: menambahkan dimensi setelah axis

Code:

```
#Generate a 100 x 100 x 3 tensor to represent a 100 x 100 three-channel color image.
expand_sample_1 = tf.random.normal([100,100,3], seed=1)
print("size of the original data:",expand_sample_1.shape)
print("add a dimension before the first dimension (axis = 0):
",tf.expand_dims(expand_sample_1, axis=0).shape)
print("add a dimension before the second dimension (axis = 1):
",tf.expand_dims(expand_sample_1, axis=1).shape)
print("add a dimension after the last dimension (axis = −1):
",tf.expand_dims(expand_sample_1, axis=-1).shape)
```

Output:

```
size of the original data: (100, 100, 3)
add a dimension before the first dimension (axis = 0):  (1, 100, 100, 3)
add a dimension before the second dimension (axis = 1):  (100, 1, 100, 3)
add a dimension after the last dimension (axis = −1):  (100, 100, 3, 1)
```

4. Dimension Squeezing

Tf.squeeze(input,axis=None,name=None):

- input: Input Tensor
- axis: jika axis di set menjadi 1, maka dimensi 1 harus di hapus

Code:

```
#Generate a 100 x 100 x 3 tensor to represent a 100 x 100 three-channel color image.
squeeze_sample_1 = tf.random.normal([1,100,100,3])
print("size of the original data:",squeeze_sample_1.shape)
squeezed_sample_1 = tf.squeeze(expand_sample_1)
print("data size after dimension squeezing:",squeezed_sample_1.shape)
```

Output:

```
size of the original data: (1, 100, 100, 3)
data size after dimension squeezing: (100, 100, 3)
```

5. **Transpose**

Tf.transpose(a,perm=None,conjugate=False,name='transpose'):

- a: Input tensor
- perm: Biasanya digunakan untuk mengtranspose array high-dimensional
- conjugate: menunjukkan transpose bilangan kompleks
- name: tensor name

Code:

```
#Input the tensor to be transposed, and call tf.transpose.
trans_sample_1 = tf.constant([1,2,3,4,5,6],shape=[2,3])
print("size of the original data:",trans_sample_1.shape)
transposed_sample_1 = tf.transpose(trans_sample_1)
print("size of transposed data:",transposed_sample_1.shape)
```

Output:

```
size of the original data: (2, 3)
size of transposed data: (3, 2)
```

6. **Broadcast**

broadcast_to digunakan untuk mengbroadcast data dari low-dimension menjadi high-dimension.

tf.broadcast_to(input,shape,name=None):

- input: Input Tensor
- shape: ukuran tensor

Code:

```
broadcast_sample_1 = tf.constant([1,2,3,4,5,6])
print("original data:",broadcast_sample_1.numpy())
broadcasted_sample_1 = tf.broadcast_to(broadcast_sample_1,shape=[4,6])
print("broadcasted data:",broadcasted_sample_1.numpy())
```

Output:

```
original data: [1 2 3 4 5 6]
broadcasted data: [[1 2 3 4 5 6]
 [1 2 3 4 5 6]
 [1 2 3 4 5 6]
 [1 2 3 4 5 6]]
```

Arithmetic Operations on Tensors

1. **Arithmetic Operators**

   Operasi aritmetika utama dalam tensor antara lain:

   - tf.add() → penjumlahan
   - tf.subtraction() → pengurangan
   - tf.multiply() → perkalian
   - tf.divide() → pembagian
   - tf.math.log() → logaritma
   - tf.pow() → pangkat

   Code:

```
a = tf.constant([[3, 5], [4, 8]])
b = tf.constant([[1, 6], [2, 9]])
print(tf.add(a, b))
```

   Output:

```
tf.Tensor(
[[ 4 11]
 [ 6 17]], shape=(2, 2), dtype=int32)
```

2. **Matrix Multiplication**

   Matrix Multiplication diterapkan dengan memanggil tf.matmul()

   Code:

```
tf.matmul(a,b)
```

   Output:

```
<tf.Tensor: shape=(2, 2), dtype=int32, numpy=
array([[13, 63],
       [20, 96]])>
```

3. Tensor Statistics Collection

Metode untuk mengambil statistik dari tensor menggunakan tf.argmax()/tf.argmin() digunakan untuk mengalkulasi posisi nilai maksimum atau minimum

tf.argmax(input,axis):

- input: Input tensor
- axis: nilai maksimal output di dalam dimensi

Code:

```
argmax_sample_1 = tf.constant([[1,3,2],[2,5,8],[7,5,9]])
print("input tensor:",argmax_sample_1.numpy())
max_sample_1 = tf.argmax(argmax_sample_1, axis=0)
max_sample_2 = tf.argmax(argmax_sample_1, axis=1)
print("locate the maximum value by column:",max_sample_1.numpy())
print("locate the maximum value by row:",max_sample_2.numpy())
```

Output:

```
input tensor: [[1 3 2]
 [2 5 8]
 [7 5 9]]
locate the maximum value by column: [2 1 2]
locate the maximum value by row: [1 2 2]
```

Dimension-based Arithmetic Operations

Pada TensorFlow, rangkaian operasi dari tf.reduce_* digunakan untuk mengurangi dimensi dari tensor. Serangkaian operasi bisa dijalakan pada elemen dimensi pada tensor, misalnya kita menghitung nilai mean per baris dan menghitung product dari semua elemen di dalam tensor

tf.reduce_sum(input_tensor,axis=None,keepdims=False,name=None):

- input_tensor: Tensor yang akan dikurangi. Harus memiliki tipe numeric
- axis: Dimensi yang akan dikurangi
- keepdims: jika bernilai True, mempertahankan dimensi yang dikurangi dengan panjang 1

Code:

```
reduce_sample_1 = tf.constant([1,2,3,4,5,6],shape=[2,3])
print("original data",reduce_sample_1.numpy())
print("calculate the sum of all elements in the tensor (axis = None):
",tf.reduce_sum(reduce_sample_1,axis=None).numpy())
print("calculate the sum of elements in each column by column (axis = 0):
",tf.reduce_sum(reduce_sample_1,axis=0).numpy())
print("calculate the sum of elements in each column by row (axis = 1):
",tf.reduce_sum(reduce_sample_1,axis=1).numpy())
```

Output:

```
original data [[1 2 3]
 [4 5 6]]
calculate the sum of all elements in the tensor (axis = None):  21
calculate the sum of elements in each column by column (axis = 0):  [5 7 9]
calculate the sum of elements in each column by row (axis = 1):  [ 6 15]
```

Tensor Concatenation and Splitting

1. Tensor Concatenation

tf.contact(): Menggabungkan vektor berdasarkan dimensi yang telah ditentukan, sembari menjaga dimensi lain tidak berubah

tf.contact(values,axis,name='concat'):

- values: Input tensor
- axis: Dimensi yang akan digabungkan

Code:

```
concat_sample_1 = tf.random.normal([4,100,100,3])
concat_sample_2 = tf.random.normal([40,100,100,3])
print("sizes of the original data:",concat_sample_1.shape,concat_sample_2.shape)
concated_sample_1 = tf.concat([concat_sample_1,concat_sample_2],axis=0)
print("size of the concatenated data:",concated_sample_1.shape)
```

Output:

```
sizes of the original data: (4, 100, 100, 3) (40, 100, 100, 3)
size of the concatenated data: (44, 100, 100, 3)
```

tf.stack (): mengubah bagian tensor dimensi R menjadi tensor dimensi R+1, yang perubahannya terjadi setelah penggabungan

tf.stack(values,axis=0,name'stack'):

- values: sebuah list dari tensor dengan shape dan type yang sama
- axis: sebuah INT. Axis yang akan di tumpuk. Default mengarah kepada dimensi pertama

Code:

```
stack_sample_1 = tf.random.normal([100,100,3])
stack_sample_2 = tf.random.normal([100,100,3])
print("sizes of the original data: ",stack_sample_1.shape, stack_sample_2.shape)
#Dimensions increase after the concatenation. If axis is set to 0, a dimension is added
before the first dimension.
stacked_sample_1 = tf.stack([stack_sample_1, stack_sample_2],axis=0)
print("size of the concatenated data:",stacked_sample_1.shape)
```

Output:

```
sizes of the original data:  (100, 100, 3) (100, 100, 3)
size of the concatenated data: (2, 100, 100, 3)
```

2. Tensor Splitting

tf.unstack(): Membagi sebuah tensor dengan dimensi tertentu

`tf.unstack(value,num=None,axis=0,name='unstack')`:

- value: Input tensor
- num: Menunjukkan bahwa list yang berisi elemen num adalah output. Nilai num harus sama dengan jumlah elemen dalam dimensi yang ditentukan. Parameter ini umunya dapat diabaikan
- axis: Menentukan dimensi berdasarkan tensor yang mana yang di split

Code:

```
#Split data based on the first dimension and output the split data in a list.
tf.unstack(stacked_sample_1,axis=0)
```

Output:

```
[<tf.Tensor: shape=(100, 100, 3), dtype=float32, numpy=
 array([[[-2.45739743e-01, -2.88914959e-03,  2.62151212e-01],
         [-8.01494956e-01, -5.14896214e-02,  8.03997815e-01],
         [-6.37233317e-01, -3.52935433e-01,  1.75227964e+00],
         ...,
```

tf.split(): membagi tensor menjadi beberapa sub tensor tententu berdasarkan dimensi tertentu

`tf.split(value, num_or_size_splits, axis=0,num=None, name'split')`:

- value: nilai tensor yang akan di split
- num_or_size_splits:

Code:

```
import numpy as np
split_sample_1 = tf.random.normal([10,100,100,3])
print("size of the original data:",split_sample_1.shape)
splited_sample_1 = tf.split(split_sample_1, num_or_size_splits=5,axis=0)
print("size of the split data when m_or_size_splits is set to 10:
",np.shape(splited_sample_1))
splited_sample_2 = tf.split(split_sample_1, num_or_size_splits=[3,5,2],axis=0)
print("sizes of the split data when num_or_size_splits is set to [3,5,2]:",
    np.shape(splited_sample_2[0]),
    np.shape(splited_sample_2[1]),
    np.shape(splited_sample_2[2]))
```

Output:

```
size of the original data: (10, 100, 100, 3)
size of the split data when m_or_size_splits is set to 10:  (5, 2, 100, 100,
3)
sizes of the split data when num_or_size_splits is set to [3,5,2]: (3, 100, 1
00, 3) (5, 100, 100, 3) (2, 100, 100, 3)
```

Tensor sorting methods

1. tf.sort() & tf.argsort() memiliki tujuan yang sama yaitu mengurutkan tensor baik itu ascending atau descending tetapi tf.sort() mengembalikan tensor yang telah di-sorting sedangkan tf.argsort() Mengembalikan index dari tensornya

   tf.sort/argsort(input,direction,axis):

   - input: Input Tensor
   - direction: perintah sorting. ASCENDING atau DESCENDING
   - axis: menentukan dimensi yang akan di-sorting. Default dari axis adalah -1, menunjukkan dimensi terakhir

   Code:

```
sort_sample_1 = tf.random.shuffle(tf.range(10))
print("input tensor:",sort_sample_1.numpy())
sorted_sample_1 = tf.sort(sort_sample_1, direction="ASCENDING")
print("tensor sorted in ascending order:",sorted_sample_1.numpy())
sorted_sample_2 = tf.argsort(sort_sample_1,direction="ASCENDING")
print("indexes of elements in ascending order:",sorted_sample_2.numpy())
```

   Output:

```
input tensor: [3 5 7 1 4 8 6 0 2 9]
tensor sorted in ascending order: [0 1 2 3 4 5 6 7 8 9]
indexes of elements in ascending order: [7 3 8 0 4 1 6 2 5 9]
```

2. tf.nn.tp_k() mengembalikan nilai maksimum k pertama

   tf.nn.tp_k(input, k=1, sorted=True):

   - input: Tensor 1-D atau lebih tinggi dengan dimensi terakhir setidaknya K
   - k: Tensor 0-D int32.
   - sorted: jika True menghasilkan elemen K yang di-sorting berdasarkan descending order

   Code:

```
values, index = tf.nn.top_k(sort_sample_1,5)
print("input tensor:",sort_sample_1.numpy())
print("first five values in ascending order:", values.numpy())
print("indexes of the first five values in ascending order:", index.numpy())
```

Output:

```
input tensor: [3 5 7 1 4 8 6 0 2 9]
first five values in ascending order: [9 8 7 6 5]
indexes of the first five values in ascending order: [9 5 2 6 1]
```

Eager Execution of TensorFlow 2.x

- Eager execution mode: Eager execution mode dari TensorFlow adalah jenis pemrograman imperatif, yang sama saja dengan native python. Saat melakukan operasi tertentu, sistem akan segera mengembalikan hasilnya
- Pada Eager execution mode, debugging kode menjadi lebih mudah, tetapi efisiensi eksekusi kode menjadi lebih rendah
- Contoh pengimplementasian perkalian sederhana dengan menggunakan TensorFlow untuk membandingkan antara eager exectuion mode dengan mode lain yaitu graph mode
  Code:

```
x = tf.ones((2, 2), dtype=tf.dtypes.float32)
y = tf.constant([[1, 2],
            [3, 4]], dtype=tf.dtypes.float32)
z = tf.matmul(x, y)
print(z)
```

Output:

```
tf.Tensor(
[[4. 6.]
 [4. 6.]], shape=(2, 2), dtype=float32)
```

Code:

```
#Use the syntax of TensorFlow 1.x in TensorFlow 2.x. You can install the v1 compatibility package in TensorFlow 2.0 to inherit the TensorFlow 1.x code and disable the eager execution mode.
import tensorflow.compat.v1 as tf
tf.disable_eager_execution()
#Create a graph and define it as a computational graph.
a = tf.ones((2, 2), dtype=tf.dtypes.float32)
b = tf.constant([[1, 2],
            [3, 4]], dtype=tf.dtypes.float32)
c = tf.matmul(a, b)
#Enable the drawing function, and perform the multiplication operation to obtain data.
with tf.Session() as sess:
   print(sess.run(c))
```

Output:

```
[[4. 6.]
 [4. 6.]]
```

Retart ulang kernel untuk mengembalikan TensorFlow 2.0 dan mengaktifkan kembali eager execution mode. Keuntungan lain dari eager execution mode terletak pada ketersediaan fungsi native Python, misal seperti kondisi berikut:
Code:

```
import tensorflow as tf
thre_1 = tf.random.uniform([], 0, 1)
x = tf.reshape(tf.range(0, 4), [2, 2])
print(thre_1)
if thre_1.numpy() > 0.5:
   y = tf.matmul(x, x)
else:
  y = tf.add(x, x)
```

Output:

```
tf.Tensor(0.36578333, shape=(), dtype=float32)
```

Dengan eager exectuion mode, dynamic control flow dapat menghasilkan nilai NumPy yang dapat diekstrak oleh tensor, tanpa menggunakan operator seperti tf.cond dan tf.while yang ada pada graph mode

AutoGraph of TensorFlow 2.x

Saat digunakan untuk mengomentari suatu fungsi, dekorator tf.function dapat dipanggil seperti fungsi lainnya. tf.function akan di-compiled menjadi graph, sehingga dapat berjalan lebih efisien pada GPU atau TPU. Dalam hal ini, fungsi tersebut menjadi operasi di TensorFlow. Fungsi dapat langsung dipanggil untuk menampilkan return value. Namun, fungsi dijalankan dalam mode graph dan nilai variabel perantara tidak dapat dilihat secara langsung.

Code:

```
@tf.function
def simple_nn_layer(w,x,b):
   print(b)
   return tf.nn.relu(tf.matmul(w, x)+b)

w = tf.random.uniform((3, 3))
x = tf.random.uniform((3, 3))
b = tf.constant(0.5, dtype='float32')
```

```
simple_nn_layer(w,x,b)
```

Output:

```
Tensor("b:0", shape=(), dtype=float32)

<tf.Tensor: shape=(3, 3), dtype=float32, numpy=
array([[1.2373953, 2.5350056, 2.0113873],
       [0.8762206, 1.4743018, 1.3564188],
       [0.9268935, 1.6092176, 1.4604295]], dtype=float32)>
```

Berdasarkan hasil output, nilai b dalam fungsi tidak dapat dilihat secara langsung, tetapi return value dapat dilihat menggunakan .numpy()

Berikut ini membandingkan kinerja mode graph dan mode eager execution dengan melakukan operasi yang sama (perhitungan satu lapisan CNN).

Code:

```
#Use the timeit module to measure the execution time of a small code segment.
import timeit
#Create a convolutional layer.
CNN_cell = tf.keras.layers.Conv2D(filters=100,kernel_size=2,strides=(1,1))

#Use @tf.function to convert the operation into a graph.
@tf.function
def CNN_fn(image):
   return CNN_cell(image)

image = tf.zeros([100, 200, 200, 3])

#Compare the execution time of the two modes.
CNN_cell(image)
CNN_fn(image)
#Call timeit.timeit to measure the time required for executing the code 10 times.
print("time required for performing the computation of one convolutional neural network (CNN)
layer in eager execution mode:", timeit.timeit(lambda: CNN_cell(image), number=10))
print("time required for performing the computation of one CNN layer in graph mode:",
timeit.timeit(lambda: CNN_fn(image), number=10))
```

Output:

```
time required for performing the computation of one convolutional neural networ
k (CNN) layer in eager execution mode: 4.884449000004679
time required for performing the computation of one CNN layer in graph mode: 2.
421844000054989
```

Perbandingan menunjukkan bahwa efisiensi eksekusi kode dalam mode graph jauh lebih tinggi. Oleh karena itu, fungsi @tf.function dapat digunakan untuk meningkatkan efisiensi eksekusi kode.