

آشنایی با TensorFlow

1. نصب Tensorflow

برای نصب پکیج مورد نظر، از روش اول، یعنی روش نصب به کمک `pip` استفاده شد. به این صورت که در ترمینال، ابتدا از آپدیت بودن `pip` اطمینان حاصل می‌کنیم. و سپس با استفاده از دستور `pip install tensorflow`، پکیج را نصب می‌کنیم. ابزارهای مورد نیاز را نصب می‌کنیم:

```
pip install numpy scipy
pip install scikit-learn
pip install pillow
pip install h5py
```

و در نهایت به کمک `pip`، کراس را نصب می‌کنیم:

```
pip install keras
```

2. اجرای یک برنامه ساده به کمک Tensorflow

پس از نصب محیط مجازی، به اجرای یک برنامه ساده برای درک بیشتر مفاهیم می‌پردازیم. برای شروع سراغ کتابخانه `numpy` رفتیم. تصاویر کدها و خروجی‌ها را در ادامه مشاهده می‌کنید:

```
EXPLORER 6-2_Numpy_2D.ipynb X
6-2_Numpy_2D.ipynb

[4] In: import numpy as np
import matplotlib.pyplot as plt

Consider the list a , the list contains three nested lists each of equal size.

[5] In: a=[[11,12,13],[21,22,23],[31,32,33]]
a
Out: [[11, 12, 13], [21, 22, 23], [31, 32, 33]]

We can cast the list to a numpy array as follow

[6] In: #every element if of the same type
A=np.array(a)
A
Out: array([[11, 12, 13],
[21, 22, 23],
[31, 32, 33]])

We can use the attribute ndim to obtain the number of axes or dimensions referred to as the rank.

[7] In: A.ndim
Out: 2
```

```
EXPLORER 6-2_Numpy_2D.ipynb X
6-2_Numpy_2D.ipynb
1.JPG
6-2_Numpy_2D.ip...

[8] In: A.shape
Out: (3, 3)

The total number of elements in the array is given by the attribute size.

[9] In: A.size
Out: 9

Accessing different elements of an Numpy Array

We can use rectangular brackets to access the different elements of the array,The correspondence between the rectangular brackets and the list and the rectangular representation is shown in the following figure for a 3x3 array:

A: [[A[0,0], A[0,1], A[0,2]], [A[1,0], A[1,1], A[1,2]], [A[2,0], A[2,1], A[2,2]]]

A[0,0] A[0,1] A[0,2]
A[1,0] A[1,1] A[1,2]
A[2,0] A[2,1] A[2,2]

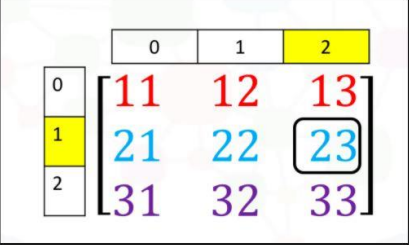
We can access the 2nd row 2nd column as shown in the following figure:
```

EXPLORER ... 6-2_Numpy_2D.ipynb X

6-2_Numpy_2D.ipynb

1.JPG
2.JPG
6-2_Numpy_2D.ip...

We can access the 2nd-row 3rd column as shown in the following figure:



We simply use the square brackets and the indices corresponding to the element we would like:

```
[10] In: A[1,2]  
Out: 23
```

We can also use the following notation to obtain the elements:

```
[11] In: A[1][2]  
Out: 23
```

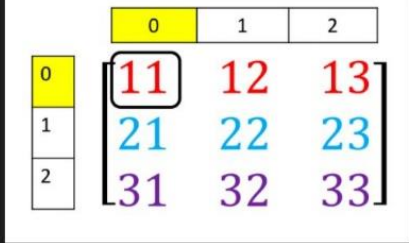
> OUTLINE

EXPLORER ... 6-2_Numpy_2D.ipynb X

6-2_Numpy_2D.ipynb

1.JPG
2.JPG
3.JPG
6-2_Numpy_2D.ip...

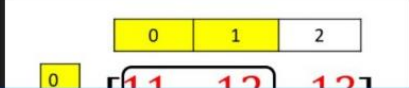
Consider the elements shown in the following figure



we can access the element as follows

```
[12] In: A[0][0]  
Out: 11
```

We can also use slicing in numpy arrays, consider the following figure; we would like to obtain the first two columns in the first row



> OUTLINE

EXPLORER

1.JPG

2.JPG

3.JPG

4.JPG

6-2_Numpy_2D.ip...

6-2_Numpy_2D.ipynb

6-2_Numpy_2D.ipynb

Trust

Jupyter Server: local

Python 3.9.4 64-bit: idle

	0	1	2
0	11	12	13
1	21	22	23
2	31	32	33

This can be done with the following syntax

```
[13] In [13]: A[0][0:2]
Out[13]: array([11, 12])
```

Similarly, we can obtain the first two rows of the 3rd column as follows:

```
[14] In [14]: A[1:3,2]
Out[14]: array([23, 33])
```

Corresponding to the following figure:

EXPLORER

1.JPG

2.JPG

3.JPG

4.JPG

5.JPG

6-2_Numpy_2D.ip...

6-2_Numpy_2D.ipynb

6-2_Numpy_2D.ipynb

Trust

Jupyter Server: local

Python 3.9.4 64-bit: idle

	0	1	2
0	11	12	13
1	21	22	23
2	31	32	33

Basic Operations

We can also add arrays; the process is identical to matrix addition. Matrix addition of X and Y is shown in the following figure:

$$X = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \quad Y = \begin{bmatrix} 2 & 1 \\ 1 & 2 \end{bmatrix}$$

$$X + Y = \begin{bmatrix} 1+2 & 0+1 \\ 0+1 & 1+2 \end{bmatrix} = \begin{bmatrix} 3 & 1 \\ 1 & 3 \end{bmatrix}$$

The numpy array is given by X and Y

EXPLORER

6-2_Numpy_2D.ipynb

1.JPG
2.JPG
3.JPG
4.JPG
5.JPG
6-2_Numpy_2D.ip...
6.JPG

6-2_Numpy_2D.ipynb

TrustedsJupyter Server: localPython 3.9.4 64-bit: Idle

[15] ▶ +≡ M6

X=np.array([[1,0],[0,1]])
X

array([[1, 0],
[0, 1]])

[16] ▶ +≡ M6

Y=np.array([[2,1],[1,2]])
Y

array([[2, 1],
[1, 2]])

We can add the numpy arrays as follows.

[17] ▶ +≡ M6

Z=X+Y
Z

array([[3, 1],
[1, 3]])

Multiplying a numpy array by a scalar is identical to multiplying a matrix by a scalar. If we multiply the matrix Y by the scalar 2, we simply multiply every element in the matrix by 2 as shown in the figure.

EXPLORER

6-2_Numpy_2D.ipynb

1.JPG
2.JPG
3.JPG
4.JPG
5.JPG
6-2_Numpy_2D.ip...
6.JPG
7.JPG

6-2_Numpy_2D.ipynb

TrustedsJupyter Server: localPython 3.9.4 64-bit: Idle

[18] ▶ +≡ M6

Y=np.array([[2,1],[1,2]])
Y

array([[2, 1],
[1, 2]])

[19] ▶ +≡ M6

Z=2*Y
Z

array([[4, 2],
[2, 4]])

Multiplication of two arrays corresponds to an element-wise product or Hadamard product. Consider matrix X and Y. The Hadamard product corresponds to multiplying each of the elements in the same position, i.e. multiplying elements contained in the same colour boxes together. The result is a new matrix that is the same size as matrix Y or X, as shown in the following figure.

$$X = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \quad Y = \begin{bmatrix} 2 & 1 \\ 1 & 2 \end{bmatrix}$$

$$X \circ Y = \begin{bmatrix} (1)2 & (0)1 \\ (0)1 & (1)2 \end{bmatrix} = \begin{bmatrix} 2 & 0 \\ 0 & 2 \end{bmatrix}$$

We can perform element-wise product of the array X and Y as follows:

```
[20] In: Y=np.array([[2,1],[1,2]])
      Y
      array([[2, 1],
             [1, 2]])
```

```
[21] In: X=np.array([[1,0],[0,1]])
      X
      array([[1, 0],
             [0, 1]])
```

```
[22] In: Z=X*Y
      Z
      array([[2, 0],
             [0, 2]])
```

We can also perform matrix multiplication with the numpy arrays A and B as follows:

First, we define matrix A and B:

```
[23] In: A=np.array([[0,1,1],[1,0,1]])
      A
      array([[0, 1, 1],
             [1, 0, 1]])
```

```
[24] In: B=np.array([[1,1],[1,1],[-1,1]])
      B
      array([[ 1,  1],
             [ 1,  1],
             [-1,  1]])
```

We use the numpy function dot to multiply the arrays together.

```
[25] In: Z=np.dot(A,B)
      Z
      array([[0, 2],
             [0, 2]])
```

```
[26] In: np.sin(Z)
      array([[0.          , 0.98929743],
             [0.          , 0.98929743]])
```

```
[27] In: C=np.array([[1,1],[2,2],[3,3]])
```

EXPLORER

1.JPG

2.JPG

3.JPG

4.JPG

5.JPG

6-2_Numpy_2D.ipynb

6.JPG

7.JPG

8.JPG

9.JPG

10.JPG

6-2_Numpy_2D.ipynb

6-2_Numpy_2D.ipynb

TrusteJupyter Server: localPython 3.9.4 64-bit: idle

[26]

np.sin(Z)

array([[0. , 0.90929743],
 [0. , 0.90929743]])

[27]

C=np.array([[1,1],[2,2],[3,3]])
C

array([[1, 1],
 [2, 2],
 [3, 3]])

[28]

C.T

array([[1, 2, 3],
 [1, 2, 3]])

About the Authors:

Joseph Santacangelo has a PhD in Electrical Engineering, his research focused on using machine learning, signal processing, and computer vision to determine how videos impact human cognition. Joseph has been working for IBM since he completed his PhD.

()

Copyright © 2017 [cognitiveclass.ai](#). This notebook and its source code are released under the terms of the [MIT License](#).

OUTLINE