Львівський національний університет імені Івана Франка

Факультет електроніки та комп'ютерних технологій

ЗВІТ

Про виконання лабораторної роботи №2

Векторизація даних

Виконав:

Студент групи ФЕП-31

Сворень Ярослав

Перевірив:

Ас. Рибак А. В.

Львів 2022

Мета роботи: Дослідити базові методи бібліотеки NumPy та провести аналіз особливостей бібліотеки NumPy на основі реалізованих методів в порівнянні Python vs Python + NumPy.

Хід Роботи:

1. **П**ерш за все, створюємо нашу програму та підключаємо до неї бібліотеки **numpy**, **random**, **time**. Для зручності описуємо змінні **N**, **M**, **numafterdecimal**, які відповідають за кількість рядків у векторі **N**, **M**, та до скількох чисел після коми варто округляти результати обчислень. Далі генеруємо наші масиви, елементи яких генерується через **random()** на проміжку значень **[0,1)**, після чого за допомогою функції **numpy.array()** створюємо дублікат нашого масиву у форматі **numpy**, щоб перевіряти правильність значень наших майбутніх функцій з вбудованими функціями **numpy**.

Lab2ZMN.py > ...

```
import numpy as np
import random as rd
import time

N = 250
M = 250
numafterdecimal = 3
```

```
Npy_array = []
Mpy_array = []

for i in range(0,N):
    Npy_array.append([round(rd.random(),numafterdecimal)])
print("Npy_array("+str(N)+"x1):\n",Npy_array,"\n")
temp = []
for i in range(0,M):
    temp.append(round(rd.random(),numafterdecimal))
Mpy_array.append(temp)
print("Mpy_array(1x"+str(M)+"):\n",Mpy_array,"\n")
############################################################
Nnum_array = np.array(Npy_array)
Mnum_array = np.array(Mpy_array)
print("Nnum_array("+str(N)+"x1):\n",Nnum_array,"\n")
print("Mnum_array(1x"+str(M)+"):\n",Mnum_array,"\n")
```

2. Складаємо зовнішню функцію **vector_mult()**, яка приймає параметрами два двовимірні масиви, кількість рядків першого масиву, та кількості рядків та стовпців другого. Перевіряємо її працездатність, передавши їй масиви та розміри **Npy**, **Mpy**, після чого виконуємо ту саму дію вже з numpy-масивами за допомогою функції **numpy.matmul()**, після чого виводимо обидва масиви у термінал.

```
9   def vector_mult(matrix_A,matrix_B,A_rows,B_rows,B_columns):
10      matrix_C = []
11      for i in range(0,A_rows):
12          temp = []
13          for j in range(0,B_columns):
14              for k in range(0,B_rows):
15                  temp.append(round(matrix_A[i][k]*matrix_B[k][j],numafterdecimal))
16          matrix_C.append(temp)
17      return matrix_C
```

```
48  start = time.time()
49  N_mult_M = vector_mult(Npy_array,Mpy_array,N,1,M)
50  end = time.time()
51  time_result11 = round(end-start,numafterdecimal)
52  print("PY N*M:\n",N_mult_M,"\n","time:",time_result11,"\n")
53  start = time.time()
54  NmultM = np.matmul(Nnum_array,Mnum_array)
55  end = time.time()
56  time_result12 = round(end-start,numafterdecimal)
57  print("NumPY N*M:\n",NmultM,"\n","time:",time_result12,"\n")
```

3. Далі описуємо зовнішню функцію **matrix_mult_vector()**, яка покриває решту прикладів, які ми маємо виконати. Трохи пізніше я продемонструю, що результати моїх функцій та вбудованих є однакові з невеликою погрішністю.

```
19  def matrix_mult_vector(matrix_A,matrix_B,A_rows,B_rows,B_columns):
20      matrix_C = []
21      for i in range(0,A_rows):
22          temp = []
23          for j in range(0,B_columns):
24              temp2 = 0
25              for k in range(0,B_rows):
26                  temp2+=round(matrix_A[i][k]*matrix_B[k][j],numafterdecimal)
27              temp.append(round(temp2,numafterdecimal))
28          matrix_C.append(temp)
29      return matrix_C
```

```
start = time.time()
N_mult_M_mult_N = matrix_mult_vector(N_mult_M,Npy_array,N,N,1)         # (Npy*Mpy)*Npy matrix
end = time.time()
time_result21 = round(end-start,numafterdecimal)
print("PY (N*M)*N:\n",N_mult_M_mult_N,"\n","time:",time_result21,"\n")
start = time.time()
NmultMmultN = np.matmul(NmultM,Nnum_array)
end = time.time()
time_result22 = round(end-start,numafterdecimal)
print("NumPY (N*M)*N:\n",NmultMmultN,"\n","time:",time_result22,"\n")
############################################################
start = time.time()
M_mult_N_mult_M = matrix_mult_vector(Mpy_array,N_mult_M,1,N,M)         # Mpy(Npy*Mpy) matrix
end = time.time()
time_result31 = round(end-start,numafterdecimal)
print("PY M*(N*M):\n",M_mult_N_mult_M,"\n","time:",time_result31,"\n")
start = time.time()
MmultNmultM = np.matmul(Mnum_array,NmultM)
end = time.time()
time_result32 = round(end-start,numafterdecimal)
print("NumPY M*(N*M):\n",MmultNmultM,"\n","time:",time_result32,"\n")
############################################################
start = time.time()
N_mult_M_mult_N_mult_M = matrix_mult_vector(N_mult_M,N_mult_M,N,N,M)   # (Npy*Mpy)*(Npy*Mpy)
end = time.time()
time_result41 = round(end-start,numafterdecimal)
print("PY (N*M)*(N*M):\n",N_mult_M_mult_N_mult_M,"\n","time:",time_result41,"\n")
start = time.time()
NmultMmultNmultM = np.matmul(NmultM,NmultM)
end = time.time()
time_result42 = round(end-start,numafterdecimal)
print("NumPY (N*M)*(N*M):\n",NmultMmultNmultM,"\n","time:",time_result42,"\n")
```

4. **П**ісля цього ми маємо написати unit-тести для наших функцій. Я намагався написати тести, які будуть порівнювати чи однакові результати у моїх функцій та функції **numpy.matmul()**, проте через невелику погрішність іноді таки програма не пропускала їх, що може трапитись, коли у тебе усі елементи перемножуються на проміжку **[0,1]**. Натомість через вбудовану команду **assert** проводимо перевірку на вірність, передавши цілі числа у наші функції, виводимо до того у терміналі правильну відповідь, обчислену через **numpy.matmul()**.

```
unit_test_1_array1 = np.array([[1],[2],[3]])
unit_test_1_array2 = np.array([[1,2,3]])
print("Unit Test 1 The answer of NumPY:\n",np.matmul(unit_test_1_array1,unit_test_1_array2))
assert vector_mult([[1],[2],[3]],[[1,2,3]],3,1,3) == [[1,2,3],[2,4,6],[3,6,9]]
print("Unit Test 1 for vector_mult() Passed")
############################################################
unit_test_2_array1 = np.array([[1,2,3],[2,4,6],[3,6,9]])
unit_test_2_array2 = np.array([[1],[2],[3]])
print("Unit Test 2 The answer of NumPY:\n",np.matmul(unit_test_2_array1,unit_test_2_array2))
assert matrix_mult_vector([[1,2,3],[2,4,6],[3,6,9]],[[1],[2],[3]],3,3,1) == [[14],[28],[42]]
print("Unit Test 2 for matrix_mult_vector() Passed")
```

5. **В**решті-решт, нам потрібно було записати тривалість виконання наших функцій на функції **numpy.matmul()** для кожного прикладу, щоб ми могли порівняти їх у терміналі. Ви могли помітити, що функції початку запису і кінця запису стоять на скріншотах вище перед кожним викликом цих функцій.

```
104   print("PY N*M time: ",time_result11, " NumPy N*M time: ",time_result12)
105   print("PY (N*M)*N time: ",time_result21, " NumPy (N*M)*N time: ",time_result22)
106   print("PY M*(N*M) time: ",time_result31, " NumPy M*(N*M) time: ",time_result32)
107   print("PY (N*M)*(N*M) time: ",time_result41, " NumPy (N*M)*(N*M) time: ",time_result42)
```

6. **Т**епер коротко розберемо вивід. Вивід починається з того, що виводяться випадкові значення наших елементів масивів **Npy**, **Mpy**. Потім виводяться значення їх numpy-аналогів, щоб переконатися, що функція **numpy.array()** правильно конвертувала значення масивів.

```
PS D:\Programs\ZMN\Lab2ZMN> python -u "d:\Programs\ZMN\Lab2ZMN\Lab2ZMN.py"
Npy_array(250x1):
 [[0.169], [0.45], [0.774], [0.397], [0.469], [0.259], [0.317], [0.992], [
```

```
Nnum_array(250x1):
 [[0.169]
 [0.45 ]
 [0.774]
 [0.397]
 [0.469]
 [0.259]
 [0.317]
 [0.992]
```

```
Mpy_array(1x250):
 [[0.508, 0.673, 0.277, 0.904, 0.14, 0.268, 0.576, 0.312, 0.117, 0.936, 0.997, 0.0, 0.911,
```

```
Mnum_array(1x250):
 [[0.508 0.673 0.277 0.904 0.14  0.268 0.576 0.312 0.117 0.936 0.997 0.
```

Далі виводяться почергово результати наших прикладів, обчислених двома способами.

```
PY N*M:
 [[0.086, 0.114, 0.047, 0.153, 0.024, 0.045, 0.097, 0.053,
```

```
NumPY N*M:
 [[0.085852 0.113737 0.046813 ... 0.165958 0.04563  0.133341]
```

```
PY (N*M)*N:
 [[10.717], [28.53], [49.056], [25.155], [29.721], [16.422], [20.097],
```

```
NumPY (N*M)*N:
 [[10.7100228 ]
 [28.5178122 ]
 [49.05063698]
 [25.15904765]
 [29.7218976 ]
 [16.41358524]
 [20.08921437]
```

```
PY M*(N*M):
 [[32.199, 42.647, 17.56, 57.294, 8.86, 16.983,
```

```
NumPY M*(N*M):
 [[32.19344133 42.64997247 17.55429773 57.28911606  8.87220824 16.98394149
```

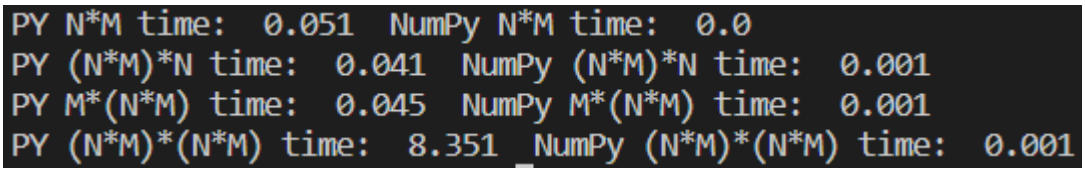
```
PY (N*M)*(N*M):
 [[5.444, 7.207, 2.968, 9.681, 1.502, 2.871, 6.167,
```

```
NumPY (N*M)*(N*M):
 [[ 5.44069158  7.20784535  2.96667632 ... 10.51724239  2.89170616
```

Далі по плану йдуть unit-тести зовнішніх функцій.

```
Unit Test 1 The answer of NumPY:
 [[1 2 3]
 [2 4 6]
 [3 6 9]]
Unit Test 1 for vector_mult() Passed
Unit Test 2 The answer of NumPY:
 [[14]
 [28]
 [42]]
Unit Test 2 for matrix_mult_vector() Passed
```

Наприкінці, виводимо час роботи кожного з прикладів обидвома способами.



В результаті бачимо, що **NumPY** функції є більш оптимізованими за мої, і тому значно ефективніше виконують обчислення.

Висновок: на цій лабораторній роботі було розглянуто алгоритми множення матриць, векторів та матриць на вектор, ознайомлено з бібліотекою NumPY та проведено часові заміри для порівняння часу виконання роботи між звичайними масивами та масивами NumPY. В результаті було виявлено, що використання вбудованих функцій NumPY є розумнішим варіантом, оскільки час виконання кожного з прикладів не зайняв довше одної тисячної секунди.