Министерство образования Республики Беларусь
Учреждение образования
«Брестский государственный технический университет»
Кафедра ИИТ

Лабораторная работа №7

По дисциплине: «Языки программирования»

Тема: «Изучение NumPy. Сравнение производительности с классическими библиотеками Python»

Выполнила:
Студентка 2 курса
Группы ПО-7
Фурсевич Д. С.
Проверил:
Бойко Д.О.

Брест, 2021

Цель работы: изучить NumPy. Сравнить производительность с классическими библиотеками Python

Ход работы:

Общие правила выполнения задания:

1. Для написания кода использовать библиотеки классического Python, NumPy и SciPy.

2. Код демонстрируется в Colaboratory

3. По каждому заданию должно быть предоставлено не менее 3-х вариантов решения, среди которых:

a) чистый NumPy (максимально оптимизованный, векторизованный)

b) любой не векторизованный вариант

c) любой другой вариант, желательно конкурентноспособный

4. Все варианты решения должны быть протестированы на скорость выполнения при помощи %timeit

5. Полученные результаты отразить в отчете и сделать выводы о производительности и комфорте использования NumPy в различных задачах.

Задание 1:

Подсчитать произведение ненулевых элементов на диагонали прямоугольной матрицы.

Пример: x = np.array([[1, 0, 1], [2, 0, 2], [3, 0, 3], [4, 4, 4]])

Ответ: 3.

1) Python

```
array = [[1, 0, 1], [2, 0, 2], [3, 0, 3], [4, 4, 4]]

def func():
    multipl = 1
    for i in range(len(array)-1):
        if array[i][i] != 0:
            multipl = multipl* array[i][i]
    return multipl

%timeit func()
func()
```

```
The slowest run took 5.02 times longer than the fastest. This could mean that an intermediate result is being cached.
1000000 loops, best of 5: 926 ns per loop
3
```

2) NumPy

```
array_ = np.array(([[1, 0, 1], [2, 0, 2], [3, 0, 3], [4, 4, 4]]))
#array_ = np.arange(12).reshape(3,4)
%timeit np.prod(array_.diagonal()[array_.diagonal() != 0])
np.prod(array_.diagonal()[array_.diagonal() != 0])
```

```
The slowest run took 654.39 times longer than the fastest. This could mean that an intermediate result is being cached.
100000 loops, best of 5: 7.04 µs per loop
3
```

3) Tensorflow

```
import tensorflow as tf
```

```
array = tf.constant(([[1, 0, 1], [2, 0, 2], [3, 0, 3], [4, 4, 4]]))
#array_ = np.arange(12).reshape(3,4)
%timeit tf.reduce_prod(tf.linalg.diag_part(array)[tf.linalg.diag_part(array)!=0])
tf.reduce_prod(tf.linalg.diag_part(array)[tf.linalg.diag_part(array)!=0])
```

```
The slowest run took 37.82 times longer than the fastest. This could mean that an intermediate result is being cached.
1000 loops, best of 5: 1.34 ms per loop
<tf.Tensor: shape=(), dtype=int32, numpy=3>
```

Задание 2:

Дана матрица x и два вектора одинаковой длины i и j. Построить вектор np.array([X[i[0], j[0]], X[i[1], j[1]], . . . , X[i[N-1], j[N-1]]]). Пример:

x = [[9 4 2], [6 0 0], [9 9 3]]

i: [1 2 1]

j: [1 0 1]

Ответ: [0 9 0]

1) Python

```
x = [[9, 4, 2], [6, 0, 0], [9, 9, 3]]
i = [1, 2, 1]
j = [1, 0, 1]
%timeit [x[a][b] for a, b in zip(i, j)]
[x[a][b] for a, b in zip(i, j)]
```

```
The slowest run took 5.29 times longer than the fastest. This could mean that an intermediate result is being cached.
1000000 loops, best of 5: 775 ns per loop
[0, 9, 0]
```

2) NumPy

```
x = np.array([[9, 4, 2], [6, 0, 0], [9, 9, 3]])
i = np.array([1, 2, 1])
j = np.array([1, 0, 1])

%timeit np.take(x, i*len(i) + j)
np.take(x, i*len(i) + j)
```

```
The slowest run took 18.71 times longer than the fastest. This could mean that an intermediate result is being cached.
100000 loops, best of 5: 4.12 µs per loop
array([0, 9, 0])
```

3) Tensorflow

```
[8]  x = tf.constant([[9, 4, 2], [6, 0, 0], [9, 9, 3]])
     i = tf.constant([1, 2, 1])
     j = tf.constant([1, 0, 1])

     def func(x, i, j):
       tf.gather(tf.reshape(x, (-1,)), i*len(i)+j)

     %timeit func(x, i, j)
     func(x, i, j)
```

```
The slowest run took 91.77 times longer than the fastest. This could mean that an intermediate result is being cached.
1000 loops, best of 5: 211 µs per loop
```

Задание 3:

Даны два вектора x и y. Проверить, задают ли они одно и то же мультимножество.

Пример: x = np.array([1,2, 2, 4]), y = np.array([4, 2, 1, 2])

Ответ: True.

1) Python

```
x = [1,2, 2, 4]
y = [4, 2, 1, 2]
%timeit set(x)==set(y)
set(x)==set(y)
```

```
The slowest run took 8.32 times longer than the fastest. This could mean that an intermediate result is being cached.
1000000 loops, best of 5: 506 ns per loop
True
```

2) NumPy

```
x = np.array([1,2, 4])
y = np.array([4, 2, 1, 3])

def func():
    ux = np.unique(x)
    uy = np.unique(y)
    return len(ux) == len(uy) and np.all(ux==uy)

%timeit func()
func()
```

```
The slowest run took 8.50 times longer than the fastest. This could mean that an intermediate result is being cached.
100000 loops, best of 5: 14.8 µs per loop
False
```

3) Tensorflow

```
x = tf.constant([1,2, 4])
y = tf.constant([4, 2, 1, 3])

def func():
    ux = tf.unique(x)
    uy = tf.unique(y)
    return ux.count == uy.count and tf.reduce_all(ux==uy)

%timeit func()
func()
```

```
The slowest run took 180.01 times longer than the fastest. This could mean that an intermediate result is being cached.
10000 loops, best of 5: 24.5 µs per loop
False
```

Задание 4:

Найти максимальный элемент в векторе x среди элементов, перед которыми стоит нулевой.

Пример: x = np.array([6, 2, 0, 3, 0, 0, 5, 7, 0])

Ответ: 5.

1) Python

```
x = [6, 2, 0, 3, 0, 0, 5, 7, 0]
%timeit max([x[i+1] for i in range(len(x)-1) if x[i]==0])
max([x[i+1] for i in range(len(x)-1) if x[i]==0])
```

```
The slowest run took 4.09 times longer than the fastest. This could mean that an intermediate result is being cached.
1000000 loops, best of 5: 1.57 µs per loop
5
```

2) NumPy

```
x = np.array([6, 2, 0, 3, 0, 0, 5, 7, 0])
%timeit np.max(x[1:][x[:-1]==0])
np.max(x[1:][x[:-1]==0])
```

```
The slowest run took 12.17 times longer than the fastest. This could mean that an intermediate result is being cached.
100000 loops, best of 5: 8.71 µs per loop
5
```

3) Tensorflow

```
[14] x = tf.constant([6, 2, 0, 3, 0, 0, 5, 7, 0])
     %timeit tf.reduce_max(x[1:][x[:-1]==0])
     tf.reduce_max(x[1:][x[:-1]==0])
```

```
The slowest run took 15.28 times longer than the fastest. This could mean that an intermediate result is being cached.
1000 loops, best of 5: 1.57 ms per loop
<tf.Tensor: shape=(), dtype=int32, numpy=5>
```

Задание 5:

Дан трёхмерный массив, содержащий изображение, размера (height, width, numChannels), а также вектор длины numChannels. Сложить каналы изображения с указанными весами, и вернуть результат в виде матрицы размера (height, width). Считать реальное изображение можно при помощи функции scipy.misc.imread (если изображение не в формате png, установите пакет pillow: conda install pillow). Преобразуйте цветное изображение в оттенки серого, использовав коэффициенты np.array([0.299, 0.587, 0.114]).

Пример:

Ответ:

```
import PIL
image = PIL.Image.open("audi-e-tron-audi-2019-cars-concept-cars.jpeg").resize((150*4,100*4))
width, height  = image.size
image
```

1) Python

```
weights = [0.299, 0.587, 0.114]
array = np.asarray(image).tolist()

def to_grayscale(pixel):
    return pixel[0]*weights[0]+pixel[1]*weights[1]+pixel[2]*weights[2]

def l(line):
    return list(map(to_grayscale, line))

%timeit list(map(l, array))
img = list(map(l, array))
PIL.Image.fromarray(np.asarray(img, dtype=np.uint8))
```

```
10 loops, best of 5: 111 ms per loop
```

2) NumPy

```
weights = np.array([0.299, 0.587, 0.114])
array = np.asarray(image)
img = np.dot(array, weights)
%timeit np.dot(array, weights)
PIL.Image.fromarray(np.asarray(img, dtype=np.uint8))
```

```
100 loops, best of 5: 8.16 ms per loop
```

3) Tensorflow

```
weights = tf.constant([0.299, 0.587, 0.114], dtype=tf.dtypes.float64)
array = tf.constant(np.asarray(image), dtype=tf.dtypes.float64)
img = tf.tensordot(array, weights, 1)
%timeit tf.tensordot(array, weights, 1)
PIL.Image.fromarray(np.asarray(img, dtype=np.uint8))
```

```
1000 loops, best of 5: 1.94 ms per loop
```

Задание 6:

Реализовать кодирование длин серий (Run-length encoding). Дан вектор x. Необходимо вернуть кортеж из двух векторов одинаковой длины. Первый содержит числа, а второй - сколько раз их нужно повторить.

Пример: x = np.array([2, 2, 2, 3, 3, 3, 5]).

Ответ: (np.array([2, 3, 5]), np.array([3, 3, 1])).

1) Python

```
[20] x = [2, 2, 2, 3, 3, 3, 5]
     def func():
       counts = dict()
       for e in x:
         if e not in counts:
           counts[e] = 1
         else:
           counts[e] += 1

       return list(counts.keys()), list(counts.values())

     %timeit func()
     func()
```

```
The slowest run took 6.21 times longer than the fastest. This could mean that an intermediate result is being cached.
1000000 loops, best of 5: 1.82 µs per loop
([2, 3, 5], [3, 3, 1])
```

2) NumPy

```
[21] x = np.array([2, 2, 2, 3, 3, 3, 5])

     def rle():
         n = len(x)
         y = x[1:] != x[:-1]
         i = np.append(np.where(y), n - 1)
         z = np.diff(np.append(-1, i))
         return(x[i], z)

     %timeit rle()
     rle()
```

```
The slowest run took 7.47 times longer than the fastest. This could mean that an intermediate result is being cached.
10000 loops, best of 5: 23.1 µs per loop
(array([2, 3, 5]), array([3, 3, 1]))
```

3) Tensorflow

```
[22] x = tf.constant([2, 2, 2, 3, 3, 3, 5])

     def rle(x):
         n = len(x)
         y = x[1:] != x[:-1]
         i = tf.concat((tf.where(y)[:,0], (n - 1,)), 0 )
         c = tf.concat(((-1,), i), 0)
         z = c[1:]-c[:-1]
         return(tf.gather(x, i), z)

     %timeit rle(x)
     rle(x)
```

```
The slowest run took 15.02 times longer than the fastest. This could mean that an intermediate result is being cached.
1000 loops, best of 5: 1.18 ms per loop
(<tf.Tensor: shape=(3,), dtype=int32, numpy=array([2, 3, 5], dtype=int32)>,
 <tf.Tensor: shape=(3,), dtype=int64, numpy=array([3, 3, 1])>)
```

Задание 7:

Даны две выборки объектов - X и Y. Вычислить матрицу евклидовых расстояний между объектами. Сравнить с функцией scipy.spatial.distance.euclidean.

Пример:

x: [2 7 6 6 9 6 3 4 9]

y: [1 0 0 7 2 2 4 3 0]

Ответ: 15.329709716755891

1) Python

```
[23] x = [2, 7, 6, 6, 9, 6, 3, 4, 9]
     y = [1, 0, 0, 7, 2, 2, 4, 3, 0]
     def l(t):
       return (t[0]-t[1])**2

     def f():
       z = list(map(l, zip(x, y)))
       sum = 0
       for i in z:
         sum += i
       return sum**0.5

     %timeit f()
     f()
```

```
The slowest run took 4.55 times longer than the fastest. This could mean that an intermediate result is being cached.
100000 loops, best of 5: 4.72 µs per loop
15.329709716755891
```

2) NumPy

```
x = np.array([2,7,6,6,9,6,3,4,9])
y = np.array([1,0,0,7,2,2,4,3,0])
def func():
  square = np.square(x - y)
  sum_square = np.sum(square)
  return np.sqrt(sum_square)

%timeit func()
func()
```

```
The slowest run took 26.92 times longer than the fastest. This could mean that an intermediate result is being cached.
100000 loops, best of 5: 8.61 µs per loop
15.329709716755891
```

3) Tensorflow

```
x = tf.constant([2,7,6,6,9,6,3,4,9])
y = tf.constant([1,0,0,7,2,2,4,3,0])
def func():
  square = tf.square(x - y)
  sum_square = tf.reduce_sum(square)
  return tf.sqrt(tf.cast(sum_square, tf.dtypes.float32))

%timeit func()
func()
```

```
The slowest run took 87.16 times longer than the fastest. This could mean that an intermediate result is being cached.
10000 loops, best of 5: 106 µs per loop
<tf.Tensor: shape=(), dtype=float32, numpy=15.32971>
```

Задание 8:

Реализовать функцию вычисления логарифма плотности многомерного нормального распределения. Входные параметры: точки X, размер (N, D), мат. ожидание m, вектор длины D, матрица ковариаций C, размер (D, D). Разрешается использовать библиотечные функции для подсчета определителя матрицы, а также обратной матрицы, в том числе в невекторизованном варианте. Сравнить с scipy.stats.multivariate_normal(m, C).logpdf(X) как по скорости работы, так и по точности вычислений

1) Python

```
[30] from numpy import *
     import scipy.sparse as sp
     import scipy.sparse.linalg as spl

     # коматрица
     matrx = array([[2.3, 0, 0, 0],
                    [0, 1.5, 0, 0],
                    [0, 0, 1.7, 0],
                    [0, 0, 0, 2]])

     # вектор средних значений
     srVect = array([2,3,8,10])
     # входные значения
     x = array([2.1, 3.5, 8, 9.5])

     def lognormpdf(x, srVect, matrx):
       lnt = len(matrx)
       norm = lnt*log(2*pi)+np.linalg.slogdet(matrx)[1]
       err = x-srVect
       numerator = spl.spsolve(matrx, err).T.dot(err)
       return -0.5*(norm+numerator)
     %timeit lognormpdf(x, srVect, matrx)
     lognormpdf(x, srVect, matrx)
```

```
/usr/local/lib/python3.7/dist-packages/scipy/sparse/linalg/dsolve/linsolve.py:138: SparseEfficiencyWarning: spsolve requires A be CSC or CSR matrix format
  SparseEfficiencyWarning)
The slowest run took 10.47 times longer than the fastest. This could mean that an intermediate result is being cached.
1000 loops, best of 5: 326 µs per loop
-5.054836210528194
```

2) NumPy

```
import numpy as np
import scipy.sparse as sp
import scipy.sparse.linalg as spl

# коматрица
matrx = np.array([[2.3, 0, 0, 0],
                  [0, 1.5, 0, 0],
                  [0, 0, 1.7, 0],
                  [0, 0, 0, 2]])

# вектор средних значений
srVect = np.array([2,3,8,10])
# входные значения
x = np.array([2.1, 3.5, 8, 9.5])

def lognormpdf(x, srVect, matrx):
  lnt = len(matrx)
  norm = lnt*log(2*pi)+np.linalg.slogdet(matrx)[1]
  err = x-srVect
  numerator = spl.spsolve(matrx, err).T.dot(err)
  return -0.5*(norm+numerator)
%timeit lognormpdf(x, srVect, matrx)
lognormpdf(x, srVect, matrx)
```

```
/usr/local/lib/python3.7/dist-packages/scipy/sparse/linalg/dsolve/linsolve.py:138: SparseEfficiencyWarning: spsolve requires A be CSC or CSR matrix format
  SparseEfficiencyWarning)
The slowest run took 5.48 times longer than the fastest. This could mean that an intermediate result is being cached.
1000 loops, best of 5: 338 µs per loop
-5.054836210528194
```

3) Pandas

```
import pandas as pd
import numpy as np
import scipy.sparse as sp
import scipy.sparse.linalg as spl

# коматрица
matrx = pd.DataFrame([[2.3, 0, 0, 0],
                  [0, 1.5, 0, 0],
                  [0, 0, 1.7, 0],
                  [0, 0, 0, 2]])

# вектор средних значений
srVect = pd.DataFrame([2,3,8,10])
# входные значения
x = pd.DataFrame([2.1, 3.5, 8, 9.5])

def lognormpdf(x, srVect, matrx):
  lnt = len(matrx)
  norm = lnt*log(2*pi)+np.linalg.slogdet(matrx)[1]
  err = x-srVect
  numerator = spl.spsolve(matrx, err).T.dot(err)
  return -0.5*(norm+numerator)
%timeit lognormpdf(x, srVect, matrx)
lognormpdf(x, srVect, matrx)
```

```
/usr/local/lib/python3.7/dist-packages/scipy/sparse/linalg/dsolve/linsolve.py:138: SparseEfficiencyWarning: spsolve requires A be CSC or CSR matrix format
  SparseEfficiencyWarning)
The slowest run took 20.30 times longer than the fastest. This could mean that an intermediate result is being cached.
1000 loops, best of 5: 1.06 ms per loop
array([-5.05483621])
```

Вывод: изучила NumPy. Сравнила производительность NumPy с производительностью обычного Python и с классической библиотекой Tensorflow.