PYTHON

Лекция 12

NUMPY: МОТИВАЦИЯ

- В Python легко и быстро пишется код, но медленная арифметика
- И громоздкое представление чисел
- В C/C++ все хорошо с эффективностью, но сложно писать код
- Под капотом numpy - C-код, с интерфейсом в Python-мир
- Арифметическая часть работает быстро
- Крупные приложения компонуются в Python-стиле

ОСНОВНЫЕ ИДЕИ И ПОНЯТИЯ

- Работаем с векторно-матричными структурами
- Разных размерностей
- В унифицированном виде
- Содержимое матриц/векторов типизировано и унифицировано

ХРАНЕНИЕ ДАННЫХ

- Основной фокус - на примитивные типы данных
- Целочисленные и с плавающей точкой
- С прямой аппаратной поддержкой
- И векторные операции с ними

ПЕРВЫЕ ШАГИ

- Устанавливаем через pip
- Или берем большой дистрибутив типа Anaconda
- ```
  import numpy
  ```
- Идиома:

```
import numpy as np
```
```

ПРИМЕР

```
1 import numpy as np
2
3 data1 = np.zeros(10)
4 data2 = np.ones((5, 8))
5
6 print(data1)
7 print(data2)
```

ЧТО СДЕЛАЛИ

- С точки зрения Python - создали два объекта
- Тип обоих - numpy.ndarray
- Это универсальный массив
- Первый - размерности 1, размера 10
- Второй - размерности 2, размера 5 на 8

ЧТО СДЕЛАЛИ

- Видим десятичные точки
- Тип элемента - с плавающей точкой
- Это numpy-тип, не Python-тип
- Тип элемента
- Можно узнать через атрибут dtype

ПРИМЕР

```
1 import numpy as np
2
3 data1 = np.zeros(10)
4 data2 = np.ones((5, 8))
5
6 print(type(data1))
7 print(data2.dtype)
8 print(data2.dtype.type)
```

ТИПИЗАЦИЯ NUMPY

- Тип описывает размер элемента данных и его интеретацию
- Каждый тип - специальный класс в модуле numpy
- Есть типы, опирающиеся на размер
- Есть типы, опирающиеся на C-аналог на данной платформе

ТИПИЗАЦИЯ NUMPY

- numpy.int8 - 8-битовый знаковый
- numpy.int16, numpy.int32, numpy.int64
- Беззнаковые аналоги: numpy.uint8 numpy.uint16, numpy.uint32, numpy.uint64
- Подбирается под задачу и предметную область

ПРИМЕР

```
 1 import numpy as np
 2
 3 data1 = np.zeros(10, dtype=np.int8)
 4 data2 = np.ones((5, 8), dtype=np.int16)
 5
 6 print(data1)
 7 print(data2)
 8
 9 data1[2] = 20
10 data1[5] = 200
11
12 print(data1)
```

ИЗДЕРЖКИ

- За скорость платим проблемами с переполнением
- Нужна осторожность и внимательность
- Особенно при вычислении средних арифметических

ОБРАЩЕНИЕ ПО ИНДЕКСАМ

- В одномерном случае - все просто, "как в списках"
- Только нет append
- Память выделятся под фиксированный размер
- Ее можно только переинтерпретировать
- Или создать новый ndarray большего размера

ОБРАЩЕНИЕ ПО ИНДЕКСАМ

- В двумерном случае поинтереснее
- Можно взять только один индекс - и получить "строку матрицы"
- И в ней - взять еще один индекс
- Но можно - и рекомендуется - использовать два индекса через запятую сразу

ПРИМЕР

```
 1 import numpy as np
 2
 3 data1 = np.zeros(10, dtype=np.int8)
 4 data2 = np.ones((5, 8), dtype=np.int16)
 5
 6 print(data2)
 7
 8 for i in range(8):
 9     data2[i % 5, i] = i * 100
10     data2[i % 5][i * 2 % 8] = i * 1000
```

ОБРАЩЕНИЕ ПО ИНДЕКСАМ

- Можно брать столбцы
- Для этого в качестве первого индекса можно указать :
- А после запятой - второй
- В общем случае каждый из индексов может быть отрезком, использовать отрицательные числа и т.п.

ГРУППОВОЕ ПРИСВАИВАНИЕ

- Можно всей вырезке присвоить значения другого ndarray
- Совпадающего по форме
- Можно - всем элементам присвоить константу
- Или выполнить обновляющее присваивание

ПРИМЕР

```
 1 import numpy as np
 2
 3 data1 = np.zeros(10, dtype=np.int8)
 4 data2 = np.ones((5, 8), dtype=np.int16)
 5
 6 data1[1:8:3] = 5
 7 print(data1)
 8
 9 data2[:, 2] = data1[1::2]
10 print(data2)
11
12 data2[2:4, 1:7] *= 100
13 print(data2)
```

ВАЖНАЯ ДЕТАЛЬ

- Можно присвоить константу всему ndarray
- Для этого надо написать:

```
data[:] = 5
```

- Сработает даже для многомерного
- А совсем без '[:]' не сработает - это будет Python-присваивание

SHAPE

- Форма данных описывается кортежем неотрицательных целых
- И хранится в атрибуте shape
- Произведение его элементов - количество элементов в ndarray
- Количество элементов в shape - количество измерений

СОЗДАНИЕ NDARRAY

- numpy.ndarray - из традиционных коллекций
- Не пойдет множество, строка, ленивые коллекции
- Пойдет список, кортеж, многомерные в любой комбинации
- Важно, чтобы была регулярность по всем размерностям
- Можно даже одно число

ПРИМЕР

```
1 import numpy as np
2
3 data0 = np.array(12345)
4 data1 = np.array([1, 2, 3], dtype=np.int8)
5 data2 = np.array((2, 3, 4, 5), dtype=np.int16)
6 data3 = np.array([(0, 1), (1, 2), (2, 3)], dtype=np.int8)
7 data4 = np.array(([2, 3 , 4, 5], (3, 4, 5, 6),
8                   [4, 5, 6, 7], [5, 6, 7, 8]),
9                   dtype=np.int16)
```

ARANGE

- Создает одномерный вектор
- По аналогии с range
- Только это материализованная структура
- linspace - вариация той же идеи
- Задаются начальная точка, конечная и количество точек

ПРИМЕР

```
1 import numpy as np
2
3 data1 = np.arange(10)
4 data2 = np.arange(2, 10, dtype=float)
5 data3 = np.arange(2, 3, 0.1)
6 data4 = np.linspace(1., 4., 6)
```

RESHAPE

- Полученные одномерные можно сделать многомерными
- Метод reshape те же данные переинтерпретирует по новому shape-у
- Важно, чтобы количество элементов в новой интерпретации совпадало с оригинальным
- Данные при этом не копируются

ПРИМЕР

```
1 import numpy as np
2
3 data1 = np.arange(10).reshape((2, 5))
4 data2 = np.arange(2, 10, dtype=float).reshape((2, 2, 2))
5 data3 = np.arange(2, 3, 0.1).reshape(2, 1, 5)
6 data4 = np.linspace(1., 4., 6).reshape((2, 3))
```

ИЗНАЧАЛЬНО МНОГОМЕРНЫЕ

- np.eye - создает диагональную матрицу (единичную)
- np.diag - создает диагональную матрицу в другом смысле
- np.vander - матрица Вандермонда
- np.zeros/ones - нули/единицы

ИНДЕКСИРОВАНИЕ

- Пусть у нас много размерностей
- Нас интересуют последняя
- Или несколько последних
- Есть отдельная конструкция - ... в качестве индекса

ПРИМЕР

```
1 import numpy as np
2
3 print(np.arange(24).reshape(2, 3, 4))
4 print(np.arange(24).reshape(2, 3, 4)[..., 0, 1, 2])
5 print(np.arange(24).reshape(2, 3, 4)[..., 1, 2])
6 print(np.arange(24).reshape(2, 3, 4)[..., 2])
7 print(np.arange(24).reshape(2, 3, 4)[...])
```

РАСШИРЕНИЕ РАЗМЕРНОСТИ

- Способ 1: reshape, добавив 1 в требуемые позиции
- Способ 2: индексация с добавлением None
- Способ 3: индексация с добавлением np.newaxis

ПРИМЕР

```
1 import numpy as np
2
3 data = np.array(5)
4
5 for _ in range(5):
6     print(data)
7     print(data.shape)
8     print()
9     data = data[np.newaxis]
```

ПРИМЕР

```
1 import numpy as np
2
3 data = np.array([2, 3])
4
5 for _ in range(5):
6     print(data)
7     print(data.shape)
8     print()
9     data = data[np.newaxis, :]
```

ПРИМЕР

```
1 import numpy as np
2
3 data = np.array([2, 3])
4
5 for _ in range(5):
6     print(data)
7     print(data.shape)
8     print()
9     data = data[:, np.newaxis]
```

ОСОБО КРАСИВЫЙ ПРИМЕР

```
1 import numpy as np
2
3 x = np.arange(1, 11)
4 print(x[:, np.newaxis] * x[np.newaxis, :])
```

NDARRAY КАК ИНДЕКС

- Один ndarray можно использовать как индекс в другом
- Или несколько через запятую
- В первом случае - выбираются элементы по этим индексам
- Во втором - по сформированным многомерным индексам

BROADCASTING

- В конечном итоге бинарные операции требуют совпадения shape-ов операндов
- Но есть правила неявного приведения
- Правило 1: если shape у одного из операндов позиция в shape-е равна 1, то это ОК
- Правило 2: если размерность у одного из операндов меньше, она дополняется единицами от начала

BROADCASTING

- В той размерности, где 1 - у нас есть законный индекс 0
- А у другого операнда - возможно, не 1
- Будем от первого операнда всегда использовать индекс 0
- Получаем такой "цилиндр"

BROADCASTING

- Операция со скаляром - частный случай
- У скаляра пустой shape
- Дополняем его единицами да размерности другого операнда
- Получается k-мерный одноэлемнентный nd-array
- Из которого всегда берется элемент по нулевым индексам

BROADCASTING

- Есть матрица a, shape (6, 10)
- Есть вектор b, shape (10,)
- a + b - законная операцмя
- a[i, j] складывается с b[j]

BROADCASTING

- a + b[:6] - операция незаконная
- a + b[:6, np.newaxis] - операция законная
- a[i, j] складывается с b[i]

ПРИМЕР

```
 1 import numpy as np
 2
 3 a = np.arange(60).reshape(6, 10)
 4 b = np.arange(10)
 5
 6 print(a)
 7 print()
 8 print(a + b)
 9 print()
10 print(a + b[:6, np.newaxis])
```

ТРАНСПОЗИЦИЯ

- Метод transpose
- Для двумерного случая очевидно
- Для многомерного - все размерности переворачиваются
- Чтобы найти элемент после транспонирования - переворачиваем набор индексов

SWAPAXES

- Можно поменять местами две оси
- В двумерном случае это сводится к транспозиции
- В многомерном - можно выбрать две оси, которые меняем
- Чтобы найти элемент после такого - меняем местами два индекса

КОНКАТЕНАЦИЯ

- В одномерном случае - склеиваем два вектора
- В общем случае выбираем ось конкатенации
- По этой оси склеиваемые объекты могут иметь разный размер
- По остальным - должны быть одинаковы

КОНКАТЕНАЦИЯ

- Вызываем numpy.concatenate
- Передаем кортеж склеиваемых объектов
- Указываем ось
- Получаем новый ndarray
- Размер по оси склейки - сумма размеров склеенных

ПРИМЕР

```
1 import numpy as np
2
3 a = np.arange(60).reshape(4, 15)
4 b = np.arange(30).reshape(2, 15)
5
6 data = np.concatenate((a, b))
7 print(data.shape) # (6, 15)
```

ПРИМЕР

- Есть картинка n пикселей в высоту и m в ширину
- Три цветовых канала
- В итоге shape (n, m, 3)
- Хотим умножить все слои на маску из 0 и 1 shape которой - (n, m)

ПРИМЕР

- Вариант 1:

```
image * mask.reshape(mask.shape + (1,))
```

- Вариант 2:

```
image * mask[:, :, np.newaxis]
```

- Вариант 3:

```
(image.transpose() * mask.transpose()).transpose()
```