Представление данных

Все современные системы машинного обучения используют тензоры в качестве основной структуры данных. Тензоры являются фундаментальной структурой данных — настолько фундаментальной, что это отразилось на названии библиотеки Google TensorFlow. В Python для работы с тензорами часто используется библиотека NumPy.

Фактически тензор — это контейнер для данных, практически всегда числовых. Другими словами, это контейнер для чисел. Например, матрица является двумерным тензором: тензоры — это обобщение матриц с произвольным количеством измерений.

Скаляры (тензор 0 ранга)

Тензор, содержащий единственное число, называется скаляром (скалярным, или тензором нулевого ранга). В Numpy число типа float32 или float64 — это скалярный тензор (или скалярный массив). Определить количество осей тензора Numpy можно с помощью атрибута ndim ; скалярный тензор имеет 0 осей (ndim == 0). Количество осей тензора также называют его рангом. Пример скаляра в NumPy:

```
import numpy as np
x = np.array(12)
print(x)
print(x.ndim)
print(type(x))
```

Векторы (тензор 1 ранга)

Одномерный массив чисел называют вектором, или тензором первого ранга. Тензор первого ранга имеет единственную ось. Пример создания вектора:

```
import numpy as np
x = np.array([12, 3, 6, 14, 7])
print(x)
print(x.ndim)
print(type(x))
```

Этот вектор содержит пять элементов и поэтому называется пятимерным вектором. Необходимо различать размерность вектора и тензора, так как пятимерный вектор и пятимерный тензор это абсолютно разные структуры. Пятимерный вектор имеет только одну ось и пять значений на этой оси, тогда как пятимерный тензор имеет пять осей (и может иметь любое количество значений на каждой из них). Мерность может обозначать или количество элементов на данной оси (как в случае с пятимерным вектором), или количество осей в тензоре (как в пятимерном тензоре), что иногда может вызывать путаницу.

Матрицы (тензоры 2 ранга)

Массив векторов — это матрица, или двумерный тензор. Матрица имеет две оси (часто их называют строками и столбцами). Матрицу можно представить как прямоугольную таблицу с числами:

```
import numpy as np
x = np.array([[5, 78, 2, 34, 0],
              [6, 79, 3, 35, 1],
              [7, 80, 4, 36, 2]])
print(x)
print(x.ndim)
print(type(x))
```

Тензоры третьего и высшего рангов

Если упаковать такие матрицы в новый массив, получится трехмерный тензор, который можно представить как числовой куб. Пример трехмерного тензора:

```
import numpy as np
x = np.array([[[5, 78, 2, 34, 0],
             [6, 79, 3, 35, 1],
             [7, 80, 4, 36, 2]],
             [[5, 78, 2, 34, 0],
             [6, 79, 3, 35, 1],
             [7, 80, 4, 36, 2]],
             [[5, 78, 2, 34, 0],
             [6, 79, 3, 35, 1],
             [7, 80, 4, 36, 2]]])
print(x)
print(x.ndim)
print(type(x))
```

Упаковав трехмерные тензоры в массив, вы получите четырехмерный тензор и т. д. В глубоком обучении чаще всего используются тензоры от нулевого ранга до четырехмерных, но иногда, например при обработке видеоданных, дело может дойти и до пятимерных тензоров.

Ключевые атрибуты тензоров

Тензор определяется тремя ключевыми атрибутами:

- Количество осей (ранг) — например, трехмерный тензор имеет три оси, а матрица — две. В библиотеках для Python, таких как Numpy, этот атрибут тензоров имеет имя ndim .
- Форма — кортеж целых чисел, описывающих количество измерений на каждой оси тензора. Например, матрица в предыдущем примере имеет форму (3, 5) , а трехмерный тензор имеет форму (3, 3, 5) . Вектор имеет форму с единственным элементом, например (5,) , тогда как скаляр имеет пустую форму (). В NumPy форма хранится в атрибуте shape.
- Тип данных (обычно в библиотеках для Python ему дается имя dtype) — это тип данных, содержащихся в тензоре; например, тензор может иметь тип float32 , uint8 , float64 и др. В редких случаях можно встретить тензоры типа char. Обратите внимание, что в Numpy (и в большинстве других библиотек) отсутствуют строковые тензоры, потому что тензоры хранятся в заранее выделенных, непрерывных сегментах памяти и строки, будучи сущностями с изменяющейся длиной, препятствуют использованию такой реализации. В NumPy форма хранится в атрибуте dtype.

Считывание тензоров из файла

NumPy представляет функционал для считывания данных из файла. Предположим, есть файл с содержимым:

```
1;2;3;
4;5;6;
7;8;9;
10;11;12;
13;14;15;
```

Код для считывания данных из файла:

```
import numpy as np

x = np.fromfile('data1.csv',dtype = 'int', sep = ';')
print(x)
print(type(x))
print(type(x[0]))
print(x.ndim)
```

В функции *np.fromfile* первый параметр отвечает за имя файла, dtype определяет тип данных элемента тензора, sep отвечает за разделитель между элементами в файле.

Если выполнить скрипт считывания, то будет следующий вывод:

```
[ 1  2  3  4  5  6  7  8  9 10 11 12 13 14 15]
<class 'numpy.ndarray'>
<class 'numpy.int32'>
1
```

Можно заметить, что данные считались в вектор (тензор 1 ранга). Для того, чтобы получить матрицу (тензор 2 ранга) такую как в файле, необходимо изменить форму тензора, для этого используется следующий скрипт:

```
y = np.reshape(x,[5,3])
print(y)
print(type(y))
print(type(y[0]))
print(y.ndim)
```

Функция *np.reshape* принимает тензор, который необходимо изменить и список размерностей нового тензора. После выполнения скрипта будет получен следующий вывод:

```
[[ 1  2  3]
 [ 4  5  6]
 [ 7  8  9]
 [10 11 12]
 [13 14 15]]
<class 'numpy.ndarray'>
<class 'numpy.ndarray'>
2
```

Изменять размерность каждый раз после считывания не всегда удобно, особенно если заранее не известны размеры тензора. Поэтому, есть более удобная функция для считывания из файла *np.genfromtxt* :

```
z = np.genfromtxt('data1.csv',dtype = 'int',delimiter=';')
print(z)
print(type(z))
print(z.ndim)
```

После выполнения скрипта будет следующий вывод:

```
[ 4  5  6 -1]
 [ 7  8  9 -1]
 [10 11 12 -1]
 [13 14 15 -1]]
<class 'numpy.ndarray'>
2
```

Можно заменить, что считывание произошло сразу в двумерный тензор, но появилась лишняя колонка. Это связано с тем, что в конце каждой строки стоит символ ‘;’. Поэтому, необходимо изменить содержимое файла следующим образом:

```
1;2;3
4;5;6
7;8;9
10;11;12
13;14;15
```

Данный формат точно соответствует формату csv.

Теперь выполним следующий код:

```
w = np.genfromtxt('data2.csv',dtype = 'int',delimiter=';')
print(w)
print(type(w))
print(w.ndim)
```

Получим следующий вывод:

```
[[ 1  2  3]
 [ 4  5  6]
 [ 7  8  9]
 [10 11 12]
 [13 14 15]]
<class 'numpy.ndarray'>
2
```

Как видно, данные сразу приобретают необходимую форму. К сожалению, у такого представления данных в файле есть ограничение, что таким образом можно представить данные в виде тензора не больше 2 ранга.

В общем случае первая ось (ось с индексом 0, потому что нумерация начинается с 0) во всех тензорах, с которыми вам придется столкнуться в глубоком обучении, будет осью образцов (иногда ее называют измерением образцов).

Практические примеры тензоров с данными

Перечислим несколько примеров тензоров с данными, которые могут встретиться вам в будущем. Данные, которыми вам придется манипулировать, почти всегда будут относиться к одной из следующих категорий:

- векторные данные — двумерные тензоры с формой (образцы, признаки). Наиболее часто встречающаяся форма данных. В таких наборах каждый образец может быть представлен вектором, а пакет, соответственно, двумерным тензором (то есть массивом векторов), где первая ось — это ось образцов, а вторая — ось признаков.
- временные ряды или последовательности — трехмерные тензоры с формой (образцы, метки_времени, признаки). Всякий раз, когда время (или понятие последовательной упорядоченности) играет важную роль в ваших данных, такие данные предпочтительнее сохранять в трехмерном тензоре с явной осью времени. Каждый образец может быть представлен как последовательность векторов (двумерных тензоров), а сам пакет данных — как трехмерный тензор, а сам пакет данных — как трехмерный тензор. В соответствии с соглашениями, ось времени всегда является второй осью (осью с индексом 1).
- изображения — четырехмерные тензоры с формой (образцы, высота, ширина, цвет) или с формой (образцы, цвет, высота, ширина). Обычно изображения имеют три измерения: высоту, ширину и цвет. Даже при том, что черно-белые изображения имеют только один канал цвета и могли бы храниться в двумерных тензорах, по соглашениям тензоры с изображениями всегда имеют три измерения, где для черно-белых изображений отводится только один канал цвета.
- видео — пятимерные тензоры с формой (образцы, кадры, высота, ширина, цвет) или с формой (образцы, кадры, цвет, высота, ширина). Видеоданные — один из немногих типов данных, для хранения которых требуются пятимерные тензоры. Видео можно представить как последовательность кадров, где каждый кадр — цветное изображение. Каждый кадр можно сохранить в трехмерном тензоре (высота, ширина, цвет) , соответственно, их последовательность можно сохранить в четырехмерном тензоре (кадры, высота, ширина, цвет) , а пакет разных видеороликов — в пятимерном тензоре с формой (образцы, кадры, высота, ширина, цвет)

Функции для работы с библиотекой numpy

arange

Функция аналогичная функции range, но возвращает сразу массив numpy.ndarray

```
import numpy as np

A = np.arange(10)

print(type(A)) #<class 'numpy.ndarray'>
print(A.shape) #(10,)
print(A) #[0 1 2 3 4 5 6 7 8 9]
```

transpose

Функция транспонирования тензора

Для одномерного случая:

```
import numpy as np

A = np.arange(5)
print(A.shape) #(5,)
print(A) #[0 1 2 3 4]
print(A.transpose().shape) #(5,)
print(A.transpose()) #[0 1 2 3 4]
```

В данном случае, так как размерность одна, то транспонирования не происходит

Если вектор представлен как матрица 5 на 1:

```
import numpy as np

A = np.arange(5).reshape([5,1])
print(A.shape) #(5, 1)
print(A) #[[0]
         #[1]
         #[2]
         #[3]
         #[4]]
print(A.transpose().shape) #(1, 5)
print(A.transpose()) #[[0 1 2 3 4]]
```

То есть для тензоров с размерностью 2 транспонирование происходит по правилам транспонирования матриц

Для тензоров большей размерности:

```
import numpy as np

A = np.arange(120).reshape([2,3,4,5])
print(A.shape) #(2, 3, 4, 5)
print(A.transpose().shape) #(5, 4, 3, 2)
```

ones, zeros, identity

Функции позволяют создать матрицу заполненную единицами, нулями, а также единичную соотвественно

```
import numpy as np

print(np.ones([3,4]))
'''
[[1. 1. 1. 1.]
 [1. 1. 1. 1.]
 [1. 1. 1. 1.]]
'''
print(np.zeros([4,3],dtype=float))
'''
[[0 0 0]
 [0 0 0]
 [0 0 0]
 [0 0 0]]
'''
print(np.identity(3))
'''
[[1. 0. 0.]
 [0. 1. 0.]
 [0. 0. 1.]]
'''
```

Математические операции

Для тензоров математические операции работают поэлементно

```
import numpy as np

A = np.array([[1,2],[3,4]])
print(A)
'''
[[1 2]
 [3 4]]
'''
B = np.array([[5,6],[7,8]])
print(B)
'''
[[5 6]
 [7 8]]
'''

print(A+B)
'''
[[ 6  8]
 [10 12]]
'''
print(A*B)
'''
[[ 5 12]
 [21 32]]
'''
print(A/B)
'''
[[0.2        0.33333333]
 [0.42857143 0.5       ]]
'''
```

Если размерности тензоров не совпадают, то меньший тензор будет преобразован для использования. Зачастую, он будет повторен несколько раз.

```
import numpy as np

A = np.array([1,2])
print(A)
'''
[1 2]
'''
B = np.array([[5,6],[7,8]])
print(B)
'''
[[5 6]
 [7 8]]
'''

print(A+B)
'''
[[ 6  8]
 [ 8 10]]
'''
print(A*B)
'''
[[ 5 12]
 [ 7 16]]
'''
```

```
print(A/B)
'''
[[0.2        0.33333333]
 [0.14285714 0.25      ]]
'''
```

Отсюда следует, что можно выполнять математические операции тензора со скаляром.

Также, в numpy есть стандартные математические функции, такие как log, asb, sin, exp, и.т.д., которые можно применять к тензорам.

dot и tensordot

NumPy обеспечивает много функций для работы с векторами и матрицами. Функция dot возвращает скалярное произведение векторов.

```
import numpy as np

a = np.array([1,2,3])
b = np.array([3,2,1])
c = np.dot(a,b)
print(c) #10
```

Также, функцией dot можно перемножать матрицы

```
import numpy as np

a = np.array([[1,2],[2,1]])
b = np.array([[1,1],[0,1]])
c = np.dot(a,b)
print(c)
'''
[[1 3]
 [2 3]]
'''
```

Для функции dot выполняются следующие условия:

- Если a и b одномерные тензоры - выполняется скалярное произведение векторов
- Если a и b двумерные тензоры - выполняется матричное произведение, но рекомендуется использоваться функцию matmul или оператор a @ b
- Если a и b скаляры - выполняется обычное умножение, что равносильно функции multiply и оператору a * b
- Если a N-мерный тензор и b 1-мерный тензор (вектор) - получается сумма произведений по последний оси a и b

```
a = np.array([[1,2],[3,4]])
b = np.array([-1,2])
c = np.dot(a,b)
print(c)
'''
[3 5]
c[0] = 3 = 1 * -1 + 2 * 2 = a[0][0] * b[0] + a[0][1] + b[1]
c[1] = 5 = 3 + -1 + 2 * 4 = a[1][0] * b[0] + a[1][1] + b[1]
Совпадают последние индексы a и b
'''
```

- Если a N-мерный тензор и b M-мерный тензор (M >= 2) - получается сумма произведений по последней оси a и предпоследней оси b

```
dot(a, b)[i,j,k,m] = sum(a[i,j,:] * b[k,:,m])
```

```
import numpy as np

a = np.arange(24).reshape([2,3,4])
b = np.arange(100).reshape([5,4,5])
c = np.dot(a,b)
print(c.shape) #(2, 3, 5, 5)
```

Функция tensordot позволяет вручную задать по каким осям производить сумму произведений.

```
import numpy as np

A = np.arange(5).reshape([1,5])
B = np.arange(5).reshape([5,1])

print(A)
'''
[[0 1 2 3 4]]
'''
print(B)
'''
[[0]
 [1]
 [2]
 [3]
 [4]]
'''

print(np.tensordot(A,B,axes=[0,1]))
'''
[[ 0  0  0  0  0]
 [ 0  1  2  3  4]
 [ 0  2  4  6  8]
 [ 0  3  6  9 12]
 [ 0  4  8 12 16]]
'''
print(np.dot(B,A)) #эквивалентная запись


print(np.tensordot(A,B,axes=[1,0]))
'''
[[30]]
'''
print(np.dot(A,B)) #эквивалентная запись
```

Также можно задать сумму сразу по нескольким осям

```
import numpy as np

a = np.arange(60.).reshape(3,4,5)
b = np.arange(24.).reshape(4,3,2)
c = np.tensordot(a,b, axes=([1,0],[0,1]))
print(c.shape) #(5, 2)
print(c)
```

```
'''
[[ 4400.,  4730.],
 [ 4532.,  4874.],
 [ 4664.,  5018.],
 [ 4796.,  5162.],
 [ 4928.,  5306.]]
'''
# Аналогичный, но более медленный способ
d = np.zeros((5,2))
for i in range(5):
    for j in range(2):
        for k in range(3):
            for n in range(4):
                d[i,j] += a[k,n,i] * b[n,k,j]
print(c == d)
'''
[[ True,  True],
 [ True,  True],
 [ True,  True],
 [ True,  True],
 [ True,  True]]
'''
```