

# Pytorch 基本数据类型

## 基本类型

在python中的数据类型在pytorch中基本都能一一对应，除了string

 image-20201101130754167

pytorch并没有string类型，但是可以使用编码类型来表示。比如10表示小狗，01表示猫。在nlp中有专门的模块来解决这个问题。

### How to denote string

- One-hot
  - [0, 1, 0, 0, ...]
- Embedding
  - Word2vec
  - glove

$\begin{bmatrix} 1 & 0 \end{bmatrix}$  dog  
 $\begin{bmatrix} 0 & 1 \end{bmatrix}$  cat

A    0x34

pytorch中的数据类型

 image-20201101132206518

## 查看类型

`a.type()` 是torch中的方法，可以返回详细的类型

`type(a)` 是python自带的方法，没有提供额外信息

`isinstance(a, torch.FloatTensor)` 一般用这种方法来比对类型

 image-20201101132652746

数据放置于cpu和GPU上所表示的数据类型是不同的

 image-20201101133040551

## dim0

`torch.tensor(1.3)`：对于标量数据类型可以使用该方法来创建一个0维数组。



`a.shape`：可以查看tensor张量的形状

`a.size()`：与[a.shape](#) 对应，不同之处在于一个直接是属性，另一个是函数。



## dim1

一维数据的生成

`torch.tensor([1])`：这个方法圆括号里面有中括号，会根据里面的值生成一维张量。

`torch.FloatTensor()`：当里面只给出了一个数字，则会根据数字生成未初始化的相应一维张量。如果里面有中括号，则会根据中括号的内容生成一维张量。



将numpy中的一维数组转换为一维tensor张量。



## dim2

下面使用随机正态分布创建了一个dim为2的张量。

`size`函数如果不给值，则返回整个的形状。如果给了参数，则返回的是该维度上的形状大小。

```
In [6]: 1 a = torch.randn(2, 3)
        2 a

        tensor([[ 5.3380e-01,  3.4226e-01, -2.3923e-01],
                  [ 6.4652e-04, -6.9289e-01, -5.1524e-01]])

In [7]: 1 a.shape

        torch.Size([2, 3])

In [8]: 1 a.size()

        torch.Size([2, 3])

In [9]: 1 a.size(0)

        2

In [10]: 1 a.size(1)

        3

In [11]: 1 a.shape[1]

        3
```

### dim3

下面使用随机均匀分布创建一个dim为3的张量。如20句话，每句话10个单词，每个单词用100个分量的向量表示，得到的Tensor就是shape=[20, 10, 100]

可以使用list方法直接将数据转换为列表。方便和python的交互。

三维数据适合文字处理。

```
In [12]: 1 a = torch.rand(1, 2, 3)
```

```
In [13]: 1 a
```

```
tensor([[[[0.8348, 0.7523, 0.7615],  
          [0.3006, 0.9928, 0.9745]]]])
```

```
In [14]: 1 a.shape
```

```
torch.Size([1, 2, 3])
```

```
In [15]: 1 a[0]
```

```
tensor([[0.8348, 0.7523, 0.7615],  
        [0.3006, 0.9928, 0.9745]])
```

```
In [16]: 1 list(a.shape)
```

```
[1, 2, 3]
```

## dim4

```
a = torch.rand(2,3,28,28)
```

**dim**为4的张量经常用来表示图片。从左往右，第一个数字表示有几张图片，第二个数字表示**rgb**的第几个通道，比如1表示灰色图片，3表示彩色图片。第三、第四分别表示图片的长和宽。

例如100张MNIST数据集的灰度图(通道数为1，如果是RGB图像通道数就是3)，每张图高28像素，宽28像素，那么这个Tensor的shape=[100, 1, 28, 28]，也就是一个batch的数据维度：[batch\_size, channel, height, width]。

```
In [17]: 1 a = torch.rand(2, 3, 28, 28)
```

```
In [18]: 1 a
```

```
tensor([[[[0.9447, 0.6063, 0.0824, ..., 0.9408, 0.2592, 0.3235],
          [0.0143, 0.6584, 0.0293, ..., 0.5035, 0.9352, 0.5687],
          [0.9418, 0.3017, 0.5470, ..., 0.5476, 0.0730, 0.5971],
          ...,
          [0.0565, 0.5171, 0.1369, ..., 0.5422, 0.8406, 0.9025],
          [0.4691, 0.0039, 0.5923, ..., 0.9344, 0.8875, 0.6200],
          [0.2026, 0.3039, 0.0047, ..., 0.2968, 0.5688, 0.2636]],
        ...,
        [[0.1914, 0.7174, 0.8890, ..., 0.9630, 0.8315, 0.5779],
          [0.0243, 0.2354, 0.6620, ..., 0.4861, 0.1339, 0.4970],
          [0.0822, 0.6797, 0.9471, ..., 0.0499, 0.9995, 0.7311],
          ...,
          [0.2012, 0.1816, 0.2162, ..., 0.6616, 0.1666, 0.5171]],
        ...,
        [[0.0954, 0.1963, 0.2051, ..., 0.2747, 0.3057, 0.8499],
          [0.8138, 0.0126, 0.1896, ..., 0.6644, 0.5349, 0.7500],
          [0.9968, 0.9643, 0.8014, ..., 0.0192, 0.3672, 0.2546],
          ...,
          [0.0677, 0.1955, 0.6148, ..., 0.2805, 0.2061, 0.7514],
          [0.4365, 0.3475, 0.4452, ..., 0.4921, 0.1533, 0.1790],
          [0.6631, 0.1556, 0.0752, ..., 0.7873, 0.0296, 0.9358]]]])
```

```
In [19]: 1 a.shape
```

```
torch.Size([2, 3, 28, 28])
```

## 额外知识

`torch.numel()` : 得到tensor数据的具体大小。

`torch.dim()` : 和`length(a.shape)` 方法一样，返回的数据表示tensor张量的维度大小。

```
In [19]: 1 a.shape
```

```
torch.Size([2, 3, 28, 28])
```

```
In [20]: 1 a.numel()
```

```
4704
```

```
In [21]: 1 a = torch.tensor(1)
```

```
In [22]: 1 a.dim()
```

```
0
```

## 创建tensor

### 导入数据

从numpy导入数据

导入的数据值和数据类型不变，只是从numpy数组变为了张量。

```
In [48]: 1 a = np.array([2, 3, 4])
          2 b = torch.from_numpy(a)
          3
          4 print(b)

tensor([2, 3, 4], dtype=torch.int32)
```

```
In [49]: 1 a = np.ones([2, 3])
          2 b = torch.from_numpy(a)
          3
          4 print(b)

tensor([[1., 1., 1.],
        [1., 1., 1.]], dtype=torch.float64)
```

从list导入数据。

`torch.tensor()` 根据数据生成张量，建议有数据的情况尽量使用小写的生成，有助于区分。

`torch.FloatTensor()` 既可以根据已有数据来生成，也可以根据给出的shape数据生成。

```
In [50]: 1 torch.tensor([2, 3.4])
```

```
tensor([2.0000, 3.4000])
```

```
In [52]: 1 torch.tensor([[2, 3.4], [1, 24]])
```

```
tensor([[ 2.0000,  3.4000],  
        [ 1.0000, 24.0000]])
```

```
In [51]: 1 torch.FloatTensor([2, 3.4])
```

```
tensor([2.0000, 3.4000])
```

```
In [54]: 1 torch.FloatTensor(2, 3)
```

```
tensor([[6.5917e-10, 1.2915e-11, 2.6367e-09],  
        [6.7944e+22, 1.6689e+22, 1.6131e-07]])
```

## 创建未初始化张量

**torch.empty()**：给定shape可以生成未初始化的数据。

**torch.FloatTensor()**：给定shape可以生成未初始化的数据。也可以根据具体的数据生成张量。

生成的数据是随机的，可能很大或者很小，一般不能直接用于计算，需要后续的赋值。

```
In [56]: 1 torch.empty(2, 3)
```

```
tensor([[0.0000e+00, 0.0000e+00, 1.8754e+28],  
        [1.2870e+22, 6.5917e-10, 1.2915e-11]])
```

```
In [60]: 1 torch.FloatTensor(2, 3)
```

```
tensor([[0.0000e+00, 1.0837e-38, 7.0295e+28],  
        [6.1949e-04, 4.7429e+30, 1.1162e+04]])
```

## 初始化张量

`torch.rand()`：初始化的每一个数据都是在从0 到1 之间随机均匀选取的。

`torch.rand_like()`：接收的是一个tensor张量，读取其shape数据，之后再用`torch.rand()`进行从0到1的随机均匀选取。

`torch.randint(low,high,size)`：这个方法前两个参数是取值的范围，左闭右开。第三个参数是传入一个tuple类型的size参数。

```
In [92]: 1 a = torch.rand(3,3)
          2 print(a)

tensor([[0.8122, 0.5964, 0.2598],
        [0.4928, 0.4003, 0.9730],
        [0.0209, 0.9446, 0.4661]])
```

```
In [91]: 1 torch.rand_like(a)

tensor([[0.0893, 0.9492, 0.1049],
        [0.8169, 0.3615, 0.4931],
        [0.5040, 0.9544, 0.7311]])
```

```
In [97]: 1 torch.randint(1,10, (3,3))

tensor([[5, 3, 6],
        [8, 8, 8],
        [4, 1, 3]])
```

`torch.randn(3,3)`：根据正态化生成数据， $N(0,1)$ ，以0为均值，以1为方差生成数据。

`torch.normal(mean=3,std=torch.arange(1,0,-0.1))`：mean是生成数据的平均值，std是数据的方差，后面表示方差从1到0以步长0.1减小。也可以只填写前面两个值，默认步长为1。后面括号中的参数，需要包含浮点数，全是小数的时候会报错。



```

In [121]: 1 torch.randn(3,3)

          tensor([[ -1.1066, -0.6213,  1.3152],
                  [-0.7086, -0.3150, -0.1513],
                  [ 0.8390,  1.4515, -2.0257]])

In [128]: 1 torch.normal(mean=3, std=torch.arange(1., 6.))

          tensor([ 5.1539,  0.8118,  6.6435, -5.4808, -3.5722])

In [130]: 1 a = torch.normal(mean=5, std=torch.arange(1, 0, -0.1))
          2 print(a)

          tensor([5.3814, 6.1390, 5.6994, 4.3170, 5.2432, 4.8204, 4.7814, 4.5104, 4.8557,
                  4.9467])

In [131]: 1 a.view((2,5))

          tensor([[5.3814, 6.1390, 5.6994, 4.3170, 5.2432],
                  [4.8204, 4.7814, 4.5104, 4.8557, 4.9467]])

```

`torch.full([2,3],5)`：表示生成一个2\*3的矩阵，里面的数据全部是5。数据的类型为默认类型。

若需要生成标量，则传入中括号空值；生成一维tensor张量只需要传入1就行。

 image-20201101205946412

`torch.arange(0,10)`：生成从0开始到但不包括10的等差数列。也可以传入一个数作为步长。

`torch.range(0,10)`：也可以实现上面函数一样的功能。但该函数之后会被删除，所以不推荐使用。

 image-20201101210900462

`torch.linspace(start, end, steps)`：从start到end之间按照steps进行均分，之后返回steps个数这么多的等差数列。

`torch.logspace(start, end, steps, base)`：取得的是两个数的对数，在两个数之间以步数取均值。比如第三个例子是从10的0次方到10的-1次方，平均来取。base是设置对数的基数，默认值为10。

 image-20201101213822693

`torch.ones()`：生成全是1的张量，维度由传入的数字确定，可以传入元组或者列表。

`torch.zeros()` : 同上

`torch.eye()`: 生成对角矩阵, 但是传入的参数, 只能是两个或者一个。更高维度的将不适用。

 image-20201101213758975

 image-20201101214457272

`torch.randperm(n)` : 生成0到n-1 的一维张量, 顺序是打散了的。

 image-20201101215205618

运用的例子。比如数据的第一排表示的是人, 后面分别表示的是数学成绩和语文成绩, 我们希望两个数据集能够同步。比如人都从第一行换到第二行。

 image-20201101220550297

## 设置默认类型

`torch.tensor()` : 里面的数据类型会给定默认的数据类型。比如整数为`LongTensor`类型张量, 小数的为默认设置的张量类型。

`torch.Tensor ()` : 无论是整数还是小数, 全部设置为默认的`tensor`类型。

`torch.set_default_tensor_type(torch.DoubleTensor)` 使用该方法可以设置默认类型。

 image-20201101151601102