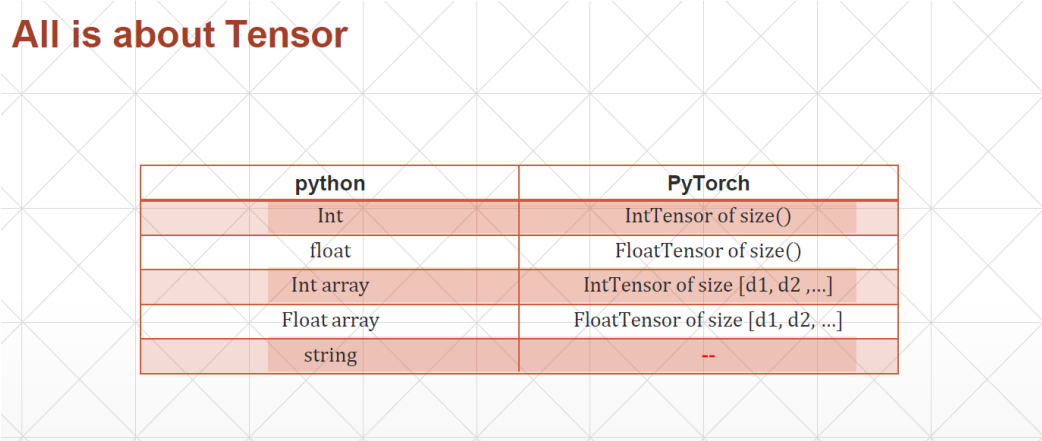
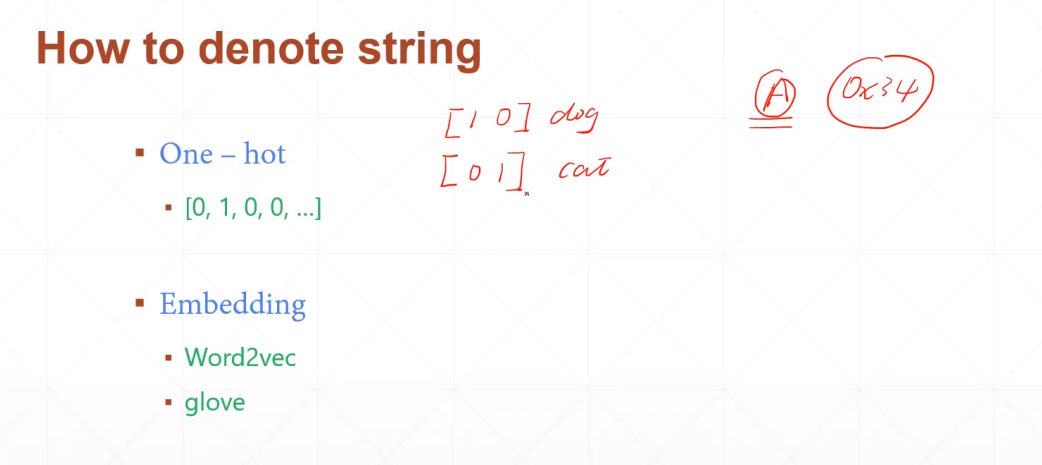
Pytorch 基本数据类型

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



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。



pytorch中的数据类型

Data type

Data type	dtype	CPU tensor	GPU tensor
32-bit floating point	`torch.float32` or `torch.float`	`torch.FloatTensor`	`torch.cuda.FloatTensor`
64-bit floating point	`torch.float64` or `torch.double`	`torch.DoubleTensor`	`torch.cuda.DoubleTensor`
16-bit floating point	`torch.float16` or `torch.half`	`torch.HalfTensor`	`torch.cuda.HalfTensor`
8-bit integer (unsigned)	`torch.uint8`	`torch.ByteTensor`	`torch.cuda.ByteTensor`
8-bit integer (signed)	`torch.int8`	`torch.CharTensor`	`torch.cuda.CharTensor`
16-bit integer (signed)	`torch.int16` or `torch.short`	`torch.ShortTensor`	`torch.cuda.ShortTensor`
32-bit integer (signed)	`torch.int32` or `torch.int`	`torch.IntTensor`	`torch.cuda.IntTensor`
64-bit integer (signed)	`torch.int64` or `torch.long`	`torch.LongTensor`	`torch.cuda.LongTensor`

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

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

In [3]:
1 a.type()

'torch.FloatTensor'

In [4]:
1 type(a)

torch.Tensor

In [5]:
1 isinstance(a, torch.FloatTensor)

True
```

数据放置于cpu和GPU上所表示的数据类型是不同的

```
In [21]: isinstance(data, torch.cuda.DoubleTensor)
Out[21]: False

In [22]: data=data.cuda()

In [23]: isinstance(data, torch.cuda.DoubleTensor)
Out[23]: True
```

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。

```
In [6]:
1 a = torch.tensor(2.3)

In [7]:
1 a.shape

torch.Size([])

In [10]:
1 len(a.shape)

0

In [11]:
1 a.size()

torch.Size([])

In [13]:
1 torch.tensor(1.)

tensor(1.)

In [14]:
1 torch.tensor(1.3)

tensor(1.3000)
```

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。

```
In [33]:
1 torch.tensor([1])
```

```
tensor([1])
```

```
In [23]:
1 torch.tensor([1.1,2.3])
```

```
tensor([1.1000, 2.3000])
```

```
In [30]:
1 torch.FloatTensor(1)
```

```
tensor([0.])
```

```
In [31]:
1 torch.FloatTensor(3)
```

```
tensor([0., 0., 0.])
```

```
In [32]:
1 torch.FloatTensor([1,2,3])
```

```
tensor([1., 2., 3.])
```

```
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]])
```

将numpy中的一维数组转换为一维tensor张量。

In [36]:
```
import numpy as np
```

In [40]:
```
data = np.ones(3)
print(data)
```

```
[1. 1. 1.]
```

In [44]:
```
#将numpy中的一维数组转换为tensor张量
data_1 = torch.from_numpy(data)

print(data_1)
print(data_1.shape)
print(data_1.size())
```

```
tensor([1., 1., 1.], dtype=torch.float64)
torch.Size([3])
torch.Size([3])
```

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

print(b)
```

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

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],
          ...[0.2871, 0.7879, 0.2751,  ..., 0.3978, 0.1388, 0.5197]],

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

print(b)
```

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

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

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.Tensor(): 若为指定生成张量类型，则生成默认的类型，一般是FlaotTensor。

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

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

生成的额数据是随机的，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 [63]:
1 torch.empty(1)

tensor([0.])

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

tensor([[1.0503e-05, 6.4108e-10, 6.5640e-07],
        [2.6540e+20, 2.7058e+23, 4.2247e-05]])

In [4]:
1 torch.Tensor(2, 3)

tensor([[0.0000e+00, 6.4108e-10, 1.8754e+28],
        [6.4978e-07, 4.2700e-08, 1.2914e-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]])

In [9]:
1 torch.IntTensor(2, 4)

tensor([[ 909716793,  959276592,  943075379,  895889464],
        [ 775240033, 1667594341, 1600484469, 1970496882]], dtype=torch.int32)
```

初始化张量

随机生成

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.randperm(n) ：生成0到n-1 的一维张量，顺序是打散了的。

```
In [173]:
1  torch.randperm(4)
```

```
tensor([1, 3, 2, 0])
```

```
In [183]:
1  a = torch.rand(2,3)
2  b = torch.rand(2,2)
3  idx = torch.randperm(2)
```

```
In [181]:
1  idx
```

```
tensor([0, 1])
```

```
In [184]:
1  idx
```

```
tensor([1, 0])
```

根据数值要求

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。

```
In [132]:
1  torch.full([2,3],5)
```

```
tensor([[5, 5, 5],
        [5, 5, 5]])
```

```
In [133]:
1  torch.full([],7)
```

```
tensor(7)
```

```
In [134]:
1  torch.full([1],7)
```

```
tensor([7])
```

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。

```
In [137]: 1 torch.arange(0.,10.)

tensor([0., 1., 2., 3., 4., 5., 6., 7., 8., 9.])

In [138]: 1 torch.arange(1,10,2)

tensor([1, 3, 5, 7, 9])

In [139]: 1 torch.range(0,10)

<ipython-input-139-dfdec9b83f7d>:1: UserWarning: torch.range is deprecated and will be removed in a future release because its behavior is inconsistent with Python's range builtin. Instead, use torch.arange, which produces values in [start, end).
  torch.range(0,10)

tensor([ 0.,  1.,  2.,  3.,  4.,  5.,  6.,  7.,  8.,  9., 10.])

In [140]: 1 torch.range(0,10,2)

<ipython-input-140-e2293d9db2d9>:1: UserWarning: torch.range is deprecated and will be removed in a future release because its behavior is inconsistent with Python's range builtin. Instead, use torch.arange, which produces values in [start, end).
  torch.range(0,10,2)

tensor([ 0.,  2.,  4.,  6.,  8., 10.])
```

torch.linspace(start, end, steps)： 从start到end之间（两数闭区间）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.

```
In [141]: 1 torch.linspace(0,10,steps=4)

tensor([ 0.0000,  3.3333,  6.6667, 10.0000])

In [143]: 1 torch.linspace(0,10,steps=11)

tensor([ 0.,  1.,  2.,  3.,  4.,  5.,  6.,  7.,  8.,  9., 10.])

In [145]: 1 torch.logspace(0,-1,steps=10)

tensor([1.0000, 0.7743, 0.5995, 0.4642, 0.3594, 0.2783, 0.2154, 0.1668, 0.1292,
        0.1000])

In [146]: 1 torch.logspace(0,1,steps=10)

tensor([ 1.0000,  1.2915,  1.6681,  2.1544,  2.7826,  3.5938,  4.6416,  5.9948,
         7.7426, 10.0000])

In [150]: 1 torch.logspace(1,2,steps=5,base=2)

tensor([2.0000, 2.3784, 2.8284, 3.3636, 4.0000])
```

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。

```
In [163]:
1 torch.ones(2,3)
```

```
tensor([[1., 1., 1.],
        [1., 1., 1.]])
```

```
In [162]:
1 torch.ones((2,3))
```

```
tensor([[1., 1., 1.],
        [1., 1., 1.]])
```

```
In [164]:
1 torch.zeros(2,3)
```

```
tensor([[0., 0., 0.],
        [0., 0., 0.]])
```

In [158]:
```
a = torch.zeros(3,3)
print(a)
```

```
tensor([[0., 0., 0.],
        [0., 0., 0.],
        [0., 0., 0.]])
```

In [159]:
```
torch.ones_like(a)
```

```
tensor([[1., 1., 1.],
        [1., 1., 1.],
        [1., 1., 1.]])
```

In [166]:
```
torch.eye(2,3)
```

```
tensor([[1., 0., 0.],
        [0., 1., 0.]])
```

In [155]:
```
torch.eye(3)
```

```
tensor([[1., 0., 0.],
        [0., 1., 0.],
        [0., 0., 1.]])
```

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。

```
In [186]:
1 a
```

```
tensor([[0.3756, 0.3711, 0.8872],
        [0.9414, 0.3014, 0.1113]])
```

```
In [185]:
1 a[idx]
```

```
tensor([[0.9414, 0.3014, 0.1113],
        [0.3756, 0.3711, 0.8872]])
```

```
In [187]:
1 b
```

```
tensor([[0.4414, 0.8936],
        [0.3061, 0.1587]])
```

```
In [188]:
1 b[idx]
```

```
tensor([[0.3061, 0.1587],
        [0.4414, 0.8936]])
```

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。生成的torsor张量数据类型不会被默认类型改变。

torch.Tensor（）：无论是整数还是小数，全部设置为默认的tensor类型。生成的tensor数据类型会受到默认设置的改变。

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

In [3]:
```
torch.tensor([1,2,3]).type()
```

'torch.LongTensor'

In [4]:
```
torch.tensor([2.1,3]).type()
```

'torch.FloatTensor'

In [5]:
```
torch.Tensor([1,2,3]).type()
```

'torch.FloatTensor'

In [6]:
```
#设置默认数据类型
torch.set_default_tensor_type(torch.DoubleTensor)
```

In [6]:
```
#设置默认数据类型
torch.set_default_tensor_type(torch.DoubleTensor)
```

In [7]:
```
torch.Tensor(2,3).type()
```

'torch.DoubleTensor'

In [9]:
```
torch.Tensor([1,2,3]).type()
```

'torch.DoubleTensor'

In [8]:
```
torch.tensor([1,2,3]).type()
```

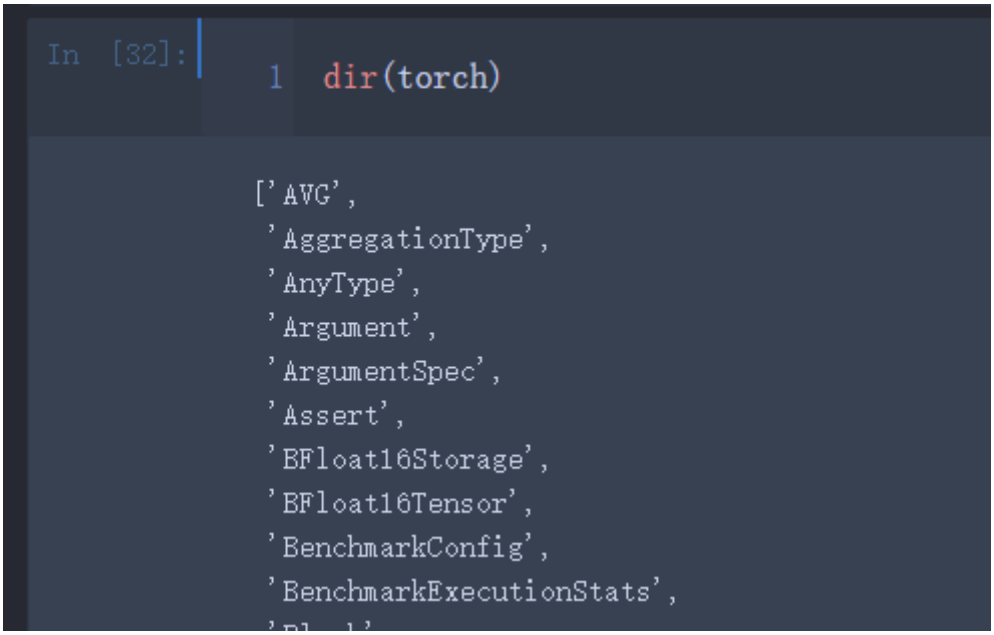
'torch.LongTensor'

In [12]:
```
torch.tensor((2,3)).type()
```

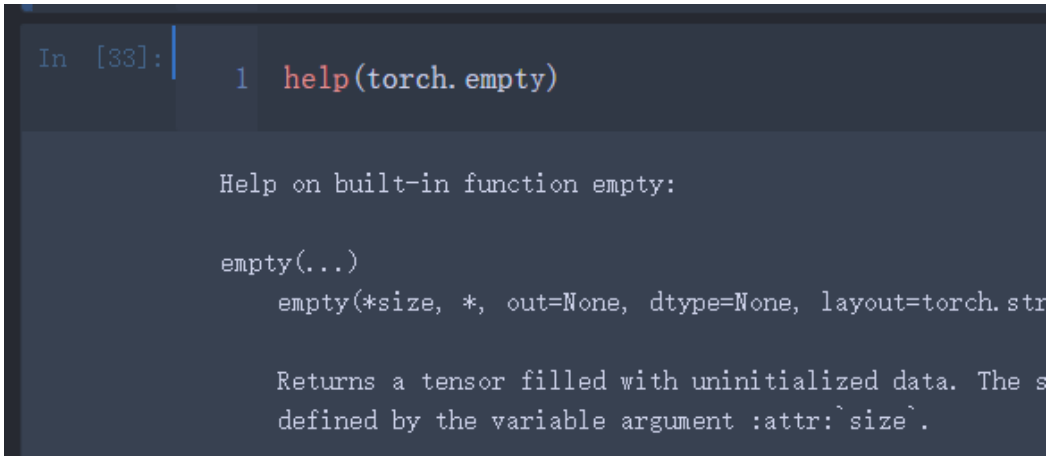
'torch.LongTensor'

参数解析

使用dir可以查看里面有哪些工具，



使用help()可以查看如何使用工具



创建一个未被初始化数值的tensor,tensor的大小是由size确定

size: 定义tensor的shape ，这里可以是一个list 也可以是一个tuple

out: 根据之前已有张量，则根据其数据复制给新的张量。之后将该数据重新复制给改变量。如果之前没有改张量，则相当于将该张量赋值。

dtype:（可选）我不设置值 默认值就是torch.set_default_tensor_type制定的值，如果需要设置那就是torch.dtype的那几个。作用是指定返回tensor的数据类型

layout:(可选)值为 torch.layout。 torch.layout表示torch.Tensor内存布局的对象。有torch.strided(dense Tensors 默认)并为torch.sparse_coo(sparse COO Tensors)提供实验支持。
torch.strided代表密集张量，是最常用的内存布局。每个strided张量都会关联 一个torch.Storage，它保存着它的数据。这些张力提供了多维度， 存储的strided视图。Strides是一个整数型列表：k-th stride表示在张量的第k维从一个元素跳转到下一个元素所需的内存。

device：表现 torch.Tensor被分配的设备类型的类，其中分为'cpu' 和 'cuda'两种。没有设置就使用当前设备。

```
requires_grad: (可选)是bool 类型的值，默认值是False 是否计算梯度

pin_memory：是否存于锁页内存
```

out参数：

```
In [17]:
y=torch.Tensor(2,3)
print(y)# 这里的输出的y的是  tensor([[1,  2,  3]])
z=torch.empty(size=[1,4],out=y)
print(z) # 这里输出的z和上一步输出的一模一样
print(y)

tensor([[1.4013e-45, 0.0000e+00, 1.4013e-45],
        [0.0000e+00, 1.4013e-45, 0.0000e+00]])
tensor([[1.4013e-45, 0.0000e+00, 1.4013e-45, 0.0000e+00]])
tensor([[1.4013e-45, 0.0000e+00, 1.4013e-45, 0.0000e+00]])
```

layout参数：

表示张量内存布局的对象。`Strides`是一个整数型列表：`k-th stride`表示在张量的第k维从一个元素跳转到下一个元素所需的内存。

```
#12表示从最高维度的一个元素到下一个元素所需内存
#4  表示从在第二个维度中一个元素跳到下一个元素所需内存
#1   表示在第三个也就是最后一个维度红一个元素到下一个元素所需内存
```

```
In [44]:
a = torch.Tensor(2,3,4)
a

tensor([[[1.0623e-05, 4.1655e-11, 2.6949e-09, 2.9572e-18],
         [6.7333e+22, 1.7591e+22, 1.7184e+25, 4.3222e+27],
         [6.1972e-04, 7.2443e+22, 1.7728e+28, 7.0367e+22]],

        [[6.0434e-07, 4.2700e-08, 1.2914e-11, 2.7303e-06],
         [1.7062e-07, 2.7593e-06, 1.6537e-04, 4.3437e-05],
         [8.5752e-07, 2.4560e-18, 7.7052e+31, 1.9447e+31]]])

In [45]:
#12表示从最高维度的一个元素到下一个元素所需内存
#4  表示从在第二个维度中一个元素跳到下一个元素所需内存
# 1   表示在第三个也就是最后一个维度红一个元素到下一个元素所需内存
a.stride()

(12, 4, 1)
```