


What's the difference between reshape and view in pytorch?



Asked 5 years, 6 months ago Modified 7 months ago Viewed 128k times


225

In numpy, we use `ndarray.reshape()` for reshaping an array.

I noticed that in pytorch, people use `torch.view(...)` for the same purpose, but at the same time, there is also a `torch.reshape(...)` existing.

So I am wondering what the differences are between them and when I should use either of them?

[python](#) [pytorch](#)

Share Follow

edited Mar 29, 2022 at 8:57


 **Mateen Ulhaq**
24.8k 19 102 137

asked Apr 4, 2018 at 5:09

 **Lifu Huang**
12k 14 56 77




5 Answers

Sorted by: Highest score (default) 


236

`torch.view` has existed for a long time. It will return a tensor with the new shape. The returned tensor will share the underlying data with the original tensor. See the [documentation here](#).

On the other hand, it seems that `torch.reshape` [has been introduced recently in version 0.4](#). According to the [document](#), this method will

Returns a tensor with the same data and number of elements as input, but with the specified shape. When possible, the returned tensor will be a view of input. Otherwise, it will be a copy. Contiguous inputs and inputs with compatible strides can be reshaped without copying, but you should not depend on the copying vs. viewing behavior.

It means that `torch.reshape` may return a copy or a view of the original tensor. You can not count on that to return a view or a copy. According to the developer:

if you need a copy use `clone()` if you need the same storage use `view()`. The semantics of `reshape()` are that it may or may not share the storage and you don't know beforehand.

Another difference is that `reshape()` can operate on both contiguous and non-contiguous tensor while `view()` can only operate on contiguous tensor. Also see [here](#) about the meaning of `contiguous`.

Share Follow


edited Jul 10, 2019 at 8:17

answered Apr 4, 2018 at 6:35

 **jdhao**
24.3k 18 135 276

69 Maybe emphasizing that `torch.view` can only operate on contiguous tensors, while `torch.reshape` can operate on both might be helpful too. – [p13rr0m](#) Apr 5, 2018 at 9:10

8 @pierrom contiguous here referring to tensors that are stored in contiguous memory or something else? – [gokul_uf](#) Dec 4, 2018 at 15:34

5 @gokul_uf Yes, you can take a look at the answer written here: stackoverflow.com/questions/48915810/pytorch-contiguous – [MBT](#) Dec 5, 2018 at 11:12 

does the phrase "a view of a tensor" mean in pytorch? – [Charlie Parker](#) Jun 29, 2020 at 21:37

1 `view()` can operate on non-contiguous tensors. See my answer for an example. – [Pierre](#) Oct 20, 2021 at 18:47



94



Although both `torch.view` and `torch.reshape` are used to reshape tensors, here are the differences between them.

1. As the name suggests, `torch.view` merely creates a *view* of the original tensor. The new tensor will *always* share its data with the original tensor. This means that if you change the original tensor, the reshaped tensor will change and vice versa.

```
>>> z = torch.zeros(3, 2)
>>> x = z.view(2, 3)
>>> z.fill_(1)
>>> x
tensor([[1., 1., 1.],
        [1., 1., 1.]])
```

2. To ensure that the new tensor always shares its data with the original, `torch.view` imposes some contiguity constraints on the shapes of the two tensors [\[docs\]](#). More often than not this is not a concern, but sometimes `torch.view` throws an error even if the shapes of the two tensors are compatible. Here's a famous counter-example.

```
>>> z = torch.zeros(3, 2)
>>> y = z.t()
>>> y.size()
torch.Size([2, 3])
>>> y.view(6)
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
RuntimeError: invalid argument 2: view size is not compatible with input tensor's
size and stride (at least one dimension spans across two contiguous subspaces).
Call .contiguous() before .view().
```

3. `torch.reshape` doesn't impose any contiguity constraints, but also doesn't guarantee data sharing. The new tensor may be a view of the original tensor, or it may be a new tensor altogether.

```
>>> z = torch.zeros(3, 2)
>>> y = z.reshape(6)
>>> x = z.t().reshape(6)
>>> z.fill_(1)
tensor([[1., 1.],
        [1., 1.],
        [1., 1.]])
>>> y
tensor([1., 1., 1., 1., 1., 1.])
>>> x
tensor([0., 0., 0., 0., 0., 0.]])
```

TL;DR:

If you just want to reshape tensors, use `torch.reshape`. If you're also concerned about memory usage and want to ensure that the two tensors share the same data, use `torch.view`.

Share Follow

answered Feb 3, 2019 at 20:49



nikhilweee

3,993 1 18 14

- 3 Maybe it's just me, but I was confused into thinking that contiguity is the deciding factor between when reshape does and does not share data. From my own experiments, it seems that this is not the case. (Your `x` and `y` above are both contiguous). Perhaps this can be clarified? Perhaps a comment on *when* reshape does and does not copy would be helpful? – [RMurphy](#) Mar 18, 2020 at 16:09

`x` and `y` are contiguous but we care here about `z` and `z.t()`. `z` is contiguous so `y` and `z` share the same data. `z` and `z.t()` share the same data, but `z.t()` is not contiguous so `z.t()` and `x` do not share the same data. Therefore `x` and `y` do not share the same data. – [Will](#) May 28 at 7:52



34



`view()` will try to change the shape of the tensor while keeping the underlying data allocation the same, thus data will be shared between the two tensors. `reshape()` will create a new underlying memory allocation if necessary.

Let's create a tensor:

```
a = torch.arange(8).reshape(2, 4)
```

0	1	2	3
4	5	6	7

The memory is allocated like below (it is *C contiguous* i.e. the rows are stored next to each other):

0	1	2	3	4	5	6	7
---	---	---	---	---	---	---	---

`stride()` gives the number of bytes required to go to the next element in each dimension:

```
a.stride()  
(4, 1)
```

We want its shape to become (4, 2), we can use `view`:

```
a.view(4, 2)
```

0	1
2	3
4	5
6	7

The underlying data allocation has not changed, the tensor is still *C contiguous*:

0	1	2	3	4	5	6	7
---	---	---	---	---	---	---	---

```
a.view(4, 2).stride()  
(2, 1)
```

Let's try with `a.t()`. `Transpose()` doesn't modify the underlying memory allocation and therefore `a.t()` is not contiguous.

```
a.t().is_contiguous()  
False
```

0	4
1	5
2	6
3	7

0	1	2	3	4	5	6	7
---	---	---	---	---	---	---	---

Although it is not contiguous, the stride information is sufficient to iterate over the tensor

```
a.t().stride()
(1, 4)
```

view() doesn't work anymore:

```
a.t().view(2, 4)
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
RuntimeError: view size is not compatible with input tensor's size and stride (at least
one dimension spans across two contiguous subspaces). Use .reshape(...) instead.
```

Below is the shape we wanted to obtain by using view(2, 4):



What would the memory allocation look like?



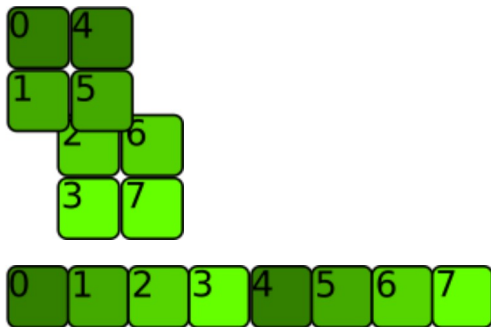
The stride would be something like (4, 2) but we would have to go back to the beginning of the tensor after we reach the end. It doesn't work.

In this case, reshape() would create a new tensor with a different memory allocation to make the transpose contiguous:



Note that we can use view to split the first dimension of the transpose. Unlike what is said in the accepted and other answers, view() can operate on non-contiguous tensors!

```
a.t().view(2, 2, 2)
```



```
a.t().view(2, 2, 2).stride()
(2, 1, 4)
```

[According to the documentation:](#)

For a tensor to be viewed, the new view size must be compatible with its original size and stride, i.e., each new view dimension must either be a subspace of an original dimension, or only span across original dimensions $d, d+1, \dots, d+k$ that satisfy the following contiguity-like condition that $\forall i=d, \dots, d+k-1$, $\text{stride}[i] = \text{stride}[i+1] \times \text{size}[i+1]$

Here that's because the first two dimensions after applying view(2, 2, 2) are subspaces of the transpose's first dimension.

For more information about contiguity have a look at my answer [in this thread](#)

Share Follow

edited Feb 20 at 14:46

answered Oct 19, 2021 at 20:28



Pierre

1,242 12 15

2 The illustration and its color darkness help me understand what `contiguous` means, it means whether indexing all of next number in one row is contiguous or not. BTW, there is a minor typo at `b.t().is_contiguous()` , it might be `a.t().is_contiguous()` , thanks all the same ! – [Wade Wang](#) Nov 6, 2021 at 9:38

Thanks for your comment and for catching the typo! It's now fixed. – [Pierre](#) Nov 6, 2021 at 11:27

"The stride would be something like (4, 2)" - it should be (2, 4), right? – [Albert](#) 2 days ago

"but we would have to go back to the beginning of the tensor after we reach the end" - if there would be a modulo operation (mod the num elems), then such stride should be correct, right? I wonder if it might make sense to allow for that. – [Albert](#) 2 days ago



18



`Tensor.reshape()` is more robust. It will work on any tensor, while `Tensor.view()` works only on tensor `t` where `t.is_contiguous()==True` .

To explain about non-contiguous and contiguous is another story, but you can always make the tensor `t` contiguous if you call `t.contiguous()` and then you can call `view()` without the error.



Share Follow

edited Jul 22, 2021 at 20:50

answered May 3, 2019 at 20:12



prosti

42.7k 15 186 152



0



I would say the answers here are technically correct but there's another reason for existing of `reshape` . `pytorch` is usually considered more convenient than other frameworks because it closer to `python` and `numpy` . It's interesting that the question involves `numpy` .

Let's look into `size` and `shape` in `pytorch` . `size` is a function so you call it like `x.size()` . `shape` in `pytorch` is not a function. In `numpy` you have `shape` and it's *not* a function - you use it `x.shape` . So it's handy to get both of them in `pytorch` . If you came from `numpy` it would be nice to use the same functions.



Share Follow

edited Oct 22, 2021 at 11:51

answered Oct 22, 2021 at 11:45



irudyak

2,281 25 20