Deep learning

1.6. Tensor internals

François Fleuret

https://fleuret.org/dlc/

Nov 29, 2020
A tensor is a view of a [part of a] storage, which is a low-level 1d vector.

```python
>>> x = torch.zeros(2, 4)
>>> x.storage()
 0.0
 0.0
 0.0
 0.0
 0.0
 0.0
 0.0
 0.0
 0.0
[torch.FloatStorage of size 8]
>>> q = x.storage()
>>> q[4] = 1.0
>>> x
  tensor([[ 0.,  0.,  0.,  0.],
           [ 1.,  0.,  0.,  0.]])
```
Multiple tensors can share the same storage. It happens when using operations such as `view()`, `expand()` or `transpose()`.

```python
>>> y = x.view(2, 2, 2)
>>> y
tensor([[ 0.,  0.],
         [ 0.,  0.]],
        [[ 1.,  0.],
         [ 0.,  0.]])
>>> y[1, 1, 0] = 7.0
>>> x
tensor([[ 0.,  0.,  0.,  0.],
         [ 1.,  0.,  7.,  0.]])
>>> y.narrow(0, 1, 1).fill_(3.0)
```
```text
>>> tensor([[ 3.,  3.],
           [ 3.,  3.]])
```
The first coefficient of a tensor is the one at `storage_offset()` in `storage()`.
The first coefficient of a tensor is the one at `storage_offset()` in `storage()`.

Incrementing index $k$ by 1 move by `stride(k)` elements in the storage.
The first coefficient of a tensor is the one at \texttt{storage\_offset()} in \texttt{storage()}.

Incrementing index \texttt{k} by \texttt{1} move by \texttt{stride(k)} elements in the storage.

```python
>>> q = torch.arange(0, 20).storage()
>>> x = torch.empty(0).set_(q, storage_offset = 5, size = (3, 2), stride = (4, 1))
>>> x
  tensor([[ 5.,  6.],
          [ 9., 10.],
          [13., 14.]])
```
The first coefficient of a tensor is the one at `storage_offset()` in `storage()`.

Incrementing index \( k \) by 1 move by `stride(k)` elements in the storage.

```python
>>> q = torch.arange(0, 20).storage()
>>> x = torch.empty(0).set_(q, storage_offset = 5, size = (3, 2), stride = (4, 1))
>>> x
tensor([[ 5.,  6.],
         [ 9., 10.],
         [13., 14.]])
```

\[
q = \begin{bmatrix}
    0 & 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 & 10 & 11 & 12 & 13 & 14 & 15 & 16 & 17 & 18 & 19
\end{bmatrix}
\]
The first coefficient of a tensor is the one at `storage_offset()` in `storage()`.

Incrementing index $k$ by 1 move by `stride(k)` elements in the storage.

```bash
>>> q = torch.arange(0, 20).storage()
>>> x = torch.empty(0).set_(q, storage_offset = 5, size = (3, 2), stride = (4, 1))
>>> x
tensor([[ 5.,  6.],
        [ 9., 10.],
        [13., 14.]])
```

```
```

$q = 
\begin{array}{cccccccccccccccccc}
0 & 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 & 10 & 11 & 12 & 13 & 14 & 15 & 16 & 17 & 18 & 19 \\
\hline
\end{array}
```

$x[0,0]$ $x[0,1]$ $x[1,0]$ $x[1,1]$ $x[2,0]$ $x[2,1]$
The first coefficient of a tensor is the one at `storage_offset()` in `storage()`.

Incrementing index $k$ by 1 move by `stride(k)` elements in the storage.

```python
>>> q = torch.arange(0, 20).storage()
>>> x = torch.empty(0).set_(q, storage_offset = 5, size = (3, 2), stride = (4, 1))
>>> x
tensor([[ 5.,  6.],
        [ 9., 10.],
        [13., 14.]])
```
The first coefficient of a tensor is the one at `storage_offset()` in `storage()`.

Incrementing index \( k \) by 1 move by \( \text{stride}(k) \) elements in the storage.

```
>>> q = torch.arange(0, 20).storage()
>>> x = torch.empty(0).set_(q, storage_offset = 5, size = (3, 2), stride = (4, 1))
>>> x
tensor([[ 5.,  6.],
         [ 9., 10.],
         [13., 14.]])
```
The first coefficient of a tensor is the one at `storage_offset()` in `storage()`.

Incrementing index $k$ by 1 move by `stride(k)` elements in the storage.

```python
>>> q = torch.arange(0, 20).storage()
>>> x = torch.empty(0).set_(q, storage_offset = 5, size = (3, 2), stride = (4, 1))
>>> x
tensor([[ 5.,  6.],
        [ 9., 10.],
        [13., 14.]])
```
We can explicitly create different “views” of the same storage

```python
>>> n = torch.linspace(1, 4, 4)
>>> n
tensor([ 1., 2., 3., 4.])
>>> torch.tensor(0.).set_(n.storage(), 1, (3, 3), (0, 1))
tensor([[ 2., 3., 4.],
        [ 2., 3., 4.],
        [ 2., 3., 4.]])
>>> torch.tensor(0.).set_(n.storage(), 1, (2, 4), (1, 0))
tensor([[ 2., 2., 2., 2.],
        [ 3., 3., 3., 3.]])
```
We can explicitly create different “views” of the same storage

```python
>>> n = torch.linspace(1, 4, 4)
>>> n
tensor([1., 2., 3., 4.])
>>> torch.tensor(0.).set_(n.storage(), 1, (3, 3), (0, 1))
tensor([[2., 3., 4.],
        [2., 3., 4.],
        [2., 3., 4.]])
>>> torch.tensor(0.).set_(n.storage(), 1, (2, 4), (1, 0))
tensor([[2., 2., 2., 2.],
        [3., 3., 3., 3.]])
```

This is in particular how transpositions and broadcasting are implemented.

```python
>>> x = torch.empty(100, 100)
>>> x.stride()
(100, 1)
>>> y = x.t()
>>> y.stride()
(1, 100)
```
This organization explains the following (maybe surprising) error

```python
>>> x = torch.empty(100, 100)
>>> x.t().view(-1)
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()
```

`x.t()` shares `x`'s storage and cannot be “flattened” to 1d.
This organization explains the following (maybe surprising) error

```python
>>> x = torch.empty(100, 100)
>>> x.t().view(-1)
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()
```

`x.t()` shares `x`'s storage and cannot be “flattened” to 1d.

This can be fixed with `contiguous()`, which returns a contiguous version of the tensor, making a copy if needed.

The function `reshape()` combines `view()` and `contiguous()`.
The end