Broadcasting semantics

Many PyTorch operations support NumPy's broadcasting semantics. See https://numpy.org/doc/stable/user/basics.broadcasting.html for details.

In short, if a PyTorch operation supports broadcast, then its Tensor arguments can be automatically expanded to be of equal sizes (without making copies of the data).

General semantics

Two tensors are "broadcastable" if the following rules hold:

- Each tensor has at least one dimension.
- When iterating over the dimension sizes, starting at the trailing dimension, the dimension sizes must either be equal, one of them is 1, or one of them does not exist.

For Example:

```
>>> x=torch.empty(5,7,3)
>>> y=torch.empty(5,7,3)
# same shapes are always broadcastable (i.e. the above rules always hold)

>>> x=torch.empty((0,))
>>> y=torch.empty(2,2)
# x and y are not broadcastable, because x does not have at least 1 dimension

# can line up trailing dimensions
>>> x=torch.empty(5,3,4,1)
>>> y=torch.empty(  3,1,1)
# x and y are broadcastable.
# 1st trailing dimension: both have size 1
# 2nd trailing dimension: y has size 1
# 3rd trailing dimension: x size == y size
# 4th trailing dimension: y dimension doesn't exist

# but:
>>> x=torch.empty(5,2,4,1)
>>> y=torch.empty(  3,1,1)
# x and y are not broadcastable, because in the 3rd trailing dimension 2 != 3
```

If two tensors :attr:`x`, :attr:`y` are "broadcastable", the resulting tensor size is calculated as follows:

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\pytorch-master\docs\source\notes\[pytorch-master][docs][source][notes]broadcasting.rst, line 45); *backlink*

Unknown interpreted text role "attr".

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\pytorch-master\docs\source\notes\[pytorch-master][docs][source][notes]broadcasting.rst, line 45); *backlink*

Unknown interpreted text role "attr".

- If the number of dimensions of :attr:`x` and :attr:`y` are not equal, prepend 1 to the dimensions of the tensor with fewer dimensions to make them equal length.

  System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\pytorch-master\docs\source\notes\[pytorch-master][docs][source][notes]broadcasting.rst, line 48); *backlink*

  Unknown interpreted text role "attr".

  System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\pytorch-master\docs\source\notes\[pytorch-master][docs][source][notes]broadcasting.rst, line 48); *backlink*

  Unknown interpreted text role "attr".

- Then, for each dimension size, the resulting dimension size is the max of the sizes of :attr:`x` and :attr:`y` along that dimension.

  System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\pytorch-master\docs\source\notes\[pytorch-master][docs][source][notes]broadcasting.rst, line 50); *backlink*

  Unknown interpreted text role "attr".

  System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\pytorch-master\docs\source\notes\[pytorch-master][docs][source][notes]broadcasting.rst, line 50); *backlink*

  Unknown interpreted text role "attr".

For Example:

```
# can line up trailing dimensions to make reading easier
>>> x=torch.empty(5,1,4,1)
>>> y=torch.empty(  3,1,1)
>>> (x+y).size()
torch.Size([5, 3, 4, 1])

# but not necessary:
>>> x=torch.empty(1)
>>> y=torch.empty(3,1,7)
>>> (x+y).size()
torch.Size([3, 1, 7])

>>> x=torch.empty(5,2,4,1)
>>> y=torch.empty(3,1,1)
>>> (x+y).size()
RuntimeError: The size of tensor a (2) must match the size of tensor b (3) at non-singleton dimension 1
```

In-place semantics

One complication is that in-place operations do not allow the in-place tensor to change shape as a result of the broadcast.

For Example:

```
>>> x=torch.empty(5,3,4,1)
>>> y=torch.empty(3,1,1)
>>> (x.add_(y)).size()
torch.Size([5, 3, 4, 1])

# but:
>>> x=torch.empty(1,3,1)
>>> y=torch.empty(3,1,7)
>>> (x.add_(y)).size()
RuntimeError: The expanded size of the tensor (1) must match the existing size (7) at non-singleton dimension 2.
```

Backwards compatibility

Prior versions of PyTorch allowed certain pointwise functions to execute on tensors with different shapes, as long as the number of elements in each tensor was equal. The pointwise operation would then be carried out by viewing each tensor as 1-dimensional. PyTorch now supports broadcasting and the "1-dimensional" pointwise behavior is considered deprecated and will generate a Python warning in cases where tensors are not broadcastable, but have the same number of elements.

Note that the introduction of broadcasting can cause backwards incompatible changes in the case where two tensors do not have the same shape, but are broadcastable and have the same number of elements. For Example:

```
>>> torch.add(torch.ones(4,1), torch.randn(4))
```

would previously produce a Tensor with size: torch.Size([4,1]), but now produces a Tensor with size: torch.Size([4,4]). In order to help identify cases in your code where backwards incompatibilities introduced by broadcasting may exist, you may set *torch.utils.backcompat.broadcast_warning.enabled* to *True*, which will generate a python warning in such cases.

For Example:

```
>>> torch.utils.backcompat.broadcast_warning.enabled=True
>>> torch.add(torch.ones(4,1), torch.ones(4))
__main__:1: UserWarning: self and other do not have the same shape, but are broadcastable, and have the same number
Changing behavior in a backwards incompatible manner to broadcasting rather than viewing as 1-dimensional.
```