PyTorch 1.7 adds a torch.fft module similar to NumPy's fft module. Unfortunately, the module's name conflicts with the existing torch.fft() function, so it is not imported by default like most PyTorch modules. This note describes how to use the new torch.fft module in PyTorch 1.7 and the differences between its functions and PyTorch's older FFT functions. It also offers recommendations for library writers who want to use the new FFT functionality and simultaneously support multiple PyTorch versions (for example, PyTorch 1.6, 1.7, and 1.8).

In PyTorch 1.8 the torch.fft module will be imported by default and the current torch.fft() function (and the torch.Tensor.fft() method) will be removed. PyTorch programs are expected to update to the torch.fft module's functionality.

Using the torch.fft module

Using functions in the torch.fft module in PyTorch 1.7 requires importing it:

```
import torch.fft

t = torch.arange(4)
torch.fft.fft(t)
: tensor([ 6.+0.j, -2.+2.j, -2.+0.j, -2.-2.j])
```

Doing so will clobber the existing torch.fft() function name.

```
import torch

t = torch.randn(4, 3, 2)
fft_result = torch.fft(t, 2)

import torch.fft
torch.fft(t, 2)
: TypeError: 'module' object is not callable
```

The torch.Tensor.fft() method can continue to be used as usual, however:

```
import torch
t = torch.randn(4, 3, 2)
fft_result = torch.fft(t, 2)

import torch.fft
t.fft(2)  # the method is not clobbered
```

Reminder: In PyTorch 1.8 the torch.fft() function and the torch.Tensor.fft() method will be removed and the torch.fft module will be automatically imported.

Differences between the old FFT functions and those in the torch.fft module

Before PyTorch 1.7 there were the following FFT-related functions:

- torch.fft() and torch.ifft()
- torch.rfft() and torch.irfft()
- torch.stft() and torch.istft()

These functions exclusively took and returned float tensors, with complex values represented as a float tensor with dimensions $(\ldots, 2)$, where the last dimension contained the real and imaginary parts, respectively, of an implicit complex number. In PyTorch 1.7, however, torch.fft(), torch.ifft(), torch.rfft(), and torch.irfft() are *deprecated* in favor of the new FFT-related functions in PyTorch's torch.fft module. In addition, torch.stft() and torch.istft() have been updated to be more consistent with Librosa's stft and istft.

Updating PyTorch programs using the four deprecated FFT functions requires:

- using complex tensors (instead of floating point tensors mimicking complex values)
- using functions from the torch.fft module

For example,

```
import torch

# Program snippet using the deprecated torch.fft
t = torch.randn(4, 3, 2)
fft_result = torch.fft(t, 1)  # Note: signal_ndim=1
# Produces a tensor with shape (4, 3, 2) and dtype float32
print(fft_result)
: tensor([[[ 0.9579, -4.2542],
           [ 2.1739, 5.6960],
           ...

# Updated version using a complex tensor
import torch.fft
c = torch.view_as_complex(t)
complex_fft_result = torch.fft.fft(c)
# Produces a tensor with shape (4, 3) and dtype complex64
print(complex_fft_result)
: tensor([[ 0.9579-4.2542j, 2.1739+5.6960j, ...
```

Here a call to torch.fft requires an update to use a complex tensor. In this case the transform is one dimensional and torch.fft.fft() is used, but higher dimensional transforms require a call to torch.fft.fftn():

```
import torch
# Program snippet using the deprecated torch.fft
```

```
t = torch.randn(4, 3, 2)
fft_result = torch.fft(t, 2)  # Note: signal_ndim=2
# Produces a tensor with shape (4, 3, 2) and dtype float32
print(fft_result)
: tensor([[[ 1.4367, -1.1135],
           [-1.9576, -2.0158],
           ...

# Updated version calling torch.fft.fftn
import torch.fft
c = torch.view_as_complex(t)
complex_fft_result = torch.fft.fftn(c)
# Produces a tensor with shape (4, 3) and dtype complex64
print(complex_fft_result)
: tensor([[ 1.4367-1.1135j, -1.9576-2.0158j, ...
```

While the deprecated functions take a signal_ndim argument, the new torch.fft module functions accept a dim argument that specifies the dimensions to transform. To translate from one to the other, if signal_ndim=x then dim should be a tuple with the last x dimensions of the complex input. For example:

```
t = torch.randn((4, 4, 4, 2))
fft_result = torch.fft(t, 3)
# Produces a tensor with shape (4, 4, 4, 2) and dtype float32
print(fft_result)
: tensor([[[[-4.8235e+00, 1.7283e+01], ...

import torch.fft
c = torch.view_as_complex(t)
# signal_ndim=3 so dim includes last 3 dimensions of c
complex_fft_result = torch.fft.fftn(c, dim=(0, 1, 2))
# Produces a tensor with shape (4, 4, 4) and dtype complex64
print(complex_fft_result)
: tensor([[[ -4.8235+1.7283e+01j, ...
```

If the original call was torch.fft(t, 1) then the updated call would be torch.fft.fftn(c, dim=2). Note that torch.fft.fft() is just sugar for the more general torch.fft.fftn().

Note that while torch.fft() was limited to one, two, or three dimensional FFT transforms, torch.fft.fftn() can transform any number of dimensions. The torch.fft module also contains the new torch.fft.hfft() and torch.fft.ihfft() functions, with more FFT-related functionality coming in PyTorch 1.8.

Recommendations for library writers

Libraries that want to use PyTorch's new FFT functionality should update to use complex tensors or, since complex tensors are still in beta, use the view_as_complex function to transform float tensors into complex inputs and

the view_as_real function to transform complex outputs into float tensors. They must also be careful not to import the torch.fft module in PyTorch 1.7, since the import affects user code, and to be aware that the torch.fft() function may not be available (since a user may have imported the torch.fft module and clobbered it). The rest of this section discusses how libraries can deal with these issues.

Using the new torch.fft module without importing it

Libraries that want to use the torch.fft functionality should not import it. Instead, they can use one of two approaches:

- Only use the new torch.fft module functions if the torch.fft module is imported.
- Require a user load the torch.fft module before loading the library.

For example, if the new functionality is only used when torch.fft is available:

```
import sys

if "torch.fft" not in sys.modules:
    # calls torch.fft
else:
    # calls torch.fft.fft
```

This approach will work in all versions of PyTorch. In PyTorch 1.7, however, the first code path will throw deprecation warnings. These warnings can be caught and then filtered:

```
import sys
import warnings

if "torch.fft" not in sys.modules:
    with warnings.catch_warnings(record=True) as w:
        # calls torch.fft
else:
    # calls torch.fft.fft
```

Alternatively, libraries can require the module be loaded on versions of PyTorch where it's available:

```
import sys
import warnings

# Acquires and parses the PyTorch version
split_version = torch.__version__.split('.')
major_version = int(split_version[0])
minor_version = int(split_version[1])
if major_version > 1 or (major_version == 1 and minor_version >= 7):
    if "torch.fft" not in sys.modules:
```

```
        raise RuntimeError("torch.fft module available but not imported")

if "torch.fft" not in sys.modules:
    # calls torch.fft, no warning is thrown because version must be < 1.7
else:
    # calls torch.fft.fft
```

Users will have to import the torch.fft module when using the library with PyTorch 1.7, but in PyTorch 1.8 and later the library will work without the user importing the module. Both approaches will work with all versions of PyTorch.

Using the old torch.fft function

If a user imports the torch.fft module then the torch.fft() function's name will be clobbered. Libraries that want to continue using the deprecated torch.fft() function in PyTorch 1.7 should use the method torch.Tensor.fft() instead:

```
t = torch.randn(4, 3, 2)

# may not work in PyTorch 1.7 and will not work in PyTorch 1.8
fft_result = torch.fft(t, 1)

# always works in PyTorch 1.7, will not work in PyTorch 1.8
fft_method_result = t.fft(1)
```

This will work in PyTorch 1.6, throw a deprecation warning in PyTorch 1.7, and will not work in PyTorch 1.8 when the torch.Tensor.fft() method is removed. Libraries should, instead, adopt one of the approaches above which will work with all PyTorch versions.