Serialization semantics

This note describes how you can save and load PyTorch tensors and module states in Python, and how to serialize Python modules so they can be loaded in C++.

Table of Contents

Saving and loading tensors

:func:`torch.save` and :func:`torch.load` let you easily save and load tensors:

System Message: ERROR/3 (`D:\onboarding-resources\sample-onboarding-resources\pytorch-master\docs\source\notes\(pytorch-master) (docs) (source) (notes) serialization.rst`, line 15); ***backlink***

Unknown interpreted text role "func".

System Message: ERROR/3 (`D:\onboarding-resources\sample-onboarding-resources\pytorch-master\docs\source\notes\(pytorch-master) (docs) (source) (notes) serialization.rst`, line 15); ***backlink***

Unknown interpreted text role "func".

```
>>> t = torch.tensor([1., 2.])
>>> torch.save(t, 'tensor.pt')
>>> torch.load('tensor.pt')
tensor([1., 2.])
```

By convention, PyTorch files are typically written with a â€˜.ptâ€™ or â€˜.pthâ€™ extension.

:func:`torch.save` and :func:`torch.load` use Pythonâ€™s pickle by default, so you can also save multiple tensors as part of Python objects like tuples, lists, and dicts:

System Message: ERROR/3 (`D:\onboarding-resources\sample-onboarding-resources\pytorch-master\docs\source\notes\(pytorch-master) (docs) (source) (notes) serialization.rst`, line 26); ***backlink***

Unknown interpreted text role "func".

System Message: ERROR/3 (`D:\onboarding-resources\sample-onboarding-resources\pytorch-master\docs\source\notes\(pytorch-master) (docs) (source) (notes) serialization.rst`, line 26); ***backlink***

Unknown interpreted text role "func".

```
>>> d = {'a': torch.tensor([1., 2.]), 'b': torch.tensor([3., 4.])}
>>> torch.save(d, 'tensor_dict.pt')
>>> torch.load('tensor_dict.pt')
{'a': tensor([1., 2.]), 'b': tensor([3., 4.])}
```

Custom data structures that include PyTorch tensors can also be saved if the data structure is pickle-able.

Saving and loading tensors preserves views

Saving tensors preserves their view relationships:

```
>>> numbers = torch.arange(1, 10)
>>> evens = numbers[1::2]
>>> torch.save([numbers, evens], 'tensors.pt')
>>> loaded_numbers, loaded_evens = torch.load('tensors.pt')
>>> loaded_evens *= 2
>>> loaded_numbers
tensor([ 1,  4,  3,  8,  5, 12,  7, 16,  9])
```

Behind the scenes, these tensors share the same "storage." See Tensor Views for more on views and storage.

When PyTorch saves tensors it saves their storage objects and tensor metadata separately. This is an implementation detail that may change in the future, but it typically saves space and lets PyTorch easily reconstruct the view relationships between the loaded tensors. In the above snippet, for example, only a single storage is written to 'tensors.pt'.

In some cases, however, saving the current storage objects may be unnecessary and create prohibitively large files. In the following snippet a storage much larger than the saved tensor is written to a file:

```
>>> large = torch.arange(1, 1000)
>>> small = large[0:5]
>>> torch.save(small, 'small.pt')
>>> loaded_small = torch.load('small.pt')
>>> loaded_small.storage().size()
999
```

Instead of saving only the five values in the *small* tensor to 'small.pt,' the 999 values in the storage it shares with *large* were saved and loaded.

When saving tensors with fewer elements than their storage objects, the size of the saved file can be reduced by first cloning the tensors. Cloning a tensor produces a new tensor with a new storage object containing only the values in the tensor:

```
>>> large = torch.arange(1, 1000)
>>> small = large[0:5]
>>> torch.save(small.clone(), 'small.pt')  # saves a clone of small
>>> loaded_small = torch.load('small.pt')
>>> loaded_small.storage().size()
5
```

Since the cloned tensors are independent of each other, however, they have none of the view relationships the original tensors did. If both file size and view relationships are important when saving tensors smaller than their storage objects, then care must be taken to construct new tensors that minimize the size of their storage objects but still have the desired view relationships before saving.

Saving and loading torch.nn.Modules

See also: Tutorial: Saving and loading modules

In PyTorch, a moduleâ€™s state is frequently serialized using a â€˜state dict.â€™ A moduleâ€™s state dict contains all of its parameters and persistent buffers:

```
>>> bn = torch.nn.BatchNorm1d(3, track_running_stats=True)
>>> list(bn.named_parameters())
[('weight', Parameter containing: tensor([1., 1., 1.], requires_grad=True)),
 ('bias', Parameter containing: tensor([0., 0., 0.], requires_grad=True))]

>>> list(bn.named_buffers())
[('running_mean', tensor([0., 0., 0.])),
 ('running_var', tensor([1., 1., 1.])),
 ('num_batches_tracked', tensor(0))]

>>> bn.state_dict()
OrderedDict([('weight', tensor([1., 1., 1.])),
             ('bias', tensor([0., 0., 0.])),
             ('running_mean', tensor([0., 0., 0.])),
             ('running_var', tensor([1., 1., 1.])),
             ('num_batches_tracked', tensor(0))])
```

Instead of saving a module directly, for compatibility reasons it is recommended to instead save only its state dict. Python modules even have a function, :meth:`~torch.nn.Module.load_state_dict`, to restore their states from a state dict:

System Message: ERROR/3 (`D:\onboarding-resources\sample-onboarding-resources\pytorch-master\docs\source\notes\(pytorch-master)(docs)(source)(notes)serialization.rst`, **line 133**); ***backlink***

Unknown interpreted text role "meth".

```
>>> torch.save(bn.state_dict(), 'bn.pt')
>>> bn_state_dict = torch.load('bn.pt')
>>> new_bn = torch.nn.BatchNorm1d(3, track_running_stats=True)
>>> new_bn.load_state_dict(bn_state_dict)
<All keys matched successfully>
```

Note that the state dict is first loaded from its file with :func:`torch.load` and the state then restored with

:meth:`~torch.nn.Module.load_state_dict`.

System Message: ERROR/3 (`D:\onboarding-resources\sample-onboarding-resources\pytorch-master\docs\source\notes\(pytorch-master) (docs) (source) (notes)serialization.rst`, line 145); ***backlink***

Unknown interpreted text role "func".

System Message: ERROR/3 (`D:\onboarding-resources\sample-onboarding-resources\pytorch-master\docs\source\notes\(pytorch-master) (docs) (source) (notes)serialization.rst`, line 145); ***backlink***

Unknown interpreted text role "meth".

Even custom modules and modules containing other modules have state dicts and can use this pattern:

```
# A module with two linear layers
>>> class MyModule(torch.nn.Module):
      def __init__(self):
        super(MyModule, self).__init__()
        self.l0 = torch.nn.Linear(4, 2)
        self.l1 = torch.nn.Linear(2, 1)

      def forward(self, input):
        out0 = self.l0(input)
        out0_relu = torch.nn.functional.relu(out0)
        return self.l1(out0_relu)

>>> m = MyModule()
>>> m.state_dict()
OrderedDict([('l0.weight', tensor([[ 0.1400, 0.4563, -0.0271, -0.4406],
                                   [-0.3289, 0.2827, 0.4588, 0.2031]])),
             ('l0.bias', tensor([ 0.0300, -0.1316])),
             ('l1.weight', tensor([[0.6533, 0.3413]])),
             ('l1.bias', tensor([-0.1112]))])

>>> torch.save(m.state_dict(), 'mymodule.pt')
>>> m_state_dict = torch.load('mymodule.pt')
>>> new_m = MyModule()
>>> new_m.load_state_dict(m_state_dict)
<All keys matched successfully>
```

Serializing torch.nn.Modules and loading them in C++

See also: Tutorial: Loading a TorchScript Model in C++

ScriptModules can be serialized as a TorchScript program and loaded using :func:`torch.jit.load`. This serialization encodes all the modulesâ€™ methods, submodules, parameters, and attributes, and it allows the serialized program to be loaded in C++ (i.e. without Python).

System Message: ERROR/3 (`D:\onboarding-resources\sample-onboarding-resources\pytorch-master\docs\source\notes\(pytorch-master) (docs) (source) (notes)serialization.rst`, line 186); ***backlink***

Unknown interpreted text role "func".

The distinction between :func:`torch.jit.save` and :func:`torch.save` may not be immediately clear. :func:`torch.save` saves Python objects with pickle. This is especially useful for prototyping, researching, and training. :func:`torch.jit.save`, on the other hand, serializes ScriptModules to a format that can be loaded in Python or C++. This is useful when saving and loading C++ modules or for running modules trained in Python with C++, a common practice when deploying PyTorch models.

System Message: ERROR/3 (`D:\onboarding-resources\sample-onboarding-resources\pytorch-master\docs\source\notes\(pytorch-master) (docs) (source) (notes)serialization.rst`, line 192); ***backlink***

Unknown interpreted text role "func".

System Message: ERROR/3 (`D:\onboarding-resources\sample-onboarding-resources\pytorch-master\docs\source\notes\(pytorch-master) (docs) (source) (notes)serialization.rst`, line 192); ***backlink***

Unknown interpreted text role "func".

System Message: ERROR/3 (`D:\onboarding-resources\sample-onboarding-resources\pytorch-master\docs\source\notes\(pytorch-master) (docs) (source) (notes) serialization.rst`, line 192); *backlink*

Unknown interpreted text role "func".

System Message: ERROR/3 (`D:\onboarding-resources\sample-onboarding-resources\pytorch-master\docs\source\notes\(pytorch-master) (docs) (source) (notes) serialization.rst`, line 192); *backlink*

Unknown interpreted text role "func".

To script, serialize and load a module in Python:

```
>>> scripted_module = torch.jit.script(MyModule())
>>> torch.jit.save(scripted_module, 'mymodule.pt')
>>> torch.jit.load('mymodule.pt')
RecursiveScriptModule( original_name=MyModule
                       (l0): RecursiveScriptModule(original_name=Linear)
                       (l1): RecursiveScriptModule(original_name=Linear) )
```

Traced modules can also be saved with :func:`torch.jit.save`, with the caveat that only the traced code path is serialized. The following example demonstrates this:

System Message: ERROR/3 (`D:\onboarding-resources\sample-onboarding-resources\pytorch-master\docs\source\notes\(pytorch-master) (docs) (source) (notes) serialization.rst`, line 212); *backlink*

Unknown interpreted text role "func".

```
# A module with control flow
>>> class ControlFlowModule(torch.nn.Module):
      def __init__(self):
        super(ControlFlowModule, self).__init__()
        self.l0 = torch.nn.Linear(4, 2)
        self.l1 = torch.nn.Linear(2, 1)

      def forward(self, input):
        if input.dim() > 1:
        return torch.tensor(0)

        out0 = self.l0(input)
        out0_relu = torch.nn.functional.relu(out0)
        return self.l1(out0_relu)

>>> traced_module = torch.jit.trace(ControlFlowModule(), torch.randn(4))
>>> torch.jit.save(traced_module, 'controlflowmodule_traced.pt')
>>> loaded = torch.jit.load('controlflowmodule_traced.pt')
>>> loaded(torch.randn(2, 4)))
tensor([[-0.1571], [-0.3793]], grad_fn=<AddBackward0>)

>>> scripted_module = torch.jit.script(ControlFlowModule(), torch.randn(4))
>>> torch.jit.save(scripted_module, 'controlflowmodule_scripted.pt')
>>> loaded = torch.jit.load('controlflowmodule_scripted.pt')
>> loaded(torch.randn(2, 4))
tensor(0)
```

The above module has an if statement that is not triggered by the traced inputs, and so is not part of the traced module and not serialized with it. The scripted module, however, contains the if statement and is serialized with it. See the TorchScript documentation for more on scripting and tracing.

Finally, to load the module in C++:

```
>>> torch::jit::script::Module module;
>>> module = torch::jit::load('controlflowmodule_scripted.pt');
```

See the PyTorch C++ API documentation for details about how to use PyTorch modules in C++.

Saving and loading ScriptModules across PyTorch versions

The PyTorch Team recommends saving and loading modules with the same version of PyTorch. Older versions of PyTorch may not support newer modules, and newer versions may have removed or modified older behavior. These changes are explicitly described in

PyTorchâ€™s release notes, and modules relying on functionality that has changed may need to be updated to continue working properly. In limited cases, detailed below, PyTorch will preserve the historic behavior of serialized ScriptModules so they do not require an update.

torch.div performing integer division

In PyTorch 1.5 and earlier :func:`torch.div` would perform floor division when given two integer inputs:

System Message: ERROR/3 (`D:\onboarding-resources\sample-onboarding-resources\pytorch-master\docs\source\notes\(pytorch-master)(docs)(source)(notes)serialization.rst`, line 279); ***backlink***

Unknown interpreted text role "func".

```
# PyTorch 1.5 (and earlier)
>>> b = torch.tensor(3)
>>> a / b
tensor(1)
```

In PyTorch 1.7, however, :func:`torch.div` will always perform a true division of its inputs, just like division in Python 3:

System Message: ERROR/3 (`D:\onboarding-resources\sample-onboarding-resources\pytorch-master\docs\source\notes\(pytorch-master)(docs)(source)(notes)serialization.rst`, line 289); ***backlink***

Unknown interpreted text role "func".

```
# PyTorch 1.7
>>> a = torch.tensor(5)
>>> b = torch.tensor(3)
>>> a / b
tensor(1.6667)
```

The behavior of :func:`torch.div` is preserved in serialized ScriptModules. That is, ScriptModules serialized with versions of PyTorch before 1.6 will continue to see :func:`torch.div` perform floor division when given two integer inputs even when loaded with newer versions of PyTorch. ScriptModules using :func:`torch.div` and serialized on PyTorch 1.6 and later cannot be loaded in earlier versions of PyTorch, however, since those earlier versions do not understand the new behavior.

System Message: ERROR/3 (`D:\onboarding-resources\sample-onboarding-resources\pytorch-master\docs\source\notes\(pytorch-master)(docs)(source)(notes)serialization.rst`, line 300); ***backlink***

Unknown interpreted text role "func".

System Message: ERROR/3 (`D:\onboarding-resources\sample-onboarding-resources\pytorch-master\docs\source\notes\(pytorch-master)(docs)(source)(notes)serialization.rst`, line 300); ***backlink***

Unknown interpreted text role "func".

System Message: ERROR/3 (`D:\onboarding-resources\sample-onboarding-resources\pytorch-master\docs\source\notes\(pytorch-master)(docs)(source)(notes)serialization.rst`, line 300); ***backlink***

Unknown interpreted text role "func".

torch.full always inferring a float dtype

In PyTorch 1.5 and earlier :func:`torch.full` always returned a float tensor, regardless of the fill value itâ€™s given:

System Message: ERROR/3 (`D:\onboarding-resources\sample-onboarding-resources\pytorch-master\docs\source\notes\(pytorch-master)(docs)(source)(notes)serialization.rst`, line 310); ***backlink***

Unknown interpreted text role "func".

```
# PyTorch 1.5 and earlier
>>> torch.full((3,), 1)  # Note the integer fill value...
tensor([1., 1., 1.])     # ...but float tensor!
```

In PyTorch 1.7, however, :func:`torch.full` will infer the returned tensorâ€™s dtype from the fill value:

System Message: ERROR/3 (`D:\onboarding-resources\sample-onboarding-resources\pytorch-master\docs\source\notes\(pytorch-master) (docs) (source) (notes) serialization.rst`, line 319); ***backlink***

Unknown interpreted text role "func".

```
# PyTorch 1.7
>>> torch.full((3,), 1)
tensor([1, 1, 1])

>>> torch.full((3,), True)
tensor([True, True, True])

>>> torch.full((3,), 1.)
tensor([1., 1., 1.])

>>> torch.full((3,), 1 + 1j)
tensor([1.+1.j, 1.+1.j, 1.+1.j])
```

The behavior of :func:`torch.full` is preserved in serialized ScriptModules. That is, ScriptModules serialized with versions of PyTorch before 1.6 will continue to see torch.full return float tensors by default, even when given bool or integer fill values. ScriptModules using :func:`torch.full` and serialized on PyTorch 1.6 and later cannot be loaded in earlier versions of PyTorch, however, since those earlier versions do not understand the new behavior.

System Message: ERROR/3 (`D:\onboarding-resources\sample-onboarding-resources\pytorch-master\docs\source\notes\(pytorch-master) (docs) (source) (notes) serialization.rst`, line 337); ***backlink***

Unknown interpreted text role "func".

System Message: ERROR/3 (`D:\onboarding-resources\sample-onboarding-resources\pytorch-master\docs\source\notes\(pytorch-master) (docs) (source) (notes) serialization.rst`, line 337); ***backlink***

Unknown interpreted text role "func".