torch.nested

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\pytorch-master\docs\source\(pytorch-master)(docs)(source)nested.rst, line 4)

Unknown directive type "automodule".

```
.. automodule:: torch.nested
```

Introduction

Warning

The PyTorch API of nested tensors is in prototype stage and will change in the near future.

Warning

torch.NestedTensor currently does not support autograd. It needs to be used in the context of torch.inference_mode().

NestedTensor allows the user to pack a list of Tensors into a single, efficient datastructure.

The only constraint on the input Tensors is that their dimension must match.

This enables more efficient metadata representations and operator coverage.

Construction is straightforward and involves passing a list of Tensors to the constructor.

```
>>> a, b = torch.arange(3), torch.arange(5) + 3
>>> a
tensor([0, 1, 2])
>>> b
tensor([3, 4, 5, 6, 7])
>>> nt = torch.nested_tensor([a, b])
>>> nt
nested_tensor([
  tensor([0, 1, 2]),
    tensor([3, 4, 5, 6, 7])
    ])
```

Data type and device can be chosen via the usual keyword arguments

```
>>> nt = torch.nested_tensor([a, b], dtype=torch.float32, device="cuda")
>>> nt
nested_tensor([
  tensor([0., 1., 2.], device='cuda:0'),
  tensor([3., 4., 5., 6., 7.], device='cuda:0')
])
```

Operator coverage

We are currently on our path to wholesale extend operator coverage guided by specific ML use cases.

Operator coverage thus is currently very limited and only unbind is supported.

```
>>> nt = torch.nested_tensor([a, b], dtype=torch.float32, device="cuda")
>>> nt
nested_tensor([
  tensor([0., 1., 2.], device='cuda:0'),
  tensor([3., 4., 5., 6., 7.], device='cuda:0')
])
>>> nt.unbind()
[tensor([0., 1., 2.], device='cuda:0'), tensor([3., 4., 5., 6., 7.], device='cuda:0')]
```