The purpose of this style guide is to provide guidance for writing `torch.nn` module documentation. It is purposefully strongly opinionated to keep documentation across modules consistent and readable. It describes which sections should be present for each module, as well as formatting details that should always be followed.

Module Docstring Formatting

Module docstrings should match the following format. Note that Google-style doc strings are utilized throughout the Python source code.

```
r"""
Applies a linear transformation to the incoming data.

.. math::
    y = xA^T + b

where :math:`x` is the input, :math:`A` is the ``weight`` parameter, :math:`b` is the
optional ``bias`` parameter, and :math:`y` is the output.

Note:
    This module supports :ref:`TensorFloat32<tf32_on_ampere>`.

Warning:
    Use of this module will melt your hard drive!

Args:
    in_features (int): The size of each input sample.
    out_features (int): The size of each output sample.
    bias (bool, optional): If set to ``False``, the layer will not learn an
        additive bias. Default: ``True``.

Shape:
    - Input: :math:`(*, H_{in})`, where :math:`*` represents any number of
      dimensions (including none) and :math:`H_{in} = \text{in\_features}`.
    - Output: :math:`(*, H_{out})`, where all but the last dimension
      match the input shape and :math:`H_{out} = \text{out\_features}`.

Attributes:
    weight: The learnable weights of the module of shape :math:`(H_{out}, H_{in})`, where
            :math:`H_{in} = \text{in\_features}` and :math:`H_{out} = \text{out\_features}`.
            The values are initialized from :math:`\mathcal{U}(-\sqrt{k}, \sqrt{k})`, where
            :math:`k = \frac{1}{\text{in1\_features}}`.
    bias: The learnable bias of the module of shape :math:`(H_{out})`. Only present when
          :attr:`bias` is ``True``. The values are initialized from
          :math:`\mathcal{U}(-\sqrt{k}, \sqrt{k})`, where :math:`k = \frac{1}{H_{in}}`.
```

```
Examples::
    >>> m = nn.Linear(20, 30)
    >>> input = torch.randn(128, 20)
    >>> output = m(input)
    >>> print(output.size())
    torch.Size([128, 30])

    >>> # Example of creating a Linear layer with no bias.
    >>> m = nn.Linear(3, 3, bias=False)
    >>> input = torch.randn(10, 3)
    >>> output = m(input)
    >>> print(output.size())
    torch.Size([10, 3])
"""
```

As shown above, the docstring for every module provided by `torch.nn` should include a thorough description of the module's computation (supporting math / references can be helpful) as well as the following sections in order:

- **Args**
- **Shape**
- **Attributes** (only if the module has learnable parameters)
- **Examples**

Details on what is expected for these sections are described below.

Args

This section describes the constructor arguments accepted by the module.

```
Args:
    in_features (int): The size of each input sample.
    out_features (int): The size of each output sample.
    bias (bool, optional): If set to ``False``, the layer will not learn an
        additive bias. Default: ``True``.
```

- Each constructor arg should be described and the supported type(s) should be included.
- Descriptions should be capitalized and end with a period.
- If an arg is optional, this should be indicated and its default value should be documented.
- Python values should be surround by double tick-marks (e.g. `False`).

Shape

This section describes accepted input tensors shapes and returned output tensor shapes.

```
Shape:
    - Input: :math:`(*, H_{in})`, where :math:`*` represents any number of
      dimensions (including none) and :math:`H_{in} = \text{in\_features}`.
    - Output: :math:`(*, H_{out})`, where all but the last dimension
      match the input shape and :math:`H_{out} = \text{out\_features}`.
```

- The shapes themselves should be tuples formatted with `:math:`(...)``.
- A 0D tensor is known as a "scalar tensor" and its shape can be denoted with `:math:`()``.
- Note that the formatting for this section is rendered differently from that of `Args` by Sphinx / napoleon. This is why list items must be explicitly tagged with `-` for this section.
- The following symbols should be used to represent dimensions consistently:
  - `C` - number of channels or number of classes
  - `D` - depth dimension
  - `E` - embedding dimension
  - `H` - height dimension (for 2d modules) or hidden state dimension (for modules have hidden state)
  - `L` - length dimension (for 1d modules)
  - `N` - batch dimension
  - `S` - source sequence dimension (for modules that operate on sequences)
  - `T` - target sequence dimension (for modules that operate on sequences)
  - `W` - width dimension (for 2d modules)
  - `*` - can be used to represent a variable number of dimensions. The first usage of this within the shape section should be documented to clearly indicate whether the `*` represents "one or more dimensions" or "any number of dimensions including none".
- Other letters are acceptable as well, but what they represent should always be explained.
- Subscripts can be useful for clarity (e.g. `:math:`(N, H_{in})``).
- Each sentence should begin with the supported shape(s) and describe what each dimension represents.

Attributes

If a module can have learnable parameters, this section is used to describe them; it should be left out if the module never has learnable parameters.

```
Attributes:
    weight: The learnable weights of the module of shape :math:`(H_{out}, H_{in})`, where
            :math:`H_{in} = \text{in\_features}` and :math:`H_{out} = \text{out\_features}`.
            The values are initialized from :math:`\mathcal{U}(-\sqrt{k}, \sqrt{k})`, where
            :math:`k = \frac{1}{\text{in1\_features}}`.
    bias: The learnable bias of the module of shape :math:`(H_{out})`. Only present when
          :attr:`bias` is ``True``. The values are initialized from
          :math:`\mathcal{U}(-\sqrt{k}, \sqrt{k})`, where :math:`k = \frac{1}{H_{in}}`.
```

- Each sentence should describe a learnable parameter, including its shape and default initialization scheme.
- Each sentence should end with a period.
- If a learnable parameter is only present for a particular arg setting, that should be made clear in the description.

Examples

This section is used to demonstrate usage of the module through example code.

```
Examples::
    >>> m = nn.Linear(20, 30)
    >>> input = torch.randn(128, 20)
    >>> output = m(input)
    >>> print(output.size())
    torch.Size([128, 30])

    >>> # Example of creating a Linear layer with no bias.
    >>> m = nn.Linear(3, 3, bias=False)
    >>> input = torch.randn(10, 3)
    >>> output = m(input)
    >>> print(output.size())
    torch.Size([10, 3])
```

- Examples should contain valid, runnable code + expected outputs.
- Examples should demonstrate instantiating a module and calling it on at least one set of inputs.
- Lines beginning with `>>>` represent code that is to be run and other lines display the expected output.
- If there are multiple types of inputs that a module works with (e.g. 1D vs. 2D inputs in different formats), consider demonstrating each of these with an example. Each example should be separated by a newline and comments should be used to describe interesting cases.
- Note the two colons in `Examples::`. This is preferred because the example section is treated differently than other sections by Sphinx / napoleon and one colon will be removed in the rendered form.
- For consistency, this section should always be named `Examples` and not `Example`, even if only one example is present.

Other Sections / Formatting

- For math formulas: use `.. math::`.
  - Used to describe computation being performed by the module.
- For warnings: use `Warning:` (don't use `.. warning::`).
  - Used to describe possible pitfalls or caveats to using the module.
- For notes: use `Note:` (don't use `.. note::`).
  - Used for other, non-critical information about the module.

- For referencing other modules: use e.g. `:class:`~torch.nn.Unfold``.
- For referencing other functions: use e.g. `:func:`torch.nn.functional.pad()``.
- For external references: use links.
  - Used to reference relevant papers or other supporting information.
- For images: use `.. image:: <relative_path>`.
- Don't specify a `Returns` section, as it contains redundant information that should be covered by other sections.