The goal of this lab is to give first-hand experience with defining and testing a new module within PyTorch.

The deliverable will be a PR (that won't be merged into master) containing all the code for the different sections below. You should add both your mentor as well as @jbschlosser as a reviewer for this PR.

Goal

For this lab, we are going to create a module named `nn.Bias` that simply adds a learnable vector parameter to the last dimension of its input (i.e. a bias-only analogue of PyTorch's [Linear] module). The module should accept a constructor argument named `num_features` that defines the size of its vector parameter, and the vector should be randomly initialized from a standard normal distribution. Example usage for this module should look like:

```
import torch
from torch import nn

m = nn.Bias(num_features=5)
input = torch.randn(10, 5)
output = m(input)  # output should be equal to input + module's bias parameter
```

Components

While this is not the case for all modules, it is common for the "meat" of a module to be defined as a single PyTorch operator. On top of this, there are generally 4 wrappers defined that utilize the underlying operator and provide a nice UX for the user, whether they are using the Python or C++ APIs:

- Python API functional form - calls the underlying operator through Python bindings
- Python API module class - calls the Python API functional form
- C++ API functional form - calls the underlying operator
- C++ API module class - calls the C++ API functional form

You should define a PyTorch operator for the "meat" of the module and implement each of the wrappers above.

Relevant Files

- `aten/src/ATen/native/native_functions.yaml` - add the definition for the underlying PyTorch operator here (see [[Tensor-and-Operator-Basics]])
- `aten/src/ATen/native/Linear.cpp` - add core logic for the operator here
- `torch/nn/modules/linear.py` - add the Python module class definition here
- `torch/nn/modules/__init__.py` - add the module class name to the list in `__all__`
- `torch/nn/functional.py` - add the Python functional form of the module here
- `torch/csrc/api/include/torch/nn/functional/linear.h` - add the C++ functional form here
- `torch/csrc/api/include/torch/nn/modules/linear.h` / `torch/csrc/api/src/nn/modules/linear.cpp` - add the C++ module class definitions here
- `torch/csrc/api/include/torch/nn/options/linear.h` / `torch/csrc/api/src/nn/options/linear.cpp` - add options for the module here (i.e. `num_features`)
- `torch/testing/_internal/common_nn.py` - add a `dict` entry to the `module_tests` list defining how to generate generic tests for the new module (note: this part is in the process of being improved)

- `test/test_nn.py` - add any module functionality tests here as functions on `TestNN` or `TestNNDeviceType` (for device-specific tests)
- `test/cpp/api/functional.cpp` - add C++ API functional tests here
- `test/cpp/api/modules.cpp` - add C++ API module tests here
- `test/cpp_api_parity/parity-tracker.md` - add module and functional entries to the parity tracker to check C++ vs. Python behavior (used by `build/bin/test_api`, which runs the C++ module and functional tests)
- `torch/overrides.py` - add override lambda here (used by `test/test_overrides.py`)
- `torch/nn/functional.pyi.in` - add skeleton signature here (used by mypy)
- ??? - other updates may be needed to pass all CI checks & tests