

Module 0.2 - Fundamentals

Module 0.2

Fundamentals

Today's Class

- Intro: Module 0
- Development Setup
- Property Testing
- Functional Python

The Guidebook

MiniTorch

Full description of the material

Module 0: Fundamentals

Learning Goals:

- Setup
- Testing
- Modules
- Visualization
- No ML yet! We'll get to it.

Code Setup: Interactive

GitHub

- <http://github.com/minitorch/Module-0>

Graduating to Code

Why not notebooks?

- Style
- Version Control
- Testing

Base Repo Template

- Each repo starts with a template
- <https://github.com/minitorch/Module-0>

Tour of Repo

- `minitorch/`
- `tests/`
- `project/`

Recommendations

- Development Setup
- Github Tutorials
- Speed of Debugging

VS Code

- Popular choice for the class
- Test
- Debugging

Contributing Guidelines

Contributing

Torch Contrib

- Style, guidelines, typing, etc.

Precommit

Command to run before commit.

```
>>> pre-commit run --all
```


Consistent Styling

Standardized formatting

```
>>> black minitorch/ tests/ project/
```


Linting

Checks for any style or documentation errors

```
>>> flake8 minitorch/ tests/ project/
```


Continuous Integration

Runs behind the scenes on every commit.

Torch CI

The screenshot displays the GitHub Actions interface for a workflow named 'GitHub Classroom Autograding Workflow'. The workflow is triggered on a push to the 'master' branch by user 'e972abe'. The 'Actions' tab is selected, showing a list of workflow runs. The most recent run, 'Autograding', is highlighted in blue and shows a failed status. The workflow run details are expanded, showing a log of the execution steps. The steps are: 'Set up job' (2s), 'Run actions/checkout@v2' (1s), and 'Run education/autograding@beta' (17s). The third step failed with an error. The log output for the failed step is as follows:

```
1 ▶ Run education/autograding@beta
4   Preparing autograding
5   Preparing workspace
6   Reading autograding test configuration
7   Running tests
8   Running Run Gradle
9   ##[error]Error: /home/runner/work/assignment-3-d12/assignment-3-d12/src/main/java/com/example/project/Calculator.java:17: error: missing return statement
10      }
11      ^
12  1 error
13
14  FAILURE: Build failed with an exception.
15
16  * What went wrong:
17  Execution failed for task ':compileJava'.
18  > Compilation failed; see the compiler error output for details.
```


Documentation

Doc style - Google

```
def index(ls, i):  
    """  
    List indexing.  
  
    Args:  
        ls: A list of any type.  
        i: An index into the list  
  
    Returns:  
        Value at ls[i].  
    """  
    ...
```


Type Checks

Modern Python support static type checks

```
>>> mypy
```

```
def mul(x: float, y: float) -> float:  
    ...
```


Type Checks

Compound types

```
from typing import Iterable

def negList(ls: Iterable[float]) -> Iterable[float]:
    ...
```


Testing

PyTorch Testing

PyTorch Tests

Running Tests

Run tests

```
>>> pytest
```

Or per task

```
>>> pytest -m task0_1
```


PyTest

- Finds files that begin with `test`
- Finds functions that begin with `test`
- Select based on filters

Gotchas

- Test output is verbose
- Read tests

Helpful Filters

Specific task

```
>>> pytest -m task0_1
```

Specific test

```
>>> pytest -k test_sum
```


How do unit tests work?

- Tries to run code
- If there is a False assert it fails
- Only prints if test fails!
- `assert` and `assert_close`

Module 0 Functions

```
def relu(x):  
    """  
    f(x) = x if x is greater than 0, else 0  
    """  
    ...
```


Mathematical Testing

- How do we know that it works?

Standard Unit Test

- Test for values with given inputs
- PyTest succeeds if no assertions are called

```
def test_relu():  
    assert operators.relu(10.0) == 10.0  
    assert operators.relu(-10.0) == 0.0
```


Ideal: Property Test

Test that all values satisfy property

```
def test_relu():  
    for a in range(0, 1e9):  
        assert operators.relu(a) == a  
  
    for a in range(-1e9, 0):  
        assert operators.relu(a) == 0.0
```


QuickCheck / Hypothesis

- <https://en.wikipedia.org/wiki/QuickCheck>
- <https://hypothesis.readthedocs.io/en/latest/>

Compromise: Randomized Property Test

Test that sampled values satisfy property.

import math

```
from hypothesis import example, given
from hypothesis.strategies import floats
```

```
def relu(a):
    if a < 5:
        return 0
    else:
        return a
```

```
@given(floats())
@example(1.0)
def test_relu(a: float):
    if math.isnan(a):
        return
    value = relu(a)
    if a >= 0:
```


Custom Generators

- Can provide your own randomized generators
- Future assignments will utilize this feature.

Functional Python

Functional Programming

- Style of programming where functions can be passed and used like other objects.
- One of several programming styles supported in Python.
- Good paradigm for mathematical programming

Function Type

```
from typing import Callable

def add(a: float, b: float) -> float:
    return a + b

def mul(a: float, b: float) -> float:
    return a * b

v: Callable[[float, float], float] = add
```


Functions as Arguments

```
def combine3(  
    fn: Callable[[float, float], float], a: float, b: float, c: float  
    ) -> float:  
    return fn(fn(a, b), c)
```

```
print(combine3(add, 1, 3, 5))  
print(combine3(mul, 1, 3, 5))
```

9

15

Functional Python

Functions as Returns

```
def combine3(  
    fn: Callable[[float, float], float]  
) -> Callable[[float, float, float], float]:  
    def new_fn(a: float, b: float, c: float) -> float:  
        return fn(fn(a, b), c)  
  
    return new_fn
```

```
add3: Callable[[float, float, float], float] = combine3(add)  
mul3: Callable[[float, float, float], float] = combine3(mul)  
  
print(add3(1, 3, 5))
```

9

```
type(add3)
```

function

Higher-order Filter

Extended example

```
def filter(fn: Callable[[float], bool]) -> Callable[[Iterable[float]], Iterable[float]]:
    def apply(ls: Iterable[float]):
        ret = []
        for x in ls:
            if fn(x):
                ret.append(x)
        return ret
    return apply
```


Higher-order Filter

Extended example

```
def more_than_4(x: float) -> bool:  
    return x > 4
```

```
filter_for_more_than_4: Callable[[Iterable[float]], Iterable[float]] = filter(  
    more_than_4  
)
```

```
filter_for_more_than_4([1, 10, 3, 5])
```

```
[10, 5]
```


Functional Python

Rules of Thumbs:

- When in doubt, write out defs
- Document the arguments that functions take and send
- Write tests in for loops to sanity check

Q&A

