

Machine Learning Engineering

Lecture 2

Fundamentals

Survey

- Thanks so much for filling out the survey
- Link: <https://forms.gle/eUhmZWX9mzZLXyMp8>
- Please fill out this week!

Today's Class

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

The Guidebook

- <https://minitorch.github.io/>
- 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/>
- Important: Link your Cornell email to your Github.

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

Contributing Guidelines

Style

- Configure your development environment to check for style errors

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

- Checks for any style or documentation errors

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

Continuous Integration

- Runs behind the scenes on every commit.

The screenshot shows the GitHub Actions interface for a workflow named "GitHub Classroom Autograding Workflow". The workflow is currently in a failed state, indicated by a red 'X' icon. The left sidebar shows the workflow's history, with the "Autograding" job selected. The main panel displays the details of the "Autograding" job, which failed 39 minutes ago. The job log shows the following steps:

- Set up job (2s) - Success
- Run actions/checkout@v2 (1s) - Success
- Run education/autograding@beta (17s) - Failure

The failure message for the "Run education/autograding@beta" step is:

```
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 (list): A list of any type.  
        i (int): An index into the list  
  
    Returns:  
        Value at ls[i].  
    """  
    ...
```

Testing

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
- Protip: minimize testing speed

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

Implement

```
def relu(x):  
    """  
    :math: f(x) = x \text{ if } x \text{ is greater than } 0, \text{ else } 0  
  
    (See \<https://en.wikipedia.org/wiki/Rectifier\_\(neural\_netwo  
    """
```

Module 0 Functions

Implement

```
def relu(x):  
    """  
    :math:`f(x) = x` if  $x$  is greater than 0, else 0  
  
    (See https://en.wikipedia.org/wiki/Rectifier\_\(neural\_networks\))  
    """
```

- Pretty basic function.

Module 0 Functions

Implement

```
def relu(x):  
    """  
    :math:`f(x) = x` if  $x$  is greater than 0, else 0  
  
    (See `<https://en.wikipedia.org/wiki/Rectifier\_\(neural\_netwo  
    """
```

- Pretty basic function.
- How do we know it works?

Standard Unit Test

Test for values with given inputs

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

- (PyTest succeeds if no assertions are called)

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
```

- Intractable

QuickCheck (Hypothesis)

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

Compromise: Randomized Property Test

Test that randomly selected values satisfy property.

```
@given(floats())
def test_relu(a):
    value = operators.relu(a)
    if a >= 0:
        assert value == a
    else:
        assert value == 0.0
```

- Greater coverage with less code

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

Functional Python

Functions as Arguments

```
def combine3(fn, a, b, c):  
    return fn(fn(a, b), c)  
  
def add(a, b):  
    return a + b  
  
def mul(a, b):  
    return a * b  
  
print(combine3(add, 1, 3, 5))  
print(combine3(mul, 1, 3, 5))
```

```
9  
15
```


Functional Python

Functions as Returns

```
def combine3(fn):  
    def apply(a, b, c):  
        return fn(fn(a, b), c)  
    return apply
```

```
add3 = combine3(add)  
mul3 = combine3(mul)
```

```
add3(1, 3, 5) # 9
```

```
9
```

Higher-order Filter

Extended example

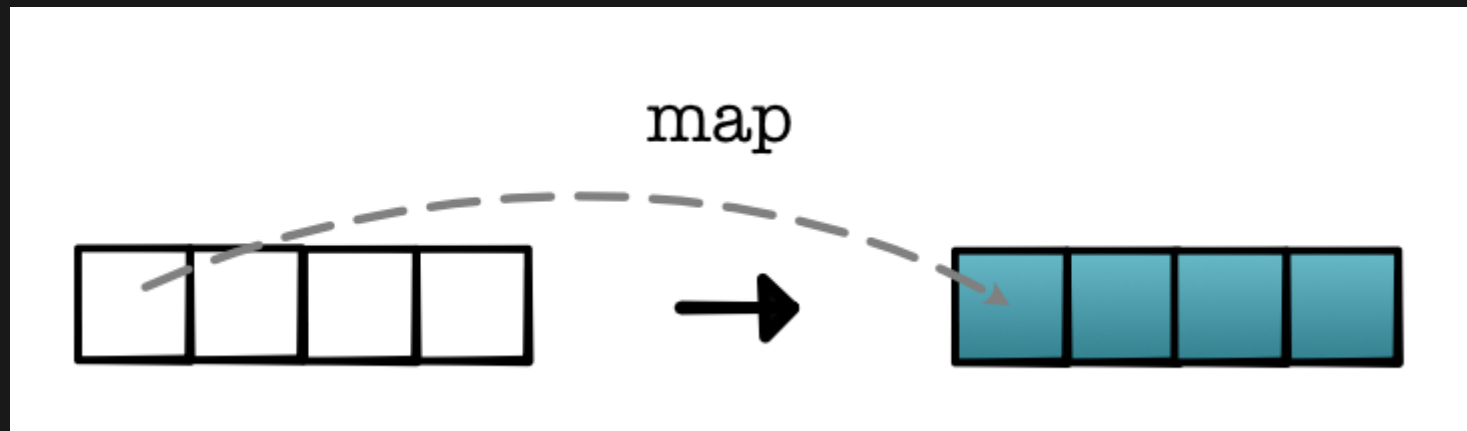
```
def filter(fn):  
    def apply(ls):  
        ret = []  
        for x in ls:  
            if fn(x):  
                ret.append(x)  
        return ret  
    return apply  
  
def more_than_4(x):  
    return x > 4  
  
filter_for_more_than_4 = filter(more_than_4)  
filter_for_more_than_4([1, 10, 3, 5])
```

```
[10, 5]
```


Module-0 Functions

`minitorch.operators.map(fn)`

Higher-order map.



See [https://en.wikipedia.org/wiki/Map_\(higher-order_function\)](https://en.wikipedia.org/wiki/Map_(higher-order_function))

order_function)

Parameters:

fn (*one-arg function*) -- Function from one value to one value.

Returns:

A function that takes a list, applies *fn* to each element, and returns a new list

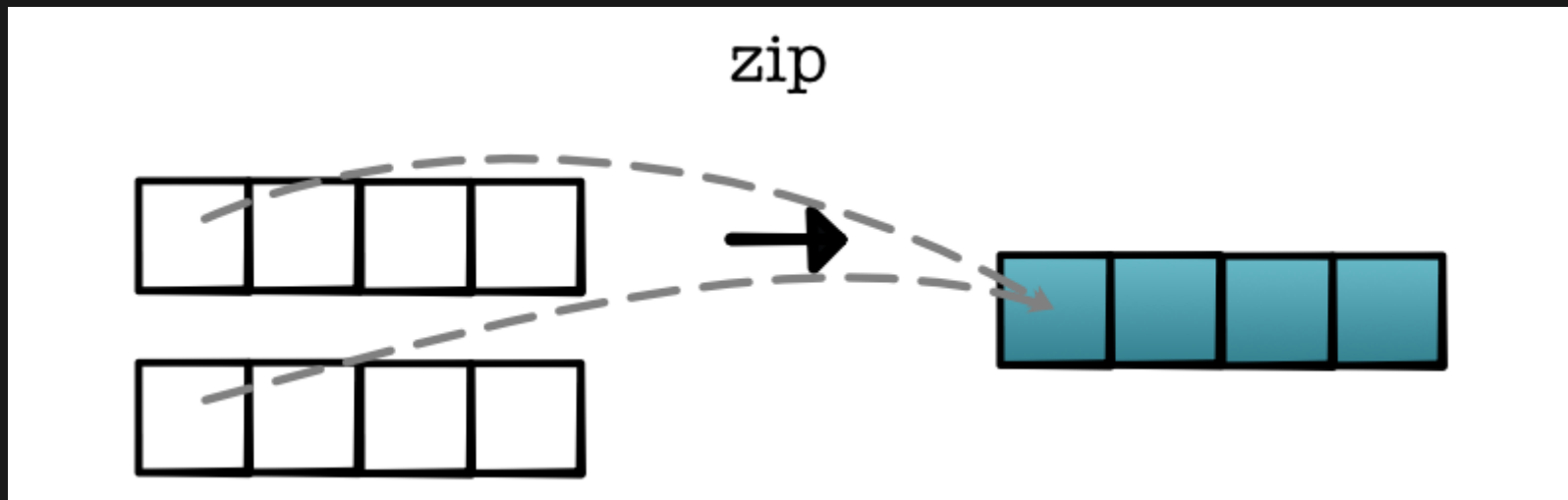
Return type:

function

Module-0 Functions

`minitorch.operators.zipWith(fn)`

Higher-order zipwith (or map2).



See [https://en.wikipedia.org/wiki/Map_\(higher-](https://en.wikipedia.org/wiki/Map_(higher-)

See [https://en.wikipedia.org/wiki/map_\(higher_order_function\)](https://en.wikipedia.org/wiki/map_(higher_order_function))

Parameters:

fn (*two-arg function*) -- combine two values

Returns:

takes two equally sized lists *ls1* and *ls2*,
produce a new list by applying `fn(x, y)` on each
pair of elements.

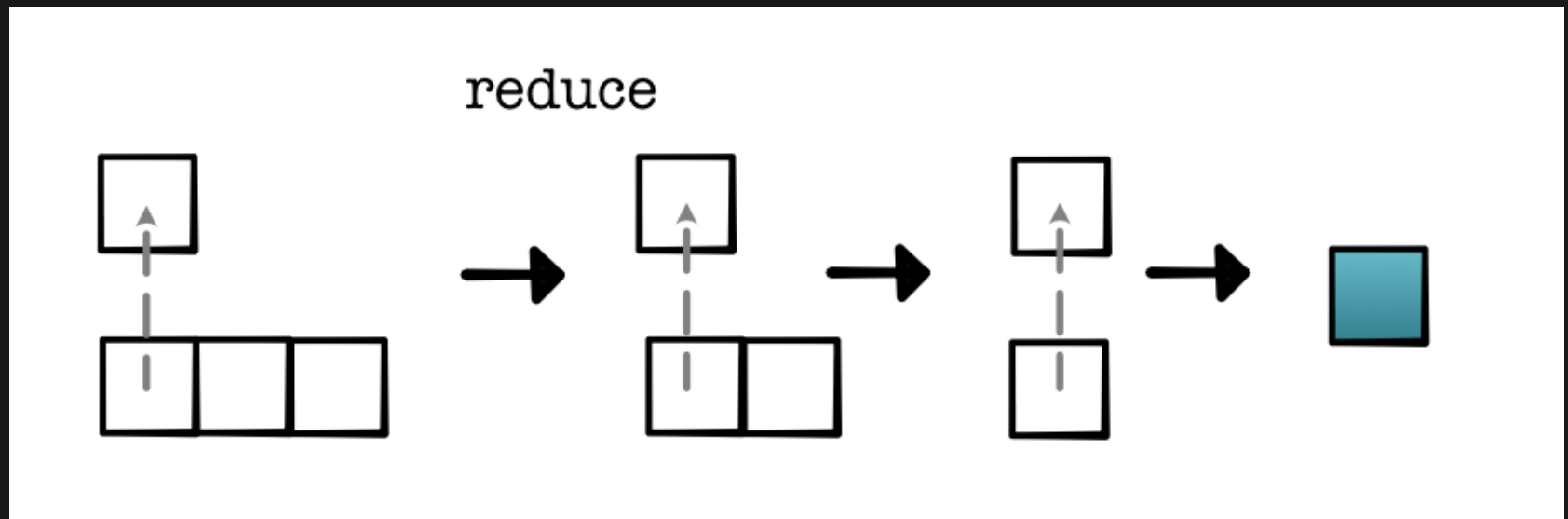
Return type:

function

Module-0 Functions

`minitorch.operators.reduce(fn, start)`

Higher-order reduce.



Parameters:

Parameters:

- **fn** (*two-arg function*) -- combine two values
- **start** (*float*) -- start value x_0

Returns:

function that takes a list ls of elements $x_1 \dots x_n$ and computes the reduction $fn(x_3, fn(x_2, fn(x_1, x_0)))$

Return type:

function

Functional Python

Rules of Thumbs

Functional Python

Rules of Thumbs

- When in doubt, write out defs

Functional Python

Rules of Thumbs

- When in doubt, write out defs
- Document the arguments that functions take and send

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