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

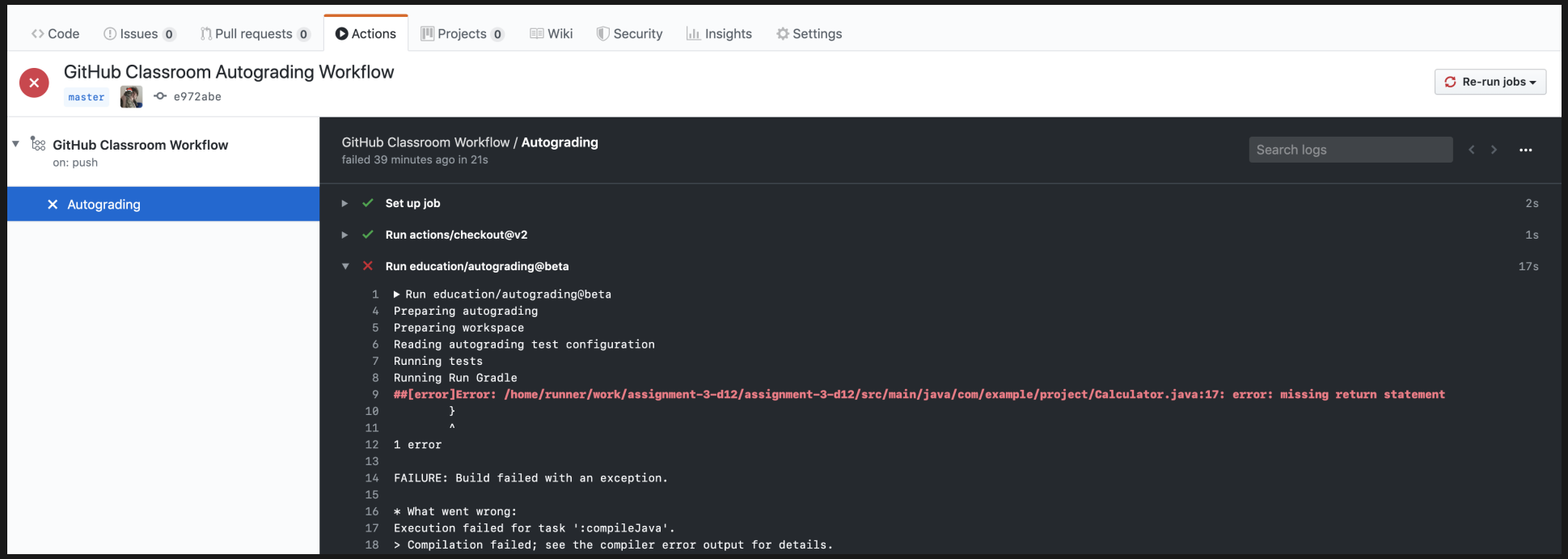
```
>>> black minitorch/ tests/ project/
```

- Checks for any style or documentation errors

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

Continuous Integration

- Runs behind the scenes on every commit.



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 if x is greater than 0, 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_netwo
  """
```

- 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

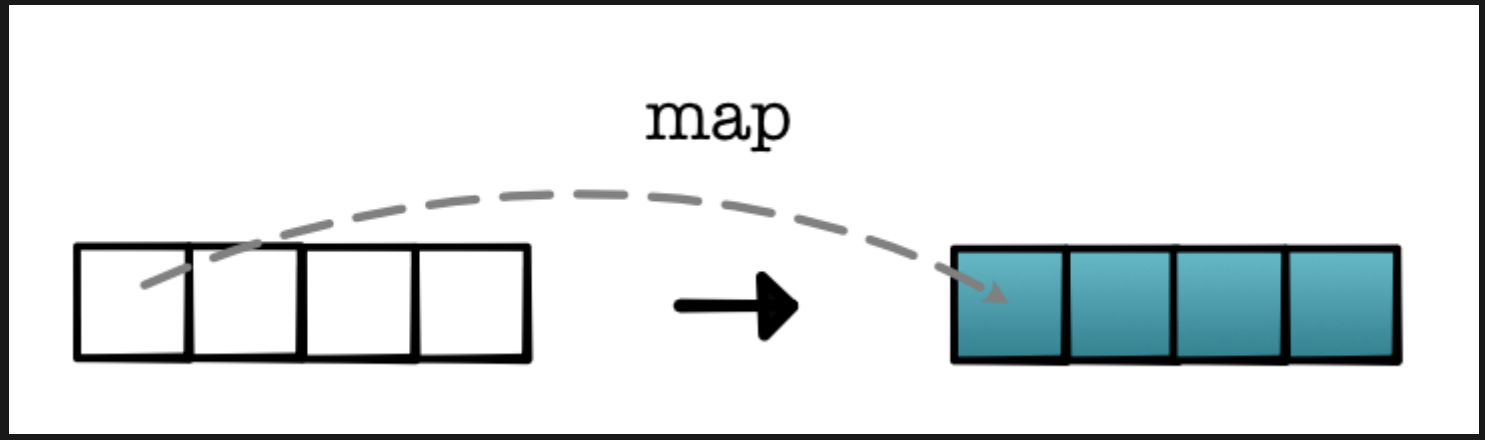
```
def filter(fn):
   def apply(ls):
      ret = []
      for x in ls:
         if fn(x):
             ret.append(x)
      return ret
   return apply

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)

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

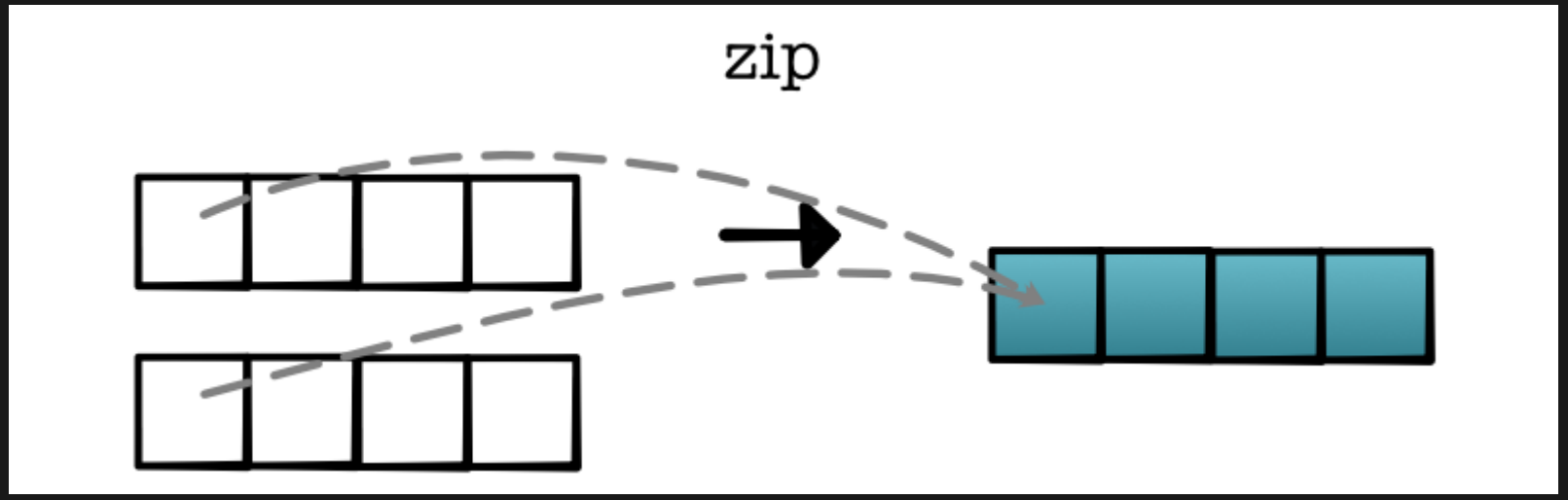
Return type:

function

Module-0 Functions

`minitorch.operators.zipWith(`*fn*`)`

Higher-order zipwith (or map2).



See https://en.wikipedia.org/wiki/Map_(higher-

See 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.

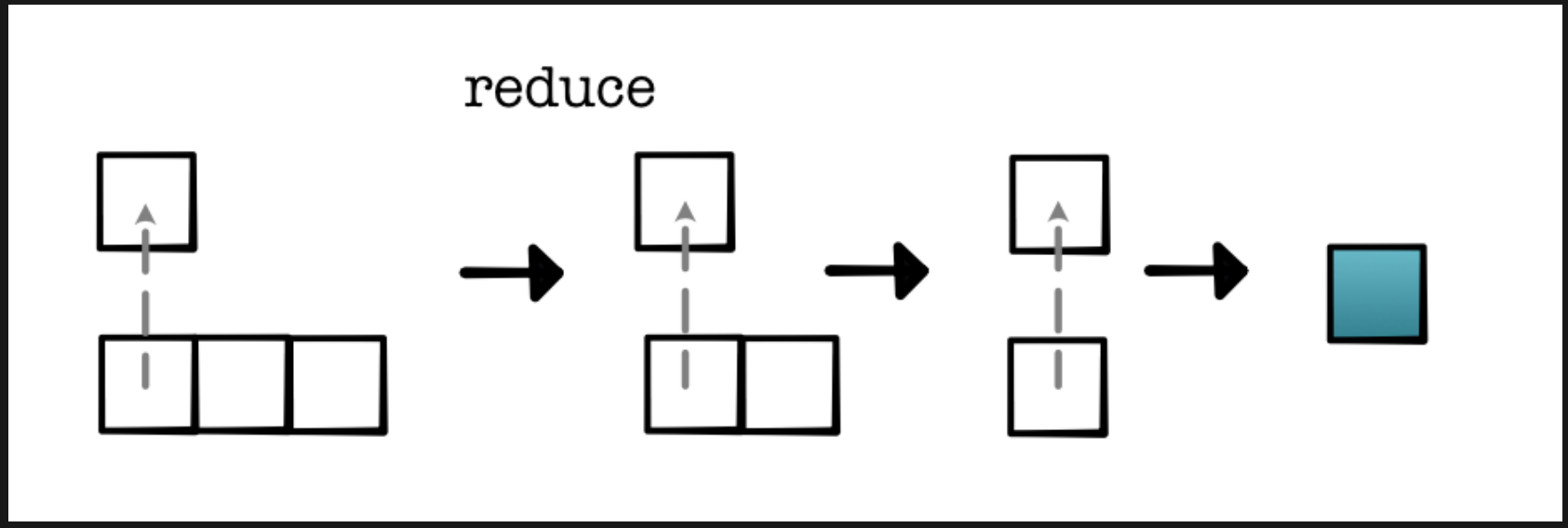
Return type:

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 $x_1 \ldots x_n$ 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