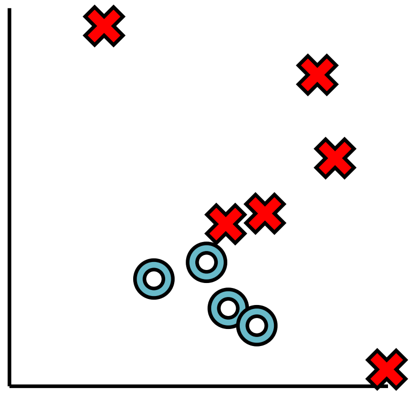
Module 1.1 - Learning With Derivatives

Training Data

- Set of datapoints, each (x, y)
- x position x_1, x_2
- y true label, color



Math

- Linear Model

$$m(x;\theta = w, b) = x_1 \times w_1 + x_2 \times w_2 + b$$

```
def forward(self, x1: float, x2: float) -> float:
    return self.w1 * x1 + self.w2 * x2 + self.b
```

Graphical Notation

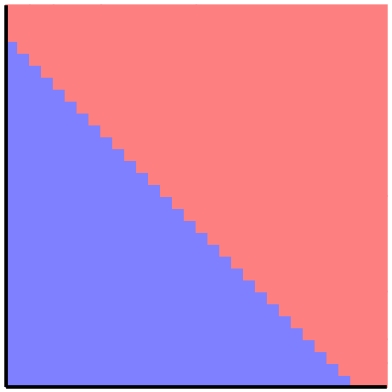
- Red is more positive, blue is more negative.
- $m(x)$ provides a value for every x_1, x_2 every point.
- Line represents seperator

Model 1

- Linear Model

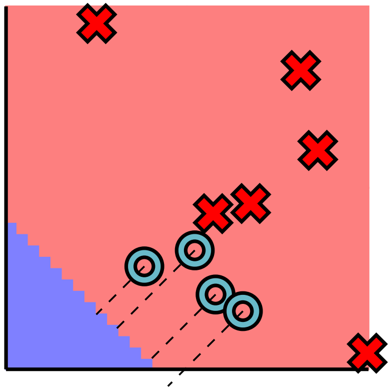
```
@dataclass
class Linear:
    # Parameters
    w1: float
    w2: float
    b: float

    def forward(self, x1: float, x2: float) -> float:
        return self.w1 * x1 + self.w2 * x2 + self.b
```

Decision Boundary: Model 1

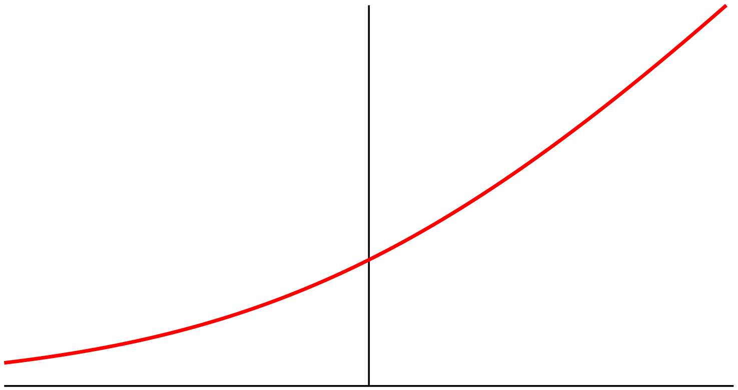


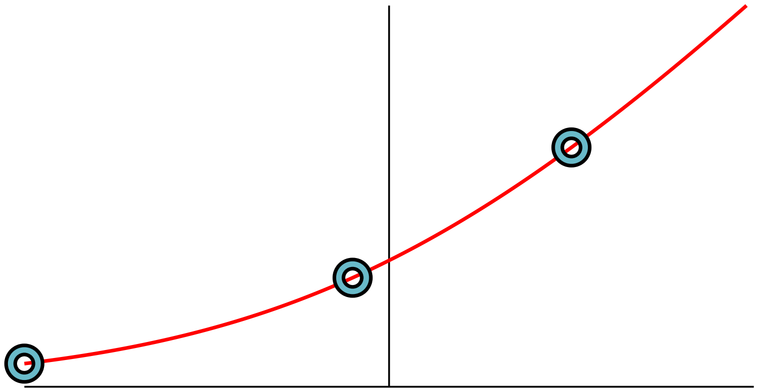
Distance Determines Fit

- $m(x)$ red or blue.



Point Loss





Loss

- $L(\theta)$ loss is a function of parameters
- We change parameters, decision boundary changes

Lecture Quiz

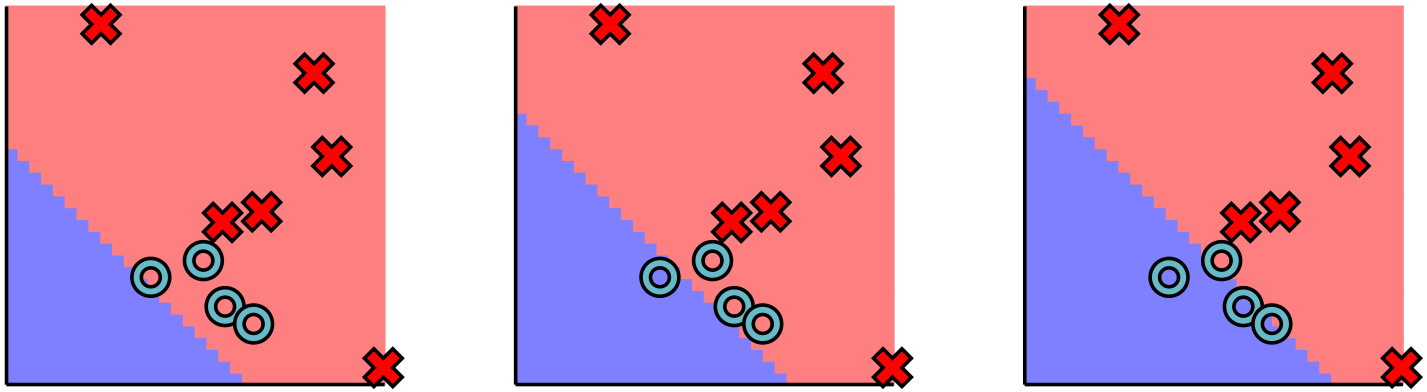
Outline

- Model Fit
- Symbolic Derivatives
- Numerical Derivatives
- Module 1

Model Fitting

Class Goal

- Find parameters that minimize loss



Numerical Optimization

- Many, many different approaches
- Our focus: *gradient descent*
- Workhorse of modern machine learning

Iterative Parameter Fitting

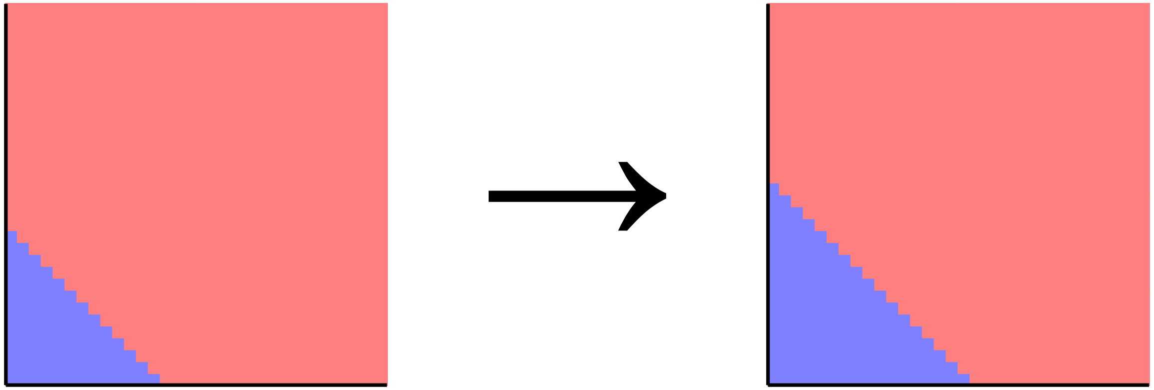
1. Compute the loss function, $L(\theta)$

2. See how small changes would change the loss

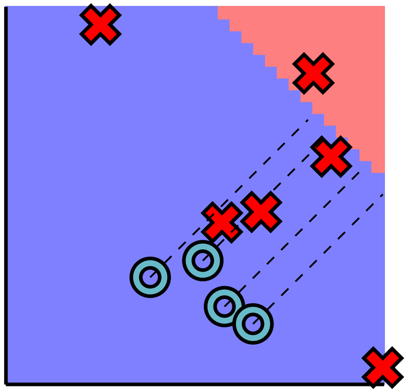
3. Update to parameters to locally reduce the loss

Example: Update Bias

```
model1 = Linear(w1=1, w2=1, b=-0.4)
model2 = Linear(w1=1, w2=1, b=-0.5)
```



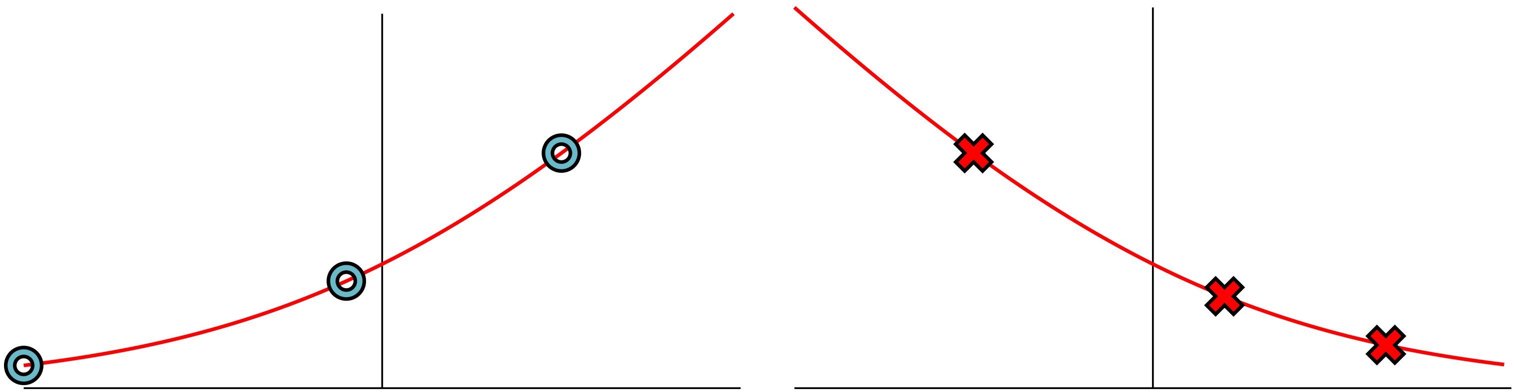
Step 1: Compute Loss



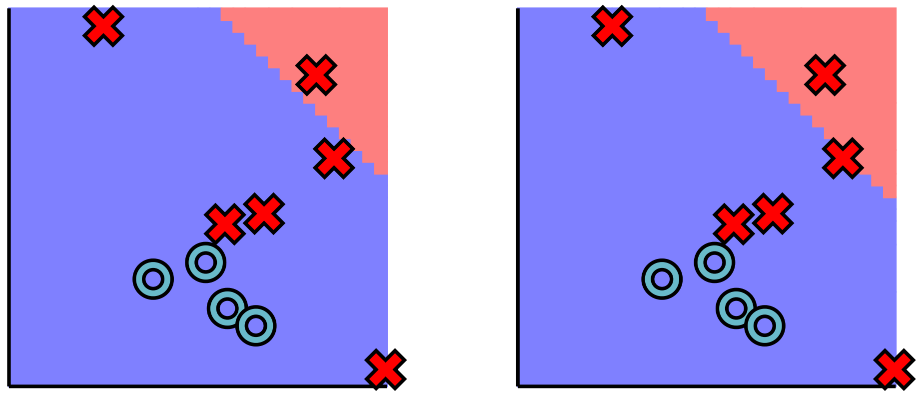
```python
def point_loss(out, y=1):
    return y * -math.log(  # Correct Side
        minitorch.operators.sigmoid(-out)  # Log-Sigmoid
    )  # Distance
```

Loss

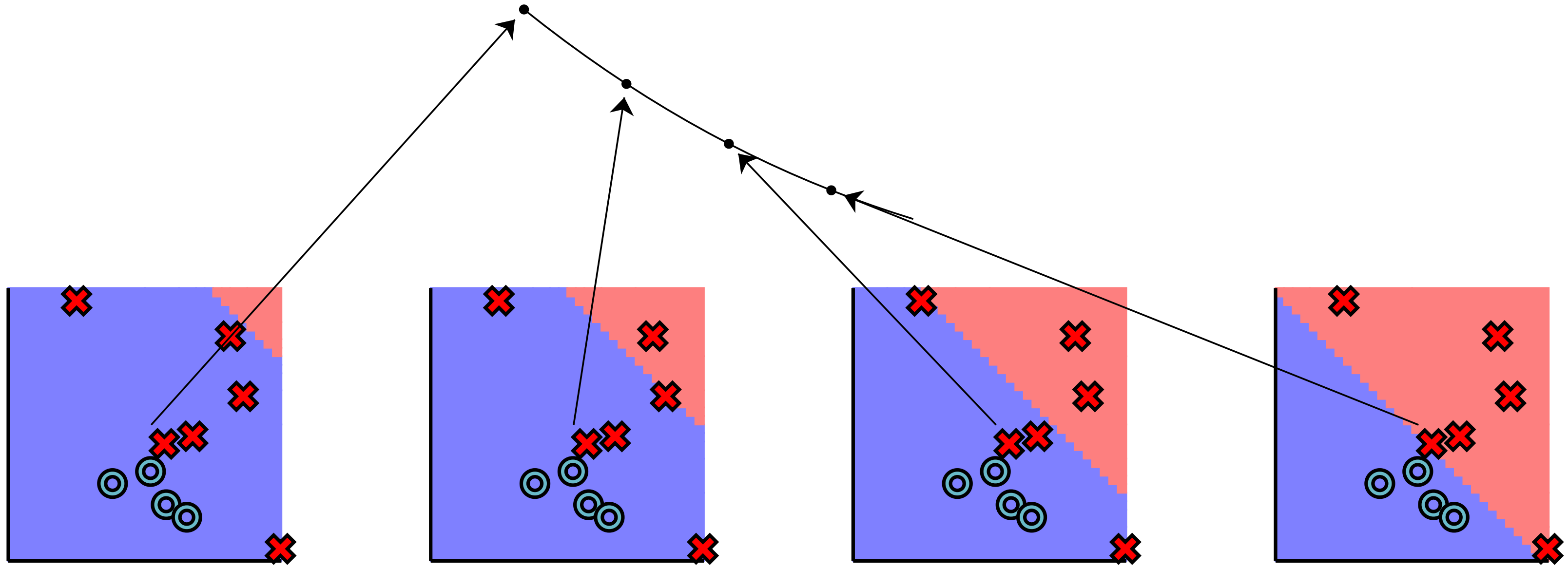
```python
def full_loss(m): # Given m( ; \theta)
    l = 0
    for x, y in zip(s.X, s.y): # For all training data
        l += point_loss(-m.forward(*x), y)
    return -l
```



Step 2: Find Direction of Improvement



Step 3: Update Parameters Iteratively



Our Challenge

How do we find the right direction?

Symbolic Derivatives

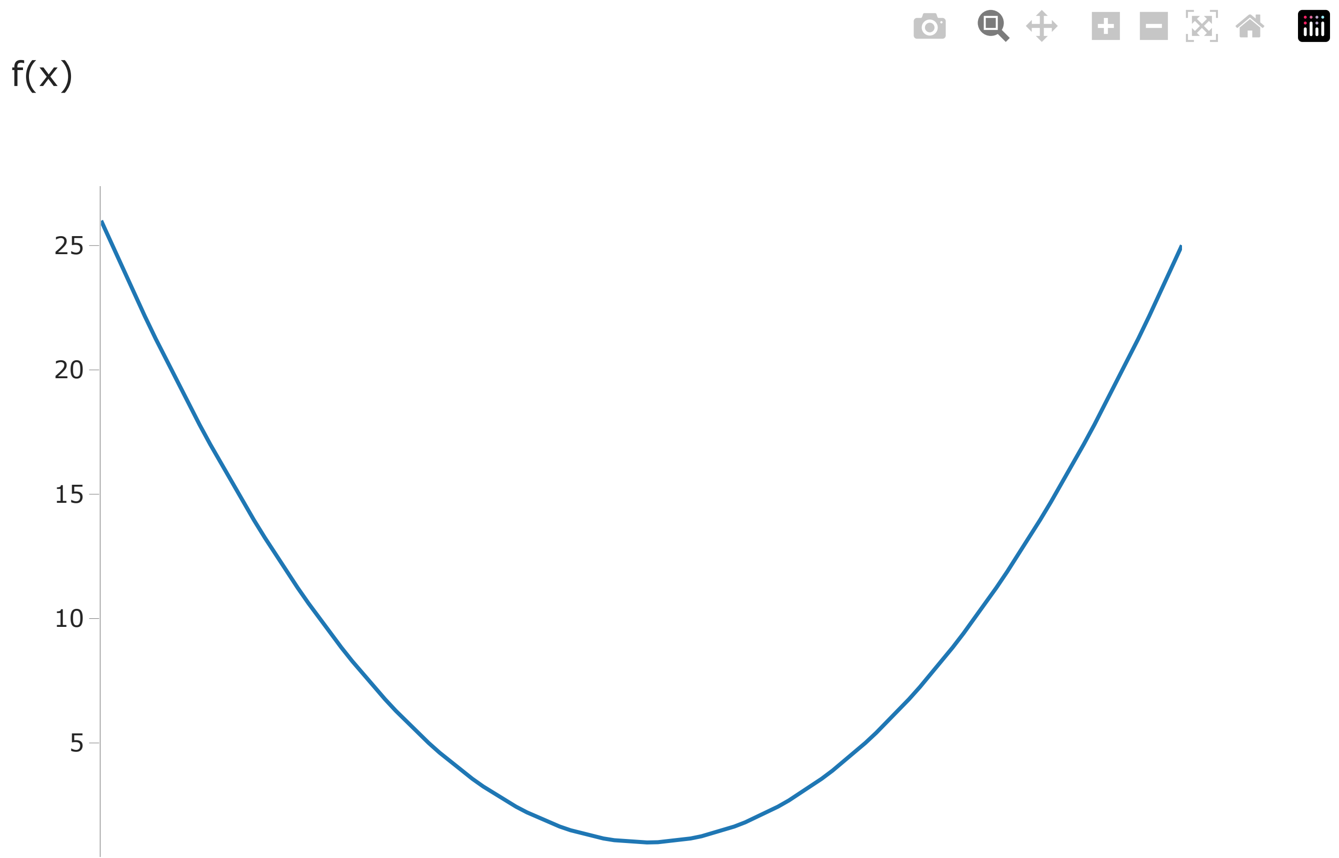
Review: What is a Derivative?

How small changes in input impact output.

- $f(x)$ - function
- x - point
- $f'(x)$ - "rise/run"

Review: Derivative

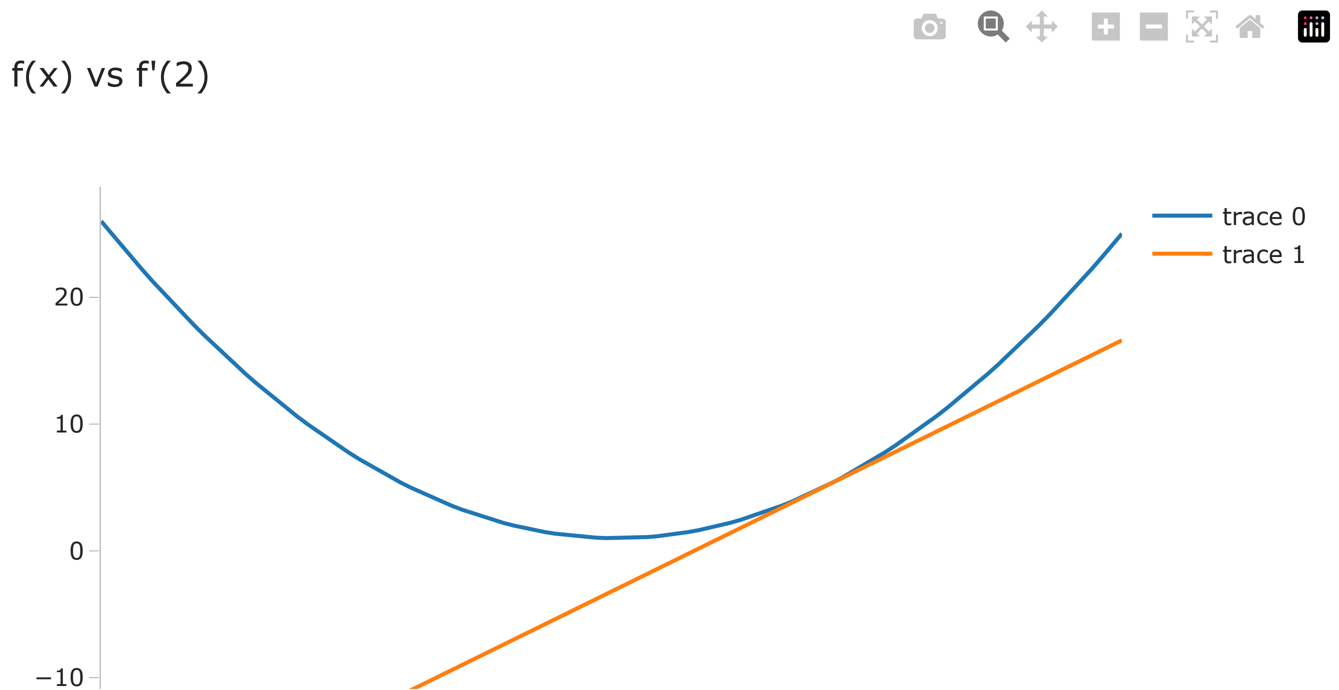
$$f(x) = x^2 + 1$$



Review: Derivative

$$f(x) = x^2 + 1$$

$$f'(x) = 2x$$

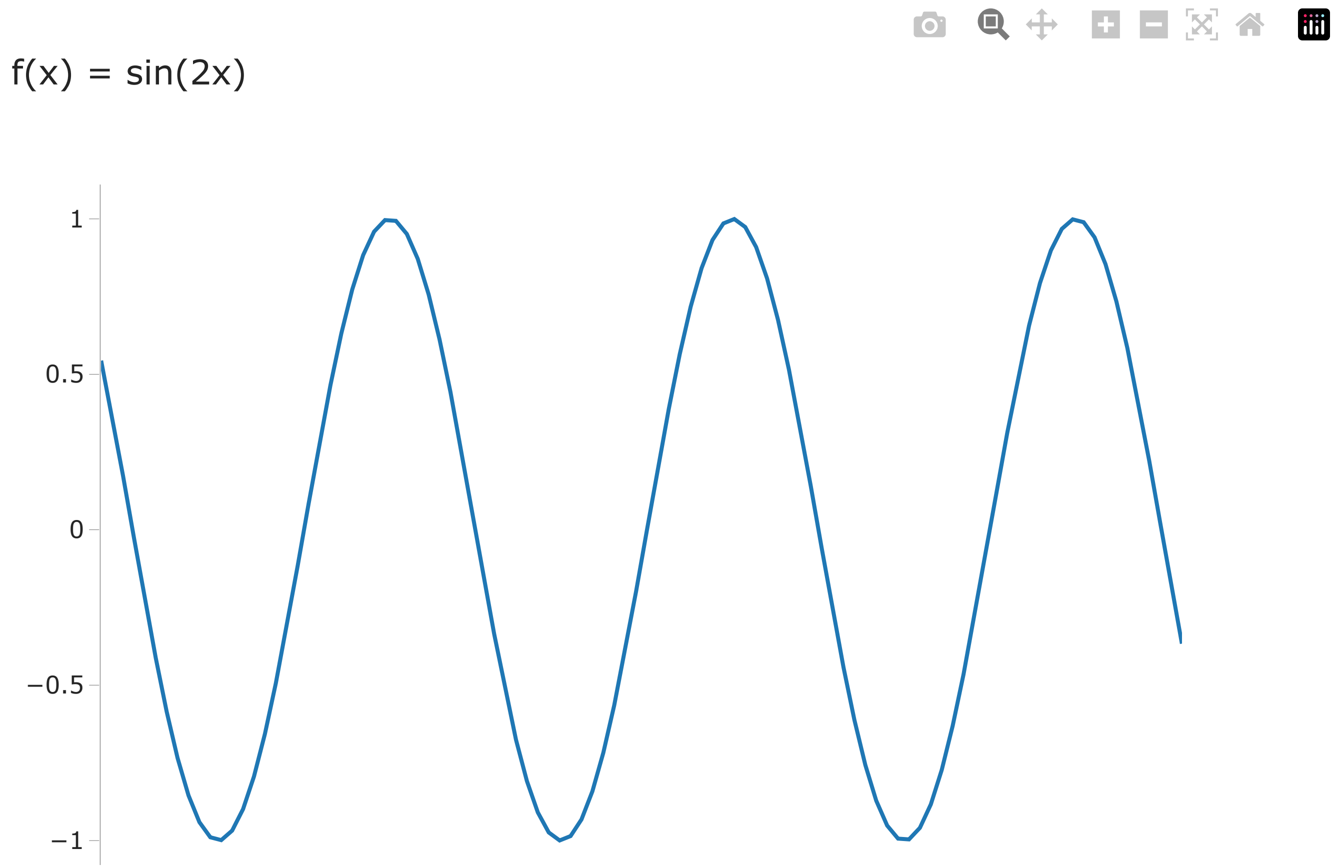


Symbolic Derivative

- Standard high-school derivatives
- Rewrite f to new form f'
- Produces mathematical function

Example Function

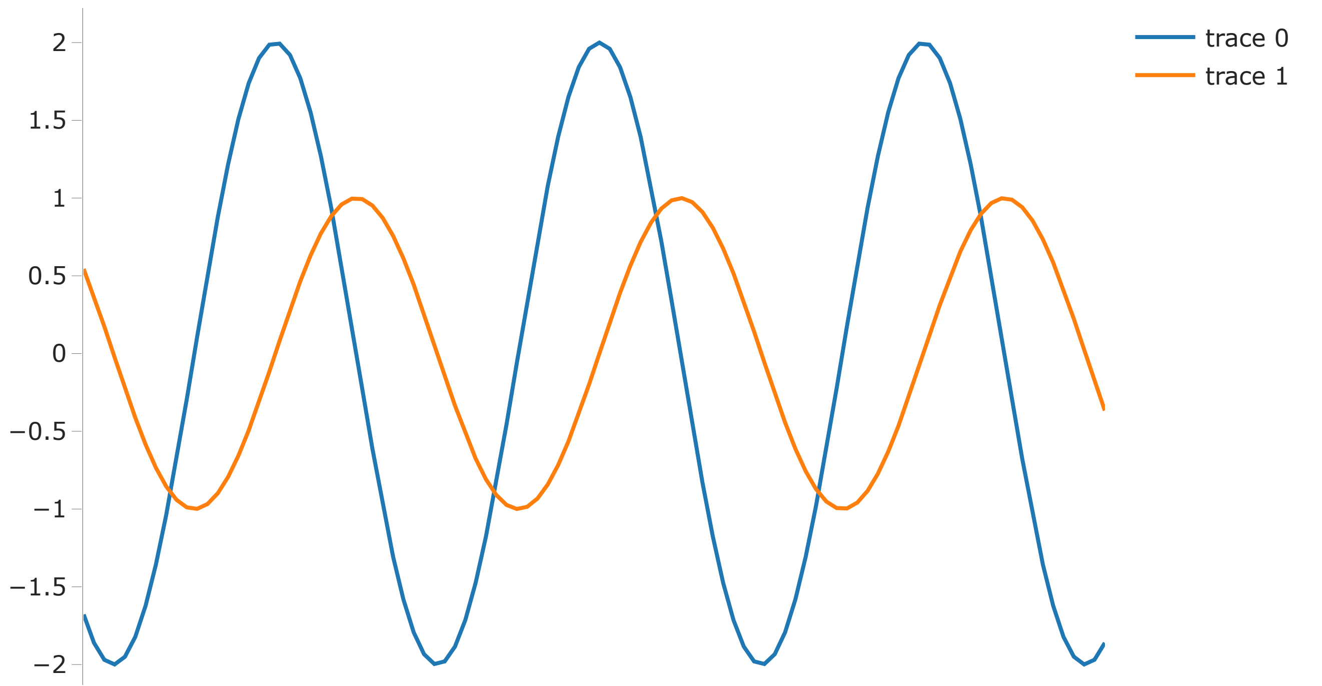
$$f(x) = \sin(2x)$$



Symbolic Derivative

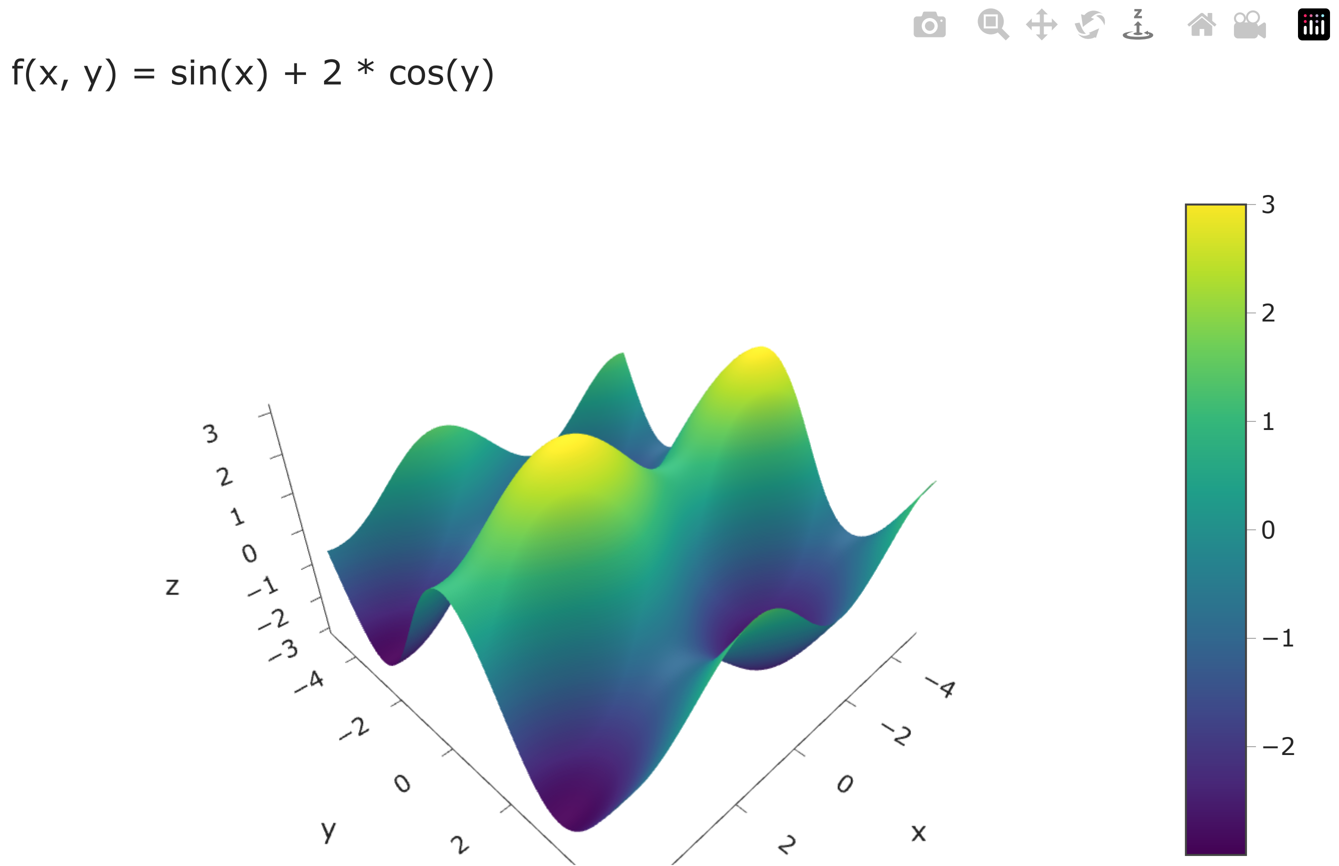
$$f(x) = \sin(2x) \Rightarrow f'(x) = 2\cos(2x)$$

f'(x) = 2*cos(2x)



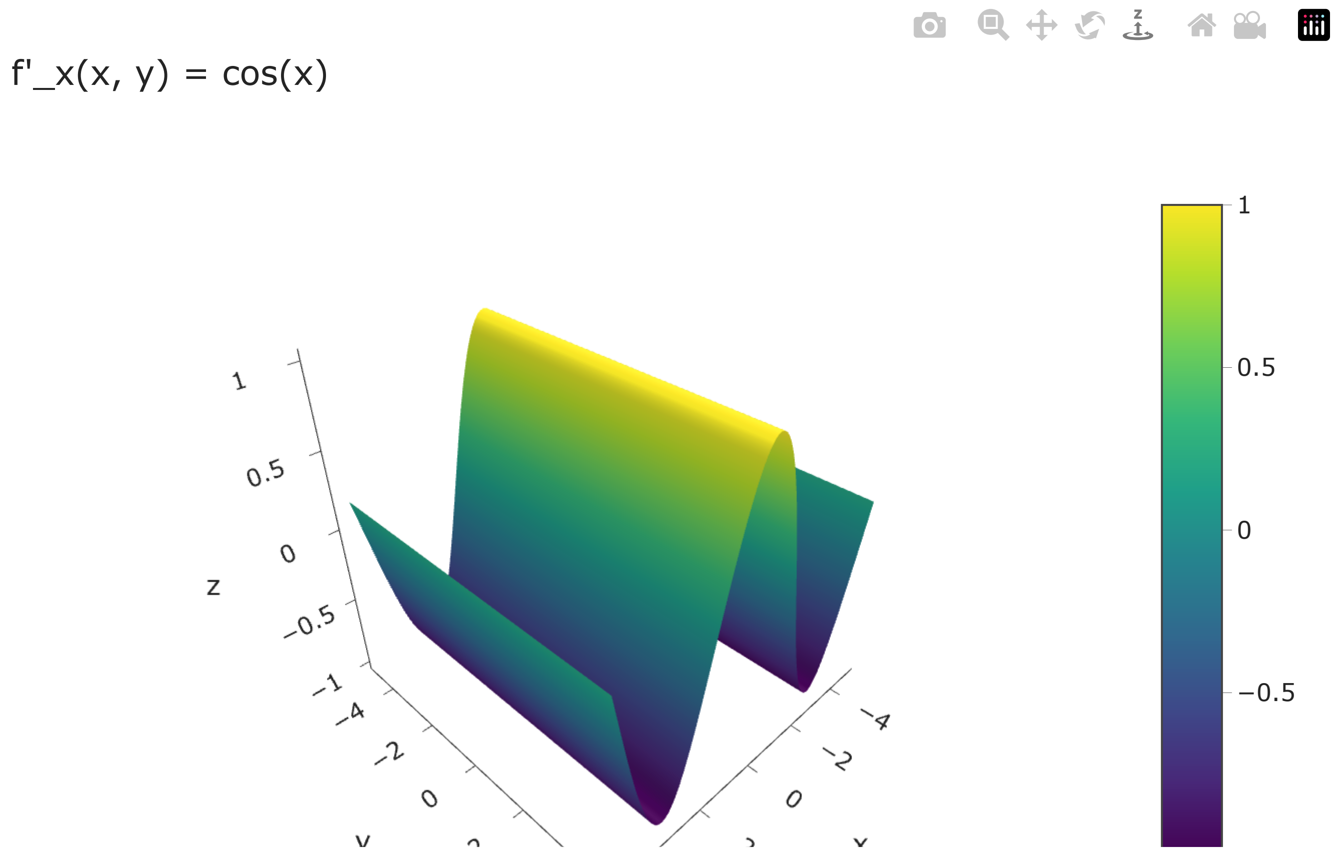
Multiple Arguments

$$f(x, y) = \sin(x) + \cos(y)$$



Derivatives with Multiple Arguments

$$f'_x(x, y) = \cos(x) \quad f'_y(x, y) = -2\sin(y)$$



Review: Symbolic Derivatives

Expectation: Apply basic derivative rules.

- Differentiation Rules

Numerical Derivatives

What if we don’t have symbols?

$$f(x) = \ldots$$

$$f'(x) = \ldots$$

For example if f is unseen code.

```python
def f(x: float) -> float:
    ...
```

Derivative as higher-order function

$$f(x) = \ldots$$

$$f'(x) = \ldots$$

```
def derivative(f: Callable[[float], float]) -> Callable[[float], float]:
    def f_prime(x: float) -> float:
        ...

    return f_prime
```

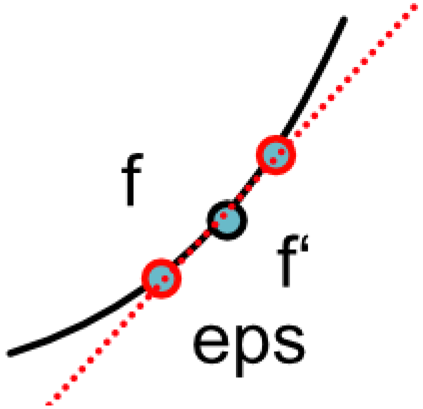
Definition of Derivative

$$f'(x) = \lim_{\epsilon \to 0} \frac{f(x+\epsilon) - f(x-\epsilon)}{2\epsilon}$$

Central Difference

Approximate derivative

$$f'(x) \approx \frac{f(x+\epsilon) - f(x-\epsilon)}{2\epsilon}$$



Approximating Derivative

Key Idea: Only need to call f.

```
def central_difference(f: Callable[[float], float], x: float) -> float:
    ...
```

Derivative as higher-order function

$$f(x) = \ldots$$

$$f'(x) = \ldots$$

```
def derivative(f: Callable[[float], float]) -> Callable[[float], float]:
    def f_prime(x: float) -> float:
        return minitorch.central_difference(f, x)

    return f_prime
```

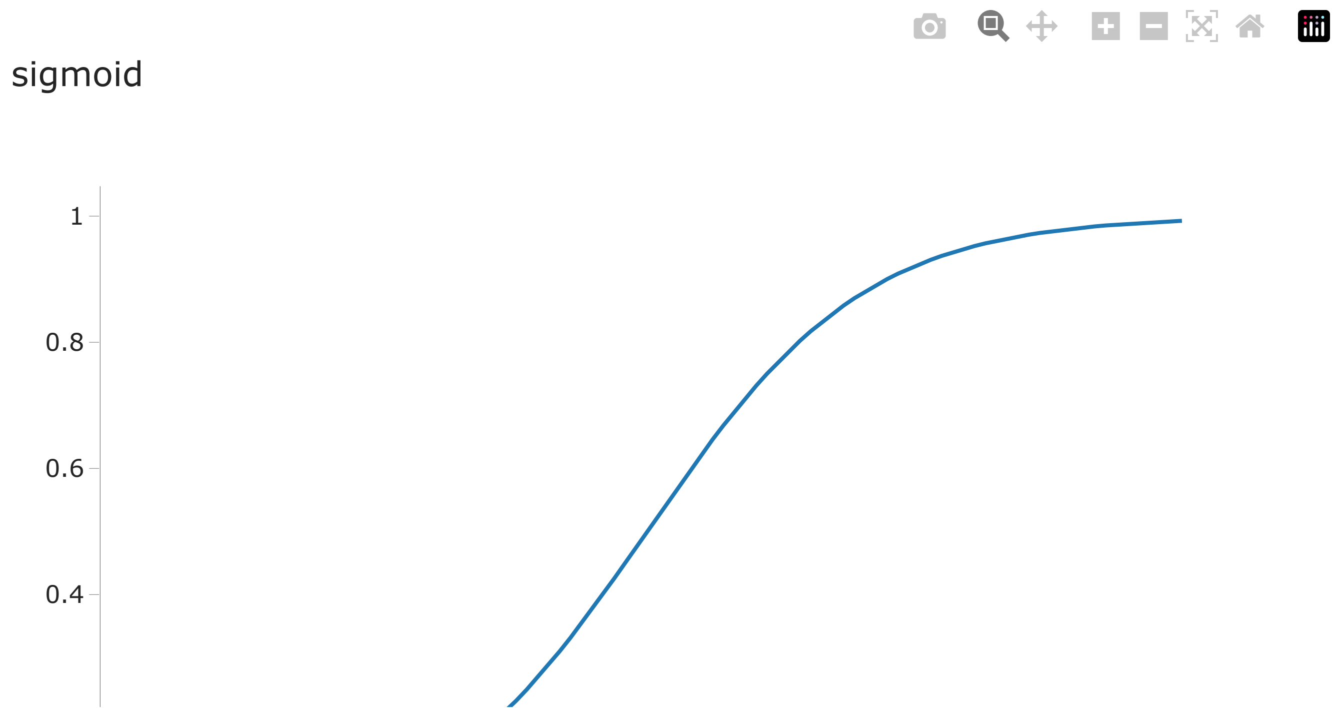
Advanced: Mulitiple Arguments

Turn 2-argument function into 1-arg.

```python
def f(x, y):
    ...


def f_x_prime(x: float, y: float) -> float:
    def inner(x: float) -> float:
        return f(x, y)

    return derivative(inner)(x)
```

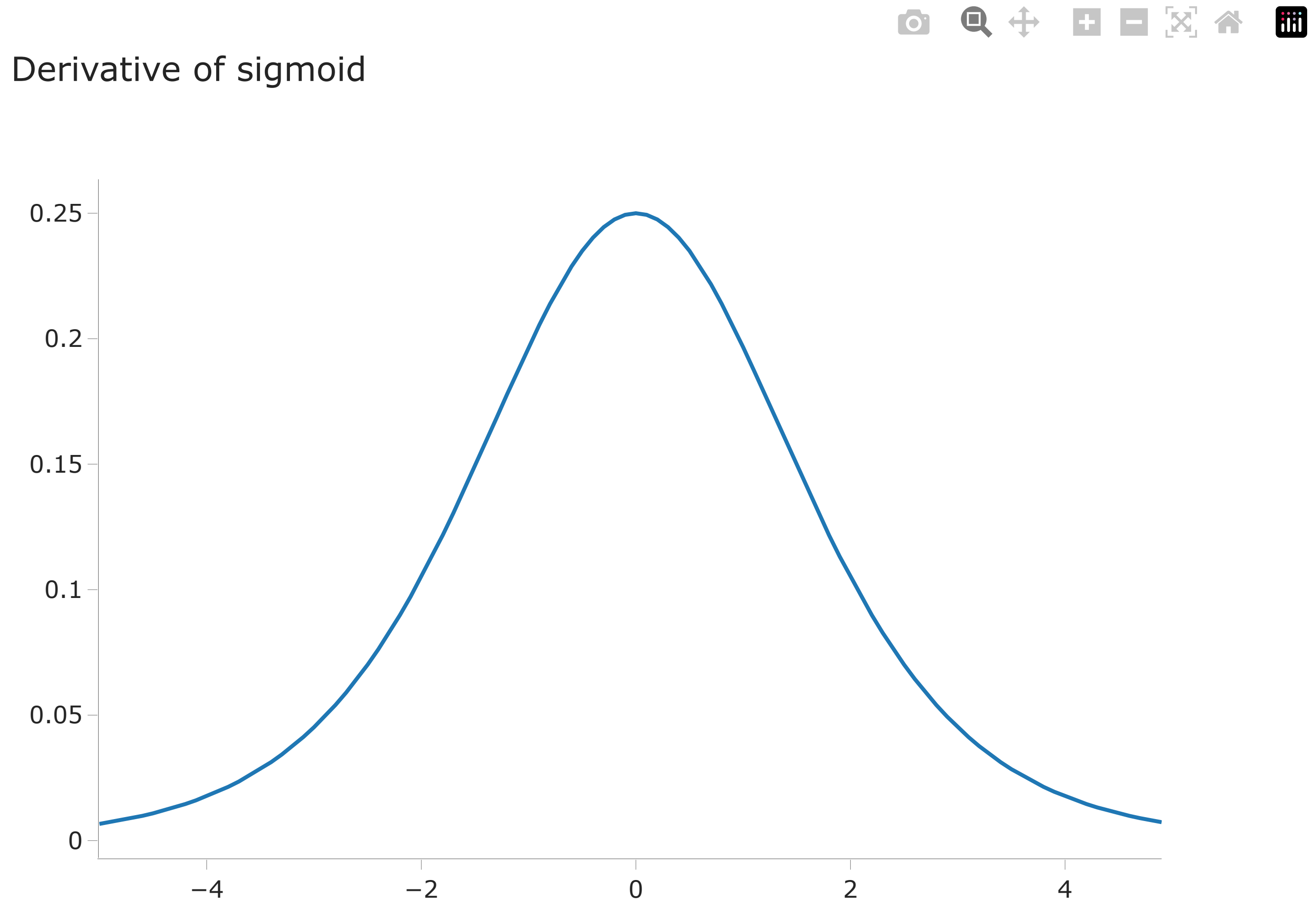
Example

```python
def sigmoid(x: float) -> float:
    if x >= 0:
        return 1.0 / (1.0 + math.exp(-x))
    else:
        return math.exp(x) / (1.0 + math.exp(x))


plot_function("sigmoid", sigmoid)
```



Example

```
sigmoid_prime = derivative(sigmoid)

plot_function("Derivative of sigmoid", sigmoid_prime)
```



Symbolic

- Transformation of mathematical function
- Gives full form of derivative
- Utilizes mathematical identities

Numerical

- Only requires evaluating function
- Computes derivative at a point
- Can be applied to fully black-box function

Next Class: Autodifferentiation

- Computes derivative on programs trace
- Efficient for large number of parameters
- Works directly on python code

Module-1

Module-1 Learning Objectives

- Practical understanding of derivatives
- Dive into autodifferentiation
- Parameters and their usage

Module-1: What is it?

- Numerical and symbolic derivatives
- Implement our numerical class
- Implement autodifferentiation
  - Everything is scalars for now (no "gradients")

Module-1 Overview

- 5 Tasks
- Module 1

Q&A