AI VIETNAM
All-in-One Course

Steps into Linear Regression

Quang-Vinh Dinh
PhD in Computer Science

Year 2023

Objective

Feature	Label
area	price
6.7	9.1
4.6	5.9
3.5	4.6
5.5	6.7

House price data

Features			Label
TV	Radio	Newspaper	Sales
230.1	37.8	69.2	22.1
44.5	39.3	45.1	10.4
17.2	45.9	69.3	12
151.5	41.3	58.5	16.5
180.8	10.8	58.4	17.9

Advertising data

if area=6.0, price=?

if TV=55.0, Radio=34.0, and Newspaper=62.0, price=?

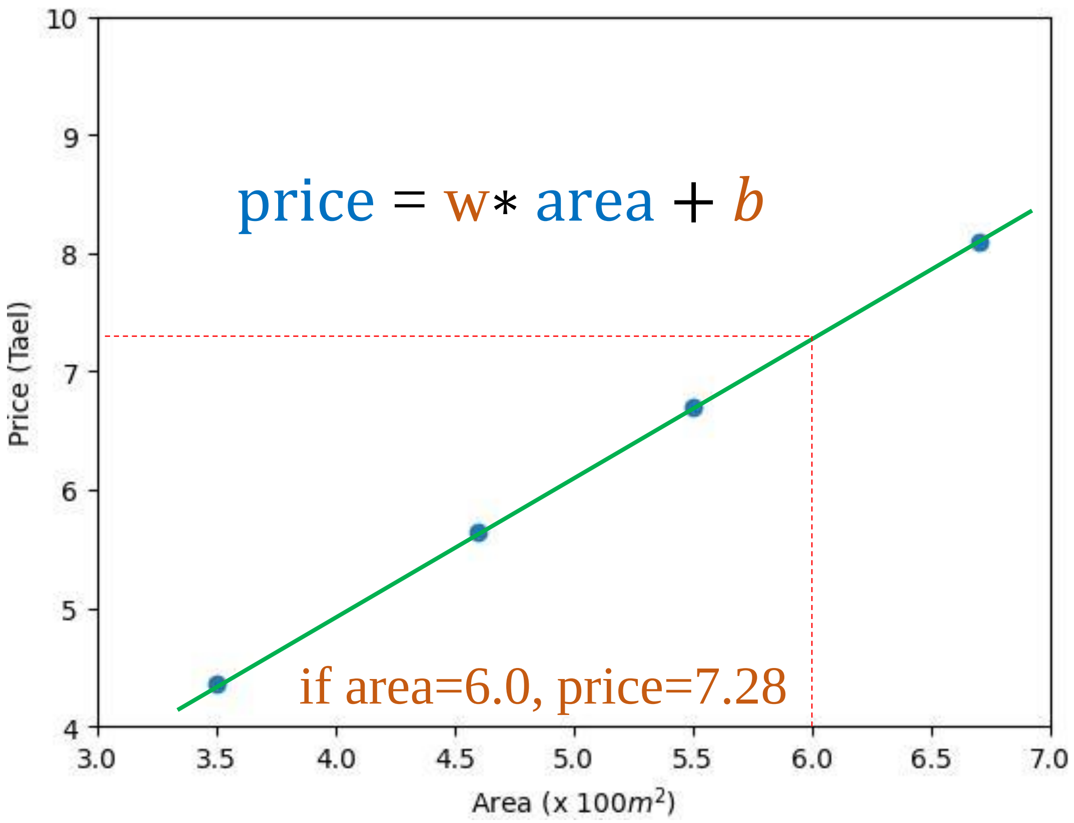
Features													Label
crim	zn	indus	chas	nox	rm	age	dis	rad	tax	ptratio	black	lstat	medv
0.00632	18	2.31	0	0.538	6.575	65.2	4.09	1	296	15.3	396.9	4.98	24
0.02731	0	7.07	0	0.469	6.421	78.9	4.9671	2	242	17.8	396.9	9.14	21.6
0.03237	0	2.18	0	0.458	6.998	45.8	6.0622	3	222	18.7	394.63	2.94	33.4
0.06905	0	2.18	0	0.458	7.147	54.2	6.0622	3	222	18.7	396.9	5.33	36.2
0.08829	12.5	7.87	0	0.524	6.012	66.6	5.5605	5	311	15.2	395.6	12.43	22.9

Boston House Price Data

House Price Prediction

Feature	Label
area	**price**
6.7	8.1
4.6	5.6
3.5	4.3
5.5	6.7

House price data



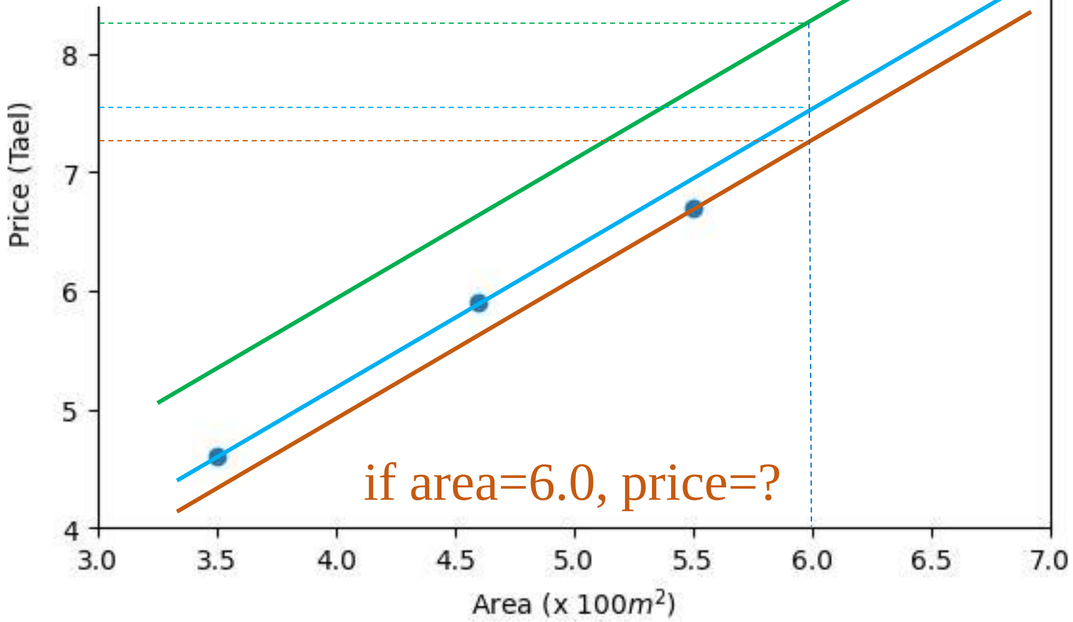
Feature	Label
area	price
6.7	9.1
4.6	5.9
3.5	4.6
5.5	6.7

House price data

$$\text{price} = w_1 * \text{area} + b_1$$

$$\text{price} = w_2 * \text{area} + b_2$$

$$\text{price} = w_3 * \text{area} + b_3$$

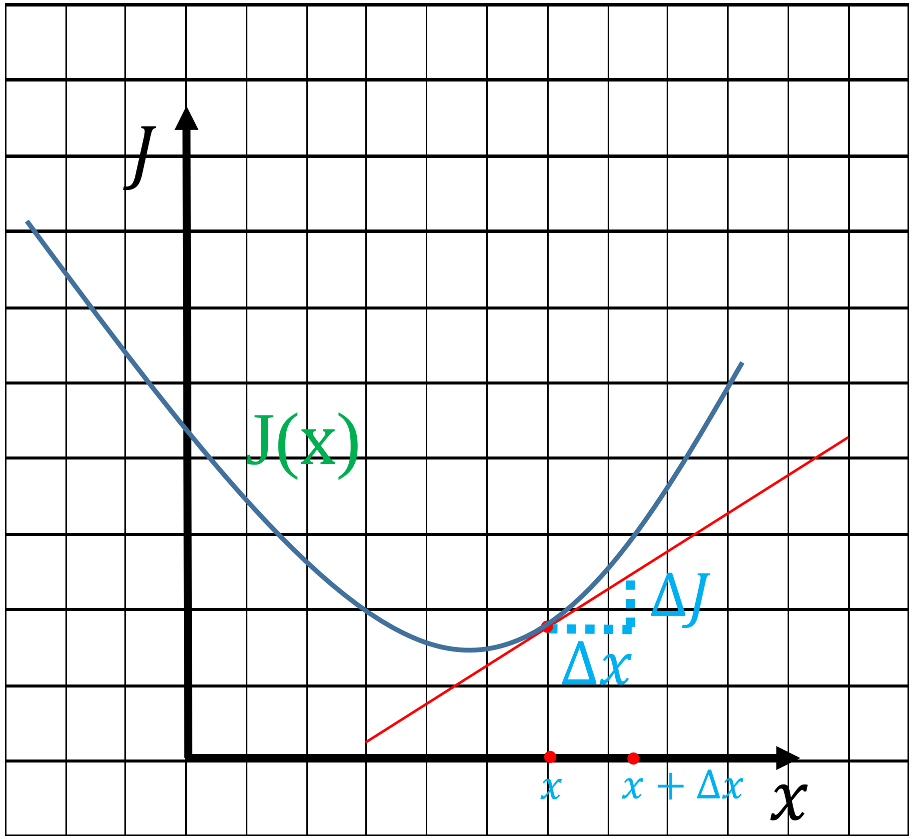


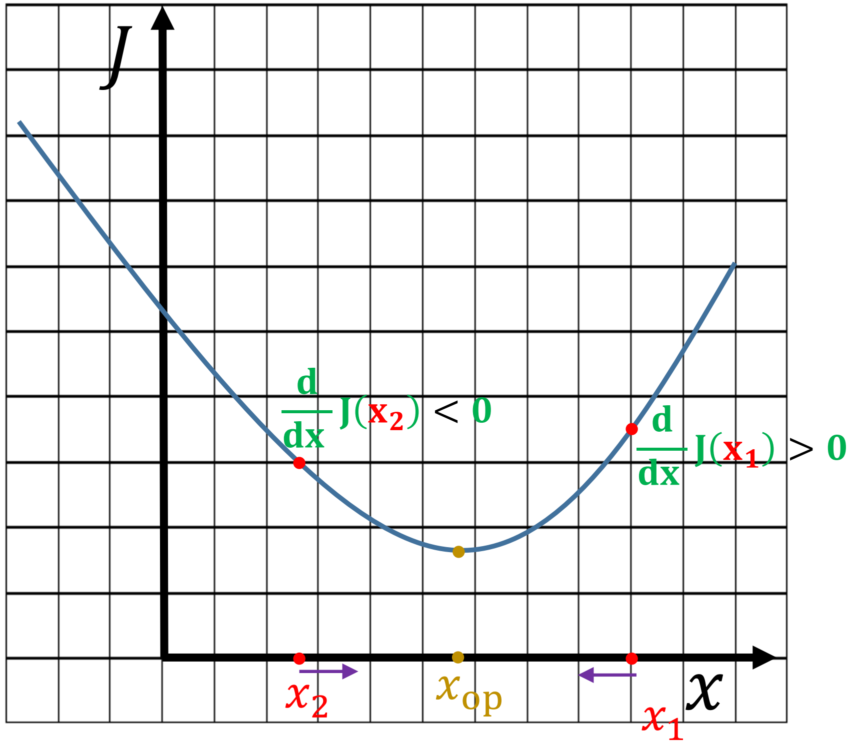
Outline

- Review on Optimization
- Partial Gradient and Chain Rules
- Finding a Line
- Linear Regression

Optimization

❖ Gradient descent





$$\frac{d}{dx}J(x) = \lim_{\Delta x \to 0} \frac{J(x + \Delta x) - J(x)}{\Delta x}$$

$$\mathbf{x} = \mathbf{x} - \eta \frac{\mathbf{d}}{\mathbf{dx}}\mathbf{J}(\mathbf{x})$$

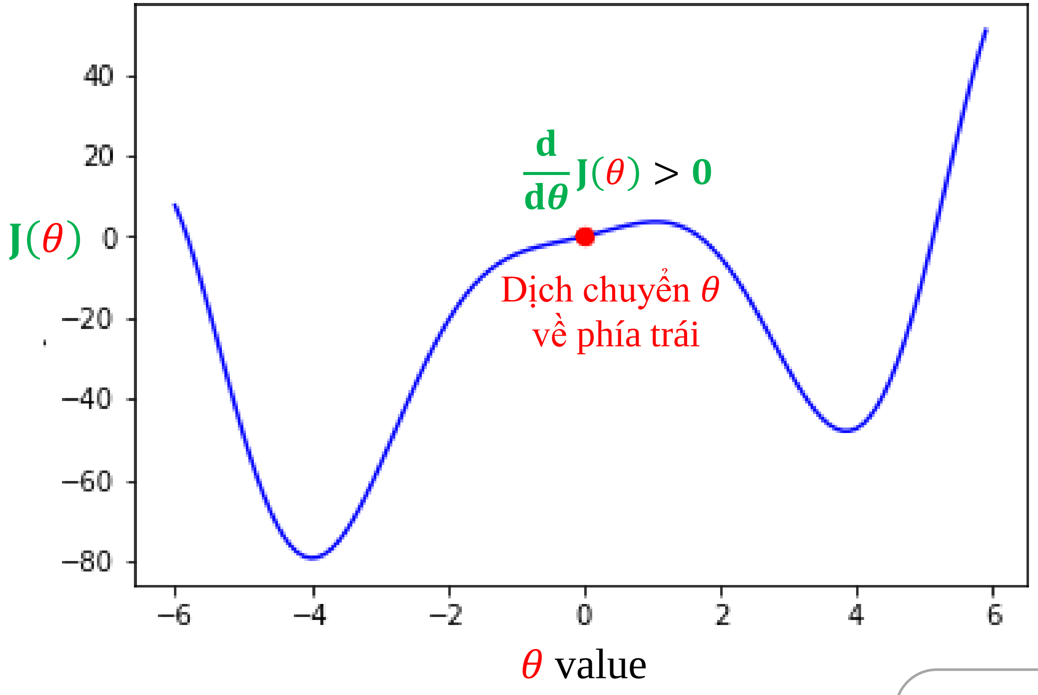
Derivate at x

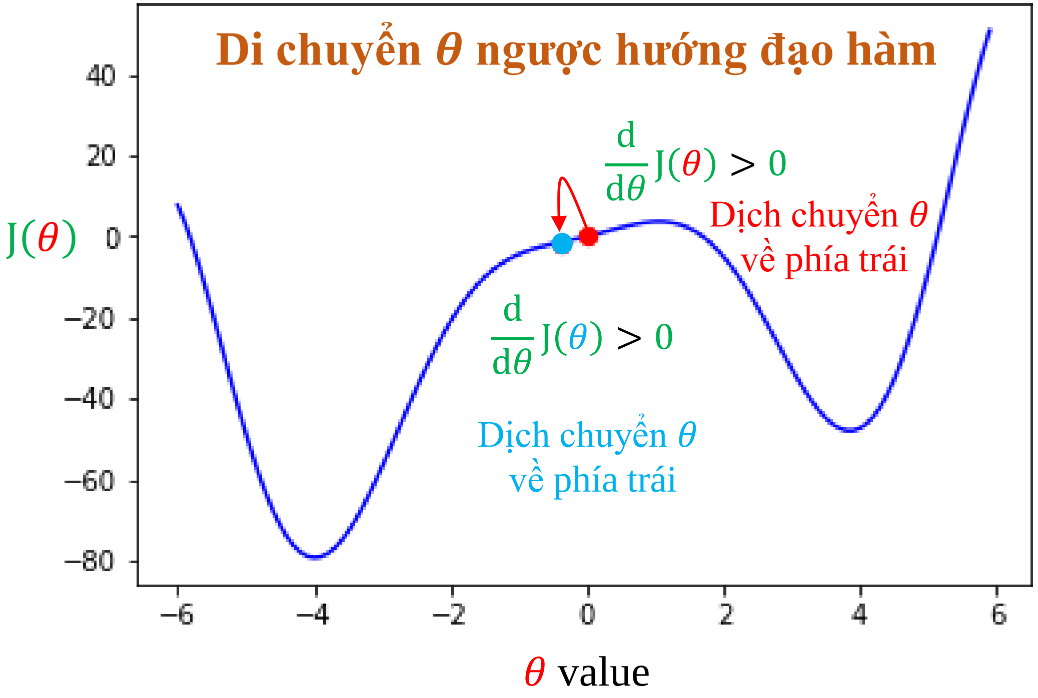
learning rate

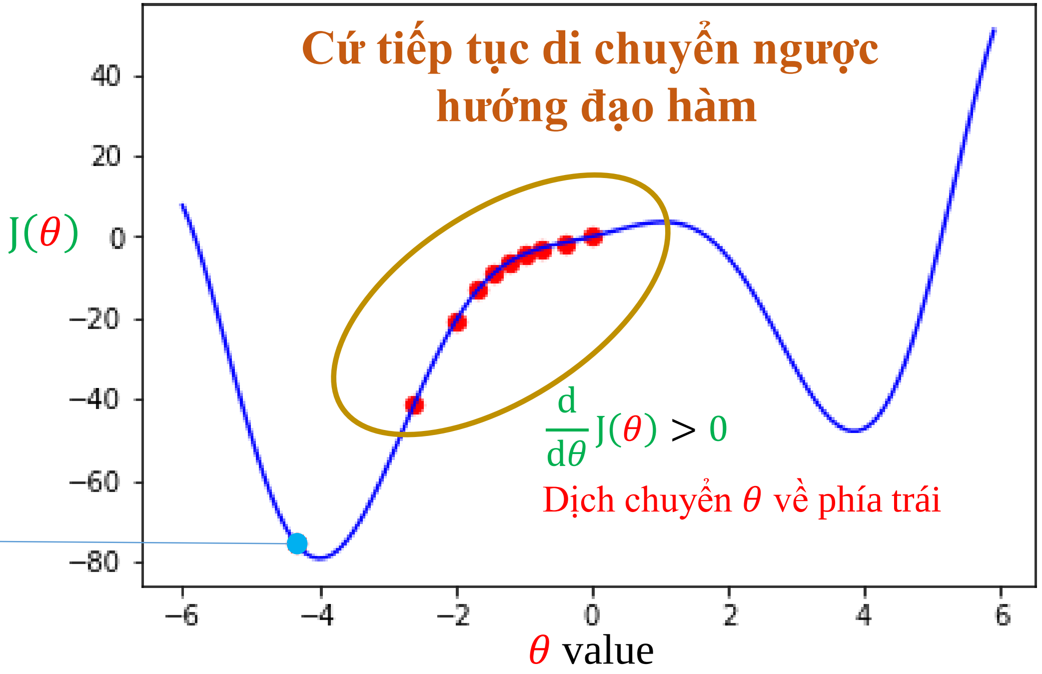
Optimization

❖ Gradient descent

Khởi tạo giá trị θ







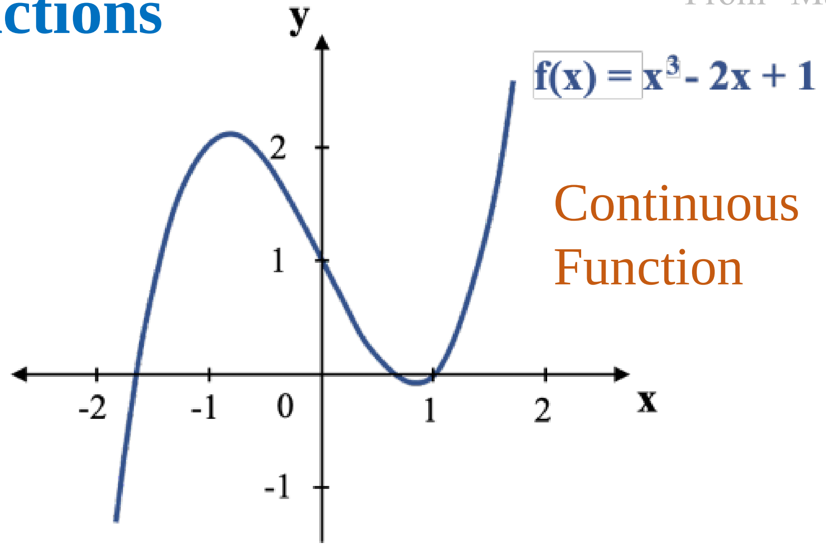
$\frac{d}{d\theta}J(\theta) < 0$

Dịch chuyển θ về phía phải

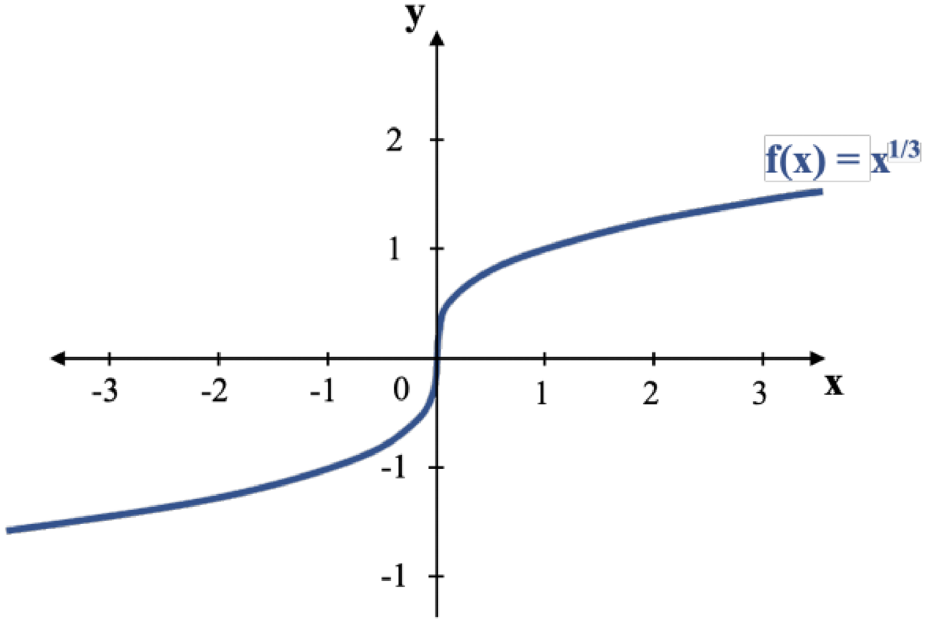
Optimization Algorithms

Loss functions

From “Machine Learning Simplified”

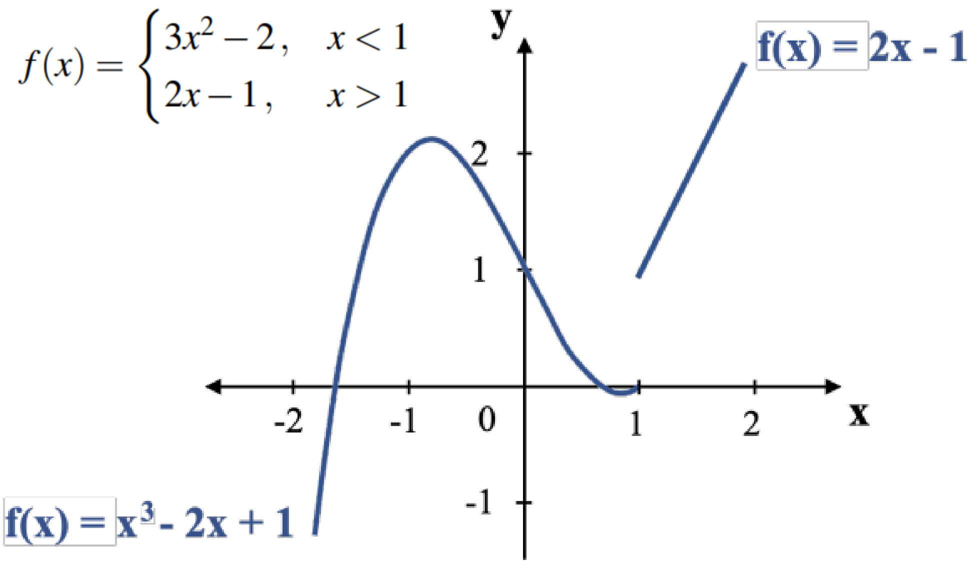


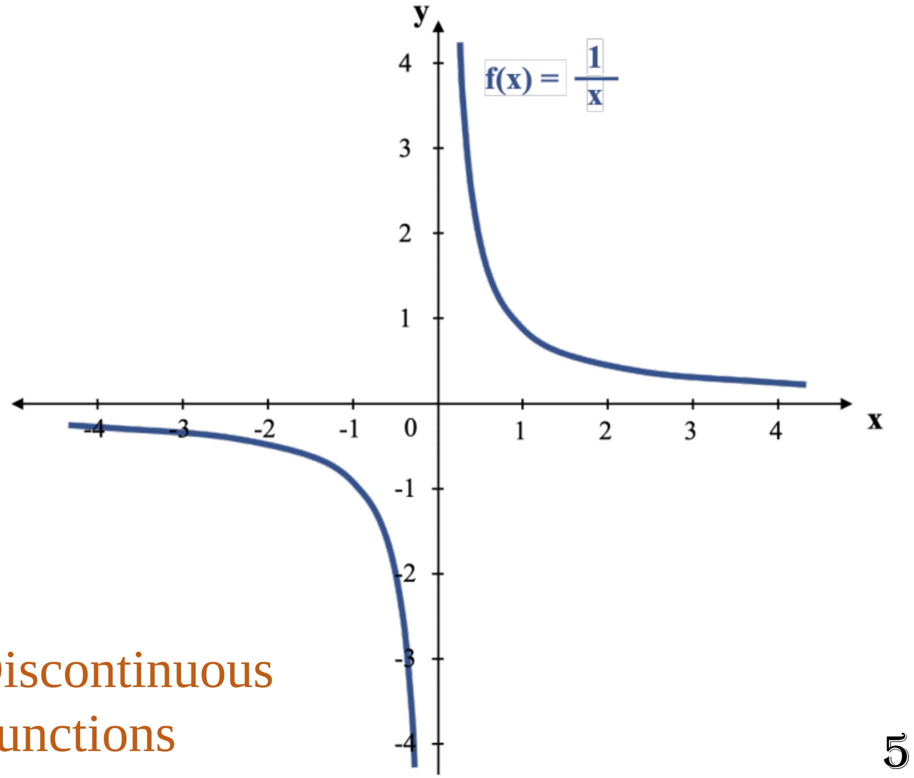
Continuous Function



Continuous non-differentiable functions

$$f(x) = \begin{cases} 3x^2 - 2, & x < 1 \\ 2x - 1, & x > 1 \end{cases}$$

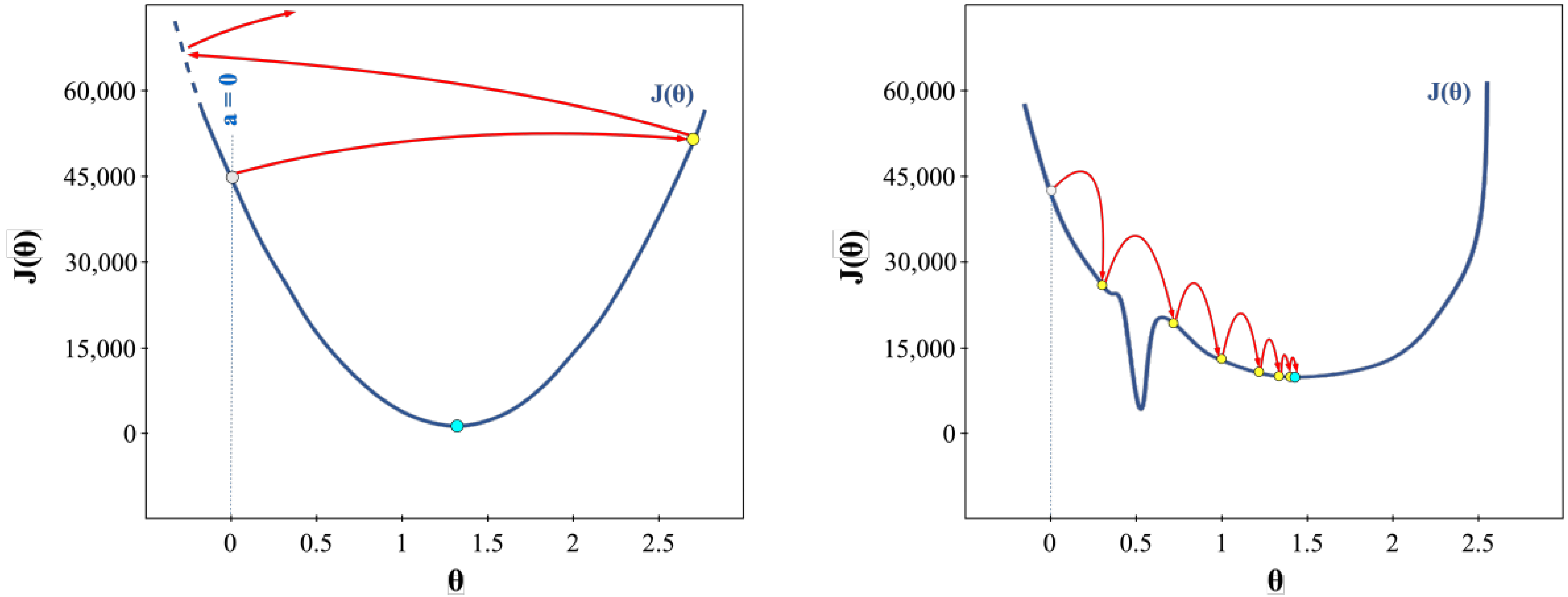




Discontinuous Functions

Optimization Algorithms

Learning rate



(a) Gradient descent missing global minimum on a convex cost function due to a very large learning rate.

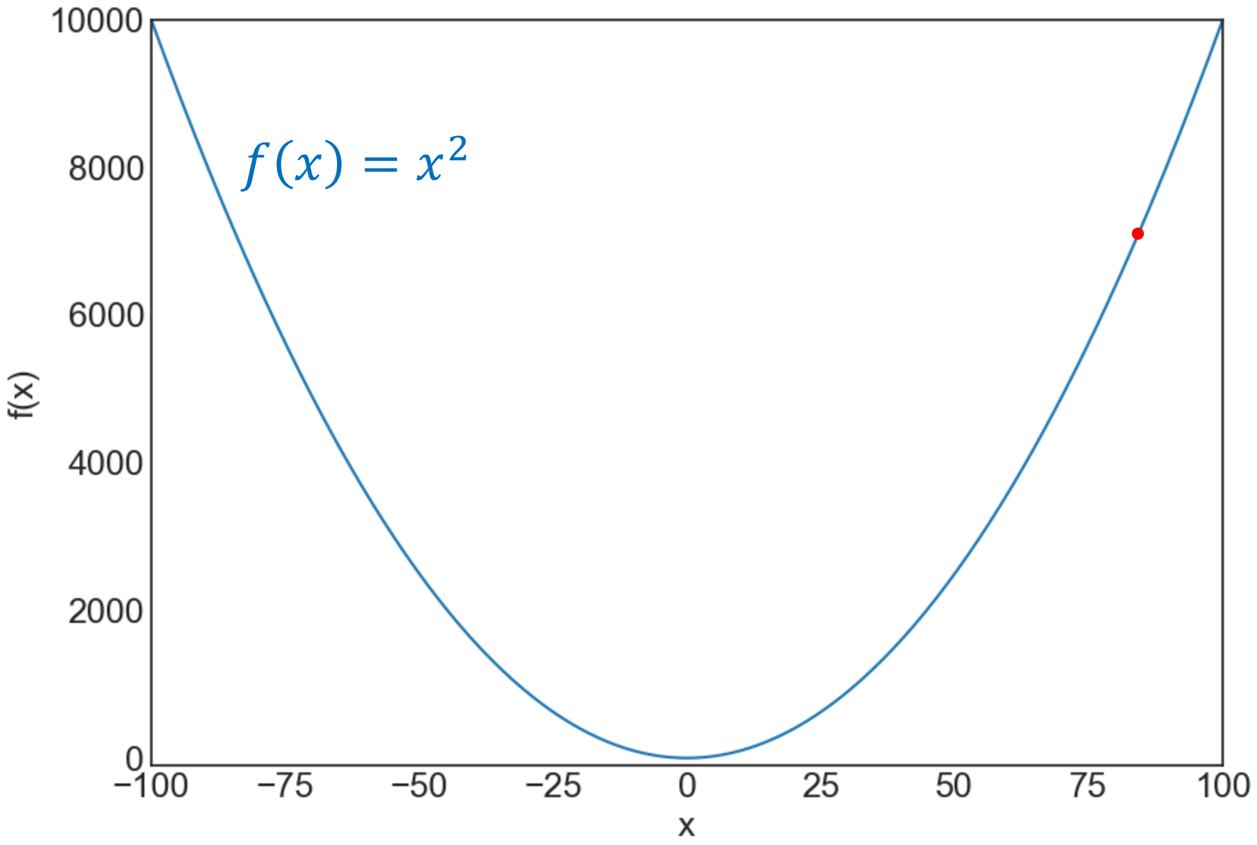
(b) Gradient Descent missing global minimum on a non-convex cost function due to a very large learning rate.

From "Machine Learning Simplified"

Gradient-based Optimization

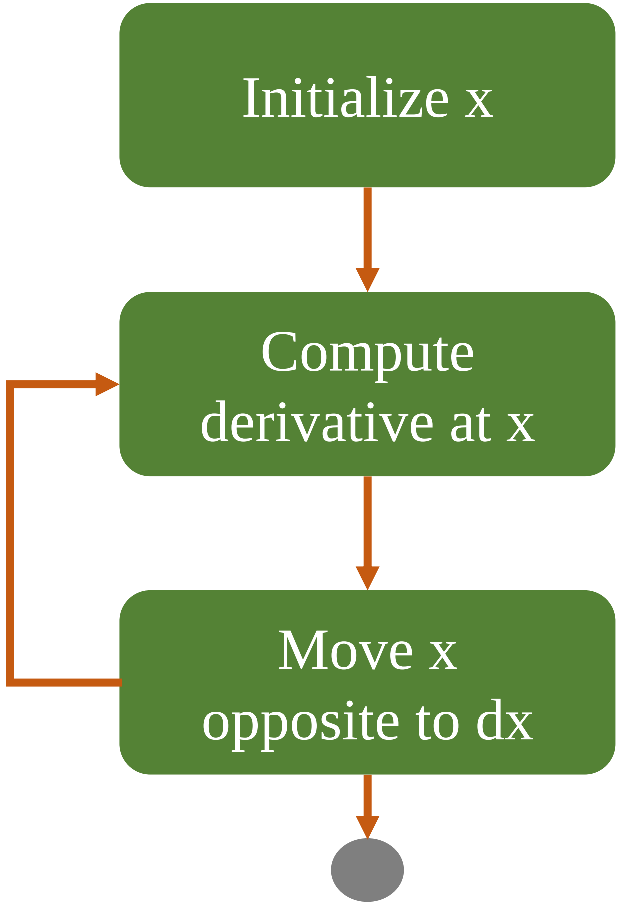
❖ Square function

$$x_t = x_{t-1} - \eta f'(x_{t-1})$$



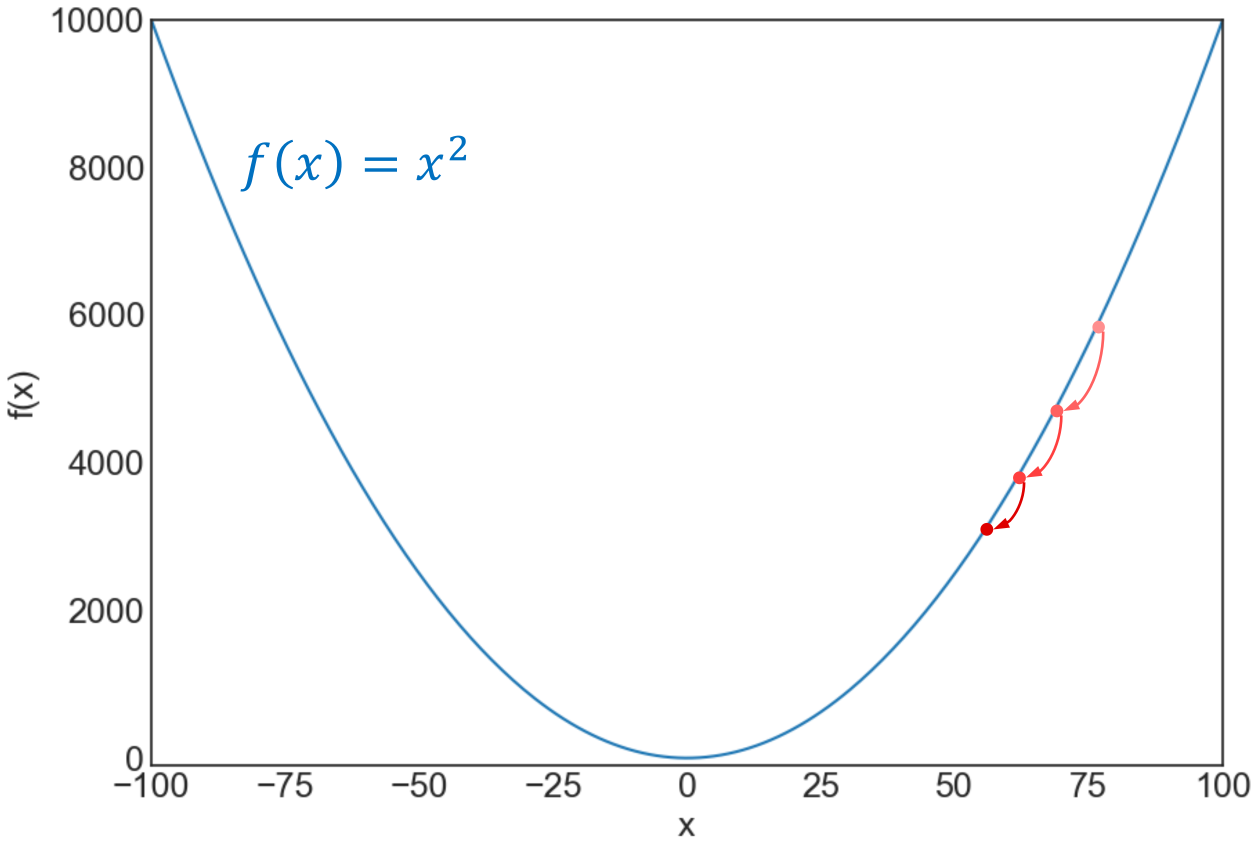
$$-100 \leq x \leq 100$$

$$x \in \mathbb{N}$$



Optimization

❖ Square function



$$-100 \leq x \leq 100$$
$$x \in \mathbb{N}$$

$$x_t = x_{t-1} - \eta f'(x_{t-1})$$

$x_0 = 70.0$ $\eta = 0.1$

$f'(x_0) = 140.0$

$x_1 = x_0 - \eta f'(x_0) = 56.0$

$f'(x_1) = 112.0$

$x_2 = x_1 - \eta f'(x_1) = 44.8$

$f'(x_2) = 89.6$

$x_3 = x_2 - \eta f'(x_2) = 35.84$

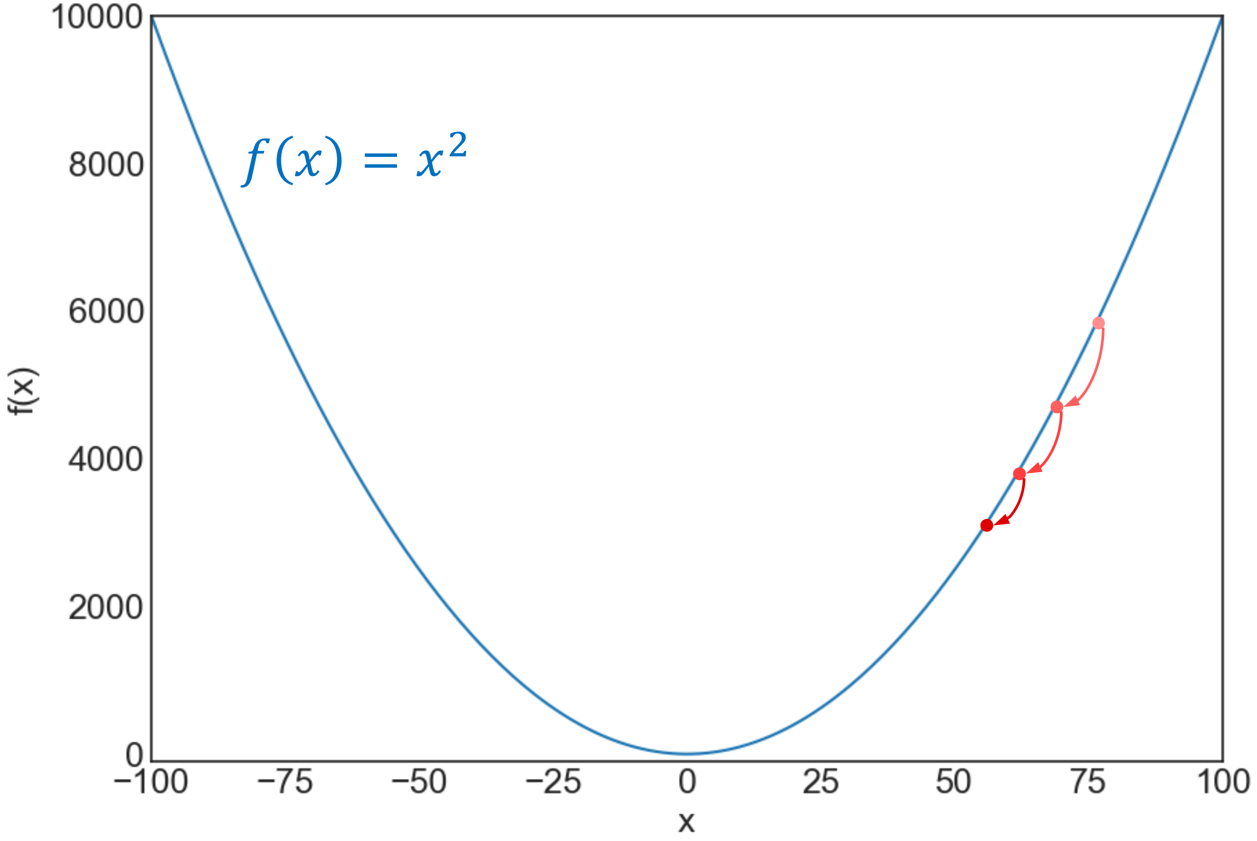
$f'(x_3) = 71.68$

$x_4 = x_3 - \eta f'(x_3) = 28.672$

Optimization

$$x_t = x_{t-1} - \eta f'(x_{t-1})$$

❖ Square function



Keep doing

$x_{10} = 6.012 \qquad \eta = 0.1$

$f'(x_{10}) = 12.02$

$x_{11} = x_{10} - \eta f'(x_{10}) = 4.81$

$f'(x_{11}) = 9.62$

$x_{12} = x_{11} - \eta f'(x_{11}) = 3.84$

$f'(x_{12}) = 7.69$

$x_{13} = x_{12} - \eta f'(x_{12}) = 3.078$

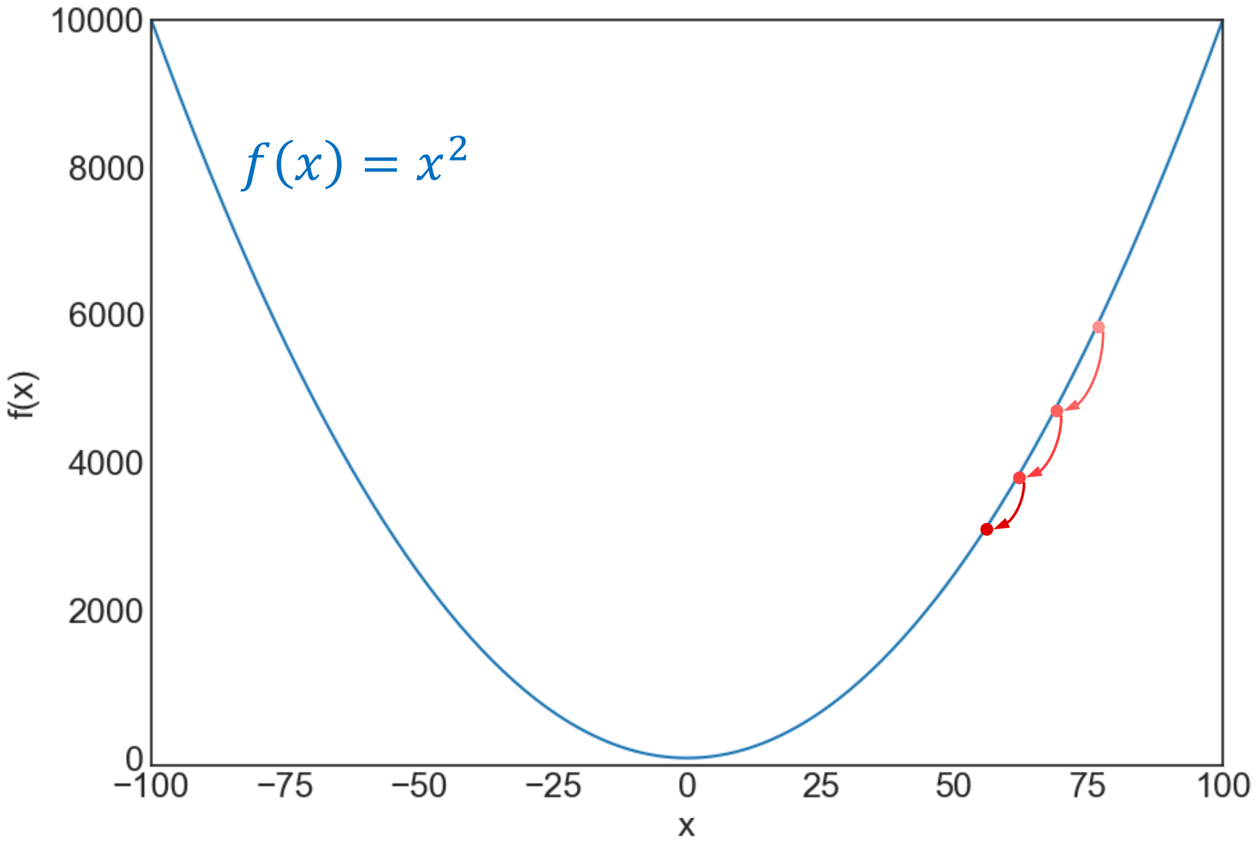
$f'(x_{13}) = 6.15$

$x_{14} = x_{13} - \eta f'(x_{13}) = 2.46$

Optimization

$$x_t = x_{t-1} - \eta f'(x_{t-1})$$

❖ Square function



Keep doing

$x_{30} = 0.069$ $\eta = 0.1$

$f'(x_{30}) = 0.138$

$x_{31} = x_{30} - \eta f'(x_{30}) = 0.055$

$\mathrm{f}'(\mathrm{x}_{31}) = 0.11$

$x_{32} = x_{31} - \eta f'(x_{31}) = 0.044$

$\mathrm{f}'(\mathrm{x}_{32}) = 0.88$

$x_{33} = x_{32} - \eta f'(x_{32}) = 0.035$

$\mathrm{f}'(\mathrm{x}_{34}) = 0.071$

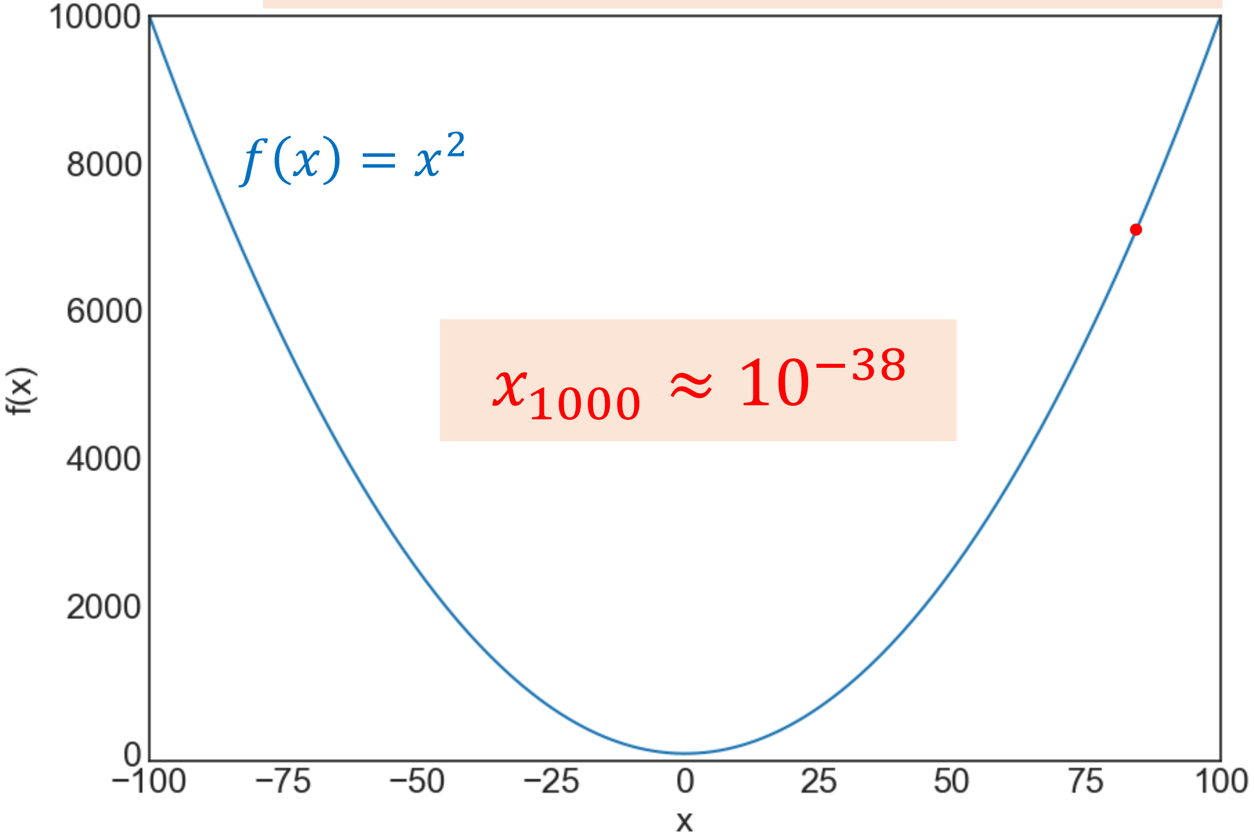
$x_{34} = x_{33} - \eta f'(x_{33}) = 0.028$

AI VIETNAM
All-in-One Course

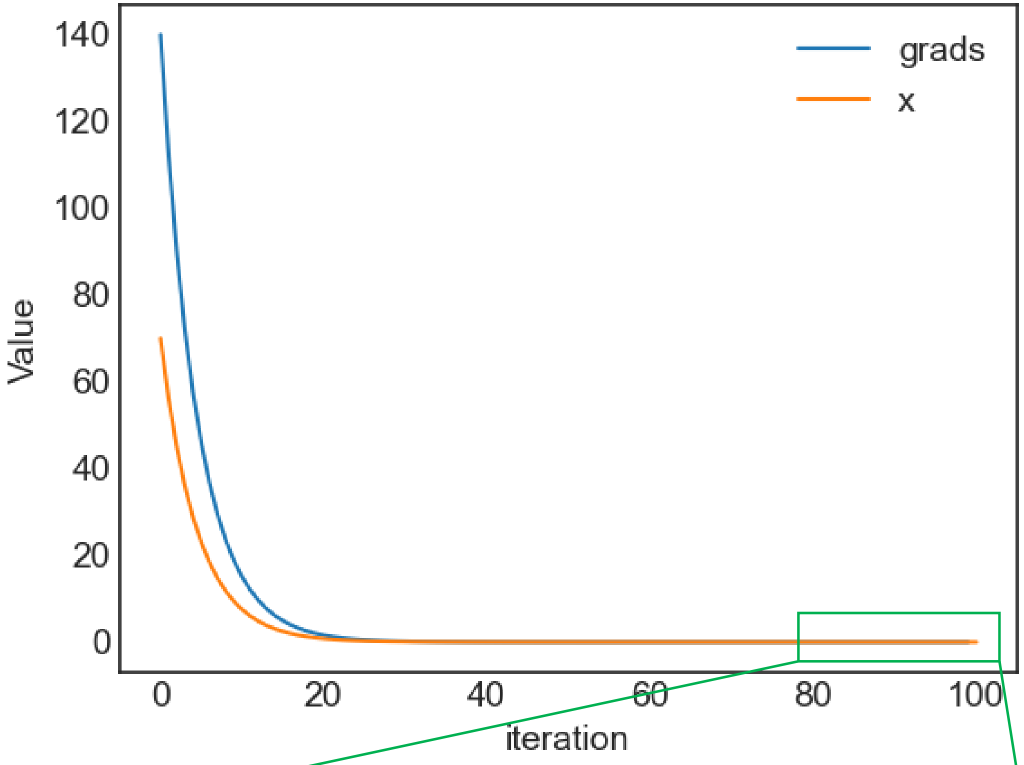
Optimization

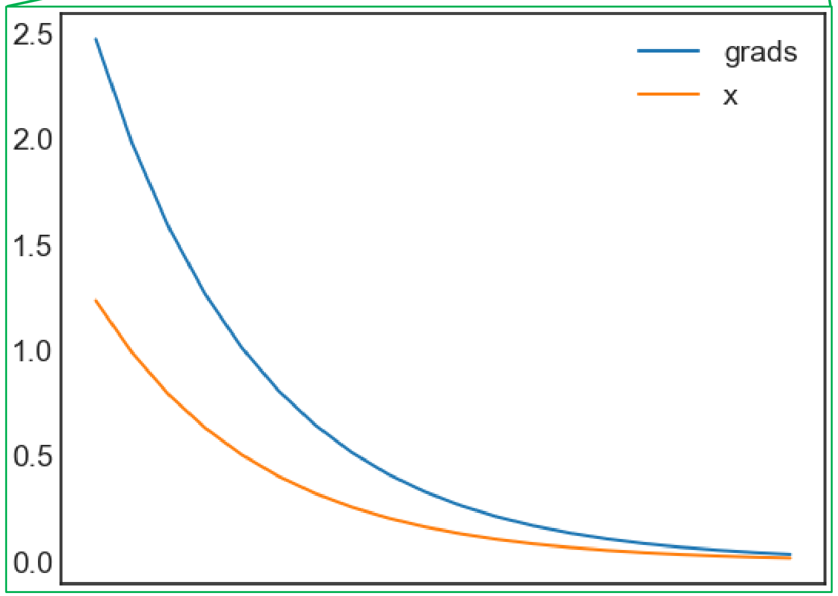
❖ Square function

$$x_t = x_{t-1} - \eta f'(x)$$



$$x_{1000} \approx 10^{-38}$$





Optimized successfully!

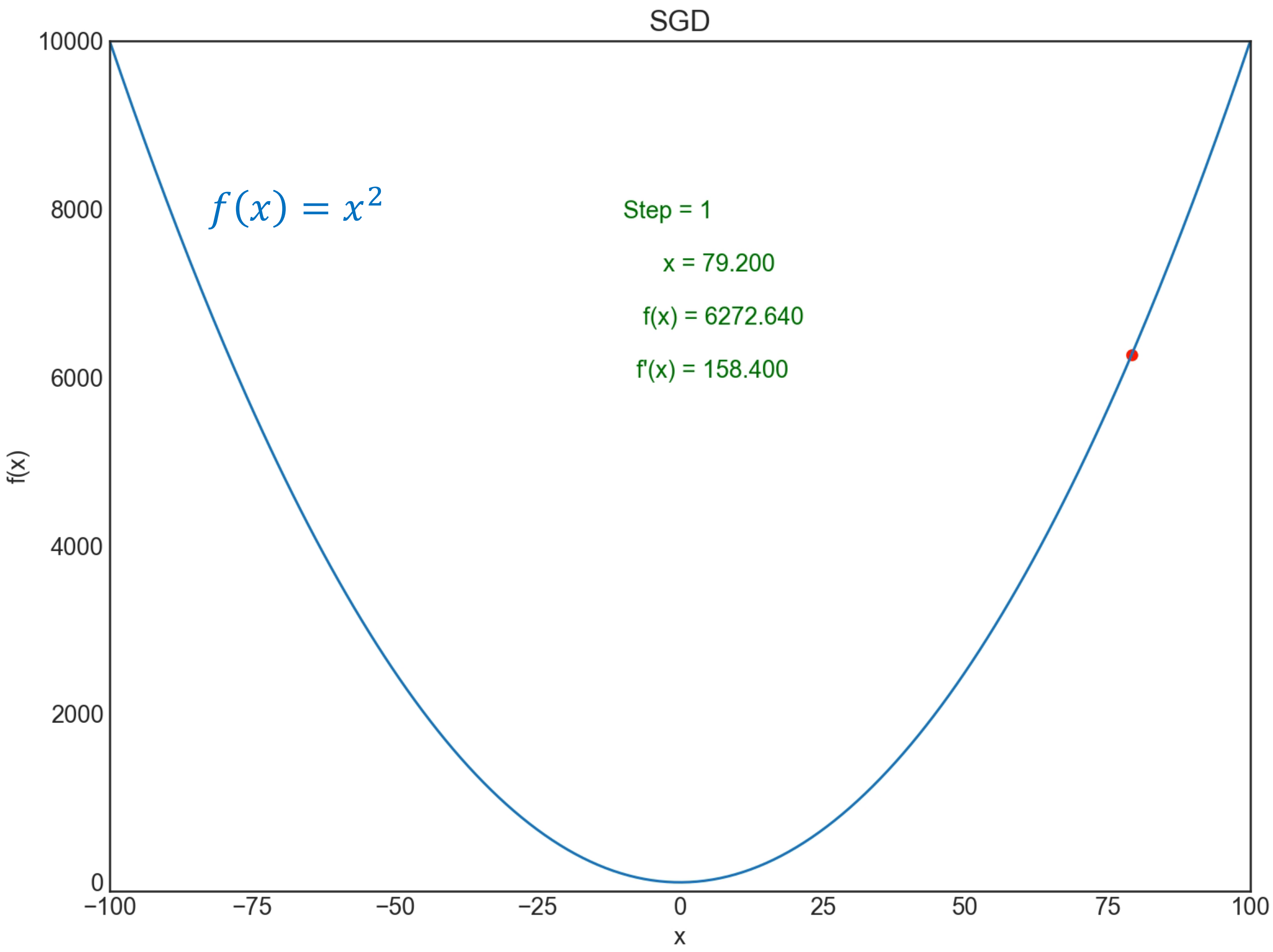
Optimization

❖ **Square function**

$$x_0 = 99.0$$

$$\eta = 0.1$$

$$x_t = x_{t-1} - \eta f'(x)$$



Optimization

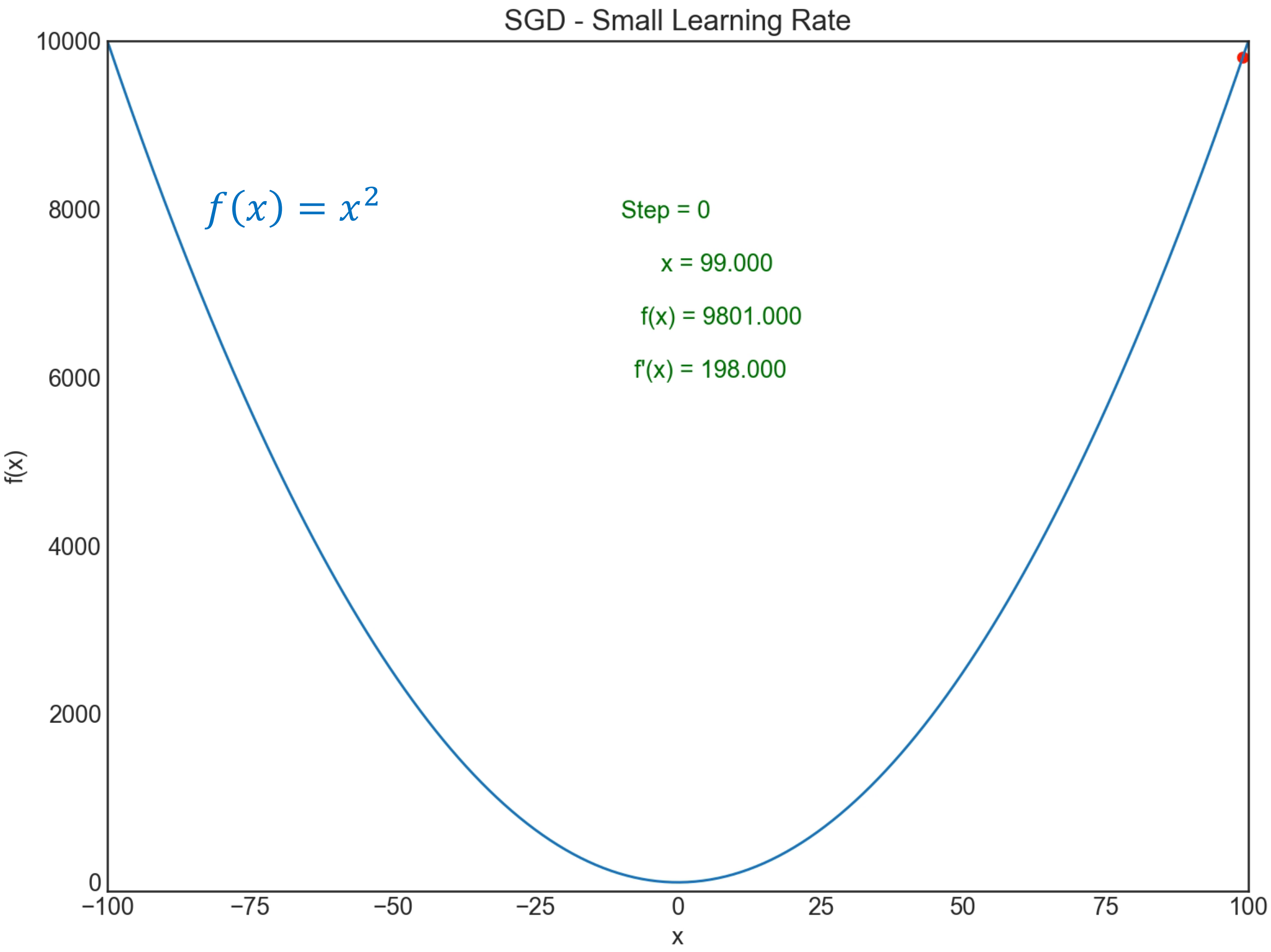
❖ **Square function**

Discussion

$$x_0 = 99.0$$

$$\eta = 0.001$$

$$x_t = x_{t-1} - \eta f'(x)$$



Optimization

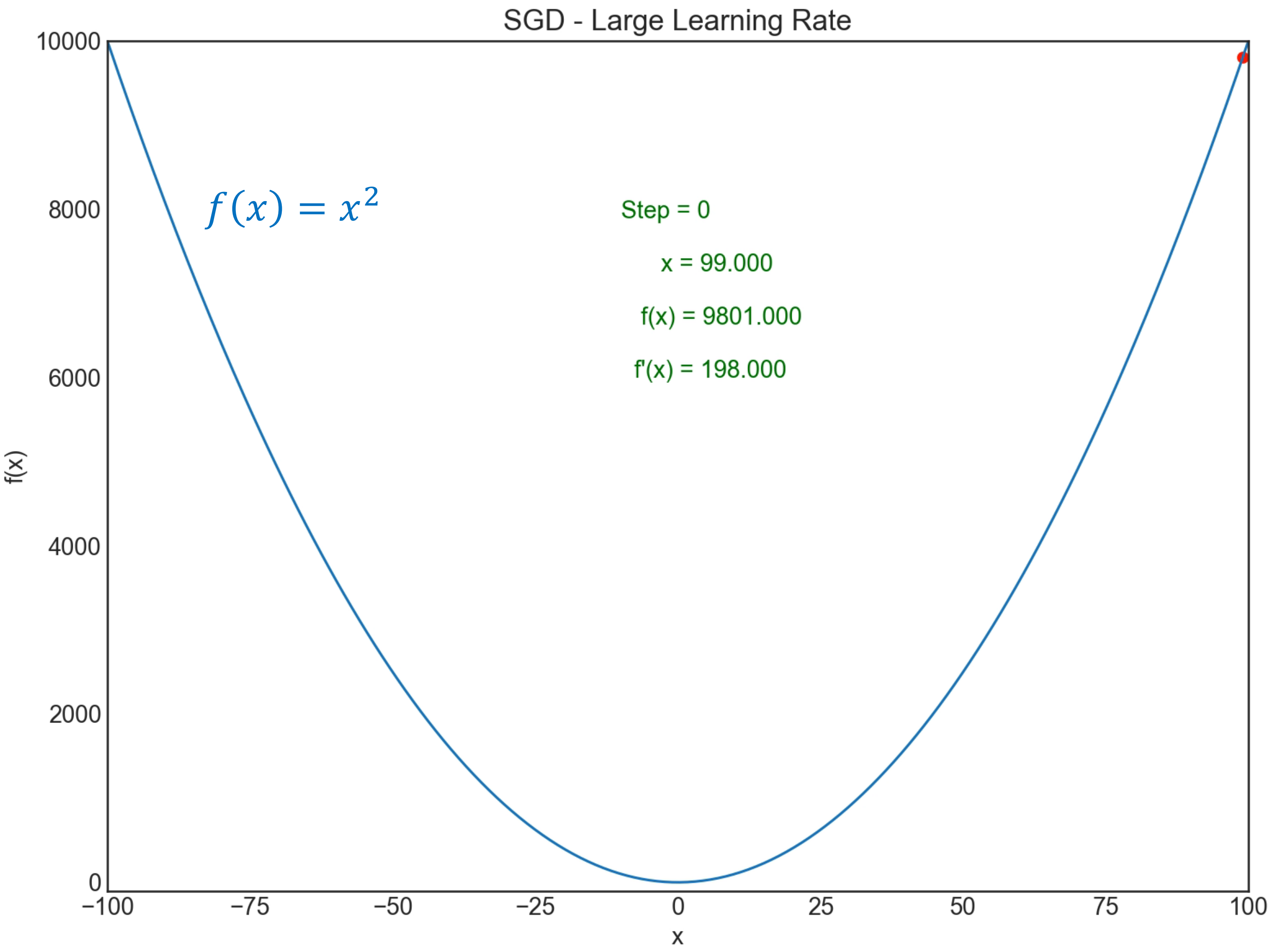
❖ **Square function**

Discussion

$$x_0 = 99.0$$

$$\eta = 0.8$$

$$x_t = x_{t-1} - \eta f'(x)$$



Optimization

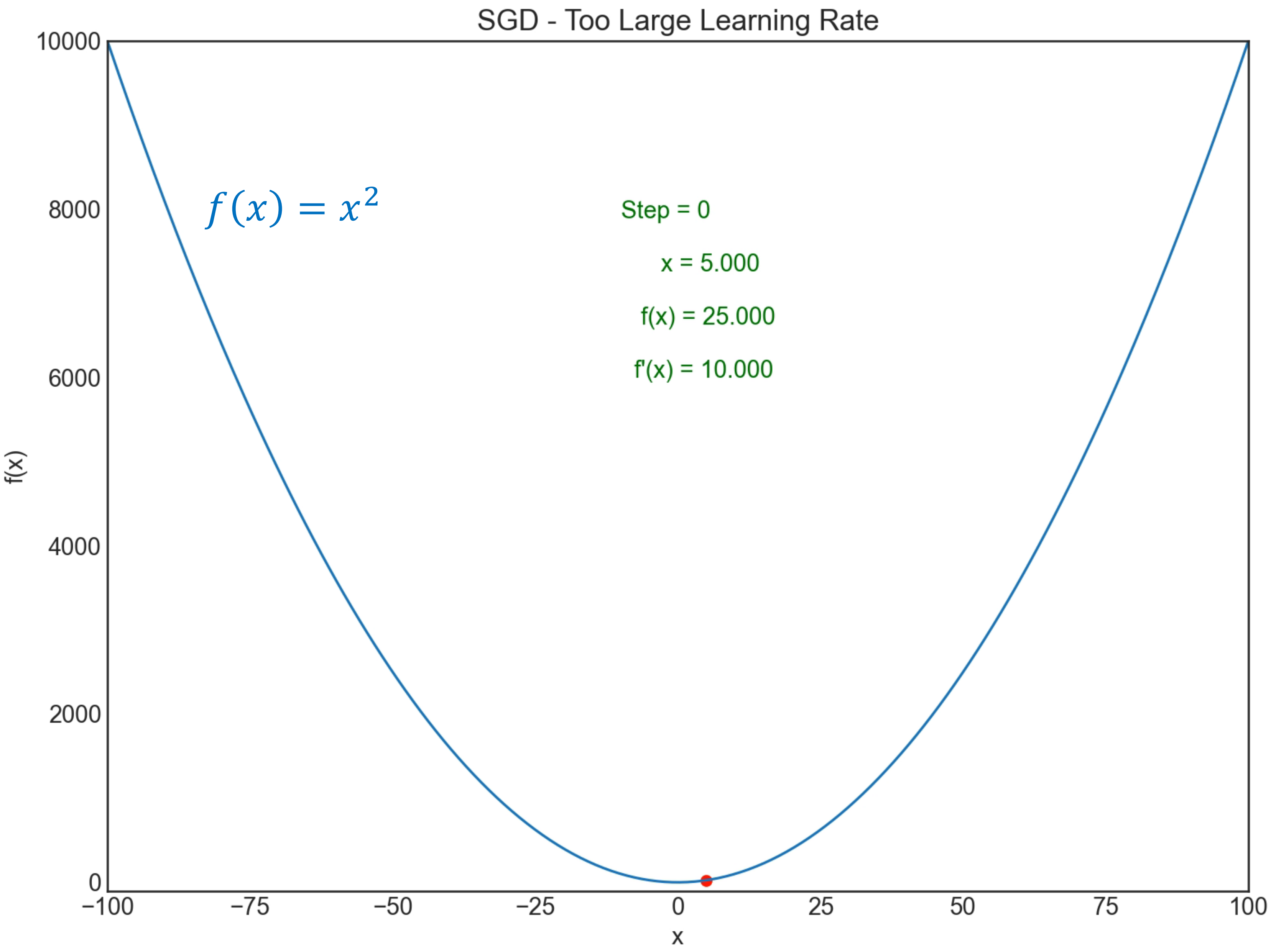
❖ **Square function**

Discussion

$$x_0 = 99.0$$

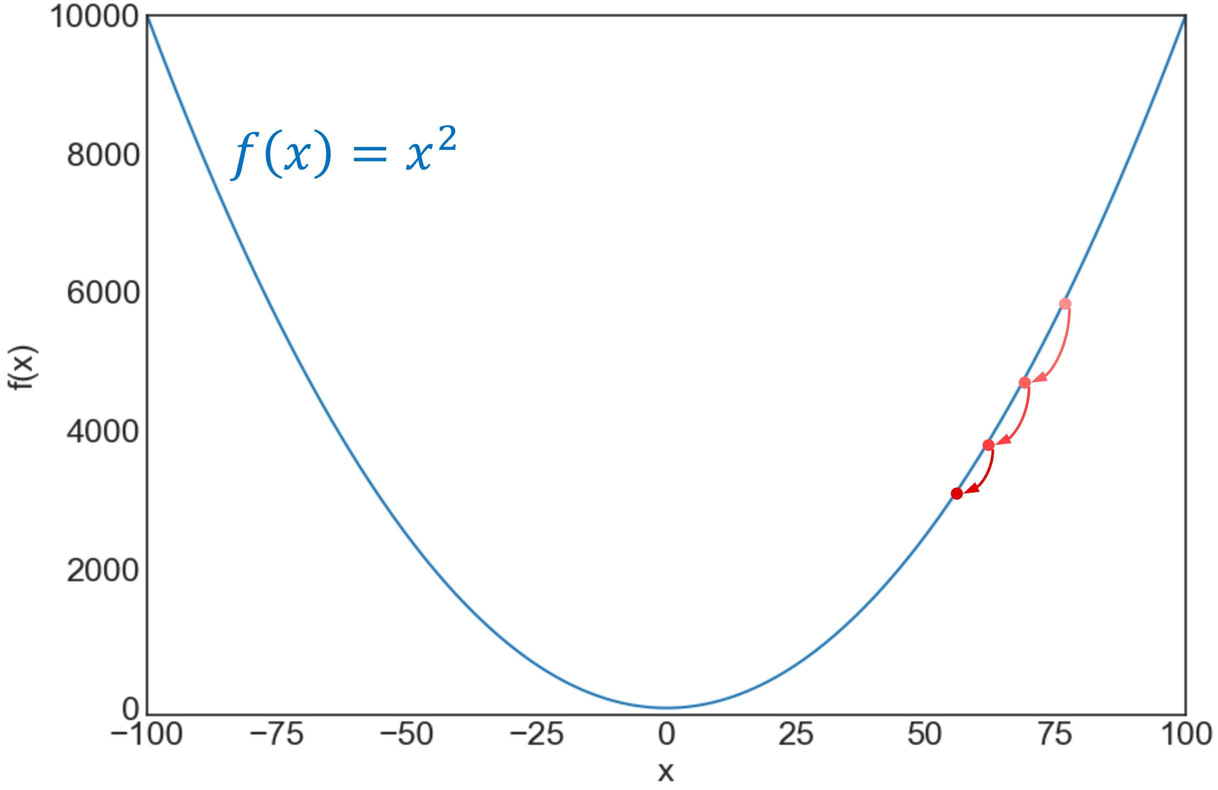
$$\eta = 1.1$$

$$x_t = x_{t-1} - \eta f'(x)$$



Optimization

❖ Square function



$$x_t = x_{t-1} - \eta f'(x_{t-1})$$

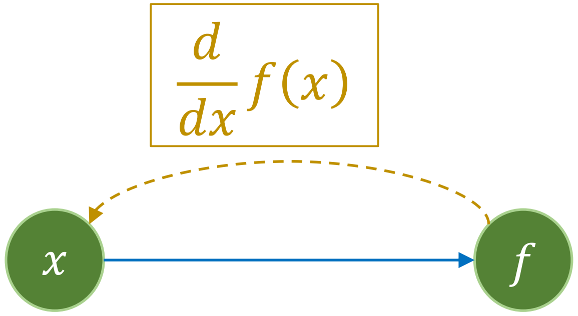
$x_{30} = 0.069$ $\eta = 0.1$

$f'(x_{30}) = 0.138$

$x_{31} = x_{30} - \eta f'(x_{30}) = 0.055$

$f'(x_{31}) = 0.11$

$x_{32} = x_{31} - \eta f'(x_{31}) = 0.044$



- Given a function f(x), find optimal x_{opt} so that $f(x_{opt})$ is minimum
- After an update, $f(x_{new}) \leq f(x_{old})$

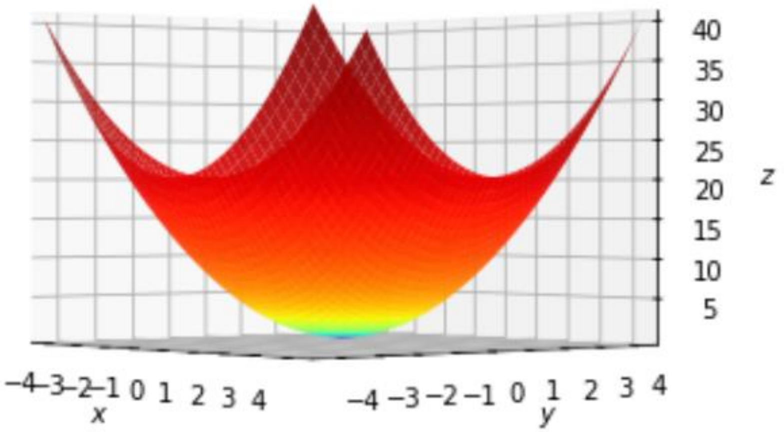
Optimization

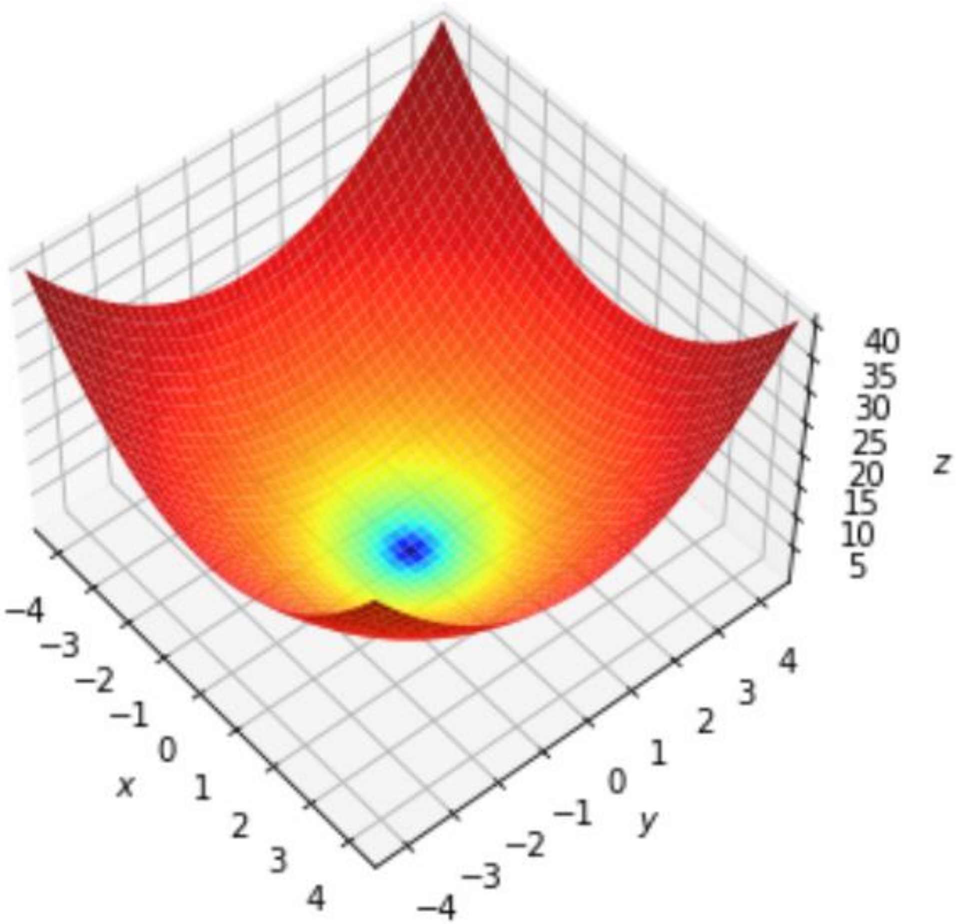
❖ Optimization: 2D function

$$f(x, y) = x^2 + y^2$$

$$-100 \leq x, y \leq 100$$

$$x, y \in \mathbb{N}$$





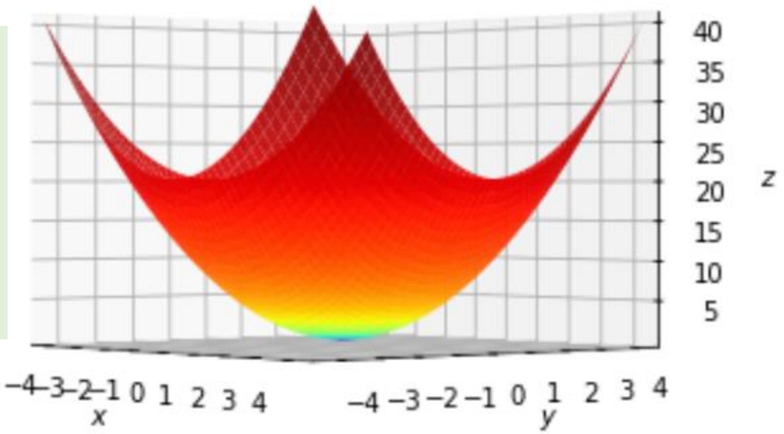
Optimization

❖ Optimization: 2D function

$$f(x, y) = x^2 + y^2$$

$$-100 \leq x, y \leq 100$$

$$x, y \in \mathbb{N}$$



$$x = x - \eta \frac{\partial f(x, y)}{\partial x}$$

$$y = y - \eta \frac{\partial f(x, y)}{\partial y}$$

$$x_0 = 6.0 \qquad y_0 = 9.0 \qquad \eta = 0.1$$

$$\frac{\partial f(x_0, y_0)}{\partial x} = 12 \qquad \frac{\partial f(x_0, y_0)}{\partial y} = 18$$

$$x_1 = 4.8 \qquad y_1 = 7.2$$

$$\frac{\partial f(x_1, y_1)}{\partial x} = 9.6 \qquad \frac{\partial f(x_1, y_1)}{\partial y} = 14.4$$

$$x_2 = 3.84 \qquad y_2 = 5.75$$

$$\frac{\partial f(x_2, y_2)}{\partial x} = 7.68 \qquad \frac{\partial f(x_2, y_2)}{\partial y} = 11.51$$

$$x_3 = 3.07 \qquad y_3 = 4.608$$

$$\frac{\partial f(x_3, y_3)}{\partial x} = 6.14 \qquad \frac{\partial f(x_3, y_3)}{\partial y} = 9.21$$

$$x_4 = 2.45 \qquad y_4 = 3.68$$

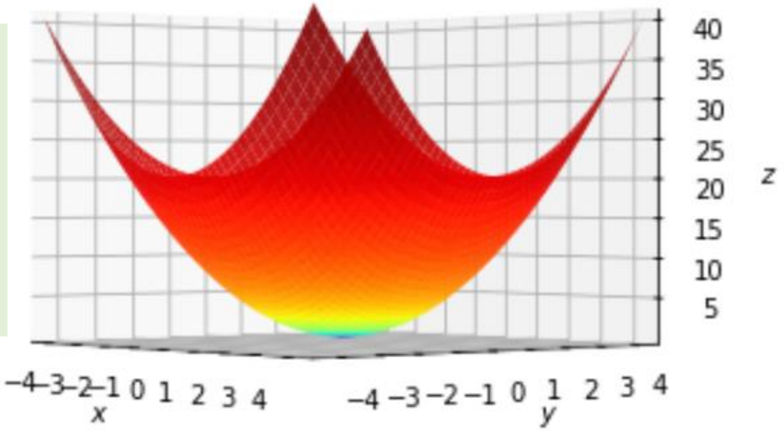
Optimization

❖ Optimization: 2D function

$$f(x, y) = x^2 + y^2$$

$$-100 \leq x, y \leq 100$$

$$x, y \in \mathbb{N}$$

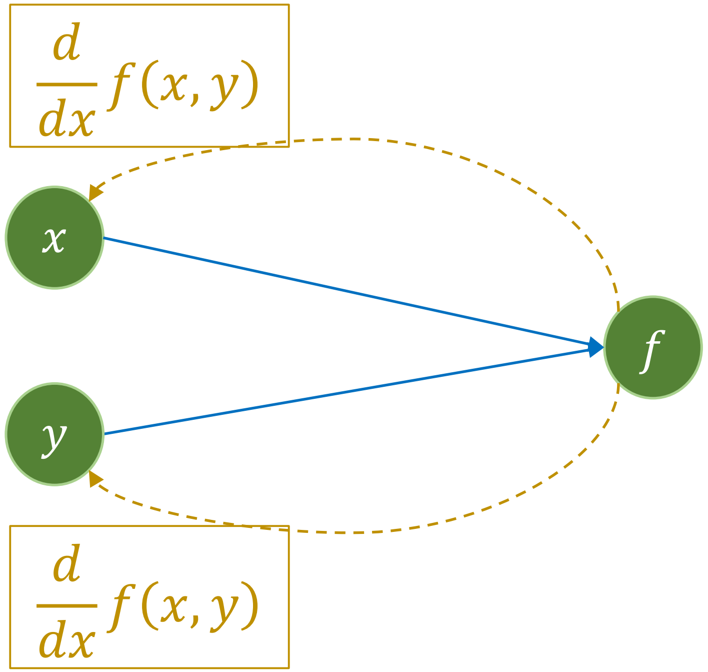


$$\frac{\partial f(x_2, y_2)}{\partial x} = 7.68 \qquad \frac{\partial f(x_2, y_2)}{\partial y} = 11.51$$

$$x_3 = 3.07 \qquad y_3 = 4.608$$

$$\frac{\partial f(x_3, y_3)}{\partial x} = 6.14 \qquad \frac{\partial f(x_3, y_3)}{\partial y} = 9.21$$

$$x_4 = 2.45 \qquad y_4 = 3.68$$



Summary:

- Given a function f(x, y), find optimal (x_{opt}, y_{opt}) so that f(x_{opt}, y_{opt}) is minimum

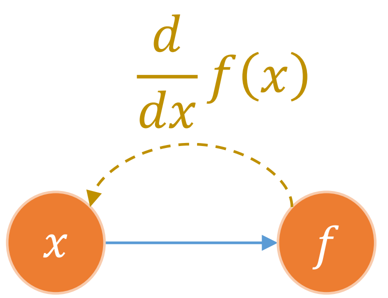
- After an update, f(x_{new}, y_{new}) ≤ f(x_{old}, y_{old})

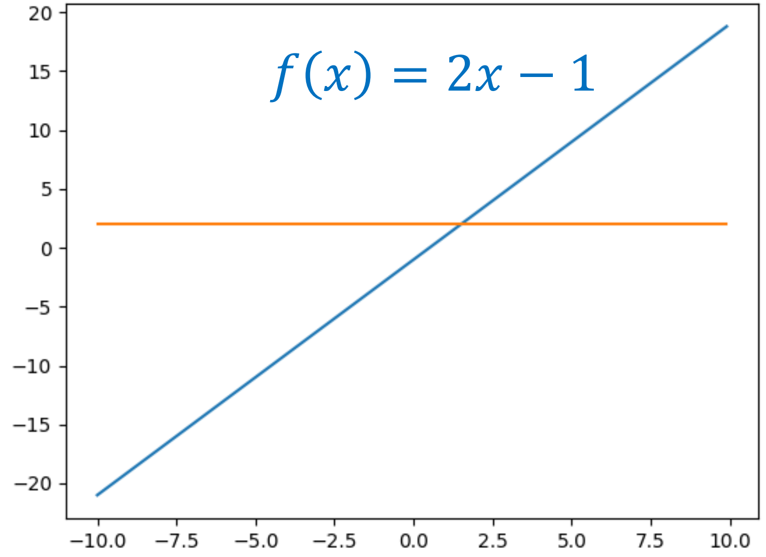
Outline

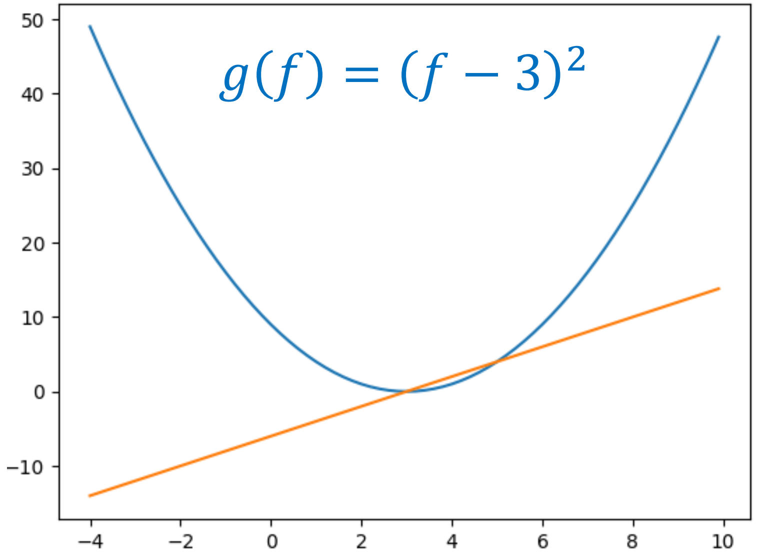
- Review on Optimization
- Partial Gradient and Chain Rules
- Finding a Line
- Linear Regression

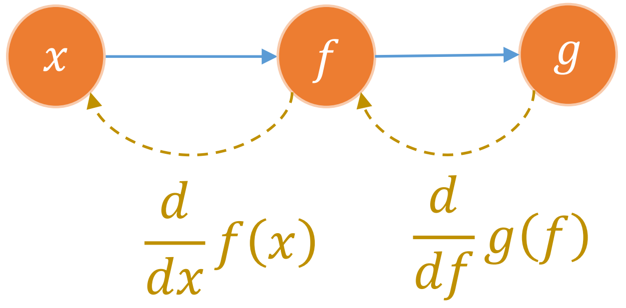
Gradient-based Optimization

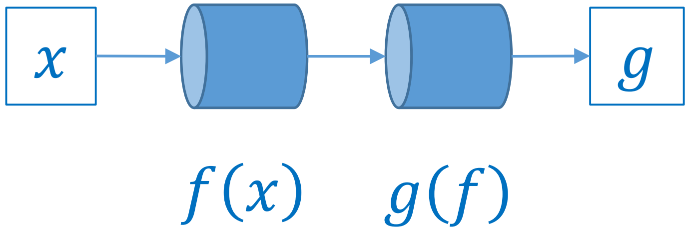
❖ For composite function

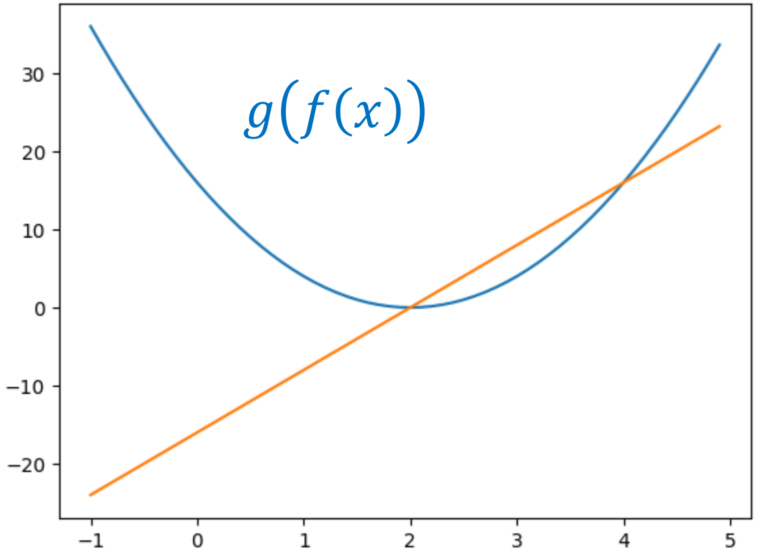








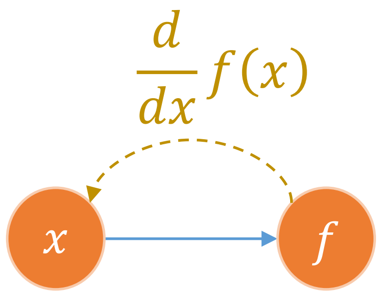


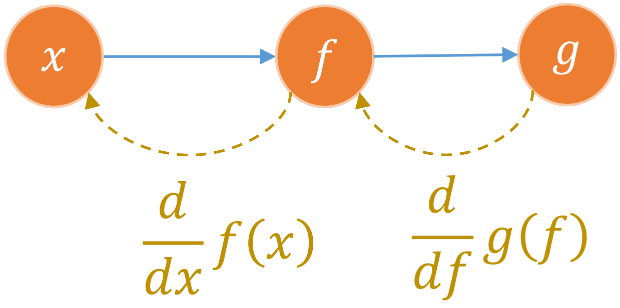


$$\frac{d}{dx}g(f(x)) = \left[\frac{d}{df}g(f)\right] * \left[\frac{d}{dx}f(x)\right]$$

Gradient-based Optimization

❖ For composite function





$$\frac{d}{dx}g\big(f(x)\big) = \left[\frac{d}{df}g(f)\right] * \left[\frac{d}{dx}f(x)\right]$$

$$f(x) = 2x - 1$$

$$g(f) = (f - 3)^2$$

➡

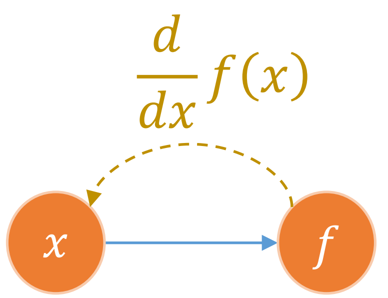
$$g(x) = (2x - 1\ - 3)^2$$
$$= (2x\ - 4)^2$$

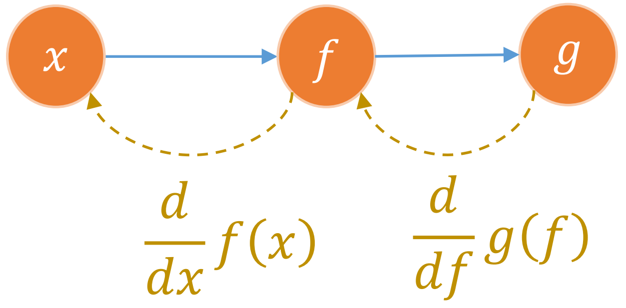
➡

$$g'(x) = 4(2x - 4)$$
$$= 8x - 16$$

Gradient-based Optimization

❖ **For composite function and chain rule**





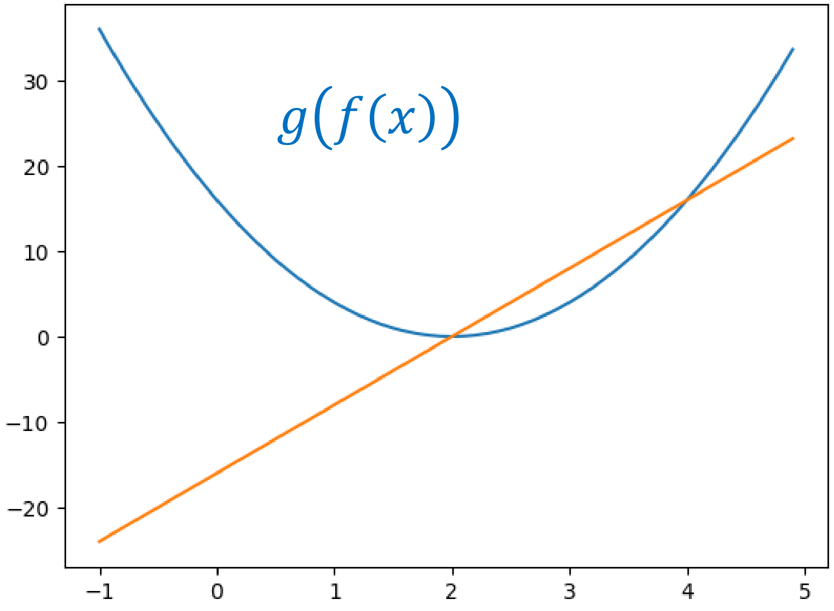
$$\frac{d}{dx}g\big(f(x)\big) = \left[\frac{d}{df}g(f)\right] * \left[\frac{d}{dx}f(x)\right]$$

$$f(x) = 2x - 1$$
$$g(f) = (f - 3)^2$$

$$f'(x) = 2$$
$$g'(f) = 2(f - 3)$$

$$\begin{aligned} \frac{dg}{dx} &= \frac{dg}{df}\frac{df}{dx} \\ &= 2(f-3)2 \\ &= 4(2x-1-3) \\ &= 8x - 16 \end{aligned}$$

Implementation



$$f(x) = 2x - 1$$

$$g(f) = (f - 3)^2$$

$$\frac{dg}{dx} = 8x - 16$$

```python
def fx(x):
    return 2*x - 1

def gf(f):
    return (f-3)**2

def dg_dx(x):
    return 8*x - 16
```

```python
import random

# parameters
lr = 0.1

# initialize x
x = 60

old_loss = gf(fx(x)) # logging
print(f'old_loss: {old_loss}')

# compute derivative
dg_dx_value = dg_dx(x)

# update
x = x - lr*dg_dx_value

new_loss = gf(fx(x)) # logging
print(f'new_loss: {new_loss}')
```

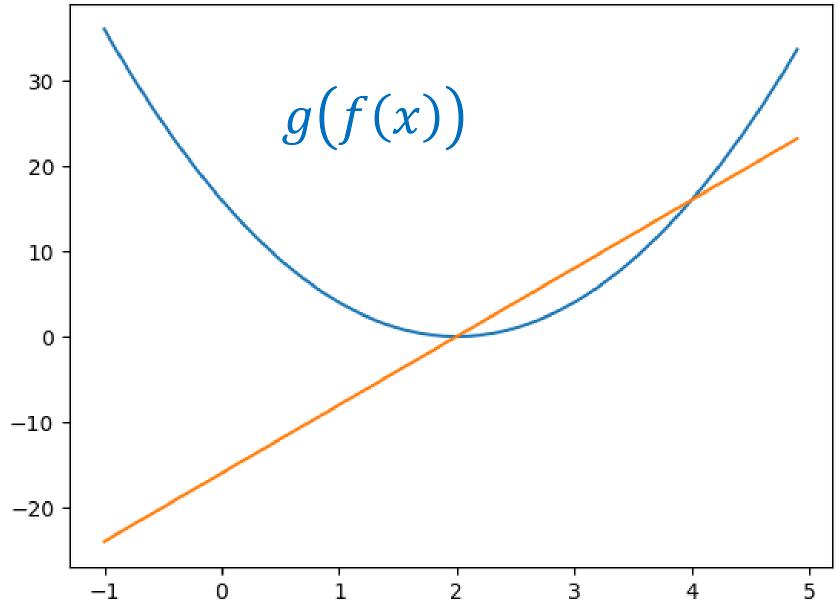
```
old_loss: 13456
new_loss: 538.2399999999994
```

Implementation

$$f(x) = 2x - 1$$

$$g(f) = (f - 3)^2$$

$$\frac{dg}{dx} = 8x - 16$$




```python
def fx(x):
    return 2*x - 1

def gf(f):
    return (f-3)**2

def dg_dx(x):
    return 8*x - 16
```

```python
import random

# parameters
num_steps = 5
lr = 0.1

# set x randomly
x = random.randint(-100, 100)

for _ in range(num_steps):
    # logging
    loss = gf(fx(x))

    # compute derivative
    dg_dx_value = dg_dx(x)

    # update
    x = x - lr*dg_dx_value
```

Gradient-based Optimization

❖ For composite function

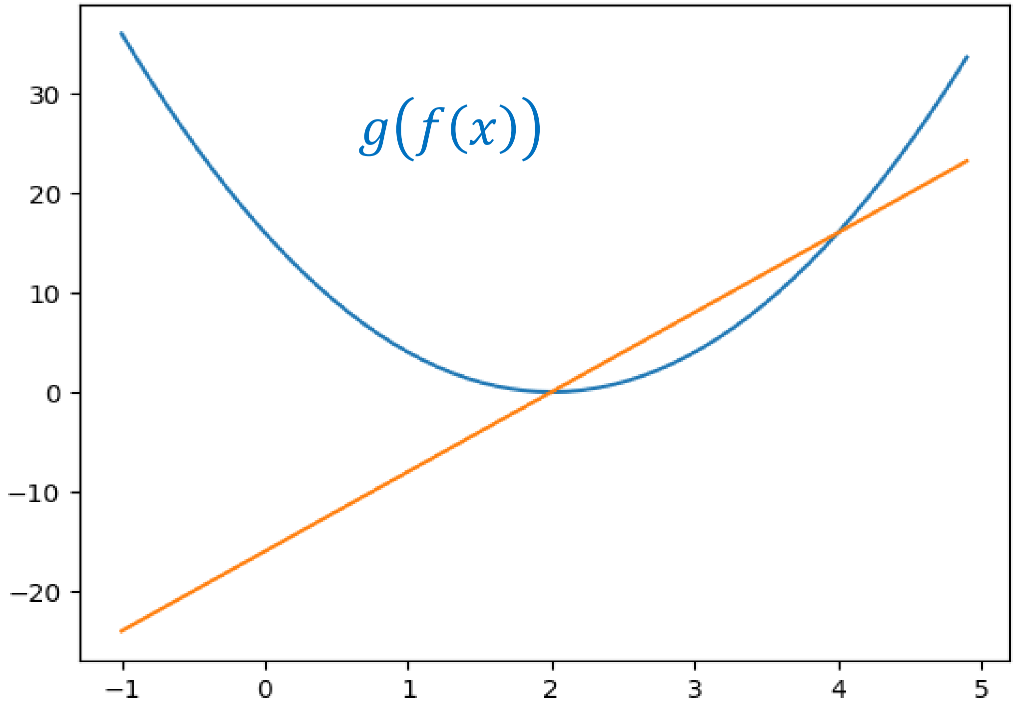
$$\frac{d}{dx}f(x)$$

$$x \rightarrow f$$

$$x \rightarrow f \rightarrow g$$

$$\frac{d}{dx}f(x) \qquad \frac{d}{df}g(f)$$

$$\frac{d}{dx}g\big(f(x)\big) = \left[\frac{d}{df}g(f)\right] * \left[\frac{d}{dx}f(x)\right]$$



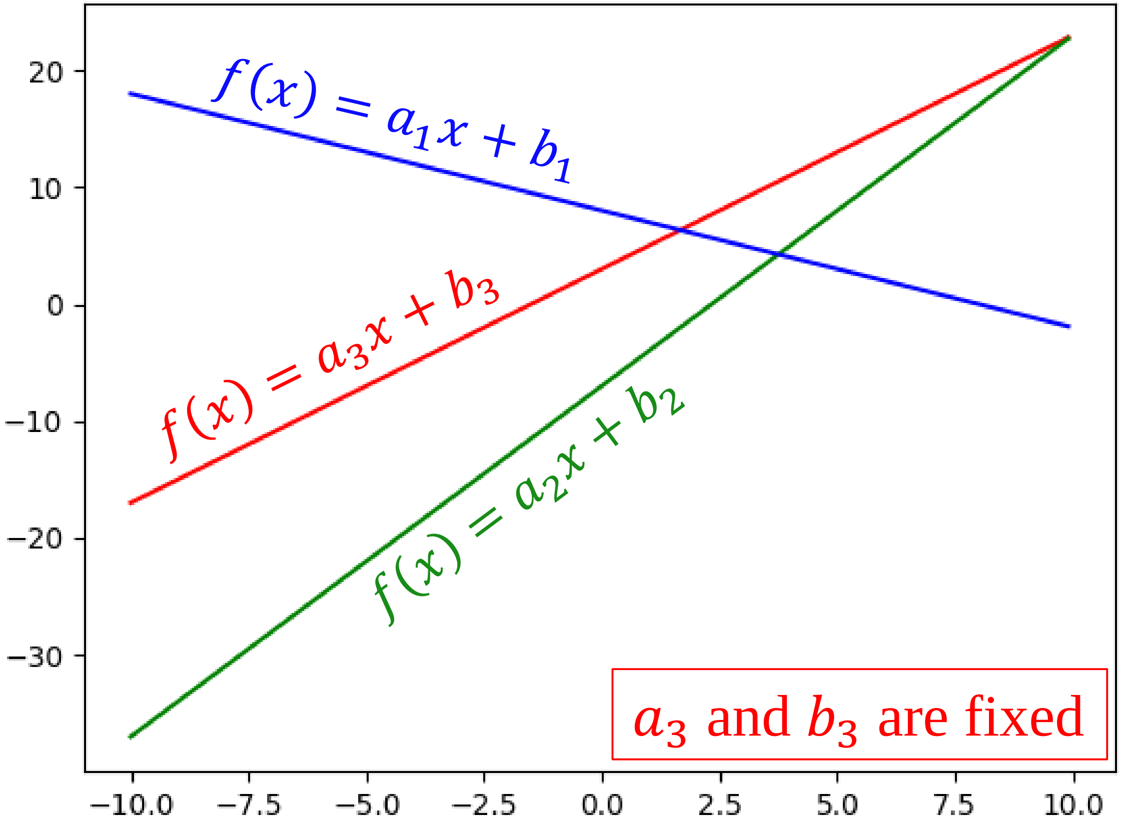
- Given a function g(f(x)), find optimal x_{opt} so that $f(x_{opt})$ is minimum

- After an update, $g(f(x_{new})) \leq g(f(x_{old}))$

Gradient-based Optimization

❖ Another context

$$f(x) = ax + b$$



❖ Constraints

$$x_t = x_{t-1} - \eta f'(x_{t-1})$$

x → The red line (Black box) → y

How to measure the distance between the red line and an predicted line?

Initialize a, b

Compute partial gradient at a, b

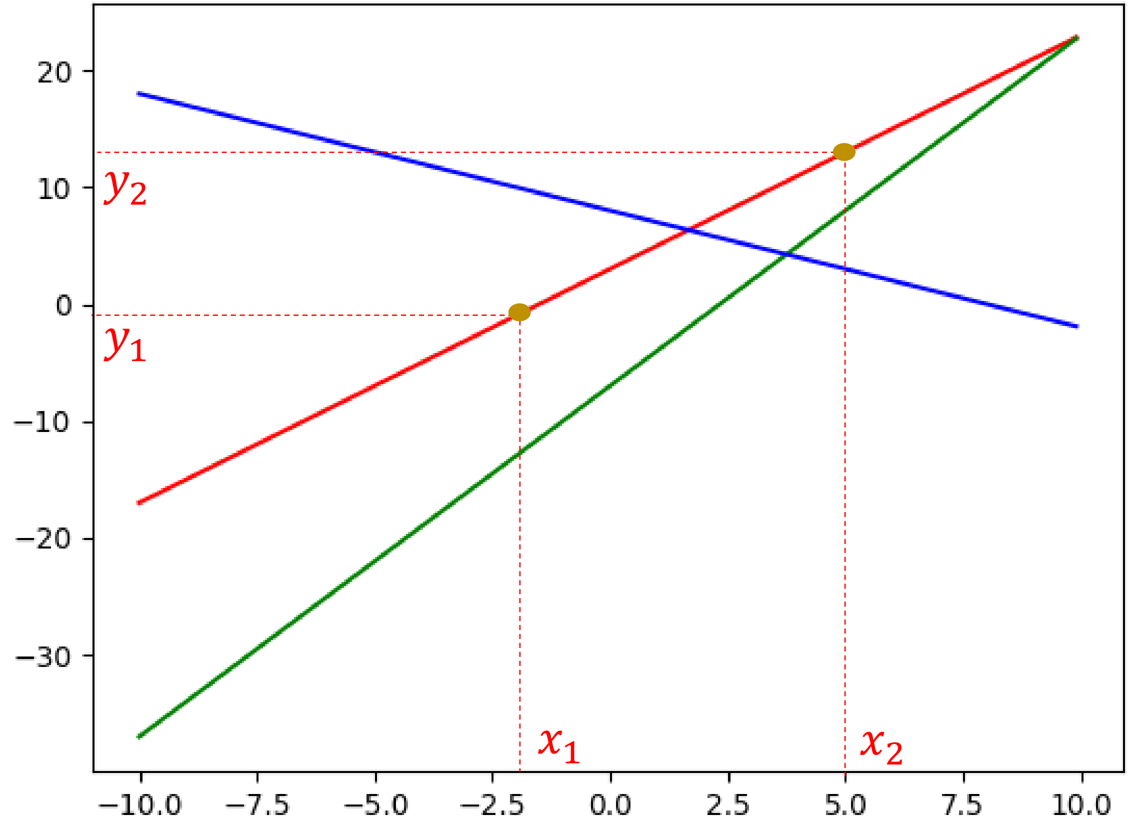
Move a, b opposite to db, db

$\eta = 0.1$

Gradient-based Optimization

❖ Another context

$$f(x) = ax + b$$



❖ Constraints

$$x_t = x_{t-1} - \eta f'(x_{t-1})$$

x → The red line (Black box) → y

$(x_1$=-2, y_1=-1)

$(x_2$=5, y_2=13)

$$g(f) = (f - y_i)^2$$

Initialize a, b

Compute partial gradient at a, b

Move a, b opposite to db, db

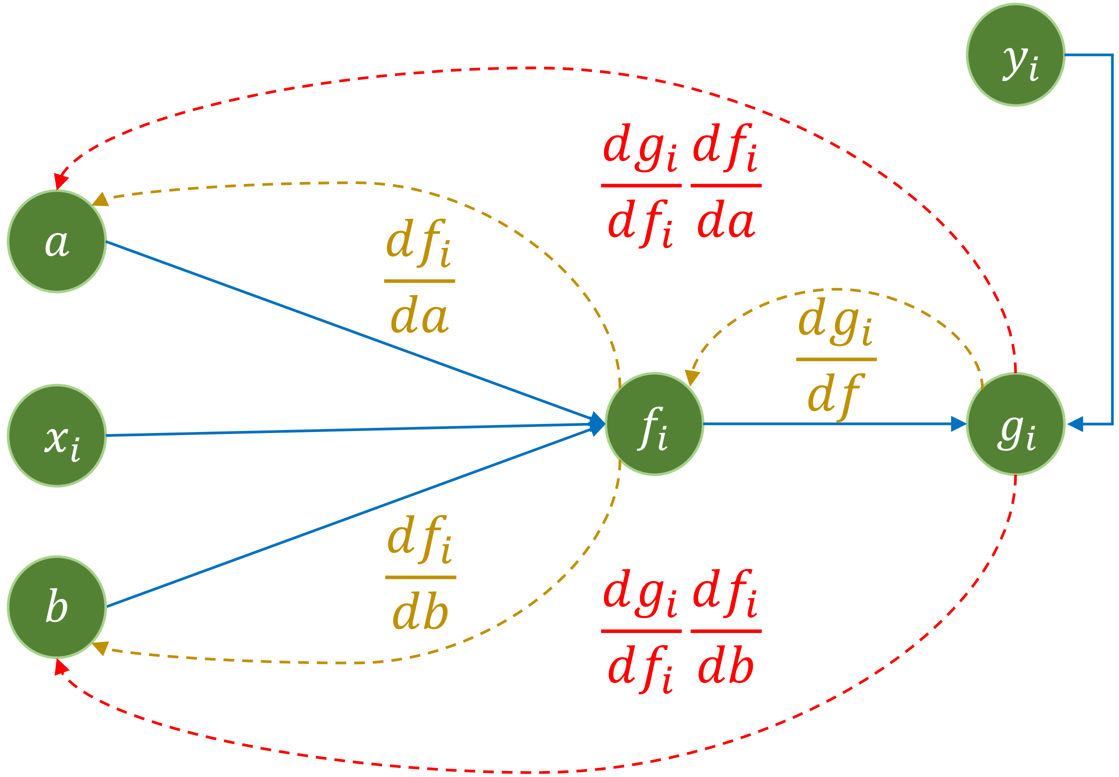
$\eta = 0.1$

Gradient-based Optimization

❖ Equations for partial gradients

$$f(x_i) = ax_i + b \qquad (x_1{=}1, y_1{=}5)$$

$$g(f) = (f - y_i)^2 \qquad (x_2{=}2, y_2{=}7)$$



$$\frac{df}{da} = x \qquad \frac{df}{db} = 1$$

$$\frac{dg}{df} = 2(f - y)$$

$$\frac{dg}{da} = \frac{dg}{df}\frac{df}{da} = 2x(f - y)$$

$$\frac{dg}{db} = \frac{dg}{df}\frac{df}{db} = 2(f - y)$$

During looking for optimal a and b, at a given time, a and b have concrete values

❖ Optimization for a composite function

Find a and b so that g(f(x)) is minimum

$f(x_i) = ax_i + b$ $\quad (x_1{=}1, y_1{=}5)$

$g(f) = (f - y_i)^2$ $\quad (x_2{=}2, y_2{=}7)$

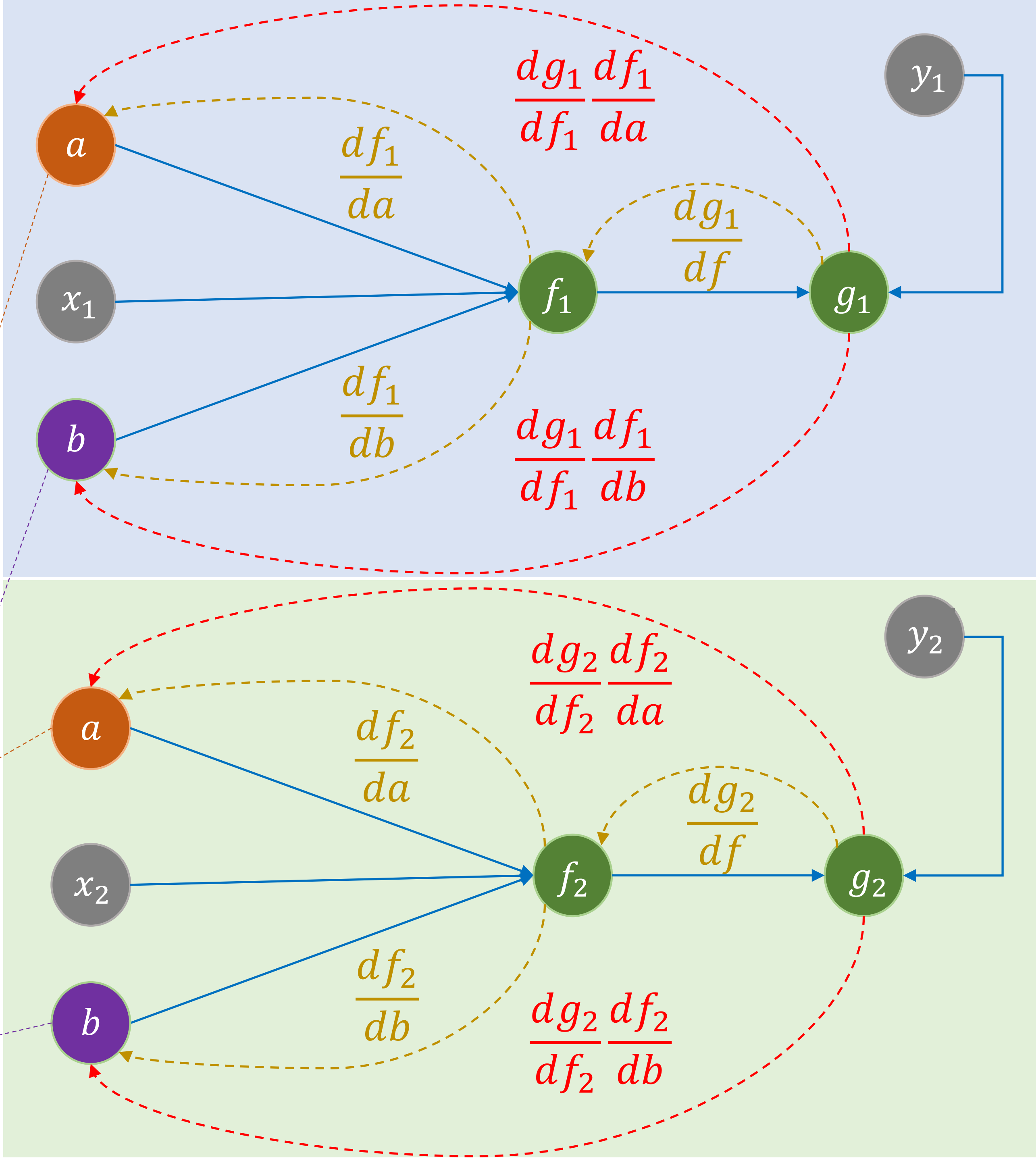
Partial derivative functions

$$\frac{dg}{da} = \frac{dg}{df}\frac{df}{da} = 2x(f-y)$$

$$\frac{dg}{db} = \frac{dg}{df}\frac{df}{db} = 2(f-y)$$

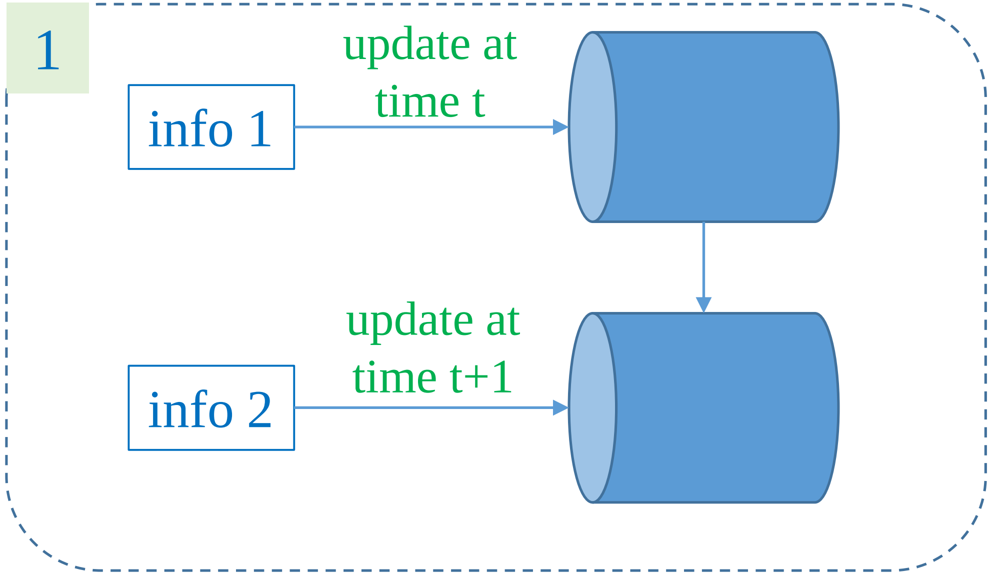
$$\sum_i \frac{dg_i}{da} = \frac{dg_1}{df_1}\frac{df_1}{da} + \frac{dg_2}{df_2}\frac{df_2}{da}$$

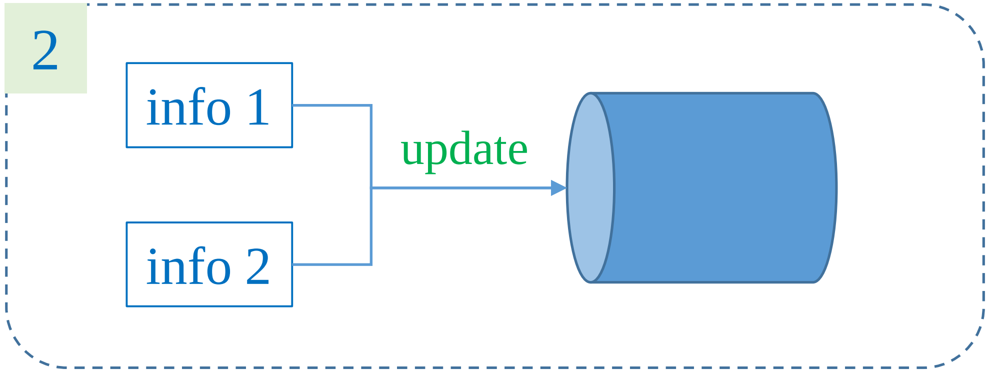
$$\sum_i \frac{dg_i}{db} = \frac{dg_1}{df_1}\frac{df_1}{db} + \frac{dg_2}{df_2}\frac{df_2}{db}$$

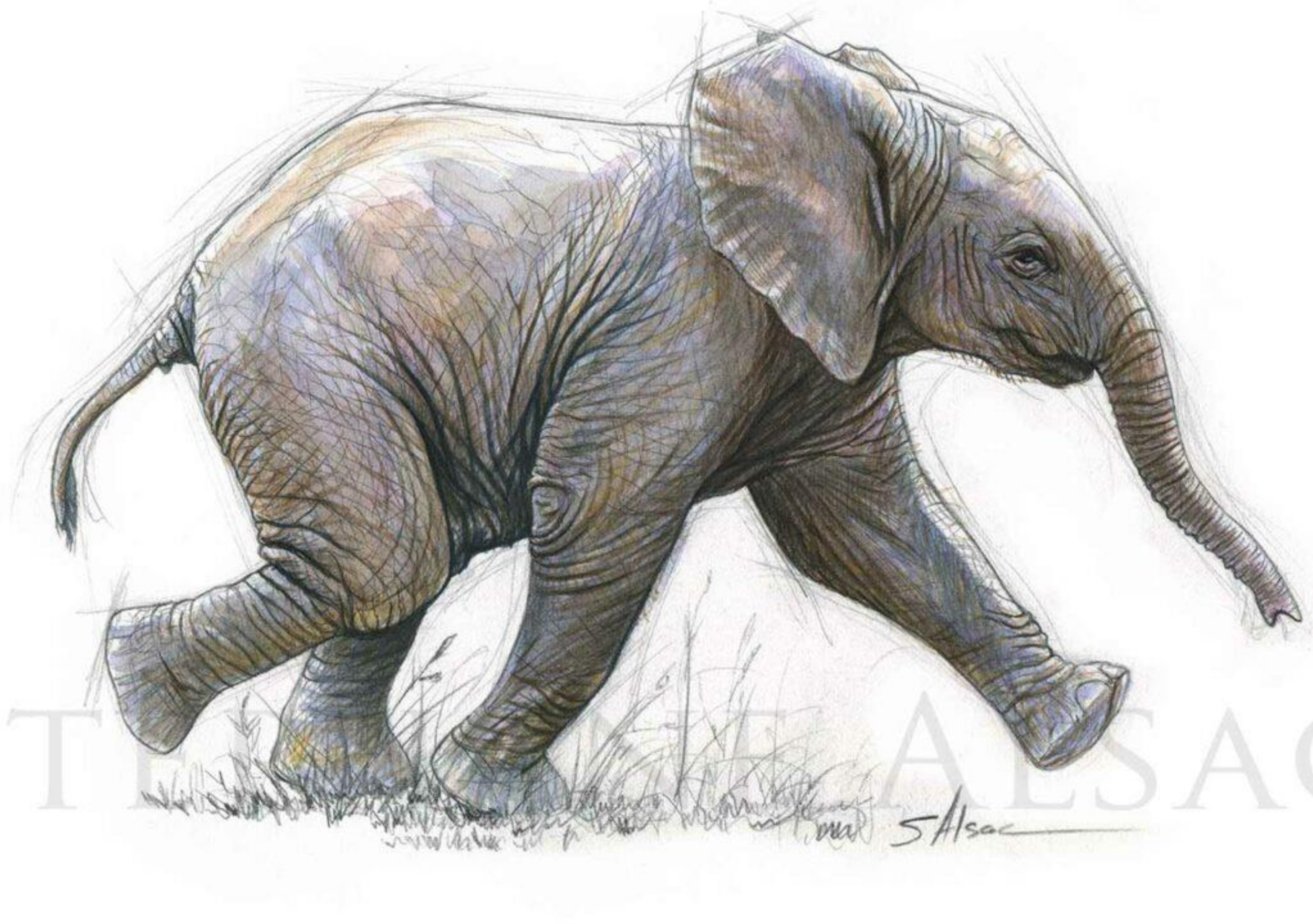


Gradient-based Optimization

❖ How to use gradient information



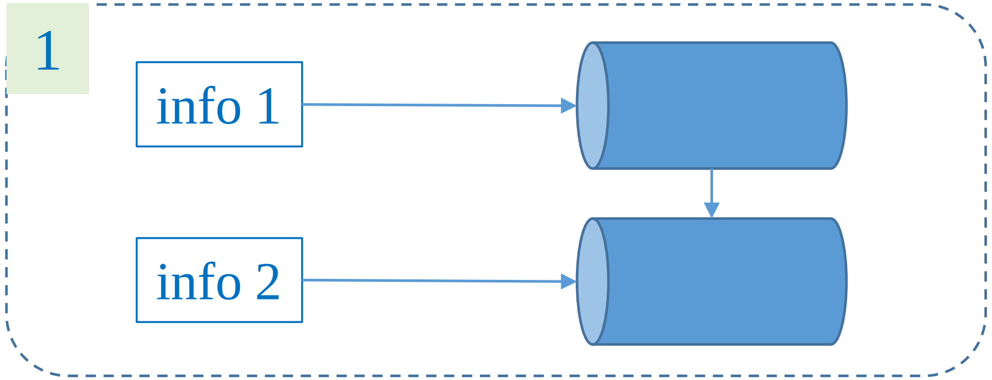




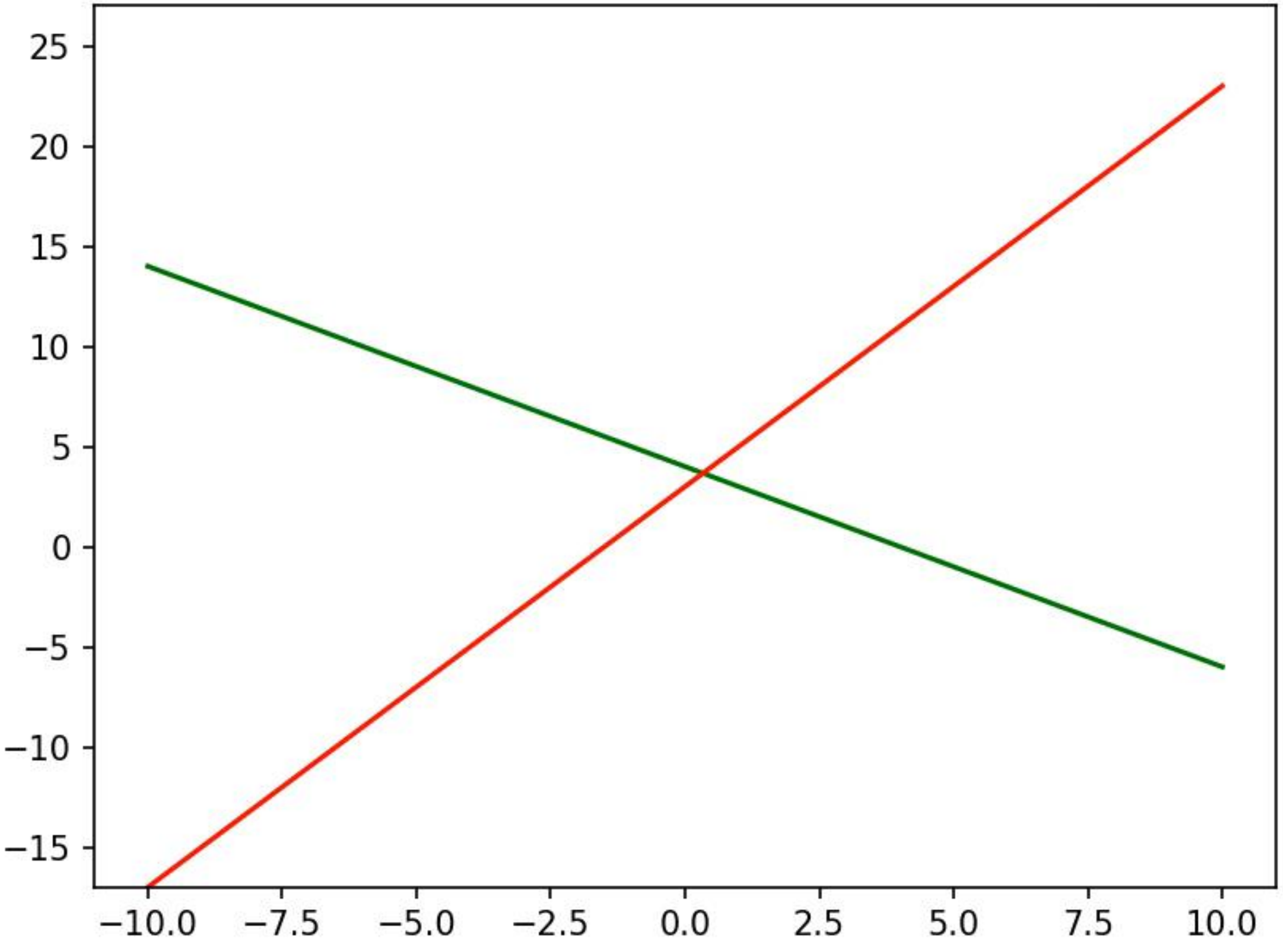
```python
import random

# predict function
def predict_func(x, a, b):
    return a*x + b

# parameters
num_steps = 100
lr = 0.01

# given data
data = [[-2, -1],
        [5, 13]]

# 1. set a, b randomly
a = random.random()*10.0 - 5.0
b = random.random()*10.0 - 5.0
```

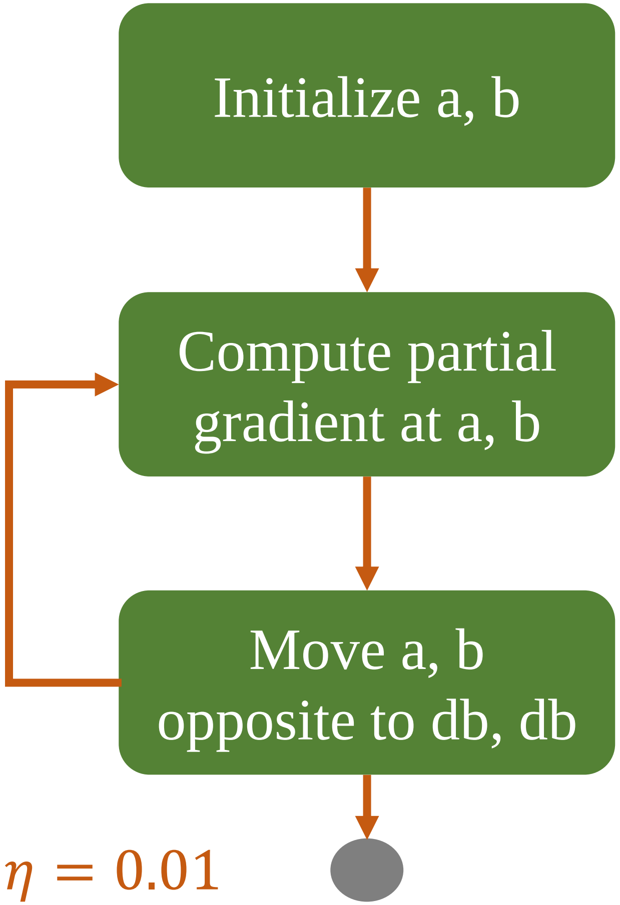
```python
for i in range(num_steps):
    for sample in data:
        x_value, y_value = sample

        # compute predicted_y
        predicted_y = predict_func(x_value, a, b)

        # compute g
        g_value = (predicted_y - y_value)**2

        # compute partial gradients for a and b
        dg_da = 2*x_value*(predicted_y - y_value)
        dg_db = 2*(predicted_y - y_value)

        # update
        a = a - lr*dg_da
        b = b - lr*dg_db
```

Summary



$$\frac{dg}{da} = \frac{dg}{df}\frac{df}{da} = 2x(f - y)$$
$$\frac{dg}{db} = \frac{dg}{df}\frac{df}{db} = 2(f - y)$$

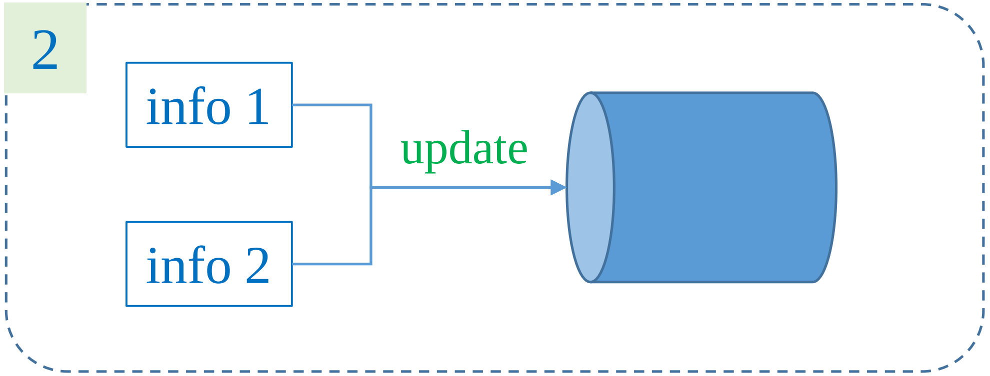




```python
import random

# predict function
def predicted_func(x, a, b):
    return a*x + b

# parameters
num_steps = 100
lr = 0.01

# given data
x1 = -2
y1 = -1

x2 = 5
y2 = 13

# 1. set a, b randomly
a = random.random()*10.0 - 5.0
b = random.random()*10.0 - 5.0
```

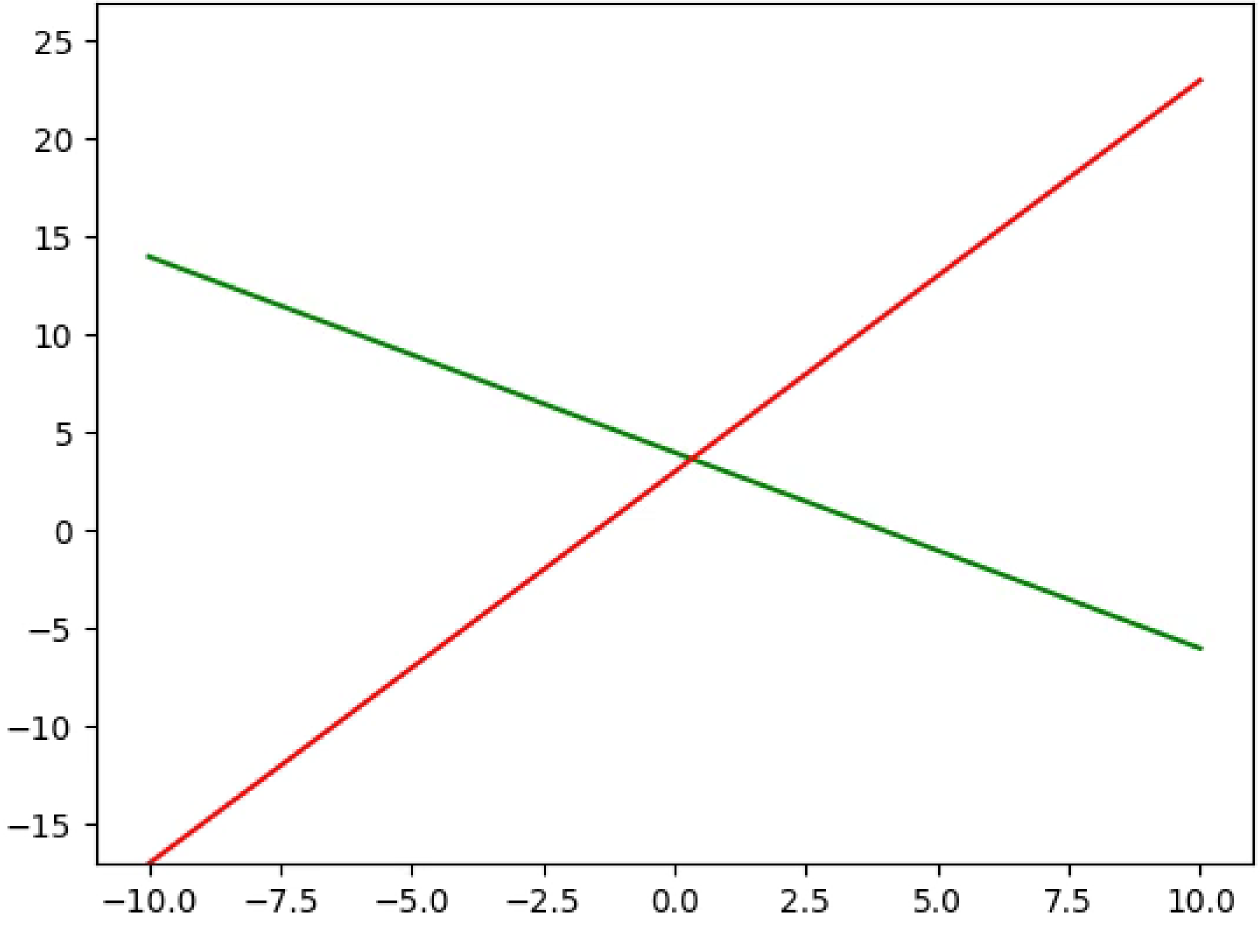
```python
for i in range(num_steps):
    # 2. compute predicted_y1 and predicted_y2
    predicted_y1 = predicted_func(x1, a, b)
    predicted_y2 = predicted_func(x2, a, b)

    # 3. compute g
    g_value_1 = (predicted_y1 - y1)**2
    g_value_2 = (predicted_y2 - y2)**2

    # logging
    # ...

    # 4. compute partial gradients for a and b
    dg_da = 2*x1*(predicted_y1 - y1) + 2*x2*(predicted_y2 - y2)
    dg_db = 2*(predicted_y1 - y1) + 2*(predicted_y2 - y2)

    # 5. update
    a = a - lr*dg_da
    b = b - lr*dg_db
```

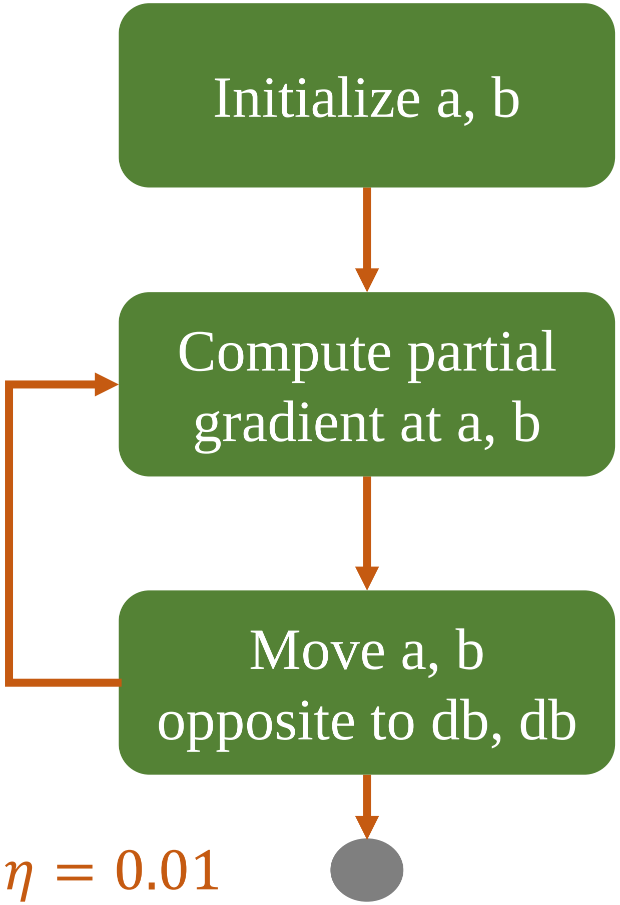
Summary



$$\frac{dg}{da} = \frac{dg}{df}\frac{df}{da} = 2x(f-y)$$

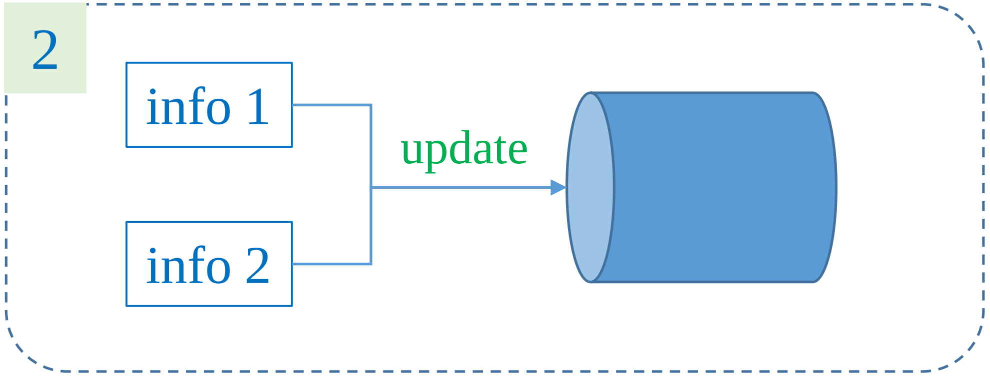
$$\frac{dg}{db} = \frac{dg}{df}\frac{df}{db} = 2(f-y)$$





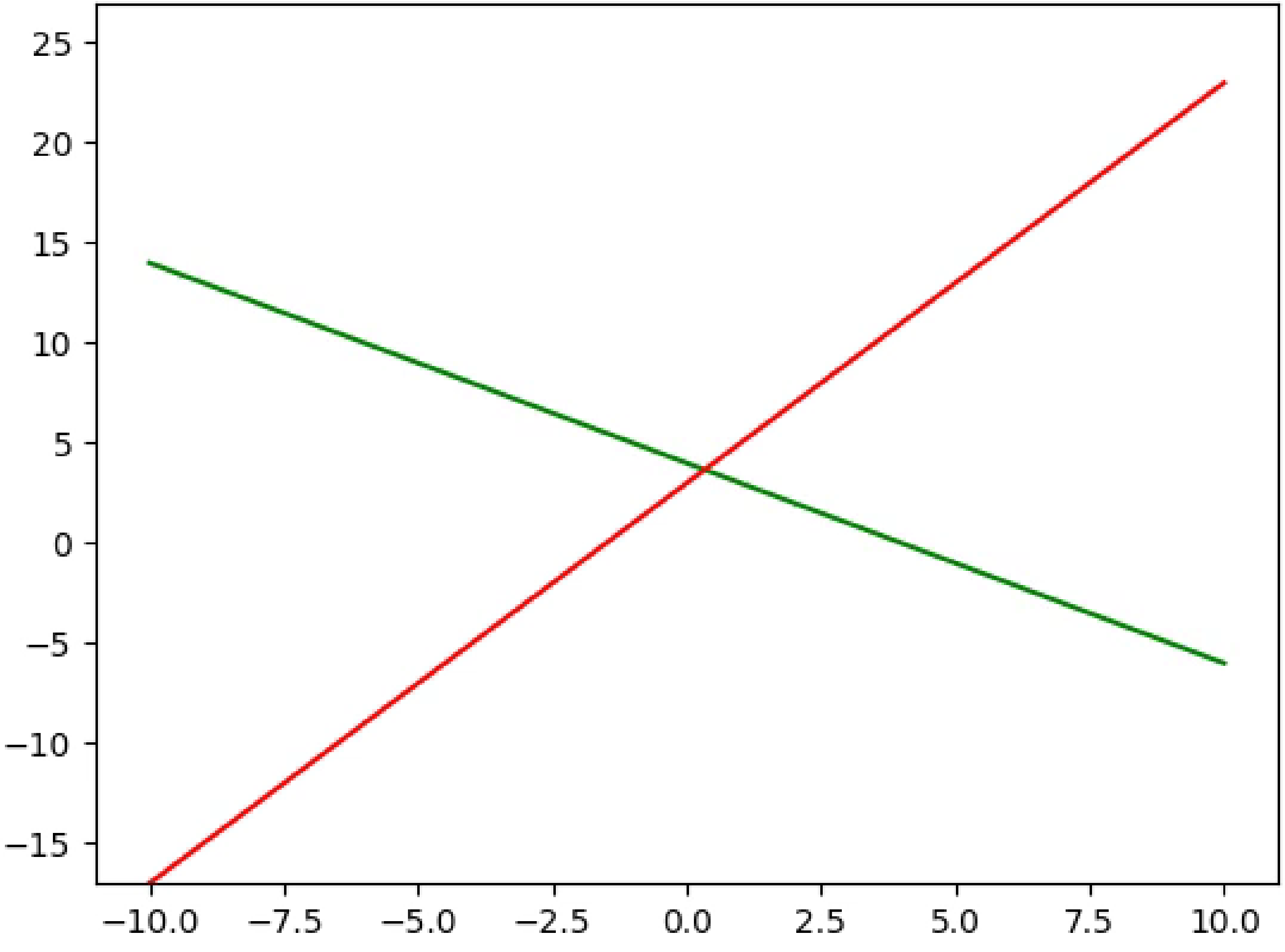
$\eta = 0.01$

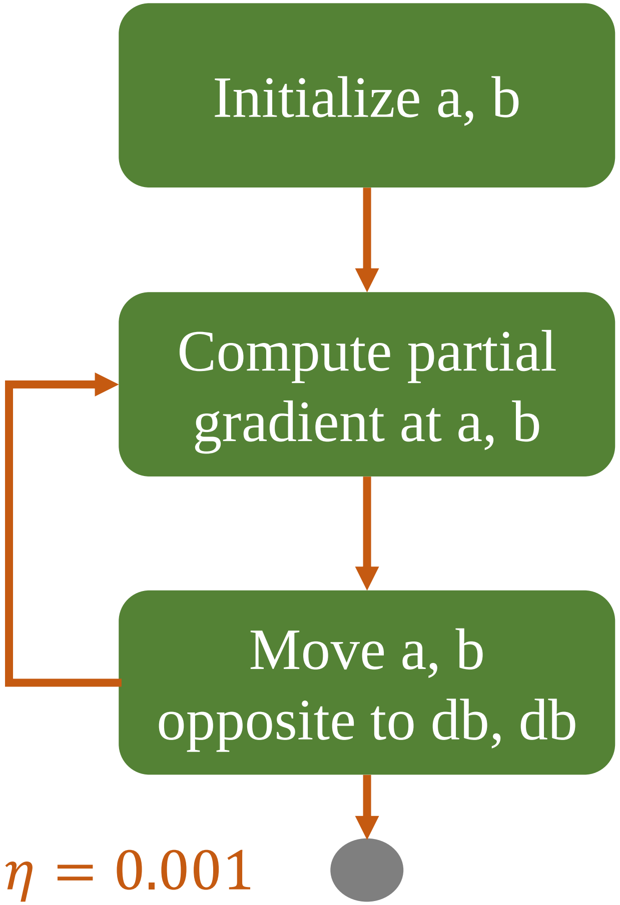
Summary



$$\frac{dg}{da} = \frac{dg}{df}\frac{df}{da} = 2x(f - y)$$

$$\frac{dg}{db} = \frac{dg}{df}\frac{df}{db} = 2(f - y)$$





$\eta = 0.001$

List-based Implementation

❖ Vectorization

$$f(x) = ax + b$$

$$f(x) = ax + b$$
$$= ax + b1$$
$$= \begin{bmatrix} x \\ 1 \end{bmatrix} \cdot \begin{bmatrix} a \\ b \end{bmatrix}$$

```python
# implement 1: naive approach
def predict_w1(x, a, b):
    return a*x + b

# test
print(predict_w1(2, 1, 3))
```

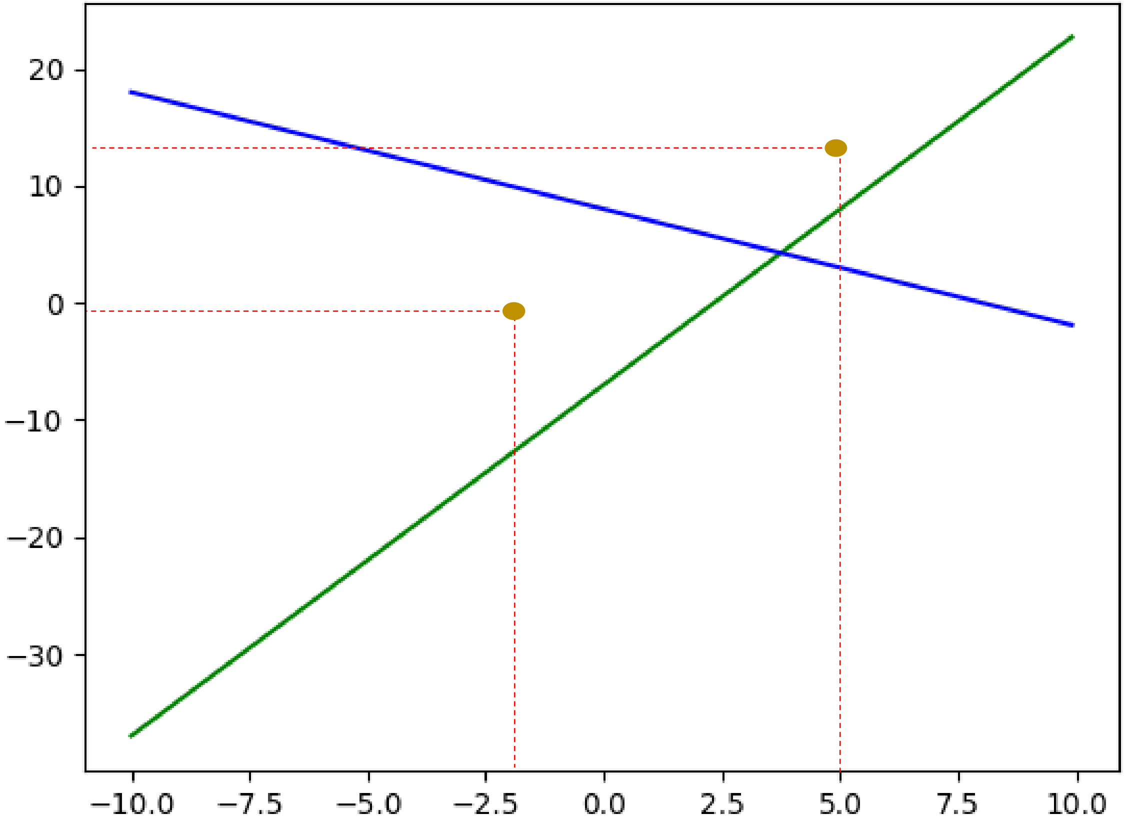
```
5
```

```python
# implement 2: using list
def predict_w2(data, weights):
    return sum([d*w for d, w in zip(data, weights)])

# test
print(predict_w2([2, 1], [1, 3]))
```

```
5
```

Discussion

Remove the red line

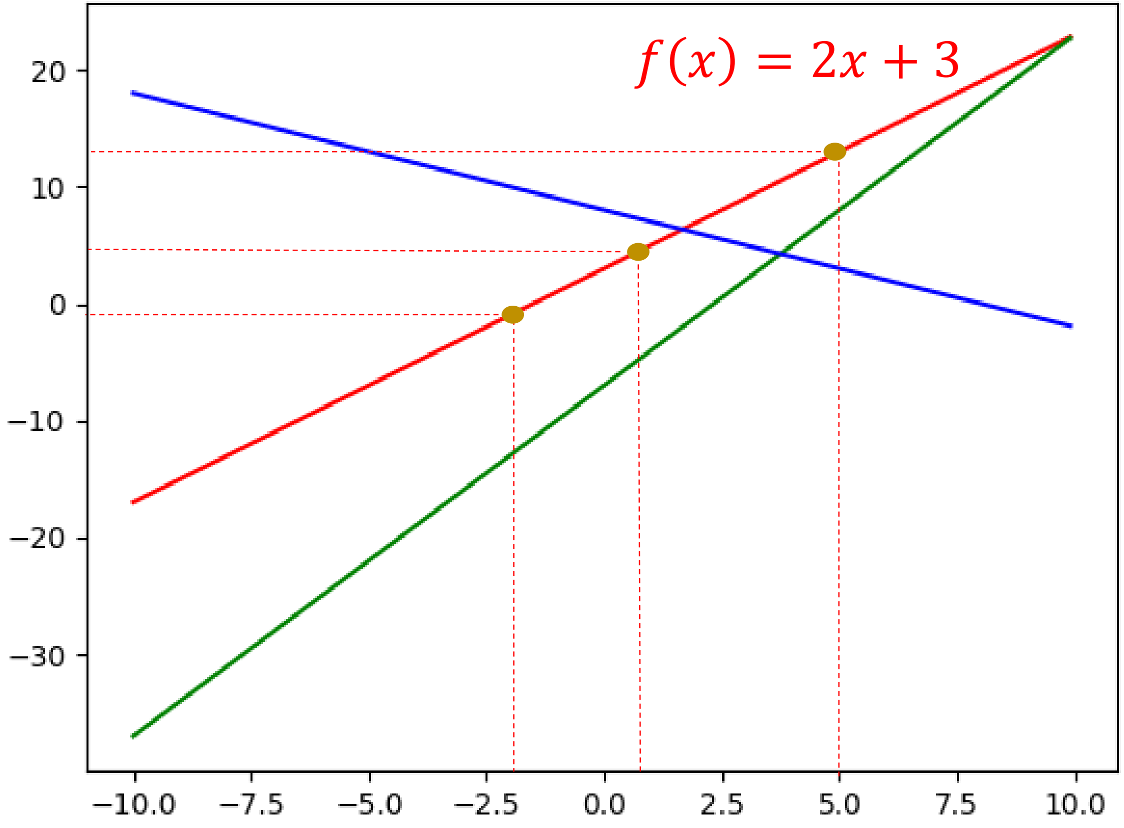


Have one more sample

$(x_1$=-2, y_1=-1)

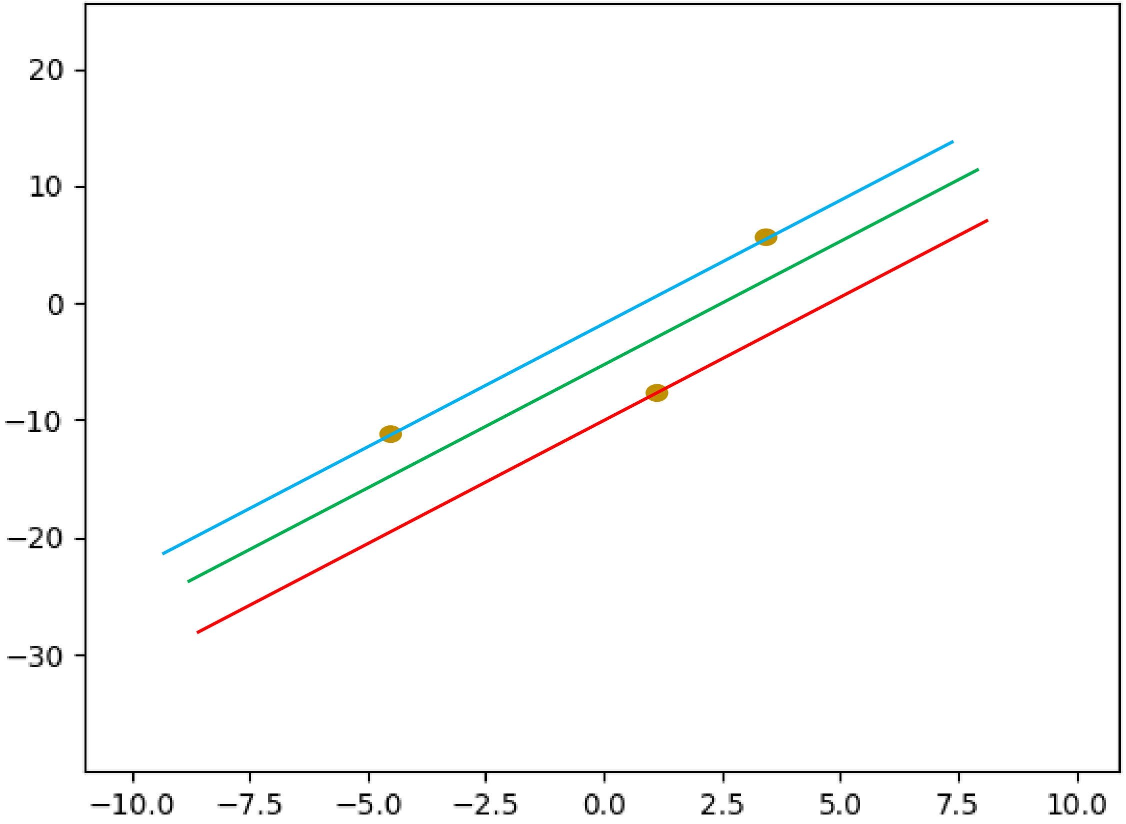
$(x_2$=5, y_2=13)

$(x_2$=1, y_2=3)



Discussion

❖ What about the given samples?



Line 1: go through two points

Line 2: smallest summation of distances

Line 3: go through one point