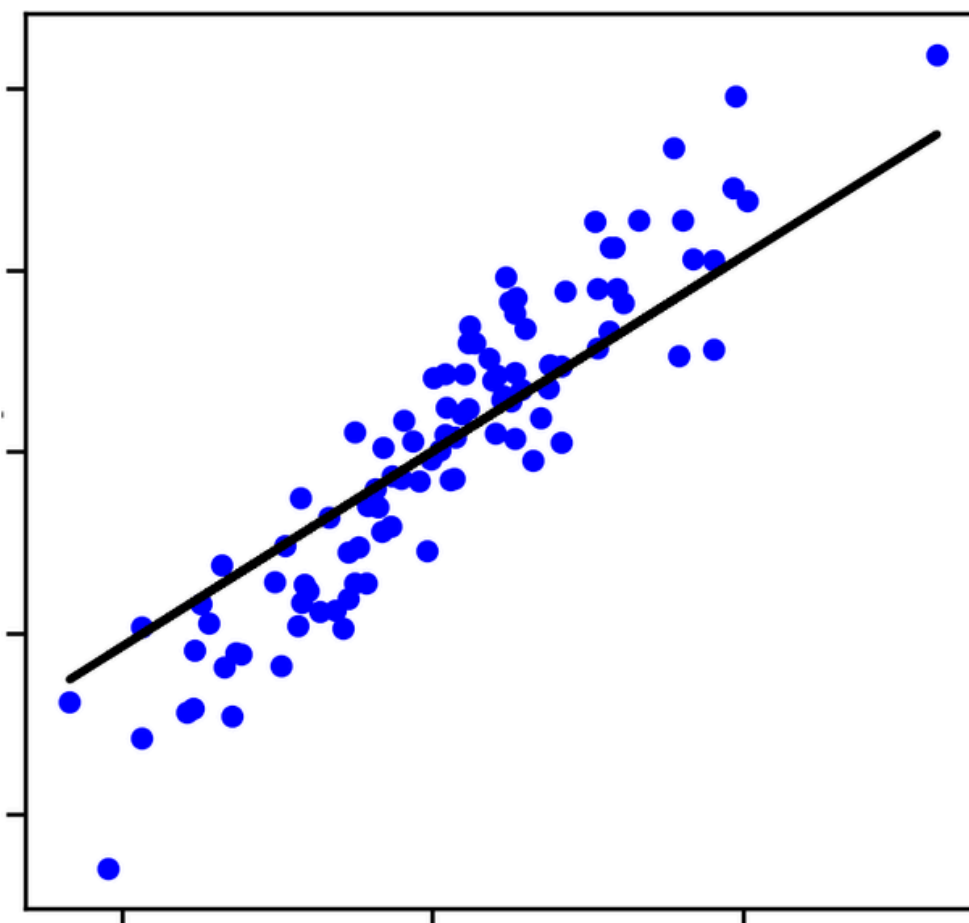


REGRESSION

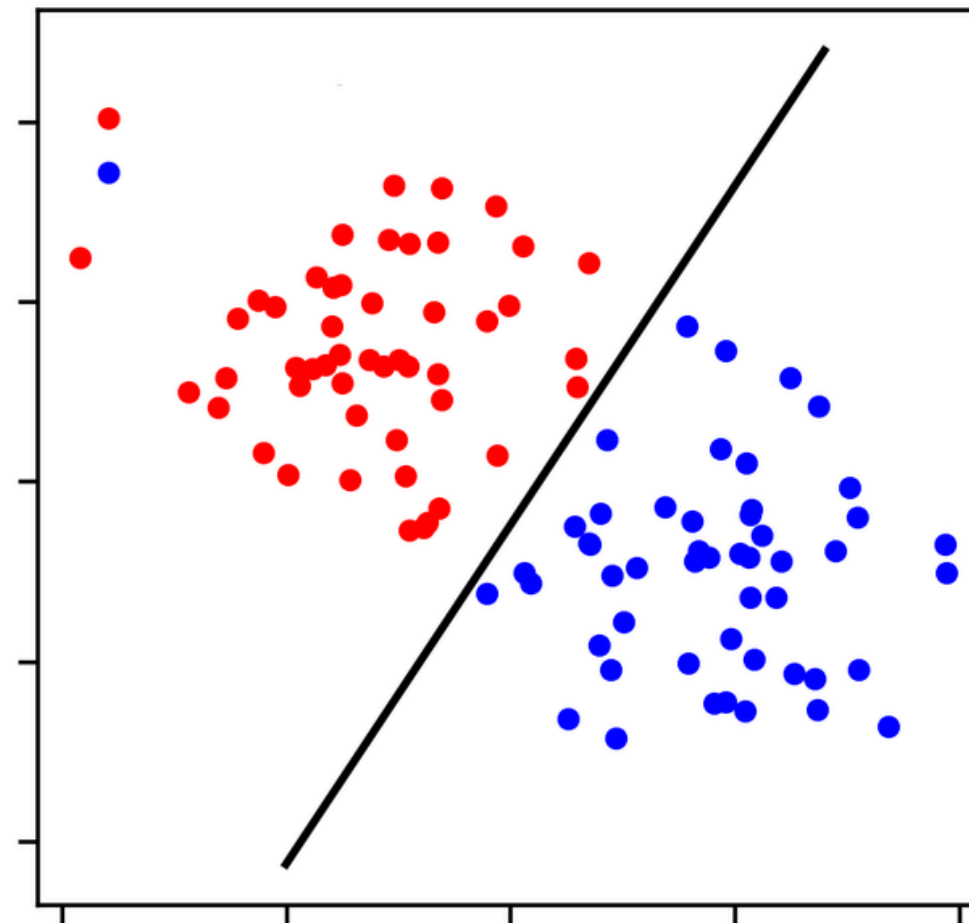
Vu Thanh Tuyen

SUPERVISED LEARNING

Regression




Classification





SUPERVISED LEARNING

Regression		Classification		
Diện tích (m2)	Giá	X1	X2	Y
100	1000\$	2	3	0
300	2000\$	1.1	3.4	1
400	5000\$	6.9	2.3	1
1000	10000\$	1.8	4	0
...



Supervised learning

$$\begin{array}{ccc} X & \rightarrow & Y \\ \text{Input} & & \text{Output} \end{array}$$

n : số mẫu

d : là số chiều của input, $x \in \mathbb{R}^d$

$X^{(i)}$ – i^{th} : mẫu thứ i

$Y^{(i)}$ – i^{th} : đầu ra thứ i

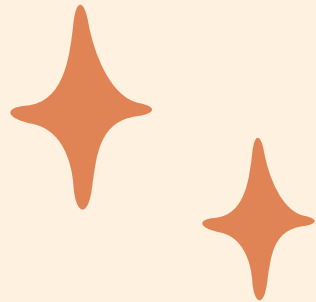
$X^{(i)}, Y^{(i)}$: là cặp input, output thứ i

– Ở đây ta thấy

+ Regression: $Y \in \mathbb{R}$

+ Classification: Y là 1 nhãn nào đó

Sơ Đồ



Traning set
 $\{(X^{(i)}, Y^{(i)})\}_{(i=1)}^n$



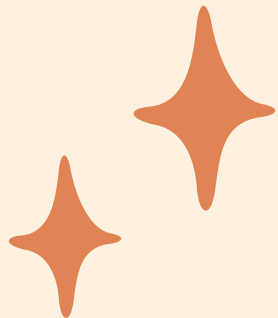
Learning algorithm

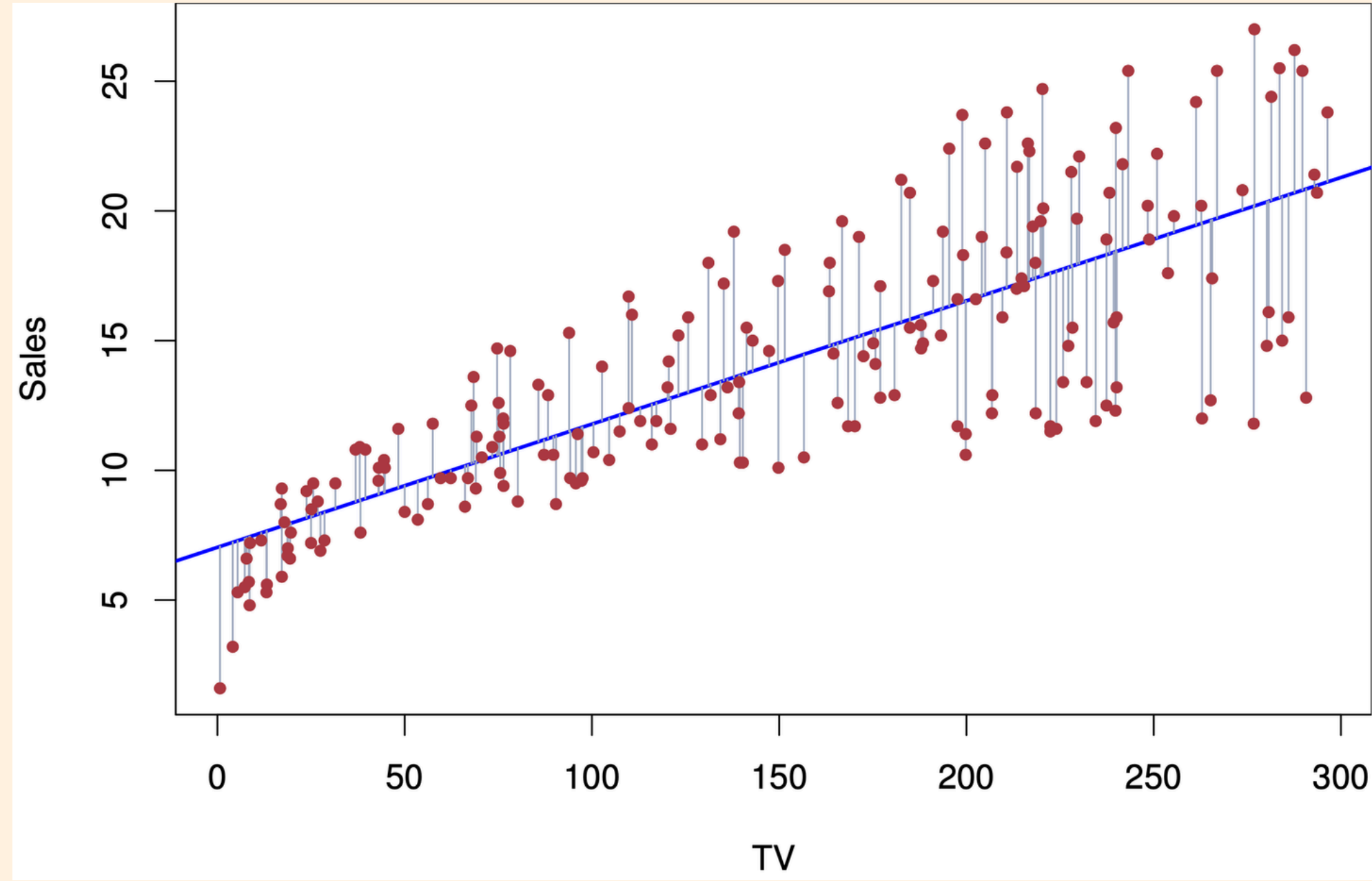


$X:input \rightarrow H(x) \approx y \rightarrow Y:output$



LINEAR REGRESSION





LINEAR REGRESSION

$X \in R^{(d)}$, $Y \in R$, n mẫu

Xét theo đường thẳng: $y = a * x + b$

Xét: $h(x) = \text{Bias} + a_1 * x_1 + a_2 * x_2 + \dots + a_d * x_d$ với $a_i \in R^{(d+1)}$

Với $x_0 = 1$, $\text{Bias} = a_0$

Viết gọn: $h(x) = \sum_{i=0}^d a_i * x_i = a^T * x$ với a, x là các vector cột

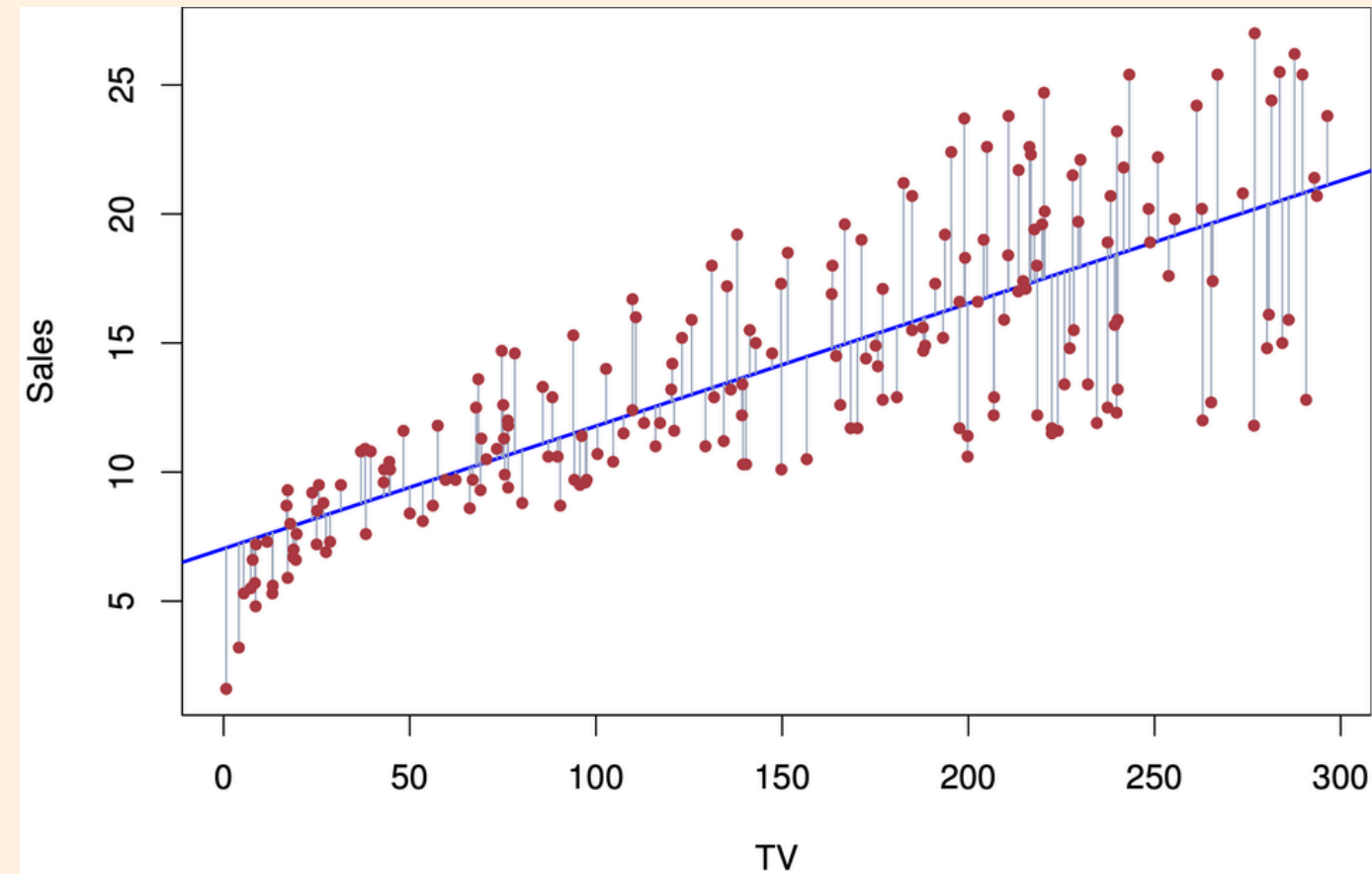
HÀM LOSS FUNCTION

Mục tiêu: là tìm hàm $h(x)$ để có thể tính toán ra được xấp xỉ với giá trị ban đầu:

$$H(x^{(i)}) \approx y^i$$

Ta có hàm
loss/cost function: \wedge^2 ?

$$loss(a) = \frac{1}{2} \sum_{i=1}^n (h(x_i) - y_i)^2$$

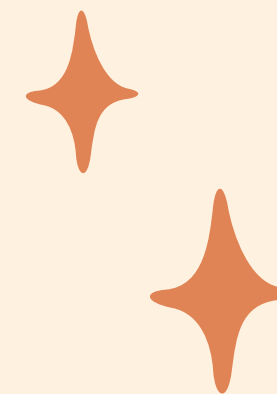
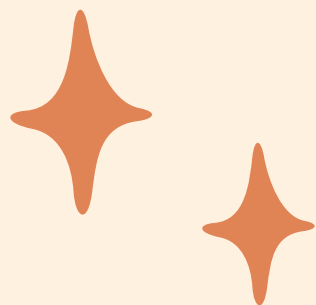
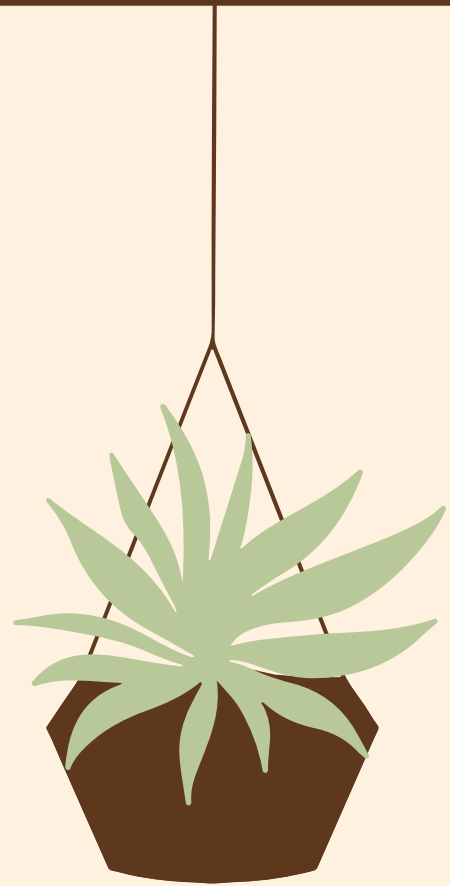


MỤC TIÊU

- Ở đây mục tiêu của chúng ta là tìm ra các số $a_0, a_1, a_2, \dots, a_d$ (hay được gọi là weight) và điều ta cần làm là tìm a sao cho hàm loss function đạt min
- Biểu diễn cụ thể:

$$a = \underset{a: a_1, a_2, \dots, a_d}{\operatorname{argmin}} \frac{1}{2} \sum_{i=1}^n (h(x_i) - y_i)^2 \quad \text{Do } h(x_i) \text{ chứa } a \text{ là hàm tạo từ } \vec{a}$$

Toán Học



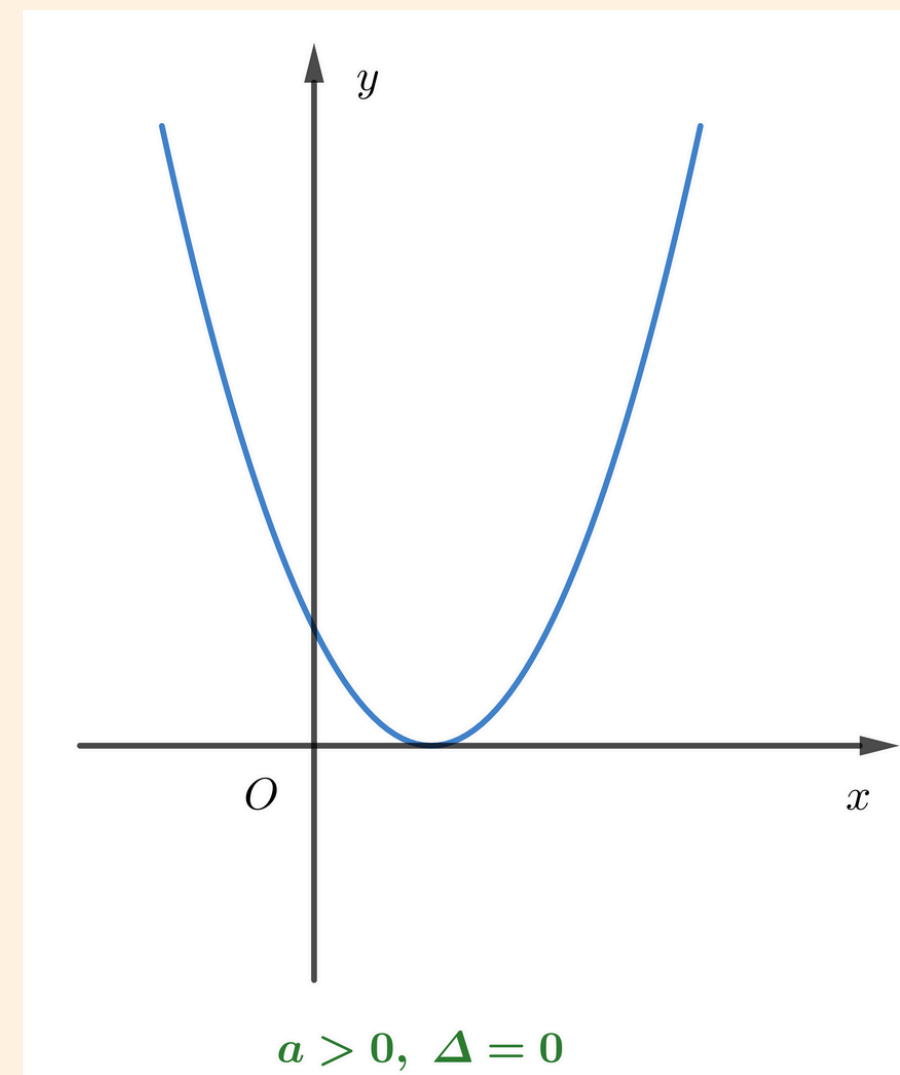
TOÁN HỌC



- Cách làm thông thường tìm min: đạo hàm như cấp 3

1. Tìm đạo hàm
 2. Tìm cực trị
 3. Xét xem giá trị đó là cực tiểu không
 4. Vẽ bảng BT
- Ta xét loss:

$$loss(a) = \frac{1}{2} \sum_{i=1}^n (h(x_i) - y_i)^2$$



TOÁN HỌC



$$\text{Xét: } a = \begin{bmatrix} a_0 \\ a_1 \\ \dots \\ a_{(d+1)} \end{bmatrix}, \quad x = \begin{bmatrix} x_1 \\ x_2 \\ \dots \\ x_n \end{bmatrix}, \quad y = \begin{bmatrix} y_1 \\ y_2 \\ \dots \\ y_n \end{bmatrix}$$

Khi đó:

+ $a \in R^{(d+1)}$, a_0 là bias

+ $x_i (i \in 1 \rightarrow n)$ là các vector feature $\in R^{(d+1)}$, với x_0 có chứa 1 $\rightarrow x \in R^{(n \times (d+1))}$

+ $y \in R^n$

TOÁN HỌC



Xét:

$$Xa - Y$$

$$\rightarrow R^{(n \times (d+1))} R^{(d+1)} - R^n$$

$$\rightarrow R^n - R^n$$

$$\rightarrow R^n$$

Ta được:

$$Xa = \begin{bmatrix} x_1^T a \\ x_2^T a \\ \dots \\ x_n^T a \end{bmatrix}, \quad Y = \begin{bmatrix} y_1 \\ y_2 \\ \dots \\ y_n \end{bmatrix}, \quad Xa - Y = \begin{bmatrix} x_1^T a - y_1 \\ x_2^T a - y_2 \\ \dots \\ x_n^T a - y_n \end{bmatrix}$$

TOÁN HỌC



Ta được hàm: $loss(a) = \frac{1}{2} (Xa - Y)^T (Xa - Y) = \frac{1}{2} \sum_{i=1}^n (h(x_i) - y_i)^2$

Tính đạo hàm:

$$loss(a)^{(1)} = X^T (Xa - Y)$$

Xét đạo hàm bằng 0:

$$loss(a)^{(1)} = 0 \rightarrow X^T (Xa - Y) = 0 \rightarrow X^T Xa - X^T Y = 0$$

$$\rightarrow \text{Nghịệm: } a = (X^T X)^{-1} X^T Y$$

TOÁN HỌC

Từ đó ta áp dụng toán học từ numpy hay torch để tìm ra a (weight cho bài toán linear)

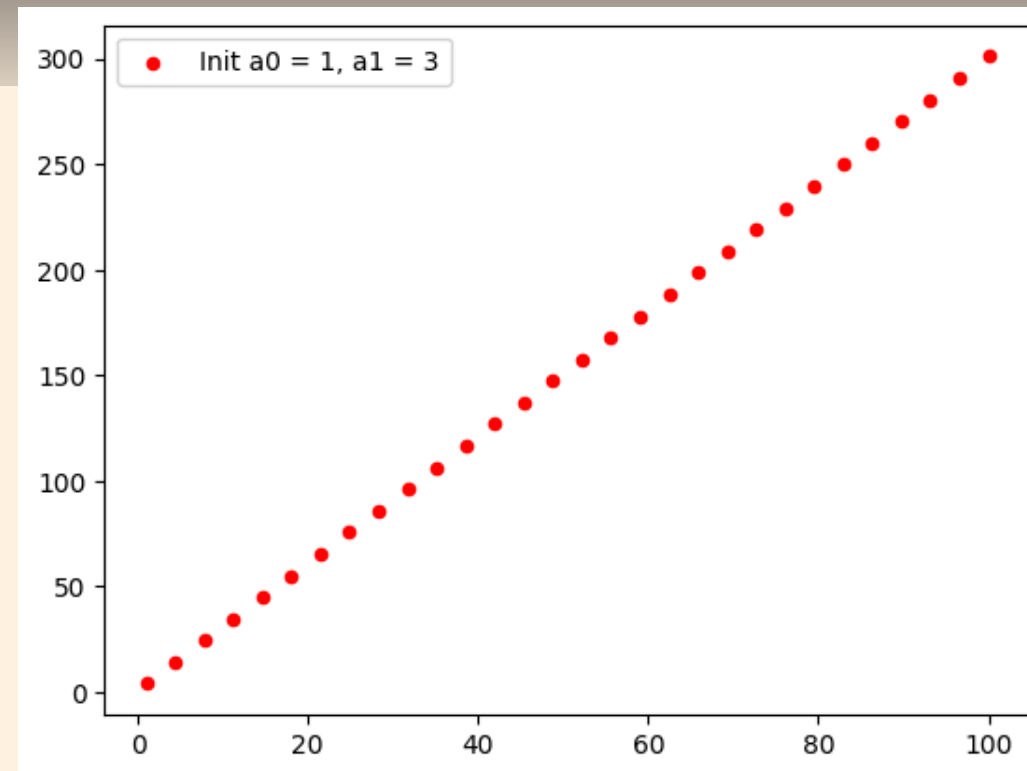


```
1 import numpy as np
2 # khởi tạo X, Y có thể khởi tạo ngẫu nhiên
3 # mục tiêu là áp dụng CT tìm ra a0, a1
4 X = np.linspace(1, 100, 30).reshape(30, 1)
5 a0 = 1
6 a1 = 3
7 Y = a0 + a1 * X
```


TOÁN HỌC



```
1 import matplotlib.pyplot as plt
2
3 plt.scatter(X, Y, c="r", s=20, label="Init a0 = 1, a1 = 3")
4 plt.legend()
5 plt.show()
```



Ta plot lên xem

TOÁN HỌC

Tính weight bằng Công thức



```
1 def weightLinear(X, Y):
2     one = np.ones(shape=(30, 1))
3     x = np.concatenate((one, X), axis=1)
4     y = Y
5     return np.linalg.pinv(x.T @ x) @ (x.T @ y)
6 a = weightLinear(X, Y).squeeze()
7 a
8 # output
9 # array([1., 3.]
```

TOÁN HỌC

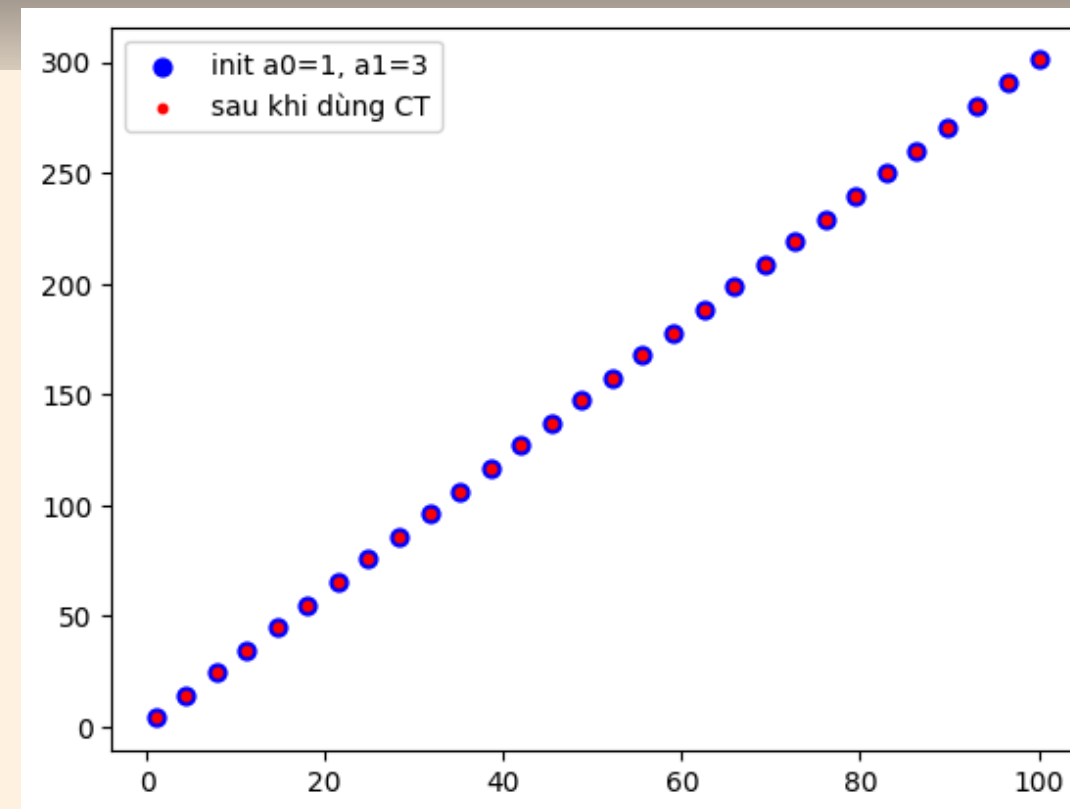


Plot:

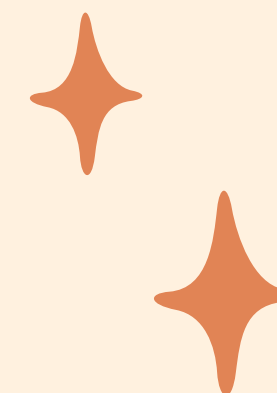
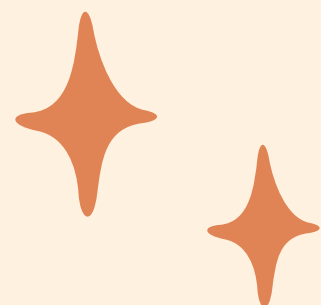
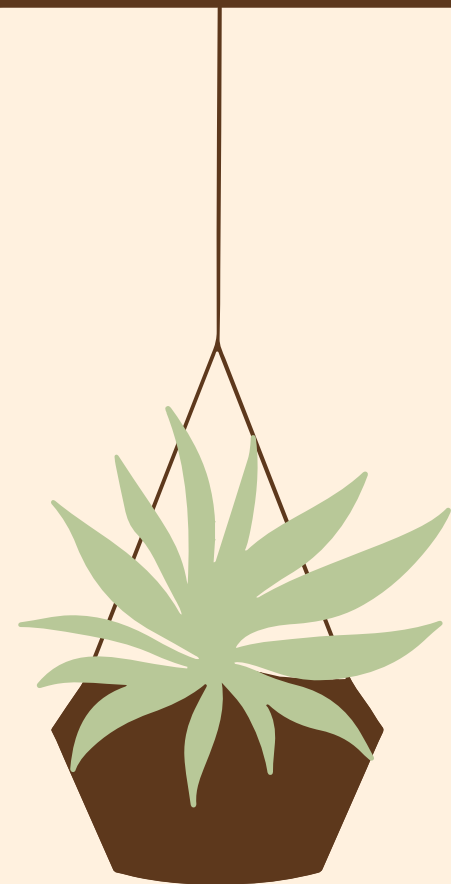
- Init: blue
- Dùng CT: red



```
1 plt.scatter(X, Y, c="b", s=40, label="init a0=1, a1=3")
2 plt.scatter(X, a[0] + a[1] * X, c="r", s=10, label="sau khi dùng CT")
3 plt.legend()
4 plt.show()
```



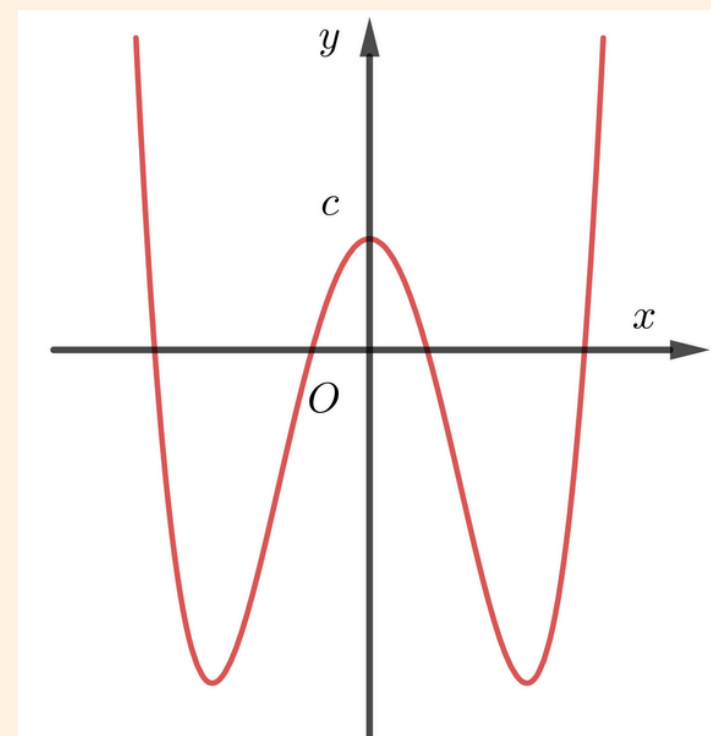
Gradient Descent



GRADIENT DESCENT

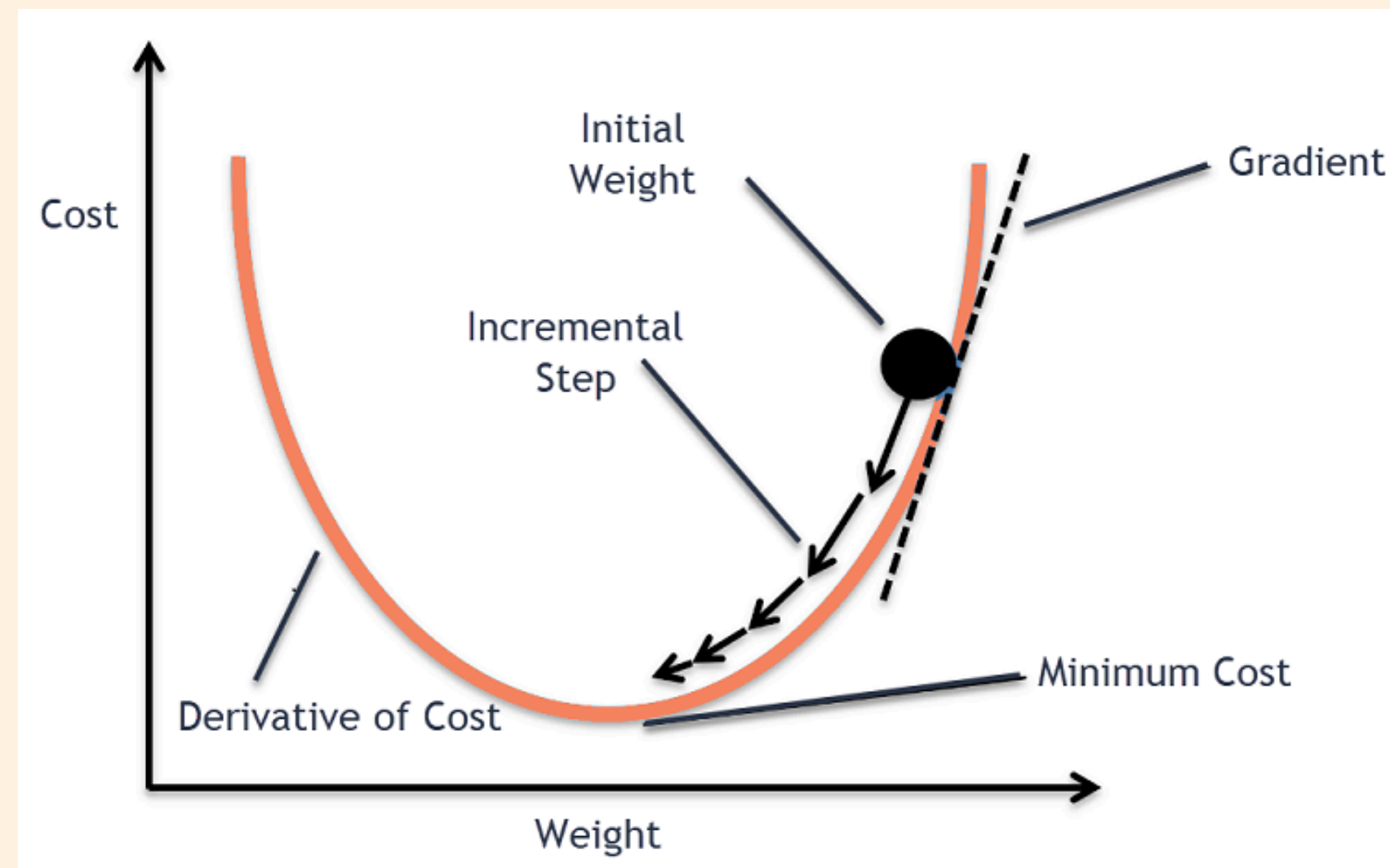


- Phương pháp trên chỉ đối với các bài có thể đạo hàm được thôi , có thể có những bài mà chúng ta không thể đạo hàm được
- Ta sẽ dùng cách chung hơn là phương pháp Gradient Descent



GRADIENT DESCENT

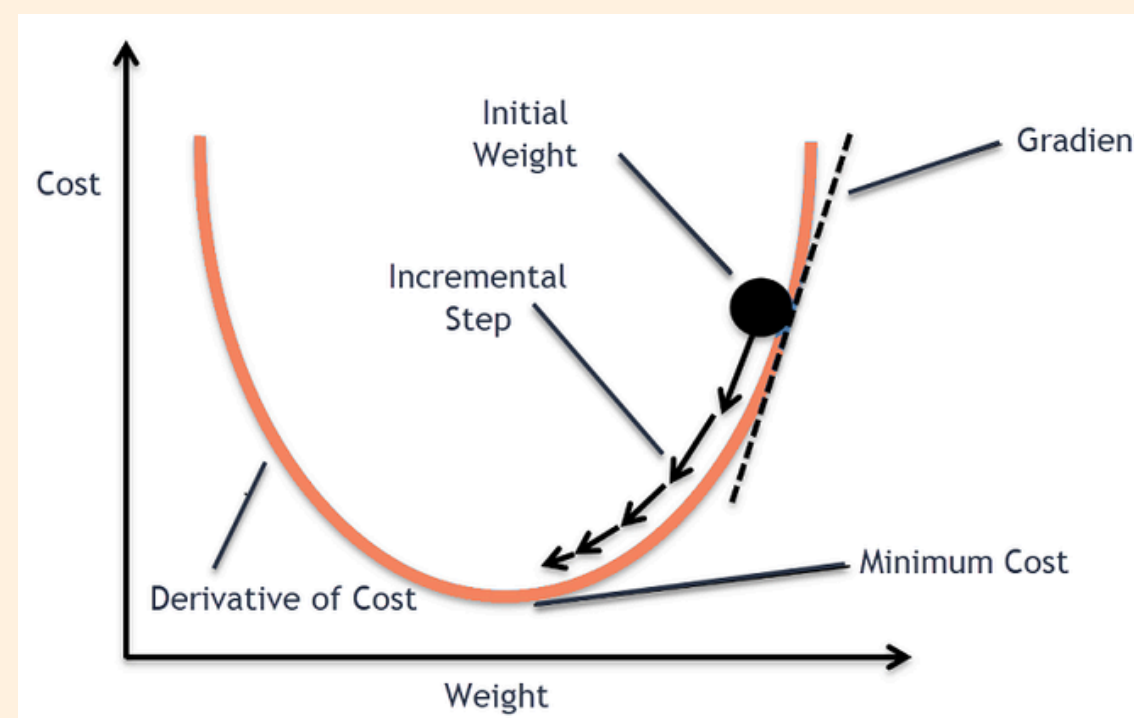
- Ví dụ trong đồ thị



GRADIENT DESCENT



- Ở đây ta cần chú ý đến tính chất cực tiểu trong đạo hàm
- Cực tiểu
 - + Bên trái nghịch biến
 - + Bên phải đồng biến
- Để tìm cực tiểu đó thì ta cần đạo hàm bằng 0



GRADIENT DESCENT

- Ta có hàm loss:

Ta được hàm: $loss(a) = \frac{1}{2} (Xa - Y)^T (Xa - \bar{Y}) = \frac{1}{2} \sum_{i=1}^n (h(x_i) - y_i)^2$

khởi tạo: \vec{a}

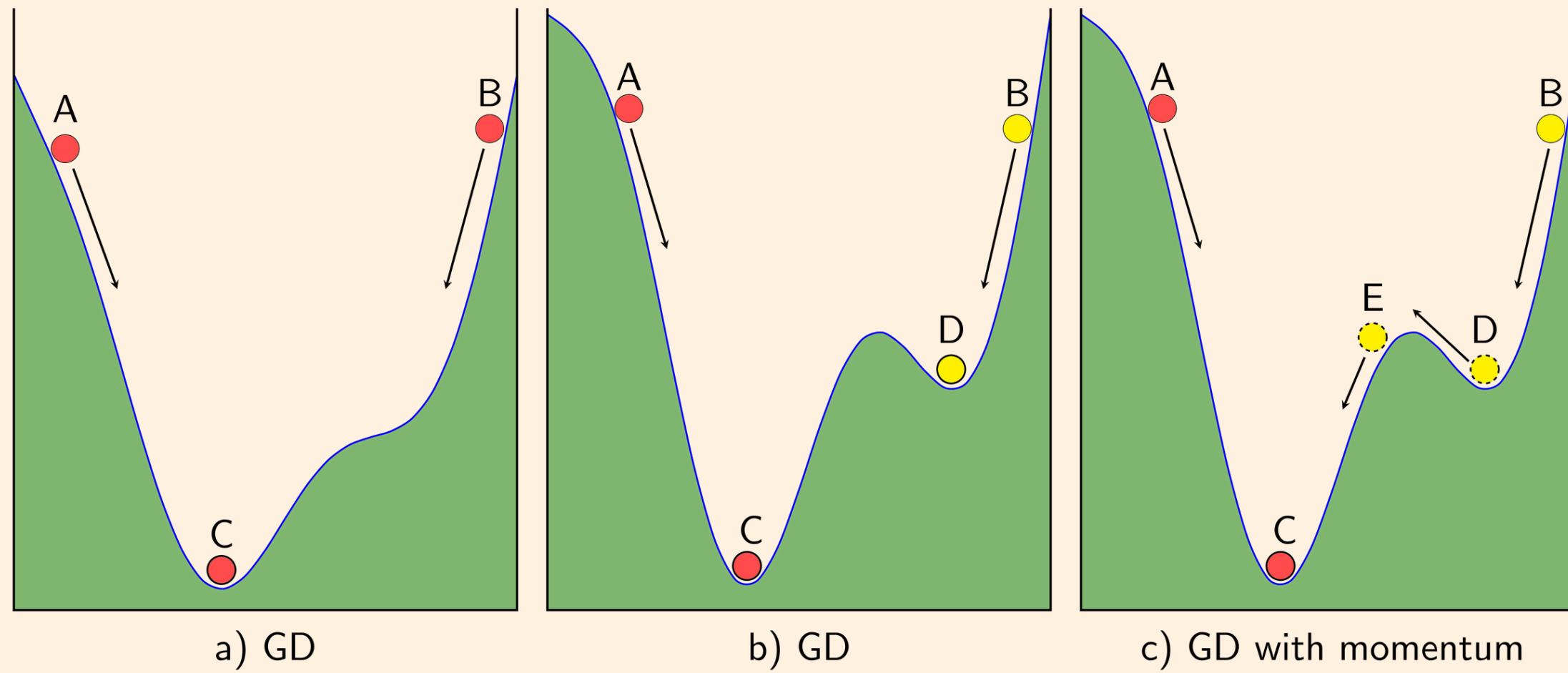
for $i \in range(0, d)$:

$$a_{inew} = a_i - lr * loss^{(1)}(\vec{a})$$

Ta sẽ viết gọn: $\vec{a}_{new} = \vec{a} - lr * loss^{(1)}(\vec{a})$

Trong đó: lr — learning rate

GRADIENT DESCENT



GRADIENT DESCENT



- Sau khi có thể cập nhật được tham số đó thì
- Ta sẽ loop cho đến khi nó hội tụ thì dừng
- Sau khi dừng thì giá trị của nó weight
- Ở đây có vấn đề:
 - + Giá trị khởi tạo như nào?
 - + Learning rate nên để bao nhiêu?

GRADIENT DESCENT

- Khởi tạo:

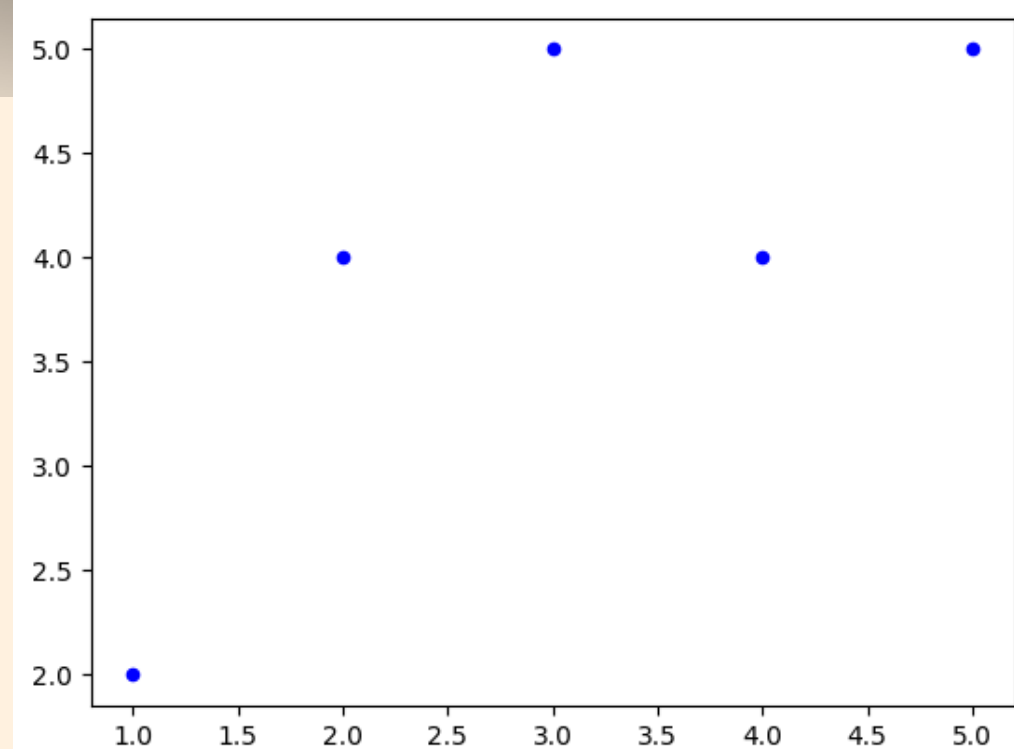
```
1 import numpy as np
2 X = np.array([1, 2, 3, 4, 5]).reshape(5, 1)
3 Y = np.array([2, 4, 5, 4, 5]).reshape(5, 1)
```

GRADIENT DESCENT




- Plot lên xem

```
1 import matplotlib.pyplot as plt
2 plt.scatter(X, Y, c="b", s=20)
3 plt.show()
```



GRADIENT DESCENT


- Function lấy grad tại a



```
1 def grad( $a$ ,  $X$ ,  $Y$ ):  
2     ones = np.ones((len( $X$ ), 1))  
3     x = np.concatenate((ones,  $X$ ), axis=1)  
4     return x.T @ (x @  $a$  -  $Y$ )  
5
```

GRADIENT DESCENT

- Function lấy grad tại a



```
1 def grad( $a$ ,  $X$ ,  $Y$ ):  
2     ones = np.ones((len( $X$ ), 1))  
3     x = np.concatenate((ones,  $X$ ), axis=1)  
4     return x.T @ (x @  $a$  -  $Y$ )  
5
```

GRADIENT DESCENT

- `a_init[-1]` là weight cuối cùng

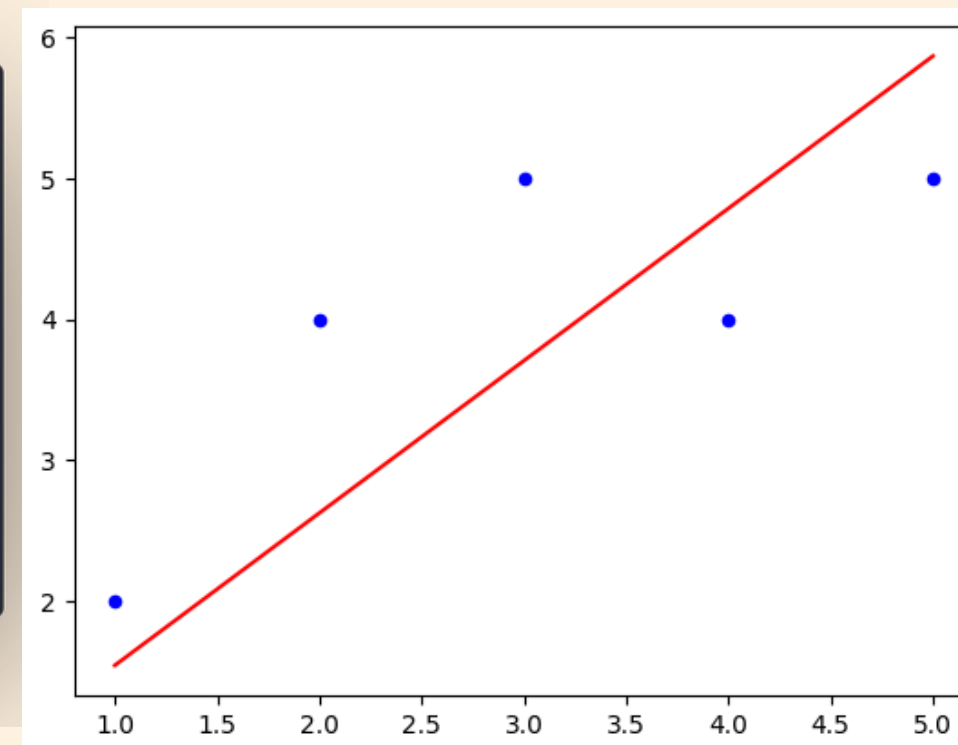


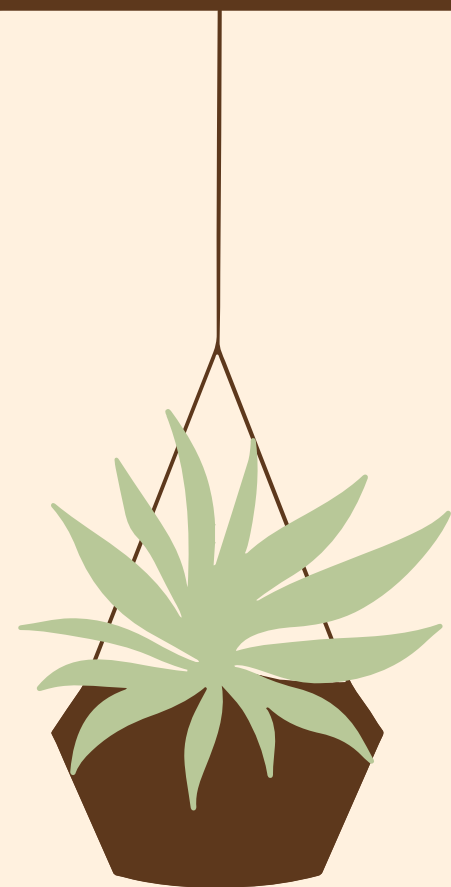
```
6 loop = 10
7 lr = 0.01
8 a_init = [np.array([0, 0]).reshape((2, 1))]
9 for iter in range(loop):
10     print(a_init[-1])
11     gr = grad(a_init[-1], X, Y)
12     if abs(gr[0][0]) < 1e-5:
13         break
14     a_new = a_init[-1] - lr * gr
15     a_init.append(a_new)
```

GRADIENT DESCENT

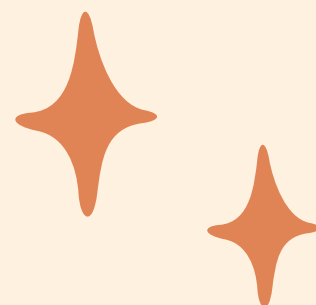
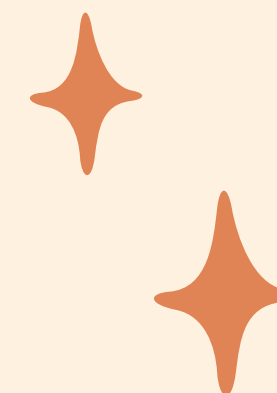
- Plot với weight: `a_init[-1]`

```
1 g = a_init[-1].squeeze()
2
3 import matplotlib.pyplot as plt
4 plt.scatter(X, Y, c="b", s=20)
5 plt.plot(X, g[0] + g[1] * X, c="r")
6 plt.show()
```

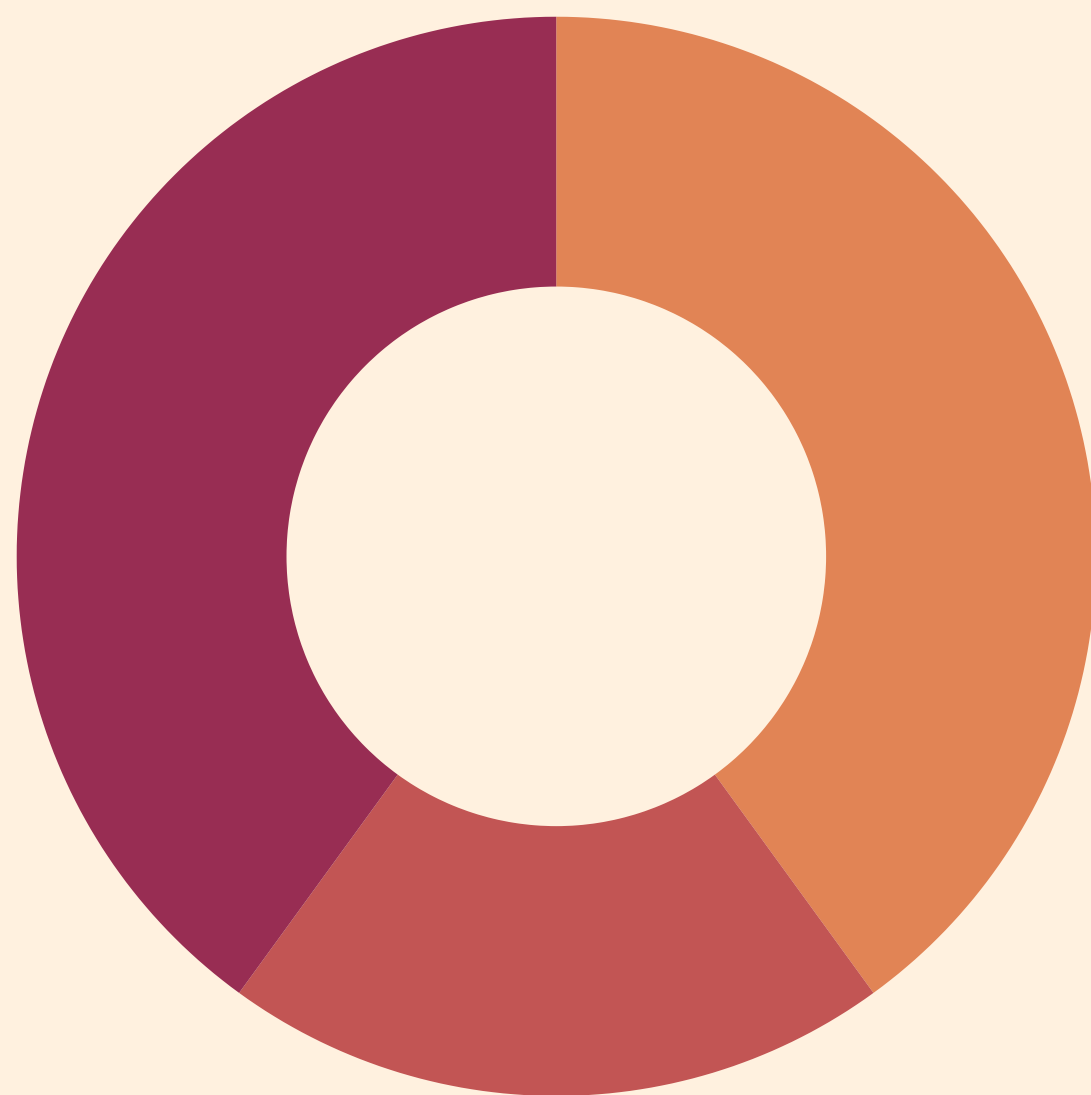




Đánh giá



LINEAR REGRESSION



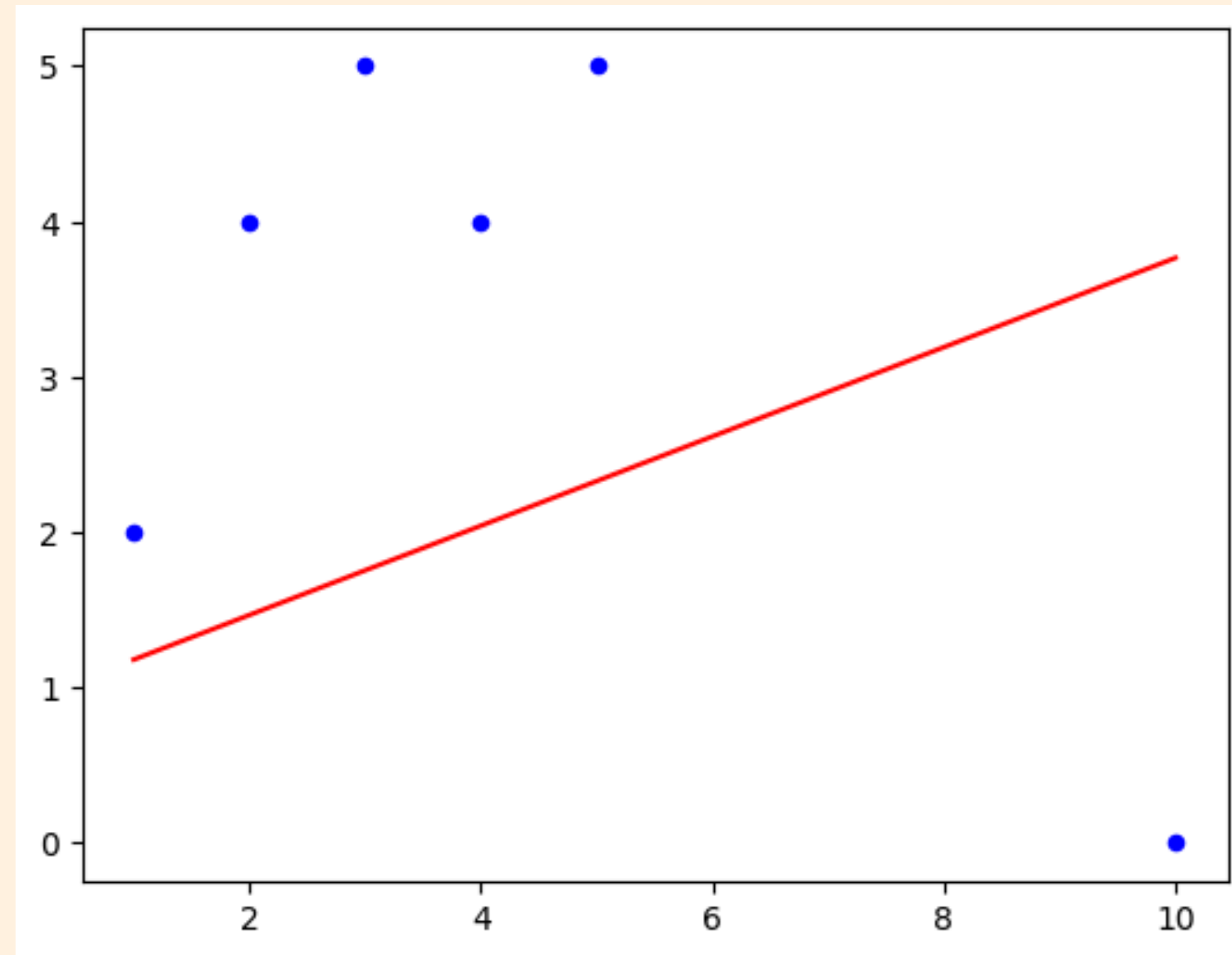
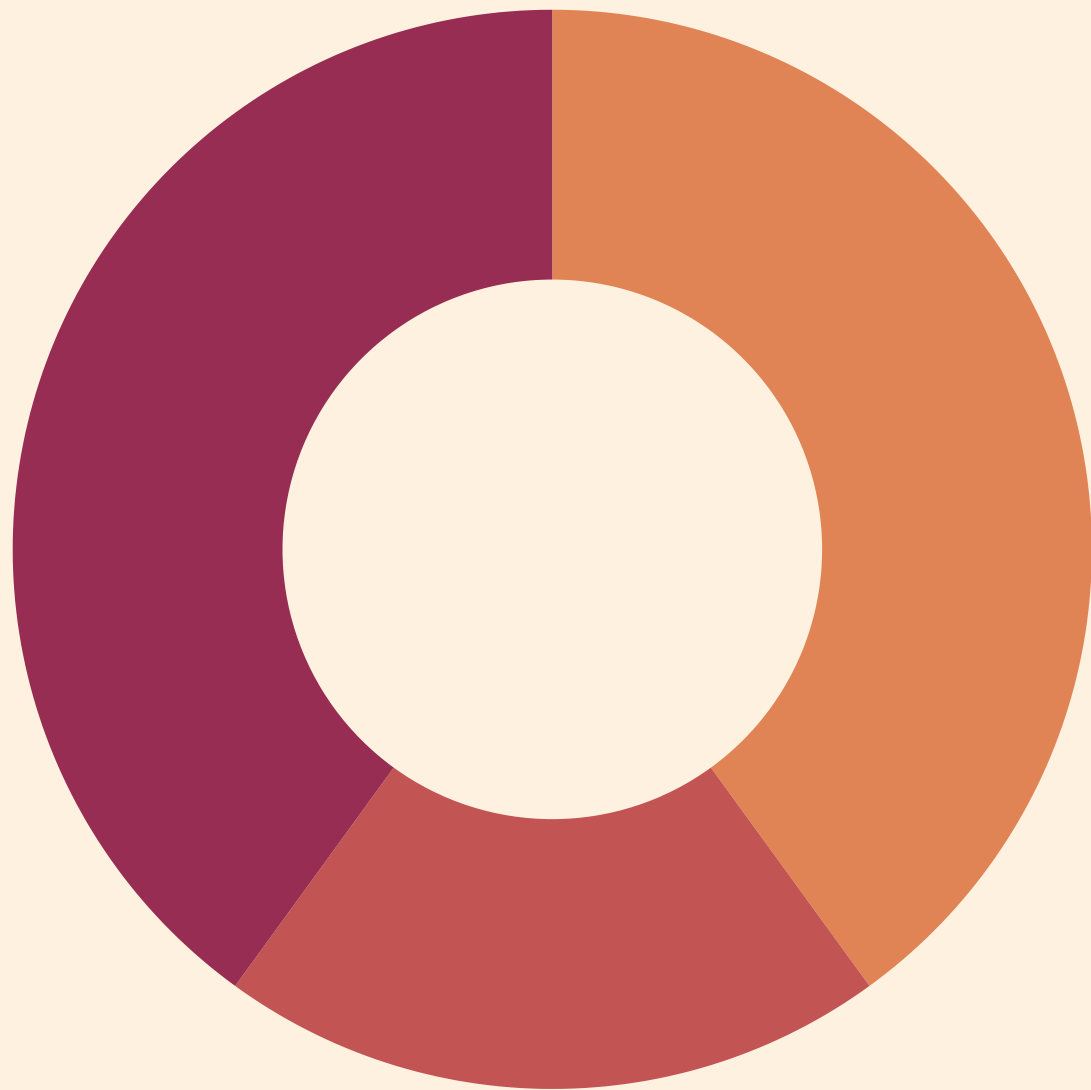
ƯU

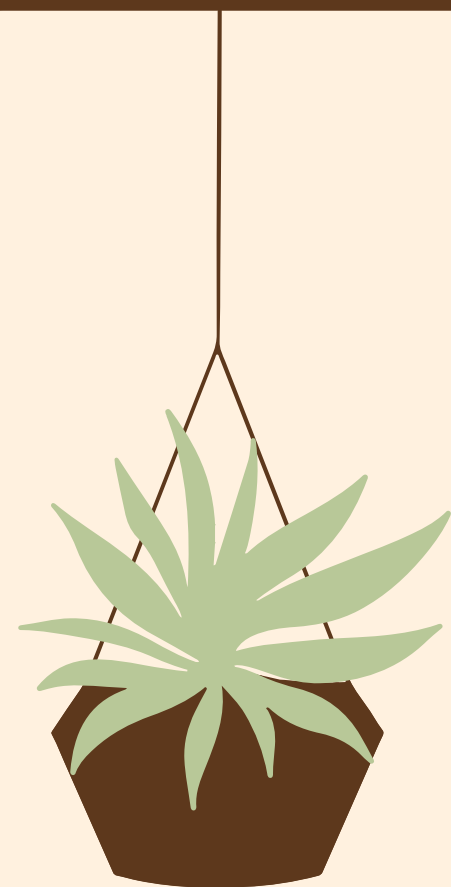
- Dễ sử dụng do có thể đạo hàm và tìm nghiệm được

NHƯỢC

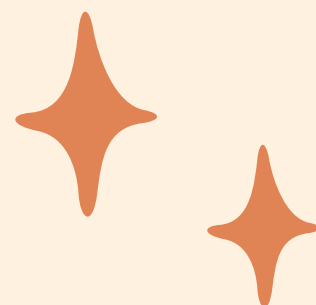
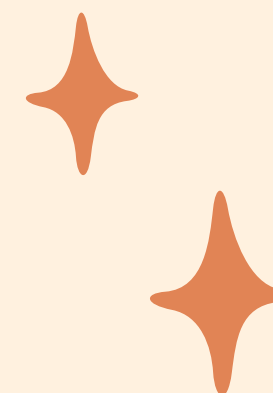
- Hạn chế của linear regression là rất nhạy cảm với nhiễu
- Khắc phục: tiền xử lí dữ liệu

LINEAR REGRESSION

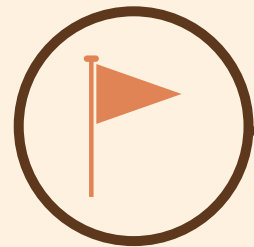




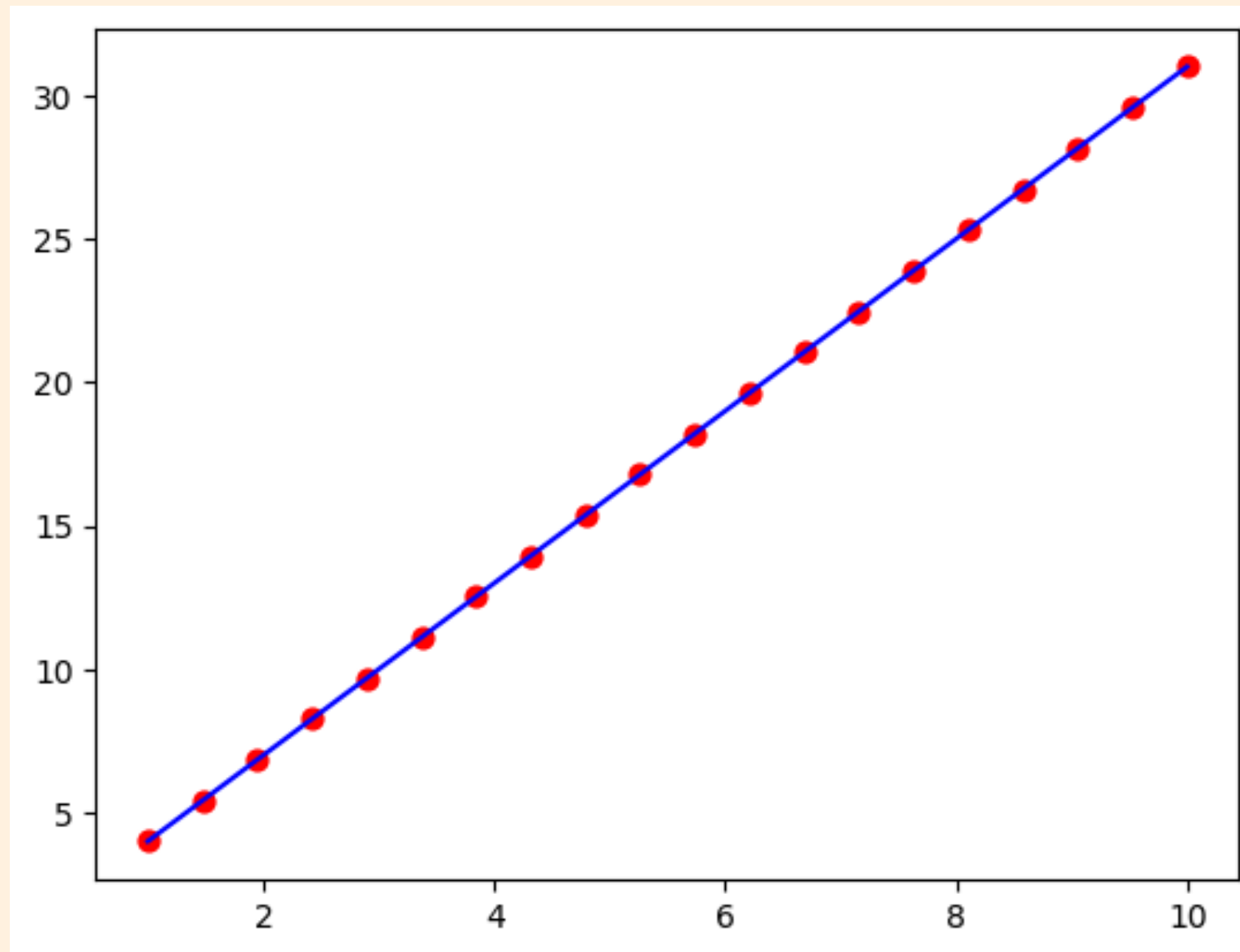
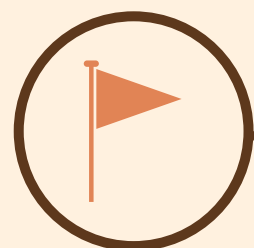
Mở Rộng



LINEAR REGRESSION MỞ RỘNG



```
1 from sklearn.linear_model import LinearRegression
2 import numpy as np
3 import matplotlib.pyplot as plt
4 X = np.linspace(1, 10, 20).reshape(20, 1)
5 y = 1 + 3 * X
6 model = LinearRegression()
7 model.fit(X, y)
8 y_pred = model.predict(X)
9
10 plt.scatter(X, y, c="r", s=40)
11 plt.plot(X, y_pred, c="b")
12
```



**LINEAR
REGRESSION
MỞ RỘNG**

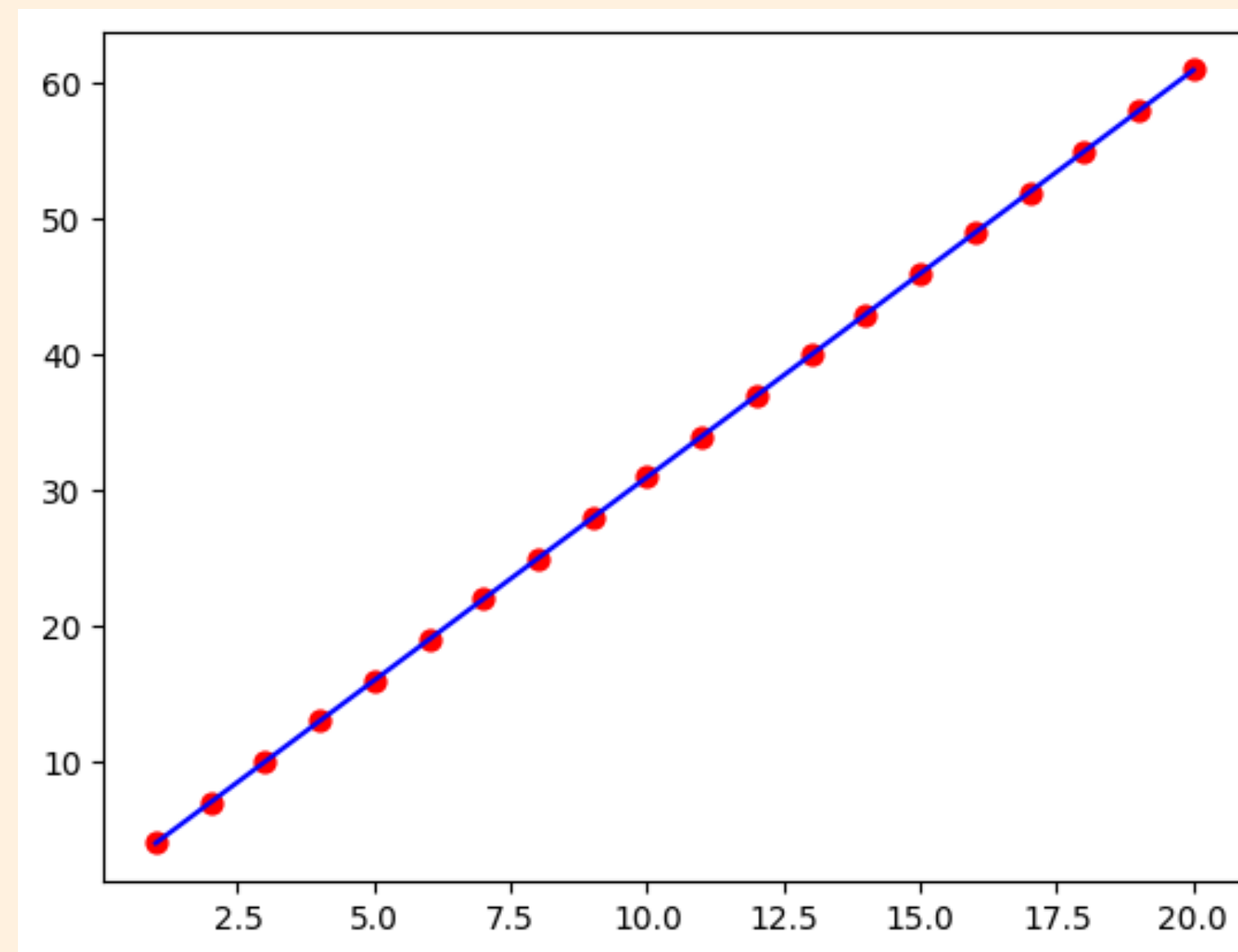
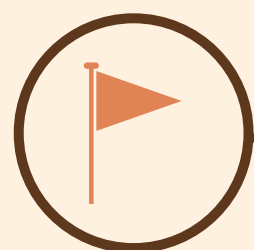




LINEAR REGRESSION MỞ RỘNG



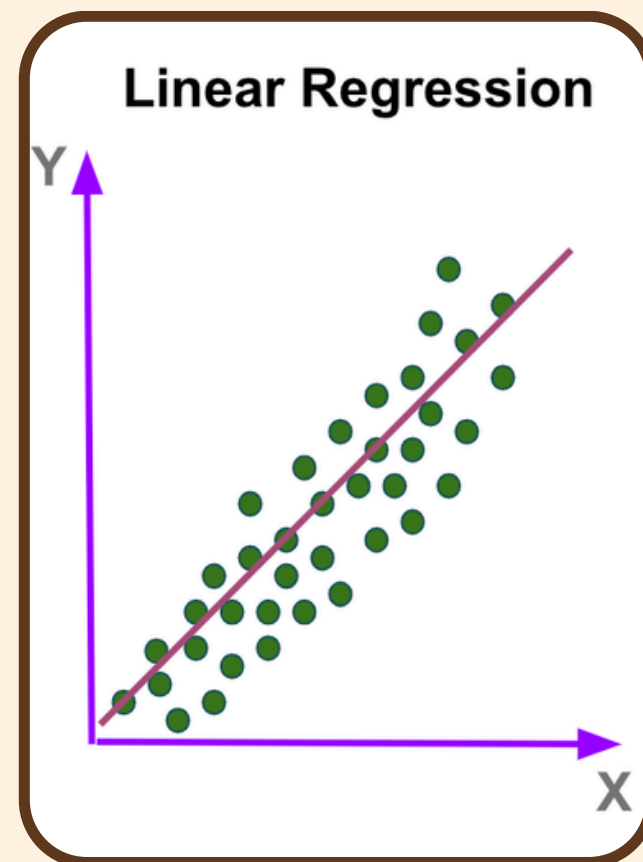
```
1
2 import torch
3 from torch import nn
4 class Linear(nn.Module, object):
5     def __init__(self) -> None:
6         super().__init__()
7         self.weight = nn.Parameter(data=torch.rand(1), requires_grad=True)
8         self.bias = nn.Parameter(data=torch.rand(1), requires_grad=True)
9
10    def forward(self, x: torch.Tensor):
11        return x * self.weight + self.bias
12 X = torch.linspace(1, 20, 20)
13 y = 1 + 3 * X
14 model = Linear()
15 loss_fn = nn.L1Loss()
16 optim = torch.optim.Adam(lr=0.01, params=model.parameters())
17 for loop in range(1000):
18     model.train()
19
20     y_pred = model(X)
21
22     loss = loss_fn(y_pred, y)
23
24     optim.zero_grad()
25     loss.backward()
26
27     optim.step()
28 y_pred = model(X)
29 plt.scatter(X.numpy(), y.numpy(), c="r", s=40)
30 plt.plot(X.numpy(), y_pred.detach().numpy(), c="b")
```



**LINEAR
REGRESSION
MỞ RỘNG**



TỔNG KẾT



- Supervised learning
 - Linear regression
 - Toán Học
 - Gradient descent
 - Đánh giá
 - Mở rộng
-



DISCUSSION
