



# **Natural Language Processing with Pytorch**

## **Bài 2: PYTORCH CƠ BẢN**



[https://csc.edu.vn/data-science-machine-learning/natural-  
language-processing-with-deep-learning\\_293](https://csc.edu.vn/data-science-machine-learning/natural-language-processing-with-deep-learning_293)

**2023**



# PYTORCH CƠ BẢN



## I. Tổng quan về Pytorch

## II. Pytorch cơ bản

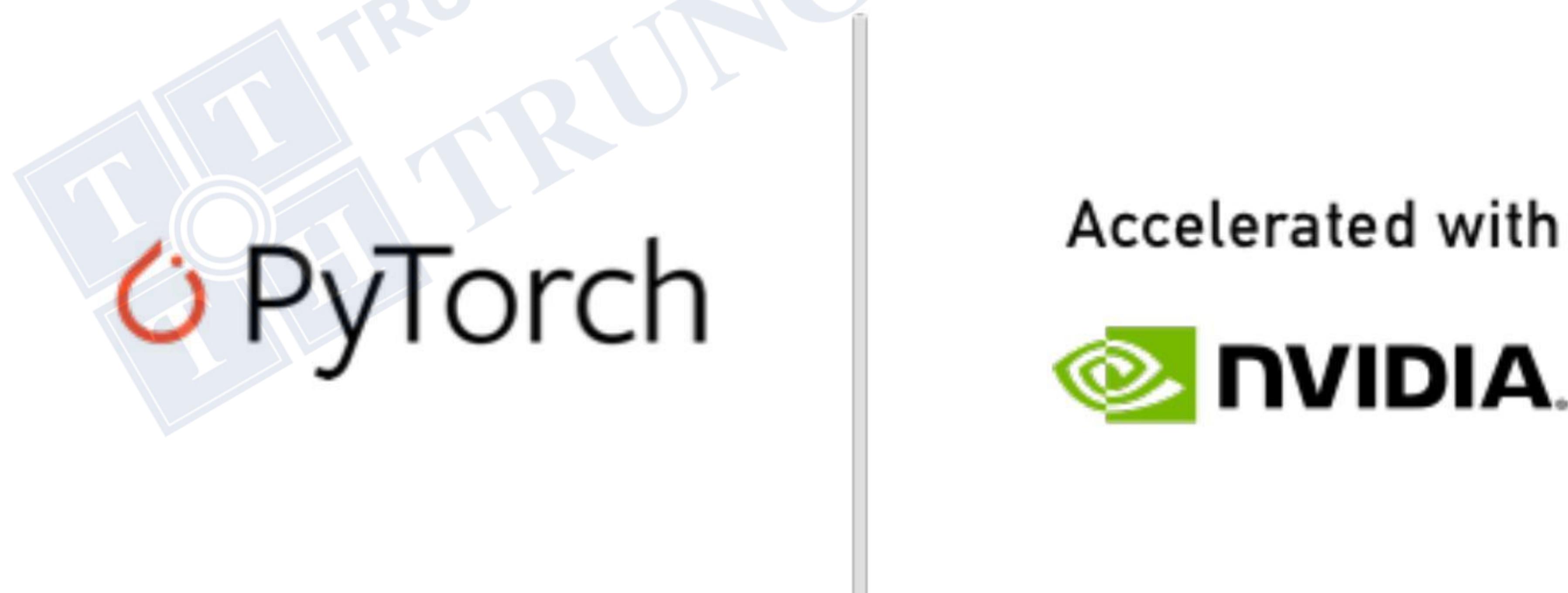
## III. Tensor, các phép toán Tensor và Gradient





# Tổng quan về Pytorch

- Pytorch là thư viện mã nguồn mở cho Machine Learning
- Được phát triển bởi Facebook's AI Research Lab
- Tận dụng được sức mạnh của GPUs
- Tính toán tự động các gradients
- Tối ưu tốt hơn → Nhanh, gọn và dễ sử dụng





# Tại sao chọn Pytorch?



```
import numpy as np
np.random.seed(0)

N, D = 3, 4

x = np.random.randn(N, D)
y = np.random.randn(N, D)
z = np.random.randn(N, D)

a = x * y
b = a + z
c = np.sum(b)

grad_c = 1.0
grad_b = grad_c * np.ones((N, D))
grad_a = grad_b.copy()
grad_z = grad_b.copy()
grad_x = grad_a * y
grad_y = grad_a * x
```



```
import numpy as np
np.random.seed(0)
import tensorflow as tf

N, D = 3, 4

with tf.device('/gpu:0'):
    x = tf.placeholder(tf.float32)
    y = tf.placeholder(tf.float32)
    z = tf.placeholder(tf.float32)

    a = x * y
    b = a + z
    c = tf.reduce_sum(b)

grad_x, grad_y, grad_z = tf.gradients(c, [x, y, z])

with tf.Session() as sess:
    values = {
        x: np.random.randn(N, D),
        y: np.random.randn(N, D),
        z: np.random.randn(N, D),
    }
    out = sess.run([c, grad_x, grad_y, grad_z],
                  feed_dict=values)
    c_val, grad_x_val, grad_y_val, grad_z_val = out
```



```
import torch
N, D = 3, 4

x = torch.rand((N, D), requires_grad=True)
y = torch.rand((N, D), requires_grad=True)
z = torch.rand((N, D), requires_grad=True)

a = x * y
b = a + z
c = torch.sum(b)

c.backward()
```

→ Pytorch nhanh (*dùng GPU*), gọn (so với NumPy và TensorFlow), có hàm chạy gradient tự động (so với NumPy).

# PYTORCH CƠ BẢN



## I. Tổng quan về Pytorch

## II. Pytorch cơ bản

## III. Tensor, các phép toán Tensor và Gradient





# Khởi chạy Pytorch

## ☐ Tải Pytorch:

Bằng *Anaconda/ Miniconda*:

```
conda install pytorch-c pytorch
```

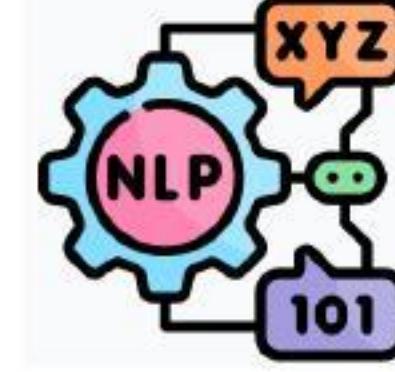
Bằng *pip*:

```
pip3 install torch
```



# Tổng quan về Tensor

- Tensors tương tự như các array của NumPy.
- Khác biệt ở chỗ tensor có thể sử dụng trên GPU để tăng hiệu suất tính toán.
- Các phép tính khởi tạo, xử lý tensor cũng tương tự như với các array của NumPy .  
*(rand, ones, zeros, reshape, transpose, ...)*



# Các dạng dữ liệu Tensor

Loại dữ liệu	Số chiều	Đặc điểm
Scalar	0	<ul style="list-style-type: none"><li>Biểu diễn 1 số duy nhất.</li><li>Vô hướng.</li><li>Ký hiệu viết thường (<math>x</math>).</li><li>Pytorch: <code>scalar = torch.tensor(7)</code></li></ul>
Vector	1	<ul style="list-style-type: none"><li>Biểu diễn được trên 1 trực, thành 1 mặt phẳng.</li><li>Có hướng.</li><li>Ký hiệu viết thường in đậm (<math>\mathbf{x}_i</math>).</li><li>Pytorch: <code>vector = torch.tensor([7, 7])</code></li></ul>
Matrix	2	<ul style="list-style-type: none"><li>Biểu diễn được trên 2 trực.</li><li>Có hướng.</li><li>Ký hiệu viết hoa (<math>X</math>).</li><li>Pytorch: <code>MATRIX = torch.tensor([7, 8], [9, 10])</code></li></ul>
Tensor	bất kỳ	<ul style="list-style-type: none"><li>Biểu diễn bất kỳ dạng dữ liệu số.</li><li>Có hướng hoặc không.</li><li>Pytorch: <code>scalar = torch.tensor([[1, 2, 3], [4, 5, 6], [7, 8, 9]])</code></li></ul>



# Các dạng dữ liệu Tensor

Ví dụ về **scalar**,  
**vector**, **matrix**  
và **tensor**

**Scalar**

7

**Vector**

7
4

or

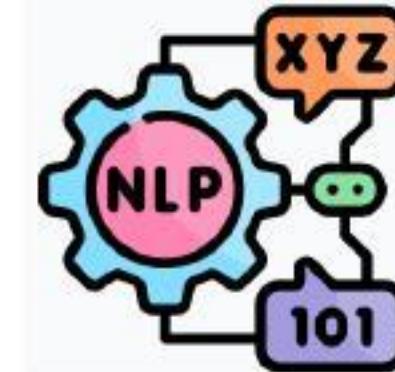
7	4
---	---

**Matrix**

7	10
4	3
5	1

**Tensor**

[7]	[4]	[0]	[1]
[1]	[9]	[2]	[3]
[5]	[6]	[8]	[8]



# Khởi tạo một Tensor

- ☐ Có hai cách chính để khởi tạo một **tensor**:

```
x = [[1, 2],[3, 4]]
```

```
x_tensor = torch.tensor(data)
```

## Trực tiếp từ data

Tensor có thể được tạo trực tiếp từ dữ liệu có sẵn.

```
array = np.arange(1.0, 8.0)
```

```
# Từ numpy sang tensor:
```

```
Tensor = torch.from_numpy(array)
```

```
# Từ tensor sang numpy:
```

```
Numpy = Tensor.numpy()
```

## Thông qua NumPy

Tensor cũng có thể được tạo từ NumPy array (và *ngược lại*).



# Lưu trữ Tensor

- ☐ Có thể lưu tensor trên **CPU** hoặc **GPU**.

## 1. Kiểm tra thiết bị có GPU không

```
!nvidia-smi
```

## 2. Kiểm tra quyền truy cập GPU

```
torch.cuda.is_available()
```

## 3. Kiểm tra số device GPU Pytorch sử dụng được

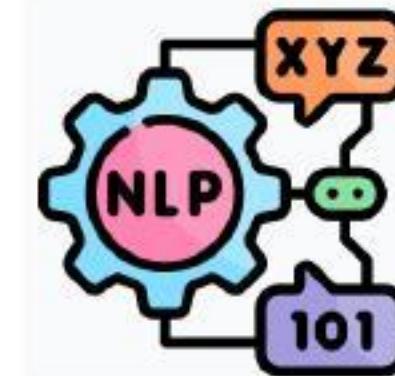
```
torch.cuda.device_count()
```

## 4. Thay đổi nơi lưu trữ tensor giữa các device

```
tensor.to(device)
```

Chuyển tensor về CPU (*nếu đang lưu trên GPU*)

```
tensor.numpy()
```



# Sử dụng Pytorch tạo Tensor

Hàm	Ý nghĩa	Ví dụ
<b>Random</b>	Tạo tensor với các phần tử con bất kỳ theo một cấu trúc nhất định	<code>tensor = torch.rand(size=(3,4))</code>
<b>Zeros</b>	Tạo tensor có tất cả phần tử con bằng 0 theo một cấu trúc nhất định	<code>tensor = torch.zeros(size=(3,4))</code>
<b>Zeros like</b>	Tạo tensor có tất cả phần tử con bằng 0 theo cấu trúc của một tensor khác	<code>tensor_a = torch.rand(size=(3,2)) tensor_b = torch.zeros_like(tensor_a)</code>
<b>Ones</b>	Tạo tensor với toàn bộ phần tử con bằng 1 theo một cấu trúc nhất định	<code>tensor = torch.ones(size=(3, 6))</code>
<b>Range</b>	Tạo tensor có các phần tử trong một khoảng cho trước, theo một cấu trúc nhất định	<code>tensor = torch.arange(start,end,step)</code>

# PYTORCH CƠ BẢN



## I. Tổng quan về Pytorch

## II. Pytorch cơ bản

## III. Tensor, các phép toán Tensor và Gradient

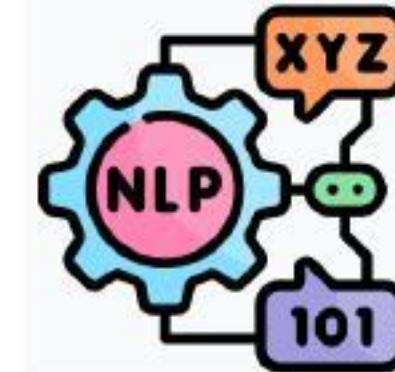




# Tensor Attributes

□ **Tensor attributes** cho biết các thông tin, đặc điểm của tensor. Một số hàm phổ biến:

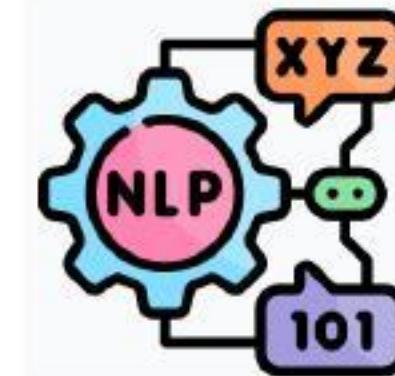
Hàm	Ý nghĩa	Ví dụ
		<code>tensor = torch.rand(3, 4)</code>
<b>shape</b>	Cấu trúc tensor như thế nào?	<code>tensor.shape()</code> → <code>torch.Size([3, 4])</code>
<b>dtype</b>	Phần tử của tensor có dạng dữ liệu nào?	<code>tensor.dtype()</code> → <code>torch.float32</code>
<b>device</b>	Tensor đang được lưu trữ trên GPU hay CPU?	<code>tensor.device()</code> → <code>cpu</code>



# Tensor Operations

□ **Tensor operations** là các hàm dùng để tính toán tensor. Một số hàm phổ biến:

Hàm	Ý nghĩa	Ví dụ
<b>Addition</b>	Cộng	<code>tensor = torch.tensor([1, 2, 3])</code> <code>tensor + 10</code> ~ <code>tensor.add(10)</code> → <code>tensor([11, 12, 13])</code>
<b>Subtraction</b>	Trừ	<code>tensor - 10</code> ~ <code>tensor.subtract(10)</code> → <code>tensor([-9, -8, -7])</code>
<b>Multiplication</b>	Nhân (tương tự với chia)	<code>tensor * 10</code> ~ <code>tensor.mul(10)</code> → <code>tensor([10, 20, 30])</code>
<b>Matrix multiplication</b>	Nhân ma trận (kiểm tra điều kiện)	<code>tensor * tensor</code> ~ <code>torch.matmul(tensor, tensor)</code> ~ <code>torch.mm(tensor, tensor)</code> → <code>tensor(14)</code>



# Tensor Aggregations (*min, max, mean, sum, ...*)

- Tensor aggregations là các hàm dùng để tính tổng và thống kê mô tả cho tensor, như sau:

Hàm	Ý nghĩa	Ví dụ
		tensor=torch.arange(0, 100, 10)
<b>Sum</b>	Tổng tất cả các phần tử	tensor.sum() → 450
<b>Mean</b>	Giá trị trung bình của các phần tử	tensor.mean() → 45.0
<b>Min</b>	Giá trị nhỏ nhất trong các phần tử	tensor.min() → 0
<b>Max</b>	Giá trị lớn nhất trong các phần tử	tensor.max() → 90
<b>ArgMin</b>	Index của giá trị nhỏ nhất	tensor.argmin() → 0
<b>ArgMax</b>	Index của giá trị lớn nhất	tensor.argmax() → 8



# Tensor Datatypes

**Tensor datatype** cho biết mức độ tính toán chính xác của các model sử dụng Pytorch.

Datatype càng lớn thì độ **chính xác** **càng cao**, thời gian tính toán **càng chậm**, và ngược lại.

`torch.int8`

Các loại **tensor datatype** phổ biến

`torch.float64`  
`torch.double`

`torch.float16`  
`torch.half`

`torch.float32`  
`torch.float`

*Pytorch định nghĩa 10 loại datatype cho tensor lưu trên CPU và GPU.*



# Thay đổi datatype của Tensor

## Tạo tensor và kiểm tra tensor datatype

```
tensor = torch.arange(10., 100., 10.)  
tensor.dtype  
→ torch.float32
```

## Tạo tensor có datatype là float16

```
tensor_float16 = tensor.type(torch.float16)  
tensor_float16  
→ tensor([10., 20., 30., 40., 50., 60., 70., 80., 90.],  
        dtype=torch.float16)
```

## Tạo tensor có datatype là int8

```
tensor_int8 = tensor.type(torch.int8)  
tensor_int8.dtype  
→ tensor([10., 20., 30., 40., 50., 60., 70., 80., 90.],  
        dtype=torch.float16)
```



# Thay đổi cấu trúc Tensor

Phương pháp	Chức năng
<code>torch.reshape(input, shape)</code>	Tổ chức lại tensor theo một cấu trúc mới.
<code>tensor.view(shape)</code>	Xem tensor theo cấu trúc mới nhưng không làm thay đổi cấu trúc gốc của tensor.
<code>torch.stack(tensors, dim=0)</code>	Chèn thêm chuỗi các tensor dọc theo một chiều (dim), các tensor phải có cùng shape.
<code>torch.squeeze(input)</code>	Loại bỏ các chiều (dim) chỉ có 1 phần tử.
<code>torch.unsqueeze(input, dim)</code>	Thêm vào một chiều một phần tử tại chiều dim cho trước.
<code>torch.permute(input, dims)</code>	Xem tensor với cấu trúc được hoán vị theo số chiều mới mà không thay đổi tensor gốc.

**Mô hình Deep Learning cần xử lý các tensor theo một cách nào đó để tính toán, việc thay đổi cấu trúc tensor là cần thiết, nếu không sẽ gặp lỗi.**



# Thay đổi cấu trúc Tensor

Phương pháp	Chức năng
<code>torch.reshape(input, shape)</code>	
<code>tensor.view(shape)</code>	
<code>torch.stack(tensors, dim=0)</code>	
<code>torch.squeeze(input)</code>	
<code>torch.unsqueeze(input, dim)</code>	
<code>torch.permute(input, dims)</code>	



# Tensor Indexing

□ Truy xuất dữ liệu tại 1 phần tử nhất định của tensor.

*Tương tự như NumPy array.*

```
x = torch.arange(1,10)  
x.reshape(1,3,3)  
x, x.shape
```

```
→ (tensor([[[1, 2, 3],  
           [4, 5, 6],  
           [7, 8, 9]]]),  
torch.Size([1, 3, 3]))
```

```
x[:, 0]
```

```
→ tensor([[1, 2, 3]])
```

```
x[:, :, 0]
```

```
→ tensor([[2, 5, 8]])
```

```
print(f"1 là: \n{x[0]})  
print(f"2 là: {x[0][0]})  
print(f"3 là: {x[0][0][0]})
```

```
→ 1 là:  
    tensor([[1, 2, 3],  
            [4, 5, 6],  
            [7, 8, 9]])  
2 là: tensor([1, 2, 3])  
3 là: 1
```

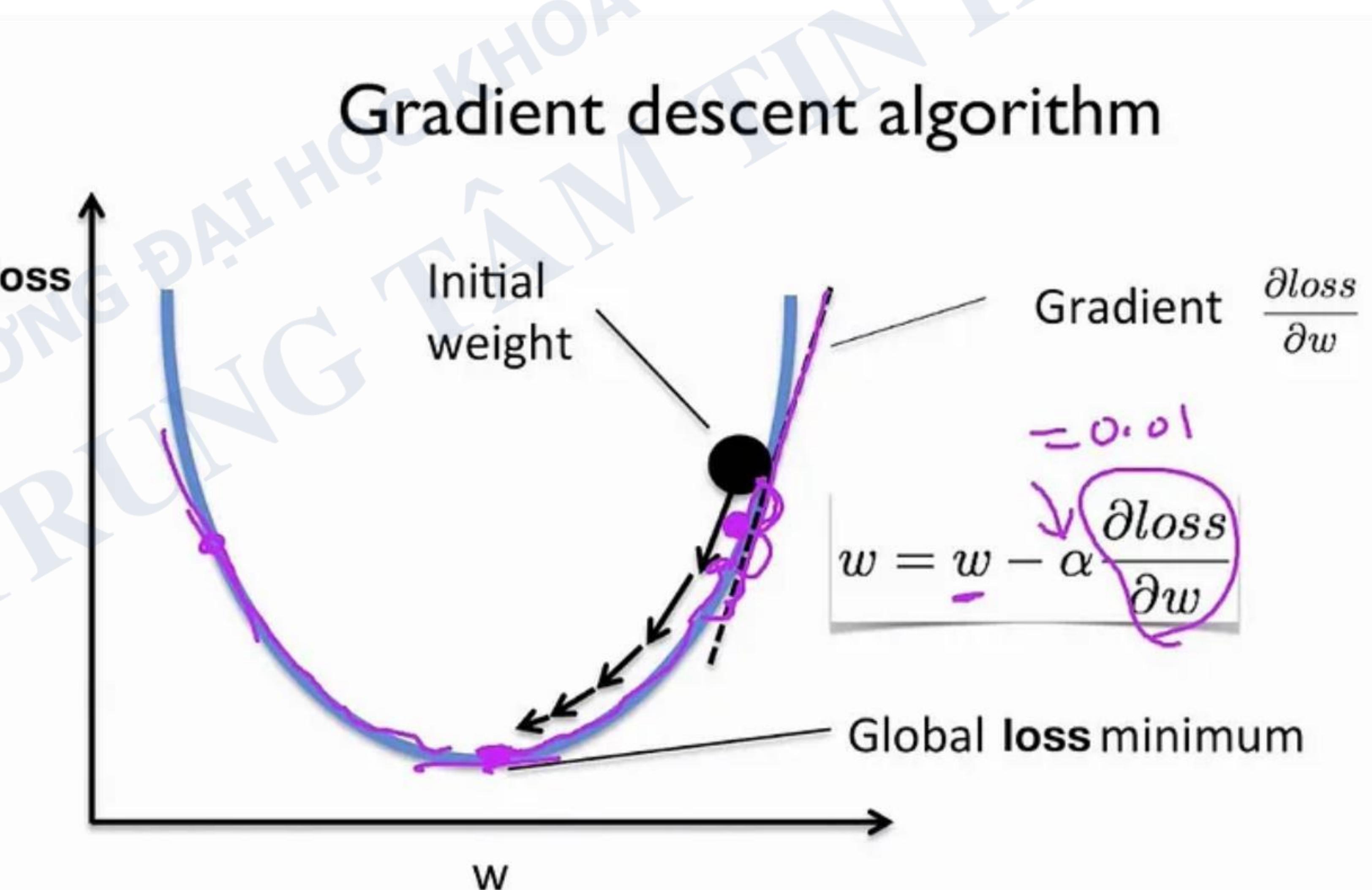
```
x[:, 1, 1]
```

```
→ tensor([5])
```

# Gradient Descent

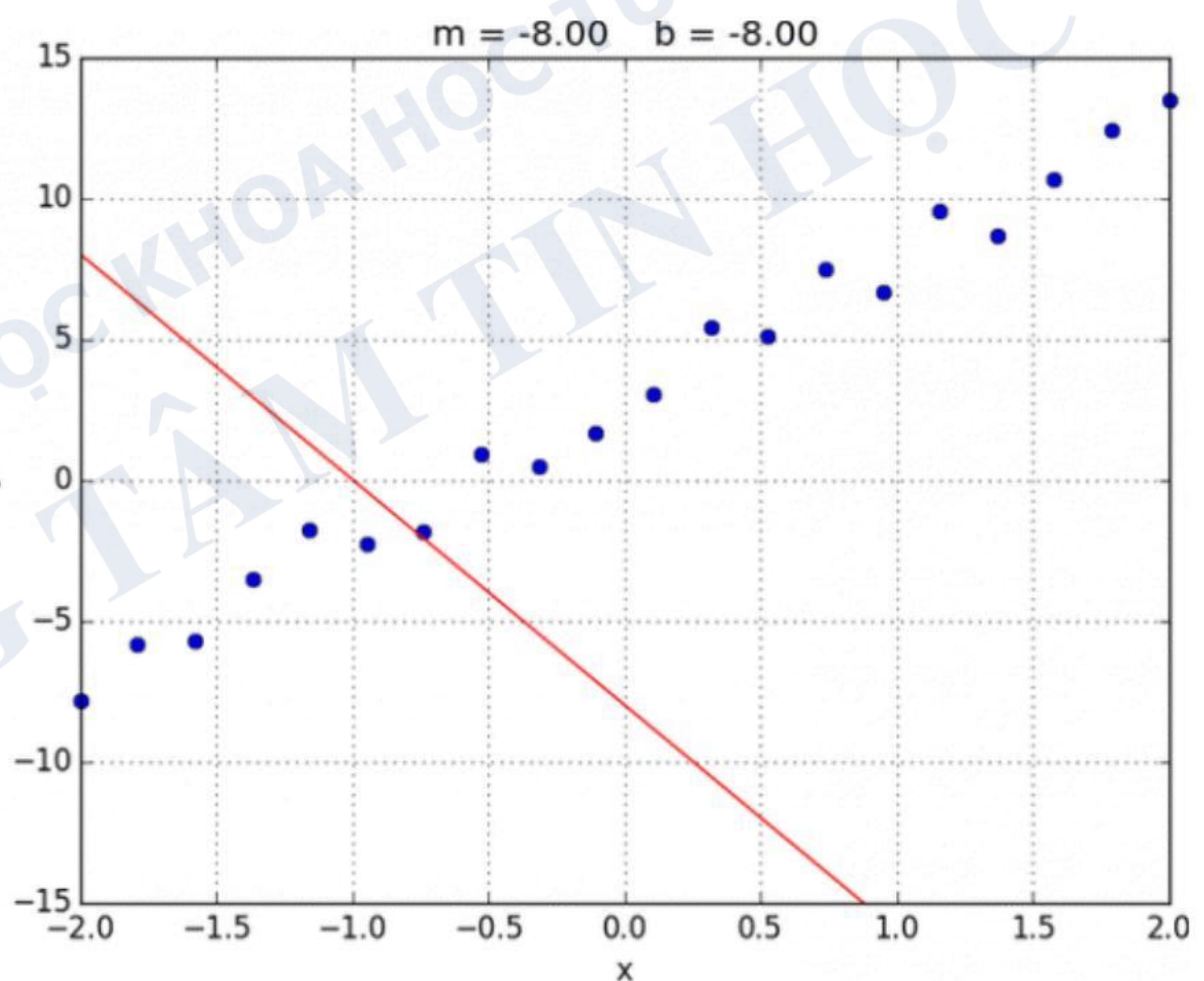
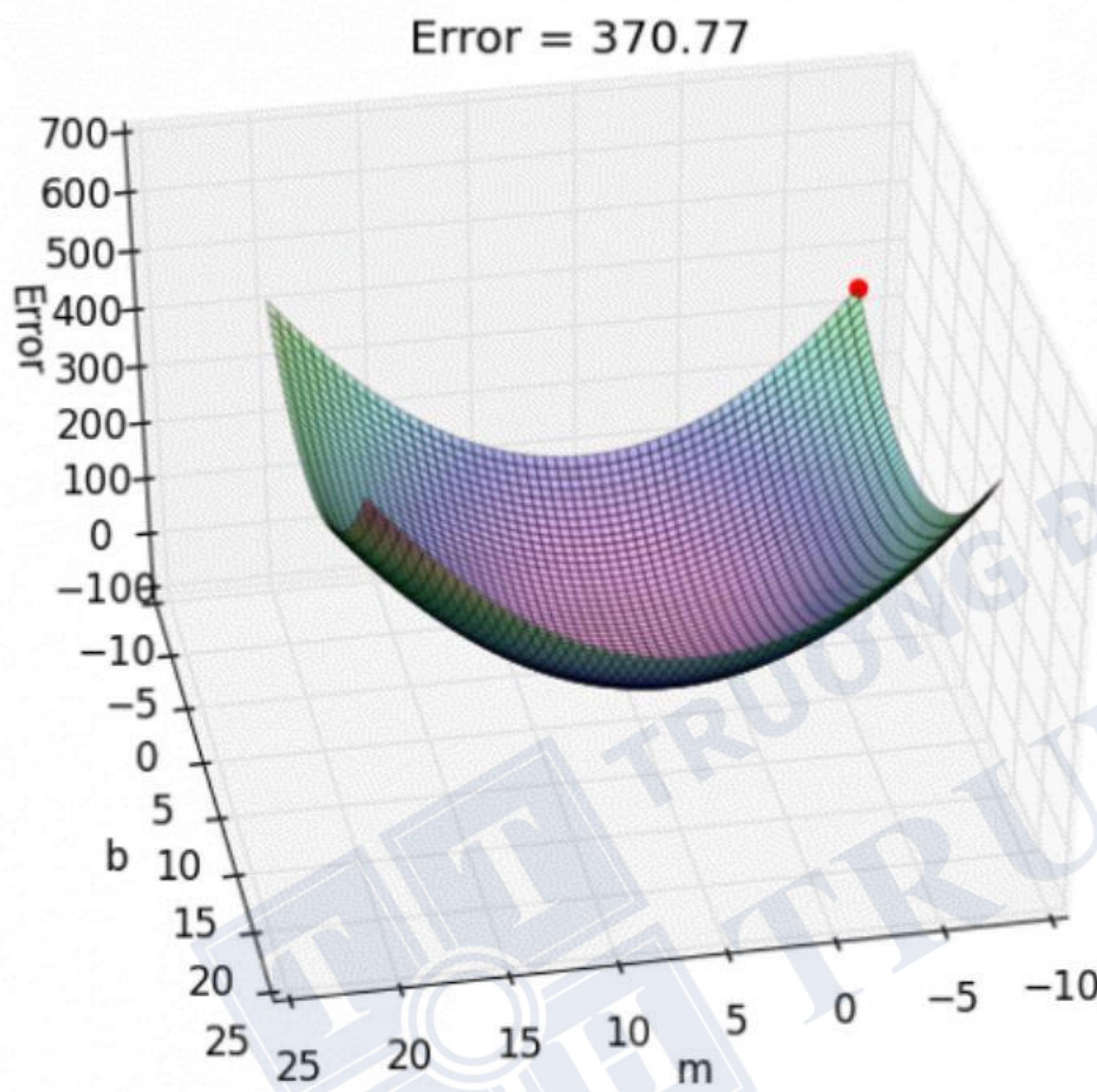
- Gradient Descent là một thuật toán tối ưu hóa mô hình Machine Learning sử dụng để tìm giá trị nhỏ nhất của một hàm số (*loss function*).

*Gradient descent điều chỉnh các tham số của một mô hình dựa trên giá trị loss nhỏ nhất.*



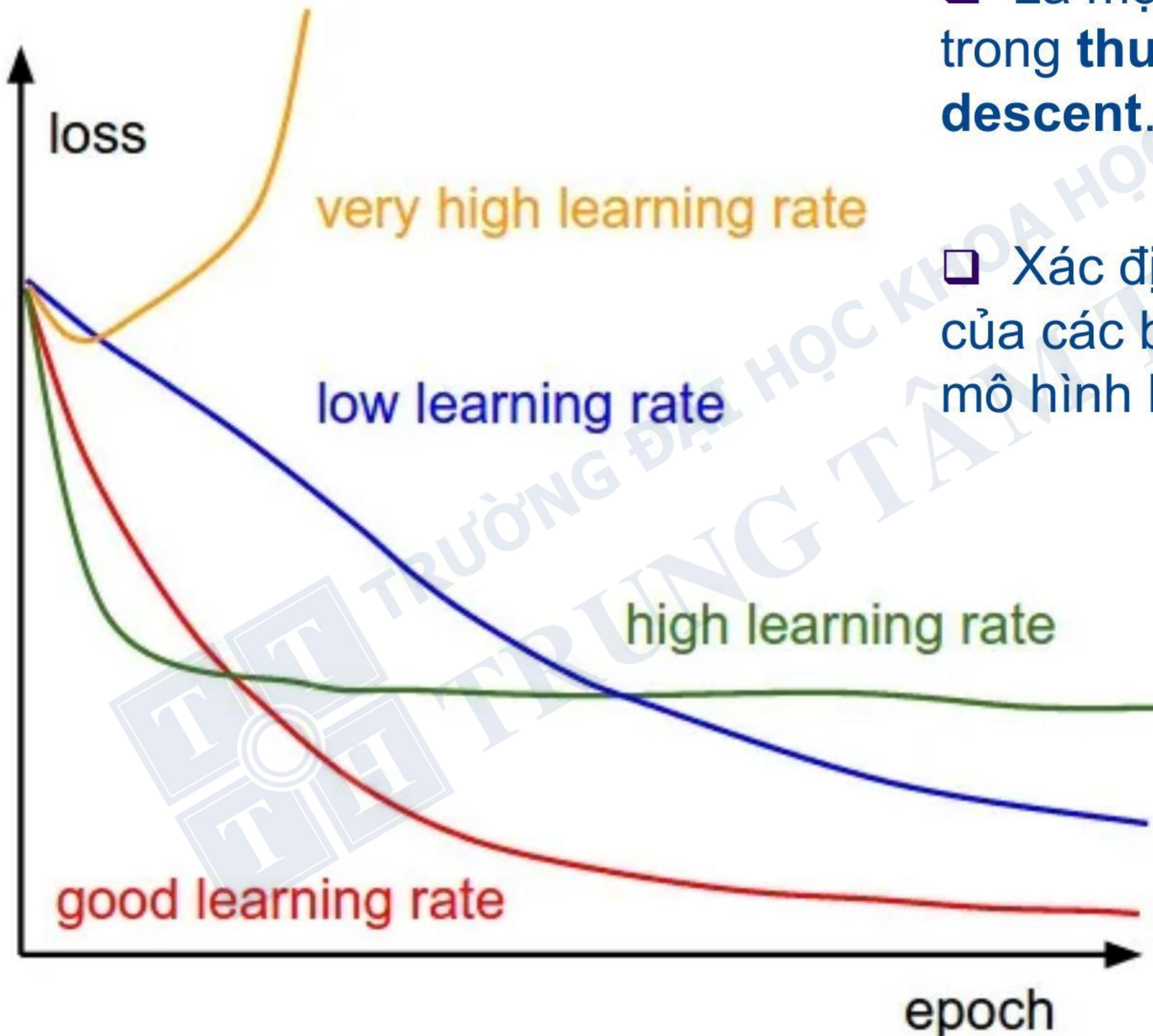
# Gradient Descent

Gradient Descent tính toán gradient của hàm số tại mỗi điểm và di chuyển theo hướng **ngược lại** để tiến gần đến điểm cực tiểu.



*Quá trình này được lặp lại cho đến khi đạt được điều kiện dừng như đạt đủ số lần lặp hay đạt đến giá trị loss nhất định.*

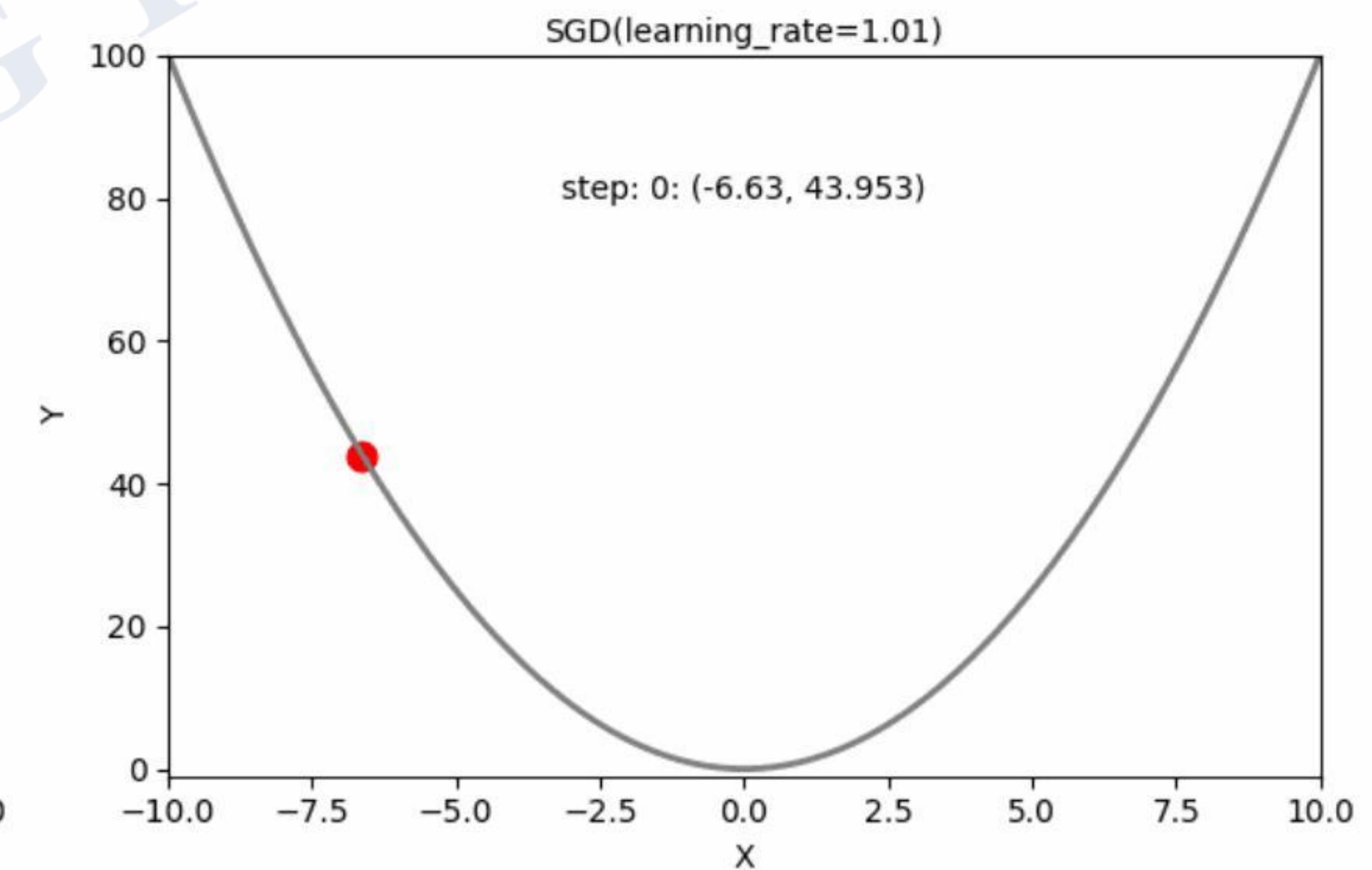
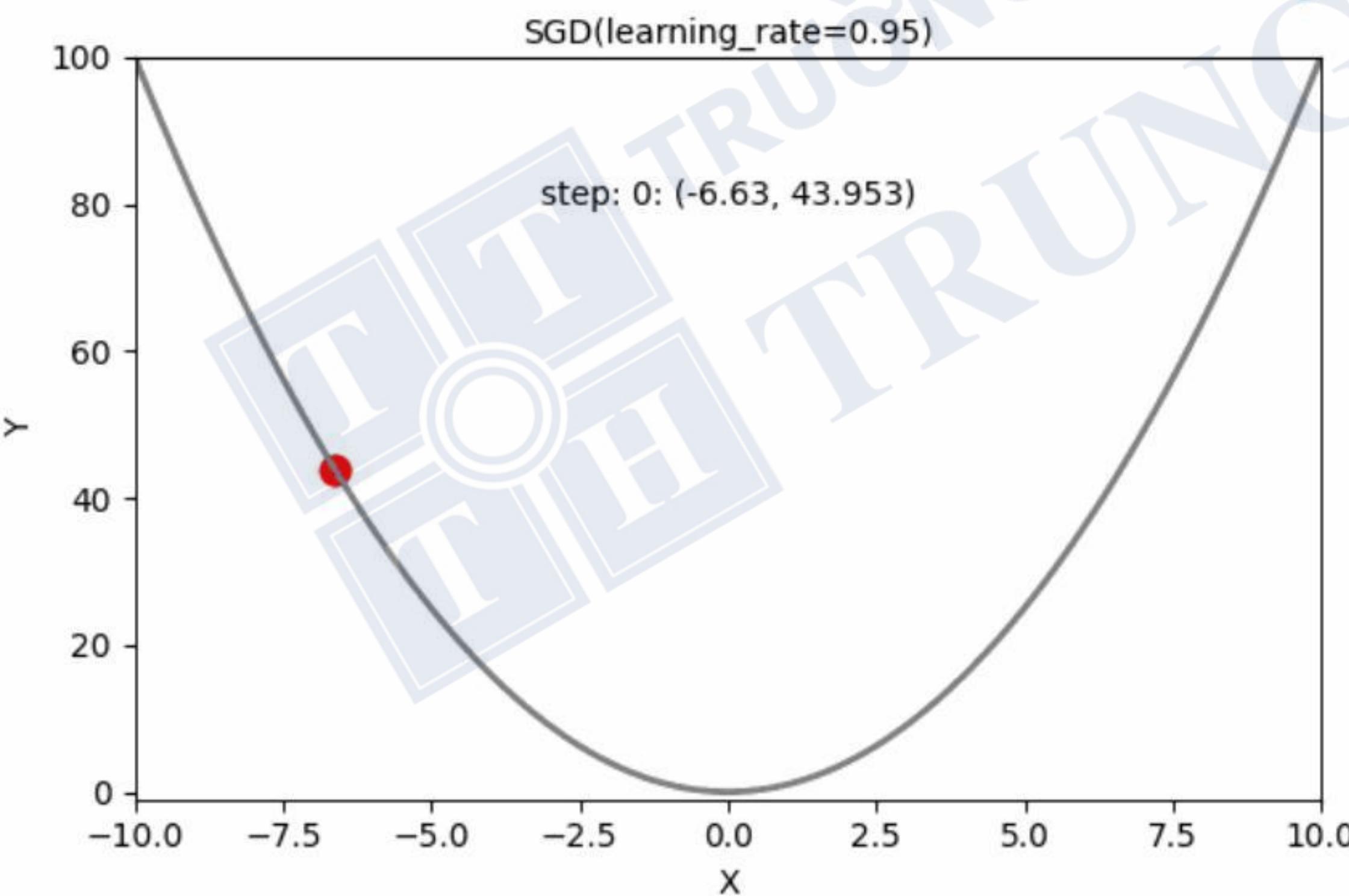
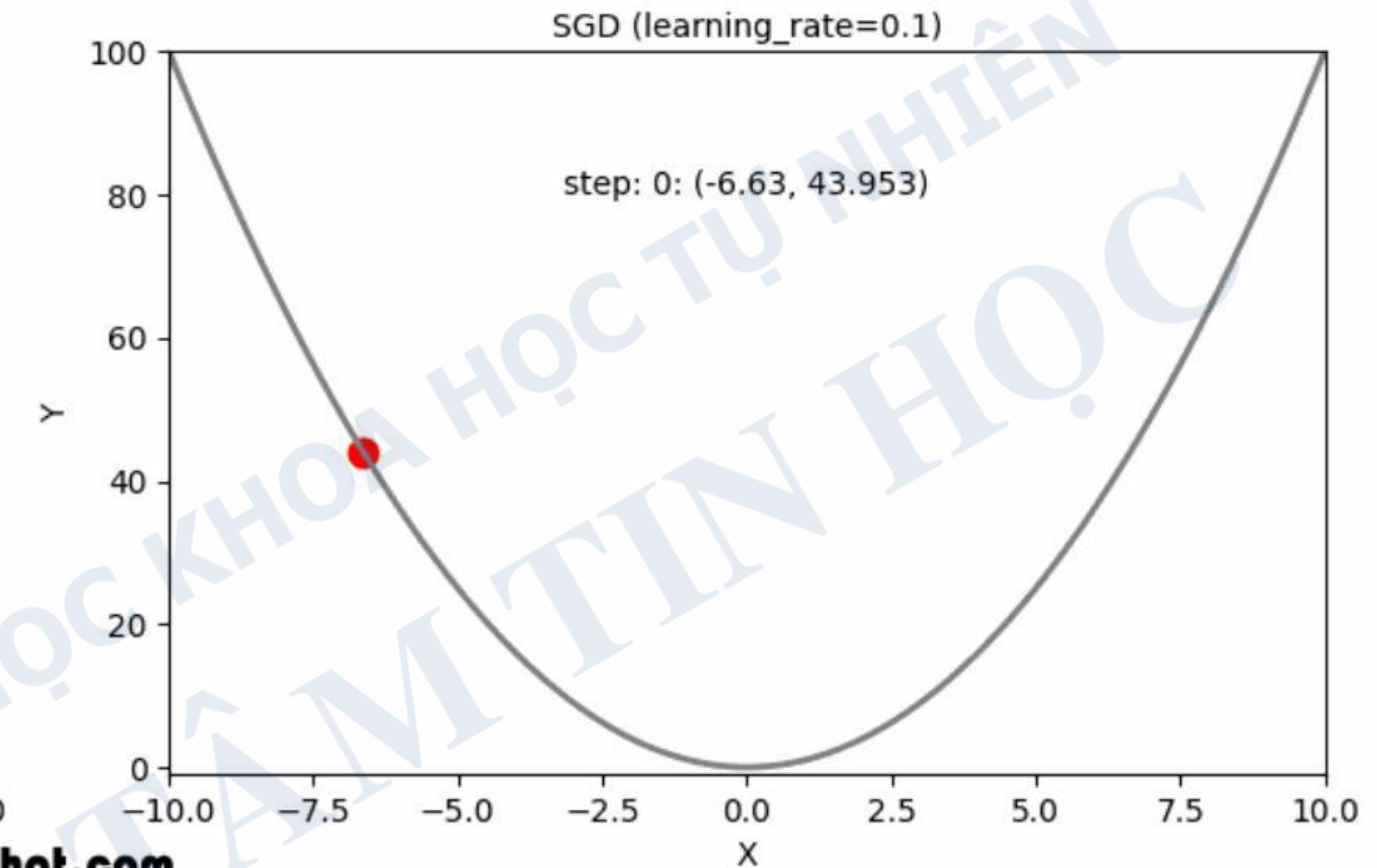
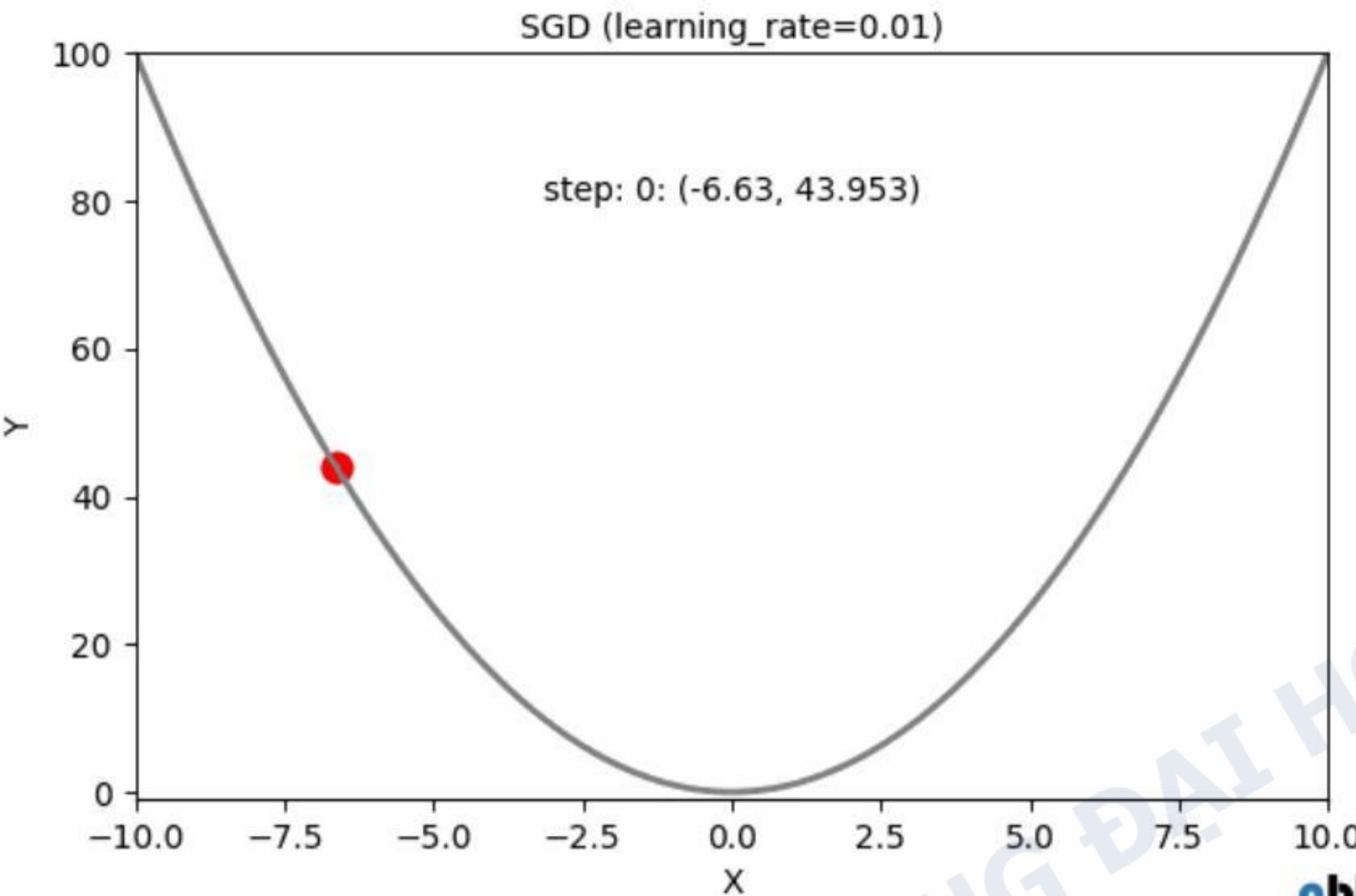
# Learning Rate



- ❑ Là một tham số quan trọng trong **thuật toán gradient descent**.
- ❑ Xác định khoảng di chuyển của các bước cập nhật tham số mô hình khi tối ưu hóa hàm loss.



# Các trường hợp của Learning Rate



# Code Demo



**DEMO**



# Q&A

---

