Số hóa và Quản trị thông tin số - Lab 6

March 14, 2025

Nguyễn Thái Nguyên - 2274802010587

1 *Pytorch Basic*

1.1 Cơ bản về Pytorch

[13]:
```
import torch
```

[14]:
```
torch.cuda.is_available()
```

[14]:
```
False
```

Sử dụng GPU và Cuda

[22]:
```
torch.cuda.current_device
```

[22]:
```
<function torch.cuda.current_device() -> int>
```

torch.cuda.get_device_name(0)

[26]:
```
# trả về mức sử dụng bộ nhớ GPU hiện  theo tensors tính bằng byte cho thiết bị
torch.cuda.memory_allocated()
```

[26]:
```
0
```

[28]:
```
# Trả về bộ nhớ GPU hiện tại được quản lý bởi bộ phân bổ bộ nhớ đệm theo byte
torch.cuda.memory_cached()
```

```
C:\Users\Admin\AppData\Local\Temp\ipykernel_51120\1580366354.py:2:
FutureWarning: `torch.cuda.memory_cached` has been renamed to
`torch.cuda.memory_reserved`
  torch.cuda.memory_cached()
```

[28]:
```
0
```

Dataset with Pytorch

[41]:
```
# Loading data iris
import torch
import numpy as np
```

```
import pandas as pd

df = pd.read_csv('iris.csv')
df.head()
```

[41]:

	Id	SepalLengthCm	SepalWidthCm	PetalLengthCm	PetalWidthCm	Species
0	1	5.1	3.5	1.4	0.2	Iris-setosa
1	2	4.9	3.0	1.4	0.2	Iris-setosa
2	3	4.7	3.2	1.3	0.2	Iris-setosa
3	4	4.6	3.1	1.5	0.2	Iris-setosa
4	5	5.0	3.6	1.4	0.2	Iris-setosa

[53]:
```
from sklearn.preprocessing import LabelEncoder
from sklearn.model_selection import train_test_split

le = LabelEncoder()
X = df.drop(["Id", "Species"], axis = 1).values
y = le.fit_transform(df["Species"].values)
```

[57]:
```
# chia dữ liệu với test size = 0,2
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.2,␣
 ↪random_state = 42)

X_train = torch.FloatTensor(X_train)
X_test = torch.FloatTensor(X_test)
y_train = torch.FloatTensor(y_train).reshape(-1, 1)
y_test = torch.FloatTensor(y_test).reshape(-1, 1)
```

[61]:
```
print(f"train_size {len(y_train)}")
```

```
train_size 120
```

[67]:
```
labels, counts = y_train.unique(return_counts=True)
print(labels, counts)
```

```
tensor([0., 1., 2.]) tensor([40, 41, 39])
```

[101]:
```
# Tính đạo hàm bằng Pytorch
# Cho y = 2x^4 + x^3 + 3x^2 + 5x + 1
# Tính y'
import torch

# Tạo một tensor với requires_grad được đặt thành True
x = torch.tensor(2.0, requires_grad=True)
print(x)
print(x.grad)

# Định nghĩa hàm
```

```
y = 2 * x**4 + x**3 + 3 * x**2 + 5 * x + 1
print(y)
print(y.grad_fn)

# Thực hiện truyền ngược và tính toán tất cả các gradient
y.backward()

# Kết quả đạo hàm
print(x.grad)
```

```
tensor(2., requires_grad=True)
None
tensor(63., grad_fn=<AddBackward0>)
<AddBackward0 object at 0x0000017E83348280>
tensor(93.)
```

[103]:
```
# BTVN

# Tính y' của y = 5x^6 + 3x^3 + 2x + x + 2x + 5x^4
# Cho biết độ dốc của đa thức trên tại điểm nào

# Khởi tạo biến x và cho phép tính gradient
x = torch.tensor(2.0, requires_grad=True)  # Ví dụ: x = 2.0

# Định nghĩa hàm y
y = 5 * x**6 + 5 * x**4 + 3 * x**3 + 5 * x

# Tính đạo hàm
y.backward()

# Giá trị của y' tại x = 2
print(f"Đạo hàm tại x = {x.item()} là y' = {x.grad.item()}")
```

```
Đạo hàm tại x = 2.0 là y' = 1161.0
```

[127]:
```
import torch

def tinh_dao_ham(x_value):
    x = torch.tensor(x_value, dtype=torch.float32, requires_grad=True)

    # Định nghĩa hàm số y
    y = 5 * x**6 + 5 * x**4 + 3 * x**3 + 5 * x + 1

    # Tính đạo hàm
    y.backward()

    # Trả về giá trị đạo hàm tại x
    return x.grad.item()
```

```
# Nhập giá trị x từ người dùng
x_input = float(input("Nhập giá trị x: "))

# Tính độ dốc tại x_input
do_doc = tinh_dao_ham(x_input)

# Hiển thị kết quả
print(f"Độ dốc của đa thức tại x = {x_input} là y' = {do_doc}")
```

```
Nhập giá trị x:  1
```

```
Độ dốc của đa thức tại x = 1.0 là y' = 64.0
```

[129]:
```
#BTVN 1
# Tạo một tensor X có giá trị ban đầu là 2.0
# Định nghĩa hàm số và tính gradient
# y = x^3 + 2x^2 + 5x + 1
# hãy tính dy/dx tại giá trị của x
# Dùng phương pháp Gradient Descent với learning rate alpha = 0,1 để cập nhật␣
↪giá trị X trong 10 vòng lặp
```

[5]:
```
import torch

def tinh_dao_ham(x_value):
    x = torch.tensor(x_value, dtype=torch.float32, requires_grad=True)

    # Định nghĩa hàm số
    y = x**3 + 2*x**2 + 5*x + 1

    # Tính đạo hàm
    y.backward()

    # Trả về giá trị đạo hàm tại x
    return x.grad.item()

# Nhập giá trị x từ người dùng
x_input = float(input("Nhập giá trị x: "))

# Tính độ dốc tại x_input
do_doc = tinh_dao_ham(x_input)

# Hiển thị kết quả
print(f"Độ dốc của đa thức tại x = {x_input} là y' = {do_doc}")

# Learning rate
alpha = 0.1
```

```
# Số vòng lặp
epochs = 10

X = torch.tensor(x_input, dtype=torch.float32, requires_grad=True)

for i in range(epochs):
    y = X**3 + 2*X**2 + 5*X + 1
    y.backward()

    with torch.no_grad():
        X -= alpha * X.grad

    X.grad.zero_()

    print(f"Epoch {i+1}: X = {X.item()}")
```

```
Nhập giá trị x:  1

Độ dốc của đa thức tại x = 1.0 là y' = 12.0
Epoch 1: X = -0.20000004768371582
Epoch 2: X = -0.6320000290870667
Epoch 3: X = -0.9990272521972656
Epoch 4: X = -1.3988330364227295
Epoch 5: X = -1.9263200759887695
Epoch 6: X = -2.7690048217773438
Epoch 7: X = -4.4616193771362305
Epoch 8: X = -9.148786544799805
Epoch 9: X = -31.099361419677734
Epoch 10: X = -309.3106994628906
```

[ ]:

[131]:
```
#BTVN 2
# Tạo một tập dữ liệu giả lập với x là số giờ học (ngẫu nhiên từ 1 đến 10) và y là số điểm được tính theo công thức y = 3x + 5 + noise
# Với noise là một giá trị ngẫu nhiên nhỏ
# 1. Khởi tạo tham số w và b ngẫu nhiên với requires_grad = True
# 2. Tính MSE
# 3. Tính gradient
# 4. Cập nhật tham số w và b bằng Gradient Descent với Learning rate alpha = 0.01
# 5. Lặp lại quá trình trên trong 100 vòng lặp và quan sát sự hội tụ của mô hình
```

[7]:
```
import torch
import numpy as np
import random

# Tạo tập dữ liệu giả lập
```

```python
np.random.seed(42)
torch.manual_seed(42)
random.seed(42)

x_data = torch.tensor(np.random.randint(1, 11, 100), dtype=torch.float32)
noise = torch.tensor(np.random.randn(100) * 0.5, dtype=torch.float32)
y_data = 3 * x_data + 5 + noise  # y = 3x + 5 + noise

# Khởi tạo tham số w và b
w = torch.randn(1, requires_grad=True)
b = torch.randn(1, requires_grad=True)

# Learning rate
alpha = 0.01
# Số vòng lặp
epochs = 100

for epoch in range(epochs):
    # Tính giá trị dự đoán
    y_pred = w * x_data + b

    # Tính MSE
    loss = torch.mean((y_pred - y_data) ** 2)

    # Tính gradient
    loss.backward()

    # Cập nhật tham số bằng Gradient Descent
    with torch.no_grad():
        w -= alpha * w.grad
        b -= alpha * b.grad

    # Đặt gradient về 0 để tránh cộng dồn
    w.grad.zero_()
    b.grad.zero_()

    if (epoch + 1) % 10 == 0:
        print(f"Epoch {epoch+1}: Loss = {loss.item():.4f}, w = {w.item():.4f},␣
↪b = {b.item():.4f}")

print(f"Ket qua cuoi: w = {w.item():.4f}, b = {b.item():.4f}")
```

```
Epoch 10: Loss = 3.6406, w = 3.5905, b = 0.7400
Epoch 20: Loss = 3.3972, w = 3.5692, b = 0.8941
Epoch 30: Loss = 3.1712, w = 3.5488, b = 1.0426
Epoch 40: Loss = 2.9615, w = 3.5291, b = 1.1856
Epoch 50: Loss = 2.7669, w = 3.5101, b = 1.3235
Epoch 60: Loss = 2.5862, w = 3.4918, b = 1.4562
```

```
Epoch 70: Loss = 2.4185, w = 3.4742, b = 1.5842
Epoch 80: Loss = 2.2629, w = 3.4572, b = 1.7074
Epoch 90: Loss = 2.1184, w = 3.4409, b = 1.8261
Epoch 100: Loss = 1.9843, w = 3.4251, b = 1.9405
Ket qua cuoi: w = 3.4251, b = 1.9405
```

Pytorch with tensor

[136]:
```
import torch
import numpy as np
```

[138]:
```
torch.__version__
```

[138]:
```
'2.6.0+cpu'
```

[142]:
```
# Chuyển đổi mảng numpy sang tensor pytorch
arr = np.array([1, 2, 3, 4 ,5])
print(arr)
print(arr.dtype)
```

```
[1 2 3 4 5]
int32
```

[144]:
```
x = torch.from_numpy(arr)
print(x)
```

```
tensor([1, 2, 3, 4, 5], dtype=torch.int32)
```

[148]:
```
print(x.dtype)
print(x.type)
```

```
torch.int32
<built-in method type of Tensor object at 0x0000017E8A938F50>
```

[154]:
```
arr2 = np.arange(0,12).reshape(4,3)
arr2
```

[154]:
```
array([[ 0,  1,  2],
       [ 3,  4,  5],
       [ 6,  7,  8],
       [ 9, 10, 11]])
```

[156]:
```
x2 = torch.from_numpy(arr2)
print(x2)
print(x2.type())
```

```
tensor([[ 0,  1,  2],
        [ 3,  4,  5],
        [ 6,  7,  8],
        [ 9, 10, 11]], dtype=torch.int32)
torch.IntTensor
```

[168]:
```
# Copying and Sharing
arr = np.arange(0,5)
x = torch.from_numpy(arr)
print(x)
```

```
tensor([0, 1, 2, 3, 4], dtype=torch.int32)
```

[160]:
```
arr[0] = 99
print(x)
```

```
tensor([99,  1,  2,  3,  4], dtype=torch.int32)
```

[162]:
```
arr = np.arange(0,5)
x = torch.tensor(arr)
print(x)
```

```
tensor([0, 1, 2, 3, 4], dtype=torch.int32)
```

[164]:
```
arr[0] = 99
print(x)
```

```
tensor([0, 1, 2, 3, 4], dtype=torch.int32)
```

[11]:
```
#BTVN3 : Giải thích lí do tại sao cho 2 trường hopwr ở trên
    #TH1: torch.from_numpy(arr) tạo một tensor chia sẻ bộ nhớ (shared memory)␣
 ↪với mảng NumPy arr.
       # Điều này có nghĩa là mọi thay đổi trong arr cũng sẽ thay đổi x, vì␣
 ↪chúng trỏ đến cùng một vùng nhớ.
       # Khi arr[0] = 99, giá trị trong x cũng bị ảnh hưởng, dẫn đến␣
 ↪tensor([99, 1, 2, 3, 4]).
    #TH2: torch.tensor(arr) tạo một bản sao độc lập của arr, không chia sẻ bộ␣
 ↪nhớ với arr.
       # Khi arr[0] = 99, giá trị của x không bị ảnh hưởng vì nó đã được copy␣
 ↪sang vùng nhớ riêng biệt.
       # Do đó, x vẫn giữ nguyên giá trị tensor([0, 1, 2, 3, 4]).
```

[13]:
```
#BTVN4: Bạn hãy giúp thầy về nhà tạo tensor với:
# Empty
# Zeros
# Ones
# Random
# Reshape với view và view as
import torch
```

[15]:
```
#Empty
empty_tensor = torch.empty(3, 3)
print("Empty Tensor:")
print(empty_tensor)
```

```
Empty Tensor:
tensor([[-9.0207e-16,  1.9884e-42,  0.0000e+00],
        [ 0.0000e+00,  0.0000e+00,  0.0000e+00],
        [ 0.0000e+00,  0.0000e+00,  0.0000e+00]])
```

[17]:
```
# Zeros
zeros_tensor = torch.zeros(3, 3)
print("\nZeros Tensor:")
print(zeros_tensor)
```

```
Zeros Tensor:
tensor([[0., 0., 0.],
        [0., 0., 0.],
        [0., 0., 0.]])
```

[19]:
```
# Ones
ones_tensor = torch.ones(3, 3)
print("\nOnes Tensor:")
print(ones_tensor)
```

```
Ones Tensor:
tensor([[1., 1., 1.],
        [1., 1., 1.],
        [1., 1., 1.]])
```

[21]:
```
# Random
random_tensor = torch.rand(3, 3)
print("\nRandom Tensor:")
print(random_tensor)
```

```
Random Tensor:
tensor([[0.3904, 0.6009, 0.2566],
        [0.7936, 0.9408, 0.1332],
        [0.9346, 0.5936, 0.8694]])
```

[23]:
```
# reshape với view
original_tensor = torch.arange(9)
reshaped_tensor_view = original_tensor.view(3, 3)
print("\nReshaped Tensor using view:")
print(reshaped_tensor_view)
```

```
Reshaped Tensor using view:
tensor([[0, 1, 2],
        [3, 4, 5],
        [6, 7, 8]])
```

[25]:
```
# reshape với view_as
example_tensor = torch.zeros_like(original_tensor)
reshaped_tensor_view_as = example_tensor.view_as(reshaped_tensor_view)
print("\nReshaped Tensor using view_as:")
print(reshaped_tensor_view_as)
```

```
Reshaped Tensor using view_as:
tensor([[0, 0, 0],
        [0, 0, 0],
        [0, 0, 0]])
```

[ ]:
```

```