lab5_pytorch_introduction

Harito ID

2025-10-30

Lab 5 (Nhập môn): Làm quen với PyTorch

1. Mục tiêu

Bài thực hành này là bước đệm để bạn làm quen với PyTorch, một trong những thư viện Deep Learning mạnh mẽ và phổ biến nhất. Trước khi xây dựng các mô hình phức tạp như RNN, chúng ta cần nắm vững các khái niệm nền tảng.

Sau bài lab này, bạn sẽ có thể: - Hiểu và thao tác với đối tượng quan trọng nhất trong PyTorch: **Tensor**. - Hiểu cách PyTorch tự động tính toán đạo hàm (gradient) thông qua `autograd`. - Biết cách xây dựng một mạng nơ-ron đơn giản bằng cách kế thừa lớp `torch.nn.Module`. - Làm quen với hai lớp (layer) cơ bản: `nn.Linear` và `nn.Embedding`.

2. Các bước thực hiện

Hãy tạo một file Jupyter Notebook hoặc một file Python để thực hiện các task dưới đây.

Phần 1: Khám phá Tensor

Tensor là cấu trúc dữ liệu cốt lõi của PyTorch, tương tự như `ndarray` của NumPy nhưng có thêm khả năng chạy trên GPU và tự động tính đạo hàm.

Task 1.1: Tạo Tensor

```
import torch
import numpy as np

# Tạo tensor từ list
data = [[1, 2], [3, 4]]
x_data = torch.tensor(data)
print(f"Tensor từ list:\n {x_data}\n")

# Tạo tensor từ NumPy array
np_array = np.array(data)
x_np = torch.from_numpy(np_array)
print(f"Tensor từ NumPy array:\n {x_np}\n")

# Tạo tensor với các giá trị ngẫu nhiên hoặc hằng số
x_ones = torch.ones_like(x_data) # tạo tensor gồm các số 1 có cùng shape với x_data
print(f"Ones Tensor:\n {x_ones}\n")
```

```
x_rand = torch.rand_like(x_data, dtype=torch.float) # tạo tensor ngẫu nhiên
print(f"Random Tensor:\n {x_rand}\n")

# In ra shape, dtype, và device của tensor
print(f"Shape của tensor: {x_rand.shape}")
print(f"Datatype của tensor: {x_rand.dtype}")
print(f"Device lưu trữ tensor: {x_rand.device}")
```

Task 1.2: Các phép toán trên Tensor

Thực hiện các phép toán sau và in kết quả: 1. Cộng `x_data` với chính nó. 2. Nhân `x_data` với 5. 3. Nhân ma trận `x_data` với `x_data.T` (ma trận chuyển vị của nó). Sử dụng toán tử `@`.

Task 1.3: Indexing và Slicing

Từ tensor `x_data`, hãy: 1. Lấy ra hàng đầu tiên. 2. Lấy ra cột thứ hai. 3. Lấy ra giá trị ở hàng thứ hai, cột thứ hai.

Task 1.4: Thay đổi hình dạng Tensor

Sử dụng `torch.rand` để tạo một tensor có shape `(4, 4)`. Sau đó, sử dụng hàm `view` hoặc `reshape` để biến nó thành một tensor có shape `(16, 1)`.

Phần 2: Tự động tính Đạo hàm với `autograd`

Đây là tính năng "ma thuật" của PyTorch. Khi bạn thực hiện các phép toán trên các tensor có `requires_grad=True`, PyTorch sẽ xây dựng một biểu đồ tính toán và tự động tính đạo hàm cho bạn.

Task 2.1: Thực hành với `autograd`

```
# Tạo một tensor và yêu cầu tính đạo hàm cho nó
x = torch.ones(1, requires_grad=True)
print(f"x: {x}")

# Thực hiện một phép toán
y = x + 2
print(f"y: {y}")

# y được tạo ra từ một phép toán có x, nên nó cũng có grad_fn
print(f"grad_fn của y: {y.grad_fn}")

# Thực hiện thêm các phép toán
z = y * y * 3

# Tính đạo hàm của z theo x
z.backward() # tương đương z.backward(torch.tensor(1.))

# Đạo hàm được lưu trong thuộc tính .grad
# Ta có z = 3 * (x+2)^2 => dz/dx = 6 * (x+2). Với x=1, dz/dx = 18
print(f"Đạo hàm của z theo x: {x.grad}")
```

Câu hỏi: Chuyện gì xảy ra nếu bạn gọi `z.backward()` một lần nữa? Tại sao?

Phần 3: Xây dựng Mô hình đầu tiên với `torch.nn`

`torch.nn` là module cung cấp các lớp và công cụ để xây dựng mạng nơ-ron.

Task 3.1: Lớp `nn.Linear`

Lớp `nn.Linear` thực hiện một phép biến đổi tuyến tính `y = xA^T + b`.

```
# Khởi tạo một lớp Linear biến đổi từ 5 chiều -> 2 chiều
linear_layer = torch.nn.Linear(in_features=5, out_features=2)

# Tạo một tensor đầu vào mẫu
input_tensor = torch.randn(3, 5) # 3 mẫu, mỗi mẫu 5 chiều

# Truyền đầu vào qua lớp linear
output = linear_layer(input_tensor)

print(f"Input shape: {input_tensor.shape}")
print(f"Output shape: {output.shape}")
print(f"Output:\n {output}")
```

Task 3.2: Lớp `nn.Embedding`

Lớp `nn.Embedding` là một bảng tra cứu, dùng để ánh xạ các chỉ số của từ thành các vector embedding.

```
# Khởi tạo lớp Embedding cho một từ điển 10 từ, mỗi từ biểu diễn bằng vector 3 chiều
embedding_layer = torch.nn.Embedding(num_embeddings=10, embedding_dim=3)

# Tạo một tensor đầu vào chứa các chỉ số của từ (ví dụ: một câu)
# Các chỉ số phải nhỏ hơn 10
input_indices = torch.LongTensor([1, 5, 0, 8])

# Lấy ra các vector embedding tương ứng
embeddings = embedding_layer(input_indices)

print(f"Input shape: {input_indices.shape}")
print(f"Output shape: {embeddings.shape}")
print(f"Embeddings:\n {embeddings}")
```

Task 3.3: Kết hợp thành một `nn.Module`

Đây là cách chuẩn để định nghĩa một mô hình trong PyTorch.

```
from torch import nn

class MyFirstModel(nn.Module):
    def __init__(self, vocab_size, embedding_dim, hidden_dim, output_dim):
        super(MyFirstModel, self).__init__()
        # Định nghĩa các lớp (layer) bạn sẽ dùng
        self.embedding = nn.Embedding(vocab_size, embedding_dim)
        self.linear = nn.Linear(embedding_dim, hidden_dim)
        self.activation = nn.ReLU() # Hàm kích hoạt
        self.output_layer = nn.Linear(hidden_dim, output_dim)

    def forward(self, indices):
        # Định nghĩa luồng dữ liệu đi qua các lớp
```

```
        # 1. Lấy embedding
        embeds = self.embedding(indices)
        # 2. Truyền qua lớp linear và hàm kích hoạt
        hidden = self.activation(self.linear(embeds))
        # 3. Truyền qua lớp output
        output = self.output_layer(hidden)
        return output

# Khởi tạo và kiểm tra mô hình
model = MyFirstModel(vocab_size=100, embedding_dim=16, hidden_dim=8, output_dim=2)
input_data = torch.LongTensor([[1, 2, 5, 9]]) # một câu gồm 4 từ

output_data = model(input_data)
print(f"Model output shape: {output_data.shape}")
```

3. Kết luận

Qua bài lab này, bạn đã làm quen với các thành phần cơ bản nhất của PyTorch. Bạn đã biết cách tạo và thao tác với Tensor, hiểu được cơ chế tự động tính đạo hàm, và quan trọng nhất là đã tự tay xây dựng một mô-rạng mạng nơ-ron đơn giản bằng `nn.Module`.

Những kỹ năng này là nền tảng vững chắc để bạn tiếp tục xây dựng các mô hình phức tạp hơn như RNN trong bài thực hành tiếp theo.