Lab 5.2: Xây dựng mô hình ANN nhận dạng chữ viết tay

ThS. Lê Thị Thùy Trang

2025-2-19

1. Giới thiệu về bài thực hành

Mạng nơ-ron là nền móng phát triển các mô hình AI hiện đại. Mạng nơ-ron (Neural Networks) mô phỏng cách hoạt động của não bộ con người, học từ dữ liệu thông qua các kết nối giữa các `nơ-ron` nhân tạo. Các mô hình ngôn ngữ lớn như GPT, BERt, T5 đều trên kiến trúc mạng nơ-ron, đặc biệt là mạng nơ-ron sâu (Deep Neural Networks) và Transformer. Mạng nơ-ron được biết đến là một công cụ mạnh mẽ cho nhiều tác vụ, từ nhận dạng hình ảnh đến xử lý ngôn ngữ tự nhiên.

Figure 1: Bộ dữ liệu MNIST

Trong bài thực hành này, sinh viên làm quen với dữ liệu dạng hình ảnh của tập dữ liệu MNIST - tập dữ liệu kinh

điển trong lĩnh vực học máy. Tập dữ liệu này bao gồm 60000 mẫu huấn luyện và 10000 mẫu thử nghiệm là các hình ảnh đa mức xám 28×28 pixel của các chữ số viết tay $(0 - 9)$ cùng với các nhãn tương ứng của chúng.

Mục tiêu của bài thực hành này là xây dựng một mạng nơ-ron có thể phân loại chính xác các chữ số này dựa trên giá trị pixel.

2. Chuẩn bị môi trường thực hành

Để thực hiện bài thực hành này, sinh viên cần cài đặt thư viện `pytorch`.

Sinh viên có thể **lựa chọn** thực hành trên `Google colab` hoặc cài đặt thư viện trên máy tính. Khuyến khích sử dụng `Google colab`.

2.1 Cài đặt thư viện trên máy tính

2.1.1 Cài đặt trên môi trường ảo.

```
python -m venv pytorch_env
```

Để làm việc trên môi trường ảo, dùng lệnh sau:

```
pytorch_env\Scripts\activate
```

Thoát môi trường

```
deactivate
```

2.1.2 Cài đặt `PyTorch` trên Windows

Cài bản CPU

```
pip install torch torchvision torchaudio
```

Nếu có GPU NVIDIA + CUDA, cài bản hỗ trợ GPU

```
pip install torch torchvision torchaudio --index-url https://download.pytorch.org/whl/cu121
```

Kiểm tra cài đặt bằng cách sau

```
import torch
print(torch.__version__)
print("CUDA is available:", torch.cuda.is_available())
```

2.1.3 Sử dụng Google Colab

Google Colab là lựa chọn tuyệt vời khi không có GPU cục bộ hoặc muốn sử dụng môi trường cấu hình sẵn.

Truy cập Google Colab

Đăng nhập bằng tài khoản Google

Tạo notebook mới: Chọn File > New notebook

Kiểm tra cài đặt Pytorch trên Google colab

```
import torch
print(torch.__version__)
print("CUDA available:", torch.cuda.is_available())
```

Google colab thường cài đặt sẵn Pytorch. Nếu muốn cài đặt bản mới, sử dụng lệnh:

```
!pip install torch torchvision torchaudio --index-url https://download.pytorch.org/whl/cu121
```

Kích hoạt GPU trên Google colab:

- Runtime > Change runtime type.
- Ở mục Hardware accelerator, chọn GPU.
- Xác nhận với code:

```
if torch.cuda.is_available():
    print(f"Đang sử dụng GPU: {torch.cuda.get_device_name(0)}")
else:
    print("GPU không khả dụng.")
```

3. Triển khai chương trình

Để bắt đầu, chúng ta cùng nhau xây dựng một mạng neuron có một lớp đầu vào, một lớp đầu ra và một lớp ẩn. Một ANN chỉ có một lớp ẩn được gọi là Mạng nơ-ron nông, ngược lại một ANN có hai hoặc nhiều lớp ẩn được gọi là mạng nơ-ron sâu.

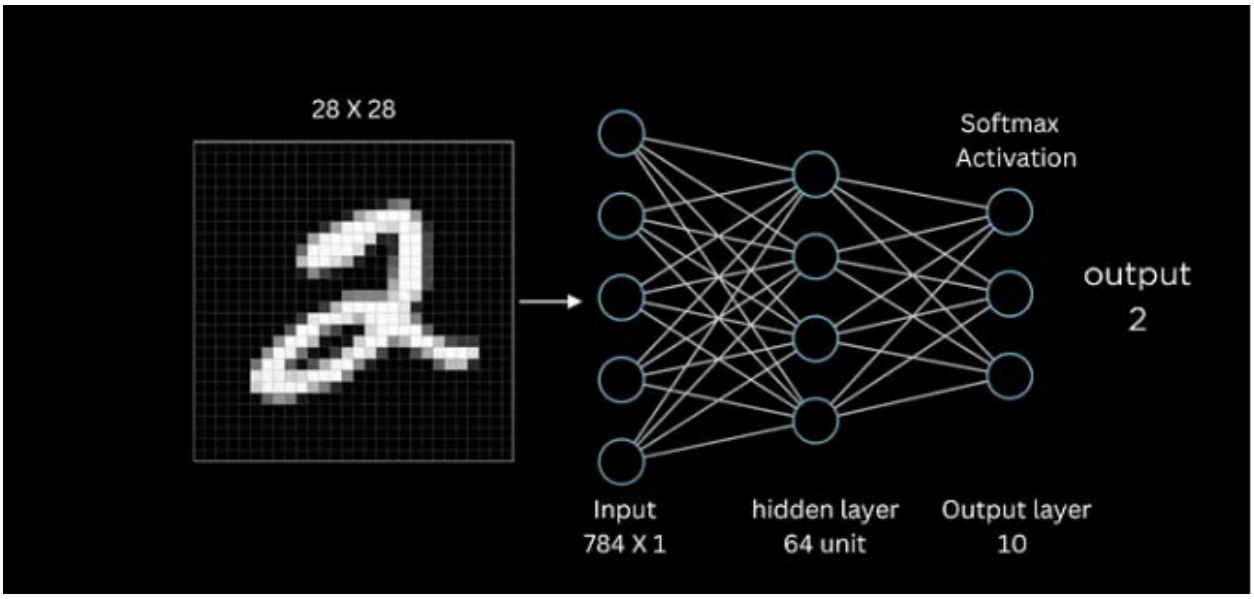

Figure 2: Mô hình mạng nơ-ron một lớp ẩn nhận dạng chữ viết tay

Mạng nơ-ron này gồm các thành phần:

- Lớp đầu vào có 784 đơn vị tương ứng với 784 pixel trong mỗi hình ảnh đầu vào có kích thước 28×28.
- Lớp ẩn có 128 đơn vị, mỗi nơ-ron trong lớp ẩn kết nối với tất cả các nơ-ron ở lớp đầu vào (fully connected)

- Lớp đầu ra gồm 10 neuron, tương ứng với 10 chữ số (0-9) trong MNIST. Sử dụng hàm kích hoạt `Softmax`, giúp xác suất đầu ra của mỗi lớp nằm trong khoảng $[0, 1]$, sao cho tổng xác suất của 10 lớp bằng 1.

Mạng này thuộc loại mạng nơ-ron truyền thẳng (Feedforward Neural Network - FNN) với một lớp ẩn duy nhất. Ngĩa là trong mạng này, thông tin chỉ truyền theo một hướng, từ `lớp đầu vào` $\rightarrow$ `lớp ẩn` $\rightarrow$ `lớp đầu ra`, mà không có vòng lặp hay phản hồi.

Để cài đặt mô hình ANN một lớp ẩn nhận diện chữ viết tay trong bộ dữ liệu MNIST, thực hiện các bước sau:

3.1 Import thư viện

```
import torch
import torch.nn as nn
import torch.optim as optim
import torchvision
from torchvision import datasets, transforms
```

3.2 Chuẩn bị dữ liệu

MNIST đã được tích hợp sẵn trong thư viện `torchvision` của `PyTorch`, nên ta có thể tải trực tiếp mà không cần tải thủ công từ nguồn bên ngoài.

Trong `PyTorch`, tensor là định dạng dữ liệu chính để huấn luyện mô hình. Khi làm việc với bộ dữ liệu MNIST (gồm ảnh số viết tay kích thước 28×28 pixel), ta cần chuyển đổi ảnh sang tensor. Tensor là một cấu trúc dữ liệu toán học, giống như một mảng (array) hoặc ma trận (matrix), nhưng có thể mở rộng lên nhiều chiều hơn. Trong `PyTorch`, tensor là định dạng dữ liệu chính để lưu trữ và xử lý dữ liệu. Nó tương tự như `numpy.array`, nhưng có khả năng chạy trên GPU, giúp tính toán nhanh hơn.

Tiếp theo, ta sử dụng `DataLoader` - một công cụ quan trọng trong PyTorch, giúp tải dữ liệu theo batch để tăng tốc quá trình huấn luyện mô hình. Nó cung cấp các tính năng như:

- Chia dữ liệu thành các batch nhỏ thay vì đưa toàn bộ vào cùng một lúc.
- Xáo trộn dữ liệu (shuffle) để giúp mô hình tổng quát tốt hơn.
- Tăng tốc độ huấn luyện bằng cách tải dữ liệu song song (num_workers).

```
# Convert images to tensors without normalization
transform = transforms.ToTensor()

# Load MNIST dataset
trainset = torchvision.datasets.MNIST(root='./data', train=True,
download=True, transform=transform)
testset = torchvision.datasets.MNIST(root='./data', train=False,
download=True, transform=transform)

trainloader = torch.utils.data.DataLoader(trainset, batch_size=32, shuffle=True)
testloader = torch.utils.data.DataLoader(testset, batch_size=32, shuffle=False)
```

3.3 Xây dựng mô hình ANN

Bước đầu, ta khởi tạo một mô hình mạng neuron đơn giản với 1 lớp ẩn chứa 64 đơn vị.

Sau đó, xây dựng phương thức `forward` để thực hiện truyền dữ liệu qua các lớp và tính toán kết quả đầu ra.

```
class ANN(nn.Module):
    def __init__(self):
        super(ANN, self).__init__()
        self.fc1 = nn.Linear(28*28, 128)  # Hidden layer with 128 neurons
        self.relu = nn.ReLU()  # Activation function
        self.fc2 = nn.Linear(128, 10)  # Output layer with 10 classes
        self.softmax = nn.Softmax(dim=1)  # Use Softmax activation function for classification

    def forward(self, x):
        x = x.view(-1, 28*28)  # Flatten 28x28 images into 784-dimensional vectors
        x = self.fc1(x)  # Apply first linear transformation
        x = self.relu(x)  # Apply ReLU activation function
        x = self.fc2(x)  # Apply second linear transformation
        x = self.softmax(x)  # Apply Softmax activation
        return x

model = ANN()
```

3.4 Định nghĩa `Loss Function`

```
criterion = nn.CrossEntropyLoss()  # Cross-entropy loss for classification
```

3.5 Huấn luyện mô hình

Mô hình ANN được huấn luyện theo các bước sau:

- Lặp qua từng epoch: Mô hình sẽ học qua nhiều epoch để cải thiện hiệu suất.
- Duyệt qua từng batch dữ liệu: Mỗi batch chứa một số lượng ảnh và nhãn nhất định.
- Truyền dữ liệu qua mô hình: Dự đoán đầu ra bằng cách gọi model(images).
- Tính toán lỗi (loss): So sánh đầu ra dự đoán với nhãn thực tế bằng criterion(outputs, labels).
- Lan truyền ngược (Backpropagation): Cập nhật trọng số bằng cách tính gradient của lỗi đối với từng tham số.
- Cập nhật trọng số thủ công: Cập nhật từng tham số bằng cách trừ đi gradient nhân với learning_rate.

```
epochs = 10
learning_rate = 0.001

for epoch in range(epochs):
    running_loss = 0.0
    for images, labels in trainloader:
        outputs = model(images)  # Forward pass
        loss = criterion(outputs, labels)  # Compute loss

        # Manually update weights
        model.zero_grad()
        loss.backward()  # Backpropagation

        with torch.no_grad():  # Disable gradient tracking for manual weight update
            for param in model.parameters():
                param -= learning_rate * param.grad
```

```
        running_loss += loss.item()
    print(f"Epoch {epoch+1}, Loss: {running_loss/len(trainloader):.4f}")
```

3.6 Đánh giá mô hình

```
correct = 0
total = 0
with torch.no_grad():  # Disable gradient computation during evaluation
    for images, labels in testloader:
        outputs = model(images)
        _, predicted = torch.max(outputs, 1)  # Get class with highest probability
        total += labels.size(0)
        correct += (predicted == labels).sum().item()

print(f"Accuracy on test set: {100 * correct / total:.2f}%")
```

4. Thực hành

Sinh viên thực thi chương trình, đọc hiểu và phân tích code mẫu. Sau đó thực hiện các câu hỏi sau:

Q1: Hiện tại, dữ liệu chỉ được chuyển thành tensor mà không chuẩn hóa. Sinh viên nghiên cứu tài liệu về chuẩn hóa dữ liệu, chọn một phương pháp chuẩn hoá dữ liệu. Sửa đổi code và so sánh hiệu suất của mô hình khi không chuẩn hoá và có chuẩn hoá dữ liệu.

Q2: Mô hình mạng ANN đang được xây dựng chỉ có một lớp ẩn. Sinh viên thực hiện thay đổi kiến trúc mạng neuron:

- Thiết kế một mạng nơ-ron có nhiều lớp ẩn hơn, tăng số lượng đơn vị ở mỗi lớp.
- Thử nghiệm mô hình với các hàm kích hoạt khác nhau (ví dụ: LeakyReLU, Tanh).

Sau đó nhận xét hiệu suất của mô hình sau khi thay đổi.