Mạng Neural Nhân Tạo (ANN), hay còn gọi là Artificial Neural Network, là một trong những mô hình máy học cơ bản và phổ biến nhất trong lĩnh vực trí tuệ nhân tạo. Mạng neural nhân tạo lấy cảm hứng từ cấu trúc và hoạt động của hệ thống não người để giải quyết các vấn đề như phân loại hình ảnh, nhận dạng văn bản, dự đoán chuỗi thời gian, và nhiều ứng dụng khác.

**Cấu Trúc Cơ Bản của ANN:**

Mạng Neural Nhân Tạo thường bao gồm các thành phần sau:

* **Tầng Đầu Vào (Input Layer)**: Tầng này nhận dữ liệu đầu vào từ bên ngoài và truyền chúng đến các tầng ẩn.
* **Tầng Ẩn (Hidden Layers)**: Đây là các tầng nằm giữa tầng đầu vào và tầng đầu ra. Mỗi tầng ẩn bao gồm một số lượng units (neurons) có thể thay đổi. Các tầng ẩn chứa các trọng số và hàm kích hoạt để biến đổi dữ liệu đầu vào.
* **Tầng Đầu Ra (Output Layer)**: Tầng này chịu trách nhiệm sản xuất kết quả đầu ra của mô hình dự báo. Số lượng units trong tầng này thường tương ứng với số lượng lớp phân loại hoặc số lượng giá trị đầu ra cần dự đoán.

**Hoạt Động Của ANN:**

* **Forward Propagation**: Quá trình truyền tiến (forward propagation) bắt đầu từ tầng đầu vào và lan truyền qua các tầng ẩn cho đến tầng đầu ra. Trong quá trình này, dữ liệu được biến đổi thông qua các trọng số và hàm kích hoạt.
* **Backward Propagation**: Sau khi thu được đầu ra dự đoán từ mô hình, chúng ta sử dụng hàm mất mát để tính toán sai số giữa kết quả dự đoán và kết quả thực tế. Sau đó, chúng ta sử dụng thuật toán lan truyền ngược (backpropagation) để cập nhật các trọng số của mạng sao cho sai số giảm đáng kể.
* **Huấn Luyện Mô Hình**: Quá trình lan truyền tiến và lan truyền ngược được lặp lại nhiều lần thông qua các vòng lặp huấn luyện để cải thiện hiệu suất của mô hình.

**Ứng Dụng của ANN:**

Mạng Neural Nhân Tạo đã được áp dụng rộng rãi trong nhiều lĩnh vực và có các ứng dụng quan trọng như:

* Nhận dạng hình ảnh và video.
* Phân loại văn bản và ngôn ngữ tự nhiên.
* Dự đoán chuỗi thời gian (ví dụ: dự báo giá cổ phiếu).
* Tự động lái xe và xe tự lái.
* Dịch máy và xử lý ngôn ngữ tự nhiên.
* Tóm lại, Mạng Neural Nhân Tạo là một công cụ mạnh mẽ và linh hoạt có thể áp dụng trong nhiều lĩnh vực khác nhau để giải quyết các vấn đề phức tạp của thế giới thực.

**Trong phần source code có 2 file, trong đó file ANN.py để huấn luyện mô hình và lưu lại kết quả huấn luyện, còn file Form.py để hiển thị form để vẻ chữ số viết tay :**

**Giải thích phần code ANN.py**

**Import thư viện và module cần thiết:**

import torch

import torch.nn as nn

import torch.optim as optim

import torchvision

import torchvision.transforms as transforms

Trong đó : torch, torch.nn, torch.optim là thư viện PyTorch và các module liên quan để tạo và huấn luyện mạng nơ-ron.

torchvision: Dùng để làm việc với dữ liệu hình ảnh và tải các tập dữ liệu chuẩn.

torchvision.transforms: Cung cấp các chức năng để chuyển đổi dữ liệu hình ảnh.

**Định nghĩa mô hình ANN:**

class ANN(nn.Module):

def \_\_init\_\_(self):

super(ANN, self).\_\_init\_\_()

self.fc1 = nn.Linear(28 \* 28, 256)

self.fc2 = nn.Linear(256, 128)

self.fc3 = nn.Linear(128, 64)

self.fc4 = nn.Linear(64, 10)

def forward(self, x):

x = x.view(-1, 28 \* 28) # Phẳng hình ảnh thành một vector

x = torch.relu(self.fc1(x))

x = torch.relu(self.fc2(x))

x = torch.relu(self.fc3(x))

x = self.fc4(x)

return x

Đây là một mạng nơ-ron truyền thẳng (feedforward neural network) đơn giản với bốn tầng fully connected (hoàn toàn kết nối).

Hàm \_\_init\_\_ khởi tạo các lớp tuyến tính (linear layers) và hàm forward xác định cách dữ liệu chuyển qua mạng.

**Kiểm tra và cấu hình thiết bị sử dụng:**

device = torch.device('cuda' if torch.cuda.is\_available() else 'cpu')

Code này kiểm tra xem GPU có sẵn hay không. Nếu có, mô hình sẽ được đào tạo trên GPU, nếu không, mô hình sẽ được đào tạo trên CPU.

**Tải và chuẩn hóa dữ liệu MNIST:**

transform = transforms.Compose([

transforms.ToTensor(),

transforms.Normalize((0.5,), (0.5,))

])

trainset = torchvision.datasets.MNIST(root='./data', train=True, download=True, transform=transform)

trainloader = torch.utils.data.DataLoader(trainset, batch\_size=64, shuffle=True)

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

testloader = torch.utils.data.DataLoader(testset, batch\_size=64, shuffle=False)

Dữ liệu MNIST là tập dữ liệu với các hình ảnh chữ số viết tay từ 0 đến 9.

transform được sử dụng để chuyển đổi hình ảnh sang tensor và chuẩn hóa dữ liệu.

Dữ liệu được chia thành tập huấn luyện và tập kiểm tra, mỗi tập được tạo thành một DataLoader để dễ dàng truy cập vào các batch dữ liệu.

**Khởi tạo mô hình, hàm mất mát và bộ tối ưu:**

model = ANN().to(device)

criterion = nn.CrossEntropyLoss()

optimizer = optim.Adam(model.parameters(), lr=0.001)

Mô hình được khởi tạo dựa trên lớp ANN, di chuyển vào thiết bị được xác định trước đó.

CrossEntropyLoss() được sử dụng làm hàm mất mát, và Adam là thuật toán tối ưu hóa được sử dụng với **learning rate** là 0.001. Cần phải thay đổi tỉ lệ học **learning rate** để kiểm tra khả năng của mô hình như thế nào này là phần quan trọng.

**Hàm huấn luyện mô hình:**

def train\_model(model, trainloader, criterion, optimizer, epochs):

for epoch in range(epochs):

running\_loss = 0.0

for inputs, labels in trainloader:

inputs, labels = inputs.to(device), labels.to(device)

optimizer.zero\_grad()

outputs = model(inputs)

loss = criterion(outputs, labels)

loss.backward()

optimizer.step()

running\_loss += loss.item()

print(f"Huấn luyện {epoch+1}/{epochs}, Thất bại: {running\_loss/len(trainloader):.4f}")

print('Hoàn tất huấn luyện')

Hàm này huấn luyện mô hình trên các epoch bằng cách lặp qua các batch từ trainloader.

Trong mỗi epoch, forward pass được thực hiện, sau đó tính toán loss, backward pass và cập nhật trọng số mạng nơ-ron.

**Lưu mô hình đã huấn luyện:**

torch.save(model.state\_dict(), 'ann\_model.pth')

Mô hình được lưu vào một tệp có tên ann\_model.pth, chỉ lưu trạng thái của các tham số mô hình, không lưu trữ cấu trúc mạng.

**Đánh giá mô hình trên tập kiểm tra:**

def evaluate\_model(model, testloader):

correct = 0

total = 0

model.eval()

with torch.no\_grad():

for inputs, labels in testloader:

inputs, labels = inputs.to(device), labels.to(device)

outputs = model(inputs)

\_, predicted = torch.max(outputs.data, 1)

total += labels.size(0)

correct += (predicted == labels).sum().item()

accuracy = 100 \* correct / total

print(f'Độ chính xác của mô hình trên 10000 hình ảnh thử nghiệm: {accuracy:.2f}%')

**Giải thích phần code Form.py**

Đoạn code trên tạo một ứng dụng đơn giản để nhận dạng chữ số viết tay bằng cách sử dụng một mô hình mạng nơ-ron đã được huấn luyện trước. Dưới đây là giải thích từng phần của đoạn mã:

**Import các thư viện và module cần thiết**:

import tkinter as tk

from tkinter import \*

import torch

import torch.nn as nn

import torch.optim as optim

import torchvision

import torchvision.transforms as transforms

from PIL import Image, ImageDraw

import numpy as np

tkinter: Được sử dụng để tạo giao diện người dùng.

torch, torch.nn, torch.optim, torchvision, transforms: Các thư viện PyTorch cần thiết để tải và sử dụng mô hình đã huấn luyện.

PIL: Được sử dụng để làm việc với hình ảnh.

**Định nghĩa mô hình ANN**:

class ANN(nn.Module):

...

Mô hình được định nghĩa giống như trong phần trước.

**Tải mô hình đã huấn luyện**:

model = ANN()

model.load\_state\_dict(torch.load('ann\_model.pth'))

model.eval()

Mô hình đã được huấn luyện trước bên phần ANN.py được tải lên từ tệp ann\_model.pth.

**Cấu hình thiết bị:**

device = torch.device('cuda' if torch.cuda.is\_available() else 'cpu')

model.to(device)

Kiểm tra xem GPU có sẵn không và di chuyển mô hình tới thiết bị phù hợp.

**Hàm dự đoán chữ số** (predict\_digit):

def predict\_digit(img):

...

Hàm này nhận một hình ảnh và trả về các dự đoán chữ số và xác suất tương ứng.

**Tạo giao diện vẽ**:

class App(tk.Tk):

...

Một ứng dụng GUI được tạo bằng cách kế thừa từ tk.Tk.

Giao diện bao gồm một canvas để vẽ và các nút để dự đoán và xóa.

**Phương thức vẽ (paint):**

def paint(self, event):

...

Khi người dùng di chuyển chuột trên canvas, các đường tròn màu đen sẽ được tạo ra và vẽ lên hình ảnh.

**Phương thức dự đoán** (classify\_handwriting):

def classify\_handwriting(self):

...

Khi nút "Dự đoán" được nhấn, hình ảnh trên canvas sẽ được đưa vào hàm predict\_digit để dự đoán chữ số và hiển thị kết quả.

**Phương thức làm mới** (clear\_canvas):

def clear\_canvas(self):

...

Khi nút "Làm mới" được nhấn, canvas sẽ được xóa sạch và chuẩn bị cho việc vẽ mới.

**Chạy ứng dụng**:

app = App()

app.mainloop()

Ứng dụng được khởi chạy bằng cách tạo một đối tượng từ lớp App và gọi phương thức mainloop() để duy trì vòng lặp chính của ứng dụng GUI.