11/8/25

BUILD A SIMPLE FEED - FORWARD NEURAL NETWORK TO RECOGNISE HANDWRITTEN CHARACTER

EXP - 4

Aim :- To design and implement a simple feed - forward neural network using python to recognize handwritten characters from a dataset.

Description :-

A feed forward network (FNN) is a type of artificial neural network where connections between nodes do not form a cycle. In this experiment, the FNN will be trained on a dataset of handwritten characters to classify them into respective categories. The model consist of an input layer, one or more hidden layers with activation functions, and an output layer using softmax for classification.

Precision & Recall -

$$\text{Precision} = \frac{\text{Correctly predicted positive observations}}{\text{Total predicted positive observations}}$$

$$\text{Recall} = \frac{\text{Correctly predicted positive observations}}{\text{All actual positive observations}}$$

These metrics evaluate classification performance beyond accuracy, especially when class distribution is imbalanced

Confusion Matrix:

Actual / Predicted	0	1	2	3
0	964	0	0	3
1	0	1122	4	4
2	8	2	984	9
3	0	0	8	981

Procedure:-

1) Load the handwritten character dataset (MNIST)
2) Normalize pixel values between 0 & 1
3) Flatten the images into 1D arrays.
4) Create a feed-forward network with:
   - Input layer
   - Hidden layer (ReLU)
   - Output layer (Softmax)
5) Compile the model with Adam optimizer & categorical cross-entropy loss.
6) Train the model on the training set.
7) Test the model & display the accuracy

Result:- The feed-forward neural network recognized handwritten digits with an accuracy of about 97%. This model which is effective is working successfully

22/6

You may also turn on the notedown plugin by default whenever you run the Jupyter Notebook. First, generate a Jupyter Notebook configuration file (if it has already been generated, you can skip this step).

```
jupyter notebook --generate-config
```

Then, add the following line to the end of the Jupyter Notebook configuration file (for Linux or macOS, usually in the path `~/.jupyter/jupyter_notebook_config.py`):

```
c.NotebookApp.contents_manager_class = 'notedown.NotedownContentsManager'
```

After that, you only need to run the `jupyter notebook` command to turn on the notedown plugin by default.

Running Jupyter Notebooks on a Remote Server

Sometimes, you may want to run Jupyter notebooks on a remote server and access it through a browser on your local computer. If Linux or macOS is installed on your local machine (Windows can also support this function through third-party software such as PuTTY), you can use port forwarding:

```
ssh myserver -L 8888:localhost:8888
```

The above string `myserver` is the address of the remote server. Then we can use http://localhost:8888 to access the remote server `myserver` that runs Jupyter notebooks. We will detail on how to run Jupyter notebooks on AWS instances later in this appendix.

Timing

We can use the `ExecuteTime` plugin to time the execution of each code cell in Jupyter notebooks. Use the following commands to install the plugin:

```
pip install jupyter_contrib_nbextensions
jupyter contrib nbextension install --user
jupyter nbextension enable execute_time/ExecuteTime
```

Summary

- Using the Jupyter Notebook tool, we can edit, run, and contribute to each section of the book.
- We can run Jupyter notebooks on remote servers using port forwarding.

Exercises

1. Edit and run the code in this book with the Jupyter Notebook on your local machine.
2. Edit and run the code in this book with the Jupyter Notebook *remotely* via port forwarding.
3. Compare the running time of the operations $\mathbf{A}^\top\mathbf{B}$ and $\mathbf{A}\mathbf{B}$ for two square matrices in $\mathbb{R}^{1024\times 1024}$. Which one is faster?

Discussions

```
import torch
import torch.nn as nn
import torch.optim as optim
from torchvision import datasets, transforms
from torch.utils.data import DataLoader
transform = transforms.Compose([transforms.ToTensor(), transforms.Normalize((0.5,), (0.5,))])
train_dataset = datasets.MNIST('.', train=True, download=True, transform=transform)
test_dataset = datasets.MNIST('.', train=False, download=True, transform=transform)
train_loader = DataLoader(train_dataset, batch_size=32, shuffle=True)
test_loader = DataLoader(test_dataset, batch_size=32, shuffle=False)
class SimpleNN(nn.Module):
    def __init__(self):
        super(SimpleNN, self).__init__()
        self.flatten = nn.Flatten()
        self.fc1 = nn.Linear(28 * 28, 128)
        self.relu = nn.ReLU()
        self.fc2 = nn.Linear(128, 10)
        self.softmax = nn.Softmax(dim=1)
    def forward(self, x):
        x = self.flatten(x)
        x = self.fc1(x)
        x = self.relu(x)
```

```
        x = self.fc2(x)
        x = self.softmax(x)
        return x
model = SimpleNN()
criterion = nn.CrossEntropyLoss()
optimizer = optim.Adam(model.parameters(), lr=0.001)
epochs = 5
for epoch in range(epochs):
    model.train()
    running_loss = 0.0
    for images, labels in train_loader:
        optimizer.zero_grad()
        outputs = model(images)
        loss = criterion(outputs, labels)
        loss.backward()
        optimizer.step()
        running_loss += loss.item()
    print(f'Epoch {epoch + 1}, Loss: {running_loss / len(train_loader)}')
model.eval()
```

```
100%|██████████| 9.91M/9.91M [00:00<00:00, 18.5MB/s]
100%|██████████| 28.9k/28.9k [00:00<00:00, 500kB/s]
100%|██████████| 1.65M/1.65M [00:00<00:00, 4.60MB/s]
100%|██████████| 4.54k/4.54k [00:00<00:00, 7.43MB/s]
Epoch 1, Loss: 1.6724406388600668
Epoch 2, Loss: 1.5657304460525512
Epoch 3, Loss: 1.5247076534907023
Epoch 4, Loss: 1.5150544836680095
Epoch 5, Loss: 1.5089014788309734
SimpleNN(
  (flatten): Flatten(start_dim=1, end_dim=-1)
  (fc1): Linear(in_features=784, out_features=128, bias=True)
  (relu): ReLU()
  (fc2): Linear(in_features=128, out_features=10, bias=True)
  (softmax): Softmax(dim=1)
)
```

```
from sklearn.metrics import confusion_matrix
import numpy as np
model.eval()  # Set model to evaluation mode

all_preds = []
all_labels = []

with torch.no_grad():  # Disable gradient calculation for inference
    for images, labels in test_loader:
        outputs = model(images)               # Forward pass
        _, predicted = torch.max(outputs, 1)  # Get class with highest score
        all_preds.extend(predicted.cpu().numpy())
        all_labels.extend(labels.cpu().numpy())

# Compute confusion matrix
cm = confusion_matrix(all_labels, all_preds)
print("Confusion Matrix:")
print(cm)
```

```
Confusion Matrix:
[[ 964    0    0    3    1    3    3    2    4    0]
 [   0 1122    4    4    0    1    1    1    2    0]
 [   8    2  984    9    9    0    3   11    6    0]
 [   0    0    8  981    0    4    0   10    3    4]
 [   1    3    2    1  948    1    6    4    3   13]
 [   6    1    0   28    2  832    5    6    7    5]
 [   7    4    3    2    7   13  913    2    7    0]
 [   0   10   18    2    1    1    0  989    1    6]
 [   3    7    5   15    4    3    6   12  916    3]
 [   6   11    0   13   28    6    1   16    9  919]]
```