14/8/25

Experiment-4

Build a simple feed-forward neural network to recognize handwritten characters

Aim:

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

Description:

A feed forward neural 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 consists of an input layer, one ore more hidden layers with activation functions, and an output layer using softmax for classification.

Precision & Recall:

- Precision = correctly predicted positive observation / Total predicted positive observations
- Recall = Correctly predicted positive observations / All actual positive observations.

Confusion Matrix:

Actual / Predicted	0	1	2	3
0	95	0	1	0
1	0	93	2	1
2	1		96	0
3	0	2	1	94

Procedure :

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

Result:

The feed forward neural network recognized handwritten digits with an accuracy of about 97%.

Lab_5.ipynb
Lab_6.ipynb
Lab_4.ipynb
Code
Notebook
Python 3 (ipykernel)

[1]:
```
import torch
import torch.nn as nn
import torch.optim as optim
from torchvision import datasets, transforms
from torch.utils.data import DataLoader
import torch.nn.functional as F
```

[2]:
```
input_size = 28*28
hidden_size = 128
num_classes = 10
batch_size = 64
learning_rate = 0.001
num_epochs = 5
```

[3]:
```
# data prep
transform = transforms.Compose([
    transforms.ToTensor(),
    transforms.Normalize((0.1307,), (0.3081,))
])
```

[4]:
```
train_dataset = datasets.MNIST(root='./data', train = True, transform = transform, download=True)
test_dataset = datasets.MNIST(root='./data', train = False, transform = transform, download = True)
train_loader = DataLoader(dataset=train_dataset, batch_size=batch_size, shuffle=True)
test_loader = DataLoader(dataset=test_dataset, batch_size=batch_size, shuffle=False)
```

Lab_5.ipynb
Lab_6.ipynb
Lab_4.ipynb
Code
Notebook
Python 3 (ipykernel)

```python
class FeedforwardNN(nn.Module):
    def __init__(self):
        super(FeedforwardNN, self).__init__()
        self.fc1 = nn.Linear(input_size, hidden_size)
        self.fc2 = nn.Linear(hidden_size, num_classes)

    def forward(self, x):
        x = x.view(-1, input_size)
        x = F.relu(self.fc1(x))
        x = self.fc2(x)
        return x

model = FeedforwardNN()
criterion = nn.CrossEntropyLoss()
optimizer = optim.Adam(model.parameters(), lr=learning_rate)

for epoch in range(num_epochs):
    model.train()
    for images, labels in train_loader:
        outputs = model(images)
        loss = criterion(outputs, labels)

        optimizer.zero_grad()
        loss.backward()
        optimizer.step()
```

Lab_5.ipynb
Lab_6.ipynb
Lab_4.ipynb
Code
Notebook
Python 3 (ipykernel)

```python
for epoch in range(num_epochs):
    model.train()
    for images, labels in train_loader:
        outputs = model(images)
        loss = criterion(outputs, labels)

        optimizer.zero_grad()
        loss.backward()
        optimizer.step()

    print(f"Epoch [{epoch+1}/{num_epochs}], Loss: {loss.item():.4f}")

model.eval()
correct = 0
total = 0
with torch.no_grad():
    for images, labels in test_loader:s
        outputs = model(images)
        probs = F.softmax(outputs, dim=1)
        _, predicted = torch.max(probs, 1)
        total += labels.size(0)
        correct += (predicted == labels).sum().item()

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