Experiment 9

Objective:

To implement multi layer neural network.

Theory

A **Multi-Layer Neural Network (MLNN)**, often referred to as a **feedforward neural network**, is a type of artificial neural network composed of multiple layers of nodes (neurons). It is one of the fundamental models in deep learning and is used for various tasks such as classification, regression, and pattern recognition. The architecture of a multi-layer neural network typically consists of an input layer, one or more hidden layers, and an output layer.

1. Structure of a Multi-Layer Neural Network:

- **Input Layer**: The input layer receives the raw data. Each neuron in this layer represents one feature of the input data.
- **Hidden Layers**: These layers exist between the input and output layers and allow the network to learn complex patterns. Each neuron in a hidden layer performs a weighted sum of inputs and passes the result through an activation function.
- **Output Layer**: The output layer produces the final prediction or classification. The number of neurons in the output layer depends on the type of task—one neuron for binary classification, multiple neurons for multi-class classification, or a single neuron for regression tasks.

2. The Neuron: Each neuron in the network performs a simple mathematical operation. Given an input vector x=[x1,x2,...,xn], the output of a neuron is calculated as:

$$y = f(w_1x_1 + w_2x_2 + ... + w_nx_n + b)$$

where:

- w1,w2,..., wn are the weights,
- bb is the bias term,
- ff is the activation function, which introduces non-linearity into the model.

3. Activation Functions: Activation functions are crucial in neural networks as they allow the network to learn complex, non-linear relationships between the inputs and outputs. Some commonly used activation functions include:

- **Sigmoid**: Maps outputs to a range between 0 and 1. Often used in binary classification problems. $\sigma(x) = \frac{1}{1+e^{-x}}$
- **ReLU (Rectified Linear Unit)**: The most widely used activation function in deep learning, which outputs the input directly if it is positive; otherwise, it outputs zero. $\text{ReLU}(x) = \max(0, x)$
- **Tanh**: Similar to sigmoid but outputs values between -1 and 1, often used when the data is centered around zero. tanh(x)=ex−e−xex+e−xtanh(x)=ex+e−xex−e−x

4. Training a Multi-Layer Neural Network: Training a neural network involves finding the optimal weights and biases that minimize the error between the predicted output and the actual target. This is achieved through the following steps:

- **Forward Propagation**: During forward propagation, the input data passes through the network, and an output is generated at the output layer.
- **Loss Function**: The loss function calculates the difference between the predicted output and the true output. Common loss functions include:
    - **Mean Squared Error (MSE)** for regression tasks.
    - **Cross-Entropy** for classification tasks.
- **Backpropagation**: The backpropagation algorithm is used to update the weights and biases by calculating the gradient of the loss function with respect to each weight in the network. This is achieved through the **chain rule** of differentiation. The gradients are then used to update the weights using an optimization algorithm such as **Stochastic Gradient Descent (SGD)** or **Adam**.

5. Optimization: Optimization algorithms are used to adjust the weights to minimize the loss function. Some common optimization techniques are:

- **Gradient Descent**: Gradually adjusts the weights in the opposite direction of the gradient to minimize the loss function.
- **Adam**: A more advanced optimization method that adapts the learning rate based on estimates of first and second moments of the gradients, providing faster convergence.

6. Overfitting and Regularization: Overfitting occurs when the model learns the training data too well, including noise and outliers, which hurts its generalization to new data. To combat overfitting:

- **Dropout**: Randomly drops neurons during training to prevent over-reliance on specific neurons.
- **L2 Regularization (Ridge)**: Adds a penalty term to the loss function based on the magnitude of the weights, encouraging smaller weight values and reducing overfitting.

7. Multi-Layer Neural Networks in Practice: Multi-layer neural networks are powerful models and are the building blocks of more complex architectures like convolutional neural networks (CNNs) and recurrent neural networks (RNNs). They are widely used in image classification, speech recognition, and natural language processing. Popular deep learning libraries such as **TensorFlow**, **Keras**, and **PyTorch** provide tools to build and train multi-layer neural networks.

Applications:

1. **Image Classification**: MLNNs are used in image classification tasks, such as identifying objects in images or recognizing handwritten digits.
2. **Natural Language Processing**: In NLP, multi-layer neural networks are used for tasks like sentiment analysis, machine translation, and named entity recognition.
3. **Speech Recognition**: Neural networks are utilized in speech-to-text systems to convert spoken language into written text.

Code & OUTPUT

In [1]:
```python
print("Experiment No 09 : To implement multi layer neural network.")
```

```
Experiment No 09 : To implement multi layer neural network.
```

In [7]:
```python
!pip install torchvision --user
```

```
Collecting torchvision
  Obtaining dependency information for torchvision from https://files.pythonhosted.org/packages/69/55/ce836703ff77bb
21582c3098d5311f8ddde7eadc7eab04be9561961f4725/torchvision-0.20.1-cp311-cp311-win_amd64.whl.metadata
  Using cached torchvision-0.20.1-cp311-cp311-win_amd64.whl.metadata (6.2 kB)
Requirement already satisfied: numpy in c:\users\hp\anaconda3\lib\site-packages (from torchvision) (1.24.3)
Requirement already satisfied: torch==2.5.1 in c:\users\hp\anaconda3\lib\site-packages (from torchvision) (2.5.1)
Requirement already satisfied: pillow!=8.3.*,>=5.3.0 in c:\users\hp\anaconda3\lib\site-packages (from torchvision)
(9.4.0)
Requirement already satisfied: filelock in c:\users\hp\anaconda3\lib\site-packages (from torch==2.5.1->torchvision)
(3.9.0)
Requirement already satisfied: typing-extensions>=4.8.0 in c:\users\hp\anaconda3\lib\site-packages (from torch==
2.5.1->torchvision) (4.12.2)
Requirement already satisfied: networkx in c:\users\hp\anaconda3\lib\site-packages (from torch==2.5.1->torchvision)
(3.1)
Requirement already satisfied: jinja2 in c:\users\hp\anaconda3\lib\site-packages (from torch==2.5.1->torchvision)
(3.1.2)
Requirement already satisfied: fsspec in c:\users\hp\anaconda3\lib\site-packages (from torch==2.5.1->torchvision) (2
023.4.0)
Requirement already satisfied: sympy==1.13.1 in c:\users\hp\anaconda3\lib\site-packages (from torch==2.5.1->torchvis
ion) (1.13.1)
Requirement already satisfied: mpmath<1.4,>=1.1.0 in c:\users\hp\anaconda3\lib\site-packages (from sympy==1.13.1->to
rch==2.5.1->torchvision) (1.3.0)
Requirement already satisfied: MarkupSafe>=2.0 in c:\users\hp\anaconda3\lib\site-packages (from jinja2->torch==
2.5.1->torchvision) (2.1.1)
Using cached torchvision-0.20.1-cp311-cp311-win_amd64.whl (1.6 MB)
Installing collected packages: torchvision
Successfully installed torchvision-0.20.1
```

In [8]:

```python
# Import necessary libraries
import torch
import torch.nn as nn
import torch.optim as optim
from torch.utils.data import DataLoader
from torchvision import datasets, transforms

# Define transformations for the dataset
transform = transforms.Compose([
    transforms.ToTensor(),
    transforms.Normalize((0.5,), (0.5,))  # Normalize to mean 0.5 and std 0.5 for simplicity
])

# Load the MNIST dataset
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(train_dataset, batch_size=64, shuffle=True)
test_loader = DataLoader(test_dataset, batch_size=64, shuffle=False)

# Define the neural network model
class NeuralNet(nn.Module):
    def __init__(self):
        super(NeuralNet, self).__init__()
        self.fc1 = nn.Linear(28*28, 128)  # First hidden layer with 128 neurons
        self.fc2 = nn.Linear(128, 64)     # Second hidden layer with 64 neurons
        self.fc3 = nn.Linear(64, 10)      # Output layer with 10 classes

    def forward(self, x):
        x = x.view(-1, 28*28)            # Flatten the input
        x = torch.relu(self.fc1(x))       # Apply ReLU activation
        x = torch.relu(self.fc2(x))       # Apply ReLU activation
        x = self.fc3(x)                   # Output layer without activation (for logits)
        return x

# Instantiate the model, define loss function and optimizer
model = NeuralNet()
criterion = nn.CrossEntropyLoss()
optimizer = optim.Adam(model.parameters(), lr=0.001)

# Training the model
epochs = 5
for epoch in range(epochs):
    for images, labels in train_loader:
        optimizer.zero_grad()               # Zero the gradients
        outputs = model(images)             # Forward pass
        loss = criterion(outputs, labels)   # Compute loss
        loss.backward()                     # Backward pass
        optimizer.step()                    # Update weights
    print(f"Epoch [{epoch+1}/{epochs}], Loss: {loss.item():.4f}")

# Evaluating the model
correct = 0
total = 0
with torch.no_grad():  # No need to calculate gradients for evaluation
    for images, labels in test_loader:
        outputs = model(images)
        _, predicted = torch.max(outputs.data, 1)
        total += labels.size(0)
        correct += (predicted == labels).sum().item()

print("OUTPUT:\n\n")
print(f"Accuracy of the model on the 10,000 test images: {100 * correct / total:.2f}%")
```

```
Downloading http://yann.lecun.com/exdb/mnist/train-images-idx3-ubyte.gz
Failed to download (trying next):
<urlopen error [SSL: CERTIFICATE_VERIFY_FAILED] certificate verify failed: certificate has expired (_ssl.c:1006)>

Downloading https://ossci-datasets.s3.amazonaws.com/mnist/train-images-idx3-ubyte.gz
Downloading https://ossci-datasets.s3.amazonaws.com/mnist/train-images-idx3-ubyte.gz to ./data\MNIST\raw\train-image
s-idx3-ubyte.gz
100%|██████████| 9.91M/9.91M [04:45<00:00, 34.7kB/s]
```

```
Extracting ./data\MNIST\raw\train-images-idx3-ubyte.gz to ./data\MNIST\raw

Downloading http://yann.lecun.com/exdb/mnist/train-labels-idx1-ubyte.gz
Failed to download (trying next):
<urlopen error [SSL: CERTIFICATE_VERIFY_FAILED] certificate verify failed: certificate has expired (_ssl.c:1006)>

Downloading https://ossci-datasets.s3.amazonaws.com/mnist/train-labels-idx1-ubyte.gz
Downloading https://ossci-datasets.s3.amazonaws.com/mnist/train-labels-idx1-ubyte.gz to ./data\MNIST\raw\train-label
s-idx1-ubyte.gz
100%|██████████| 28.9k/28.9k [00:00<00:00, 110kB/s]
Extracting ./data\MNIST\raw\train-labels-idx1-ubyte.gz to ./data\MNIST\raw

Downloading http://yann.lecun.com/exdb/mnist/t10k-images-idx3-ubyte.gz
Failed to download (trying next):
<urlopen error [SSL: CERTIFICATE_VERIFY_FAILED] certificate verify failed: certificate has expired (_ssl.c:1006)>

Downloading https://ossci-datasets.s3.amazonaws.com/mnist/t10k-images-idx3-ubyte.gz
Downloading https://ossci-datasets.s3.amazonaws.com/mnist/t10k-images-idx3-ubyte.gz to ./data\MNIST\raw\t10k-images-
idx3-ubyte.gz
100%|██████████| 1.65M/1.65M [01:13<00:00, 22.5kB/s]
Extracting ./data\MNIST\raw\t10k-images-idx3-ubyte.gz to ./data\MNIST\raw

Downloading http://yann.lecun.com/exdb/mnist/t10k-labels-idx1-ubyte.gz
Failed to download (trying next):
<urlopen error [SSL: CERTIFICATE_VERIFY_FAILED] certificate verify failed: certificate has expired (_ssl.c:1006)>

Downloading https://ossci-datasets.s3.amazonaws.com/mnist/t10k-labels-idx1-ubyte.gz
Downloading https://ossci-datasets.s3.amazonaws.com/mnist/t10k-labels-idx1-ubyte.gz to ./data\MNIST\raw\t10k-labels-
idx1-ubyte.gz
100%|██████████| 4.54k/4.54k [00:00<00:00, 911kB/s]
Extracting ./data\MNIST\raw\t10k-labels-idx1-ubyte.gz to ./data\MNIST\raw

Epoch [1/5], Loss: 0.2970
Epoch [2/5], Loss: 0.1825
Epoch [3/5], Loss: 0.0771
Epoch [4/5], Loss: 0.0327
Epoch [5/5], Loss: 0.0104
OUTPUT:


Accuracy of the model on the 10,000 test images: 96.79%
```

In [ ]:

Result
As a result of this Experiment, we successfully wrote and executed the to implement multi layer neural network in python.

Learning Outcomes
Understand and implement a multi-layer neural network, including forward propagation, backpropagation, and optimization, for classification and regression tasks.