**✅ What is Deep Learning?**

**Deep Learning** is a subset of **Machine Learning** that uses **neural networks with multiple layers** to automatically learn patterns from large amounts of data. It is especially powerful in tasks such as image recognition, natural language processing, and time series prediction.

**✅ What is a Neural Network?**

A **Neural Network** is inspired by the human brain. It consists of layers of **neurons (nodes)**:

* **Input Layer**: Takes the input data.
* **Hidden Layers**: Perform computations through weights and activations.
* **Output Layer**: Produces the final result (like classification or prediction).

Each neuron is connected to others with **weights**, and uses an **activation function** like ReLU or sigmoid to introduce non-linearity.

**🔍 Example: Deep Learning with Neural Networks in Python**

We will:

* Use the **MNIST digits dataset** (handwritten digits).
* Build a **Deep Neural Network** using **TensorFlow/Keras**.
* Train the model to recognize digits from 0 to 9.

**🛠️ Step-by-Step Python Code**

# Step 1: Import Required Libraries

import numpy as np

import matplotlib.pyplot as plt

from tensorflow.keras.datasets import mnist

from tensorflow.keras.models import Sequential

from tensorflow.keras.layers import Dense, Flatten

from tensorflow.keras.utils import to\_categorical

# Step 2: Load the Dataset

(x\_train, y\_train), (x\_test, y\_test) = mnist.load\_data()

# Step 3: Preprocess the Data

x\_train = x\_train / 255.0 # Normalize pixel values

x\_test = x\_test / 255.0

y\_train = to\_categorical(y\_train, 10) # One-hot encode labels

y\_test = to\_categorical(y\_test, 10)

# Step 4: Visualize Sample

plt.imshow(x\_train[0], cmap='gray')

plt.title("Sample Image")

plt.show()

# Step 5: Build the Neural Network Model

model = Sequential([

Flatten(input\_shape=(28, 28)), # Convert 28x28 to 784 input vector

Dense(128, activation='relu'), # Hidden Layer 1

Dense(64, activation='relu'), # Hidden Layer 2

Dense(10, activation='softmax') # Output Layer (10 classes)

])

# Step 6: Compile the Model

model.compile(optimizer='adam',

loss='categorical\_crossentropy',

metrics=['accuracy'])

# Step 7: Train the Model

model.fit(x\_train, y\_train, epochs=5, batch\_size=32, validation\_split=0.2)

# Step 8: Evaluate the Model

loss, accuracy = model.evaluate(x\_test, y\_test)

print(f"Test Accuracy: {accuracy:.4f}")

# Step 9: Make Predictions

predictions = model.predict(x\_test)

predicted\_label = np.argmax(predictions[0])

print(f"Predicted Label: {predicted\_label}")

# Show the predicted image

plt.imshow(x\_test[0], cmap='gray')

plt.title(f"Predicted Digit: {predicted\_label}")

plt.show()

**📊 Output**

* Training logs showing accuracy improvement.
* Final **test accuracy** (typically around 97–98%).
* Image of a digit and the model's prediction.

**📦 Dataset Used**

* **MNIST** dataset: 70,000 images of handwritten digits.
* Format: 28x28 grayscale images.
* 60,000 for training, 10,000 for testing.

**🧠 Recap: Key Concepts**

| **Concept** | **Description** |
| --- | --- |
| Deep Learning | ML technique with multi-layered neural networks |
| Neural Network | Layers of nodes that simulate human brain learning |
| Activation Function | Introduces non-linearity (e.g., ReLU, softmax) |
| Epoch | One full pass through the dataset |
| One-hot Encoding | Converts labels to binary class matrix |
| Accuracy | Correct predictions / total predictions |

**🧠 What is a Neural Network?**

A **Neural Network (NN)** is a **computational model inspired by the human brain**. It is composed of **layers of interconnected nodes** (called neurons), and it learns from data by adjusting **weights** during training.

**🧬 Structure of a Neural Network**

A typical neural network has:

1. **Input Layer** – Takes the raw data as input.
2. **Hidden Layers** – Perform computations with activation functions.
3. **Output Layer** – Produces the final prediction.

Each connection between neurons has a **weight**, and neurons apply **activation functions** (e.g., ReLU, sigmoid, softmax) to introduce non-linearity.

**🧪 Step-by-Step Example using Python**

Let’s build a **Neural Network to classify the Iris dataset**.

We’ll use:

* scikit-learn to load and preprocess data.
* TensorFlow/Keras to build and train the NN.

**✅ Step 1: Install Required Libraries (if needed)**

pip install tensorflow scikit-learn matplotlib

**✅ Step 2: Full Python Code – Neural Network on Iris Dataset**

# Step 1: Import Libraries

import numpy as np

import matplotlib.pyplot as plt

from sklearn.datasets import load\_iris

from sklearn.model\_selection import train\_test\_split

from sklearn.preprocessing import StandardScaler, OneHotEncoder

from tensorflow.keras.models import Sequential

from tensorflow.keras.layers import Dense

from tensorflow.keras.utils import to\_categorical

# Step 2: Load and Prepare the Dataset

iris = load\_iris()

X = iris.data # Features: Sepal & Petal length and width

y = iris.target # Labels: 0 - setosa, 1 - versicolor, 2 - virginica

# Step 3: Preprocess the Data

scaler = StandardScaler()

X\_scaled = scaler.fit\_transform(X)

y\_encoded = to\_categorical(y) # One-hot encode labels

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X\_scaled, y\_encoded, test\_size=0.2, random\_state=42)

# Step 4: Build the Neural Network Model

model = Sequential([

Dense(10, input\_shape=(4,), activation='relu'), # Hidden layer with 10 neurons

Dense(8, activation='relu'), # Another hidden layer

Dense(3, activation='softmax') # Output layer (3 classes)

])

# Step 5: Compile the Model

model.compile(optimizer='adam',

loss='categorical\_crossentropy',

metrics=['accuracy'])

# Step 6: Train the Model

history = model.fit(X\_train, y\_train, epochs=50, batch\_size=5, validation\_split=0.1)

# Step 7: Evaluate the Model

loss, accuracy = model.evaluate(X\_test, y\_test)

print(f"Test Accuracy: {accuracy:.2f}")

# Step 8: Make Predictions

predictions = model.predict(X\_test)

predicted\_classes = np.argmax(predictions, axis=1)

true\_classes = np.argmax(y\_test, axis=1)

# Step 9: Visualize Accuracy

plt.plot(history.history['accuracy'], label='Train Accuracy')

plt.plot(history.history['val\_accuracy'], label='Validation Accuracy')

plt.xlabel("Epoch")

plt.ylabel("Accuracy")

plt.title("Training vs Validation Accuracy")

plt.legend()

plt.show()

**📌 Output**

* **Model accuracy** on test set (typically > 95%)
* **Graph** showing training vs validation accuracy

**📚 Summary of Each Step**

| **Step** | **Description** |
| --- | --- |
| 1 | Import necessary libraries (Keras, scikit-learn, etc.) |
| 2 | Load Iris dataset |
| 3 | Scale features & one-hot encode labels |
| 4 | Define a neural network architecture |
| 5 | Compile the model with optimizer and loss function |
| 6 | Train the model on training data |
| 7 | Evaluate on test data |
| 8 | Predict and compare outputs |
| 9 | Plot accuracy for visualization |