Multi-Layer Perceptron (MLP) in TensorFlow

# 1. Introduction

A Multi-Layer Perceptron (MLP) is a type of Artificial Neural Network that consists of:  
- Input Layer – Takes input features.  
- Hidden Layers – Perform transformations using weighted connections and activation functions.  
- Output Layer – Provides the final prediction (classification/regression).  
  
It is called multi-layer because it has more than one layer between input and output.

# 2. Components of MLP

1. Input Layer – Each neuron corresponds to one input feature.

2. Hidden Layers – Perform weighted sum + activation function.

3. Output Layer – Produces final prediction (Softmax/Sigmoid/Linear).

# 3. Working of MLP

## a) Forward Propagation

Weighted Sum: z = Σ(wi\*xi + b)

Activation Functions: Sigmoid, ReLU, Tanh

## b) Loss Function

Classification: Binary/Categorical Cross-Entropy  
Regression: Mean Squared Error (MSE)

## c) Backpropagation

Gradient Descent Update: w = w - η \* ∂L/∂w

## d) Optimization Algorithms

SGD, Adam Optimizer

# 4. Implementation in TensorFlow

Step 1: Import Libraries & Load Dataset

import tensorflow as tf  
import numpy as np  
import matplotlib.pyplot as plt  
from tensorflow.keras.models import Sequential  
from tensorflow.keras.layers import Flatten, Dense  
  
(x\_train, y\_train), (x\_test, y\_test) = tf.keras.datasets.mnist.load\_data()

Step 2: Normalize Data

x\_train = x\_train.astype('float32') / 255  
x\_test = x\_test.astype('float32') / 255

Step 3: Visualize Data

fig, ax = plt.subplots(10, 10)  
k = 0  
for i in range(10):  
 for j in range(10):  
 ax[i][j].imshow(x\_train[k], cmap='gray')  
 ax[i][j].axis('off')  
 k += 1  
plt.show()

Step 4: Build Model

model = Sequential([  
 Flatten(input\_shape=(28, 28)),  
 Dense(256, activation='sigmoid'),  
 Dense(128, activation='sigmoid'),  
 Dense(10, activation='softmax')  
])

Step 5: Compile Model

model.compile(optimizer='adam', loss='sparse\_categorical\_crossentropy', metrics=['accuracy'])

Step 6: Train Model

mod = model.fit(x\_train, y\_train, epochs=10, batch\_size=2000, validation\_split=0.2)

Step 7: Evaluate Model

results = model.evaluate(x\_test, y\_test, verbose=0)  
print('Test Loss, Test Accuracy:', results)

Step 8: Visualize Accuracy & Loss

plt.figure(figsize=(12, 5))  
plt.subplot(1, 2, 1)  
plt.plot(mod.history['accuracy'], label='Train Accuracy')  
plt.plot(mod.history['val\_accuracy'], label='Validation Accuracy')  
plt.legend(); plt.title('Accuracy')  
  
plt.subplot(1, 2, 2)  
plt.plot(mod.history['loss'], label='Train Loss')  
plt.plot(mod.history['val\_loss'], label='Validation Loss')  
plt.legend(); plt.title('Loss')  
  
plt.show()

# 5. Advantages of MLP

✅ Versatile – works for classification & regression

✅ Handles non-linear data

✅ Trainable using GPUs → fast parallel computation

# 6. Disadvantages of MLP

❌ Training is expensive for large data

❌ Overfitting without regularization

❌ Sensitive to data scaling

# 7. Conclusion

MLP is a powerful supervised learning model capable of learning complex mappings between input and output. Our TensorFlow implementation achieved ~92% accuracy on the MNIST dataset. With regularization and hyperparameter tuning, performance can be improved further.