## EX 2 MULTI-LAYER PERCEPTRON (MLP) FOR CLASSIFICATION

#### **Problem Statement:**

Develop a Multi-Layer Perceptron (MLP) for a simple classification task. Experiment with different numbers of hidden layers and activation functions, and evaluate the model's performance using accuracy and loss.

Suggested Dataset: Iris Dataset

# **Objectives:**

- 1. Understand the structure and purpose of MLPs for classification.
- 2. Experiment with various hidden layer configurations and activation functions.
- Train the MLP using the Iris dataset and evaluate its accuracy and loss.
- 4. Visualize training progress and use the trained model for prediction.

#### Scope:

This experiment provides insights into how neural networks with multiple layers (MLPs) perform on structured classification tasks. It demonstrates model tuning using different architectures and activation functions, an essential concept in designing effective deep learning models.

Tools and Libraries Used:

- 1. Python 3.x
- 2. TensorFlow / Keras
- scikit-learn (for data preprocessing and dataset loading)
- 4. Matplotlib (for visualization)

## **Implementation Steps:**

#### Step 1: Import Necessary Libraries

import matplotlib.pyplot as plt import numpy as np 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

## Step 2: Load and Preprocess Data

```
iris = load_iris()

X = iris.data

y = iris.target.reshape(-1, 1)
```

```
classes = iris.target_names
encoder = OneHotEncoder(sparse_output=False)
y_encoded = encoder.fit_transform(y)
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)
X_train, X_test, y_train, y_test = train_test_split(X_scaled, y_encoded, test_size=0.2,
random_state=42)
Step 3: Define MLP Model Creation Function
def create_mlp(input_dim, output_dim, hidden_layers, activation='relu'):
  model = Sequential()
  model.add(Dense(hidden_layers[o], input_dim=input_dim, activation=activation))
  for units in hidden layers[1:]:
    model.add(Dense(units, activation=activation))
  model.add(Dense(output dim, activation='softmax'))
  model.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accuracy'])
  return model
Step 4: Train and Evaluate MLP with Various Configurations
hidden_layer_configs = [[8], [16, 8], [32, 16, 8]]
activations = ['relu', 'tanh', 'sigmoid']
for hidden layers in hidden layer configs:
  for activation in activations:
    print(f"\nTesting MLP with hidden_layers={hidden_layers}, activation={activation}")
    model = create_mlp(input_dim=4, output_dim=3, hidden_layers=hidden_layers,
activation=activation)
    history = model.fit(X_train, y_train, epochs=50, batch_size=5,
validation split=0.1)
    test_loss, test_acc = model.evaluate(X_test, y_test, verbose=o)
    print(f"Test Accuracy: {test_acc:.4f}, Test Loss: {test_loss:.4f}")
Step 5: Plot Accuracy and Loss Curves
    # Plot accuracy and loss
    plt.figure(figsize=(10, 4))
    plt.suptitle(f"Config: {hidden_layers}, Activation: {activation}", fontsize=14)
    plt.subplot(1, 2, 1)
    plt.plot(history.history['accuracy'], label='Train Acc')
    plt.plot(history.history['val_accuracy'], label='Val Acc')
    plt.xlabel('Epoch')
    plt.ylabel('Accuracy')
```

AI23531 Deep Learning

```
plt.title('Model Accuracy')
plt.legend()

plt.subplot(1, 2, 2)
plt.plot(history.history['loss'], label='Train Loss')
plt.plot(history.history['val_loss'], label='Val Loss')
plt.xlabel('Epoch')
plt.ylabel('Loss')
plt.title('Model Loss')
plt.tight_layout()
plt.tight_layout()
plt.show()
```

# **Step 6: Predict on New Input**

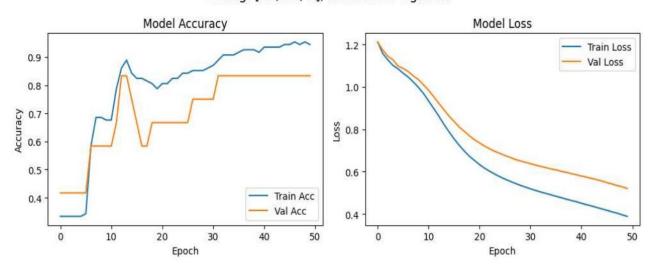
```
sepal_length = float(input("Sepal length (cm): "))
sepal_width = float(input("Sepal width (cm): "))
petal_length = float(input("Petal length (cm): "))
petal_width = float(input("Petal width (cm): "))

user_input = np.array([[sepal_length, sepal_width, petal_length, petal_width]])
user_input_scaled = scaler.transform(user_input)

# Predict using the last trained model
prediction = model.predict(user_input_scaled)
predicted_class_index = np.argmax(prediction)
predicted_class_name = classes[predicted_class_index]
print(f"\n* Predicted Iris Species: {predicted_class_name}")
```

## **Output:**

Config: [32, 16, 8], Activation: sigmoid



AI23531 Deep Learning