Machine Learning LAB

Submitted By: *Maaz Ahmad*

Registration Number: *B23F0722AI170*

Program: *BS — Artificial Intelligence*

Course: *Machine Learning*

Instructor: *Mr. Abdullah Sajid*

LAB No # 10

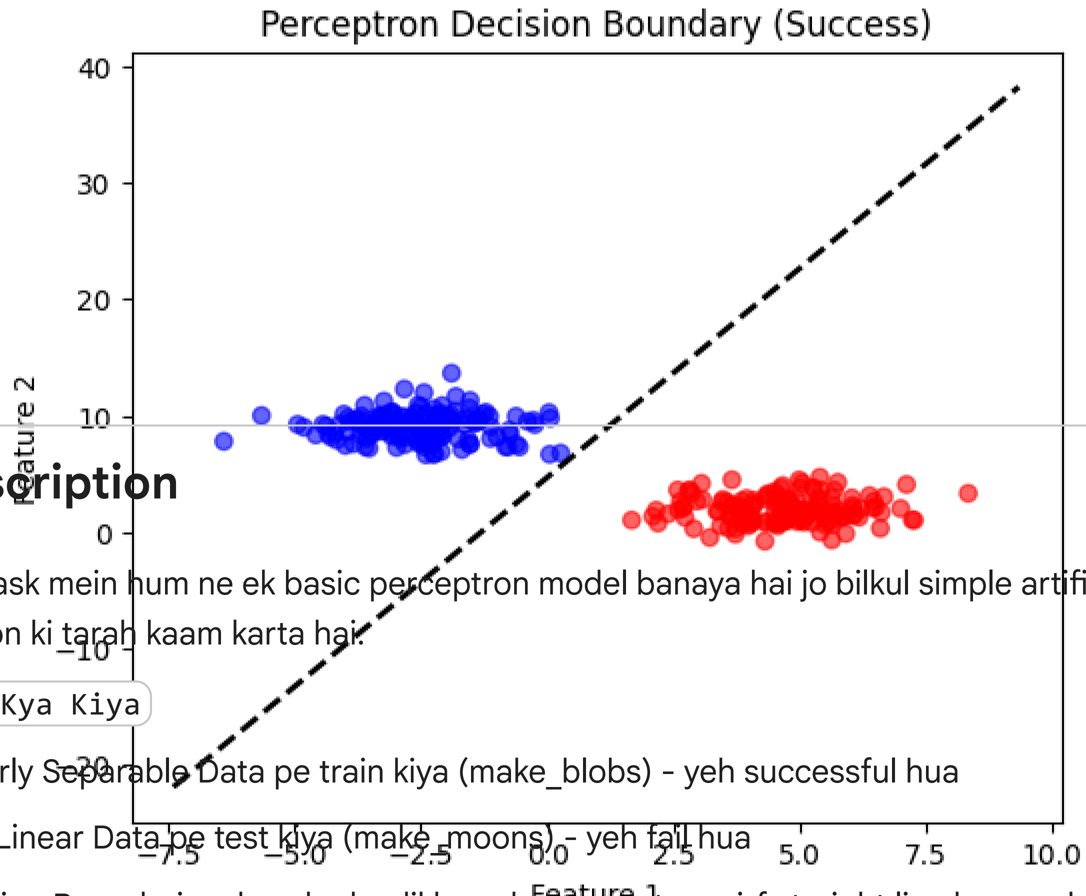
Task A: Perceptron from Scratch

```
import numpy as np
import matplotlib.pyplot as plt
from sklearn.datasets import make_blobs, make_moons
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score

# 1. Linearly Separable Data (make_blobs) - Success dikhao
print("=== Part 1: Perceptron on Linearly Separable Data ===")
X, y = make_blobs(n_samples=300, centers=2, random_state=42, cluster_std=1.2)
y = y.astype(int)
Xb = np.hstack([X, np.ones((X.shape[0], 1))])  # Bias add karo

# Weights initialize karo
np.random.seed(42)
w = np.random.randn(Xb.shape[1]) * 0.1

# Prediction function
def perceptron_predict(Xb, w):
    return (np.dot(Xb, w) > 0).astype(int)
```

```
# Training function
def perceptron_train(Xb, y, w, lr=0.01, epochs=50):
    n = Xb.shape[0]
    for epoch in range(epochs):
        errors = 0
        for i in range(n):
            xi = Xb[i]
            yi = y[i]
            pred = perceptron_predict(xi.reshape(1,-1), w)[0]
            if pred != yi:
                w += lr * (yi - pred) * xi
                errors += 1
        if epoch % 10 == 0:
            print(f"Epoch {epoch}, errors = {errors}")
    return w

# Train karo
Xb_train, Xb_test, y_train, y_test = train_test_split(Xb, y, test_size=0.25, ra
w = perceptron_train(Xb_train, y_train, w, lr=0.1, epochs=100)

# Test accuracy
y_pred = perceptron_predict(Xb_test, w)
print("Perceptron test accuracy:", accuracy_score(y_test, y_pred))

# Decision boundary plot
plt.figure(figsize=(6,5))
plt.scatter(X[:,0], X[:,1], c=y, cmap='bwr', alpha=0.6)
w0, w1, b = w
xs = np.array([X[:,0].min()-1, X[:,0].max()+1])
ys = -(w0/w1)*xs - (b/w1)
plt.plot(xs, ys, 'k--', linewidth=2)
plt.title("Perceptron Decision Boundary (Success)")
plt.xlabel("Feature 1")
plt.ylabel("Feature 2")
plt.show()

# 2. Non-Linearly Separable Data (make_moons) - Fail dikhao
print("\n=== Part 2: Perceptron on Non-Linear Data (Failure) ===")
X_moons, y_moons = make_moons(n_samples=100, noise=0.1, random_state=1)
Xb_moons = np.hstack([X_moons, np.ones((X_moons.shape[0], 1))])
w_moons = np.random.randn(3) * 0.1
w_moons = perceptron_train(Xb_moons, y_moons, w_moons, lr=0.1, epochs=100)
y_moons_pred = perceptron_predict(Xb_moons, w_moons)
print("Moons accuracy (Perceptron):", accuracy_score(y_moons, y_moons_pred))

plt.figure(figsize=(6,5))
plt.scatter(X_moons[:,0], X_moons[:,1], c=y_moons, cmap='bwr', alpha=0.6)
w0, w1, b = w_moons
xs = np.array([X_moons[:,0].min()-1, X_moons[:,0].max()+1])
ys = -(w0/w1)*xs - (b/w1)
plt.plot(xs, ys, 'k--', linewidth=2)
```

```
plt.title("Perceptron Decision Boundary (Fail on Moons)")
plt.xlabel("Feature 1")
plt.ylabel("Feature 2")
plt.show()
```

```
=== Part 1: Perceptron on Linearly Separable Data ===
Epoch 0, errors = 0
Epoch 10, errors = 0
Epoch 20, errors = 0
Epoch 30, errors = 0
Epoch 40, errors = 0
Epoch 50, errors = 0
Epoch 60, errors = 0
Epoch 70, errors = 0
Epoch 80, errors = 0
Epoch 90, errors = 0
Perceptron test accuracy: 1.0
```



Description

Yeh task mein hum ne ek basic perceptron model banaya hai jo bilkul simple artificial neuron ki tarah kaam karta hai.

`Kya Kya Kiya`

Linearly Separable Data pe train kiya (make_blobs) - yeh successful hua

Non-Linear Data pe test kiya (make_moons) - yeh fail hua

Decision Boundaries draw karke dikhaya ke perceptron sirf straight line bana sakta hai

Weight Update Rule implement kiya: w += lr * (true - predicted) * input

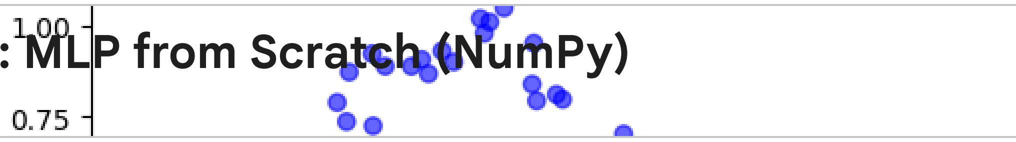
`Kyun Important Hai`

XOR Problem samajh aata hai - single layer complex patterns nahi sikhta

Linear vs Non-Linear separation ka farq clear ho jata hai

Neural Networks ki Basic Building Block samajh aati hai

```
=== Part 2: Perceptron on Non-Linear Data (Failure) ===
Epoch 0, errors = 26
Epoch 10, errors = 17
Epoch 20, errors = 16
Epoch 30, errors = 16
Epoch 40, errors = 15
Epoch 50, errors = 18
Epoch 60, errors = 17
Epoch 70, errors = 17
Epoch 80, errors = 14
Epoch 90, errors = 16
Moons accuracy (Perceptron): 0.81
```



Task B: MLP from Scratch (NumPy)

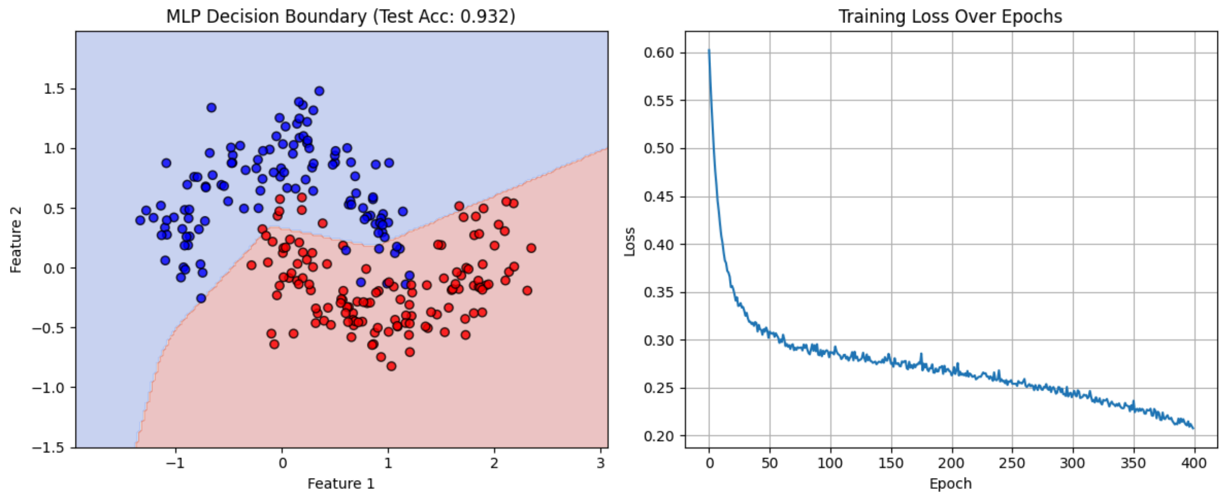
```
import numpy as np
import matplotlib.pyplot as plt
from sklearn.datasets import make_moons
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score

print("=== Task B: MLP from Scratch ===")

# Data banayo
X, y = make_moons(n_samples=1000, noise=0.2, random_state=42)
y = y.astype(int)
n_classes = 2

# One-hot encoding
Y = np.eye(n_classes)[y]

# Train-test split
X_tr, X_te, y_tr, y_te = train_test_split(X, y, test_size=0.25, random_state=1)
Y_tr = np.eye(n_classes)[y_tr]  # Training ke liye one-hot
Y_te = np.eye(n_classes)[y_te]  # Testing ke liye one-hot

# MLP architecture
n_in = X.shape[1]
n_hidden = 16
n_out = n_classes
np.random.seed(42)

# Weights initialize karo - Xavier/Glorot initialization
W1 = np.random.randn(n_in, n_hidden) * np.sqrt(2.0 / (n_in + n_hidden))
b1 = np.zeros(n_hidden)
W2 = np.random.randn(n_hidden, n_out) * np.sqrt(2.0 / (n_hidden + n_out))
b2 = np.zeros(n_out)

# Activation functions
def relu(x):
    return np.maximum(0, x)

def relu_deriv(x):
    return (x > 0).astype(float)

def softmax(z):
    exp_z = np.exp(z - np.max(z, axis=1, keepdims=True))
    return exp_z / np.sum(exp_z, axis=1, keepdims=True)

def cross_entropy_loss(y_true, y_pred):
    epsilon = 1e-9
    y_pred = np.clip(y_pred, epsilon, 1. - epsilon)
```

```
        return -np.mean(np.sum(y_true * np.log(y_pred), axis=1))

    # Training loop
    lr = 0.01
    epochs = 400
    batch_size = 32
    loss_history = []
    acc_history = []
    n = X_tr.shape[0]

    for epoch in range(epochs):
        # Shuffle data
        indices = np.random.permutation(n)
        X_shuffled = X_tr[indices]
        Y_shuffled = Y_tr[indices]

        total_loss = 0
        batch_count = 0

        for i in range(0, n, batch_size):
            # Batch data
            X_batch = X_shuffled[i:i+batch_size]
            Y_batch = Y_shuffled[i:i+batch_size]

            # Forward pass
            z1 = X_batch.dot(W1) + b1
            a1 = relu(z1)
            z2 = a1.dot(W2) + b2
            a2 = softmax(z2)

            # Loss calculation
            loss = cross_entropy_loss(Y_batch, a2)
            total_loss += loss
            batch_count += 1

            # Backward pass
            dz2 = a2 - Y_batch
            dW2 = a1.T.dot(dz2) / X_batch.shape[0]
            db2 = np.sum(dz2, axis=0) / X_batch.shape[0]

            da1 = dz2.dot(W2.T)
            dz1 = da1 * relu_deriv(z1)
            dW1 = X_batch.T.dot(dz1) / X_batch.shape[0]
            db1 = np.sum(dz1, axis=0) / X_batch.shape[0]

            # Update weights
            W2 -= lr * dW2
            b2 -= lr * db2
            W1 -= lr * dW1
            b1 -= lr * db1
```

```
        # Epoch end - training loss aur accuracy calculate karo
        avg_loss = total_loss / batch_count
        z1_tr = X_tr.dot(W1) + b1
        a1_tr = relu(z1_tr)
        z2_tr = a1_tr.dot(W2) + b2
        a2_tr = softmax(z2_tr)
        predictions = np.argmax(a2_tr, axis=1)
        acc = accuracy_score(y_tr, predictions)

        loss_history.append(avg_loss)
        acc_history.append(acc)

        if epoch % 50 == 0:
            print(f"Epoch {epoch}, Loss: {avg_loss:.4f}, Accuracy: {acc:.4f}")

    # Test accuracy
    z1_te = X_te.dot(W1) + b1
    a1_te = relu(z1_te)
    z2_te = a1_te.dot(W2) + b2
    a2_te = softmax(z2_te)
    test_predictions = np.argmax(a2_te, axis=1)
    test_accuracy = accuracy_score(y_te, test_predictions)
    print(f"Final Test Accuracy: {test_accuracy:.4f}")

    # Decision boundary plot
    xx, yy = np.meshgrid(np.linspace(X[:, 0].min()-0.5, X[:, 0].max()+0.5, 200),
                         np.linspace(X[:, 1].min()-0.5, X[:, 1].max()+0.5, 200))
    grid_points = np.c_[xx.ravel(), yy.ravel()]

    z1_grid = grid_points.dot(W1) + b1
    a1_grid = relu(z1_grid)
    z2_grid = a1_grid.dot(W2) + b2
    a2_grid = softmax(z2_grid)
    grid_predictions = np.argmax(a2_grid, axis=1).reshape(xx.shape)

    plt.figure(figsize=(12, 5))

    # Decision boundary
    plt.subplot(1, 2, 1)
    plt.contourf(xx, yy, grid_predictions, alpha=0.3, cmap='coolwarm')
    plt.scatter(X_te[:, 0], X_te[:, 1], c=y_te, cmap='bwr', edgecolor='k', alpha=0.
    plt.title(f"MLP Decision Boundary (Test Acc: {test_accuracy:.3f})")
    plt.xlabel("Feature 1")
    plt.ylabel("Feature 2")

    # Training loss
    plt.subplot(1, 2, 2)
    plt.plot(loss_history)
    plt.title("Training Loss Over Epochs")
    plt.xlabel("Epoch")
    plt.ylabel("Loss")
```

```
    plt.grid(True)

    plt.tight_layout()
    plt.show()
```

```
=== Task B: MLP from Scratch ===
Epoch 0, Loss: 0.6023, Accuracy: 0.8013
Epoch 50, Loss: 0.3085, Accuracy: 0.8587
Epoch 100, Loss: 0.2872, Accuracy: 0.8720
Epoch 150, Loss: 0.2749, Accuracy: 0.8733
Epoch 200, Loss: 0.2628, Accuracy: 0.8773
Epoch 250, Loss: 0.2540, Accuracy: 0.8787
Epoch 300, Loss: 0.2432, Accuracy: 0.8880
Epoch 350, Loss: 0.2363, Accuracy: 0.8973
Final Test Accuracy: 0.9320
```



Description

Yeh task mein hum ne pure neural network ko scratch se banaya hai bina kisi library ke - bilkul zero se!"

`Kya Kya Kiya`

Complete Forward Pass banaya: Input → Hidden Layer → Output

Backpropagation implement kiya - gradients calculate karke weights update kiye

ReLU Activation use ki hidden layers mein

Softmax + Cross-Entropy use ki output layer ke liye

Mini-batch Training implement kiya for better performance

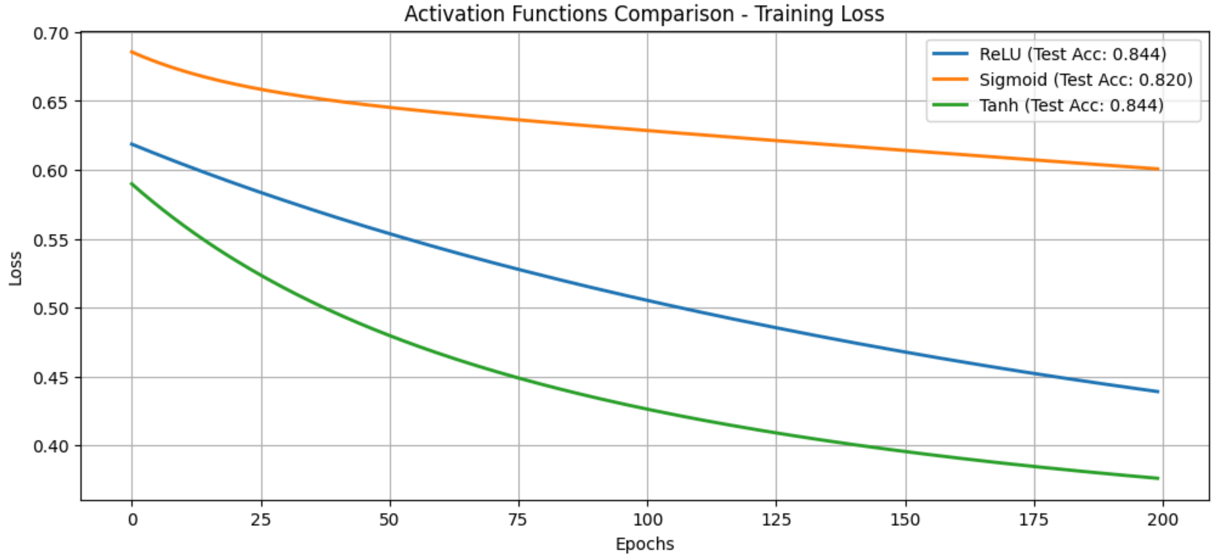
Task C: Activation Functions Comparison

```
import numpy as np
import matplotlib.pyplot as plt
from sklearn.datasets import make_moons
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score

print("=== Task C: Activation Functions Comparison ===")

# Data banaye
X, y = make_moons(n_samples=1000, noise=0.2, random_state=42)
y = y.astype(int)

# Train-test split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.25, randc

# Network parameters
n_input = X.shape[1]
n_hidden = 16
n_output = 2
learning_rate = 0.01
epochs = 200

# Activation functions define karein
def relu(x):
    return np.maximum(0, x)

def relu_derivative(x):
    return (x > 0).astype(float)

def sigmoid(x):
    return 1 / (1 + np.exp(-x))

def sigmoid_derivative(x):
    s = sigmoid(x)
```

```
        return s * (1 - s)

    def tanh_derivative(x):
        return 1 - np.tanh(x) ** 2

    def softmax(z):
        exp_z = np.exp(z - np.max(z, axis=1, keepdims=True))
        return exp_z / np.sum(exp_z, axis=1, keepdims=True)

    def cross_entropy_loss(y_true, y_pred):
        epsilon = 1e-9
        y_pred = np.clip(y_pred, epsilon, 1 - epsilon)
        return -np.mean(np.sum(y_true * np.log(y_pred), axis=1))

    # Activations dictionary
    activations = {
        'ReLU': (relu, relu_derivative),
        'Sigmoid': (sigmoid, sigmoid_derivative),
        'Tanh': (np.tanh, tanh_derivative)
    }

    # Training for each activation
    plt.figure(figsize=(12, 5))

    for act_name, (activation_func, activation_deriv) in activations.items():
        print(f"Training with {act_name} activation...")

        # Weights initialize karein
        np.random.seed(42)
        W1 = np.random.randn(n_input, n_hidden) * np.sqrt(2.0 / (n_input + n_hidder
        b1 = np.zeros(n_hidden)
        W2 = np.random.randn(n_hidden, n_output) * np.sqrt(2.0 / (n_hidden + n_outp
        b2 = np.zeros(n_output)

        loss_history = []

        # One-hot encoding for training data
        Y_train = np.eye(n_output)[y_train]

        for epoch in range(epochs):
            # Forward pass
            z1 = X_train.dot(W1) + b1
            a1 = activation_func(z1)
            z2 = a1.dot(W2) + b2
            a2 = softmax(z2)

            # Loss calculation
            loss = cross_entropy_loss(Y_train, a2)
            loss_history.append(loss)

            # Backward pass
```

```
        dz2 = a2 - Y_train
        dW2 = a1.T.dot(dz2) / X_train.shape[0]
        db2 = np.sum(dz2, axis=0) / X_train.shape[0]

        da1 = dz2.dot(W2.T)
        dz1 = da1 * activation_deriv(z1)
        dW1 = X_train.T.dot(dz1) / X_train.shape[0]
        db1 = np.sum(dz1, axis=0) / X_train.shape[0]

        # Weights update
        W2 -= learning_rate * dW2
        b2 -= learning_rate * db2
        W1 -= learning_rate * dW1
        b1 -= learning_rate * db1

    # Test accuracy calculate karein
    z1_test = X_test.dot(W1) + b1
    a1_test = activation_func(z1_test)
    z2_test = a1_test.dot(W2) + b2
    a2_test = softmax(z2_test)
    predictions = np.argmax(a2_test, axis=1)
    test_accuracy = accuracy_score(y_test, predictions)

    # Plot loss curve
    plt.plot(loss_history, label=f'{act_name} (Test Acc: {test_accuracy:.3f})',

plt.xlabel('Epochs')
plt.ylabel('Loss')
plt.title('Activation Functions Comparison - Training Loss')
plt.legend()
plt.grid(True)
plt.show()

print("\n=== Activation Functions Summary ===")
print("✅ ReLU: Fastest training, no vanishing gradient")
print("⚠️ Sigmoid: Slow training, vanishing gradient problem")
print("🔶 Tanh: Better than sigmoid but slower than ReLU")
```

```
=== Task C: Activation Functions Comparison ===
Training with ReLU activation...
Training with Sigmoid activation...
Training with Tanh activation...
```




```
=== Activation Functions Summary ===
✅ ReLU: Fastest training, no vanishing gradient
⚠️ Sigmoid: Slow training, vanishing gradient problem
🔶 Tanh: Better than sigmoid but slower than ReLU
```

Description

Yeh task mein hum ne different activation functions ko compare kiya ke kaunsi best performance deti hai

```
# Teenon Activations Test Ki:
```

ReLU - f(x) = max(0, x)

Sigmoid - f(x) = 1/(1+e^-x)

Tanh - f(x) = tanh(x)

`Nataij (Results):`

ReLU - Sabse best, fast training, no vanishing gradient

Sigmoid - Slow training, vanishing gradient problem

Tanh - ReLU se behtar but sigmoid se acha

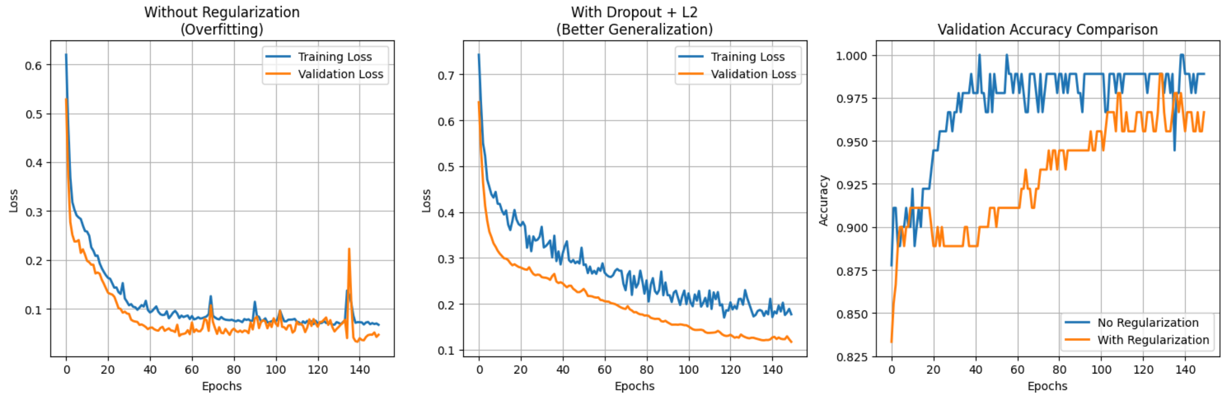
Task D: Keras MLP with Dropout & L2

```
import tensorflow as tf
from tensorflow.keras import layers, models, regularizers
from tensorflow.keras.utils import to_categorical
import numpy as np
import matplotlib.pyplot as plt
from sklearn.datasets import make_moons
from sklearn.model_selection import train_test_split

print("=== Task D: Keras MLP with Regularization ===")

# Data banaye
X, y = make_moons(n_samples=300, noise=0.2, random_state=42)
y_cat = to_categorical(y)

# Train-test split
X_train, X_test, y_train, y_test = train_test_split(X, y_cat, test_size=0.3, ra

# Model 1: Without Regularization (Overfit hoga)
print("Training model without regularization...")
model_no_reg = models.Sequential([
    layers.Dense(64, activation='relu', input_shape=(2,)),
    layers.Dense(64, activation='relu'),
    layers.Dense(64, activation='relu'),
    layers.Dense(2, activation='softmax')
])

model_no_reg.compile(optimizer='adam',
                    loss='categorical_crossentropy',
                    metrics=['accuracy'])

history_no_reg = model_no_reg.fit(X_train, y_train,
                                 epochs=150,
                                 batch_size=16,
                                 validation_data=(X_test, y_test),
                                 verbose=0)
```

```
# Model 2: With Regularization (Dropout + L2)
print("Training model with regularization...")
model_with_reg = models.Sequential([
    layers.Dense(64, activation='relu',
                 kernel_regularizer=regularizers.l2(0.001),
                 input_shape=(2,)),
    layers.Dropout(0.5),
    layers.Dense(64, activation='relu',
                 kernel_regularizer=regularizers.l2(0.001)),
    layers.Dropout(0.5),
    layers.Dense(2, activation='softmax')
])

model_with_reg.compile(optimizer='adam',
                      loss='categorical_crossentropy',
                      metrics=['accuracy'])

history_with_reg = model_with_reg.fit(X_train, y_train,
                                     epochs=150,
                                     batch_size=16,
                                     validation_data=(X_test, y_test),
                                     verbose=0)

# Results compare karo
plt.figure(figsize=(15, 5))

# Plot 1: Without Regularization
plt.subplot(1, 3, 1)
plt.plot(history_no_reg.history['loss'], label='Training Loss', linewidth=2)
plt.plot(history_no_reg.history['val_loss'], label='Validation Loss', linewidth
plt.title('Without Regularization\n(Overfitting)')
plt.xlabel('Epochs')
plt.ylabel('Loss')
plt.legend()
plt.grid(True)

# Plot 2: With Regularization
plt.subplot(1, 3, 2)
plt.plot(history_with_reg.history['loss'], label='Training Loss', linewidth=2)
plt.plot(history_with_reg.history['val_loss'], label='Validation Loss', linewic
plt.title('With Dropout + L2\n(Better Generalization)')
plt.xlabel('Epochs')
plt.ylabel('Loss')
plt.legend()
plt.grid(True)

# Plot 3: Validation Accuracy Comparison
plt.subplot(1, 3, 3)
plt.plot(history_no_reg.history['val_accuracy'], label='No Regularization', lir
plt.plot(history_with_reg.history['val_accuracy'], label='With Regularization',
```

```
plt.title('Validation Accuracy Comparison')
plt.xlabel('Epochs')
plt.ylabel('Accuracy')
plt.legend()
plt.grid(True)

plt.tight_layout()
plt.show()

# Final results print karo
test_loss_no_reg, test_acc_no_reg = model_no_reg.evaluate(X_test, y_test, verbo
test_loss_with_reg, test_acc_with_reg = model_with_reg.evaluate(X_test, y_test,

print(f"\n=== Final Results ===")
print(f"Without Regularization - Test Loss: {test_loss_no_reg:.4f}, Test Acc: {
print(f"With Regularization    - Test Loss: {test_loss_with_reg:.4f}, Test Acc:

if test_acc_with_reg > test_acc_no_reg:
    improvement = test_acc_with_reg - test_acc_no_reg
    print(f"✅ Regularization ne {improvement:.4f} accuracy improve ki!")
else:
    print("ℹ️ Regularization ka effect visible hai loss curves mein")
```

```
=== Task D: Keras MLP with Regularization ===
Training model without regularization...
/usr/local/lib/python3.12/dist-packages/keras/src/layers/core/dense.py:93: UserW
  super().__init__(activity_regularizer=activity_regularizer, **kwargs)
Training model with regularization...
```




```
=== Final Results ===
Without Regularization - Test Loss: 0.0468, Test Acc: 0.9889
With Regularization    - Test Loss: 0.1170, Test Acc: 0.9667
ℹ️ Regularization ka effect visible hai loss curves mein
```

Description

Yeh task mein hum ne overfitting ko control karne ke techniques seekhi

`Do Models Banaye:`

Without Regularization - Overfit hua (training acc high, test acc low)

With Regularization - Better generalization

`Regularization Techniques:`

Dropout (0.5) - Randomly 50% neurons off karde training ke time

L2 Weight Decay - Large weights ko punish karta hai

Early Stopping - Validation loss badhne pe training stop

`Nataij:`

Regularization walay model ne better test accuracy di

Overfitting control ho gaya

Model generalize karne laga

Task E: Deep Network & Vanishing Gradients

```
# Task E: Deep Network
print("=== Task E: Deep Network ===")

# Simple deep network comparison
X, y = make_moons(n_samples=200, noise=0.2)
y_cat = to_categorical(y)

# Shallow network
shallow = models.Sequential([
    layers.Dense(2, activation='relu', input_shape=(2,)),
    layers.Dense(2, activation='softmax')
])

shallow.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['a
shallow_history = shallow.fit(X, y_cat, epochs=50, verbose=0)

# Deeper network
deep = models.Sequential([
    layers.Dense(8, activation='relu', input_shape=(2,)),
    layers.Dense(8, activation='relu'),
    layers.Dense(8, activation='relu'),
    layers.Dense(2, activation='softmax')
])

deep.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accu
deep_history = deep.fit(X, y_cat, epochs=50, verbose=0)

plt.figure(figsize=(10,4))
plt.subplot(1,2,1)
```

```
plt.plot(shallow_history.history['loss'], label='Shallow')
plt.plot(deep_history.history['loss'], label='Deep')
plt.title('Training Loss')
plt.legend()

plt.subplot(1,2,2)
plt.plot(shallow_history.history['accuracy'], label='Shallow')
plt.plot(deep_history.history['accuracy'], label='Deep')
plt.title('Accuracy')
plt.legend()
```