DL Assignment

Name - Rohit Dahale

PRN - 202201070052

Problem Statement:

1) Logistic Regression from Scratch and using the Scikit-learn library for comparison.

2) ReLU and Tanh activation function plotting.

3) Log Loss calculation for binary classication.

4) KNN-like classication using Keras for a neural network model.

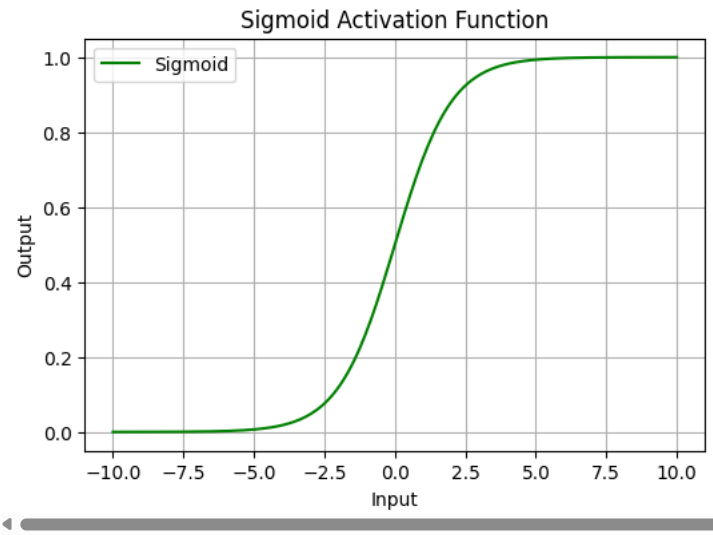
Activation Function Plotting

1. Sigmoid

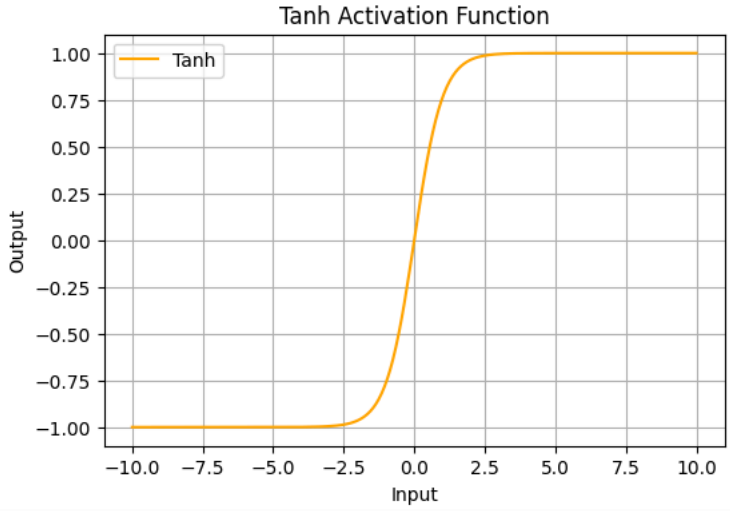
2. TanH

3. Relu

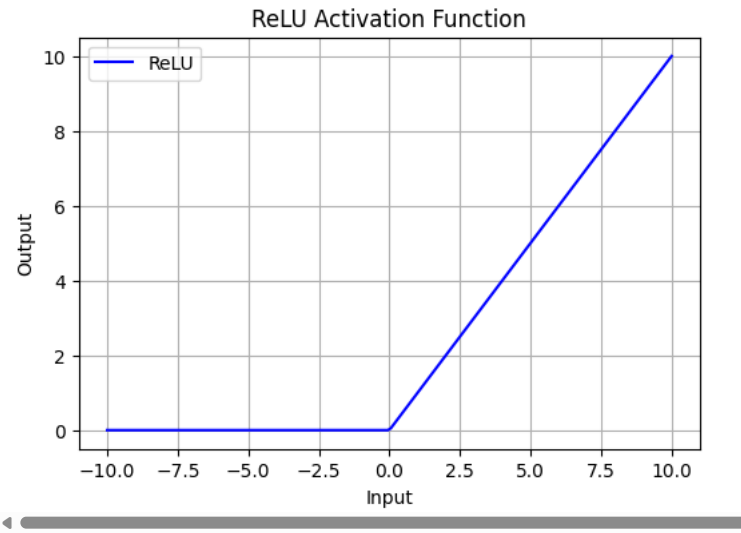
```
import numpy as np
import matplotlib.pyplot as plt

# Input range
x = np.linspace(-10, 10, 200)

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

def tanh(x):
    return np.tanh(x)

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


# Plot Sigmoid
plt.figure(figsize=(6, 4))
plt.plot(x, sigmoid(x), label='Sigmoid', color='green')
plt.title('Sigmoid Activation Function')
plt.xlabel('Input')
plt.ylabel('Output')
plt.grid(True)
plt.legend()
plt.show()
```



```
# Plot Tanh
plt.figure(figsize=(6, 4))
plt.plot(x, tanh(x), label='Tanh', color='orange')
plt.title('Tanh Activation Function')
plt.xlabel('Input')
plt.ylabel('Output')
plt.grid(True)
plt.legend()
plt.show()
```




```
# Plot ReLU
plt.figure(figsize=(6, 4))
plt.plot(x, relu(x), label='ReLU', color='blue')
plt.title('ReLU Activation Function')
plt.xlabel('Input')
plt.ylabel('Output')
plt.grid(True)
plt.legend()
plt.show()
```



Logistic Regression From Scratch

Binary: Setosa vs Not Setosa

```
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler


# Load the dataset
iris = pd.read_csv('Iris.csv')

# Drop unnecessary columns if present
if 'Id' in iris.columns:
```

```
    iris = iris.drop(columns=['Id'])

# Features and target
X = iris.drop(columns=['Species']).values  # All features
y = iris['Species'].values

# Convert to binary classification: Classify if it's Setosa (1) or not (0)
y_binary = (y == 'Iris-setosa').astype(int)

# Normalize features
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)

# Add bias term
X_scaled = np.c_[np.ones(X_scaled.shape[0]), X_scaled]  # shape: (150, 5)


# Sigmoid activation function
def sigmoid(x):
    return 1 / (1 + np.exp(-np.clip(x, -500, 500)))

# Log loss function
def compute_loss(y, y_pred):
    epsilon = 1e-9
    y_pred = np.clip(y_pred, epsilon, 1 - epsilon)
    return -np.mean(y * np.log(y_pred) + (1 - y) * np.log(1 - y_pred))

# Gradient descent training
def gradient_descent(X, y, lr=0.01, epochs=2000, clip_value=1.0):
    m, n = X.shape
    theta = np.random.randn(n) * 0.01
    for epoch in range(epochs):
        z = np.dot(X, theta)
        y_pred = sigmoid(z)
        loss = compute_loss(y, y_pred)
        gradient = np.dot(X.T, (y_pred - y)) / m
        gradient = np.clip(gradient, -clip_value, clip_value)
        theta -= lr * gradient
        if epoch % 200 == 0:
            print(f"Epoch {epoch}, Loss: {loss:.4f}")
    return theta


# Train-Test Split
X_train, X_test, y_train, y_test = train_test_split(X_scaled, y_binary, test_size=0.2, random_state=42)

# Flatten targets
y_train = y_train.flatten()
y_test = y_test.flatten()

# Train model
theta = gradient_descent(X_train, y_train, lr=0.01, epochs=2000)

# Prediction function
def predict(X, theta):
    return (sigmoid(np.dot(X, theta)) >= 0.5).astype(int)

# Evaluate performance
y_train_pred = predict(X_train, theta)
y_test_pred = predict(X_test, theta)

train_acc = np.mean(y_train_pred == y_train) * 100
test_acc = np.mean(y_test_pred == y_test) * 100

print(f"\nTrain Accuracy: {train_acc:.2f}%")
print(f"Test Accuracy: {test_acc:.2f}%")
print(f"Final Log Loss: {compute_loss(y_test, sigmoid(np.dot(X_test, theta))):.4f}")
```

```
Epoch 0, Loss: 0.6861
Epoch 200, Loss: 0.2533
Epoch 400, Loss: 0.1598
Epoch 600, Loss: 0.1186
Epoch 800, Loss: 0.0950
Epoch 1000, Loss: 0.0796
Epoch 1200, Loss: 0.0687
Epoch 1400, Loss: 0.0606
Epoch 1600, Loss: 0.0544
Epoch 1800, Loss: 0.0493

Train Accuracy: 100.00%
Test Accuracy: 100.00%
Final Log Loss: 0.0396
```

Log Loss

```
import numpy as np

def log_loss(y_true, y_pred):
    epsilon = 1e-15
    y_pred = np.clip(y_pred, epsilon, 1 - epsilon)
    return -np.mean(y_true * np.log(y_pred) + (1 - y_true) * np.log(1 - y_pred))

y_true = np.random.randint(0, 2, 10)  # Labels (0 or 1)
y_pred = np.random.rand(10)

loss = log_loss(y_true, y_pred)
print(f"Log Loss: {loss:.4f}")
```

```
Log Loss: 0.6483
```

Keras Binary Classification

```
from keras.models import Sequential
from keras.layers import Dense
from sklearn.datasets import load_iris
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
import numpy as np


# Load Iris CSV dataset
iris = pd.read_csv('Iris.csv')

# Drop 'Id' column if it exists
if 'Id' in iris.columns:
    iris = iris.drop(columns=['Id'])

# Convert to binary classification: Setosa vs Not Setosa
X = iris.drop(columns=['Species']).values
y = (iris['Species'] == 'Iris-setosa').astype(int).values  # 1 if Setosa, else 0

# Normalize features
scaler = StandardScaler()
X = scaler.fit_transform(X)

# Train-test split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)


# Build Keras model
model = Sequential()
model.add(Dense(16, input_dim=X.shape[1], activation='relu'))
model.add(Dense(8, activation='relu'))
model.add(Dense(1, activation='sigmoid'))

# Compile model
model.compile(loss='binary_crossentropy', optimizer='adam', metrics=['accuracy'])

# Train model
model.fit(X_train, y_train, epochs=100, batch_size=8, verbose=1)

# Evaluate
loss, accuracy = model.evaluate(X_test, y_test)
print(f"\n✅ Final Test Accuracy on Iris Binary Classification: {accuracy * 100:.2f}%")
```

```
15/15 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - accuracy: 1.0000 - loss: 0.0014
Epoch 83/100
15/15 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - accuracy: 1.0000 - loss: 0.0015
Epoch 84/100
15/15 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - accuracy: 1.0000 - loss: 0.0011
Epoch 85/100
15/15 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - accuracy: 1.0000 - loss: 0.0012
Epoch 86/100
15/15 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - accuracy: 1.0000 - loss: 7.9318e-04
Epoch 87/100
15/15 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - accuracy: 1.0000 - loss: 9.0900e-04
Epoch 88/100
15/15 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - accuracy: 1.0000 - loss: 7.8472e-04
Epoch 89/100
15/15 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - accuracy: 1.0000 - loss: 0.0012
Epoch 90/100
15/15 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - accuracy: 1.0000 - loss: 0.0011
Epoch 91/100
15/15 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - accuracy: 1.0000 - loss: 8.7002e-04
Epoch 92/100
15/15 ━━━━━━━━━━━━━━━━━━━━ 0s 7ms/step - accuracy: 1.0000 - loss: 9.4010e-04
Epoch 93/100
15/15 ━━━━━━━━━━━━━━━━━━━━ 0s 6ms/step - accuracy: 1.0000 - loss: 8.9173e-04
Epoch 94/100
15/15 ━━━━━━━━━━━━━━━━━━━━ 0s 6ms/step - accuracy: 1.0000 - loss: 7.7238e-04
Epoch 95/100
15/15 ━━━━━━━━━━━━━━━━━━━━ 0s 8ms/step - accuracy: 1.0000 - loss: 6.6343e-04
Epoch 96/100
15/15 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step - accuracy: 1.0000 - loss: 0.0011
Epoch 97/100
15/15 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step - accuracy: 1.0000 - loss: 0.0010
Epoch 98/100
15/15 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step - accuracy: 1.0000 - loss: 6.8758e-04
Epoch 99/100
15/15 ━━━━━━━━━━━━━━━━━━━━ 0s 6ms/step - accuracy: 1.0000 - loss: 7.3135e-04
Epoch 100/100
15/15 ━━━━━━━━━━━━━━━━━━━━ 0s 6ms/step - accuracy: 1.0000 - loss: 9.2071e-04
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 323ms/step - accuracy: 1.0000 - loss: 7.3555e-04

✅ Final Test Accuracy on Iris Binary Classification: 100.00%
```

Logistic Regression using Sciket Learn

```
import numpy as np
from sklearn.datasets import load_iris
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score, confusion_matrix, classification_report

# Load the Iris dataset
iris = load_iris()
X = iris.data
y = iris.target

# Normalize features
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)

# Train-test split
X_train, X_test, y_train, y_test = train_test_split(X_scaled, y, test_size=0.2, random_state=42)

# Build and train the logistic regression model (multi-class)
model = LogisticRegression(multi_class='multinomial', solver='lbfgs', max_iter=200)
model.fit(X_train, y_train)

# Predictions
y_pred = model.predict(X_test)

# Evaluation
acc = accuracy_score(y_test, y_pred)
print(f"✅ Accuracy: {acc * 100:.2f}%")
print("\nConfusion Matrix:")
print(confusion_matrix(y_test, y_pred))

print("\nClassification Report:")
print(classification_report(y_test, y_pred, target_names=iris.target_names))
```

```
✅ Accuracy: 100.00%

Confusion Matrix:
[[10  0  0]
 [ 0  9  0]
```

```
 [ 0  0 11]]

Classification Report:
              precision    recall  f1-score   support

      setosa       1.00      1.00      1.00        10
  versicolor       1.00      1.00      1.00         9
   virginica       1.00      1.00      1.00        11

    accuracy                           1.00        30
   macro avg       1.00      1.00      1.00        30
weighted avg       1.00      1.00      1.00        30

/usr/local/lib/python3.11/dist-packages/sklearn/linear_model/_logistic.py:1247: FutureWarning: 'multi_class' was deprecated in versi
  warnings.warn(
```

Start coding or generate with AI.