



Name : Sanket Dahotre

Class : T.E.

Roll No : 3029

ASSIGNMENT 1

1. Title

Write a Python program to plot a few activation functions that are being used in neural networks.

2. Code

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

class Neuron:
    def __init__(self, weights, bias, activation):
        self.weights = np.array(weights)
        self.bias = bias
        self.activation = activation

    def forward(self, inputs):
        inputs = np.array(inputs)
        z = np.dot(inputs, self.weights) + self.bias
        return self.activation(z)

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

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

    def tanh(x):
```

```

    return np.tanh(x)

# create neurons using the same class : # Same weights and bias for fair
comparison
weights = [1.0]
bias = 0.0

sigmoid_neuron = Neuron(weights, bias, sigmoid)
relu_neuron = Neuron(weights, bias, relu)
tanh_neuron = Neuron(weights, bias, tanh)

# plotting the activation function :

import matplotlib.pyplot as plt

# Input range
x = np.linspace(-10, 10, 400)

# Forward pass through neurons
y_sigmoid = [sigmoid_neuron.forward([i]) for i in x]
y_relu = [relu_neuron.forward([i]) for i in x]
y_tanh = [tanh_neuron.forward([i]) for i in x]

# Plotting
plt.figure(figsize=(10, 6))

plt.subplot(3, 1, 1)
plt.plot(x, y_sigmoid, label="Sigmoid", color="blue")
plt.title("Sigmoid Neuron Output")
plt.grid(True)
plt.legend()

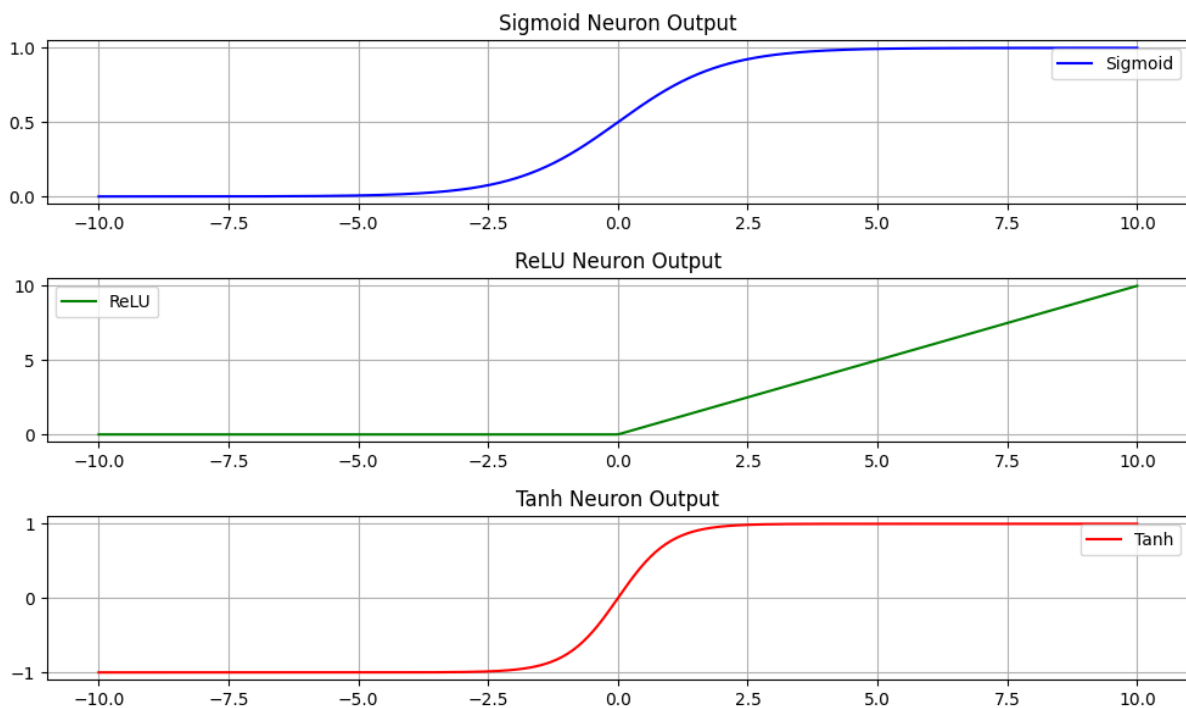
```

```
plt.subplot(3, 1, 2)
plt.plot(x, y_relu, label="ReLU", color="green")
plt.title("ReLU Neuron Output")
plt.grid(True)
plt.legend()
```

```
plt.subplot(3, 1, 3)
plt.plot(x, y_tanh, label="Tanh", color="red")
plt.title("Tanh Neuron Output")
plt.grid(True)
plt.legend()
```

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

3. Output





Name : Sanket Dahotre

Class : T.E.

Roll No : 3029

ASSIGNMENT 2

1. Title

Generate ANDNOT function using McCulloch-Pitts neural net by a python program.

2. Code

McCulloch-Pitts Neuron for AND-NOT Function

```
def mp_neuron(A, B):
```

```
    w1 = 1    # weight for A
```

```
    w2 = -1   # weight for B (NOT operation)
```

```
    threshold = 1
```

```
    net_input = A * w1 + B * w2
```

```
    if net_input >= threshold:
```

```
        return 1
```

```
    else:
```

```
        return 0
```

Testing all input combinations

```
inputs = [(0, 0), (0, 1), (1, 0), (1, 1)]
```

```
print("A B | Output (A AND NOT B)")
```

```
print("-----")
```

```
for A, B in inputs:
```

```
    output = mp_neuron(A, B)
```

```
print(A, B, "|", output)
```

3. Output

A	B		Output (A AND NOT B)
0	0		0
0	1		0
1	0		1
1	1		0



**Pune Vidyarthi Griha's
COLLEGE OF ENGINEERING, TECHNOLOGY AND
MANAGEMENT, PUNE-09**



Name : Sanket Dahotre

Class : T.E.

Roll No : 3029

ASSIGNMENT 3

1. Title

Write a Python Program using Perceptron Neural Network to recognise even and odd numbers. Given numbers are in ASCII form 0 to 9

2. Code

```
import numpy as np

# ASCII values for '0' to '9'
ascii_values = list(range(48, 58))

# Convert ASCII values to 6-bit binary
inputs = [list(map(int, format(a, '06b')) for a in ascii_values]

# Target: Even = 1, Odd = 0
targets = [1 if (a % 2 == 0) else 0 for a in ascii_values]

# Initialize weights and bias
weights = np.zeros(6)
bias = 0
learning_rate = 0.1

# Step activation function
def step(x):
    return 1 if x >= 0 else 0
```

```

# Training
for epoch in range(25):
    for x, target in zip(inputs, targets):
        net_input = np.dot(weights, x) + bias
        output = step(net_input)
        error = target - output

        weights += learning_rate * error * np.array(x)
        bias += learning_rate * error

print("Training Completed")
print("Final Weights:", weights)
print("Final Bias:", bias)

# Testing
print("\nASCII | Digit | Prediction (1=Even, 0=Odd)")
print("-----")

for a in ascii_values:
    x = list(map(int, format(a, '06b')))
    prediction = step(np.dot(weights, x) + bias)
    print(" ", a, " | ", chr(a), " | ", prediction)

```

3. Output

```
Training Completed  
Final Weights: [ 0.   0.   0.   0.1  0.  -0.2]  
Final Bias: 0.0
```

```
ASCII | Digit | Prediction (1=Even, 0=Odd)  
-----  
48 | 0 | 1  
49 | 1 | 0  
50 | 2 | 1  
51 | 3 | 0  
52 | 4 | 1  
53 | 5 | 0  
54 | 6 | 1  
55 | 7 | 0  
56 | 8 | 1  
57 | 9 | 0
```




**Pune Vidyarthi Griha's
COLLEGE OF ENGINEERING, TECHNOLOGY AND
MANAGEMENT, PUNE-09**



Name : Sanket Dahotre

Class : T.E.

Roll No : 3029

ASSIGNMENT 4

1. Title

With a suitable example demonstrate the perceptron learning law with its decision regions using python. Give the output in graphical form.

2. Code

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

```
X_or = np.array([
    [0, 0],
    [0, 1],
    [1, 0],
    [1, 1]
])
```

```
y_or = np.array([0, 1, 1, 1])
```

```
class Perceptron:
    def __init__(self, learning_rate=0.1, epochs=20):
        self.lr = learning_rate
        self.epochs = epochs
        self.weights = None
        self.bias = None
        self.errors_per_epoch = []
```

```

def predict(self, X):
    linear_output = np.dot(X, self.weights) + self.bias
    return np.where(linear_output >= 0, 1, 0)

def fit(self, X, y):
    n_samples, n_features = X.shape
    self.weights = np.zeros(n_features)
    self.bias = 0.0

    for _ in range(self.epochs):
        errors = 0
        for xi, target in zip(X, y):
            linear_output = np.dot(xi, self.weights) + self.bias
            y_pred = 1 if linear_output >= 0 else 0

            update = self.lr * (target - y_pred)
            self.weights += update * xi
            self.bias += update

            errors += int(update != 0)

        self.errors_per_epoch.append(errors)

p_or = Perceptron(learning_rate=0.1, epochs=20)
p_or.fit(X_or, y_or)

print("Weights:", p_or.weights)
print("Bias:", p_or.bias)
print("Predictions:", p_or.predict(X_or))

def plot_decision_boundary(X, y, model, title):

```

```

x_min, x_max = X[:, 0].min() - 1, X[:, 0].max() + 1
y_min, y_max = X[:, 1].min() - 1, X[:, 1].max() + 1

xx, yy = np.meshgrid(
    np.linspace(x_min, x_max, 300),
    np.linspace(y_min, y_max, 300)
)

grid = np.c_[xx.ravel(), yy.ravel()]
Z = model.predict(grid)
Z = Z.reshape(xx.shape)

plt.figure(figsize=(6, 5))
plt.contourf(xx, yy, Z, alpha=0.3, cmap="coolwarm")

for label in np.unique(y):
    pts = X[y == label]
    plt.scatter(
        pts[:, 0], pts[:, 1],
        s=100,
        edgecolor='black',
        label=f"Class {label}"
    )

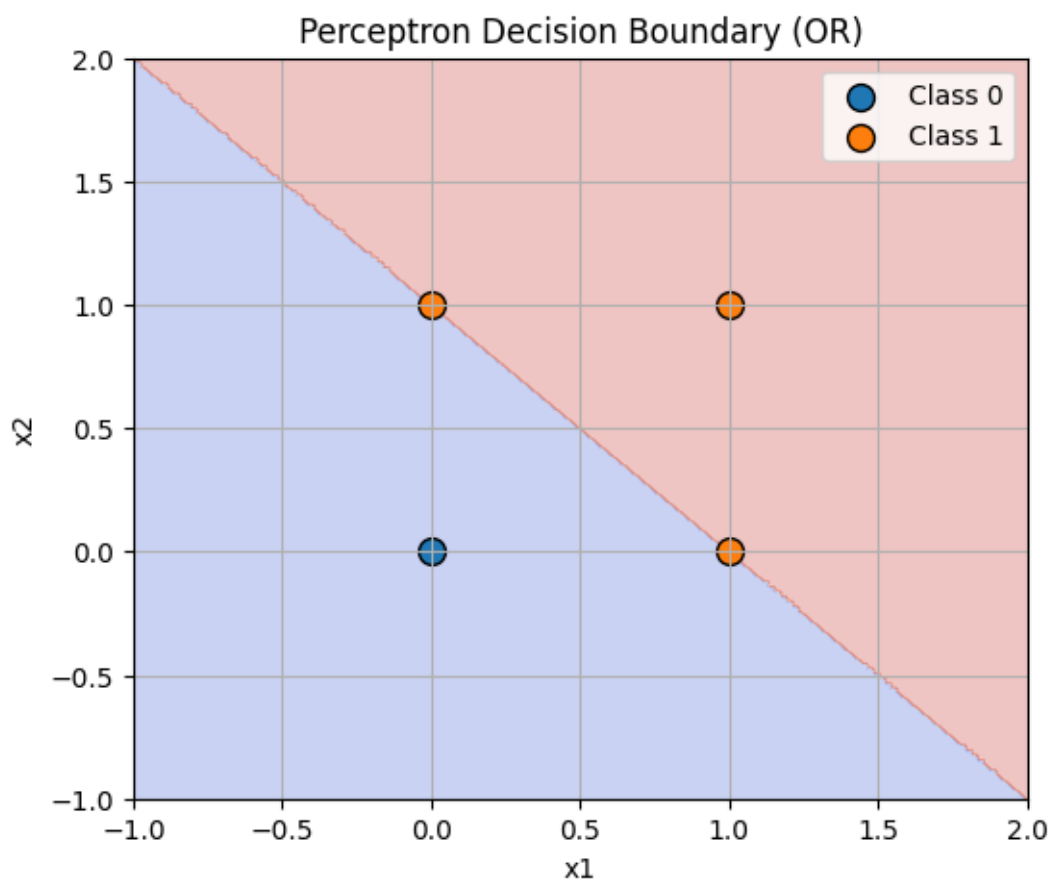
plt.title(title)
plt.xlabel("x1")
plt.ylabel("x2")
plt.legend()
plt.grid(True)
plt.show()

```

```
plot_decision_boundary(X_or, y_or, p_or, "Perceptron Decision Boundary (OR)")
```

3. Output

```
Weights: [0.1 0.1]  
Bias: -0.1  
Predictions: [0 1 1 1]
```





**Pune Vidyarthi Griha's
COLLEGE OF ENGINEERING, TECHNOLOGY AND
MANAGEMENT, PUNE-09**



Name : Sanket Dahotre

Class : T.E.

Roll No : 3029

ASSIGNMENT 5

1. Title

Implement Artificial Neural Network training process in Python by using Forward Propagation, Back Propagation.

2. Code

```
import numpy as np
```

```
# Input Dataset (XOR)
```

```
X = np.array([
    [0, 0],
    [0, 1],
    [1, 0],
    [1, 1]
])
```

```
Y = np.array([
    [0],
    [1],
    [1],
    [0]
])
```

```
# Activation Functions
```

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

```

def sigmoid_derivative(x):
    return x * (1 - x)

# Initialize Weights
np.random.seed(42)

W1 = np.random.rand(2, 2)
b1 = np.random.rand(1, 2)

W2 = np.random.rand(2, 1)
b2 = np.random.rand(1, 1)

# Training Parameters
learning_rate = 0.5
epochs = 5000

# Training Loop
for epoch in range(epochs):

    # Forward Propagation
    Z1 = np.dot(X, W1) + b1
    A1 = sigmoid(Z1)

    Z2 = np.dot(A1, W2) + b2
    A2 = sigmoid(Z2)

    # Loss (Mean Squared Error)
    error = Y - A2
    loss = np.mean(np.square(error))

    # Backpropagation

```

```

dA2 = error * sigmoid_derivative(A2)

error_hidden = dA2.dot(W2.T)
dA1 = error_hidden * sigmoid_derivative(A1)

# Update Weights
W2 += A1.T.dot(dA2) * learning_rate
b2 += np.sum(dA2, axis=0, keepdims=True) * learning_rate

W1 += X.T.dot(dA1) * learning_rate
b1 += np.sum(dA1, axis=0, keepdims=True) * learning_rate

# Print loss occasionally
if epoch % 1000 == 0:
    print(f"Epoch {epoch}, Loss: {loss:.4f}")

# Final Output
print("\nFinal Output:")
print(np.round(A2))

```

3. Output

```

Final Output:
[[0.]
 [1.]
 [1.]
 [0.]]

```




Name : Sanket Dahotre

Class : T.E.

Roll No : 3029

ASSIGNMENT 6

1. Title

Create a Neural network architecture from scratch in Python and use it to do multi-class classification on any data.

Parameters to be considered while creating the neural network from scratch are specified as:

- (1) No of hidden layers : 1 or more
- (2) No. of neurons in hidden layer: 100
- (3) Non-linearity in the layer : Relu
- (4) Use more than 1 neuron in the output layer.

Use a suitable threshold value Use appropriate Optimisation algorithm

2. Code

```
import numpy as np
from sklearn.datasets import load_iris
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler

# Load Dataset
data = load_iris()
X = data.data
y = data.target

# One-hot encoding
num_classes = len(np.unique(y))
```

```

y_onehot = np.zeros((y.shape[0], num_classes))
y_onehot[np.arange(y.shape[0]), y] = 1

# 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_onehot, test_size=0.2, random_state=42
)

# Add slight noise to avoid overfitting
X_train = X_train + 0.1 * np.random.randn(*X_train.shape)

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

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

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

# Network Architecture
input_size = X.shape[1]
hidden_size = 20 # reduced from 100 to avoid overfitting
output_size = num_classes

np.random.seed(42)

```

```

W1 = np.random.randn(input_size, hidden_size) * 0.01
b1 = np.zeros((1, hidden_size))

W2 = np.random.randn(hidden_size, output_size) * 0.01
b2 = np.zeros((1, output_size))

# Training Parameters
learning_rate = 0.02
epochs = 20

# Training Loop
for epoch in range(epochs):

    # Forward
    Z1 = np.dot(X_train, W1) + b1
    A1 = relu(Z1)

    Z2 = np.dot(A1, W2) + b2
    A2 = softmax(Z2)

    # Loss
    loss = -np.mean(y_train * np.log(A2 + 1e-8))

    # Backprop
    dZ2 = A2 - y_train
    dW2 = np.dot(A1.T, dZ2)
    db2 = np.sum(dZ2, axis=0, keepdims=True)

    dA1 = np.dot(dZ2, W2.T)
    dZ1 = dA1 * relu_derivative(Z1)
    dW1 = np.dot(X_train.T, dZ1)

```

```

db1 = np.sum(dZ1, axis=0, keepdims=True)

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

print(f"Epoch {epoch+1}, Loss: {loss:.4f}")

# Prediction
def predict(X):
    Z1 = np.dot(X, W1) + b1
    A1 = relu(Z1)
    Z2 = np.dot(A1, W2) + b2
    A2 = softmax(Z2)
    return np.argmax(A2, axis=1)

# Accuracy
y_pred = predict(X_test)
y_true = np.argmax(y_test, axis=1)

accuracy = np.mean(y_pred == y_true)
print("\nFinal Accuracy:", accuracy)

```

3. Output

```
Epoch 1, Loss: 0.3661
Epoch 2, Loss: 0.3651
Epoch 3, Loss: 0.3599
Epoch 4, Loss: 0.3298
Epoch 5, Loss: 0.2411
Epoch 6, Loss: 0.1792
Epoch 7, Loss: 0.1457
Epoch 8, Loss: 0.1234
Epoch 9, Loss: 0.1078
Epoch 10, Loss: 0.0978
Epoch 11, Loss: 0.0977
Epoch 12, Loss: 0.1195
Epoch 13, Loss: 0.1854
Epoch 14, Loss: 0.2024
Epoch 15, Loss: 0.1094
Epoch 16, Loss: 0.0857
Epoch 17, Loss: 0.0709
Epoch 18, Loss: 0.0599
Epoch 19, Loss: 0.0532
Epoch 20, Loss: 0.0480

Final Accuracy: 0.9666666666666667
```




**Pune Vidyarthi Griha's
COLLEGE OF ENGINEERING, TECHNOLOGY AND
MANAGEMENT, PUNE-09**



Name : Sanket Dahotre

Class : T.E.

Roll No : 3029

ASSIGNMENT 7

1. Title

Write a python program to show Back Propagation Network for XOR function with Binary Input and Output

2. Code

```
import numpy as np
```

```
# Input (XOR)
```

```
X = np.array([  
    [0, 0],  
    [0, 1],  
    [1, 0],  
    [1, 1]  
])
```

```
# Output
```

```
Y = np.array([  
    [0],  
    [1],  
    [1],  
    [0]  
])
```

```
# Activation Functions
```

```
def sigmoid(x):
```

```

    return 1 / (1 + np.exp(-x))

def sigmoid_derivative(x):
    return x * (1 - x)

# Initialize Weights
np.random.seed(1)

W1 = np.random.rand(2, 2) # input → hidden
b1 = np.random.rand(1, 2)

W2 = np.random.rand(2, 1) # hidden → output
b2 = np.random.rand(1, 1)

# Training Parameters
learning_rate = 0.5
epochs = 10000

# Training
for epoch in range(epochs):

    # Forward Propagation
    Z1 = np.dot(X, W1) + b1
    A1 = sigmoid(Z1)

    Z2 = np.dot(A1, W2) + b2
    A2 = sigmoid(Z2)

    # Error
    error = Y - A2

    # Backpropagation

```

```

dA2 = error * sigmoid_derivative(A2)

error_hidden = dA2.dot(W2.T)
dA1 = error_hidden * sigmoid_derivative(A1)

# Update weights and biases
W2 += A1.T.dot(dA2) * learning_rate
b2 += np.sum(dA2, axis=0, keepdims=True) * learning_rate

W1 += X.T.dot(dA1) * learning_rate
b1 += np.sum(dA1, axis=0, keepdims=True) * learning_rate

# Output
print("Final Output:")
print(np.round(A2))

```

3. Output

```

Final Output:
[[0.]
 [1.]
 [1.]
 [0.]]

```




Name : Sanket Dahotre

Class : T.E.

Roll No : 3029

ASSIGNMENT 8

1. Title

Write a python program to illustrate ART neural network.

2. Code

```
import numpy as np

# Parameters
num_features = 4
num_clusters = 3
vigilance = 0.6 # similarity threshold

# Input Patterns (Binary)
X = np.array([
    [1, 1, 0, 0],
    [1, 1, 0, 1],
    [0, 0, 1, 1],
    [0, 0, 1, 0]
])

# Initialize Weights
# Bottom-up weights
W = np.ones((num_clusters, num_features)) / (1 + num_features)
```

```

# Top-down weights
T = np.ones((num_clusters, num_features))

# ART1 Algorithm
def art1(X, W, T, vigilance):
    clusters = []

    for i, x in enumerate(X):
        print(f"\nInput Pattern {i+1}: {x}")

        while True:
            # Compute matching scores
            net = np.dot(W, x)

            # Choose best matching neuron
            j = np.argmax(net)

            # Compute match (similarity)
            match = np.sum(np.minimum(x, T[j])) / np.sum(x)

            if match >= vigilance:
                print(f"Assigned to cluster {j}")

                # Update weights
                T[j] = np.minimum(x, T[j])
                W[j] = T[j] / (0.5 + np.sum(T[j]))

                clusters.append(j)
                break
            else:
                # Reset this neuron and try next
                W[j] = -1

```

```

        if np.all(W == -1):
            print("No suitable cluster found")
            clusters.append(-1)
            break

    return clusters

# ----- Run ART1 -----
cluster_result = art1(X, W.copy(), T.copy(), vigilance)

print("\nFinal Cluster Assignments:", cluster_result)

```

3. Output

```

Input Pattern 1: [1 1 0 0]
Assigned to cluster 0

Input Pattern 2: [1 1 0 1]
Assigned to cluster 0

Input Pattern 3: [0 0 1 1]
Assigned to cluster 1

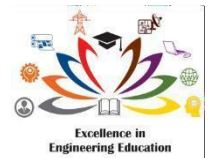
Input Pattern 4: [0 0 1 0]
Assigned to cluster 1

Final Cluster Assignments: [np.int64(0), np.int64(0), np.int64(1), np.int64(1)]

```




**Pune Vidyarthi Griha's
COLLEGE OF ENGINEERING, TECHNOLOGY AND
MANAGEMENT, PUNE-09**



Name : Sanket Dahotre

Class : T.E.

Roll No : 3029

ASSIGNMENT 9

1. Title

Write a python program in python program for creating a Back Propagation Feed-forward neural network

2. Code

```
import numpy as np
```

```
# Input Dataset (XOR Example)
```

```
X = np.array([  
    [0, 0],  
    [0, 1],  
    [1, 0],  
    [1, 1]  
])
```

```
Y = np.array([  
    [0],  
    [1],  
    [1],  
    [0]  
])
```

```
# Activation Functions
```

```

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

def sigmoid_derivative(x):
    return x * (1 - x)

# Initialize Network
np.random.seed(42)

input_neurons = 2
hidden_neurons = 3
output_neurons = 1

W1 = np.random.rand(input_neurons, hidden_neurons)
b1 = np.random.rand(1, hidden_neurons)

W2 = np.random.rand(hidden_neurons, output_neurons)
b2 = np.random.rand(1, output_neurons)

# Training Parameters
learning_rate = 0.5
epochs = 10000

# Training (Feedforward + Backpropagation)
for epoch in range(epochs):

    # Feed Forward
    Z1 = np.dot(X, W1) + b1
    A1 = sigmoid(Z1)

    Z2 = np.dot(A1, W2) + b2
    A2 = sigmoid(Z2)

```

```

# Error
error = Y - A2

# Backpropagation
dA2 = error * sigmoid_derivative(A2)

error_hidden = dA2.dot(W2.T)
dA1 = error_hidden * sigmoid_derivative(A1)

# Update Weights
W2 += A1.T.dot(dA2) * learning_rate
b2 += np.sum(dA2, axis=0, keepdims=True) * learning_rate

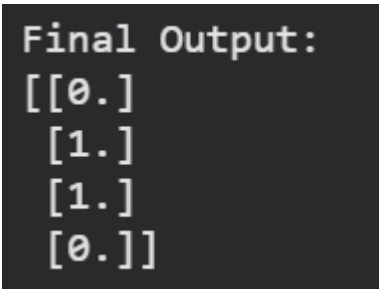
W1 += X.T.dot(dA1) * learning_rate
b1 += np.sum(dA1, axis=0, keepdims=True) * learning_rate

# Print Loss Occasionally
if epoch % 2000 == 0:
    loss = np.mean(np.square(error))
    print(f"Epoch {epoch}, Loss: {loss:.4f}")

# Final Output
print("\nFinal Output:")
print(np.round(A2))

```

3. Output



```

Final Output:
[[0.]
 [1.]
 [1.]
 [0.]]

```




Name : Sanket Dahotre

Class : T.E.

Roll No : 3029

ASSIGNMENT 10

1. Title

Write a python program to design a Hopfield Network which stores 4 vectors

2. Code

```
import numpy as np

# Sign Function
def sign(x):
    return np.where(x >= 0, 1, -1)

# Stored Patterns (Bipolar: -1, +1)
patterns = np.array([
    [1, -1, 1, -1],
    [-1, 1, -1, 1],
    [1, 1, -1, -1],
    [-1, -1, 1, 1]
])

# Training (Hebbian Learning)
n = patterns.shape[1]
W = np.zeros((n, n))
```

```

for p in patterns:
    W += np.outer(p, p)

# Remove self-connections
np.fill_diagonal(W, 0)

print("Weight Matrix W:\n", W)

# Recall Function
def recall(input_pattern, W, steps=5):
    x = input_pattern.copy()

    for _ in range(steps):
        x = sign(np.dot(W, x)) # synchronous update

    return x

# Test with Noisy Input
test_pattern = np.array([1, -1, -1, -1]) # noisy version

recalled = recall(test_pattern, W)

print("\nNoisy Input:", test_pattern)
print("Recalled Pattern:", recalled)

```

3. Output

```

Weight Matrix W:
[[ 0.  0.  0. -4.]
 [ 0.  0. -4.  0.]
 [ 0. -4.  0.  0.]
 [-4.  0.  0.  0.]]

Noisy Input: [ 1 -1 -1 -1]
Recalled Pattern: [ 1  1  1 -1]

```



Name : Sanket Dahotre

Class : T.E.

Roll No : 3029

ASSIGNMENT 11

1. Title

How to Train a Neural Network with TensorFlow and evaluation of logistic regression using tensorflow

2. Code

```
import tensorflow as tf  
from sklearn.datasets import load_iris  
from sklearn.model_selection import train_test_split  
from sklearn.preprocessing import StandardScaler
```

```
# Load Dataset
```

```
data = load_iris()
```

```
X, y = data.data, data.target
```

```
# Preprocessing
```

```
scaler = StandardScaler()
```

```
X = scaler.fit_transform(X)
```

```
X_train, X_test, y_train, y_test = train_test_split(
```

```
    X, y, test_size=0.2, random_state=42
```

```
)
```

```
# 1. Neural Network Model
```

```
nn_model = tf.keras.Sequential([
```

```

    tf.keras.layers.Dense(100, activation='relu', input_shape=(4,)),
    tf.keras.layers.Dense(3, activation='softmax')
])

nn_model.compile(
    optimizer='adam',
    loss='sparse_categorical_crossentropy',
    metrics=['accuracy']
)

# Train NN
nn_model.fit(X_train, y_train, epochs=50, verbose=0)

# Evaluate NN
nn_loss, nn_acc = nn_model.evaluate(X_test, y_test, verbose=0)

# 2. Logistic Regression Model
log_model = tf.keras.Sequential([
    tf.keras.layers.Dense(3, activation='softmax', input_shape=(4,))
])

log_model.compile(
    optimizer='adam',
    loss='sparse_categorical_crossentropy',
    metrics=['accuracy']
)

# Train Logistic Regression
log_model.fit(X_train, y_train, epochs=50, verbose=0)

# Evaluate Logistic Regression
log_loss, log_acc = log_model.evaluate(X_test, y_test, verbose=0)

```

Results

```
print("Neural Network Accuracy :", nn_acc)
```

```
print("Logistic Regression Accuracy :", log_acc)
```

3. Output

```
Neural Network Accuracy : 0.9666666388511658  
Logistic Regression Accuracy : 0.06666667014360428
```



Name : Sanket Dahotre

Class : T.E.

Roll No : 3029

ASSIGNMENT 12

1. Title

Pytorch implementation of CNN

2. Code

```
import torch
import torch.nn as nn
import torch.optim as optim
from torchvision import datasets, transforms
from torch.utils.data import DataLoader

# Load dataset
transform = transforms.Compose([
    transforms.ToTensor()
])

train_data = datasets.MNIST(root='./data', train=True, download=True,
transform=transform)

test_data = datasets.MNIST(root='./data', train=False, download=True,
transform=transform)

train_loader = DataLoader(train_data, batch_size=64, shuffle=True)
test_loader = DataLoader(test_data, batch_size=64)

# Define CNN
class CNN(nn.Module):
    def __init__(self):
```

```

super().__init__()
self.conv1 = nn.Conv2d(1, 32, 3)
self.pool = nn.MaxPool2d(2, 2)
self.conv2 = nn.Conv2d(32, 64, 3)

self.fc1 = nn.Linear(64 * 5 * 5, 100)
self.fc2 = nn.Linear(100, 10)

def forward(self, x):
    x = self.pool(torch.relu(self.conv1(x)))
    x = self.pool(torch.relu(self.conv2(x)))

    x = x.view(-1, 64 * 5 * 5)
    x = torch.relu(self.fc1(x))
    x = self.fc2(x)
    return x

model = CNN()

criterion = nn.CrossEntropyLoss()
optimizer = optim.Adam(model.parameters(), lr=0.001)

# Training
for epoch in range(5):
    for images, labels in train_loader:
        optimizer.zero_grad()
        outputs = model(images)
        loss = criterion(outputs, labels)
        loss.backward()
        optimizer.step()

# Evaluation

```

total = 0

for images, labels in test_loader:

```
_, predicted = torch.max(outputs, 1)
```

```
correct += (predicted == labels).sum().item()
```

3. Output

```
(EDU) PS C:\Users\Divesh\OneDrive\Desktop\New folder> python .\speed.py
Downloading http://yann.lecun.com/exdb/mnist/train-images-idx3-ubyte.gz
Failed to download (trying next):
HTTP Error 404: Not Found

Downloading https://oss-ci-datasets.s3.amazonaws.com/mnist/train-images-idx3-ubyte.gz
Downloading https://oss-ci-datasets.s3.amazonaws.com/mnist/train-images-idx3-ubyte.gz to ./data\MNIST\raw\train-images-idx3-ubyte.gz
100%|██████████████████████████████████████████████████████████████████████████████| 9.91M/9.91M [00:17<00:00, 560kB/s]
Extracting ./data\MNIST\raw\train-images-idx3-ubyte.gz to ./data\MNIST\raw

Downloading http://yann.lecun.com/exdb/mnist/train-labels-idx1-ubyte.gz
Failed to download (trying next):
HTTP Error 404: Not Found

Downloading https://oss-ci-datasets.s3.amazonaws.com/mnist/train-labels-idx1-ubyte.gz
Downloading https://oss-ci-datasets.s3.amazonaws.com/mnist/train-labels-idx1-ubyte.gz to ./data\MNIST\raw\train-labels-idx1-ubyte.gz
100%|██████████████████████████████████████████████████████████████████████████████| 28.9k/28.9k [00:00<00:00, 84.8kB/s]
Extracting ./data\MNIST\raw\train-labels-idx1-ubyte.gz to ./data\MNIST\raw

Downloading http://yann.lecun.com/exdb/mnist/t10k-images-idx3-ubyte.gz
Failed to download (trying next):
HTTP Error 404: Not Found

Downloading https://oss-ci-datasets.s3.amazonaws.com/mnist/t10k-images-idx3-ubyte.gz
Downloading https://oss-ci-datasets.s3.amazonaws.com/mnist/t10k-images-idx3-ubyte.gz to ./data\MNIST\raw\t10k-images-idx3-ubyte.gz
100%|██████████████████████████████████████████████████████████████████████████████| 1.65M/1.65M [00:03<00:00, 469kB/s]
Extracting ./data\MNIST\raw\t10k-images-idx3-ubyte.gz to ./data\MNIST\raw

Downloading http://yann.lecun.com/exdb/mnist/t10k-labels-idx1-ubyte.gz
Failed to download (trying next):
HTTP Error 404: Not Found

Downloading https://oss-ci-datasets.s3.amazonaws.com/mnist/t10k-labels-idx1-ubyte.gz
Downloading https://oss-ci-datasets.s3.amazonaws.com/mnist/t10k-labels-idx1-ubyte.gz to ./data\MNIST\raw\t10k-labels-idx1-ubyte.gz
100%|██████████████████████████████████████████████████████████████████████████████| 4.54k/4.54k [00:00<00:00, 1.52MB/s]
Extracting ./data\MNIST\raw\t10k-labels-idx1-ubyte.gz to ./data\MNIST\raw

PyTorch CNN Accuracy: 0.991
(EDU) PS C:\Users\Divesh\OneDrive\Desktop\New folder>
```



Name : Sanket Dahotre

Class : T.E.

Roll No : 3029

ASSIGNMENT 13

1. Title

MNIST Handwritten Character Detection using PyTorch

2. Code

```
import torch
import torch.nn as nn
import torch.optim as optim
from torchvision import datasets, transforms
from torch.utils.data import DataLoader

# Load dataset
transform = transforms.ToTensor()

train_data = datasets.MNIST('./data', train=True, download=True,
transform=transform)
test_data = datasets.MNIST('./data', train=False, download=True,
transform=transform)

train_loader = DataLoader(train_data, batch_size=64, shuffle=True)
test_loader = DataLoader(test_data, batch_size=64)

# CNN Model
class CNN(nn.Module):
    def __init__(self):
        super().__init__()
        self.conv1 = nn.Conv2d(1, 32, 3)
```

```

self.pool = nn.MaxPool2d(2,2)
self.conv2 = nn.Conv2d(32, 64, 3)

self.fc1 = nn.Linear(64*5*5, 100)
self.fc2 = nn.Linear(100, 10)

def forward(self, x):
    x = self.pool(torch.relu(self.conv1(x)))
    x = self.pool(torch.relu(self.conv2(x)))

    x = x.view(-1, 64*5*5)
    x = torch.relu(self.fc1(x))
    return self.fc2(x)

model = CNN()

criterion = nn.CrossEntropyLoss()
optimizer = optim.Adam(model.parameters(), lr=0.001)

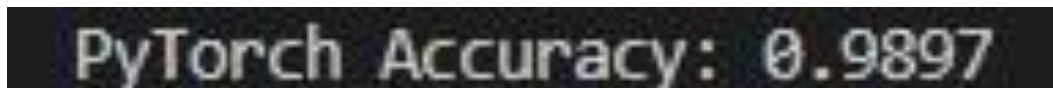
# Training
for epoch in range(5):
    for images, labels in train_loader:
        optimizer.zero_grad()
        outputs = model(images)
        loss = criterion(outputs, labels)
        loss.backward()
        optimizer.step()

# Evaluation
correct = 0
total = 0

```

```
with torch.no_grad():  
    for images, labels in test_loader:  
        outputs = model(images)  
        _, predicted = torch.max(outputs, 1)  
        total += labels.size(0)  
        correct += (predicted == labels).sum().item()  
  
print("PyTorch Accuracy:", correct / total)
```

3. Output



```
PyTorch Accuracy: 0.9897
```