J.N.T.U.H. UNIVERSITY COLLEGE OF ENGINEERING, SCIENCE AND TECHNOLOGY HYDERABAD

KUKATPALLY, HYDERABAD – 500085

CERTIFICATE

This is to certify that **KURAPATI TOYESH** of B.Tech III year II Semester bearing the Hall-Ticket number **21011A0524** has fulfilled his/her DEEP LEARNING LAB record for the academic year 2023-2024.

Signature of the Head of the Department   Signature of the staff member

Date of Examination: 11-06-2024

Internal Examiner   External Examiner

TABLE OF CONTENTS

1. Logic Gates using Perceptron – OR gate

```
def activate(x):
    return 1 if x >= 0 else 0

def perceptron(inputs):
    w1, w2, b = 0, 0, 0
    desired_outputs = [0, 1, 1, 1]
    learning_rate = 0.1
    epochs = 100

    for epoch in range(epochs):
        total_error = 0
        for i in range(len(inputs)):
            A, B = inputs[i]
            target_output = desired_outputs[i]
            output = activate(w1 * A + w2 * B + b)
            error = target_output - output
            w1 += learning_rate * error * A
            w2 += learning_rate * error * B
            b += learning_rate * error
            total_error += abs(error)
        if total_error == 0:
            break
```

```
    if total_error == 0:
        return w1, w2, b

inputs = [(0, 0), (0, 1), (1, 0), (1, 1)]
w1, w2, b = perceptron(inputs)

print("Weights:", w1, w2)
print("Bias:", b)
for i in range(len(inputs)):
    A, B = inputs[i]
    output = activate(w1 * A + w2 * B + b)
    print("Input:", A, B, "Output:", output)
```

OUTPUT:

```
Weights: 0.1 0.1
Bias: -0.1
Input: 0 0 Output: 0
Input: 0 1 Output: 1
Input: 1 0 Output: 1
Input: 1 1 Output: 1
```

2. Logic Gates using Perceptron – AND gate

```
def activate(x):
    return 1 if x >= 0 else 0


def perceptron(inputs):
    w1, w2, b = 0, 0, 0
    desired_outputs = [0, 0, 0, 1]
    learning_rate = 0.1
    epochs = 100

    for epoch in range(epochs):
        total_error = 0
        for i in range(len(inputs)):
            A, B = inputs[i]
            target_output = desired_outputs[i]
            output = activate(w1 * A + w2 * B + b)
            error = target_output - output
            w1 += learning_rate * error * A
            w2 += learning_rate * error * B
            b += learning_rate * error
            total_error += abs(error)
```

```python
        if total_error == 0:
            break

    if total_error == 0:
        return w1, w2, b

inputs = [(0, 0), (0, 1), (1, 0), (1, 1)]
w1, w2, b = perceptron(inputs)

print("Weights:", w1, w2)
print("Bias:", b)
for i in range(len(inputs)):
    A, B = inputs[i]
    output = activate(w1 * A + w2 * B + b)
    print("Input:", A, B, "Output:", output)
```

OUTPUT:

```
Weights: 0.2 0.1
Bias: -0.20000000000000004
Input: 0 0 Output: 0
Input: 0 1 Output: 0
Input: 1 0 Output: 0
Input: 1 1 Output: 1
```

3. Logic Gates using Perceptron – NOR gate

```
def activate(x):
    return 1 if x >= 0 else 0


def perceptron(inputs):
    w1, w2, b = 0, 0, 0
    desired_outputs = [1, 0, 0, 0]
    learning_rate = 0.1
    epochs = 100


    for epoch in range(epochs):
        total_error = 0
        for i in range(len(inputs)):
            A, B = inputs[i]
            target_output = desired_outputs[i]
            output = activate(w1 * A + w2 * B + b)
            error = target_output - output
            w1 += learning_rate * error * A
            w2 += learning_rate * error * B
            b += learning_rate * error
            total_error += abs(error)
        if total_error == 0:
```

```
            break

    if total_error == 0:
        return w1, w2, b

inputs = [(0, 0), (0, 1), (1, 0), (1, 1)]
w1, w2, b = perceptron(inputs)

print("Weights:", w1, w2)
print("Bias:", b)
for i in range(len(inputs)):
    A, B = inputs[i]
    output = activate(w1 * A + w2 * B + b)
    print("Input:", A, B, "Output:", output)
```

OUTPUT:

```
Weights: -0.1 -0.1
Bias: 0.0
Input: 0 0 Output: 1
Input: 0 1 Output: 0
Input: 1 0 Output: 0
Input: 1 1 Output: 0
```

4. Logic Gates using Perceptron – NAND gate

```
def activate(x):
    return 1 if x >= 0 else 0


def perceptron(inputs):
    w1, w2, b = 0, 0, 0
    desired_outputs = [0, 1, 1, 1] #[0, 0, 0, 1] #[1, 0, 0, 0] #[1, 1, 1, 0]
    learning_rate = 0.1
    epochs = 100

    for epoch in range(epochs):
        total_error = 0
        for i in range(len(inputs)):
            A, B = inputs[i]
            target_output = desired_outputs[i]
            output = activate(w1 * A + w2 * B + b)
            error = target_output - output
            w1 += learning_rate * error * A
            w2 += learning_rate * error * B
            b += learning_rate * error
            total_error += abs(error)
        if total_error == 0:
```

```
            break

    if total_error == 0:
        return w1, w2, b

inputs = [(0, 0), (0, 1), (1, 0), (1, 1)]
w1, w2, b = perceptron(inputs)

print("Weights:", w1, w2)
print("Bias:", b)
for i in range(len(inputs)):
    A, B = inputs[i]
    output = activate(w1 * A + w2 * B + b)
    print("Input:", A, B, "Output:", output)
```

OUTPUT:

```
Weights: -0.2 -0.1
Bias: 0.2
Input: 0 0 Output: 1
Input: 0 1 Output: 1
Input: 1 0 Output: 1
Input: 1 1 Output: 0
```

5. Logic Gates using Perceptron – NOT gate

```
def activate(x):
    return 1 if x >= 0 else 0


def perceptron(inputs):
    w1, b = 0, -1
    desired_outputs = [1, 0]
    learning_rate = 0.1
    epochs = 100

    for epoch in range(epochs):
        total_error = 0
        for i in range(len(inputs)):
            A = inputs[i]
            target_output = desired_outputs[i]
            output = activate(w1 * A + b)
            error = target_output - output
            w1 += learning_rate * error * A
            b += learning_rate * error
            total_error += abs(error)
        if total_error == 0:
            break
```

```
        if total_error == 0:
            return w1, b

inputs = [0, 1]
w1, b = perceptron(inputs)


print("NOT Gate Output:")
print("Weight:", w1)
print("Bias:", b)
for i in range(len(inputs)):
    A = inputs[i]
    output = activate(w1 * A + b)
    print("Input:", A, "Output:", output)
```

OUTPUT:

```
NOT Gate Output:
Weight: -0.1
Bias: 0.09999999999999987
Input: 0 Output: 1
Input: 1 Output: 0
```

6. Logic Gates using Perceptron – XOR gate

```
import numpy as np
# define Unit Step Function
def unitStep(v):
  if v >= 0:
    return 1
  else:
    return 0

# design Perceptron Model
def perceptronModel(x, w, b):
  v = np.dot(w, x) + b
  y = unitStep(v)
  return y

# NOT Logic Function
# wNOT = -1, bNOT = 0.5
def NOT_logicFunction(x):
  wNOT = -1
  bNOT = 0.5
  return perceptronModel(x, wNOT, bNOT)
```

```
# AND Logic Function
# here w1 = wAND1 = 1,
# w2 = wAND2 = 1, bAND = -1.5
def AND_logicFunction(x):
  w = np.array([1, 1])
  bAND = -1.5
  return perceptronModel(x, w, bAND)

# OR Logic Function
# w1 = 1, w2 = 1, bOR = -0.5
def OR_logicFunction(x):
  w = np.array([1, 1])
  bOR = -0.5
  return perceptronModel(x, w, bOR)

# XOR Logic Function
# with AND, OR and NOT
# function calls in sequence
def XOR_logicFunction(x):
  y1 = AND_logicFunction(x)
  y2 = OR_logicFunction(x)
  y3 = NOT_logicFunction(y1)
  final_x = np.array([y2, y3])
```

```
    finalOutput = AND_logicFunction(final_x)
    return finalOutput

# testing the Perceptron Model
test1 = np.array([0, 1])
test2 = np.array([1, 1])
test3 = np.array([0, 0])
test4 = np.array([1, 0])

print("XOR({}, {}) = {}".format(0, 1, XOR_logicFunction(test1)))
print("XOR({}, {}) = {}".format(1, 1, XOR_logicFunction(test2)))
print("XOR({}, {}) = {}".format(0, 0, XOR_logicFunction(test3)))
print("XOR({}, {}) = {}".format(1, 0, XOR_logicFunction(test4)))
```

OUTPUT:

```
XOR(0, 1) = 1
XOR(1, 1) = 0
XOR(0, 0) = 0
XOR(1, 0) = 1
```

7. ADALINE Neural Network

```
import numpy as np


def Adaline(Input, Target, lr=0.2, stop=0.001):
    weight = np.random.random(Input.shape[1])
    bias = np.random.random(1)[0]  # Extract scalar from array


    Error = [stop + 1]
    # Check the stop condition for the network
    while Error[-1] > stop or Error[-1] - Error[-2] > 0.0001:
        error = []
        for i in range(Input.shape[0]):
            Y_input = np.dot(weight, Input[i]) + bias


            # Update the weight
            for j in range(Input.shape[1]):
                weight[j] = weight[j] + lr * (Target[i] - Y_input) * Input[i][j]


            # Update the bias
            bias = bias + lr * (Target[i] - Y_input)
```

```
            # Store squared error value
            error.append((Target[i] - Y_input) ** 2)

        # Store sum of squared errors
        Error.append(sum(error))

    return weight, bias

# Input dataset
x = np.array([[1.0, 1.0, 1.0],
              [1.0, -1.0, 1.0],
              [-1.0, 1.0, 1.0],
              [-1.0, -1.0, -1.0]])

# Target values
t = np.array([1, 1, 1, -1])

# Train the Adaline model
w, b = Adaline(x, t, lr=0.2, stop=0.001)

# Print the final weights and bias
print('Weights:', w)
print('Bias:', b)
```

```python
# Predict outputs
predicted_outputs = []
for i in range(x.shape[0]):
    predicted_output = np.dot(w, x[i]) + b
    predicted_outputs.append(predicted_output)

# Display inputs and predicted outputs
for i in range(x.shape[0]):
    print("Input:", x[i][0], x[i][1], "Output:", predicted_outputs[i])
```

OUTPUT:

```
Weights: [0.00916354 0.00916354 0.988777  ]
Bias: 0.009163537511307641
Input: 1.0 1.0 Output: 1.016267616963145
Input: 1.0 -1.0 Output: 0.9979405419405291
Input: -1.0 1.0 Output: 0.9979405419405288
Input: -1.0 -1.0 Output: -0.9979405419405297
```

8. MADALINE Neural Network

```
import numpy as np


# Activation function
def activation_fn(z):
    return 1 if z >= 0 else -1


def Madaline(Input, Target, lr, epoch):
    weight = np.random.random((Input.shape[1], Input.shape[1]))
    bias = np.random.random(Input.shape[1])

    w = np.array([0.5 for _ in range(weight.shape[1])])
    b = 0.5
    k = 0
    while k < epoch:
        error = []
        z_input = np.zeros(bias.shape[0])
        z = np.zeros(bias.shape[0])
        for i in range(Input.shape[0]):
            for j in range(Input.shape[1]):
                z_input[j] = sum(weight[j] * Input[i]) + bias[j]
```

```
            z[j] = activation_fn(z_input[j])

        y_input = sum(z * w) + b
        y = activation_fn(y_input)

        # Update the weight & bias
        if y != Target[i]:
            for j in range(weight.shape[1]):
                weight[j] = weight[j] + lr * (Target[i] - z_input[j]) * Input[i][j]
                bias[j] = bias[j] + lr * (Target[i] - z_input[j])

        # Store squared error value
        error.append((Target[i] - y_input) ** 2)

    # Compute sum of square error
    Error = sum(error)
    k += 1

  return weight, bias

# Prediction function
def prediction(X, w, b):
  y = []
```

```
    for i in range(X.shape[0]):
        x = X[i]
        z1 = x * w
        z_1 = []
        for j in range(z1.shape[1]):
            z_1.append(activation_fn(sum(z1[j]) + b[j]))
        y_in = sum(np.array(z_1) * np.array([0.5 for _ in range(w.shape[1])]))
+ 0.5
        y.append(activation_fn(y_in))
    return y

# Input dataset
x = np.array([[1.0, 1.0, 1.0],
              [1.0, -1.0, 1.0],
              [-1.0, 1.0, 1.0],
              [-1.0, -1.0, -1.0]])

# Target values
t = np.array([1, 1, 1, -1])

# Train the MADALINE model
w, b = Madaline(x, t, lr=0.0001, epoch=3)
```

```
# Print the final weights and bias
print('Weights:', w)
print('Bias:', b)

# Predict outputs
predicted_outputs = prediction(x, w, b)

# Display inputs and predicted outputs
for i in range(x.shape[0]):
    print("Input:", x[i][0], x[i][1], "Output:", predicted_outputs[i])
```

OUTPUT:

```
Weights: [[0.57658    0.55301413 0.42396924]
 [0.93762437 0.91234253 0.12067558]
 [0.64274761 0.60354512 0.335578  ]]
Bias: [0.98596296 0.28329168 0.31065159]
Input: 1.0 1.0 Output: 1
Input: 1.0 -1.0 Output: 1
Input: -1.0 1.0 Output: 1
Input: -1.0 -1.0 Output: -1
```

9. Image Classification using MNIST dataset

```
import tensorflow as tf

import keras

from tensorflow.keras import datasets, layers, models

import matplotlib.pyplot as plt

%matplotlib inline

import numpy as np


(X_train, y_train), (X_test, y_test) = keras.datasets.mnist.load_data()

X_train.shape, X_test.shape
```

```
Downloading data from https://storage.googleapis.com/tensorflow/tf-keras-datasets/mnist.npz
11490434/11490434 [==============================] - 0s 0us/step
((60000, 28, 28), (10000, 28, 28))
```

```
#Normalisation

X_train=X_train/255

X_test=X_test/255


cnn=models.Sequential([

    #cnn

    layers.Conv2D(filters=32,kernel_size=(3,3),activation='relu',input_shape
=(28,28,1)),
```

```
    layers.MaxPooling2D((2,2)),


    layers.Conv2D(filters=64,kernel_size=(3,3),activation='relu'),
    layers.MaxPooling2D((2,2)),


    #dense
    layers.Flatten(),
    layers.Dense(50, activation='relu'),
    layers.Dense(10, activation='softmax')
])
cnn.compile(optimizer='adam',
            loss='sparse_categorical_crossentropy',
            metrics=['accuracy'])


cnn.fit(X_train, y_train, epochs=10)
```

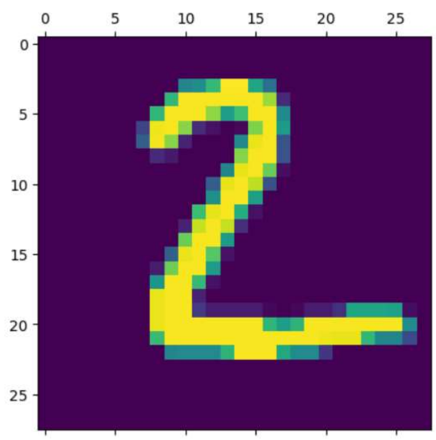
```
Epoch 1/10
1875/1875 [==============================] - 53s 28ms/step - loss: 0.1517 - accuracy: 0.9538
Epoch 2/10
1875/1875 [==============================] - 51s 27ms/step - loss: 0.0477 - accuracy: 0.9854
Epoch 3/10
1875/1875 [==============================] - 54s 29ms/step - loss: 0.0344 - accuracy: 0.9895
Epoch 4/10
1875/1875 [==============================] - 51s 27ms/step - loss: 0.0254 - accuracy: 0.9922
Epoch 5/10
1875/1875 [==============================] - 51s 27ms/step - loss: 0.0186 - accuracy: 0.9940
Epoch 6/10
1875/1875 [==============================] - 50s 27ms/step - loss: 0.0158 - accuracy: 0.9949
Epoch 7/10
1875/1875 [==============================] - 51s 27ms/step - loss: 0.0117 - accuracy: 0.9962
Epoch 8/10
1875/1875 [==============================] - 52s 28ms/step - loss: 0.0095 - accuracy: 0.9971
Epoch 9/10
1875/1875 [==============================] - 50s 27ms/step - loss: 0.0086 - accuracy: 0.9973
Epoch 10/10
1875/1875 [==============================] - 52s 28ms/step - loss: 0.0066 - accuracy: 0.9978
<keras.src.callbacks.History at 0x78a2550a1fc0>
```

```
y_pred=cnn.predict(X_test)
y_classes=[np.argmax(element) for element in y_pred]
```

```
313/313 [==============================] - 4s 12ms/step
```

```
plt.matshow(X_test[1])
print("Test Label:",y_test[1])
print("Predicted Value:",y_classes[1])
```

```
Test Label: 2
Predicted Value: 2
```



```
from sklearn.metrics import confusion_matrix, classification_report
print("Classification Report:\n",classification_report(y_test, y_classes))
```

```
Classification Report:
              precision    recall  f1-score   support

           0       0.99      1.00      0.99       980
           1       0.99      1.00      0.99      1135
           2       0.99      0.99      0.99      1032
           3       0.99      0.99      0.99      1010
           4       0.99      0.99      0.99       982
           5       0.99      0.99      0.99       892
           6       0.99      0.99      0.99       958
           7       0.99      0.98      0.98      1028
           8       0.99      0.99      0.99       974
           9       0.99      0.99      0.99      1009

    accuracy                           0.99     10000
   macro avg       0.99      0.99      0.99     10000
weighted avg       0.99      0.99      0.99     10000
```

10. Image Classification using CIFAR10 dataset

```
import tensorflow as tf

import keras

from tensorflow.keras import datasets, layers, models

import matplotlib.pyplot as plt

%matplotlib inline

import numpy as np


(X_train, y_train), (X_test, y_test)=datasets.cifar10.load_data()

X_train.shape, X_test.shape
```

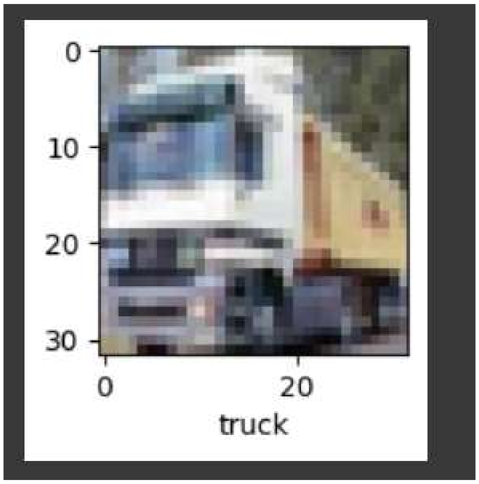
```
((50000, 32, 32, 3), (10000, 32, 32, 3))
```

```
classes=["airplane","automobile","bird","cat","deer","dog","frog","horse","shi
p","truck"]


y_train=y_train.reshape(-1,)

y_train
```

```
array([6, 9, 9, ..., 9, 1, 1], dtype=uint8)
```

```
def plot_sample(X, y,index):

  plt.figure(figsize=(15,2))
```

```
  plt.imshow(X[index])
  plt.xlabel(classes[y[index]])
plot_sample(X_train, y_train, 1)
```




```
#Normalisation
X_train=X_train/255
X_test=X_test/255

cnn=models.Sequential([
    #cnn

layers.Conv2D(filters=32,kernel_size=(3,3),activation='relu',input_shape=(3
2,32,3)),
    layers.MaxPooling2D((2,2)),

    layers.Conv2D(filters=64,kernel_size=(3,3),activation='relu'),
    layers.MaxPooling2D((2,2)),
```

```
    #dense
    layers.Flatten(),
    layers.Dense(50, activation='relu'),
    layers.Dense(10, activation='softmax')
])

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

cnn.fit(X_train, y_train, epochs=10)
```

```
Epoch 1/10
1563/1563 [==============================] - 58s 36ms/step - loss: 2.4180 - accuracy: 0.0986
Epoch 2/10
1563/1563 [==============================] - 49s 31ms/step - loss: 2.3036 - accuracy: 0.0981
Epoch 3/10
1563/1563 [==============================] - 49s 31ms/step - loss: 2.3028 - accuracy: 0.0973
Epoch 4/10
1563/1563 [==============================] - 48s 31ms/step - loss: 2.3030 - accuracy: 0.0950
Epoch 5/10
1563/1563 [==============================] - 50s 32ms/step - loss: 2.3028 - accuracy: 0.0980
Epoch 6/10
1563/1563 [==============================] - 49s 32ms/step - loss: 2.3028 - accuracy: 0.0958
Epoch 7/10
1563/1563 [==============================] - 48s 31ms/step - loss: 2.3028 - accuracy: 0.0999
Epoch 8/10
1563/1563 [==============================] - 49s 32ms/step - loss: 2.3028 - accuracy: 0.0999
Epoch 9/10
1563/1563 [==============================] - 48s 31ms/step - loss: 2.3028 - accuracy: 0.0985
Epoch 10/10
1563/1563 [==============================] - 47s 30ms/step - loss: 2.3028 - accuracy: 0.0985
<keras.src.callbacks.History at 0x7af8d054a6e0>
```

```
y_pred=cnn.predict(X_test)
y_classes=[np.argmax(element) for element in y_pred]
```

```
313/313 [==============================] - 4s 12ms/step
```

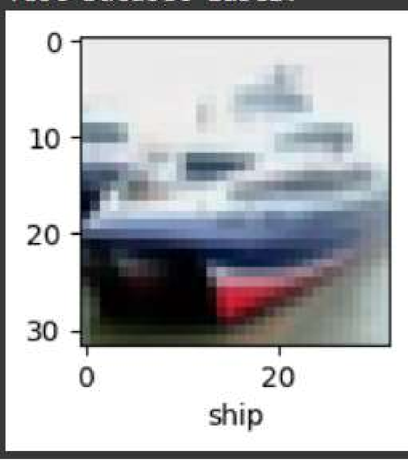
y_test=y_test.reshape(-1,)

print("Calculated Class:",classes[y_classes[1]])

print("Test Dataset Label:")

plot_sample(X_test, y_test, 1)

```
Calculated Class: ship
Test Dataset Label:
```



from sklearn.metrics import confusion_matrix, classification_report

print("Classification Report: \n",classification_report(y_test, y_classes))

```
Classification Report:
              precision    recall  f1-score   support

           0       0.75      0.73      0.74      1000
           1       0.82      0.84      0.83      1000
           2       0.65      0.51      0.57      1000
           3       0.54      0.47      0.50      1000
           4       0.58      0.69      0.63      1000
           5       0.56      0.68      0.62      1000
           6       0.75      0.79      0.77      1000
           7       0.75      0.75      0.75      1000
           8       0.81      0.81      0.81      1000
           9       0.85      0.73      0.79      1000

    accuracy                           0.70     10000
   macro avg       0.70      0.70      0.70     10000
weighted avg       0.70      0.70      0.70     10000
```