EXPERIMENT – 7

CREATE A SIMPLE PYTHON PROGRAM THAT CREATES AN ENCODER FOR AN AUTOENCODER

Aim: - The aim of this program is to create a simple python program that creates an encoder for an autoencoder.

Procedure: -

```
import numpy as np

import tensorflow as tf

from tensorflow.keras import layers, models

import matplotlib.pyplot as plt


# Load the MNIST dataset

(x_train, _), (x_test, _) = tf.keras.datasets.mnist.load_data()


# Normalize pixel values to the range [0, 1]

x_train = x_train.astype('float32') / 255.0

x_test = x_test.astype('float32') / 255.0


# Flatten the images

x_train = x_train.reshape((len(x_train), np.prod(x_train.shape[1:])))

x_test = x_test.reshape((len(x_test), np.prod(x_test.shape[1:])))


# Build the autoencoder model with an encoder and decoder

encoding_dim = 32  # Number of neurons in the bottleneck layer


# Input layer

input_img = tf.keras.Input(shape=(784,))


# Encoder
```

```
encoded = layers.Dense(encoding_dim, activation='relu')(input_img)

# Decoder (not used in this example, but included for completeness)
decoded = layers.Dense(784, activation='sigmoid')(encoded)

# Autoencoder model
autoencoder = models.Model(input_img, decoded)

# Encoder model (only includes the encoder part)
encoder = models.Model(input_img, encoded)

# Compile the autoencoder (not used in this example, but included for completeness)
autoencoder.compile(optimizer='adam', loss='binary_crossentropy')

# Display the encoder model summary
encoder.summary()
```

Output: -

```
Model: "model_1"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
input_1 (InputLayer)         [(None, 784)]             0
_________________________________________________________________
dense_6 (Dense)              (None, 32)                25120
=================================================================
Total params: 25,120
Trainable params: 25,120
Non-trainable params: 0
_________________________________________________________________
```

Result: - Program executed successfully.

EXPERIMENT – 8

WRITE A PYTHON PROGRAM THAT CREATES A DECODER FOR AN AUTOENCODER

Aim: - The aim of this program is to create a python program that creates a decoder for an autoencoder.

Procedure: -

```
import numpy as np

import tensorflow as tf

from tensorflow.keras import layers, models

import matplotlib.pyplot as plt


# Load the MNIST dataset

(x_train, _), (x_test, _) = tf.keras.datasets.mnist.load_data()


# Normalize pixel values to the range [0, 1]

x_train = x_train.astype('float32') / 255.0

x_test = x_test.astype('float32') / 255.0


# Flatten the images

x_train = x_train.reshape((len(x_train), np.prod(x_train.shape[1:])))

x_test = x_test.reshape((len(x_test), np.prod(x_test.shape[1:])))


# Build the autoencoder model with an encoder and decoder

encoding_dim = 32  # Number of neurons in the bottleneck layer


# Input layer

input_img = tf.keras.Input(shape=(784,))


# Encoder
```

```
encoded = layers.Dense(encoding_dim, activation='relu')(input_img)

# Decoder
decoded = layers.Dense(784, activation='sigmoid')(encoded)

# Autoencoder model
autoencoder = models.Model(input_img, decoded)

# Decoder model (only includes the decoder part)
decoder_input = tf.keras.Input(shape=(encoding_dim,))
decoder_layer = autoencoder.layers[-1]  # Use the last layer of the autoencoder
decoder = models.Model(decoder_input, decoder_layer(decoder_input))

# Compile the autoencoder (not used in this example, but included for completeness)
autoencoder.compile(optimizer='adam', loss='binary_crossentropy')

# Display the decoder model summary
decoder.summary()
```

Output: -

```
Model: "model_3"
_________________________________________________________________
Layer (type)                Output Shape              Param #
=================================================================
input_3 (InputLayer)        [(None, 32)]              0
_________________________________________________________________
dense_9 (Dense)             (None, 784)               25872
=================================================================
Total params: 25,872
Trainable params: 25,872
Non-trainable params: 0
_________________________________________________________________
```

Result: - Program executed successfully.

EXPERIMENT – 9

CREATE A PROGRAM OF AN ENCODER USING A BASIC NEURAL NETWORK WITH KERAS IN PYTHON

Aim: - To use the trained encoder to encode the original data, and we display the original and encoded data for the first example.

Procedure: -

```
# Import necessary libraries
from keras.layers import Input, Dense
from keras.models import Model
import numpy as np

# Create a simple dataset for demonstration
# Each data point is a vector of length 10
# You can replace this with your own dataset
data = np.random.random((1000, 10))

# Define the architecture of the encoder
input_data = Input(shape=(10,))
encoded = Dense(5, activation='relu')(input_data)  # Encoder layer with 5 neurons

# Create the encoder model
encoder_model = Model(input_data, encoded)

# Compile the encoder model (not necessary for an encoder, but included for completeness)
encoder_model.compile(optimizer='adam', loss='mse')  # Use mean squared error as a dummy loss

# Display the architecture of the encoder
encoder_model.summary()
```

Encode the data using the trained encoder

encoded_data = encoder_model.predict(data)

Display the original and encoded data for the first example

print("Original Data:")

print(data[0])

print("Encoded Data:")

print(encoded_data[0])

Output: -

```
Model: "model_4"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
input_4 (InputLayer)         [(None, 10)]              0
_________________________________________________________________
dense_10 (Dense)             (None, 5)                 55
=================================================================
Total params: 55
Trainable params: 55
Non-trainable params: 0
_________________________________________________________________
Original Data:
[0.84292158 0.52975721 0.79121258 0.28516277 0.08029    0.43078745
 0.43391456 0.47488537 0.5187685  0.48892646]
Encoded Data:
[0.5135934 0.598009  0.        0.9055654 0.        ]
```

Result: - Program executed successfully.

EXPERIMENT – 10

PROGRAM TO DEMONSTRATE THE FLOW OF DATA THROUGH A NEURAL NETWORK DURING FORWARD PROPAGATION

Aim: - Create a program demonstrates the flow of data through a neural network during forward propagation.

Procedure: -

```
import numpy as np


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


# Define a simple neural network with random weights and biases
def initialize_parameters(input_size, hidden_size, output_size):
    np.random.seed(42)
    W1 = np.random.randn(hidden_size, input_size)  # Weights for the first layer
    b1 = np.zeros((hidden_size, 1))  # Biases for the first layer
    W2 = np.random.randn(output_size, hidden_size)  # Weights for the second layer
    b2 = np.zeros((output_size, 1))  # Biases for the second layer

    parameters = {"W1": W1, "b1": b1, "W2": W2, "b2": b2}
    return parameters


def forward_propagation(X, parameters):
    # Retrieve parameters
    W1, b1, W2, b2 = parameters["W1"], parameters["b1"], parameters["W2"], parameters["b2"]

    # Forward pass through the first layer
    Z1 = np.dot(W1, X) + b1
    A1 = sigmoid(Z1)
```

```
    # Forward pass through the second layer
    Z2 = np.dot(W2, A1) + b2
    A2 = sigmoid(Z2)

    # Output of the neural network
    output = A2
    return output

# Example usage
input_size = 3
hidden_size = 4
output_size = 1

# Initialize random parameters
parameters = initialize_parameters(input_size, hidden_size, output_size)

# Generate random input data
X = np.random.randn(input_size, 1)

# Perform forward propagation
output = forward_propagation(X, parameters)

# Display the input data and the output of the neural network
print("Input Data:")
print(X)
print("\nOutput of the Neural Network:")
print(output)
```

Output: -

```
Input Data:
[[-1.01283112]
 [ 0.31424733]
 [-0.90802408]]

Output of the Neural Network:
[[0.25942616]]
```

Result: - Program executed successfully.

EXPERIMENT – 11

PROGRAM TO DEFINE A BASIC NEURAL NETWORK LAYER WITH BIAS AND PERFORMS A FORWARD PASS WITH A GIVEN INPUT

Aim: - Create a program defines a basic neural network layer with bias and performs a forward pass with a given input and generates output predictions for a given input.

Procedure: -

```
import numpy as np

def linear_activation(inputs, weights, bias):
    """
    Perform a linear activation (weighted sum + bias) for a neural network layer.
    Args:
    - inputs: Input data (numpy array)
    - weights: Weights for the layer (numpy array)
    - bias: Bias for the layer (scalar)
    Returns:
    - output: Output of the layer (numpy array)
    """
    weighted_sum = np.dot(inputs, weights) + bias
    return weighted_sum

# Define a simple neural network layer with bias
input_size = 3
output_size = 1

# Randomly initialize weights and bias
weights = np.random.randn(input_size)
bias = np.random.randn()

# Create input data
```

```
input_data = np.array([0.5, 0.3, 0.2])


# Perform a forward pass through the layer
output = linear_activation(input_data, weights, bias)


# Display the input data, weights, bias, and output
print("Input Data:", input_data)
print("Weights:", weights)
print("Bias:", bias)
print("Output:", output)
```

Output: -

```
Input Data: [0.5 0.3 0.2]
Weights: [-1.4123037   1.46564877 -0.2257763 ]
Bias: 0.06752820468792384
Output: -0.2440842754005629
```

Result: - Program executed successfully.

EXPERIMENT – 12

WRITE A SIMPLE PYTHON PROGRAM THAT DEMONSTRATES GRADIENT DESCENT

Aim: - To create a simple program that demonstrates gradient descent.

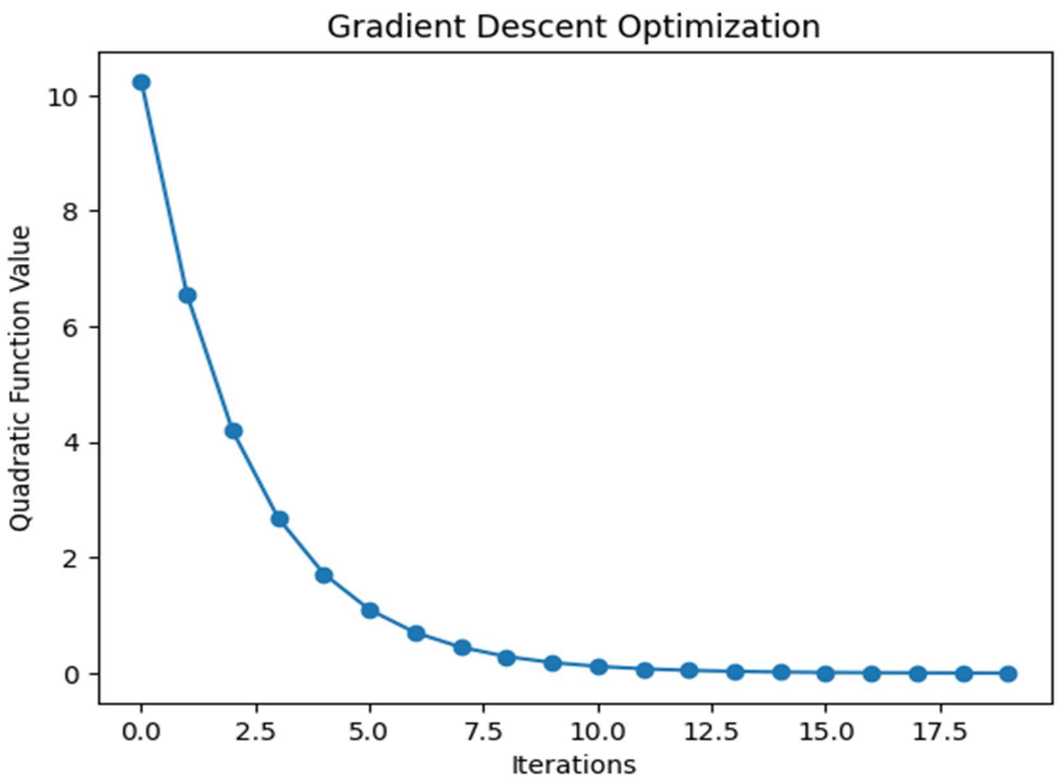
Procedure: -

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


# Function to compute the gradient of a quadratic function
def compute_gradient(x):
    return 2 * x


# Gradient Descent function
def gradient_descent(initial_x, learning_rate, num_iterations):
    """
    Perform gradient descent optimization on a quadratic function.
    Args:
    - initial_x: Initial guess for the minimum (scalar)
    - learning_rate: Step size for each iteration (scalar)
    - num_iterations: Number of iterations to perform (integer)
    Returns:
    - x_values: List of x values during the optimization (list)
    - y_values: List of y values (quadratic function) during the optimization (list)
    """
    x_values = []
    y_values = []

    x = initial_x
```

```
    for _ in range(num_iterations):
        # Compute the gradient
        gradient = compute_gradient(x)

        # Update the value of x using gradient descent
        x = x - learning_rate * gradient

        # Calculate the corresponding y value (quadratic function)
        y = x**2

        # Store the values for visualization
        x_values.append(x)
        y_values.append(y)

    return x_values, y_values

# Initial parameters
initial_guess = 4.0
learning_rate = 0.1
num_iterations = 20

# Perform gradient descent
x_values, y_values = gradient_descent(initial_guess, learning_rate, num_iterations)

# Display the results
print("Optimal x value:", x_values[-1])
print("Optimal y value (minimized):", y_values[-1])
```

```
# Plot the optimization process
plt.plot(range(num_iterations), y_values, marker='o')
plt.xlabel('Iterations')
plt.ylabel('Quadratic Function Value')
plt.title('Gradient Descent Optimization')
plt.show()
```

Output: -



Result: - Program executed successfully.

EXPERIMENT – 13

CREATE A PYTHON PROGRAM USING TENSORFLOW AND KERAS TO DEFINE A NEURAL NETWORK WITH WEIGHTS

Aim: - The aim of this program is to create simple Python program using TensorFlow and Keras to define a neural network with weights.

Procedure: -

```
import tensorflow as tf
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense
import numpy as np

# Create a simple dataset for demonstration
X = np.array([[0, 0], [0, 1], [1, 0], [1, 1]])
y = np.array([[0], [1], [1], [0]])

# Define a simple neural network with one hidden layer
model = Sequential()

# Input layer (2 input nodes)
model.add(Dense(units=2, input_dim=2, activation='relu', name='input_layer'))

# Hidden layer with weights to be defined
model.add(Dense(units=1, activation='sigmoid', name='output_layer'))

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

# Display the model summary
model.summary()
```

```
# Train the model on the dataset
model.fit(X, y, epochs=1000, verbose=0)

# Evaluate the trained model
loss, accuracy = model.evaluate(X, y)
print(f'\nEvaluation - Loss: {loss}, Accuracy: {accuracy}')

# Display the learned weights
print("\nLearned Weights:")
for layer in model.layers:
    if 'Dense' in layer.name:
        weights, biases = layer.get_weights()
print(f"{layer.name} Weights:\n{weights}\n")
```

Output: -

```
Model: "sequential_7"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
input_layer (Dense)          (None, 2)                 6
_________________________________________________________________
output_layer (Dense)         (None, 1)                 3
=================================================================
Total params: 9
Trainable params: 9
Non-trainable params: 0
_________________________________________________________________
1/1 [==============================] - 0s 434ms/step - loss: 0.5028 - accu
racy: 0.7500

Evaluation - Loss: 0.5028179883956909, Accuracy: 0.75

Learned Weights:
output_layer Weights:
[[ 1.913241 ]
 [-0.6327765]]
```

Result: - Program executed successfully.

EXPERIMENT – 14

WRITE A PYTHON PROGRAM USING TENSORFLOW AND KERAS TO DEFINE A NEURAL NETWORK WITH BIASES

Aim: - The aim of this program is to create a Python program using TensorFlow and Keras to define a neural network with biases.

Procedure: -

```
import tensorflow as tf

from tensorflow.keras.models import Sequential

from tensorflow.keras.layers import Dense

import numpy as np


# Create a simple dataset for demonstration

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

y = np.array([[0], [1], [1], [0]])


# Define a simple neural network with one hidden layer

model = Sequential()


# Input layer (2 input nodes)

model.add(Dense(units=2, input_dim=2, activation='relu', use_bias=True, name='input_layer'))


# Hidden layer with biases to be defined

model.add(Dense(units=1, activation='sigmoid', use_bias=True, name='output_layer'))


# Compile the model

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


# Display the model summary

model.summary()
```

Train the model on the dataset

model.fit(X, y, epochs=1000, verbose=0)

Evaluate the trained model

loss, accuracy = model.evaluate(X, y)

print(f'\nEvaluation - Loss: {loss}, Accuracy: {accuracy}')

Display the learned biases

print("\nLearned Biases:")

for layer in model.layers:

    if 'Dense' in layer.name:

        weights, biases = layer.get_weights()

print(f"{layer.name} Biases:\n{biases}\n")

Output: -

```
Model: "sequential_9"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
input_layer (Dense)          (None, 2)                 6
_________________________________________________________________
output_layer (Dense)         (None, 1)                 3
=================================================================
Total params: 9
Trainable params: 9
Non-trainable params: 0
_________________________________________________________________
1/1 [==============================] - 1s 585ms/step - loss: 0.5683 - accu
racy: 0.7500

Evaluation - Loss: 0.5682690739631653, Accuracy: 0.75

Learned Biases:
output_layer Biases:[-0.41911563]
```

Result: - Program executed successfully.

EXPERIMENT – 15

DEMONSTRATE A PYTHON PROGRAM THAT CREATES AN ARTIFICIAL NEURAL NETWORK (ANN)

Aim: - Creating a Python program that creates an Artificial Neural Network (ANN) using TensorFlow and Keras for a binary classification task with uses a synthetic dataset for illustration purposes. The ANN consists of an input layer, a hidden layer with ReLU activation, and an output layer with a sigmoid activation function.

Procedure: -

```
import numpy as np

import tensorflow as tf

from sklearn.model_selection import train_test_split

from sklearn.preprocessing import StandardScaler

from tensorflow.keras import models, layers


# Generate synthetic data for a binary classification task

np.random.seed(42)

X = np.random.randn(1000, 10)  # 1000 samples with 10 features

y = np.random.randint(2, size=(1000, 1))  # Binary labels (0 or 1)


# Split the data into training and testing sets

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)


# Standardize the features using StandardScaler

scaler = StandardScaler()

X_train = scaler.fit_transform(X_train)

X_test = scaler.transform(X_test)


# Build the ANN model

model = models.Sequential()
```

```
model.add(layers.Dense(32, activation='relu', input_shape=(X_train.shape[1],)))
model.add(layers.Dense(1, activation='sigmoid'))

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

# Display the model summary
model.summary()

# Train the model
model.fit(X_train, y_train, epochs=10, batch_size=32, validation_data=(X_test, y_test))

# Evaluate the model on the test set
test_loss, test_acc = model.evaluate(X_test, y_test)
print(f'Test accuracy: {test_acc}')
```

Output: -

```
Model: "sequential_10"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense (Dense)                (None, 32)                352
_________________________________________________________________
dense_1 (Dense)              (None, 1)                 33
=================================================================
Total params: 385
Trainable params: 385
Non-trainable params: 0
_________________________________________________________________
Epoch 1/10
25/25 [==============================] - 3s 38ms/step - loss: 0.7165 - acc
uracy: 0.5125 - val_loss: 0.7165 - val_accuracy: 0.5050
Epoch 2/10
25/25 [==============================] - 0s 15ms/step - loss: 0.7058 - acc
uracy: 0.5138 - val_loss: 0.7099 - val_accuracy: 0.5050
Epoch 3/10
25/25 [==============================] - 0s 15ms/step - loss: 0.6992 - acc
uracy: 0.5188 - val_loss: 0.7063 - val_accuracy: 0.5150
Epoch 4/10
```

```
25/25 [==============================] - 0s 17ms/step - loss: 0.6945 - acc
uracy: 0.5275 - val_loss: 0.7039 - val_accuracy: 0.5100
Epoch 5/10
25/25 [==============================] - 0s 15ms/step - loss: 0.6907 - acc
uracy: 0.5312 - val_loss: 0.7027 - val_accuracy: 0.5250
Epoch 6/10
25/25 [==============================] - 0s 16ms/step - loss: 0.6880 - acc
uracy: 0.5362 - val_loss: 0.7018 - val_accuracy: 0.5000
Epoch 7/10
25/25 [==============================] - 1s 23ms/step - loss: 0.6853 - acc
uracy: 0.5425 - val_loss: 0.7020 - val_accuracy: 0.4800
Epoch 8/10
25/25 [==============================] - 0s 20ms/step - loss: 0.6831 - acc
uracy: 0.5575 - val_loss: 0.7018 - val_accuracy: 0.4700
Epoch 9/10
25/25 [==============================] - 1s 21ms/step - loss: 0.6812 - acc
uracy: 0.5638 - val_loss: 0.7014 - val_accuracy: 0.4750
Epoch 10/10
25/25 [==============================] - 1s 27ms/step - loss: 0.6799 - acc
uracy: 0.5612 - val_loss: 0.7017 - val_accuracy: 0.4800
7/7 [==============================] - 0s 10ms/step - loss: 0.7017 - accur
acy: 0.4800
```

Test accuracy: 0.47999998927116394

<u>Result:</u> - Program executed successfully.