EXPERIMENT – 1

PYTHON PROGRAM THAT DEMONSTRATES THE USE OF THE ReLU ACTIVATION FUNCTION IN A BASIC NEURAL NETWORK

Aim: - The aim of this program is to use the ReLU activation function in a basic neural network.

Procedure: -

```
import tensorflow as tf
from tensorflow.keras import layers, models

# Build a simple neural network with ReLU activation
model = models.Sequential([
    layers.Dense(32, input_shape=(10,), activation='relu'),
    layers.Dense(1, activation='sigmoid')
])

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

Output: -

```
Model: "sequential"
_________________________________________________________________
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
_________________________________________________________________
```

Result: - Program executed successfully.

EXPERIMENT – 2

SIMPLE PYTHON PROGRAM THAT CREATES A BASIC NEURON AND PERFORMS A FORWARD PASS

Aim: - To create a neuron, performs a forward pass with random weights and bias, and displays the relevant information.

Procedure: -

```
import numpy as np

class Neuron:
    def __init__(self, input_size):
        # Initialize weights and bias randomly
        self.weights = np.random.rand(input_size)
        self.bias = np.random.rand(1)

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

    def forward(self, inputs):
        # Perform a forward pass through the neuron
        weighted_sum = np.dot(inputs, self.weights) + self.bias
        output = self.sigmoid(weighted_sum)
        return output

# Example usage
if __name__ == "__main__":
    # Create a neuron with 3 input features
    neuron = Neuron(input_size=3)
```

```
# Input data for a single example
input_data = np.array([0.5, 0.3, 0.2])

# Perform a forward pass through the neuron
output = neuron.forward(input_data)

# Display the results
print("Input Data:", input_data)
print("Weights:", neuron.weights)
print("Bias:", neuron.bias)
print("Weighted Sum:", np.dot(input_data, neuron.weights) + neuron.bias)
print("Output after Sigmoid Activation:", output)
```

Output:-

```
Input Data: [0.5 0.3 0.2]
Weights: [0.46353624 0.80029125 0.23403905]
Bias: [0.06518426]
Weighted Sum: [0.58384757]
Output after Sigmoid Activation: [0.64195225]
```

Result: - Program executed successfully.

EXPERIMENT – 3

WRITE A PROGRAM USING TENSORFLOW AND KERAS TO CREATE A NEURAL NETWORK WITH AN OPTIMIZER

Aim: - To sets up a basic neural network with the SGD optimizer for the MNIST digit classification task.

Procedure: -

```
import tensorflow as tf

from tensorflow.keras.models import Sequential

from tensorflow.keras.layers import Dense

# Create a simple dataset for demonstration

X = [[0, 0], [0, 1], [1, 0], [1, 1]]

y = [0, 1, 1, 0]


# Define a fully connected neural network with SGD optimizer

model = Sequential()


# Input layer with 2 neurons

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


# Output layer with 1 neuron (binary classification)

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


# Compile the model with Stochastic Gradient Descent (SGD) optimizer

sgd_optimizer = tf.keras.optimizers.SGD(learning_rate=0.01)

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


# Display the model summary

model.summary()
```

Output: -

```
Model: "sequential_1"
_________________________________________________________________
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
_________________________________________________________________
```

Result: - Program executed successfully.

EXPERIMENT – 4

SIMPLE PROGRAM THAT BUILDS A GENERATOR FOR A GENERATIVE ADVERSARIAL NETWORK (GAN)

Aim: - To generates images that resemble handwritten digits using a simple generator architecture.

Procedure:-

```
import tensorflow as tf

from tensorflow.keras import layers, models

import numpy as np

import matplotlib.pyplot as plt


# Define the generator model

def build_generator(latent_dim):

    model = models.Sequential()

    model.add(layers.Dense(128, input_dim=latent_dim, activation='relu'))

    model.add(layers.Dense(784, activation='sigmoid'))

    model.add(layers.Reshape((28, 28, 1)))

    return model


# Build the generator

latent_dim = 100

generator = build_generator(latent_dim)


# Display the generator summary

generator.summary()
```

Output:-

```
Model: "sequential_2"
_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
dense_2 (Dense)              (None, 128)               12928     
_________________________________________________________________
dense_3 (Dense)              (None, 784)               101136    
_________________________________________________________________
reshape (Reshape)            (None, 28, 28, 1)         0         
=================================================================
Total params: 114,064
Trainable params: 114,064
Non-trainable params: 0
_________________________________________________________________
```

Result: - Program executed successfully.

EXPERIMENT – 5

WRITE A PROGRAM THAT BUILDS A DISCRIMINATOR FOR A GENERATIVE ADVERSARIAL NETWORK (GAN)

Aim: - To discriminate between real and generated images, particularly focusing on recognizing handwritten digits.

Procedure: -

```
import tensorflow as tf

from tensorflow.keras import layers, models


# Define the discriminator model

def build_discriminator(input_shape=(28, 28, 1)):

    model = models.Sequential()

    model.add(layers.Flatten(input_shape=input_shape))

    model.add(layers.Dense(128, activation='relu'))

    model.add(layers.Dense(1, activation='sigmoid'))

    return model


# Build the discriminator

discriminator = build_discriminator()


# Display the discriminator summary

discriminator.summary()
```

<u>Output</u>:-

```
Model: "sequential_3"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
flatten (Flatten)            (None, 784)               0

_________________________________________________________________
dense_4 (Dense)              (None, 128)               100480

_________________________________________________________________
dense_5 (Dense)              (None, 1)                 129
=================================================================
Total params: 100,609
Trainable params: 100,609
Non-trainable params: 0
_________________________________________________________________
```

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

EXPERIMENT – 6

CREATE A PYTHON PROGRAM USING TENSORFLOW AND KERAS TO DEFINE A NEURAL NETWORK WITH A SPECIFIC ACTIVATION FUNCTION

Aim: - By creating function create_neural_network() that takes the input shape as an argument and returns a simple neural network model.

Procedure: -

```
import tensorflow as tf

from tensorflow.keras.models import Sequential

from tensorflow.keras.models import Model

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 and a specific activation function

model = Sequential()

# Input layer (2 input nodes)

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

# Hidden layer with a specific activation function (e.g., sigmoid)

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 activations of the hidden layer for a sample input
sample_input = np.array([[0, 1]])
hidden_layer_activation = Model(inputs=model.input,
                                outputs=model.get_layer('input_layer').output).predict(sam
                                ple_input)
output_activation = Model(inputs=model.input,
                          outputs=model.get_layer('output_layer').output).predict(sa
                          mple_input)

print("\nHidden Layer Activation:")
print(hidden_layer_activation)

print("\nOutput Layer Activation:")
print(output_activation)
```

Output: -

```
Model: "sequential_17"
_________________________________________________________________
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 425ms/step - loss: 0.5255 - accu
racy: 0.7500

Evaluation - Loss: 0.5255492329597473, Accuracy: 0.75

Hidden Layer Activation:
[[0.        1.1805922]]

Output Layer Activation:
[[0.84069824]]
```

Result: - Program executed successfully.