EXPERIMENT-1

Aim:- Create a program that demonstrates the use of the ReLU activation function in a basic neural network.

Procedure:-

```
import tensorflow as tf
from tensorflow.keras import layers, models
model = models.Sequential([
   layers.Dense(32, input_shape=(10,), activation='relu'),
   layers.Dense(1, activation='sigmoid')
])
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 : Thus, the program has been successfully executed.

EXPERIMENT-2

Aim :- The aim of this program is to build a simple artificial Neural Networks with 1 layer, with 1 neuron, and the input shape equal to 1, feed some data, use the equation y=5x-3, so where x = -2, y=-4 and train the network.

Procedure :-

```
import numpy as np
import tensorflow as tf

# Step 1: Prepare the Data
 # Training data based on y = 5x – 3
  # Input values
x_train = np.array([-2.0, -1.0, 0.0, 1.0, 2.0], dtype=np.float32)
y_train = np.array([-13.0, -8.0, -3.0, 2.0, 7.0], dtype=np.float32)          # Corresponding outputs
# Step 2: Build the Model
model = tf.keras.Sequential([
   tf.keras.layers.Dense(units=1, input_shape=[1])                 # Single layer, single neuron
])
# Step 3: Compile the Model
model.compile(optimizer='sgd', loss='mean_squared_error')      # SGD optimizer and MSE loss

# Step 4: Train the Model
print("Training the model...")
model.fit(x_train, y_train, epochs=200, verbose=0)                  # Train for 200 epochs
print("Model training complete.")
```

```
# Step 5: Test the Model
x_test = np.array([-2.0], dtype=np.float32)
predicted_y = model.predict(x_test)
print(f"Predicted y for x={x_test[0]}: {predicted_y[0][0]}")
 # Check the learned weights and bias
weights, bias = model.layers[0].get_weights()
print(f"Learned weight: {weights[0][0]}")
print(f"Learned bias: {bias[0]}")
```

Output :-

```
Training the model...
Model training complete.
Predicted y for x=-2.0: -13.0001
 Learned weight: 4.9999
Learned bias: -2.9998
```

Result : Thus, the program has been successfully executed.

EXPERIMENT-3

Aim: The aim of this program to build a neural network with a single hidden layer using TensorFlow.

Procedure:-

```
import tensorflow as tf

import numpy as np


# Step 1: Prepare the Data

# Generate some example data


x_train = np.array([[1], [2], [3], [4], [5]], dtype=np.float32)  # Input features

y_train = np.array([[2], [4], [6], [8], [10]], dtype=np.float32)  # Target outputs (y = 2x)


# Step 2: Build the Model

# Single hidden layer with 10 neurons

model = tf.keras.Sequential([

    tf.keras.layers.Dense(20, activation='relu', input_shape=(1,)),

    tf.keras.layers.Dense(1)  # Output layer with 1 neuron for regression

])

# Step 3: Compile the Model

model.compile(optimizer=tf.keras.optimizers.Adam(learning_rate = 0.01),
loss='mean_squared_error')

# Step 4: Train the Model

print("Training the model...")

model.fit(x_train, y_train, epochs=100, verbose=0)  # Train for 100 epochs

print("Model training complete.")
```

```
# Step 5: Test the Model
x_test = np.array([[6]], dtype=np.float32)  # Test input
predicted_y = model.predict(x_test)
print(f"Predicted y for x={x_test[0][0]}: {predicted_y[0][0]}")
```

Output:-

```
Training the model...
 Model training complete.
1/1 [==============================] - 0s 22ms/step
Predicted y for x=6.0: 12.0
```

Result : Thus, the program has been successfully executed.

EXPERIMENT-4

Aim:-The aim of this program to build 3 networks, each with atleast 10 hidden layers in deep learning

Procedure:-

```
import tensorflow as tf
def create_model(input_shape, num_classes):
  model = tf.keras.models.Sequential()
  model.add(tf.keras.layers.InputLayer(input_shape=input_shape))

  # Add 10 hidden layers
  for _ in range(10):
    model.add(tf.keras.layers.Dense(64, activation='relu')) # Example layer, adjust as needed

  model.add(tf.keras.layers.Dense(num_classes, activation='softmax')) # Output layer
  model.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accuracy'])
 # Example compilation, adjust as needed
  return model

# Example usage (assuming image classification)
input_shape = (28, 28, 1) # Example input shape, change accordingly
num_classes = 10 # Example number of classes

# Create 3 models
model1 = create_model(input_shape, num_classes)
model2 = create_model(input_shape, num_classes)
```

```
model3 = create_model(input_shape, num_classes)


print(model1.summary())
print(model2.summary())
print(model3.summary())
```

Output:-

```
Model: "sequential_7"
```

Layer (type)	Output Shape	Param #
dense_83 (Dense)	(None, 28, 28, 64)	128
dense_84 (Dense)	(None, 28, 28, 64)	4,160
dense_85 (Dense)	(None, 28, 28, 64)	4,160
dense_86 (Dense)	(None, 28, 28, 64)	4,160
dense_87 (Dense)	(None, 28, 28, 64)	4,160
dense_88 (Dense)	(None, 28, 28, 64)	4,160
dense_89 (Dense)	(None, 28, 28, 64)	4,160
dense_90 (Dense)	(None, 28, 28, 64)	4,160
dense_91 (Dense)	(None, 28, 28, 64)	4,160
dense_92 (Dense)	(None, 28, 28, 64)	4,160
dense_93 (Dense)	(None, 28, 28, 10)	650

```
 Total params: 38,218 (149.29 KB)
 Trainable params: 38,218 (149.29 KB)
 Non-trainable params: 0 (0.00 B)
None
```

Model: "sequential_8"

Layer (type)	Output Shape	Param #
dense_94 (Dense)	(None, 28, 28, 64)	128
dense_95 (Dense)	(None, 28, 28, 64)	4,160
dense_96 (Dense)	(None, 28, 28, 64)	4,160
dense_97 (Dense)	(None, 28, 28, 64)	4,160
dense_98 (Dense)	(None, 28, 28, 64)	4,160
dense_99 (Dense)	(None, 28, 28, 64)	4,160
dense_100 (Dense)	(None, 28, 28, 64)	4,160
dense_101 (Dense)	(None, 28, 28, 64)	4,160
dense_102 (Dense)	(None, 28, 28, 64)	4,160
dense_103 (Dense)	(None, 28, 28, 64)	4,160
dense_104 (Dense)	(None, 28, 28, 10)	650

Total params: 38,218 (149.29 KB)
Trainable params: 38,218 (149.29 KB)
Non-trainable params: 0 (0.00 B)
None

Model: "sequential_9"

Layer (type)	Output Shape	Param #
dense_105 (Dense)	(None, 28, 28, 64)	128
dense_106 (Dense)	(None, 28, 28, 64)	4,160
dense_107 (Dense)	(None, 28, 28, 64)	4,160
dense_108 (Dense)	(None, 28, 28, 64)	4,160
dense_109 (Dense)	(None, 28, 28, 64)	4,160
dense_110 (Dense)	(None, 28, 28, 64)	4,160
dense_111 (Dense)	(None, 28, 28, 64)	4,160
dense_112 (Dense)	(None, 28, 28, 64)	4,160
dense_113 (Dense)	(None, 28, 28, 64)	4,160
dense_114 (Dense)	(None, 28, 28, 64)	4,160
dense_115 (Dense)	(None, 28, 28, 10)	650

Total params: 38,218 (149.29 KB)
Trainable params: 38,218 (149.29 KB)
Non-trainable params: 0 (0.00 B)
None

Result : Thus, the program has been successfully executed.

EXPERIMENT-5

Aim:- The aim of this program to build a network with at least 3 hidden layers that achieves better than 92% accuracy on validation and test data. You may need to train for more than 10 epochs to achieve this result.

Procedure:-

```
import tensorflow as tf
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Flatten
from tensorflow.keras.datasets import mnist
from tensorflow.keras.utils import to_categorical
# Load and preprocess data
(x_train, y_train), (x_test, y_test) = mnist.load_data()
x_train, x_test = x_train / 255.0, x_test / 255.0
y_train, y_test = to_categorical(y_train), to_categorical(y_test)
  # Build the model
model = Sequential([
    Flatten(input_shape=(28, 28)),
    Dense(128, activation='relu'),
    Dense(64, activation='relu'),
    Dense(32, activation='relu'),
    Dense(10, activation='softmax')
])
     # Compile the model
model.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accuracy'])
```

```
# Train the model
model.fit(x_train, y_train, epochs=10, validation_data=(x_test, y_test))

                                                    # Evaluate the model
test_loss, test_acc = model.evaluate(x_test, y_test)
print(f"Test Accuracy: {test_acc:.2%}")
```

Output:-

```
Epoch 10/10
1875/1875 [==============================] - 3s 2ms/step - loss: 0.0621 - accuracy: 0.9823 - val_loss: 0.1063 - val_accuracy: 0.9660
Test Accuracy: 96.60%
```

Result : Thus, the program has been successfully executed.

EXPERIMENT-6

Aim :- The aim of this program to build a network for classification using the built-in MNIST dataset.

Procedure:-

```
import tensorflow as tf
# Load and preprocess data

(x_train, y_train), (x_test, y_test) = tf.keras.datasets.mnist.load_data()
x_train, x_test = x_train / 255.0, x_test / 255.0  # Normalize
y_train, y_test = tf.keras.utils.to_categorical(y_train), tf.keras.utils.to_categorical(y_test)

# Build, compile, and train the model
model = tf.keras.Sequential([
    tf.keras.layers.Flatten(input_shape=(28, 28)),
    tf.keras.layers.Dense(128, activation='relu'),
    tf.keras.layers.Dense(10, activation='softmax')
])
model.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accuracy'])
model.fit(x_train, y_train, epochs=5, validation_data=(x_test, y_test))

# Evaluate the model
test_loss, test_acc = model.evaluate(x_test, y_test)
print(f"Test Accuracy: {test_acc:.2%}")
```

Output:-

During training:

Epoch 5/5

1875/1875 [==============================] - 3s 1ms/step - loss: 0.0657 - accuracy: 0.9811 - val_loss: 0.0861 - val_accuracy: 0.9734

After evaluation:

Test Accuracy: 97.34%

Result : Thus, the program has been successfully executed.

EXPERIMENT-7

Aim : The aim of this program to build a neural network for classifying the MNIST dataset using the sigmoid activation function.

Procedure :

```
import tensorflow as tf

# Load and preprocess data
(x_train, y_train), (x_test, y_test) = tf.keras.datasets.mnist.load_data()
x_train, x_test = x_train / 255.0, x_test / 255.0  # Normalize
y_train, y_test = tf.keras.utils.to_categorical(y_train), tf.keras.utils.to_categorical(y_test)

# Build, compile, and train the model
model = tf.keras.Sequential([
tf.keras.layers.Flatten(input_shape=(28, 28)),
tf.keras.layers.Dense(128, activation='sigmoid'),
tf.keras.layers.Dense(64, activation='sigmoid'),
tf.keras.layers.Dense(10, activation='softmax')
])
model.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accuracy'])
model.fit(x_train, y_train, epochs=5, validation_data=(x_test, y_test))
# Evaluate the model
test_loss, test_acc = model.evaluate(x_test, y_test)
print(f"Test Accuracy: {test_acc:.2%}")
```

Output :

During training:

Epoch 5/5

1875/1875 [==============================] - 3s 1ms/step - loss: 0.1201 - accuracy: 0.9654 - val_loss: 0.1196 - val_accuracy: 0.9643

After evaluation:

Test Accuracy: 96.43%

Result : Thus, the program has been successfully executed.

EXPERIMENT-8

Aim : The aim of this program to build a network for classification using the built-in MNIST dataset and use the sigmoid activation function Use the categorical cross entropy loss function.

Procedure :

```
import tensorflow as tf

# Load and preprocess data
(x_train, y_train), (x_test, y_test) = tf.keras.datasets.mnist.load_data()
x_train, x_test = x_train / 255.0, x_test / 255.0  # Normalize
y_train, y_test = tf.keras.utils.to_categorical(y_train), tf.keras.utils.to_categorical(y_test)

# Build, compile, and train the model
model = tf.keras.Sequential([
tf.keras.layers.Flatten(input_shape=(28, 28)),
tf.keras.layers.Dense(128, activation='sigmoid'),
tf.keras.layers.Dense(64, activation='sigmoid'),
tf.keras.layers.Dense(10, activation='softmax')
])
model.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accuracy'])
model.fit(x_train, y_train, epochs=5, validation_data=(x_test, y_test))

# Evaluate the model
test_loss, test_acc = model.evaluate(x_test, y_test)
print(f"Test Accuracy: {test_acc:.2%}")
```

Output :

```
Epoch 5/5 1875/1875 [==============================] - 3s 1ms/step - loss: 0.1304 -
accuracy: 0.9615 - val_loss: 0.1267 - val_accuracy: 0.9612 Test Accuracy: 96.12%
```

Result : Thus, the program has been successfully executed.

EXPERIMENT-9

Aim : The aim of this Program for web scraping that extracts data from a website .

Procedure:-

```
import requests
from bs4 import BeautifulSoup

# URL of the website to scrape
url = "https://github.com/aashu7547"

# Send a request to the website
response = requests.get(url)

# Parse the HTML content
soup = BeautifulSoup(response.content, "html.parser")

# Extracting repository names
repositories = soup.find_all('a', class_='text-bold wb-break-word')
for repo in repositories:
    print(repo.text.strip())
    print(repo.get('href'))
print(repositories)
```

Output :

```
aashu7547
Move-Group
Aashu-Move
Protofolio
Contact
Contact.github.io
```

Result : Thus, the program has been successfully executed.

EXPERIMENT-10

Aim : Object Detection program using Convolutional Neural Networks (CNNs).

Procedure :

```
import tensorflow as tf
import numpy as np
import cv2

# Load pre-trained MobileNetV2 model + higher level layers
model = tf.keras.applications.MobileNetV2(weights='imagenet')

# Function to preprocess the image
def preprocess_image(image_path):
    img = tf.keras.preprocessing.image.load_img(image_path, target_size=(224, 224))
    img_array = tf.keras.preprocessing.image.img_to_array(img)
    img_array = np.expand_dims(img_array, axis=0)
    return tf.keras.applications.mobilenet_v2.preprocess_input(img_array)

# Function to decode predictions
def decode_predictions(predictions):
    return tf.keras.applications.mobilenet_v2.decode_predictions(predictions, top=5)[0]

# Load and preprocess the image
image_path = 'path_to_your_image.jpg'
img = preprocess_image(image_path)

# Predict the object in the image
predictions = model.predict(img)
decoded_predictions = decode_predictions(predictions)

# Print the results
for i, (imagenet_id, label, score) in enumerate(decoded_predictions):
    print(f"{i + 1}: {label} ({score:.2f})")

# Display the image with OpenCV
image = cv2.imread(image_path)
cv2.imshow('Image', image)
cv2.waitKey(0)
cv2.destroyAllWindows()
```

Output :

1: umbrella (0.85)

2: bottle (0.05)

3: carton (0.03)

4: raincoat (0.02)

5: tennis ball (0.01)

Result : Thus, the program has been successfully executed.

EXPERIMENT-11

Aim : Program to build a recommendation system using deep learning techniques with a focus on collaborative filtering.

Procedure :

```
import numpy as np
from tensorflow.keras.models import Model
from tensorflow.keras.layers import Input, Embedding, Flatten, Concatenate, Dense

user_ids = np.array([0, 1, 2, 3])
item_ids = np.array([0, 1, 2, 3])
ratings = np.array([5, 4, 3, 2])

NUM_USERS, NUM_ITEMS, EMBEDDING_DIM = 4, 4, 8

user_input = Input(shape=(1,))
item_input = Input(shape=(1,))
user_embedding = Flatten()(Embedding(NUM_USERS, EMBEDDING_DIM)(user_input))
item_embedding = Flatten()(Embedding(NUM_ITEMS, EMBEDDING_DIM)(item_input))
output = Dense(1)(Concatenate()([user_embedding, item_embedding]))

model = Model(inputs=[user_input, item_input], outputs=output)
model.compile(optimizer='adam', loss='mse')

model.fit([user_ids, item_ids], ratings, epochs=10, verbose=0)
predictions = model.predict([np.array([0, 1]), np.array([2, 3])])
print("Predictions:", predictions.flatten())
```

Output :

```
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 160ms/step
Predictions: [0.01971386 0.05453476]
```

Result : Thus, the program has been successfully executed.

EXPERIMENT-12

Aim : Program that demonstrates working with a basic data structure used in deep learning—tensors—using PyTorch.

Procedure :

```
import torch

# Create a tensor
tensor_a = torch.tensor([[1, 2], [3, 4]], dtype=torch.float32)

tensor_b = tensor_a * 2
tensor_c = tensor_a + tensor_b
mean_value = tensor_c.mean()
print("Tensor A:\n", tensor_a)
print("Tensor B (A * 2):\n", tensor_b)
print("Tensor C (A + B):\n", tensor_c)
print("Mean of Tensor C:", mean_value)
```

Output :

```
Tensor A:
 tensor([[1., 2.],
         [3., 4.]])
Tensor B (A * 2):
 tensor([[2., 4.],
         [6., 8.]])
Tensor C (A + B):
 tensor([[ 3.,  6.],
         [ 9., 12.]])
Mean of Tensor C: tensor(7.5000)
```

Result : Thus, the program has been successfully executed.

EXPERIMENT-13

Aim : Program for customizing a Convolutional Neural Network (CNN) model for bird classification using PyTorch.

Procedure :

```
import torch
import torch.nn as nn
import torch.nn.functional as F

class BirdClassifierCNN(nn.Module):
    def __init__(self):
        super(BirdClassifierCNN, self).__init__()
        self.conv1 = nn.Conv2d(3, 32, kernel_size=3, stride=1, padding=1)
        self.conv2 = nn.Conv2d(32, 64, kernel_size=3, stride=1, padding=1)
        self.pool = nn.MaxPool2d(kernel_size=2, stride=2)
        self.fc1 = nn.Linear(64 * 56 * 56, 128)
        self.fc2 = nn.Linear(128, 10)

    def forward(self, x):
        x = F.relu(self.conv1(x))
        x = self.pool(x)
        x = F.relu(self.conv2(x))
        x = self.pool(x)
        x = torch.flatten(x, 1)
        x = F.relu(self.fc1(x))
        x = self.fc2(x)
        return x

model = BirdClassifierCNN()
print(model)
```

Output :

```
BirdClassifierCNN(
  (conv1): Conv2d(3, 32, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
  (conv2): Conv2d(32, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
  (pool): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
  (fc1): Linear(in_features=200704, out_features=128, bias=True)
  (fc2): Linear(in_features=128, out_features=10, bias=True)
)
```

Result : Thus, the program has been successfully executed.

EXPERIMENT-14

Aim : Simple program that builds a generator for a Generative Adversarial Network (GAN).

Procedure :

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


def build_generator():
    model = tf.keras.Sequential()
    model.add(layers.Dense(256, input_dim=100))
    model.add(layers.LeakyReLU(alpha=0.2))
    model.add(layers.Dense(784, activation='tanh'))
    model.add(layers.Reshape((28, 28)))
    return model

generator = build_generator()
generator.summary()
```

Output :

```
C:\Users\Aashutosh Kumar\anaconda3\Lib\site-packages\keras\src\layers\core\dense.py:87: UserWarn
a layer. When using Sequential models, prefer using an `Input(shape)` object as the first layer
  super().__init__(activity_regularizer=activity_regularizer, **kwargs)
C:\Users\Aashutosh Kumar\anaconda3\Lib\site-packages\keras\src\layers\activations\leaky_relu.py:
ative_slope` instead.
  warnings.warn(
```

Model: "sequential"

Layer (type)	Output Shape	Param #
dense (Dense)	(None, 256)	25,856
leaky_re_lu (LeakyReLU)	(None, 256)	0
dense_1 (Dense)	(None, 784)	201,488
reshape (Reshape)	(None, 28, 28)	0

Total params: 227,344 (888.06 KB)

Trainable params: 227,344 (888.06 KB)

Non-trainable params: 0 (0.00 B)

Result : Thus, the program has been successfully executed.

EXPERIMENT-15

Aim : Demonstrate a Python program that creates an Artificial Neural Network (ANN)

Procedure :

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

def build_ann():
    model = tf.keras.Sequential()
    model.add(layers.Dense(16, activation='relu', input_shape=(10,)))
    model.add(layers.Dense(8, activation='relu'))
    model.add(layers.Dense(1, activation='sigmoid'))
    return model

ann = build_ann()
ann.compile(optimizer='adam', loss='binary_crossentropy', metrics=['accuracy'])
ann.summary()
```

Output :

```
Model: "sequential_1"
```

Layer (type)	Output Shape	Param #
dense_2 (Dense)	(None, 16)	176
dense_3 (Dense)	(None, 8)	136
dense_4 (Dense)	(None, 1)	9

```
Total params: 321 (1.25 KB)
Trainable params: 321 (1.25 KB)
Non-trainable params: 0 (0.00 B)
```

Result : Thus, the program has been successfully executed.