22. Write a program in Python to implement Linear Regression with TensorFlow.

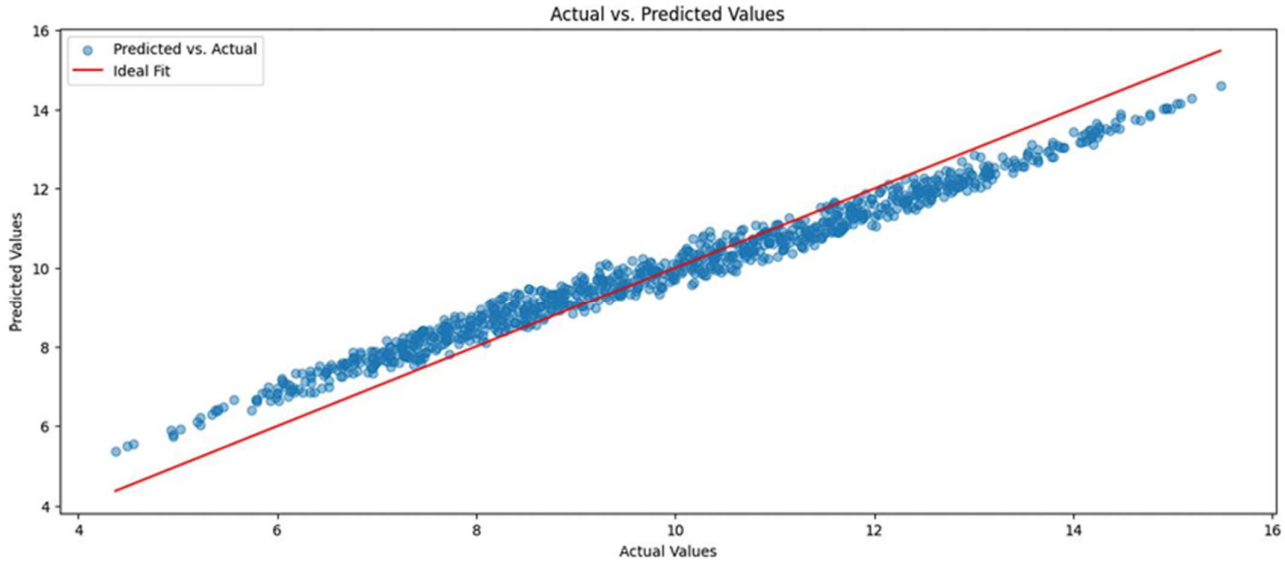
```
import tensorflow as tf
import numpy as np
import matplotlib.pyplot as plt

tf.compat.v1.disable_eager_execution()

test_data_size = 1000
iterations = 1000
learn_rate = 0.005

def generate_test_values():
    train_x = []
    train_y = []
    for _ in range(test_data_size):
        x1, x2, x3 = np.random.rand(), np.random.rand(),
np.random.rand()
        yf = 2*x1 + 3*x2 + 7*x3 + 4
        train_x.append([x1, x2, x3])
        train_y.append(yf)
    return np.array(train_x), np.array(train_y).reshape(-1, 1)

x = tf.compat.v1.placeholder(tf.float32, [None, 3], name="x")
y = tf.compat.v1.placeholder(tf.float32, [None, 1], name="y")
w = tf.Variable(tf.zeros([3, 1]), name="w")
b = tf.Variable(tf.zeros([1]), name="b")
y_model = tf.add(tf.matmul(x, w), b)

cost = tf.reduce_mean(tf.square(y - y_model))
```

```
train =
tf.compat.v1.train.GradientDescentOptimizer(learn_rate).minimize(
 cost)
train_dataset, train_values = generate_test_values()
init = tf.compat.v1.global_variables_initializer()

with tf.compat.v1.Session() as session:
    session.run(init)
    for i in range(iterations):
        session.run(train, feed_dict={x: train_dataset, y:
train_values})

    final_cost = session.run(cost, feed_dict={x: train_dataset,
y: train_values})
    final_w = session.run(w)
    final_b = session.run(b)
    predicted_values = session.run(y_model, feed_dict={x:
train_dataset})


print("Final Cost:", final_cost)
print("Learned Weights (w):", final_w.flatten())
print("Learned Bias (b):", final_b)

plt.figure(figsize=(15, 6))
plt.scatter(train_values, predicted_values, alpha=0.5,
label="Predicted vs. Actual")
plt.plot([min(train_values), max(train_values)],
[min(train_values), max(train_values)], 'r', label="Ideal Fit")
plt.xlabel("Actual Values")
```

```
plt.ylabel("Predicted Values")
plt.title("Actual vs. Predicted Values")
plt.legend()
plt.show()
```

Output: –

```
Final Cost: 0.3295301
Learned Weights (w): [2.1542575 2.7820535 5.0548334]
Learned Bias (b): [5.042796]
```



23. Write a program in Python to implement RNN (Recurrent Neural Network) using encoding method.

```
from keras.datasets import imdb
from keras.preprocessing.sequence import pad_sequences
from keras.models import Sequential
from keras.layers import Embedding, SimpleRNN, Dense

(X_train, y_train), (X_test, y_test) = imdb.load_data(num_words=10000)

X_train = pad_sequences(X_train, padding='post', maxlen=50)
X_test = pad_sequences(X_test, padding='post', maxlen=50)

model = Sequential()
model.add(Embedding(input_dim=10000, output_dim=32, input_length=50))  # Converts word indices to dense vectors
model.add(SimpleRNN(32))
model.add(Dense(1, activation='sigmoid'))

model.compile(loss='binary_crossentropy', optimizer='adam', metrics=['accuracy'])
model.summary()
model.fit(X_train, y_train, epochs=5, validation_data=(X_test, y_test))
```

Output: –

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 simple_rnn (SimpleRNN)      (None, 32)                1088

 dense (Dense)               (None, 1)                 33

=================================================================
Total params: 1,121
Trainable params: 1,121
Non-trainable params: 0
_________________________________________________________________
```

```
Epoch 1/5
782/782 [==============================] - 10s 12ms/step - loss: 0.6929 - accuracy: 0.5061 - val_loss: 0.6955 - val_accuracy: 0.5048
Epoch 2/5
782/782 [==============================] - 9s 12ms/step - loss: 0.6931 - accuracy: 0.5031 - val_loss: 0.6945 - val_accuracy: 0.5021
Epoch 3/5
782/782 [==============================] - 9s 12ms/step - loss: 0.6927 - accuracy: 0.5085 - val_loss: 0.6967 - val_accuracy: 0.5008
Epoch 4/5
782/782 [==============================] - 9s 12ms/step - loss: 0.6927 - accuracy: 0.5090 - val_loss: 0.6942 - val_accuracy: 0.5043
Epoch 5/5
782/782 [==============================] - 9s 12ms/step - loss: 0.6928 - accuracy: 0.5056 - val_loss: 0.6957 - val_accuracy: 0.5048
<keras.callbacks.History at 0x7f8dc97f8810>
```

24. Write a program in Python to implement RNN (Recurrent Neural Network) using ensemble method.

```
from keras.datasets import imdb
from keras.preprocessing.sequence import pad_sequences
from keras.models import Sequential
from keras.layers import Dense, SimpleRNN, Embedding

(X_train, y_train), (X_test, y_test) = imdb.load_data(num_words=10000)

X_train = pad_sequences(X_train, padding='post', maxlen=50)
X_test = pad_sequences(X_test, padding='post', maxlen=50)

model = Sequential()
model.add(Embedding(input_dim=10000, output_dim=2, input_length=50))
model.add(SimpleRNN(32))
model.add(Dense(1, activation='sigmoid'))

model.compile(optimizer='adam', loss='binary_crossentropy',
metrics=['accuracy'])
model.summary()
history = model.fit(X_train, y_train, epochs=5, validation_data=(X_test, y_test))
```

Output: –

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 embedding (Embedding)       (None, None, 2)           20000

 simple_rnn (SimpleRNN)      (None, 32)                1120

 dense (Dense)               (None, 1)                 33

=================================================================
Total params: 21,153
Trainable params: 21,153
Non-trainable params: 0
_________________________________________________________________
```

```
Epoch 1/5
782/782 [==============================] - 40s 50ms/step - loss: 0.5801 - acc: 0.6731 - val_loss: 0.4433 - val_acc: 0.7978
Epoch 2/5
782/782 [==============================] - 39s 50ms/step - loss: 0.3524 - acc: 0.8510 - val_loss: 0.4430 - val_acc: 0.8051
Epoch 3/5
782/782 [==============================] - 39s 50ms/step - loss: 0.2040 - acc: 0.9224 - val_loss: 0.5460 - val_acc: 0.7828
Epoch 4/5
782/782 [==============================] - 38s 49ms/step - loss: 0.0823 - acc: 0.9735 - val_loss: 0.6990 - val_acc: 0.7824
Epoch 5/5
782/782 [==============================] - 38s 49ms/step - loss: 0.0352 - acc: 0.9894 - val_loss: 0.8774 - val_acc: 0.7670
```