**Exp No 7 LINEAR REGRESSION MODEL FOR FITTING A RANDOMLY GENERATED DATA**

**AIM:**

**To implement linear regression model using TensorFlow for fitting a randomly generated data,**

**and visualizing the output.**

**INTEGRATED DEVELOPMENT ENVIRONMENT (IDE) REQUIRED:**

**JUPYTER NOTEBOOK**

**REQUIRED LIBRARIES FOR PYTHON:**

**Tensorflow**

**Numpy**

**Matplotlib**

**PROCEDURE:**

**1. Data Generation:**

**Generate random data (x). Generate output (y) with a linear relationship using the**

**formula: y=4+3⋅ x+noise (random).**

**2. Model Implementation:**

**Implement a simple linear regression model using TensorFlow. Choose Stochastic**

**Gradient Descent (SGD) as the optimizer and Mean Squared Error (MSE) as the loss**

**function. Set the learning rate and number of epochs.**

**3. Training:**

**Train the model on the generated data for a specified number of epochs. Monitor the**

**training process to observe how the loss decreases over epochs.**

**4. Visualization:**

**Plot the original data points along with the fitted line obtained from the trained model.**

**Plot the training loss over epochs to visualize the model learning process.**

**5. Analysis:**

**Print and analyze the trained parameters (slope and intercept) of the linear regression**

**model. Interpret the training loss curve and understand how well the model fits the data.**

**PROGRAM:**

5) import tensorflow as tf

import numpy as np

import matplotlib.pyplot as plt

# Generate some random data for a linear regression problem

np.random.seed(42)

X = 2 \* np.random.rand(100, 1)

y = 4 + 3 \* X + np.random.randn(100, 1)

# Convert data to TensorFlow tensors

X\_tensor = tf.constant(X, dtype=tf.float32)

y\_tensor = tf.constant(y, dtype=tf.float32)

# Initialize variables (weights and bias)

W = tf.Variable(tf.random.normal(shape=(1, 1), mean=0.0, stddev=1.0), trainable=True)

b = tf.Variable(tf.zeros(shape=(1,)), trainable=True)

# Define the linear regression model

def linear\_regression(x):

return tf.matmul(x, W) + b

# Define the mean squared error loss function

def mean\_squared\_error(y\_true, y\_pred):

return tf.reduce\_mean(tf.square(y\_true - y\_pred))

# Set hyperparameters

learning\_rate = 0.01

num\_epochs = 100

# Optimization using gradient descent

for epoch in range(num\_epochs):

with tf.GradientTape() as tape:

predictions = linear\_regression(X\_tensor)

loss = mean\_squared\_error(y\_tensor, predictions)

# Calculate gradients

gradients = tape.gradient(loss, [W, b])

# Update weights and bias using gradient descent

W.assign\_sub(learning\_rate \* gradients[0])

b.assign\_sub(learning\_rate \* gradients[1])

# Print the loss every 10 epochs

if (epoch + 1) % 10 == 0:

print(f'Epoch {epoch + 1}/{num\_epochs}, Loss: {loss.numpy()}')

# Plot the data points and the linear regression line

plt.scatter(X, y, label='Data points')

plt.plot(X, linear\_regression(X\_tensor).numpy(), color='red', label='Linear regression')

plt.xlabel('X')

plt.ylabel('y')

plt.legend()

plt.show()

**Result:**

∙ Final hidden weights 1: Displays the weights of the connections between input layer

and first hidden layer.

∙ Final hidden weights 2: Displays the weights of the connections between first hidden

layer and second hidden layer.

∙ Final output weights: Displays the weights of the connections between second hidden

layer and output layer.