**11.Building Linear Regressor using ANN**

**Program Code:**

import tensorflow as tf

import numpy as np

import matplotlib.pyplot as plt

# Generate some random data for demonstration

np.random.seed(0)

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

y\_train = 2 \* X\_train + 1 + 0.1 \* np.random.randn(100, 1)

# Build the model

model = tf.keras.Sequential([

    tf.keras.layers.Dense(units=1, input\_shape=(1,))

])

# Compile the model

model.compile(optimizer='sgd', loss='mean\_squared\_error')

# Train the model

history = model.fit(X\_train, y\_train, epochs=100, verbose=0)

# Plot the training loss over epochs

plt.plot(history.history['loss'])

plt.xlabel('Epochs')

plt.ylabel('Mean Squared Error Loss')

plt.title('Training Loss')

plt.show()

# Make predictions on new data

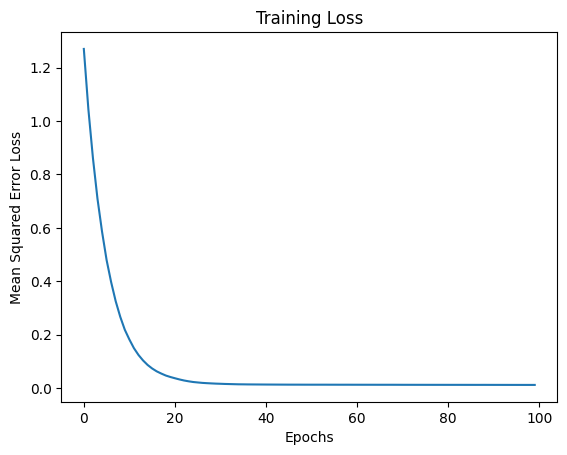
X\_test = np.array([[0.2], [0.5], [0.8]])

predictions = model.predict(X\_test)

# Print the predictions

for i in range(len(X\_test)):

    print(f"Input: {X\_test[i][0]}, Predicted Output: {predictions[i][0]}")

****1/1 [==============================] - 0s 116ms/step

Input: 0.2, Predicted Output: 1.3859466314315796

Input: 0.5, Predicted Output: 2.01832914352417

Input: 0.8, Predicted Output: 2.6507115364074707

**In this example, we generate some random data for training, with a linear relationship between X\_train and y\_train. The model consists of a single dense layer with one neuron, which is equivalent to a linear regression model. The optimizer used is Stochastic Gradient Descent (sgd), and the loss function is Mean Squared Error (mean\_squared\_error).**

**The training process is visualized by plotting the training loss over epochs. After training, the model is used to make predictions on new data (X\_test).**

**Note: Depending on your specific use case and requirements, you might need to customize the architecture, optimizer, and other hyperparameters.**