Developing a Neural Network Regression Model

AIM

To develop a neural network regression model for the given dataset.

THEORY

Regression problems involve predicting a continuous output variable based on input features. Traditional linear regression models often struggle with complex patterns in data. Neural networks, specifically feedforward neural networks, can capture these complex relationships by using multiple layers of neurons and activation functions. In this experiment, a neural network model is introduced with a single linear layer that learns the parameters weight and bias using gradient descent.

Neural Network Model

Include the neural network model diagram.

DESIGN STEPS

STEP 1: Generate Dataset

Create input values from 1 to 50 and add random noise to introduce variations in output values .

STEP 2: Initialize the Neural Network Model

Define a simple linear regression model using torch.nn.Linear() and initialize weights and bias values randomly.

STEP 3: Define Loss Function and Optimizer

Use Mean Squared Error (MSE) as the loss function and optimize using Stochastic Gradient Descent (SGD) with a learning rate of 0.001.

STEP 4: Train the Model

Run the training process for 100 epochs, compute loss, update weights and bias using backpropagation.

STEP 5: Plot the Loss Curve

Track the loss function values across epochs to visualize convergence.

STEP 6: Visualize the Best-Fit Line

Plot the original dataset along with the learned linear model.

STEP 7: Make Predictions

Use the trained model to predict for a new input value .

PROGRAM

Name: AKASH CT

REGISTERN NO: 212224240007

Register Number:

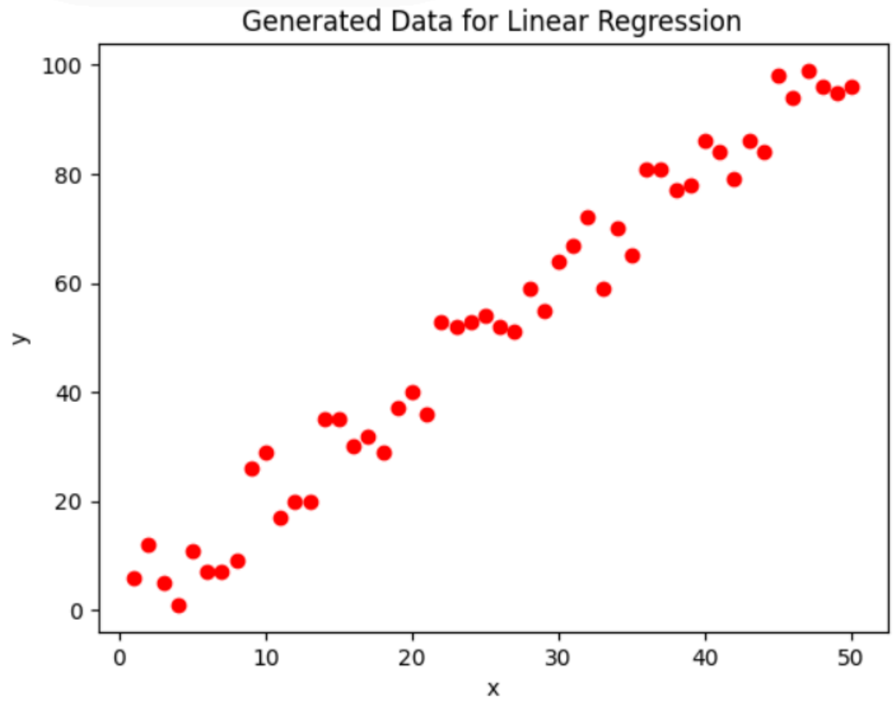
```
import torch
import torch.nn as nn
import numpy as np
import matplotlib.pyplot as plt
%matplotlib inline
```

```
x=torch.linspace(1,50,50).reshape(-1,1)
torch.manual_seed(9999)
e= torch.randint(low=-8, high=9, size=(50,1),dtype=torch.float)


y = 2*X + 1 + e
print(y.shape)

plt.scatter(X.numpy(), y.numpy(),color='red')
plt.xlabel('x')
plt.ylabel('y')
plt.title('Generated Data for Linear Regression')
plt.show()


torch.manual_seed(59)


class Model(nn.Module):
    def __init__(self, in_features, out_features):
        super().__init__()
        self.linear = nn.Linear(in_features, out_features)

    def forward(self, x):
        y_pred = self.linear(x)
        return y_pred


torch.manual_seed(59)
model = Model(1, 1)
print('Weight:', model.linear.weight.item())
print('Bias:  ', model.linear.bias.item())

loss_function = nn.MSELoss()

optimizer = torch.optim.SGD(model.parameters(), lr=0.0001)

epochs = 50
losses = []

for epoch in range(1, epochs + 1):
    optimizer.zero_grad()
    y_pred = model(X)
    loss = loss_function(y_pred, y)
    losses.append(loss.item())

    loss.backward()
    optimizer.step()
```

```
plt.plot(range(epochs), losses)
plt.ylabel('Loss')
plt.xlabel('epoch');
plt.show()


x1 = torch.tensor([X.min().item(), X.max().item()])


w1, b1 = model.linear.weight.item(), model.linear.bias.item()


y1 = x1 * w1 + b1


print(f'Final Weight: {w1:.8f}, Final Bias: {b1:.8f}')
print(f'X range: {x1.numpy()}')
print(f'Predicted Y values: {y1.numpy()}')


plt.scatter(X.numpy(), y.numpy(), label="Original Data")
plt.plot(x1.numpy(), y1.numpy(), 'r', label="Best-Fit Line")
plt.xlabel('x')
plt.ylabel('y')
plt.title('Trained Model: Best-Fit Line')
plt.legend()
plt.show()
```

Dataset Information

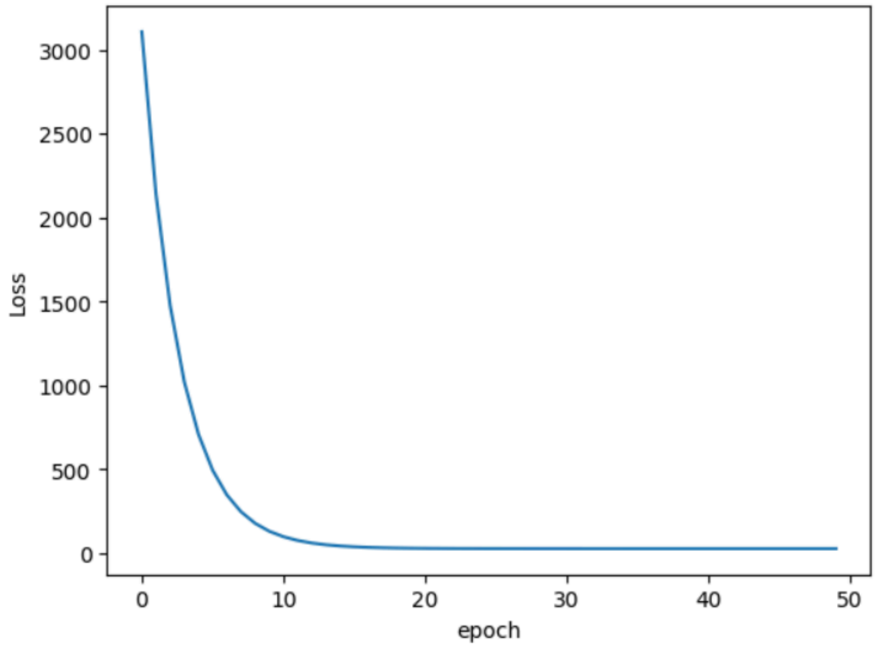
```
torch.Size([50, 1])
```



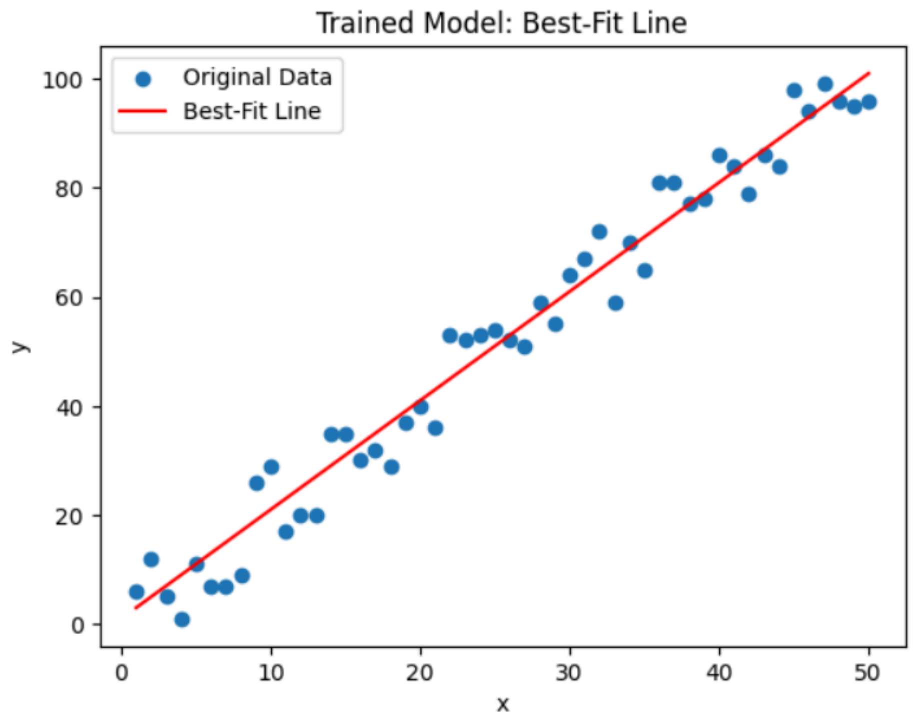
OUTPUT

Training Loss Vs Iteration Plot

```
Weight: 0.10597813129425049
Bias:   0.9637961387634277
```



Best Fit line plot



New Sample Data Prediction

SAMPLE INPUT

```
Weight: 0.10597813129425049
Bias:   0.9637961387634277
```

SAMPLE OUTPUT

```
Final Weight: 1.99847031, Final Bias: 1.01699948
X range: [ 1. 50.]
Predicted Y values: [   3.0154698 100.94051   ]
```

RESULT

Thus, a neural network regression model was successfully developed and trained using PyTorch.