

- [Getting Started with cuML's accelerator mode](#)
- [Linear regression with tf.keras using synthetic data](#)

Using Accelerated Hardware

- [TensorFlow with GPUs](#)
- [TPUs in Colab](#)

▼ Featured examples

- [Retraining an Image Classifier](#): Build a Keras model on top of a pre-trained image classifier to distinguish flowers.
- [Text Classification](#): Classify IMDB movie reviews as either *positive* or *negative*.
- [Style Transfer](#): Use deep learning to transfer style between images.
- [Multilingual Universal Sentence Encoder Q&A](#): Use a machine learning model to answer questions from the SQuAD dataset.
- [Video Interpolation](#): Predict what happened in a video between the first and the last frame.

```
import torch

import torch.nn as nn

import matplotlib.pyplot as plt
%matplotlib inline

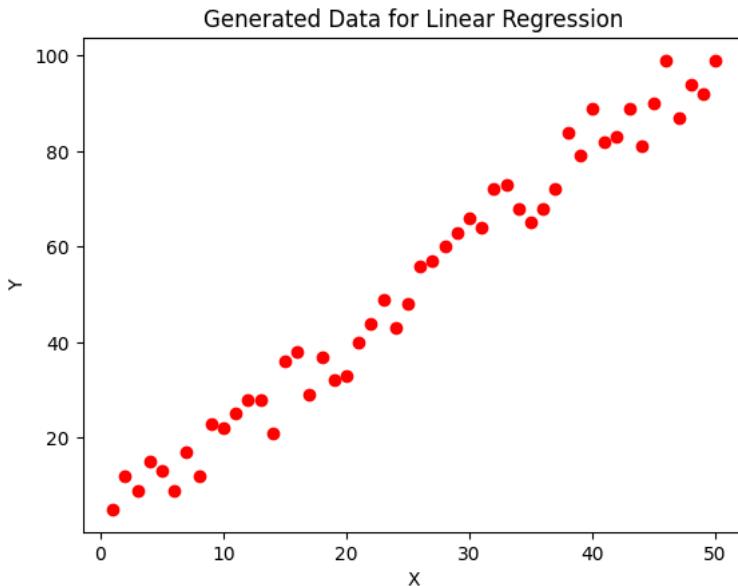
torch.manual_seed(71)
X=torch.linspace(1,50,50).reshape(-1,1)
e=torch.randint(-8,9,(50,1),dtype=torch.float)

e

tensor([[ 2.],
       [ 7.],
       [ 2.],
       [ 6.],
       [ 2.],
       [-4.],
       [ 2.],
       [-5.],
       [ 4.],
       [ 1.],
       [ 2.],
       [ 3.],
       [ 1.],
       [-8.],
       [ 5.],
       [ 5.],
       [-6.],
       [ 0.],
       [-7.],
       [-8.],
       [-3.],
       [-1.],
       [ 2.],
       [-6.],
       [-3.],
       [ 3.],
       [ 2.],
       [ 3.],
       [ 4.],
       [ 5.],
       [ 1.],
       [ 7.],
       [ 6.],
       [-1.],
       [-6.],
       [-5.],
       [-3.],
       [ 7.],
       [ 0.],
       [ 8.],
       [-1.],
       [-2.],
       [ 2.],
       [-8.],
       [-1.],
       [ 6.],
       [-8.],
       [-3.],
       [-7.],
       [-2.]])
```

```
y=2*X+1+e
```

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



```
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):
        return self.linear(x)
```

```
torch.manual_seed(59)
model=Model(1,1)
```

```
initial_weight=model.linear.weight.item()
initial_bias=model.linear.bias.item()
print("\nName: PRIYADHARSHINI RAJA")
```

Name: PRIYADHARSHINI RAJA

```
print("\nRegister Number: 212223230160")
```

Register Number: 212223230160

```
print(f"\nInitial Weight: {initial_weight:.8f}, \nInitial Bias: {initial_bias:.8f}")
```

```
Initial Weight: 0.10597813,
Initial Bias: 0.96379614
```

```
loss_function=nn.MSELoss()
optimizer=torch.optim.SGD(model.parameters(),lr=0.001)
epochs=100
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()
    print(f"epoch: {epoch:2} loss: {loss.item():10.8f}"
          f"weight: {model.linear.weight.item():10.8f}"
          f"bias: {model.linear.bias.item():10.8f}")
```

```
plt.plot(range(epocns),losses,color='blue')
plt.xlabel("Epoch")
plt.ylabel("Loss")
plt.title("Loss Curve")
plt.show()
```

```
final_weight=model.linear.weight.item()
final_bias=model.linear.bias.item()
print("\nName: PRIYADHARSHINI RAJA")
```

Name: PRIYADHARSHINI RAJA

```
print(f"\nFinal Weight: {final_weight:.8f}, Final Bias: {final_bias:.8f}")
```

Final Weight: 1.98277164, Final Bias: 1.09101629

Double-click (or enter) to edit

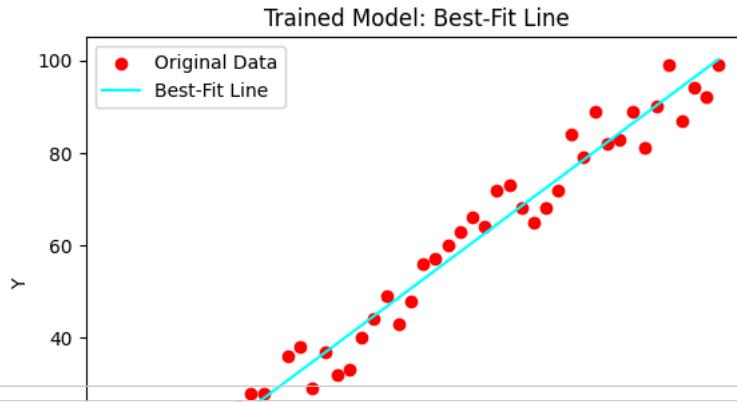
```
x1=torch.tensor([X.min().item(), X.max().item()])
y1=x1*final_weight+final_bias
print(x1)
```

```
tensor([ 1., 50.])
```

```
print(y1)
```

```
tensor([ 3.0738, 100.2296])
```

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



```
x_new=torch.tensor([[120.0]])
y_new_pred=model(x_new).item()
print("\nName: PRIYADHARSHINI RAJA")

Name: PRIYADHARSHINI RAJA
X
print(f"\nPredicted for x = 120: {y_new_pred:.8f}")

epoch: 29 loss: 21.06936836weight: 1.98439610bias: 1.04068720
epoch: 30 loss: 21.06883812weight: 1.98415291bias: 1.04140162
epoch: 31 loss: 21.06830788weight: 1.98429084bias: 1.04212701
epoch: 32 loss: 21.06778145weight: 1.98415494bias: 1.04284394
epoch: 33 loss: 21.06726074weight: 1.98421574bias: 1.04356635
epoch: 34 loss: 21.06674004weight: 1.98413551bias: 1.04428422
epoch: 35 loss: 21.06622505weight: 1.98415649bias: 1.04500473
epoch: 36 loss: 21.06570816weight: 1.98410451bias: 1.04572272
epoch: 37 loss: 21.06518745weight: 1.98410523bias: 1.04644191
epoch: 38 loss: 21.06466866weight: 1.98406804bias: 1.04715967
epoch: 39 loss: 21.06415749weight: 1.98405814bias: 1.04787791
epoch: 40 loss: 21.06363678weight: 1.98402870bias: 1.04859519
epoch: 41 loss: 21.06312561weight: 1.98401320bias: 1.04931259
epoch: 42 loss: 21.06260681weight: 1.98398757bias: 1.05002928
epoch: 43 loss: 21.06209564weight: 1.98396957bias: 1.05074584
epoch: 44 loss: 21.06157684weight: 1.98394585bias: 1.05146194
epoch: 45 loss: 21.06106949weight: 1.98392630bias: 1.05217779
epoch: 46 loss: 21.06055450weight: 1.98390377bias: 1.05289316
epoch: 47 loss: 21.06004333weight: 1.98388338bias: 1.05360830
epoch: 48 loss: 21.05953217weight: 1.98386145bias: 1.05432308
epoch: 49 loss: 21.05901337weight: 1.98384094bias: 1.05503750
epoch: 50 loss: 21.05850983weight: 1.98381913bias: 1.05575156
```