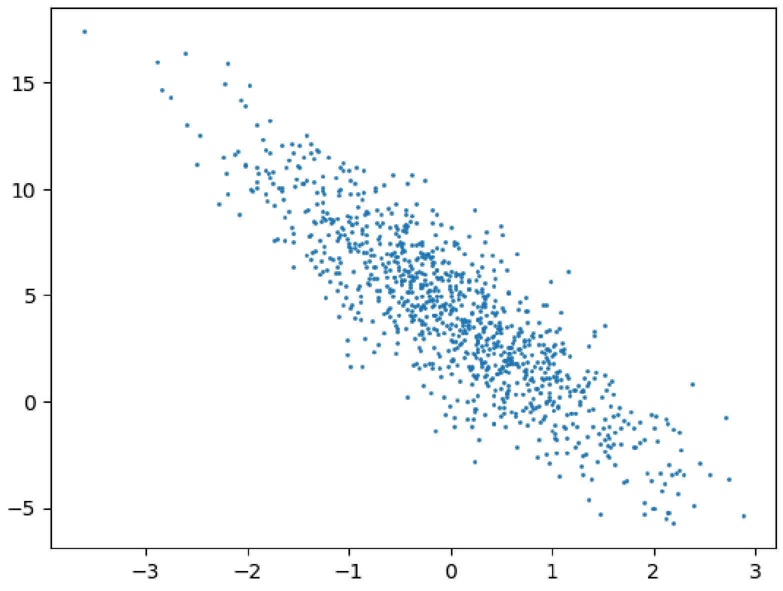
```
# Set up the environment
%matplotlib inline
import numpy as np
import torch
import matplotlib.pyplot as plt
from torch.utils.data import DataLoader, TensorDataset
```

```
# Data generation
def synthetic_data(w, b, num_examples):
    """Generate y = Xw + b + noise."""
    X = torch.normal(0, 1, (num_examples, len(w)))
    y = torch.matmul(X, w) + b
    y += torch.normal(0, 0.01, y.shape)
    return X, y.reshape((-1, 1))

true_w = torch.tensor([2, -3.4])
true_b = 4.2
features, labels = synthetic_data(true_w, true_b, 1000)
```

```
# Visualize generated data
plt.scatter(features[:, 1].numpy(), labels.numpy(), 1)
plt.show()
```



```
# Model definition
class LinearRegressionScratch:
    def __init__(self, num_inputs, lr):
        self.w = torch.normal(0, 0.01, size=(num_inputs, 1), requires_grad=True)
        self.b = torch.zeros(1, requires_grad=True)
        self.lr = lr

    def forward(self, X):
        return torch.matmul(X, self.w) + self.b

    def loss(self, y_hat, y):
        return ((y_hat - y) ** 2).mean() / 2

    def step(self):
        with torch.no_grad():
            self.w -= self.lr * self.w.grad
            self.b -= self.lr * self.b.grad
            self.w.grad.zero_()
            self.b.grad.zero_()
```

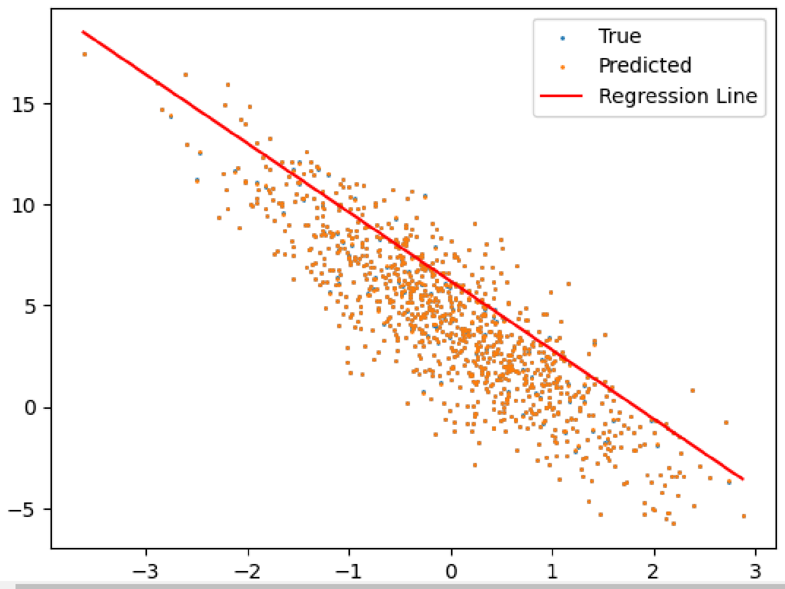
```
# DataLoader setup
batch_size = 10
dataset = TensorDataset(features, labels)
dataloader = DataLoader(dataset, batch_size, shuffle=True)


# Training loop
def train(model, dataloader, num_epochs):
    for epoch in range(num_epochs):
        for X, y in dataloader:
            # Forward pass
            y_hat = model.forward(X)
            l = model.loss(y_hat, y)

            # Backward pass
            l.backward()
            model.step()

        with torch.no_grad():
            train_l = model.loss(model.forward(features), labels)
            print(f'epoch {epoch + 1}, loss {train_l.item():f}')


# Instantiate and train the model
lr = 0.03
num_epochs = 5
model = LinearRegressionScratch(2, lr)
train(model, dataloader, num_epochs)
```

```
epoch 1, loss 0.042858
epoch 2, loss 0.000161
epoch 3, loss 0.000048
epoch 4, loss 0.000048
epoch 5, loss 0.000048
```

```
# Evaluate the model's performance
with torch.no_grad():
    print(f'Error in estimating w: {true_w - model.w.reshape(true_w.shape)}')
    print(f'Error in estimating b: {true_b - model.b}')
```

```
Error in estimating w: tensor([-0.0007, -0.0004])
Error in estimating b: tensor([0.0004])
```

```
# Visualize learned parameters vs true parameters
preds = model.forward(features).detach().numpy()

# Scatter plot of the true labels
plt.scatter(features[:, 1].numpy(), labels.numpy(), 1, label='True')

# Scatter plot of the predicted values
plt.scatter(features[:, 1].numpy(), preds, 1, label='Predicted')

# Create a smooth range of values for plotting the regression line
x_vals = torch.linspace(features[:, 1].min(), features[:, 1].max(), 100).reshape(-1, 1)  # Create 100 points between min and max
X_smooth = torch.cat((torch.ones_like(x_vals), x_vals), dim=1)  # Add ones for the intercept term

# Get predicted values for the smooth range
with torch.no_grad():
    y_smooth = model.forward(X_smooth).detach().numpy()

# Plot the regression line
plt.plot(x_vals.numpy(), y_smooth, color='red', label='Regression Line')

plt.legend()
plt.show()
```



Positive Slope (Left to Right): When the weight associated with the feature is positive, the line will slope upwards from left to right.

In this case, since the second feature has a significant negative weight (-3.4), it contributes to a downward slope, which is why the regression line moves from right to left.

Overall, the plot looks accurate given the data generated. The "Predicted" points align well with the "True" points, and the regression line fits them nicely, which is exactly what is in a linear regression model.