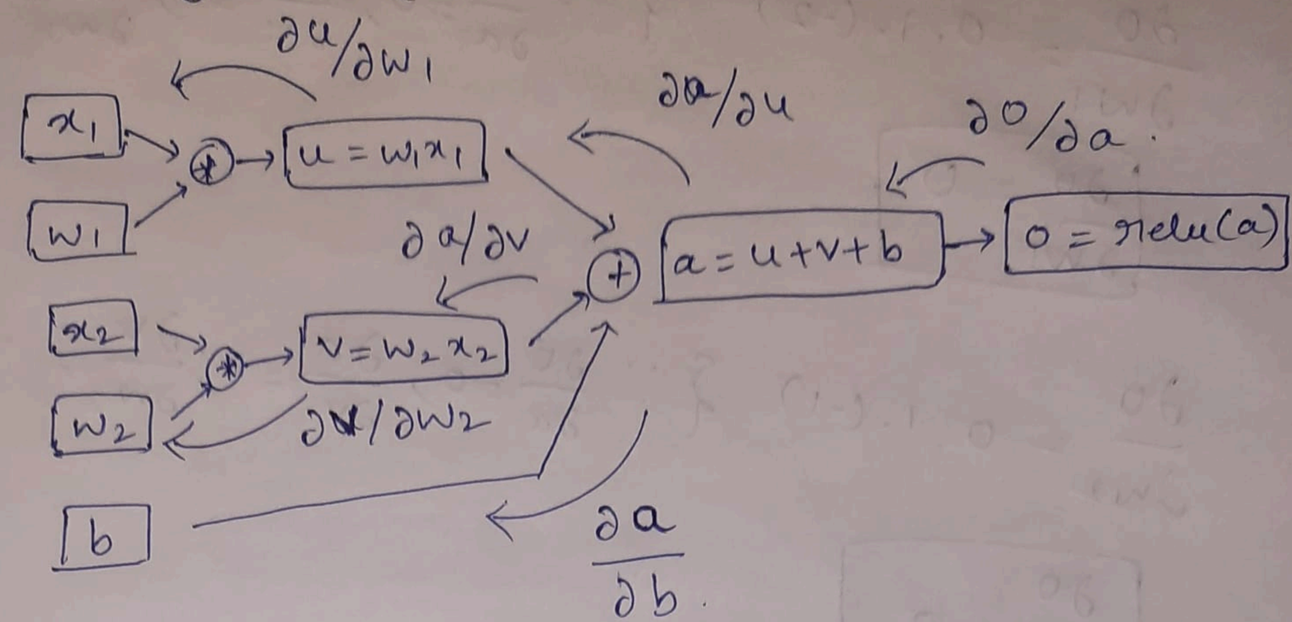
Lecture 09 Submission Pranav

1 Answer

Drawing the computation graph for the given equation and writing the derivatives for each

1a) $y = relu(w_1 x_1 + w_2 x_2 + b)$

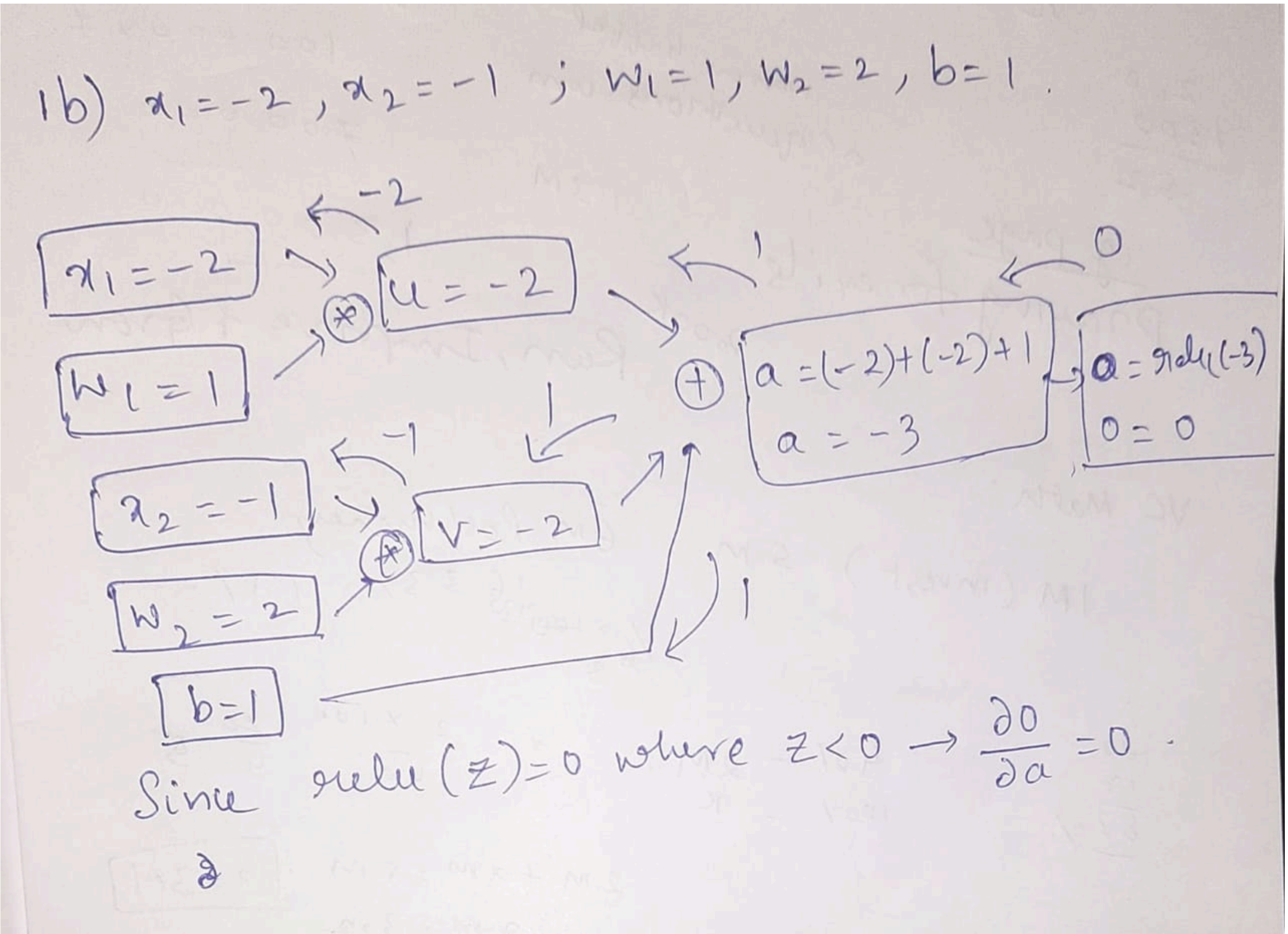


$$\frac{\partial o}{\partial w_1} = \frac{\partial o}{\partial a} \cdot \frac{\partial a}{\partial u} \cdot \frac{\partial u}{\partial w_1}$$

$$\frac{\partial o}{\partial w_2} = \frac{\partial o}{\partial a} \cdot \frac{\partial a}{\partial v} \cdot \frac{\partial v}{\partial w_2}$$

$$\frac{\partial o}{\partial b} = \frac{\partial o}{\partial a} \cdot \frac{\partial a}{\partial b}$$

1b - Substituting the given values into the graph and then using the above derivied gradients to compute values



$$\frac{\partial o}{\partial w_1} = 0 \cdot 1 \cdot (-2) \quad \left\{ \because \frac{\partial o}{\partial a} = 0;\ \frac{\partial a}{\partial u} = 1;\ \frac{\partial u}{\partial w_1} = (-2) \right\}$$

$$\boxed{\frac{\partial o}{\partial w_1} = 0}$$

$$\frac{\partial o}{\partial w_2} = 0 \cdot 1 \cdot (-1) \quad \left\{ \because \frac{\partial o}{\partial a} = 0;\ \frac{\partial a}{\partial v} = 1;\ \frac{\partial v}{\partial w_2} = -1 \right\}.$$

$$\boxed{\frac{\partial o}{\partial w_2} = 0}$$

$$\frac{\partial o}{\partial b} = 0 \cdot 1 \quad \left\{ \because \frac{\partial o}{\partial a} = 0;\ \frac{\partial a}{\partial b} = 1 \right\}.$$

$$\boxed{\frac{\partial o}{\partial b} = 0}$$

2 Answer

```
import torch
from torch.autograd import grad
import torch.nn.functional as F
from torch.nn import ReLU

import matplotlib.pyplot as plt
%matplotlib inline
```

```
import pandas as pd
import matplotlib.pyplot as plt
import torch
%matplotlib inline
```

```
df = pd.read_csv('./linreg-data.csv', index_col=0)
df.tail()
```

	x1	x2	y
995	-0.942094	-0.835856	-22.324428
996	1.222445	-0.403177	-52.121493
997	-0.112466	-1.688230	-57.043196
998	-0.403459	-0.412272	-27.701833
999	0.021351	-0.499017	-9.804714

```
# Assign features and target

X = torch.tensor(df[['x1', 'x2']].values, dtype=torch.float)
y = torch.tensor(df['y'].values, dtype=torch.float) + 144

# Shuffling & train/test split

torch.manual_seed(123)
shuffle_idx = torch.randperm(y.size(0), dtype=torch.long)

X, y = X[shuffle_idx], y[shuffle_idx]

percent70 = int(shuffle_idx.size(0)*0.7)

X_train, X_test = X[shuffle_idx[:percent70]], X[shuffle_idx[percent70:]]
y_train, y_test = y[shuffle_idx[:percent70]], y[shuffle_idx[percent70:]]

# Normalize (mean zero, unit variance)

mu, sigma = X_train.mean(dim=0), X_train.std(dim=0)
X_train = (X_train - mu) / sigma
X_test = (X_test - mu) / sigma
```

Computing gradients manually

Here I've taken X as an n x 2 matrix and initialised the weights and biases accordingly as well, I've added an extra relu function to simulate the model.

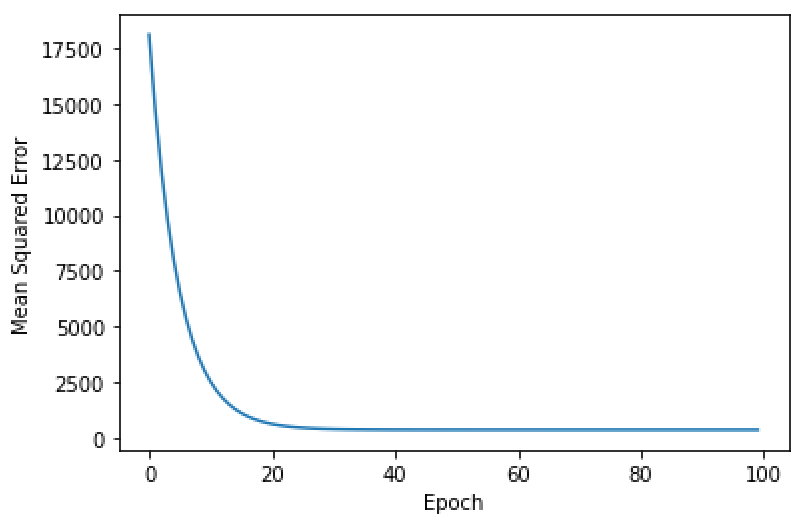
However since we added 144 to y tensor and all values are positive as well as the derivative of relu for positive values is 1, I just calculated the loss and the derivatives accordingly

```
class LinearRegression():
    def __init__(self, num_features):
        self.num_features = num_features
        self.weights = torch.rand(num_features, 1,
                                   dtype=torch.float)
        self.bias = torch.rand(1, dtype=torch.float)

    def forward(self, x):
        netinputs = torch.add(torch.mm(x, self.weights), self.bias)
        activations = netinputs
        relu = ReLU()
        return relu(activations).view(-1)

    def backward(self, x, yhat, y):

        grad_loss_yhat = 2*(yhat - y)

        grad_yhat_weights = x
        grad_yhat_bias = 1.

        grad_loss_weights =  torch.mm(grad_yhat_weights.t(),
                                         grad_loss_yhat.view(-1, 1)) / y.size(0)

        grad_loss_bias = torch.sum(grad_yhat_bias*grad_loss_yhat) / y.size(0)

        return (-1)*grad_loss_weights, (-1)*grad_loss_bias


def loss(yhat, y):
    return torch.mean((yhat - y)**2)


def train(model, x, y, num_epochs, learning_rate=0.01):
    cost = []
    for e in range(num_epochs):

        #### Compute outputs ####
        yhat = model.forward(x)

        #### Compute gradients ####
        negative_grad_w, negative_grad_b = model.backward(x, yhat, y)

        #### Update weights ####
        model.weights += learning_rate * negative_grad_w
        model.bias += learning_rate * negative_grad_b

        #### Logging ####
```

```
        yhat = model.forward(x) # not that this is a bit wasteful here
        curr_loss = loss(yhat, y)
        print('Epoch: %03d' % (e+1), end="")
        print(' | MSE: %.5f' % curr_loss)
        cost.append(curr_loss)

    return cost


model = LinearRegression(num_features=X_train.size(1))
cost = train(model,
             X_train, y_train,
             num_epochs=100,
             learning_rate=0.05)
```

```
Epoch: 083 | MSE: 371.61090
Epoch: 084 | MSE: 371.61084
Epoch: 085 | MSE: 371.61069
Epoch: 086 | MSE: 371.61066
Epoch: 087 | MSE: 371.61057
Epoch: 088 | MSE: 371.61050
Epoch: 089 | MSE: 371.61050
Epoch: 090 | MSE: 371.61050
Epoch: 091 | MSE: 371.61047
Epoch: 092 | MSE: 371.61041
Epoch: 093 | MSE: 371.61044
Epoch: 094 | MSE: 371.61041
Epoch: 095 | MSE: 371.61038
Epoch: 096 | MSE: 371.61038
Epoch: 097 | MSE: 371.61038
Epoch: 098 | MSE: 371.61038
Epoch: 099 | MSE: 371.61032
Epoch: 100 | MSE: 371.61038
```

```
plt.plot(range(len(cost)), cost)
plt.ylabel('Mean Squared Error')
plt.xlabel('Epoch')
plt.show()
```




```
train_pred = model.forward(X_train)
test_pred = model.forward(X_test)

print('Train MSE: %.5f' % loss(train_pred, y_train))
print('Test MSE: %.5f' % loss(test_pred, y_test))
```

```
Train MSE: 371.61038
Test MSE: 406.87973
```

```
print('Weights', model.weights)
print('Bias', model.bias)
```

```
Weights tensor([[ 0.3623],
        [37.8791]])
Bias tensor([143.4497])
```

2 Linear Regression Semi Manual

Computing the gradients semi manually

Here, I've defined the loss function and used the grad function from pytorch and used required_grad to calculate the gradients

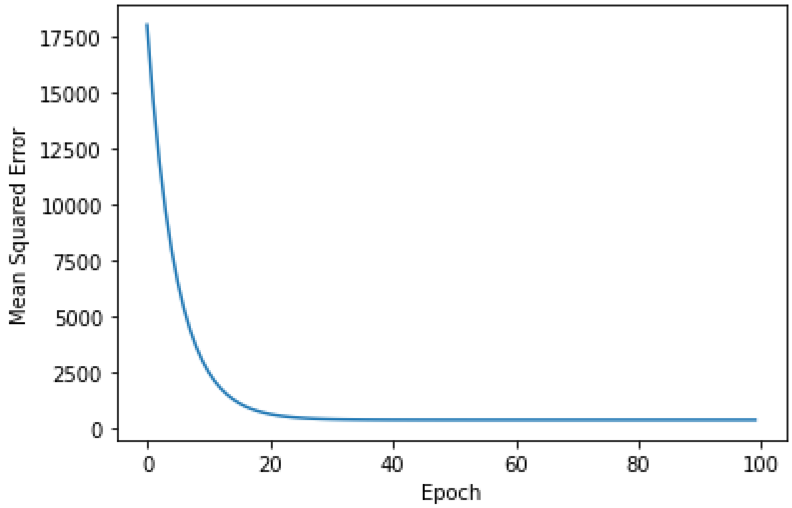
I used the retain_graph=True as well to retain the graph while calculating derivatives of weights and then using that graph for biases

```
class LinearRegression2():
    def __init__(self, num_features):
        self.num_features = num_features
        self.weights = torch.rand(num_features, 1,
                                   dtype=torch.float,
                                  requires_grad=True)
        self.bias = torch.rand(1, dtype=torch.float, requires_grad=True)

    def forward(self, x):
        netinputs = torch.add(torch.mm(x, self.weights), self.bias)
        activations = netinputs
        return F.relu(activations).view(-1)

    def backward(self, x, yhat, y):

        grad_loss_yhat = 2*(yhat - y)

        grad_yhat_weights = x
        grad_yhat_bias = 1.

        # Chain rule: inner times outer
        grad_loss_weights =  torch.mm(grad_yhat_weights.t(),
                                        grad_loss_yhat.view(-1, 1)) / y.size(0)

        grad_loss_bias = torch.sum(grad_yhat_bias*grad_loss_yhat) / y.size(0)

        # return negative gradient
        return (-1)*grad_loss_weights, (-1)*grad_loss_bias


def loss_func(yhat, y):
    return torch.mean((yhat - y)**2)
```

```
def train(model, x, y, num_epochs, learning_rate=0.01):
    cost = []
    for e in range(num_epochs):

        yhat = model.forward(x)
        loss = loss_func(yhat, y)

        #### Compute gradients using the grad function of pytorch ####
        negative_grad_w = grad(loss, model.weights, retain_graph=True)[0] * (-1)
        negative_grad_b = grad(loss, model.bias)[0] * (-1)

        model.weights = model.weights + learning_rate * negative_grad_w
        model.bias = model.bias + learning_rate * negative_grad_b

        #### Logging ####
        with torch.no_grad():
          yhat = model.forward(x)
          curr_loss = loss_func(yhat, y)
          print('Epoch: %03d' % (e+1), end="")
          print(' | MSE: %.5f' % curr_loss)
          cost.append(curr_loss)

    return cost


model = LinearRegression2(num_features=X_train.size(1))
cost = train(model,
             X_train, y_train,
             num_epochs=100,
             learning_rate=0.05)
```

```
Epoch: 063 | MSE: 371.64777
Epoch: 064 | MSE: 371.64066
Epoch: 065 | MSE: 371.63489
Epoch: 066 | MSE: 371.63028
Epoch: 067 | MSE: 371.62646
Epoch: 068 | MSE: 371.62338
Epoch: 069 | MSE: 371.62091
Epoch: 070 | MSE: 371.61890
Epoch: 071 | MSE: 371.61728
Epoch: 072 | MSE: 371.61600
Epoch: 073 | MSE: 371.61490
Epoch: 074 | MSE: 371.61404
Epoch: 075 | MSE: 371.61334
Epoch: 076 | MSE: 371.61279
Epoch: 077 | MSE: 371.61234
Epoch: 078 | MSE: 371.61197
Epoch: 079 | MSE: 371.61163
Epoch: 080 | MSE: 371.61136
Epoch: 081 | MSE: 371.61121
Epoch: 082 | MSE: 371.61105
Epoch: 083 | MSE: 371.61090
Epoch: 084 | MSE: 371.61081
Epoch: 085 | MSE: 371.61069
Epoch: 086 | MSE: 371.61063
Epoch: 087 | MSE: 371.61057
Epoch: 088 | MSE: 371.61053
Epoch: 089 | MSE: 371.61050
Epoch: 090 | MSE: 371.61044
Epoch: 091 | MSE: 371.61044
Epoch: 092 | MSE: 371.61044
Epoch: 093 | MSE: 371.61041
Epoch: 094 | MSE: 371.61044
Epoch: 095 | MSE: 371.61041
Epoch: 096 | MSE: 371.61038
Epoch: 097 | MSE: 371.61038
Epoch: 098 | MSE: 371.61038
Epoch: 099 | MSE: 371.61041
Epoch: 100 | MSE: 371.61032
```

```
plt.plot(range(len(cost)), cost)
plt.ylabel('Mean Squared Error')
plt.xlabel('Epoch')
plt.show()
```




```
train_pred = model.forward(X_train)
test_pred = model.forward(X_test)

print('Train MSE: %.5f' % loss(train_pred, y_train))
print('Test MSE: %.5f' % loss(test_pred, y_test))
```

```
Train MSE: 371.61032
Test MSE: 406.87982
```

```
print('Weights', model.weights)
print('Bias', model.bias)
```

```
Weights tensor([[ 0.3623],
        [37.8791]], grad_fn=<AddBackward0>)
Bias tensor([143.4498], grad_fn=<AddBackward0>)
```

2 Linear Regression Automatic

Here I've removed backward function completely and used the optimiser function from pytorch and then just used to step and compute the gradients as well.

```
class LinearRegression3(torch.nn.Module):
    def __init__(self, num_features):
        super(LinearRegression3, self).__init__()
        self.linear = torch.nn.Linear(num_features, 1)

    def forward(self, x):
        netinputs = self.linear(x)
        activations = netinputs
        return F.relu(activations).view(-1)
```

```
def train(model, x, y, num_epochs, learning_rate=0.01):
    cost = []

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

    for e in range(num_epochs):

        yhat = model.forward(x)
        loss = F.mse_loss(yhat, y)

        optimizer.zero_grad()

        loss.backward()

        optimizer.step()

        #### Logging ####
        with torch.no_grad():
            yhat = model.forward(x)
            curr_loss = F.mse_loss(yhat, y)
            print('Epoch: %03d' % (e+1), end="")
            print(' | MSE: %.5f' % curr_loss)
            cost.append(curr_loss)

    return cost


model = LinearRegression3(num_features=X_train.size(1))
cost = train(model,
             X_train, y_train,
             num_epochs=100,
             learning_rate=0.05)
```

```
Epoch: 001 | MSE: 18315.29297
Epoch: 002 | MSE: 14911.81543
Epoch: 003 | MSE: 12149.74121
Epoch: 004 | MSE: 9912.19141
Epoch: 005 | MSE: 8099.72217
Epoch: 006 | MSE: 6631.58008
Epoch: 007 | MSE: 5442.34912
Epoch: 008 | MSE: 4479.04297
Epoch: 009 | MSE: 3698.74146
Epoch: 010 | MSE: 3066.67676
Epoch: 011 | MSE: 2554.68774
Epoch: 012 | MSE: 2139.96460
Epoch: 013 | MSE: 1804.02747
Epoch: 014 | MSE: 1531.90967
Epoch: 015 | MSE: 1311.48657
Epoch: 016 | MSE: 1132.93799
Epoch: 017 | MSE: 988.30878
Epoch: 018 | MSE: 871.15466
Epoch: 019 | MSE: 776.25659
Epoch: 020 | MSE: 699.38641
```

```
Epoch: 021 | MSE: 637.11926
Epoch: 022 | MSE: 586.68115
Epoch: 023 | MSE: 545.82477
Epoch: 024 | MSE: 512.72974
Epoch: 025 | MSE: 485.92166
Epoch: 026 | MSE: 464.20621
Epoch: 027 | MSE: 446.61624
Epoch: 028 | MSE: 432.36777
Epoch: 029 | MSE: 420.82593
Epoch: 030 | MSE: 411.47665
Epoch: 031 | MSE: 403.90338
Epoch: 032 | MSE: 397.76883
Epoch: 033 | MSE: 392.79965
Epoch: 034 | MSE: 388.77441
Epoch: 035 | MSE: 385.51382
Epoch: 036 | MSE: 382.87262
Epoch: 037 | MSE: 380.73331
Epoch: 038 | MSE: 379.00027
Epoch: 039 | MSE: 377.59653
Epoch: 040 | MSE: 376.45938
Epoch: 041 | MSE: 375.53821
Epoch: 042 | MSE: 374.79208
Epoch: 043 | MSE: 374.18768
Epoch: 044 | MSE: 373.69809
Epoch: 045 | MSE: 373.30151
Epoch: 046 | MSE: 372.98029
Epoch: 047 | MSE: 372.72003
Epoch: 048 | MSE: 372.50925
Epoch: 049 | MSE: 372.33850
Epoch: 050 | MSE: 372.20020
Epoch: 051 | MSE: 372.08813
Epoch: 052 | MSE: 371.99738
Epoch: 053 | MSE: 371.92383
Epoch: 054 | MSE: 371.86432
Epoch: 055 | MSE: 371.81610
Epoch: 056 | MSE: 371.77701
Epoch: 057 | MSE: 371.74533
Epoch: 058 | MSE: 371.71970
```

Start coding or generate with AI.

Start coding or generate with AI.

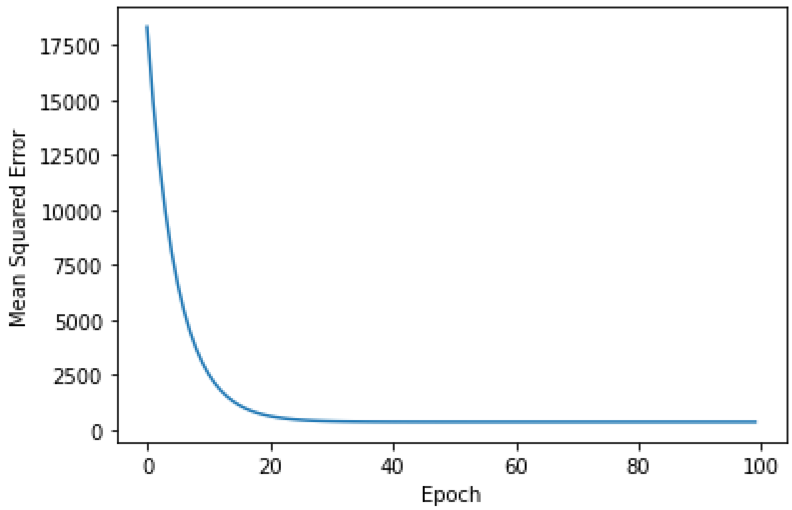
Start coding or generate with AI.

Start coding or generate with AI.

Start coding or generate with AI.

```
plt.plot(range(len(cost)), cost)
plt.ylabel('Mean Squared Error')
```

```
plt.xlabel('Epoch')
plt.show()
```



Start coding or generate with AI.

Start coding or generate with AI.

```
train_pred = model.forward(X_train)
test_pred = model.forward(X_test)

print('Train MSE: %.5f' % F.mse_loss(train_pred, y_train))
print('Test MSE: %.5f' % F.mse_loss(test_pred, y_test))
```

```
Train MSE: 371.61035
Test MSE: 406.88037
```

```
list(model.parameters())
```

```
[Parameter containing:
 tensor([[ 0.3622, 37.8791]], requires_grad=True), Parameter containing:
 tensor([143.4497], requires_grad=True)]
```

3 Answer

Writing the automatic version using the sequential class and computing the loss and showing it is no different

```
class LinearRegressionSequential(torch.nn.Module):
    def __init__(self, num_features):
```

```
        super(LinearRegressionSequential, self).__init__()

        self.my_network = torch.nn.Sequential(
            torch.nn.Linear(num_features, 1),
            torch.nn.ReLU()
        )


    def forward(self, x):
        netinputs = self.my_network(x)
        activations = netinputs
        return activations.view(-1)



def train(model, x, y, num_epochs, learning_rate=0.01):
    cost = []

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

    for e in range(num_epochs):

        yhat = model.forward(x)
        loss = F.mse_loss(yhat, y)

        optimizer.zero_grad()

        loss.backward()

        optimizer.step()

        #### Logging ####
        with torch.no_grad():
            yhat = model.forward(x)
            curr_loss = F.mse_loss(yhat, y)
            print('Epoch: %03d' % (e+1), end="")
            print(' | MSE: %.5f' % curr_loss)
            cost.append(curr_loss)

    return cost


model = LinearRegressionSequential(num_features=X_train.size(1))
cost = train(model,
             X_train, y_train,
             num_epochs=100,
             learning_rate=0.05)
```

```
Epoch: 001 | MSE: 20488.75781
Epoch: 002 | MSE: 16986.41211
Epoch: 003 | MSE: 13830.63574
Epoch: 004 | MSE: 11274.26953
```

```
Epoch: 005 | MSE: 9203.45898
Epoch: 006 | MSE: 7525.97559
Epoch: 007 | MSE: 6167.10986
Epoch: 008 | MSE: 5066.34521
Epoch: 009 | MSE: 4174.65527
Epoch: 010 | MSE: 3452.33032
Epoch: 011 | MSE: 2867.20117
Epoch: 012 | MSE: 2393.20850
Epoch: 013 | MSE: 2009.24353
Epoch: 014 | MSE: 1698.20642
Epoch: 015 | MSE: 1446.24585
Epoch: 016 | MSE: 1242.14075
Epoch: 017 | MSE: 1076.80212
Epoch: 018 | MSE: 942.86633
Epoch: 019 | MSE: 834.36884
Epoch: 020 | MSE: 746.47845
Epoch: 021 | MSE: 675.28107
Epoch: 022 | MSE: 617.60626
Epoch: 023 | MSE: 570.88525
Epoch: 024 | MSE: 533.03815
Epoch: 025 | MSE: 502.37915
Epoch: 026 | MSE: 477.54321
Epoch: 027 | MSE: 457.42416
Epoch: 028 | MSE: 441.12634
Epoch: 029 | MSE: 427.92389
Epoch: 030 | MSE: 417.22876
Epoch: 031 | MSE: 408.56479
Epoch: 032 | MSE: 401.54648
Epoch: 033 | MSE: 395.86099
Epoch: 034 | MSE: 391.25531
Epoch: 035 | MSE: 387.52438
Epoch: 036 | MSE: 384.50195
Epoch: 037 | MSE: 382.05365
Epoch: 038 | MSE: 380.07031
Epoch: 039 | MSE: 378.46365
Epoch: 040 | MSE: 377.16205
Epoch: 041 | MSE: 376.10776
Epoch: 042 | MSE: 375.25366
Epoch: 043 | MSE: 374.56174
Epoch: 044 | MSE: 374.00125
Epoch: 045 | MSE: 373.54718
Epoch: 046 | MSE: 373.17935
Epoch: 047 | MSE: 372.88138
Epoch: 048 | MSE: 372.64001
Epoch: 049 | MSE: 372.44443
Epoch: 050 | MSE: 372.28604
Epoch: 051 | MSE: 372.15771
Epoch: 052 | MSE: 372.05380
Epoch: 053 | MSE: 371.96954
Epoch: 054 | MSE: 371.90137
Epoch: 055 | MSE: 371.84607
Epoch: 056 | MSE: 371.80133
Epoch: 057 | MSE: 371.76508
Epoch: 058 | MSE: 371.73569
```

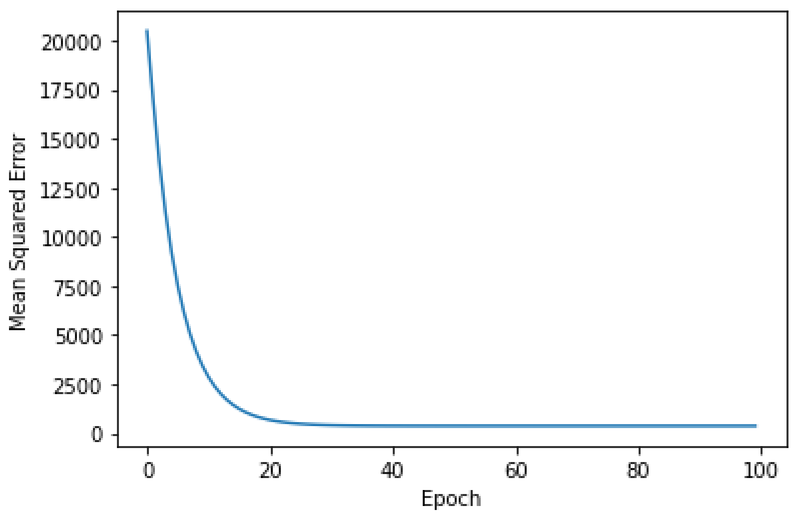
Start coding or generate with AI.

Start coding or generate with AI.

Start coding or generate with AI.

Start coding or generate with AI.

```
plt.plot(range(len(cost)), cost)
plt.ylabel('Mean Squared Error')
plt.xlabel('Epoch')
plt.show()
```



Start coding or generate with AI.

Start coding or generate with AI.

```
train_pred = model.forward(X_train)
test_pred = model.forward(X_test)

print('Train MSE: %.5f' % F.mse_loss(train_pred, y_train))
print('Test MSE: %.5f' % F.mse_loss(test_pred, y_test))
```