**Pratik Shukla**

Follow 88 Followers About

# Linear Regression With Gradient Descent From Scratch

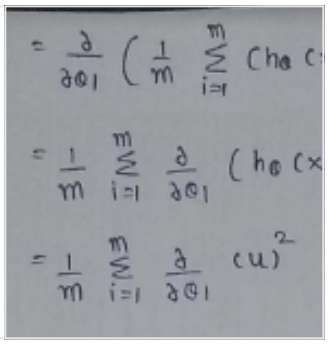
 Pratik Shukla May 17, 2020 · 5 min read

In the last article we saw that how the formula for finding the regression line with gradient descent works. In that article we started with some basic cost function and then made our way through our original cost function which was Mean Squared Error(MSE). I recommend you to read that article first,if you haven't already!

**What is Gradient Descent ?**

Gradient Descent is a machine learning algorithm which operates iteratively to find the optimal values for it's...

medium.com



**All my articles are available on my blog : patrickstar0110.blogspot.com**

Here we will use a slightly different cost function that MSE. Here we will use our basic Sum of Squared Residual (SSR) function to find the optimal parameter values. First we'll find the parameters for one iteration by hand. That will give us ample idea of how this algorithm works.

So what are we waiting for? **Let's dig in!**

## Gradient Descent For Linear Regression By Hand:

In this, I will take some random numbers to solve the problem. But it is also applicable for any datasets. Here we'll use the SSR cost function for ease of calculations.

(1) Dataset :

$$X = [2, 4, 5]$$

$$Y = [1.2, 2.8, 5.3]$$

(2) Initial parameter values:

$$LR = 0.001$$

$$\beta_0 = 0$$

$$\beta_1 = 1$$

(3) Cost function :

$$\text{cost} = \text{Sum of Squared Residuals (SSR)}$$

$$\text{cost} = J = \sum (Y - Y_{pred})^2$$

$$Y_{Pred} = \beta_0 + \beta_1 X$$

(5) Predicted values :

$$Y_{Pred1} = 0 + 1(2) = 2$$
$$Y_{Pred2} = 0 + 1(4) = 4$$
$$Y_{Pred3} = 0 + 1(5) = 5$$

(6) Calculating cost :

$$\text{Cost} = J = (1.2-2)^2 = 0.64$$
$$= (2.8-4)^2 = 1.44$$
$$= (5.3-5)^2 = 0.09$$
$$\text{Total cost} = 2.17$$

(7) Finding derivatives :

$$\frac{\partial}{\partial \beta_0} J = \frac{\partial}{\partial \beta_0} \left( \Sigma (Y - Y_{Pred})^2 \right)$$
$$= \frac{\partial}{\partial \beta_0} \left( \Sigma (u)^2 \right)$$

(8) Defining u :

$$u = Y - Y_{Pred}$$

(9) Calculating the derivative :

$$\frac{\partial}{\partial \beta_0} J = \frac{\partial}{\partial u} J \times \frac{\partial u}{\partial \beta_0}$$

(10) Simplifying :

$$\frac{\partial J}{\partial u} = \frac{\partial}{\partial u} (\Sigma (u)^2)$$

$$= 2 \Sigma u$$

$$\frac{\partial u}{\partial \beta_0} = \frac{\partial}{\partial \beta_0} (Y - Y_{pred})$$

$$= \frac{\partial}{\partial \beta_0} (Y - (\beta_0 + \beta_1 X))$$

$$= \frac{\partial}{\partial \beta_0} (Y - \beta_0 - \beta_1 X)$$

$$= \boxed{-1}$$

(11) Merging calculated values :

$$\frac{\partial}{\partial \beta_0} J = 2 \Sigma u \cdot -1$$

$$= -2 \Sigma u$$

$$= -2 \Sigma (Y - Y_{pred})$$

(12) Find the other parameter:

$$\frac{\partial}{\partial \beta_1} J = \frac{\partial}{\partial \beta_1} \left( \Sigma (Y - Y_{pred})^2 \right)$$

$$= \frac{\partial}{\partial \beta_1} \left( \Sigma (u)^2 \right)$$

(13) Simplifying:

$$\frac{\partial}{\partial \beta_1} J = \frac{\partial}{\partial u} J \times \frac{\partial u}{\partial \beta_1}$$

(14) Calculating :

$$\frac{\partial J}{\partial u} = \frac{\partial}{\partial u} \left( \Sigma (u)^2 \right)$$

$$= 2 \Sigma u$$

$$\frac{\partial u}{\partial \beta_1} = \frac{\partial}{\partial \beta_1} (Y - Y_{pred})$$

$$= \frac{\partial}{\partial \beta_1} (Y - (\beta_0 + \beta_1 x))$$

$$= \frac{\partial}{\partial \beta_1} (Y - \beta_0 - \beta_1 x)$$

$$= \boxed{-x}$$

$$\frac{\partial J}{\partial \beta_1} = 2\Sigma u \cdot -x$$

$$= -2 \times \Sigma u$$

$$= -2 \times \Sigma (Y - (\beta_0 + \beta_1 x))$$

(16) Calculating values for derivatives:

$$\frac{\partial J}{\partial \beta_0} = -2 \times (1.2 - (0 + 1(2)))$$

$$= -2 \times (-0.8)$$

$$= 1.6$$

$$\frac{\partial J}{\partial \beta_0} = -2 \times (2.8 - (0 + 1(4)))$$

$$= -2 \times -1.2$$

$$= 2.4$$

$$\frac{\partial J}{\partial \beta_0} = -2 \times (5.3 - (0 + 1(5)))$$

$$= -2 \times 0.3$$

$$= -0.6$$

$$\Sigma \frac{\partial J}{\partial \beta_0} = 1.6 + 2.4 - 0.6$$

$$= 3.4$$

$$\frac{\partial J}{\partial \beta_1} = -2 \cdot (2) \cdot (1.2 - [0 + 1(2)])$$
$$= -4 \cdot -0.8$$
$$= 3.2$$

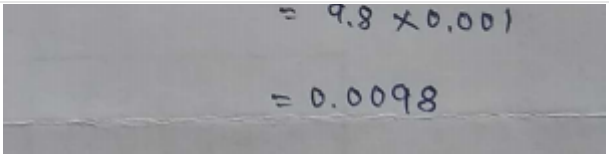
$$\frac{\partial J}{\partial \beta_1} = -2 \cdot (4) \cdot (2.8 - (0 + 1(4)))$$
$$= -8 \cdot -1.2$$
$$= 9.6$$

$$\frac{\partial J}{\partial \beta_1} = -2 \cdot (5) \cdot (5.3 - (0 + 1(5)))$$
$$= -10 \cdot 0.3$$
$$= -3$$

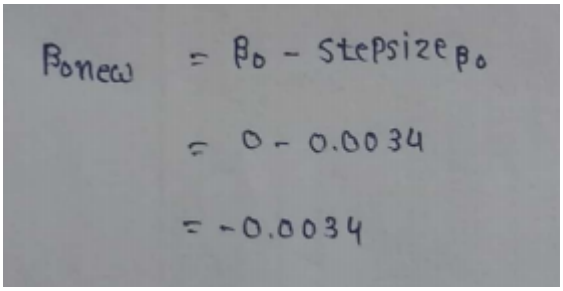
$$\sum \frac{\partial J}{\partial \beta_1} = 3.2 + 9.6 - 3$$
$$= 9.8$$

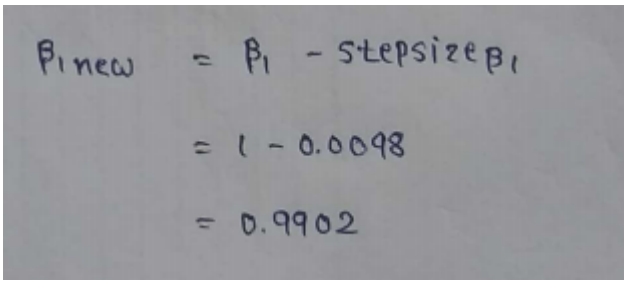
(18) Calculating the step size :

$$\text{Stepsize}_{\beta_0} = J\beta_0 \times LR$$
$$= 3.4 \times 0.001$$
$$= 0.0034$$




(19) Updating the values :




(20) Repeat the same process for all the iterations

. . .

Here we can see that if we are going to do 1000 iterations by hand, it is going to take forever for some slow kids like me. That's why we implement it in python! We will use the derived formulas and some "for" loops to write our python code.

Let's get statted!

Outline of the process :

(1) Initialize beta 0 and beta 1 values.

(2) Initialize learning rate and desired number of iterations.

(4) Initialize the variables which will hold the error for a particular iteration.

(5) Make prediction using the line equation.

(6) Calculate the error and append it to the error array.

(7) Calculate partial derivatives for both coefficients.

(8) Increase the cost of both coefficients (As there are 3 data points in our dataset.)

(9) Update the values of coefficients.

. . .

## # Let's code :

(1) Import some required libraries :

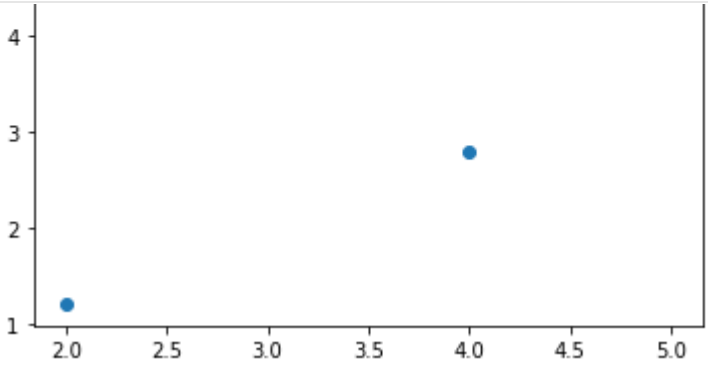
```
# Import required libraries :

import numpy as np
import matplotlib.pyplot as plt
```

(2) Define the dataset :

```
# Dataset :

X = np.array([2,4,5])
Y = np.array([1.2,2.8,5.3])
```

(3) Plot the data points :

```
# Plotting our dataset :

plt.scatter(X,Y)
plt.show()
```




(4) Main function to calculate values of coefficients :

```
# Initialize the parametes :

b0 = 0                # Intercept
b1 = 1                # Slope
lr = 0.001            # Learning Rate
iterations = 1000     # Number of iterations
error = []            # Error array to calculate cost for each iteartions

for itr in range(iterations):
    error_cost = 0
    cost_b0 = 0
    cost_b1 = 0

    for i in range(len(X)):
        y_pred = (b0+b1*X[i])    # Predict the value for given X

        error_cost = error_cost + (Y[i] - y_pred)**2   # Calculate the error in prediction for all 3 points.

        for j in range(len(X)):
            partial_wrt_b0 = -2 * (Y[j] - (b0 + b1*X[j]))        # Partial derivative 1
            partial_wrt_b1 = (-2*X[j]) *(Y[j] - (b0 + b1*X[j]))  # Partial derivative 2

            cost_b0 = cost_b0 + partial_wrt_b0                   # calculate cost for each number and add
            cost_b1 = cost_b1 + partial_wrt_b1                   # calculate cost for each number and add

        b0 = b0 - lr*cost_b0         # Update values
        b1 = b1 - lr*cost_b1         # Update values

    error.append(error_cost)         # Append the data in array
```

(5) Value of coefficients :

```
# Value of coefficient 1:

b0
```

```
-1.5675513508588914
```

```
# Value of coefficient 2:
```

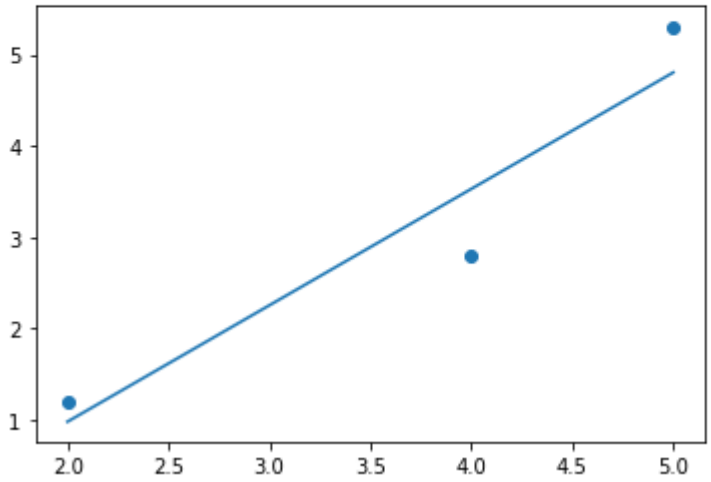
(6) Predicting the values :

```
# predict new values :

y_pred = b0 + b1*X
```

(7) Plot the regression line :

```
# Plot the regression line :
plt.scatter(X,Y)
plt.plot(X , y_pred)
```

```
[<matplotlib.lines.Line2D at 0xa96c524a88>]
```




(8) Predicting values :

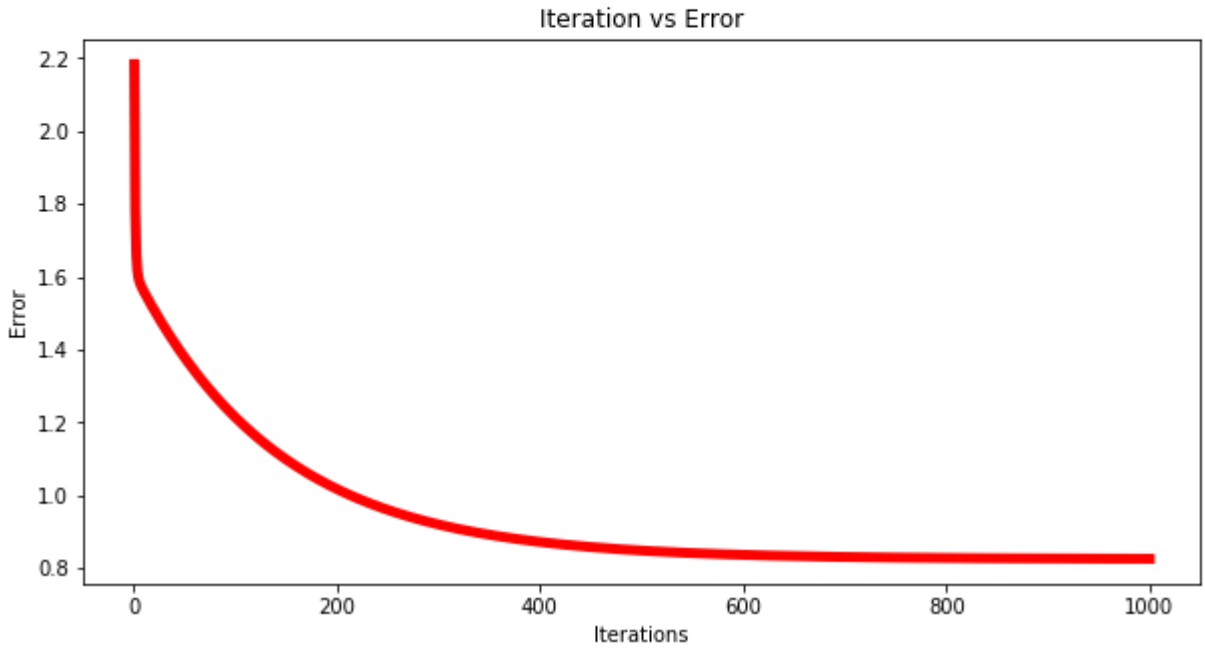
```
# predicting new value :

y_new_pred = b0 + b1*3
print (y_new_pred)
```

```
2.2550946738005173
```

(9) Plotting the error for each iterations :

```
plt.xlabel( 'Iterations' )
plt.ylabel("Error")
```

```
Text(0, 0.5, 'Error')
```




That's it. We did it! We have our optimal parameters for 1000 iterations and decreased error. However we can see that this method is less efficient if we take into account only a few iterations(i.e. 10).

. . .

**To find more such detailed explanation, visit my blog:**
**patrickstar0110.blogspot.com**

(1) Simple Linear Regression Explained With It's Derivation:
https://youtu.be/od2boSsFtnY

(2)How to Calculate The Accuracy Of A Model In Linear Regression From Scratch :
https://youtu.be/bM3KmaghclY

(3) Simple Linear Regression Using Sklearn :
https://youtu.be/_VGjHF1X9oU

Machine Learning | Gradient Descent | Algorithms | Regression | Derivation