# Objectives

* Introduction to Gradient Descent
* Implement Linear regression using Gradient Descent

# Lab Tool(s)

<https://www.kaggle.com/>

# Lab Deliverables

Submit a pdf document to Blackboard containing your solution to the lab assessment at the end of this document.

# Introduction to Gradient Descent

Gradient Descent is a very generic optimization algorithm capable of finding optimal

solutions to a wide range of problems. The general idea of Gradient Descent is to

tweak parameters iteratively in order to minimize a cost function.

Suppose you are lost in the mountains in a dense fog; you can only feel the slope of

the ground below your feet. A good strategy to get to the bottom of the valley quickly

is to go downhill in the direction of the steepest slope. This is exactly what Gradient

Descent does: it measures the local gradient of the error function with regards to the

parameter vector θ, and it goes in the direction of descending gradient. Once the gradient

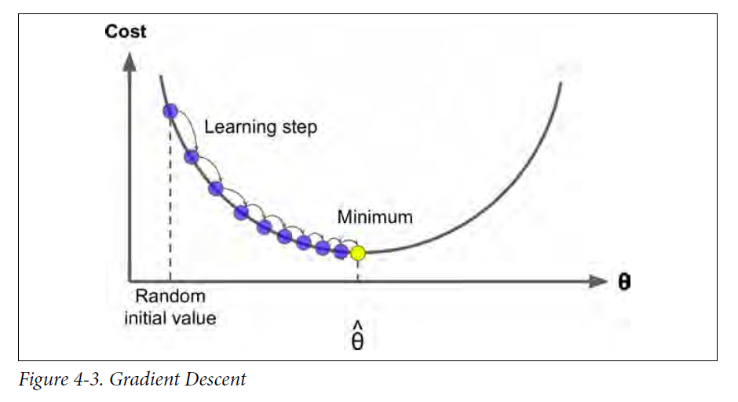
is zero, you have reached a minimum!

Concretely, you start by filling θ with random values (this is called random initialization),

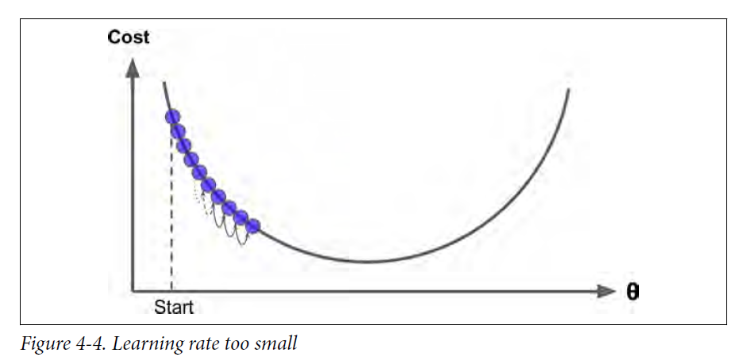
and then you improve it gradually, taking one baby step at a time, each step

attempting to decrease the cost function (e.g., the MSE), until the algorithm converges

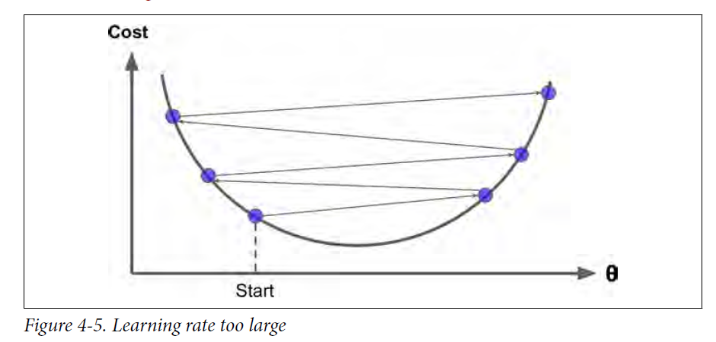
to a minimum.



An important parameter in Gradient Descent is the size of the steps, determined by the learning rate hyperparameter. If the learning rate is too small, then the algorithm will have to go through many iterations to converge, which will take a long time.



On the other hand, if the learning rate is too high, you might jump across the valley and end up on the other side, possibly even higher up than you were before. This might make the algorithm diverge, with larger and larger values, failing to find a good solution.



Finally, not all cost functions look like nice regular bowls. There may be holes, ridges, plateaus, and all sorts of irregular terrains, making convergence to the minimum very difficult.

**Note:** When using Gradient Descent, you should ensure that all features have a **similar scale** (e.g., using Scikit-Learn’s StandardScaler class), or else it will take much longer to converge.

# Implementing ‘Batch’ Gradient Descent

To implement gradient descent, we need to compute the **gradient of the cost function** with regards to each model parameter θj. In other words, you need to calculate how much the cost function will change if you change θj just a little bit. This is called a *partial derivative*. It is like asking “what is the slope of the mountain under my feet if I face east?” and then asking the same question facing north (and so on for all other dimensions, if you can imagine a universe with more than three dimensions).

In Batch gradient descent, in each gradient step, the calculations involve all training set. Thus, it is slow on large training sets (i.e., when the number records/rows is large)

Then, we subtract the gradient from the current theta, taking into account the learning rate. This step is repeated for a number of iterations, or until convergence.

**Step1:** Generate data

**import** **numpy** **as** **np**

m = 100 # size of training data

X = 2 \* np.random.rand(m, 1)

y = 4 + 3 \* X + np.random.randn(m, 1)

X\_b = np.c\_[np.ones((m, 1)), X] *# add x0 = 1 to each instance*

**Step1:** Implement Batch Gradient Descent

eta = 0.1 # learning rate

n\_iterations = 1000 # number of iterations

theta = np.random.randn(2,1) # random initialization

**for** iteration **in** range(n\_iterations):

gradients = 2/m \* X\_b.T.dot(X\_b.dot(theta) - y)

theta = theta - eta \* gradients

**Step2:** Check theta and compare it to the original (remember, *y* = 4 + 3*x*1 + Gaussian noise)

theta

The first value represents theta\_0 and the second is for theta\_1. Remember that the true ones are 4 and 3 respectively. How did the learned ones compare to the true ones?

# Lab Assessment

**Step1:** Create a new notebook and name it “CCAI312\_YOURSTUDENTID\_Lab5”

**Step2:** Generate the following data

num\_hours\_studied = np.array([1, 3, 3, 4, 5, 6, 7, 7, 8, 8, 10])

exam\_score = np.array([18, 26, 31, 40, 55, 62, 71, 70, 75, 85, 97])

where num\_hours\_studied represents X and exam\_score is y.

**Step3:** visualize the data using scatter plots

**Step4:** Fit a linear regression using Batch Gradient Descent and plot the results. Report the learned theta.

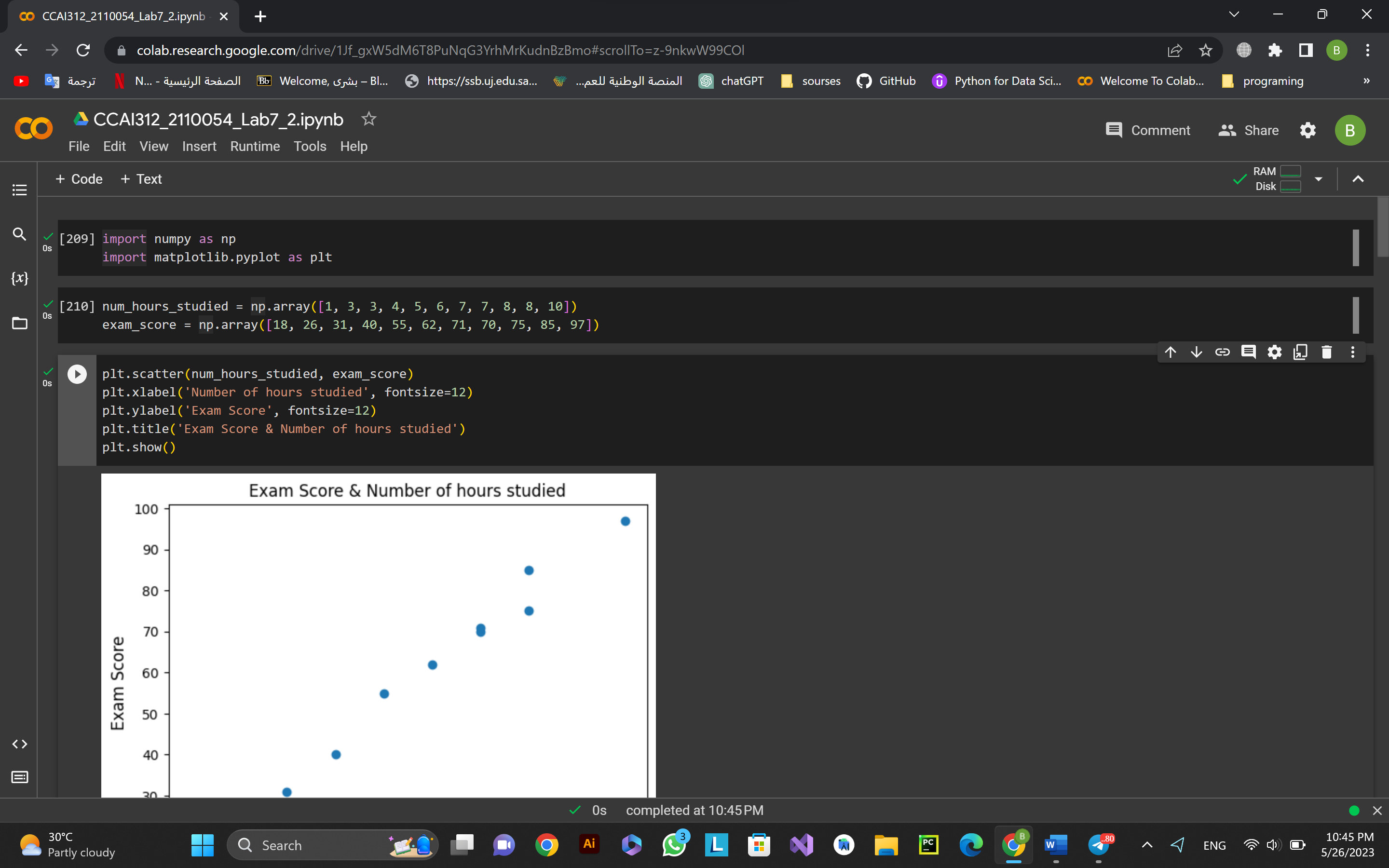
**Step5:** Fit a linear regression using Batch Gradient Descent with a different learning rate and plot the results. Report the learned theta. Repeat this step with two different learning rates (0.02 and 0.5). What did you observe?

**Step5:** Fix the learning rate to 0.1. Then, fit a linear regression using Batch Gradient Descent with number of iterations = 10. Report the learned theta. Repeat this step with two different number of iterations of your choice. What do you observe?

**Step6:** Submit a pdf document containing **your code and answers** to Blackboard, name the file as: CCAI312\_YOURSTUDENTID\_Lab5.pdf.

References:

Hands-on machine learning with sickit-learn and tensor flow, Chapter 4



A screenshot of a computer

Description automatically generated

A screenshot of a computer

Description automatically generated

A screenshot of a computer

Description automatically generated

A screenshot of a computer

Description automatically generated

A screenshot of a computer

Description automatically generated

A screenshot of a computer

Description automatically generated

A screenshot of a computer

Description automatically generated

A screenshot of a computer

Description automatically generated

A screenshot of a computer

Description automatically generated

A screenshot of a computer

Description automatically generated

A screenshot of a computer

Description automatically generated

A screenshot of a computer

Description automatically generated