Table of Contents

[Review 1](#_Toc123565844)

[Linear Regression Review 1](#_Toc123565845)

[Stochastic Gradient Descent (SGD) Regression 3](#_Toc123565846)

[SGD Scaling 4](#_Toc123565847)

[Epochs 5](#_Toc123565848)

[Logistic Regression Review 5](#_Toc123565849)

[SGDClassifier 8](#_Toc123565850)

[A Simplified Explanation of the Gradient Descent Algorithm 8](#_Toc123565851)

[The Loss (Cost) Function 9](#_Toc123565852)

[Implementing Gradient Descent 11](#_Toc123565853)

[Calculating the Derivative 12](#_Toc123565854)

[The Learning Rate 14](#_Toc123565855)

[Gradient Descent Summary 19](#_Toc123565856)

[Stochastic Gradient Descent 19](#_Toc123565857)

Welcome back ☺

## Review

This term will actually be quite a bit different than the last one but the difference

will not be apparent today. The content in this document will seem excessive at a first glance

but I will strive to keep it manageable. I will put things into context during class so you

can get through it quickly.

A goal of this document is to provide a quick review how to apply regression and classification

routines but perfect recall is not expected. We will also introduce the Stochastic GradientDescent

to perform regression and classification just like we have done with OLS and logistic regression.

You do not need to understand the new routines at a deep level. We will examine these

algorithms in more detail later.

The reason we are spending time on gradient descent algorithms now is because they

are a foundation for other machine learning algorithms which we will introduce in the course

in the near future.

## Linear Regression Review

Example : Linear Regression

Hopefully you remember ordinary least squares, OLS, to create a best fit line. The very first example of multiple regression that we covered in COMP3948 was used to predict wine quality.

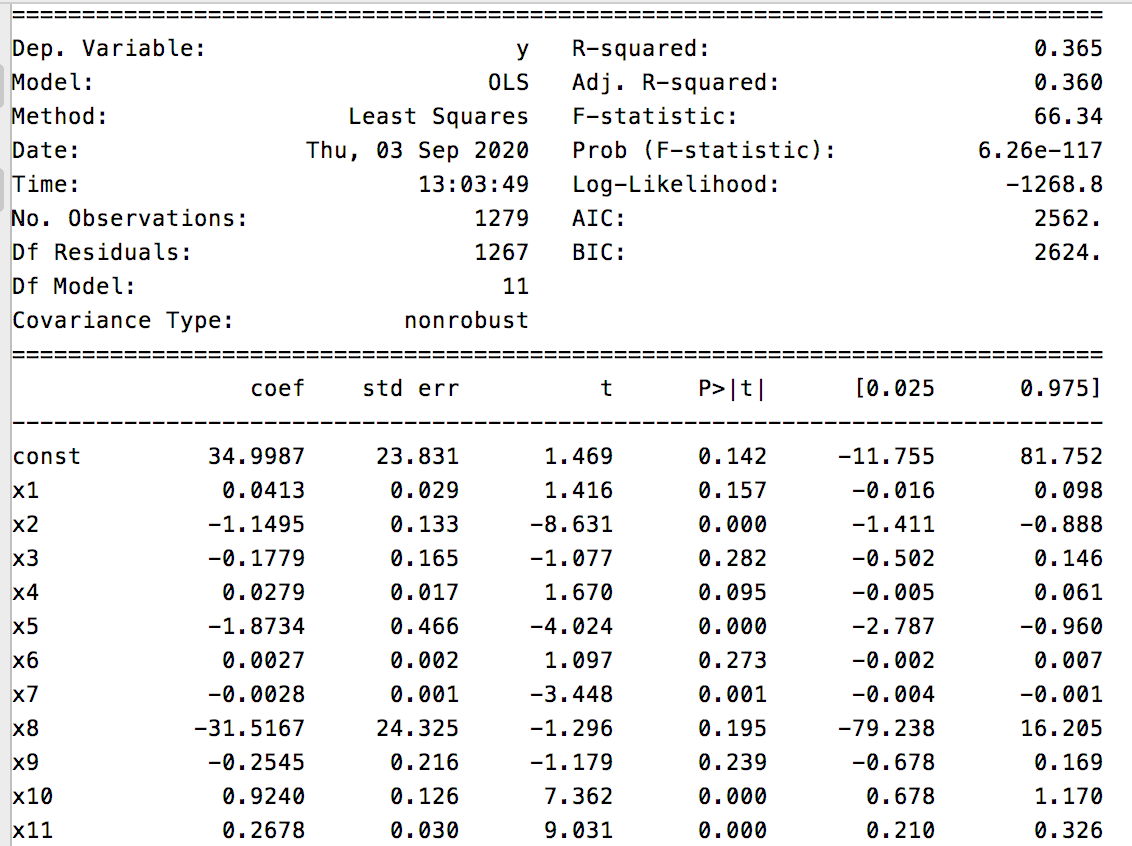
Here is the code.

|  |
| --- |
| import pandas as pd  import numpy as np  from sklearn import metrics  from sklearn.model\_selection import train\_test\_split  import statsmodels.api as sm  PATH = "/Users/pm/Desktop/DayDocs/data/"  CSV\_DATA = "winequality.csv"  dataset = pd.read\_csv(PATH + CSV\_DATA,  skiprows=1, # Don't include header row as part of data.  encoding = "ISO-8859-1", sep=',',  names=('fixed acidity', 'volatile acidity', 'citric acid',  'residual sugar', 'chlorides', 'free sulfur dioxide',  'total sulfur dioxide', 'density', 'pH', 'sulphates',  'alcohol', 'quality'))  # Show all columns.  pd.set\_option('display.max\_columns', None)  # Increase number of columns that display on one line.  pd.set\_option('display.width', 1000)  print(dataset.head())  print(dataset.describe())  X = dataset[['fixed acidity', 'volatile acidity', 'citric acid', 'residual sugar',  'chlorides', 'free sulfur dioxide', 'total sulfur dioxide', 'density',  'pH', 'sulphates','alcohol']].values  # Adding an intercept \*\*\* This is required \*\*\*. Don't forget this step.  # The intercept centers the error residuals around zero  # which helps to avoid over-fitting.  X = sm.add\_constant(X)  y = dataset['quality'].values  X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.2, random\_state=0)  model = sm.OLS(y\_train, X\_train).fit()  predictions = model.predict(X\_test) # make the predictions by the model  print(model.summary())  print('Root Mean Squared Error:',  np.sqrt(metrics.mean\_squared\_error(y\_test, predictions))) |

The result generates a root mean square error (RMSE) of **0.620**. The root mean square error represents the average deviation from the mean. Where y is the actual value and is the predicted value:

RMSE =

Here is the model summary which shows that , the coefficient of determination is **0.365**.



Exercise (2 marks)

Modify the code so only the statistically significant predictor variables are included. Show your code here:

|  |
| --- |
| import pandas as pd import numpy as np from sklearn import metrics from sklearn.model\_selection import train\_test\_split import statsmodels.api as sm from pathlib import Path  PATH = Path("C:\datasets\winequality.csv")   dataset = pd.read\_csv(PATH,  skiprows=1, # Don't include header row as part of data.  encoding = "ISO-8859-1", sep=',',  names=('fixed acidity', 'volatile acidity', 'citric acid',  'residual sugar', 'chlorides', 'free sulfur dioxide',  'total sulfur dioxide', 'density', 'pH', 'sulphates',  'alcohol', 'quality')) # Show all columns. pd.set\_option('display.max\_columns', None)  # Increase number of columns that display on one line. pd.set\_option('display.width', 1000) print(dataset.head()) print(dataset.describe()) X = dataset[[ 'volatile acidity',  'chlorides', 'total sulfur dioxide', 'sulphates','alcohol']].values  # Adding an intercept \*\*\* This is required \*\*\*. Don't forget this step. # The intercept centers the error residuals around zero # which helps to avoid over-fitting. X = sm.add\_constant(X) y = dataset['quality'].values  X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.2, random\_state=0)  model = sm.OLS(y\_train, X\_train).fit() predictions = model.predict(X\_test) # make the predictions by the model print(model.summary()) print('Root Mean Squared Error:',  np.sqrt(metrics.mean\_squared\_error(y\_test, predictions))) |

What is the root mean squared error after making the changes?

|  |
| --- |
| Root Mean Squared Error: 0.6259157889490543 |

## Stochastic Gradient Descent (SGD) Regression

Stochastic gradient descent is one of the most fundamental machine learning algorithms. We need to cover it early because it will form the basis for many other more advanced machine learning algorithms. We will not discuss it in much detail for this class but we will cover it more detail during the lab.

### SGD Scaling

Stochastic Gradient Descent is sensitive to feature scaling so it is highly recommended to scale your data. For example, scale each attribute on the input vector X to [0,1] or [-1,+1], or standardize it to have mean 0 and variance 1. Note that the same scaling must be applied to the test vector to obtain meaningful results. This task is often performed using StandardScaler.

**Note:** Remember to use the training set to fit the scaler. Also remember that only the x-values are scaled.

|  |
| --- |
| # Stochastic gradient descent models are sensitive to differences  # in scale so a StandardScaler is usually used.  from sklearn.preprocessing import StandardScaler  scaler = StandardScaler()  scaler.fit(X\_train) # Don't cheat - fit only on training data  X\_trainScaled = scaler.transform(X\_train)  X\_testScaled = scaler.transform(X\_test) |

Example : Stochastic Gradient Descent

This code generates a regression model to predict the wine quality using stochastic gradient descent with the **SGDRegressor** class. The root mean square error from this process is varies between roughly **0.619** and **0.622** each time the program is run. The RMSE is actually very similar to the linear regression. Notice how scaling is used to tame the magnitude of the predictor variables.

To build this example, add this code onto the end of Example 1.

|  |
| --- |
| ###########################################################  print("\nStochastic Gradient Descent")  from sklearn.linear\_model import SGDRegressor  from sklearn.metrics import mean\_squared\_error  # Stochastic gradient descent models are sensitive to differences  # in scale so a StandardScaler is usually used.  from sklearn.preprocessing import StandardScaler  scaler = StandardScaler()  scaler.fit(X\_train) # Don't cheat - fit only on training data  X\_trainScaled = scaler.transform(X\_train)  X\_testScaled = scaler.transform(X\_test)  # SkLearn SGD classifier  sgd = SGDRegressor(verbose=1)  sgd.fit(X\_trainScaled, y\_train)  predictions = sgd.predict(X\_testScaled)  print('Root Mean Squared Error:',  np.sqrt(mean\_squared\_error(y\_test, predictions))) |

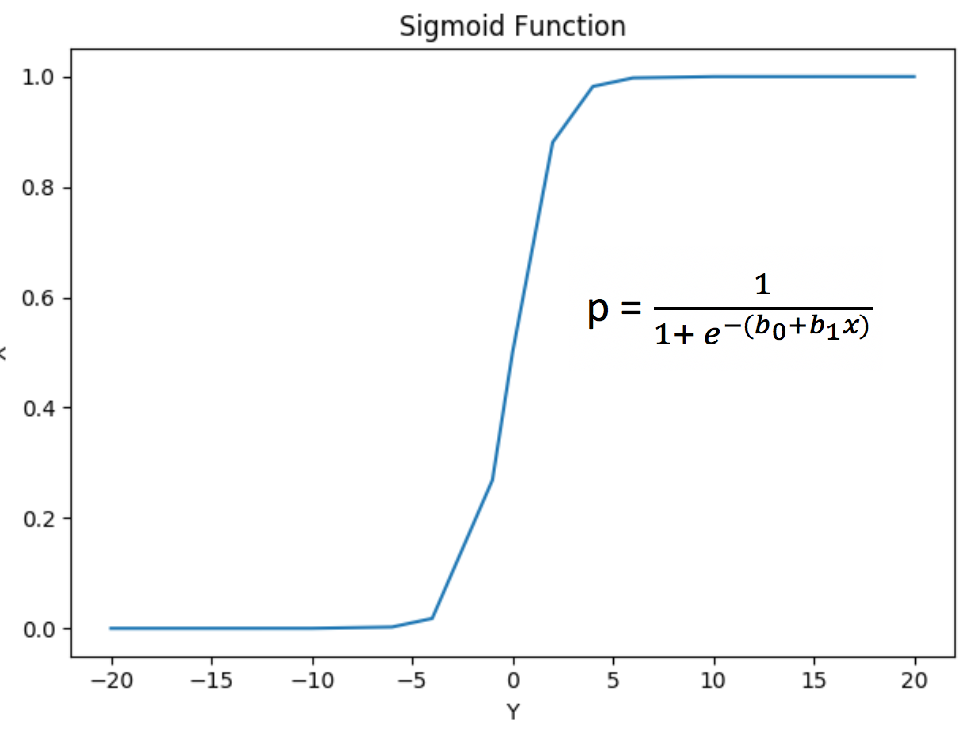
### Epochs

You will notice some differences when examining the output when building a regression model with stochastic gradient descent. SGD regressors output loss values for each epoch. **An epoch is an iteration of a machine learning algorithm which works towards minimizing a cost / loss function.**

|  |
| --- |
| -- Epoch 6  Norm: 0.41, NNZs: 11, Bias: 5.650050, T: 7674, Avg. loss: 0.215492  Total training time: 0.00 seconds.  -- Epoch 7  Norm: 0.42, NNZs: 11, Bias: 5.646630, T: 8953, Avg. loss: 0.215903  Total training time: 0.00 seconds.  -- Epoch 8  Norm: 0.42, NNZs: 11, Bias: 5.652358, T: 10232, Avg. loss: 0.215620  Total training time: 0.00 seconds.  Convergence after 8 epochs took 0.00 seconds  Root Mean Squared Error: 0.6223729015084277 |

## Logistic Regression Review

In COMP3948 we used logistic regression for classification. Remember, logistic regression uses a sigmoid function to clamp ‘y’ between 0 and 1 for all values of X.



Example : Logistic Regression

This is actually the first logistic regression example that we covered in COMP3948. GMAT and GPA scores and work experience were used to determine if a student applicant would be admitted. A chi-square test found GMT and work experience to be statistically significant.

# GMT # GPA #Work

﻿Predictor Chi-Square Scores: [123.062 3.307 21.457]

To avoid overfitting, we dropped the GPA column. The revised model uses GMT scores and work experience to make the prediction.

Intercept:

[0.176]

[[1.546 1.711]]

The logistic regression model succeeded with 100% accuracy. Note that the results without scaling only offered 80%accuracy*.*

Accuracy: 1.0

Confusion Matrix

Predicted 0 1

Actual

0 5 0

1 0 5

Here is the code:

|  |
| --- |
| import pandas as pd  import numpy as np  from sklearn.model\_selection import train\_test\_split  from sklearn.linear\_model import LogisticRegression  from sklearn import metrics  # Setup data.  candidates = {'gmat': [780,750,690,710,680,730,690,720,  740,690,610,690,710,680,770,610,580,650,540,590,620,  600,550,550,570,670,660,580,650,660,640,620,660,660,  680,650,670,580,590,690],  'gpa': [4,3.9,3.3,3.7,3.9,3.7,2.3,3.3,  3.3,1.7,2.7,3.7,3.7,3.3,3.3,3,2.7,3.7,2.7,2.3,  3.3,2,2.3,2.7,3,3.3,3.7,2.3,3.7,3.3,3,2.7,4,  3.3,3.3,2.3,2.7,3.3,1.7,3.7],  'work\_experience': [3,4,3,5,4,6,1,4,5,  1,3,5,6,4,3,1,4,6,2,3,2,1,4,1,2,6,4,2,6,5,1,2,4,6,  5,1,2,1,4,5],  'admitted': [1,1,1,1,1,1,0,1,1,0,0,1,  1,1,1,0,0,1,0,0,0,0,0,0,0,1,1,0,1,1,0,0,1,1,1,0,0,  0,0,1]}  df = pd.DataFrame(candidates,columns= ['gmat', 'gpa',  'work\_experience','admitted'])  print(df)  # Separate into x and y values.  X = df[['gmat', 'gpa','work\_experience']]  y = df['admitted']  # Import the necessary libraries first  from sklearn.feature\_selection import SelectKBest  from sklearn.feature\_selection import chi2  # Show chi-square scores for each feature.  # There is 1-degree freedom since 1 predictor during feature evaluation.  # Generally, >=3.8 is good)  test = SelectKBest(score\_func=chi2, k=3)  chiScores = test.fit(X, y) # Summarize scores  np.set\_printoptions(precision=3)  print("\nPredictor Chi-Square Scores: " + str(chiScores.scores\_))  # Re-assign X with significant columns only after chi-square test.  X = df[['gmat', 'work\_experience']]  # Split data.  X\_train,X\_test,y\_train,y\_test = train\_test\_split(  X, y, test\_size=0.25,random\_state=0)  # Perform logistic regression.  logisticModel = LogisticRegression(fit\_intercept=True, random\_state = 0,  solver='liblinear')  # Stochastic gradient descent models are sensitive to differences  # in scale so a StandardScaler is usually used.  from sklearn.preprocessing import StandardScaler  scaler = StandardScaler()  scaler.fit(X\_train)  X\_trainScaled = scaler.transform(X\_train)  X\_testScaled = scaler.transform(X\_test)  logisticModel.fit(X\_trainScaled,y\_train)  y\_pred=logisticModel.predict(X\_testScaled)  # Show model coefficients and intercept.  print("\nModel Coefficients: ")  print("\nIntercept: ")  print(logisticModel.intercept\_)  print(logisticModel.coef\_)  # Show confusion matrix and accuracy scores.  confusion\_matrix = pd.crosstab(y\_test, y\_pred,  rownames=['Actual'],  colnames=['Predicted'])  print('\nAccuracy: ',metrics.accuracy\_score(y\_test, y\_pred))  print("\nConfusion Matrix")  print(confusion\_matrix) |

## SGDClassifier

The SGDClassifier class is an altered version of the SGDRegressor class. However, SGDClassifier is designed to make binary and multi-class predictions.

Example : Stochastic Gradient Descent

For your reference this code shows how you can implement the SGDClassifier class to perform the same classification as logistic regression did in Example 3. The results are not dramatic since the SGDClassifier score is also perfect just like with logistic regression. To build this example, add this code to the end of Example 3.

|  |
| --- |
| print("\nStochastic Gradient Descent")  from sklearn.linear\_model import SGDClassifier  clf = SGDClassifier()  clf.fit(X\_trainScaled, y\_train)  y\_pred = clf.predict(X\_testScaled)  # Show confusion matrix and accuracy scores.  confusion\_matrix = pd.crosstab(y\_test, y\_pred,  rownames=['Actual'],  colnames=['Predicted'])  print('\nAccuracy: ',metrics.accuracy\_score(y\_test, y\_pred))  print("\nConfusion Matrix")  print(confusion\_matrix) |

### A Simplified Explanation of the Gradient Descent Algorithm

Example : Gradient Descent Explained in Simple Terms

This example is based on the video by Josh Stormer at <https://youtu.be/sDv4f4s2SB8>. For this case gradient descent is used to find a best-fit linear equation to predict height based on weight.

**Height = intercept + weight**

To keep this explanation simple, assume the slope is known to be 0.64. To further simplify the demonstration, we will only discuss how to optimize the intercept. The equation then becomes:

**Height = intercept + weight**

To begin, the algorithm starts with a random value for the intercept of 0. Figure 1 shows a scatter plot with three samples. For this case we are trying to find the intercept of a best fit line between them.

Figure : Scatter of Samples

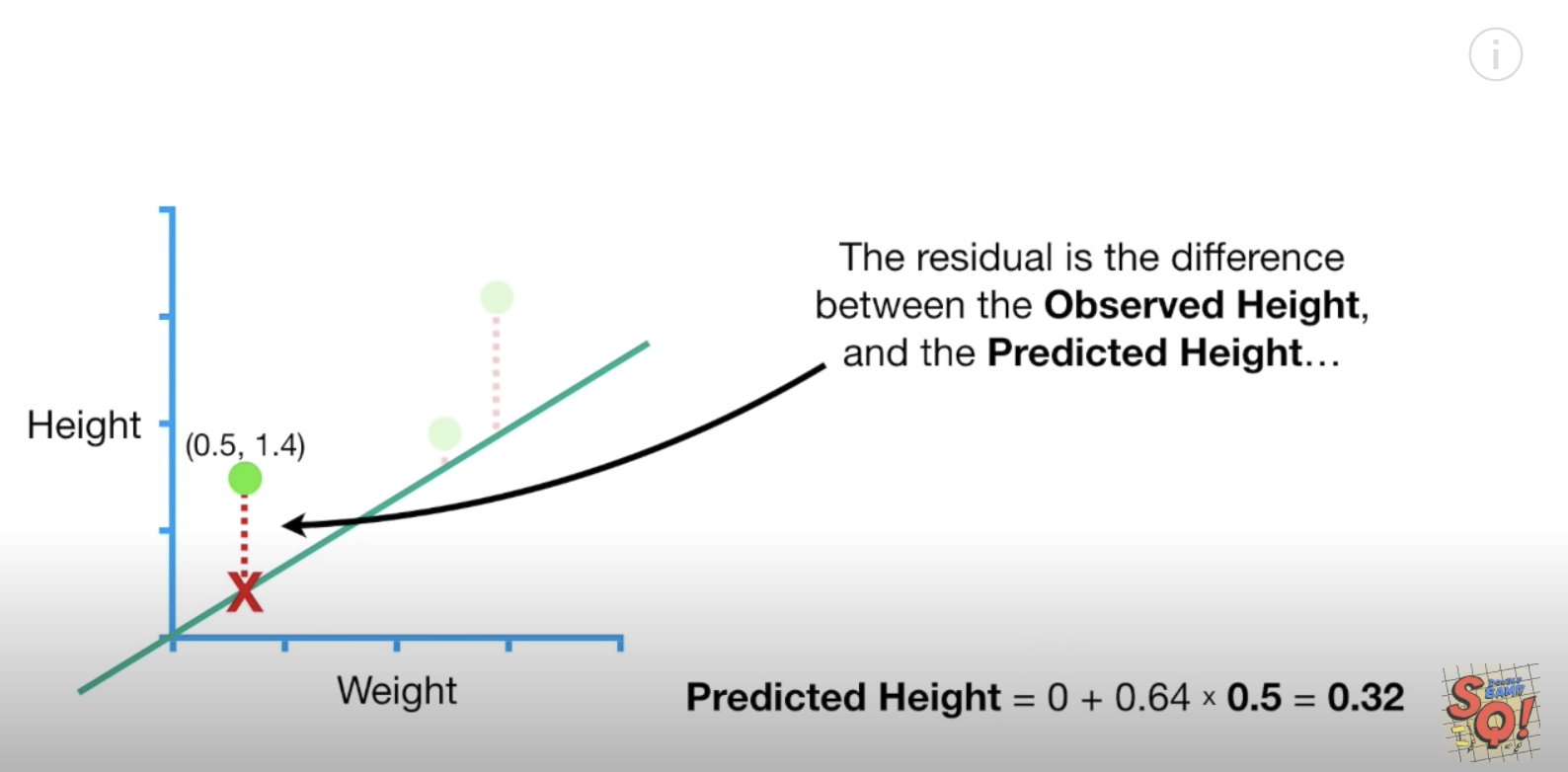
|  |  |  |
| --- | --- | --- |
|  | Weight | Height |
| 0.5  2.3  2.9 | 1.4  1.9  3.2 |

When a line with a slope of 0.64 and an intercept of 0 is drawn (refer to Figure 2), the equation for predicting weight becomes:

**Height = intercept + weight = 0 + weight**

**Height = 0 + 0.5 = 0.32**

Figure : Regression Line

****

### The Loss (Cost) Function

Loss is used to measure the effectiveness of the model. Loss really is just the sum of the squared residuals. When actual height is 1.4 and our estimated height is 0.32 when the residual value is:

**1.4 – 0.32 1.08**

When summing the squared residuals for all three data points in our example the result is:

= 3.155

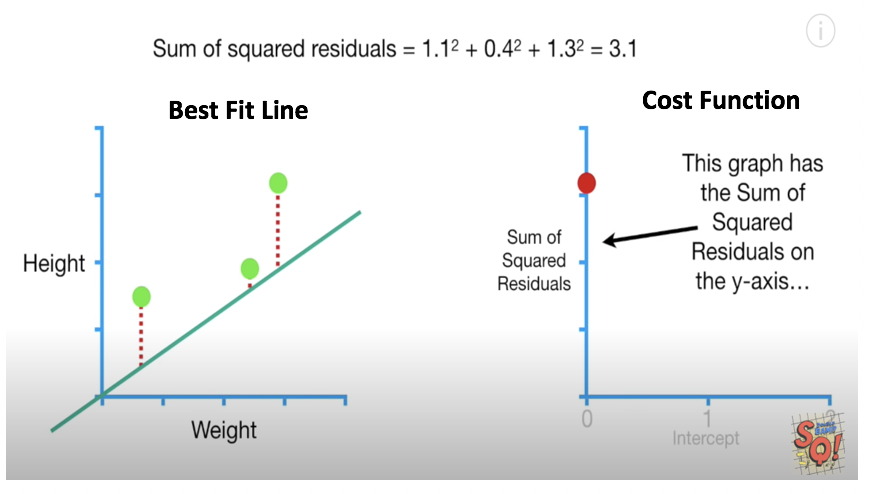
Exercise (2 marks)

Show the calculation by hand that is used to generate the residual value of -0.428 in the sum of squared residual values calculation above. To do this you will have to plug the actual weight value of 1.9 into the model equation.

|  |
| --- |
| Actual height = 2.3  Actual weight = 1.9  Predicted height = 0 + 0.64\*height (2.3) = 1.216  Residual = 1.216 – 1.9 = -0.428 |

The red dot in Figure 3 is the sum of the squared residuals when intercept is 0. The sum of the squared residuals is the **cost function**.

Figure : Regression Line Versus Cost Function

****

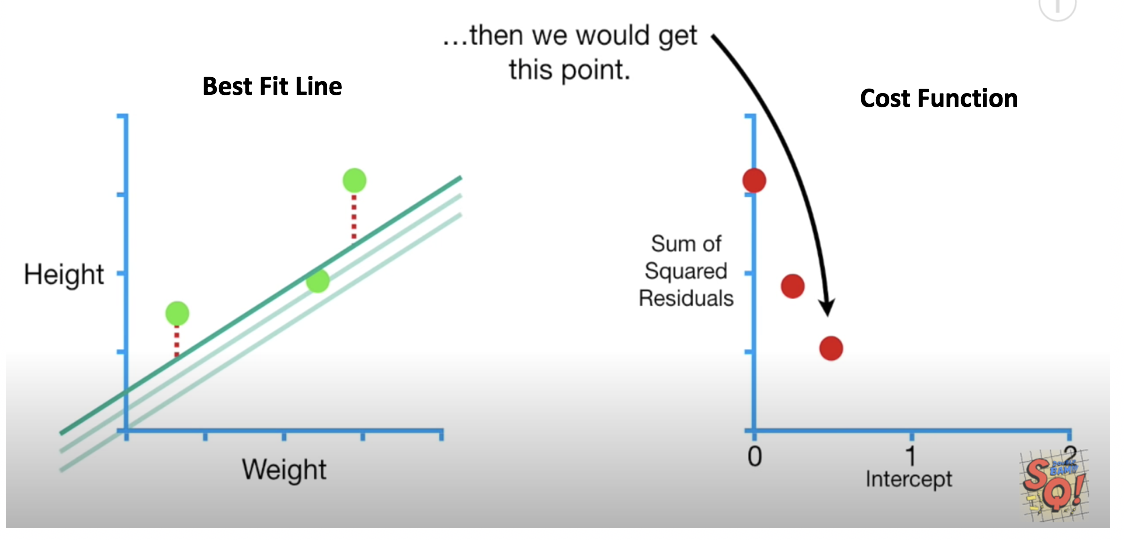
Exercise (1 mark)

I know this question may seem trivial and even obvious but it is important that you understand. What does the value of the red dot in the graph in Figure 3 represent in terms of the horizontal and vertical axis?

|  |
| --- |
| The red dot shows the sum of the squared residuals. It tells you how off the predictions are. |

We can change the intercept to obtain a lower sum squared residuals. Lower squared residuals imply a more accurate regression. The red dots in the right of Figure 4 show how the error is reduced as the intercept value is increased.

Figure : Best Fit Versus Cost

****

Exercise (1 mark)

Which dot in highlights the most optimized point of the cost function.

1. The green dot at the very left.
2. The red dot on the left.
3. The middle red dot.
4. The red dot on the right.

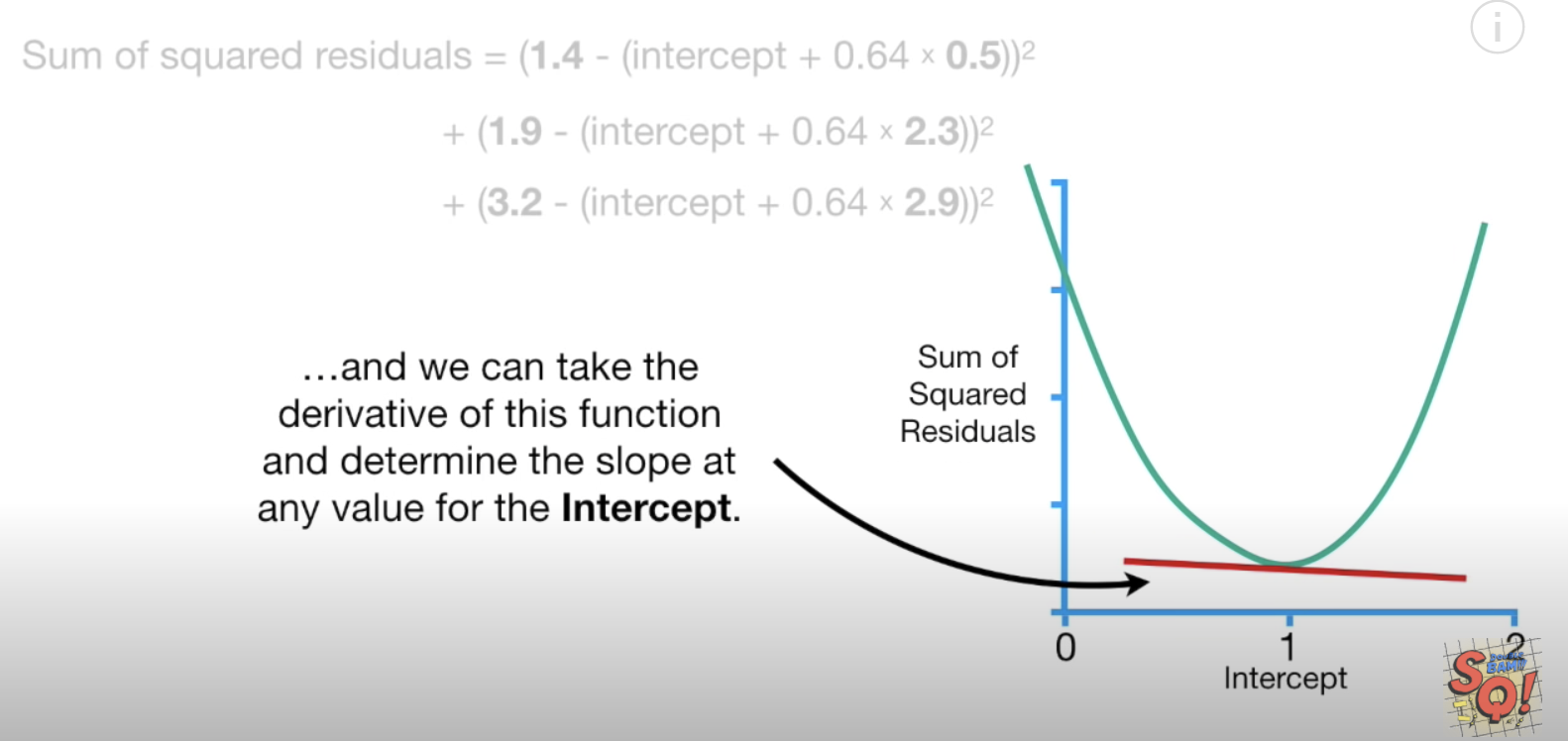
Exercise (1 mark)

In figure 4 for , each red dot represents the sum of the squared residuals for all samples when using a specific intercept.

T / F

The slope of the cost function approaches 0 as the regression becomes more accurate. See Figure 5.

Figure : Seeking the Minimal Slope of the Cost Function



## Implementing Gradient Descent

We place Chad the robot at a random position in our bowl. However, Chad has only one sensor. The sensor detects the loss value at the exact position where he stands. Using this sensor, how could Chad get to the bottom? If you guessed gradient descent you guessed right. (see Figure 6).

Figure : Chad the Robot is. Finding His Way to the Bottom of the Bowl

|  |  |
| --- | --- |
|  |  |

Gradient descent finds the slope of the cost function that is closest to zero. We can calculate the slope using several methods. You could calculate the change in y over the change in x. You can also take the derivative of the cost function to do the same thing. When the intercept is 0 the slope is -5.7.

Figure : Finding the Slope of the Cost (Loss) Function

Where 0.64 is the slope of the best fit line:

|  |  |  |
| --- | --- | --- |
|  | Weight  0.5  2.3  2.9 | Height  1.4  1.9  3.2 |

Exercise (1 mark)

By examining the graph, suggest an intercept value for the example above when the cost / loss function is minimized?

|  |
| --- |
| About 1 |

## Calculating the Derivative

Figure 7 summarizes the calculates the sum of the squared residuals relative to the change in the intercept. In other words, we are calculating the derivative with respect to the intercept.

We will not cover calculus in this course but for curious minds here is how the equation for the derivative of the loss function is derived.

**intercept + = 0 +**

= i +

Where i=intercept, h= height and =estimated height:

Residual =

=

=

=

= 0 – 2h – 0 +2i +2 +

= -2(h – i - )

Since the intercept is a constant, the derivative of the intercept is 0. In the end we can determine that the change in the sum of squared residuals relative to the change in the intercept is -5.7 as shown in Figure 7. Therefore, the slope of the cost function when the intercept is 0 is -5.7. Here is some code to perform the same calculation:

|  |
| --- |
| weights = [0.5, 2.3, 2.9]  heights = [1.4, 1.9, 3.2]  def getSlopeOfLossFunction(weights, heights, intercept):  sum = 0  BETA = 0.64  for i in range(0, len(weights)):  sum += -2 \* (heights[i] - intercept - BETA \* weights[i])  print("Intercept: " + str(intercept) + " Res: " + str(round(sum, 2)))  intercept = 0  getSlopeOfLossFunction(weights, heights, intercept) |

Exercise (1 mark)

What is the approximate derivative of the sum of the squared residuals when the intercept is 0.95?

1. 0
2. -1
3. 0.213
4. 0.296

Exercise (1 mark)

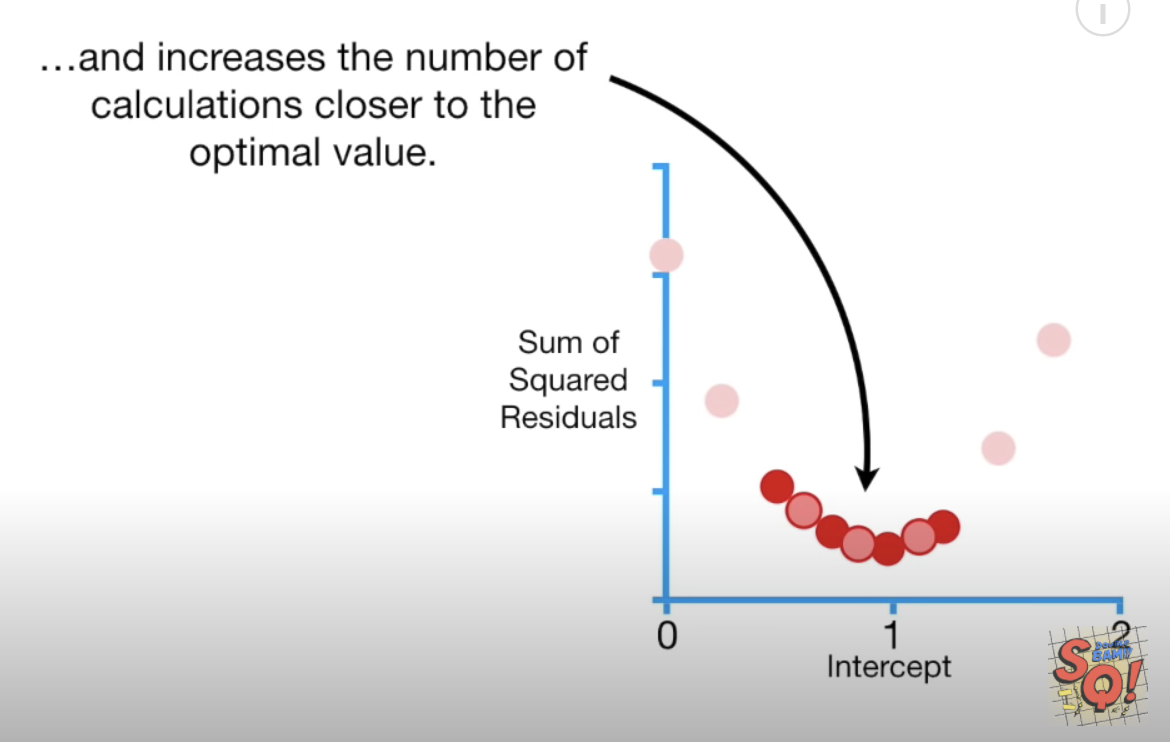
Show a picture of the slope of the loss function when the intercept is 0.95 for this example. You can use MS Paint, a drawing tool or you can draw it on paper and embed your image in this document. Show your graph below.

|  |
| --- |
|  |

### The Learning Rate

When plotting the gradient descent cost function values in Figure 8, notice how the width between intercept estimates become closer as the slope of the cost function approaches zero.

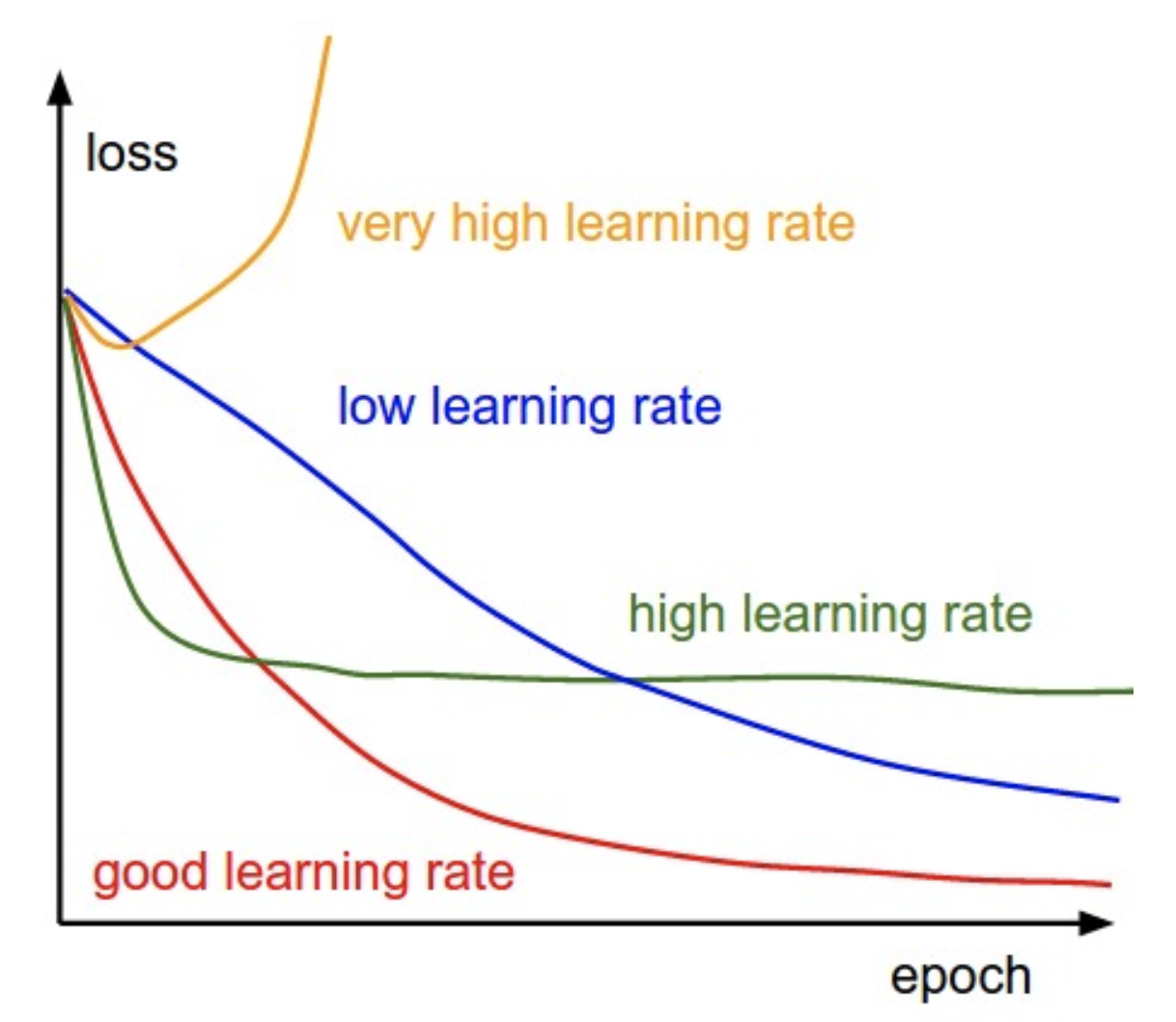
Figure : Width Between Cost Function Points



A higher learning rate can be used to speed up the descent to the minimum. However, a learning rate that is too high can cause the algorithm to overshoot the minimum and miss it. A learning rate that is too low can cause the algorithm to take too long to find the minimum. Often, manual adjustments to the learning rate are needed to find an optimal balance between speed and accuracy.

Figure 9 shows various learning rates which are applied over several iterations which are called **epochs**. The yellow and green learning rates are too fast so they missed the minimum.

Figure : Various Learning Rates

****

Exercise (1 mark)

Which line in Figure 9 shows a learning rate that is too slow?

|  |
| --- |
| blue |

#### Step Size

Step size in this case refers to the amount of change in the intercept value as we search for the optimum.

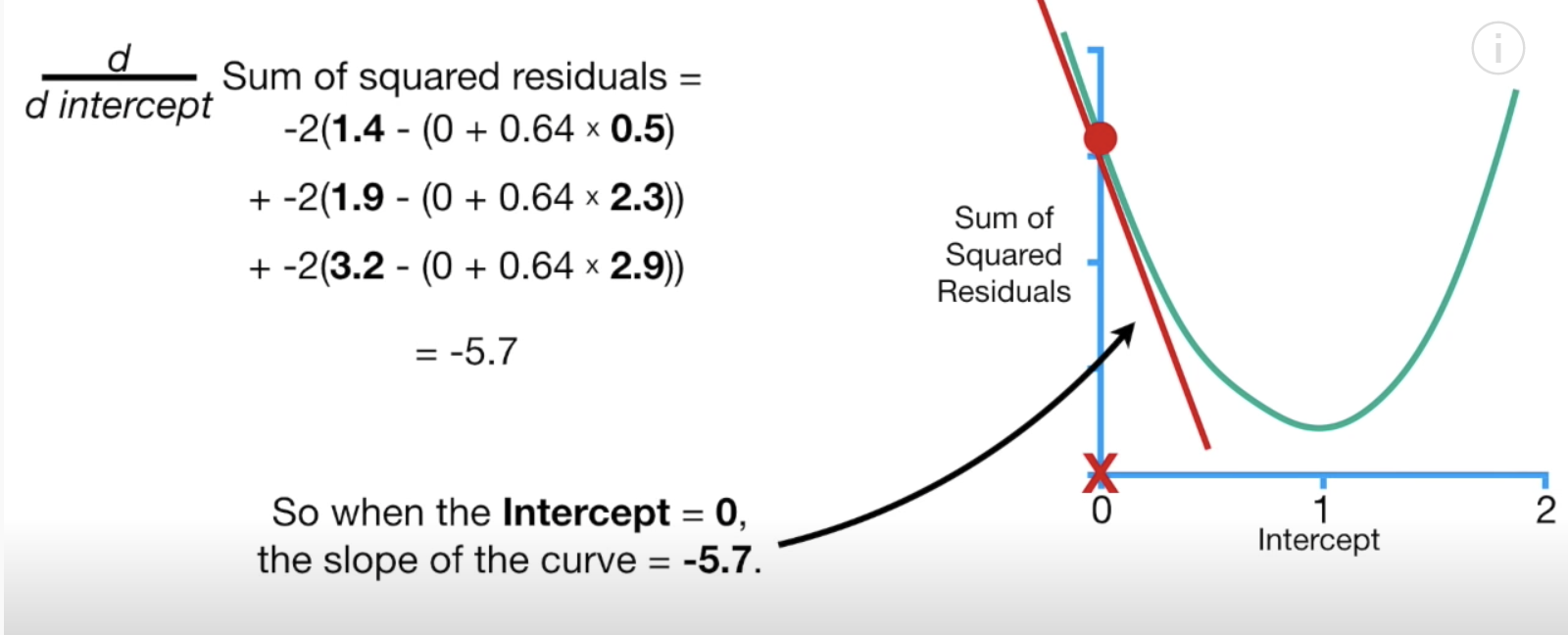
To avoid over or under-shooting the minimum we need to find an appropriate step size at a rate that is suitable when far or close to the minimum. Ideally, as the slope of the loss function approaches 0 the step size becomes smaller and smaller (refer to Figure 8).

**Step size = slope learning rate**

Remember, we showed that when the intercept is zero, the slope of the cost (loss) function is

-5.7

Figure : Slope of the Cost Function when the Intercept is 0.



Therefore, if the **learning rate** is 0.1, the step size when the intercept is zero is:

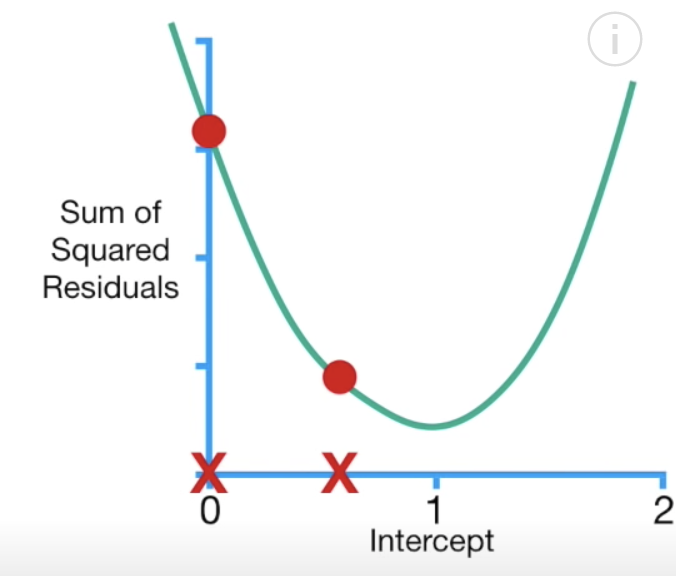
Step size = slope learning rate = -5.7 0.1 = -0.57

The new intercept becomes the old intercept - the step size:

New intercept = old intercept – step size = 0 - -0.57 = 0.57

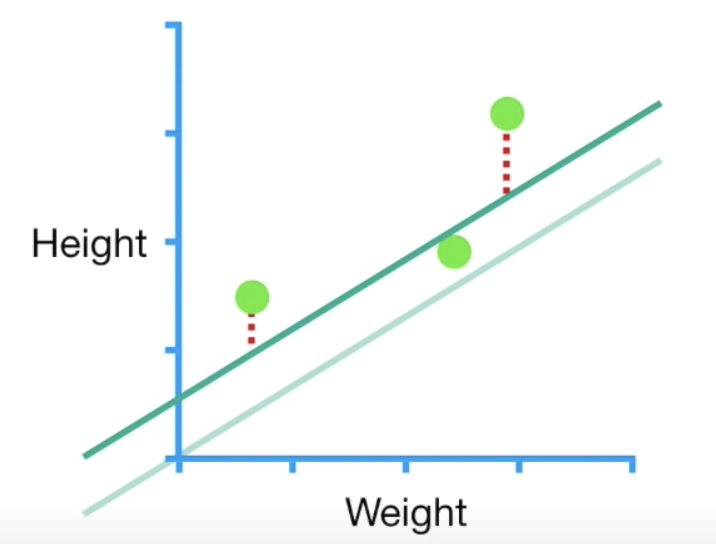
We can represent our new intercept on our graph (see Figure 11).

Figure : Plotting intercepts 0 and 0.57 on the cost function graph



When the intercept is 0.57, notice how the sum of the squared residuals is reduced (see Figure 12).

Figure : Reducing the Sum of the Squared Residuals



Exercise (3 marks)

What is the slope of the cost function when the intercept is 0.57? Show the setup for your calculation. Refer to Figure 10. Also show the final answer.

|  |
| --- |
| Derivative of the sum of the square residuals =  -2(1.4-(0.57 + 0.64\*0.5) + -2(1.9-(0.57 + 0.64\*2.3) + -2(3.2-(0.57 + 0.64\*2.9)  = -2.3 |

Exercise (3 marks)

If the learning rate is 0.1, what is the next step size when the intercept is 0.57? What intercept is generated with the new step size? Show your calculations.

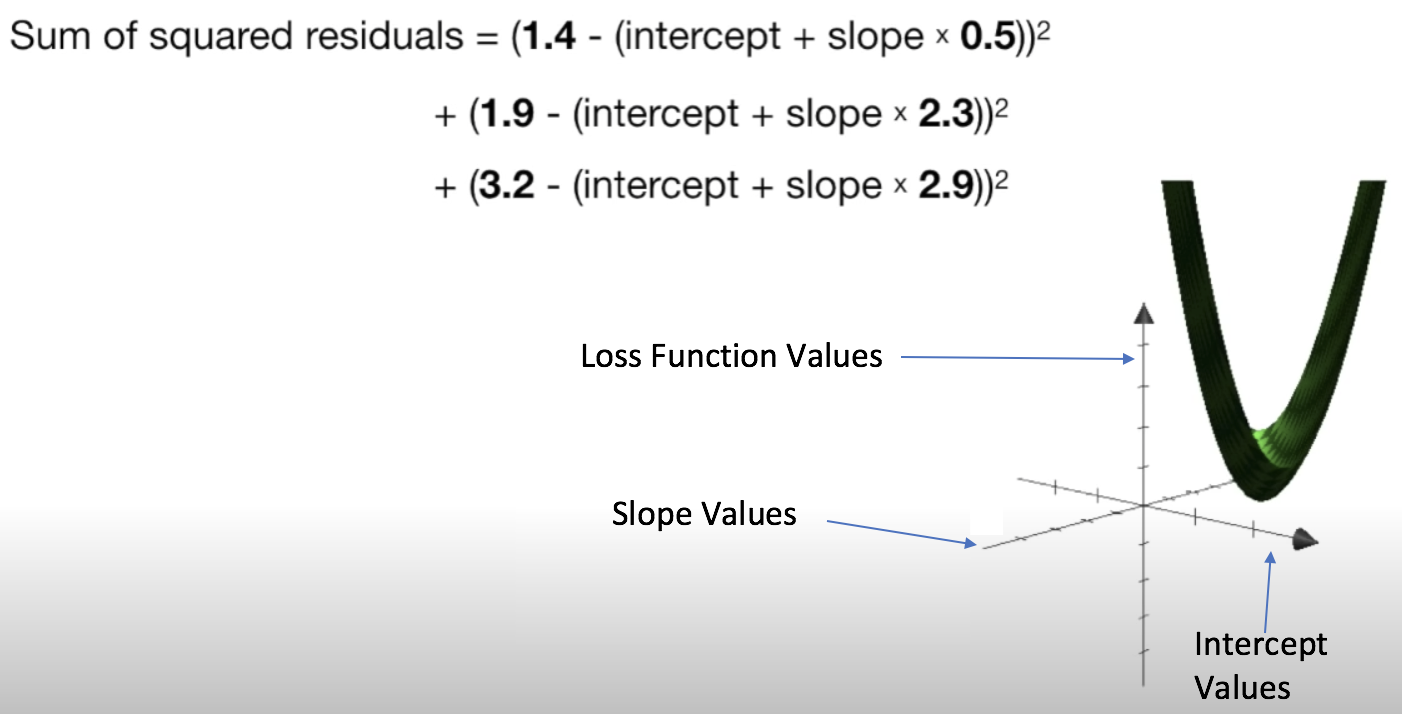
|  |
| --- |
| Step size = -2.3 \* 0.1 = -0.23  New intercept is 0.57- (-0.23) = 0.8 |

Example : Calculating the Minimum Loss When Estimating the Intercept and Slope

In Example 5 we examined how to apply gradient descent to find the best fit line by only adjusting the intercept. This example continues from Example 5. However, in this case we will consider adjustments to both the intercept and the slope of the best fit line.

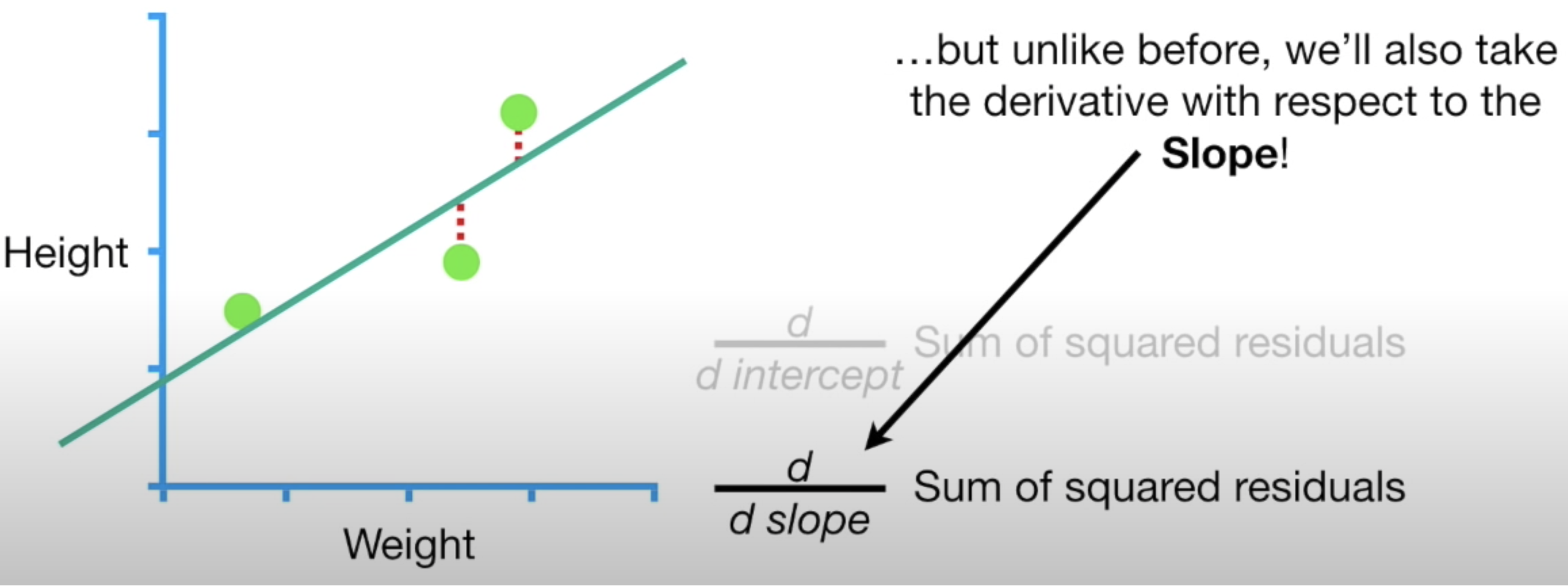
Just like before we will calculate the sum of the squared residuals to determine the slope of the loss function. This time though, we will adjust both the slope and the intercept. Since we are now calculating loss function values for different slope and intercept values it helps to visualize the cost function in 3D space. To minimize the cost function, we must now find the minimum using three different axes. In other words, we need to find the bottom of the bowl shape shown in Figure 13.

Figure : Cost Function with Changing Slope and Intercept Values



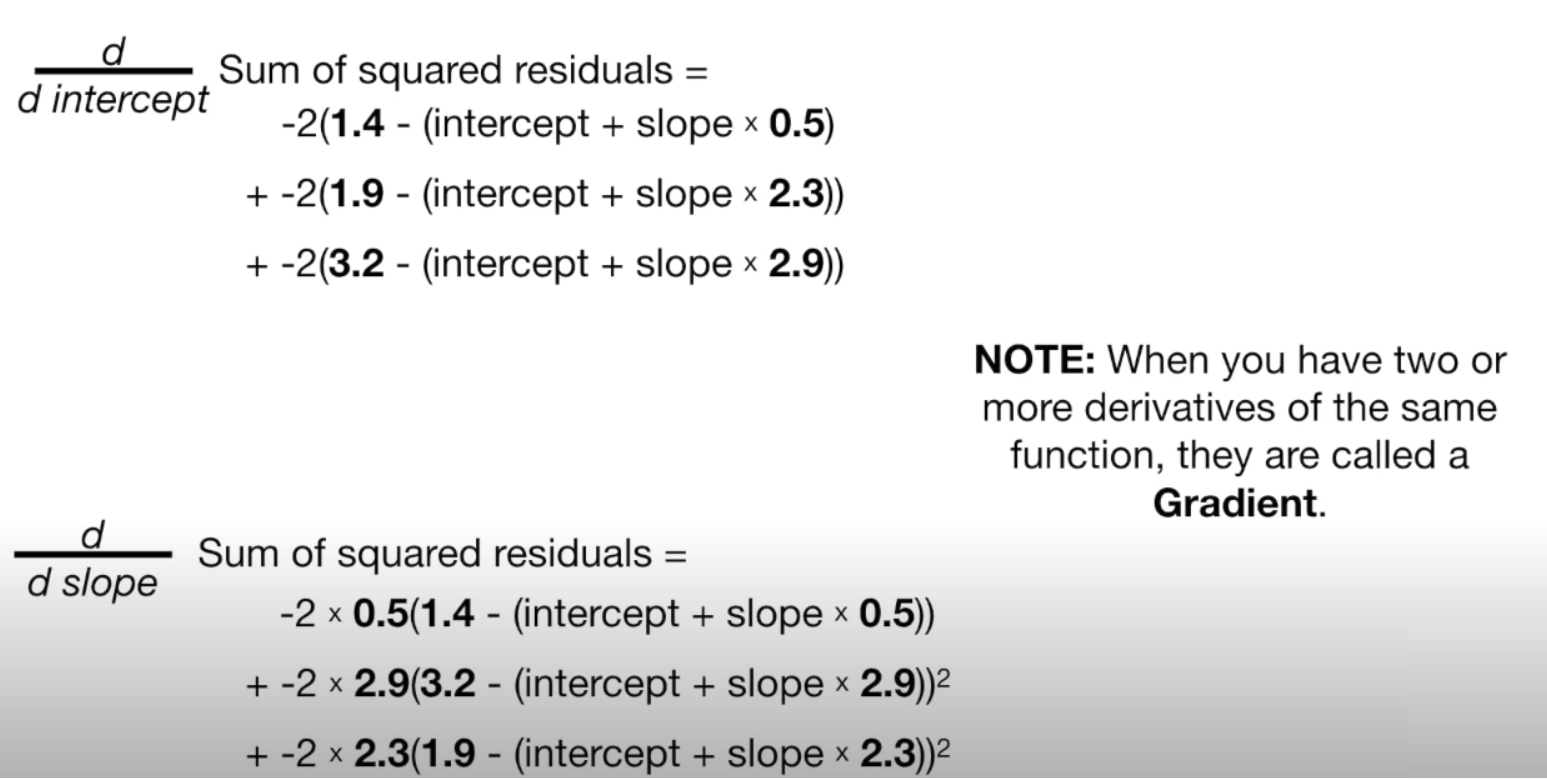
In Example 5, we were only searching for the changes in the intercept so we only took the derivative of the loss function with respect to the intercept. Now, since we are adjusting the intercept and the slope of the best fit line we must take the derivative of the loss function with respect to the intercept and the slope of the best fit line (see Figure 14).

Figure : Calculating the Slope of the Loss Function using Two Variables



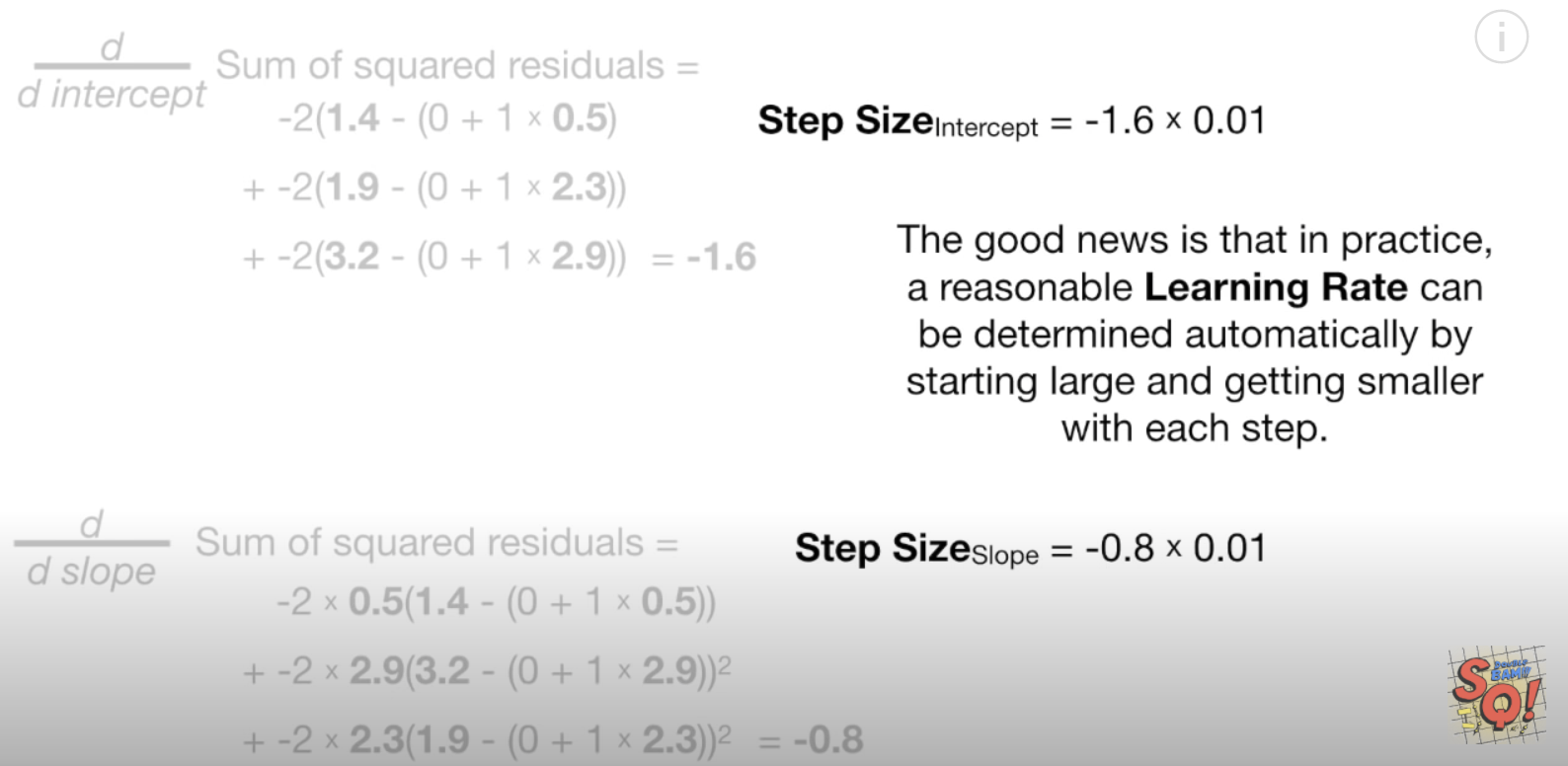
When you have two or more derivatives of the same function this is called a **gradient**. Figure 15 shows the calculations needed for determining both derivatives.

Figure : Derivatives with respect to the intercept and the slope



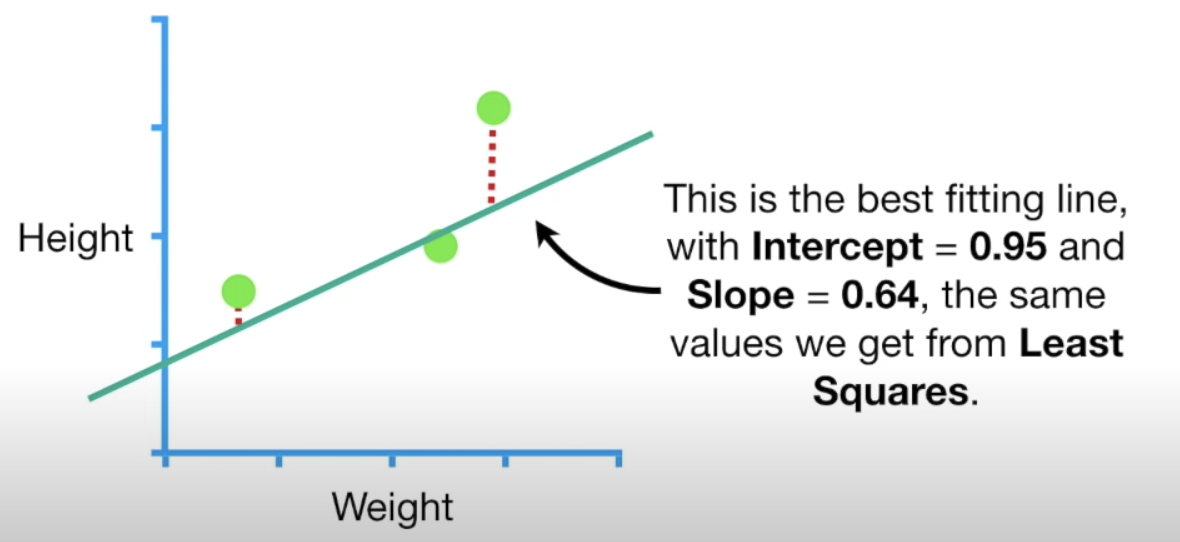
Essentially, we are using two different slopes to find minimums in two different dimensions If we were to calculate both derivatives by starting with an intercept of 0 and a slope of 1, notice how the step sizes are calculated when the learning rate is 0.1 (see Figure 16).

Figure : Calculating Step Size for each Dimension

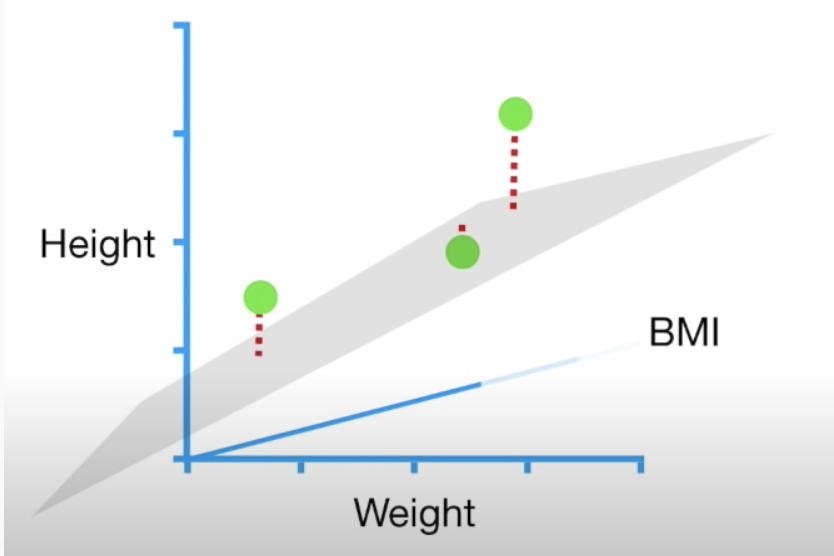


After taking several steps we arrive at the optimal slope and intercept for the best fitting line. This happens to be the same result that is generated by the least squares method (Figure 17).

Figure : The optimal intercept and slope from gradient descent



If we had more parameters than height and weight then we take only need to take more derivatives and everything else would stay the same.



### Gradient Descent Summary

The sum of the squared residuals is just one type of loss function, there are many other types of loss functions but gradient descent works the same way. Specifically, the steps of the gradient descent algorithm are:

1. Take the derivative of the loss function for each parameter in it. In other words, take the gradient of the loss function.
2. Pick random values for the parameters.
3. Plug the parameter values into the derivatives.
4. Calculate the step sizes: (Step size = Slope x learning rate)
5. New parameter = old parameter – step size
6. Go back to step 3 and repeat until finding a minimum of 0.001 or less or until generally 1000 iterations (epochs).

## Stochastic Gradient Descent

For the three data points in Example 5 and Example 6 the calculations did not take very long. However, with millions of data records, the calculations can be very time consuming. Stochastic gradient descent works like the gradient descent algorithm. However, stochastic gradient descent takes a random subset of data to perform the calculations for the cost function.

Exercise (1 mark)

In your own words, explain how stochastic gradient descent is different than gradient descent.

|  |
| --- |
| A gradient descent uses the whole data set to calculate the cost function, while the stochastic gradient descent uses a random subset of the data to calculate the cost function. |

Exercise (1 mark)

A gradient:

1. Is used for grating cheese.
2. Could be used to find the minimum of a single dimension.
3. Is the combination of two or more derivatives of the same equation in different dimensions.
4. Is the mid-point of a curvature.