Welcome to Deep Learning Online Bootcamp

Day 4

dφ

Democratizing Data Science Learning

Learning Objectives

Linear Regression

Cost

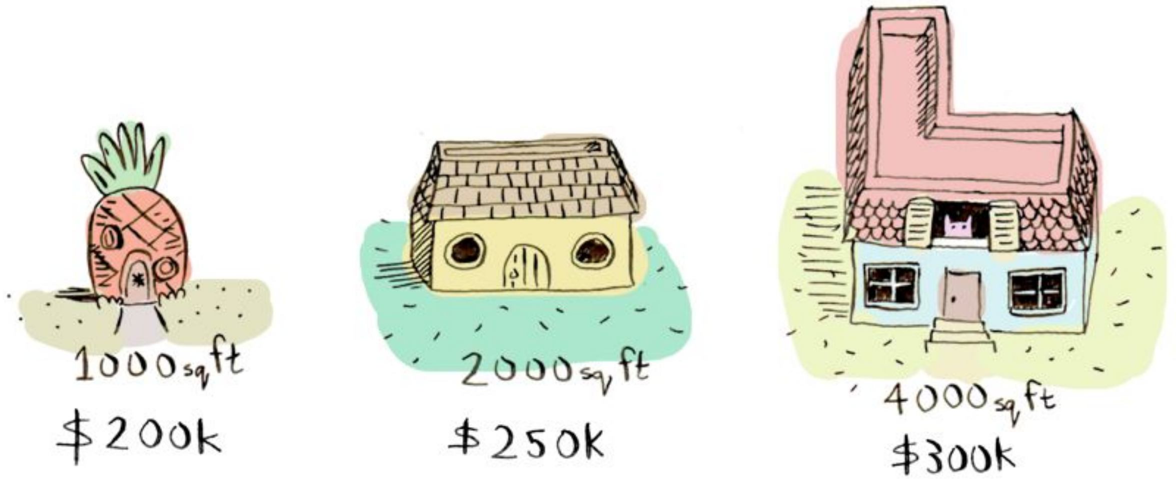
Learning Rate

Epochs

Batch Size

What is linear regression? - an example

Suppose you are thinking of selling your home. And, various houses with different sizes (area in sq.ft) around you have sold for different prices as listed below:

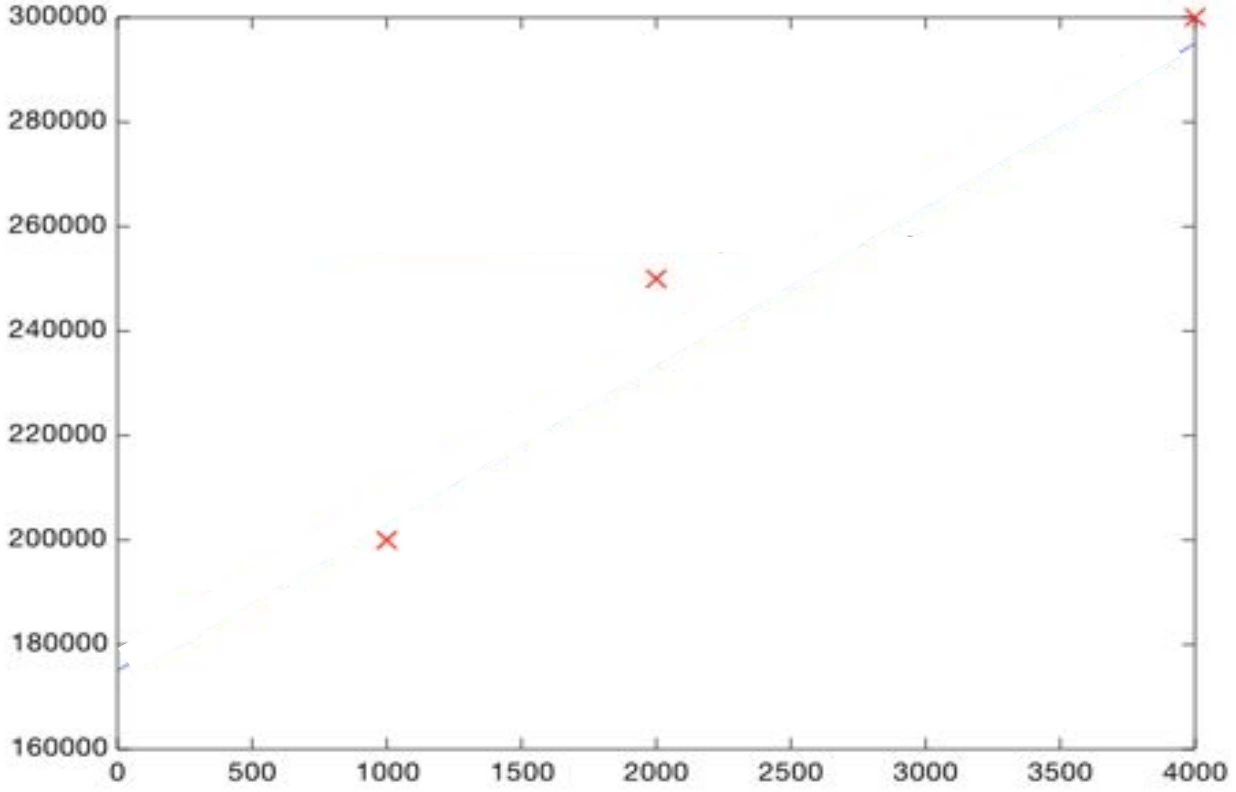


And considering, **your home is 3000 square feet**. How much should you sell it for?

When you **look at the existing price patterns (data) and predict a price for your home** that is an example of **linear regression.**

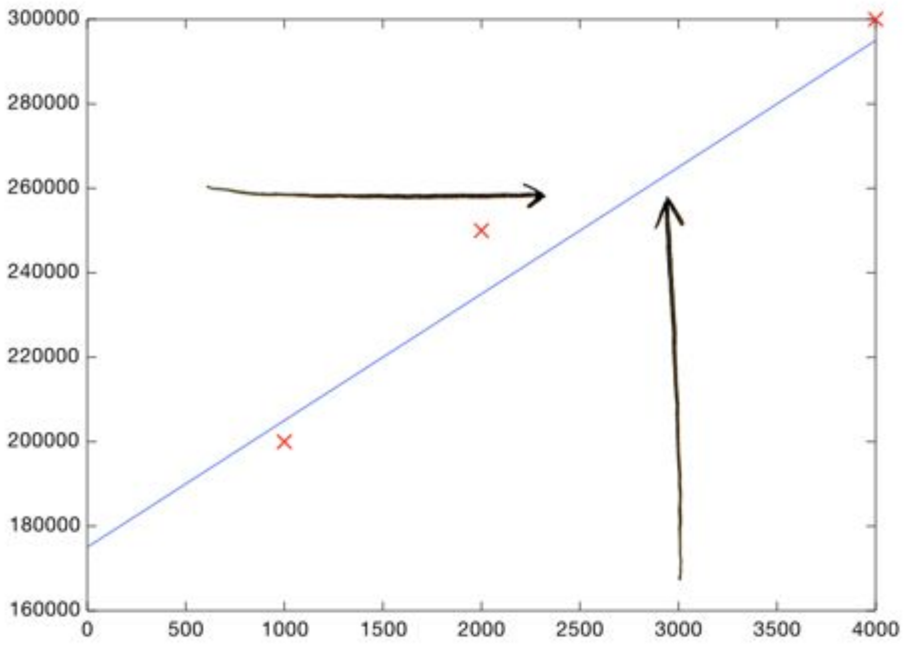
What is linear regression? - an example

Here's an easy way to do it. Plotting the 3 data points (information about previous homes) we have so far:



What is linear regression? - an example

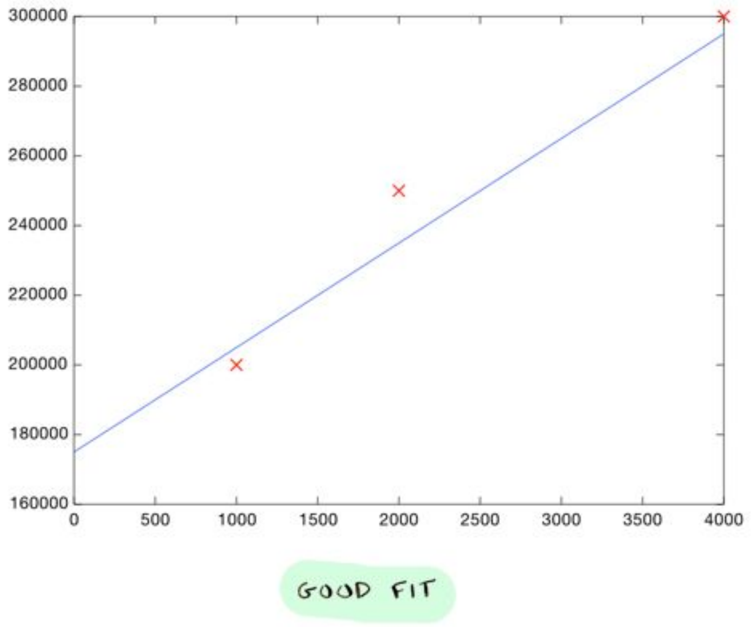
Now you can eyeball it and roughly draw a line that gets pretty close to all of these points. Then look at the price shown by the line, where the square footage is 3000:

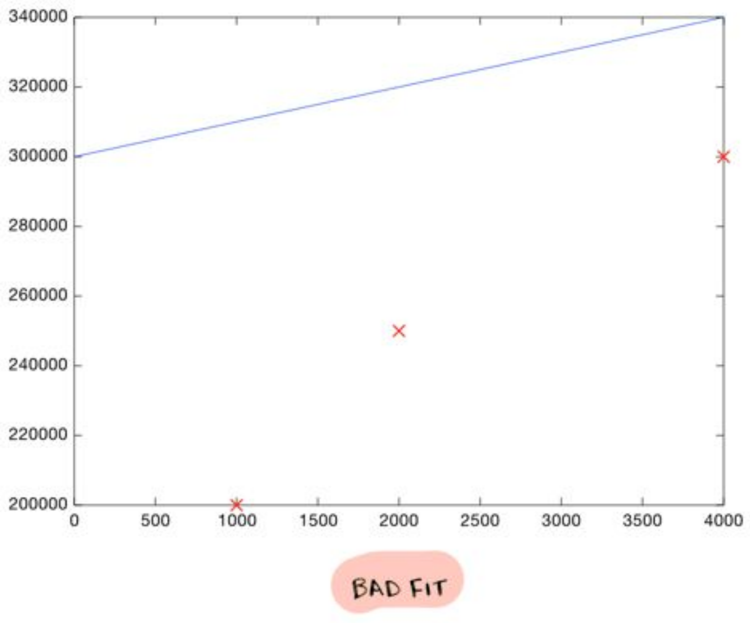


Boom! Your home should sell for $260,000.

What is linear regression? - an example

That's all! (Well kinda) You plot your data, make a rough line, and use the line to make predictions. You just need to make sure **your line fits the data well** (but not too well :)





But of course we don't want to roughly make a line, we want to compute the exact line that best "fits" our data. That's where

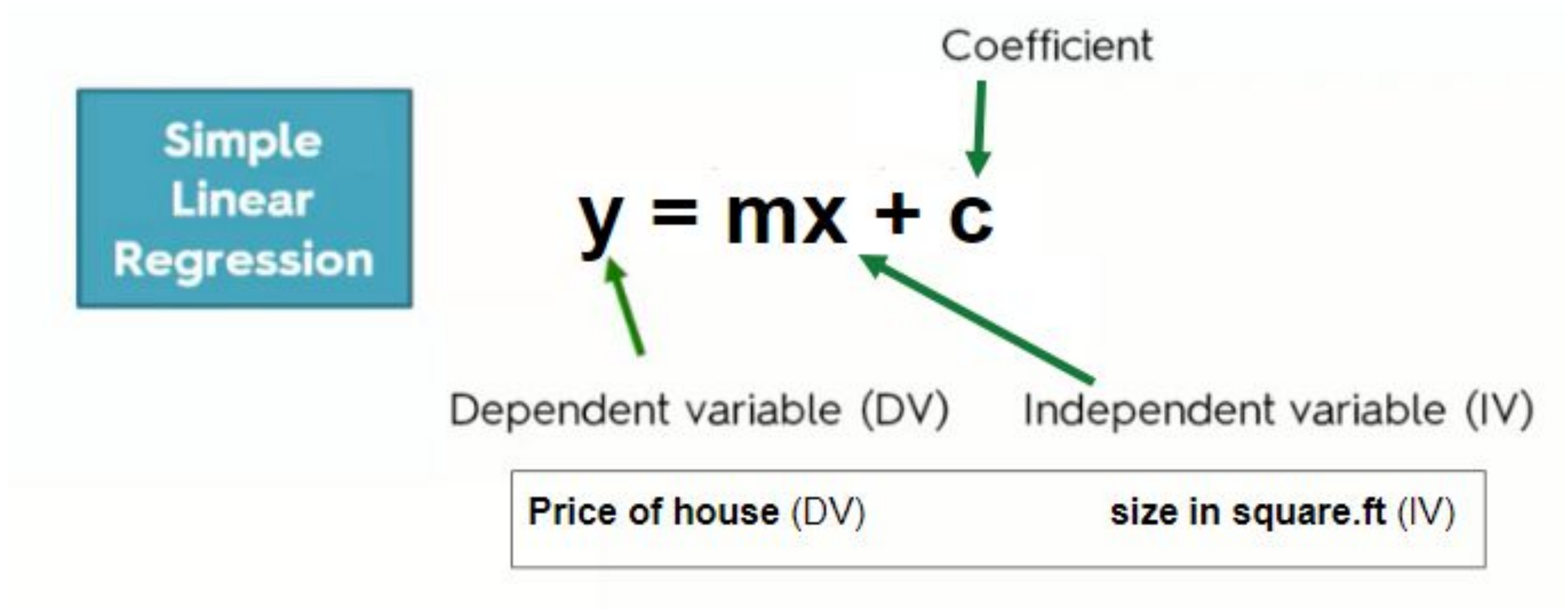
What is linear regression? - to summarize

- Linear regression is a linear model i.e. a model that assumes a **linear relationship** (straight-line relationship) between the input variables (x) and the single output variable (y).

- When there is a single input variable (x), the method is referred to as **simple linear regression** or just linear regression.
For eg. The area of houses in this case.

- When there are multiple input variables, it is often referred to as **multiple linear regression**.
For eg. If area of house, neighbourhood, rooms, etc were given.

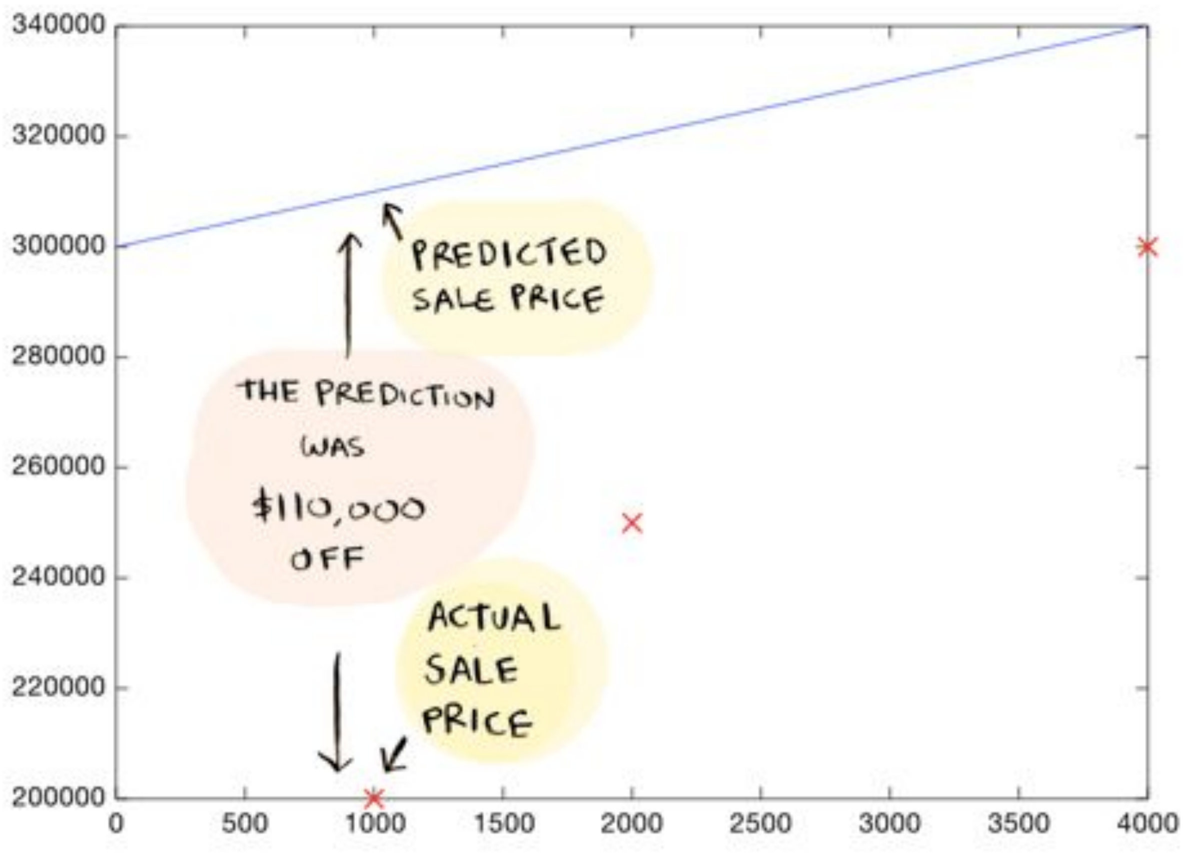
Simple Linear Regression



In case of linear regression, the relationship between the dependant and the independent variable is a simple equation of a straight line (**y = mx + c**)

Which line is good?

Consider this line and the predictions that it makes



This above drawn line is way off. For example, according to the line, a 1000 sq foot house should sell for $310,000, whereas we know it actually sold for $200,000.

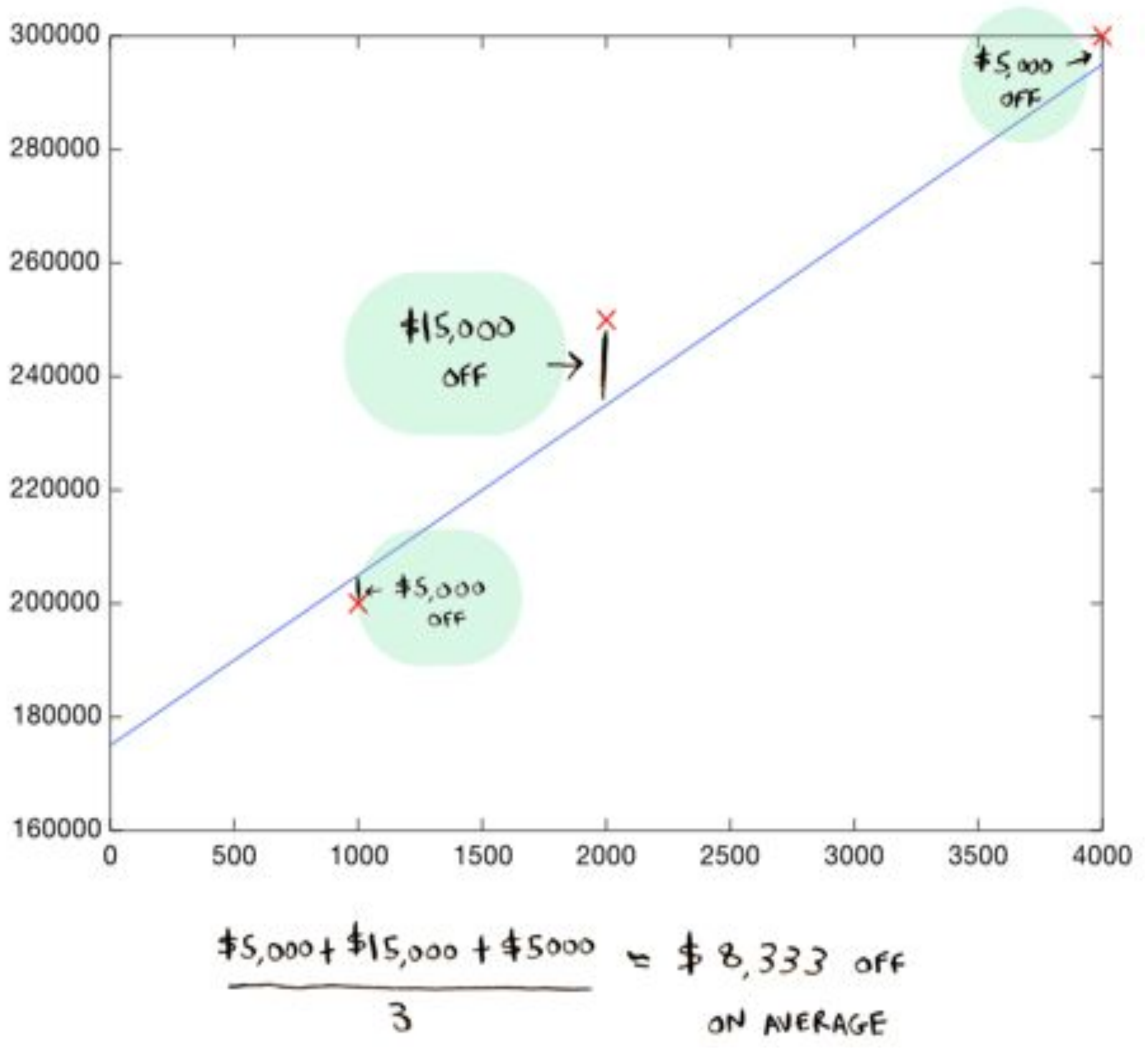
Which line is good?

Here's a better line:

The **residual** (absolute difference between the actual and the predicted value) in this case are: \$5000, \$15000 and \$5000.

This line is an average of **\$8,333** dollars off (adding all the distances and dividing by 3).

This **\$8,333** is called the **cost** of using this line.



Cost

The **cost** is how far off the **learned** line is from the **real data**. The best line is the one that is the least off from the real data.

To find out what line is the best line (to find the values of coefficients), we need to define and use a cost function.

In ML, cost functions are used to estimate how badly models are performing.

Put simply, a cost function is a measure of how wrong the model is in terms of its ability to estimate the relationship between X and y.

Cost Function

Now let’s take a step back and reflect on why we are performing Linear Regression. It is being performed so that when we are finally using our model for prediction, it will predict the value of target variable (or dependent variable) y for the input value (s) of x (even on a new unseen data).

By achieving the best-fit regression line, the model aims to predict y value such that the **error difference between predicted value and true value is minimum.**

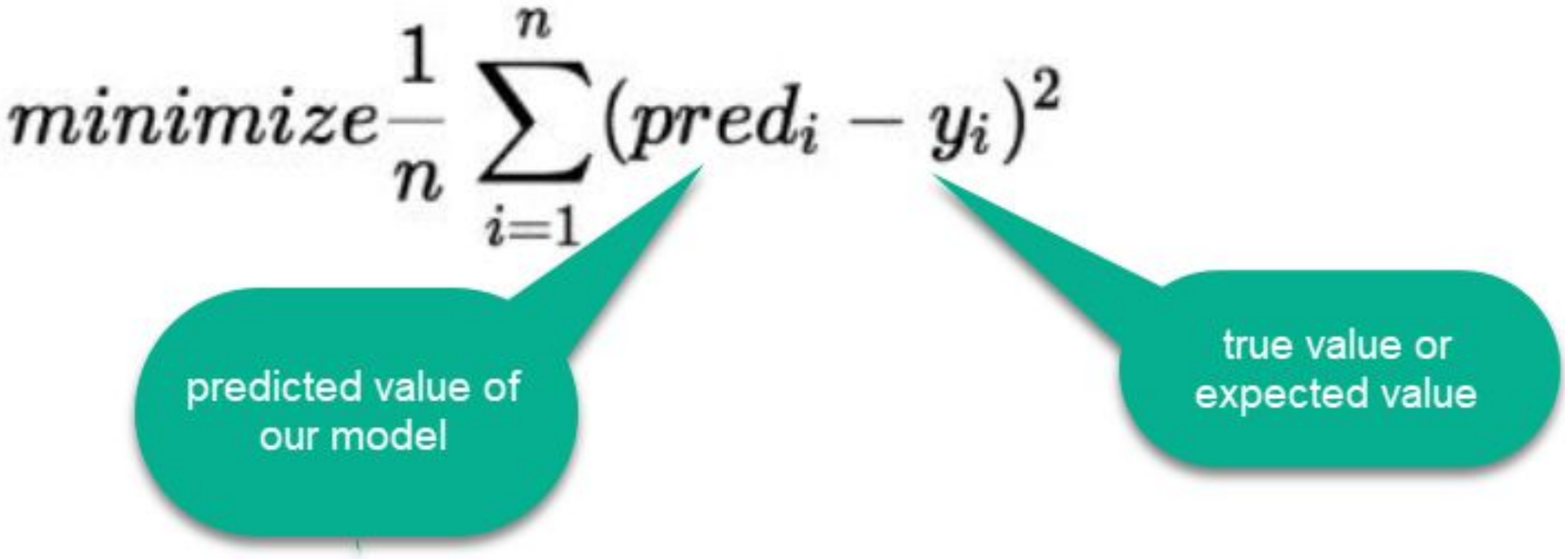
So, it is very important to update the *m* and *b* values, to **reach the best value that minimize the error between predicted value (pred)** and **true value (y)** per sample *i*.

$$minimize \frac{1}{n} \sum_{i=1}^{n} (pred_i - y_i)^2$$

Here, *n* is the total no. of observations/data points.

Now let’s have a look at how exactly we got the above formula

Cost Function



$$minimize \frac{1}{n} \sum_{i=1}^{n} (pred_i - y_i)^2$$

Here, **pred = mx + c.** Pred can also be termed as the hypothesis function

The difference between the model’s **predicted value** and the **true value** is called **RESIDUAL or COST or LOSS**, which needs to be minimized.

Cost Function - Types

There are three primary metrics used to evaluate linear models (to find how well a model is performing):

1. Mean Squared Error
2. Root Mean Squared Error
3. Mean Absolute Error

Mean Squared Error (MSE)

- MSE is simply the average of the squared difference between the true target value and the value predicted by the regression model.

- As it squares the differences, it **penalizes** (gives some penalty or weight for deviating from the objective) **even a small error** which leads to **over-estimation of how bad the model is.**

$$MSE = \frac{1}{n} \Sigma \underbrace{\left(y - \widehat{y} \right)^2}_{\text{The square of the difference between actual and predicted}}$$

Root Mean Squared Error (RMSE)

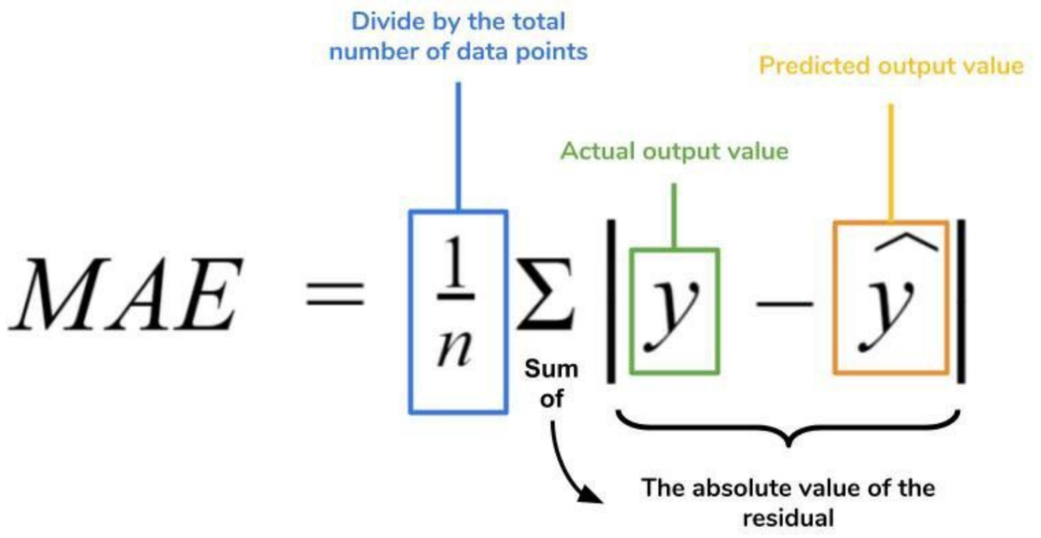
- It is just the square root of the mean square error.

- It is preferred more in some cases because the errors are first squared before averaging which poses a high penalty on large errors. This implies that **RMSE is useful when large errors are undesired.**

$$RMSE = \sqrt{\frac{\sum_{i=1}^{N} (Predicted_i - Actual_i)^2}{N}}$$

Mean Absolute Error(MAE)

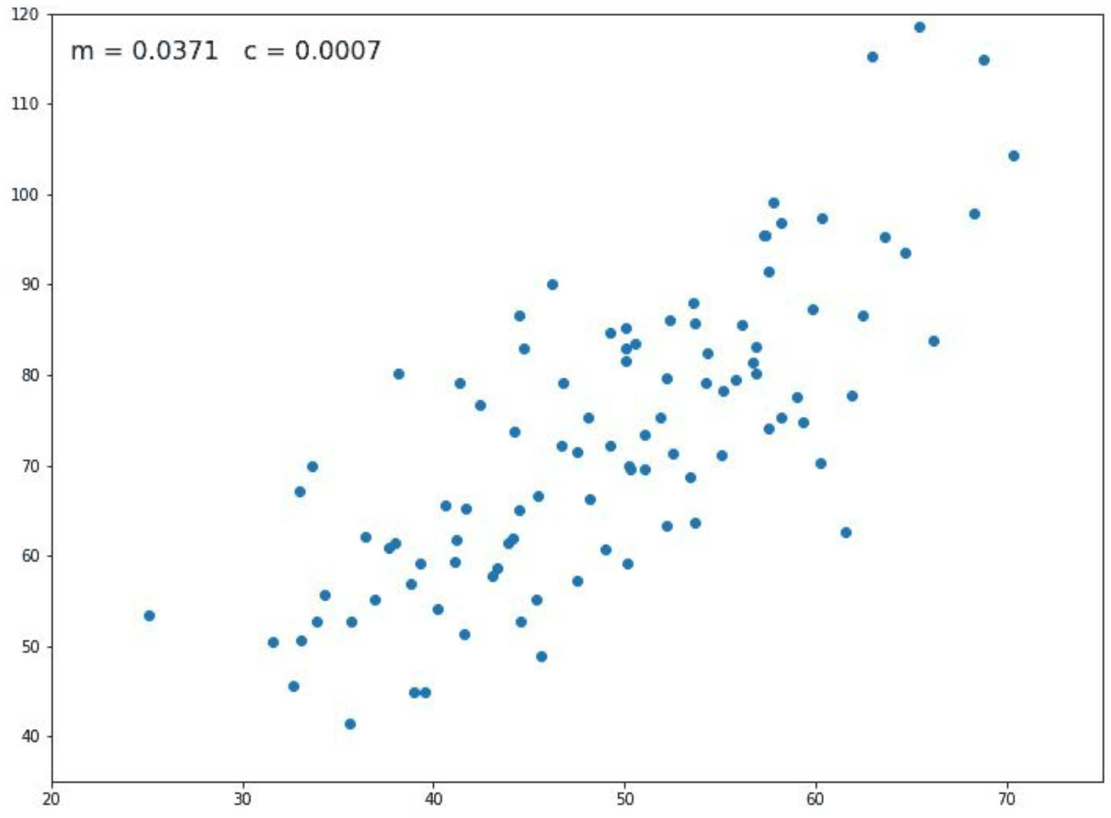
- MAE is the absolute difference between the target value and the value predicted by the model.
- MAE **does not penalize the errors as effectively as mse** making it not suitable for use-cases where you want to pay more attention to the outliers.


Divide by the total number of data points
Predicted output value
Actual output value
$MAE = \frac{1}{n}\sum\left|y-\hat{y}\right|$
Sum of
The absolute value of the residual

Objective

Find Optimal Values that provide efficient/effective predictions

The values of *m (coefficient of x)* and *c* (y intercept) are updated repeatedly until the line fits the data well enough to get the optimal solution



$y = 1.47 * x + 0.0296$

How do we find the optimal values, you ask?

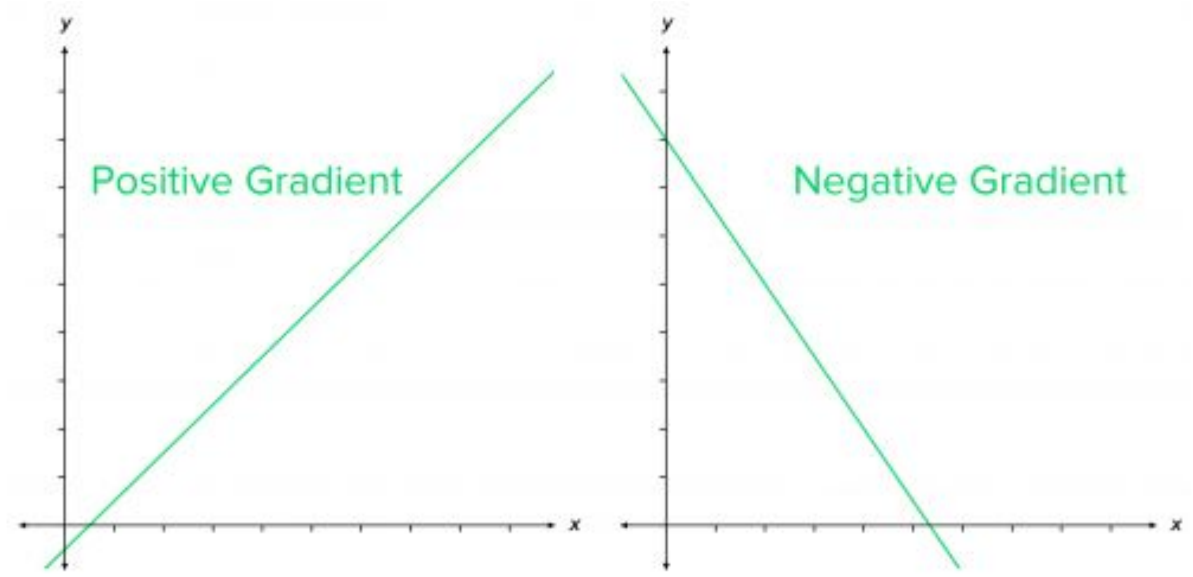
Gradient Descent!

Gradient Descent (GD) is an optimization technique in the machine learning process which helps us **minimize the cost function** by learning the weights of the model such that it fits the training data well.

Gradient Descent - let’s break it down

Gradient = rate of inclination or declination of a slope (**how steep** a slope is and in **which direction** it is going) of a function at some point.

Descent = an act of moving downwards.



Simple way to memorize:

- line going up as we move right → positive slope, gradient
- line going down as we move right → negative slope, gradient

Gradient Descent- A technique to minimize cost

At the start, the parameter values of the hypothesis are randomly initialized (can be 0)

Then, Gradient Descent **iteratively** (one step at a time) adjusts the parameters and gradually finds the best values for them by minimizing the cost

In this case we'll use it to find the parameters **m** and **c** of the linear regression model

m = **slope** of a line that we just learned about
c = bias : where the line crosses the Y axis.

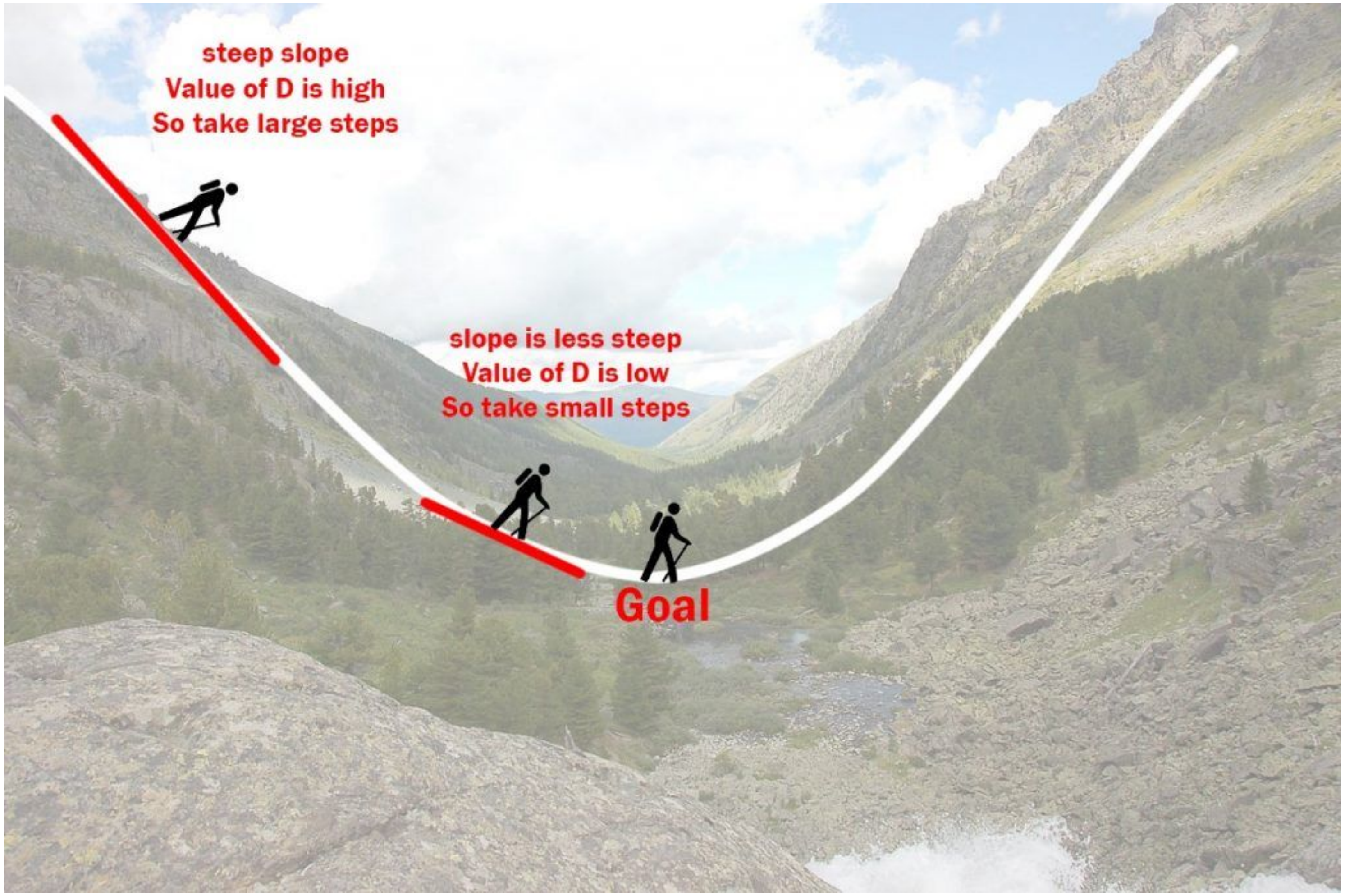
Gradient Descent - An Example

Imagine a person at the top of a mountain who wants to get to the bottom of the valley below the mountain through the fog (he cannot see clearly ahead and only can look around the point he’s standing at)

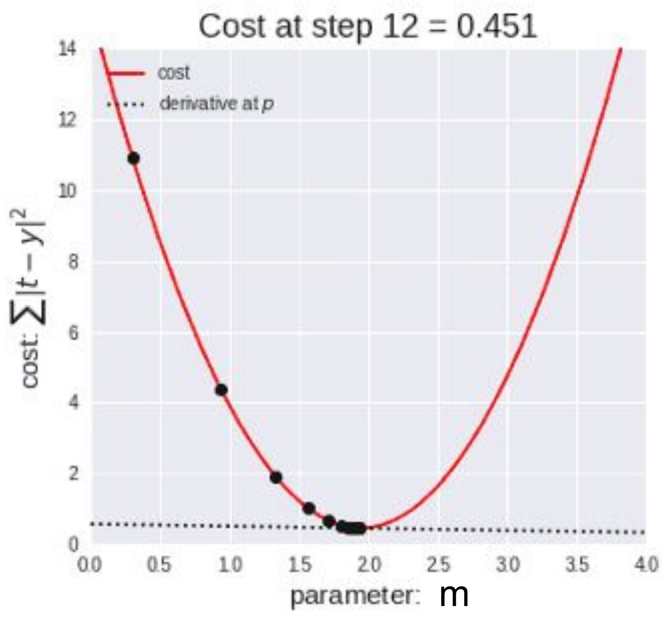
He goes down the slope and takes **large steps when the slope is steep** and **small steps when the slope is less steep**.

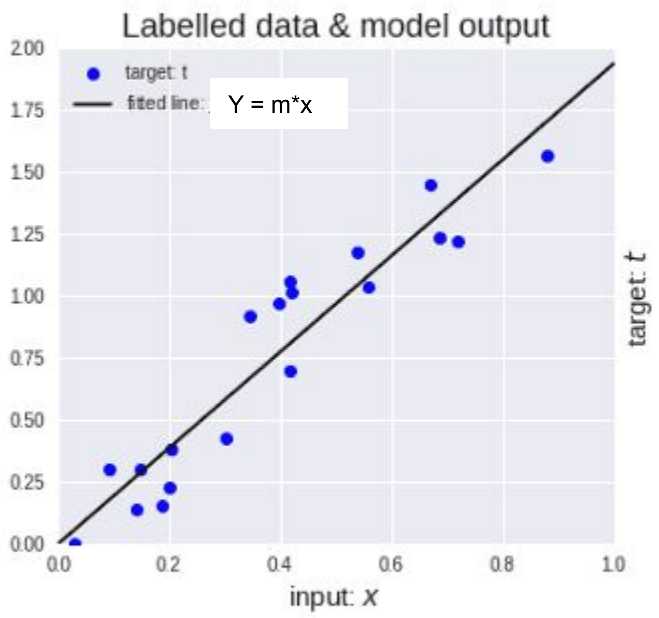
He decides his next position based on his current position and **stops when he gets to the bottom of the valley which was his goal.**

Gradient Descent



Gradient Descent in action





Now, think of the valley as a cost function. The objective of GD is to minimise the cost function (by updating it's parameter values). In this case the minimum value would be 0.

$$MSE = \frac{1}{n}\Sigma\underbrace{\left(y - \widehat{y}\right)^2}_{\text{The square of the difference between actual and predicted}}$$

dφ

Democratizing Data Science Learning

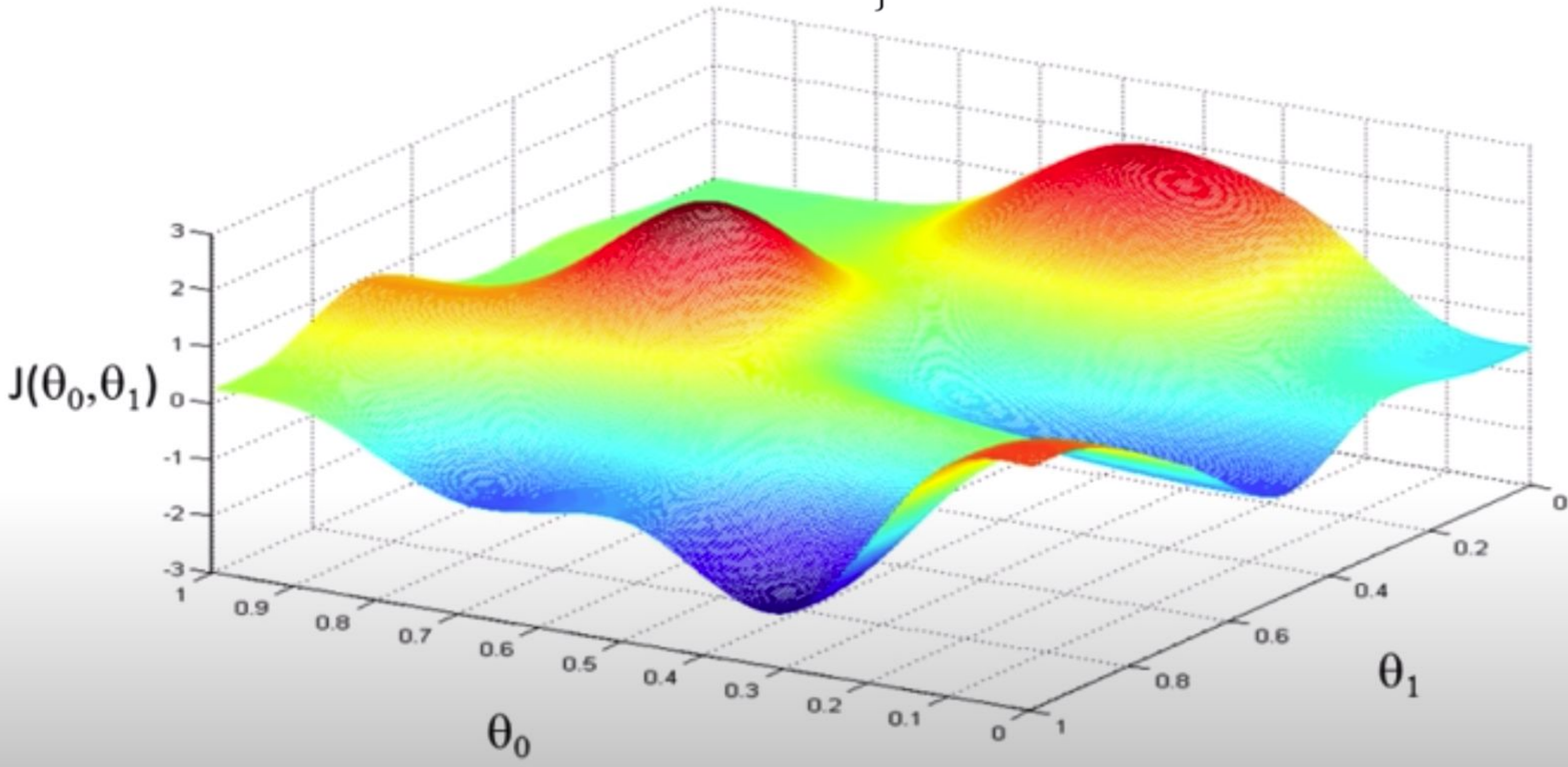
What does a cost function look like?

J is the cost function, plotted along it’s parameter values. Notice the crests and troughs.

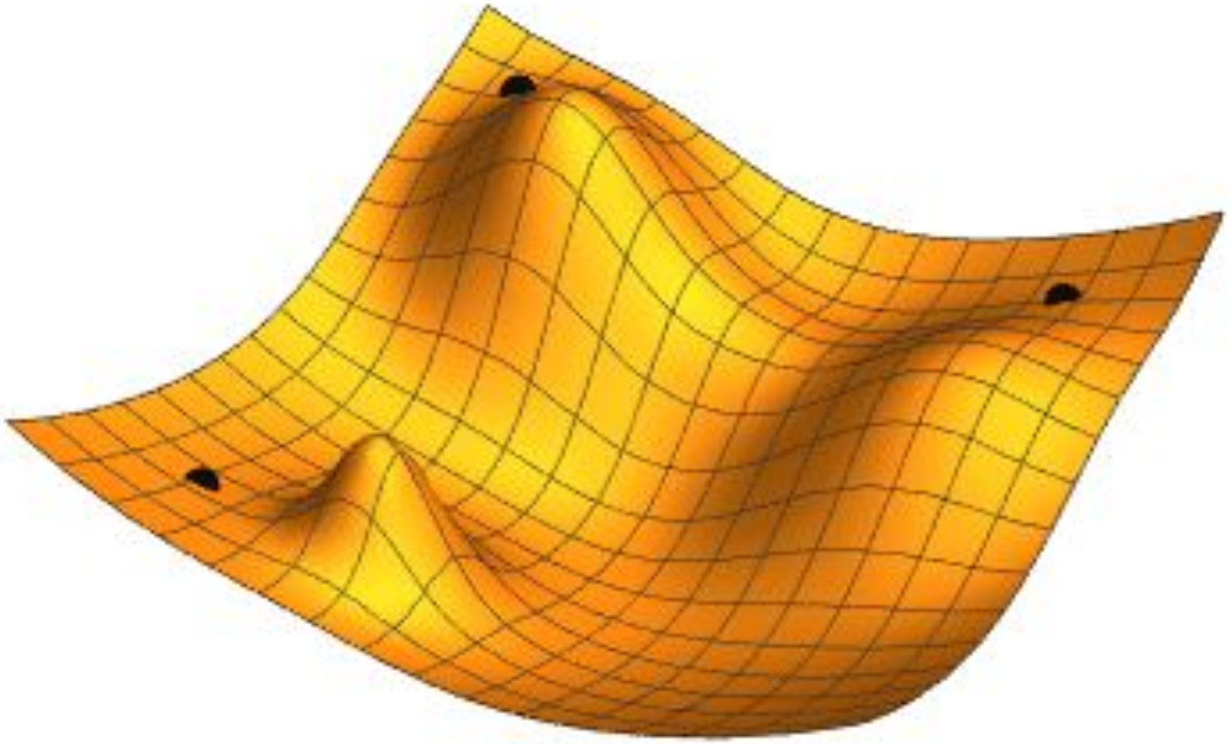
repeat until convergence {

$$\theta_j := \theta_j - \alpha \frac{\partial}{\partial \theta_j} J(\theta_0, \theta_1) \quad \text{(simultaneously update } j = 0 \text{ and } j = 1)$$

}



How we navigate the cost function via GD



Hyperparameters

- **Hyperparameters** are parameters which determine the model’s performance (or how well we are able to train it)

- These are **parameters outside of the model** which are used to **tune** the model so that it **fits** the training data for the given hypothesis

- We can change the values of these hyperparameters during successive runs of training a model.

- Here you will learn three important hyper parameters:
  - **Learning Rate**
  - **Epoch**
  - **Batch Size**

Learning Rate

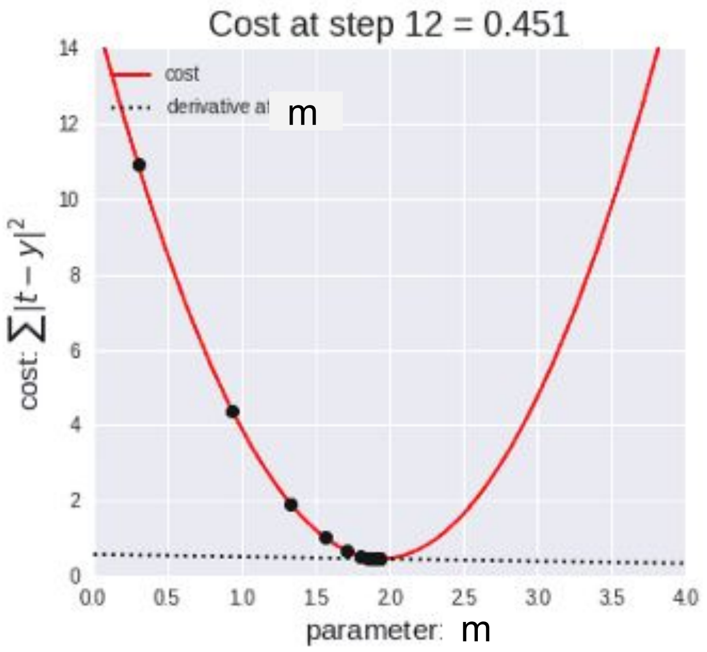
How does learning rate play a role in getting the optimal weights?

Learning Rate

- Learning rate is the rate at which we update (increase or decrease) the values of the parameters (m) during gradient descent

- As you can see in the image, **initially the steps are bigger (huge jumps)**

- As the point goes down, the **steps become shorter(smaller jumps)**

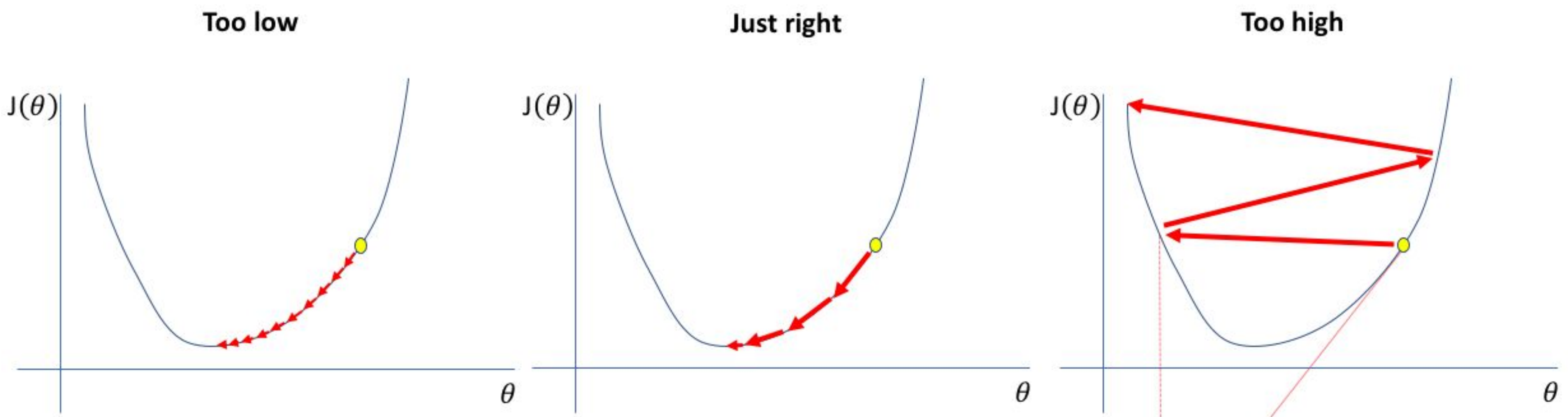


- When updating weights during GD, if the learning rate is high, the weights will be changed aggressively and when it is low, they'll be updated slowly

dφ

Democratizing Data Science Learning

Learning Rate



A small learning rate requires many updates before reaching the minimum point

The optimal learning rate swiftly reaches the minimum point

Too large of a learning rate causes drastic updates which lead to divergent behaviors

Epochs

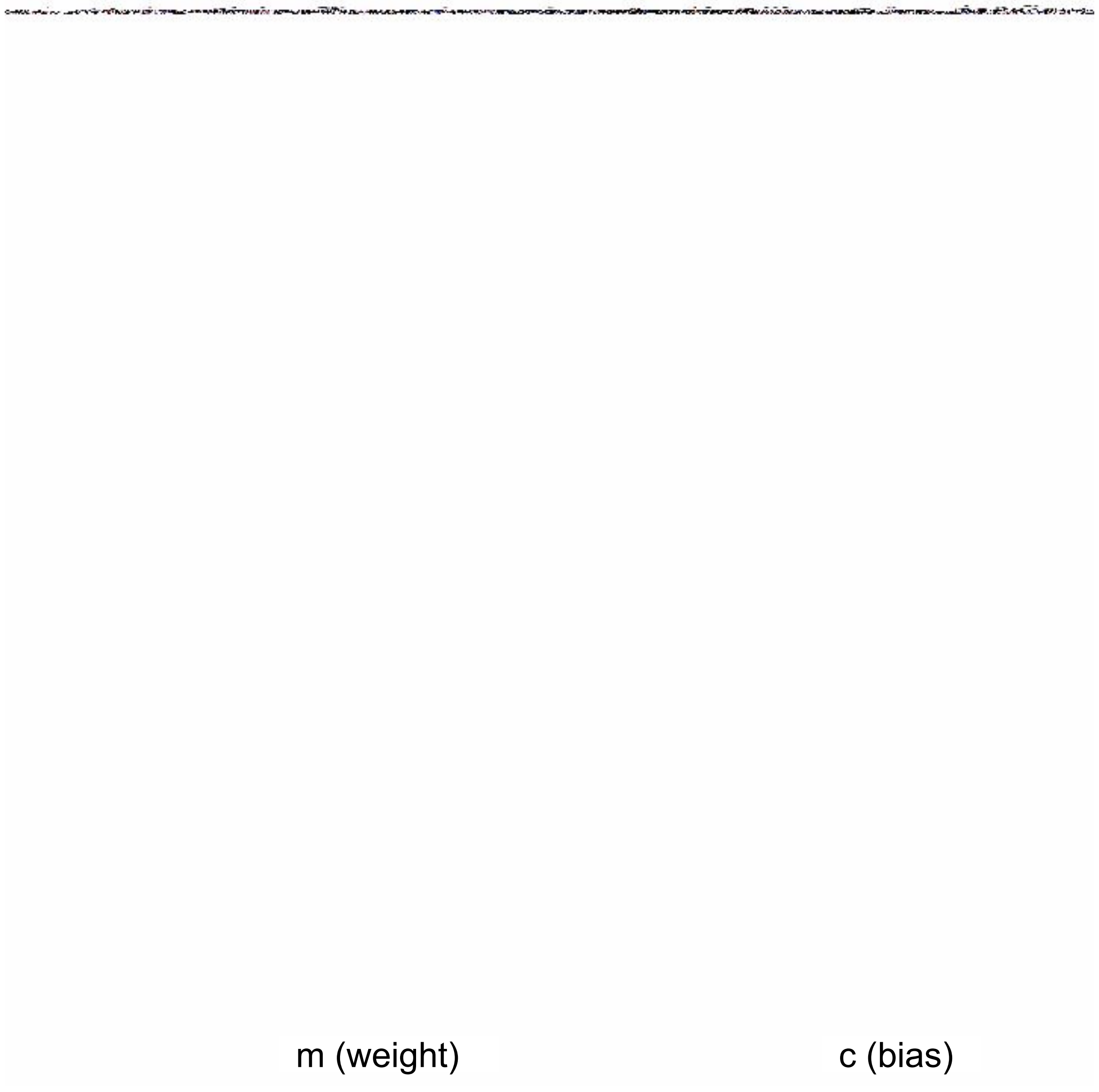
One epoch is when an **ENTIRE dataset is passed forward and backward through the neural network only ONCE.**

Usually, training a neural network takes more than a few epochs. It is because just passing the dataset once through the Neural Network is not enough (depends on the problem too).

We need to pass the full dataset multiple times to the same neural network so that the model is able to **learn** the patterns present in the data

Note: There is no guarantee a network will **converge** or **get better** by letting it learn the data for multiple epochs. It is an art in machine learning to decide the number of epochs sufficient for a network.

Epochs



Batch

When the data is too big, we can’t pass all the data to the computer at once (think on the order of millions records of data).

To overcome this problem we need to **divide the data into smaller sizes** and **give it to our computer one by one** and update the weights of the neural networks at the end of every step.

Batch Size

Batch Size = **Total number of training examples present in a single batch.**

But What is a **Batch**?

As we said, you can’t pass the entire dataset into the neural net at once. So, you divide dataset into **Number of Batches** or sets or parts.

Batch Size

Let’s say we have **2000 training examples** that we are going to use .

We can divide the dataset of 2000 samples into **batches of 500**.

What is the **Batch Size** here? → **500**

What will be the **number of batches**? → Simple, 2000/500 = **4**

Thus, when all the 4 batches (the whole data) pass through the model back and forth once, it’ll complete **1 epoch**.

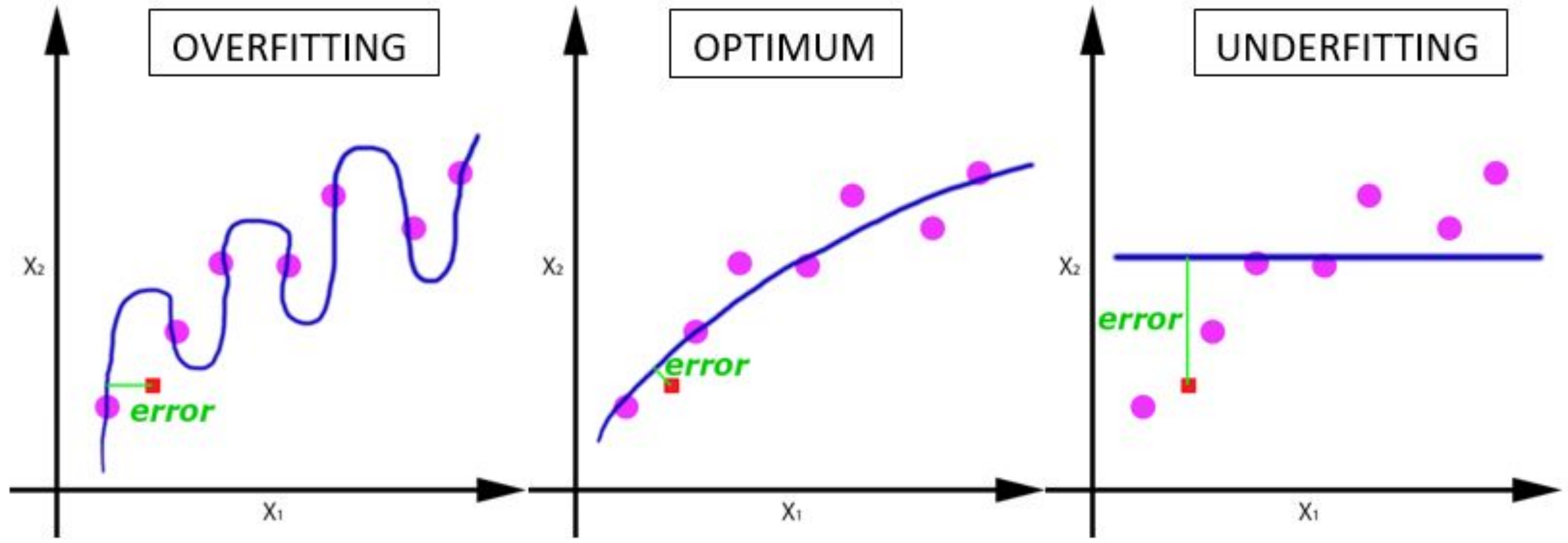
The 5 Step Model Life-Cycle

- A model has a life-cycle, and this very simple knowledge provides the backbone for both modeling a dataset and understanding the tf.keras API.
- The five steps in the life-cycle are as follows:
  - Define the model.
  - Compile the model.
  - Fit the model.
  - Make predictions on the test data.
  - Evaluate the model.
- The above 5 steps are explained in the notebook with practical implementation.

Notebook for practice

- Beginners’ Notebook
- Intermediate Learners Notebook

Overfit - Optimal - Underfit Example



Slide Download Link

- You can download the slides here: https://docs.google.com/presentation/d/1VuNfO4p5Gumm_UT_t84n9jptTN-REoDIZip5Dx2JxB8/edit?usp=sharing

References

- https://towardsdatascience.com/epoch-vs-iterations-vs-batch-size-4dfb9c7ce9c9

That's it for the day. Thank you!

Feel free to post any queries on Discuss
or in the #help channel on Slack