Gradient Descent in MLR

Course	
Topic	
Type	Lecture
Done	☐
Date	@July 18, 2022
Materials	https://drive.google.com/file/d/1nJT4W5YT7dK_6JI6uvikMsLkWsy_OK6z/view

▼ Summary

Why Feature Scaling?

- Feature Scaling helps in making gradient descent much faster.

It is explained in the below process:

- As a concrete example, let's predict the price of a house using two features **x1:** the size of the house and **x2:** the number of bedrooms. Let's say that x1 typically ranges from 300 to 2000 square feet. And x2 in the data set ranges from 0 to 5 bedrooms. So for this example, x1 takes on a relatively large range of values and x2 takes on a relatively small range of values.

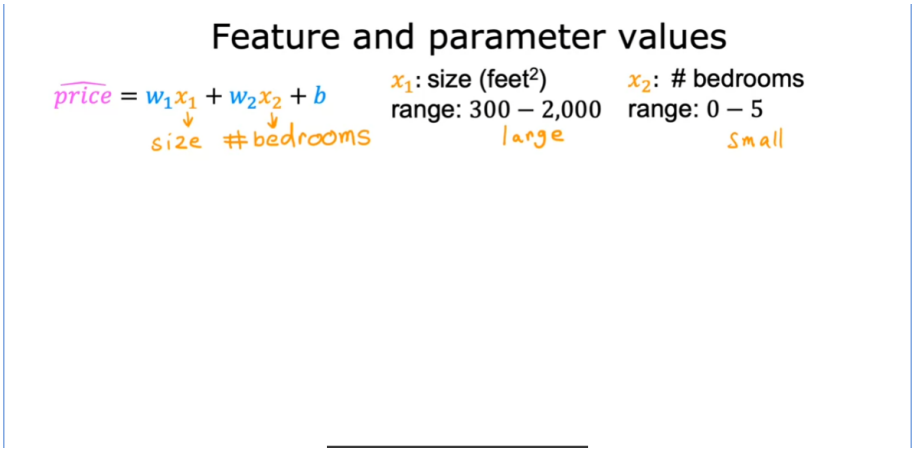

Figure 1: Depicting the house prediction example

- Now let's take an example of a house that has a size of 2000 square feet has five bedrooms and a price of 500k or $500,000.

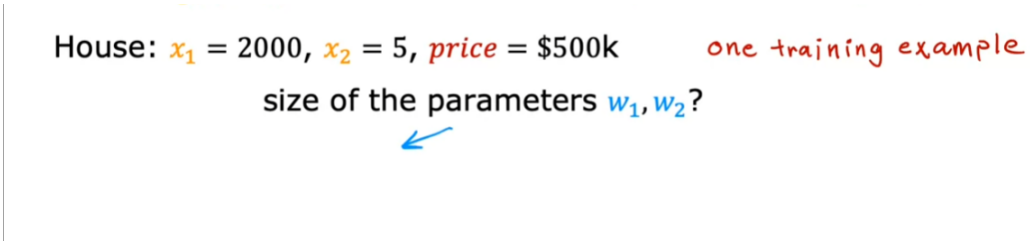

Figure 2: Explaining the above example

- For this one training example, what do you think are reasonable values for the size of parameters w1 and w2?

Case 1:

- **w1=50 ,w2=0.1,b=50**

Case 2:

- **w1=0.1,w2= 50 , b=50**

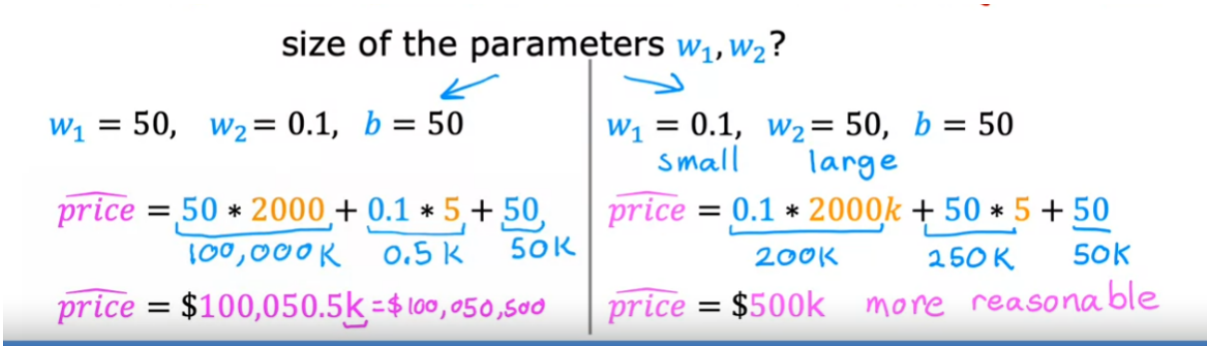

Figure 3: Comparing the two cases

Therefore, when a possible range of values of a feature is **large i**t's more likely that a good model will learn to choose a **relatively small parameter value.** Likewise, when the possible values of the feature are **small** then a reasonable value for its parameters will be relatively **large.**

How does this relate to Gradient Descent ?

- Let's take a look at the scatter plot of the features (**figure 4 yellow**) where the size square feet is the horizontal axis x1 and the number of bedrooms is on the vertical axis.If you plot the training data, you notice that the horizontal axis is on a much larger scale or much larger range of values compared to the vertical axis.
- Let's look at how the cost function (**figure 4 blue**) might look in a contour plot. You might see a contour plot where the horizontal axis has a much narrower range, say between zero and one, whereas the vertical axis takes on much larger values, say between 10 and 100. So the contours form ovals or ellipses and they're short on one side and longer on the other. And this is because a very small change to **w1** can have a very large impact on the estimated price and that's a very large impact on the cost **J.** Because **w1** tends to be multiplied by a very large number, the size and square feet. In contrast, it takes a much larger change in **w2** in order to change the predictions much. And thus small changes to **w2**, don't change the cost function nearly as much.

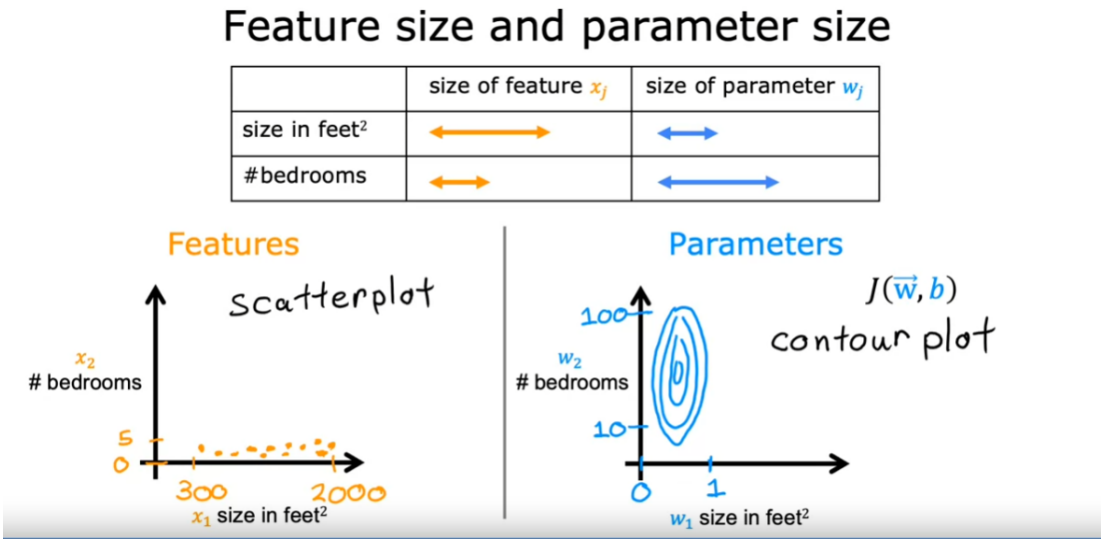

Figure 4: How does different values of w affect cost function?

- So, if you were to run gradient dissent on the training data without scaling the features, the contours are so tall and skinny that gradient descent may end up bouncing back and forth(Figure 5) for a long time before it can finally find its way to the global minimum.

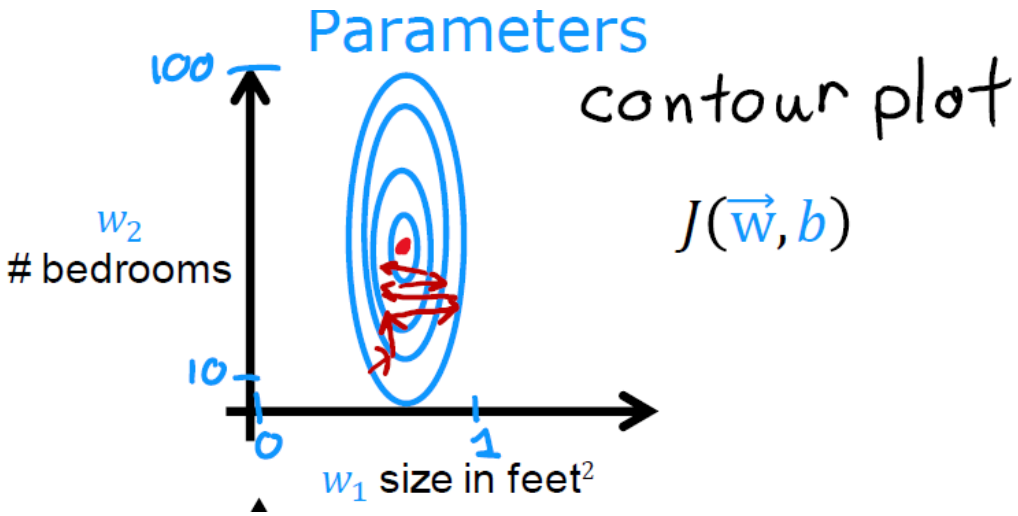

Figure 5: Contour plot if features are not scaled

- In situations like this, a useful thing to do is to **scale the features**. This means performing some transformation of your training data so that x1 say might now range from 0 to 1 and x2 might also range from 0 to . So, the contours will look more like this more like circles and less tall and skinny. Therefore, gradient descent can find a much more direct path to the global minimum

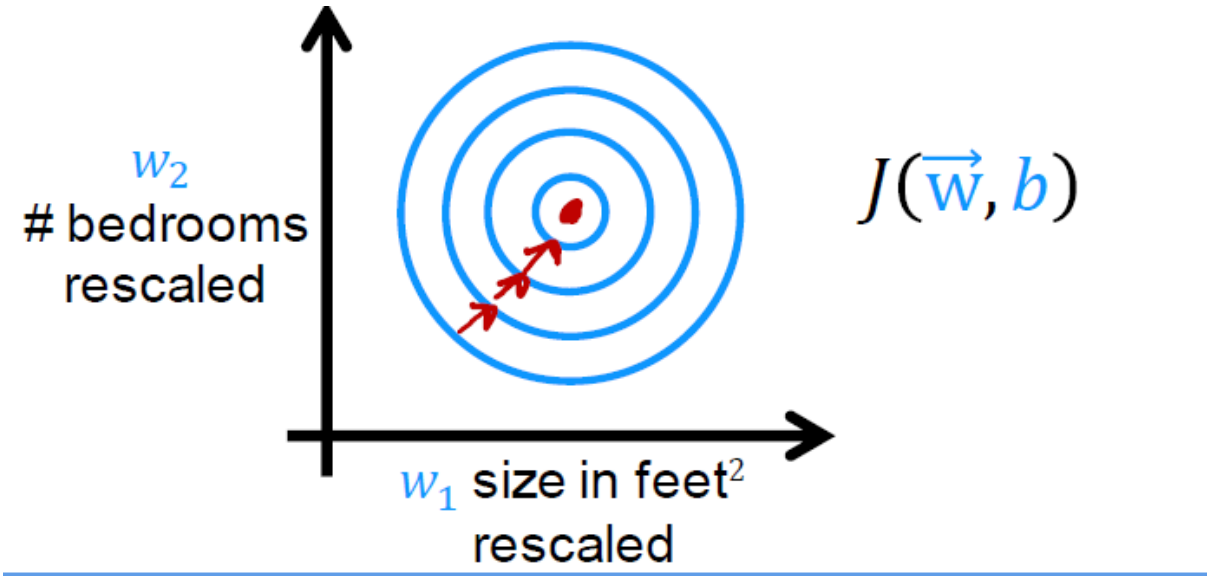

Figure 5: Contour plot if features are scaled

> To summarize, when you have different features that take on very different ranges of values, can cause gradient descent to run slowly but re scaling the different features so they all take on comparable range of values can speed up gradient descent significantly.

Implementing Feature Scaling

There are three methods to rescale our given features

1) Dividing by the maximum value of the range :

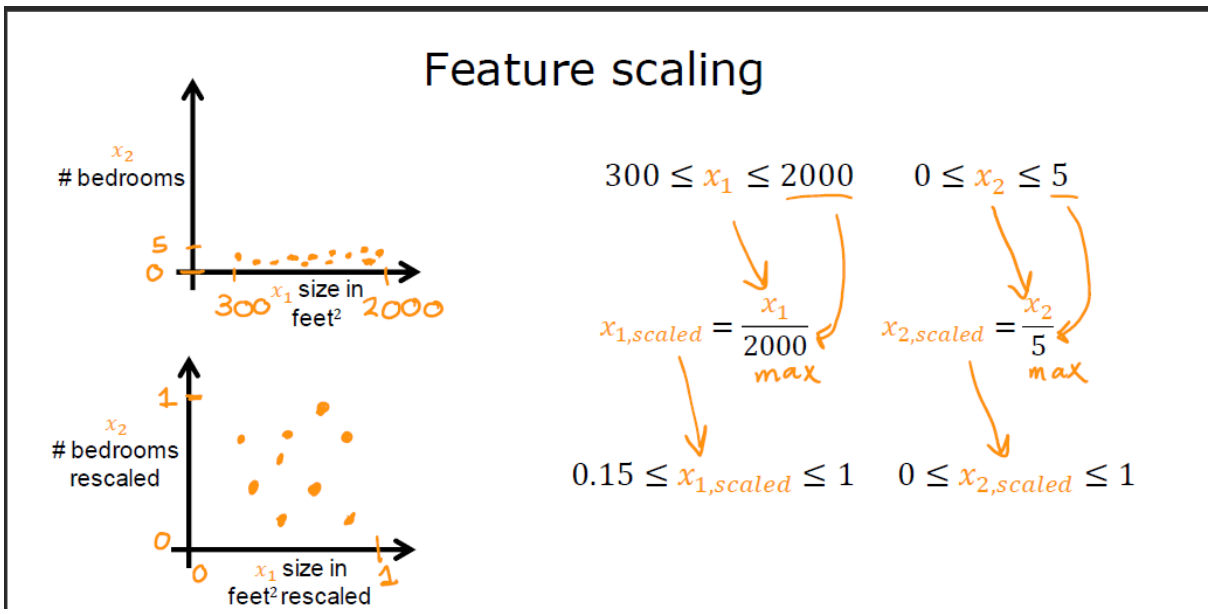

Figure 6: Feature Scaling by dividing each value of x by the maximum value

2) Mean Normalization:

- In mean normalization, you re-scale the features so that all of them are centered around zero.
- Before the feature only had values greater than zero, now they have both negative and positive values that may be usually between **-1 and 1.**

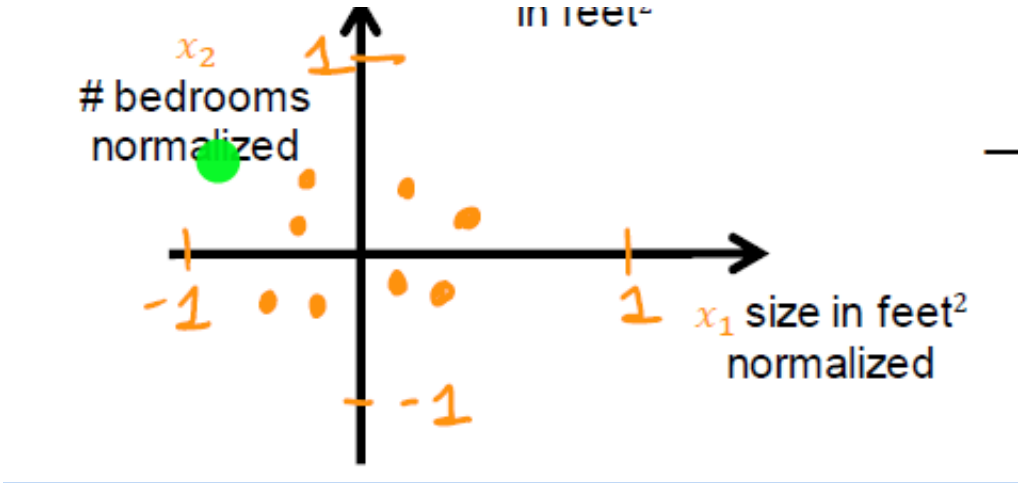

Figure 7 : Feature Scaling by mean normalization

How to perform mean normalization?

- To calculate the mean normalization of x_1, first find the average, also called the mean of x_1 on your training set, and let's call this mean μ_1.

Mean normalization = $\frac{x_1 - \mu_1}{max - min}$

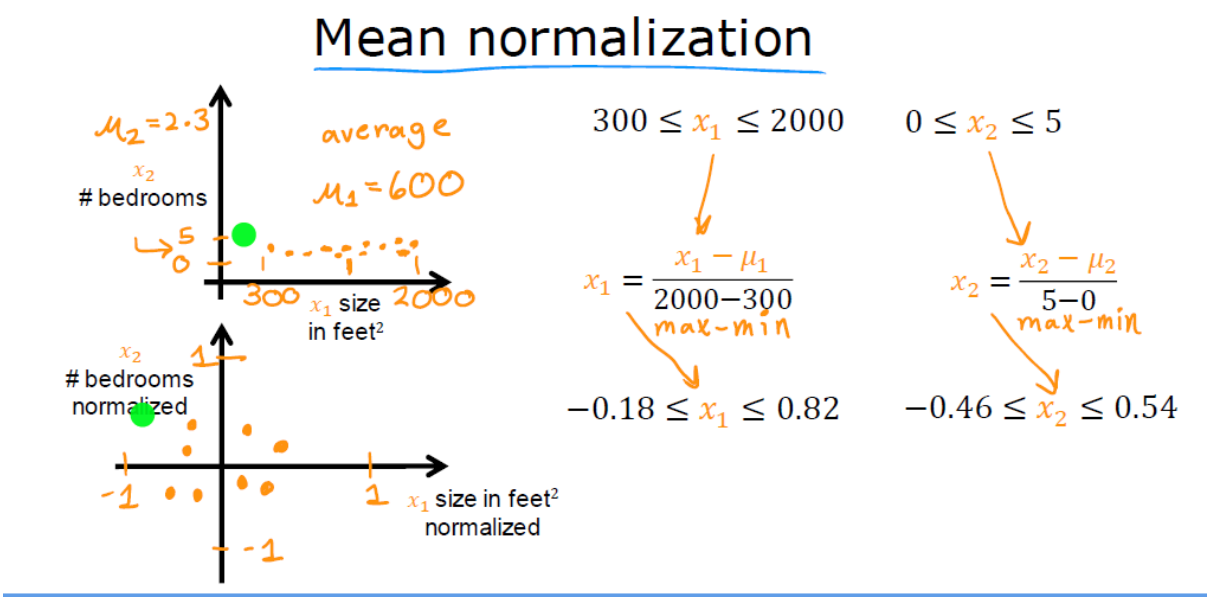

Figure 8 : Demonstating mean normalization

3) Z score normalization:

Z score normalization = $\frac{x_1 - \mu_1}{\sigma_1}$

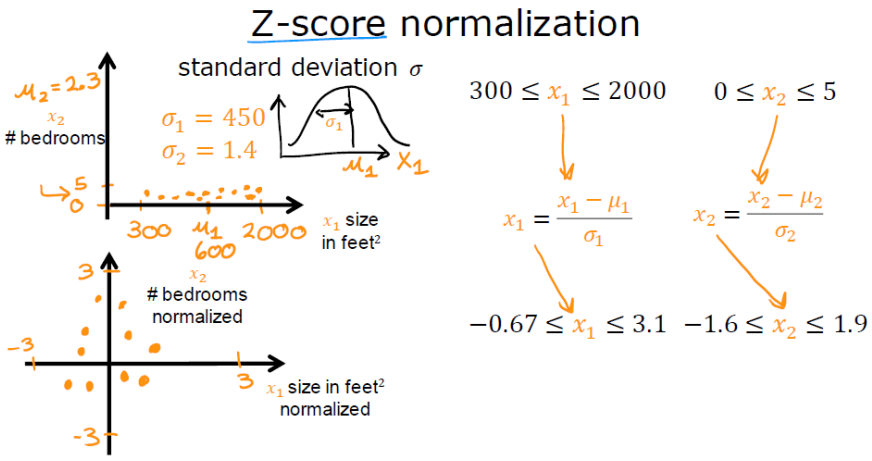

Figure 8 : Demonstating Z-score normalization

When to apply Feature Scaling?

- When performing feature scaling, you might want to aim for getting the features to range from maybe anywhere around **-1 to +1** for each feature x.
- But these values, **-1 to +1** can be a little bit loose. If the features range from **-3 to +3** or **-0.3 to +0.3,** all of these are completely okay.

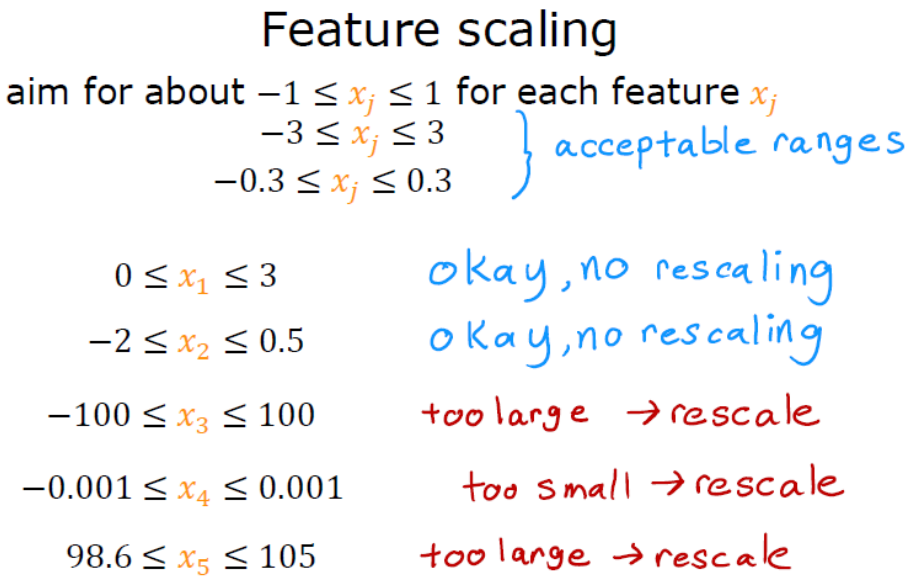

Figure 9: Some examples of when one should rescale the features

Checking gradient descent for covergence:

- The job of gradient descent is to find parameters **w and b** that hopefully minimize the cost function **J**.
- There are **2 methods** to check if gradient descent is working properly or not.

Method1: Plot a Learning Curve

- Plot the cost function J, which is calculated on the training set, and plot the value of J at each iteration of gradient descent.
- Each iteration means after each simultaneous update of the parameters w and b.
- In this plot, the horizontal axis is the number of iterations of gradient descent.

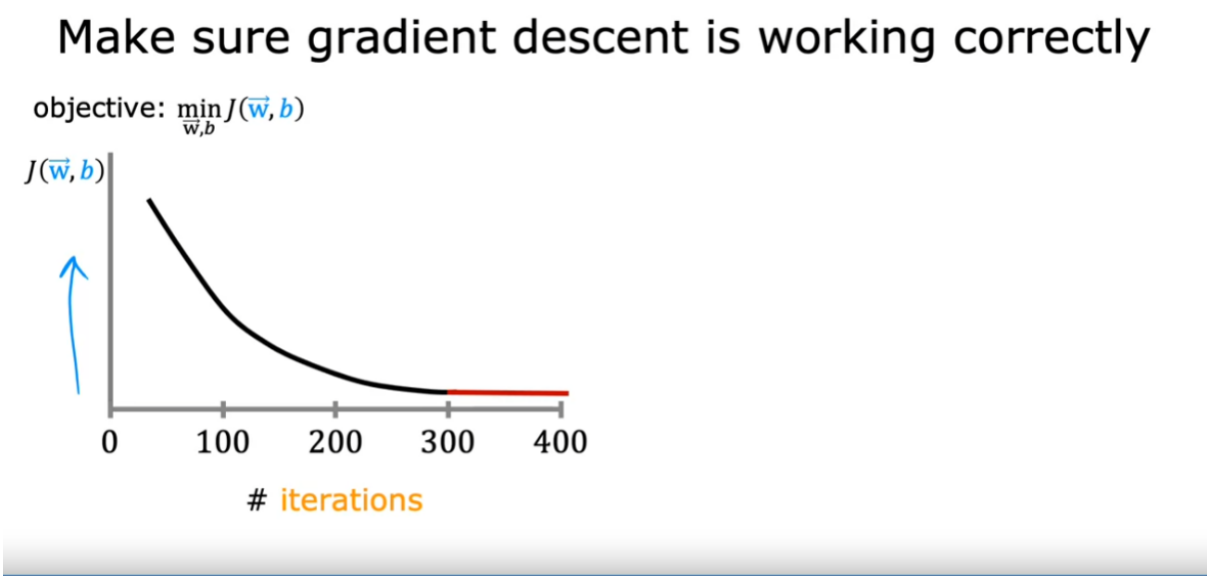

Figure 10: A learning curve

- Looking at this graph helps you to see how your **cost J changes** after each iteration of gradient descent. If gradient descent is working properly, then the cost J should decrease after every single iteration. If J ever increases after one iteration, that means either Alpha is chosen poorly, and it usually means Alpha is too large, or there could be a bug in the code.
- Looking at this learning curve, you can try to spot whether or not **gradient descent is converging**(i.e remains constant and close to the global minimum).

Method 2: Automatic Convergence test

- Let ϵ be a variable representing a small number, such as 0.001 or 10^-3.
- If the **cost J decreases by less than** ϵ on one iteration, then you're likely on the flattened part of the curve and you can declare convergence(you found parameters w and b that are close to the minimum possible value of J).

Which method is better and why?

The learning curve method is better because:

- Choosing the right threshold ϵ is pretty difficult.

- Looking at the learning curve, we get some advanced warning if maybe gradient descent is not working correctly.

Choosing the learning rate

- If you plot the cost for a number of iterations and notice that the **cost function J** sometimes goes up and sometimes goes down, you should take that as a clear sign that gradient descent is not working properly.

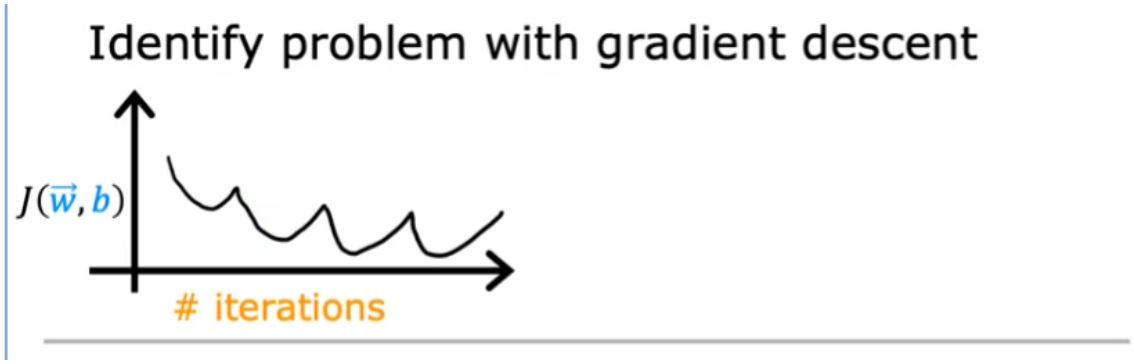

Figure 11: Identifying problem with gradient descent

- This could mean that there's a bug in the code or sometimes it could mean that your learning rate is too large.
- If the gradient descent is too high, something like the below figure might happen.

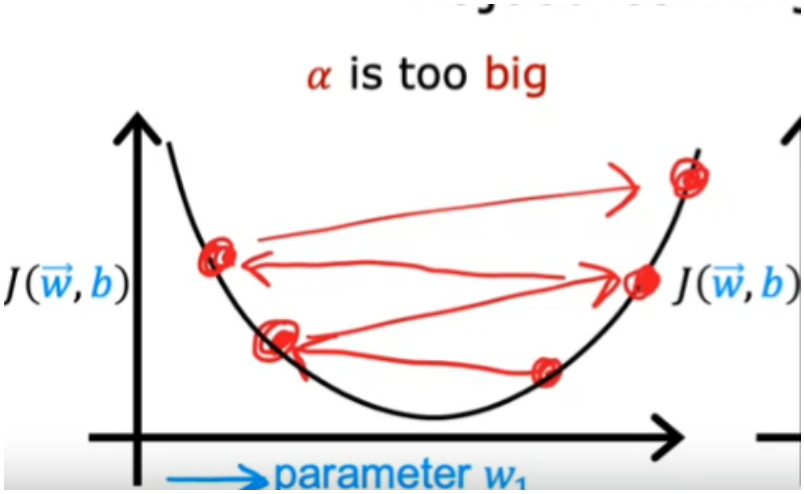

Figure 12: If α is too high

- So one should use a smaller α.

How to choose the correct α:

- Try a range of values like 0.001,0.003……..1.
- **After trying 0.001, then increase the learning rate threefold to 0.003**.
- After that, try 0.01, which is again about three times as large as 0.003.
- So we are roughly trying out gradient descents with each value of α being roughly **three times** bigger than the previous value.
- What one should do is try a range of values until you find the value of that's too small as well as find a value that's too large.
- Then slowly try to pick the largest possible learning rate, or just something slightly smaller than the largest reasonable value that I found.

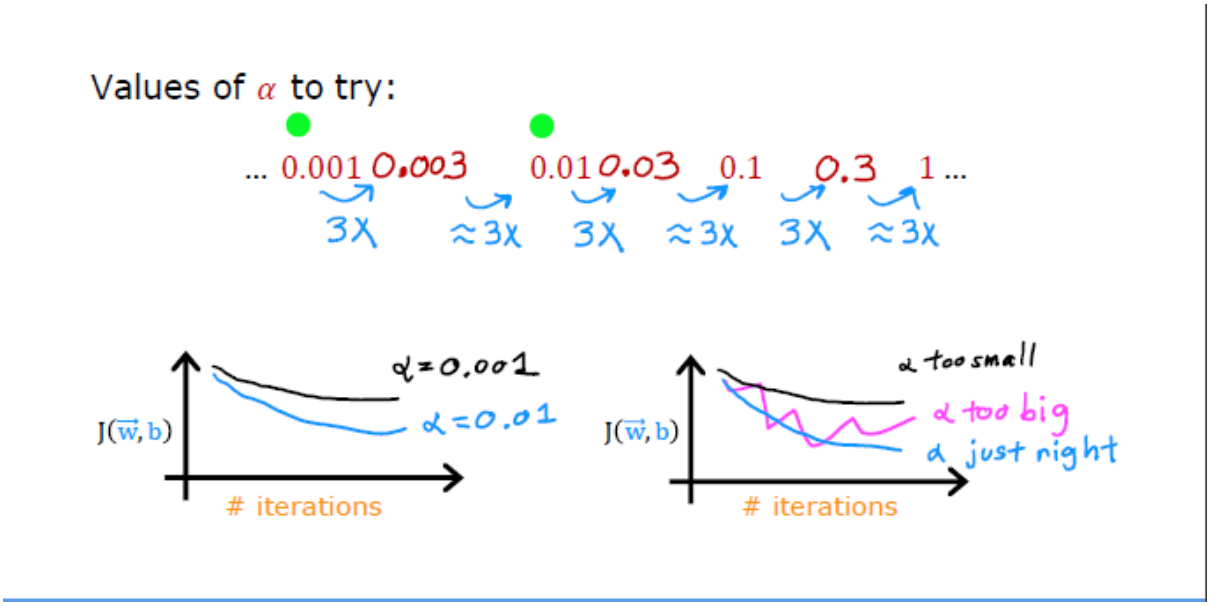

Figure 13: Choosing different values of α

Feature Engineering:

The choice of features can have a huge impact on your learning algorithm's performance. In fact, for many practical applications, choosing or entering the right features is a critical step to making the algorithm work well.

Choosing the correct features for our learning algorithm:

- Let's take a look at feature engineering by revisiting the example of predicting the price of a house. Say you have two features for each house where x_1 is the width of the plots of land and the second feature, x_2, is the depth of rectangular plot of land.
- Given these two features, $x_1$and x_2, you might build a model like this where:

$$f(x) = w_1x_1 + w_2x_2 + b$$

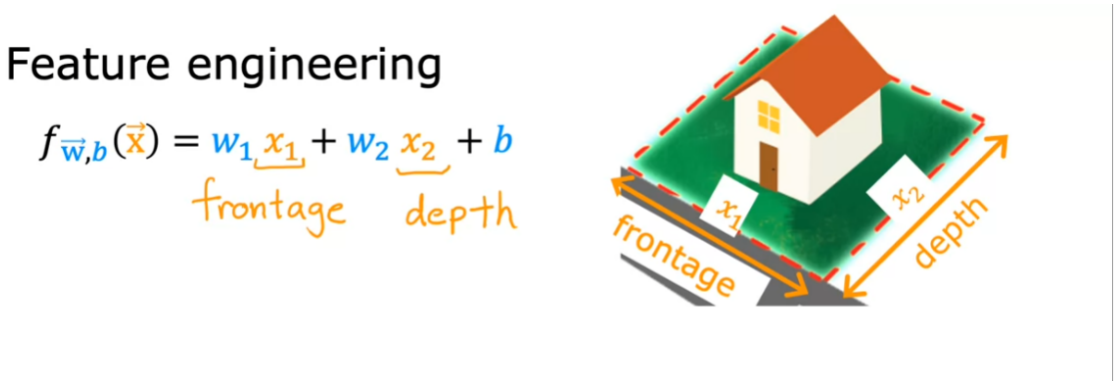

Figure 14: Building a model to predict the cost of house

- This model might work okay. But here's another option for how you might choose a different way to use these features in the model that could be even more effective.
- You might notice that the area of the land can be calculated as $x_1 * x_2$. You may have an intuition that the area of the land is more predictive of the price, than the frontage and depth as separate features.
- You might define a new feature, $x_3 = x_1 * x_2$.
- This new feature x_3 is equal to the **area of the plot of land**.
- With this feature, you can then have a model:

$$f(x) = w_1x_1 + w_2x_2 + w_3x_3 + b$$

- The model can now choose parameters $w_1, w_2, and\ w_3$.

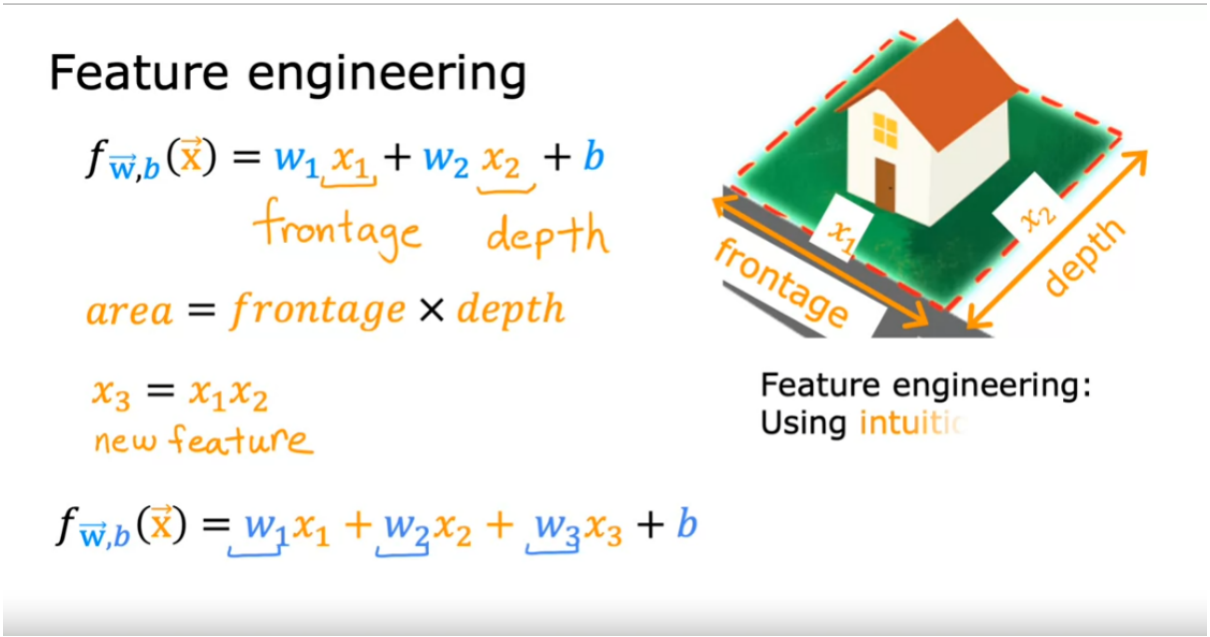

Figure 15: Explaing feature engineering for the house price example

- What we just did, creating a new feature is an example of what's called **feature engineering,** in which you might use your knowledge or intuition about the problem to design new features usually by transforming or combining the original features of the problem in order to make it easier for the learning algorithm to make accurate predictions.
- Depending on what insights you may have into the application, rather than just taking the features that you happen to have started off with, sometimes by defining new features, you might be able to get a much better model. That's **feature engineering.**

Polynomial Regression

Suppose you have a dataset like this:

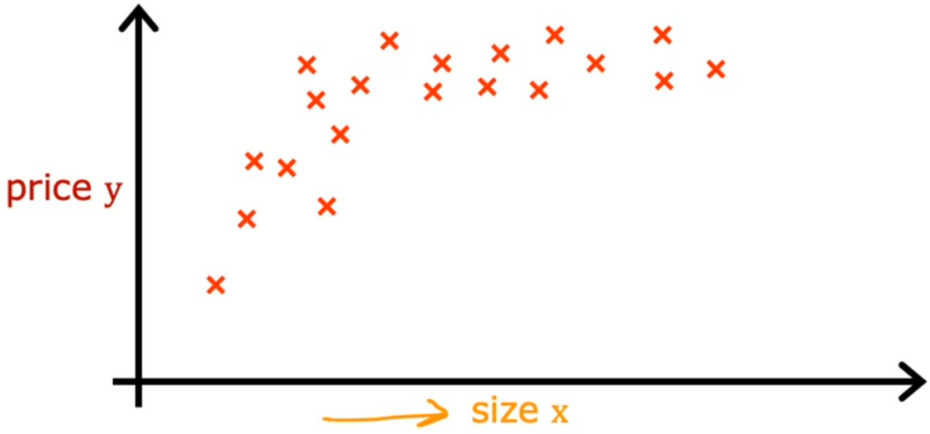

Figure 16: Sample dataset for polynomial regression

Here we can find that a curve is better than a straight line to fit the data. So we may use a polynomial function for regression.

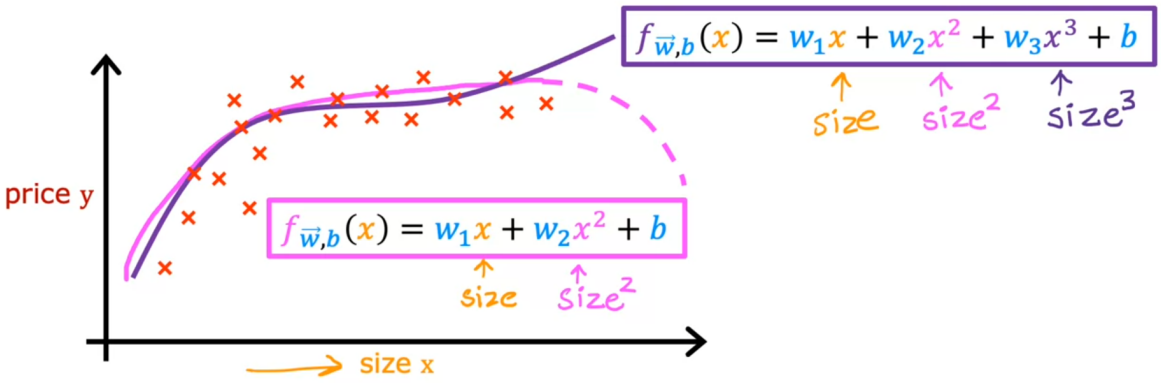

Figure 17: Cubic function(purple) and quadratic function(pink) to fit our training data

In polynomial function, feature scaling becomes extremely important to get features into comparable ranges of values.