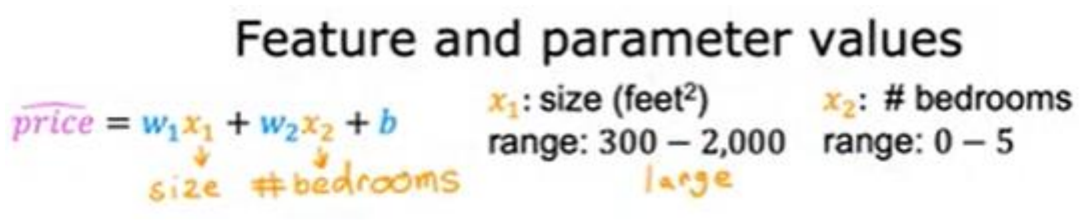
Mise à l'échelle des caractéristiques, partie 1

Let's start by taking a look at the relationship between the size of a feature that is how big are the numbers for that feature and the size of its associated parameter. 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.



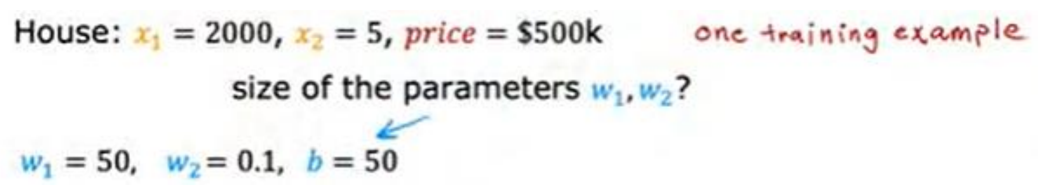
Feature and parameter values

$\widehat{price} = w_1 \underset{\text{size}}{\underset{\downarrow}{x_1}} + w_2 \underset{\text{\#bedrooms}}{\underset{\downarrow}{x_2}} + b$

x_1: size (feet^2)
range: 300 – 2,000
large

x_2: # bedrooms
range: 0 – 5

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. For this one training example, what do you think are reasonable values for the size of parameters w1 and w2? Well, let's look at one possible set of parameters. Say w1 is 50 and w2 is 0.1 and b is 50 for the purposes of discussion.



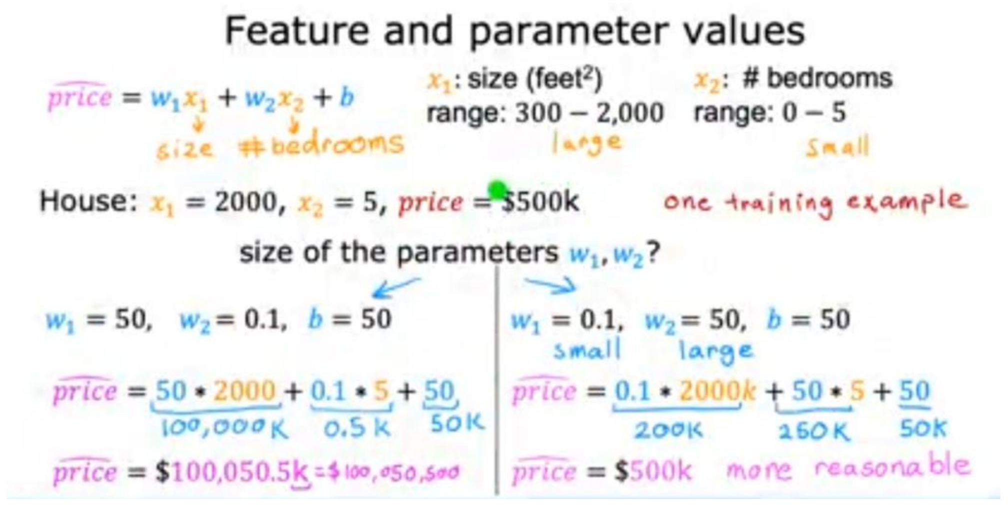
House: $x_1 = 2000$, $x_2 = 5$, $price = \$500k$ one training example

size of the parameters w_1, w_2?

$w_1 = 50$, $w_2 = 0.1$, $b = 50$

So in this case the estimated price in thousands of dollars is 100,000k here plus 0.5 k plus 50 k. Which is slightly over 100 million dollars. So that's clearly very far from the actual price of $500,000. And so this is not a very good set of parameter choices for w1 and w2.

Now let's take a look at another possibility. Say w1 and w2 were the other way around. W1 is 0.1 and w2 is 50 and b is still also 50. In this choice of w1 and w2, w1 is relatively small and w2 is relatively large, 50 is much bigger than 0.1. So here the predicted price is 0.1 times 2000 plus 50 times five plus 50. The first term becomes 200k, the second term becomes 250k, and the plus 50. So this version of the model predicts a price of $500,000 which is a much more reasonable estimate and happens to be the same price as the true price of the house.



Feature and parameter values

$\widehat{price} = w_1 \underset{\text{size}}{\underset{\downarrow}{x_1}} + w_2 \underset{\text{\#bedrooms}}{\underset{\downarrow}{x_2}} + b$

x_1: size (feet^2)
range: 300 – 2,000
large

x_2: # bedrooms
range: 0 – 5
small

House: $x_1 = 2000$, $x_2 = 5$, $price = \$500k$ one training example

size of the parameters w_1, w_2?

$w_1 = 50$, $w_2 = 0.1$, $b = 50$

$\widehat{price} = \underbrace{50 * 2000}_{100{,}000K} + \underbrace{0.1 * 5}_{0.5K} + \underbrace{50}_{50K}$

$\widehat{price} = \$100{,}050.5k = \$100{,}050{,}500$

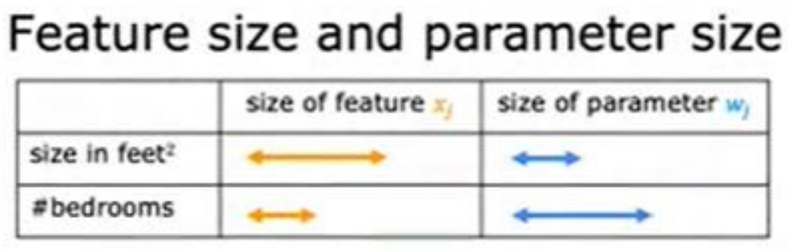
$w_1 = 0.1$, $w_2 = 50$, $b = 50$
small large

$\widehat{price} = \underbrace{0.1 * 2000k}_{200K} + \underbrace{50 * 5}_{250K} + \underbrace{50}_{50K}$

$\widehat{price} = \$500k$ more reasonable

So hopefully you might notice that when a possible range of values of a feature is large, like the size and square feet which goes all the way up to 2000. It's more likely that a good model will learn to choose a relatively small parameter value, like 0.1. Likewise, when the possible values of the feature are small, like the number of bedrooms, then a reasonable value for its parameters will be relatively large like 50.

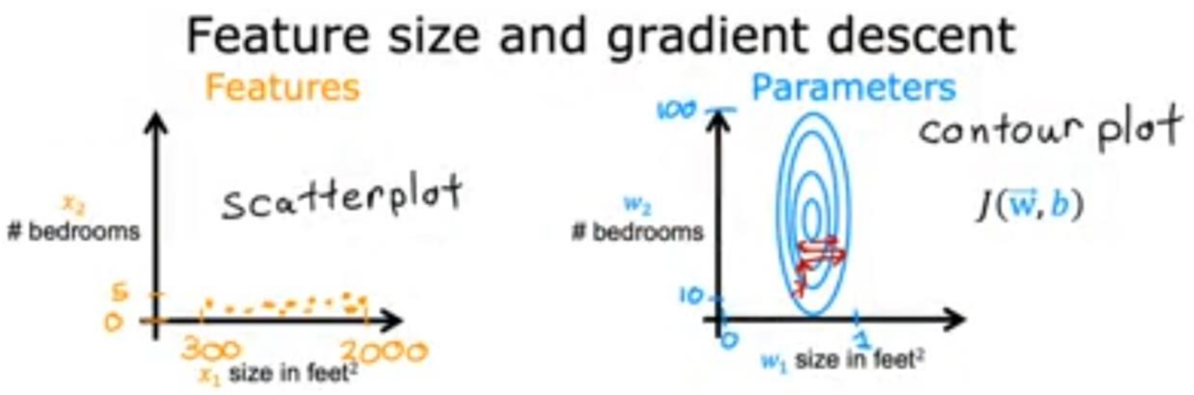
So how does this relate to grading descent? Well, let's take a look at the scatter plot of the features where the size square feet is the horizontal axis x1 and the number of bedrooms exudes is on the vertical axis. If you plot the training data, you notice that the horizontal axis is on a much larger scale or much larger range of values compared to the vertical axis.



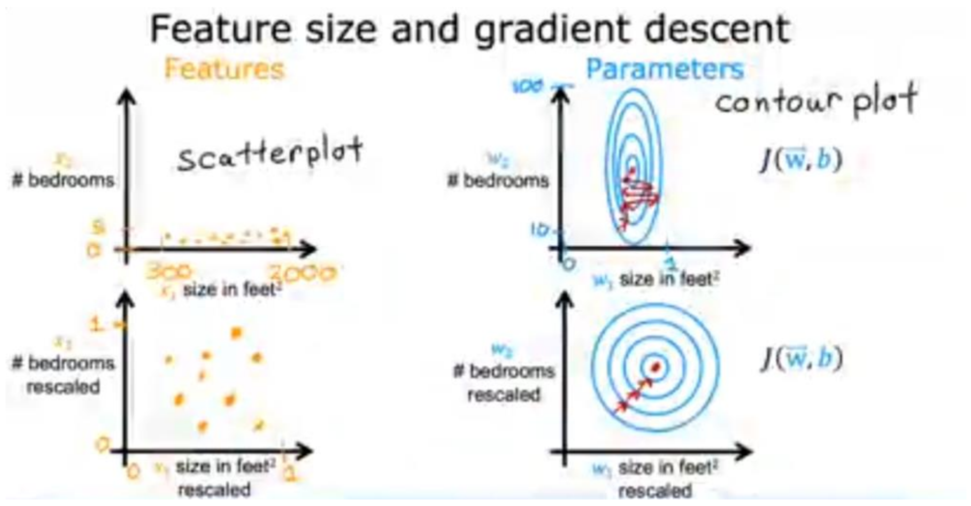
Feature size and parameter size

	size of feature x_j	size of parameter w_j
size in feet^2	⟵⟶	⟷
#bedrooms	⟷	⟵⟶

Next let's look at how the cost function 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. So where does this leave us? This is what might end up happening if you were to run great in dissent, if you were to use your training data as is. Because the contours are so tall and skinny gradient descent may end up bouncing back and forth for a long time before it can finally find its way to the global minimum.



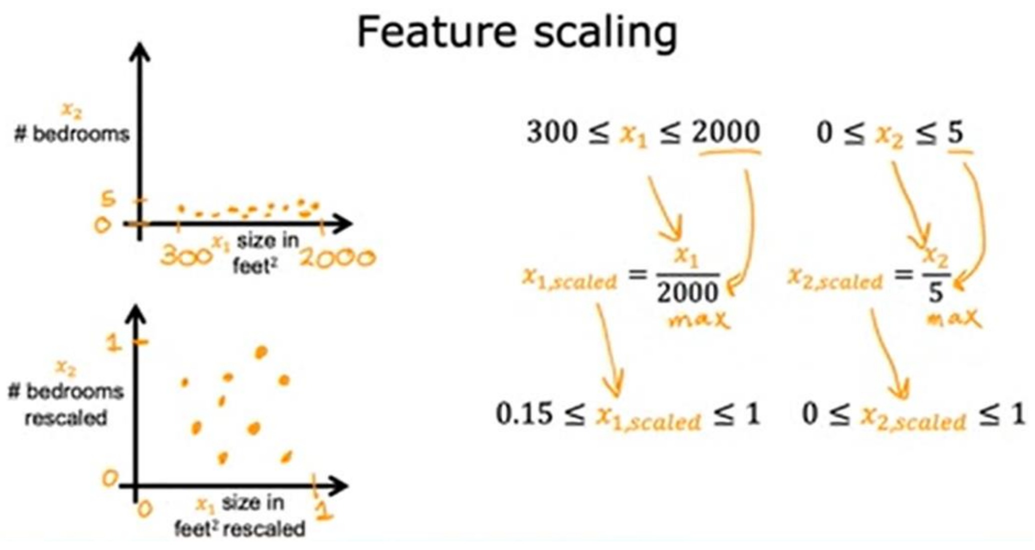
In situations like this, a useful thing to do is to scale the features. This means performing some transformation of your training data so that x1 say might now range from 0 to 1 and x2 might also range from 0 to 1. So the data points now look more like this and you might notice that the scale of the plot on the bottom is now quite different than the one on top.

The key point is that the re scale x1 and x2 are both now taking comparable ranges of values to each other. And if you run gradient descent on a cost function to find on this, re scaled x1 and x2 using this transformed data, then the contours will look more like this more like circles and less tall and skinny. And gradient descent can find a much more direct path to the global minimum.

Mise à l'échelle des caractéristiques, partie 2

Let's look at how you can implement feature scaling, to take features that take on very different ranges of values and skill them to have comparable ranges of values to each other. How do you actually scale features? Well, if x_1 ranges from 3-2,000, one way to get a scale version of x_1 is to take each original x1_ value and divide by 2,000, the maximum of the range. The scale x_1 will range from 0.15 up to one. Similarly, since x_2 ranges from 0-5, you can calculate a scale version of x_2 by taking each original x_2 and dividing by five, which is again the maximum. So the scale is x_2 will now range from 0-1.

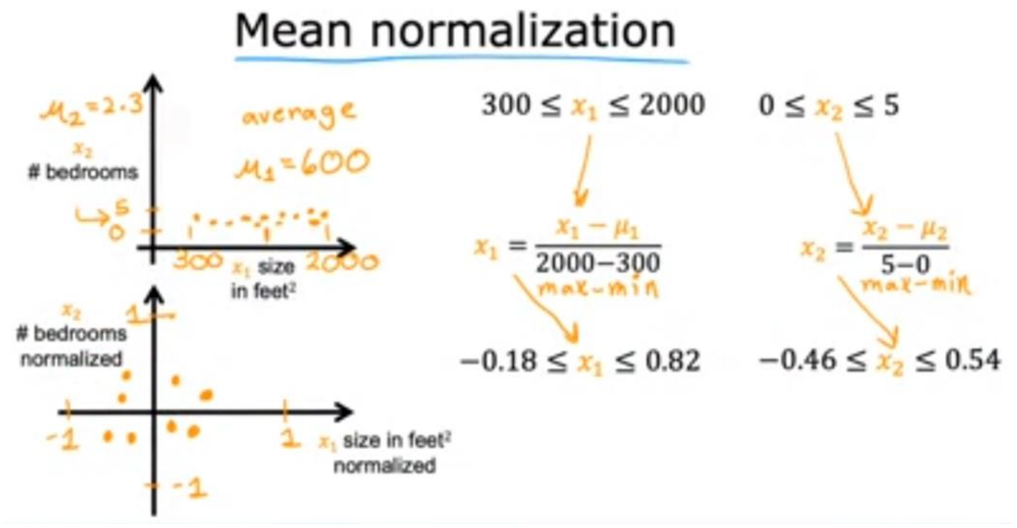
If you plot the scale to x_1 and x_2 on a graph, it might look like this.



In addition to dividing by the maximum, you can also do what's called mean normalization. What this looks like is, you start with the original features and then you re-scale them so that both of them are centered around zero. Whereas before they only had values greater than zero, now they have both negative and positive values that may be usually between negative one and plus one. 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 Mu_1, with this being the Greek alphabets Mu.

For example, you may find that the average of feature 1, Mu_1 is 600 square feet. Let's take each x_1, subtract the mean Mu_1, and then let's divide by the difference 2,000 minus 300, where 2,000 is the maximum and 300 the minimum, and if you do this, you get the normalized x_1 to range from negative 0.18-0.82. Similarly, to mean normalized x_2, you can calculate the average of feature 2. For instance, Mu_2 may be 2.3.

Then you can take each x_2, subtract Mu_2 and divide by 5 minus 0. Again, the max 5 minus the mean, which is 0. The mean normalized x_2 now ranges from negative 0.46-0 54. If you plot the training data using the mean normalized x_1 and x_2, it might look like this.

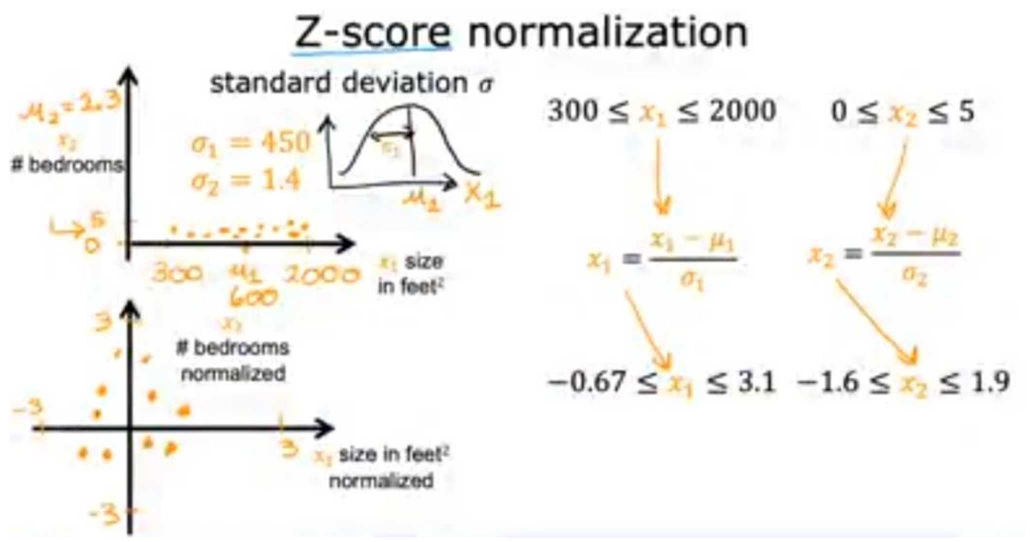


There's one last common re-scaling method call Z-score normalization. To implement Z-score normalization, you need to calculate something called the standard deviation of each feature. If you don't know what the standard deviation is, don't worry about it, you won't need to know it for this course.

Or if you've heard of the normal distribution or the bell-shaped curve, sometimes also called the Gaussian distribution, this is what the standard deviation for the normal distribution looks like. But if you haven't heard of this, you don't need to worry about that either. But if you do know what is the standard deviation, then to implement a Z-score normalization, you first calculate the mean Mu, as well as the standard deviation, which is often denoted by the lowercase Greek alphabet Sigma of each feature.

For instance, maybe feature 1 has a standard deviation of 450 and mean 600, then to Z-score normalize x_1, take each x_1, subtract Mu_1, and then divide by the standard deviation, which I'm going to denote as Sigma 1. What you may find is that the Z-score normalized x_1 now ranges from negative 0.67-3.1.

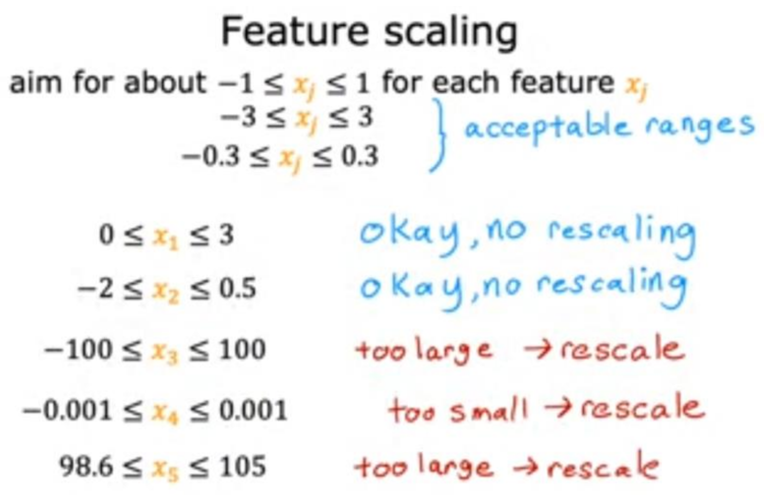
Similarly, if you calculate the second features standard deviation to be 1.4 and mean to be 2.3, then you can compute x_2 minus Mu_2 divided by Sigma_2, and in this case, the Z-score normalized by x_2 might now range from negative 1.6-1.9. If you plot the training data on the normalized x_1 and x_2 on a graph, it might look like this.



As a rule of thumb, when performing feature scaling, you might want to aim for getting the features to range from maybe anywhere around negative one to somewhere around plus one for each feature x. But these values, negative one and plus one can be a little bit loose. If the features range from negative three to plus three or negative 0.3 to plus 0.3, all of these are completely okay. If you have a feature x_1 that winds up being between zero and three, that's not a problem. You can re-scale it if you want, but if you don't re-scale it, it should work okay too.

Or if you have a different feature, x_2, whose values are between negative 2 and plus 0.5, again, that's okay, no harm re-scaling it, but it might be okay if you leave it alone as well. But if another feature, like x_3 here, ranges from negative 100 to plus 100, then this takes on a very different range of values, say something from around negative one to plus one. You're probably better off re-scaling this feature x_3 so that it ranges from something closer to negative one to plus one.

Similarly, if you have a feature x_4 that takes on really small values, say between negative 0.001 and plus 0.001, then these values are so small. That means you may want to re-scale it as well. Finally, what if your feature x_5, such as measurements of a hospital patients by the temperature ranges from 98.6-105 degrees Fahrenheit? In this case, these values are around 100, which is actually pretty large compared to other scale features, and this will actually cause gradient descent to run more slowly. In this case, feature re-scaling will likely help.



Feature scaling

aim for about $-1 \leq x_j \leq 1$ for each feature x_j

$-3 \leq x_j \leq 3$
$-0.3 \leq x_j \leq 0.3$ $\}$ acceptable ranges

$0 \leq x_1 \leq 3$	okay, no rescaling
$-2 \leq x_2 \leq 0.5$	okay, no rescaling
$-100 \leq x_3 \leq 100$	too large → rescale
$-0.001 \leq x_4 \leq 0.001$	too small → rescale
$98.6 \leq x_5 \leq 105$	too large → rescale

There's almost never any harm to carrying out feature re-scaling. When in doubt, I encourage you to just carry it out. That's it for feature scaling. With this little technique, you'll often be able to get gradient descent to run much faster. That's features scaling.