Gradient Descent in Practice

Feature Scaling

Feature and parameter values

❓ **Example:**

Size in feet²	Number of bedrooms	Number of floors	Age of home in years	Price ($) in $1000's
X_1	X_2	X_3	X_4	
2104	5	1	45	460
1416	3	2	40	232
1534	3	2	30	315
852	2	1	36	178
...	...	...	...	...

x_j = j^{th} feature
n = number of features
$\vec{x}^{(i)}$ = features of i^{th} training example
$x_j^{(i)}$ = value of feature j in i^{th} training example

With the same above example, we consider:

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

x_1: size ($feet^2$), range: $300 - 2000$

⇒ $Large$

x_2: # bedrooms, range: $0 - 5$

⇒ $Small$

With **House**: $x_1 = 2000,\ x_2 = 5,\ price = \$500k$. Size of parameter w_1, w_2?

The first tuple (w_1, w_2, b)

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

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

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

⇒ Far from $price = \$500k$

The second tuple (w_1, w_2, b)

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

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

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

⇒ more ***Reasonable***

⇒ The above example shows that large ranges can lead to poor parameter choices.

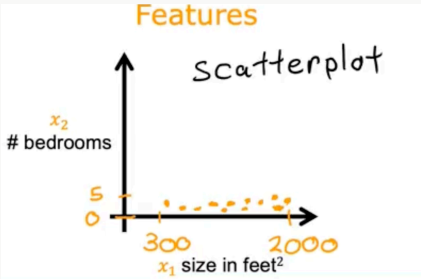
Visualization by Scatter Plot and Contour Plot for deeper comprehension of Feature Scaling

With the range of x_1 (size in $feet^2$) is **LARGE**, the range of x_2 is **SMALL**. Otherwise, the range of parameter w_1 is **SMALL** and w_2 is **LARGE** as following:

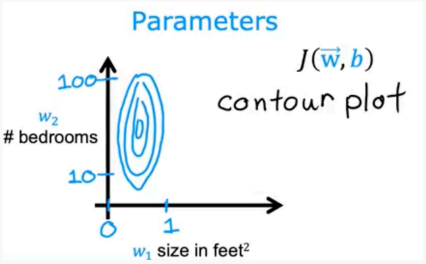
	size of feature x_j	size of parameter w_j
size in feet^2	⟷	⟷
#bedrooms	⟷	⟷

We can visualize it by

Scatter plot for features



Contour plot for parameters



⇒ The contours appear **elongated** or **elliptical**.

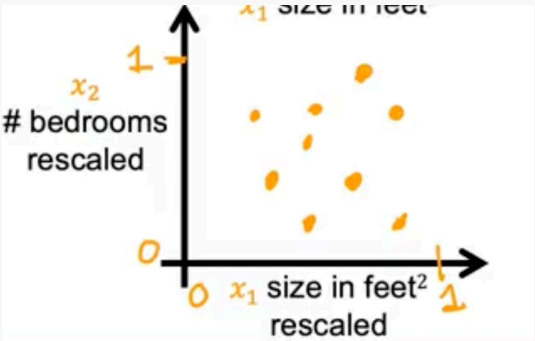
Impact on Gradient Descent:

- 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.
- The steepness of the contours means that small changes in one parameter (like w_1) can lead to **significant changes** in the cost, while larger changes in another parameter (like w_2) may **not** affect the cost as much. This can cause gradient descent to oscillate and take longer to converge to the global minimum.

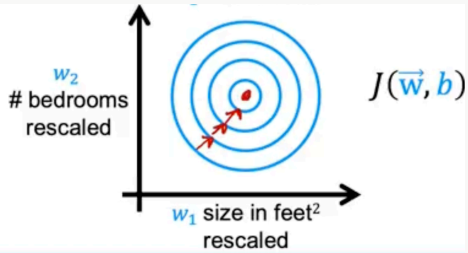
⇒ Useful Solution is to scale the feature

With Feature Scaling

Scale x_1, x_2 to range: $0 - 1$:

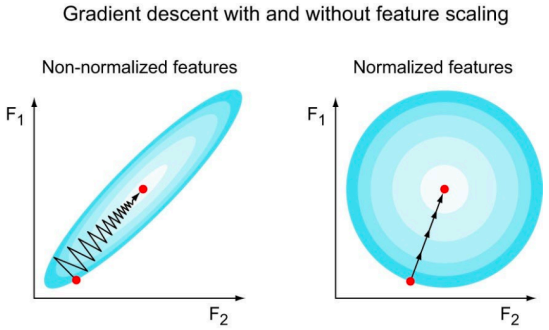


The contour plot is circle, less tall and skinny.



- **Circular Contours**: After scaling the features to a similar range (e.g., 0 to 1), the contours become more **circular**.
- **Improved Gradient Descent**: The circular shape indicates that changes in both parameters have a more **balanced** effect on the cost function. This allows gradient descent to move more *directly* towards the global minimum, *improving convergence speed and efficiency.*

If carefully selected w values are not chosen, the algorithm can often overshoot its target, which will lead to divergence or much longer training times. Properly scaled features should fill an entire graph, when plotted relatively to each other.



Feature Scaling Techniques

Considering the following range for each feature x_i, where $min \leq x_j \leq max$, the goal is to aim for a range of $-1 \leq x_{j,scaled} \leq 1$

Min-Max Scaling

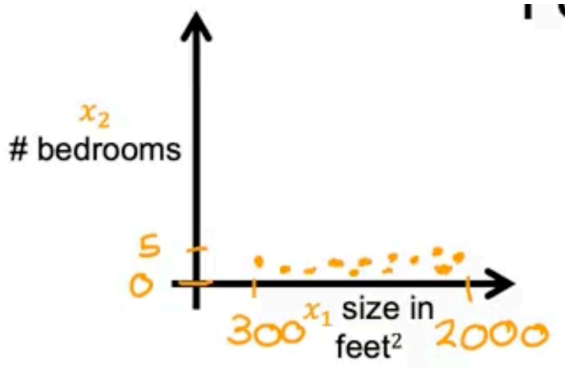
This method involves dividing each feature value by the maximum value of that feature.

$$x_{j,scaled} = \frac{x_i}{max} \qquad \frac{min}{max} \leq x_{j,scaled} \leq 1 \ (\frac{max}{max})$$

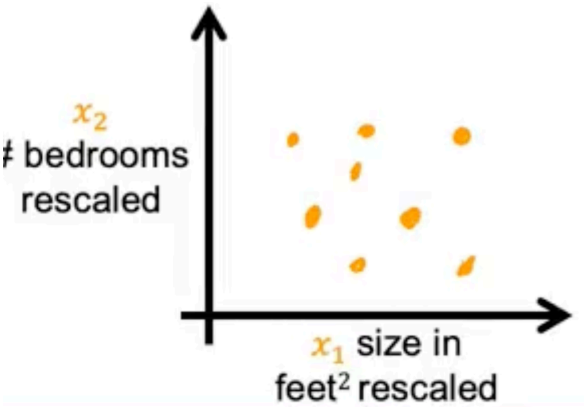
- **For example:**
  - If a feature ranges from $(3, 2000)$, dividing by 2000 scales it to a range of $(0.15, 1)$.

- If a feature ranges from $(0, 5)$, dividing by 5 scales it to a range of $(0, 1)$.

Before Scaling:



After Scaling:



Mean Normalization

This technique centers the features around zero. It involves subtracting the mean of the feature from each value and then dividing by the range $(max - min)$. This results in values typically between -1 and 1.

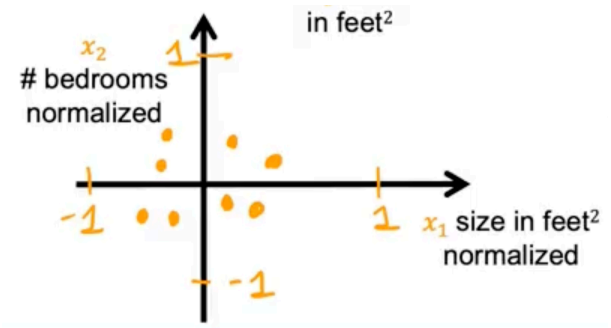
$$\mu_j = \frac{1}{m} \sum_{i=1}^{m} x_j^{(i)} \qquad x_j = \frac{x_j - \mu_j}{\text{max} - \text{min}}$$

Example: Consider a feature x_1, with values ranging from $(30, 2000)$ and x_2, ranging from $(0, 5)$

- Calculate the mean $\mu_1 = 600$
- $x_1 = \frac{x_1 - \mu_1}{2000 - 300}$
- Range: $-0.18 \leq x_1 \leq 0.82$

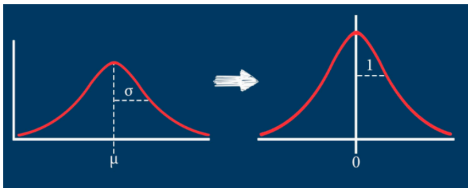
- Calculate the mean $\mu_2 = 2.3$
- $x_2 = \frac{x_2 - \mu_2}{5 - 0}$
- Range: $-0.46 \leq x_2 \leq 0.54$

After nomalization:



Z-score normalization

Z-score normalization uses the **mean** and **standard deviation (σ)** of each feature. It's based on the **normal distribution** (bell curve). Unlike mean normalization, it centers data around zero but doesn't restrict values to between -1 and 1. The method is applied across **m training examples** per feature.



$$\sigma_j = \sqrt{\frac{1}{m} \sum_{i=1}^{m} (x_j{}^{(i)} - \mu_j)} \qquad x_j = \frac{x_j - \mu_j}{\sigma_j}$$

Feature engineering

- Choosing the right features is crucial for the success of machine learning models.
- An example is provided using house price prediction, where the width and depth of a lot are initial features.

$$f_{\vec{w},b}(\vec{x}) = w_1 \underbrace{x_1}_{\text{frontage}} + w_2 \underbrace{x_2}_{\text{depth}} + b$$

We have: $\text{area} = \text{frontage} \cdot \text{depth}$

So we create new feature: $x_3 = x_1 x_2$

$$f_{\vec{w},b}(\vec{x}) = w_1 \underbrace{x_1}_{\text{frontage}} + w_2 \underbrace{x_2}_{\text{depth}} + w_3 \underbrace{x_3}_{\text{area}} + b$$

Feature engineering: Using intuition to design new features, by transforming or combining original features.

Polynomial Regression

In many situations, the relationship between features and outcomes is not purely linear but instead follows a more complex pattern.

For example, house prices might not rise in a straight line with respect to living space; very small and very large houses may not fit such a simple trend. With standard linear regression, adjusting only the weights w and the bias b is not enough to capture these non-linear behaviors.

⇒ a quadratic function may not fit housing prices well, while a cubic function can provide a better fit as it allows for increasing prices with size.

To address this, we can construct additional features from the original ones. For instance, starting with two features x_1 and x_2, we might introduce a new one $x_3 = x_1 x_2$ (**feature engineering**). A special case occurs when we expand features by including their higher powers. In that case, the model becomes:

$$f_{\vec{w},b}(x) = w_1 x + w_2 x^2 + w_3 x^3 + \cdots + w_k x^k + b$$