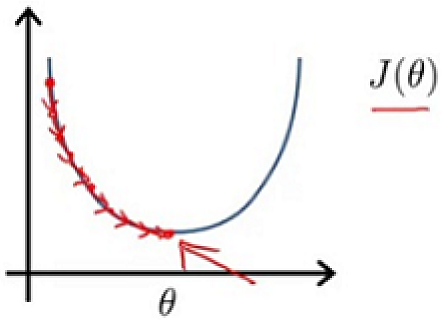
Normal Equation

Concretely, so far the algorithm that we've been using for linear regression is gradient descent where in order to minimize the cost function $J(\theta)$, we would take this iterative algorithm that takes many steps, multiple iterations of gradient descent to converge to the global minimum.

Gradient Descent



So that rather than needing to run this iterative algorithm, we can instead just solve for the optimal value for θ all at one go, so that in basically one step we get to the optimal value.

Normal equation: Method to solve for θ analytically.

For this explanatory example let's take a very simplified cost function $J(\theta)$, that's just the function of a real number θ. So, for now, we imagine that θ is just a scalar value or that θ is just a row value.

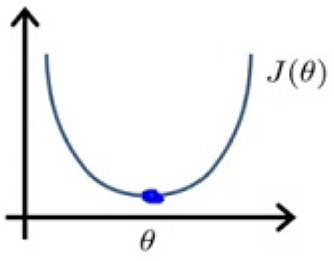
It's just a number, rather than a vector, and that we have a cost function J that's a quadratic function of this real value parameter θ.

Intuition: If 1D $(\theta \in \mathbb{R})$

$\rightarrow$ $J(\theta) = a\theta^2 + b\theta + c$

$$\frac{\partial}{\partial \theta} J(\theta) = \dots \overset{\text{set}}{=} 0$$

Solve for θ



So, how do we minimize a quadratic function? the way to minimize a function is to take derivatives and to set derivatives equal to zero, and this allows us to solve for the value of θ that minimizes $J(\theta)$. That was a simpler case of when data was just real number.

In the problem that we are interested in, θ is no longer just a real number, but, instead, is this $n + 1$ - dimensional parameter vector, and, a cost function J is a function of the vector theta (θ_0 through θ_m).

$$\theta \in \mathbb{R}^{n+1} \qquad J(\theta_0, \theta_1, \dots, \theta_m) = \frac{1}{2m}\sum_{i=1}^{m}(h_\theta(x^{(i)}) - y^{(i)})^2$$

$$\frac{\partial}{\partial \theta_j} J(\theta) = \cdots \overset{\text{set}}{=} 0 \qquad \text{(for every } j\text{)}$$

Solve for $\theta_0, \theta_1, \dots, \theta_n$

How do we minimize the cost function J? One way to do so, is to take the partial derivative of J, with respect to every parameter of θ in turn, and then, to set all of these to 0. If we do that, and we solve for the values of θ_0, θ_1, up to θ_n, then, this would give us that values of θ to minimize the cost function J.

In order to implement the normal equation at big, what we're going to do is take the data set (with 4 training examples) and add an extra column that corresponds to the extra feature, x_0, that is always takes on the value of 1. In this case let's assume that, these four examples is all the data that we have.

	Size (feet²)	Number of bedrooms	Number of floors	Age of home (years)	Price ($1000)
$\downarrow$ x_0	x_1	x_2	x_3	x_4	y
1	2104	5	1	45	460
1	1416	3	2	40	232
1	1534	3	2	30	315
1	852	2	1	36	178

What we're going to do is construct a matrix called X that's a matrix are basically contains all of the features from our training data and we're going to do something similar for y's and call that vector y.

So X is going to be a $m \times (n + 1)$ - dimensional matrix, and y is going to be a m-dimensional vector where m is the number of training examples and n is a number of features $n + 1$, because of this extra feature X_0 that we had.

$$X = \underbrace{\begin{bmatrix} 1 & 2104 & 5 & 1 & 45 \\ 1 & 1416 & 3 & 2 & 40 \\ 1 & 1534 & 3 & 2 & 30 \\ 1 & 852 & 2 & 1 & 36 \end{bmatrix}}_{m \times (n+1)} \qquad y = \underbrace{\begin{bmatrix} 460 \\ 232 \\ 315 \\ 178 \end{bmatrix}}_{m\text{-dimensional vector}}$$

Finally if we take our matrix X and we take our vector y, and if we just compute the following formula:

$\theta = (X^T X)^{-1} X^T y$

$$\theta = \underline{(X^T X)^{-1}} X^T y$$

$(X^T X)^{-1}$ is inverse of matrix $\underline{X^T X}$.

Set $\underline{A} = \underline{X^T}\underline{X}$

$$\boxed{(X^T X)^{-1}} = A^{-1}$$

Octave: `pinv(X'*X)*X'*y`

Video Question: Suppose you have the training in the table below:

age (x_1)	height in cm (x_2)	weight in kg (y)
4	89	16
9	124	28
5	103	20

You would like to predict a child's weight as a function of his age and height with the model

$weight = \theta_0 + \theta_1 age + \theta_2 height.$

What are X and y?

- $X = \begin{bmatrix} 4 & 89 \\ 9 & 124 \\ 5 & 103 \end{bmatrix}, y = \begin{bmatrix} 16 \\ 28 \\ 20 \end{bmatrix}$

- $X = \begin{bmatrix} 1 & 4 & 89 \\ 1 & 9 & 124 \\ 1 & 5 & 103 \end{bmatrix}, y = \begin{bmatrix} 1 & 16 \\ 1 & 28 \\ 1 & 20 \end{bmatrix}$

> $$X = \begin{bmatrix} 4 & 89 & 1 \\ 9 & 124 & 1 \\ 5 & 103 & 1 \end{bmatrix}, y = \begin{bmatrix} 16 \\ 28 \\ 20 \end{bmatrix}$$

- $X = \begin{bmatrix} 1 & 4 & 89 \\ 1 & 9 & 124 \\ 1 & 5 & 103 \end{bmatrix}, y = \begin{bmatrix} 16 \\ 28 \\ 20 \end{bmatrix}$

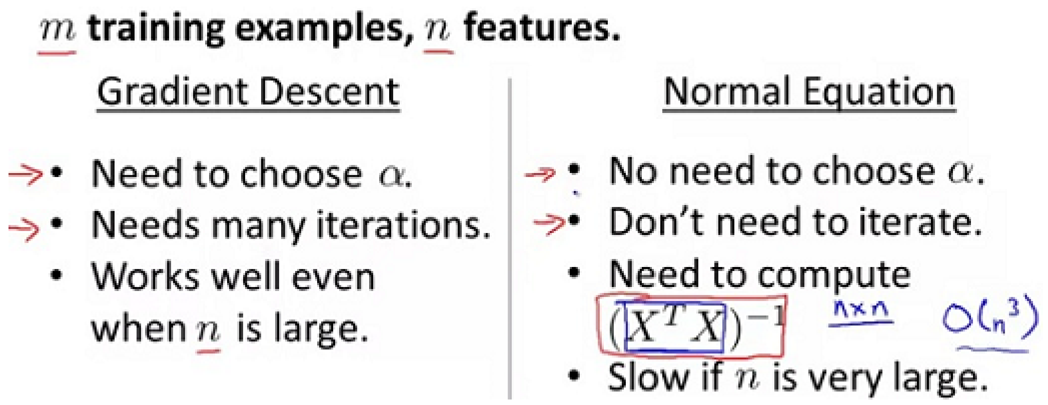
If we are using the normal equation method then feature scaling isn't actually necessary, and it's fine if we have features with:

$0 \leqslant x_1 \leqslant 1$

$0 \leqslant x_2 \leqslant 1000$

$0 \leqslant x_3 \leqslant 10^{-5}$

Advantages and disadvantages:



$\underline{m}$ **training examples,** $\underline{n}$ **features.**

Gradient Descent	Normal Equation
$\rightarrow$ • Need to choose α.	$\rightarrow$ • No need to choose α.
$\rightarrow$ • Needs many iterations.	$\rightarrow$ • Don't need to iterate.
• Works well even when $\underline{n}$ is large.	• Need to compute $\boxed{(\boxed{X^TX})^{-1}}$ $\underline{n \times n}$ $\underline{O(n^3)}$
	• Slow if n is very large.

So long as the number of features is not too large, the normal equation gives us a great alternative method to solve for the parameter θ.

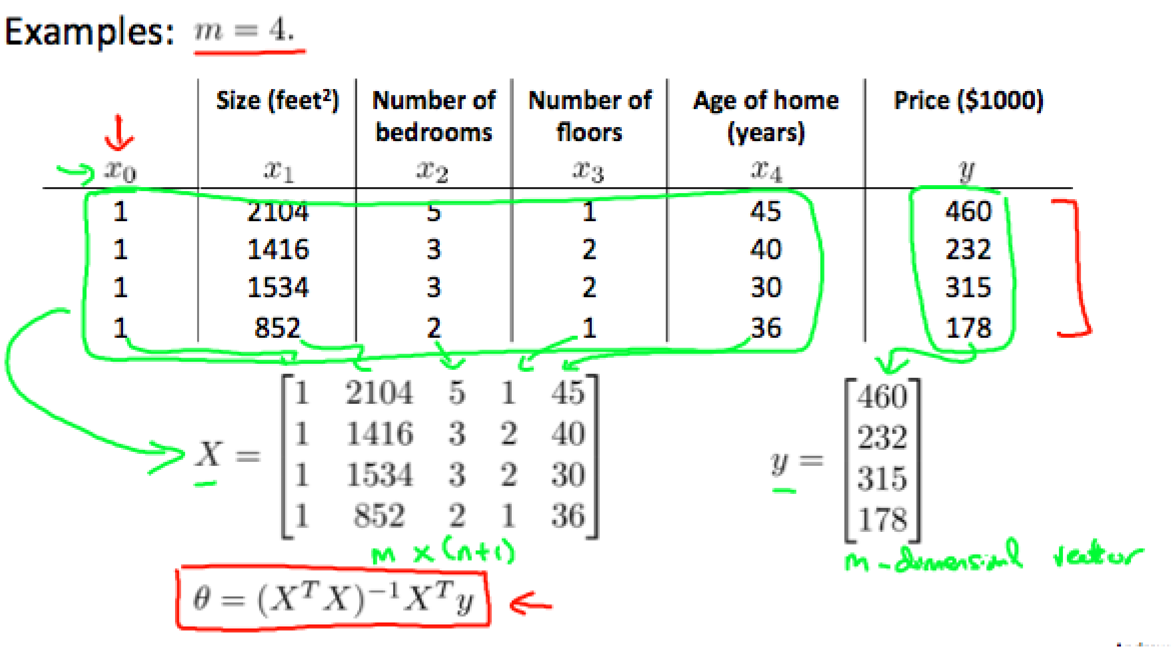
Summary

Gradient descent gives one way of minimizing J. Let's discuss a second way of doing so, this time performing the minimization explicitly and without resorting to an iterative algorithm.

In the "Normal Equation" method, we will minimize J by explicitly taking its derivatives with respect to the θ_j 's, and setting them to zero. This allows us to find the optimum theta without iteration. The normal equation formula is given below:

$\theta = (X^T X)^{-1} X^T y$

There is **no need** to do feature scaling with the normal equation.



Examples: $m = 4.$

x_0	Size (feet²) x_1	Number of bedrooms x_2	Number of floors x_3	Age of home (years) x_4	Price ($1000) y
1	2104	5	1	45	460
1	1416	3	2	40	232
1	1534	3	2	30	315
1	852	2	1	36	178

$$X = \underbrace{\begin{bmatrix} 1 & 2104 & 5 & 1 & 45 \\ 1 & 1416 & 3 & 2 & 40 \\ 1 & 1534 & 3 & 2 & 30 \\ 1 & 852 & 2 & 1 & 36 \end{bmatrix}}_{m \times (n+1)} \qquad y = \underbrace{\begin{bmatrix} 460 \\ 232 \\ 315 \\ 178 \end{bmatrix}}_{m\text{-dimensional vector}}$$

$$\boxed{\theta = (X^T X)^{-1} X^T y}$$

The following is a comparison of gradient descent and the normal equation:

Gradient Descent	Normal Equation
Need to choose alpha	No need to choose alpha
Needs many iterations	No need to iterate
$O(kn^2)$	$O(n^3)$,need to calculate inverse of X^TX
Works well when n is large	Slow if n is very large

With the normal equation, computing the inversion has complexity $O(n^3)$. So if we have a very large number of features, the normal equation will be slow. In practice, when n exceeds 10,000 it might be a good time to go from a normal solution to an iterative process.