Multiple Features Linear Regression

Given 4 features, x_1: size

x_2: number of bedrooms

x_3: number of floors

x_4: age of house

x_j = j-th feature

n = number of features

$X^{(i)}$ = features of i-th training examples

$X_j^{(i)}$ = value of j-th feature in the i-th training example

$$f_{w,b}(X) = WX + b = w_1x_1 + w_2x_2 + w_3x_3 + w_4x_4 + b$$

$$= w_1x_1 + \cdots + w_nx_n + b = \vec{w} \cdot \vec{x} + b$$

dot product

$\vec{w} = [w_1, w_2, \ldots, w_n]$ $1 \times n$

$\vec{x} = [x_1, x_2, \ldots, x_n]$ for m samples, $n \times m$

It's called Multiple Linear Regression (NOT multivariate regression).

Gradient Descent for Multiple Linear Regression with Vectorization

Given $\vec{w} = [w_1, \ldots, w_n]$, $\vec{x} = [x_1, \ldots, x_n]$ and b (still a number)

repeat until convergence { n features ($n \geq 2$)

$$w_1 = w_1 - \alpha \frac{1}{m} \sum_{i=1}^{m} (f_{\vec{w},b}(\vec{x}^{(i)}) - y^{(i)}) x_1^{(i)} \rightarrow \frac{\partial}{\partial w_1} J(\vec{w}, b)$$

$$\vdots$$

$$w_n = w_n - \alpha \frac{1}{m} \sum_{i=1}^{m} (f_{\vec{w},b}(\vec{x}^{(i)}) - y^{(i)}) x_n^{(i)}$$

$$b = b - \alpha \frac{1}{m} \sum_{i=1}^{m} (f_{\vec{w},b}(\vec{x}^{(i)}) - y^{(i)})$$

}

Simultaneously update w_j (for $j = 1, \ldots, n$) and b

Gradient Descent in Practice

Feature Scaling:

Feature and parameter values

$\widehat{price} = w_1 x_1 + w_2 x_2 + b$ (x_1: size, x_2: # bedrooms)

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?

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

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

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

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

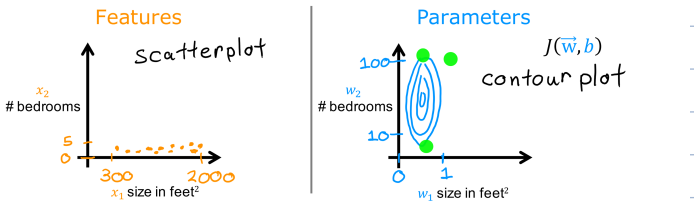
$\widehat{price} = 0.1 * 2000k + 50 * 5 + 50$ (200K, 250K, 50K)

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

Stanford ONLINE DeepLearning.AI Andrew Ng

Feature size and parameter size

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



Stanford ONLINE DeepLearning.AI Andrew Ng

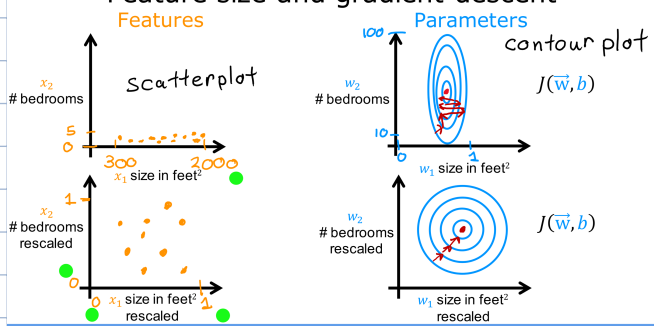
The larger the scale of the feature, the smaller the weight.

The smaller the scale of the feature, the bigger the weight.

This is BAD for gradient descent, because it may end up bouncing back and forth for long time.

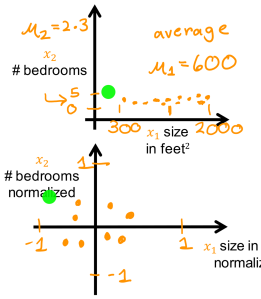
We can rescale the feature. But what should the output range be? 0~+1, -1~+1, -3~+3, etc

Feature size and gradient descent



Stanford ONLINE DeepLearning.AI Andrew Ng

Mean normalization



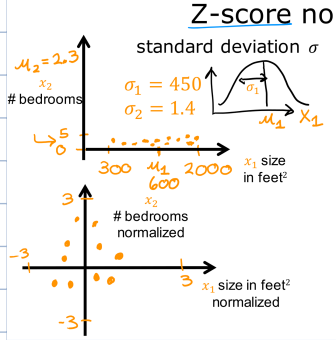
$300 \le x_1 \le 2000$ $\quad 0 \le x_2 \le 5$

$x_1 = \frac{x_1 - \mu_1}{2000-300}$ (max-min) $\quad x_2 = \frac{x_2 - \mu_2}{5-0}$ (max-min)

$-0.18 \le x_1 \le 0.82$ $\quad -0.46 \le x_2 \le 0.54$

Stanford ONLINE DeepLearning.AI Andrew Ng

Z-score normalization

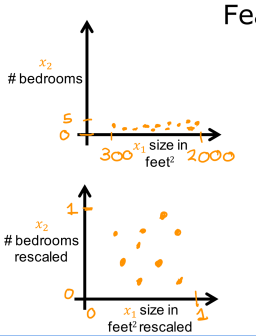


$300 \le x_1 \le 2000$ $\quad 0 \le x_2 \le 5$

$x_1 = \frac{x_1 - \mu_1}{\sigma_1}$ $\quad x_2 = \frac{x_2 - \mu_2}{\sigma_2}$

$-0.67 \le x_1 \le 3.1$ $\quad -1.6 \le x_2 \le 1.9$

Feature scaling



$300 \le x_1 \le 2000$ $\quad 0 \le x_2 \le 5$

$x_{1,scaled} = \frac{x_1}{2000}$ (max) $\quad x_{2,scaled} = \frac{x_2}{5}$ (max)

$0.15 \le x_{1,scaled} \le 1$ $\quad 0 \le x_{2,scaled} \le 1$

Stanford ONLINE DeepLearning.AI Andrew Ng

Here, the output ranges are all different after rescaling. And they're all fine. The rule of thumb is aiming for $-1 \le x_j \le 1$ for each feature x_j. Here is a list of examples.

OK
$-1 \le x_j \le 1$
$-3 \le x_j \le 3$
$-0.3 \le x_j \le 0.3$

okay $0 \le x_1 \le 3$

okay $-2 \le x_2 \le 0.5$

rescale $-100 \le x_3 \le 100$

rescale $-0.001 \le x_4 \le 0.001$

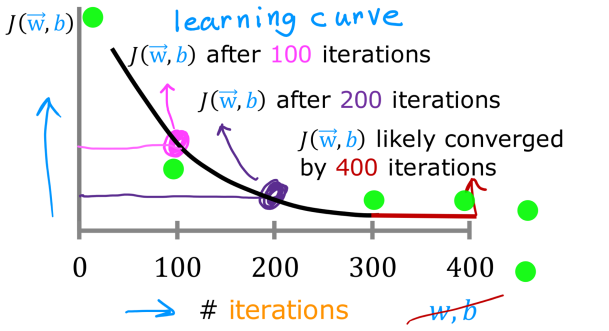
rescale $98 \le x_5 \le 105$

Checking Gradient Descent for Convergence

Make sure gradient descent is working correctly

objective: $\min_{\vec{w},b} J(\vec{w},b)$

$J(\vec{w},b)$ should decrease after every iteration



iterations needed varies 30 1,000 100,000

Automatic convergence test

Let ε "epsilon" be 10^{-3}. 0.001

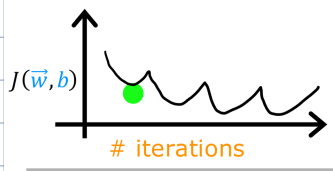
If $J(\vec{w},b)$ decreases by $\leq \varepsilon$ in one iteration, declare convergence.

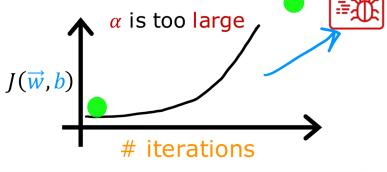
(found parameters $\vec{w},b$ to get close to global minimum)

Stanford ONLINE DeepLearning.AI Andrew Ng

Choosing the Learning Rate α

Identify problem with gradient descent





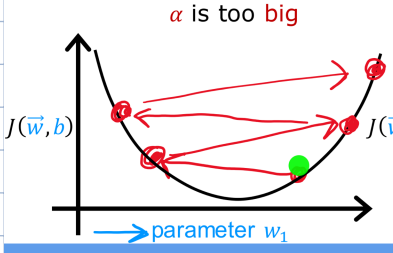
or learning rate is too large

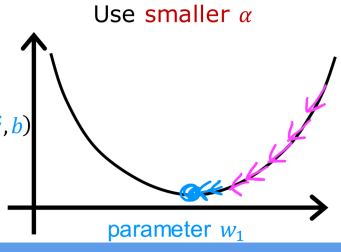
$w_1 = w_1 + \alpha d_1$

use a minus sign

$w_1 = w_1 - \alpha d_1$

Adjust learning rate



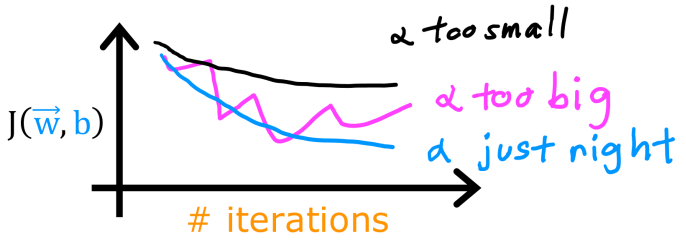


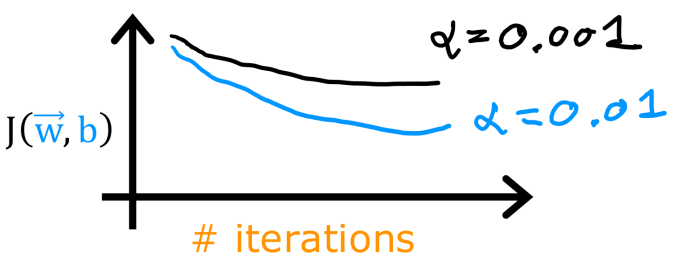
With a small enough α, $J(\vec{w},b)$ should decrease on every iteration

If α is too small, gradient descent takes a lot more iterations to converge

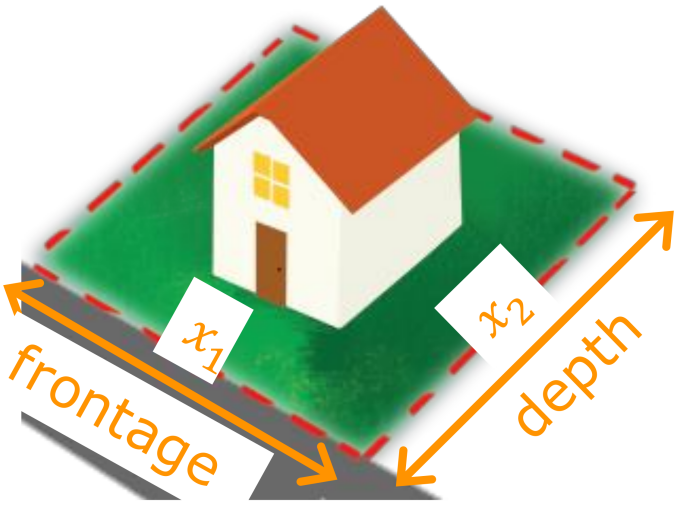
Stanford ONLINE DeepLearning.AI Andrew Ng

Values of α to try: 0.001, 0.003, 0.01, 0.03, 0.1, 0.3, 1





Feature Engineering



Naive model: $f(X) = w_1 x_1 + w_2 x_2 + b$

Better, define a new feature area X_3

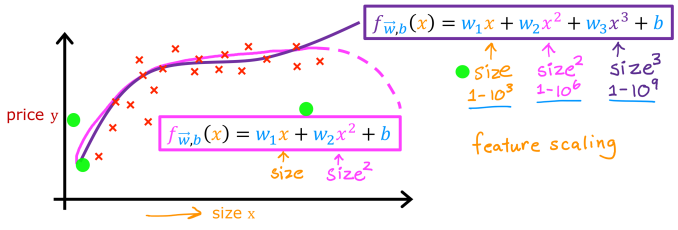
$X_3 = X_1 X_2$

Thus, new model $f(X) = w_1 x_1 + w_2 x_2 + w_3 x_3 + b$

We must use our knowledge, creativity and intuition to design new features, by transforming or combining original features.

Polynomial Regression:

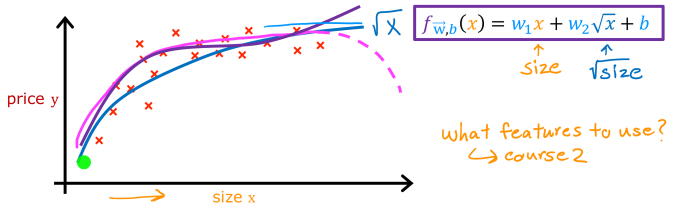
Polynomial regression



Stanford ONLINE DeepLearning.AI Andrew Ng

Extra attention must be put on the scale of designed features X^2 and X^3. They need to be rescaled properly.

Choice of features

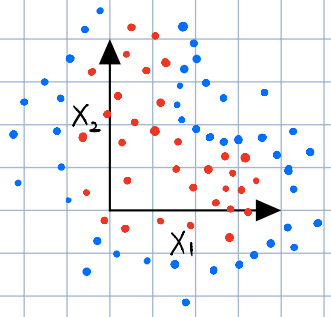


Stanford ONLINE DeepLearning.AI Andrew Ng

How do I know which feature to use? Should it be a straight line, parabolic or a random curve?

One issue with linear regression is that I may have to hand-engineer features.

For instance,



We can define $g(\theta_1 X_1 + \theta_2 X_2 + \theta_3 X_1 X_2 + \theta_4 X_1^2 X_2 + \cdots)$

There are just too many possible features.

Another example is given an $50 \times 50 \times 3$ RGB image.

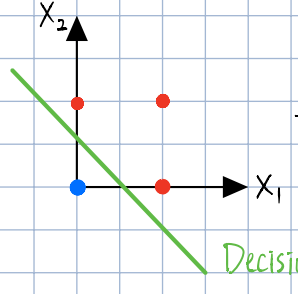
There are 7500 features. I.e. every pixel X_i is a feature.

If creating a new feature $X_i X_j$ from a given pixel X_i and another pixel X_j,

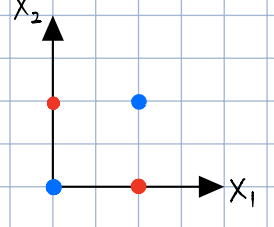
then there will be $\frac{7500^2}{2} \approx 3$ million, which is just too many features.

The solution is using Deep Neural Networks.

Here's another example where neural networks can create features automatically from input.

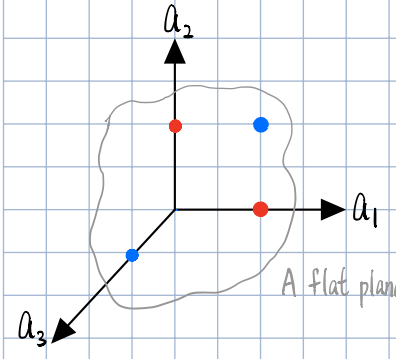


This case can be easily solved with linear regression.



This case cannot be solved with linear regression.

The training data is not linearly seperable. Note: non-linear neuron (sigmoid, tanh) can learn non-linear features, but linear neuron can't.



We can use examples above with two features X_1, X_2 as input to a neural network.

Then it may automatically create features a_1, a_2, a_3 to form a decision boundry.