Lecture 1 - Multivariate Linear Regression

Notation

- $x_j^{(i)}$: value of the feature j in the i^{th} training example
- $x^{(i)}$: the input (features) of the i^{th} training example
- m: number of training examples
- n: number of features

Multivariable form

Consider the following table:

Size (feet)^2	Number of bedrooms	Number of floors	Age of home (years)	Price ($1000)
2104	5	1	45	460
1416	3	2	40	232
1534	3	2	30	315
852	2	1	36	178
...	...	...	...	...

The multivariable form of the hypothesis function is:

$$h_\theta(x) = \theta_0 + \theta_1 x_1 + \theta_2 x_2 + \cdots + \theta_n x_n \tag{1}$$

We can think of θ_0 as the basic price of a house, θ_1 as the price per square meter, θ_2 as the price per floor, etc. x_1 will be the number of square meters in the house, x_2 the number of floors, etc.

> Conveniently, we can represent it in **matrix multiplication** form:
>
> $$h_\theta(x) = \begin{bmatrix} \theta_0 & \theta_1 & \dots & \theta_n \end{bmatrix} \begin{bmatrix} x_0 \\ x_1 \\ \vdots \\ x_n \end{bmatrix} = \theta^T x \tag{2}$$
>
> This is a vectorization of our hypothesis function for **one** training example.

Note: for convenience, we assume $x_0^{(i)} = 1, \; \forall \, i = 1, \dots, m$.

Gradient Descent for Multiple Variables

To sum up...

Hypothesys: $h_\theta(x) = \theta^T x = \theta_0 + \theta_1 x_1 + \theta_2 x_2 + \cdots + \theta_n x_n$
Parameters: $\theta_0,\ \theta_1,\ \ldots,\ \theta_n$
Cost Function: $J(\theta_0,\ \theta_1,\ \ldots,\ \theta_n) = \frac{1}{2m} \sum_{i=1}^{m} \left[(h_\theta(x^{(i)}) - y^{(i)}) \right]^2$
Gradient Descent: Repeat: $\left\{ \theta_j := \theta_j - \alpha \frac{\partial}{\partial \theta_j} J(\theta_0,\ \theta_1,\ \ldots,\ \theta_n) \right\}$ (simultaneously update for every $j = 0,\ 1,\ \ldots,\ n$)

The new algorithm of Gradient Descent for multiple variables is as follows:

Repeat:
{
$\theta_0 := \theta_0 - \alpha \frac{1}{m} \sum_{i=1}^{m} \left[h_\theta(x^{(i)}) - y^{(i)} \right] x_0^{(i)}$
$\theta_1 := \theta_1 - \alpha \frac{1}{m} \sum_{i=1}^{m} \left[h_\theta(x^{(i)}) - y^{(i)} \right] x_1^{(i)}$
$\theta_2 := \theta_2 - \alpha \frac{1}{m} \sum_{i=1}^{m} \left[h_\theta(x^{(i)}) - y^{(i)} \right] x_2^{(i)}$
$\ldots$
}

Gradient Descent in Practice I: Feature Scaling

Idea: make sure variables are on a similar scale.

We can speed up gradient descent by having each of our input values in roughly the same range. This is because θ will descend quickly on small ranges and slowly on large ranges, and so will oscillate inefficiently down to the optimum when the variables are very uneven.

Two techniques to help with this are **feature scaling** and **mean normalization**. Feature scaling involves dividing the input values by the range (i.e. the maximum value minus the minimum value) of the input variable, resulting in a new range of just 1. Mean normalization involves subtracting the average value for an input variable from the values for that input variable resulting in a new average value for the input variable of just zero.

Feature Scaling: what we want to do is to get every feature into approximately a $-1 \leq x \leq 1$ range.

Mean Normalization: get $-1 \leq x - \mu \leq 1$, where μ is the mean average of the feature in question.

To implement both of these techniques, just do

$$\frac{x_j - \mu_j}{range(x_j)} \text{ or } \frac{x_j - \mu_j}{s_j} \tag{3}$$

Gradient Descent in Practice II: Learning Rate

Debugging: Make a plot with number of iterations on the x-axis. Now plot the cost function, $J(\theta)$ over the number of iterations of gradient descent. If $J(\theta)$ ever increases, then you probably need to decrease the learning rate α.

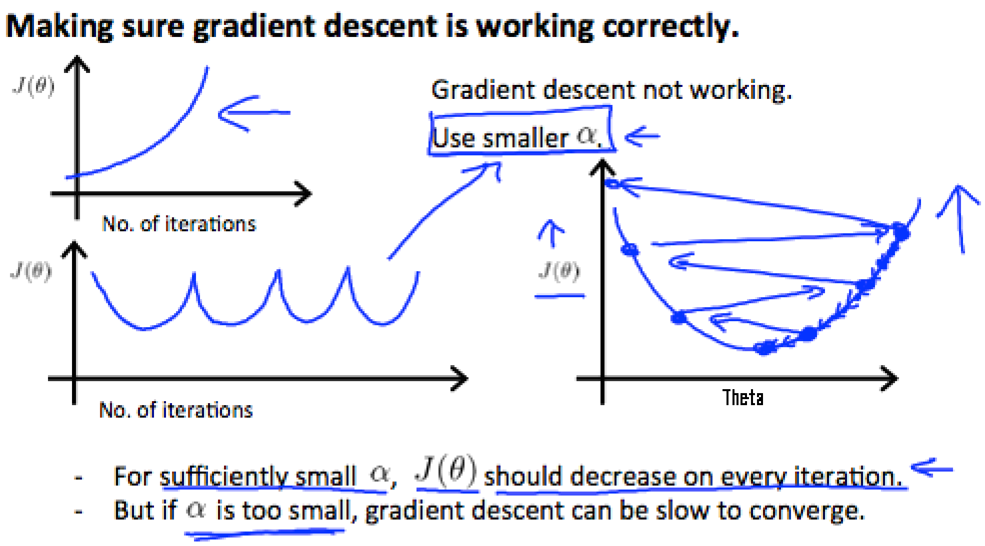
Automatic convergence test: Declare convergence if $J(\theta)$ decreases by less than ϵ in one iteration, where ϵ is some small value such as 10^{-3}. However in practice it's difficult to choose this threshold value.

It has been proven that if learning rate α is sufficiently small, then $J(\theta)$ will decrease on every iteration.

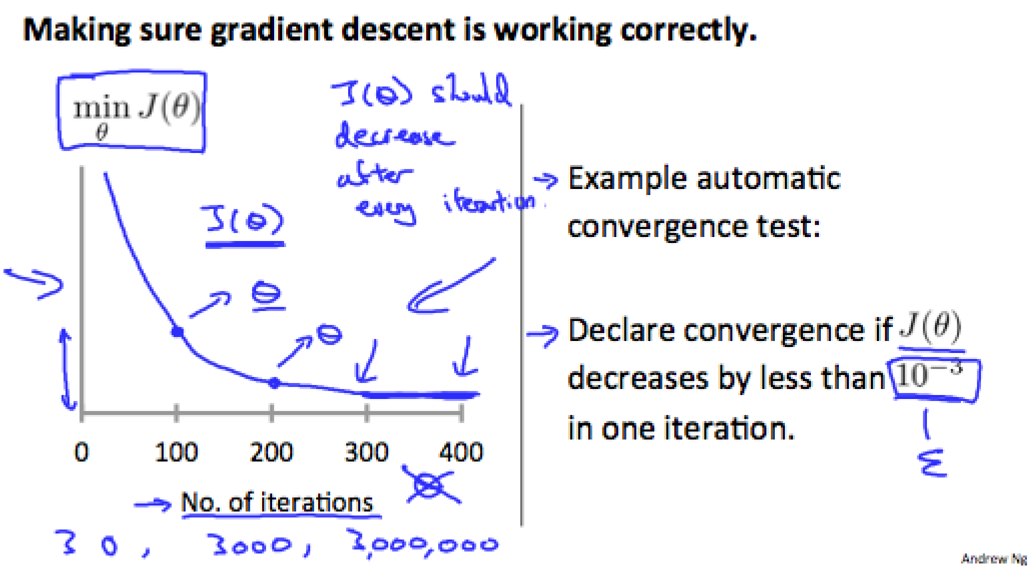
To summarize...

If α is too small: slow convergence. If α is too large: $J(\theta)$ may not decrease on every iteration and thus may not converge.

Example: in all cases below, we need to decrease α



Example: Good convergence case



Features and Polynomial Regression

Note: We can combine multiple features into one. For example, we can combine x_1 and x_2 into a new feature x_3 by taking $x_3 = x_1 \cdot x_2$.

Our hypothesis function need not be linear (a straight line) if that does not fit the data well.

We can change the behavior or curve of our hypothesis function by making it a quadratic, cubic or square root function (or any other form).

Examples:

- cubic function: $h_\theta(x) = \theta_0 + \theta_1 x_1 + \theta_2 x_1^2 + \theta_3 x_1^3$
- square root function: $h_\theta(x) = \theta_0 + \theta_1 x_1 + \theta_2 \sqrt{x_1}$

Important! If x_1 has range 1 - 1000, then range of x_1^2 becomes 1 - 1000000 and that of x_1^3 becomes 1 - 1000000000. That's why it's important to apply feature scaling.

References

[1] Machine Learning - Stanford University (https://www.coursera.org/learn/machine-learning)