Cost Function: $h(\theta) = \theta_0 + \theta_1 x$ minimize$(\theta_0, \theta_1) \frac{1}{2m} \sum_{i=1}^{m} (h_\theta(x^i) - y^i)^2$. Minimize sum of squared differences.

$J(\theta_0, \theta_1) = \frac{1}{2m} \sum_{i=1}^{m} (h_\theta(x^i) - y^k)^2$ minimize $J(\theta_0, \theta_1)$

Simplified: $h_\theta(x) = \theta_1 x$, set $\theta_0 = 0$

Assignment: $a := b, a := a + 1$

Truth assertion: $a = b$. $a = a + 1$ is wrong.

$$\theta_j := \theta_j - \alpha \frac{\partial}{\partial \theta_j} J(\theta_0, \theta_1)$$

where α is the learning rate. Simultaneously update θ_0, θ_1.

Correct: Simultaneous update.

$$\begin{aligned} temp0 &:= \theta_0 - \alpha \frac{\partial}{\partial \theta_0} J(\theta_0, \theta_1) \\ temp1 &:= \theta_1 - \alpha \frac{\partial}{\partial \theta_1} J(\theta_0, \theta_1) \\ \theta_0 &:= temp0 \\ \theta_1 &:= temp1 \end{aligned}$$

Incorrect:

$$\begin{aligned} temp0 &:= \theta_0 - \alpha \frac{\partial}{\partial \theta_0} J(\theta_0, \theta_1) \\ \theta_0 &:= temp0 \\ temp1 &:= \theta_1 - \alpha \frac{\partial}{\partial \theta_1} J(\theta_0, \theta_1) \\ \theta_1 &:= temp1 \end{aligned}$$

Gradient Descent. Previously,

$$\begin{aligned} \theta_0 &= \theta_0 - \alpha \frac{1}{m} \sum_{i=1}^{m} (h_\theta(x^{(i)}) - y^{(i)}) \\ \theta_1 &= \theta_1 - \alpha \frac{1}{m} \sum_{i=1}^{m} (h_\theta(x^{(i)}) - y^{(i)}) x^{(i)} \end{aligned}$$

Multiple Features. $x_j^{(i)}$: value of feature j in i^{th} training example. $x^{(i)}$ is a n dimension vector.

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

For convenience, think of $x_0 = 1$. Then,

$$x = \begin{bmatrix} x_0 \\ x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix} \in \mathbb{R}^{n+1} \qquad \theta = \begin{bmatrix} \theta_0 \\ \theta_1 \\ \theta_2 \\ \vdots \\ \theta_n \end{bmatrix} \in \mathbb{R}^{n+1}$$

$h_\theta(x) = \theta_0 x_0 + \theta_1 x_1 + \cdots + \theta_n x_n = \theta^T x$

Gradient Descent for Multivariate. New algorithm.

$$\theta_j = \theta_j - \alpha \frac{1}{m} \sum_{i=1}^{m} (h_\theta(x^{(i)}) - y^{(i)}) x_j^i$$

Feature Scaling. Make sure features are on a similar scale. Get every feature into approximately $-1 \leq x_i \leq 1$.

Mean normalization. Replace x_i with $x_i = \mu_i$ to take features have approximately zero measure. $-0.5 \leq x_1 \leq 0.5, -0.5 \leq x_2 \leq 0.5$. $x := \frac{x_1 - \mu_1}{s_1}$, where s_1 is range (max - min) or standard deviation.

Learning Rate. Plot min $J(\theta)$ vs. number of iteration. Get concave up graph and level off when converged. $J(\theta)$ should decrease after every iteration.

Automatic convergence test. Declare convergence if $J(\theta)$ decreases by less than 10^{-3} per iteration. If gradient descent graph is increase, i.e. not working, use smaller α. If learning descent is shooting up, use smaller α because overshoots the minimum. If graph oscillates, use small α.

If α is too small, gradient descent is slow.

Polynomial Regression. Modeling housing price, $\theta_0 + \theta_1 x + \theta_2 x^2 + \theta_3 x^3$, where x_1 is the size, x_2 is size squared. Apply feature scaling.

Another reasonable choice $h_\theta(x) = \theta_0 + \theta_1(size) + \theta_2\sqrt{size}$.

Normal Equation. Method to solve for θ analytically.

Intuition: If 1D, $J(\theta) = a\theta^2 + b\theta + c$. $\frac{d}{d\theta} J(\theta)$. Multi: $\frac{\partial}{\partial \theta_j} J(\theta) \cdots = 0$

Example: $m = 4$, has 4 training examples. X is a $m \times (n+1)$ matrix where column 1 is all ones, and the rest are coefficients for each feature. y is m-dimensional vector, m = number of training examples.

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

Let

$$x^{(i)} = \begin{bmatrix} x_0^{(i)} \\ x_1^{(i)} \\ x_2^{(i)} \\ \vdots \\ x_n^{(i)} \end{bmatrix} \in \mathbb{R}^{n+1}$$

Take X, there design matrix to be $m \times (n+1)$

$$\begin{bmatrix} (x^{(1)})^T \\ (x^{(2)})^T \\ \vdots \\ (x^{(m)})^T \end{bmatrix}$$

Gradient descent: need to choose α, need many iterations. Works well even when n, the feature is large.
Normal equation: no need to choose α, no need to iterate. Need to compute $(X^T X)^{-1}$, slow if n is large. Use gradient descent, if $n \geq 10,000$.