Machine Learning Notes

Coursera

Table of Contents

Week 1

First Module

What is Machine Learning?

Arthur Samuel : "the field of study that gives computers the ability to learn without being explicitly programmed."

Tom Mitchell provides a more modern definition: "A computer program is said to learn from experience E with respect to some class of tasks T and performance measure P, if its performance at tasks in T, as measured by P, improves with experience E."

Example: playing checkers.

E = the experience of playing many games of checkers

T = the task of playing checkers.

P = the probability that the program will win the next game.

In general, any machine learning problem can be assigned to one of two broad classifications:

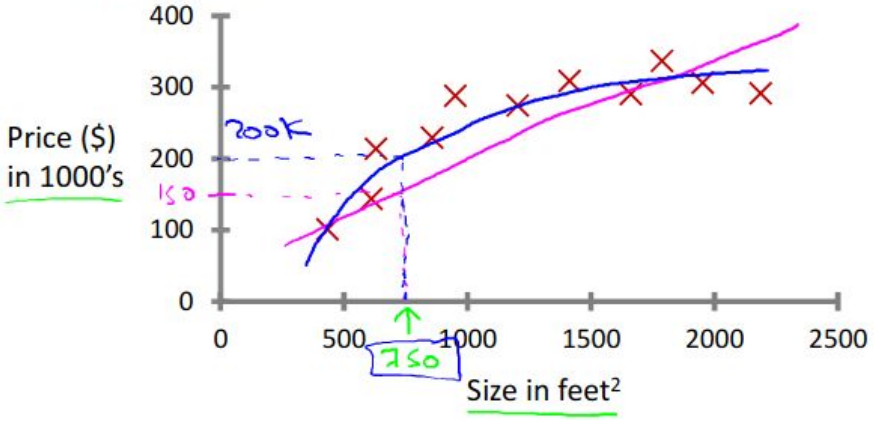
Supervised learning and Unsupervised learning.

Supervised Learning

In supervised learning, we are given a data set and already know what our correct output should look like, having the idea that there is a relationship between the input and the output.

Supervised learning problems are categorized into "regression" and "classification" problems. In a regression problem, we are trying to predict results within a continuous output. In a classification problem, we are instead trying to predict results in a discrete output.

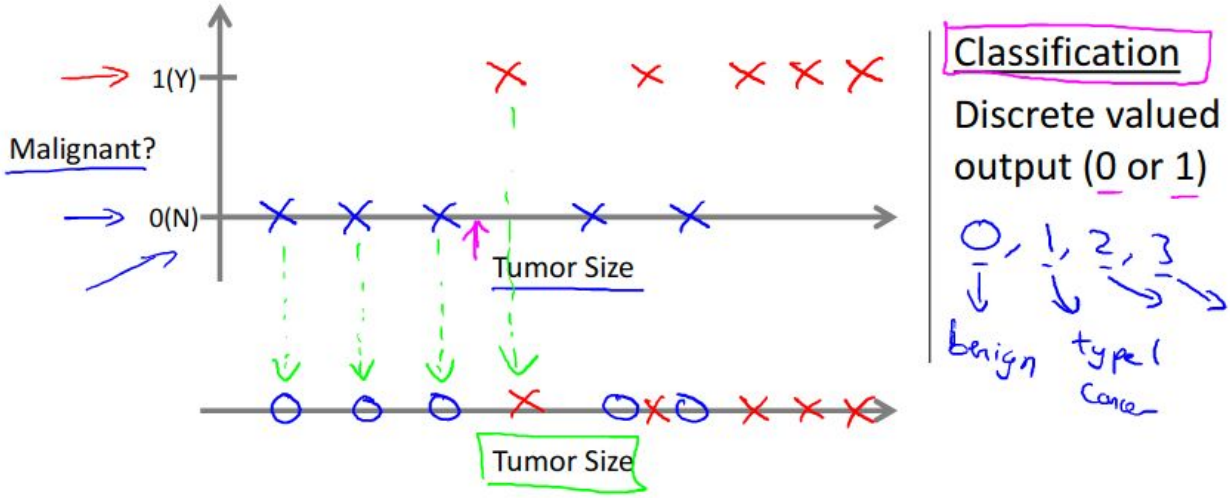
Housing price prediction.



Supervised Learning
"right answers" given

Regression: Predict continuous valued output (price)

Breast cancer (malignant, benign)



Example:

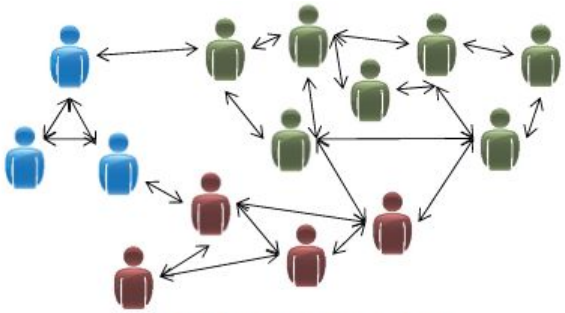
(a) Regression - Given a picture of a person, we have to predict their age on the basis of the given picture

(b) Classification - Given a patient with a tumor, we have to predict whether the tumor is malignant or benign.

Unsupervised Learning

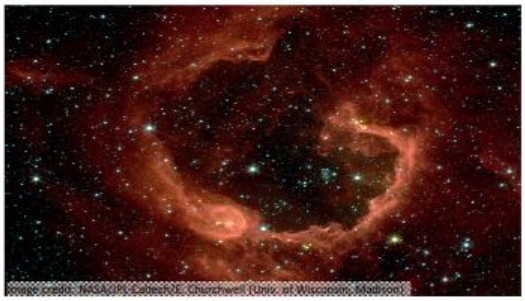
Unsupervised learning allows us to approach problems with little or no idea what our results should look like. We can derive structure from data where we don't necessarily know the effect of the variables.

Organize computing clusters



Social network analysis

Market segmentation



Astronomical data analysis

Example:

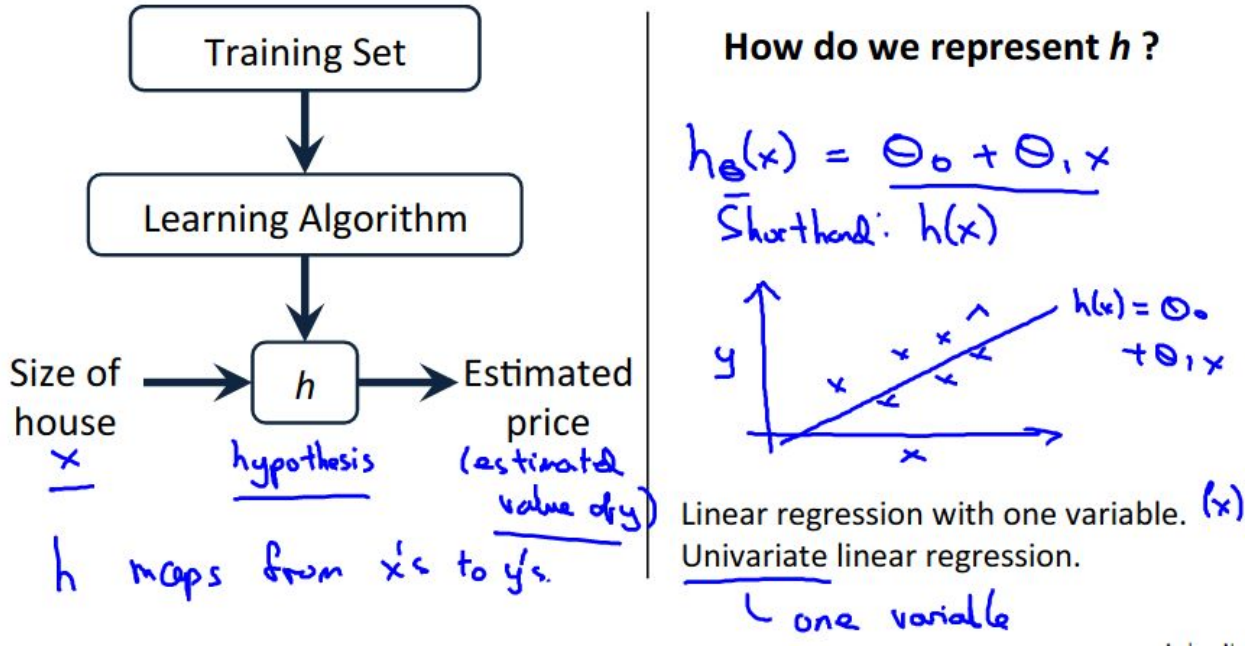
Clustering: Take a collection of 1,000,000 different genes, and find a way to automatically group these genes into groups that are somehow similar or related by different variables, such as lifespan, location, roles, and so on.

Non-clustering: The "Cocktail Party Algorithm", allows you to find structure in a chaotic environment. (i.e. identifying individual voices and music from a mesh of sounds at a cocktail party).

Second Module

Model Representation

$x^{(i)}$ denotes the "input" variables and $y^{(i)}$ to denote the "output" or target variable that we are trying to predict. A pair $(x^{(i)},y^{(i)})$ is called a training example, and the dataset that we'll be using to learn—a list of m training examples $(x^{(i)},y^{(i)});i=1,...,m$—is called a training set. We will use X to denote the space of input values, and Y to denote the space of output values.

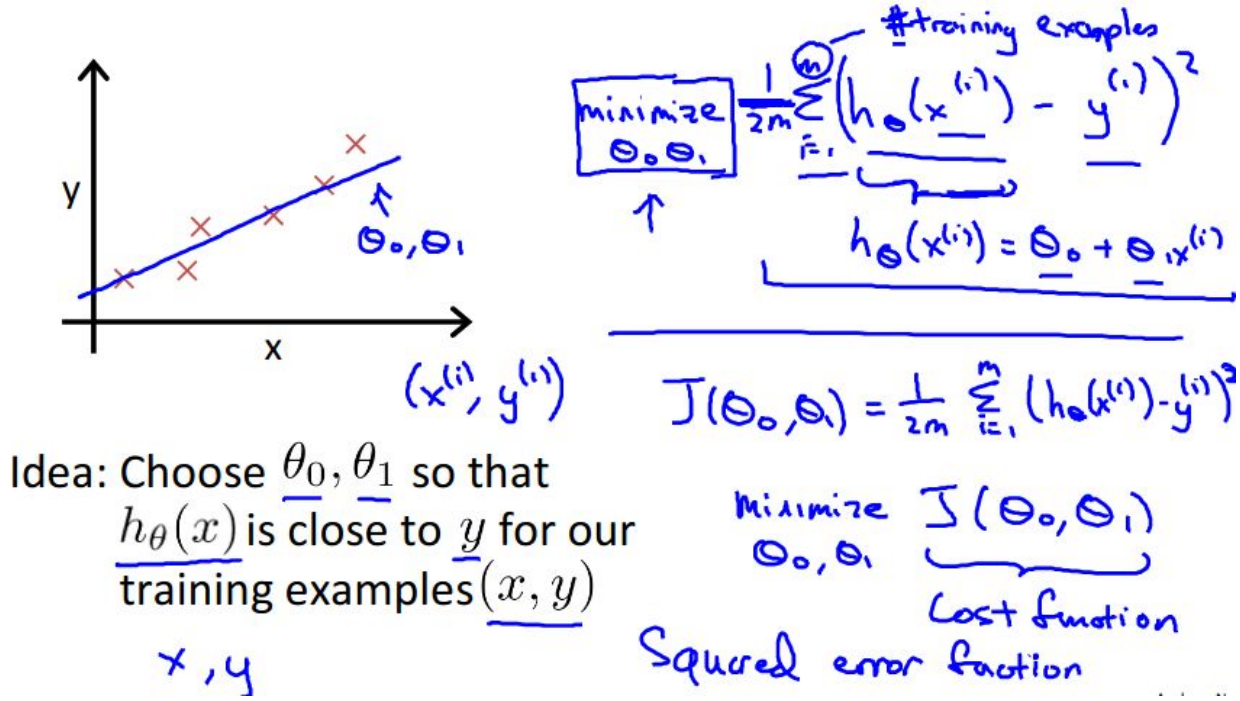


Cost Function

We can measure the accuracy of our hypothesis function by using a cost function. This takes an average difference of all the results of the hypothesis with inputs from x's and the actual output y's.

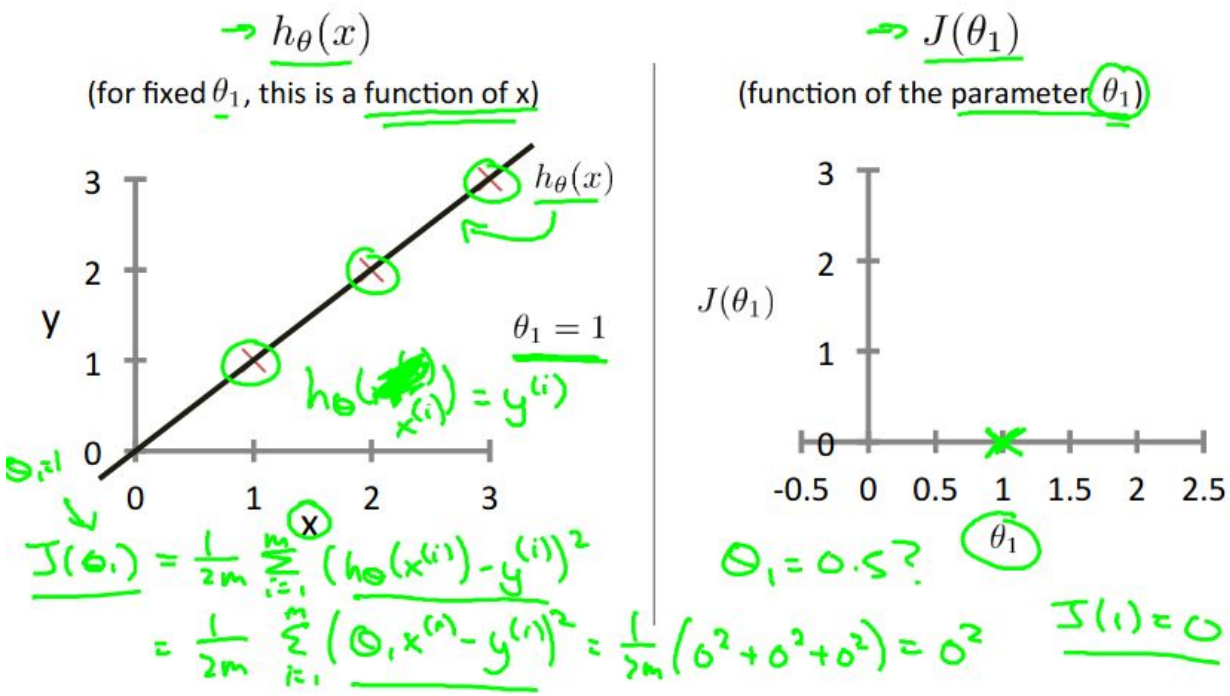
$$J(\theta_0, \theta_1) = \frac{1}{2m} \sum_{i=1}^{m} (\hat{y}_i - y_i)^2 = \frac{1}{2m} \sum_{i=1}^{m} (h_\theta(x_i) - y_i)^2$$

This function is otherwise called the "Squared error function", or "Mean squared error". The mean is halved (½) as a convenience for the computation of the gradient descent, as the derivative term of the square function will cancel out the (½) term.



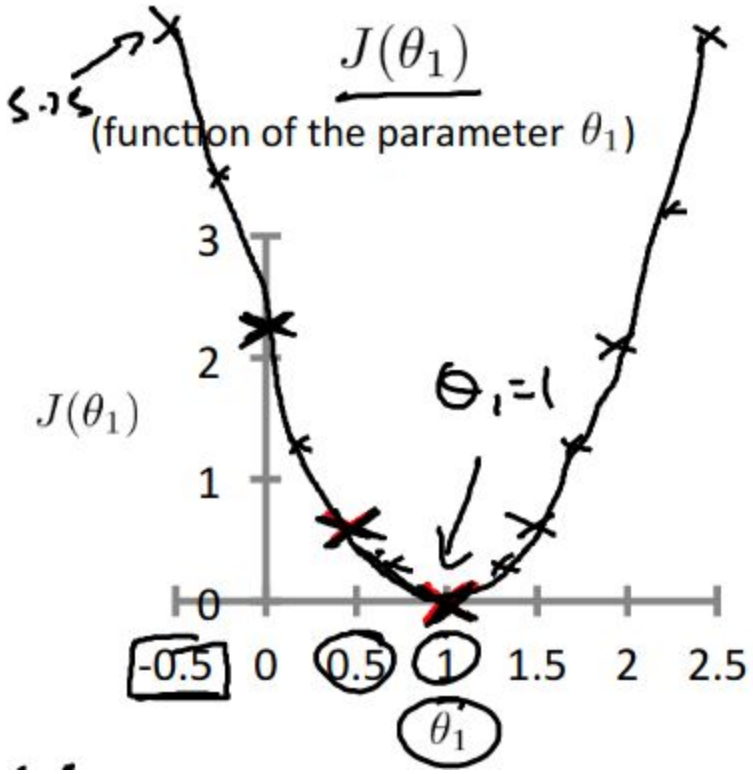
Cost Function - Intuition 1

Our objective is to get the best possible line. The best possible line will be such that the average squared vertical distances of the scattered points from the line will be the least. The following example shows the ideal situation where we have a cost function of 0.



When θ_1 = 1, we get a slope of 1 which goes through every single data point in our model. Conversely, when θ_1=0.5, we see the vertical distance from our fit to the data points increase.

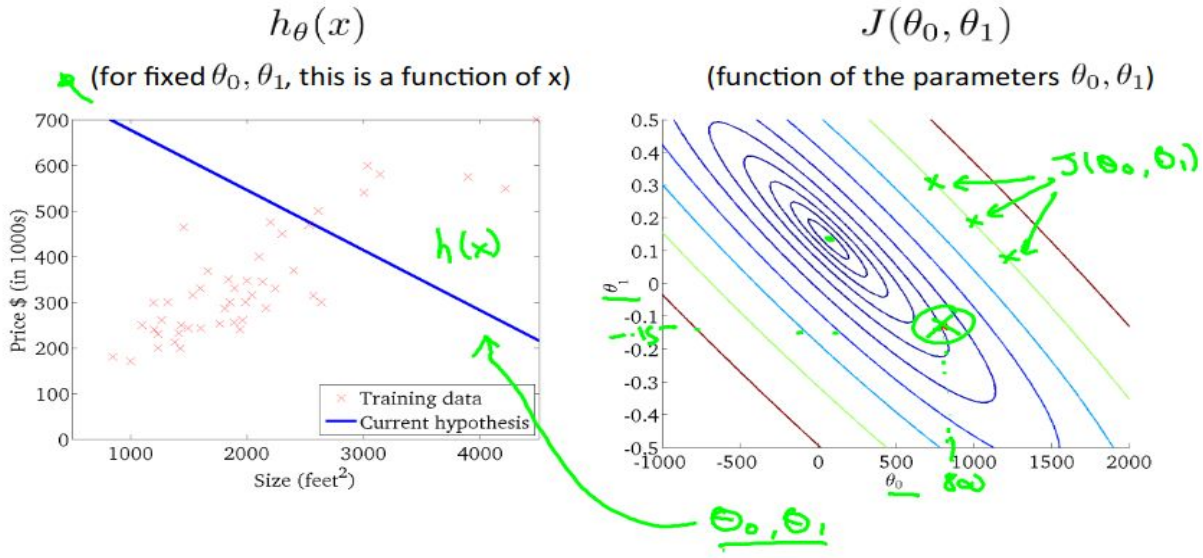
Plotting several other points yields to the following graph:



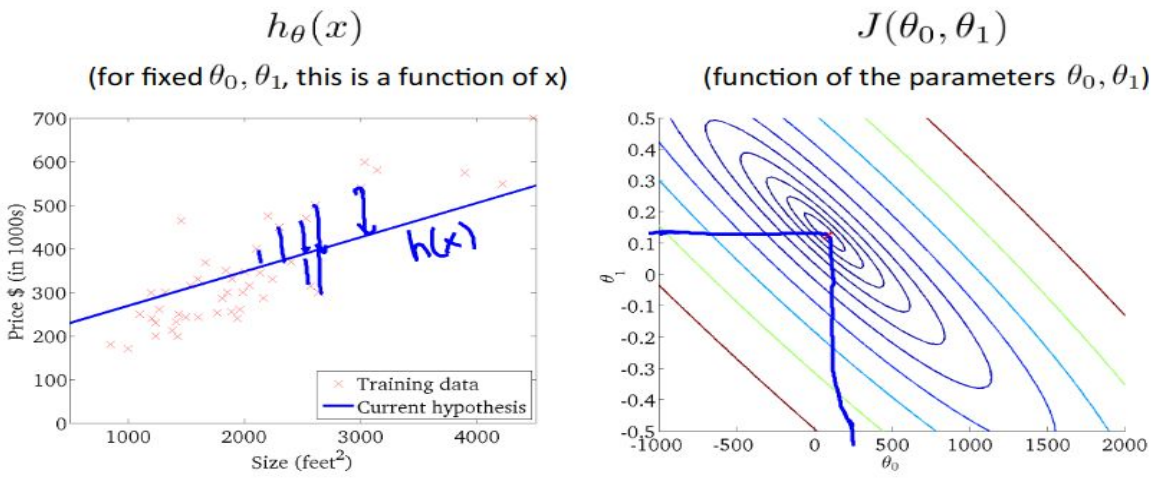
Thus as a goal, we should try to minimize the cost function. In this case, θ_1=1 is our global minimum.

Cost Function - Intuition 2

A contour plot is a graph that contains many contour lines. A contour line of a two variable function has a constant value at all points of the same line. An example of such a graph is the one to the right below.



The circled x displays the value of the cost function for the graph on the left when θ_0= 800 and θ_1= -0.15. When θ_0 = 360 and θ_1 = 0, the value of $J(\theta_0,\theta_1)$ in the contour plot gets closer to the center thus reducing the cost function error.

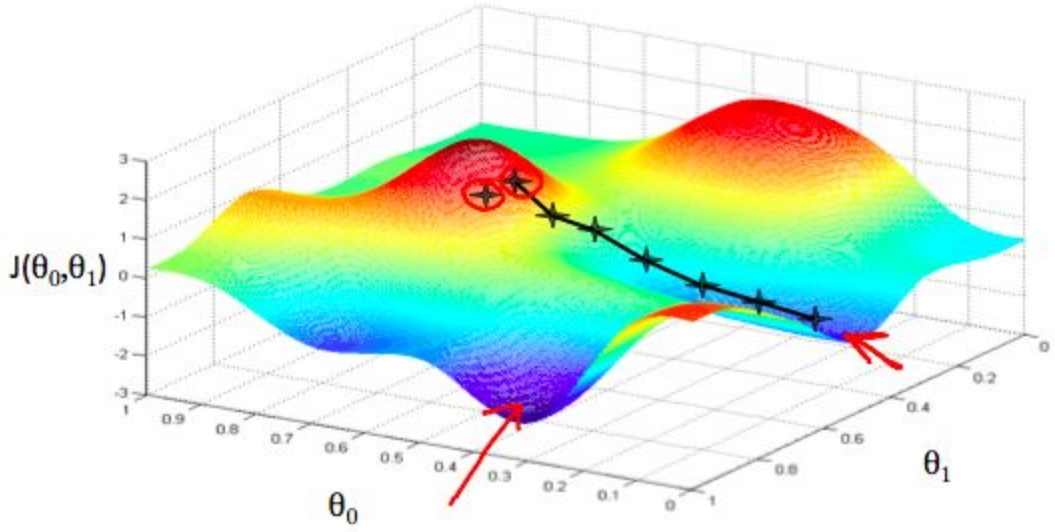


The graph above minimizes the cost function as much as possible and consequently, the result of θ_1 and θ_0 tend to be around 0.12 and 250 respectively. Plotting those values on our graph to the right seems to put our point in the center of the inner most 'circle'.

Gradient Descent

Gradient descent algorithm is an iterative process that takes us to the minimum of a function.

We put θ_0 on the x axis and θ_1 on the y axis, with the cost function on the vertical z axis. The points on our graph will be the result of the cost function using our hypothesis with those specific theta parameters. The graph below depicts such a setup.



We will know that we have succeeded when our cost function is at the very bottom of the pits in our graph, i.e. when its value is the minimum. The red arrows show the minimum points in the graph.

The gradient descent algorithm is:

repeat until convergence:

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

where

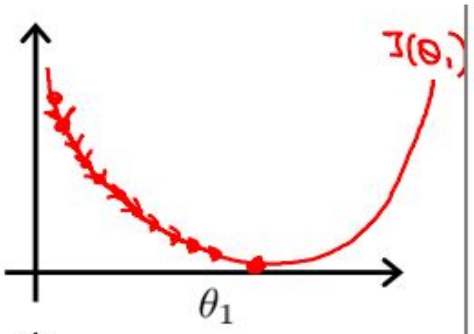
j=0,1 represents the feature index number.

At each iteration j, one should simultaneously update the parameters $\theta_1,\theta_2,...,\theta_n$. Updating a specific parameter prior to calculating another one on the $j^{(th)}$ iteration would yield to a wrong implementation.

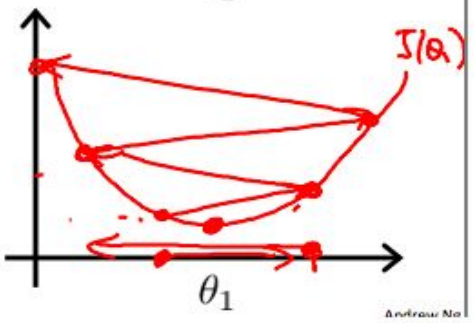
we should adjust our parameter α to ensure that the gradient descent algorithm converges in a reasonable time. Failure to converge or too much time to obtain the minimum value imply that our step size is wrong.

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

If α is too small, gradient descent can be slow.



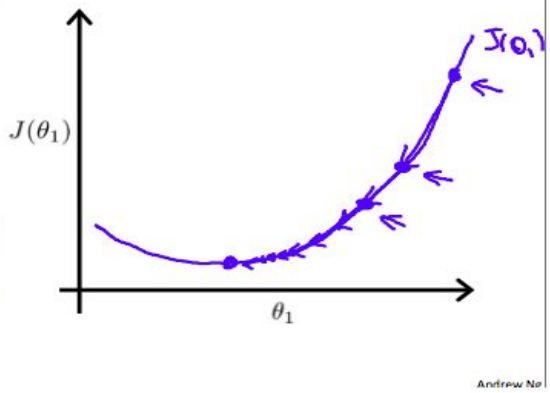
If α is too large, gradient descent can overshoot the minimum. It may fail to converge, or even diverge.



Gradient descent can converge to a local minimum, even with the learning rate α fixed.

$$\theta_1 := \theta_1 - \alpha \frac{d}{d\theta_1} J(\theta_1)$$

As we approach a local minimum, gradient descent will automatically take smaller steps. So, no need to decrease α over time.



Gradient descent for linear regression:

We can substitute our actual cost function and our actual hypothesis function and modify the equation to :

repeat until convergence: {

$$\theta_0 := \theta_0 - \alpha \frac{1}{m} \sum_{i=1}^{m} (h_\theta(x_i) - y_i)$$

$$\theta_1 := \theta_1 - \alpha \frac{1}{m} \sum_{i=1}^{m} (h_\theta(x_i) - y_i) x_i$$

}

where m is the size of the training set, θ_0 a constant that will be changing simultaneously with θ_1 and x_i,y_i are values of the given training set (data).

The following is a derivation $\frac{\partial}{\partial \theta_j} J(\theta)$ of for a single example :

$$\begin{aligned}
\frac{\partial}{\partial \theta_j} J(\theta) &= \frac{\partial}{\partial \theta_j} \frac{1}{2} (h_\theta(x) - y)^2 \\
&= 2 \cdot \frac{1}{2} (h_\theta(x) - y) \cdot \frac{\partial}{\partial \theta_j} (h_\theta(x) - y) \\
&= (h_\theta(x) - y) \cdot \frac{\partial}{\partial \theta_j} \left(\sum_{i=0}^{n} \theta_i x_i - y \right) \\
&= (h_\theta(x) - y) \, x_j
\end{aligned}$$

The point of all this is that if we start with a guess for our hypothesis and then repeatedly apply these gradient descent equations, our hypothesis will become more and more accurate.

This is simply gradient descent on the original cost function J. This method looks at every example in the entire training set on every step, and is called **batch gradient descent**.

Important Link and References:

- Understanding the Mathematics behind Gradient Descent.
- Gradient Descent — ML Glossary documentation
- An overview of gradient descent optimization algorithms

Week 2

Multivariate linear regression

Notation for equations where we can have any number of input variables:

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

$x^{(i)}$ = the input (features) of the i^{th} training example

m = the number of training examples

n = the number of features

Gradient descent for multiple variables

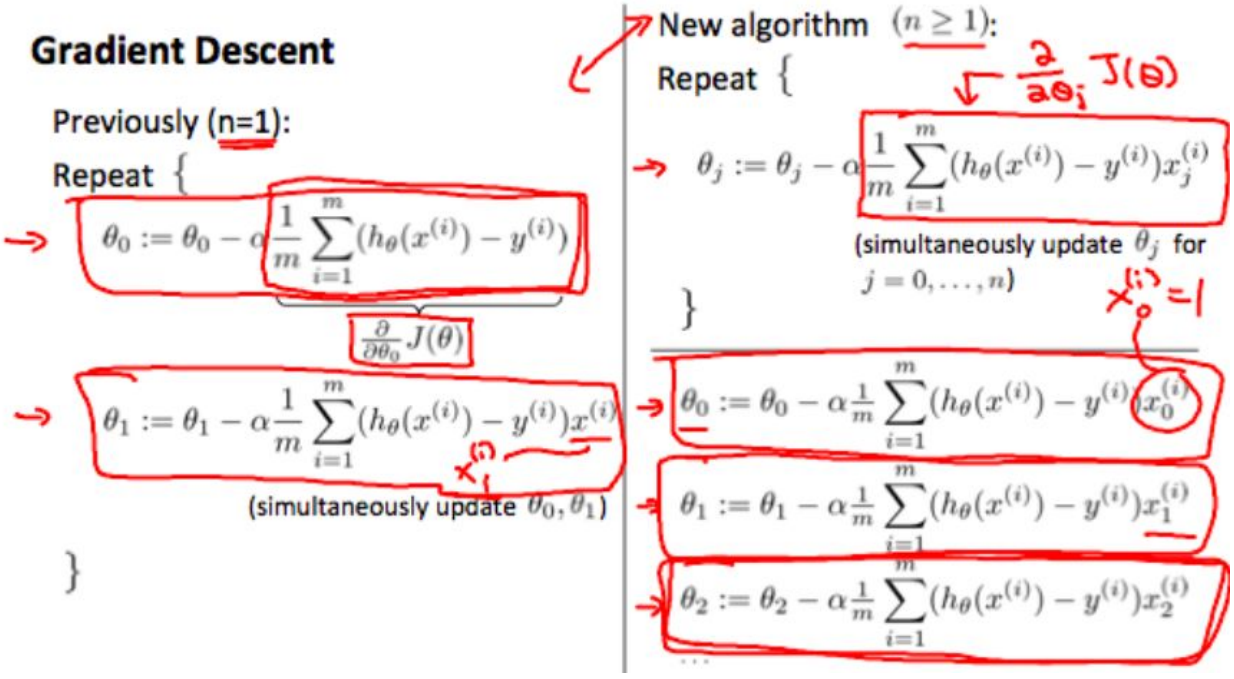
The gradient descent equation itself is generally the same form; we just have to repeat it for our 'n' features:

repeat until convergence: {

$$\theta j := \theta j - \alpha \frac{1}{m} \sum_{i=1}^{m} (h\theta(x^{(i)}) - y^{(i)}) \cdot x_j^{(i)} \qquad \text{for j := 0...n}$$

}

The following image compares gradient descent with one variable to gradient descent with multiple variables:



Before doing gradient descent

Feature Scaling & Mean Normalization

We can speed up gradient descent by having each of our input values in roughly the same range.
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. To implement both of these techniques, adjust your input values as shown in this formula:

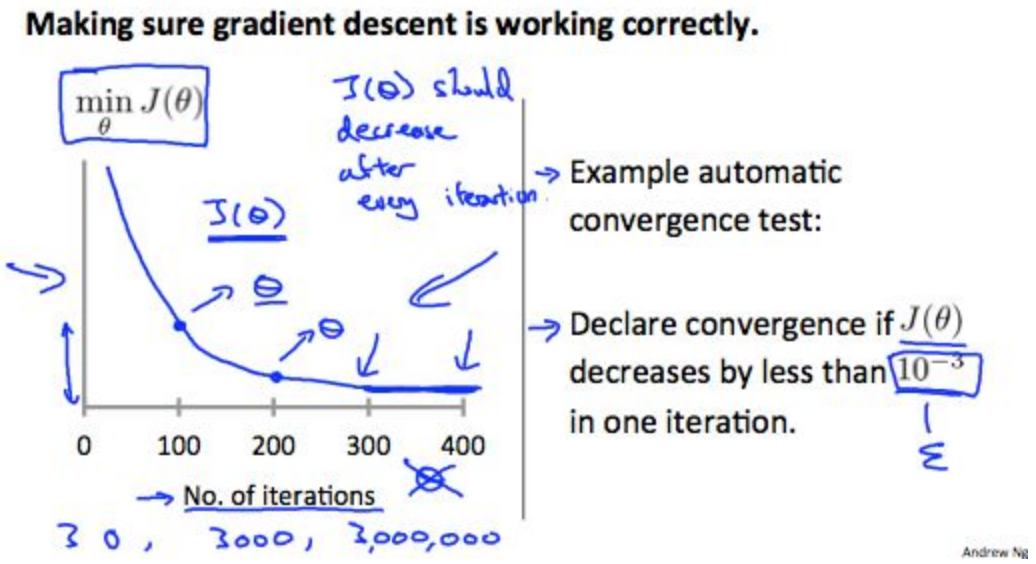
$$x_i := \frac{x_i - \mu_i}{s_i}$$

Where μ_i is the **average** of all the values for feature (i) and s_i is the range of values (max - min), or s_i is the standard deviation.

Learning Rate

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

Automatic convergence test. Declare convergence if J(θ) decreases by less than E in one iteration, where E is some small value such as 10^{-3} .



If α is too small: slow convergence.

If α is too large: J(θ) may not decrease on every iteration and thus may not converge.

Features and Polynomial Regression

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 $x_1 \cdot x_2$.
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).

Normal Equation

Normal equation 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.

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 $X^T X$
Works well when n is large	Slow if n is very large