Mathematical Approach of Machine Learning Algorithm I

– A Short Introduction –

Julien Lin

Department of Bioengineering | Imperial College London | 2015-2016

Part I

Introduction to

Supervised Machine Learning

Part I

Chapter 1

Regression

Table of content

Regression

Linear Regression is an approach to model a linear relationship between a scalar dependent **continuous** output variable ***y*** given one or more independent input variable ***x*.**

I. Univariate Linear Regression

Univariate Linear Regression refers to a Linear Regression with only one variable ***x*** and is used to:

- Fit the best predictive model to forecast/predict accurately a single and continuous known output values $y^{(i)}$ given a single known input $x^{(i)}$.
- Quantify the strength of the relationship between a single known output values ***y*** given a single known input ***x***.
-

1) *Starting Example: "Predicting Housing Prices"*

Given a training set of house prices with **m** *(total number of training examples)*, **x** (*input variable/features*), and **y** *(output/ target variable)*:

Index of training examples	Size in feet^2 (x)	Price in $ (y)
i=1	2014	460
i=2	1490	244
i=3	900	190
i=m	…	…

This dataset will be used to define the best predictive model to predict with the highest accuracy possible the prices of the houses (y) according to the size (x)

For each single known value of size $x^{(i)}$ correspond an actual value of the housing prices $y^{(i)}$ (**Figure 1**)

Linear Regression problem: given a the dataset represented by the value of each known $(x^{(i)},y^{(i)})$ pair, what will be the house price according to a specific size (e.g. 1200 feet^2) that has not been collected in the dataset yet?

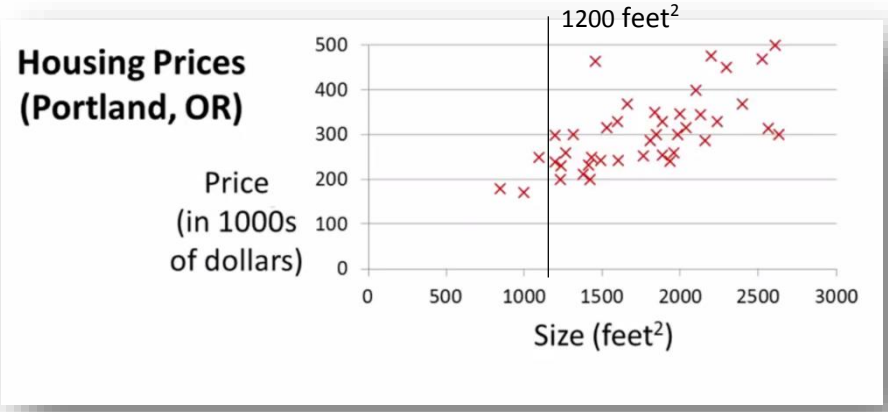


Figure 1: Graphic representation of housing prices in Portland according to the feet^2

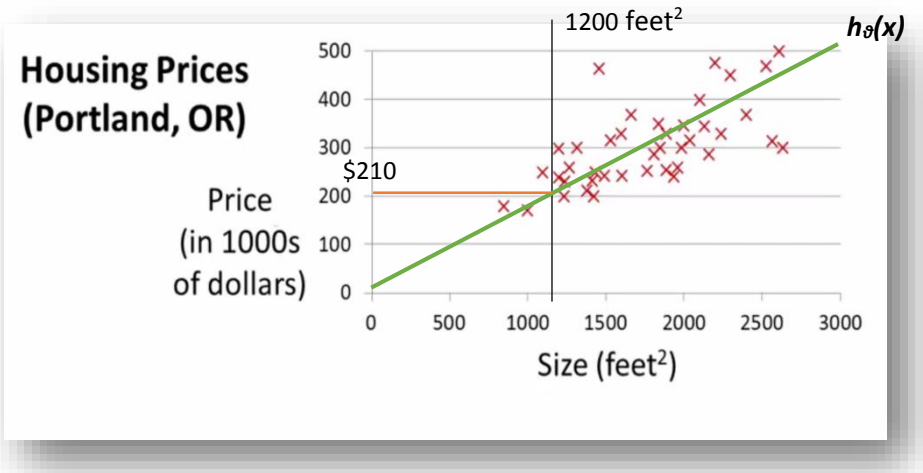


Figure 2: Graphic representation of an implementation of a linear regression on housing prices data according to the feet^2

Solution: As the dataset is labelled and the output and input variable are continuous, a linear relationship between **y** and **x** can be drawn. This mean that it is possible to fit a model $h_\vartheta(x)$ to the training set using linear regression. x=1200 is then projected on the predicted function $h_\vartheta(x)$ and it is now possible to find the corresponding y=210. Hence, for a house of 1200 feet^2, it is predicted that the corresponding price will be $210 (**Figure 2**).

2) *Hypothesis/Predictive Function* $h_\theta(x)$

The Univariate Linear Regression equation can be described by its Hypothesis/Predictive Function:

$$h_\theta = \theta_0 + \theta_1 x \ ,$$

in which θ_0 and θ_1 are parameters that enable to build the best predictive model $h_\vartheta(x)$. More precisely, the choice of (θ_0, θ_1) parameters determine how accurate $h_\vartheta(x^{(i)})$ will predict the actual value $y^{(i)}$ given $x^{(i)}$, such as:

Input x	Output y
0	8
1	5
2	3
3	5

If (θ_0,θ_1) are chosen randomly, $h_\vartheta(x^{(i)}) \neq y^{(i)}$ systematically for each $(x^{(i)},y^{(i)})$ pair, and the predictive function $h_\vartheta(x)$ will not represent or fit graphically the dataset (**Figure 3**)

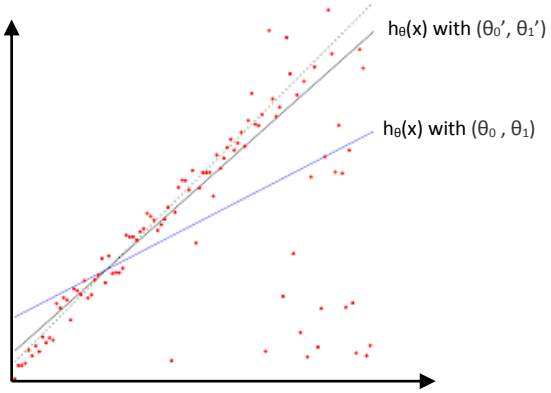


Figure 3: Graphic representation of several predictive function with different (θ_0,θ_1) parameters

Issue: Knowing that $h_\theta(x) = \theta_0 + \theta_1 x$, how can we measure the accuracy when building the predictive function $h_\vartheta(x)$ and how to choose the optimal value of (θ_0, θ_1) such as $h_\vartheta(x^{(i)}) \approx y^{(i)}$ (i.e. low errors) for each $(x^{(i)},y^{(i)})$ pair?

3) *Cost Function* $J(\theta_0,\theta_1)$

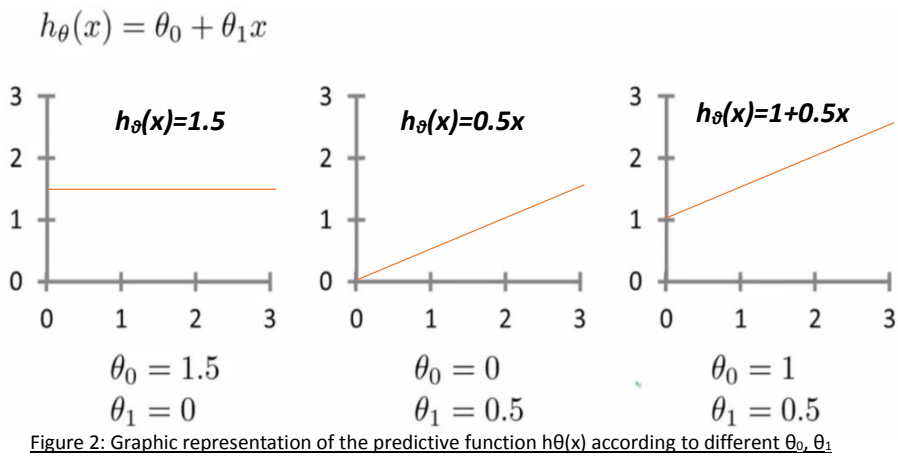


Figure 2: Graphic representation of the predictive function hθ(x) according to different θ_0, θ_1 parameters

Since different value of θ_0, θ_1 result in different predictive/hypothesis function $h_\vartheta(x)$, θ_0, θ_1 will need to be chosen so that each $h_\vartheta(x^{(i)})$ is closed to $y^{(i)}$ for each $(x^{(i)},y^{(i)})$ pair.

For each $h_\vartheta(x^{(i)})$ to be closed to $y^{(i)}$, the distance between each $h_\vartheta(x^{(i)})$ and each $y^{(i)}$ need to be minimised to increase accuracy of the prediction, which means that the error between the predicted and the actual value need to be reduced.

In other words, (**higher accuracy**) $h_\theta(x^{(i)}) \approx y^{(i)} \Leftrightarrow h_\theta(x^{(i)}) - y^{(i)} \approx 0$ (**lower errors**). Thus, to fit the predictive function $h_\vartheta(x)$ well to the training set, $h_\theta(x^{(i)}) - y^{(i)}$ that defined the error between the predicted value $h_\vartheta(x^{(i)})$ and the actual output $y^{(i)}$ at given $x^{(i)}$ need to be minimised, to tend as much as possible to 0 (**Figure 5**).

Graphically, $\sum_{i=1}^{m}(h\theta(x^{(i)}) - y^{(i)})$ is the sum of all the distance between the predicted value $h_\vartheta(x^{(i)})$ and the actual output $y^{(i)}$. Usually, $h_\theta(x^{(i)}) - y^{(i)}$ is squared to $(h_\theta(x^{(i)}) - y^{(i)})^2$ to avoid negative values when minimizing errors. Hence, $\frac{1}{m}\sum_{i=1}^{m}(h_\theta(x^{(i)}) - y^{(i)})^2$ is the mean of the squares of errors $h_\theta(x^{(i)})$-$y^{(i)}$ (i.e. the distance/difference between the predicted value $h_\vartheta(x)$ and the actual output **y**), also called "*Squared error function*" or "*Mean squared error*".

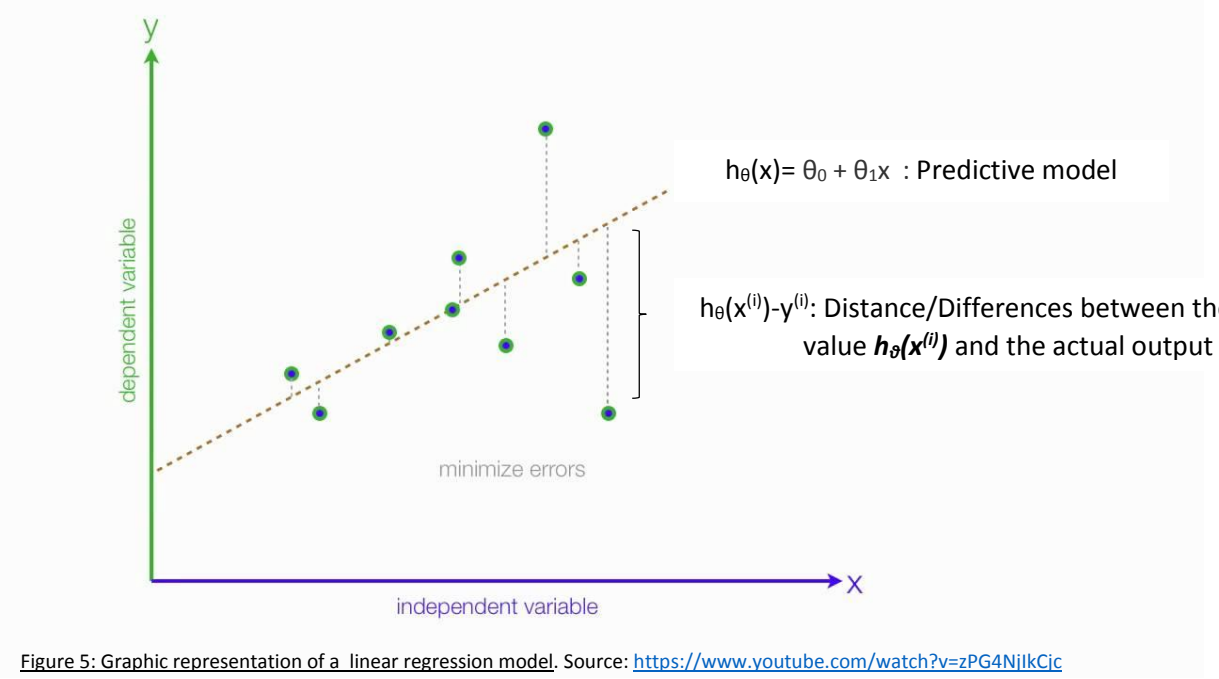


Figure 5: Graphic representation of a linear regression model. Source: https://www.youtube.com/watch?v=zPG4NjIkCjc

The **accuracy** of the hypothesis/predictive function can be measured using a **cost function, $J(\theta_0,\theta_1)$.** The cost function measures specifically the halved average of the squared difference $\left(h_\theta\left(x^{(i)}\right)-y^{(i)}\right)^2$ between **all** the predictive values $h_\vartheta(x^{(i)})$ of the hypothesis function $h_\vartheta(x)$ for a given $x^{(i)}$ and the actual value of the output $y^{(i)}$.

The cost function can be described as:

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

With $h_\theta = \theta_0+\theta_1 x$, and *m* the total number of training example.

Thus, the **optimization objective** will be to minimize the cost function **$J(\theta_0,\theta_1)$ by defining the (θ_0,θ_1) parameter** in order to find the most accurate predictive function $h_\vartheta(x)$ such as $h_\vartheta(x^{(i)}) \approx y^{(i)}$ for each given $(x^{(i)},y^{(i)})$ pair, such as

$$minimize_{\theta_0,\theta_1} \quad J(\theta_0,\theta_1)$$

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

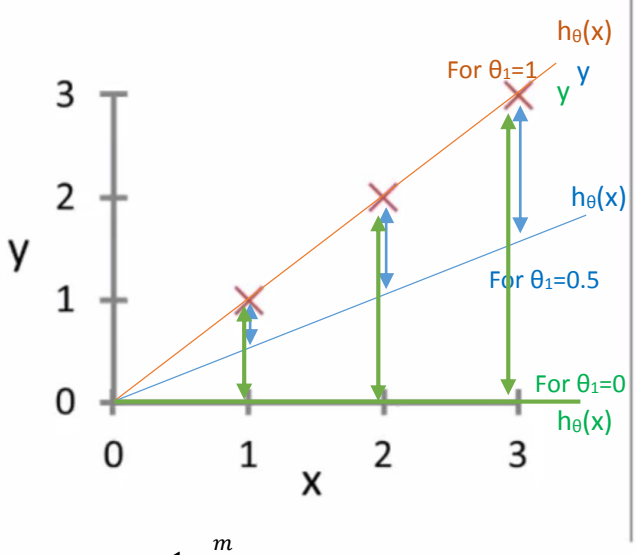
Mathematical Intuition:

	General Expression of the Cost Function	Simplified Expression of the Cost Function
Hypothesis	$h_\theta = \theta_0+\theta_1 x$	$h_\theta = \theta_1 x$ with θ_0
Paramaters	θ_0,θ_1	θ_1
Cost function	$J(\theta_0,\theta_1)=\frac{1}{2m}\sum_{i=1}^{m}(h_\theta(x^{(i)})-y^{(i)})^2$	$J(\theta_0,\theta_1)=\frac{1}{2m}\sum_{i=1}^{m}(h_\theta(x^{(i)})-y^{(i)})^2$
Goal	$minimize_{\theta_0,\theta_1} \quad J(\theta_0,\theta_1)$	$minimize_{\theta_0,\theta_1} \quad J(\theta_1)$

➔ **For the Simplified expression of the Cost Function J(θ₁)**:

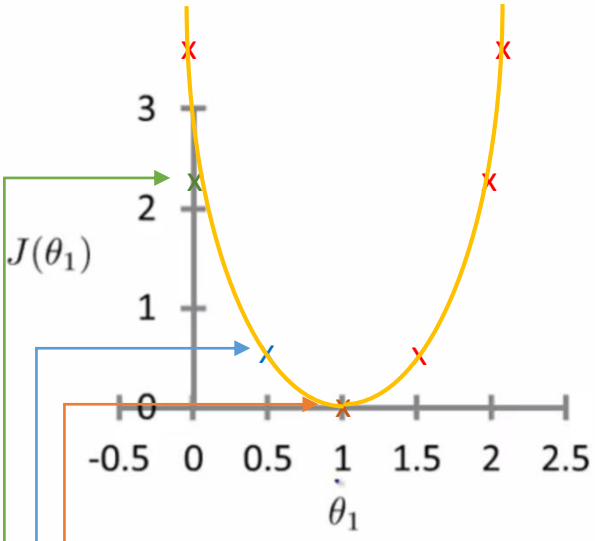
$h_\theta(x)$

For a fixed value of θ_1, h_θ is a function of x



$J(\theta_1)$

$J(\theta_1)$ is a function of the parameter θ_1



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

$$J(\theta_1) = \frac{1}{2m}\sum_{i=1}^{m}(\theta_1 x^{(i)} - y^{(i)})^2$$

For **θ₁=1**

$$J(1) = \frac{1}{2*3}[(1*1-1)^2 + (1*2-2)^2 + (1*3-3)^2]$$

$$J(1) = \frac{1}{2m}(0^2 + 0^2 + 0^2) = 0$$

For **θ₁=0.5**

$$J(0.5) = \frac{1}{2*3}[(0.5*1-1)^2 + (0.5*2-2)^2 + (0.5*3-3)^2]$$

$$J(0.5) = \frac{1}{2*3}(3.5) = 0.68$$

For **θ₁=0**

$$J(0) = \frac{1}{2*3}(1^2 + 2^2 + 3^2]$$

$$J(0) = \frac{1}{2*3}(14) = 2.3$$

Etc…

Each value of θ_1 corresponds to a different hypothesis $h_\theta(x)$, and for each value of θ_1, it is possible to ***derive*** a different value of $J(\theta_1)$.

The $J(\theta_1)$ function in univariate linear regression will be always convex, which mean that $J(\theta_1)$ will always have a unique global minimum without local minima. Therefore, when derive $J(\theta_1)$, $\frac{\delta}{\delta\theta_j}J(\theta_0,\theta_1)$=0 will always be represented by a straight line parallel to the θ_1 axis, and will be always assigned the global minima of the $J(\theta_1)$ function

The optimization objective was to choose a θ_1 to minimise the cost function $J(\theta_1)$.

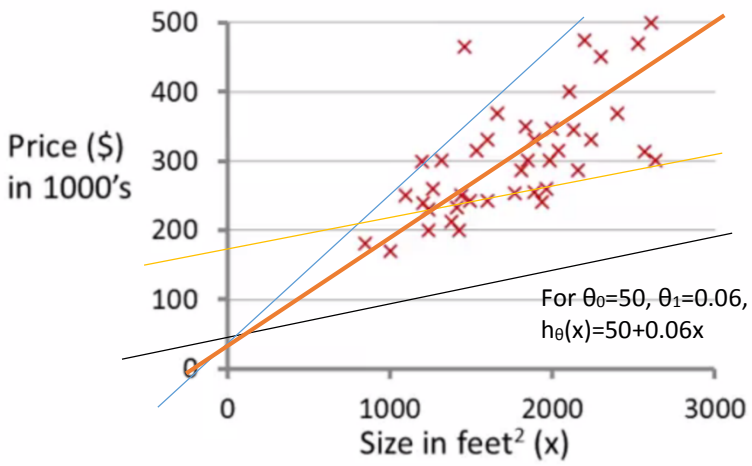
Thus, by look of the 2D plot $J(\theta_1)$ vs. θ_1, the minimum value of $J(\theta_1)$ is J(1)=0 with θ_1=1.

That is why minimizing $J(\theta_1)$ corresponds to finding a straight line that fits the data well in the 2D plot *y* vs. *x*.

➔ **For the General expression of the Cost Function $J(\theta_0\theta_1)$:**

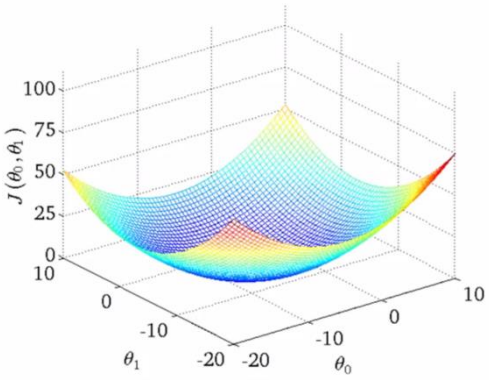
$h_\theta(x)$

For a fixed value of $\theta_0\,\theta_1$, h_θ is a function of x



$J(\theta_0, \theta_1)$

$J(\theta_0\theta_1)$ is a function of the parameter $\theta_0\,\theta_1$, h_θ is a function of



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

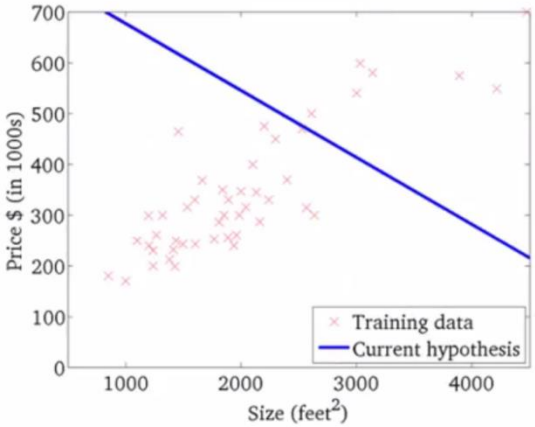
$$J(\theta_0\theta_1) = \frac{1}{2m}\sum_{i=1}^{m}(\theta_0 + \theta_1 x^{(i)} - y^{(i)})^2$$

Each value of θ_0 *and* θ_1 corresponds to a different hypothesis $h_\theta(x)$, and at each coordinate (θ_0, θ_1) it is possible to ***derive*** a different value of $J(\theta_1)$, in order to find the global minima (i.e. the smallest of $J(\theta_0,\theta_1)$ given parameters θ_0 *and* θ_1) that enable to fit the predictive model $h_\theta(x)$ optimally to the dataset. The $J(\theta_0,\theta_1)$ function in univariate linear regression will be always convex, which mean that $J(\theta_0,\theta_1)$ will always have a unique global minimum without local minima. Therefore, when derive $J(\theta_1)$, $\frac{\delta}{\delta\theta_j}J(\theta_0,\theta_1)$=0 will always be represented by a straight line parallel to the θ_1 axis, and will be always assigned the global minima of the $J(\theta_1)$ function.

For a better understanding of the 3D plot, **contour plots/figure** is usually used to show the surfaces of (θ_0, θ_1) from the 3D plot.

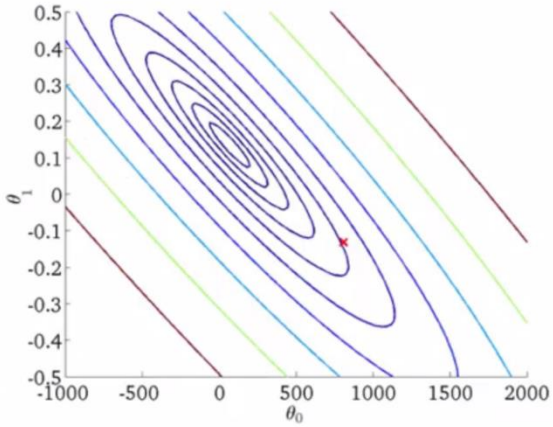
$h_\theta(x)$

For a fixed value of $\theta_0\,\theta_1$, h_θ is a function of x

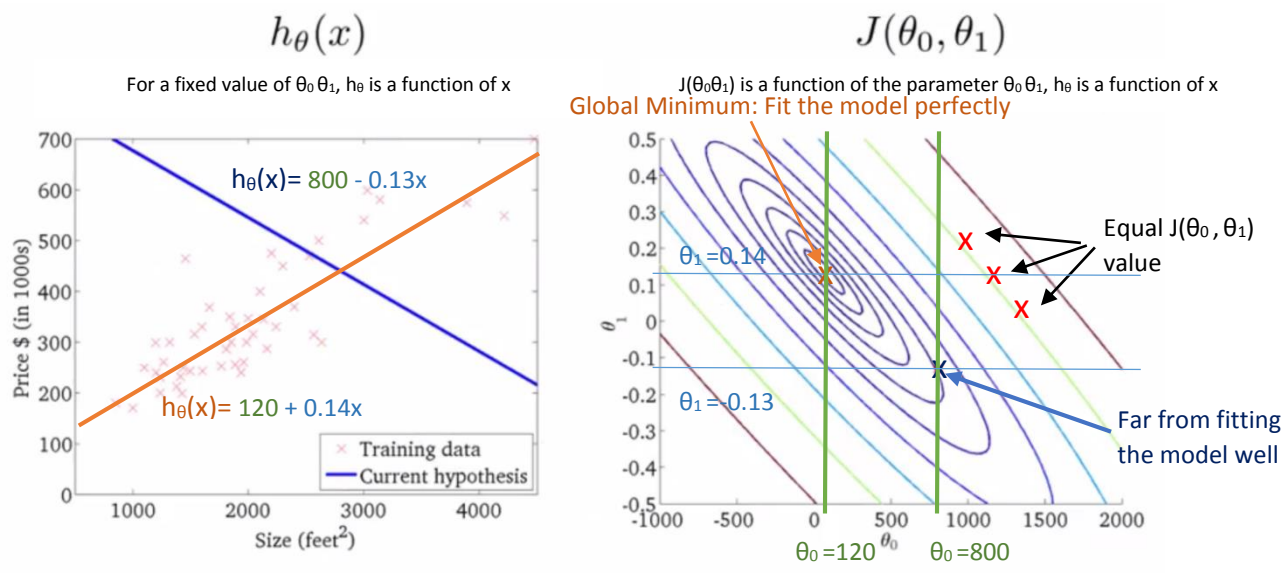


$J(\theta_0, \theta_1)$

$J(\theta_0\theta_1)$ is a function of the parameter $\theta_0\,\theta_1$, h_θ is a function of x



Each of the ellipse in the $J(\theta_0, \theta_1)$ 2D plot represent a set of points that takes on an identical value of $J(\theta_0, \theta_1)$. Thus, each colour of the ellipse stand for a different value of $J(\theta_0, \theta_1)$.



The centre of the smallest ellipse always represents the global minimum value of of $J(\theta_0, \theta_1)$, in which the corresponding values of θ_0, θ_1 enable the fitting of the model perfectly **$h_\theta(x)$**.

Testing all parameter θ_0, θ_1 is fastidious. Thus, in order to find **automatically** the value of θ_0, θ_1 that minimize the cost function $J(\theta_0, \theta_1)$, **Gradient Descent** algorithm is usually used.

4) *Gradient Descent*

Gradient Descent is a first-order optimization algorithm that can be broadly applied to a lot of model used in machine learning. Here, we will focus on gradient descent used for linear regression.
The goal of gradient descent applied to linear regression is to $minimize_{\theta_0,\theta_1}\ J(\theta_0, \theta_1)$.
Outline:

- Start with some θ_0, θ_1 (e.g. $\theta_0 = 0, \theta_1 = 0$)
- Keep changing θ_0, θ_1 to reduce $J(\theta_0, \theta_1)$ until end up at a minimum.

Gradient Descent Algorithm:

Repeat until convergence {

$$\boldsymbol{\theta_j := \theta_j - \alpha \frac{\partial}{\partial \theta_j} J(\theta_0, \theta_1)} \qquad \text{(for j=0 and j=1)}$$

}

with := represent the implementation of θ_j at each **step**.

For $\boldsymbol{\frac{\partial}{\partial \theta_j} J(\theta_0, \theta_1) \neq 0}$, gradient descent will take a **step** downhill while for $\boldsymbol{\frac{\partial}{\partial \theta_j} J(\theta_0, \theta_1) = 0}$, the global minima is found and the gradient descent algorithm converge and stop.

α refers the learning rate. It controls how big a **step** is take downhill with gradient descent. Thus, if α is very large, this correspondent to a very aggressive gradient descent that take a huge steps downhill whereas if α is very small, gradient descent is taking tiny steps downhill.

In gradient descent, $\boldsymbol{\theta_0, \theta_1}$ have to be updated simultaneously. This mean that $\boldsymbol{\theta_j - \alpha \frac{\partial}{\partial \theta_j} J(\theta_0, \theta_1)}$ (for j=0 and j=1) need to be **<u>first</u>** compute before implementation $\boldsymbol{\theta_j := \theta_j - \alpha \frac{\partial}{\partial \theta_j} J(\theta_0, \theta_1)}$, such as

Compute first:

$$\boldsymbol{temp0 := \theta_0 - \alpha \frac{\partial}{\partial \theta_0} J(\theta_0, \theta_1)}$$

$$\boldsymbol{temp1 := \theta_1 - \alpha \frac{\partial}{\partial \theta_1} J(\theta_0, \theta_1)}$$

Then, update simultaneously:

Simultaneous

$$\boldsymbol{\theta_0 := temp0}$$

$$\boldsymbol{\theta_1 := temp1}$$

The incorrect implementation that does not simultaneously update will be:

$$\boldsymbol{temp0 := \theta_0 - \alpha \frac{\partial}{\partial \theta_0} J(\theta_0, \theta_1)}$$

$$\boldsymbol{\theta_0 := temp0}$$

Not Simultaneous

$$\boldsymbol{temp1 := \theta_1 - \alpha \frac{\partial}{\partial \theta_1} J(\theta_0, \theta_1)}$$

$$\boldsymbol{\theta_1 := temp1}$$

Where in that case $\boldsymbol{\frac{\partial}{\partial \theta_1} J(\theta_0, \theta_1)}$ from ***temp1*** will have a $\boldsymbol{J(\theta_0, \theta_1)}$ with an updated $\boldsymbol{\theta_0}$ while $\boldsymbol{\theta_1}$ will not be.

<u>In more details,</u>

➔ **For the Simplified expression of the Cost Function J(θ_1):**

The optimization objective is to $minimize_{\theta_0,\theta_1}\ J(\theta_1)$. Gradient Descent will update $J(\theta_1)$ at each step until converging (i.e. finding the global minimum where $\boldsymbol{\frac{\partial}{\partial \theta_j} J(\theta_1) = 0}$).

Repeat until convergence {

$$\boldsymbol{\theta_1 := \theta_1 - \alpha \frac{\partial}{\partial \theta_1} J(\theta_1)} \qquad \theta_1 \in R$$

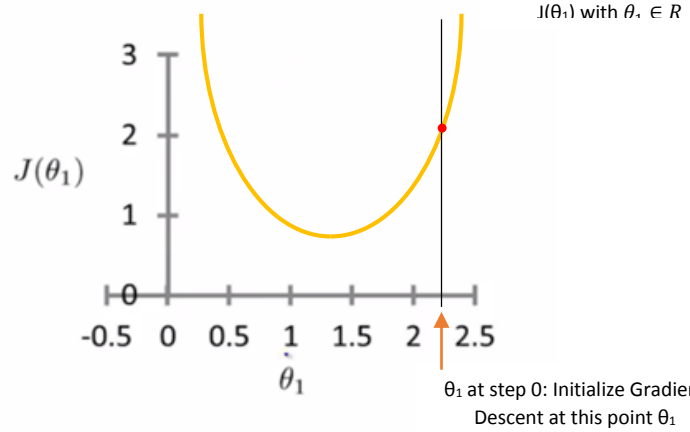
}

First, Gradient Descent starts at a random initialized θ_1 value such as

$J(\theta_1)$

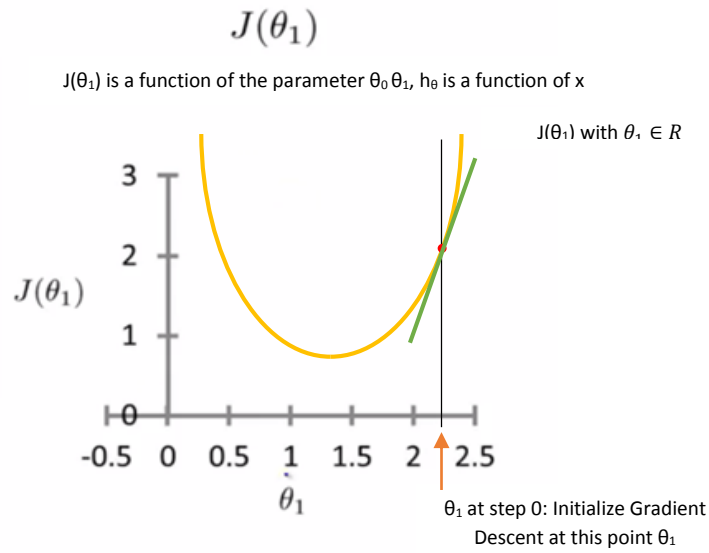
J(θ_1) is a function of the parameter θ_1, h_θ is a function of x

J(θ_1) with $\theta_1 \in R$

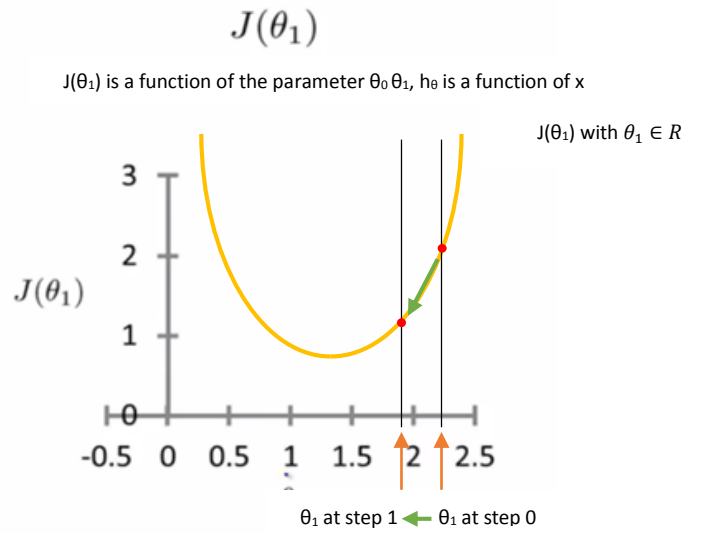


θ_1 at step 0: Initialize Gradient Descent at this point θ_1

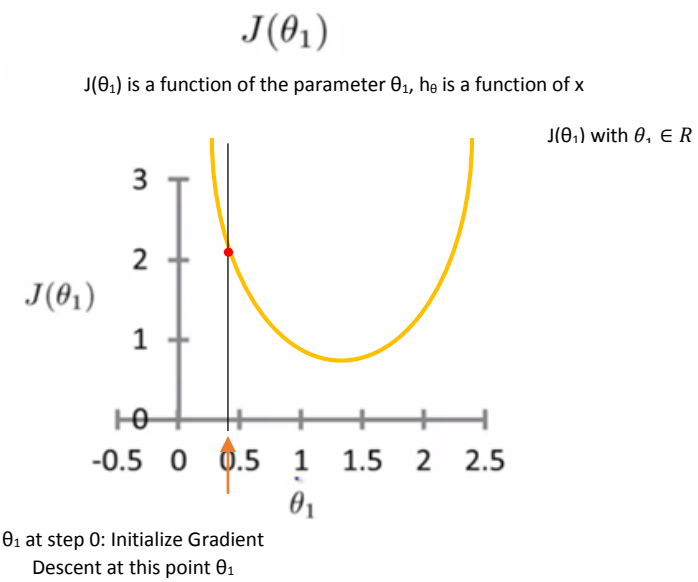
Then, the slope of the derivative $\frac{d}{d\theta_1}J(\boldsymbol{\theta_1})$ at initialized θ_1 is calculated

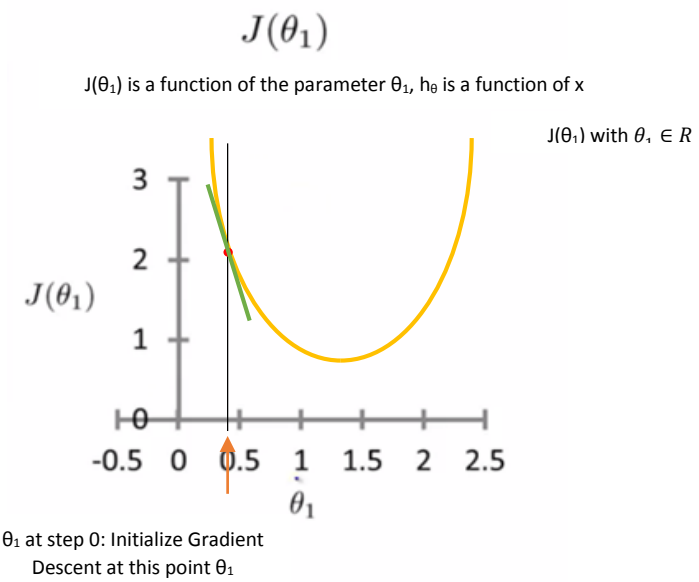


In this case, $\frac{d}{d\theta_1}J(\boldsymbol{\theta_1}) \geq \mathbf{0}$ so that $\boldsymbol{\theta_1} := \boldsymbol{\theta_1} - \boldsymbol{\alpha(positive\ number)}$, which means that $\boldsymbol{\theta_1}$ is going to decrease after being updated. Thus, graphically, gradient descent is going to take a value of $\boldsymbol{\theta_1}$ at step 1 that will be smaller than the $\boldsymbol{\theta_1}$ at step 0.

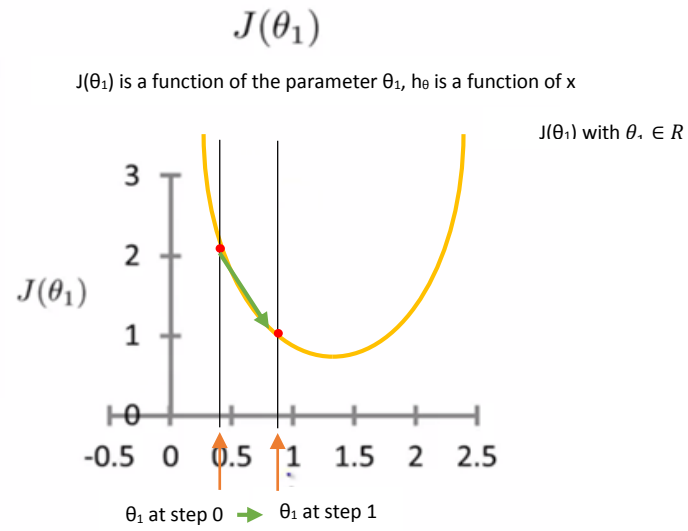


If the gradient descent was initialized this way:





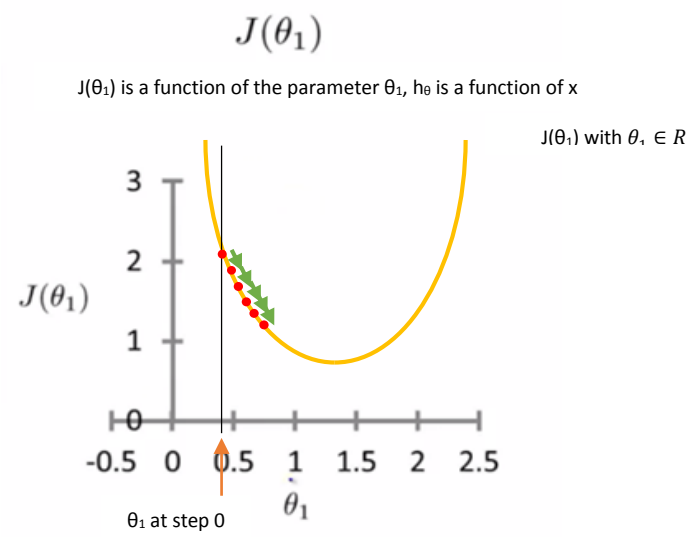
In this case, $\frac{d}{d\theta_1}\boldsymbol{J(\theta_1)} \leq \mathbf{0}$ so that $\boldsymbol{\theta_1} := \boldsymbol{\theta_1} - \boldsymbol{\alpha(negative\ number)}$, which means that $\boldsymbol{\theta_1}$ is going to increase after being updated. Thus, graphically, gradient descent is going to take a value of $\boldsymbol{\theta_1}$ at step 1 that will be bigger than the $\boldsymbol{\theta_1}$ at step 0.



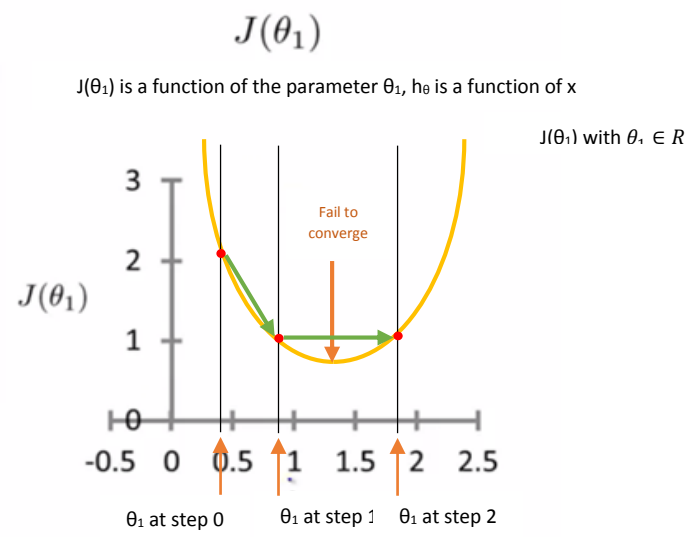
Moreover, choosing the right learning rate $\boldsymbol{\alpha}$ enable to control how fast gradient descent will step downhill after each update before converging (i.e. finding the global minimum):

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

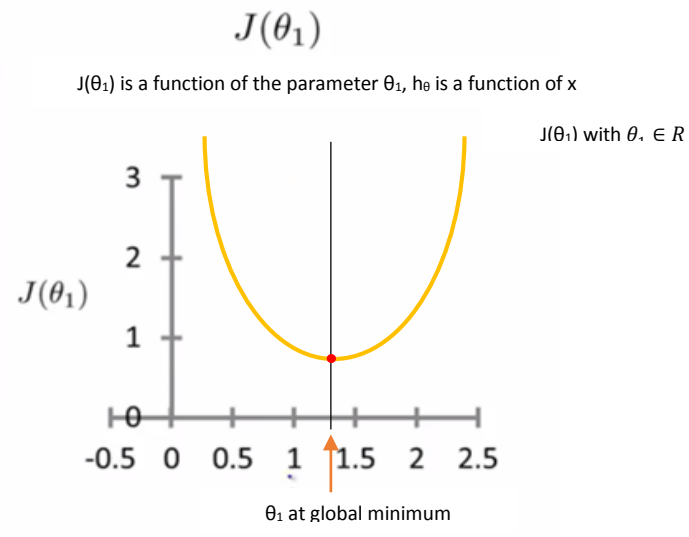
If $\boldsymbol{\alpha}$ is too small, gradient descent can be slow as each update will be very small. Gradient Descent will then need a lot of step before reaching closed to the global minima



If $\boldsymbol{\alpha}$ is too large, gradient descent can overshoot the minimum. It may fail to converge, or even diverge



When gradient descent has reached the local global minimum:



$$\boldsymbol{\theta_1} := \boldsymbol{\theta_1} - \boldsymbol{\alpha}\frac{\partial}{\partial \boldsymbol{\theta_1}} \boldsymbol{J(\theta_1)} \text{ with } \frac{\partial}{\partial \boldsymbol{\theta_1}} \boldsymbol{J(\theta_1)} = \mathbf{0}$$

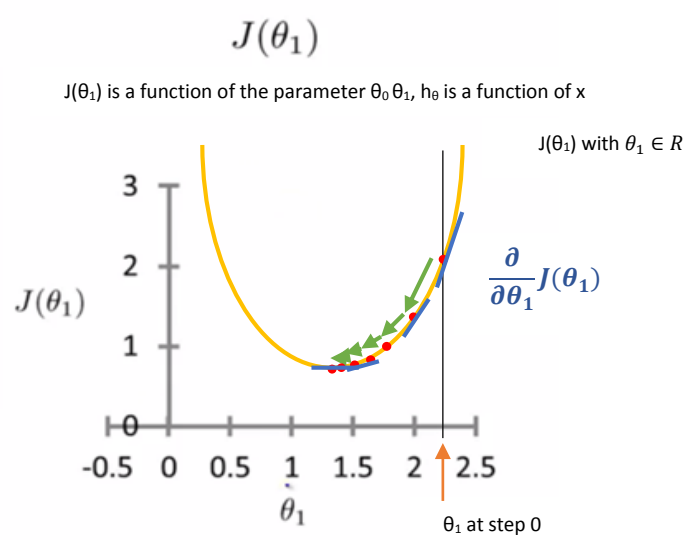
$$\therefore \boldsymbol{\theta_1} := \boldsymbol{\theta_1}$$

Gradient descent will stop update and will stay at global minimum, defining thus, the optimal $\boldsymbol{\theta_1}$parameter.

Additionally, gradient descent can converge to a global minimum, even with an appropriate learning rate $\boldsymbol{\alpha}$ fixed.

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

As we approach a local minimum, gradient descent will automatically take smaller steps as $\boldsymbol{\frac{\partial}{\partial \theta_1} J(\theta_1)}$ will adjust by itself, which mean that the gradient descent will take smaller and smaller steps when approaching the global minimum.



Thus, there is no need to decrease $\boldsymbol{\alpha}$ manually over time. The only concern is to choose and to fix the most appropriate value of $\boldsymbol{\alpha}$ that enable to find quickly the global minimum without overshooting it.

➔ **For the General expression of the Cost Function J(θ₀θ₁): Gradient descent for Linear Regression**

Gradient descent algorithm	**Linear Regression Model**
Repeat until convergence { $\boldsymbol{\theta_j := \theta_j - \alpha \frac{\partial}{\partial \theta_j} J(\theta_0, \theta_1)}$ **(for j=0 and j=1) }**	$h_\theta = \theta_0 + \theta_1 x$ $J(\theta_0, \theta_1) = \frac{1}{2m} \sum_{i=1}^{m} (h_\theta(x^{(i)}) - y^{(i)})^2$

The goal is to apply gradient descent to $minimize_{\theta_0, \theta_1} \quad J(\theta_0, \theta_1)$.

Thus,

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

$$\boldsymbol{\frac{\partial}{\partial \theta_j} J(\theta_0, \theta_1)} = \frac{\partial}{\partial \theta_j} \frac{1}{2m} \sum_{i=1}^{m} (\theta_0 + \theta_1 x^{(i)}) - y^{(i)})^2$$

For $j = 0$,

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

For $j = 1$,

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

N.B: If $\frac{1}{m}\sum_{i=1}^{m}(h\theta(x^{(i)}) - y^{(i)})^2$ can be represented as $\overline{x}$, it will be noticed that the cost function is equal to half of the mean squared error such as $J(\theta_0, \theta_1) = \frac{1}{2}\,\overline{x}$. This is convenient for finding the global minima of the **cost function, J(θ₀,θ₁)** when 3D plotting **J(θ) over θ.** When deriving the cost function **J(θ₀,θ₁)** to find the global minima at $\frac{\delta}{\delta \theta_j} J(\theta_0, \theta_1)$, the square term of the function $(h\theta(x^{(i)}) - y^{(i)})^2$ will cancel out the $\frac{1}{2}$ term.

Thus, gradient descent for linear regression

Repeat until convergence {

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

$$\theta_1 := \theta_1 - \alpha \frac{1}{m} \sum_{i=1}^{m} (h_\theta(x^{(i)}) - y^{(i)}) * x^{(i)}$$

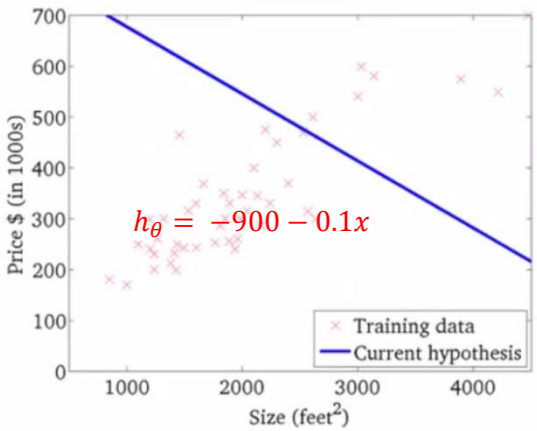
Update θ_0, θ_1 simultaneously

}

Gradient descent for linear regression will always end up at global minimum and not at a local minimum because the cost function $\boldsymbol{J(\theta_0, \theta_1)}$ of a linear regression model is always a convex function (also called Bowl-shaped function), which only have a global optima and no local optima.

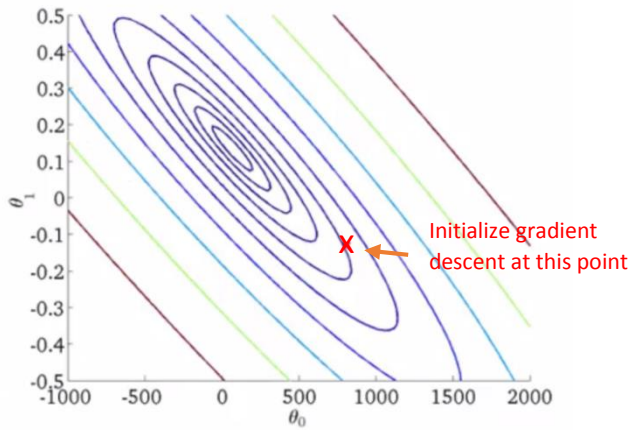
$h_\theta(x)$

For a fixed value of θ₀ θ₁, hθ is a function of x



$J(\theta_0, \theta_1)$

J(θ₀θ₁) is a function of the parameter θ₀ θ₁, hθ is a function of x



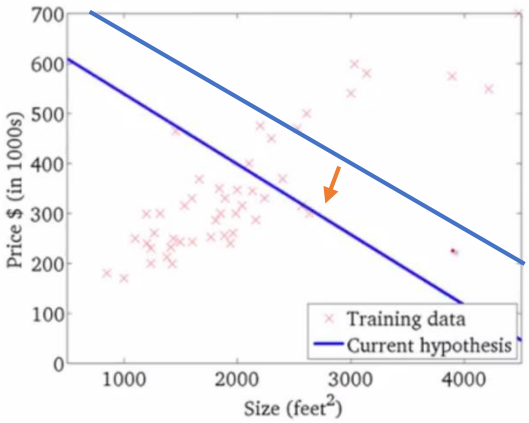
Gradient descent is first initialized at this point. Then, gradient descent is run.

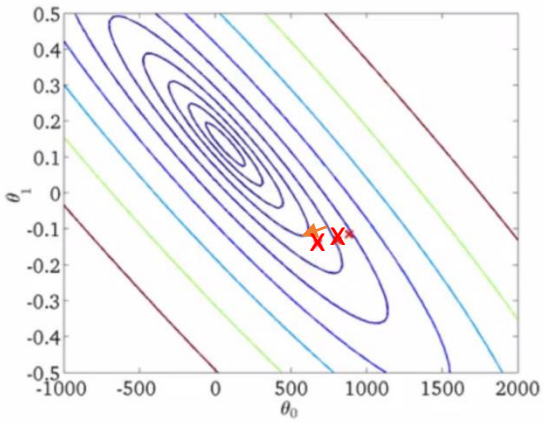
$h_\theta(x)$

For a fixed value of $\theta_0\,\theta_1$, h_θ is a function of x

$J(\theta_0, \theta_1)$

$J(\theta_0\theta_1)$ is a function of the parameter $\theta_0\,\theta_1$, h_θ is a function of x





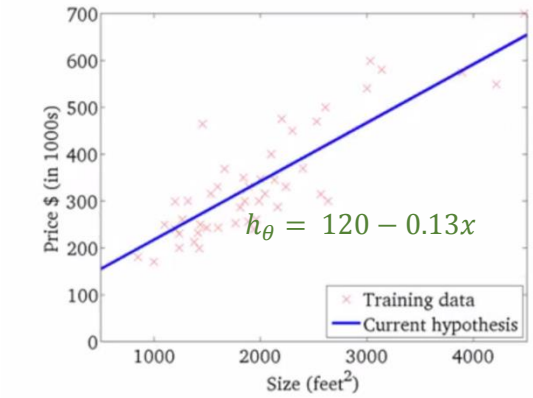
Until reaching the global minima,

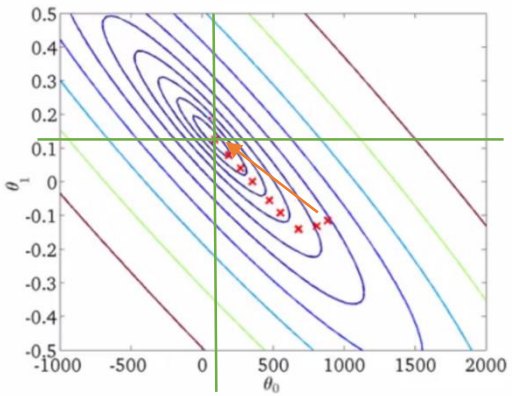
$h_\theta(x)$

For a fixed value of $\theta_0\,\theta_1$, h_θ is a function of x

$J(\theta_0, \theta_1)$

$J(\theta_0\theta_1)$ is a function of the parameter $\theta_0\,\theta_1$, h_θ is a function of x





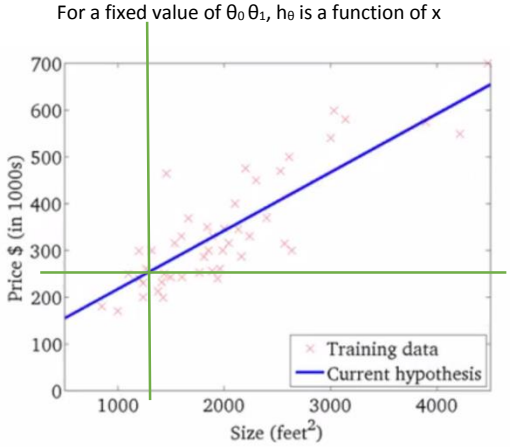
By reaching the global minimum, gradient descent finds out the optimal value of (θ_0=120, θ_1=0.13), which enable to fit well the predictive model $h_\theta = 120 - 0.13x$ to the training set.

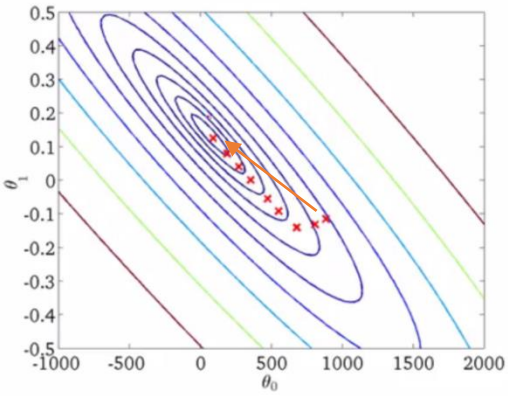
$h_\theta(x)$

For a fixed value of $\theta_0\,\theta_1$, h_θ is a function of x

$J(\theta_0, \theta_1)$

$J(\theta_0\theta_1)$ is a function of the parameter $\theta_0\,\theta_1$, h_θ is a function of x





Thus, for a 1250 feet^2 house, it is possible to predict that the housing price will be around $260 000.

This type of gradient descent that is used in this section is called "*Batch*" Gradient Descent. In "*Batch*" Gradient Descent, all the training examples (e.g. $\sum_{i=1}^{m}(h_\theta(x^{(i)}) - y^{(i)})$) are used at each step of the gradient descent when updating the algorithm. Other type of gradient descent might use only one or a subset of the training examples instead of all the training set at each step when updating the algorithm.

II. Multivariate Linear Regression

Multivariate Linear Regression refers to a Linear Regression with more than one variable/features ***x*** and is used to:

- Fit the best predictive model to forecast/predict accurately a single and continuous known/labelled output values $y^{(i)}$ given **several** known input $x^{(i)}$.
- Quantify the strength of the relationship between a single known output values ***y*** given **several** known/labelled input ***x***.

1) *Starting Example: "Predicting Housing Prices"*

In Univariate Linear Regression, there is only a single variable/feature **x** (the size of the house) that was used to predict the output variable **y** (the housing price) using the predicted/hypothesis function $h_\theta = \theta_0 + \theta_1 x$

Index of training examples	Size in feet^2 (x)	Price in $ (y)
i=1	2014	460
i=2	1490	244
i=3	900	190
i=m	…	…

However, in Multivariate Linear Regression, there are several variable/feature $\mathbf{x_n}$ that could be used to predict the output variable **y** (the housing price).

Index j of variable/features	j=1	j=2	j=3	j=**n**	
Index i of training examples	Size in feet^2 ($\mathbf{x_1}$)	Number of Bedrooms ($\mathbf{x_2}$)	Number of floors ($\mathbf{x_3}$)	… ($\mathbf{x_n}$)	Price in $ (**y**)
i=1	2014	1	45	…	460
i=2	1490	2	40	…	244
i=3	$x_1^{(3)} = 900$	2	35	…	190
i=**m**	…	…	…	…	…

Given a training set of house prices:

m is the *total number of training examples*

n is the *total number of feature*

x is the *input variables/features*

$x^{(i)}$ is the *input variables/features of i^{th} training example*

$x_j^{(i)}$ is the *value of features j in i^{th} training example*

y is the *output/ target variable*

For instance:

$$x_1^{(3)} = 900; \quad x^{(2)} = \begin{bmatrix} 1490 \\ 2 \\ 40 \end{bmatrix}; \quad x_2 = \begin{bmatrix} 1 \\ 2 \\ 2 \end{bmatrix}$$

2) *Hypothesis/Predictive Function* $\mathbf{h_\theta(x)}$

Previously, the predicted/hypothesis function for the Univariate Linear Regression was represented by

$$h_\theta(x) = \theta_0 + \theta_1 x$$

In **Multivariate Linear Regression**, the predicted/hypothesis function is represented by:

$$h_\theta = \theta_0 + \theta_1 x_1 + \theta_2 x_2 + \theta_3 x_3 + \ldots + \theta_n x_n$$

Where $x_1 is\ the\ housing\ size, x_2 the\ number\ of\ bedrooms, x_3 the\ number\ of\ floor$

Where $\boldsymbol{x_0^{(i)} = 1}$

For more convenience, it is also possible to define all the feature $\boldsymbol{x} \in \boldsymbol{R^{n+1}}$ *(indexed from 0)* as $\boldsymbol{x} = \begin{bmatrix} \boldsymbol{x_0} \\ \boldsymbol{x_1} \\ \ldots \\ \boldsymbol{x_n} \end{bmatrix}$

And to define all the parameter $\theta \in \boldsymbol{R^{n+1}}$ *(indexed from 0)* as $\boldsymbol{\theta} = \begin{bmatrix} \theta_0 \\ \theta_1 \\ \ldots \\ \theta_n \end{bmatrix}$, such as

$h_\theta = \theta_0 + \theta_1 x_1 + \theta_2 x_2 + \theta_3 x_3 + \ldots + \theta_n x_n$ can also be written as

$$h_\theta = \boldsymbol{\theta}^T \boldsymbol{x}$$

θ is transposed to enable a matrix multiplication such as $\boldsymbol{\theta}^T = \theta_0 \quad \theta_1 \quad \ldots \quad \theta_n$ and

$$h_\theta = \theta^T \boldsymbol{x} = [\theta_0 \quad \theta_1 \quad \ldots \quad \theta_n] * \begin{bmatrix} \boldsymbol{x_0} \\ \boldsymbol{x_1} \\ \ldots \\ \boldsymbol{x_n} \end{bmatrix} \qquad \textit{(inner product)}$$

3) *Cost Function $J(\vartheta_0, \vartheta_1)$ and Gradient Descent*

Mathematical Intuition:

	Univariate Linear Regression	**Multivariate Linear Regression**
Hypothesis	$h_\theta = \theta_0 + \theta_1 x$	$h_\theta = \theta_0 x_0 + \theta_1 x_1 + \theta_2 x_2 + \theta_3 x_3 + \ldots + \theta_n x_n$ with $x_0 = 1$
Paramaters	θ_0, θ_1	$\theta_0, \theta_1, \ldots, \theta_n = \boldsymbol{\theta}$
Cost function	$J(\theta_0, \theta_1) = \frac{1}{2m} \sum_{i=1}^{m} (h_\theta(x^{(i)}) - y^{(i)})^2$	$J(\theta_0, \theta_1, \ldots, \theta_n) = \frac{1}{2m} \sum_{i=1}^{m} (h_\theta(x^{(i)}) - y^{(i)})^2$ $J(\boldsymbol{\theta}) = \frac{1}{2m} \sum_{i=1}^{m} (h_\theta(x^{(i)}) - y^{(i)})^2$
Goal	$minimize_{\theta_0, \theta_1} \quad J(\theta_0, \theta_1)$	$minimize_{\theta_0, \theta_1, \ldots, \theta_n} \quad J(\theta_0, \theta_1, \ldots, \theta_n)$ $minimize_{\boldsymbol{\theta}} \quad J(\boldsymbol{\theta})$

Gradient Descent Algorithm:

Repeat {

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

} (simultaneously update for every j=0, 1, …, n)

Previously for univariate linear regression (n=1),

$\frac{\partial}{\partial \theta_0} J(\theta_0, \theta_1)$

Repeat {

$$\boldsymbol{\theta_0} := \boldsymbol{\theta_0} - \boldsymbol{\alpha} \frac{1}{m} \sum_{i=1}^{m} (h_\theta(x^{(i)}) - y^{(i)})$$

$$\boldsymbol{\theta_1} := \boldsymbol{\theta_1} - \boldsymbol{\alpha} \frac{1}{m} \sum_{i=1}^{m} (h_\theta(x^{(i)}) - y^{(i)}) * x_1{}^{(i)}$$

Update θ_0, θ_1 simultaneously

}

$\frac{\partial}{\partial \theta_1} J(\theta_0, \theta_1)$

For multivariate linear regression (n≥1):

Repeat {

$$\boldsymbol{\theta_1} := \boldsymbol{\theta_1} - \boldsymbol{\alpha} \frac{1}{m} \sum_{i=1}^{m} (h_\theta(x^{(i)}) - y^{(i)}) * x_j{}^{(i)}$$

Update $\theta_0, \theta_1, \dots, \theta_n$ simultaneously

}

$\frac{\partial}{\partial \theta_j} J(\theta)$

For $j = 0$,

$$\boldsymbol{\theta_0} := \boldsymbol{\theta_0} - \boldsymbol{\alpha} \frac{1}{m} \sum_{i=1}^{m} (h_\theta(x^{(i)}) - y^{(i)}) * x_0{}^{(i)}$$

Since it is assumed that $x_0{}^{(i)} = 1$ by convention,

$$\boldsymbol{\theta_0} := \boldsymbol{\theta_0} - \boldsymbol{\alpha} \frac{1}{m} \sum_{i=1}^{m} (h_\theta(x^{(i)}) - y^{(i)})$$

Similar to the Univariate Linear Regression for *j=0*

For $j = 1$,

$$\boldsymbol{\theta_1} := \boldsymbol{\theta_1} - \boldsymbol{\alpha} \frac{1}{m} \sum_{i=1}^{m} (h_\theta(x^{(i)}) - y^{(i)}) * x_1{}^{(i)}$$

For $j = 2$,

$$\boldsymbol{\theta_2} := \boldsymbol{\theta_2} - \boldsymbol{\alpha} \frac{1}{m} \sum_{i=1}^{m} (h_\theta(x^{(i)}) - y^{(i)}) * x_2{}^{(i)}$$

…

4) Feature Scaling and Mean Normalisation

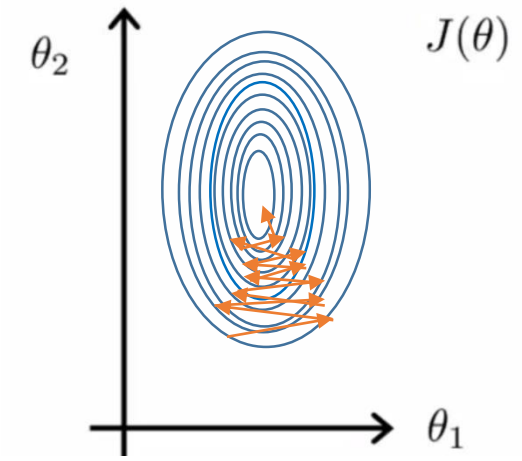
Feature Scaling is a practical **optimization** tool that is often used in Multivariate Linear Regression to ensure that features are on a similar scale.

For instance, given

$\boldsymbol{x_1} = size$ (between 0-2000 feet^2)

$\boldsymbol{x_2} = number\ of\ bedrooms$ (between 1-5)

Since the ratio of $\frac{\boldsymbol{x_1}, size\ (\text{between } 0-2000\ \text{feet}^2)}{\boldsymbol{x_2}, number\ of\ bedrooms\ (\text{between } 1-5)}$ is going to be very big, the contour plot of the cost function $J(\boldsymbol{\theta})$ will appear as a skewed, tall, skinny ellipse in which the gradient descent can run for a long time before finding the global minimum.

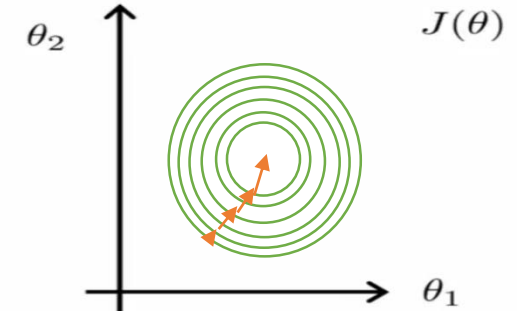


The solution is to scale the feature such as $\frac{\boldsymbol{x_1}}{\boldsymbol{S_1}}$ with **S_1 the range (maximum value – minimum value)**

$$\boldsymbol{x_1} = \frac{size}{\boldsymbol{2000}}$$

$$\boldsymbol{x_2} = \frac{number\ of\ bedrooms}{\boldsymbol{4}}$$

So that the contour of the cost function $J(\boldsymbol{\theta})$ become less skewed, and gradient descent will run much faster



Generally speaking, when performing feature scaling, every features are scaled approximately from a range of

$$\boldsymbol{-1 \leq x_i \leq 1}$$

Additionally, when performing feature scaling, it is usually to do a **Mean Normalization**. It consists of replacing the $\boldsymbol{x_i}$ by $\boldsymbol{x_i - \mu_i}$ to make every features have approximately zero mean (except for $\boldsymbol{x_0}$). For instance

$\frac{\boldsymbol{x_i - \mu_i}}{\boldsymbol{S_i}}$ **is used** Such that $\boldsymbol{-0.5 \leq x_1 \leq 0.5}$ **and** $\boldsymbol{-0.5 \leq x_2 \leq 0.5}$

$\boldsymbol{x_1} = \frac{size-1000}{\boldsymbol{2000}}$ with $\mu_1 = 1000$ and $S_1 = 2000 - 0 = 2000$

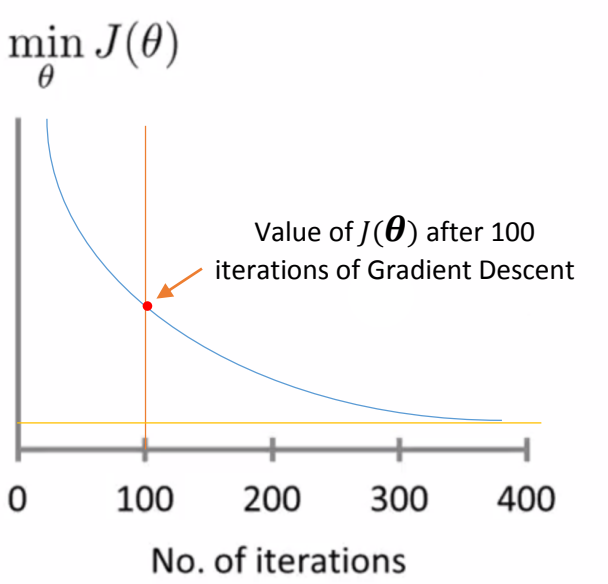
$\boldsymbol{x_2} = \frac{number\ of\ bedrooms-2}{\boldsymbol{4}}$ with $\mu_2 = 2$ and $S_2 = 5 - 1 = 4$

5) *Learning rate Optimization*

To run the gradient descent faster, it is important to choose an optimal learning rate $\boldsymbol{\alpha}$.

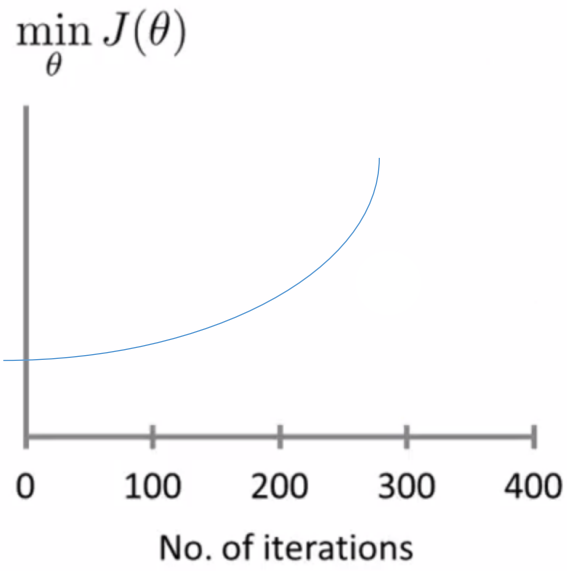
To make sure that gradient descent is working correctly in $minimizing_{\boldsymbol{\theta}} \; J(\boldsymbol{\theta})$, the cost function $\boldsymbol{J(\theta)}$ **can be plotted over the number of iterations** that gradient descent run.

Normal behaviour of gradient descent over numerous iterations should behave like that:

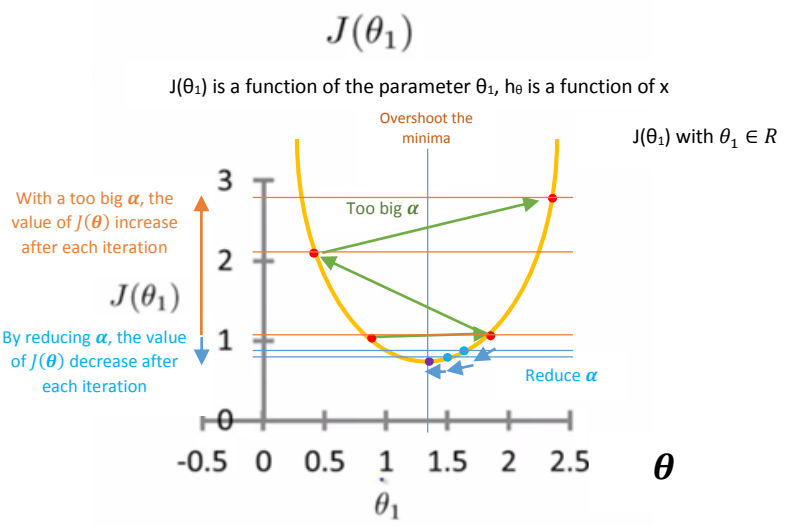


The cost function $J(\boldsymbol{\theta})$ should normally decrease after every iteration until converging to a minimum. Some algorithm might take 30, 3000, 3000000 iterations for the Gradient Descent to find the minimum. Thus, the number of iterations before converging can be reduced by choosing the appropriate value of $\boldsymbol{\alpha}$. ***Alternatively***, instead of plotting $J(\boldsymbol{\theta})$ over the number of iterations to figure out whether Gradient Descent works correctly, it is possible to know when $J(\boldsymbol{\theta})$ converges by using automatic convergence test to declare convergence if $J(\boldsymbol{\theta})$ decreases by less than threshold $\varepsilon=10^{-3}$ in one iteration.

Abnormal behaviour of gradient descent over numerous iterations:



When $J(\boldsymbol{\theta})$ is increasing over numerous iterations of gradient descent, this is a sign that gradient descent is not working correctly. In this case, using a **smaller $\boldsymbol{\alpha}$** help to avoid gradient descent to overshoot the minimum (i.e. the smallest value of $J(\boldsymbol{\theta})$ such as in:



For an appropriate small $\boldsymbol{\alpha}$, $J(\boldsymbol{\theta})$ should decrease on every iteration. However, if $\boldsymbol{\alpha}$ is too small, gradient descent can run very slowly before converging. By contrast, for a too large $\boldsymbol{\alpha}$, $J(\boldsymbol{\theta})$ may not decrease on every iteration and converge.

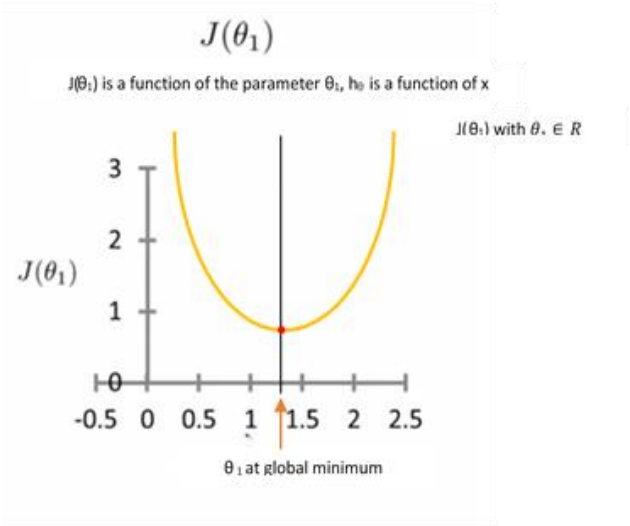
The only way to choose the right value of $\boldsymbol{\alpha}$, is to try to run gradient descent with different value of $\boldsymbol{\alpha}$ such as $\boldsymbol{\alpha = 0.001}$, $\boldsymbol{\alpha = 0.003}$, $\boldsymbol{\alpha = 0.01}$, $\boldsymbol{\alpha = 0.03}$, $\boldsymbol{\alpha = 0.1}$, $\boldsymbol{\alpha = 0.3}$, $\boldsymbol{\alpha = 1}$, and to pick the best learning rate $\boldsymbol{\alpha}$ that enable to find quickly the minimum with optimal value of $\boldsymbol{\theta}$ can fit the model well to the training set**.**

6) *Normal Equation algorithm*

An alternative to Gradient Descent algorithm that solve for optimal value of $\boldsymbol{\theta}$ iteratively is the use of Normal Equation algorithm that solve for optimal value of $\boldsymbol{\theta}$ analytically. Suppose $J(\boldsymbol{\theta})$ as a quadratic function such as:

$$J(\boldsymbol{\theta}) = a\theta^2 + b\theta + c$$

To find the optimal value of $\boldsymbol{\theta}$ that minimises$_{\boldsymbol{\theta}}$ $J(\boldsymbol{\theta})$, $\frac{\boldsymbol{d}}{\boldsymbol{d\theta_j}}\boldsymbol{J(\theta)}$ **= 0** need to be solved (*for every j*).



<u>In gradient descent algorithm</u>,

Knowing that $J(\boldsymbol{\theta}) = \frac{1}{2m}\sum_{i=1}^{m}(h_\theta(x^{(i)}) - y^{(i)})^2$ and $\boldsymbol{\theta} \in \boldsymbol{R^{n+1}}$,

$\frac{\boldsymbol{\partial}}{\boldsymbol{\partial\theta_j}}\boldsymbol{J(\theta)}$ is set to **0** (*for every j*) in order to solve for $\theta_0, \theta_1, \ldots, \theta_n$

In Normal Equation,

Having the following dataset,

Index j of variable/features	j=1	j=2	j=3	j=4	
Index i of training examples	Size in feet^2 (x_1)	Number of Bedrooms (x_2)	Number of floors (x_3)	Housing Age (years) (x_4)	Price in \$ (**y**)
i=1	2014	5	1	45	460
i=2	1490	2	2	40	244
i=3	900	2	2	35	190
i=4	540	1	1	36	…

The x_0 column will be added such as

Index j of variable/features	j=0	j=1	j=2	j=3	j=4	
Index i of training examples	X_0	Size in feet^2 (x_1)	Number of Bedrooms (x_2)	Number of floors (x_3)	Housing Age (years) (x_4)	Price in \$ (**y**)
i=1	1	2014	5	1	45	460
i=2	1	1490	2	2	40	244
i=3	1	900	2	2	35	190
i=4	1	540	1	1	36	50

So that,

The design matrix $X = \begin{bmatrix} 1 & 2014 & 5 & 1 & 45 \\ 1 & 1490 & 2 & 2 & 40 \\ 1 & 900 & 2 & 2 & 35 \\ 1 & 540 & 1 & 1 & 36 \end{bmatrix}$ *and* $y = \begin{bmatrix} 460 \\ 244 \\ 190 \\ 50 \end{bmatrix}$

m * (n+1) dimensional matrix m-dimensional vector

In more details,

For *m* **example** $(x^{(1)}, y^{(1)}), \dots, (x^{(m)}, y^{(m)})$; *n* **features,**

$$x^{(i)} = \begin{bmatrix} x_0^{(i)} \\ x_1^{(i)} \\ \dots \\ x_n^{(i)} \end{bmatrix} ; \ (x^{(i)})^T = [x_0^{(i)} \ x_1^{(i)} \ \dots \ x_n^{(i)}] \ ; \ X = \begin{bmatrix} -(x^{(1)})^T - \\ -(x^{(2)})^T - \\ \dots \\ -(x^{(m)})^T - \end{bmatrix} ; y = \begin{bmatrix} y^{(1)} \\ y^{(2)} \\ \dots \\ y^{(m)} \end{bmatrix}$$

m * (n+1) dimensional matrix

As a concrete example,

If $x^{(i)} = \begin{bmatrix} x_0^{(i)} = 1 \\ x_1^{(i)} \end{bmatrix}$; $(x^{(i)})^T = [1 \ x_1^{(i)}]$; $X = \begin{bmatrix} 1 & x_1^{(1)} \\ 1 & x_2^{(1)} \\ \dots & \dots \\ 1 & x_m^{(1)} \end{bmatrix}$

m * 2 dimensional matrix

Finally, the optimal value of θ that minimises$_\theta$ $J(\theta)$ can be found by computing

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

N.B: *The Matlab function that inverse a matrix is* ***inv()****.*

Theta = *inv*(X'*X)*X'*y

If using Normal Equation instead of Gradient Descent, applying feature scaling is not necessary.

Issue: when is it more appropriate to use Normal Equation instead of Gradient Descent?

	Normal Equation	**Gradient Descent**	
Advantage	- No need to choose $\boldsymbol{\alpha}$ - Don't need to iterate (i.e. do not need to plot $J(\boldsymbol{\theta})$, wait for it to converge, etc..)	- Need to choose $\boldsymbol{\alpha}$ - Need to iterate	Disadvantage
Disadvantage	-Need to compute $(\boldsymbol{X^T X})^{-1}$ (*n*n matrix*). Computing an inverse matrix cost roughly the cube of the dimension of the matrix (~ O(n^3)). -Very slow if *n* is very large. Better to use when *n is small (*<10000)	-Works well even when *n* is large (i.e. 1,000,000)	Advantage

Issue: What if $\boldsymbol{X^T X}$ is non-invertible (i.e. singular/degenerate)?

Non-invertible $\boldsymbol{X^T X}$ matrices occur rarely, but if this happens, there are usually two causes for that:

- Redundant features (linearly dependent)
  E.g. for $\boldsymbol{x_1} = size\ in\ feet^2$ and $\boldsymbol{x_2} = size\ in\ m^2$, $\boldsymbol{x_2} = 3.28 * \boldsymbol{x_1}$
  → Solution: Delete redundant features
- Too many features (e.g. $m \geq n$)
  → Solution: Delete useless features to have $\boldsymbol{m \geq n}$, or use **regularization** techniques (*see the chapter in Overfitting*)

III. Polynomial regression

Choosing different features results in several learning algorithm. Polynomial regression enable the use of the machinery of linear regression to fit very complicated (*sometimes non-linear*) functions by creating interesting features.

1) *Example of Classification problem*

Assuming two features ($\boldsymbol{x_1}$= *frontage,* $\boldsymbol{x_2}$ =*depth*) such as:

$$h_\theta(x) = \theta_0 + \theta_1 * frontage + \theta_2 * depth$$

With the frontage is defined as the width of the property and the depth is defined as the length of the house.

When running linear regression, instead of using $\boldsymbol{x_1}$= *frontage,* $\boldsymbol{x_2}$ =*depth*, it is possible to create a third feature $\boldsymbol{x_3}$ from $\boldsymbol{x_1}, \boldsymbol{x_2}$ such as:

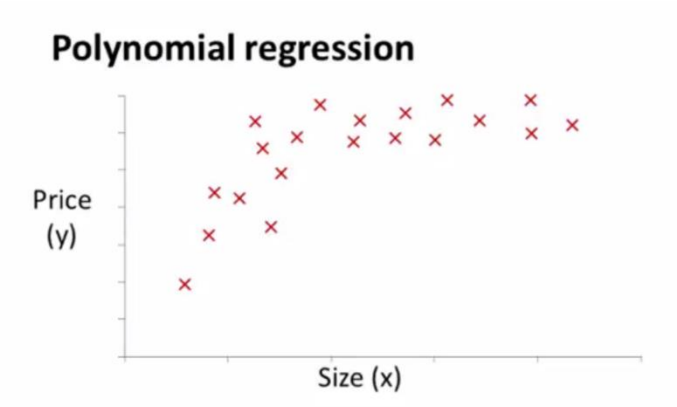
$$\boldsymbol{x_3 = Area = frontage * depth = x_1 * x_2}$$

$$\therefore h_\theta(x) = \theta_0 + \theta_1 * Area$$

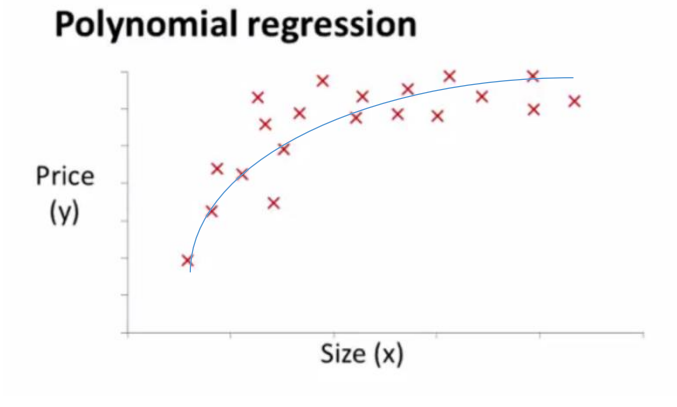
$$h_\theta(x) = \theta_0 + \theta_1 x_3$$

Creating new features enable to end up with better model sometimes.

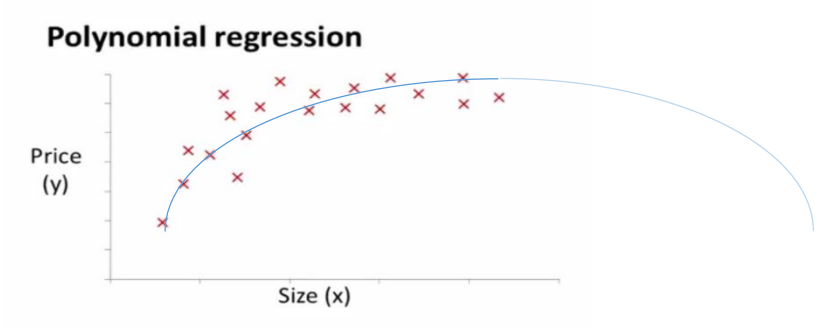
Closely related to choosing better features, let look at polynomial regression:



To fit to this training set (*above*) well, a quadratic model might be used: $h_\theta(x) = \theta_0 + \theta_1 x + \theta_2 x^2$

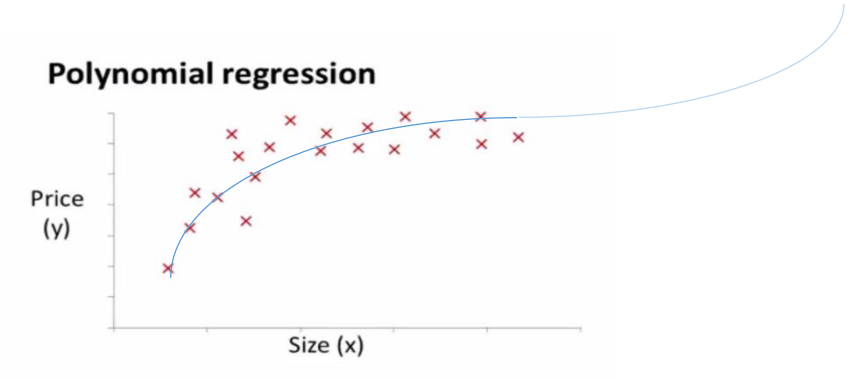


However a quadratic function might not make sense since it usually have a “bowl shape”. The housing price will go down when the size of the house increase too much.



Then, maybe a cubic function $h_\theta(x) = \theta_0 + \theta_1 x + \theta_2 x^2 + \theta_3 x^3$ might fit to the training set better, in which the housing price continue to increase when the housing size increase:

Issue: How can a model like a cubic function $h_\theta(x) = \theta_0 + \theta_1 x + \theta_2 x^2 + \theta_3 x^3$ be implemented to the training set more easily? Using the machinery of multivariate regression, it is possible to apply at simple modification to this cubic function:



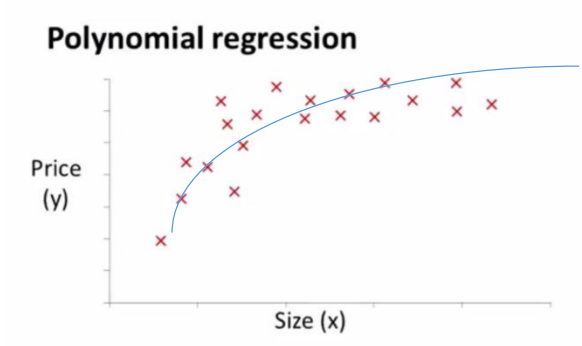
From the machinery of multivariate regression $h_\theta(x) = \theta_0 + \theta_1 x + \theta_2 x_2 + \theta_3 x_3$,

We can choose $x = (size)$, $x_2 = (size)^2$, $x_3 = (size)^3$, such as

$$h_\theta(x) = \theta_0 + \theta_1 (size) + \theta_2 (size)^2 + \theta_3 (size)^3$$

Where the range of $size$: $1 - 1000$, of $(size)^2$: $1 - 1,000,000$, of $(size)^3$: $1 - 1,000,000,000$, which might cause a problem when implementing the model. In this case, it is important to use **feature scaling and mean normalization.**

Similarly, in the case of trying to fit a quadratic function $h_\theta(x) = \theta_0 + \theta_1 (size) + \theta_2 (size)^2$ the housing price dataset in which the housing price will go down when the size of the house increase too much, it would be better to use a root function $h_\theta(x) = \theta_0 + \theta_1 (size) + \theta_2 \sqrt{(size)}$ in which the fitted model converge such as:



Feature can be automatically selected by using algorithm such as Principal Component Analysis algorithm. For more details, please refer to http://www.jmlr.org/papers/volume3/guyon03a/guyon03a.pdf.