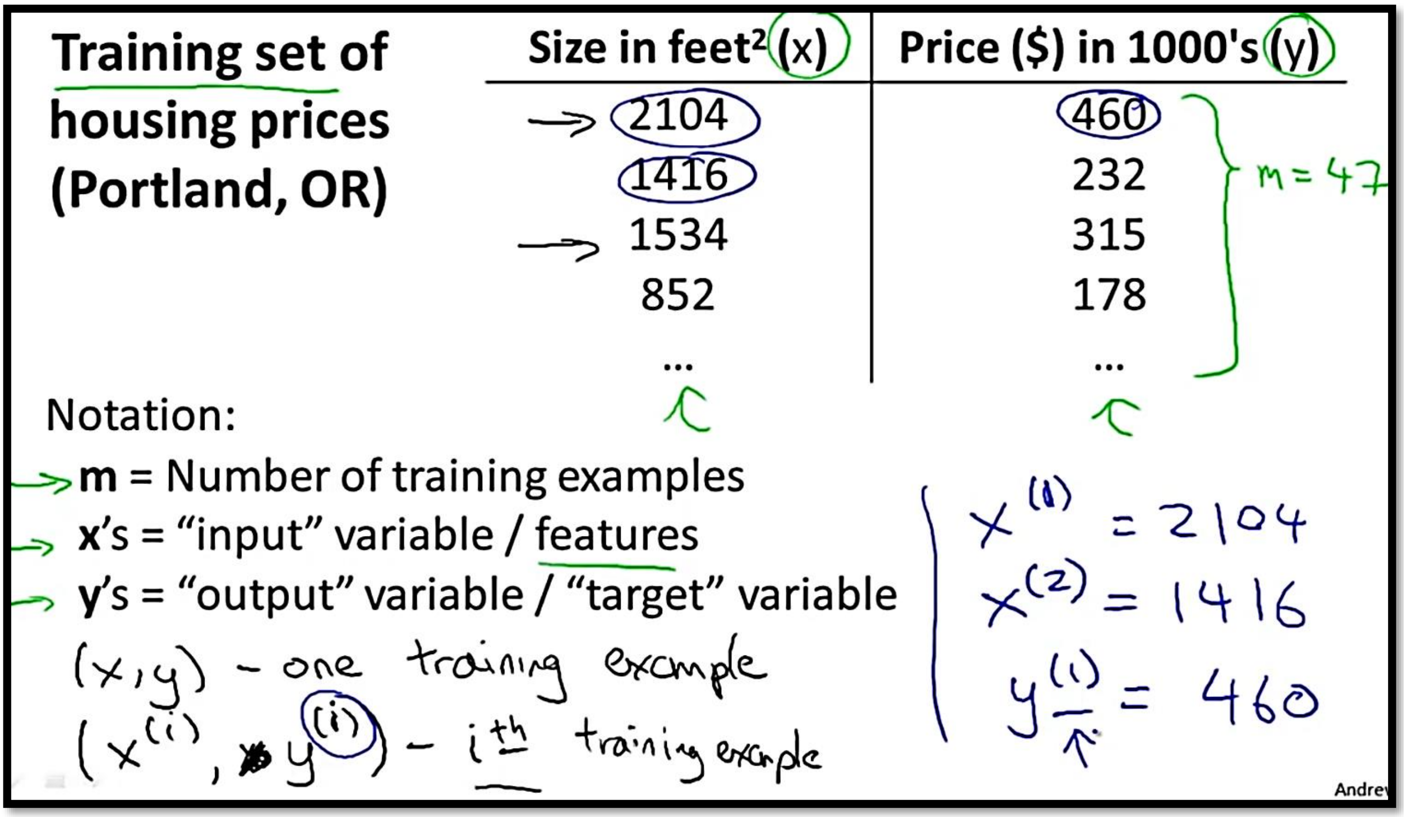
2. Linear Regression with One Variable – or Univariate Linear Regression

⇨ MODEL REPRESENTATION:



➢ **Other Notations:**

X = space of input values

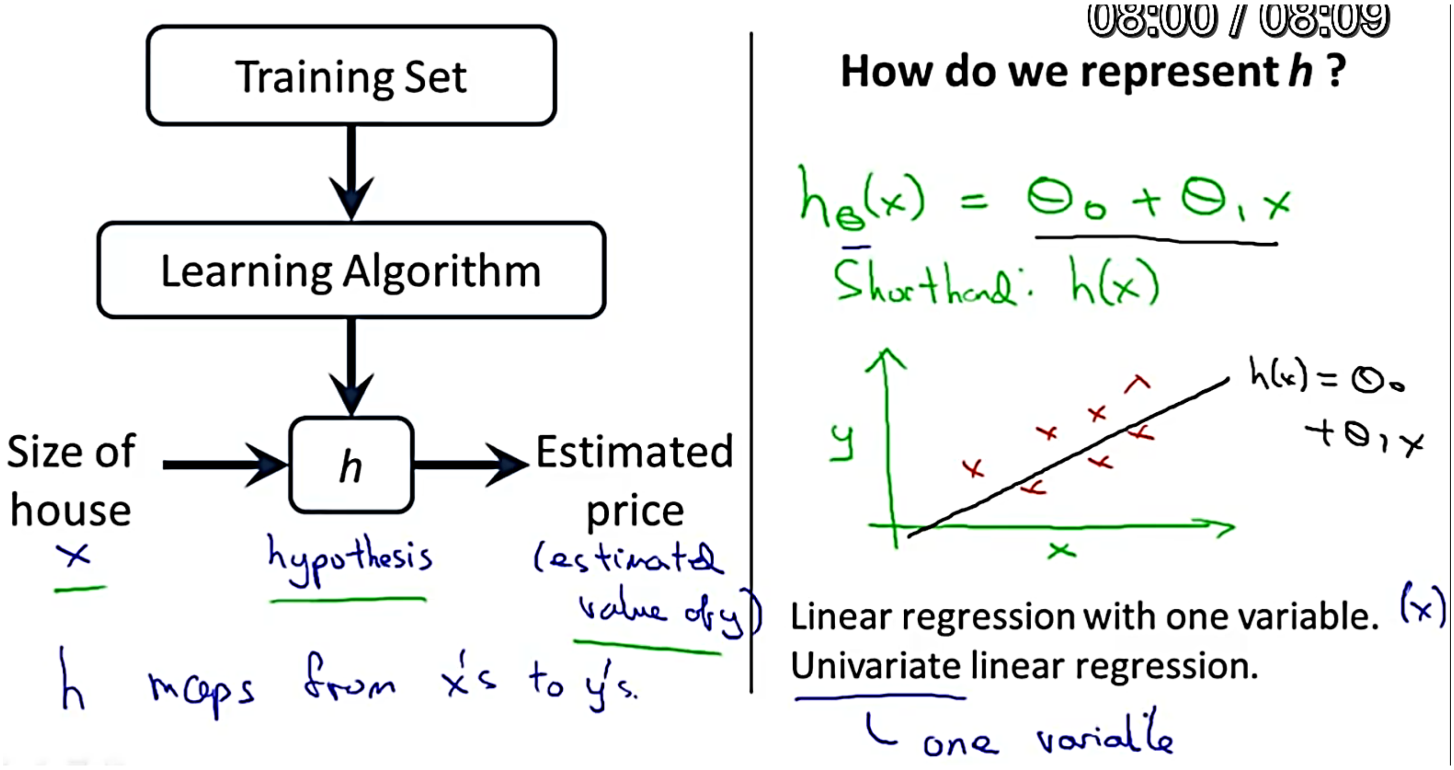
Y = space of output values

Dataset = list of m training examples **→ $(x^{(i)}, y^{(i)})$;** i = 1,2, . . . ,m

---

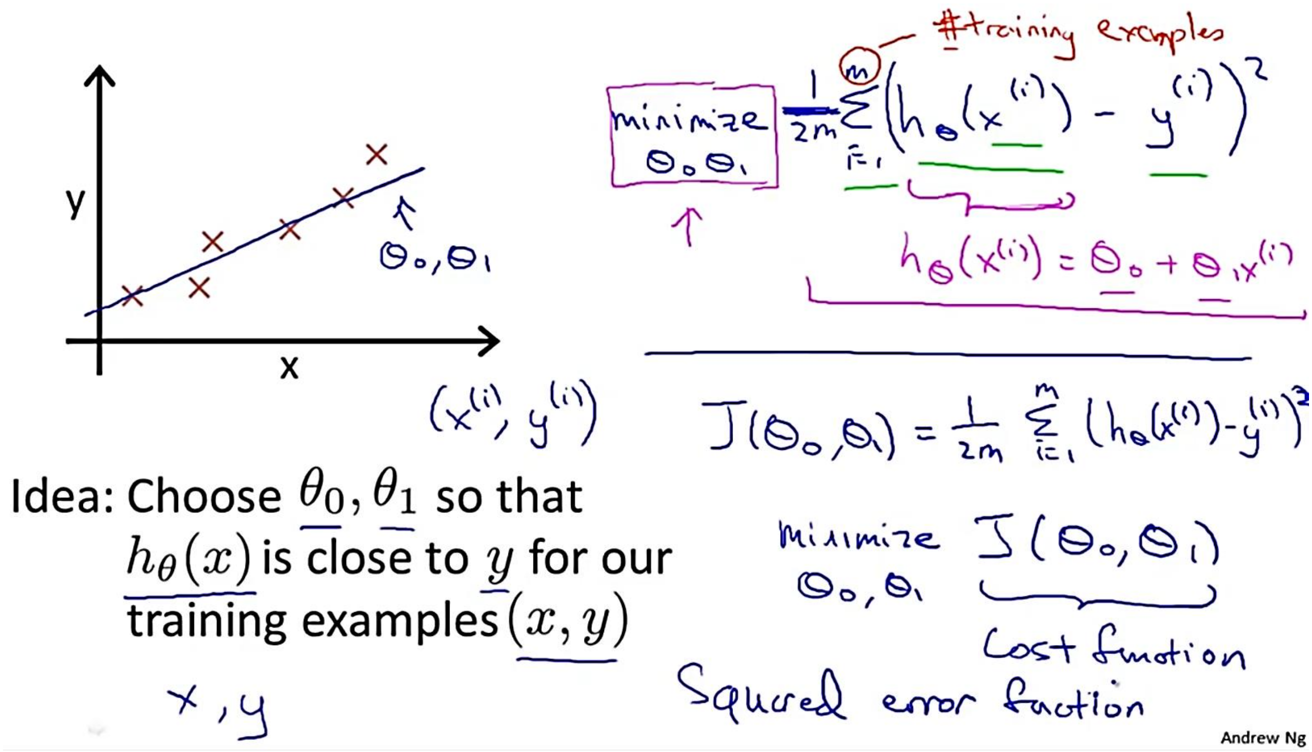
Hypothesis Function: Function which is derived by feeding training data (Input and Output (Supervised)) to the learning algorithm, which can then be used to predict o/p for new input data.

- For a supervised problem: **h : X → Y** so that h(x) is a "*good*" predictor for the corresponding value of y.



COST FUNCTION: Can measure the **accuracy** of our hypothesis function by **Choosing parameters** of h(x) such that **h(x) is close to y** for data in the training set.

Cost function is a **minimization** function:



Here, we try to **minimize** the *(1/2m) x (sum of squared differences b/w predicted value and actual value given in dataset).*

- **"J"** is the cost function or **squared error cost function** or **Mean squared error**
- **(1/2m)** is for averaging the squared difference.

Minimize means, we try to find values of **θ** parameters such that the cost function is minimized.

Cost ➔ An **average difference** of all the results of the hypothesis with inputs from x's and the actual output y's.

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

To break it apart, it is $\frac{1}{2}\bar{x}$ where $\bar{x}$ is the mean of the squares of $h_\theta(x_i) - y_i$, or the difference between the predicted value and the actual value.

- The mean is **halved** as a convenience for the computation of the **gradient descent**, as the derivative term of the square function will cancel out the 1/2 term.

COST FUNCTION INTUITION: Training set data is scattered on x-y plane. We try to draw a straight line through it. **Goal → find best fitting line.**

Ideally, the line should pass through all the points of our training data set. In such a case, the value of "J" will be 0.

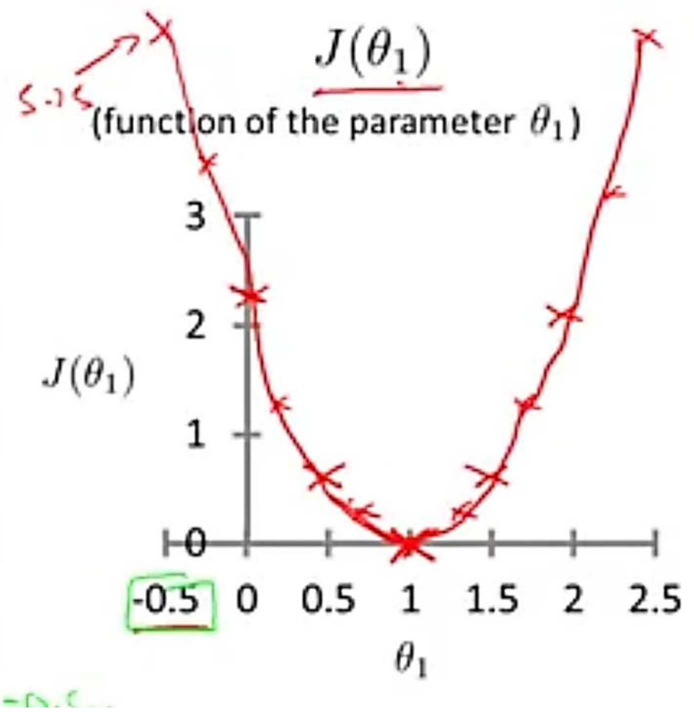
- For simplicity: let $h_\theta(x) = \theta_1 x$

$h_\theta(x)$	$J(\theta_1)$
(for fixed θ_1, this is a function of x)	(function of the parameter θ_1)

For Θ=1 ➔ h(Θ)=x ➔ J(Θ)=0

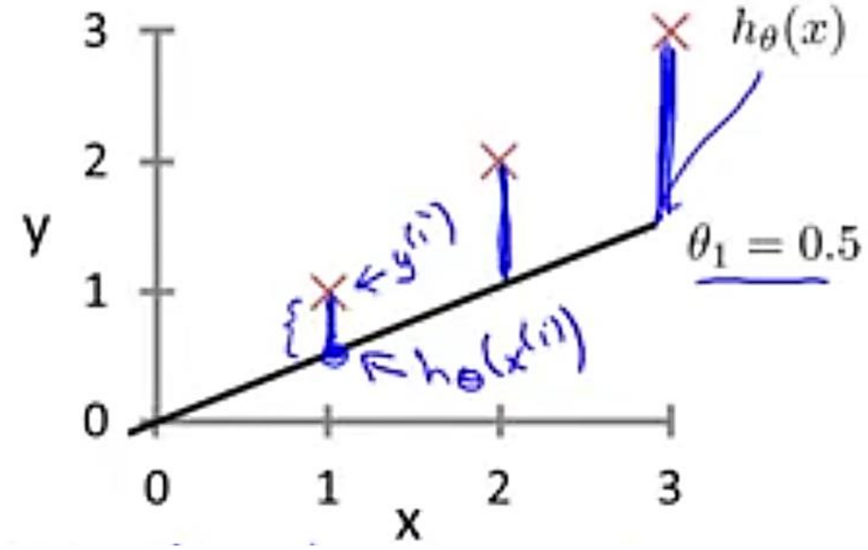
For Θ=0.5 ➔ h(Θ)=0.5x ➔ J(Θ)=0.58

For Θ=0 ➔ h(Θ)=0 ➔ J(Θ)=2.3



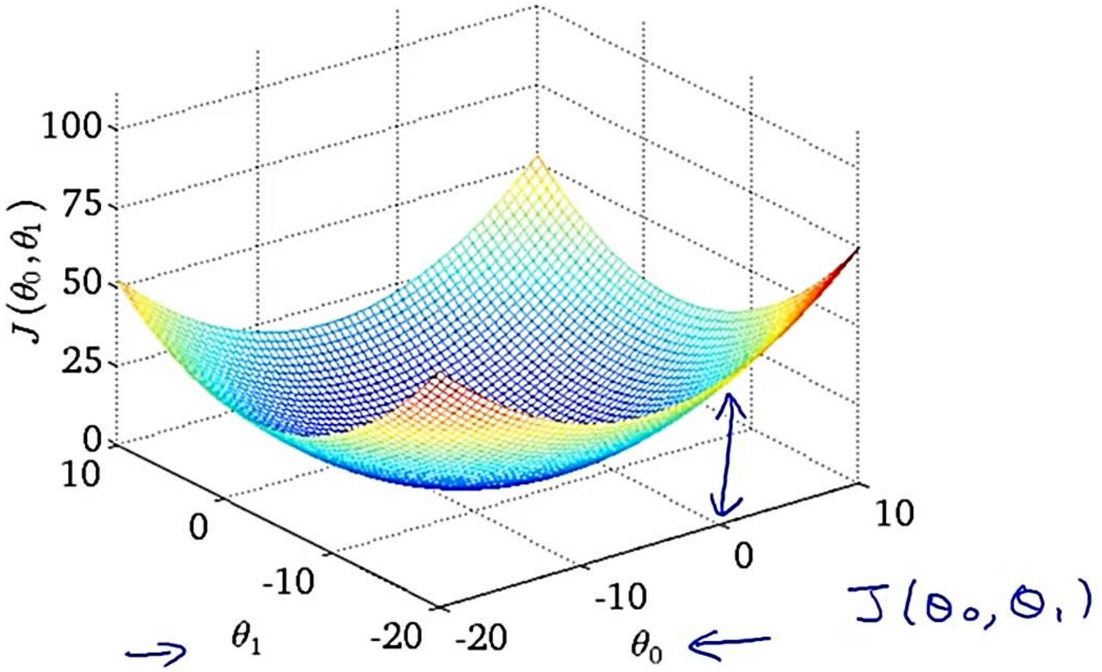
- J(Θ) is the average of sqr of diff bw h(x) and y:
- h(x) = **predicted** value at given training data
- y = **actual** value of o/p in training data

- **Vertical lines** represent the given **difference**



- For different values of Θ, we try to minimize **J(Θ)** {**error**}, which occurs at Θ=1. Therefore, we choose Θ=1 as out best fitting curve: h(x)=x

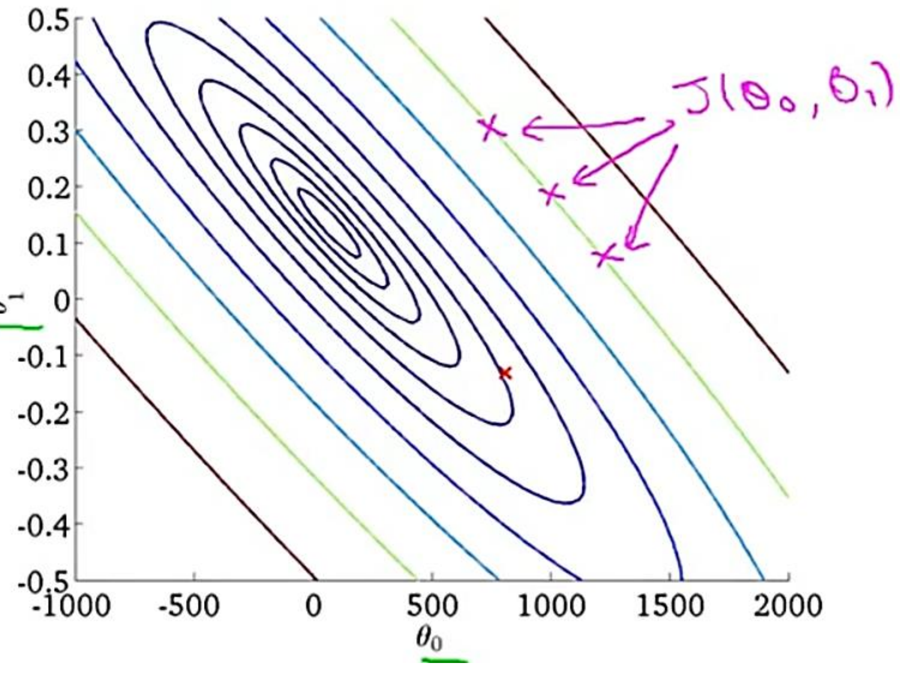
≫ For more **complex** h(x) fxn like **h(x) = Θ_0 + Θ_1x** : we have to plot **J(Θ_0, Θ_1)** in **3D**. As, for different combinations of Θ_0 and Θ_1, J can be different.



These can be more easily represented using **contour figures**: 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 on the line.

$$J(\theta_0, \theta_1)$$

(function of the parameters θ_0, θ_1)

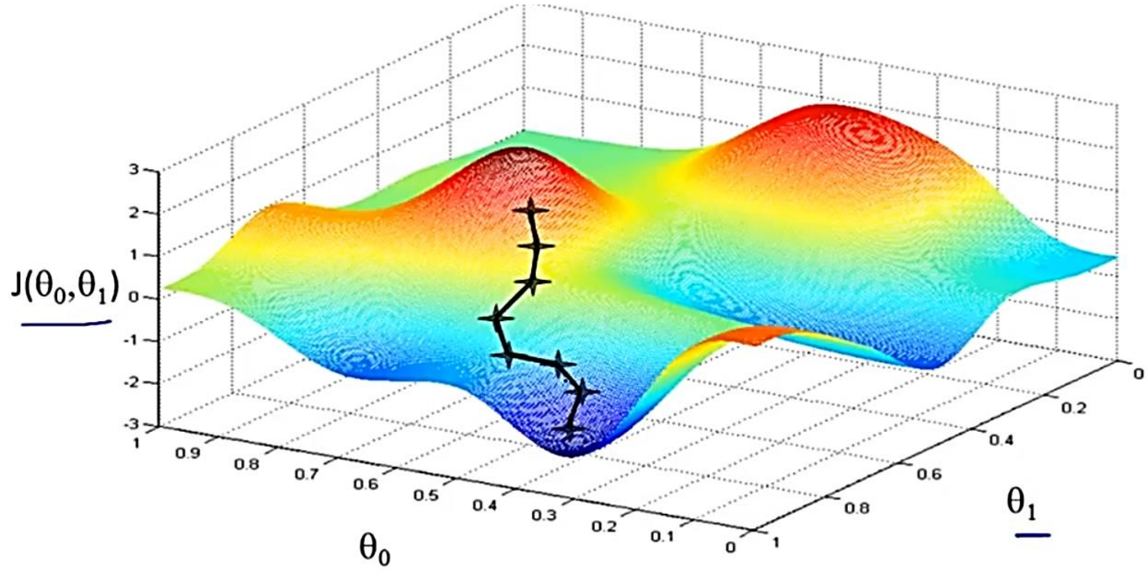


Graph b/w Θ_0 and Θ_1: these **ellipses** are the combinations of Θ_0 and Θ_1 for which value of J is same. Points other than on ellipses are also valid points, they also correspond to a unique value of J.

- The **best** combination (one which **minimizes** $J(\Theta_0, \Theta_1)$) of Θ_0 and Θ_1 lies around **center** of the innermost circle.

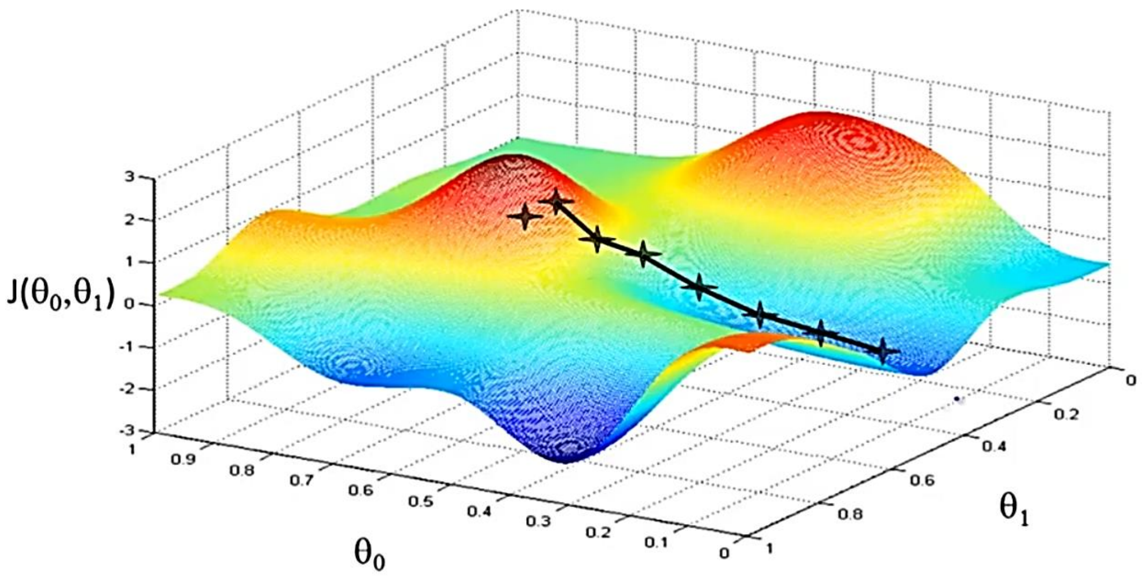
⇨ **GRADIENT DESCENT:** an algorithm to minimize $J(\Theta_0, \Theta_1)$

- Start with some Θ_0, Θ_1
- Keep changing Θ_0, Θ_1 to reduce J until minimum is reached



Here, we start at a value of Θ_0, Θ_1 and keep going down on $J(\Theta_0, \Theta_1)$ curve until we reach a **local minima**. We will know that we have succeeded when our cost function is at the very bottom of the pits in our graph.

- If we start at a diff value of Θ_0, Θ_1, we end having different minima.



We are not graphing x and y itself, but the parameter range of our hypothesis function and the cost resulting from selecting a particular set of parameters.

Gradient descent algorithm

repeat until convergence {

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

}

The **slope of the tangent** is the **derivative** at that point and it will give us a **direction** to move towards. We make steps down the cost function in the direction with the steepest descent. The size of each step is determined by the parameter **α**, which is called the **learning rate**.

α = Learning rate

SIMULTANEOUS UPDATE: first we calculate new value for both Θ_0 and Θ_1, then only we update their values. So order of execution of statements is:

Correct: Simultaneous update

$$\text{temp0} := \theta_0 - \alpha \frac{\partial}{\partial \theta_0} J(\theta_0, \theta_1)$$
$$\text{temp1} := \theta_1 - \alpha \frac{\partial}{\partial \theta_1} J(\theta_0, \theta_1)$$
$$\theta_0 := \text{temp0}$$
$$\theta_1 := \text{temp1}$$

Incorrect:

$$\text{temp0} := \theta_0 - \alpha \frac{\partial}{\partial \theta_0} J(\theta_0, \theta_1)$$
$$\theta_0 := \text{temp0}$$
$$\text{temp1} := \theta_1 - \alpha \frac{\partial}{\partial \theta_1} J(\theta_0, \theta_1)$$
$$\theta_1 := \text{temp1}$$

Here, if update Θ_0 before actually calculating the value of new Θ_1, the Θ_0 used in equation of Θ_1 will be new Θ_0, not the one we wanted to minimize for.

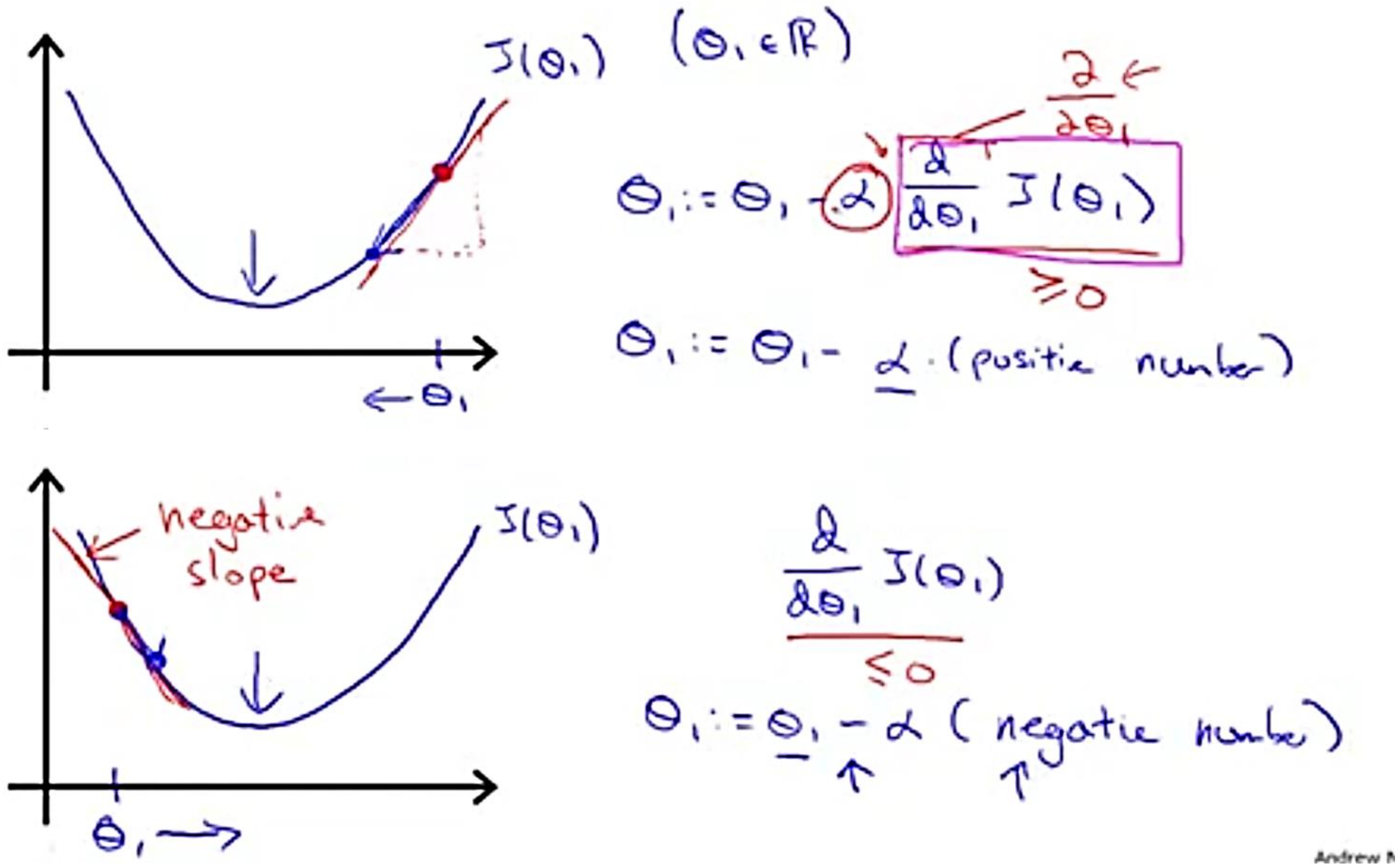
At each iteration j, one should simultaneously update the parameters .. Θ_0, Θ_1…Θ_n. Updating a specific parameter prior to calculating another one on the j^{th} iteration would yield a wrong implementation.

GRADIENT DESCENT INTUITION: for simplicity we only use one parameter:

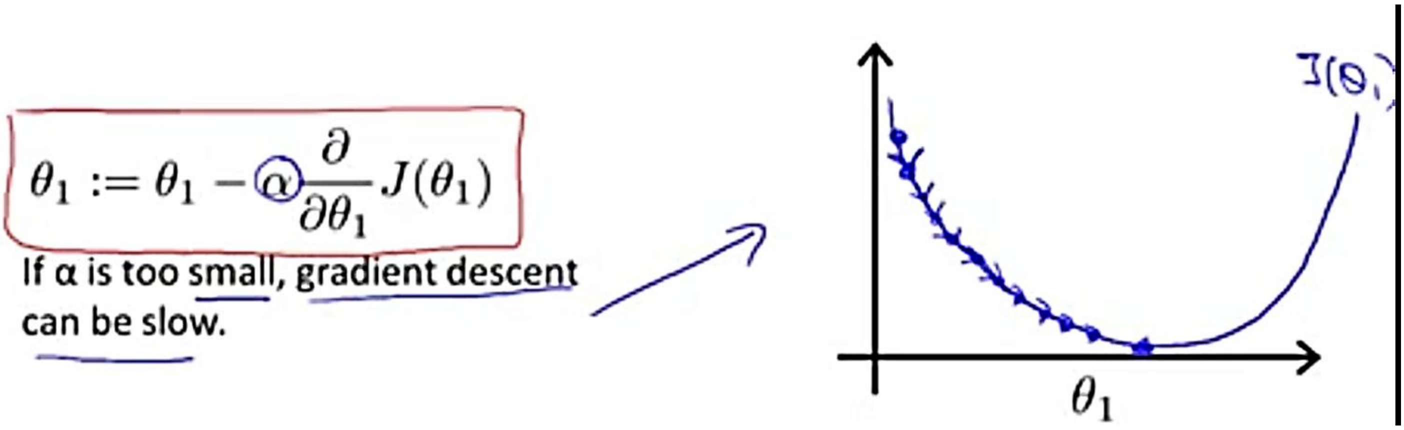
$h(x) = \Theta_1.x$

→α is positive

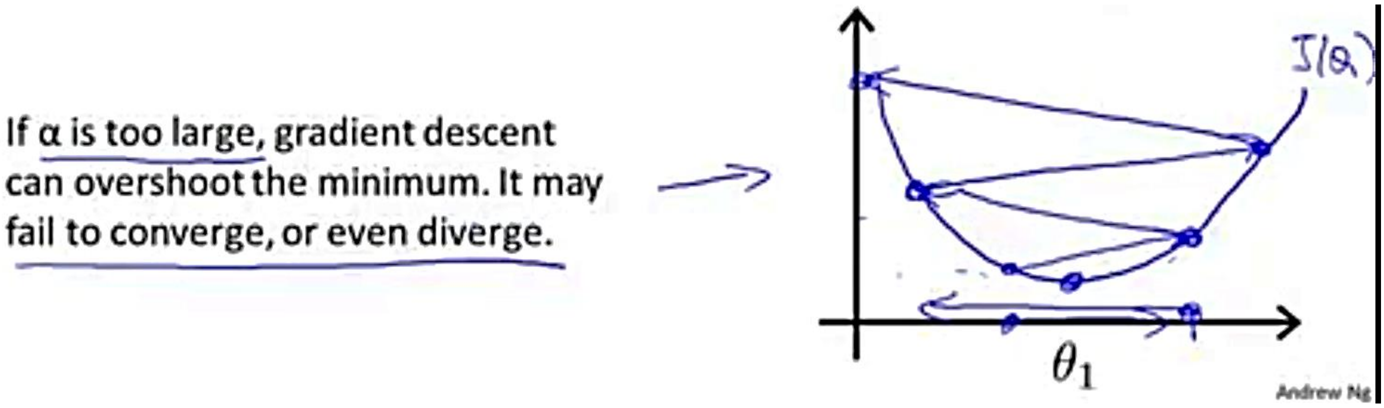
- ⇨For a value of Θ_1, if the **slope** of J(Θ) is **positive**: Θ_1 decreases
- ⇨For **negative slope** of J(Θ): Θ_1 increases
- ⇨θ_1 eventually **converges** to its minimum.



- ⇨**If** the value of **α is too small**: gradient descent takes baby steps towards the min.



⇨ **If α is too large**: gradient descent takes huge steps.. in such case the gradient descent may even **overshoot the min** if the diff b/w initial Θ and Θ_{min} is less than the value of **jump in Θ (α * derivative of J)** and it may start going further and further from the min.



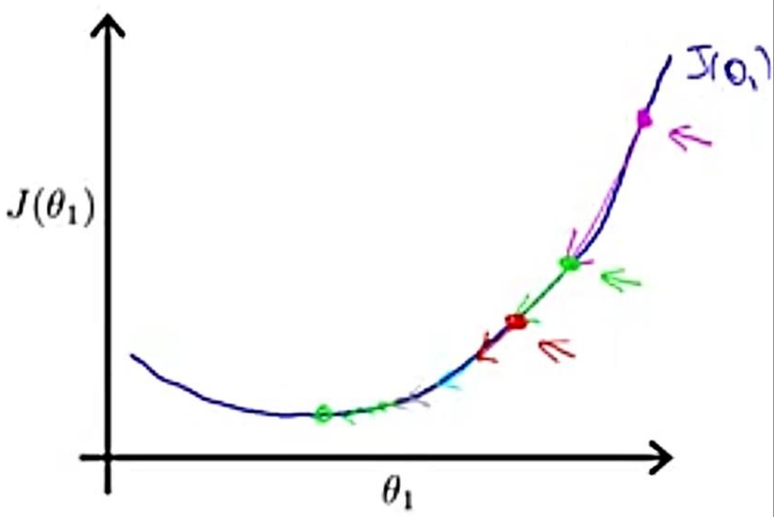
- Therefore, we should adjust our parameter to ensure that the gradient descent algorithm **converges** in a **reasonable time**.
- If the Θ is already at its **local minimum**, the **slope will be 0**.. thus Θ won't change

⇨ Even if the learning rate **α is fixed**, the **slope gets smaller as we reach** towards the **minima**.. so the steps automatically become smaller

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

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

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



⇨For a **linear regression** model: **Derivative of J**:

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

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

repeat until convergence: {

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

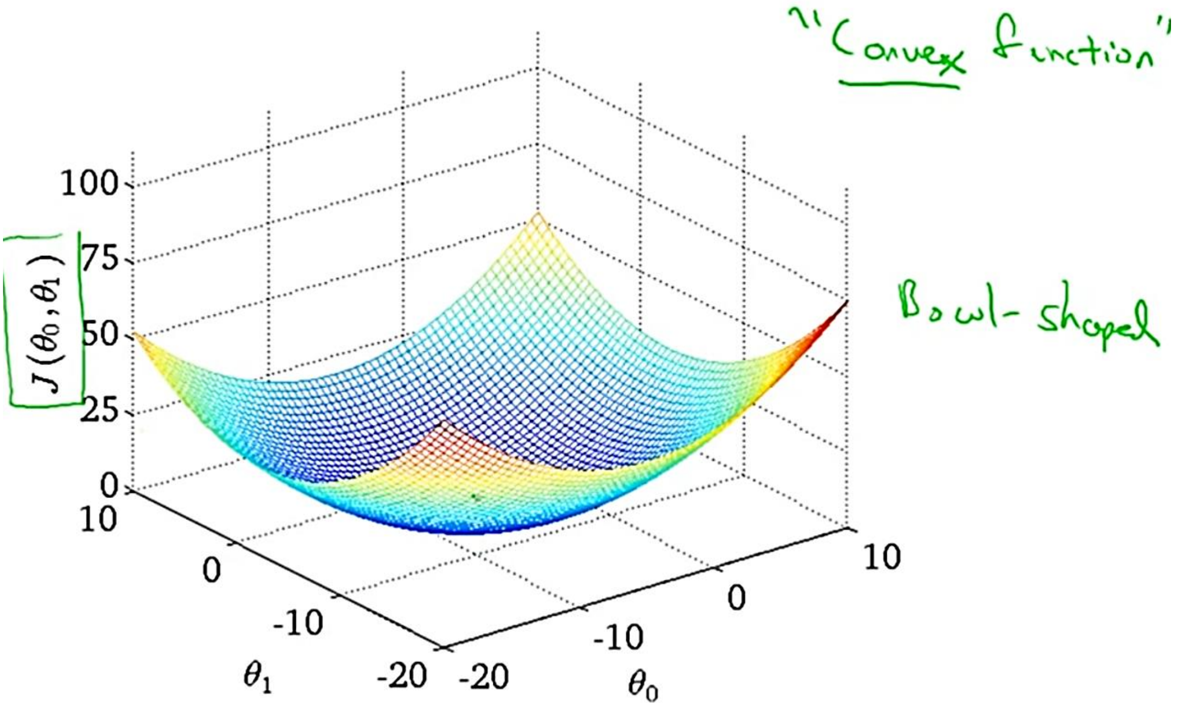
$$\theta_1 := \theta_1 - \alpha \frac{1}{m} \sum_{i=1}^{m} ((h_\theta(x_i) - y_i) x_i)$$

}

⇨**For Θ_0** – derivate wrt Θ_0

⇨**For Θ_1** – derivate wrt Θ_1

- For a **linear regression** model: the curve is always a **convex curve** (bowl shaped).
  It has only **one optimum** → **global minima** (assuming the learning rate α is not too large).

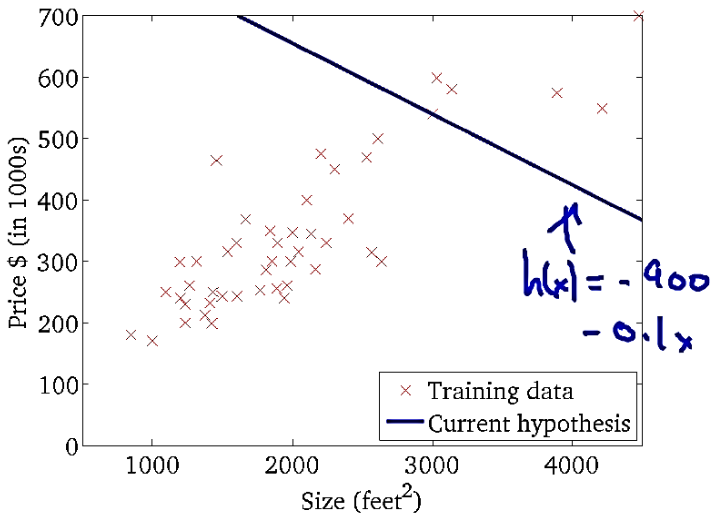


- In the contour curve: we start of with any value of Θ_0 and Θ_1 and then we minimize the J.
- **We approach the min as we reach towards the center**.

We start at an **arbitrary** value for **Θ_0 and Θ_1:**

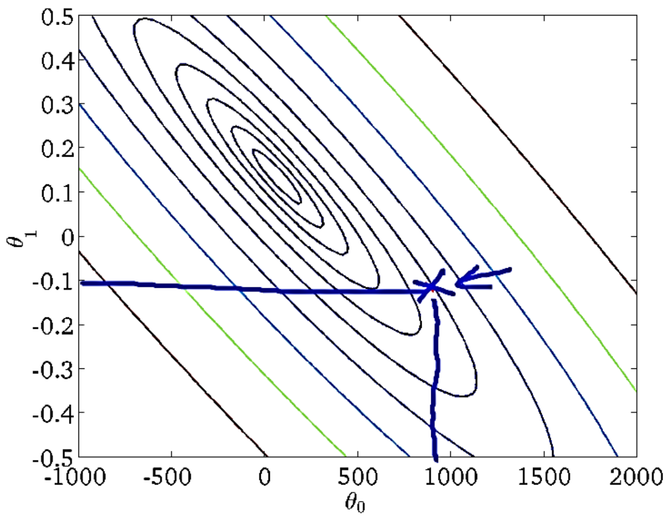
$h_\theta(x)$

(for fixed θ_0, θ_1, this is a function of x)

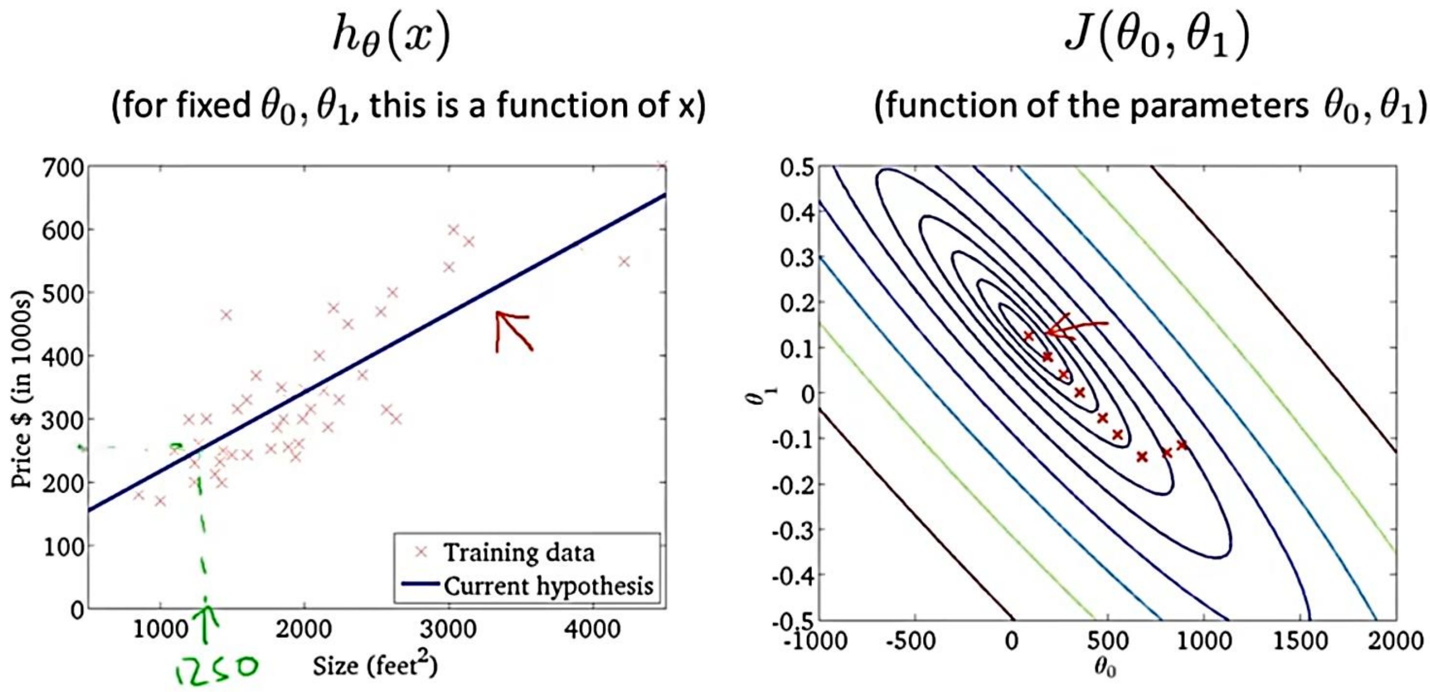


$J(\theta_0, \theta_1)$

(function of the parameters θ_0, θ_1)



We start minimizing $J(\Theta_0, \Theta_1)$ with our gradient descent algo:



J is a complicated quadratic function

The ellipses shown above are the **contours** of a quadratic function

Batch Gradient Descent → Each step of gradient descent uses all training examples.

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**.
