Linear Regression.

A number of slides and screenshots from : Andrew Ng's course on machine learning and Sebastian Raschka's course on deep learning
Both can be found for free on youtube

What does regression mean ?

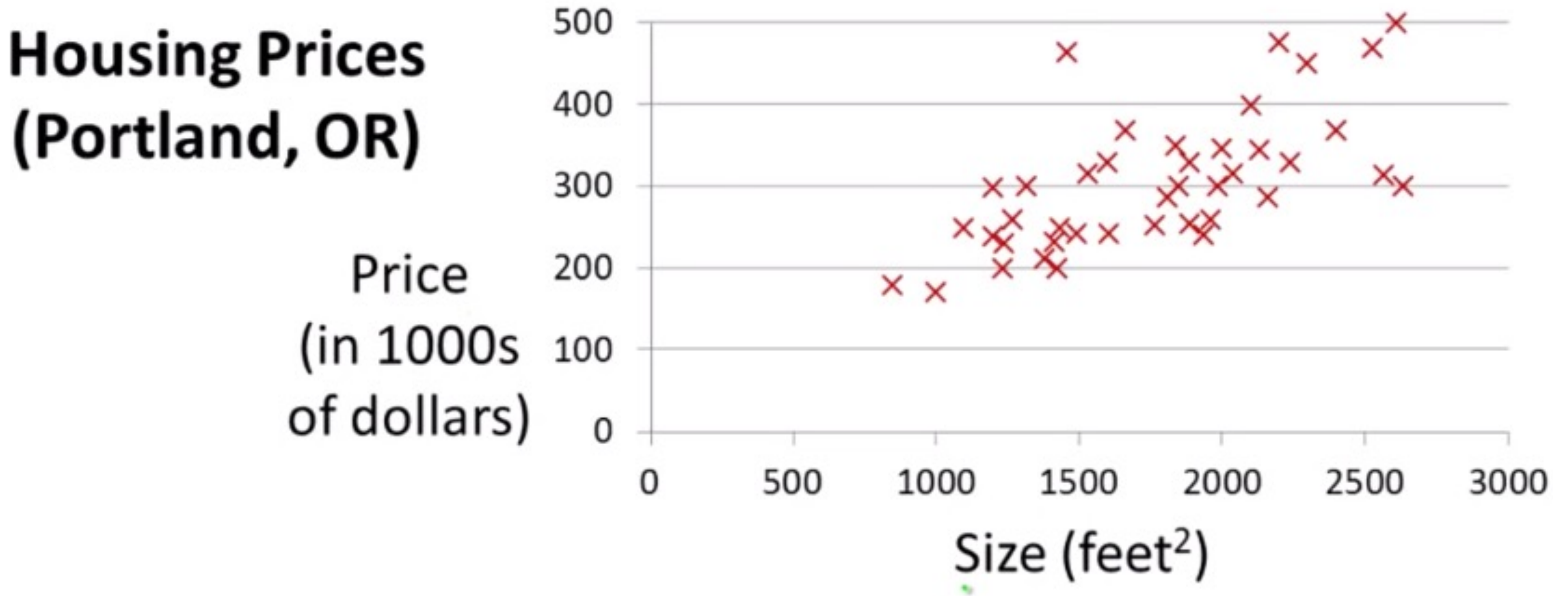
- Seen in intro, but :
- Regression means predictiong real-valued outputs.

- An essential type of supervised machine learning task : for each exmaple in the data, we want to get as close as possible to the real-valued label.
- Often contrasted with classification (discrete labels).

- Example :
  - Predicting height => many many real-valued outputs are possible…
  - Vs. Predicting a « height class » : short | medium-height | tall

Dataset and problem example

- Imagine we want to create an ML algorithm to predict the price of a house, using only as information the size of the house. This is the dataset we can use to train our algorithm.



Training Set and Notation

Training set of housing prices (Portland, OR)

Size in feet^2 (x)	Price ($) in 1000's (y)
2104	460
1416	232
1534	315
852	178
…	…

Notation:

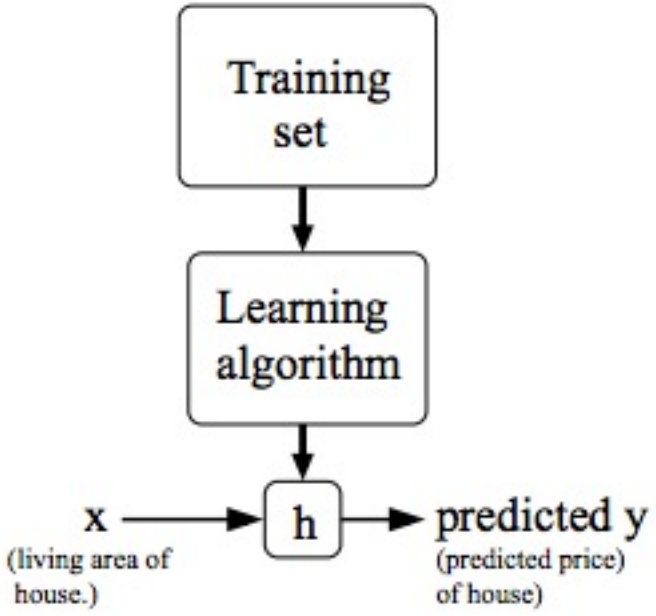
m = Number of training examples
x's = "input" variable / features
y's = "output" variable / "target" variable

The supervised learning workflow

- h: hypothesis
- h is a function which maps x's to y's
- Our goal will be to find the function which takes

x as input and predicts the correct y for that

x.

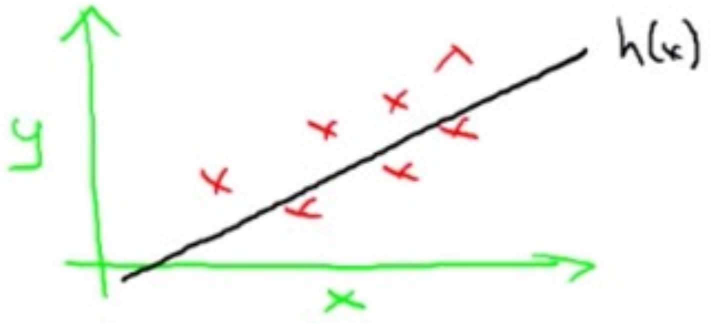


Model function h

- To start with, we will use a simple model, a function which is the equation of a line (maybe you remember y = ax + b from school ?)

$$h(x) = \theta_0 + \theta_1 x$$

- This model will predict that y is some linear (straight line) function :



If this seems a bit odd to you…

- Remember we want our function to predict the examples we have in our training set correctly,
- which our simple model will probably not do very well….

- What if we can’t get to all the points using a straight line ?

- Don’t worry for now, this is still a good starting point !

Cost function

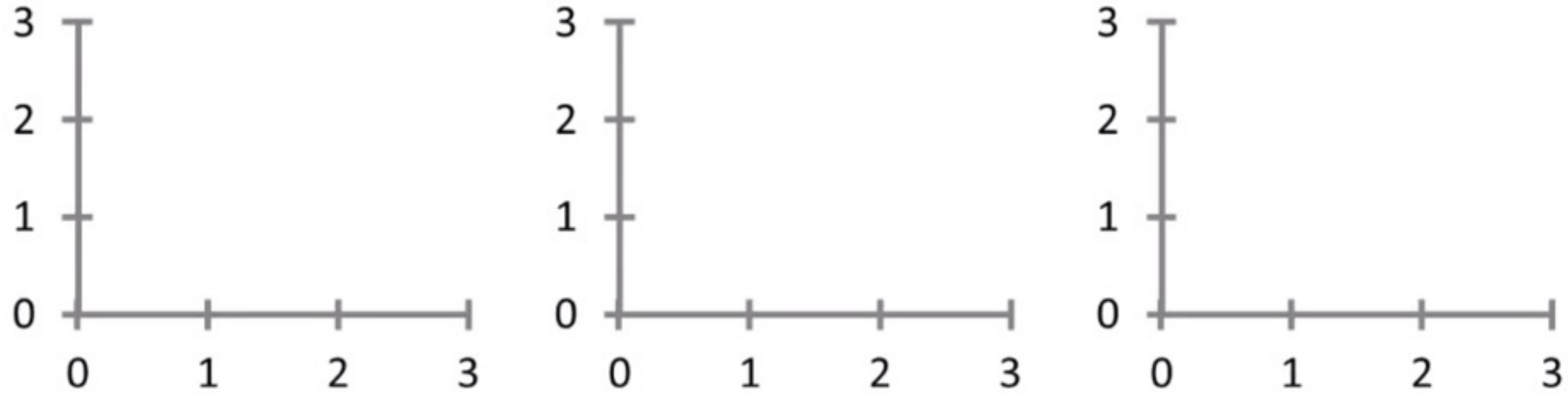
- This is a **second** function we will use to judge how well our straight line fits the data and to find the best possible straight line.

- $h(x) = \theta_0 + \theta_1 x$ => our model

- θ_i are what we call **parameters** and we want to find the right combination of those parameters to get the best line.
- So **how do we choose the right parameters** ?

Different parameter choices/hypotheses

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



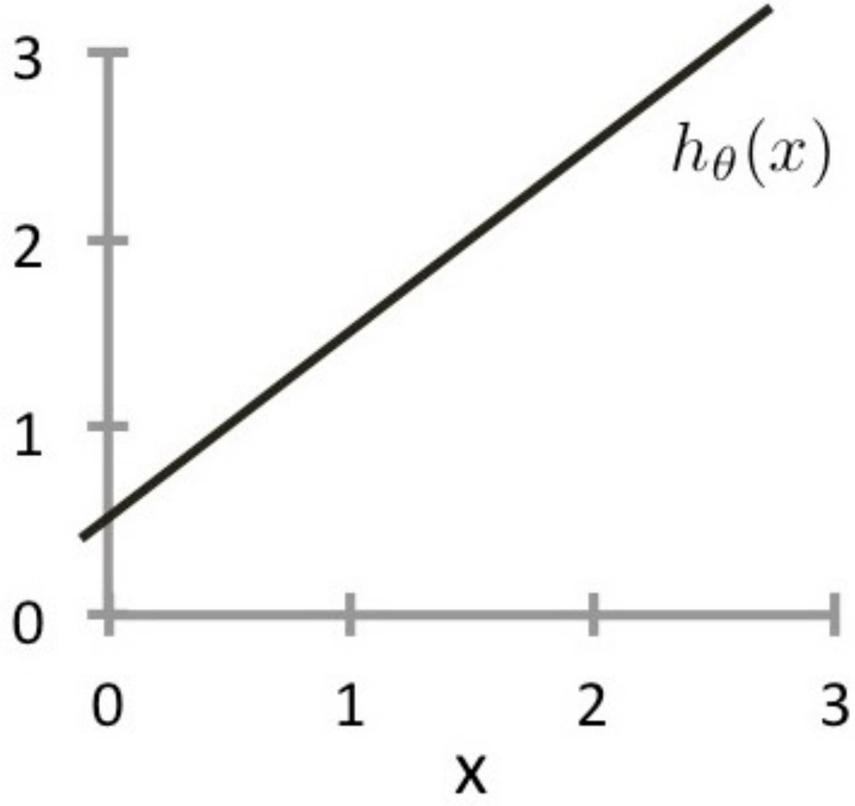
$\theta_0 = 1.5$
$\theta_1 = 0$

$\theta_0 = 0$
$\theta_1 = 0.5$

$\theta_0 = 1$
$\theta_1 = 0.5$

Exercise

- Look at the plot of $h(x) = \theta_0 + \theta_1 x$



- What are the values of θ_0 and θ_1 ?

Minimization Problem

- We want to choose θ_0 and θ_1 so that
- $h(x)$is close to y for out training examples (x, y)…
- So this is actually a **minimization problem**,
- where we want to minimize $(h(x) - y)^2$ for example, by tweaking our parameters θ_0 and θ_1

Cost function = Quantifying the model's error

- For all of our examples m the average error is :

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

The 2 is just there to make the math easier but doesn't change anything fundamentally, you can regard this as the average error.

- This function is known as the MSE (we'll see how it works in a few slides) and is the most commonly used:

Mean Squared Error

To recap

Hypothesis:

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

Parameters:

$$\theta_0, \theta_1$$

Cost Function:

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

Goal: $\underset{\theta_0, \theta_1}{\text{minimize}}\ J(\theta_0, \theta_1)$

https://www.coursera.org/learn/machine-learning/home/welcome

Cost Function Intuition

- Let’s use a simplified model hypothesis to understand what’s going on:

$$h(x) \ = \theta_1 x$$

- Our objective is now to minimize

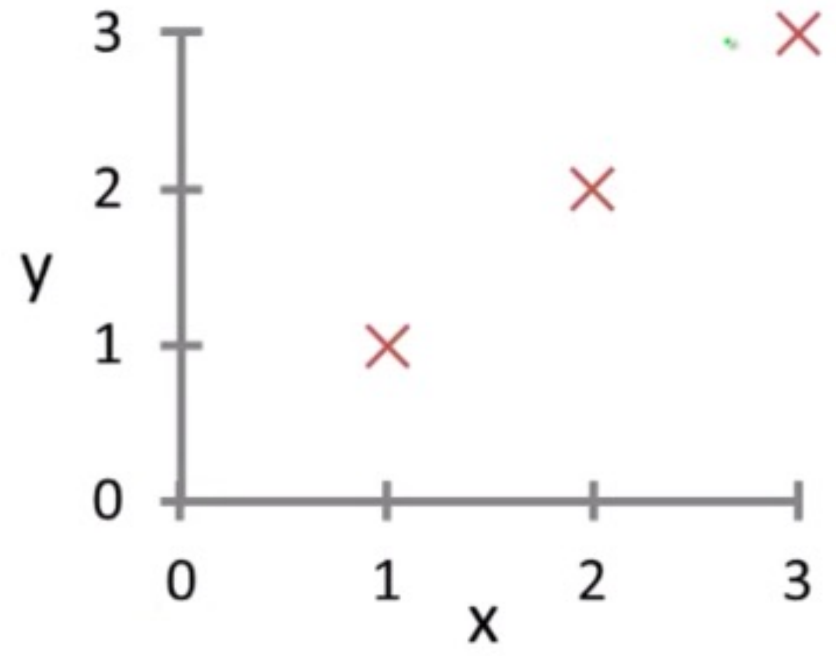
$$J(\theta_1)$$

- And our cost function looks like

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

Hypothesis function vs. Cost function

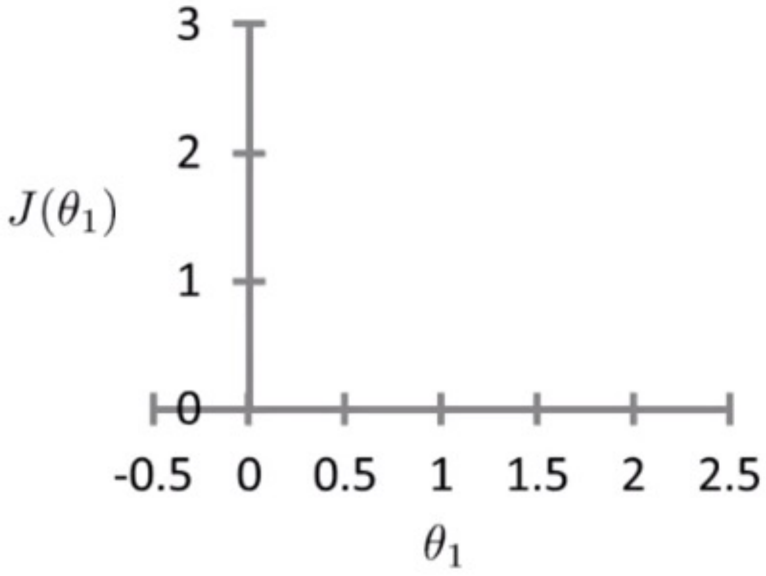
- If the points on the graph represent our training data and $\theta_1 = 1$, what does our hypothesis (line) look like ?



- What is the cost ?
- Remember : $\frac{1}{2m}\sum_{i=1}^{m}(\theta_1 x^i - y^i)^2$

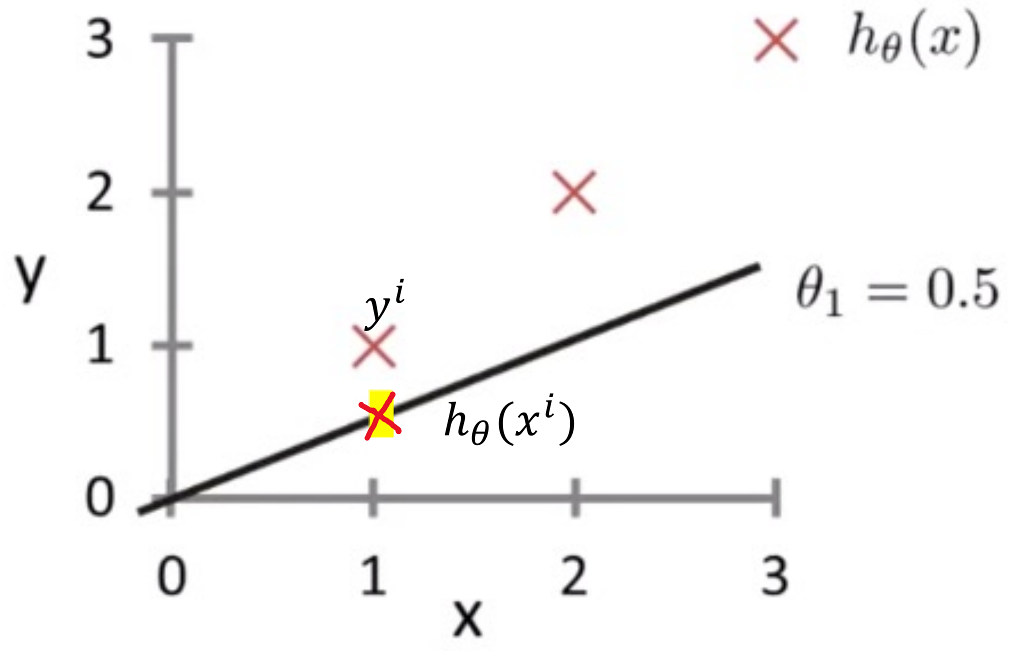
Hypothesis function vs. Cost function

- $J(\theta_1 = 1) = 0$
- We can now plot our error rate
- Notice that the values for θ_1 are on the horizontal axis. This is not the same graph as before !!
- This is a plot for the **cost function** :



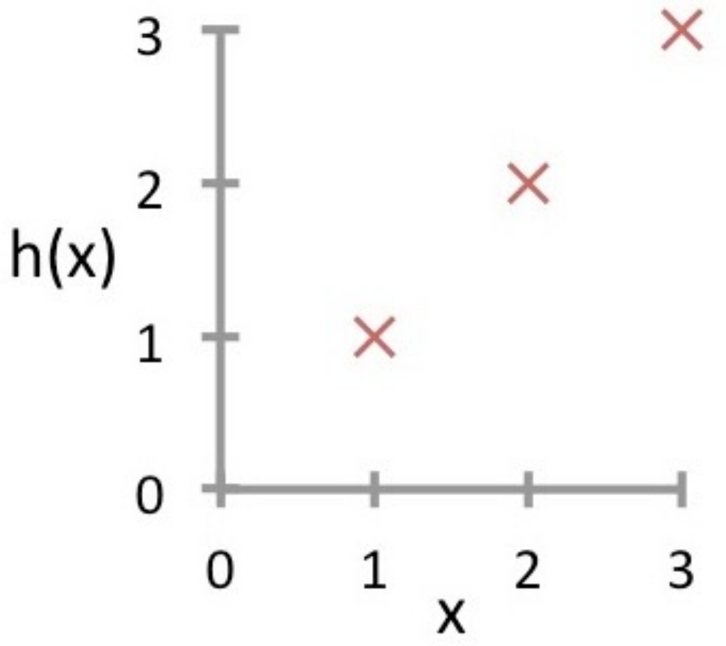
Hypothesis function vs. Cost function

- Now let's look at $\theta_1 = 0.5$
- And compute $J(\theta_1 = 0.5)$ (approx. 0.58)
- The error for each point is actually the height which seperates the data point and the line for a given x.



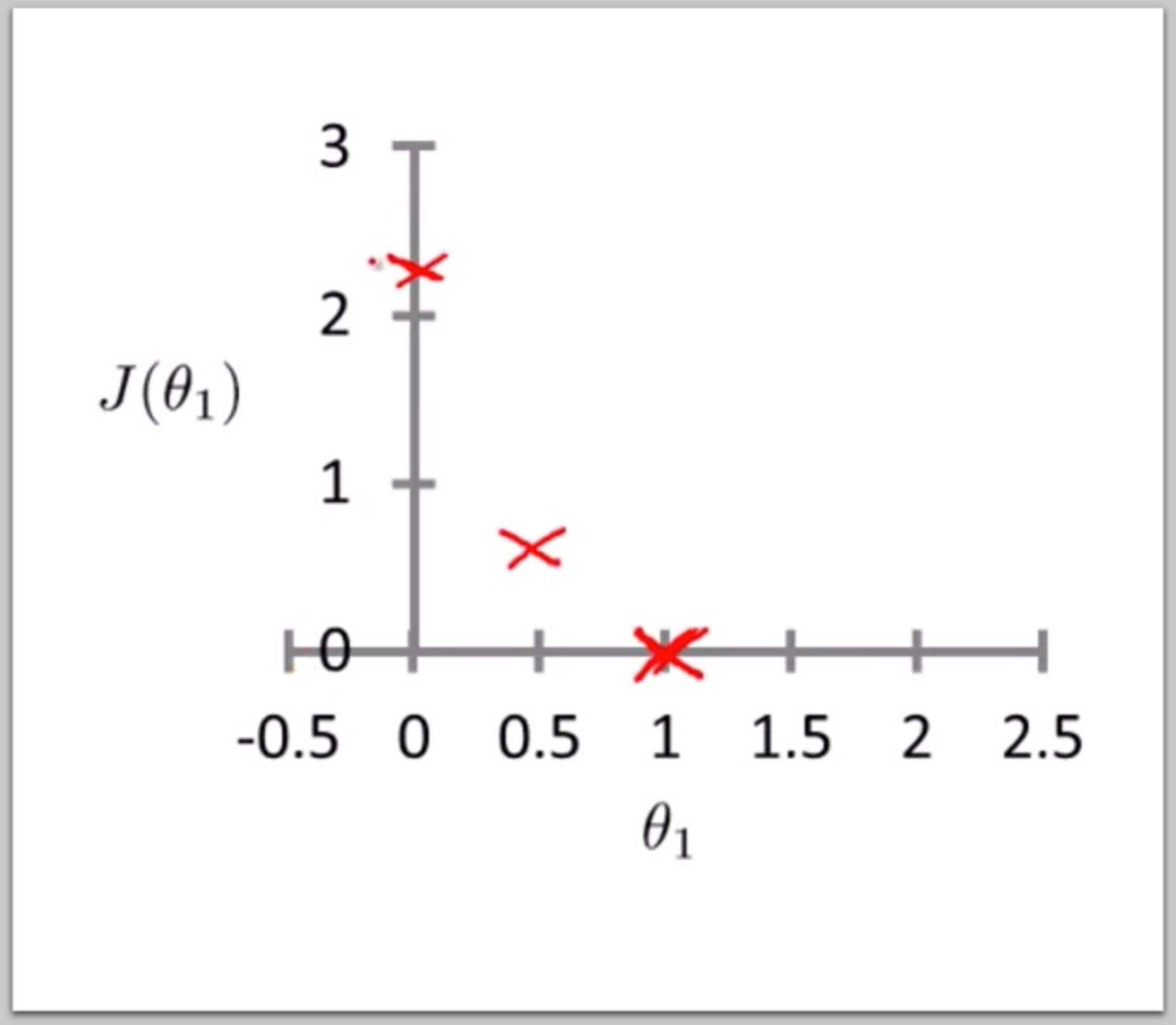
Your turn !

- Suppose this is our training set. $m = 3$.
- Given the same hypothesis and cost
- functions as before, what is $J(0)$?
- ie. $\theta_1 = 0$
- Should be approx. 2.3



Hypothesis function vs. Cost function

- We could continue plotting points but we'll stop here.
- With the error calculated for the different values of θ_1, we start to see part of the general shape of the function
- It turns out the function is convex/looks like a parabola.



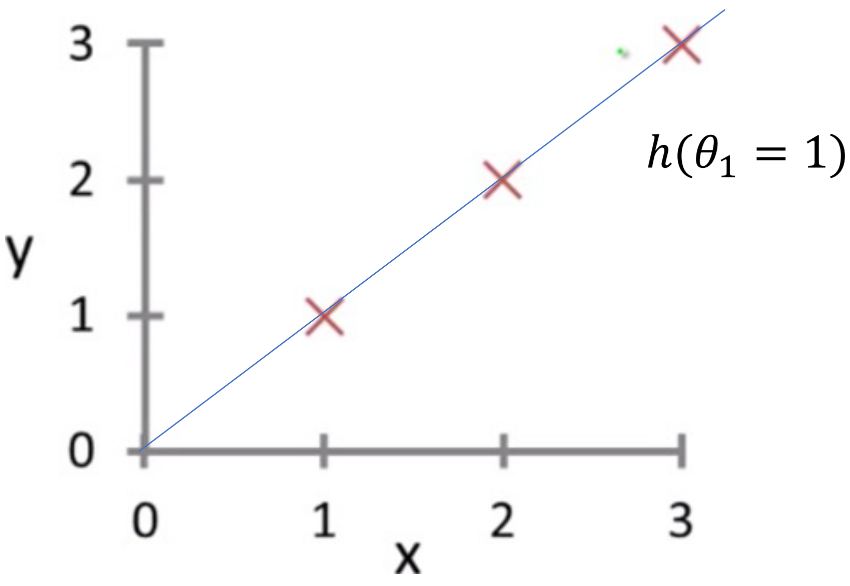
Quick recap

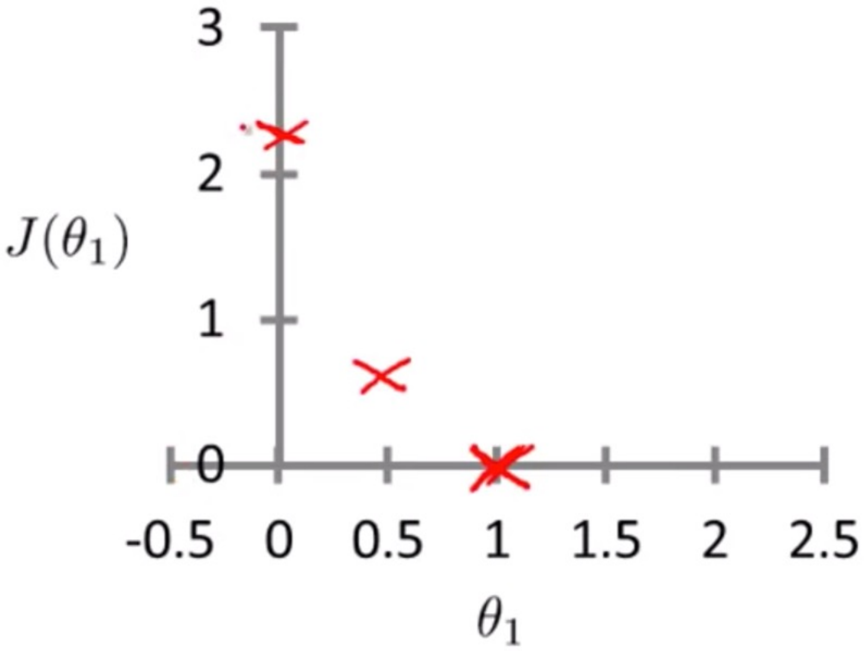
- Each value of θ_1 plotted corresponds to a different hypothesis / model on the data point graphs shown previously.

- For each value of θ_1 we can compute a value $J(\theta_1)$ to trace out the cost function.

- Now remember, we wanted to find the value of θ_1 which minimized $J(\theta_1)$… Looking at the graph we can now do so !

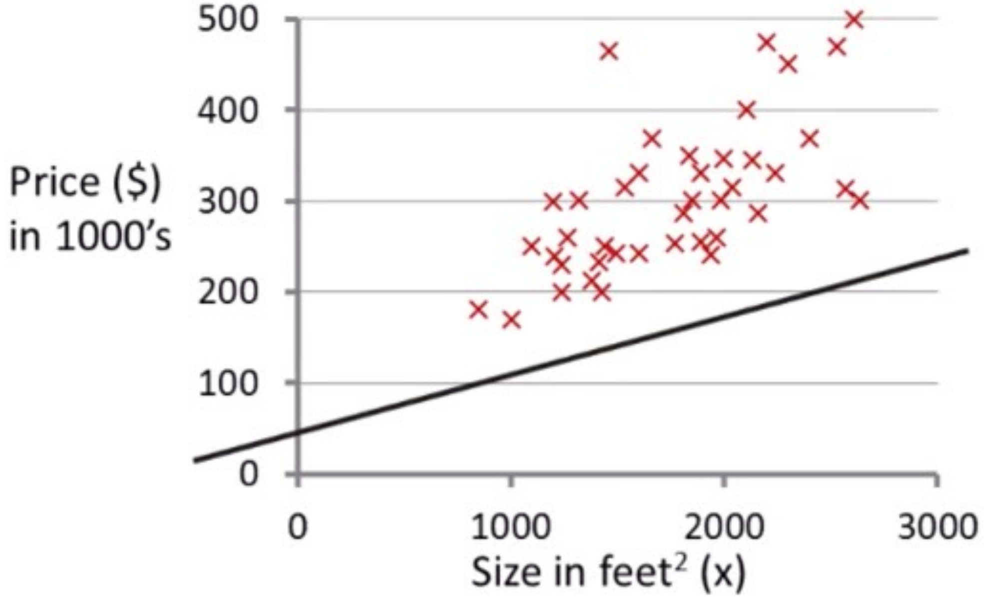
- No surprise, the value of θ_1 which minimizes the error, is associated with the model which fits the data perfectly





Back to 2 parameters

- Now, going back to our original data and model, we use a 2 parameter hypothesis to draw our line.
- For :
- $\theta_0 = 50$
- $\theta_1 = 0.06$
- We get this straight line as our model



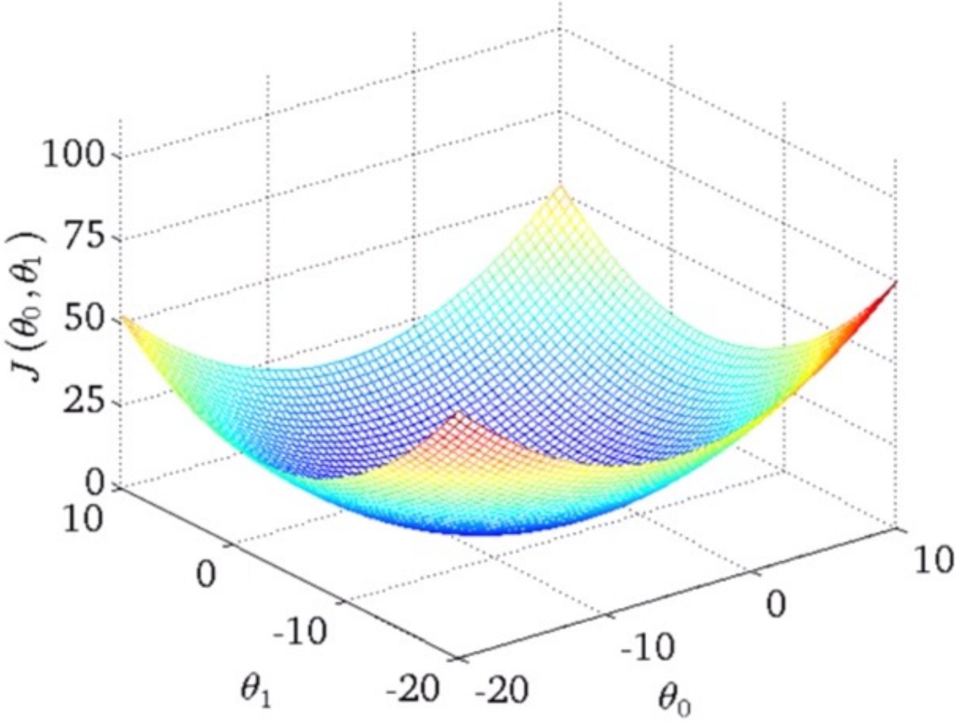
$$h_\theta(x) = 50 + 0.06x$$

Corresponding Cost function

- Now we have two parameters, the error graph will be slightly harder to plot as it has 3 dimensions:

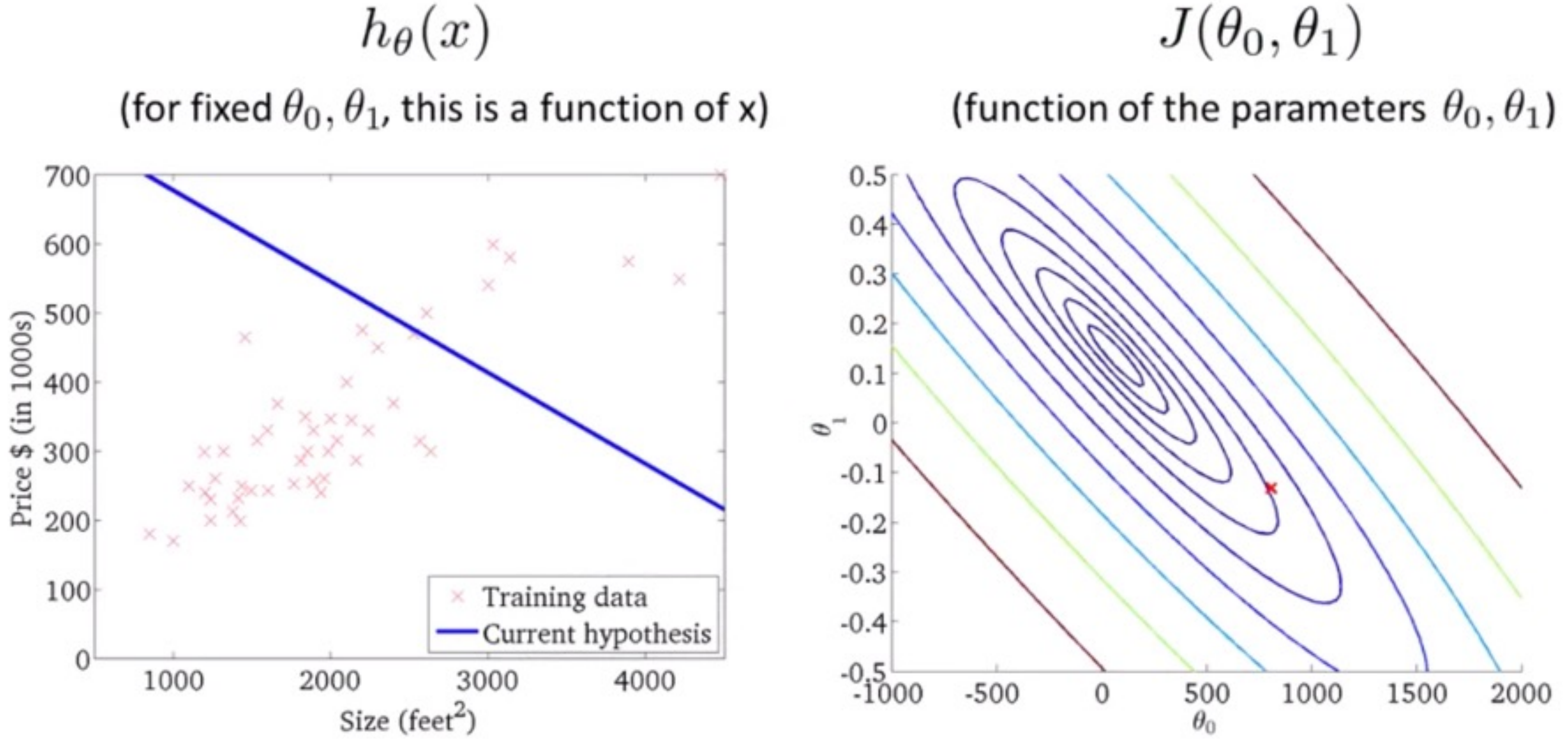
$$\theta_1, \theta_2, cost$$

- Indeed , $J(\theta_1, \theta_2)$ now has 2 inputs,
- So it will look like this in 3D:



Contour Plots

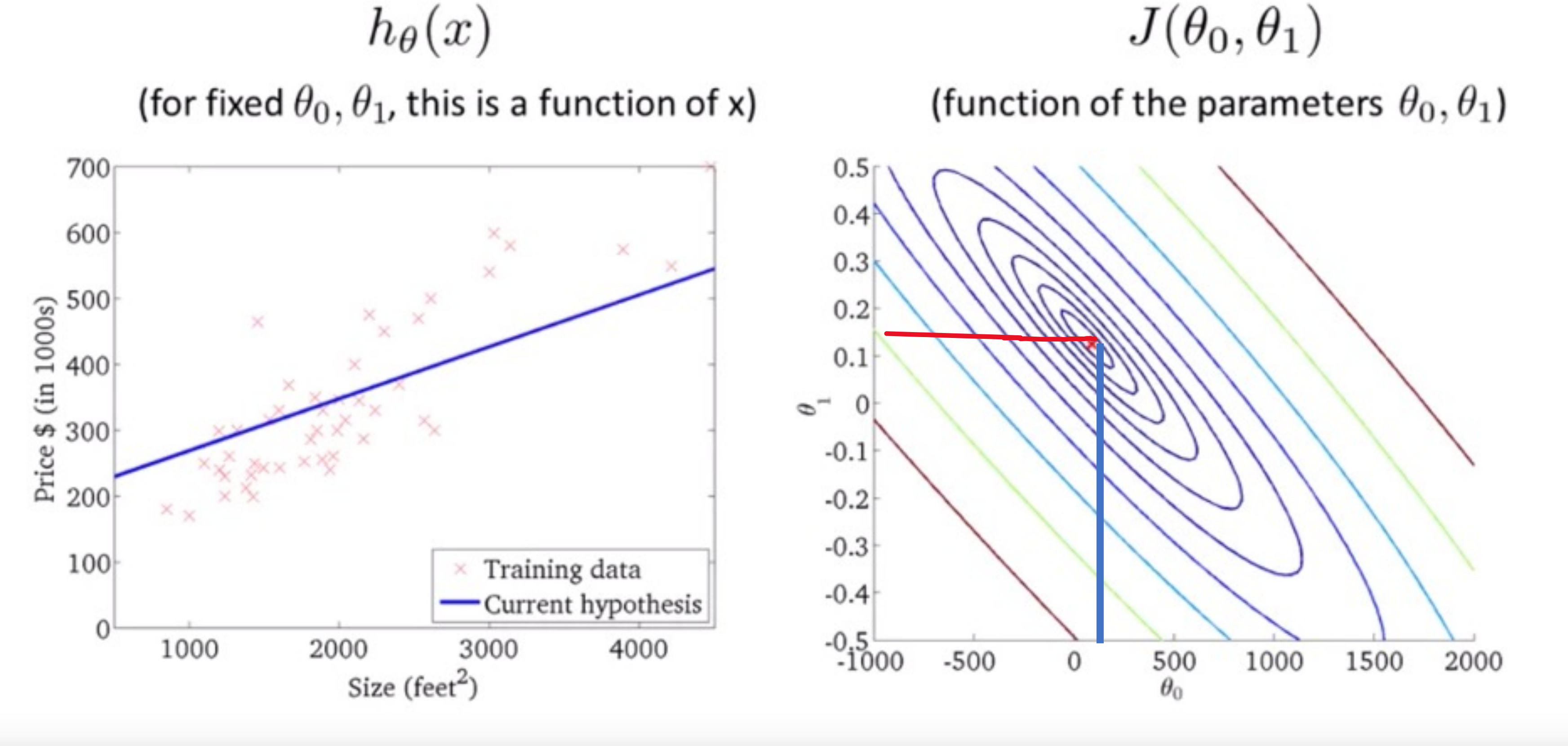
- To stay in 2D, you will see the cost function represented by a contour plot :



The ovals/ellipses show the set of points which take on the same value for given values of θ_0, θ_1

Countour Plots

- The minimum is at the center of all the « ellipses ».
- This shows a model very close to the minimum.



Gradient Descent

- Now we know how to evaluate a model, using a cost function, how do we make the model *learn* the optimal parameters ?
- In other words, how do we minimize the cost function without testing all the different possible models ?

- The algorithm used to do this is called *Gradient Descent*, and is essential to most machine learning algorithms, not just linear regression !
- In DL libraries this type of algorithm is called an ‘optimizer’ and other variants exist.

Gradient Descent

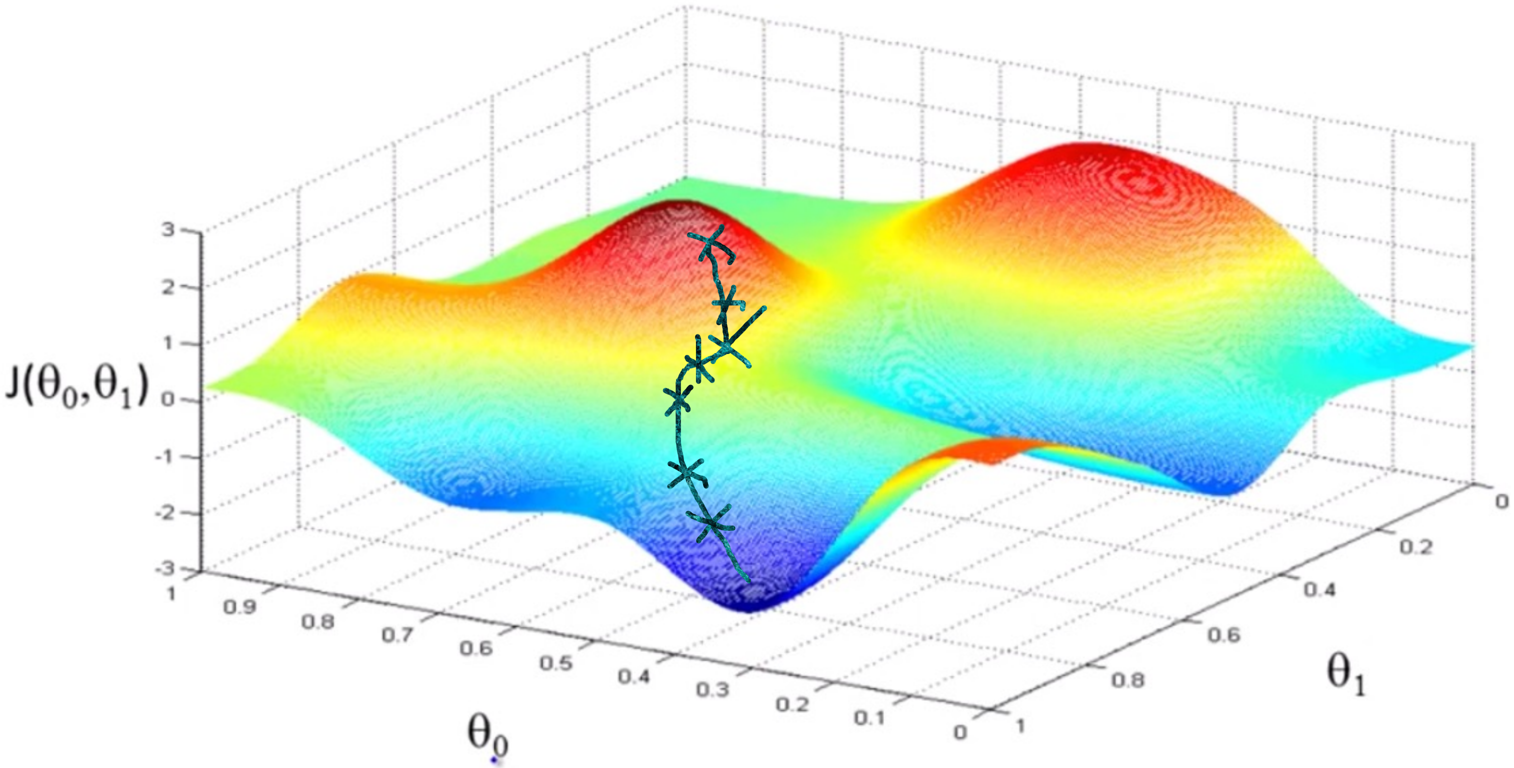
- We have some function $J(\theta_1, \theta_2)$ which we want to minimize…

Outline :

- Start with some inital guess, some random values for θ_1, θ_2
- Keep updating θ_1, θ_2 a little bit to reduce $J(\theta_1, \theta_2)$ until we hopefully end up at a minimum

GD intuition

- This is your cost function in 3D
- Imagine you start somewhere near the top of one of the « hills » and your goal is to walk in the direction which will take you down to the bottom the fastest.



GD formula

$$\text{repeat until convergence } \{$$
$$\theta_j := \theta_j - \alpha \boxed{\frac{\partial}{\partial \theta_j} J(\theta_0, \theta_1)} \quad (\text{for } j = 0 \text{ and } j = 1)$$
$$\}$$

- This is the update formula for each of the parameters
- := signifies assignment
- α is a number called the *learning rate.* If α is very large, then it corresponds to an aggressive learning procedure and big steps being taken « downhill » and vice versa.
- $\boxed{\frac{\partial}{\partial \theta_j} J(\theta_0, \theta_1)}$ is a derivative term, for which we need to do a bit of calculus !

GD Intuition

- Why does this update make sense ?
- Why are we putting those 2 terms together ?

- Let’s try and get a basic understanding of derivatives before we go any further.

Derivatives

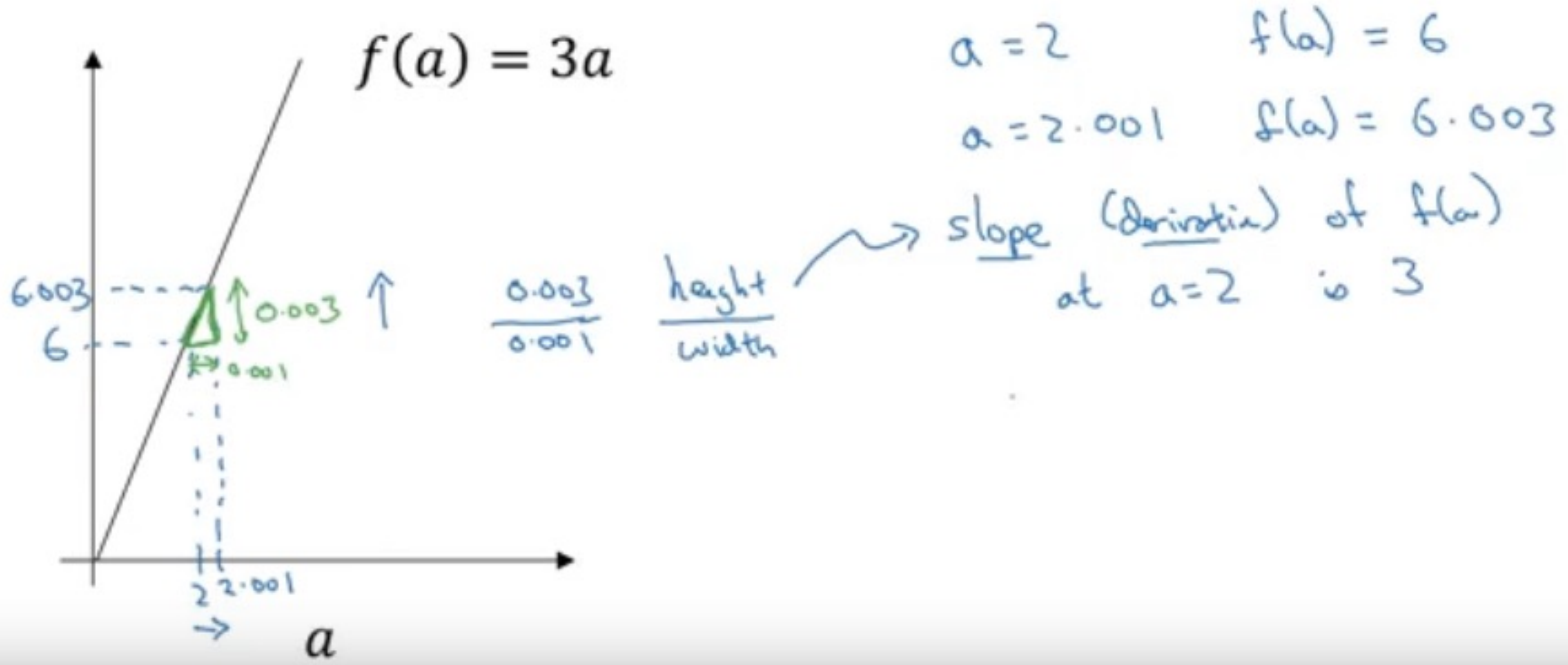
- The derivative describes how the output of a function varies with regard to a very very very tiny positive nudge to the input, to the point where we consider *almost* no variation in input….

- Informally, the deriviative tells you how a function behaves at a particular « instant », i.e. for a given input value.

- The derivative is commonly refered to as an « instantaneous rate of change »

Derivatives

- Here is a linear function as an example. What happens when we shift the input by a ‘small’ value like 0.001
- when a=2 ?
- when a=5 ?



[Andrew Ng’s course on DL]

Derivatives

- With this function, we expect a small positive nudge in the input to make the ouput increase by 3 times the value of that nudge.

$$f(5.001) = 15.003$$

- In other words the **ratio** between the change in output and the change in input is 3 :

$$\frac{change\ in\ f(a)}{change\ in\ a} = \frac{df(a)}{da} = \frac{0.003}{0.001} = 3$$

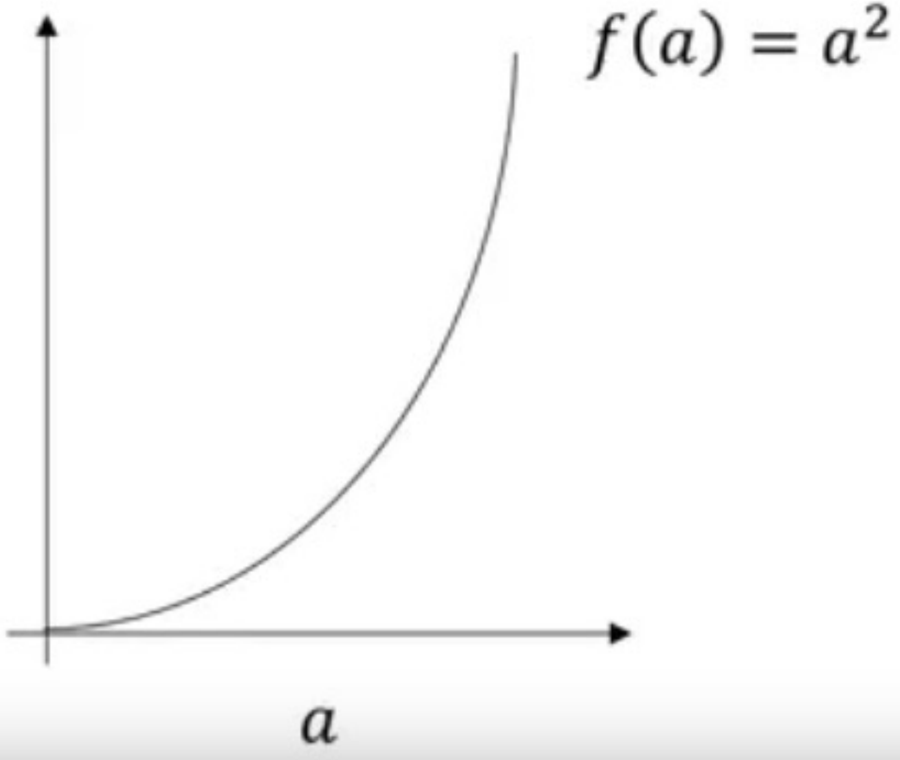
Derivatives

- This is just an example, but formally, the derivative considers this ratio when the input is increased by a much tinier amount !
- Nudge < 0.00000000000…..1 => nudge gets as close to 0 as possible

- In this previous example, whatever input value we pick, the derivative will be the same.
- The little triangle will be the same for any value of a : this makes sense since the function is a line and the ouput increases at a constant rate
- Question : What if the derivative was negative everywhere ? What would the function look like ?

Derivatives

- What if our function isn't a line ?

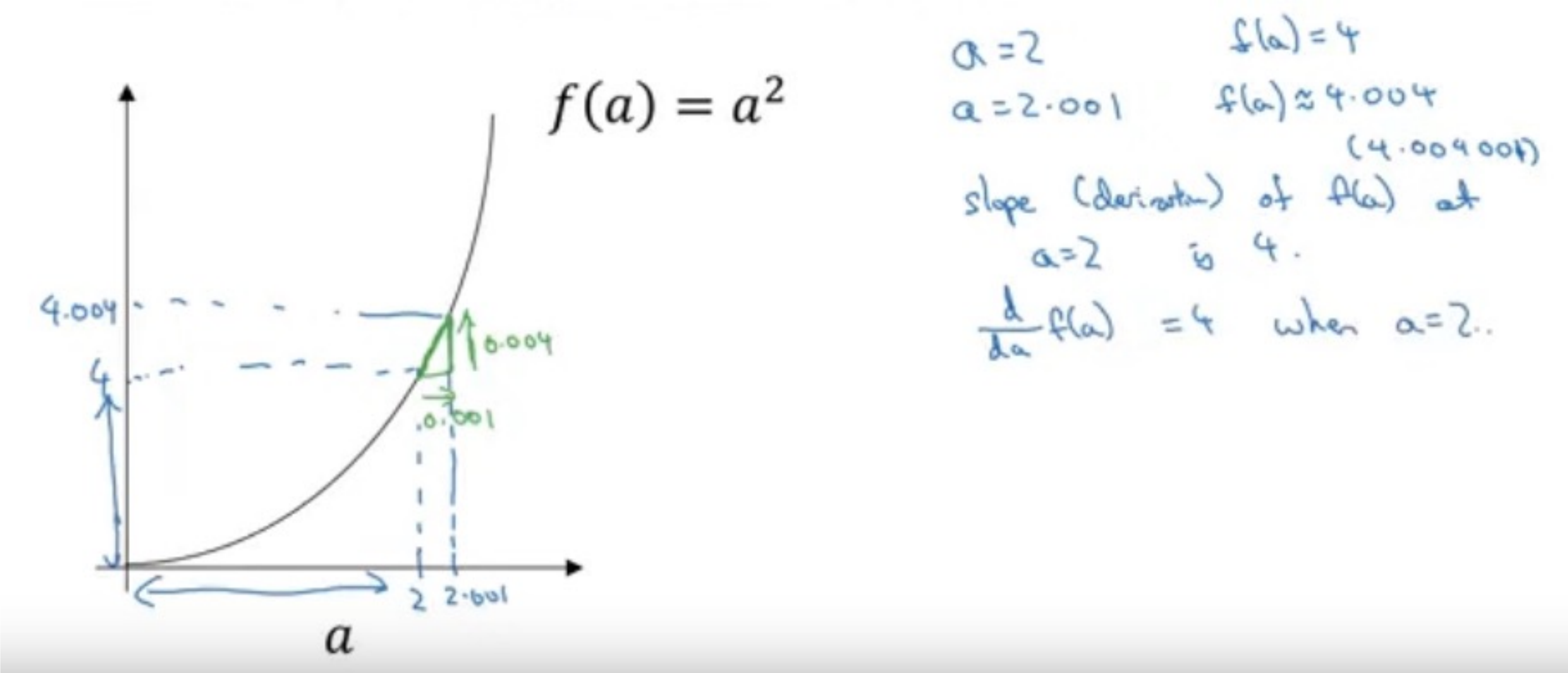


$a = 2$ $\quad f(a) = 4$

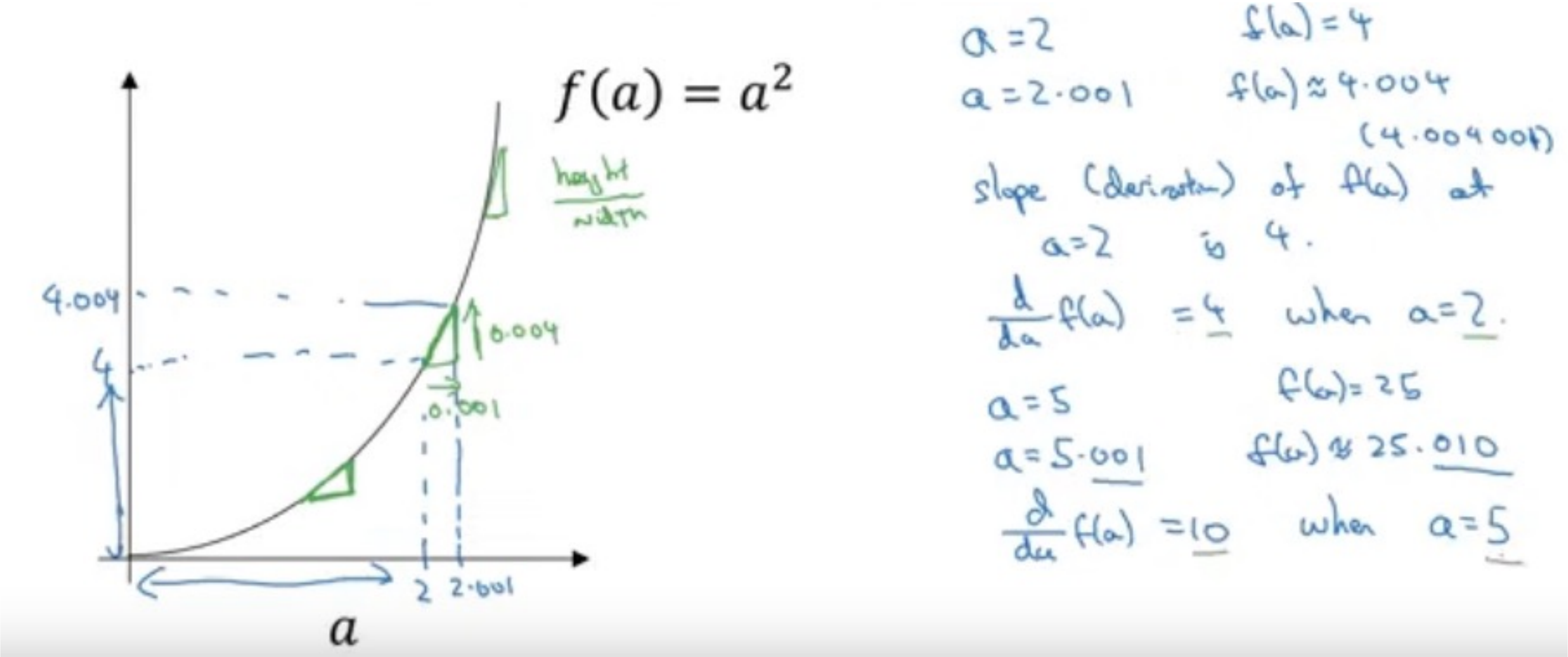
$a = 2.001$ $\quad f(a) \approx 4.004$

(4.004001)

- The derivative at a=2 is ...



- The derivative at a=5 is …



- Rules actually exist to compute derivatives by hand quickly !
- For the function

$$f(a) = a^2$$

$$f'(a) = \frac{d}{da} f(a) = 2a$$

(The notations are called Lagrange and Leibniz notations and are both common)

- If we look at the derivatives/slopes/ratios we calculated previously, this does indeed seem to work ! (and this can be proven of course)
- Note: the derivative is equal to the slope of the tangent on the graph at our input value.

Derivatives: (optional)

$$f'(x) = \frac{df}{dx} = \lim_{\Delta x \to 0} \frac{f(x + \Delta x) - f(x)}{\Delta x}$$

<u>Example 2:</u> $f(x) = x^2$

$$\begin{aligned}
\frac{df}{dx} &= \lim_{\Delta x \to 0} \frac{f(x + \Delta x) - f(x)}{\Delta x} \\
&= \lim_{\Delta x \to 0} \frac{x^2 + 2x\Delta x + (\Delta x)^2 - x^2}{\Delta x} \\
&= \lim_{\Delta x \to 0} \frac{2x\Delta x + (\Delta x)^2}{\Delta x} \\
&= \lim_{\Delta x \to 0} 2x + \Delta x.
\end{aligned}$$

- As Δx approaches 0, the derivative
- Approaches 2x.

GD Intuition

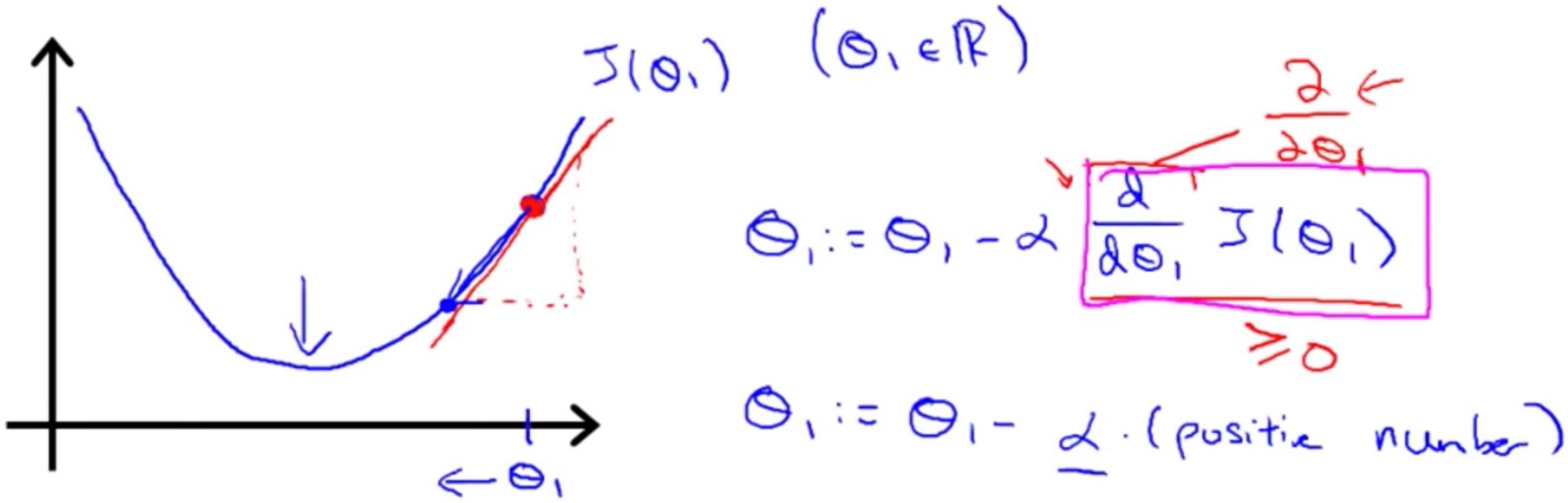
- Now we have a basic understanding of derivatives, let’s apply this understanding to the gradient descent algorithm by using a simpler example, with a cost function of only 1 single parameter.

- We use $J(\theta_1)$ instead of $J(\theta_0, \theta_1)$

- Let’s look at a couple scenarios to see how Gradient Descent updates our parameter θ_1.

When the derivative is positive…

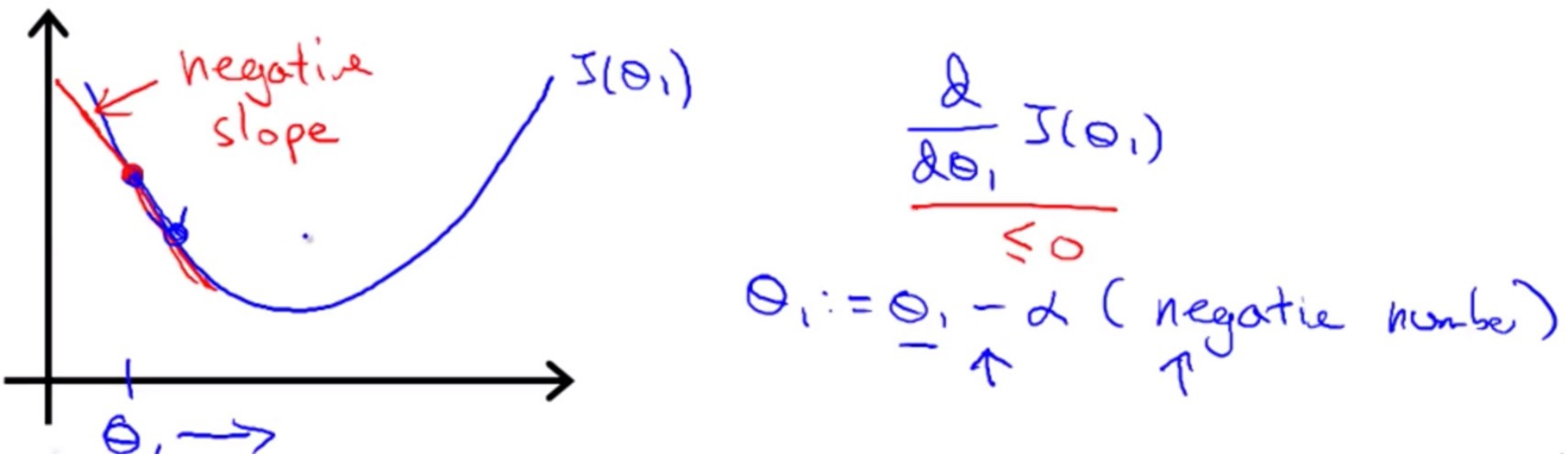
- Remember, our cost function looks like a parabola.
- When θ_1 is too high, we would want Gradient Descent to **reduce** this parameter and bring it closer to the « sweet spot », where the cost is minimized.
- Let's see if it does the right thing :



See [Andrew Ng's course]()

When the derivative is negative...

- When θ_1 is too low, let's see if Gradient Descent **increases** it and brings it closer to the « sweet spot », where tht cost is minimized :



Recap

- When the the parameter value is too high, the derivative is positive and the update rule decreases the value for the parameter.

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

> 0

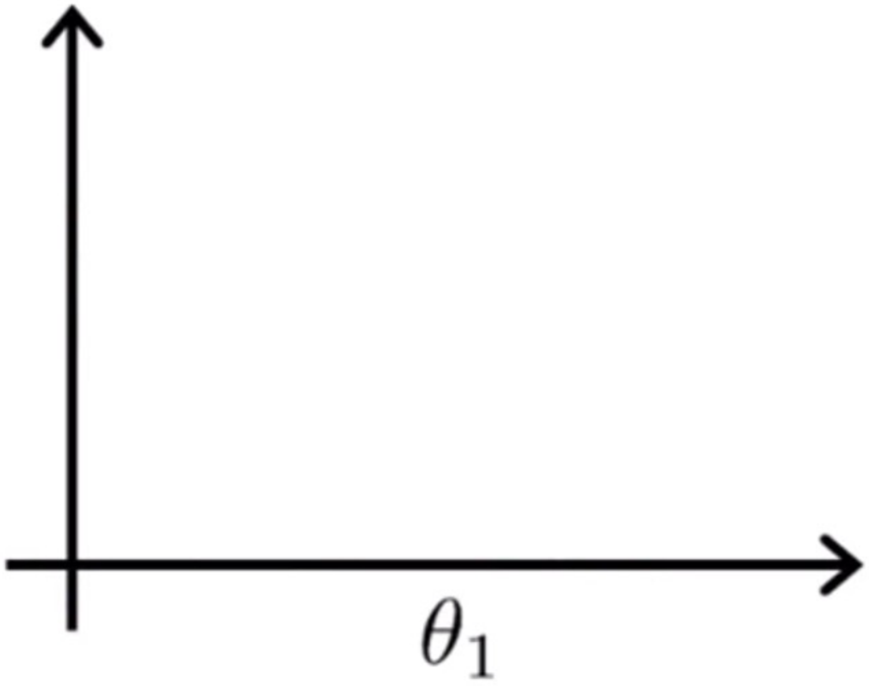
- Conversely, when the parameter value is too low, the parameter value will be increased by the update rule.

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

< 0

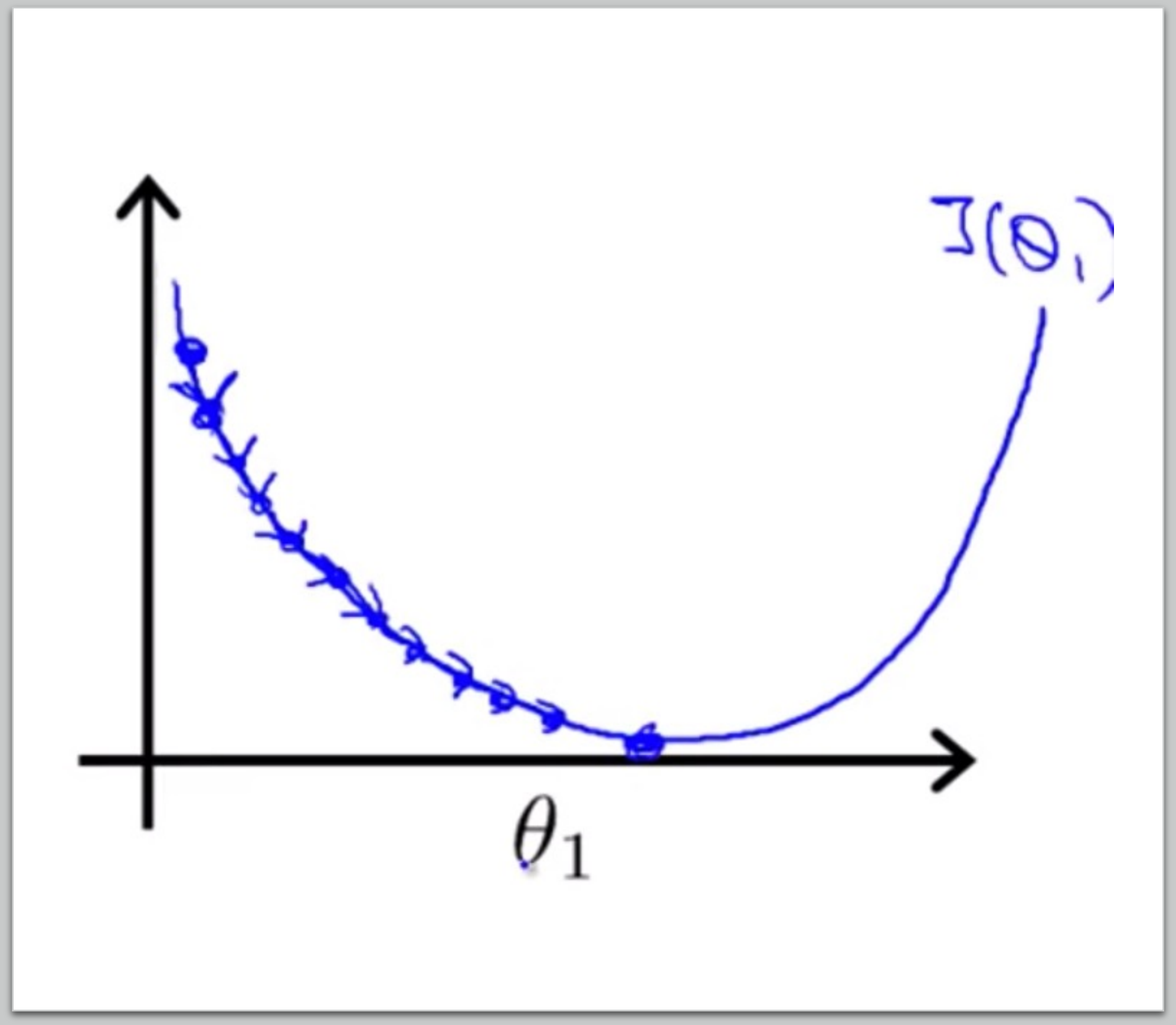
Okay so now what about α ?

- Remember the update rule : $\theta_1 := \theta_1 - \alpha \frac{d}{d\theta_1} J(\theta_1)$
- How does α influence the update of our parameter θ_1 ?
- If α is too small :



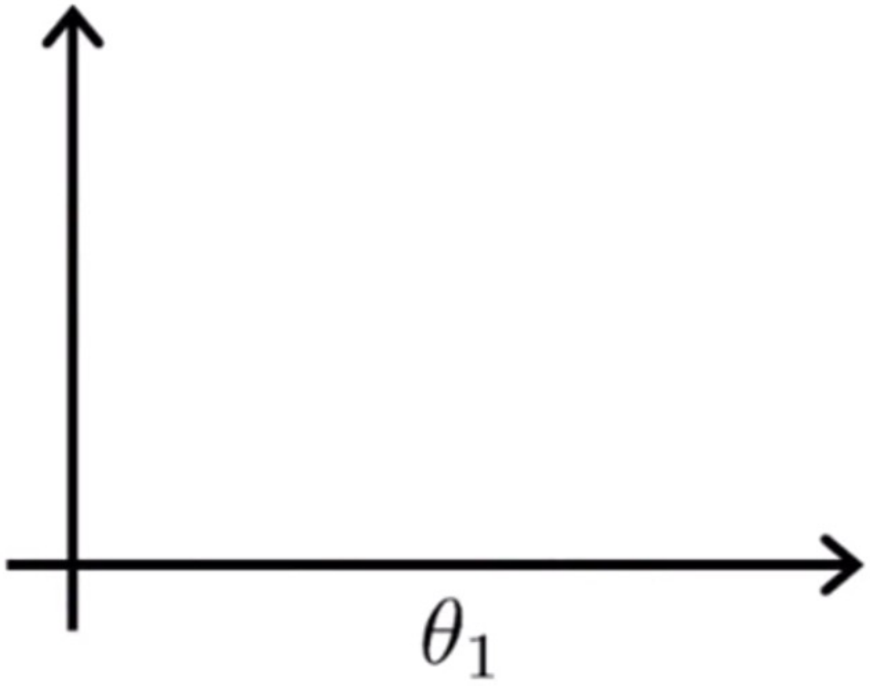
If α is too small

- Many small steps will be taken , which maks Gradient Descent very slow

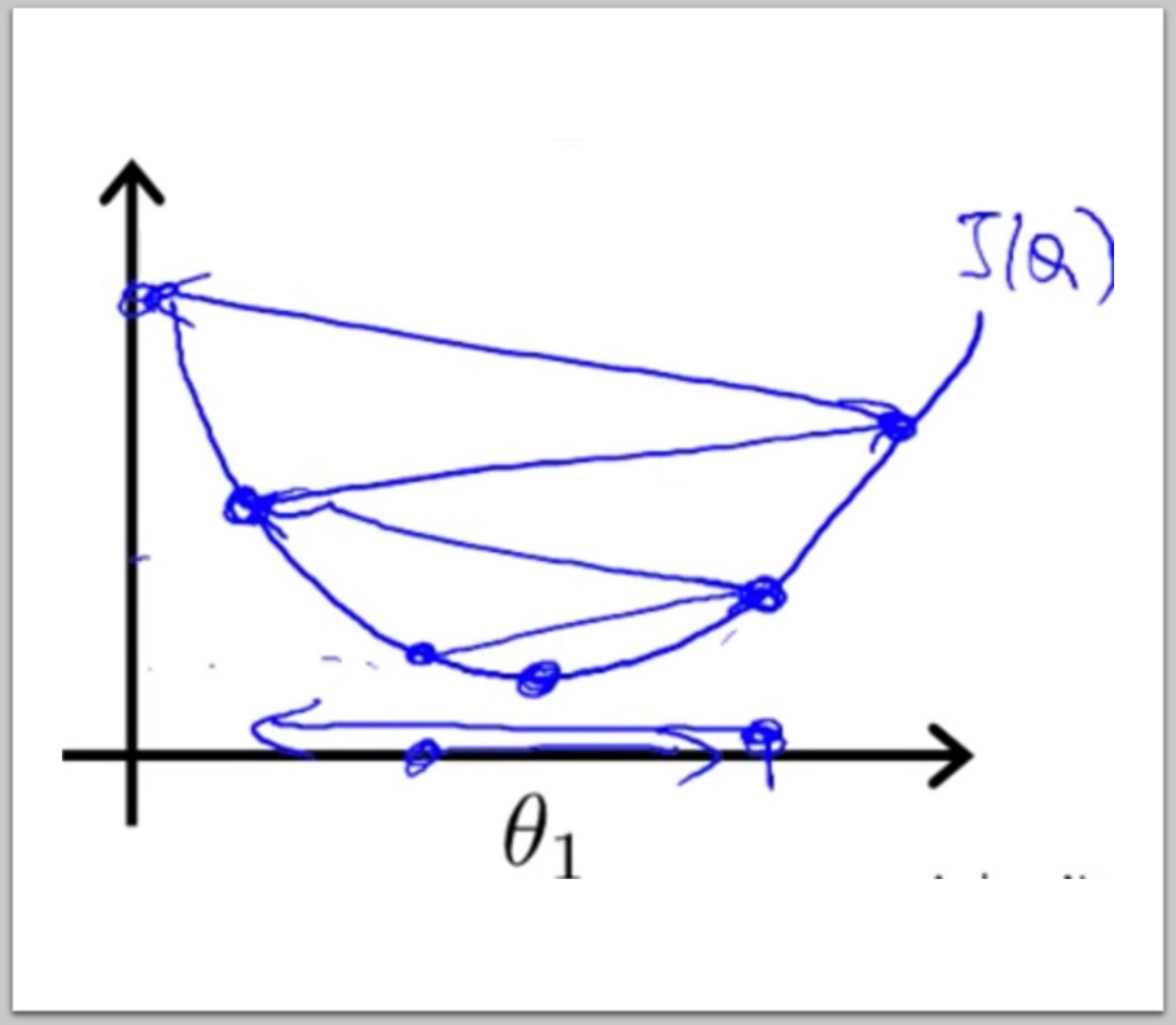


If α is too large…

- Gradient descent may « overshoot », go past the minimum. It may even never converge (never find the minimum) and keep jumping around.



If alpha too large

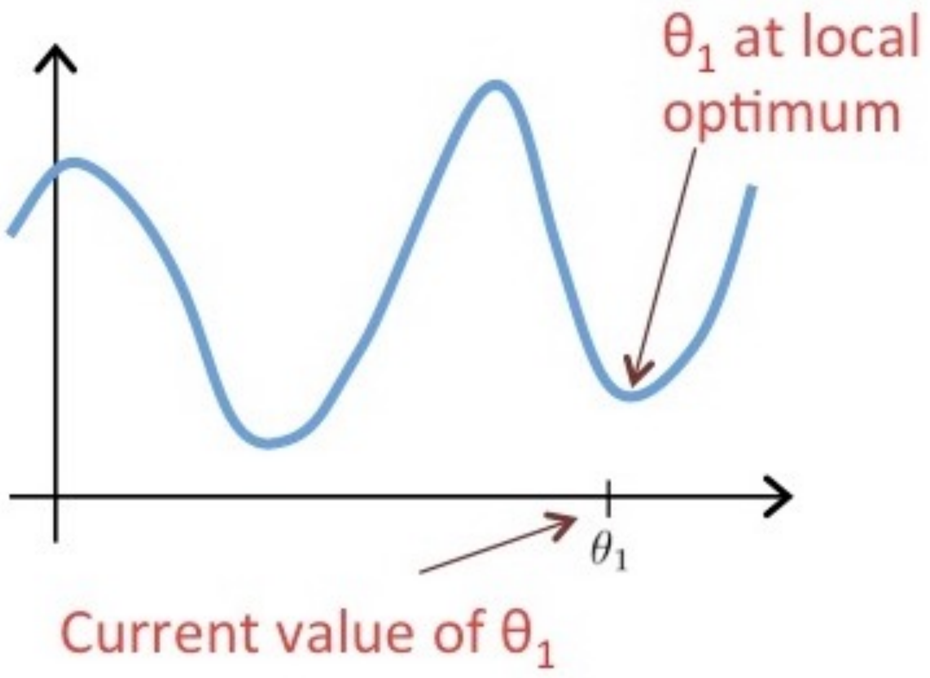


Question

1. Change θ_1 in a random direction ?
2. Move θ_1 in the direction of the global minimum of $J(\theta_1)$?
3. Leave θ_1 unchanged ?
4. Decrease θ_1 ?

Suppose θ_1 is at a local optimum of $J(\theta_1)$, such as shown in the figure.

What will one step of gradient descent $\theta_1 := \theta_1 - \alpha \frac{d}{d\theta_1} J(\theta_1)$ do?



Recap

- To update our parameter with the Gradient Descent algorithm, we perform 2 essential steps :

1. Compute the derivative of the parameter with respect to the value we want to minimize (ie. our cost: a score to express how good our model is doing)

2. Take an optimization step/update the parameter. This update will be proportional to the derivative and the learning rate.

Large derivative (steep tangent line) + large learning rate = big update

Piecing everything together

- This is all we need :
  - A **hypothesis** function (our model)
  - A **cost function** (to tell us how well/bad our model is doing)
  - **Gradient Descent** (to update our parameters and get closer to a better model)

Gradient descent algorithm

$$
\begin{aligned}
&\text{repeat until convergence } \{ \\
&\quad \theta_j := \theta_j - \alpha \frac{\partial}{\partial \theta_j} J(\theta_0, \theta_1) \\
&\qquad (\text{for } j = 1 \text{ and } j = 0) \\
&\}
\end{aligned}
$$

Linear Regression Model

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

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

Derivatives vs. Partial derivatives

- Except, instead of having a cost function with a single input, we are back to 2 inputs, our 2 parameters $\theta_0$and θ_1.

- When we have functions with multiple inputs (known as multivariate functions), computing 1 single derivative is no longer enough!

- The function's « **instantaneous rate of change** » for a given combination of parameters is now determined by 2 values :

  - How does a tiny change in $\theta_0$change $J(\theta_0, \theta_1)$?

  - How does a tiny change in $\theta_1$change $J(\theta_0, \theta_1)$?

=> Packed together, these 2 derivatives make up what is referred to as the **gradient**

- Each derivative is a **partial derivative**. (you need both together to get the whole picture !)

Derivatives vs. Partial derivatives

- *Partial Derivative :*

This comes down to calculating the derivative at each input value, treating the other input as a constant

- We pretend for a second that the other input value has basiclally no effect on the function

  - when looking at θ_0 , we treat θ_1 as a constant

  - when looking at θ_1 , we treat θ_0 as a constant

Partial derivatives and graphs intuition

- To help illustrate things and relate them to our simple Gradient Descent intuition:

Gradient Descent

- Each partial derivative tells us how the function behaves (increases/decreases, quickly/slowly, stays constant…) with respect to a single input
- We can then use this information to know if we should increase or decrease each input to get closer to our minimum cost value !
- **Gradient** : the partial derivatives packed together in a vector
- **Descent** : we want to find the cost function’s minimum, using the gradient as a source of information to tell us if the cost is increasing/decreasing with respect to each input.

Update rule

- So we need to figure out the partial derivatives for each parameter !

- the partial derivative of $J(\theta_0, \theta_1)$ with respect to θ_1
- the partial derivative of $J(\theta_0, \theta_1)$ with respect to θ_2

Partial derivatives of $J(\theta_1, \theta_2)$

- You can treat these results as being **given**, in order not to go into the details of the derivation, which involves a couple of the rules mentioned in the optional slides at the end

- General formula

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

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

See here for the [MSE derivative]

Partial derivatives of $J(\theta_1, \theta_2)$

- Here are the partial derivatives obtained (take these at face value for now):

$$j = 0 : \quad \frac{\partial}{\partial \theta_0} J(\theta_0, \theta_1) = \frac{1}{m} \sum_{i=1}^{m} (h_\theta(x^i) \; - y^i)$$

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

- These formulas allow us to compute the partial derivatives for each of the parameters, which we can then plug into our Gradient Descent algorithm.

Gradient Descent

- We now have formulas to update our parameters !

$$
\begin{aligned}
&\text{repeat until convergence } \{ \\
&\quad \theta_0 := \theta_0 - \alpha \frac{1}{m} \sum_{i=1}^{m} \left(h_\theta(x^{(i)}) - y^{(i)} \right) \\
&\quad \theta_1 := \theta_1 - \alpha \frac{1}{m} \sum_{i=1}^{m} \left(h_\theta(x^{(i)}) - y^{(i)} \right) \cdot x^{(i)} \\
&\}
\end{aligned}
$$

Quick recap to put things into perspective

- We have :
- a **model**, which is a line :

$$h(x) = \theta_0 + \theta_1 x$$

- a **cost function,** to tell us how good/bad our model is fitting the data:

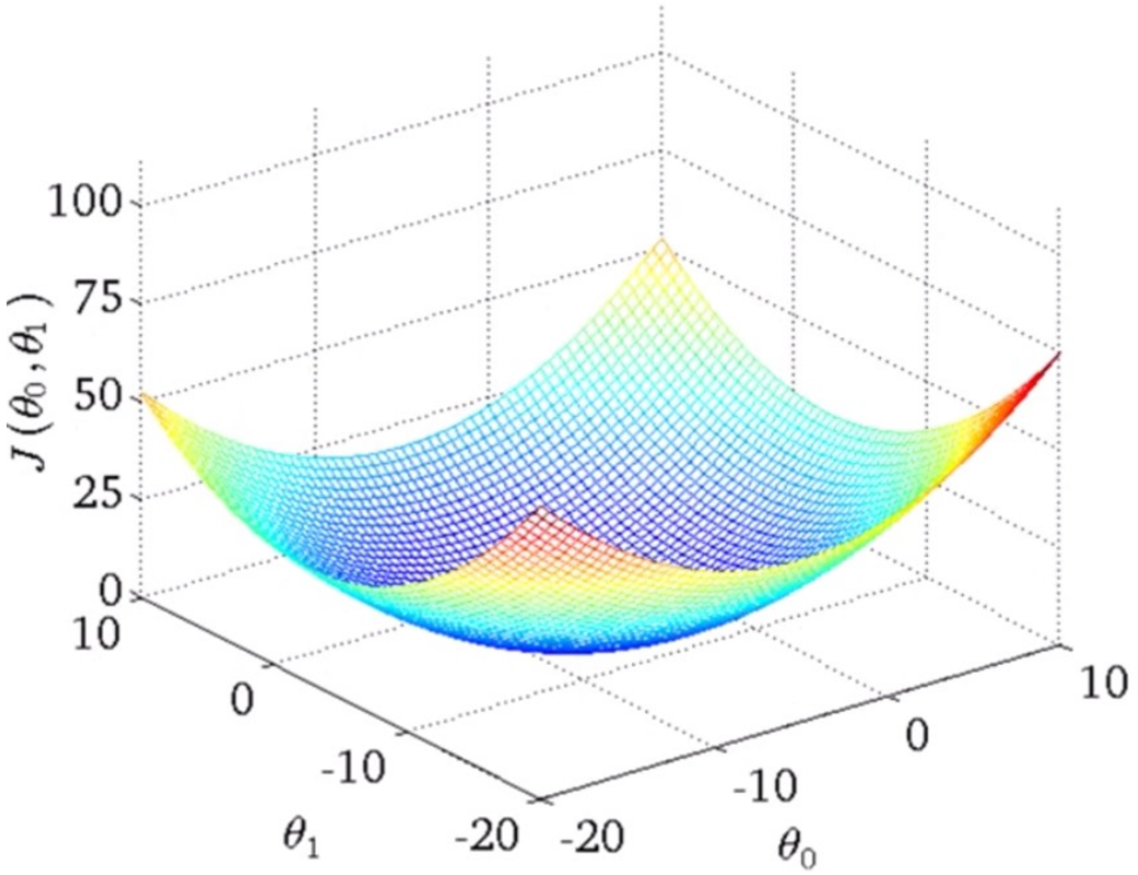
$$J(\theta_0,\theta_1) = \frac{1}{2m}\sum_{i=1}^{m}(h\left(x^{(i)}\right) - y^{(i)})^2$$

- **Gradient Descent,** a method to update our parameters so as to minimize the cost function:

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

Update examples

- For linear regression, the cost function will always be bowl-shaped

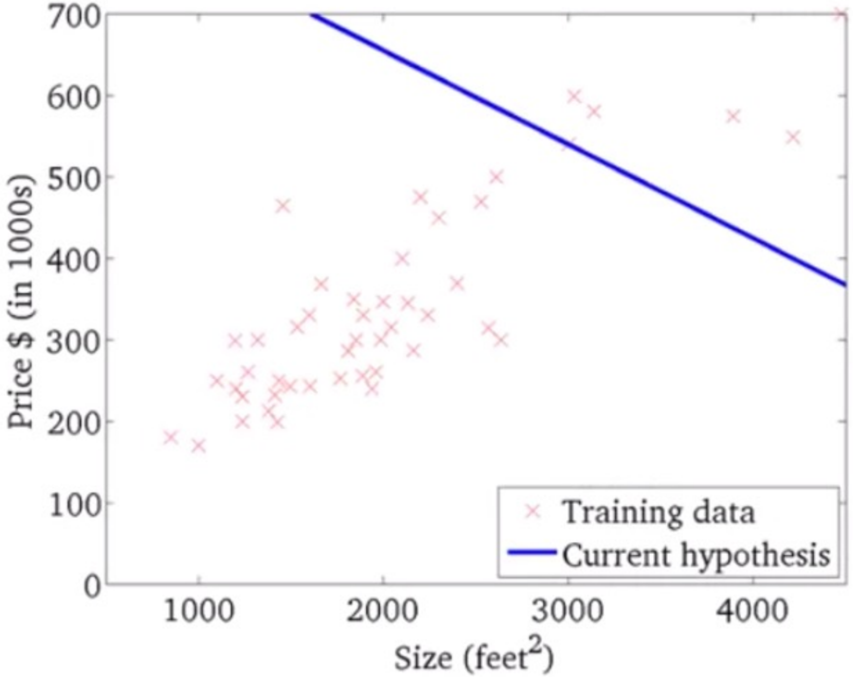


Update examples

Say we initialize our parameters randomly, this is the model and cost :

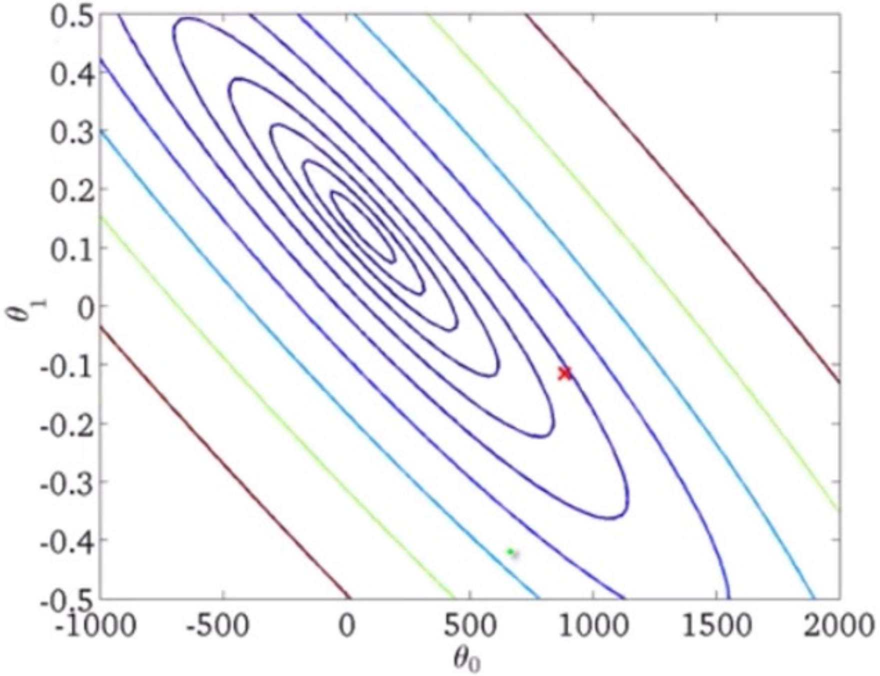
$$h_\theta(x)$$

(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)

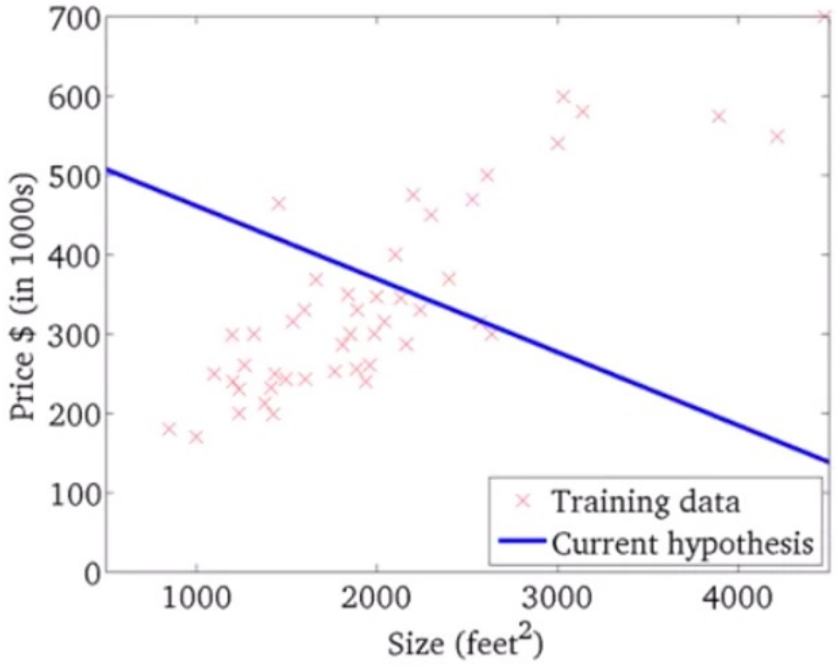


Update examples

As we take Gradient Descent steps, the model (line) seems to be fitting the data better

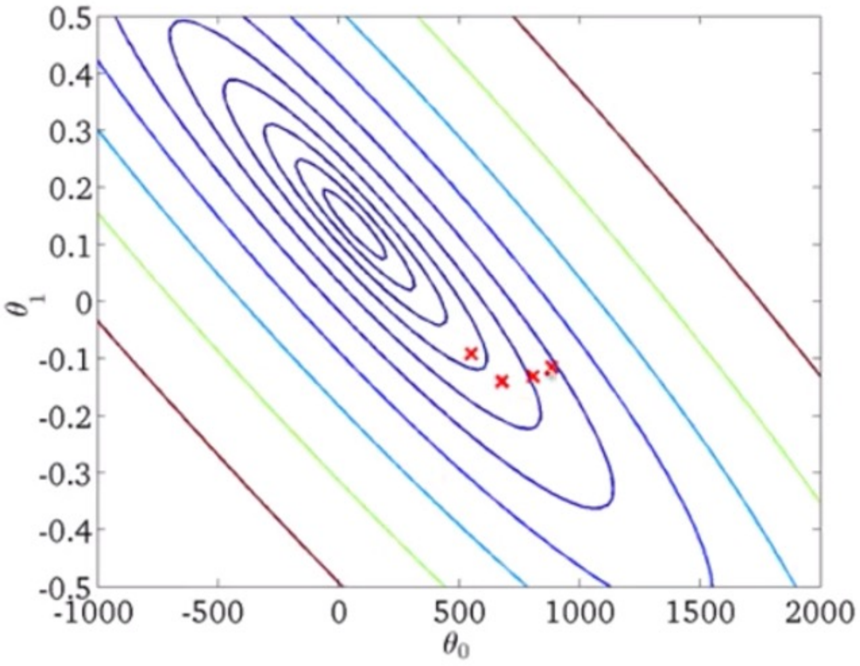
$h_\theta(x)$

(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)

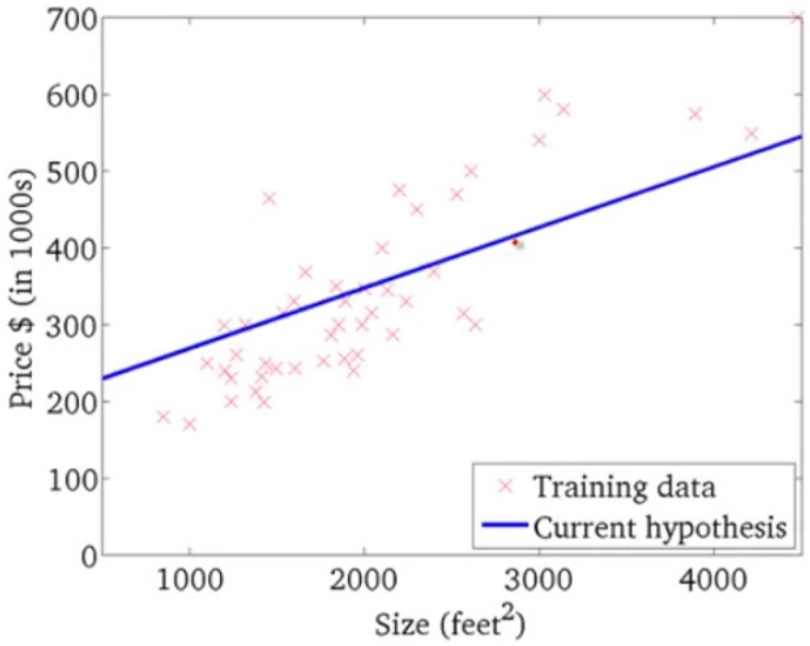


Update examples

Until we reach the global minimum

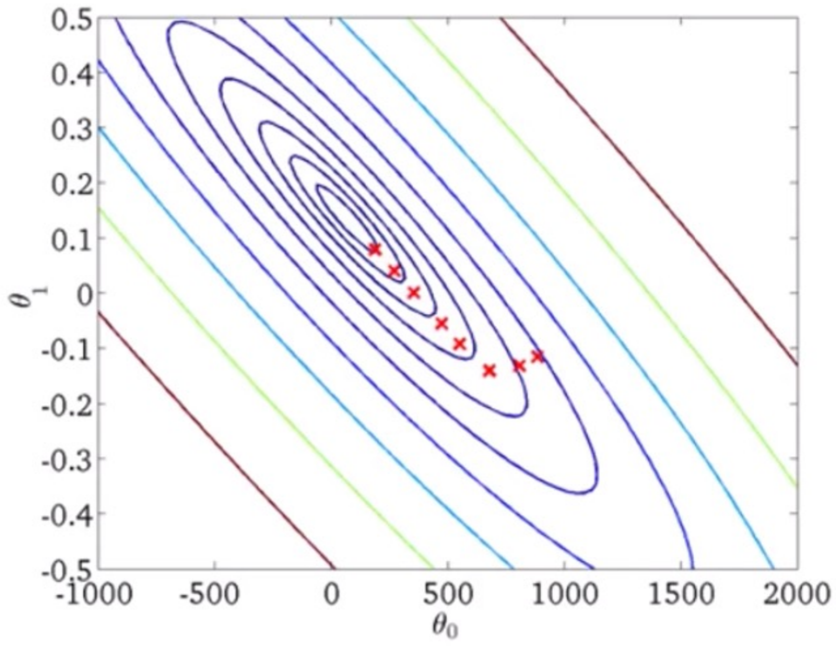
$h_\theta(x)$

(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)



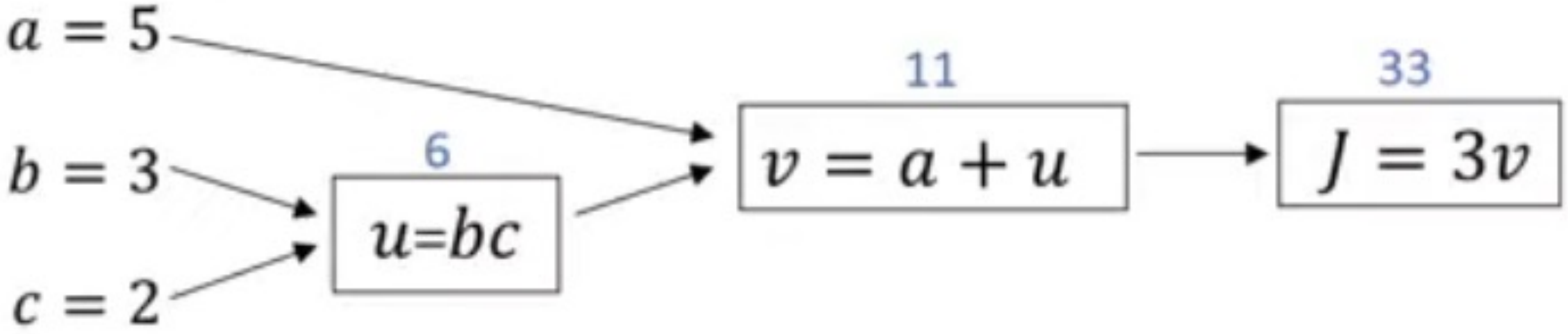
Chain Rule (plus intro to backpropagation)

- Just a little intuition on how to the derivative of the MSE was calculated.

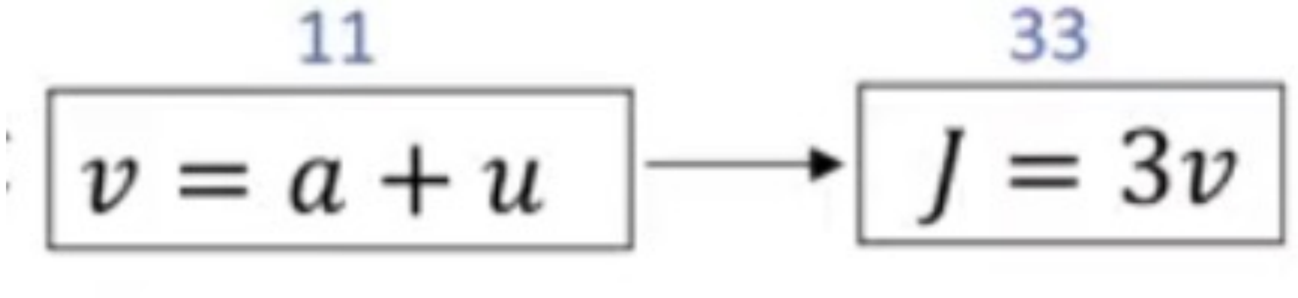
- The chain rule is the main mathematical tool used to compute partial derivatives and is essential to any algorithm that uses gradient descent !

Chain Rule

- Let’s apply derivatives to a computation graph that has multiple nodes, each representing a function :



Finding the derivative the final output J with respect to each node

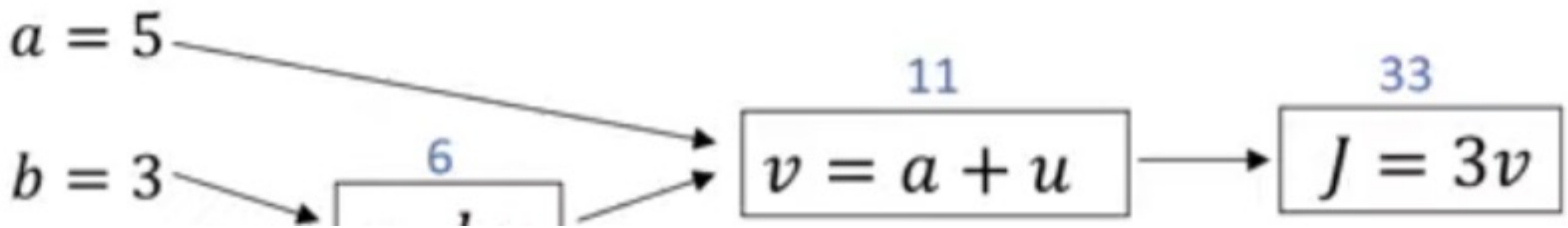


- $\frac{dJ}{dv} = ?$

$$v = 11 \ \rightarrow 11.001$$
$$J = 33 \ \rightarrow 33.003$$

- $\frac{dJ}{da} = ?$
- To figure this out, let's look at how a affects v, and then how this change in v affects J :

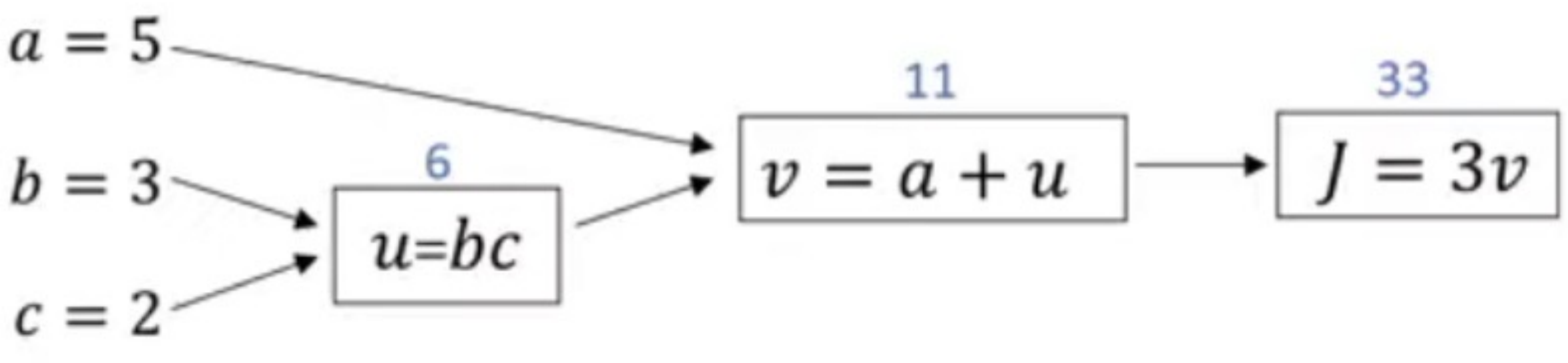
$$a = 5 \ \rightarrow 5.001$$
$$v = 11 \ \rightarrow 11.001$$
$$\mathrm{J} = 33 \ \rightarrow 33.003$$



- Reasoning by looking at the waterfall effect of nudging a is the basis of the chain rule : $a \rightarrow v \rightarrow J$
- We can know how a nudge in a changes J by multiplying how much this nudge **first** changes v **and then** J.

$$\frac{dJ}{da} = \frac{dJ}{dv} \times \frac{dv}{da}$$

- $\frac{dJ}{du} = ?$
- $\frac{dJ}{db} = ?$
- $\frac{dJ}{dc} = ?$



- To figure out $\frac{dJ}{du}$, the chain rule tells us

$$\frac{dJ}{du} = \frac{dJ}{dv} \times \frac{dv}{du} = 3 \times 1 = 3$$

- We can re-use the previous computations as we go back through the graph !
- Same applies for the other derivatives.

Multivariate Linear Regression

But what if we have **several features** vs. only 1 ?

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
...	...	...	...	...

Notation

- Features will be denoted by

$$x_1, x_2, \dots, x_n,$$

Where :

Notation:

- n = number of features

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

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

Notation

- With this notation, the example $\boldsymbol{x}^{(2)}$ is a 4-D vector :

x_1 Size (feet²)	x_2 Number of bedrooms	x_3 Number of floors	x_4 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
…	…	…	…	…

- $\boldsymbol{x}^{(2)} = \begin{bmatrix} 1416 \\ 3 \\ 2 \\ 40 \end{bmatrix}$
- $\boldsymbol{x}_3^{(2)} = 2$

The model

- Previously $h_\theta(x) = \cancel{\theta_0 + \theta_1 x}$
- Now,

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

- We can no longer represent the model using a 2D graph…

Gathering features and parameters into vectors

- For convenience, let's group up features and parameters into vectors:
- $\boldsymbol{x} = \begin{bmatrix} x_1 \\ x_2 \\ \dots \\ x_n \end{bmatrix}$ $\quad \boldsymbol{\theta} = \begin{bmatrix} \theta_0 \\ \theta_1 \\ \theta_2 \\ \dots \\ \theta_n \end{bmatrix}$ $\quad \boldsymbol{x} = \begin{bmatrix} x_0 \\ x_1 \\ x_2 \\ \dots \\ x_n \end{bmatrix}$

Size => n $\neq$ Size => n+1

- To avoid this size mismatch we can add a « dummy » feature, $x_0 = 1$

The model formula simplified

$$h_\theta(x) = \theta_0 x_0 + \theta_1 x_1 + \theta_2 x_2 + \cdots + \theta_n x_n$$

Instead of using a loop, which is slow, we can vectorize both the inputs and the parameters and compute their dot product !

- $h_\theta(x) = \boldsymbol{\theta}^T \boldsymbol{x}$
- ${}_{1}\overset{n+1}{[\theta_0\ \theta_1 \dots \theta_n]} \begin{bmatrix} x_0 \\ x_1 \\ \dots \\ x_n \end{bmatrix}^{1}_{n+1}$

Multivariate Gradient Descent

- The intuitions and formulas we saw previously are the same, there are just more partial derivatives to compute !

New algorithm $(n \geq 1)$:

Repeat $\{$

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

(simultaneously update θ_j for $j = 0, \ldots, n$)

$\}$

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

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

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

$\ldots$

Derivative rules (optional)

- Just so you are aware, no need to learn these by heart. But useful if you want to try and derive a function on your own !

- You can find these kinds of « cheatsheets » online if you need to.

	Function $f(x)$	Derivative with respect to x
1	a	0
2	x	1
3	ax	a
4	x^2	$2x$
5	x^a	ax^{a-1}
6	a^x	$\log(a)a^x$
7	$\log(x)$	$1/x$
8	$\log_a(x)$	$1/(x\log(a))$
9	$\sin(x)$	$\cos(x)$
10	$\cos(x)$	$-\sin(x)$
11	$\tan(x)$	$\sec^2(x)$

Derivative rules (optional)

- More useful rules

	Function	Derivative
Sum Rule	$f(x) + g(x)$	$f'(x) + g'(x)$
Difference Rule	$f(x) - g(x)$	$f'(x) - g'(x)$
Product Rule	$f(x)g(x)$	$f'(x)g(x) + f(x)g'(x)$
Quotient Rule	$f(x)/g(x)$	$[g(x)f'(x) - f(x)g'(x)]/[g(x)]^2$
Reciprocal Rule	$1/f(x)$	$-[f'(x)]/[f(x)]^2$
Chain Rule	$f(g(x))$	$f'(g(x))g'(x)$

Quick but useful example using the rules (optional)

- $f(\theta_1) = \frac{(\theta_1 x_1 - y_1)^2}{2}$
- Power rule : $x^2 \Rightarrow 2x$
- Scalar multiplication rule: $ax \Rightarrow a$
- Chain rule: $\frac{d}{dx} f\big(g(h(x))\big) \Rightarrow \frac{df}{dg} \times \frac{dg}{dh} \times \frac{dh}{dx}$

(Optional)

- $f(a) = \frac{(ax - y)^2}{2}$
- Let’s decompose this into 3 functions:
  - $g(a) = ax \ - y$
  - $h(g) = g^2 \ where \ g \ is \ (ax - y)$
  - $i(h) = \frac{h}{2} \ where \ h \ \ is (ax - y)^2$
- Let’s derive these functions 1 by 1 using the rules in the previous slides.

(Optional)

- $g'(a) = \frac{d}{da}(ax - y) = x$
- $h'(g) = \frac{\mathrm{d}}{dg}(\mathrm{g}^2) = 2\mathrm{g}$
- $i'(h) = \frac{d}{dh}\left(\frac{1}{2}h\right) = \frac{1}{2}$

- Using the chaine rule, we multiply these dervatives to get the derivative of our original function :

$$f'(a) = \frac{di}{dh} \times \frac{dh}{dg} \times \frac{dg}{da}$$

$$= \frac{1}{2} \times 2(ax - y) \times x$$

$$= (ax - y) \times x$$