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

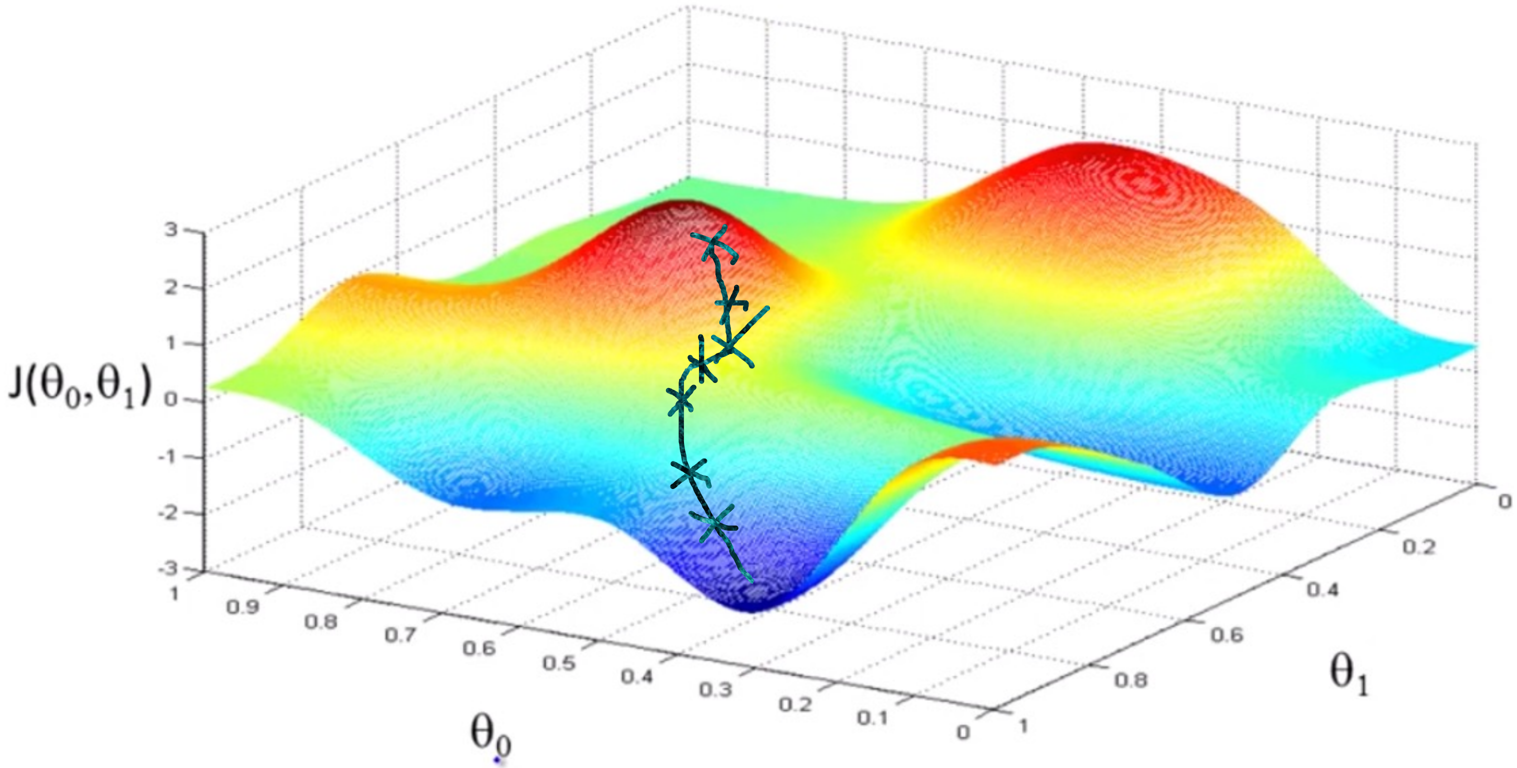
- We have some function $J(\theta_1, \theta_2)$ which we want to minimize...

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 end up at a **minimum** (global or local)

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, which requires 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

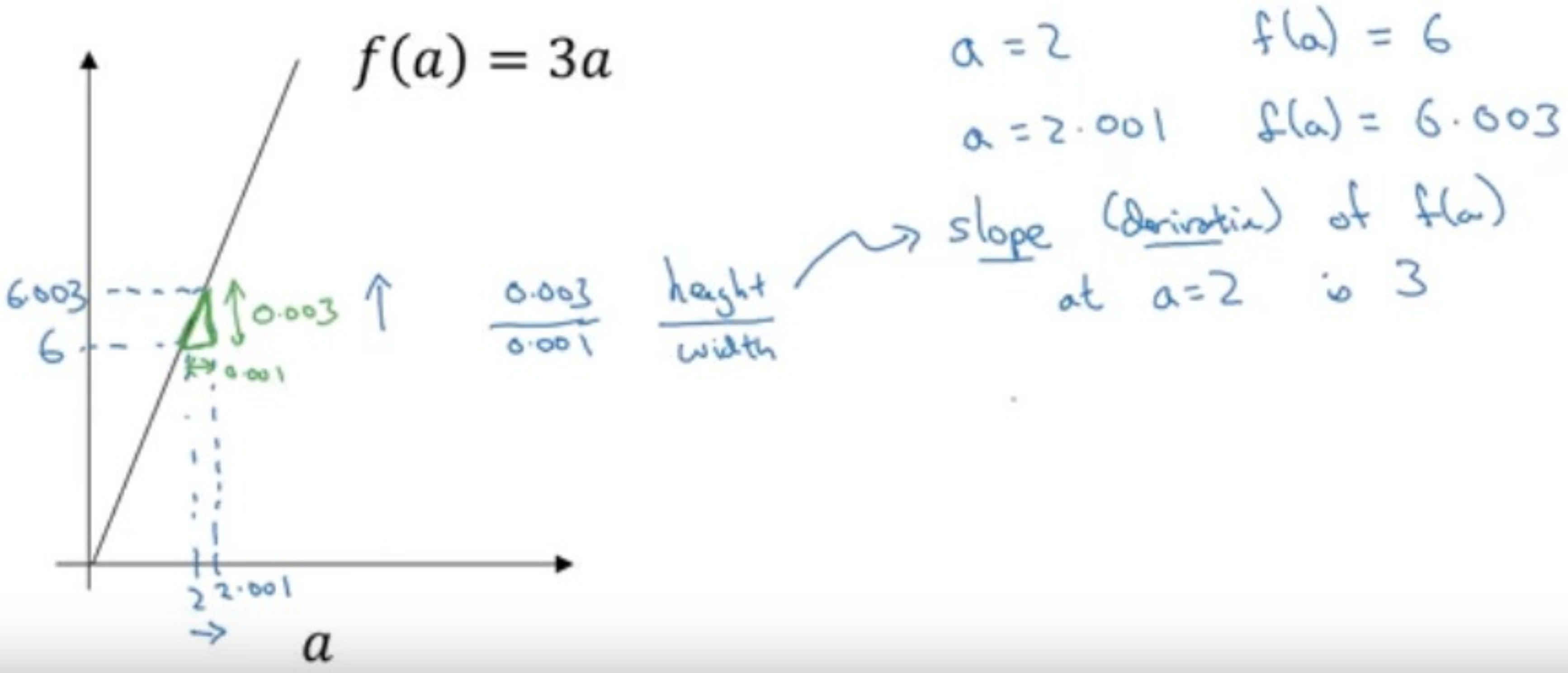
- The derivative describes **how the output of a function varies** with regard to a very very very **tiny positive nudge** to the **input**, to the point where we consider *almost* no variation in input....

- 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 « **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 ?



cf. 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 :

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

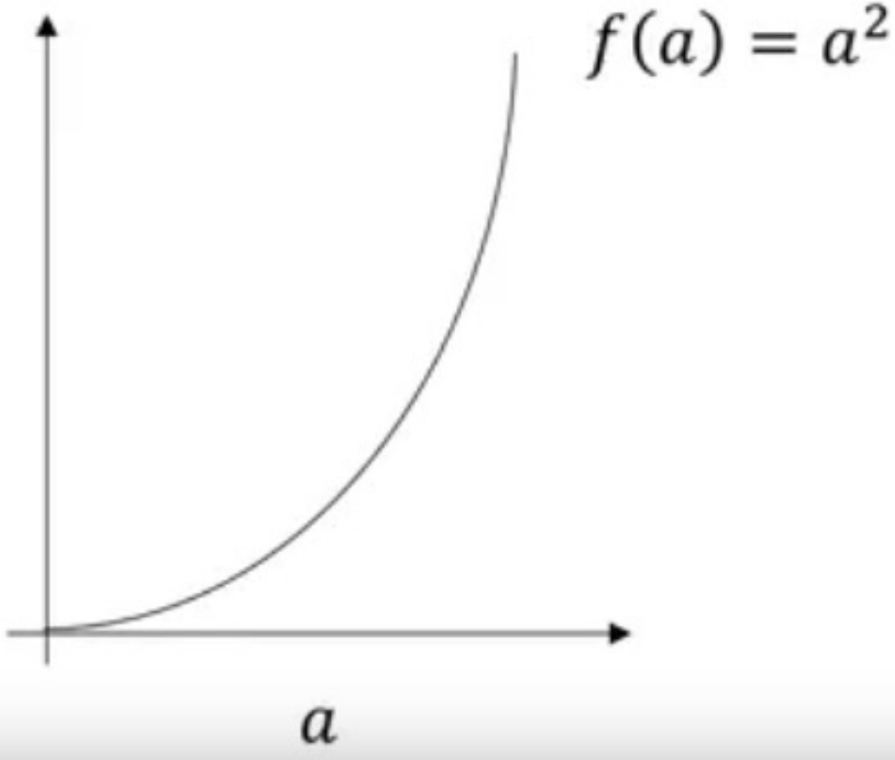
Derivatives

- 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.
- This makes sense since the **function is a line** and the output 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 ?

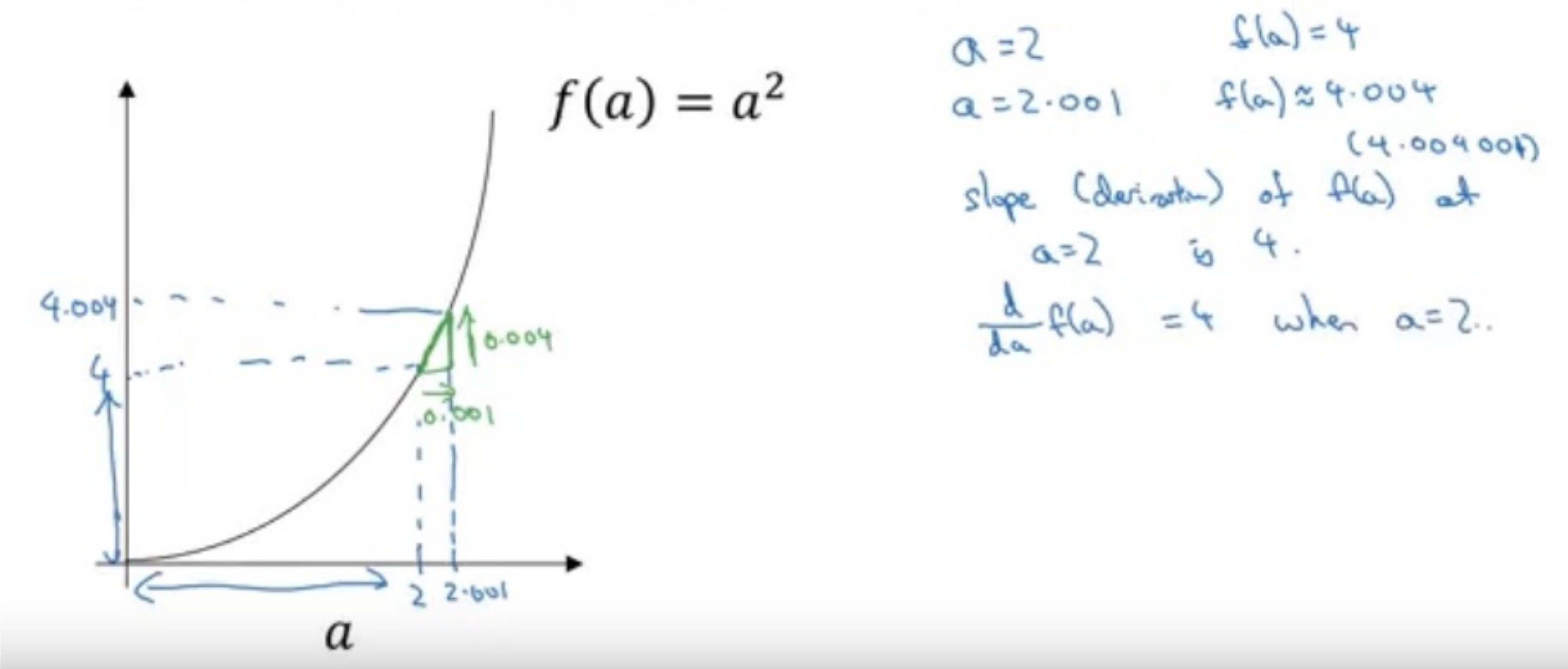


$a = 2$ $\quad f(a) = 4$

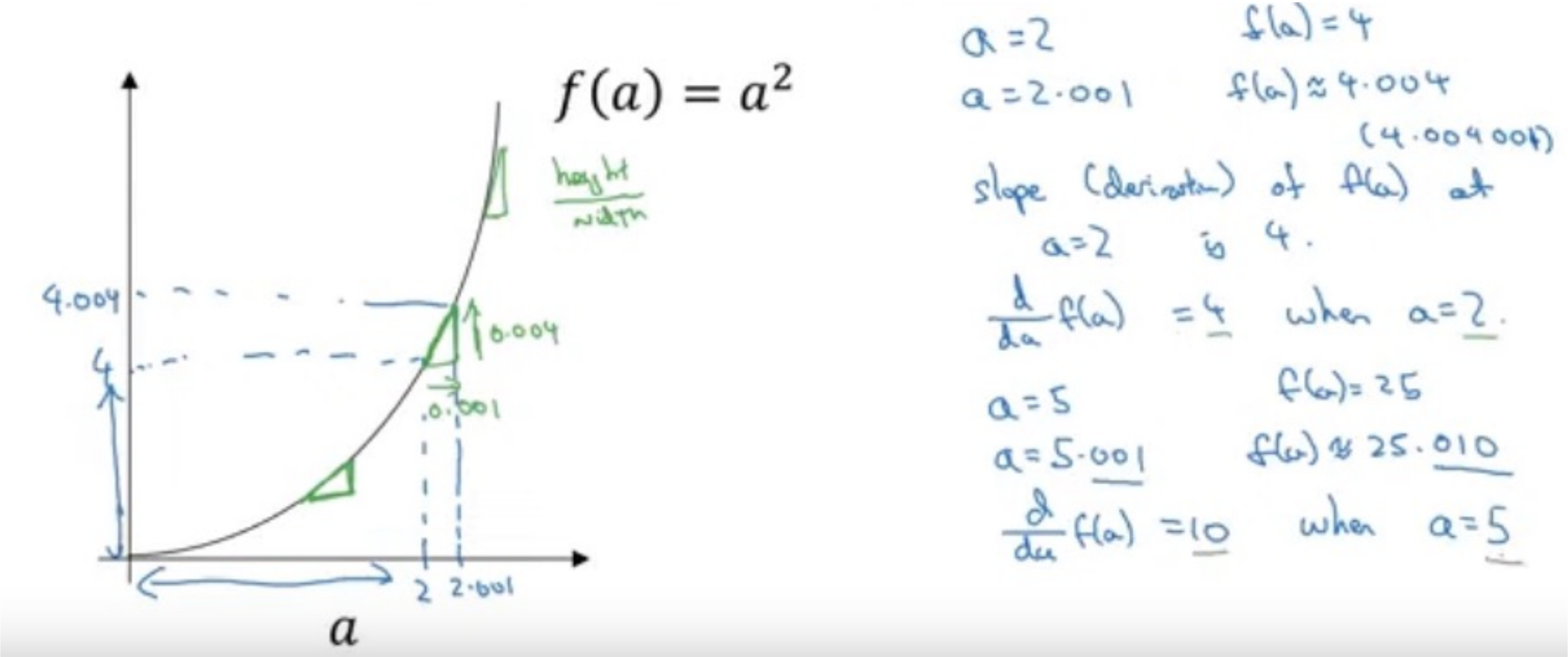
$a = 2.001$ $\quad f(a) \approx 4.004$

(4.004001)

- The derivative at a=2 is …



- The derivative at a=5 is ...



- Rules exist to compute derivatives
- For example, 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 !
- Note: the derivative is equal to the **slope of the tangent line** 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

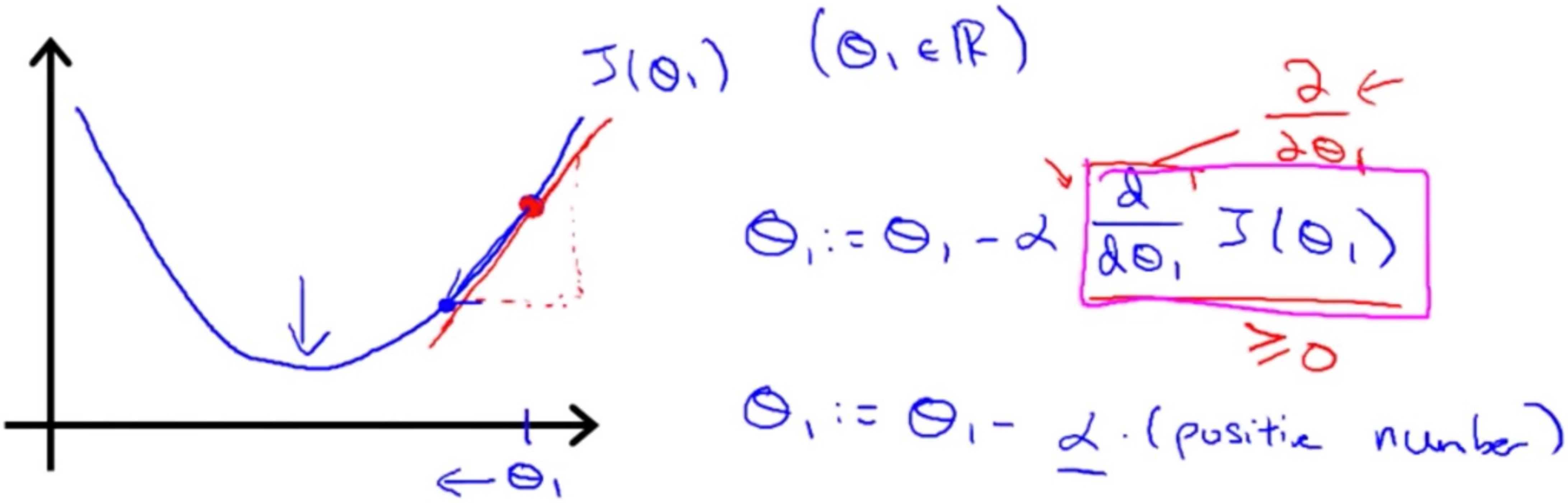
- Now we have a basic understanding of derivatives, let's apply this understanding to the gradient descent algorithm by using **a simpler example**, with a **cost function of only 1 single parameter**.

- We use $\boldsymbol{J(\theta_1)}$ 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…

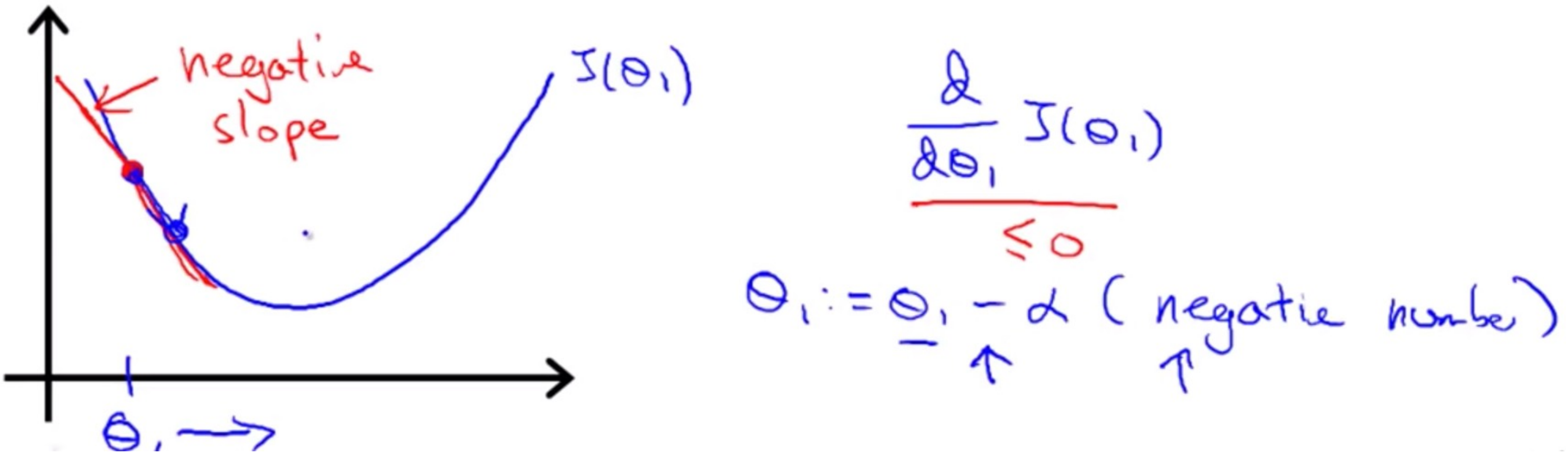
- Remember, our cost function looks like a parabola.
- When θ_1 is too high, we want our optimizer to **reduce** this parameter and bring it closer to the « **sweet spot** », where the cost is minimized.
- Let's see if it does the right thing :



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

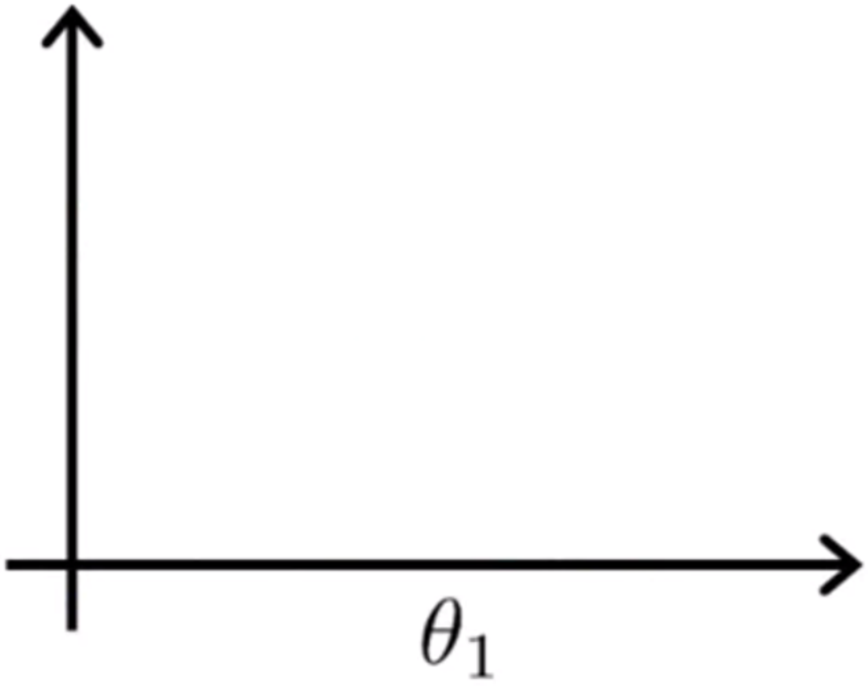
- Conversely, when the parameter value is too low, the parameter value will be increased by the update rule.

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

< 0

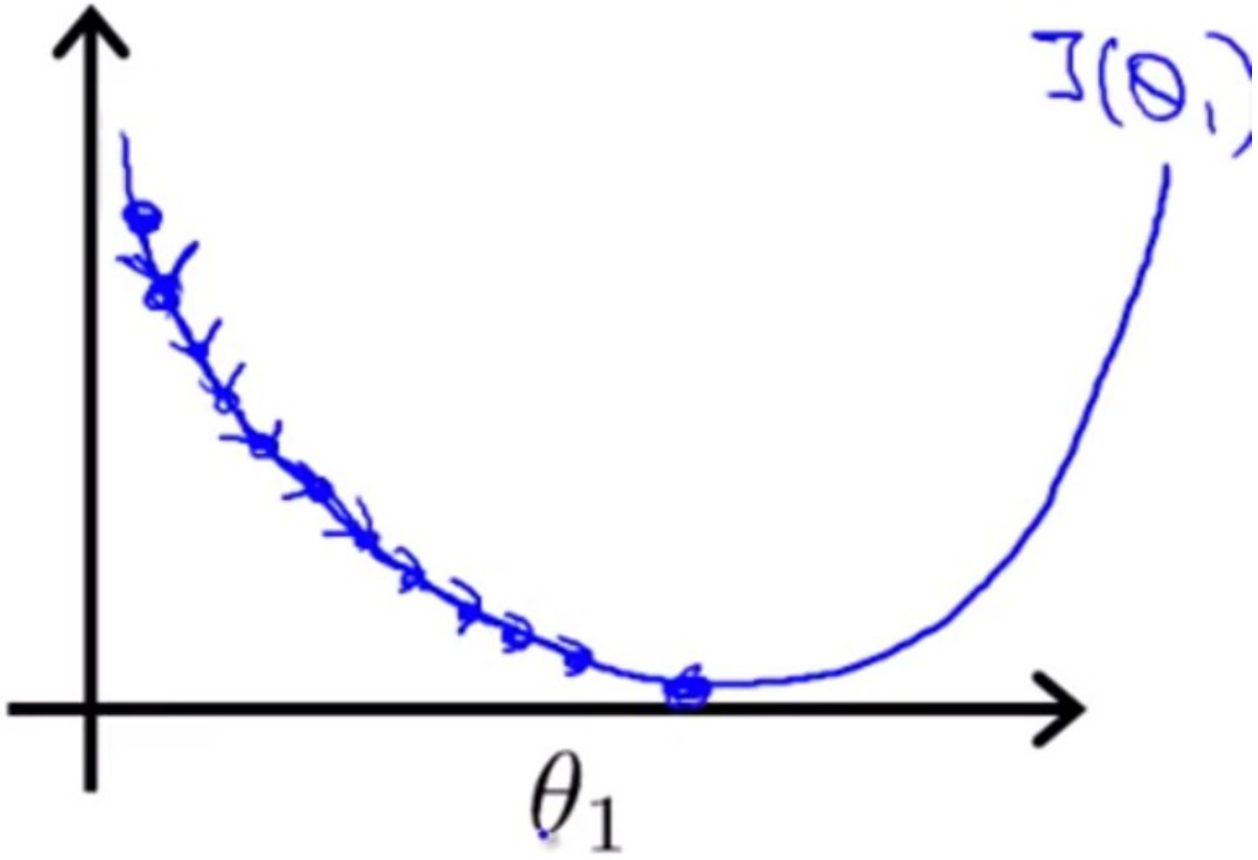
Okay so now what about α ?

- 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 :



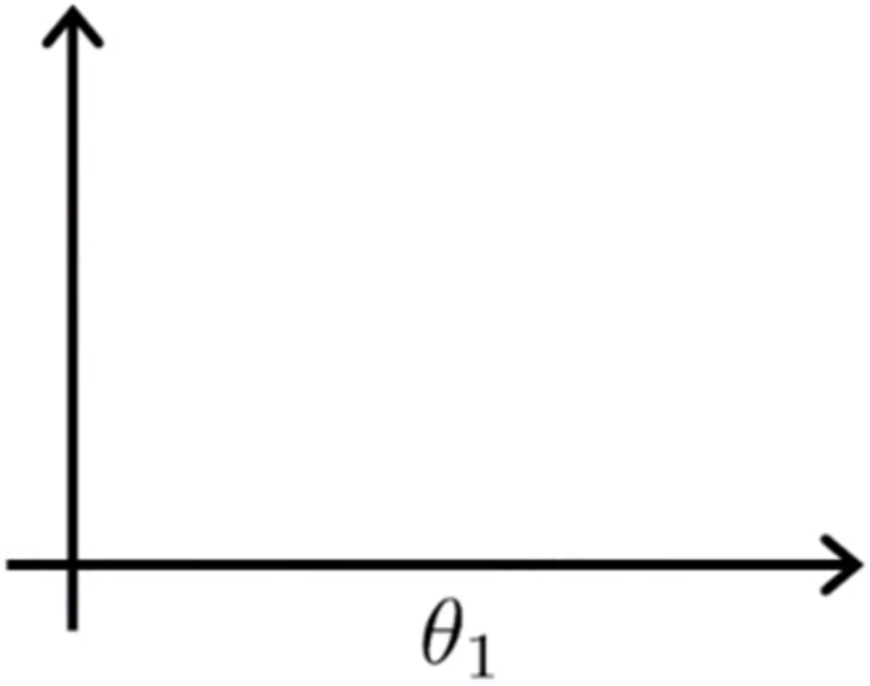
If α is too small

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

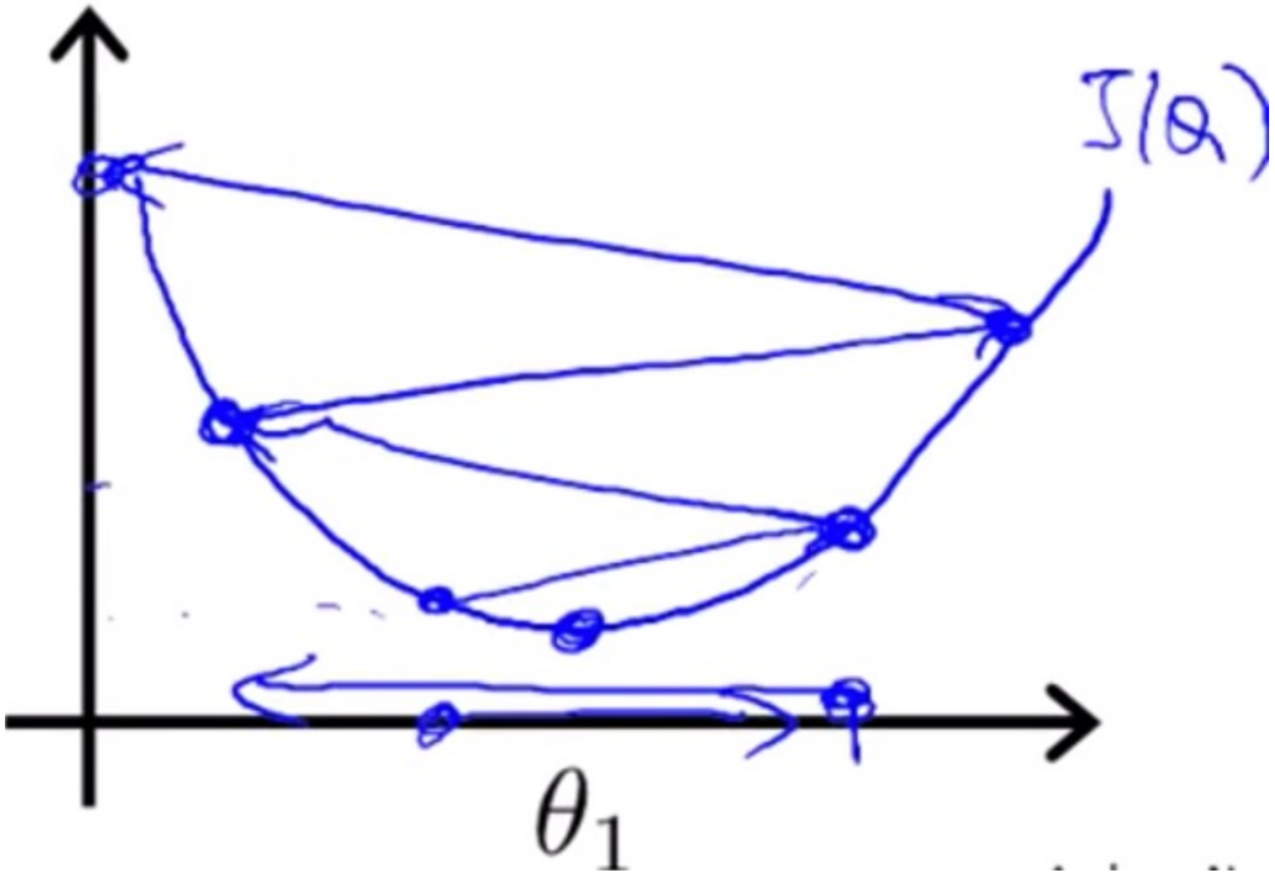


If α is too large…

- 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

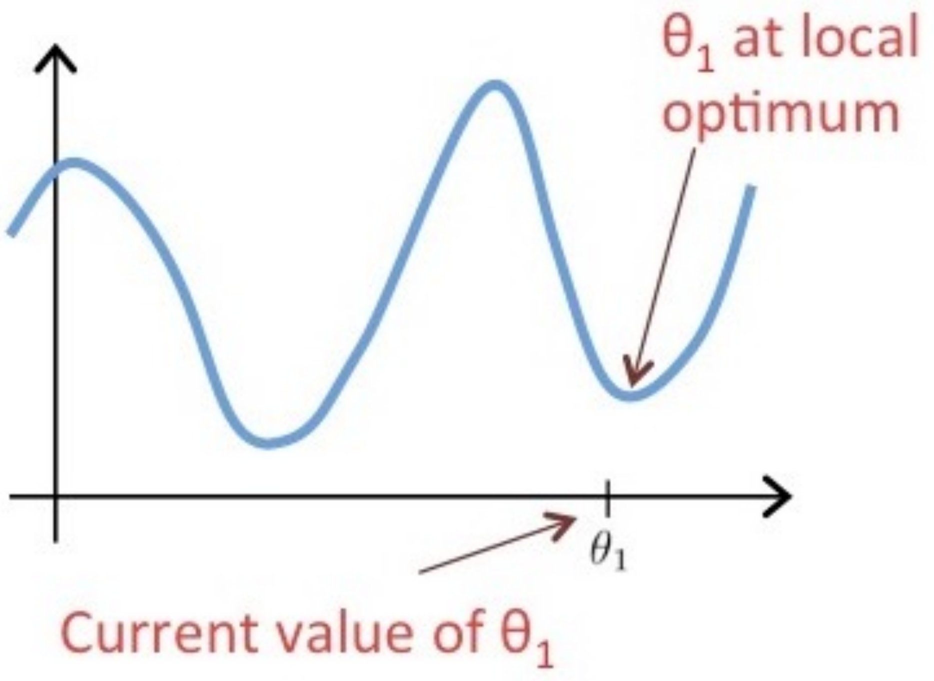


Question

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?

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 ?



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 or variant** (to update our parameters and get closer to a better model)

Gradient descent algorithm

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

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 into a vector, 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 basically 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 visually

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

https://www.youtube.com/watch?v=dfvnCHqzK54&t=1s

Gradient Descent

- Each partial derivative tells us how the function **behaves** (increases/decreases quickly/slowly, or 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** : go the opposite direction. If **gradient** values are **positive**, this means a positive nudge applied to the weights **increases** the cost: so we need to decrease the weights. And vice-versa.

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.

- 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 know how 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 fits the data:

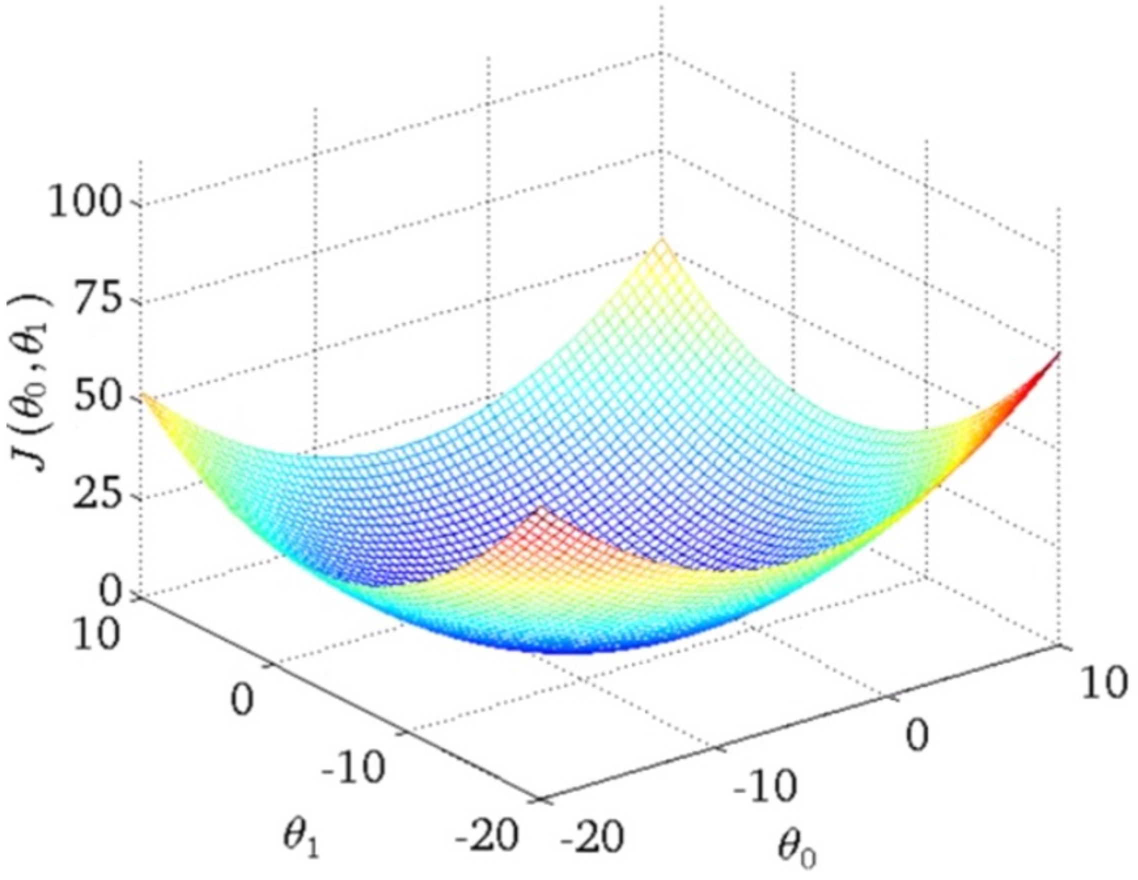
$$J(\theta_0,\theta_1) = \frac{1}{2m}\sum_{i=1}^{m}(h\left(x^{(i)}\right) - y^{(i)})^2$$

- **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

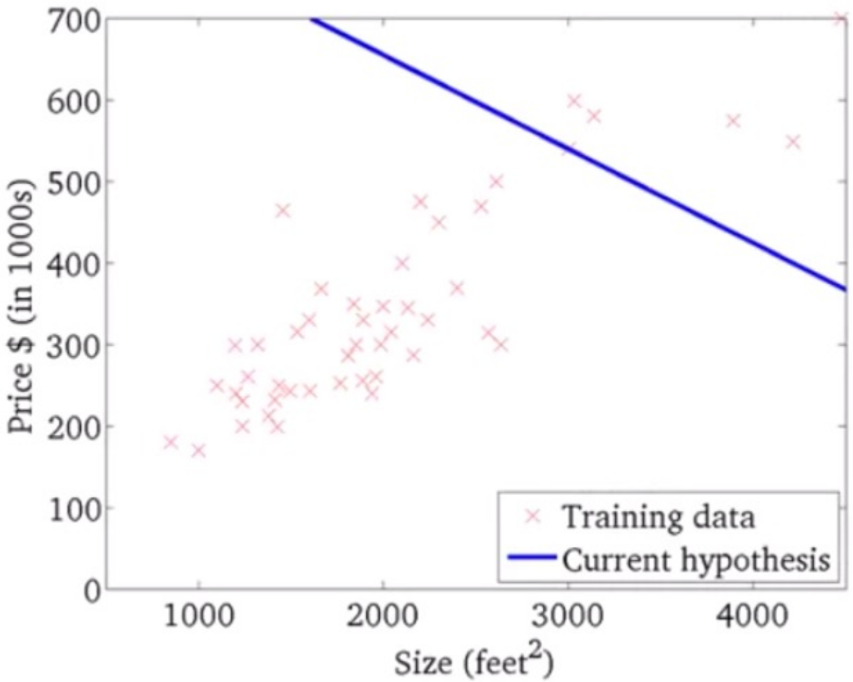


Update examples

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

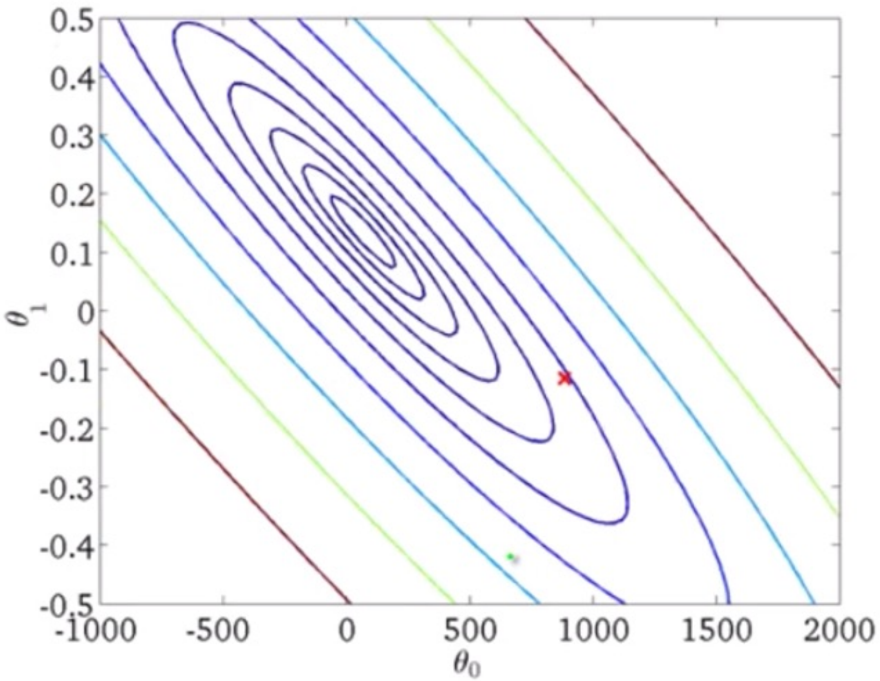
$h_\theta(x)$

(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)

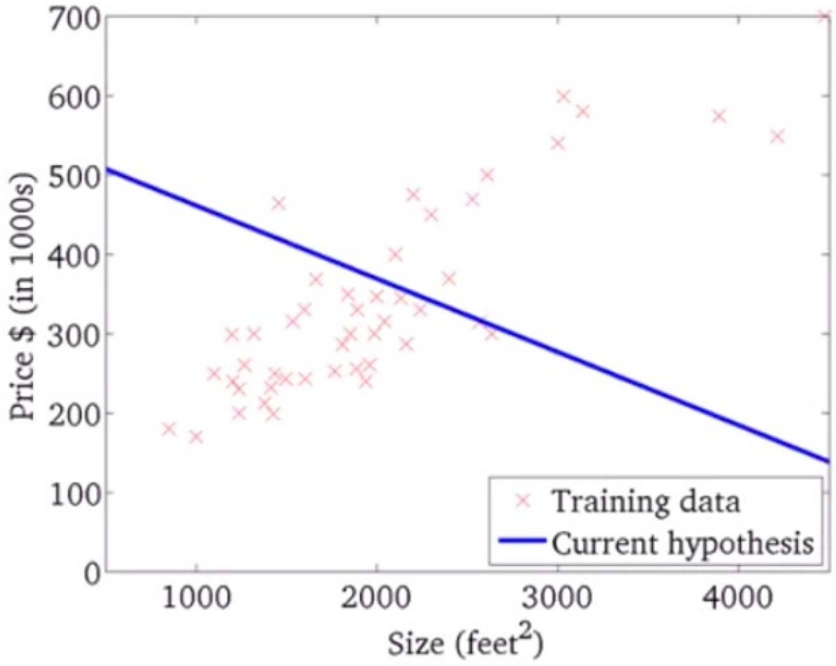


Update examples

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

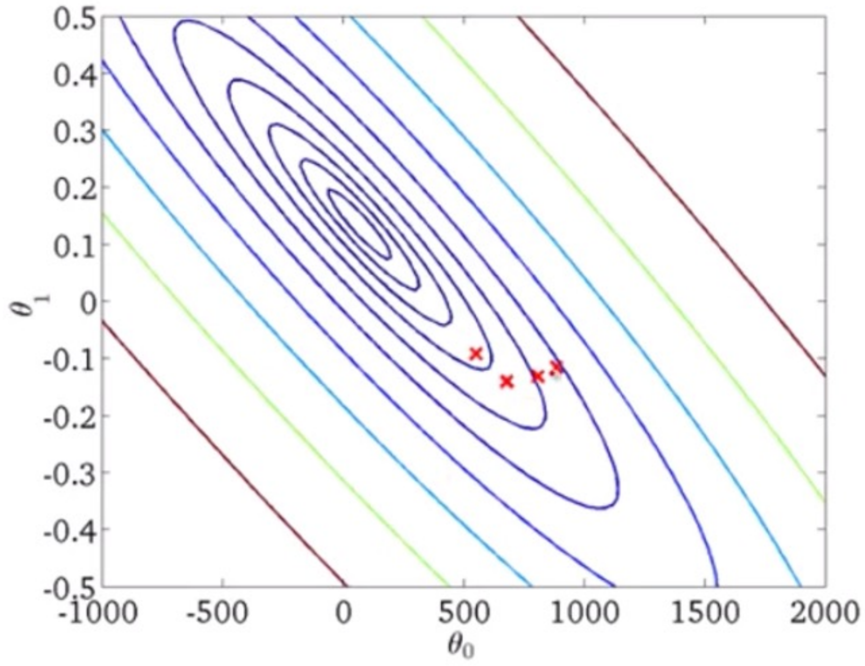
$h_\theta(x)$

(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)

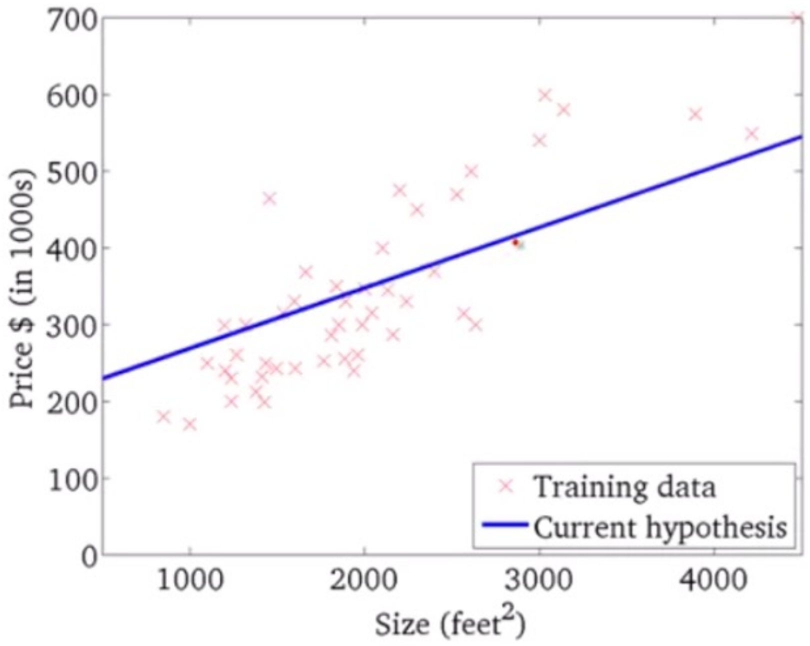


Update examples

Until we reach the global minimum

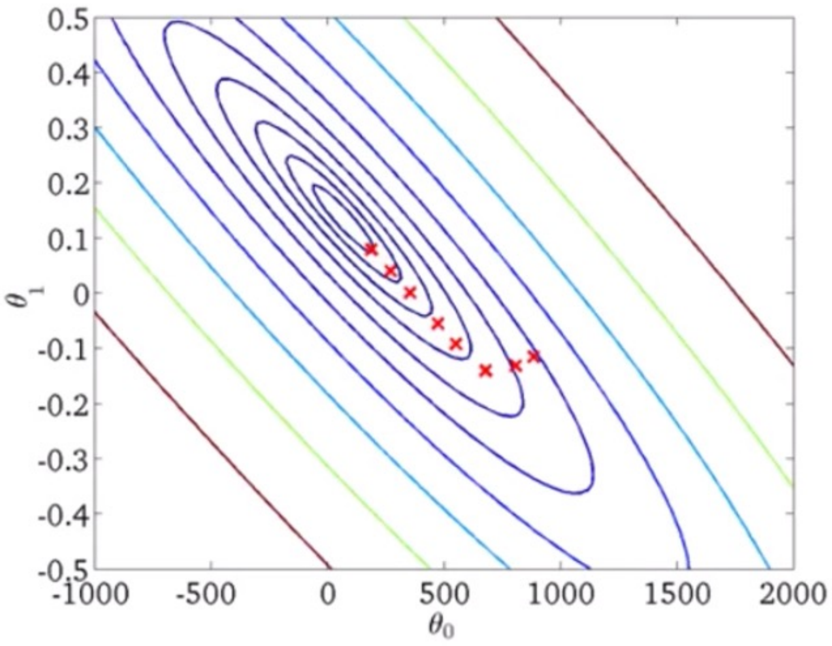
$h_\theta(x)$

(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)



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 $\boldsymbol{x}^{(1)}$ 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
…	…	…	…	…

- $\boldsymbol{x}^{(1)} = \begin{bmatrix} 1416 \\ 3 \\ 2 \\ 40 \end{bmatrix}$
- $\boldsymbol{x}_3^{(1)} = 2$

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:

- $\boldsymbol{x} = \begin{bmatrix} x_1 \\ x_2 \\ \dots \\ x_n \end{bmatrix}$ $\neq$ $\boldsymbol{\theta} = \begin{bmatrix} \theta_0 \\ \theta_1 \\ \theta_2 \\ \dots \\ \theta_n \end{bmatrix}$ $\qquad \boldsymbol{x} = \begin{bmatrix} x_0 \\ x_1 \\ x_2 \\ \dots \\ x_n \end{bmatrix}$

  Size = n — 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** !

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

Multivariate Gradient Descent

- The intuitions and formulas we saw previously are the same, there are just more partial derivatives to compute (1 per param.) !

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)

	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	$f(g(x))$	$f'(g(x))g'(x)$