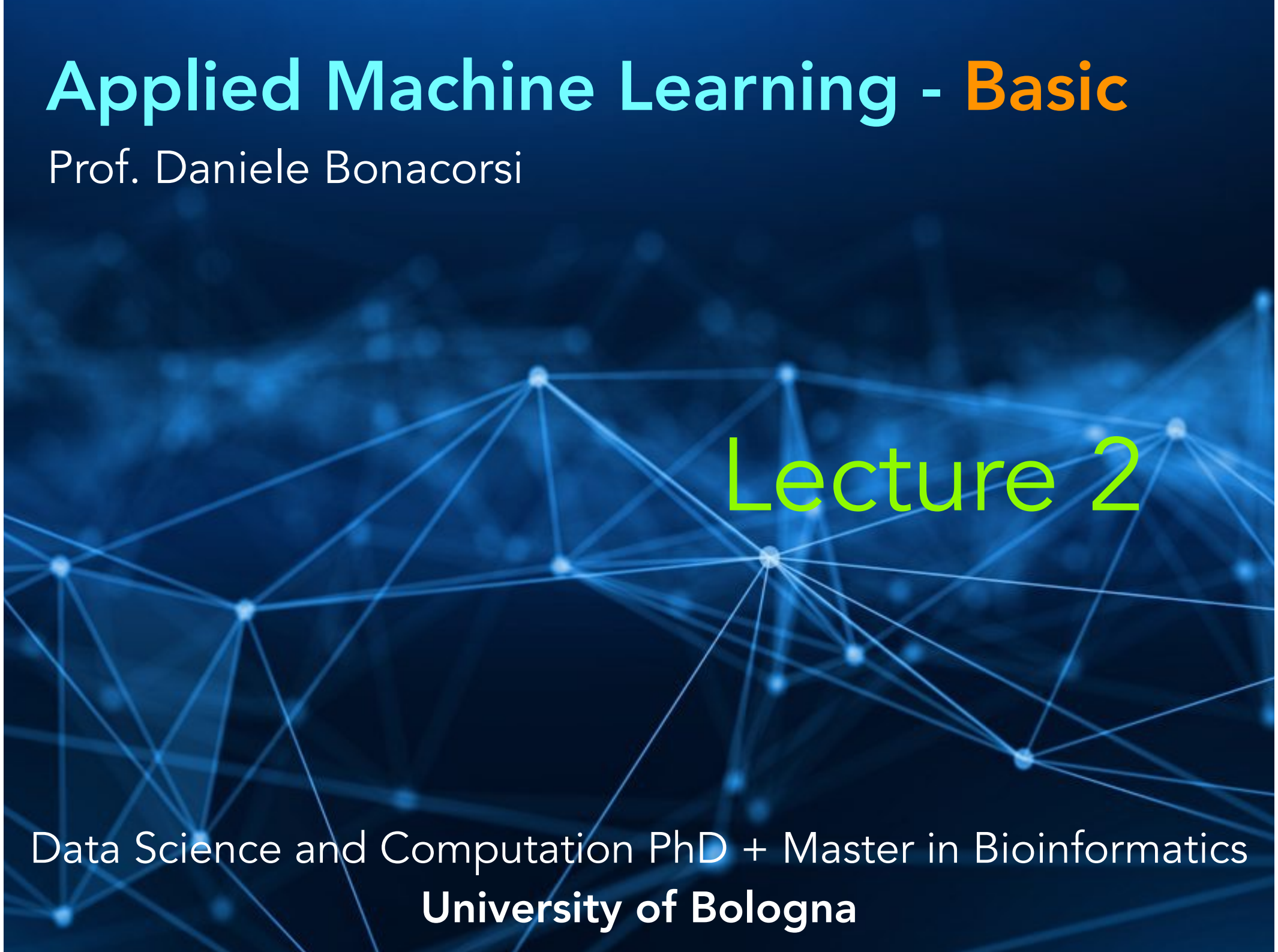
Applied Machine Learning - Basic

Prof. Daniele Bonacorsi



Lecture 2

Data Science and Computation PhD + Master in Bioinformatics

University of Bologna


Model representation
Univariate Linear regression
Gradient Descent

Note

Gradient Descent introduced here in linear regression, but used elsewhere.

Gradient Descent (GD) algorithm

You have some function $J(\theta_0, \theta_1)$

You want $\min_{\theta_0, \theta_1} J(\theta_0, \theta_1)$

Algo:

- start with some θ_0, θ_1
- Keep changing θ_0, θ_1 to reduce $J(\theta_0, \theta_1)$
- stop when you (hopefully!) reach a min

Gradient Descent (GD) algorithm

You have some function $J(\theta_0, \theta_1)$

You want $\min_{\theta_0, \theta_1} J(\theta_0, \theta_1)$

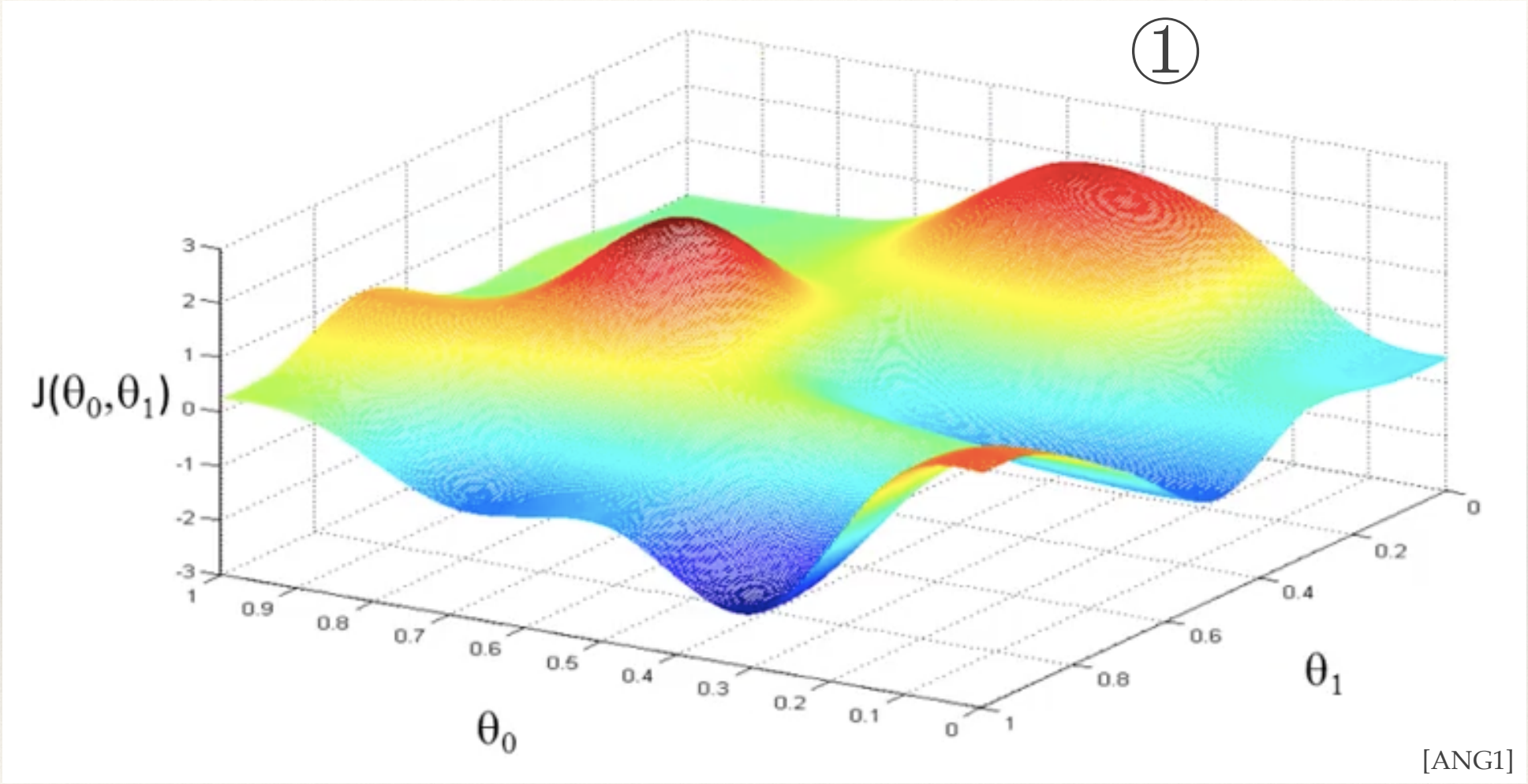
you can generalize to $J(\theta_0, \theta_1, \dots, \theta_n)$

$\min_{\theta_0, \theta_1, \dots \theta_n} J(\theta_0, \theta_1, \dots, \theta_n)$

Algo:

- start with some θ_0, θ_1
- Keep changing θ_0, θ_1 to reduce $J(\theta_0, \theta_1)$
- stop when you (hopefully!) reach a min

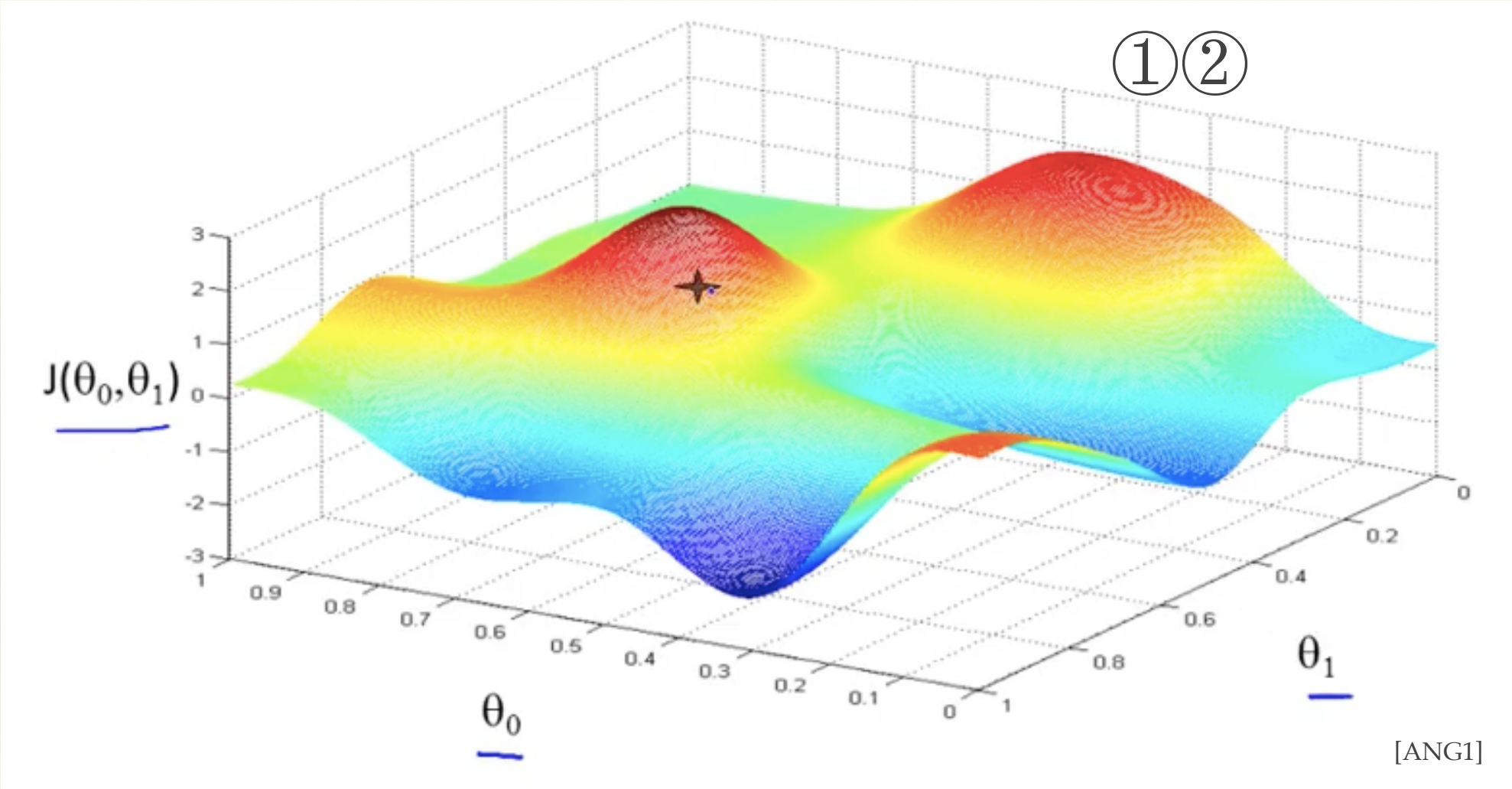
Note the axes.



[ANG1]

(with 2 θs you can at least plot it 3D - but you can think and generalise to n θs..)

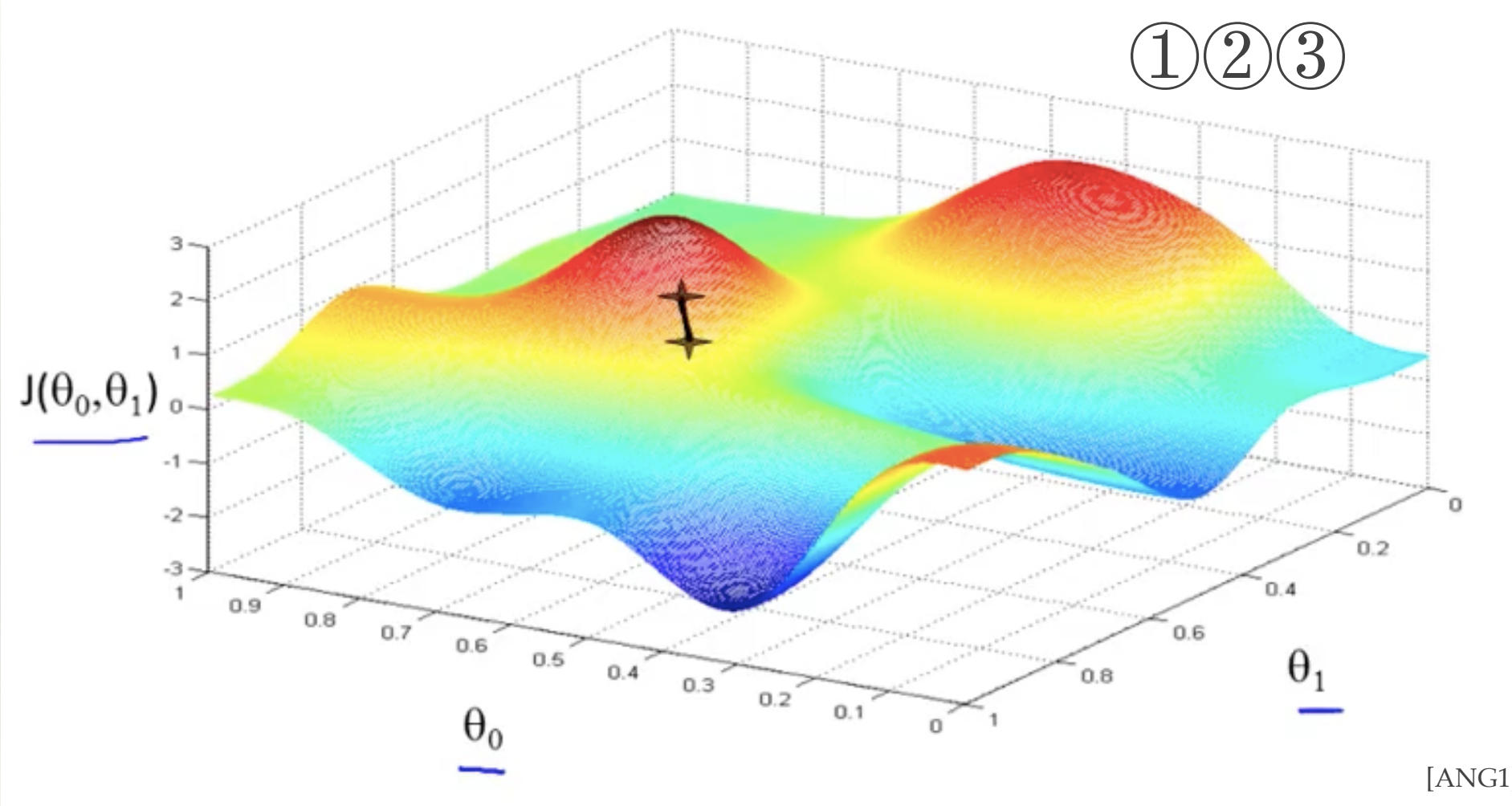
Pick a starting point, i.e. a given (θ_1, θ_2) couple.



[ANG1]

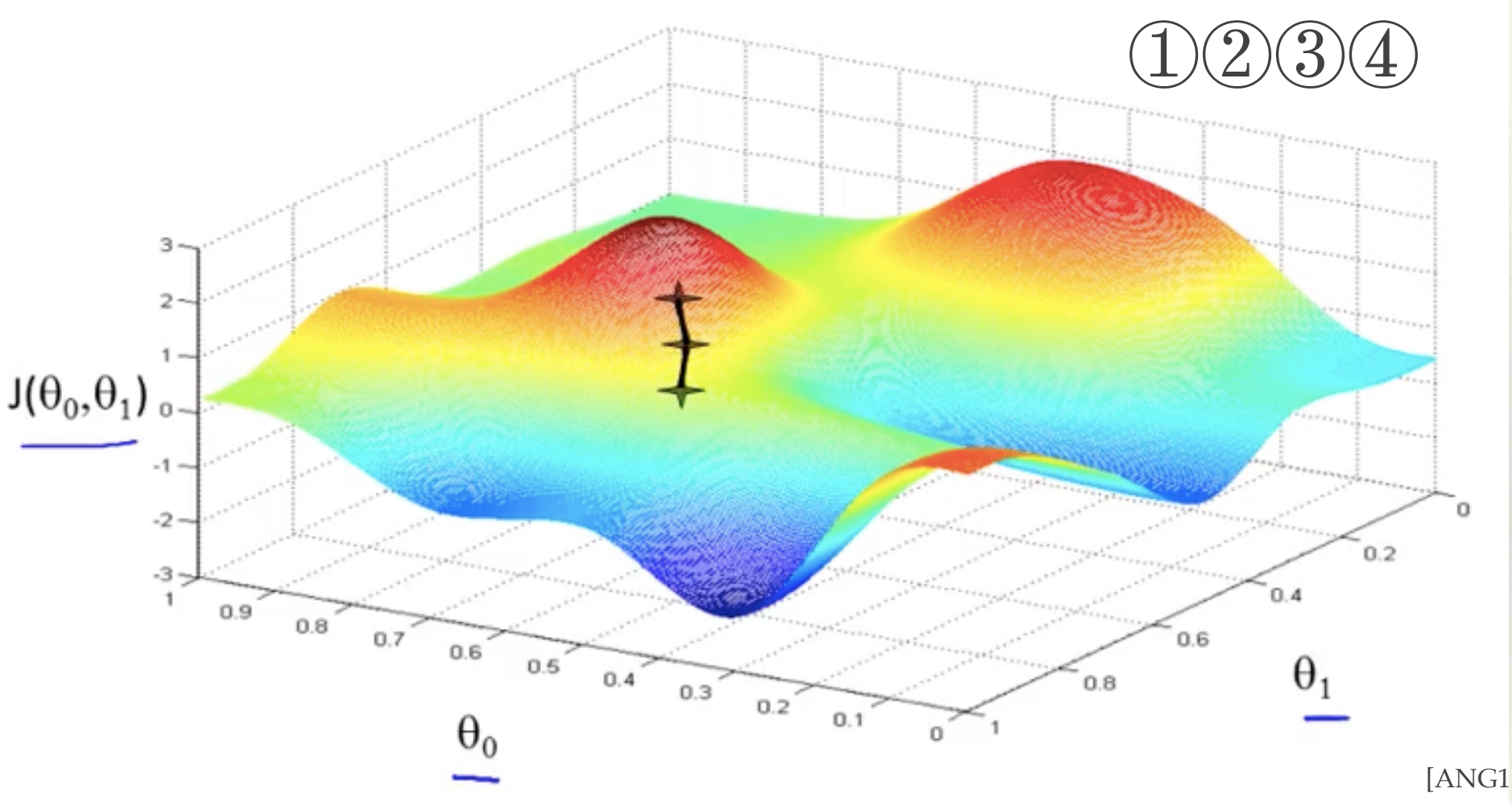
From the starting point, think physically as if these were hills: look around 360 degrees and make a step in the direction where I am going down quicker.

This is roughly the direction I should take.

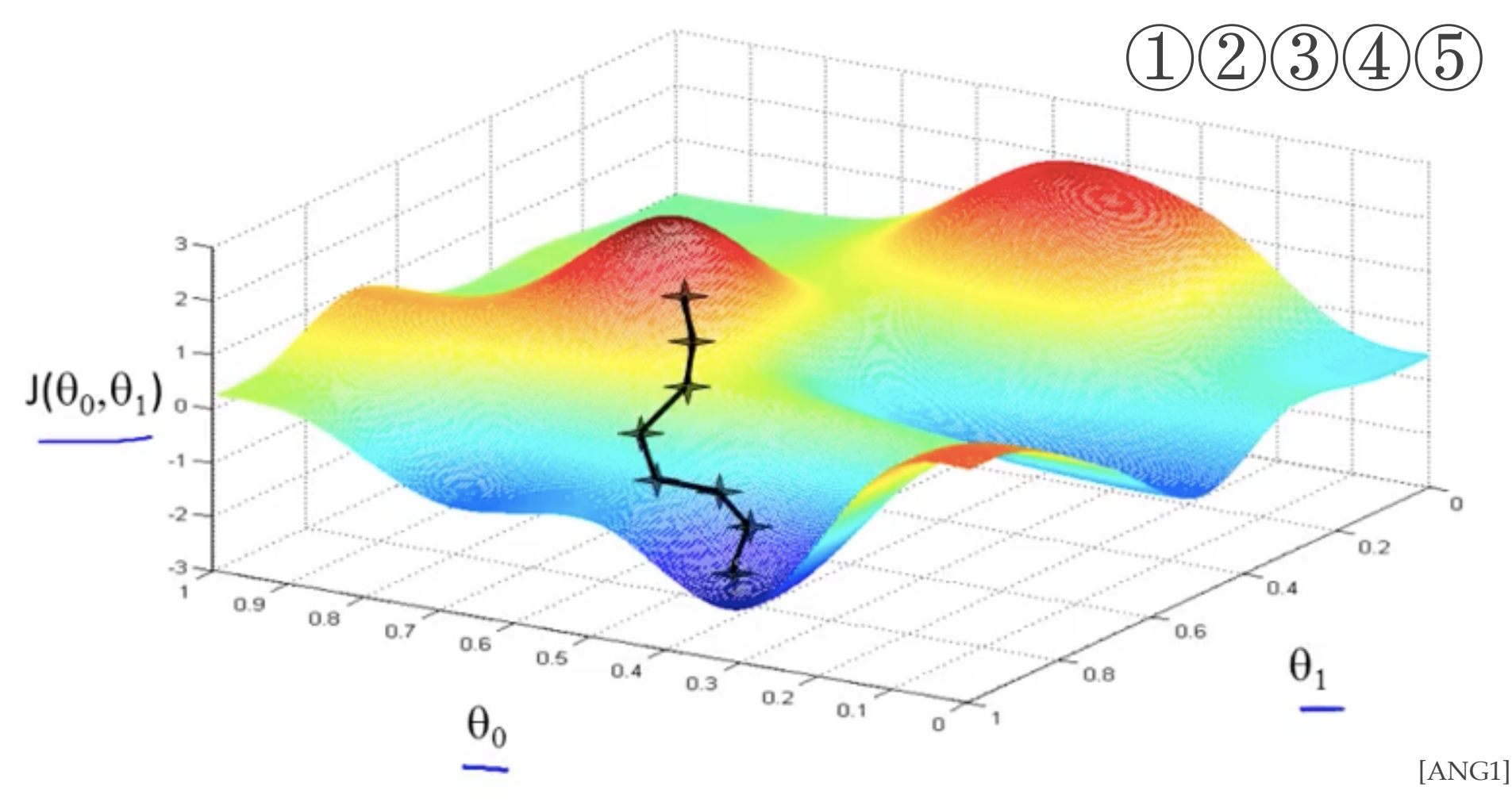


[ANG1]

And now?

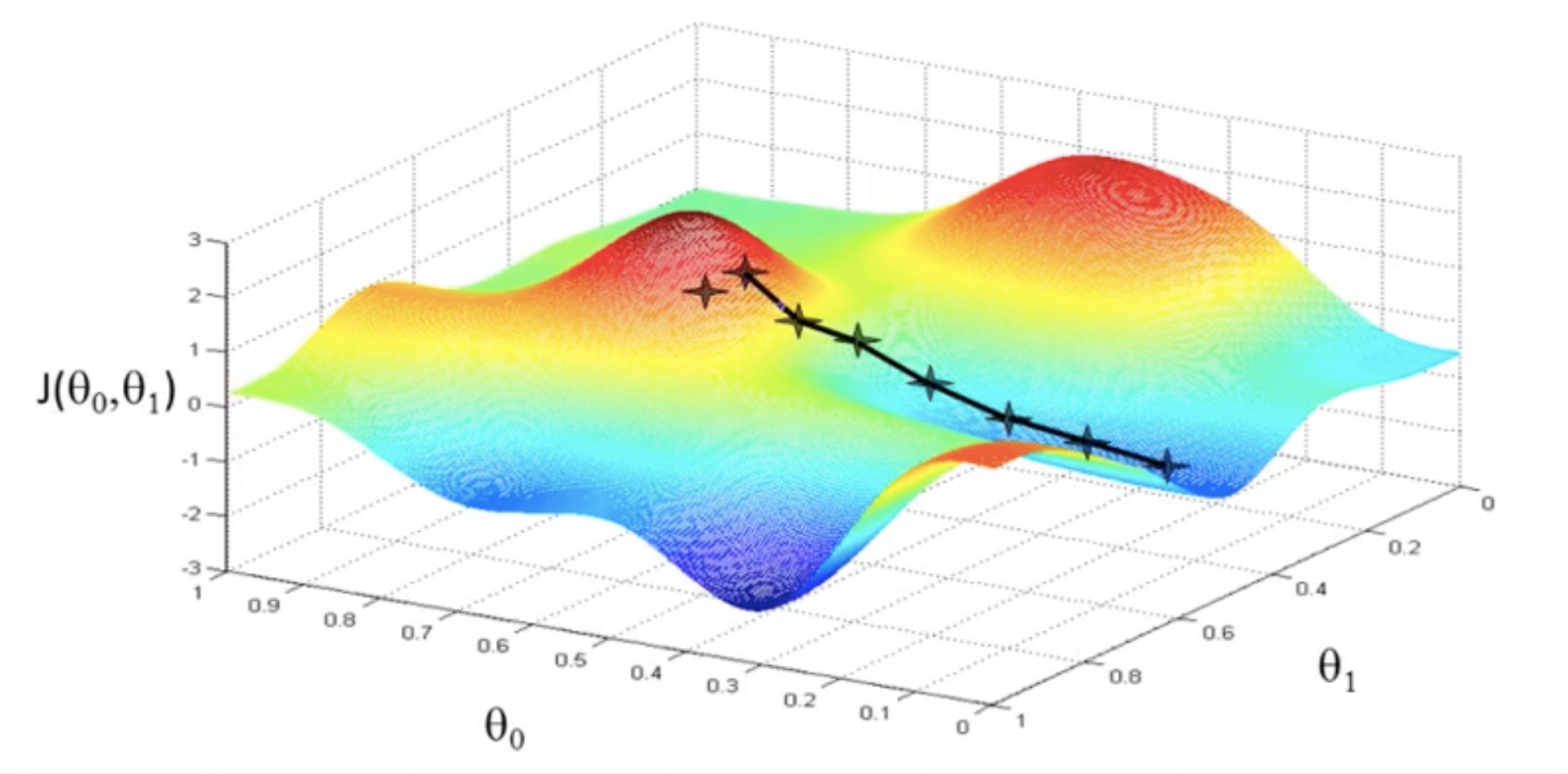


[ANG1]


①②③④⑤
J(θ0,θ1)
θ0
θ1
3
2
1
0
-1
-2
-3
1
0.9
0.8
0.7
0.6
0.5
0.4
0.3
0.2
0.1
0
0.2
0.4
0.6
0.8
1

[ANG1]

If I had started just a couple of steps to the right…



[ANG1]

… GD would have taken you to a **different local minimum**. This is a property of GD.

Definition of GD algo

repeat until convergence {

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

simultaneous update $j = 0$ and $j = 1$

}

feature index number

LEARNING RATE

it controls how aggressive the GD is

Implementation of GD algo

Let's look and digest all its parts.

- Firstly, let's look at the **θs**

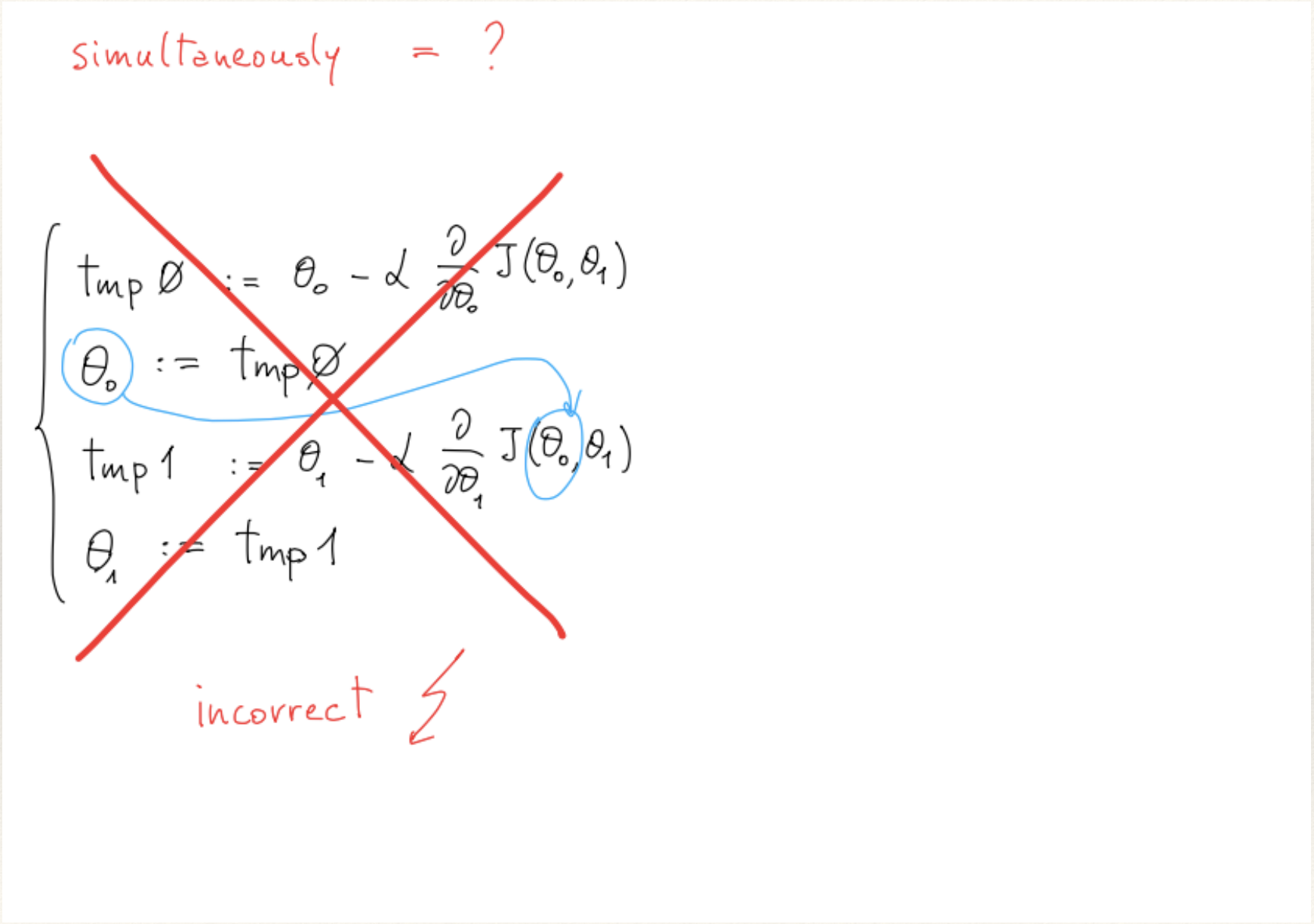
Implementation of GD algo: θs

simultaneously = ?

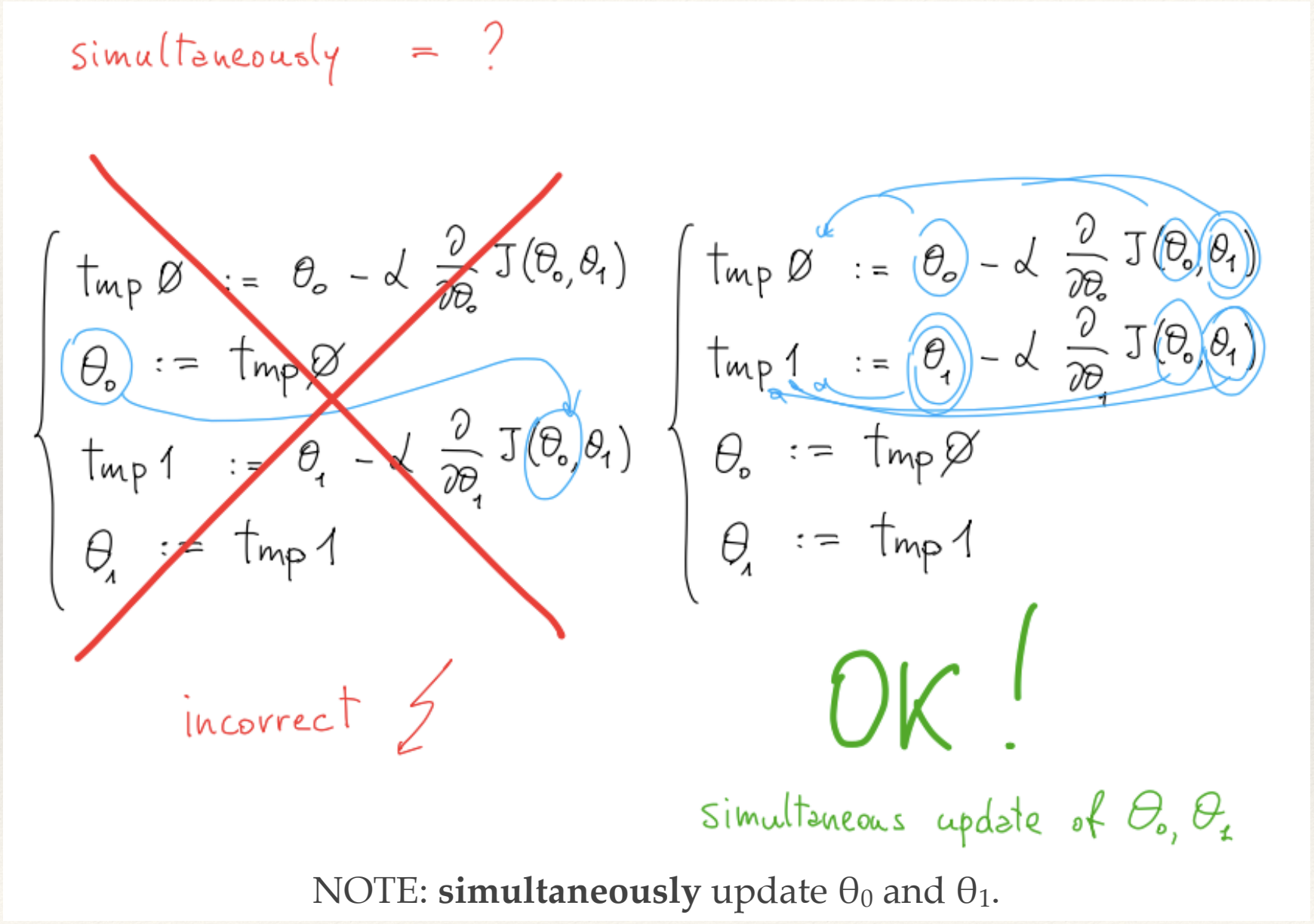
$$\begin{cases} tmp\,\emptyset := \theta_0 - \alpha \dfrac{\partial}{\partial \theta_0} J(\theta_0, \theta_1) \\ \theta_0 := tmp\,\emptyset \\ tmp\,1 := \theta_1 - \alpha \dfrac{\partial}{\partial \theta_1} J(\theta_0, \theta_1) \\ \theta_1 := tmp\,1 \end{cases}$$

QUIZ: is this correct?

Implementation of GD algo: θs



Implementation of GD algo: θs



NOTE: **simultaneously** update θ_0 and θ_1.
Be careful about a correct implementation of GD!

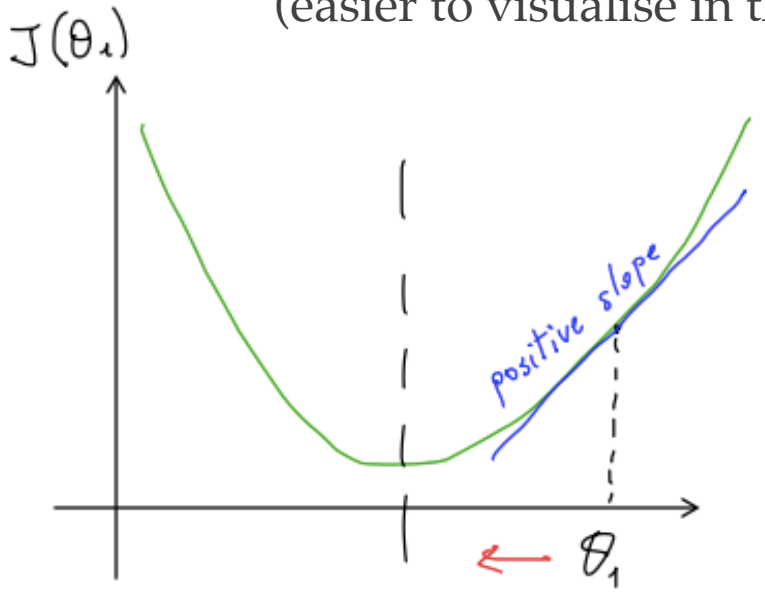
Implementation of GD algo

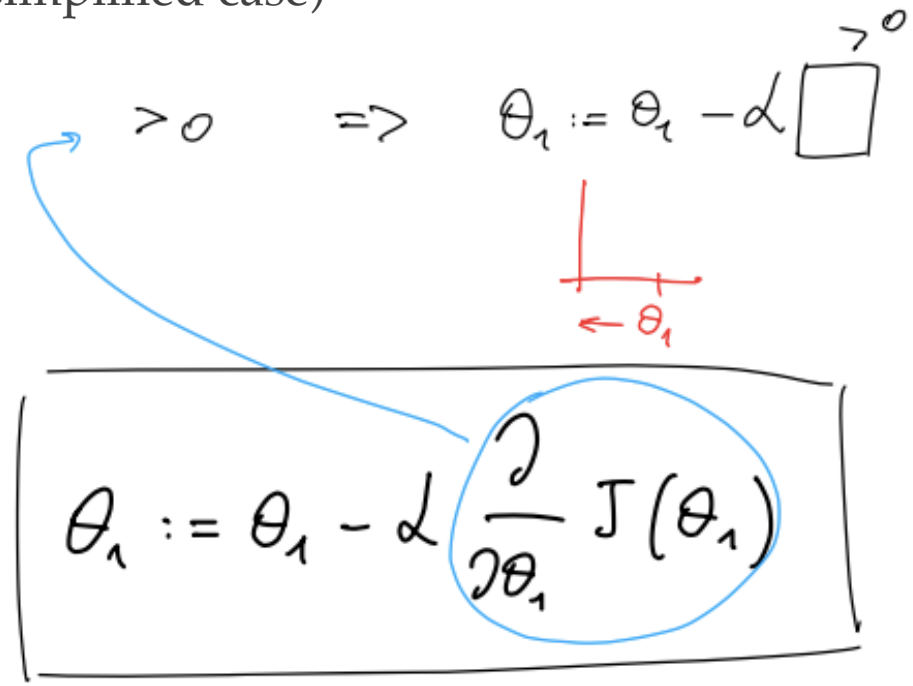
Let's look and digest all its parts.

- Firstly, let's look at the **θs**
- Secondly, let's look at the **derivative** term

Implementation of GD algo: **derivative**

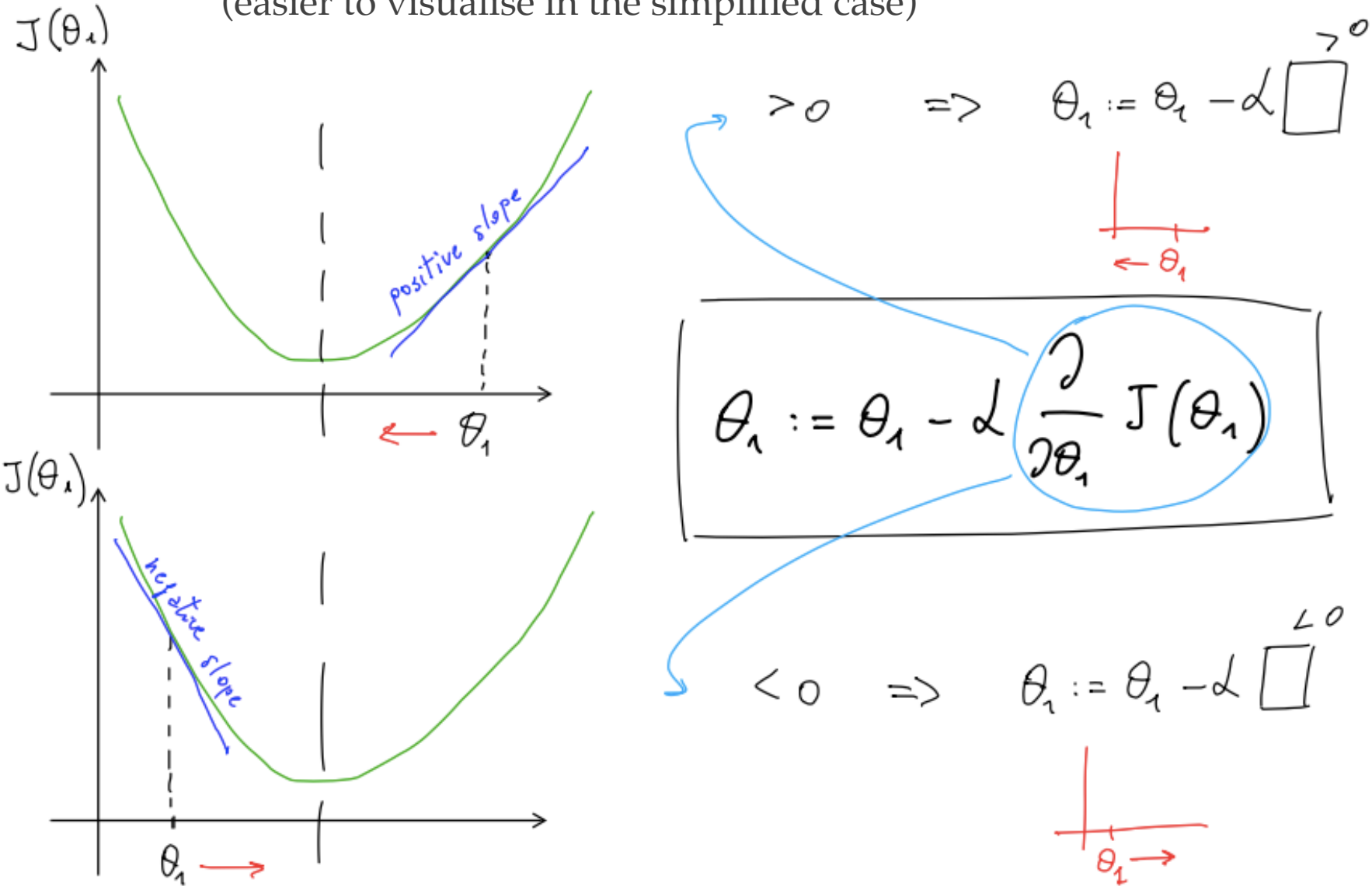
(easier to visualise in the simplified case)





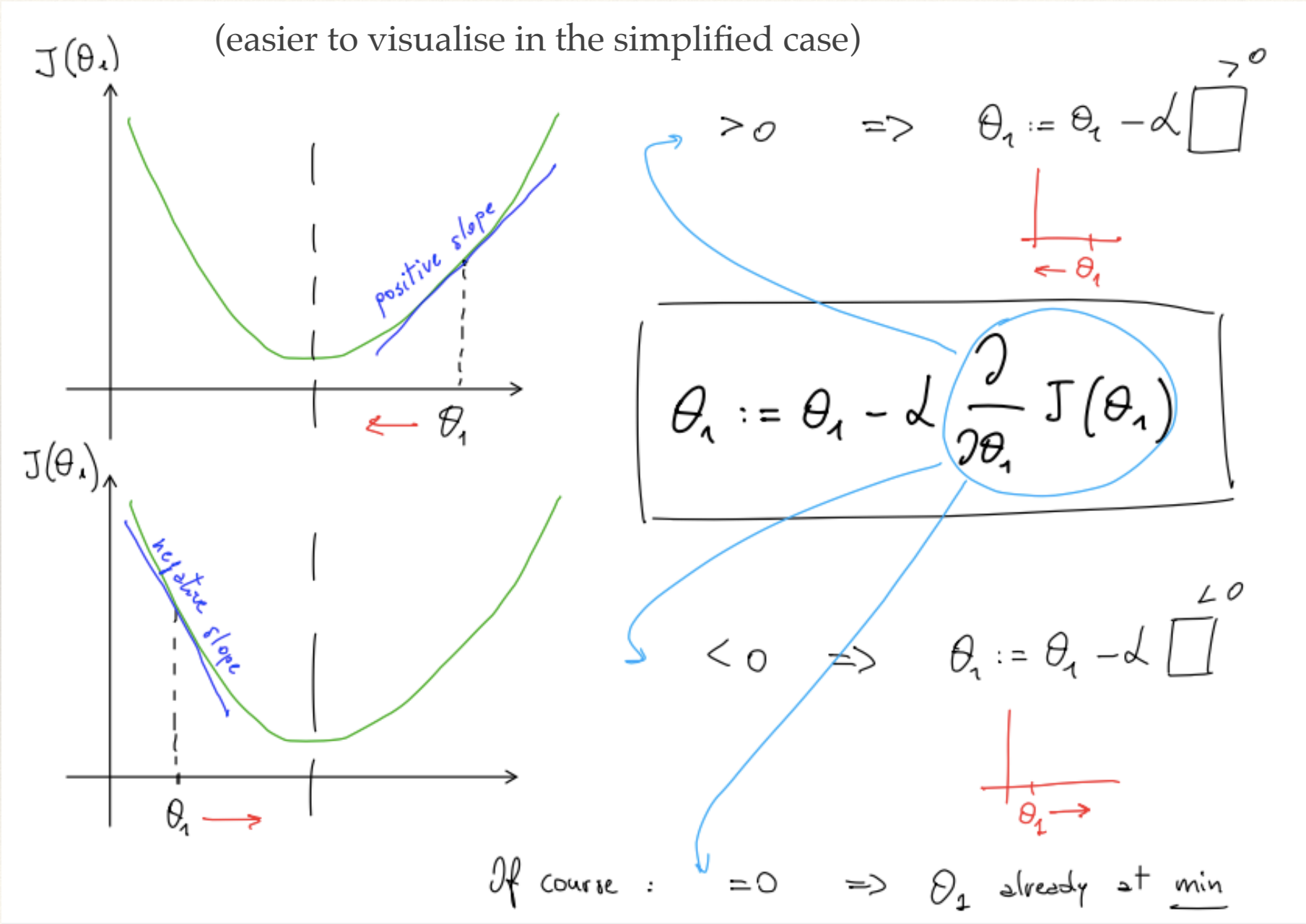
Implementation of GD algo: **derivative**

(easier to visualise in the simplified case)



Gradient tells you in which direction to go.

Implementation of GD algo: **derivative**



Implementation of GD algo

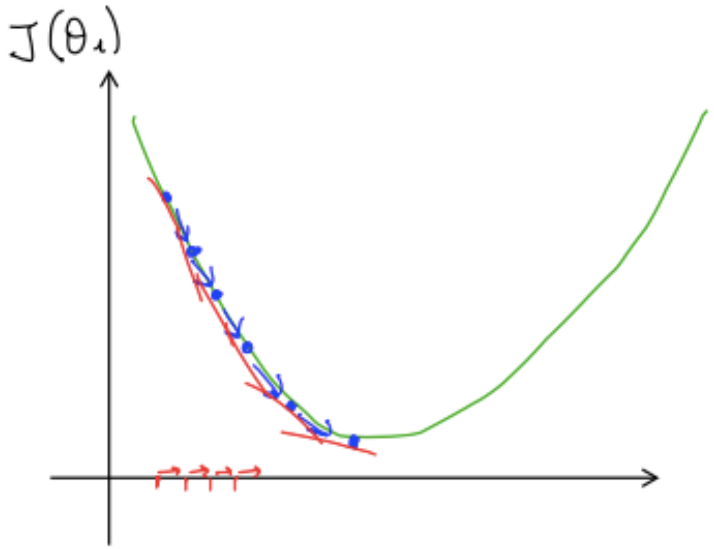
Let's look and digest all its parts.

- Firstly, let's look at the **θs**
- Secondly, let's look at the **derivative** term
- Thirdly, look at the **learning rate** term

Implementation of GD algo: **learning rate**

α too small $\Rightarrow$ GD can be slow

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



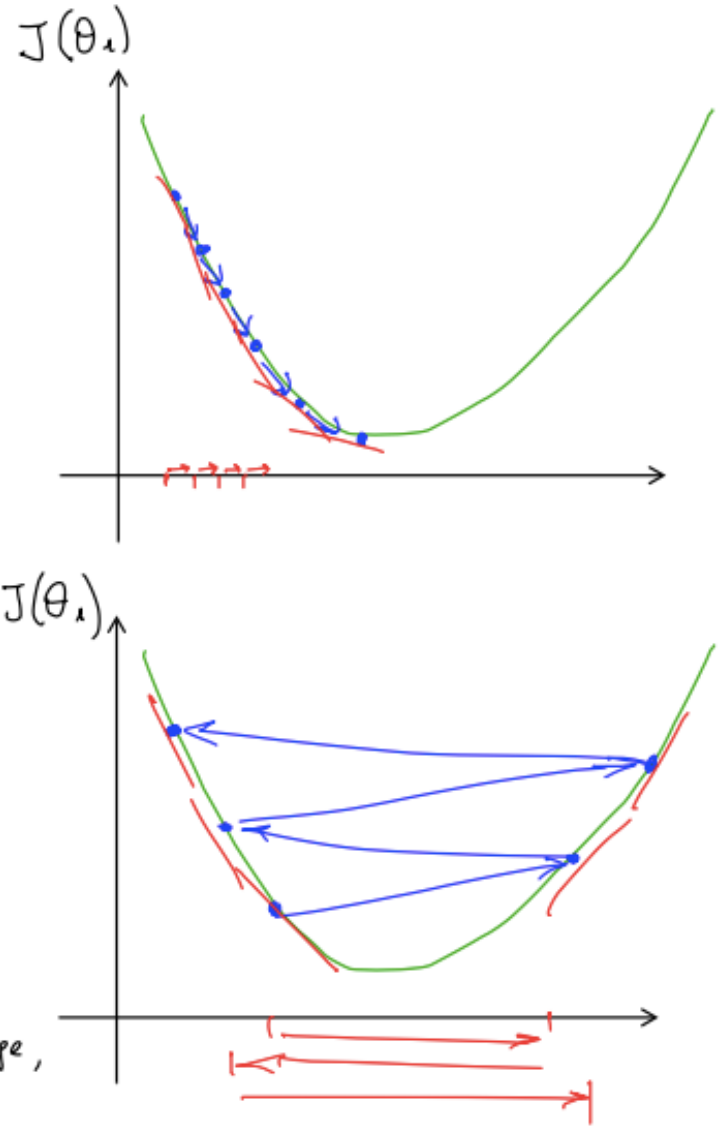
Implementation of GD algo: **learning rate**

α too small $\Rightarrow$ GD can be slow

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

α too large $\Rightarrow$ GD may overshoot the minimum

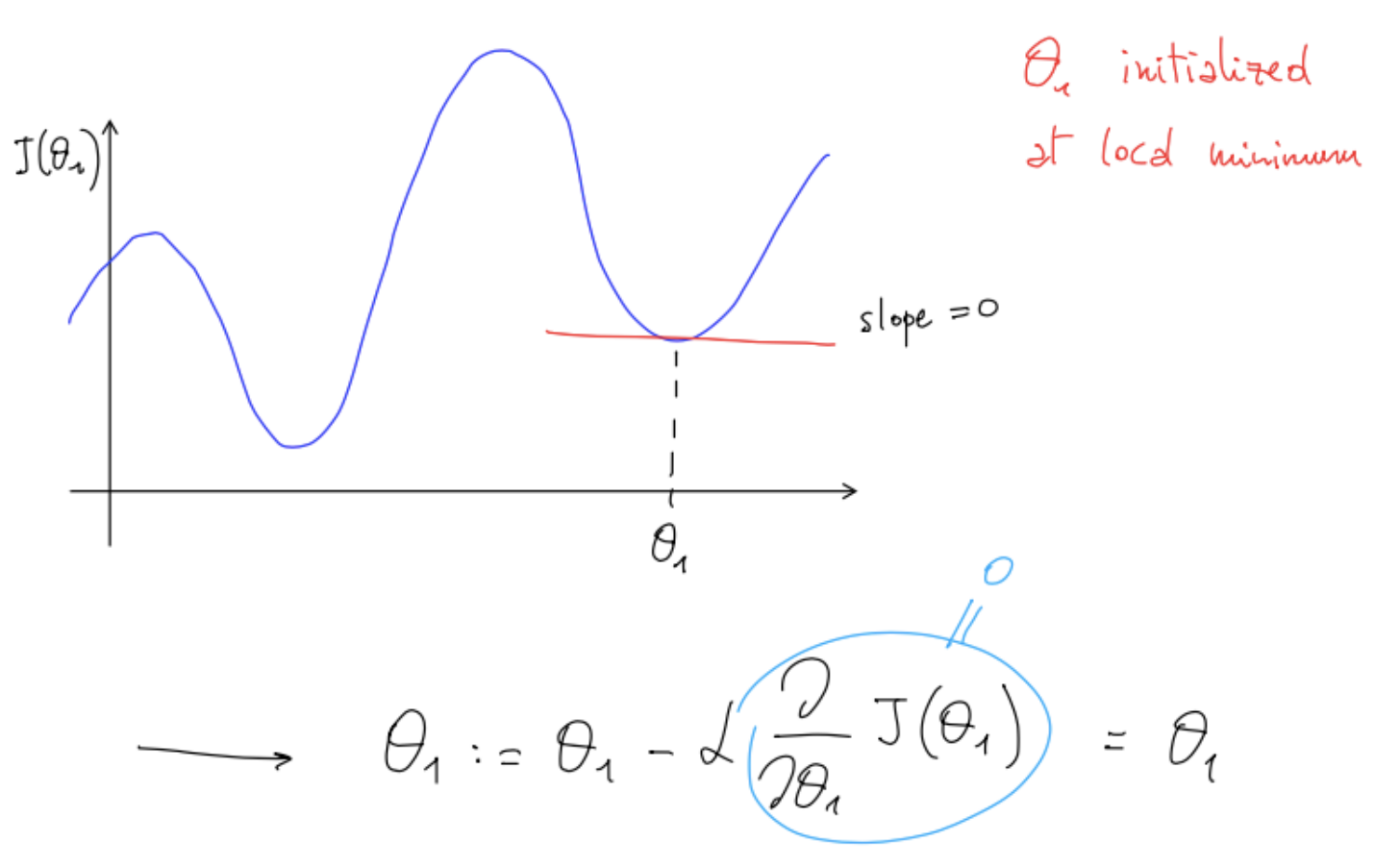
$\rightarrow$ it may fail to converge, or even diverge



Bonus feature! GD in action

Local minimum

What if you are at a **local minimum**? What will GD do?

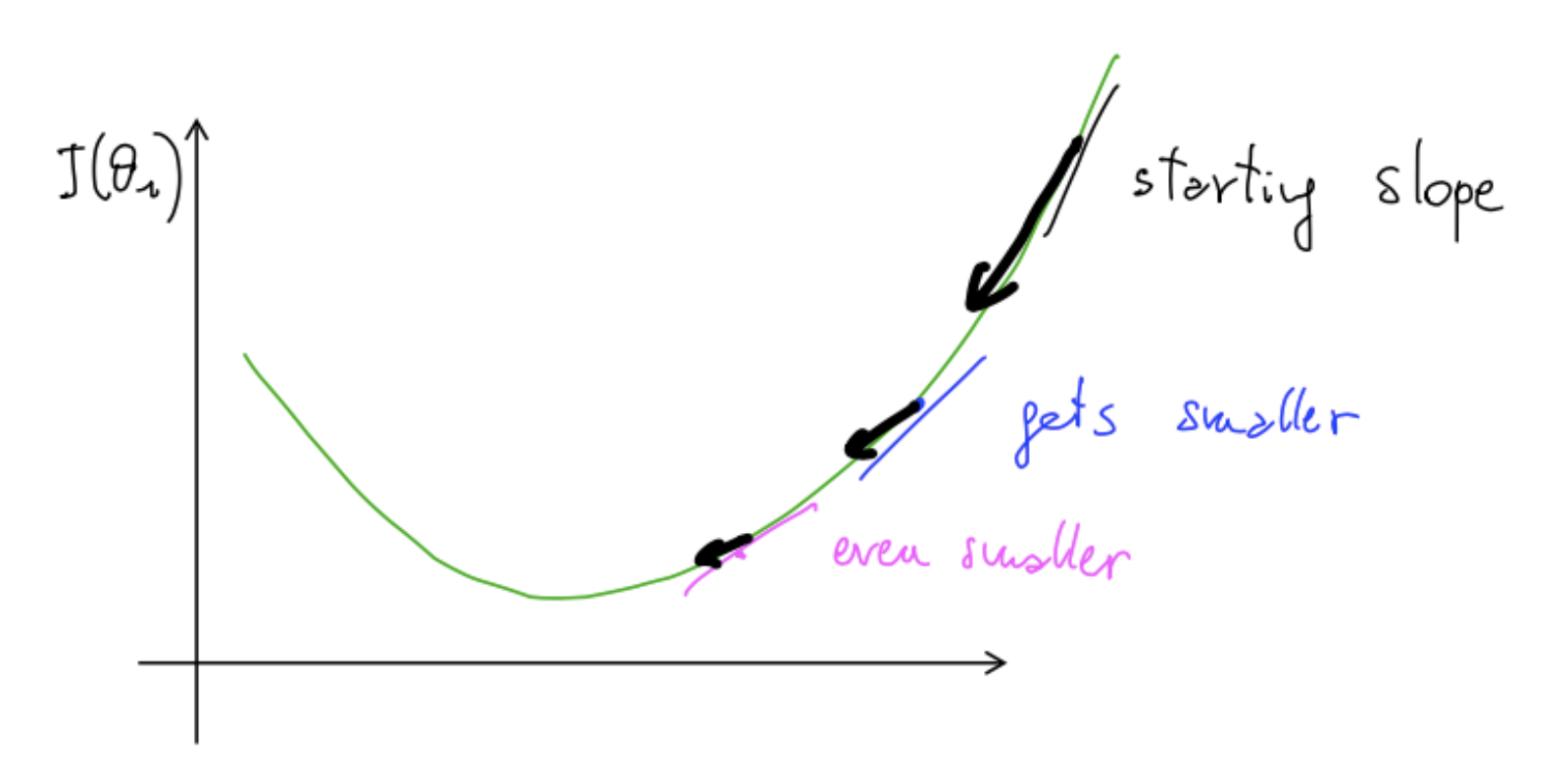


Absolutely nothing! It does not change the parameter any more. Which is precisely what you want as it keeps your solution at the local optimum. This is a property of basic GD.

Does GD converge with _fixed_ learning rate?

This also explains why GD can converge to a local minimum even with the learning rate *fixed*.

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



Summary [1/3]

Once we have made our hypothesis via the function **h**, we need to estimate the θs parameters.

For this task we introduced **GD**.

We graph the cost function **J** as a function of the θs estimates

- Note we are not graphing x and y themselves, but the **θ** parameter range of our **h** function and the cost **J** resulting from selecting a particular set of parameters.
  - E.g. 2 parameters: we put θ_0 on the x axis and θ_1 on the y axis, with the cost function J on the vertical z axis. The points on our J graph will be the result of the cost function using our hypothesis **h** with those specific θ parameters.

Summary [2/3]

GD gives you a protocol to follow:

1. take the derivative of the cost function
   - The slope of the tangent is the derivative at that point, it gives us a direction to move towards on the next step
2. does it converge? Yes. Regardless of the slope's sign for the derivative, it eventually converges to the parameters' minimum values
   - the direction in which the step is taken is determined by the partial derivative of J; when the slope is negative, the value of a parameter increases and when it is positive, the value decreases
3. at each iteration j, one should simultaneously update the parameters θ_j.
   - Careful! Updating a specific parameter prior to calculating another one on the same iteration would yield to a wrong implementation of GD!

Summary [3/3]

- 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
- We should adjust our parameter α to ensure that the GD algo converges in a reasonable time.
  - a smaller/larger α would result in a smaller/larger step, respectively
  - failure to converge or too much time to obtain the minimum value imply that our step size is wrong
- we will know that we have succeeded when our cost function J is at the very bottom of the pits in the graph, i.e. when its value is the minimum.
- How does GD converge with a fixed step size α? The derivative approaches 0 as we approach the bottom of our convex function. At the minimum, the derivative will always be 0 and thus no parameters update are done
- NOTE: depending on where one starts on the graph, one could end up at different points. This is a propriety of GD.

Applying GD
to linear regression

Apply GD to minimize J for Linear Regression

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

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

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

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

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

$$= \begin{cases} j=0 : & \frac{\partial}{\partial \theta_0} J(\theta_0, \theta_1) = \frac{1}{m} \sum_{i=1}^{m} \left(h_\theta(x^{(i)}) - y^{(i)} \right) \\ j=1 : & \frac{\partial}{\partial \theta_1} J(\theta_0, \theta_1) = \frac{1}{m} \sum_{i=1}^{m} \left(h_\theta(x^{(i)}) - y^{(i)} \right) \cdot x^{(i)} \end{cases}$$

GD algo for Linear Regression

repeat until convergence {

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

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

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

} (update θ_0, θ_1 simultaneously)

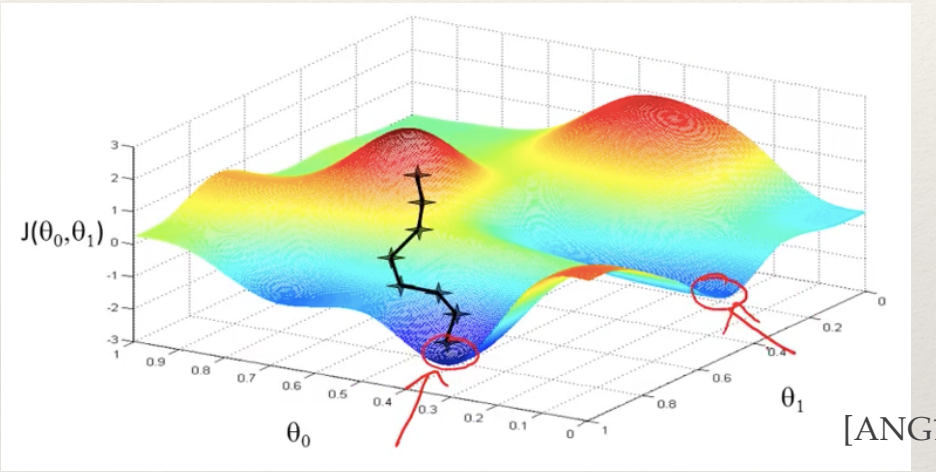
$$\frac{\partial}{\partial \theta_1} J(\theta_0, \theta_1)$$

Local vs global optima

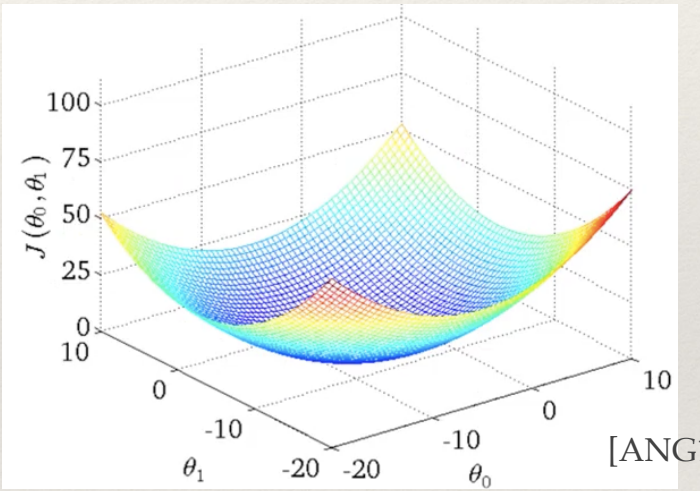
One of the properties we saw of GD is that it can be susceptible to local optima.

- depending on where you initialise it, you can end up at different local optima

It turns out that that the cost function for linear regression is always going to be a convex quadratic function (i.e. "bow shaped")



[ANG1]



[ANG1]

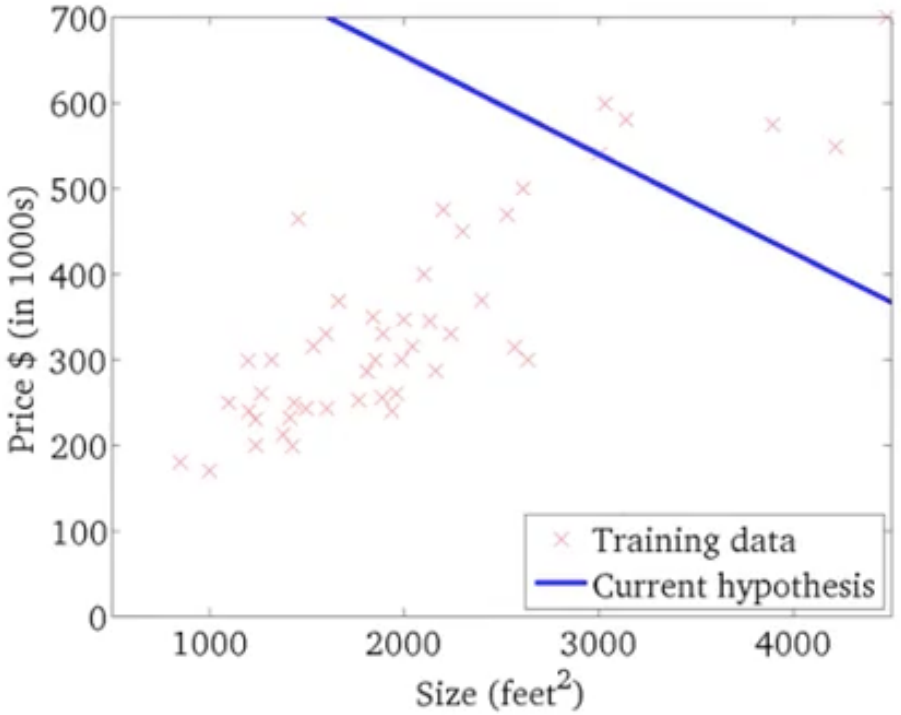
A convex function does **not** have any local optima except for the one global optimum. GD on this type of cost function (which you get whenever you are dealing with linear regression) will always converge to the global optimum simply because there are no other local optima

- NOTE: assuming the learning rate α is not too large..

GD in action!

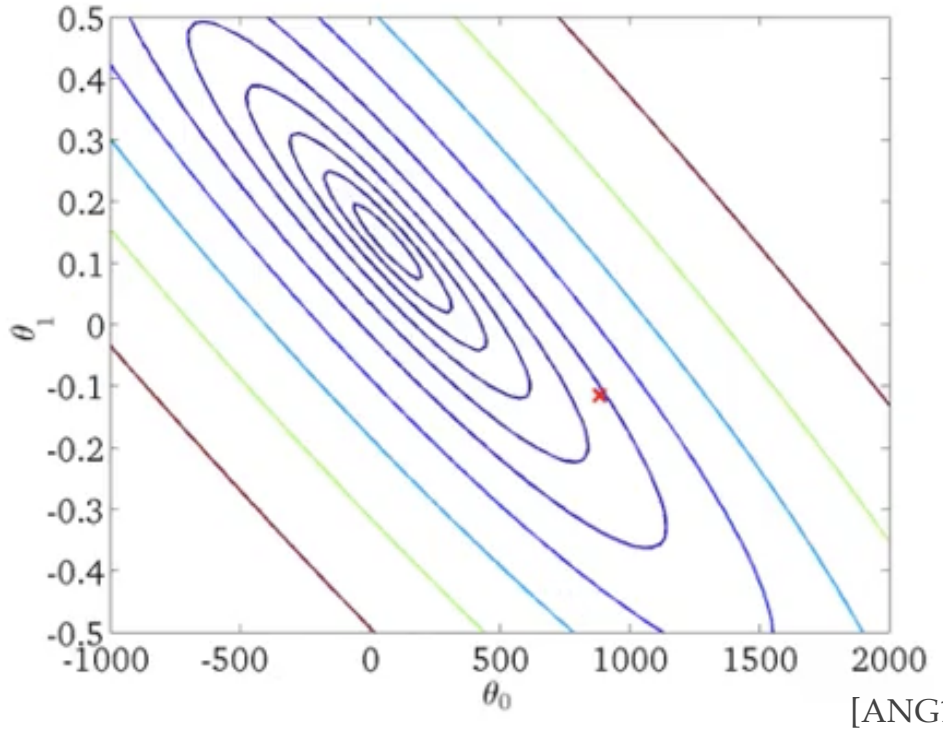
$h_\theta(x)$

(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)



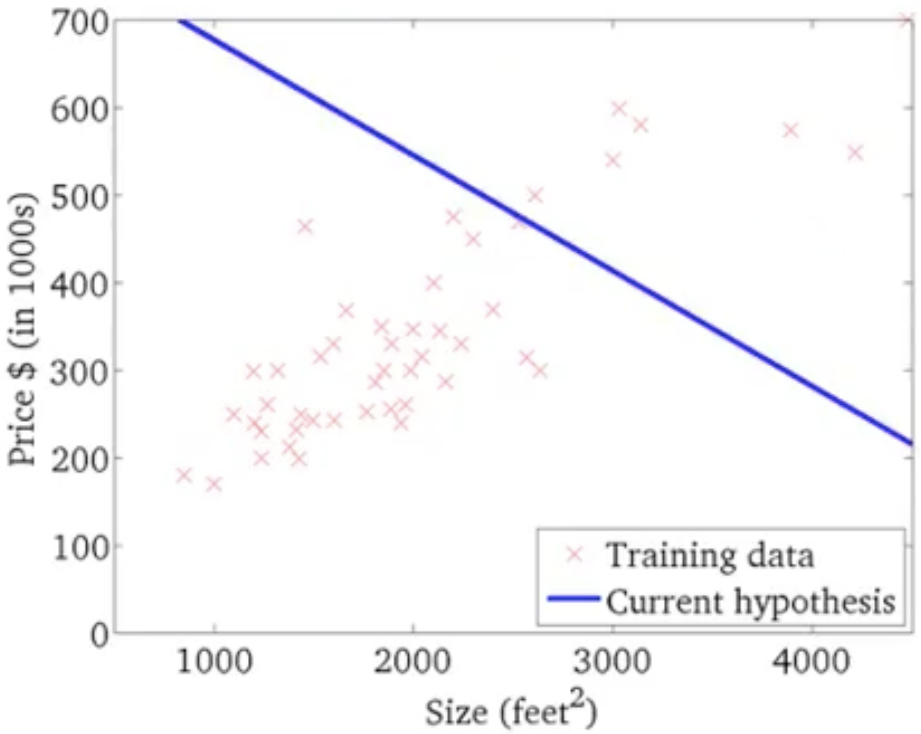
[ANG1]

Initialize θs and go!

①②③④⑤⑥⑦⑧⑨

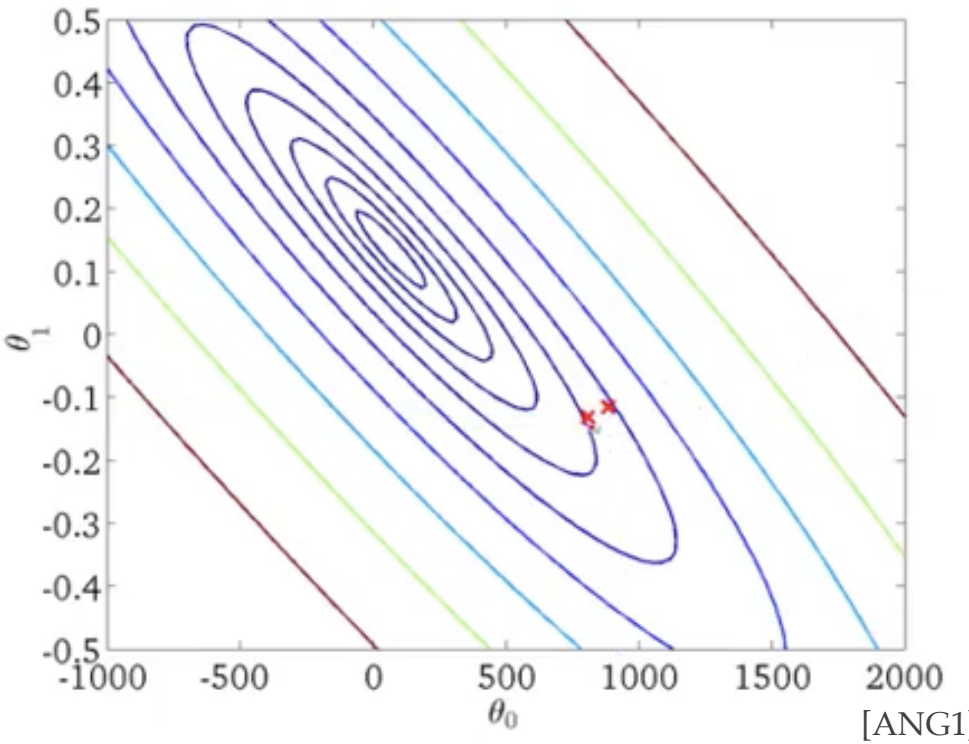
$h_\theta(x)$

(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)



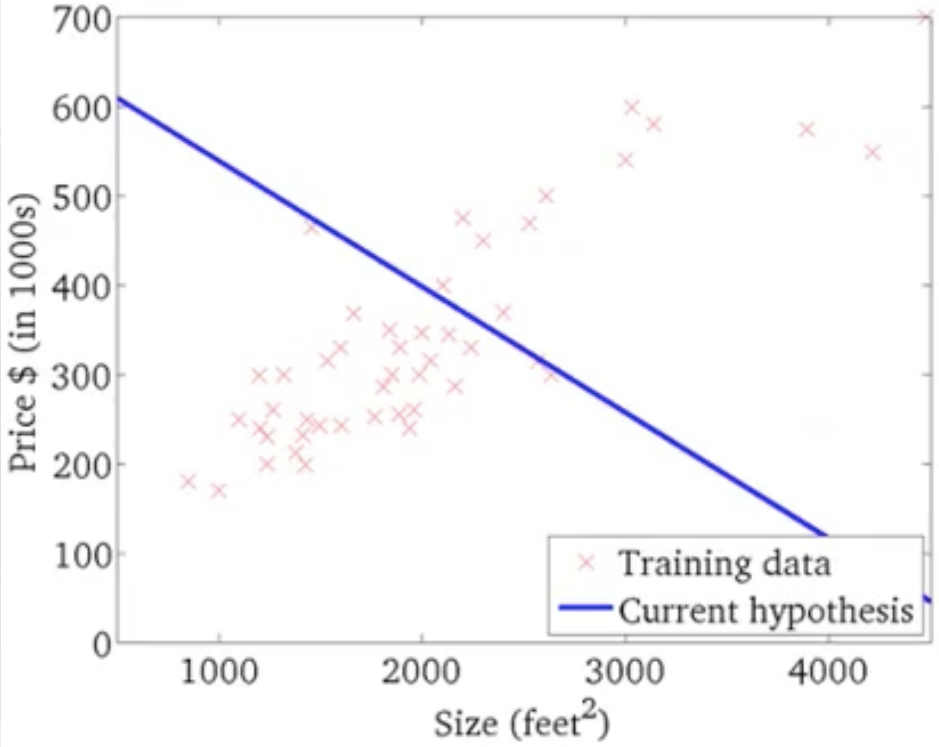
[ANG1]

Line changed a little bit.. go on..

①②③④⑤⑥⑦⑧⑨

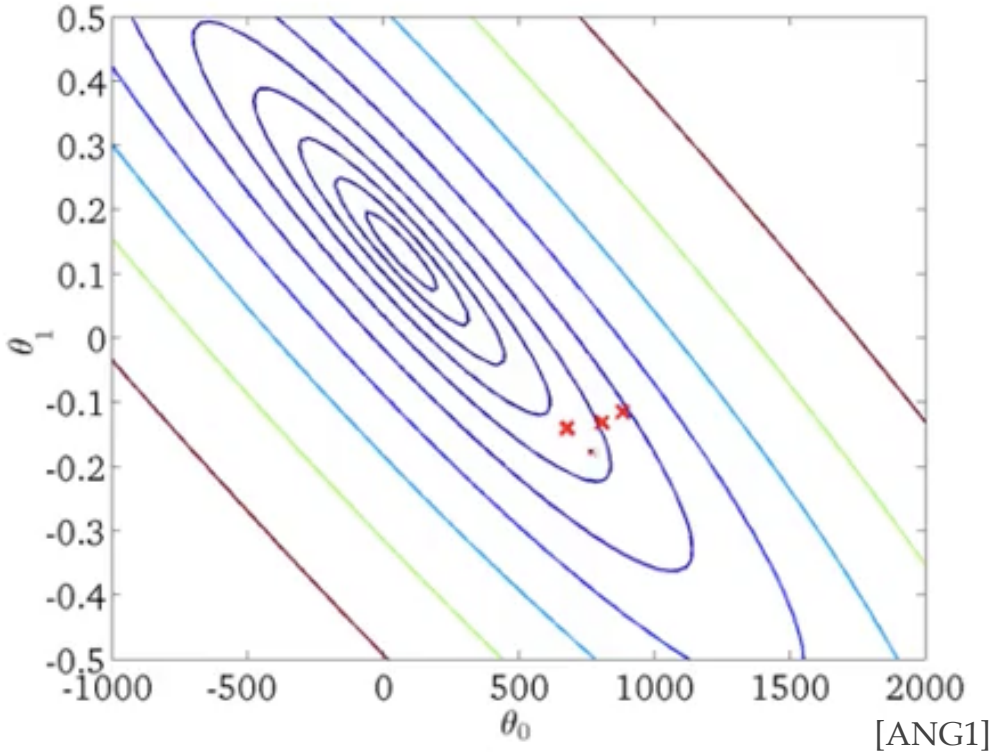
$h_\theta(x)$

(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)



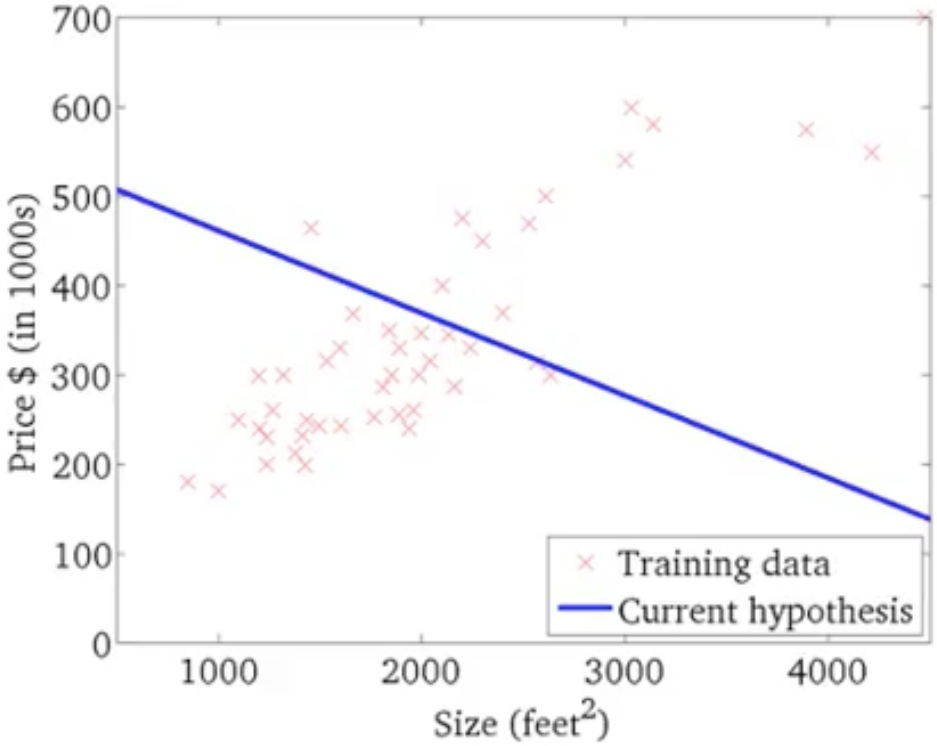
[ANG1]

More..

①②③④⑤⑥⑦⑧⑨

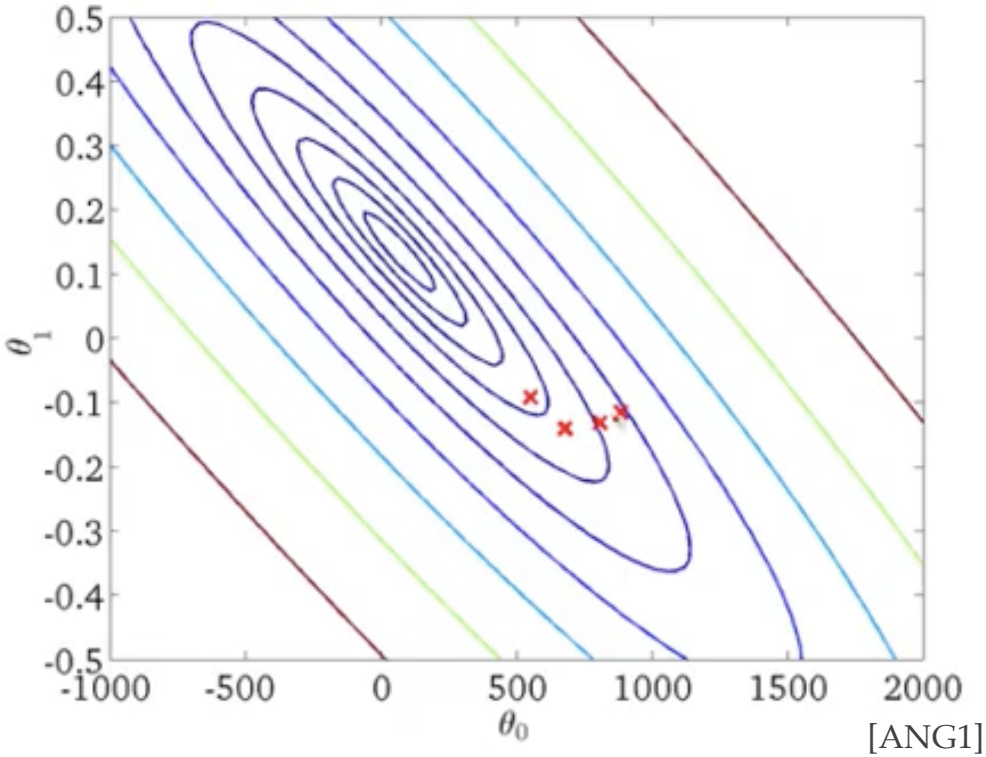
$h_\theta(x)$

(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)



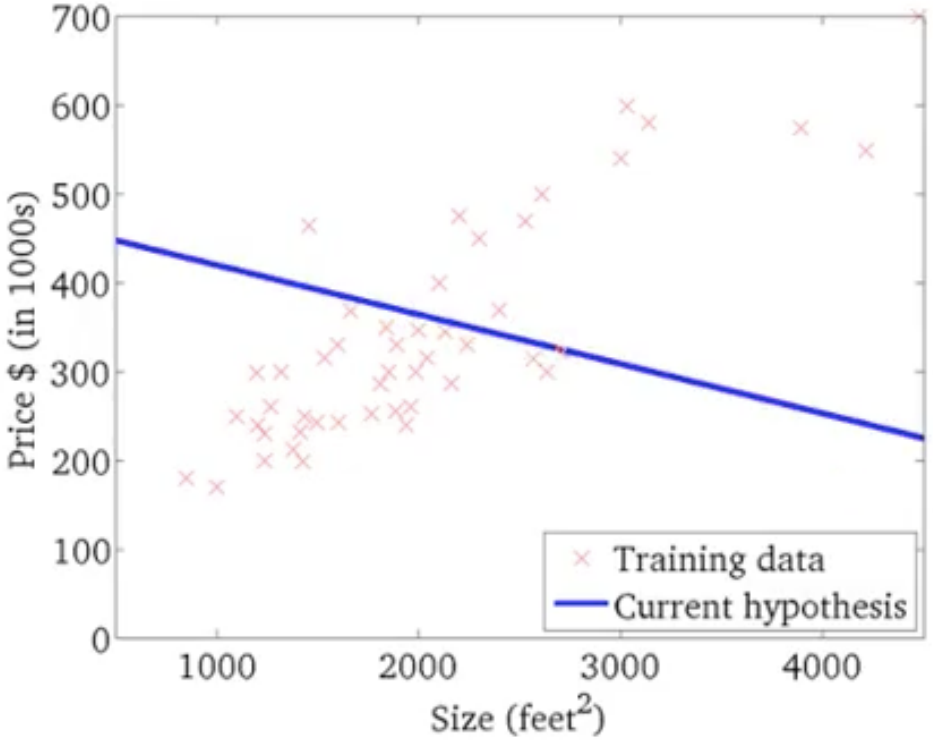
[ANG1]

More..

①②③④⑤⑥⑦⑧⑨

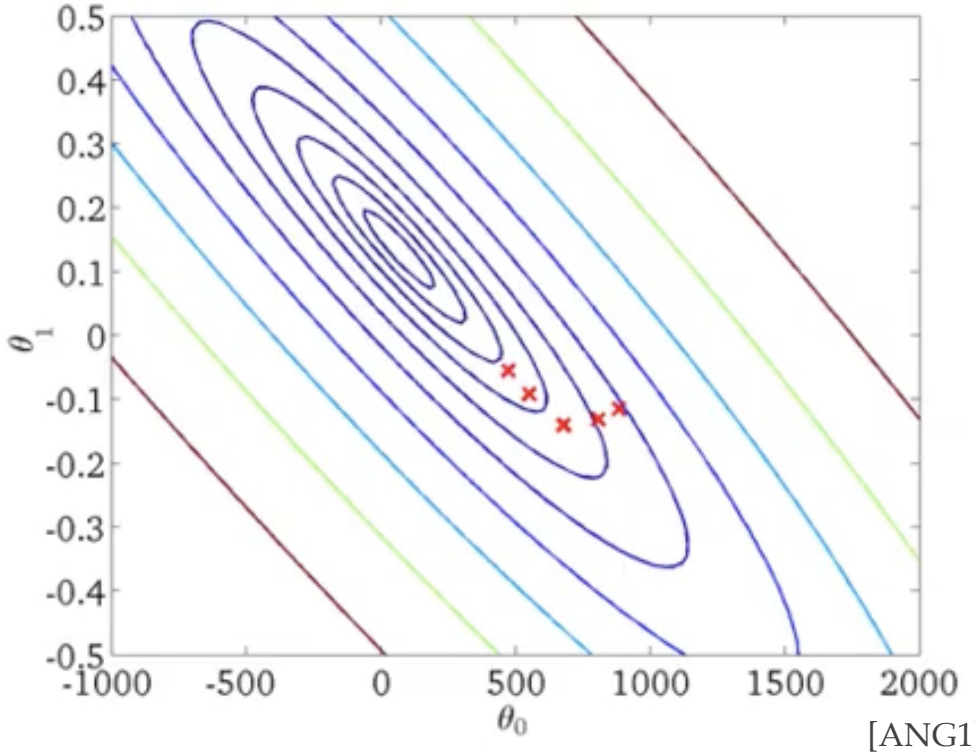
$h_\theta(x)$

(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)

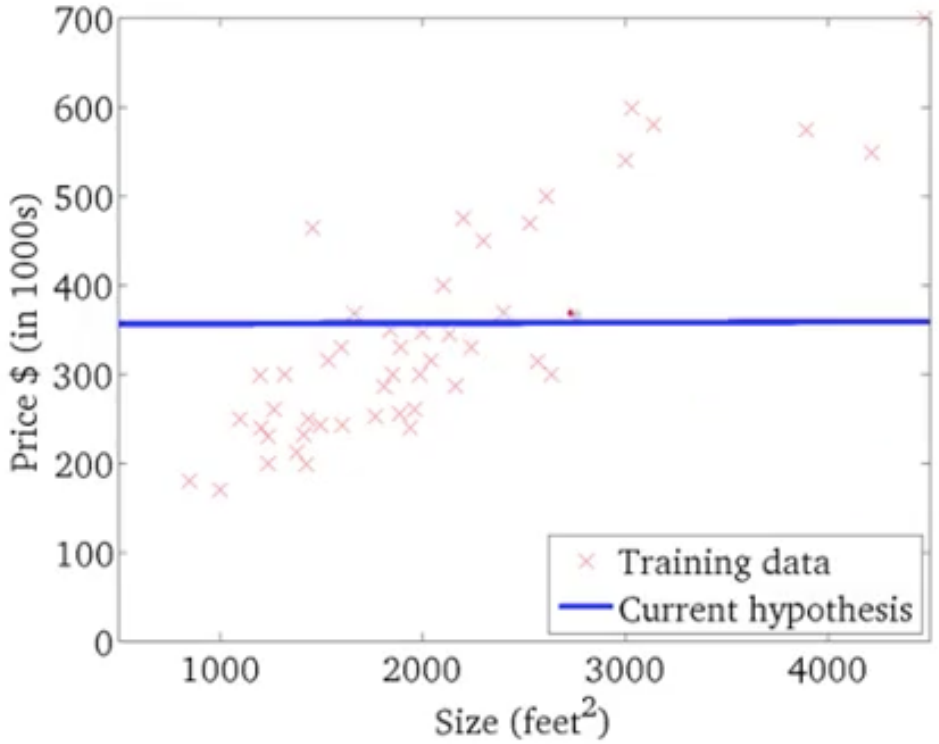


[ANG1]

More..

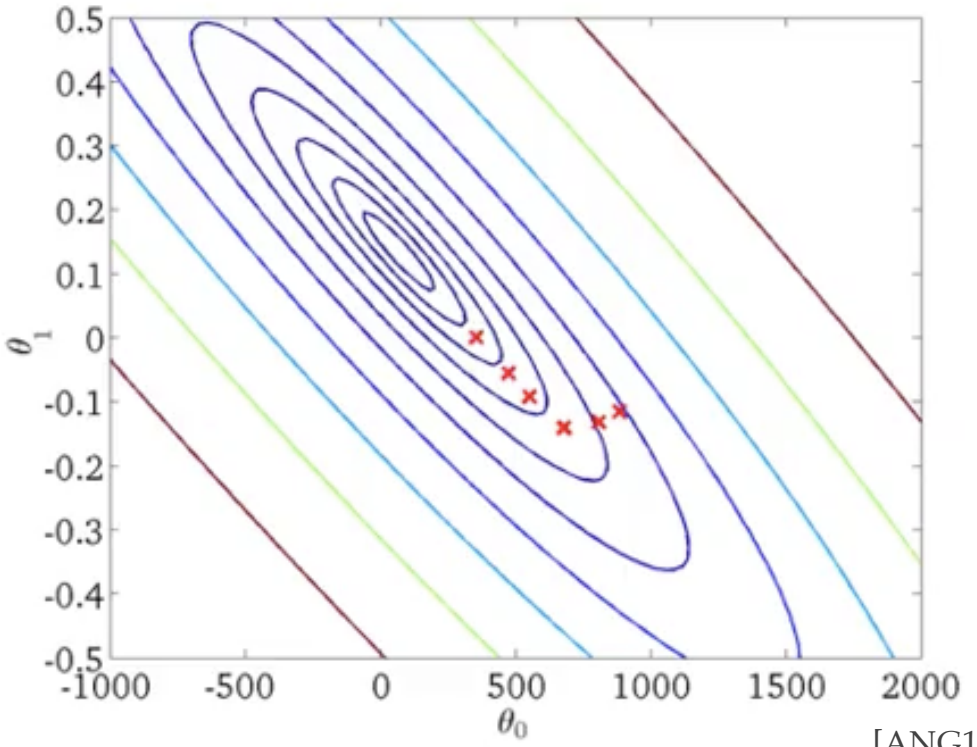
$h_\theta(x)$

(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)



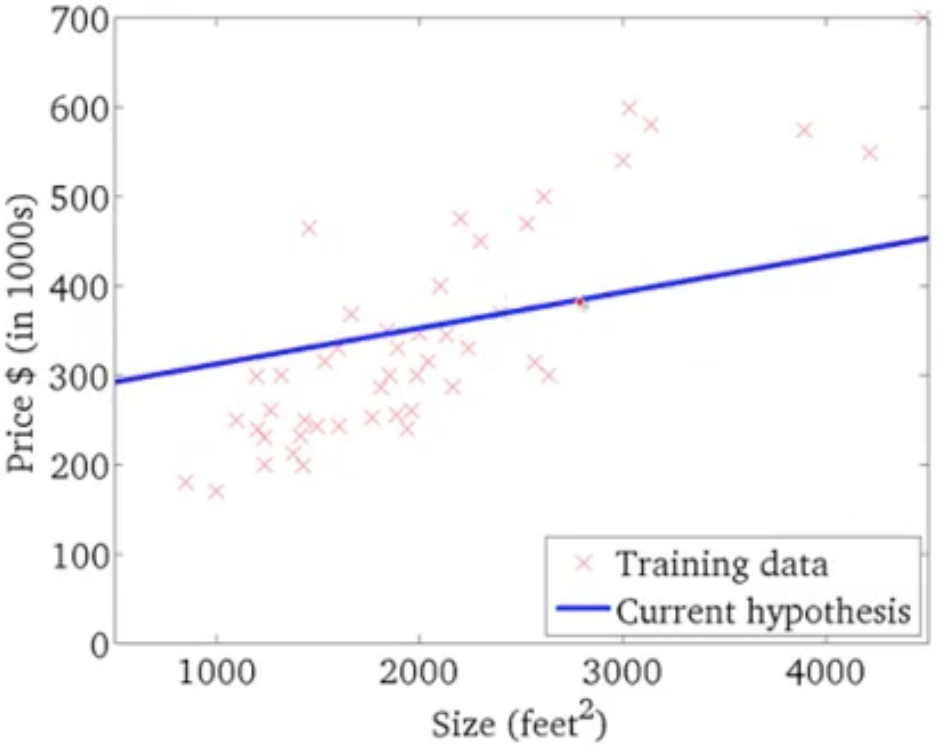
[ANG1]

$\theta_1 = 0$, so horizontal line.. More..

①②③④⑤⑥⑦⑧⑨

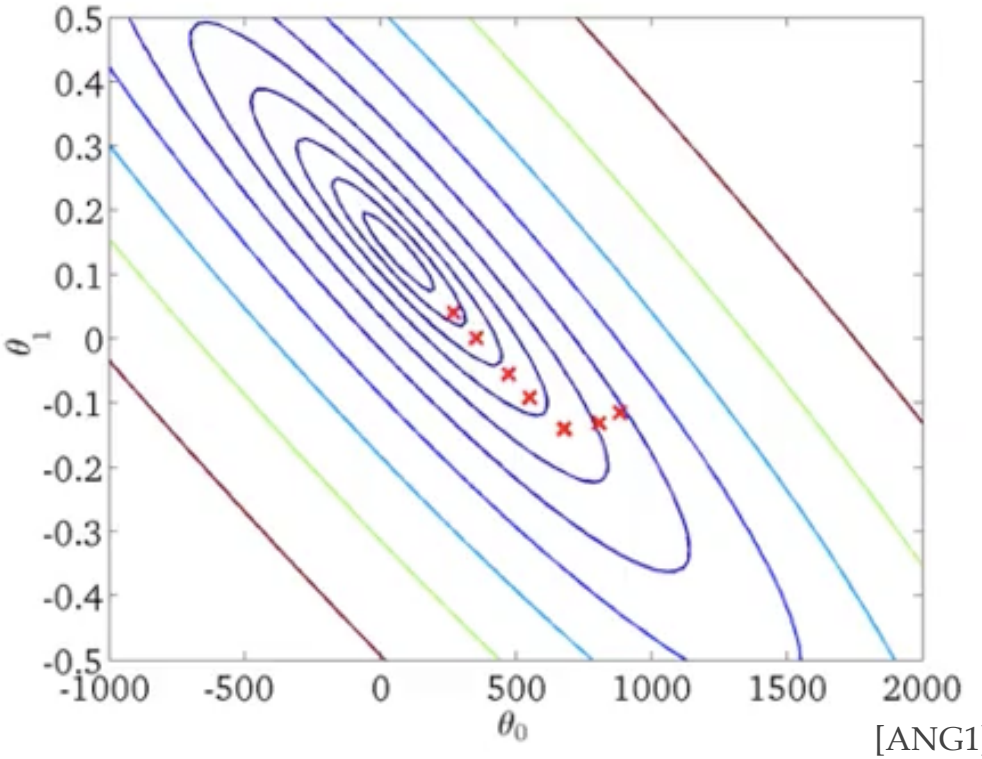
$h_\theta(x)$

(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)



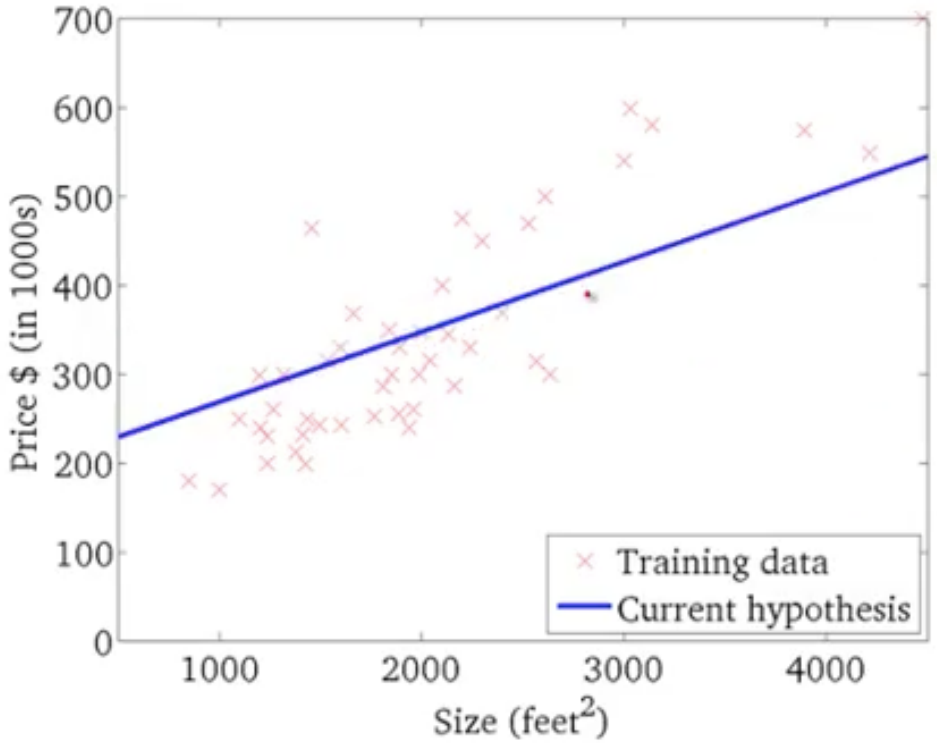
[ANG1]

More..

①②③④⑤⑥⑦⑧⑨

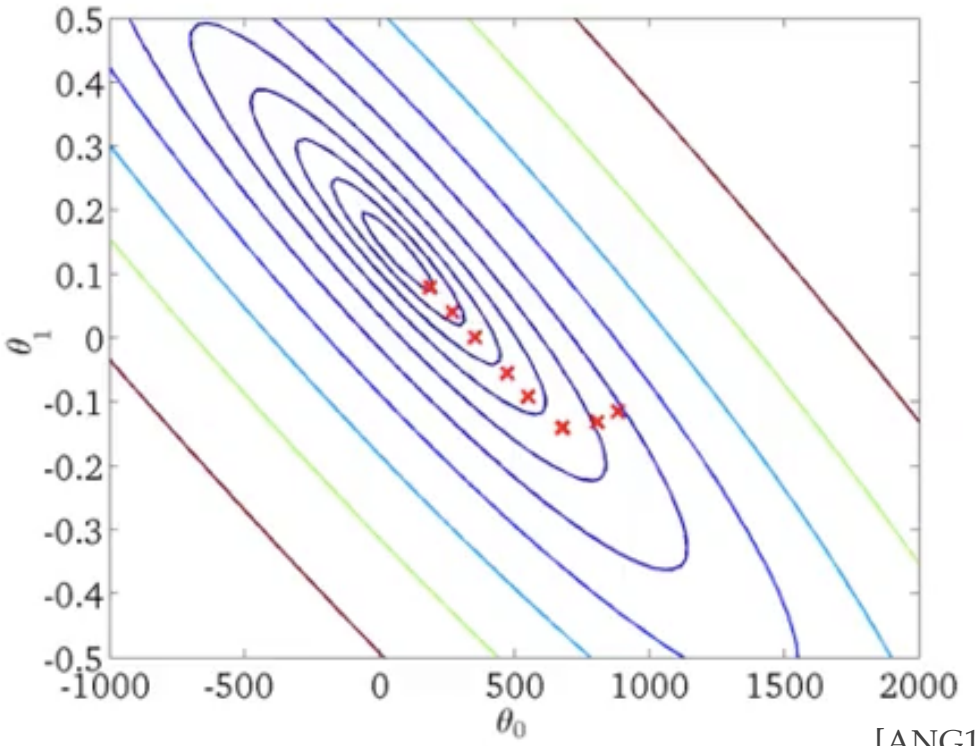
$h_\theta(x)$

(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)



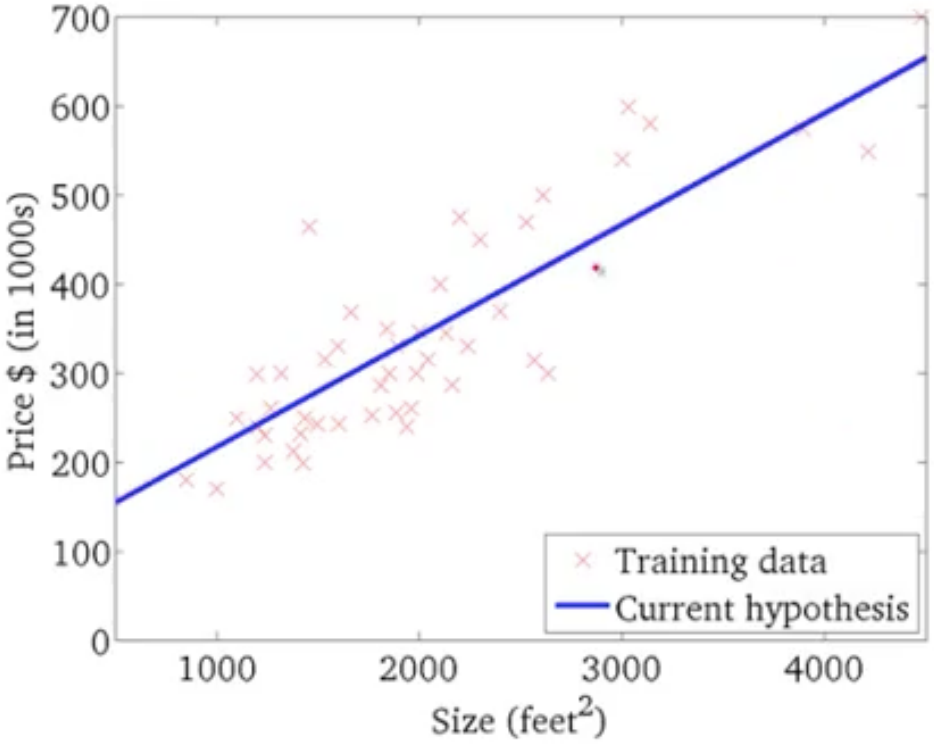
[ANG1]

OK. It starts to fit the data.. More..

①②③④⑤⑥⑦⑧⑨

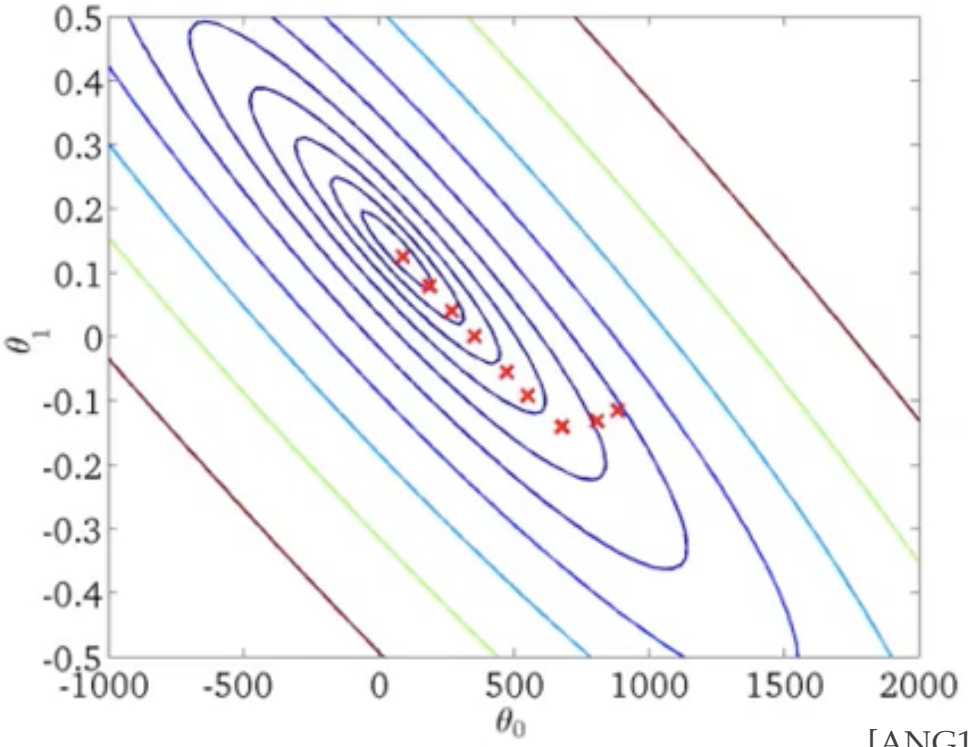
$h_\theta(x)$

(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)



[ANG1]

①②③④⑤⑥⑦⑧⑨

Eventually, it reaches the global minimum corresponds to having an hypothesis that gives me a good fit to the data

"Batch" GD

So, we know how **GD for linear regression** works: the point of all this is that if we start with a guess for our hypothesis and then repeatedly apply the GD equations for linear regression, our hypothesis will become more and more accurate.

This algo is usually called **batch GD**.

- it refers to the fact that in every step of GD, we are looking at all (the entire batch) of the training examples
  - see the formula: when computing the derivatives, we are computing the sums on i from 1 to m

There are other versions of GD, i.e. looking at small subsets of the set

- more later on..

Alternatives to GD?

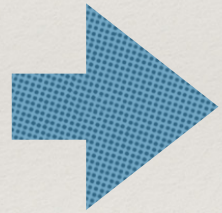
NOTE: there exists a solution for numerically solving for the minimum of the cost function J without needing to use an iterative algorithm like GD

- the normal equation method

Why don't we use it?

- (SPOILER): ıʇ ʇnɹus onʇ ʇɥɐʇ ɓp ʍill sɔɐlə qəʇʇəɹ ʇo lɐɹɓəɹ pɐʇɐ səʇs

Ready for Quiz 2




Model representation
Multivariate linear regression

Univariate vs Multivariate

Linear regression with multiple variables is also known as "**multivariate linear regression**"

- Multivariate = a fancy term for saying we have multiple features, or multi-variables with which to try to make predictions

Univariate Linear Regression

you have 1 single feature
to use to predict the price of the house

Size (feet²) x	Price ($1000) y
2104	460
1416	232
1534	315
852	178
...	...

Multivariate Linear Regression

you have ≥1 features
to use to predict the price of the house

Size (feet²)	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

$$j : 1, 2, ..., n$$

Size (feet²)	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
...	...	...	...	...

$$i : 1, 2, ..., m$$

m # examples (i.e. rows)

n # features (denoted as $x^{(i)}$, …)

$x^{(i)}$ input (features) of i^{th} training example
(**m** examples in total, **i: 1→m**)

e.g. $x^{(4)} = \begin{bmatrix} 852 \\ 2 \\ 1 \\ ... \end{bmatrix}$

$x_j^{(i)}$ value of feature j in i^{th} training example
(**n** features in total, **j: 1→n**)

e.g. $x_4^{(2)} = 40$

Form of our hypothesis

Size (feet²)	Price ($1000)
x	y
2104	460
1416	232
1534	315
852	178
...	...

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

we only had **x** as a single feature

Form of our hypothesis

Size (feet²)	Price ($1000)
x	y
2104	460
1416	232
1534	315
852	178
...	...

Size (feet²)	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
...	...	...	...	...

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

we only had **x** as a single feature

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

now we have multiple $\mathbf{x_j}$ features, even thousands!

Form of our hypothesis

Size (feet²)	Price ($1000)
x	y
2104	460
1416	232
1534	315
852	178
...	...

Size (feet²)	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
...	...	...	...	...

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

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

we only had **x** as a single feature

now we have multiple $\mathbf{x_j}$ features, even thousands!

not convenient… see next slide

Notation: add a zero feature

For convenience of notation, let's introduce: $x_0 = 1$

Which means that for each of the **m** examples, I have: $x_0^{(i)} = 1$

I am basically adding a "**zero feature**" that always takes value 1 in all examples.

So, for each example (i.e. not drawing **(i)** below), my feature vector becomes n+1 dimensional:

$$\mathbf{x} = \begin{bmatrix} x_0 \\ x_1 \\ \dots \\ x_n \end{bmatrix} \in \mathbb{R}^{n+1} \qquad \boldsymbol{\theta} = \begin{bmatrix} \theta_0 \\ \theta_1 \\ \dots \\ \theta_n \end{bmatrix} \in \mathbb{R}^{n+1}$$

feature vector

$$\boldsymbol{\theta}^T = [\theta_0 \ \theta_1 \ \dots \ \theta_n]$$

transpose of the parameters vector

Then I can easily think of making my notation more compact.. (see next)

Notation: vectorise!

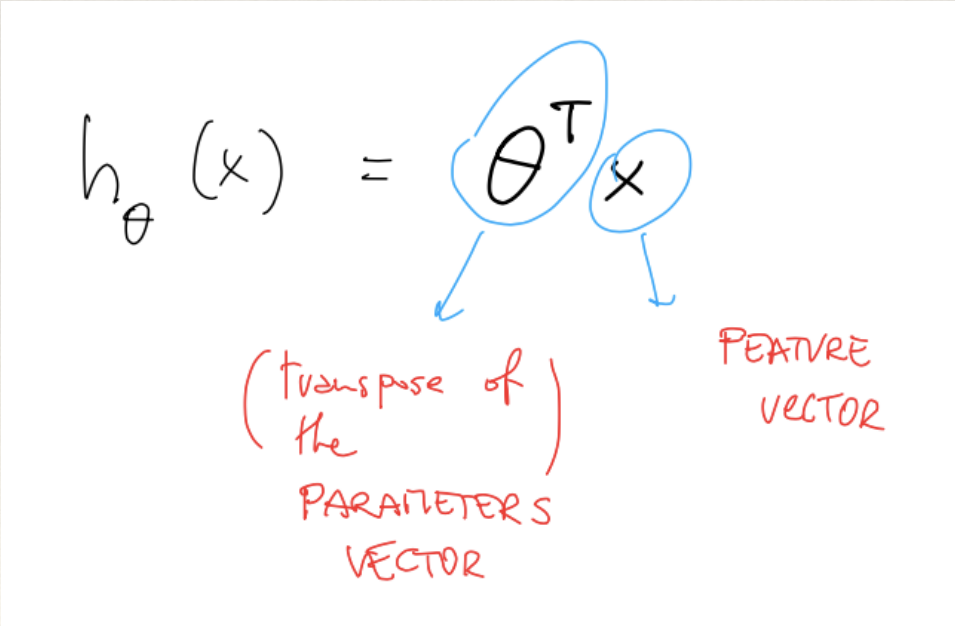
$$h_\theta(x) = \theta_0 x_0 + \theta_1 x_1 + \ldots + \theta_n x_n = \boldsymbol{\theta}^T \mathbf{x}$$

Notation: vectorised, and more compact now

$$h_\theta(x) = \theta_0 x_0 + \theta_1 x_1 + \ldots + \theta_n x_n = \boldsymbol{\theta}^T \mathbf{x}$$

This is a vectorisation of our hypothesis function for one training example

- on purpose, we have omitted all indices **(i)**




Model representation
Multivariate linear regression
Gradient Descent

GD for multivariate linear regression

We discussed univariate linear regression, and GD for it.

We introduced multivariate linear regression.

We saw the form of the **h** for multivariate linear regression.

Now, let's see how to fit the parameters of the latter hypothesis. In particular let's see how to use GD for linear regression with multiple features.

Notation for the multivariate case

hypothesis

$$h_\theta(x) = \theta_0 x_0 + \theta_1 x_1 + ... + \theta_n x_n = \boldsymbol{\theta}^T \mathbf{x}$$

parameters
(n+1 values)

$$\theta_0, \theta_1, ..., \theta_n$$

cost function

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

Notation for the multivariate case

hypothesis

$$h_\theta(x) = \theta_0 x_0 + \theta_1 x_1 + \ldots + \theta_n x_n = \boldsymbol{\theta}^T \mathbf{x}$$

parameters
(n+1 values)

$$\theta_0, \theta_1, \ldots, \theta_n$$

cost function

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

Think of "**vectors**" as much as possible…

parameters
(n+1 dimensional **vector**)

$$\boldsymbol{\theta}$$

cost function

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

Notation for the multivariate case

gradient descent

Repeat $\{$

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

$\}$ (simultaneously update for every $j = 0, ..., n$)

Notation for the multivariate case

gradient descent

Repeat {

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

} (simultaneously update for every $j = 0, ..., n$)

Think of "**vectors**" as much as possible…

gradient descent

Repeat {

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

} (simultaneously update for every $j = 0, ..., n$)

Let's see how it looks like when you do this:

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

Notation for the multivariate case

$n = 1$:

Repeat {

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

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

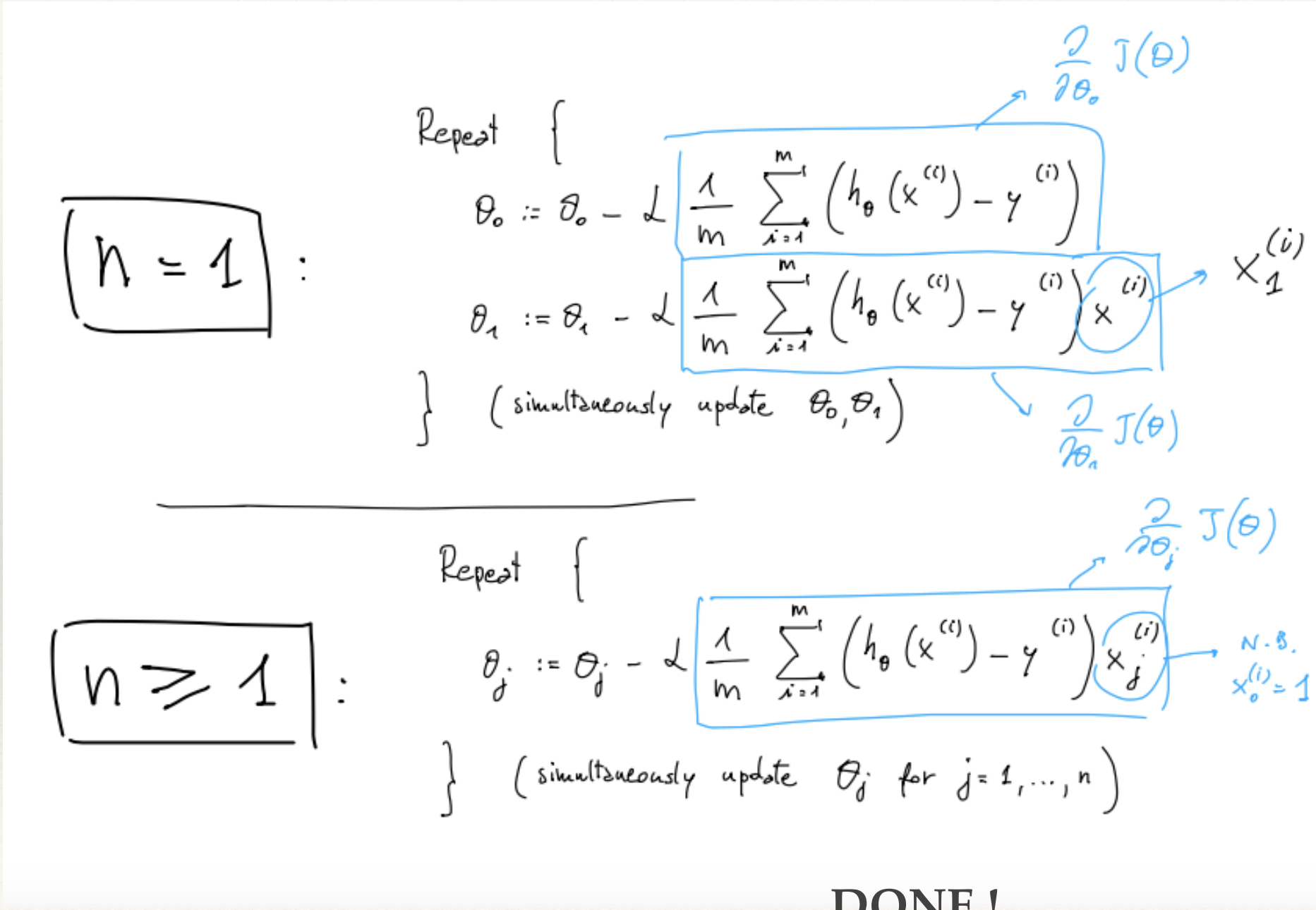
} (simultaneously update θ_0, θ_1)

$\frac{\partial}{\partial \theta_0} J(\theta)$

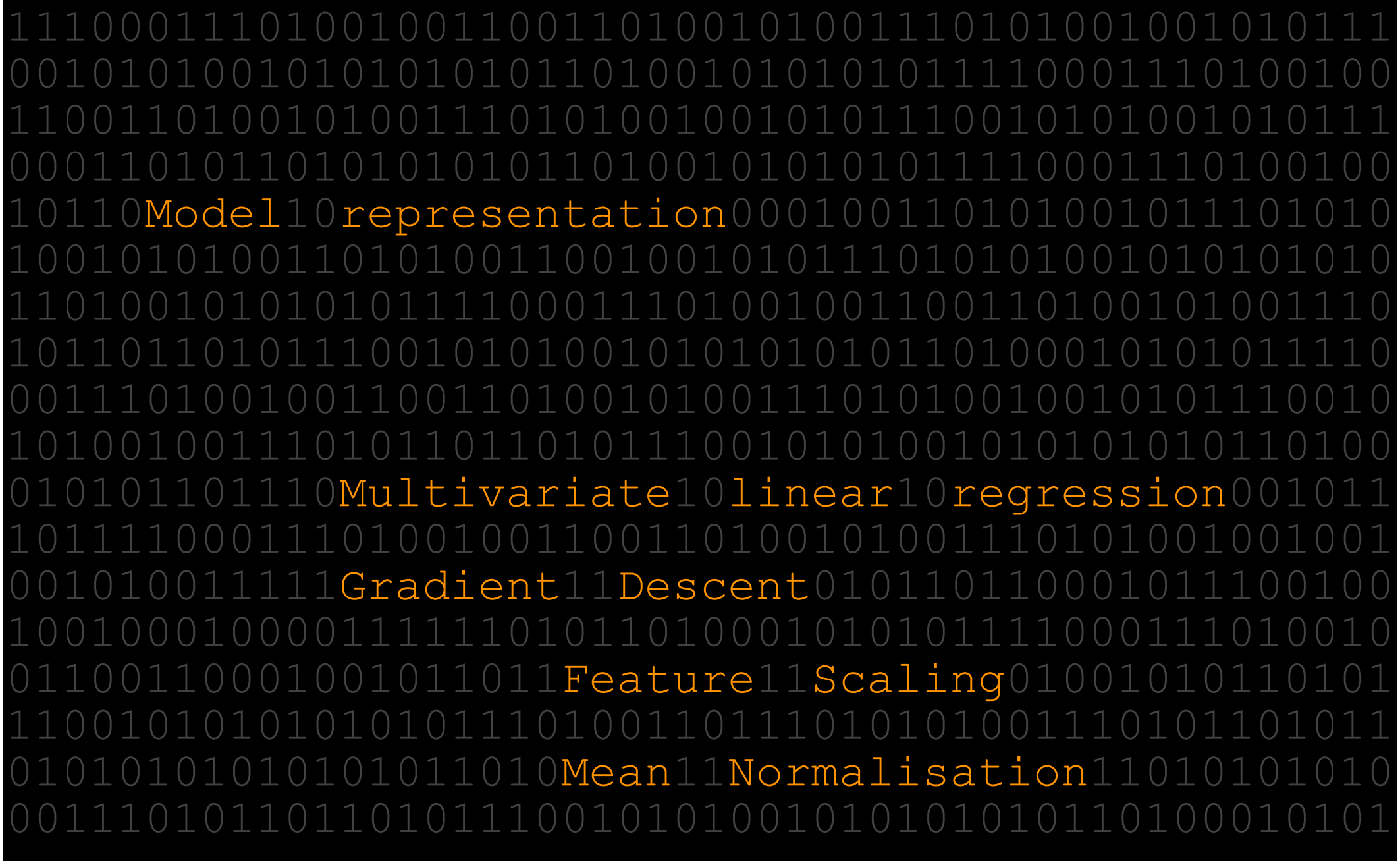
$\frac{\partial}{\partial \theta_1} J(\theta)$

$x_1^{(i)}$

Notation for the multivariate case



DONE !


Model representation
Multivariate linear regression
Gradient Descent
Feature Scaling
Mean Normalisation

A couple of "tricks"

The are some practical tricks for making GD work well.

One idea is called **feature** scaling

Another one is **mean normalisation**

- may become more important when indeed you have n>1 features..

*COMMENT: this trick (and more) would be part of what's called **data pre-processing** (or **data preparation**) in AML machinery. We will get back on it later in this respect - but it is introduced here to help you develop an "intuition" of its need, which is connected to GD*

Feature scaling

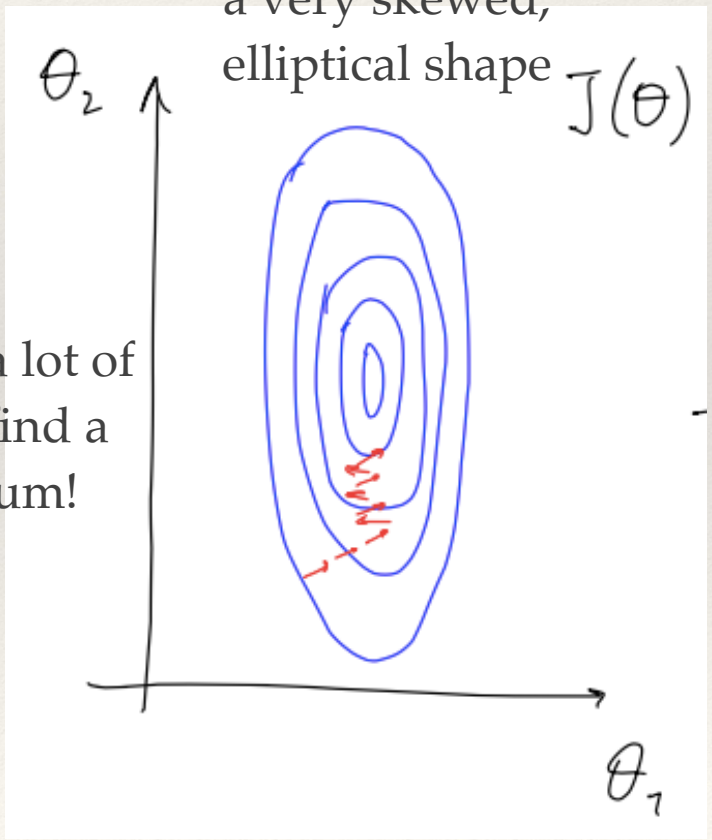
Very simple idea: **make sure features are on a similar scale**.

- i.e different features should take on similar ranges of values
  - (not entirely intuitive why this is needed, this is why we discuss it here!)

Why?

- very pragmatic reason: because then GD can converge more quickly!

cost function with a very skewed, elliptical shape



can take a lot of time to find a minimum!

Example: easily orders of magnitude!

- x_1 = house size (50-2000 mq)
- x_2 = # bathrooms (1-5)

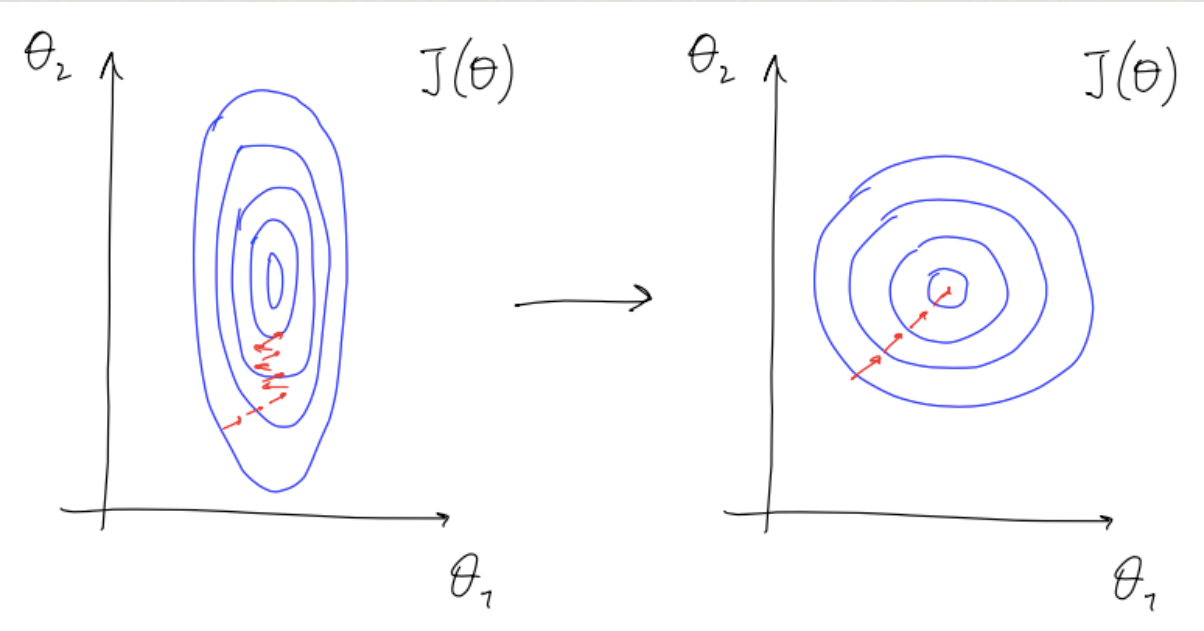
Feature scaling (cont'd)

This example indeed shows conditions in which a useful thing to do is to scale the features.

Concretely if you instead define:

- x_1 = house size (50-2000 mq) / 2000
- x_2 = # bathrooms (1-5) / 5

then **the contours are less like ellipses and more like circumferences**, and running GD on a cost function like the latter (it could be showed mathematically that) you can find a much more direct path to the global minimum



Feature scaling: which scaling ranges?

In previous example, we ended up with both features between 0 and 1.

More generally, in feature scaling, we often want to get every feature into **"approximately" a [-1,+1]** range

- remember your feature x_0 is always equal to 1, so that's already in that range

Be encouraged to adopt some **AML practical tips** as of when doing feature scaling:

- features e.g. in [0,3] or [-3,+3] or [-2,0.5]? **maybe not**. You are still close enough and no scaling is probably needed
- features in e.g. in [-100,+100]? **yes**. You might benefit from rescaling
- features in e.g. in [-0.0001,+0.0001]? **yes**. This is within [-1,1] but their relative difference might still be large, so you may benefit from rescaling

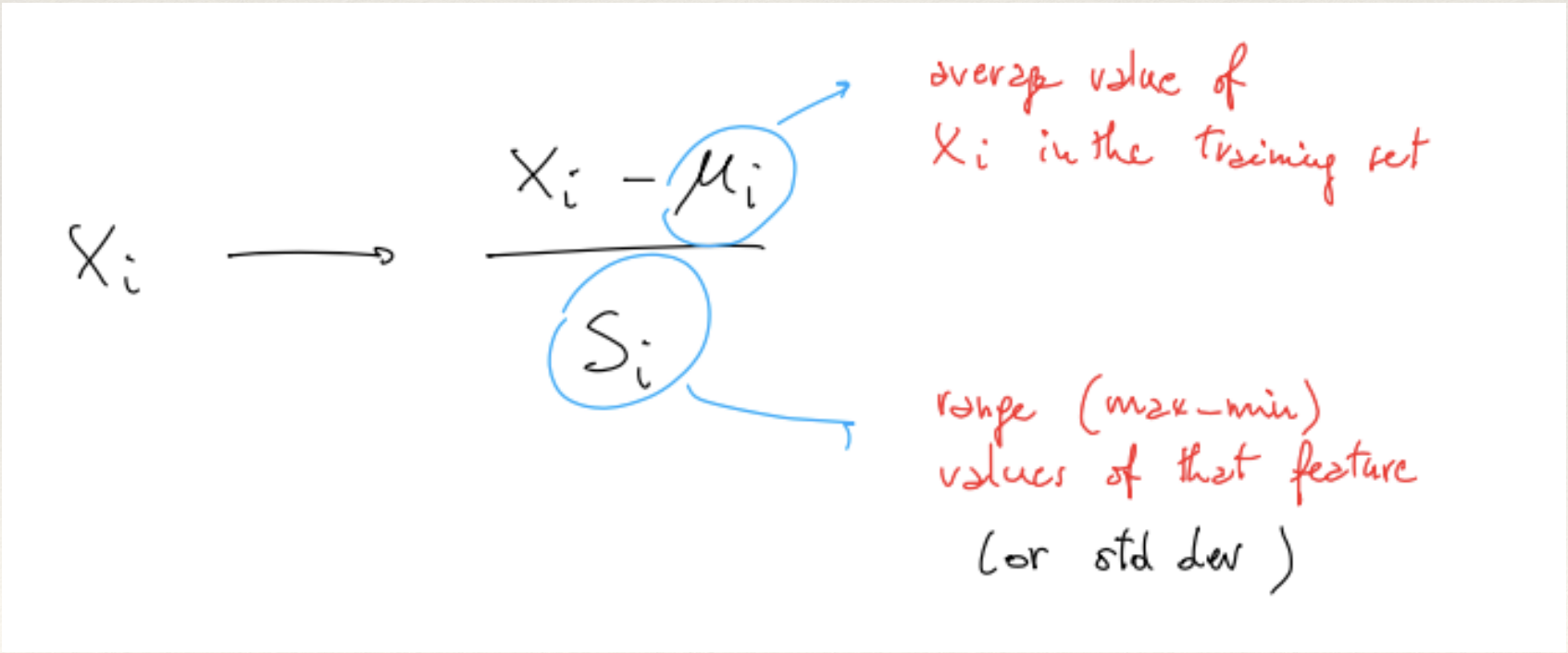
It is not the absolute value that matters, but more the proximity to a [-1,+1] range and - most important - the similarity of possible ranges of values across different features

Mean normalisation

As another often suggested action, in addition to dividing by the maximum value when performing feature scaling, ML practitioners also often do what's called **mean normalisation**.

Pretty easy:

- replace x_i (not for i=0) by $(x_i-\mu_i)/S_i$ to make features have approximately 0 mean



Summary

Feature scaling and **mean normalisation** are simple tricks that help to make GD run much faster and converge in a lot fewer iterations.

- This is something, but there is more than this to make GD work well in practice.. we will see more and focus on some hands-on on data preparation.


Model representation
Multivariate Linear regression
Gradient Descent
Learning Rate

Learning rate α

The learning rate is a **hyper-parameter** of your model

$$\text{Repeat} \quad \{$$
$$\theta_j := \theta_j - \alpha \frac{\partial}{\partial \theta_j} J(\boldsymbol{\theta})$$
$$\} \quad (\text{simultaneously update for every } \mathrm{j} = 0, \ldots, \mathrm{n})$$

Here is the GD update rule. You implement this in code and you use this iterative algorithm.

In order to debug that GD is working correctly, one thing to pay attention to is: how do I choose the learning rate α?

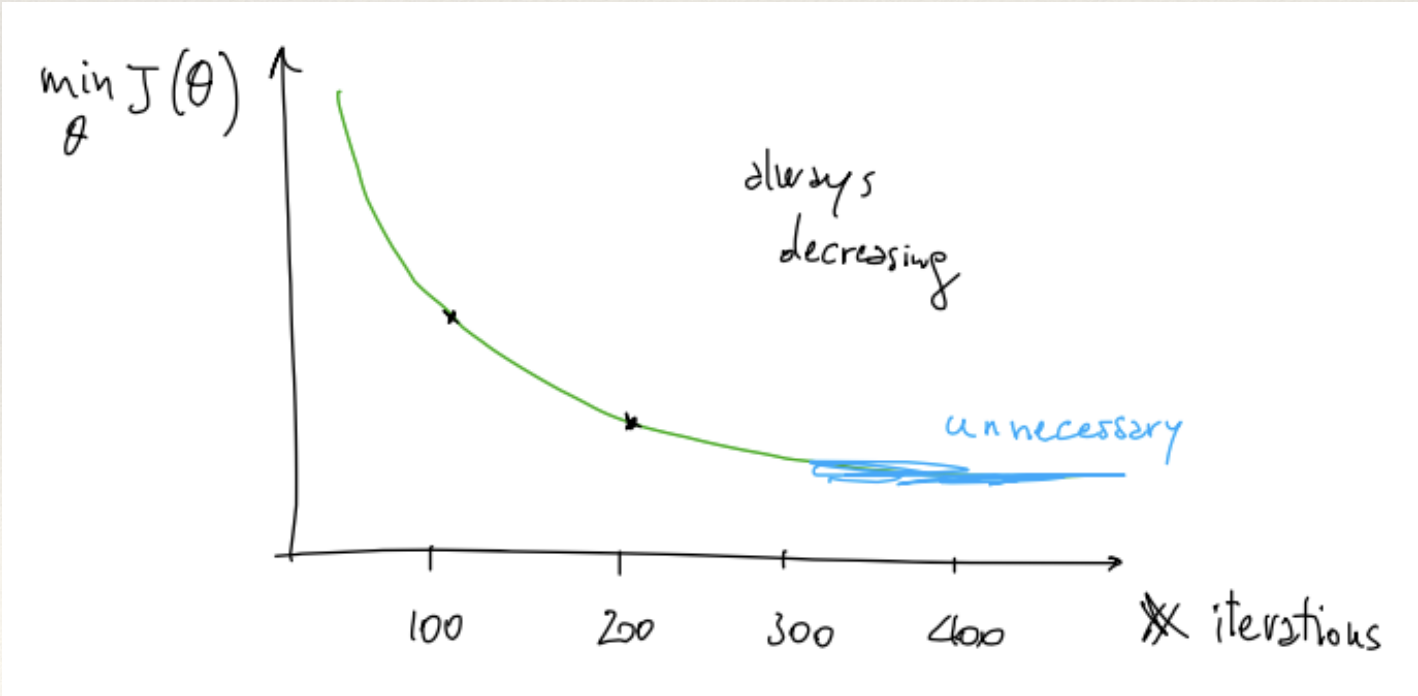
J(θ) versus # iterations

The job of GD is to find the value of θs that minimises the cost function J(θ).

To make sure that GD is working correctly, you need to visualise it: **just plot the cost function J(θ) as GD runs**.

You hopefully get a plot that shows a decreasing curve, as J should decrease after every iteration [*]

- [*] (not discussed here, but - under some assumptions about the cost function J, that are anyway true for linear regression, (math tells us that) if your learning rate α is small enough, then J(θ) should indeed decrease on every iteration



Note that the # iterations needed to stabilise may be very different from one application to another, and may even not be so smooth!

J(θ) versus # iterations: automatically?

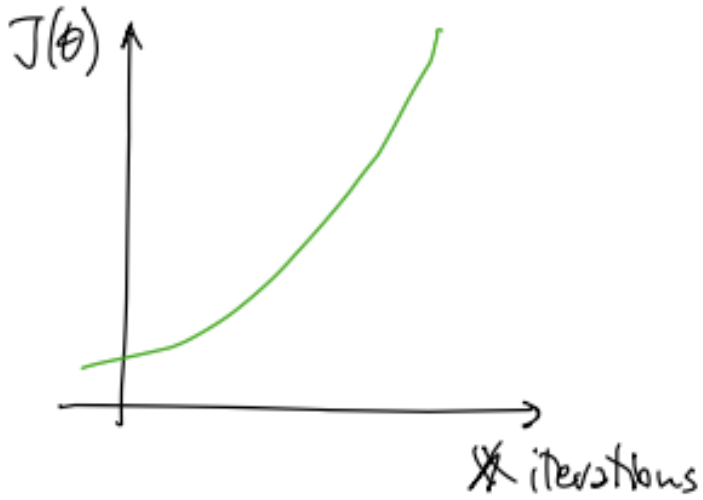
It's also possible to come up with automatic convergence test

- namely have a algorithm tell you if GD has converged
- e.g. such a test may declare convergence if your cost function J(θ) decreases by less than some small value, e.g. 10^{-3}, in 1 iteration
- usually difficult to choose such threshold, though - so perhaps checking manually the plot is better than blindly relying on automatic converge tests

As it often happens: a human eye here might just work better.

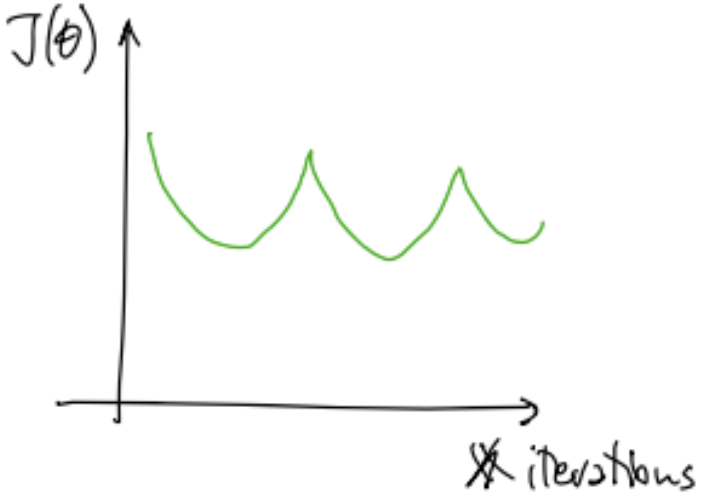
- how? See next slide.

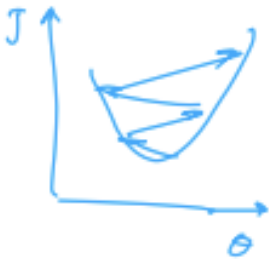
Human eye on α and GD convergence



=> GD not working

(usually means you should use smaller α)





Choosing alpha

So: we said that if your learning rate α is small enough, then $J(\theta)$ should decrease on every iteration

- if this doesn't happen, it probably means that alpha is **too large**

Cool, so just set it smaller!

- well, yes, but of course, you also don't want your learning rate to be **too small** because if you do so then GD can be really **slow to converge**

So what?

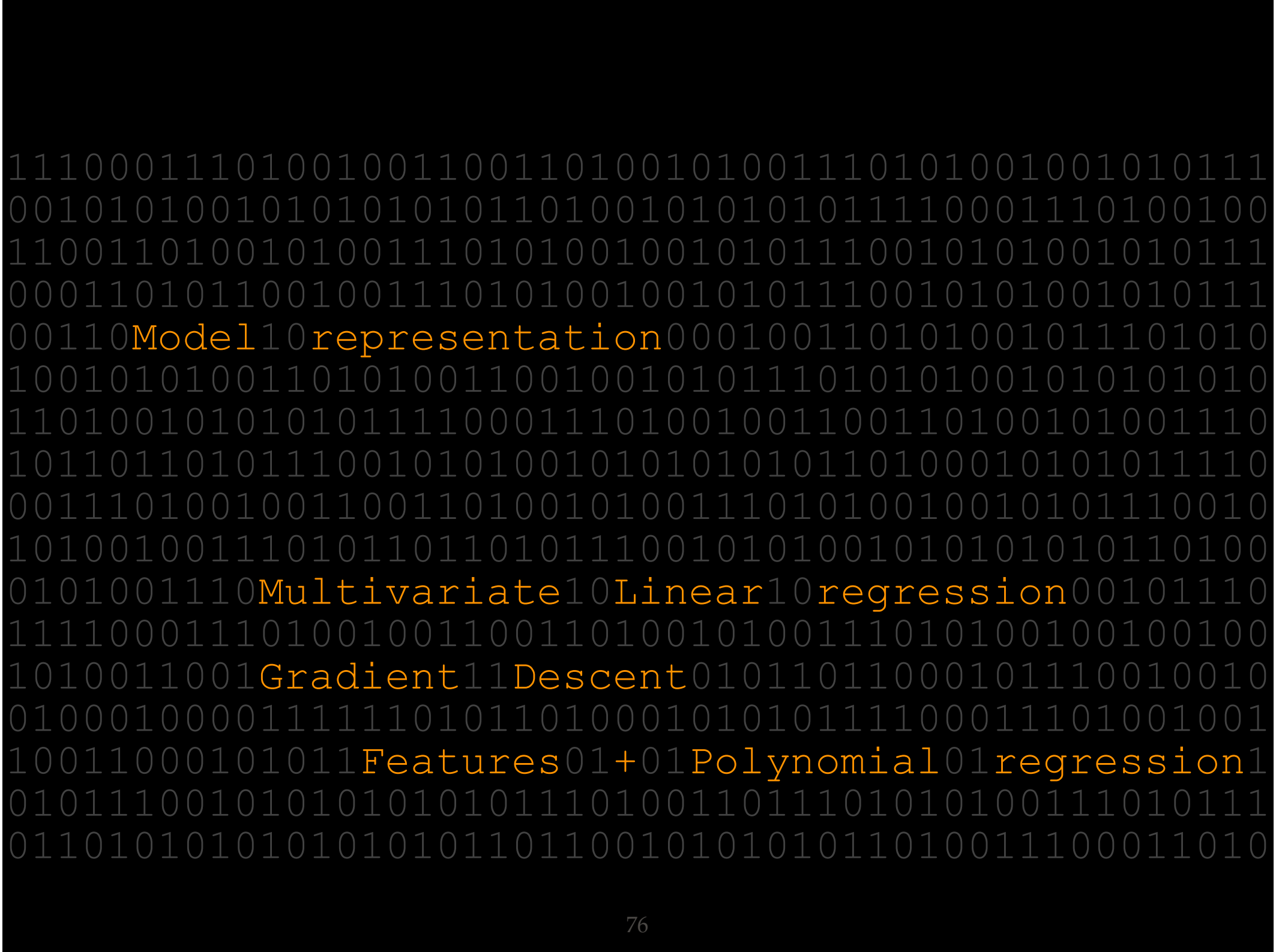
- alpha too small -> slow convergence
- alpha too large -> J may not decrease on every iteration, may eventually not converge

AML trick for beginners: to choose α, try a factor 10

- **0.001 ... 0.01 ... 0.1 ... 1**

AML trick for intermediate: to choose α, try a factor ~3

- **0.001 ... 0.003 ... 0.01 ... 0.03 ... 0.1 ... 0.3 ... 1**


Model representation
Multivariate Linear regression
Gradient Descent
Features + Polynomial regression

We now know about linear regression with multiple variables.

Now, it is time to discuss the possibility to choose features how you can build different models - sometimes very powerful ones! - by indeed choosing appropriate features.

In particular, **polynomial regression** allows you to use the (SIMPLE) machinery of linear regression to fit (complicated - even very non-linear) functions.

Use or create features?

Let's expand the example of predicting the price of the house.

Suppose you are considering two specific features of the house:

- the width (facade and the depth of the house: both can be meaningful in the dataset)

You might build a linear regression model as we did so far

- e.g. facade is your first feature x_1, depth is your second feature x_2

You don't necessarily have to use just the features x_1 and x_2 that you're given: you can **create new features** by yourself.

…

(think about it one sec.. doesn't it sound CRAZY?)

…

Use or create features?

Yes, it might sound crazy - as if you were "inventing data" - but you are actually **NOT adding any info to your dataset**, you are just **choosing how to present your data to the learning algo**

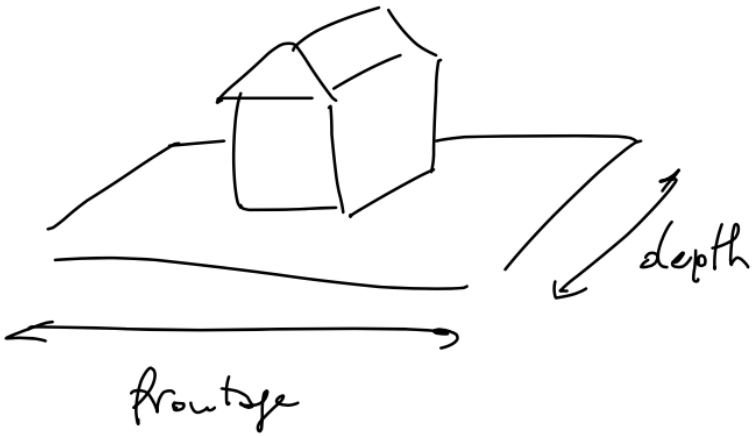
Example:

- If houses are in Amsterdam, for historical (tax) reasons the houses were built very narrow. The facade width is a meaningful parameter to predict the price, so retain x_1 as a feature!
- If houses are elsewhere, the same above is close to meaningless (if taken alone). If I want to predict the price of a house, what I might do instead is to focus on total size in m^2, or other parameters.

In this example, depending on the situation, I might either use x_1 and x_2, or create a new feature: $x_1 * x_2$. **This largely depends on what insight you might have into a particular problem, rather than just taking the features that you happen to have started off with**

- sometimes by defining new features you might actually get a better model

$$h_\theta(x) = \theta_0 + \theta_1 * \underbrace{\text{frontage}}_{x_1} + \theta_2 * \underbrace{\text{depth}}_{x_2}$$



$$\text{Area} \quad x = \text{frontage} * \text{depth}$$

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

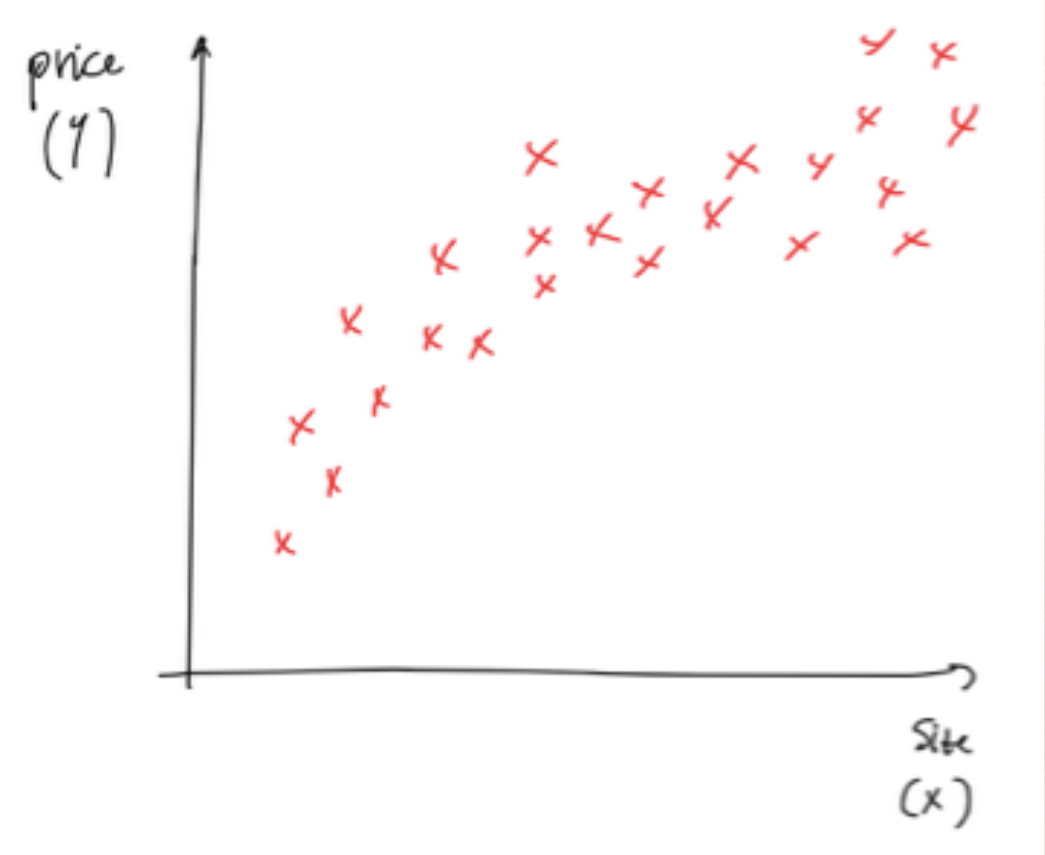
Another example: … angles vs difference of angles as features …

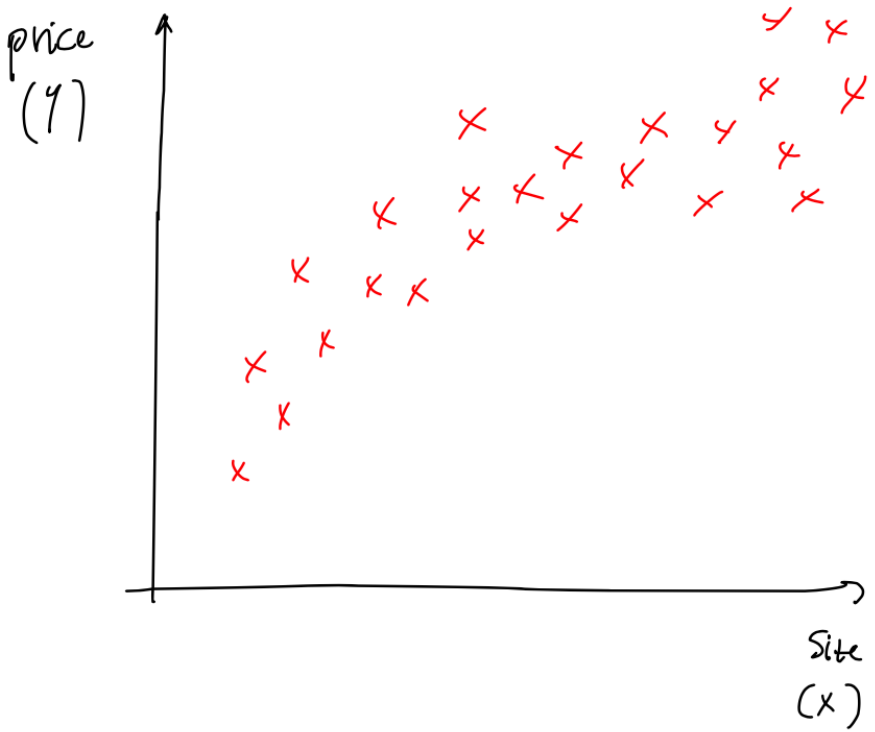
Polynomial regression

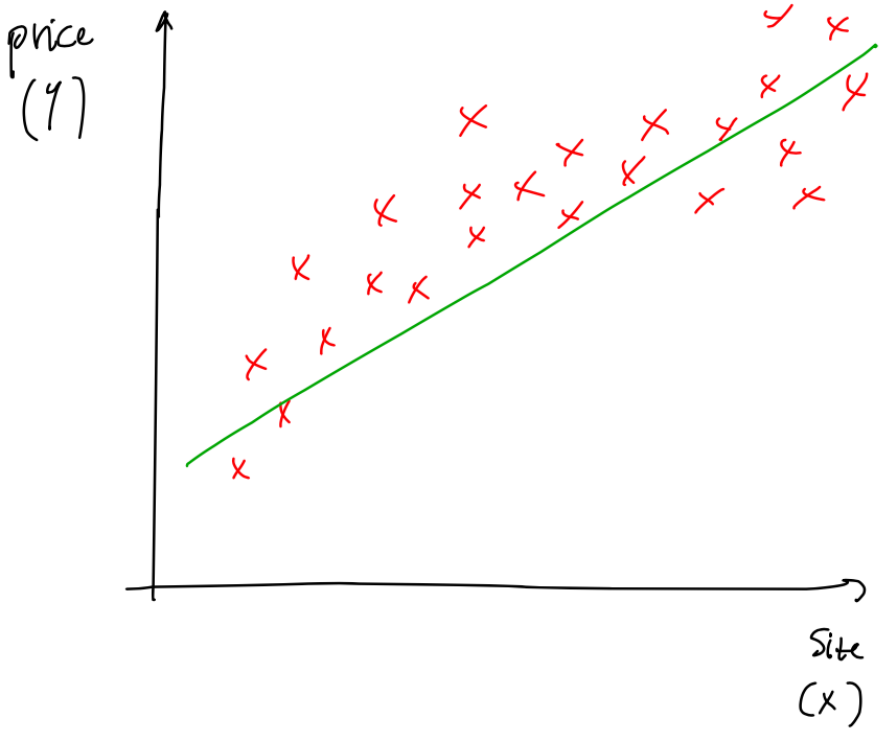
Closely related to the idea of "choosing your features" is the so-called **polynomial regression**.

Let's say you have a housing price dataset that looks like in the picture (right)

How would you build your hypothesis?

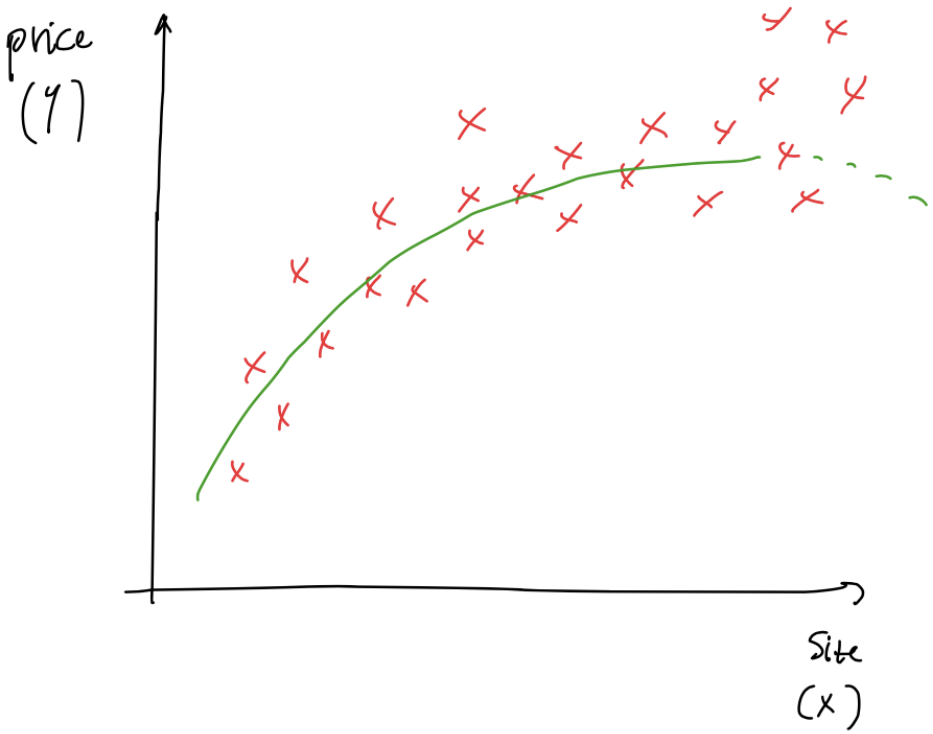



price
(y)
Site
(x)



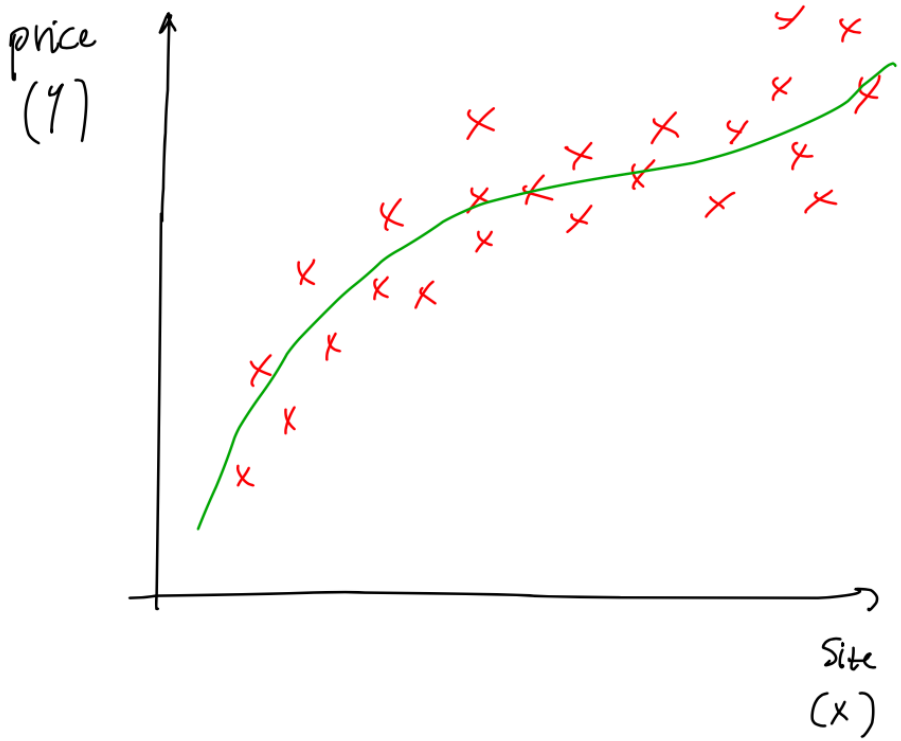
$\theta_0 + \theta_1 x$

?



$\theta_0 + \theta_1 x$

$\theta_0 + \theta_1 x + \theta_2 x^2$?

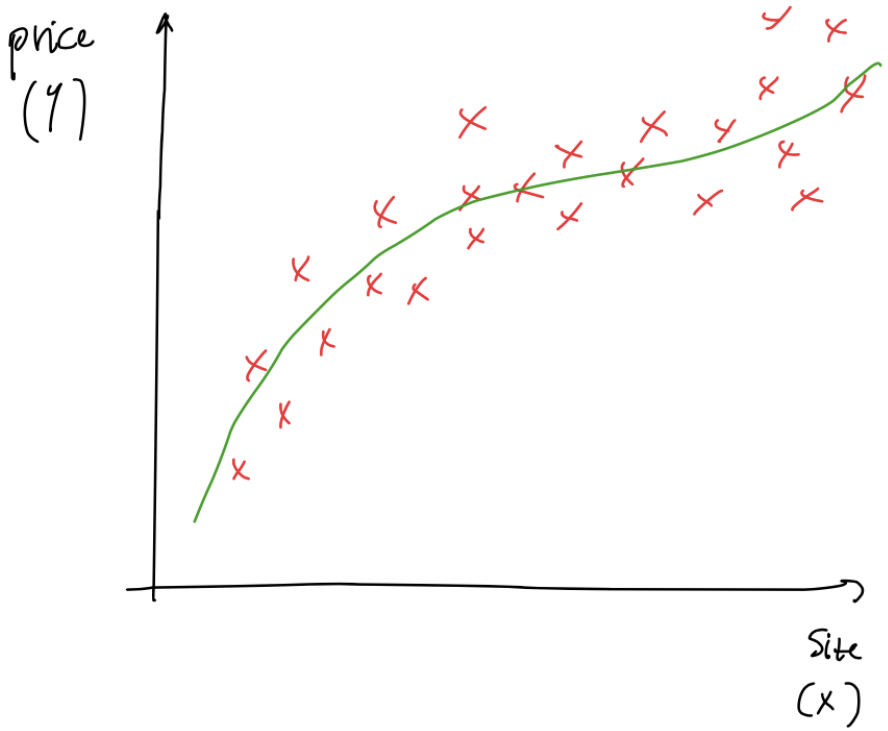


$\theta_0 + \theta_1 x$

$\theta_0 + \theta_1 x + \theta_2 x^2$

$\theta_0 + \theta_1 x + \theta_2 x^2 + \theta_3 x^3$

ONE POSSIBLE CHOICE



$\theta_0 + \theta_1 x$

$\theta_0 + \theta_1 x + \theta_2 x^2$

$\theta_0 + \theta_1 x + \theta_2 x^2 + \theta_3 x^3$

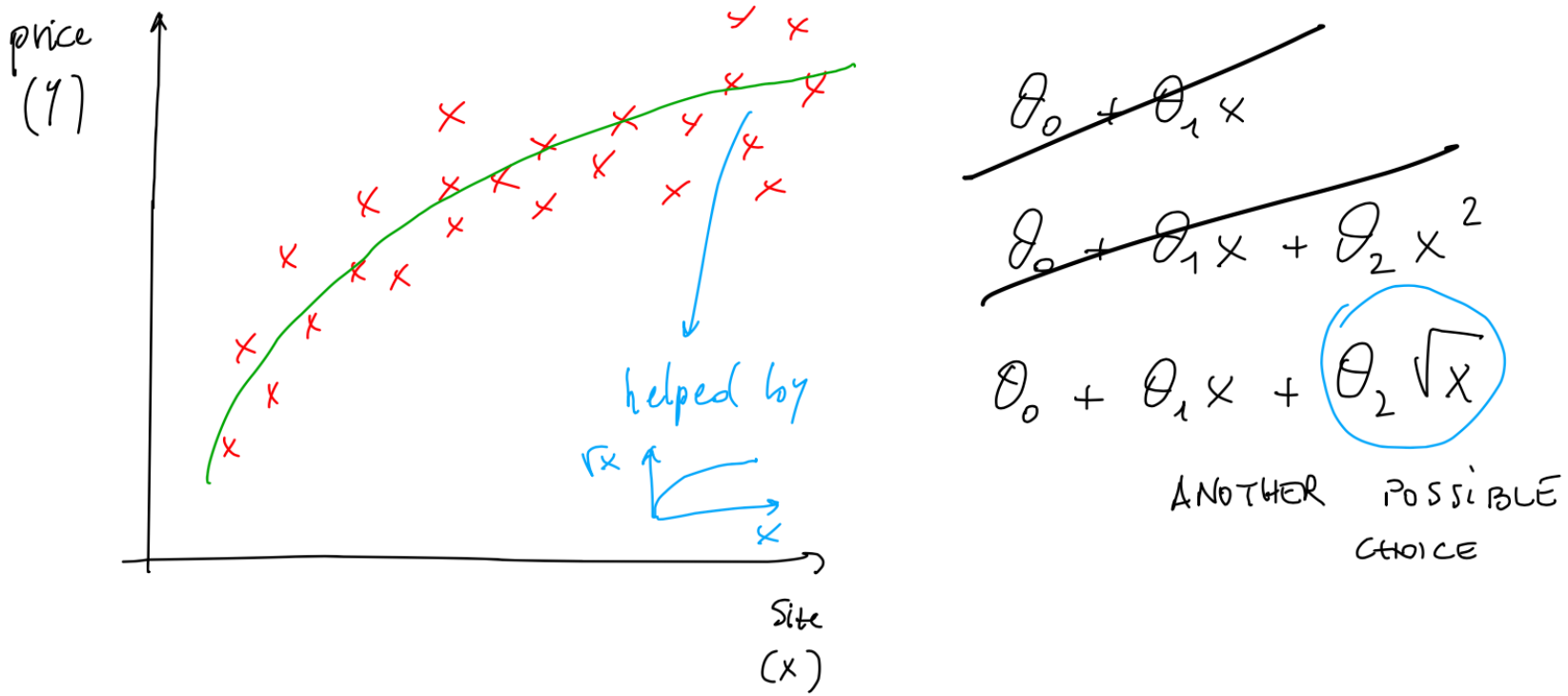
$$h_\theta(x) = \theta_0 + \theta_1 x_1 + \theta_2 x_2 + \theta_3 x_3$$
$$= \theta_0 + \theta_1 x + \theta_2 x^2 + \theta_3 x^3$$

3 FEATURES

(CAREFUL ABOUT FEATURE SCALING !!!)

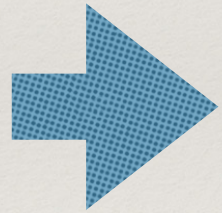
It is always the size !

size	1 - 1000 mq
size^2	1 - 10^6
size^3	1 - 10^9



This a clearly built-up feature.. but it may well work just fine for your case, so give it a try!

Ready for Quiz 3



Summary

Polynomial regression: how to fit a polynomial (like a quadratic or cubic functions) to your data - Fine, but most importantly an idea was thrown out: i.e. that **I have plenty of choice in what features to use**

- e.g. multiply two features
- e.g. take square of a feature, or cubic, or square root, …

Bewildering! how do I decide what features to use?

- experience… and there are algos that automatically choose what features to use (so you can have an algorithm look at the data and automatically choose for you whether you want to fit a quadratic function, or a cubic function, or something else)… but experience always wins..

Key point is that:

- by designing different features you can fit more complex functions to your data than just fitting a straight line to the data
- appropriate insight into the features simply give you a much better model


Model representation
Computing parameters analytically
Normal equation

The **normal equation** - at least for some linear regression problems - gives a good way to solve for the optimal value of the θ parameters.

Why stepping away from GD?

- Well, more like "stepping aside"..
- GD, in order to minimise the cost function $J(\theta)$, take an **iterative approach to derive θs**, we run an algo that by construction takes many steps, multiple iterations, to converge to the global minimum
- In contrast, the normal equation gives us a method towards an **analytical solution for θs**, so no need to 'iterate' over anything, just solve for the optimal value for θs in one go: in 1 step you get to the optimal value

Sounds like: "why did we introduce GD in the first place, then?!"

- **because the reality in which you apply ML does matter!**

One step at a time.

Road to the normal equation

The normal equation gives you the values of θs that minimises the cost function.

Let's see how.

First, we need to re-express our vectors and matrices a bit.

Example: m=4 training examples

Size (feet²) x_1	Number of bedrooms x_2	Number of floors x_3	Age of home (years) x_4	Price ($1000) y
2104	5	1	45	460
1416	3	2	40	232
1534	3	2	30	315
852	2	1	36	178

extra column added

x_0	Size (feet²) x_1	Number of bedrooms x_2	Number of floors x_3	Age of home (years) x_4	Price ($1000) y
1	2104	5	1	45	460
1	1416	3	2	40	232
1	1534	3	2	30	315
1	852	2	1	36	178

then build a "**design matrix**" that contains all features from the training data

$$X = \begin{bmatrix} 1 & 2104 & 5 & 1 & 45 \\ 1 & 1416 & 3 & 2 & 40 \\ 1 & 1534 & 3 & 2 & 30 \\ 1 & 852 & 2 & 1 & 36 \end{bmatrix}$$

m x (n+1)

and a vector with all the values I am going to predict

$$y = \begin{bmatrix} 460 \\ 232 \\ 315 \\ 178 \end{bmatrix}$$

m-dim vector

First: assume for simplicity $\theta \in \mathbb{R}$ (not vector)

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

$$\min_{\theta} \quad \Leftrightarrow \quad \frac{d}{d\theta} J(\theta) \overset{\text{set}}{=} 0 \text{ , and solve for } \theta$$

Second: $\theta \in \mathbb{R}^{n+1}$

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

$$\min_{(\theta_0, \theta_1, \dots, \theta_n)} \quad \Leftrightarrow \quad \frac{\partial}{\partial \theta_j} J(\theta_0, \theta_1, \dots, \theta_n) \overset{\text{set}}{=} 0 \text{ , and solve for } \theta_0, \theta_1, \dots, \theta_n$$

m examples $(x^{(1)}, y^{(1)}), \dots (x^{(m)}, y^{(m)})$ (vectors)

n features $x_1, x_2, \dots x_n$

$$x^{(i)} = \begin{pmatrix} x_0^{(i)} \\ x_1^{(i)} \\ \vdots \\ x_n^{(i)} \end{pmatrix} \in \mathbb{R}^{n+1} \qquad (x_0^{(i)} = 1)$$

$$y = \begin{pmatrix} y^{(1)} \\ y^{(2)} \\ \vdots \\ y^{(m)} \end{pmatrix}$$

m-dim vector

example : 1 feature only

$$x^{(i)} = \begin{pmatrix} x_0^{(i)} \\ x_1^{(i)} \end{pmatrix} = \begin{pmatrix} 1 \\ x_1^{(i)} \end{pmatrix}$$

$$X = \begin{pmatrix} 1 & x_1^{(1)} \\ 1 & x_1^{(2)} \\ \vdots & \\ 1 & x_1^{(m)} \end{pmatrix}$$

$m \times 2$ matrix

"DESIGN MATRIX"

$m \times (n+1)$ matrix

$$X = \begin{pmatrix} — & (x^{(1)})^T & — \\ — & (x^{(2)})^T & — \\ & \vdots & \\ — & (x^{(m)})^T & — \end{pmatrix} = \begin{pmatrix} x_0^{(1)} & x_1^{(1)} & \cdots & x_n^{(1)} \\ x_0^{(2)} & x_1^{(2)} & \cdots & x_n^{(2)} \\ & & \vdots & \\ x_0^{(m)} & x_1^{(m)} & \cdots & x_n^{(m)} \end{pmatrix}$$

$$X = \begin{pmatrix} x_0^{(1)} & x_1^{(1)} & \cdots & x_n^{(1)} \\ x_0^{(2)} & x_1^{(2)} & \cdots & x_n^{(2)} \\ & & \vdots & \\ x_0^{(m)} & x_1^{(m)} & \cdots & x_n^{(m)} \end{pmatrix} \qquad \theta = \begin{pmatrix} \theta_0 \\ \theta_1 \\ \vdots \\ \theta_n \end{pmatrix} \in \mathbb{R}^{n+1}$$

$$X\theta = \begin{pmatrix} x_0^{(1)} & x_1^{(1)} & \cdots & x_n^{(1)} \\ x_0^{(2)} & x_1^{(2)} & \cdots & x_n^{(2)} \\ & & \vdots & \\ x_0^{(m)} & x_1^{(m)} & \cdots & x_n^{(m)} \end{pmatrix} \begin{pmatrix} \theta_0 \\ \theta_1 \\ \vdots \\ \theta_n \end{pmatrix} = \begin{pmatrix} \theta_0 x_0^{(1)} + \theta_1 x_1^{(1)} + \ldots + \theta_n x_n^{(1)} \\ \theta_0 x_0^{(2)} + \theta_1 x_1^{(2)} + \ldots + \theta_n x_n^{(2)} \\ \\ \\ \theta_0 x_0^{(m)} + \theta_1 x_1^{(m)} + \ldots + \theta_n x_n^{(m)} \end{pmatrix}$$

$m \times (n+1)$ $\quad$ $(n+1)$-dim vector $\quad$ m-dim vector

Start from: $h_\theta(x) = \theta_0 x_0 + \theta_1 x_1 + \dots + \theta_n x_n$

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

and rewrite: $h_\theta(x) = \theta^T x$

$$J(\theta) = \frac{1}{2m} (X\theta - y)^T (X\theta - y)$$

(drop $\frac{1}{2m}$ as we are comparing a derivative to 0 anyway)

$$(X\theta - y)^T (X\theta - y) = ((X\theta)^T - y^T)(X\theta - y) =$$

$$= (X\theta)^T (X\theta) - (X\theta)^T y - y^T (X\theta) + y^T y$$

(note: $X\theta$ and y are both m-dim vectors)

$$= \theta^T X^T X\theta - 2(X\theta)^T y + y^T y$$

$\therefore$

$$\frac{\partial J}{\partial \theta} = \frac{\partial}{\partial \theta}\left(\theta^T X^T X\theta - 2(X\theta)^T y + y^T y\right) = 0$$

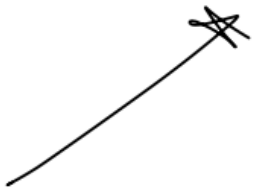
$$2X^T X\theta - 2X^T y = 0$$

$$X^T X\theta = X^T y \qquad \text{(assuming } X^T X \text{ is invertible, we get:)}$$

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

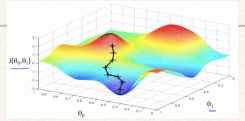
normal equation

This is the θ that $\min_\theta J(\theta)$.



GD vs normal equation

aka: an iterative process



aka: an analytic solution

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

need to choose alpha

- run it a few time and pick the best

needs many iterations

- depending on the details it would make it slower

needs feature scaling

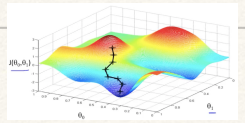
no need to choose alpha

do not need any iterations

feature scaling is irrelevant

GD vs normal equation

aka: an iterative process



aka: an analytic solution

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

need to choose alpha

- run it a few time and pick the best

needs many iterations

- depending on the details it would make it slower

needs feature scaling

works well even when n is large

- even millions of features
- cost ~scales as $O(kn^2)$

for some tasks (e.g. logistic regression algorithms) you need GD..

no need to choose alpha

do not need any iterations

feature scaling is irrelevant

slow if n is large. Need to compute $(X^T X)^{-1}$

- nxn matrix, so very high for high n
- matrix inversion cost ~scales as $O(n^3)$

empirically, a limit at $n \sim 10^4$..