Hes·so
Haute Ecole Spécialisée
de Suisse occidentale
Fachhochschule Westschweiz
University of Applied Sciences
Western Switzerland

MSE | MASTER OF SCIENCE IN ENGINEERING

Machine Learning

T-MachLe

4. Linear Regression (with gradient descent)

Jean Hennebert
Andres Perez Uribe

Plan - Linear Regression Revisited

4.1 Recap from last week and examples

4.2 Problem formulation

4.3 Batch Gradient Descent Algorithm

4.4 Stochastic Gradient Descent Algorithm

4.5 Extensions

Practical Work 4

DISCLAIMER !

- This week is going to be less funny than last weeks :(
  - We need to formalise things
  - There will be maths
  - The algorithm that we will study will not be as sexy as you may think

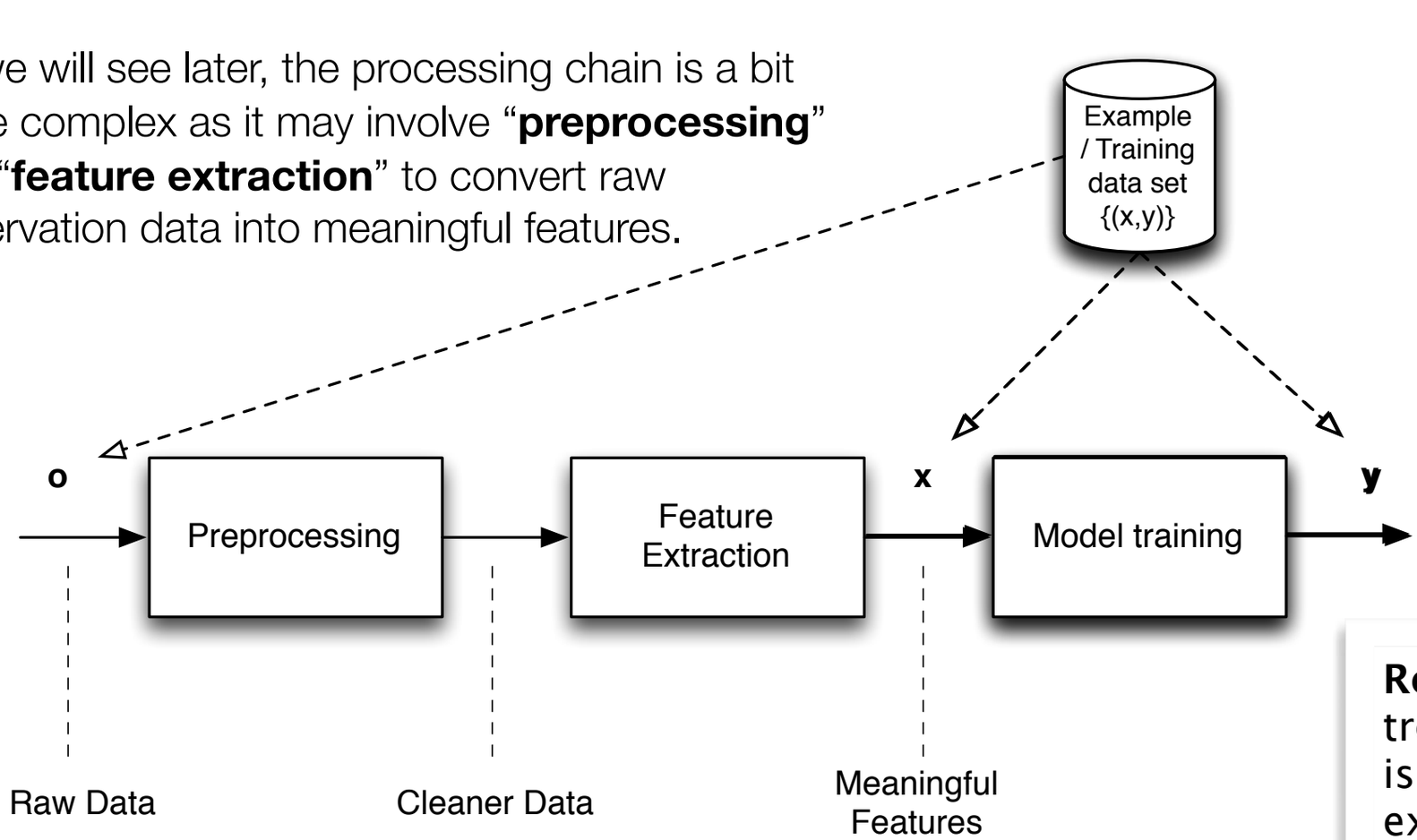
- But…
  - If you understand fully this one, a lot of things about machine learning will be understood later on!
  - So keep courage my braves!

4.1 Recap from last week and examples

Supervised learning

With **supervised learning**, the goal is to learn a mapping from inputs **x** to outputs **y** given a set of example data called the training set.

As we will see later, the processing chain is a bit more complex as it may involve "**preprocessing**" and "**feature extraction**" to convert raw observation data into meaningful features.



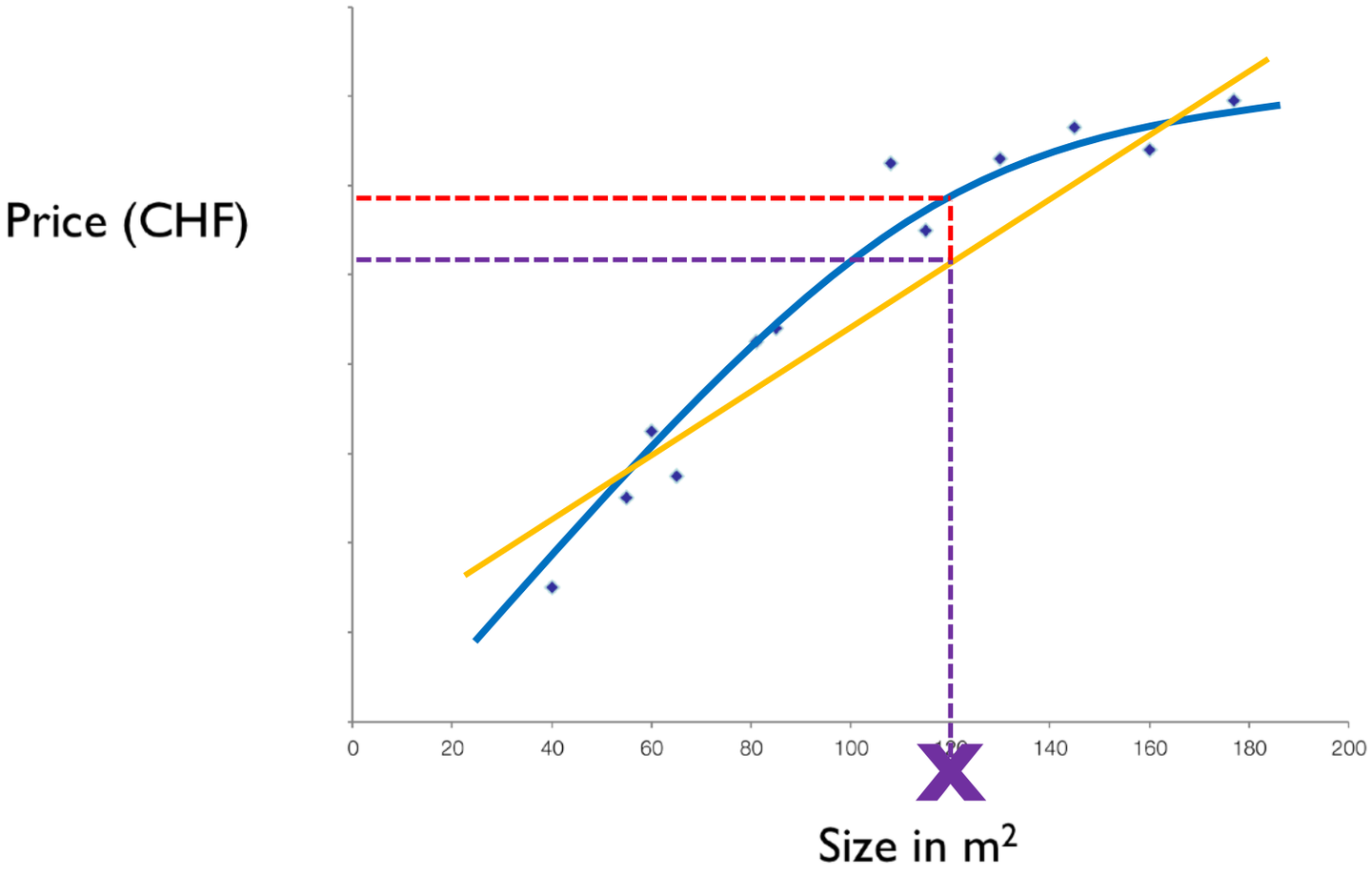
Remark: the recent trend of deep learning is to train the feature extraction step

Supervised learning - regression tasks

A **regression task** maps inputs **x** to an infinite set of continuous outputs **y**. The outputs are numeric values corresponding to the variable we want to predict.

- **x** are said to be "**explicative variables**"
- **y** are said to be the "**response variables**" or "dependent variables"
- in $\widehat{y} = h(x)$
  - if **x** is a scalar (vector of dimension 1), it is a **monovariate regression**, i.e. the response is influenced only by one input factor
  - if **x** is a vector (of dimension D), it is a **multivariate regression** i.e. the response is influenced by multiple factors

Supervised learning example 1 - house price prediction



The output of the system is a continuous value: this is called a **regression**

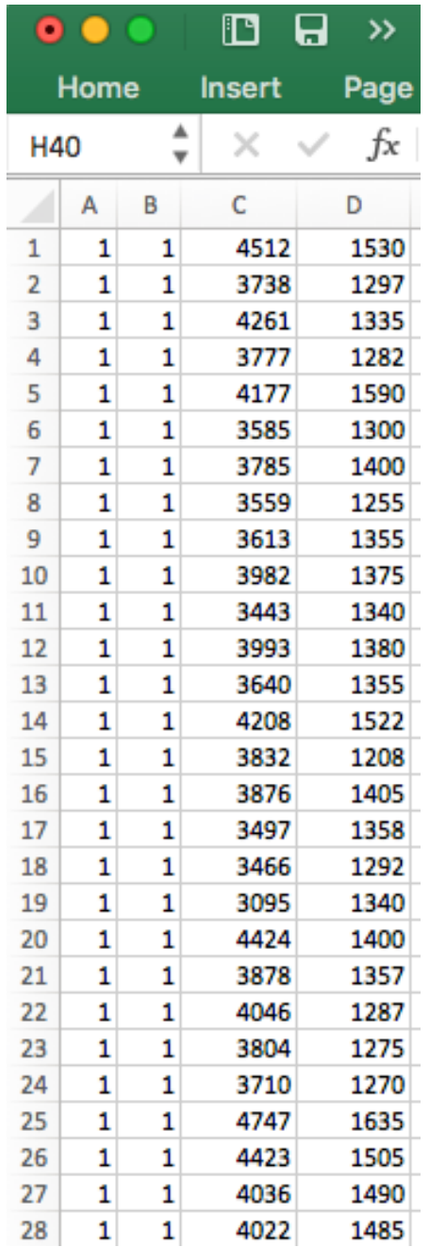
- We want to **predict** the price from the size.
- We have a set of price values with the *right answers*
- We build a mathematical model explaining the mapping: *price = h(size)*
  - yellow: linear model
  - blue: non linear model
- We can use the model to predict new house prices

Regression example - brain weight and head size - Part 1

Activity

- get ‘x02.csv’ from moodle
  - Source: R.J. Gladstone (1905). "A Study of the Relations of the Brain to the Size of the Head", Biometrika, Vol. 4, pp105-123
- open a new iPython notebook
- read column 3 (x) and column 2 (y) from the file
  - x is brain weight
  - y is head size
- visualize the data with a scatter plot

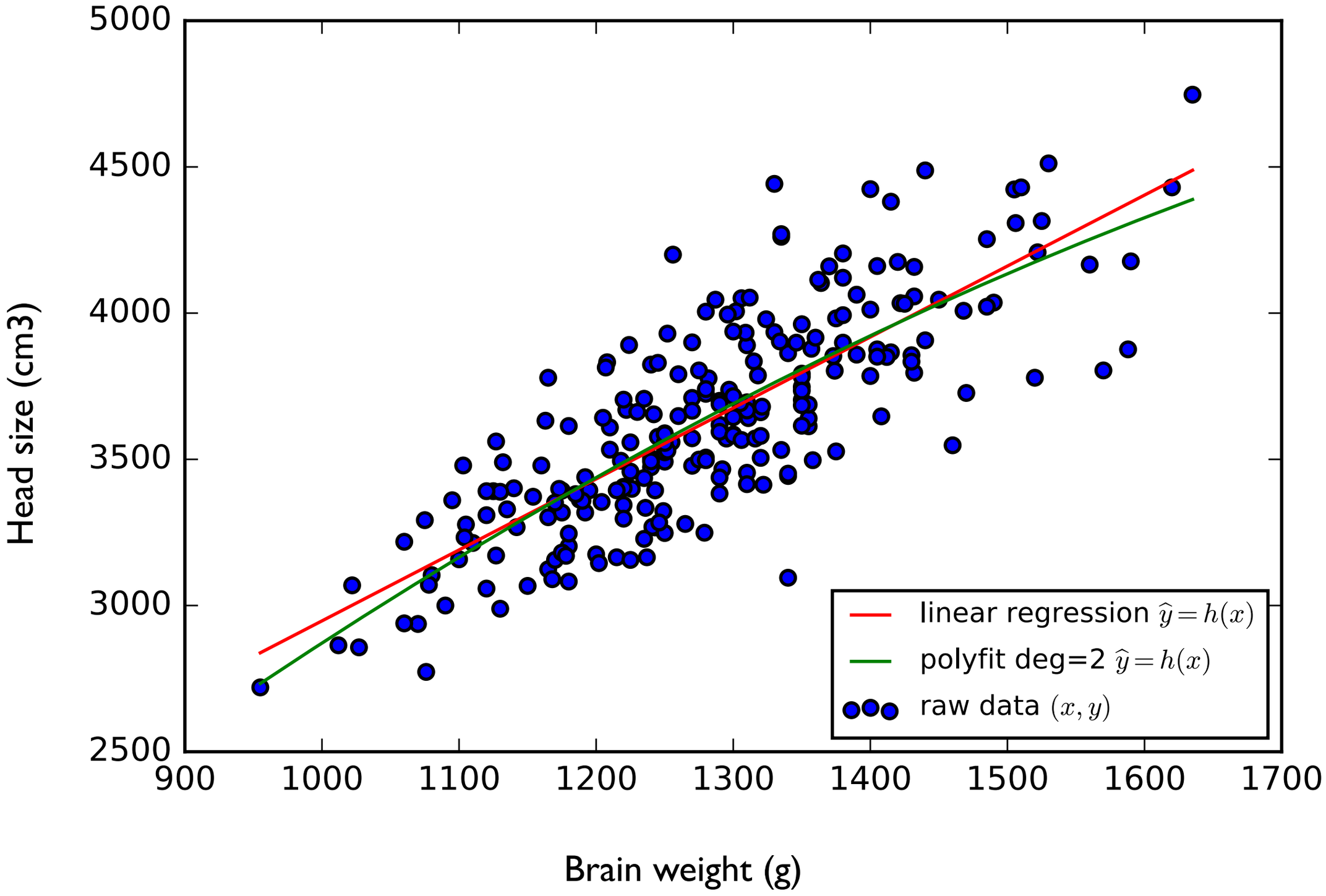
x2.csv



	A	B	C	D
1	1	1	4512	1530
2	1	1	3738	1297
3	1	1	4261	1335
4	1	1	3777	1282
5	1	1	4177	1590
6	1	1	3585	1300
7	1	1	3785	1400
8	1	1	3559	1255
9	1	1	3613	1355
10	1	1	3982	1375
11	1	1	3443	1340
12	1	1	3993	1380
13	1	1	3640	1355
14	1	1	4208	1522
15	1	1	3832	1208
16	1	1	3876	1405
17	1	1	3497	1358
18	1	1	3466	1292
19	1	1	3095	1340
20	1	1	4424	1400
21	1	1	3878	1357
22	1	1	4046	1287
23	1	1	3804	1275
24	1	1	3710	1270
25	1	1	4747	1635
26	1	1	4423	1505
27	1	1	4036	1490
28	1	1	4022	1485

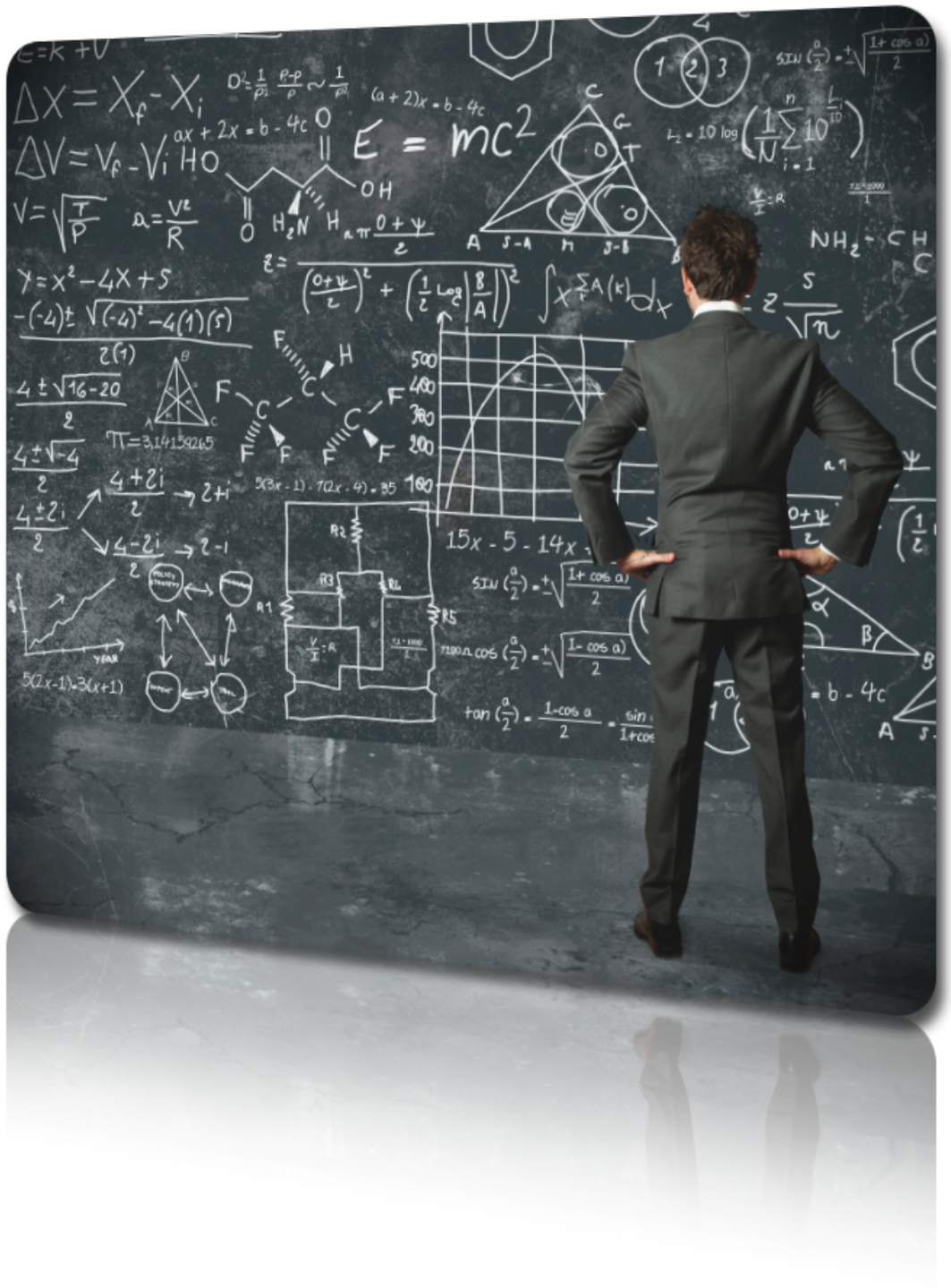
Regression example - brain weight and head size - Part 1

- Use the `polyfit` numpy function to find the coefficients of the polynomial that minimises the MSE
  - for a linear regression of the data
  - for a second order polynomial fit of the data
- Use the `poly1d` numpy function to encapsulate the polynomial and plot the regressions


5000
4500
4000
3500
3000
2500
900
1000
1100
1200
1300
1400
1500
1600
1700
Head size (cm3)
Brain weight (g)
linear regression $\widehat{y} = h(x)$
polyfit deg=2 $\widehat{y} = h(x)$
raw data (x, y)

4.2 Problem formulation

Linear regression
Cost function



Some notations

$\mathbf{x}$	input variable of the system, also called **input feature**
y	output variable of the system, also called **target output**
$\widehat{y}$	**gotten output**, as computed by the mapping function
$(\mathbf{x}, y)$	a **training sample** (pair of feature and corresponding target)
$(\mathbf{x}_n, y_n)$	the training sample at index n in the training set ("in row" n)
N	the total number of training samples in the training set

The **training set** can then be noted

$$\{(\mathbf{x}_n, y_n); n = 1, \ldots, N\}$$

Appart surface	Monthly rent price
26	890
37	955
57	1630
48	1390
60	1700
57	1630
76	2060
80	2270
103	2550
142	4260
…	…

Training data set

Appart surface	Monthly rent price
26	890
37	955
57	1630
48	1390
60	1700
57	1630
76	2060
80	2270
103	2550
142	4260
…	…

1

$\vdots$

n

$\vdots$

N

x y

We want to build a mathematical model to predict the rent price from the surface

We make the assumption that there is an unknown function $f(x)$ that we want to "learn" from the set of training examples given in the training set. Our goal is to approximate this function f by a function h that we learn on the data set.

$$\widehat{y} = h(x)$$

Parameters of the mapping function

- The mapping function $\widehat{y} = h(x)$ has **parameters** (also called *weights*) that we want to discover through the learning process.
- The set of parameters is usually noted the θ's.
- To emphasise the fact that the mapping function is parametrised by θ , we note

$$\widehat{y} = h_\theta(x)$$

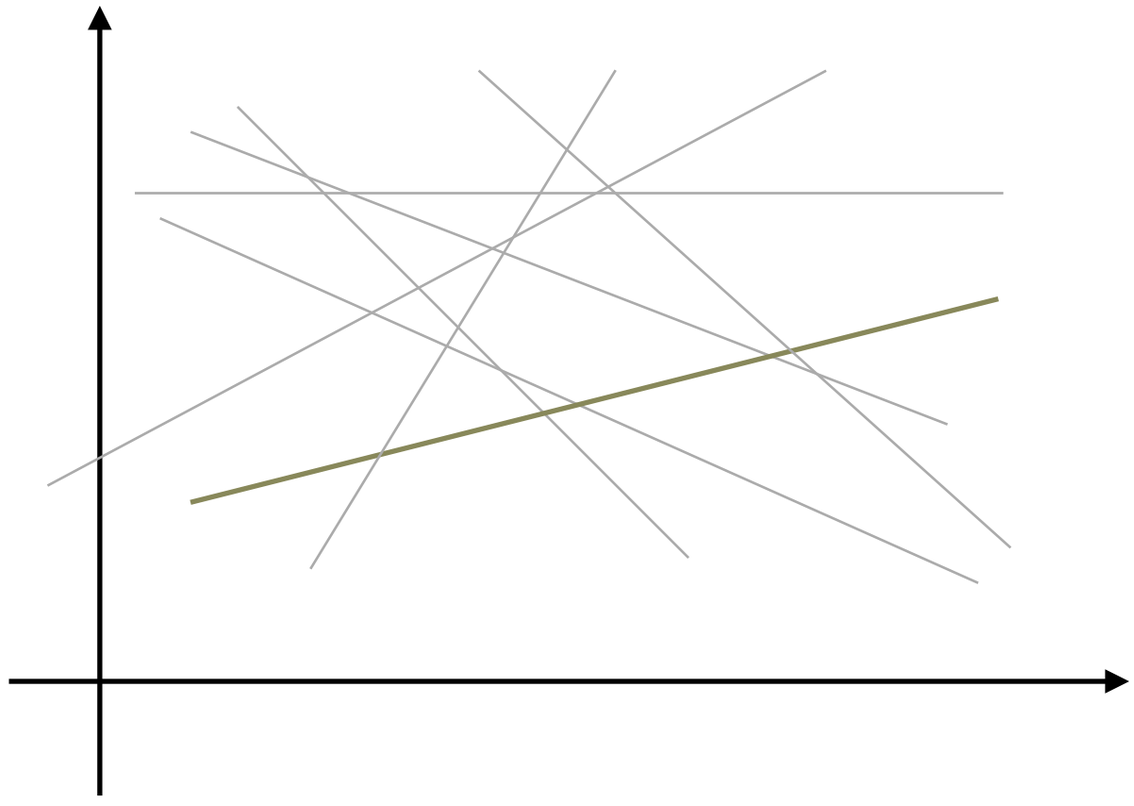
- In the case of a mono variable linear regression:

$$\widehat{y} = h_\theta(x) = \theta_0 + \theta_1 x \qquad \theta = \{\theta_0, \theta_1\}$$

"intercept" **"slope"**

$\theta = \{\theta_0, \theta_1\}$ defines the “family” of hypothesis functions, in this case all the lines in the (x,y) plane are candidates

$\theta = \{\theta_0, \theta_1\}$ defines the “family” of hypothesis functions, in this case all the lines in the (x,y) plane are candidates



One line is going to be better than all the other ones, and we want to find it!

A potential “stupid brute force” algorithm would be to explore a very large set of parameters and to retain the best.

Another algorithm would be to generate two variations from one hypothesis, to eliminate the worst and to iterate (kind of “genetic” approach)

Generalised notations

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

- Let's rewrite this equation using vector notations, defining $x_0 = 1$

Bold notation to denote a vector

$$h_\theta(\mathbf{x}) = \theta_0 x_0 + \theta_1 x_1$$

$$h_\theta(\mathbf{x}) = \sum_{i=0}^{1} \theta_i x_i$$

$$h_\theta(\mathbf{x}) = \theta^T \mathbf{x} \qquad \text{with} \qquad \mathbf{x} = \begin{pmatrix} x_0 \\ x_1 \end{pmatrix}$$

$$\theta = \begin{pmatrix} \theta_0 \\ \theta_1 \end{pmatrix}$$

Defining a cost function

- A reasonable approach is to chose the θ's so that we minimise the difference between the prediction $\widehat{y} = h_\theta(x)$ and the target example value y
- In plain english: minimise $(h_\theta(x) - y)$ for all training example (x, y)

"Minimise the difference between gotten output and target"

- In math notations:

$$\min_\theta \frac{1}{2N} \sum_{n=1}^{N} (h_\theta(\mathbf{x}_n) - y_n)^2$$

This is for later math simplifications

This is to normalise the cost regarding the size of training set

The square has some reasons we don't develop now – let's take it as is for now

MSE cost function - Mean Square Error

- Let's define the cost function $J(\theta)$ for which we will have to discover the minimum

$$J(\theta) = \frac{1}{2N} \sum_{n=1}^{N} (h_\theta(\mathbf{x}_n) - y_n)^2$$

- If we can solve $\min_\theta J(\theta)$, then we find the θ's defining our regression line

Remark: in some books, the normalisation term 1/N is not used. We prefer to use is as it makes the cost function independent to the size of the training set. It may also help by making the learning step alpha (see next slides) independent to the training set size.

From pure math approach: closed form solution

- There is a “closed form” solution to the previous problem

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

- With

Remember: D = dim of features,
N = number of samples in train set

Remember: $x_0 = 1$

$$X = \begin{pmatrix} 1 & x_{1,1} & \cdots & x_{1,D} \\ 1 & & \ddots & \\ 1 & \vdots & x_{n,d} & \vdots \\ 1 & & \ddots & \\ 1 & x_{N,1} & \cdots & x_{N,D} \end{pmatrix} \qquad y = \begin{pmatrix} y_1 \\ \vdots \\ y_N \end{pmatrix}$$

- From a Machine Learning perspective:
  - We may get into troubles if X is large
  - We are also more interested into another algorithm called **gradient descent** that demonstrates nice learning properties

4.3 Batch gradient descent algorithm

Properties

Principles

Mathematical formulation

Properties of gradient descent

- As we will see soon, gradient descent algorithms have some nice **properties**:
  - it can handle **very large sets of data** (especially the stochastic form of gradient descent - see next Section 4.4)
  - it allows for **incremental learning**, i.e. on-the-fly adaptation of the model on new incoming data
  - the learning principle is **generalisable** to many other form of "hypothesis families" $h_\theta(x)$, including neural networks, deep neural networks, etc.

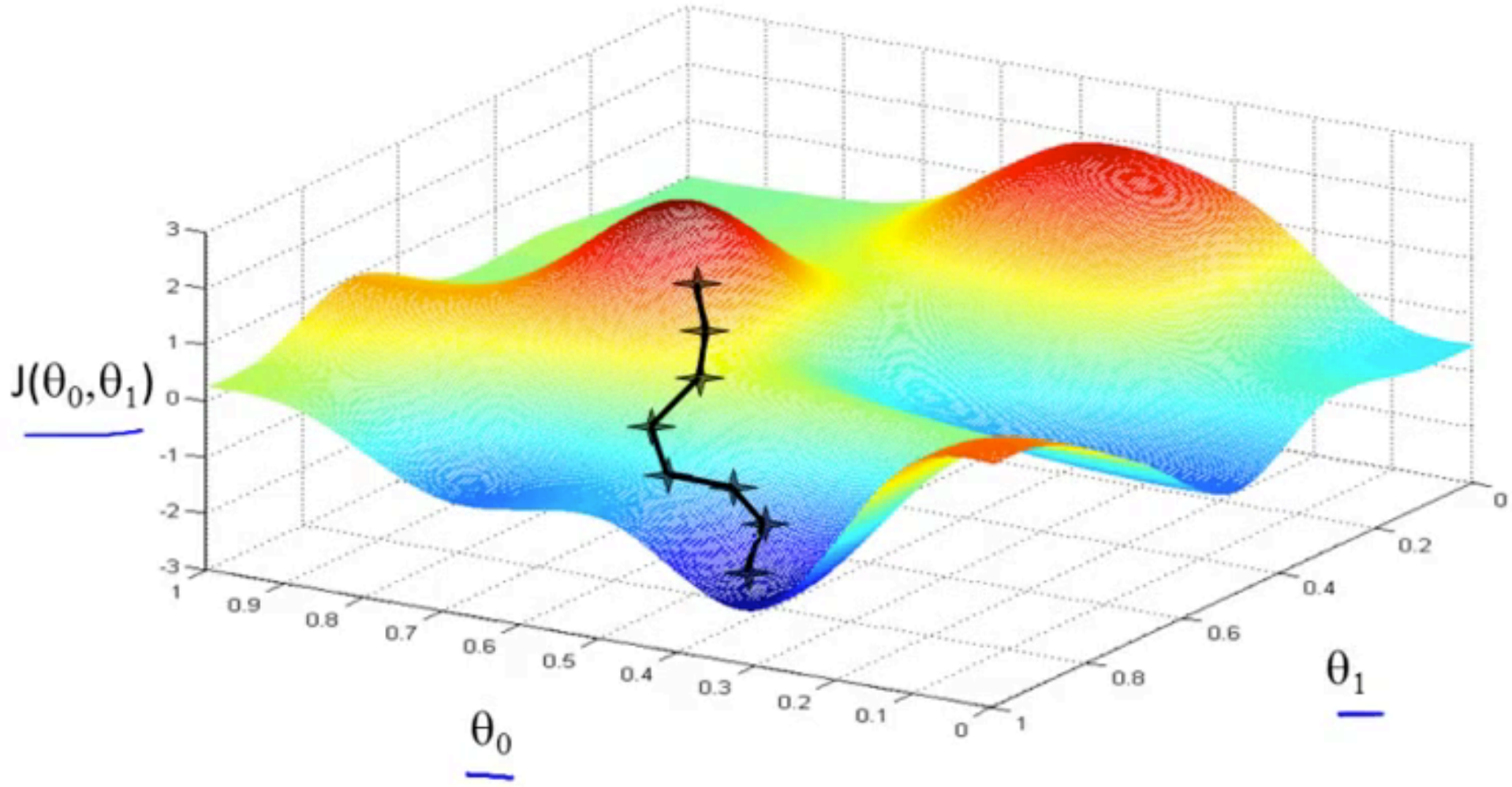
“Full” batch gradient descent principles

1. Start with some initial θ’s (for example random or null)
2. Visit the full training set to compute new values of the θ’s reducing $J(\theta)$
3. Loop in 2 until convergence

A visit of the training set is called an **epoch**

- The new values of θ’s are chosen according to the “gradient” of $J(\theta)$, i.e. in the opposite direction of the slope.

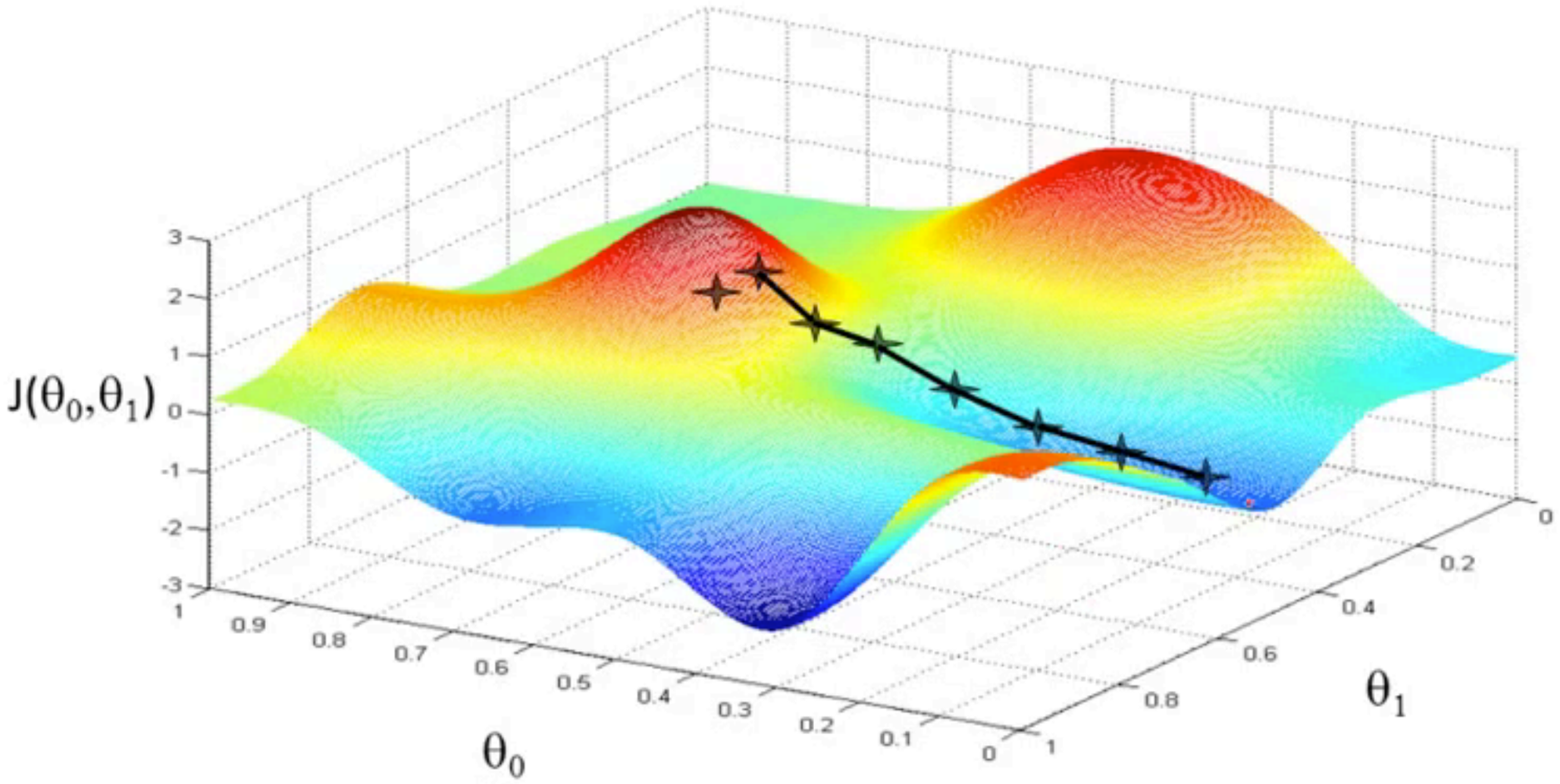
Gradient descent intuition



We take steps in the inverse direction of the slope, “going downhill”, until we reach a minimum.

Source: Andrew Ng - Machine Learning class - Stanford

Gradient descent intuition



We **depend** to the initial conditions. In this case, a slight difference in the initial parameter values, we end in another minimum, a local minimum which is above the global minimum reached in the previous run.

Actually, in the case of a linear regression, the J() function is a quadratic one, “bowl shaped” with a unique minimum, so we don’t really care so far by “getting stuck in a local minimum”. For more complex functions J( ) as the ones in deep neural networks, different runs may lead to different networks (and performances)

Source: Andrew Ng - Machine Learning class - Stanford

Gradient descent update rule

Explanation: the parameter theta is replaced by its last value plus a step alpha in the opposite direction (minus sign) of the gradient of the cost function.

$$\theta_0 \leftarrow \theta_0 - \alpha \frac{\partial J(\theta)}{\partial \theta_0}$$

$$\theta_1 \leftarrow \theta_1 - \alpha \frac{\partial J(\theta)}{\partial \theta_1}$$

We need now to compute this term, see math development next pages

- Generalised notations:

$$\theta_i \leftarrow \theta_i - \alpha \frac{\partial J(\theta)}{\partial \theta_i}$$

Mathematical development for θ_0

$$J(\theta) = \frac{1}{2N}\sum_{n=1}^{N}(h_\theta(\mathbf{x}_n) - y_n)^2$$

$$\frac{\partial J(\theta)}{\partial \theta_0} = \frac{\partial}{\partial \theta_0}\frac{1}{2N}\sum_{n=1}^{N}(h_\theta(\mathbf{x}_n) - y_n)^2$$

$$= 2 \cdot \frac{1}{2N}\sum_{n=1}^{N}(h_\theta(\mathbf{x}_n) - y_n)\frac{\partial}{\partial \theta_0}(h_\theta(\mathbf{x}_n) - y_n)$$

Chain rule

$$= \frac{1}{N}\sum_{n=1}^{N}(h_\theta(\mathbf{x}_n) - y_n)\frac{\partial}{\partial \theta_0}(\theta_0 x_{n,0} + \theta_1 x_{n,1} - y_n)$$

Only this term depends to θ_0

$$= \frac{1}{N}\sum_{n=1}^{N}(h_\theta(\mathbf{x}_n) - y_n)x_{n,0}$$

Gotten output

Target value

Actually $x_{n,0} = 1.0$ in this case, see slide 21

- The gradient for parameter θ_i is the sum over all training data of the product of the difference between gotten output and target, and the training coefficient $x_{n,i}$

Mathematical development

- Similarly, for θ_1 we get

$$\frac{\partial J(\theta)}{\partial \theta_1} = \frac{\partial}{\partial \theta_1} \frac{1}{2N} \sum_{n=1}^{N} (h_\theta(\mathbf{x}_n) - y_n)^2$$

$$= \frac{1}{N} \sum_{n=1}^{N} (h_\theta(\mathbf{x}_n) - y_n) x_{n,1}$$

Gotten output

Target value

Self exercise: try to re-do this math development by yourself.

Gradient descent update rule

$$\theta_0 \leftarrow \theta_0 - \alpha \frac{1}{N} \sum_{n=1}^{N} (h_\theta(\mathbf{x}_n) - y_n)$$

$$\theta_1 \leftarrow \theta_1 - \alpha \frac{1}{N} \sum_{n=1}^{N} (h_\theta(\mathbf{x}_n) - y_n) x_{n,1}$$

As $x_{n,0} = 1$

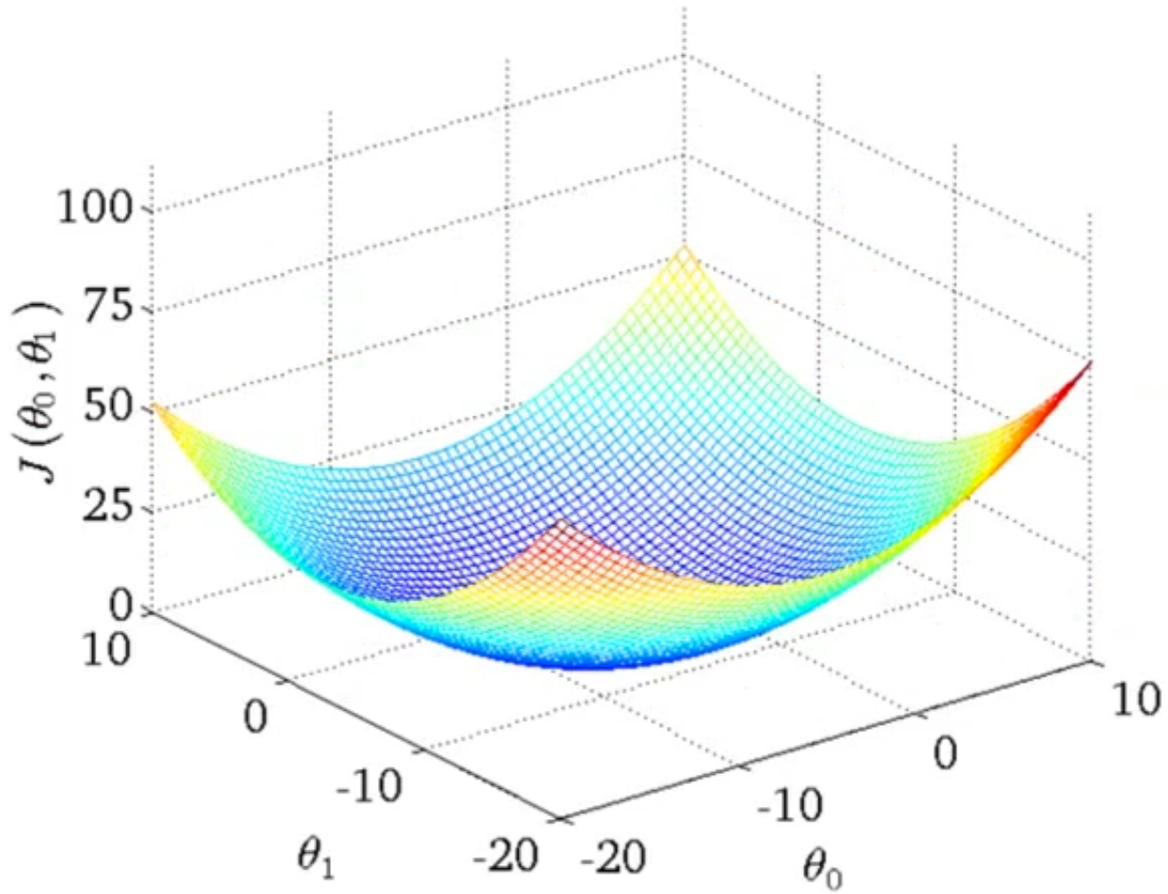
- Generalised notations:

$$\theta_i \leftarrow \theta_i - \alpha \frac{1}{N} \sum_{n=1}^{N} (h_\theta(\mathbf{x}_n) - y_n) x_{n,i}$$

This equation makes sense. If the system predicts perfectly, there is no update.

"error signal", real value

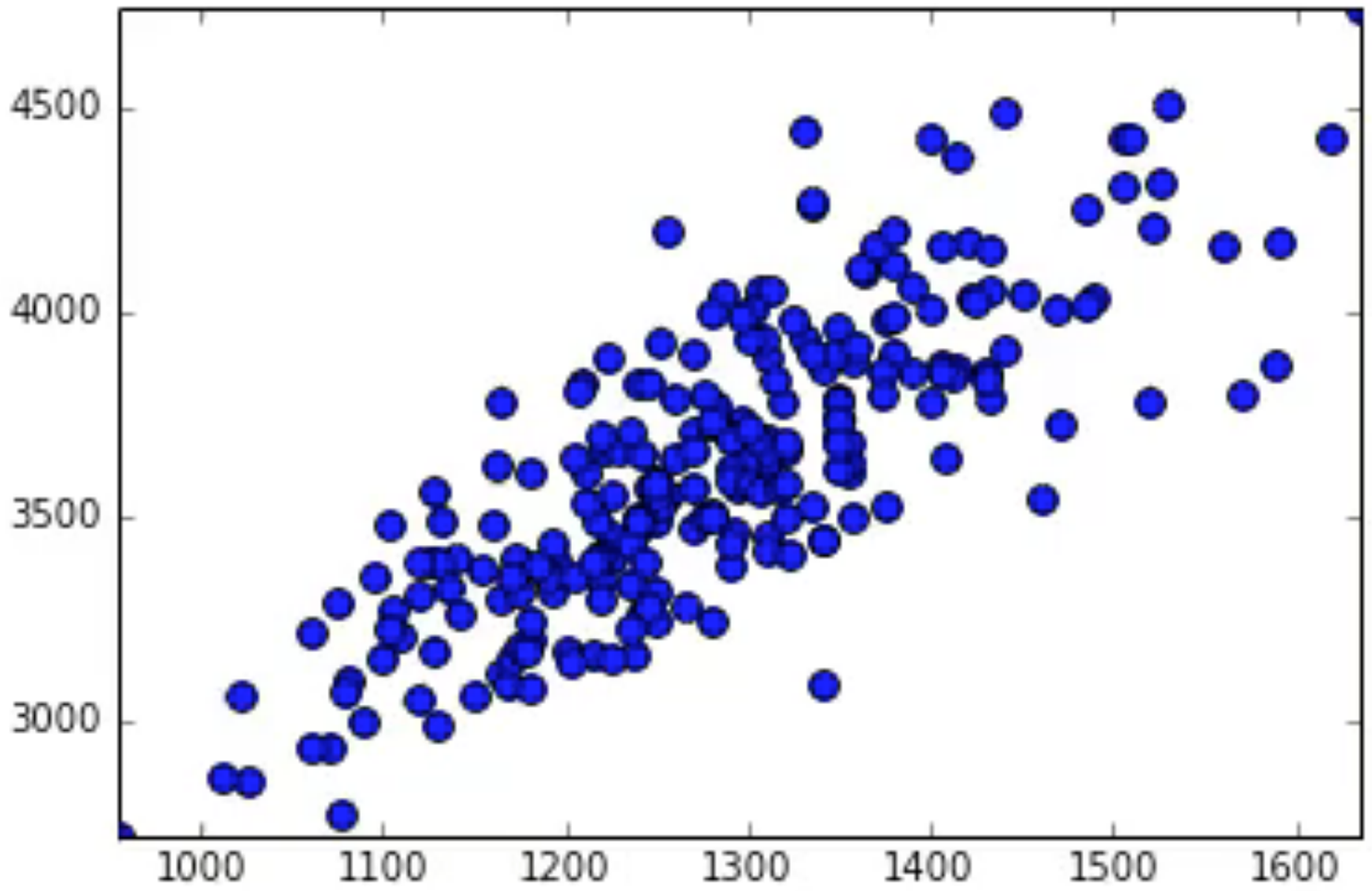
Quadratic cost function



- For linear regression, $J()$ is a quadratic function
  - We are guaranteed to approach, epoch after epoch the global minimum (unless the step is too large)
  - As the slope gets decreasing when approaching the minimum, the gradient term is also decreasing, slowing down the adaptation of the parameters.
    - This is not bad as we slow down when approaching the solution
    - On the other side, we may never reach the perfect solution

Source: Andrew Ng - Machine Learning class - Stanford

Batch gradient descent animation



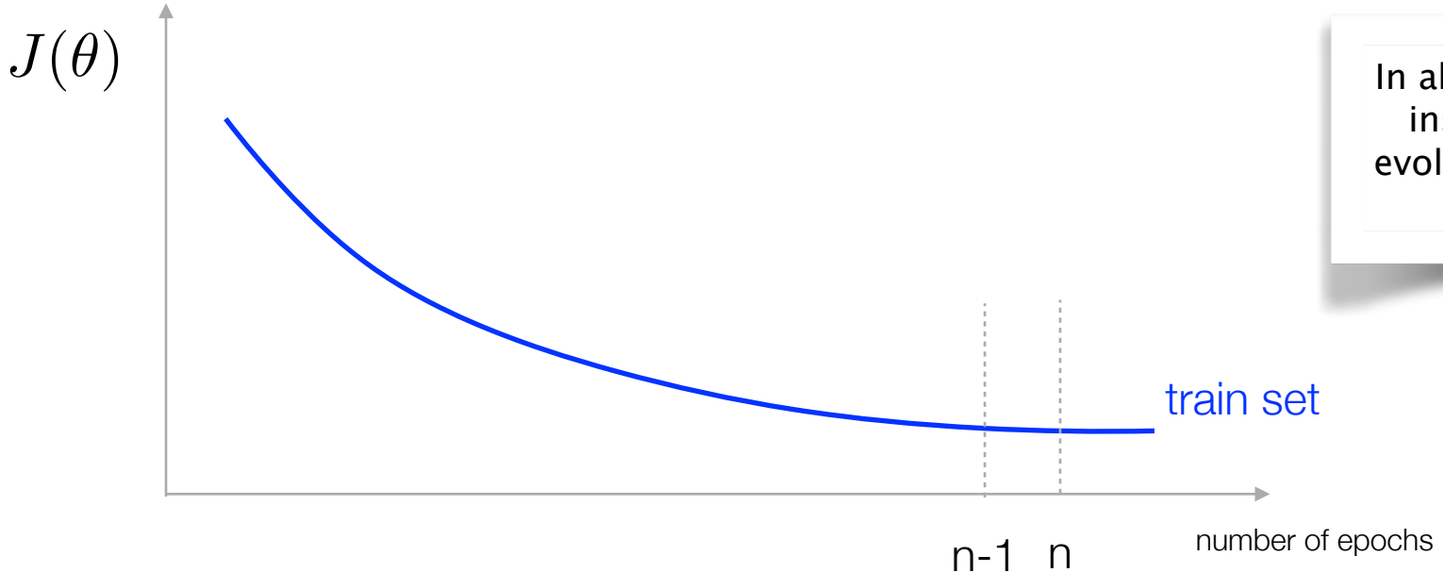
Remark on learning step α

- α will condition the learning speed
  - if too large, we may oscillate around the minimum and even diverge
  - if too small, we may spend a lot of time getting to the minimum
- α needs to be tuned to a given problem
  - an optimal α value for a given problem may not work for another problem
  - α is an hyper-parameter

When to stop iterating ?

- Simple strategy: fix a maximum number of epochs
- Advanced strategy: inspect the evolution of the cost function with, for example

$$\frac{J(\theta)^{epoch_{n-1}} - J(\theta)^{epoch_n}}{J(\theta)^{epoch_n}} < \epsilon$$



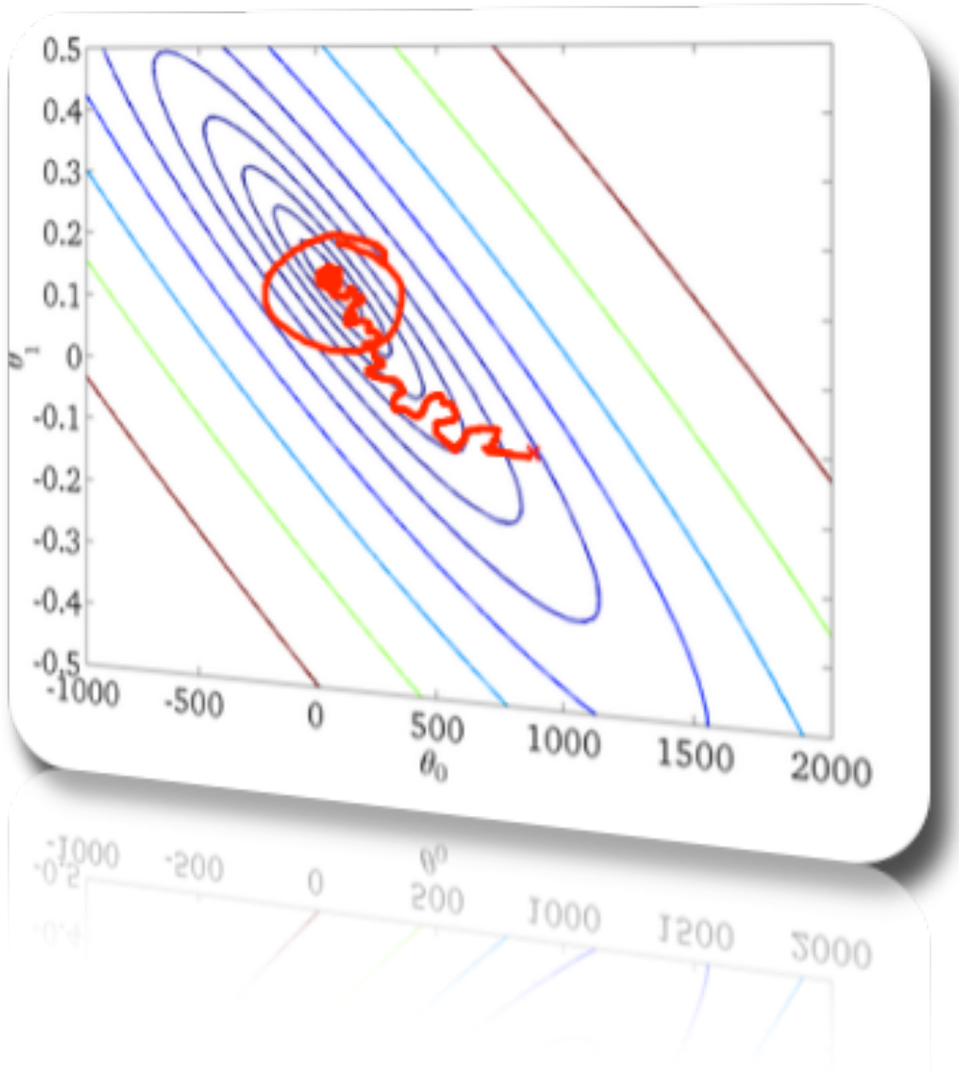
In all case, we recommend to inspect this cost function evolution over the epochs, as illustrated here.

4.4 Stochastic gradient descent algorithm

Principles

Properties

Extension to mini-batch



Stochastic gradient descent principle

1. Start with some initial θ's (for example random or null)
2. Select **1** training sample randomly in the set and perform the update of the θ's with this example

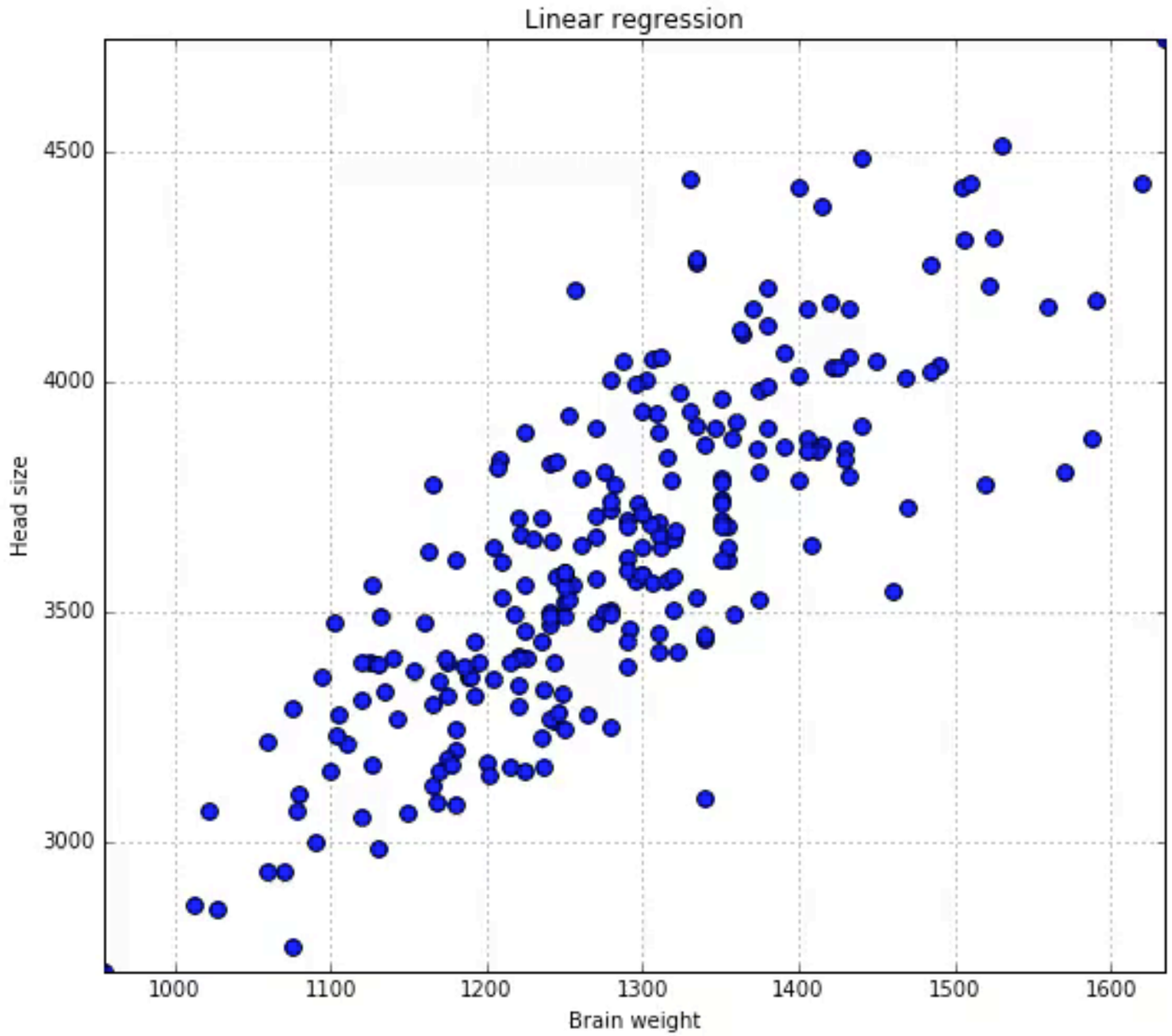
$$\theta_i \leftarrow \theta_i - \alpha(h_\theta(\mathbf{x}_n) - y_n)x_{n,i}$$

3. Loop in 2 until convergence

The **convergence** is trickier to observe as we may select favorable and less favorable points in the training set.

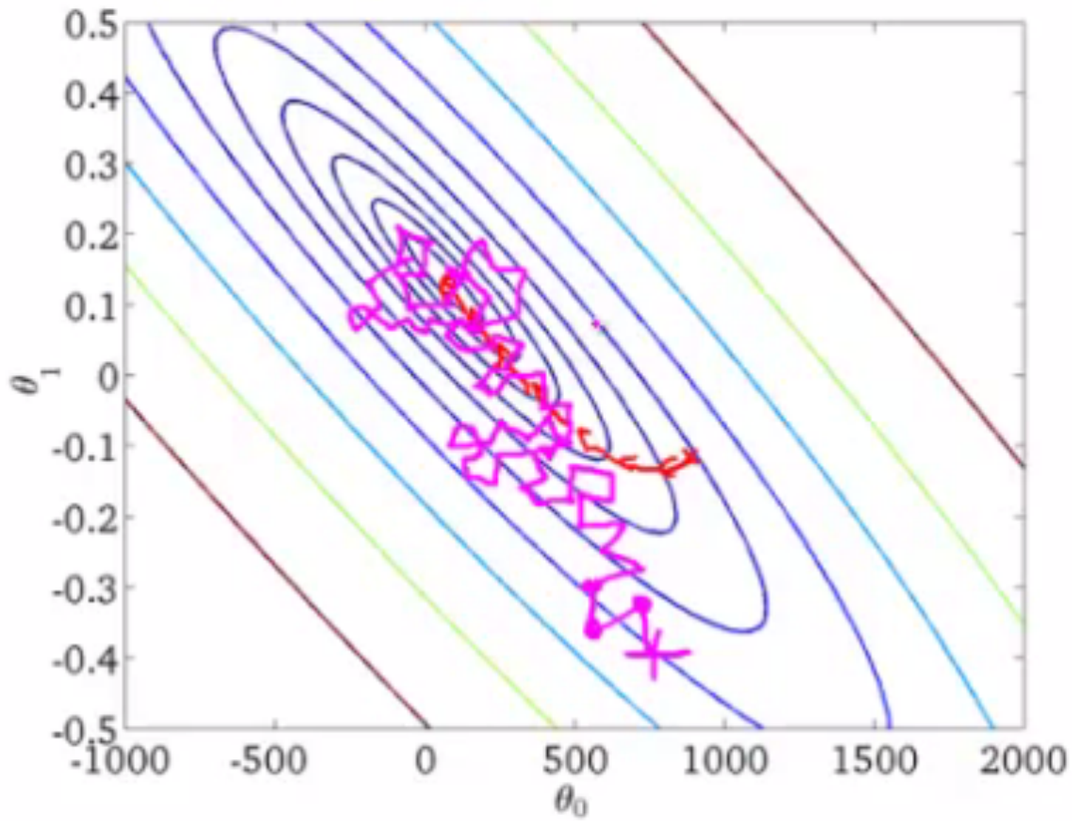
In practice we may compute an "averaged" cost function over the past U updates. Typically U<=N and U<<N if N is really large.

Stochastic gradient descent animation



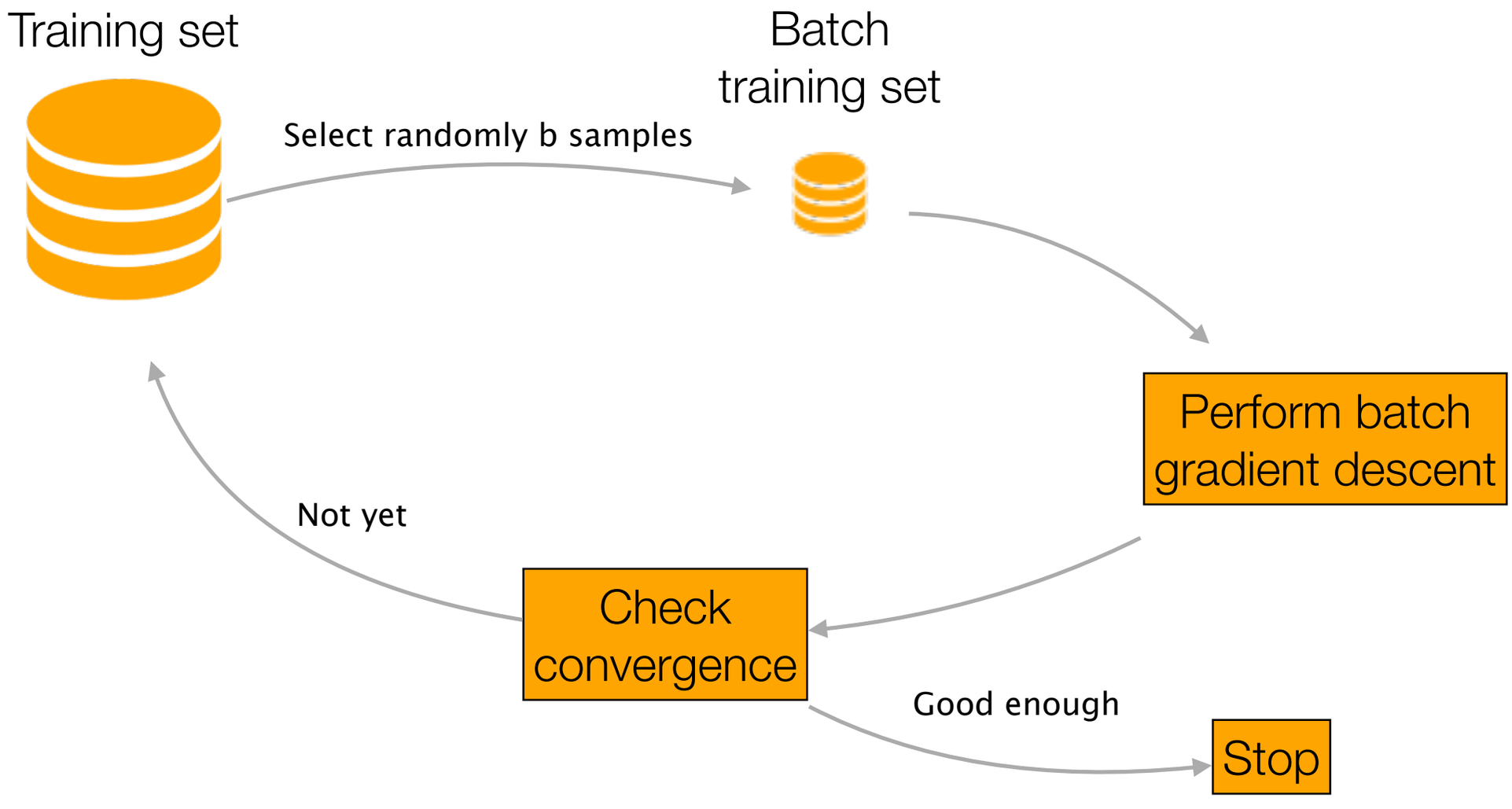
Stochastic gradient descent properties

- We update the parameters for each training samples, which is faster
- We "generally" move in the direction of the global minimum, but not always
- We never actually converges like batch gradient descent does, but ends up wandering around close to the global minimum
  - in practice, this isn't a problem - as long as we are close to the minimum that's probably OK
- **Advantages**
  - It can handle very large sets of data
  - It allows for incremental learning, i.e. on-the-fly adaptation of the model on new incoming data
  - The learning principle is generalisable to many other forms of “hypothesis families”



Extension to mini-batch

- Apply batch approach on sub-parts of the training set



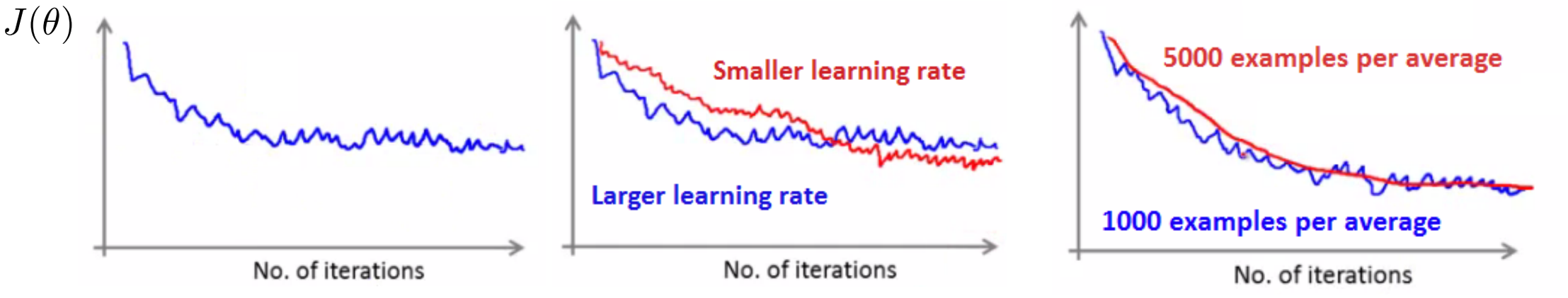
Advantages of mini-batch

- Advantage: the adaptation noise of stochastic gradient descent is reduced
- It allows for vectorised implementations
  - more efficient (pipelining), use of gpu
- It allows to distribute mini-batches on several machines or cores
  - map/reduce pattern
    - map = distribute the mini-batches, for example 50
    - reduce = update parameters with the results of the 50 mini-batches

- A disadvantage is that *b* needs to be optimised

Convergence of stochastic gradient descent

- We don't want to pause the algorithm to compute the cost function on the whole training data set
  - the point of stochastic gradient descent is to avoid those whole-data summations
- In practice we compute an "averaged" cost function over the past U updates. Typically U<=N and U<<N if N is really large.
  - For example, we may plot the cost $J(\theta)$ after 1'000 stochastic updates easy to cumulate it as we anyway compute it for the update



Source of figures: http://www.holehouse.org/mlclass/17_Large_Scale_Machine_Learning.html

2.5 Extensions of linear regression

Multi-variables

Polynomial regression

So far we assumed only one variable x

Training data set

Appart surface	Monthly rent price	
26	890	1
37	955	
57	1630	⋮
48	1390	
60	1700	
57	1630	n
76	2060	
80	2270	⋮
103	2550	
142	4260	
…	…	N

x y

Now moving to multi-variables $x_1, x_2, \ldots$

Training data set

Surface (m2)	#bedrooms	Monthly	
26	1	890	1
37	1.5	955	
57	2	1630	
48	2	1390	
60	2.5	1700	
57	2.5	1630	n
76	3	2060	
80	3.5	2270	
103	3.5	2550	
142	4	4260	
…	…	…	N
x_1	x_2	y	

Multi-variable problem formulation

$$h_\theta(\mathbf{x}) = \theta_0 + \theta_1 x_1 + +\theta_2 x_2 + \ldots + \theta_D x_D$$

$$h_\theta(\mathbf{x}) = \sum_{i=0}^{D} \theta_i x_i = \theta^T \mathbf{x}$$

- We have *D* parameters to discover through the learning process
- The good news is that all the equations we had before are working

$$\theta_i \leftarrow \theta_i - \alpha \frac{\partial J(\theta)}{\partial \theta_i}$$

$$\theta_i \leftarrow \theta_i - \alpha \frac{1}{N} \sum_{n=1}^{N} (h_\theta(\mathbf{x}_n) - y_n) x_{n,i}$$

Moving to polynomial regression

- Let's assume we "suspect" a dependency of *y* to the square of the living area

$$h_\theta(\mathbf{x}) = \theta_0 + \theta_1 x_1 + \theta_2 x_1^2$$

- We just need to build the following table and apply multi-variables as in the previous slide

surface	surface^2	Monthly
26	26*26 = 676	890
37	1369	955
57	3249	1630
48	2304	1390
…	…	…

x_1 x_2 y

Compute a new column $x_2 = x_1^2$ and treat it as before! This is **magical** isn't it?

Conclusions

- A regression task is a supervised learning task where we build a model $h_\theta(x)$ that maps inputs x to an infinite set of continuous outputs y
- Models for linear regression can be computed either mathematically (closed form solution) or with **gradient descent**
- Gradient descent can handle **very large data sets**, can do **incremental learning**, is **generalisable to many forms of model**, including the most advanced deep neural architectures
- **Principles** of gradient descent algorithm
  - **Batch gradient descent**: use all N examples in each iteration
  - **Stochastic gradient descent**: use 1 example in each iteration
  - **Mini-batch gradient descent**: use b examples in each iteration