Machine Learning
Lecture 6 - Linear Regression

Profa. Dra. Esther Luna Colombini
esther@ic.unicamp.br

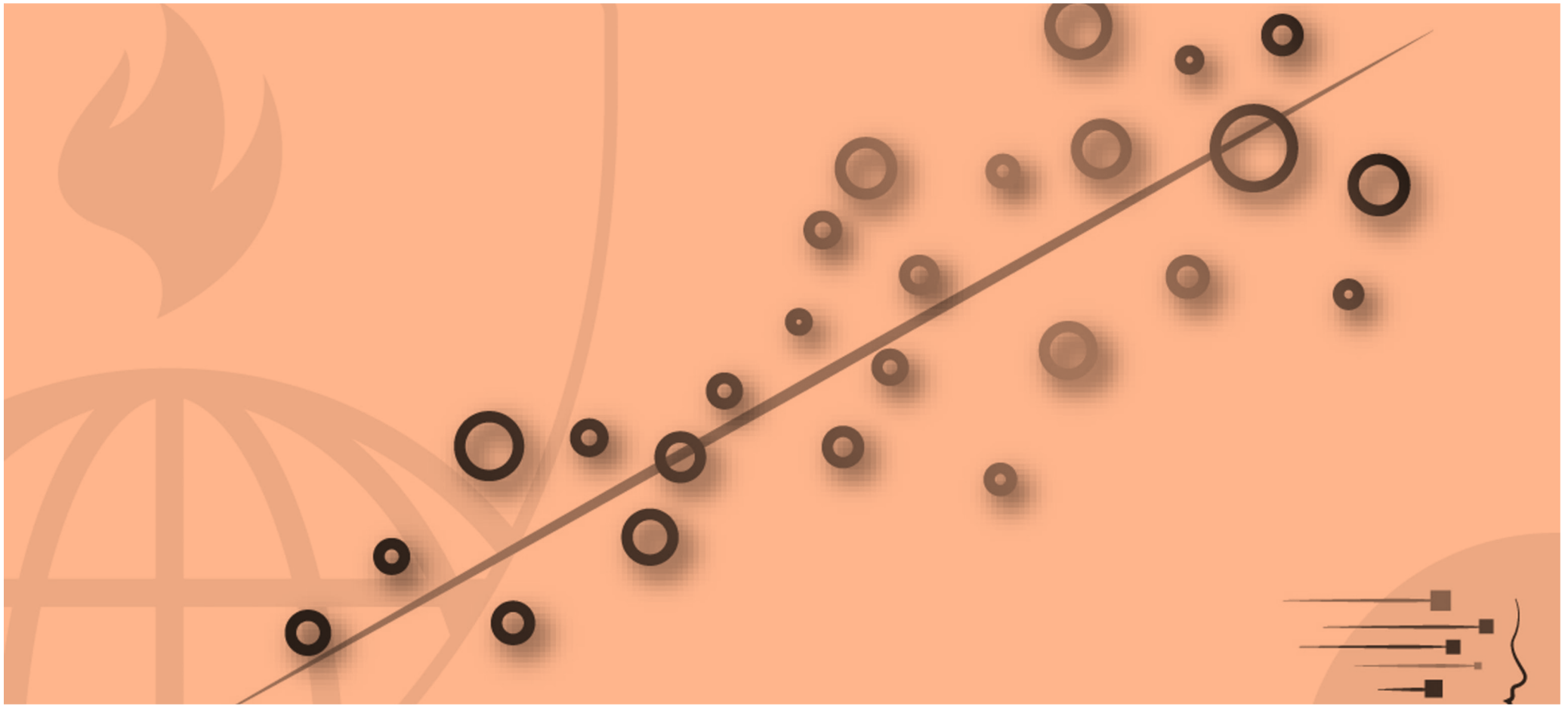
Prof. Dr. Alexandre Simoes
alexandre.simoes@unesp.br

LaRoCS – Laboratory of Robotics and Cognitive Systems

Advanced Institute for Artificial Intelligence

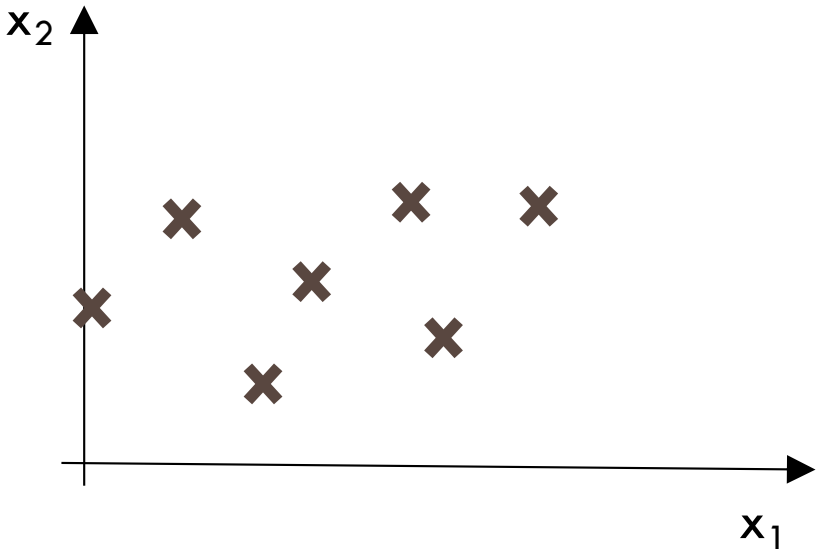
Summary

- Regression task
- Linear Regression
  - One Variable
    - Model Representation
    - Cost Function
    - Gradient Descent
    - Learning Rate
  - Multiple Variables
    - Gradient Descent for Multiple Variables
- Polynomial Regression

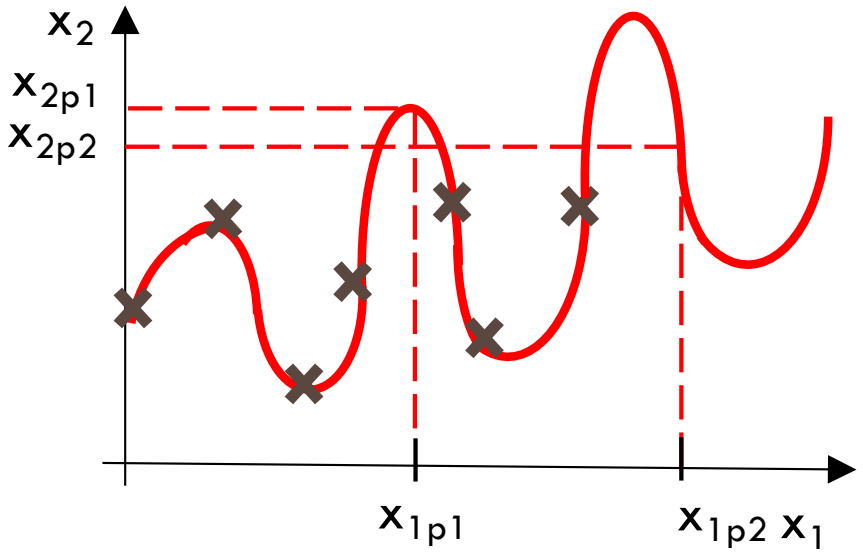


Regression task

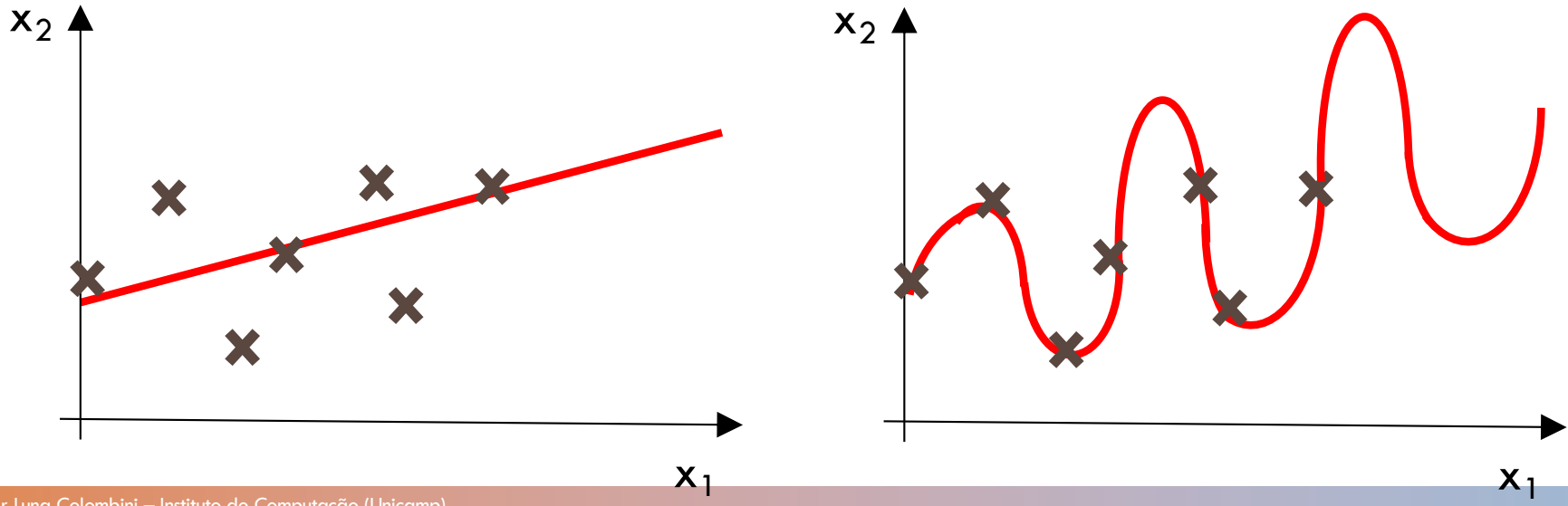
- Given some data points, the goal is to learn a **mathematical model** capable to fit data with a **curve**, so that the curve passes as close as possible to all of the data points
- This approach typically allows to **predict** the **numerical** output for points that were not part of the original data

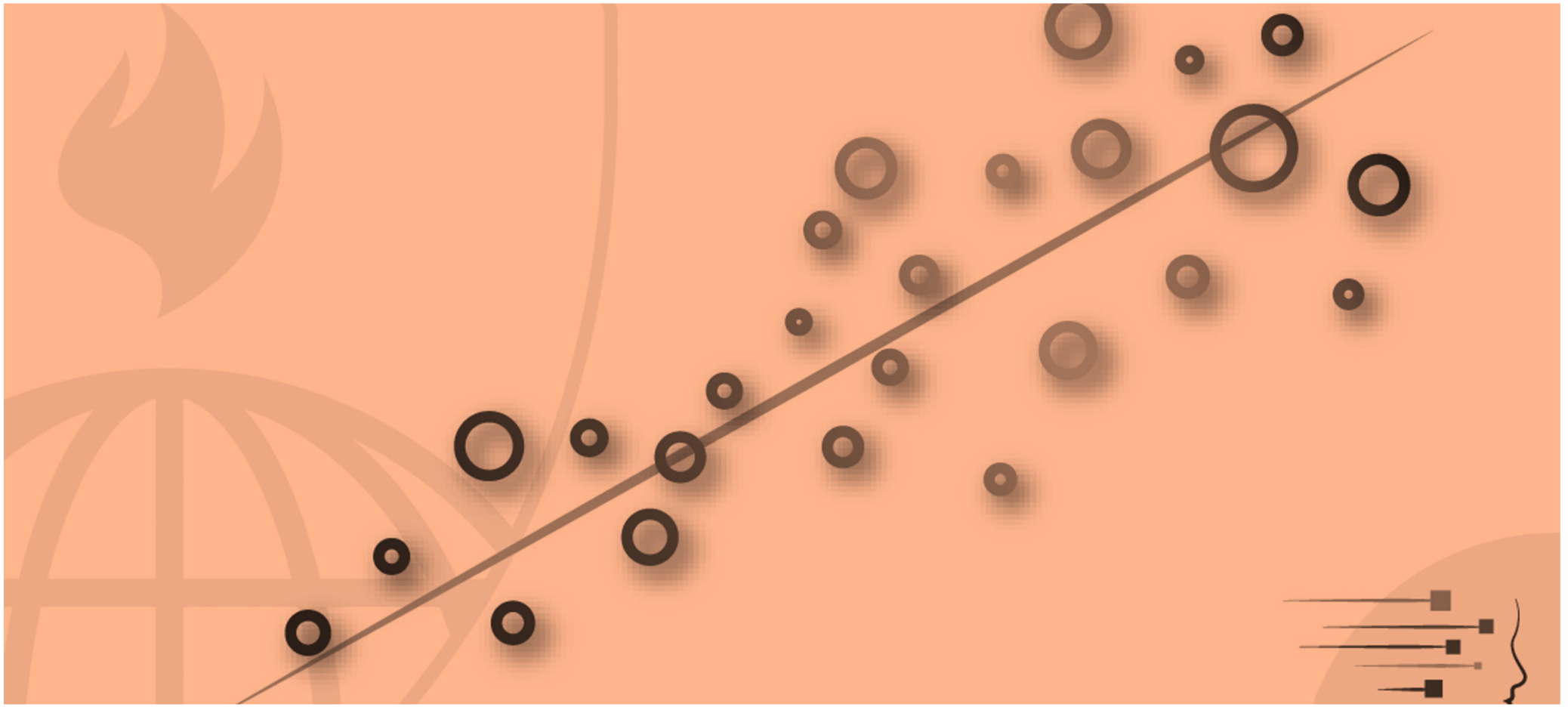


- Given some data points, the goal is to learn a **mathematical model** capable to fit data with a **curve**, so that the curve passes as close as possible to all of the data points
- This approach allows to **predict** the output for points that were not part of the original data



- What is the curve that best fits the data?
  - **Linear regression:** assumes that the relationship between variables is approximately linear
  - **Polynomial regression:** assumes that the relationship between variables is expressed by a polynomial of some degree
- Since we have a function model, its parameters that are estimated from the data

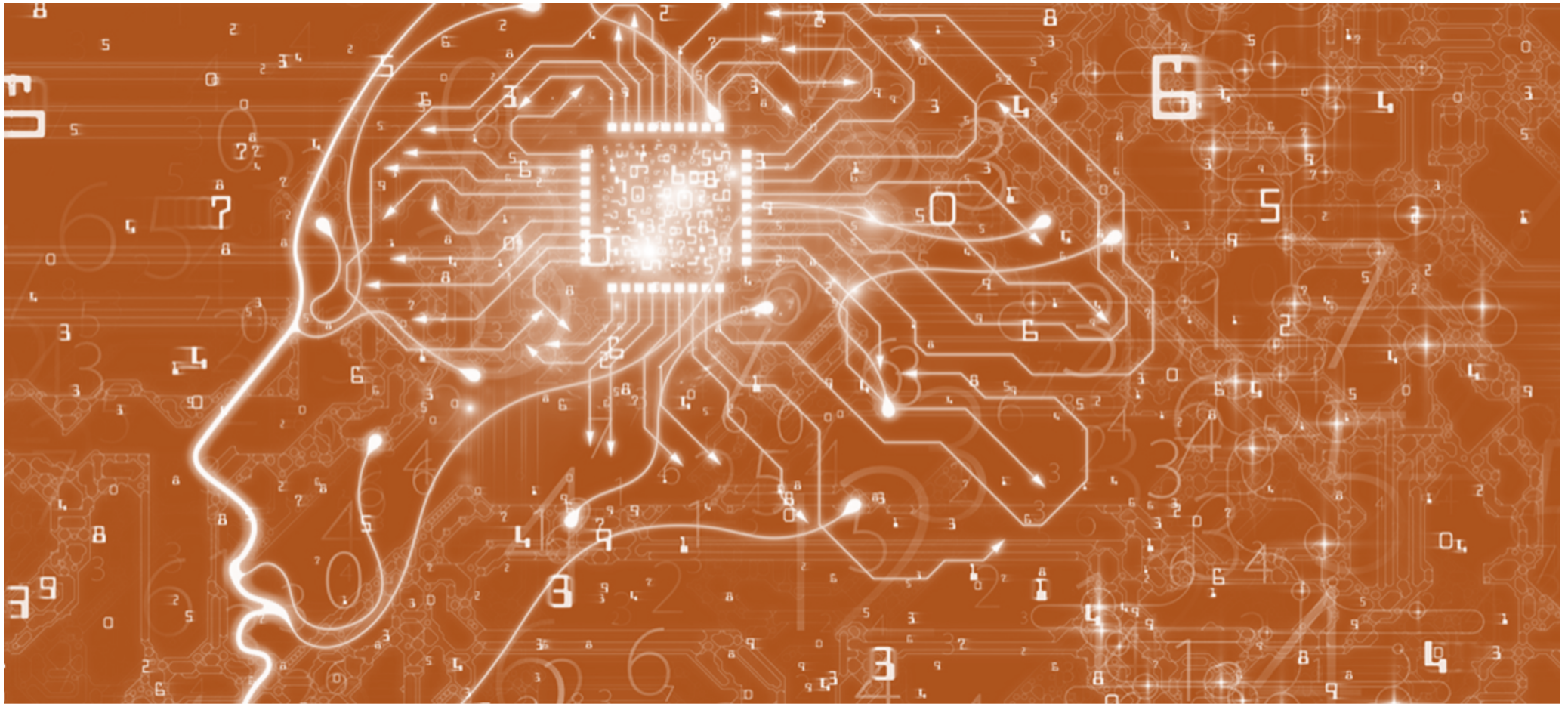




Linear regression

Summary

- Introduction to regression problem
- Linear Regression with One Variable
  - Model Representation
  - Cost Function
  - Gradient Descent
  - Learning Rate
- Linear Regression with Multiple Variables
  - Gradient Descent for Multiple Variables

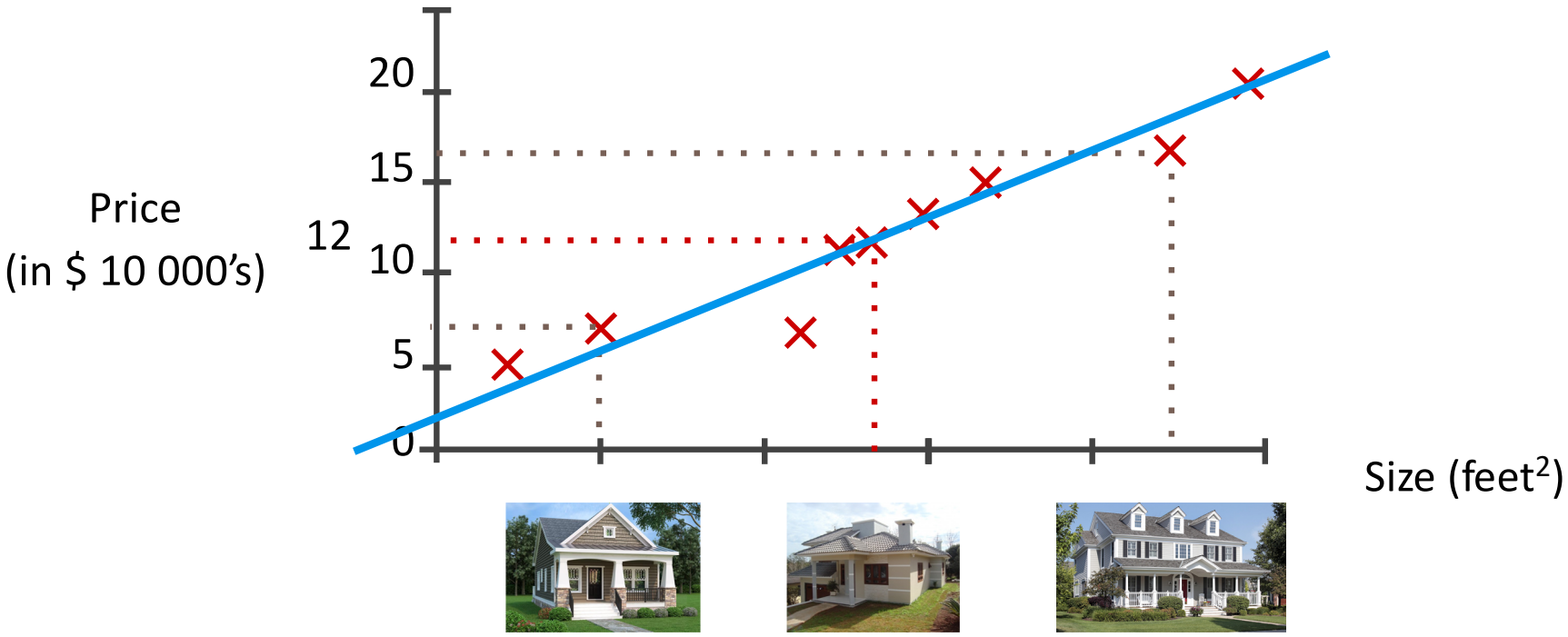


Linear regression with one variable

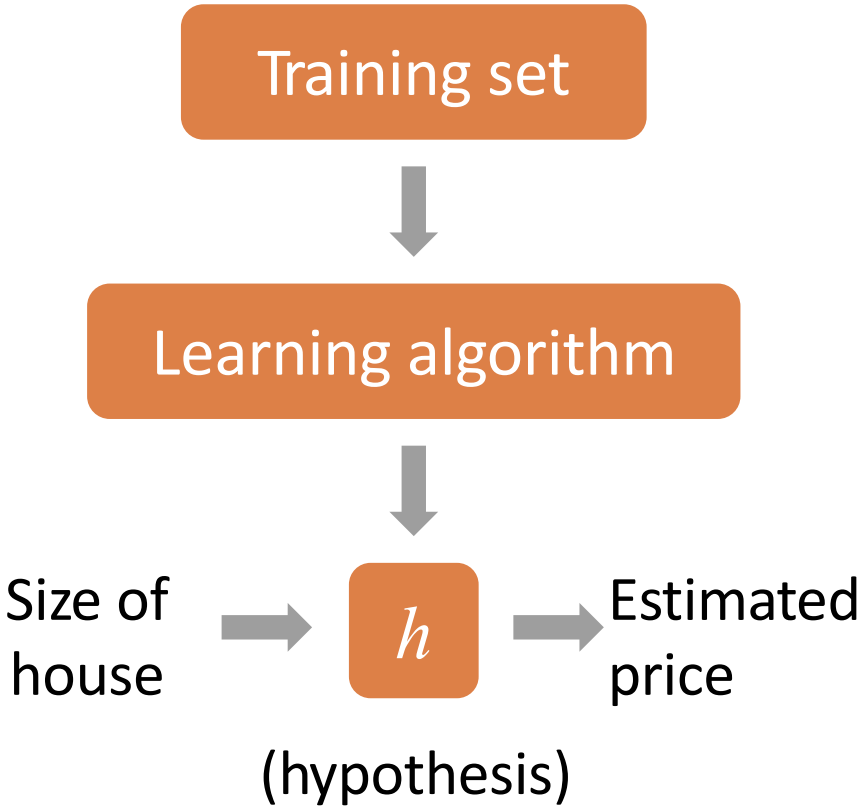
- **Regression**: Given some data points, the goal is to learn a mathematical model capable to fit the data with a curve, so that the curve passes as close as possible to all of the data points

- In **linear regression**, the relationships are modeled using linear predictor functions with unknown model parameters that are estimated from the data

House price prediction



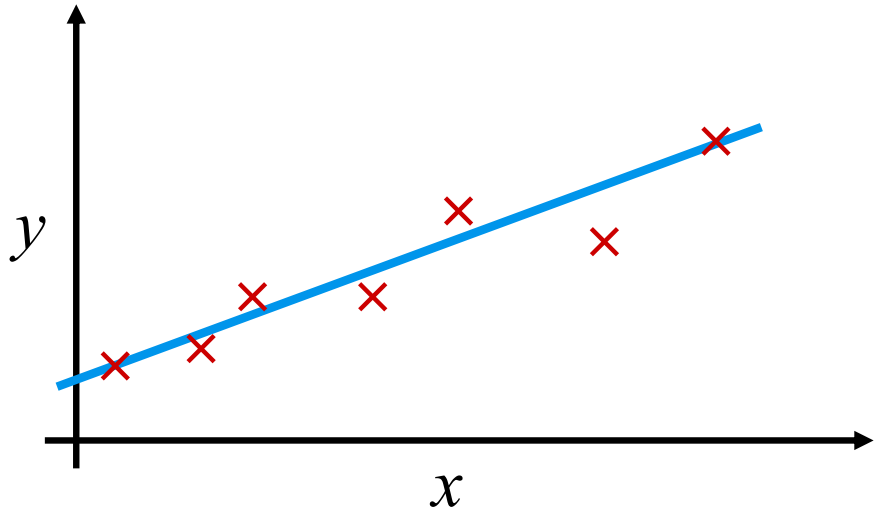
What is Regression?



h maps x's to y's

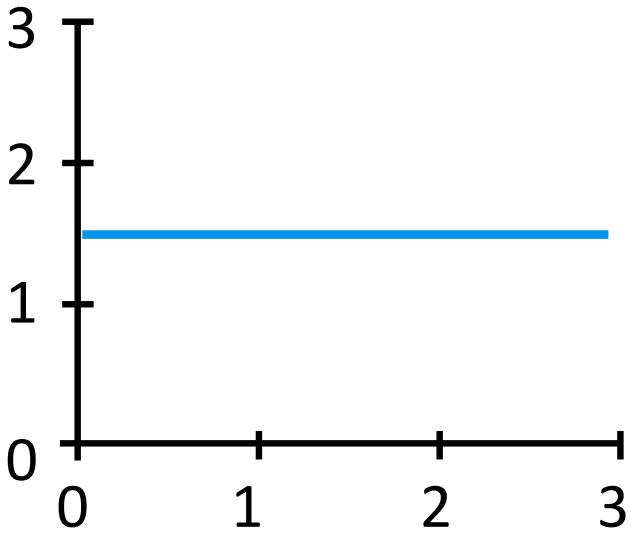
How do we represent h ?

$$h_\theta(x) = \theta_o + \theta_1 . x$$

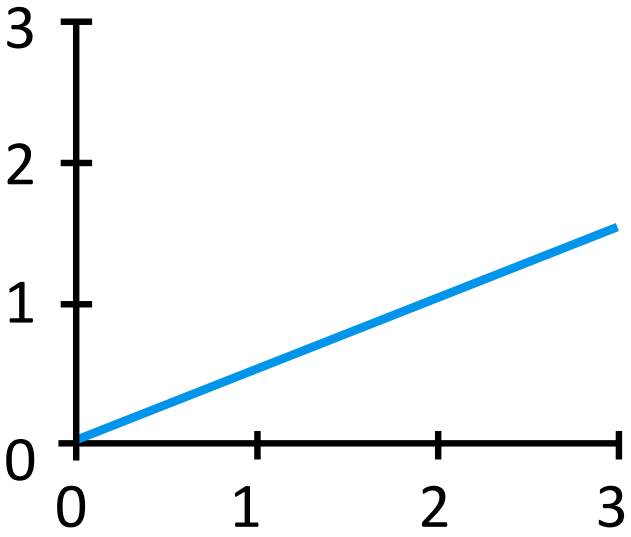


Linear regression with one variable.
Univariate linear regression.

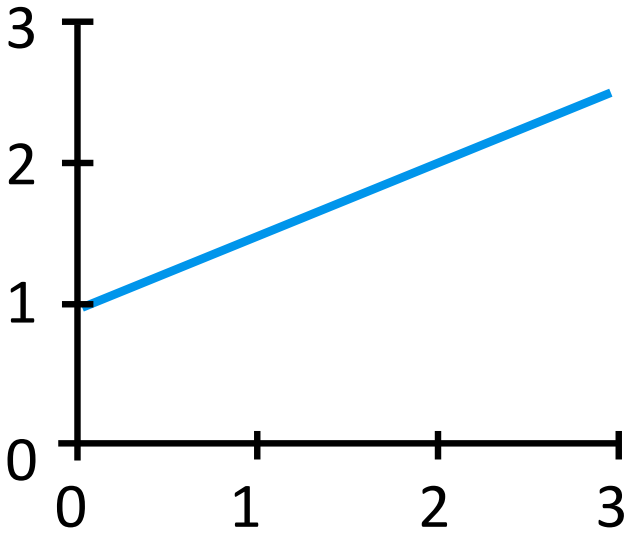
Parameters tuning



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

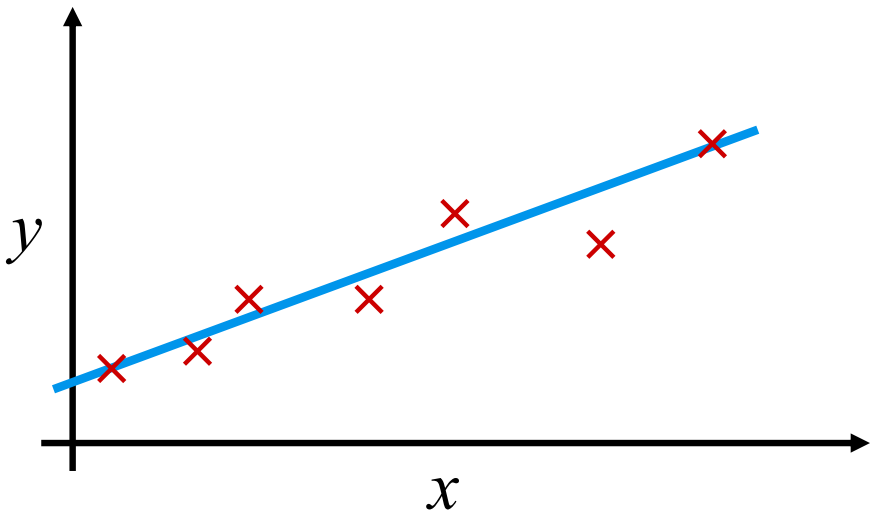


$\theta_0 = 0$
$\theta_1 = 0.5$



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

Cost function (J)



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

$$h_\theta(x) = \theta_o + \theta_1 . x$$

Idea: Choose θ_0, θ_1 so that $h_\theta(x)$ is close to y for our training examples (x,y)

$$\underset{\theta_o, \theta_1}{Minimize}\ J(\theta_o, \theta_1)$$

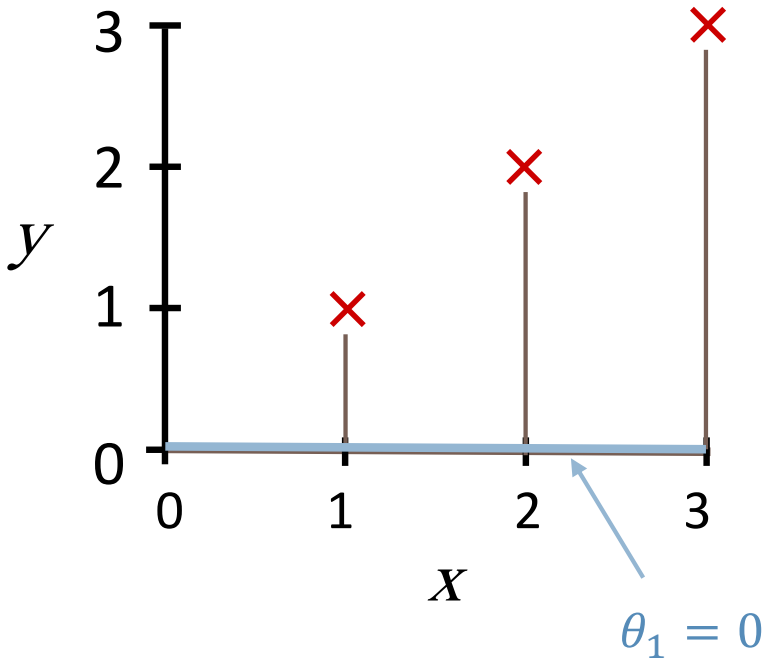
Cost function
(Squared error function)

Minimizing the the least squared loss is the same thing as minimizing the variance!

Cost function

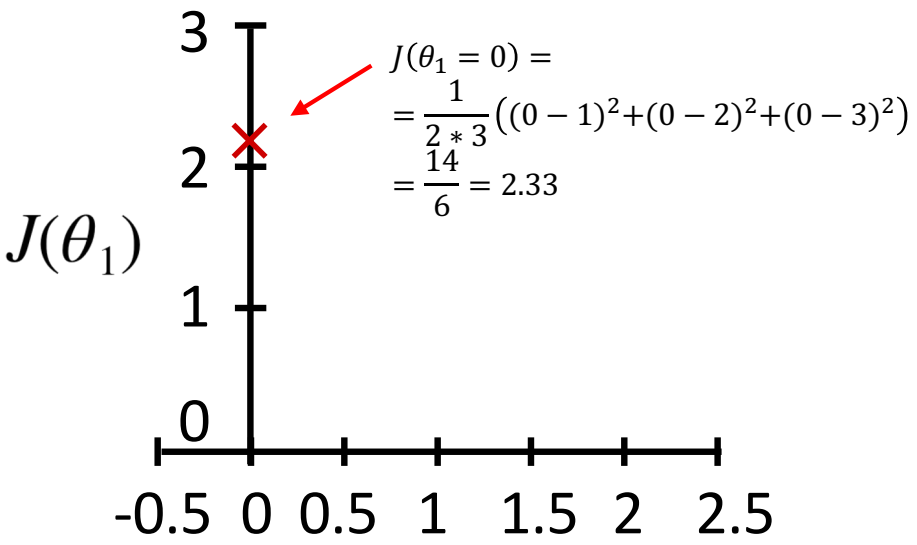
Hypotesis:

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



Cost function:

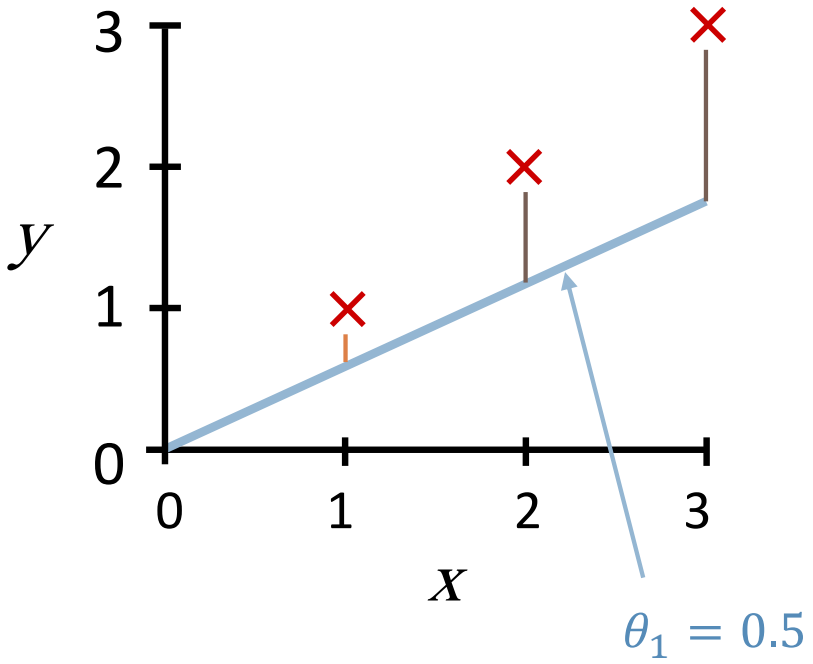
$$J(\theta_1)$$



Cost function

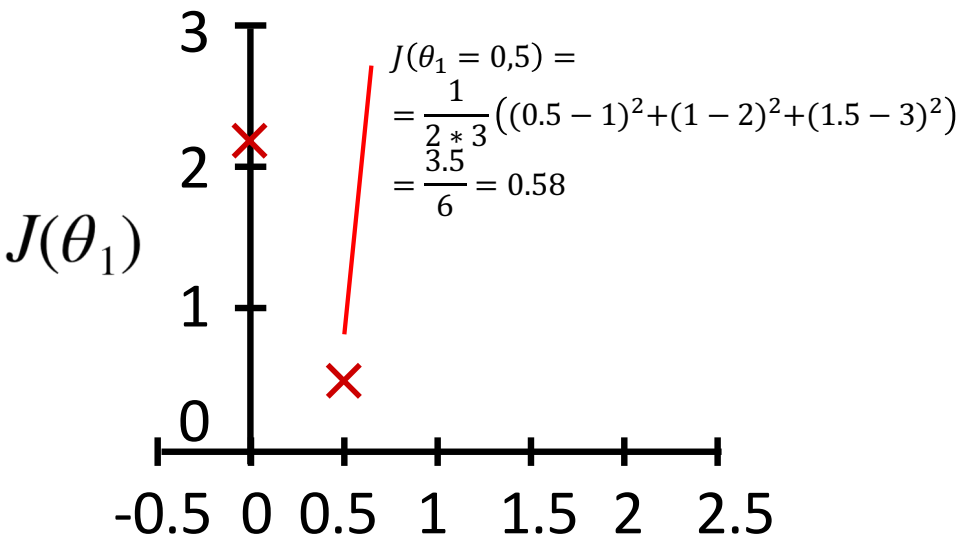
Hypotesis:

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



Cost function:

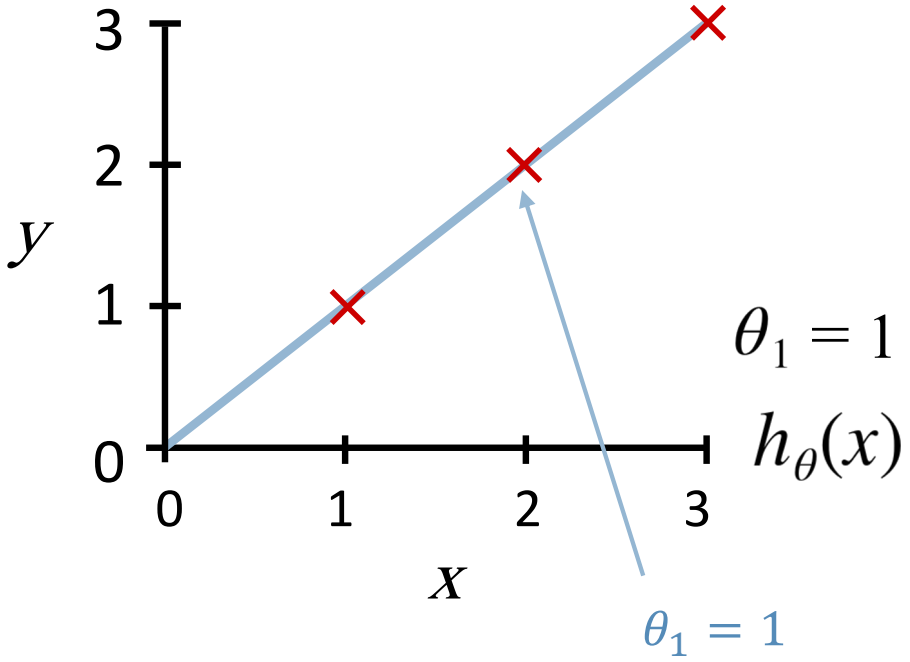
$$J(\theta_1)$$



Cost function

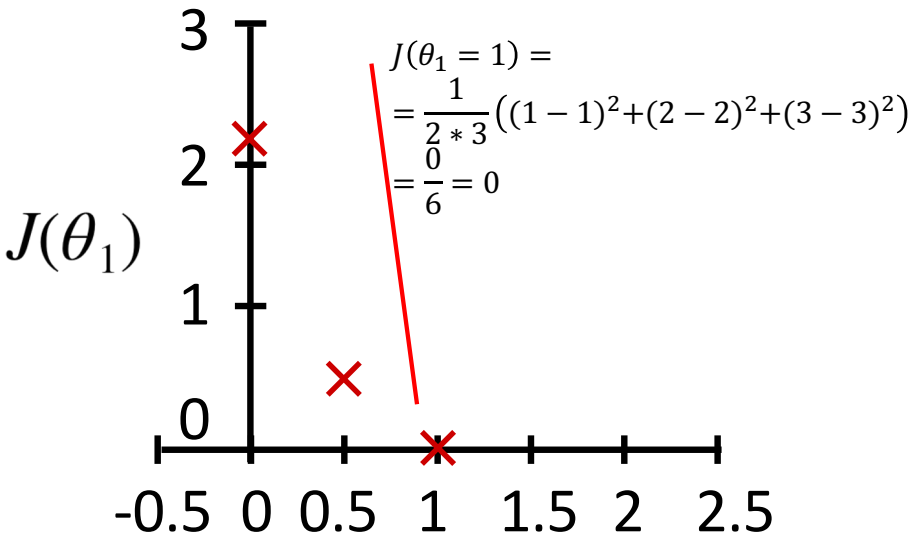
Hypotesis:

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



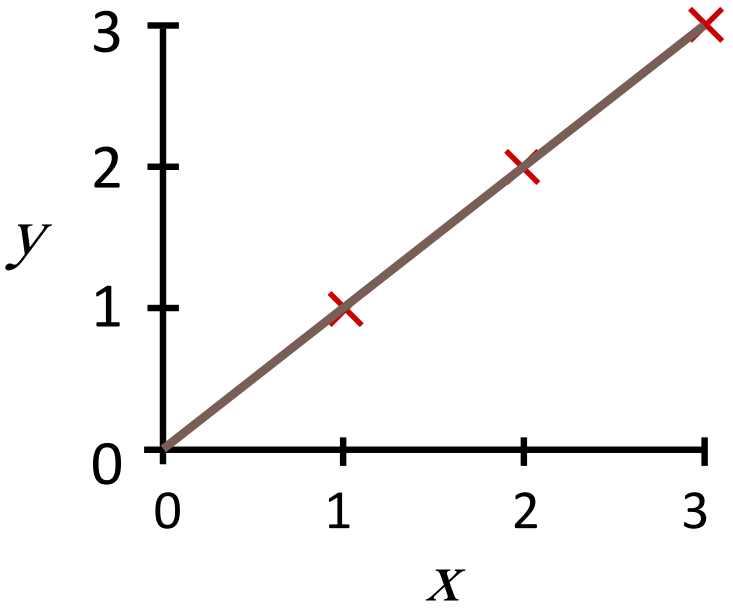
Cost function:

$J(\theta_1)$



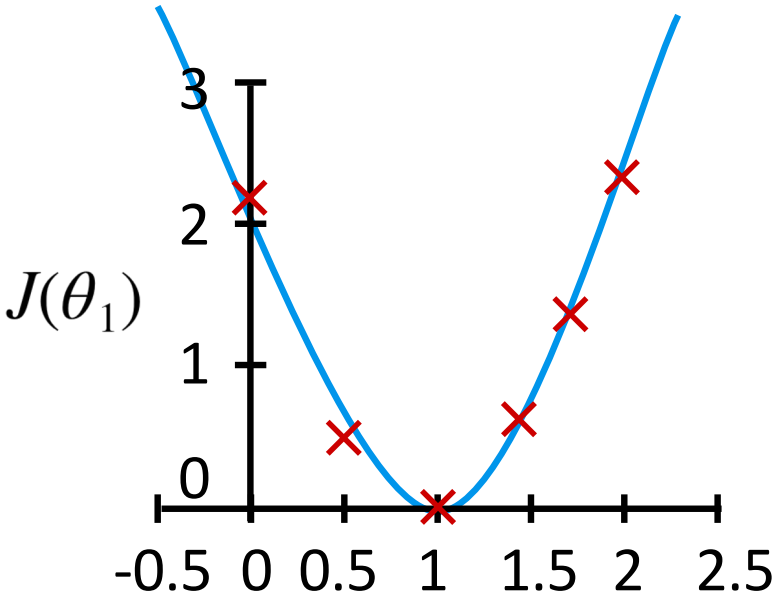
Hypotesis:

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



Cost function:

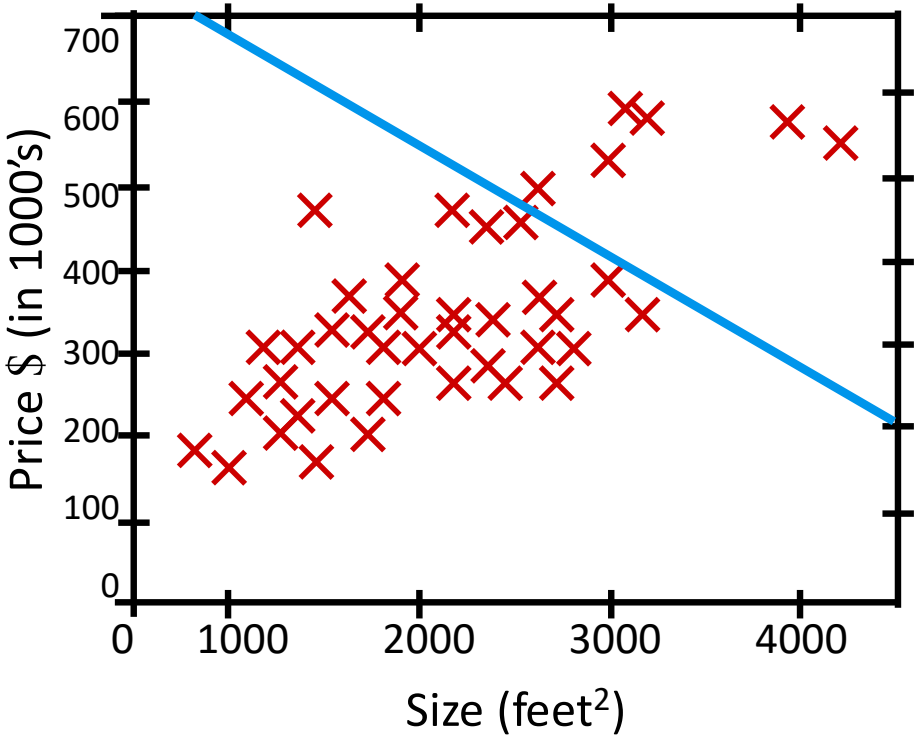
$J(\theta_1)$



Cost function

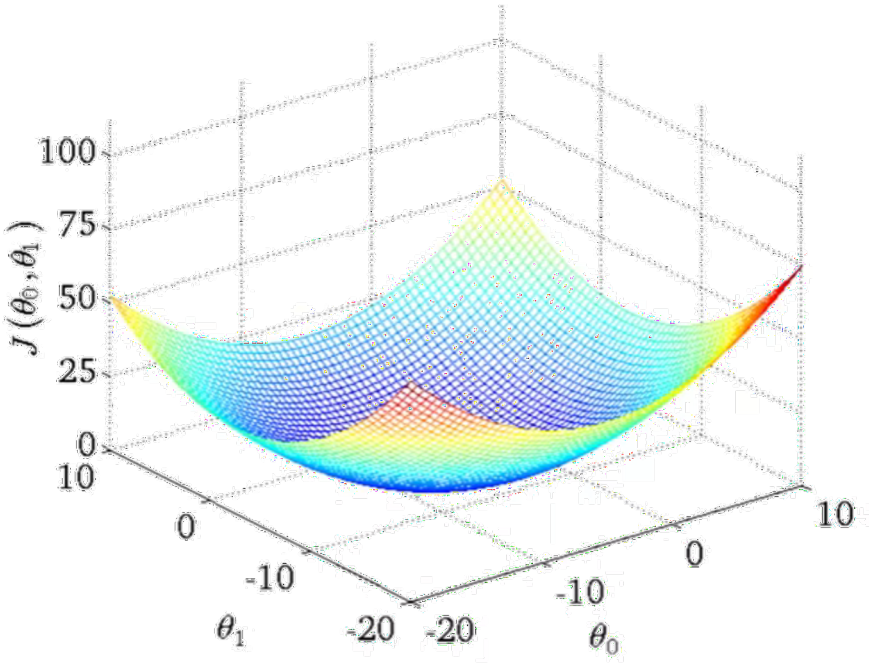
Hypotesis:

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



Cost function:

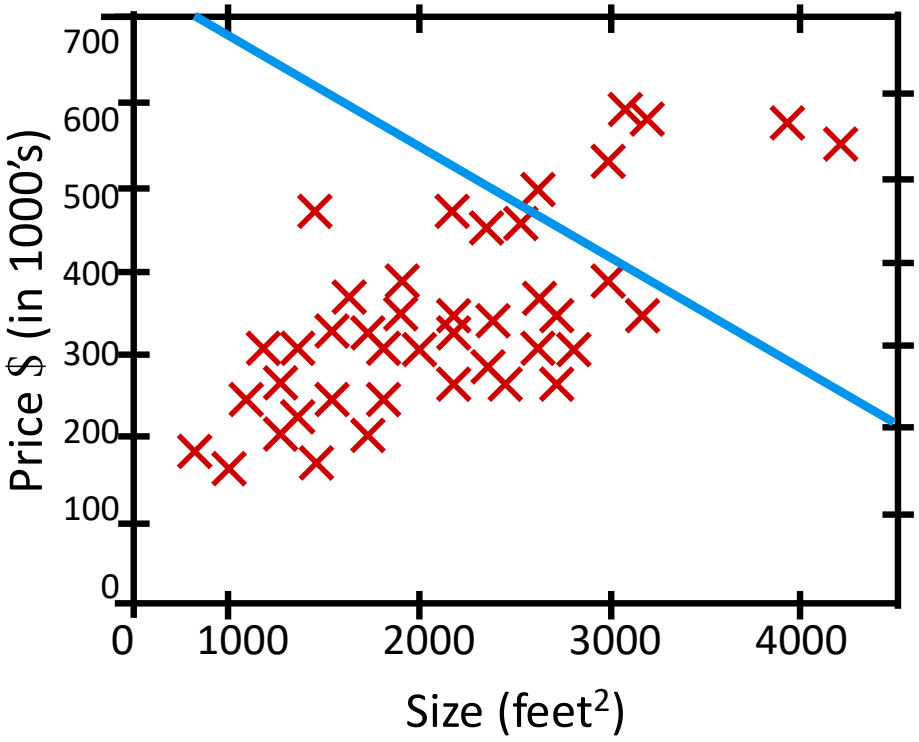
$$J(\theta_0, \theta_1)$$



Cost function

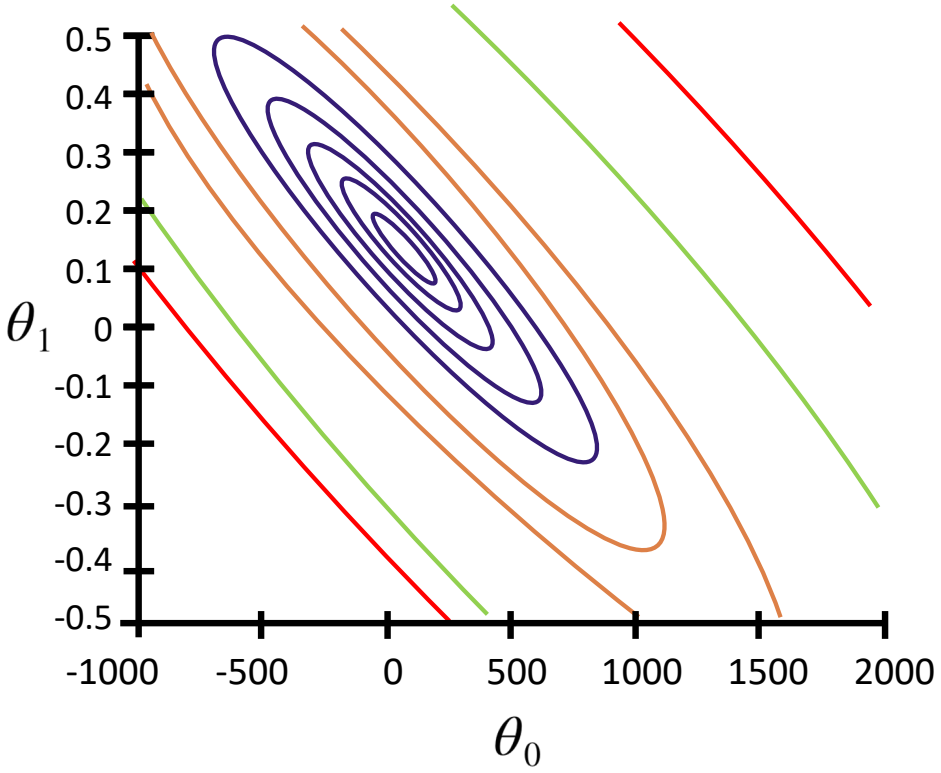
Hypotesis:

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



Cost function:

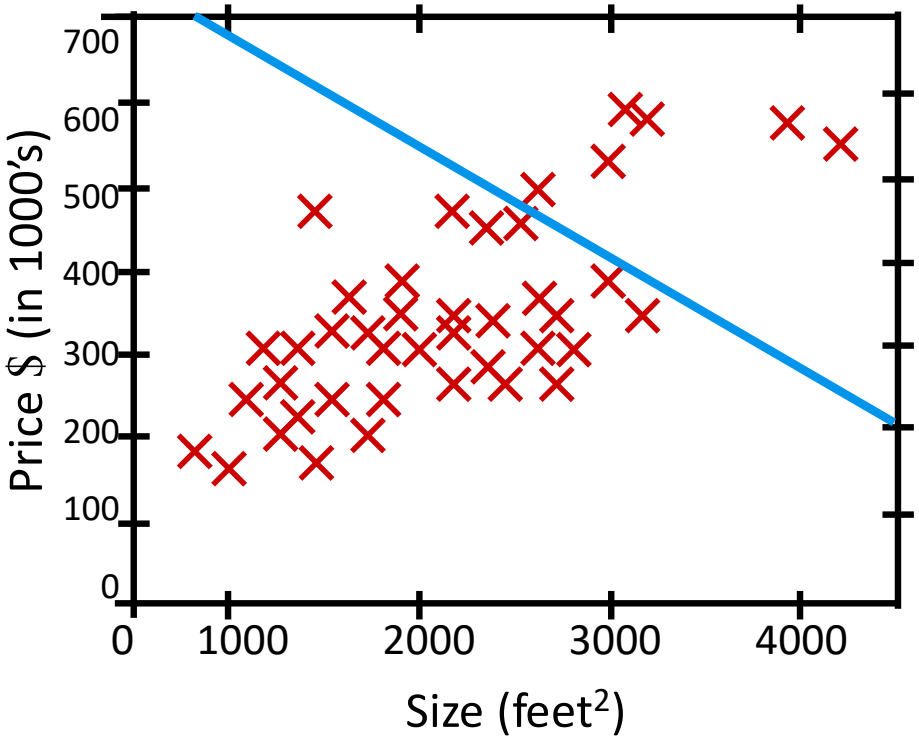
$$J(\theta_0, \theta_1)$$



Cost function

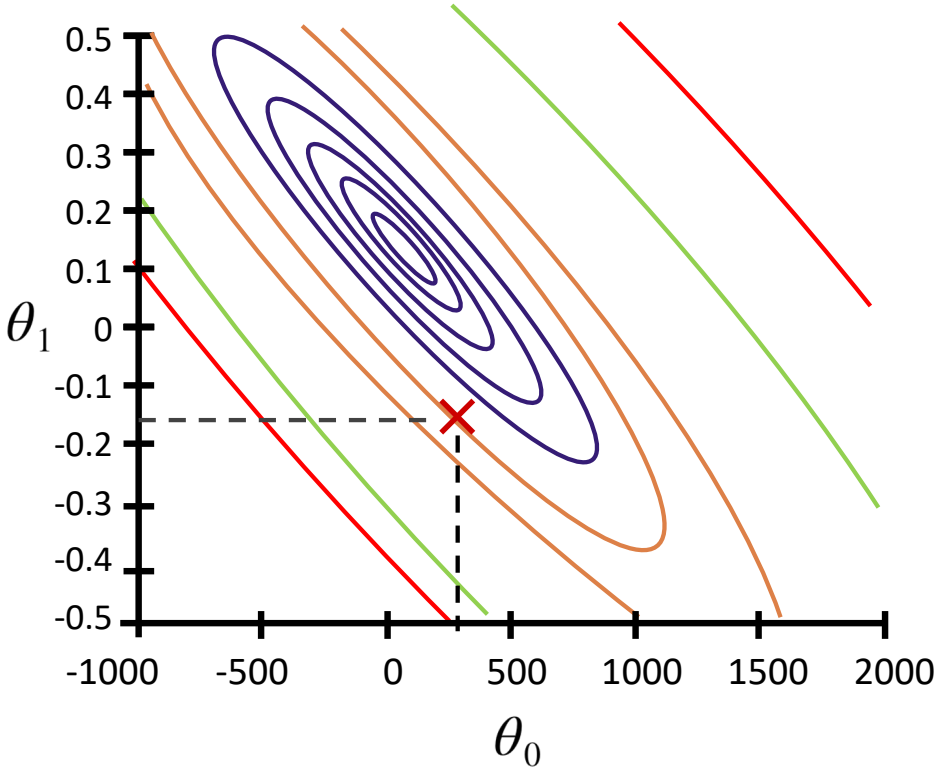
Hypotesis:

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



Cost function:

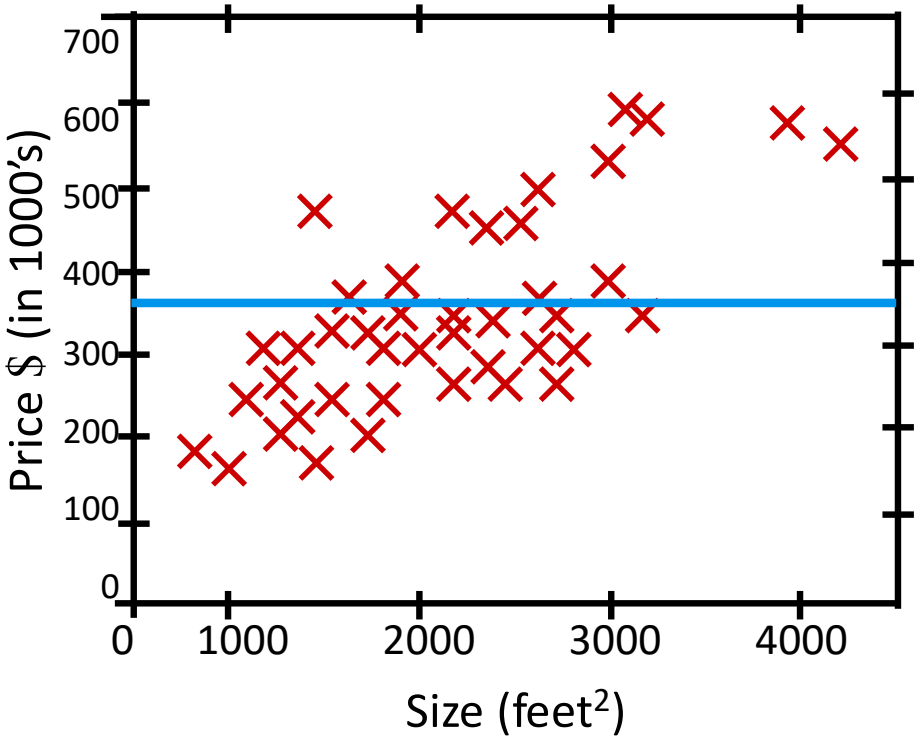
$$J(\theta_0, \theta_1)$$



Cost function

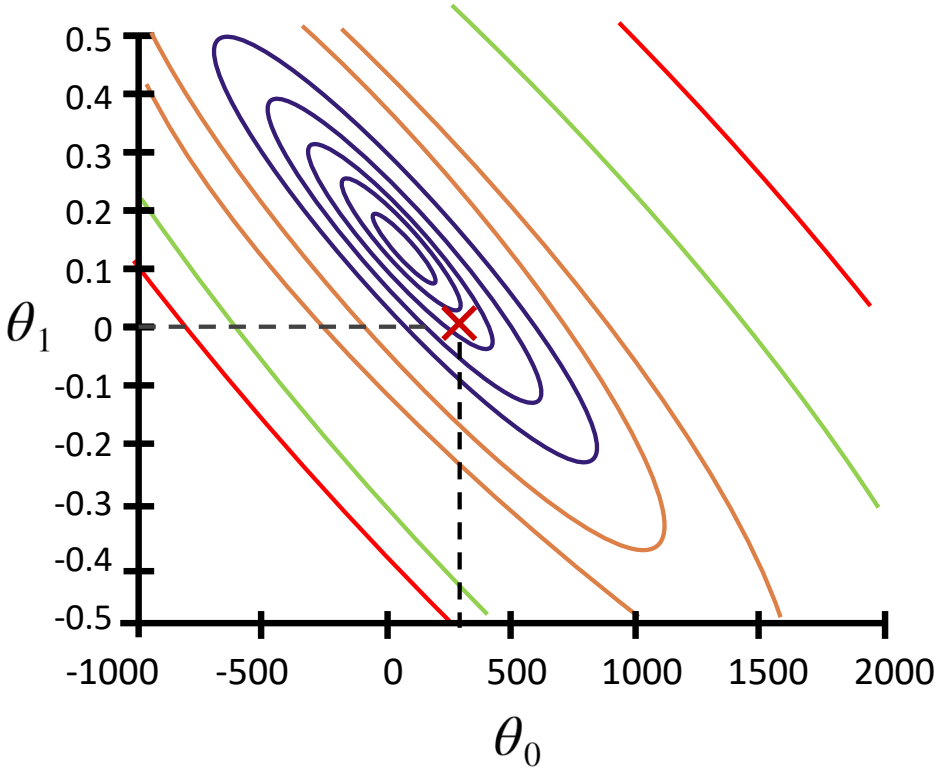
Hypotesis:

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



Cost function:

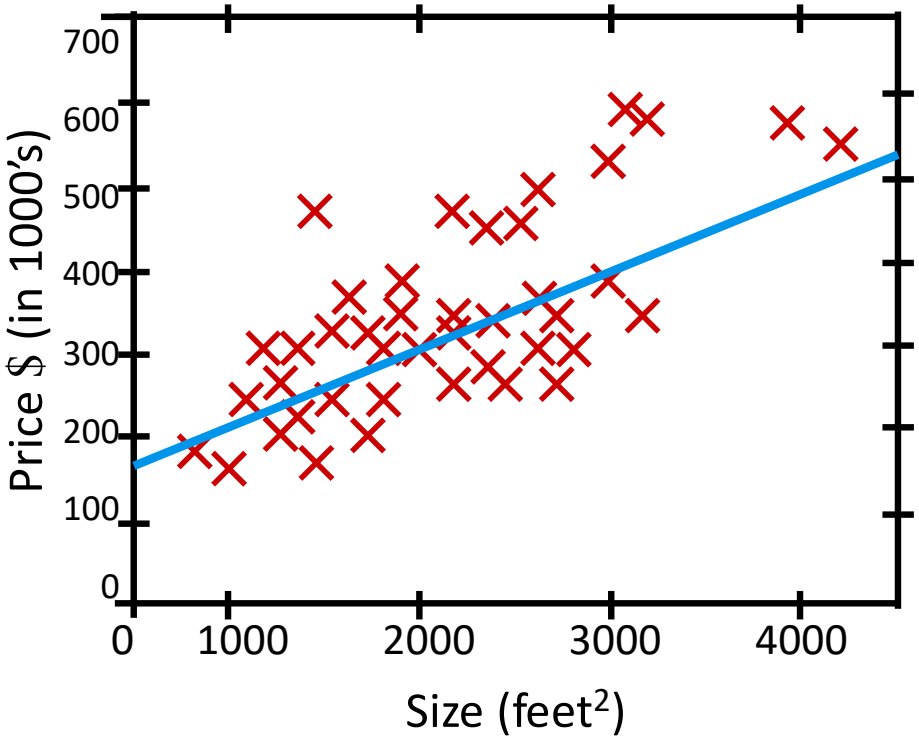
$$J(\theta_0, \theta_1)$$



Cost function

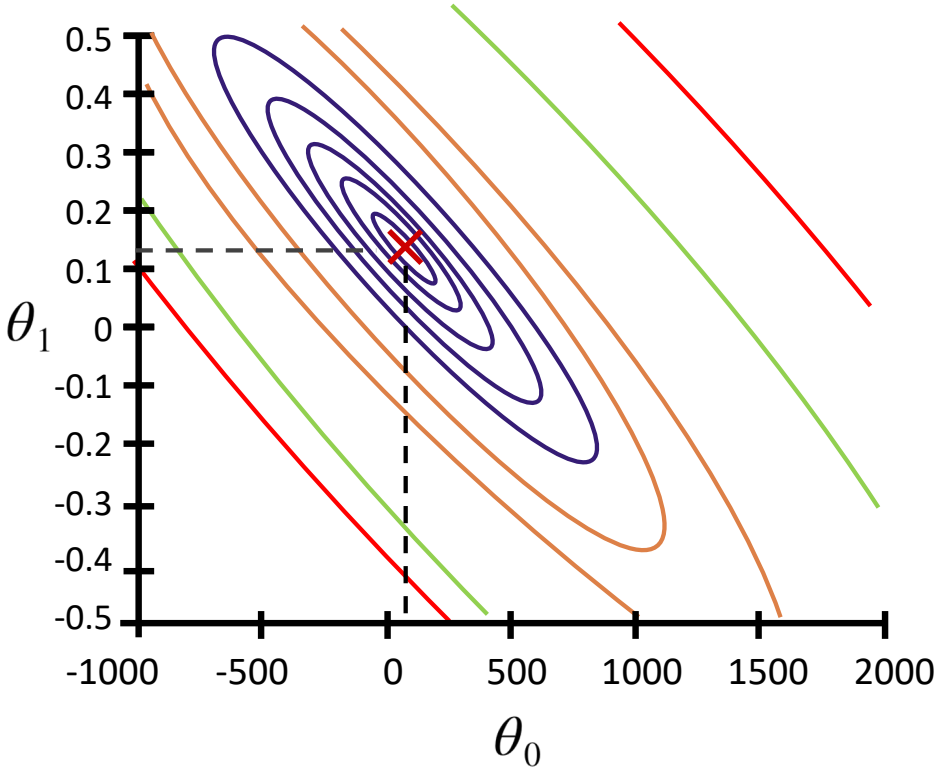
Hypotesis:

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



Cost function:

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



Gradient descent

Algorithm: Gadient Descent

1: Given some cost function $J(\theta_0, \theta_1, \ldots, \theta_n)$:

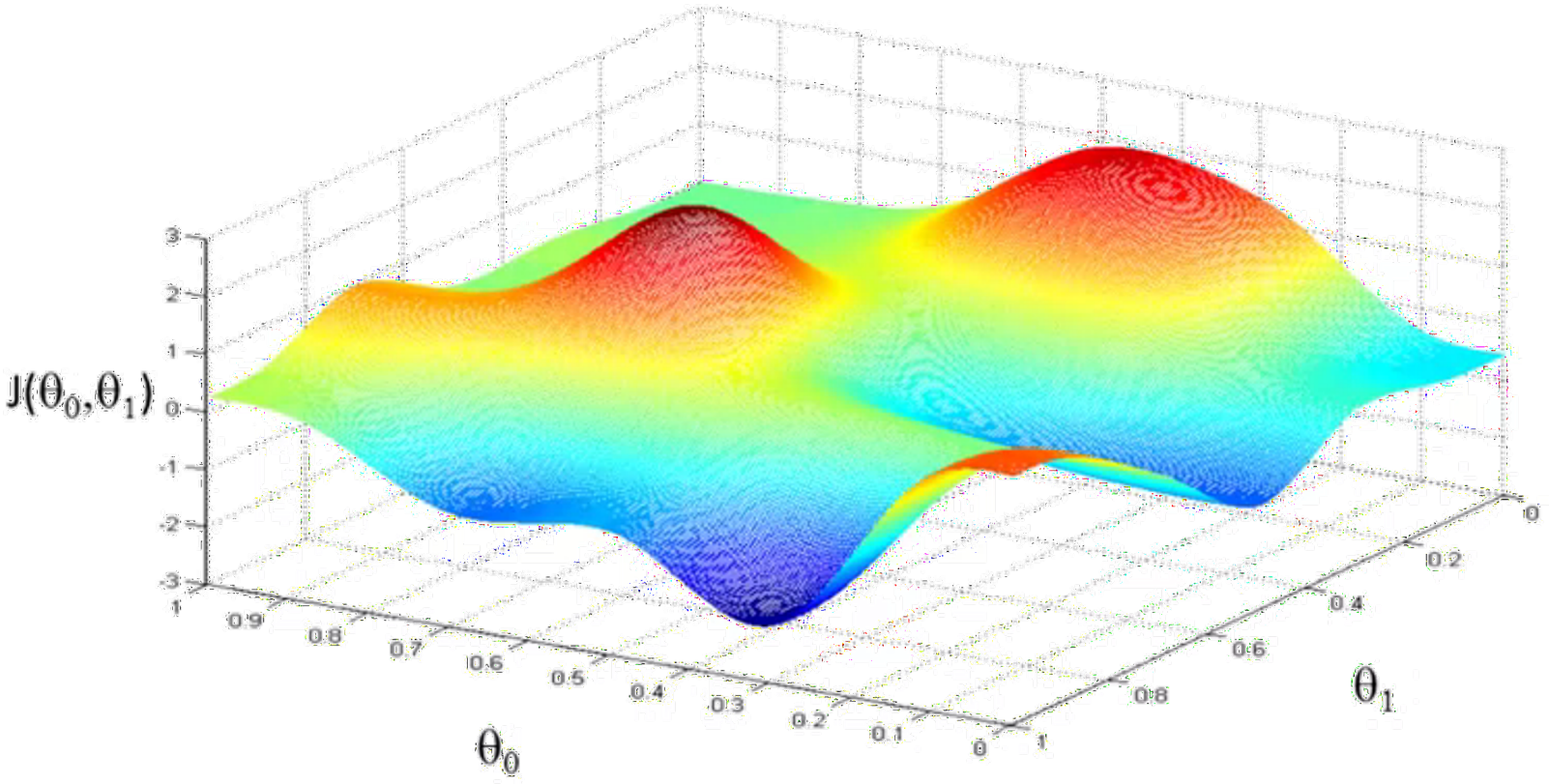
2: Algorithm parameter: learning rate $\alpha \in (0, 1]$

3: *Initialize* $\theta_0, \theta_1, \ldots, \theta_n$ arbitrarily

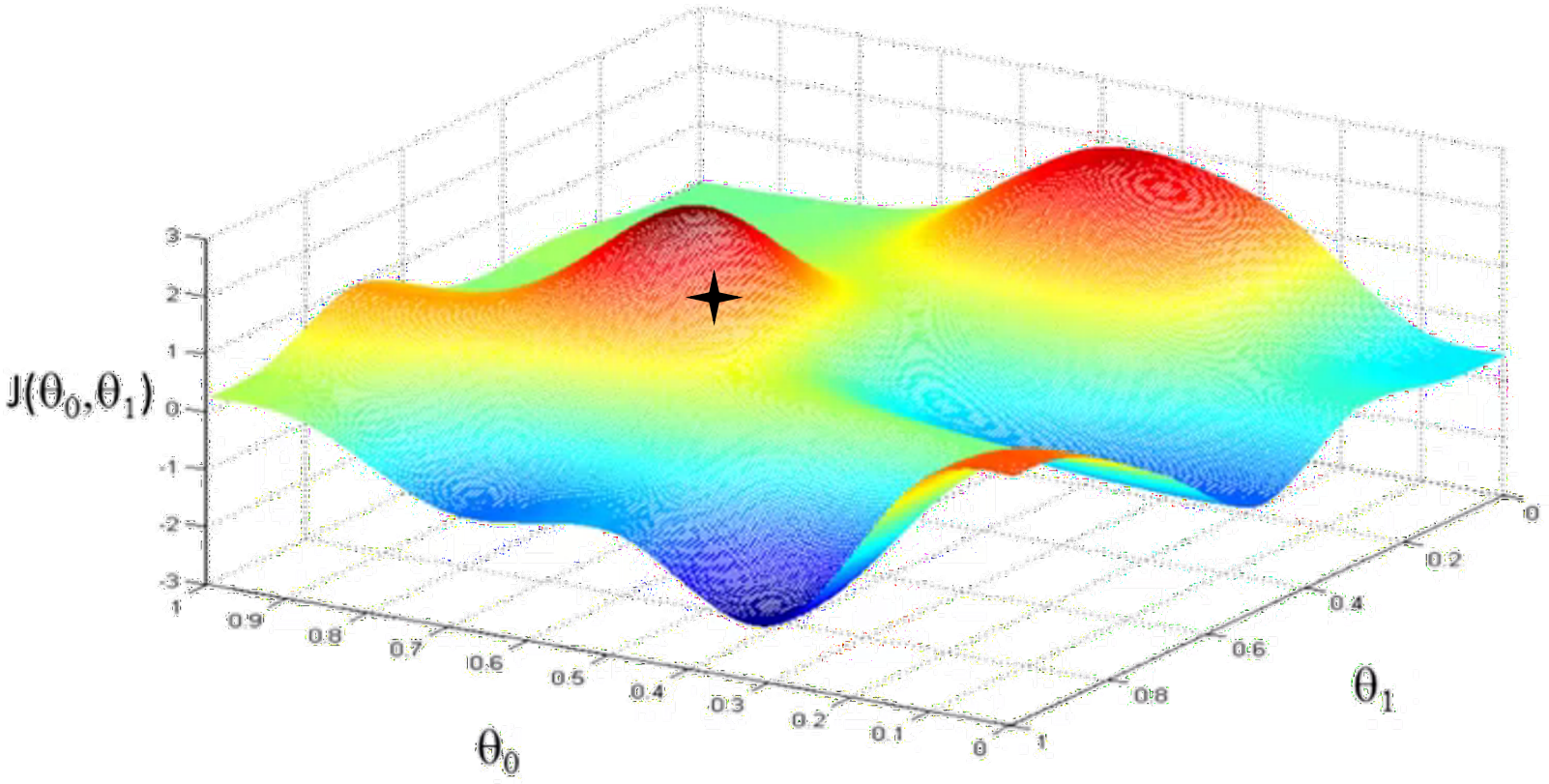
4: ***repeat (for each epoch)***

5: Change $\theta_0, \theta_1, \ldots, \theta_n$ to reduce $J(\theta_0, \theta_1, \ldots, \theta_n)$

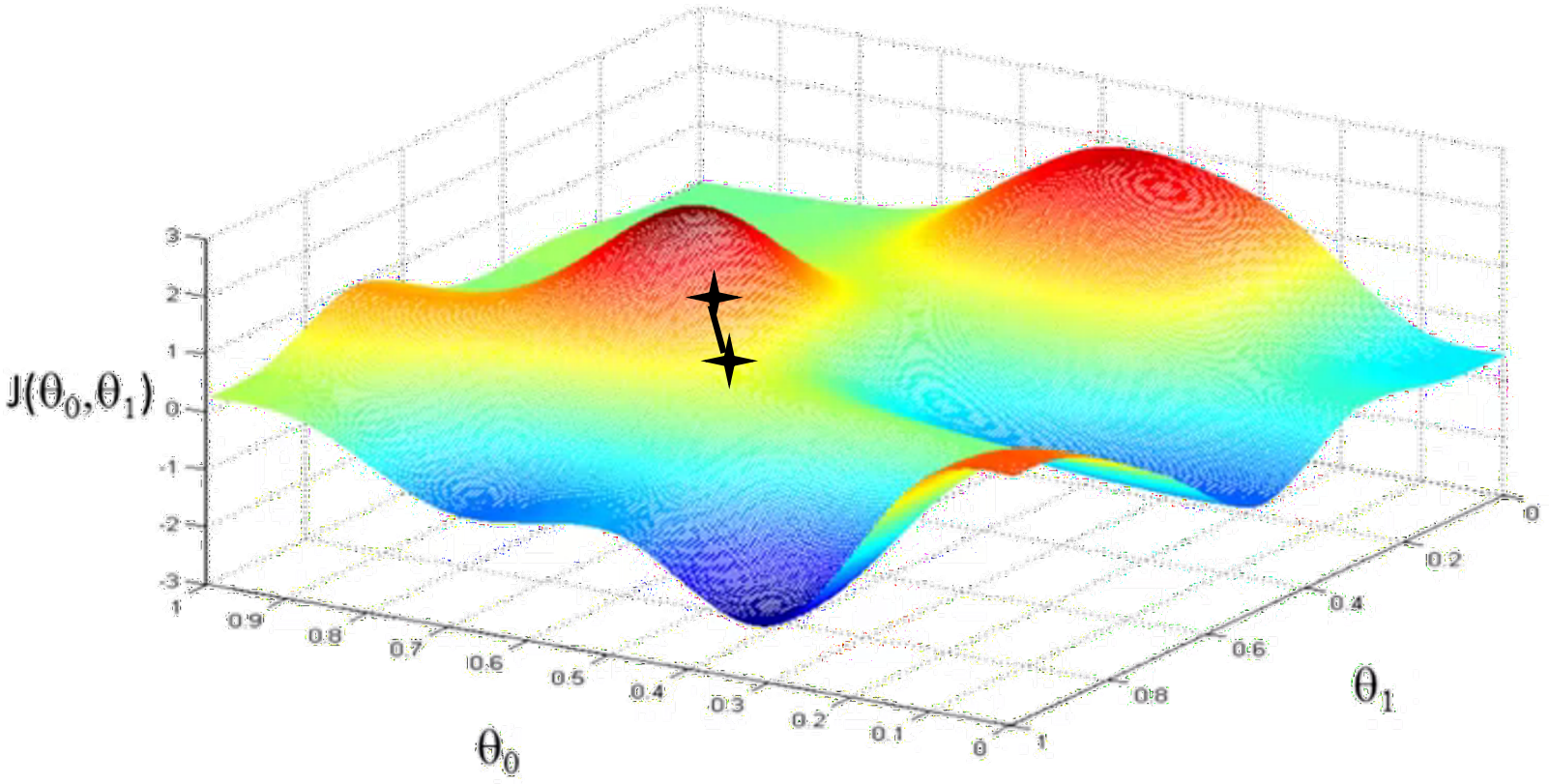
Gradient descent



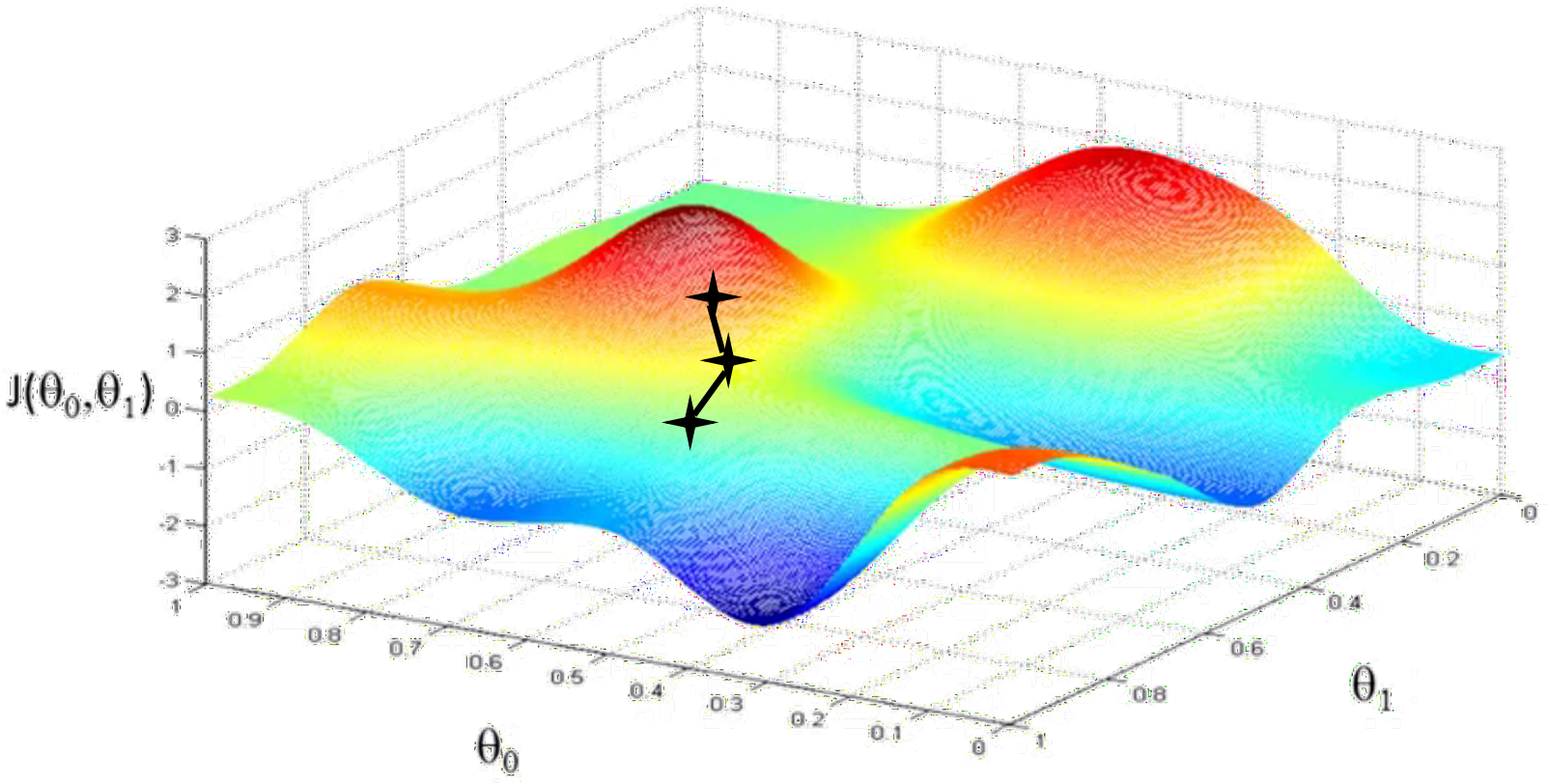
Gradient descent



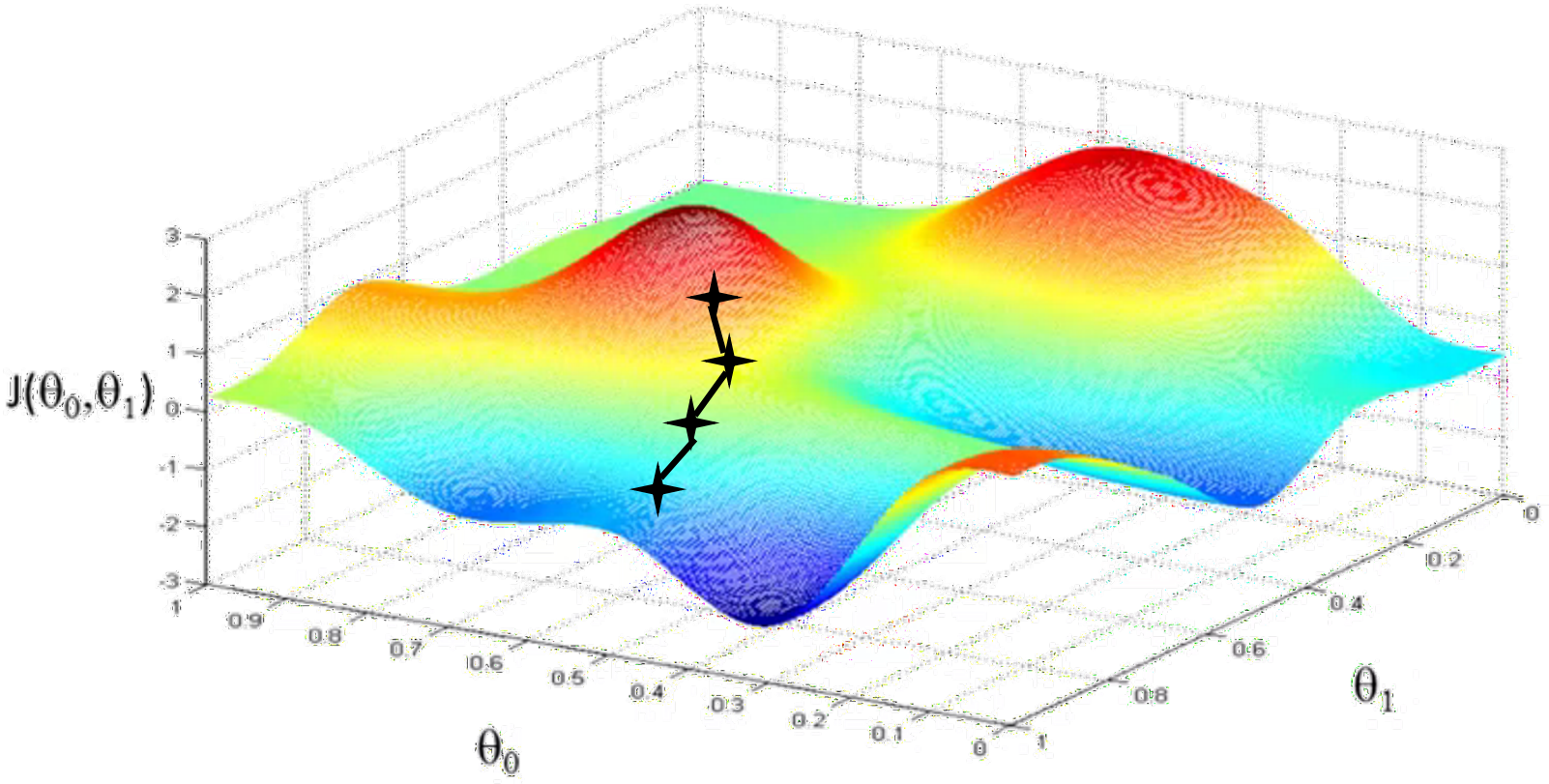
Gradient descent



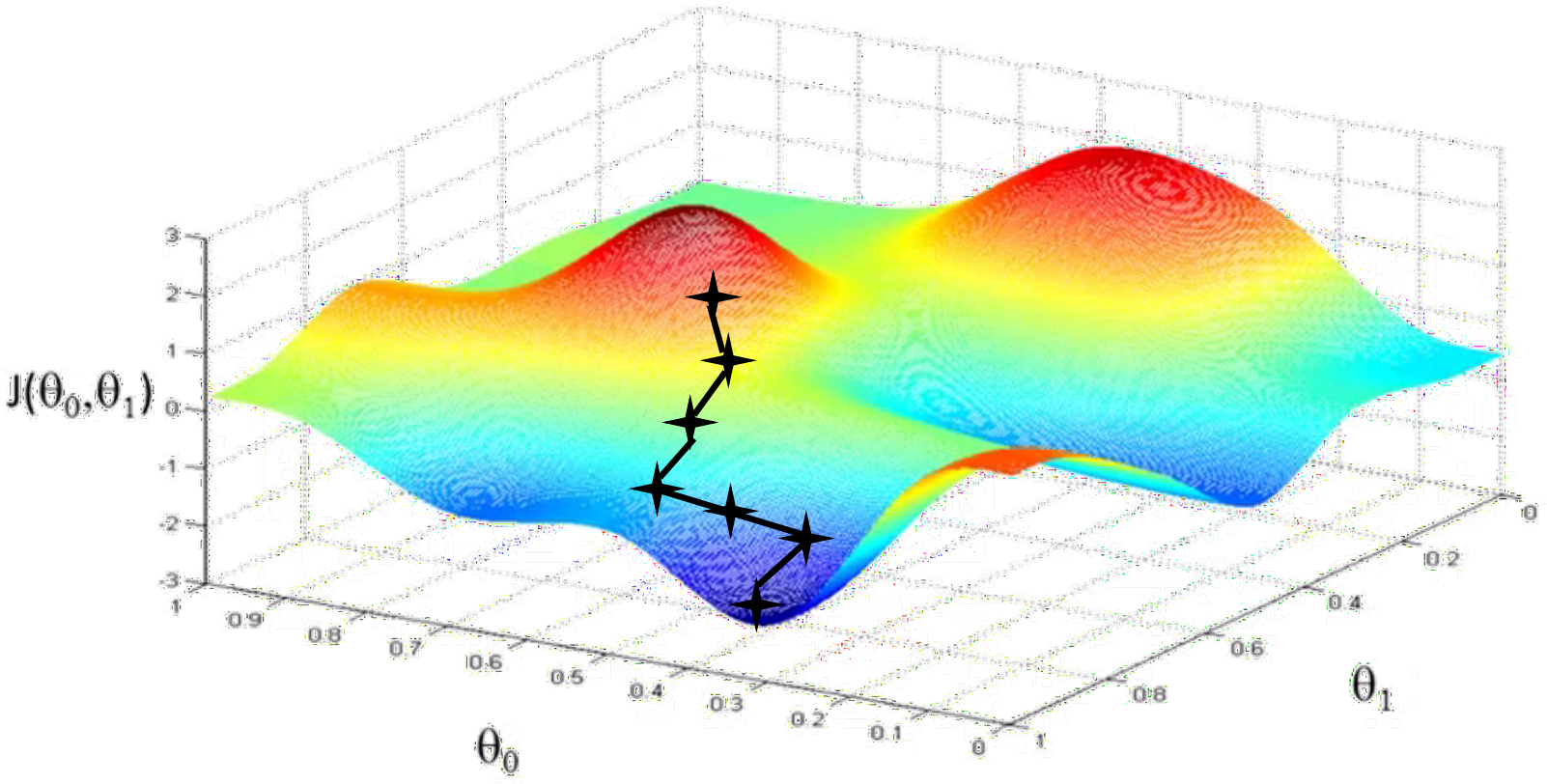
Gradient descent



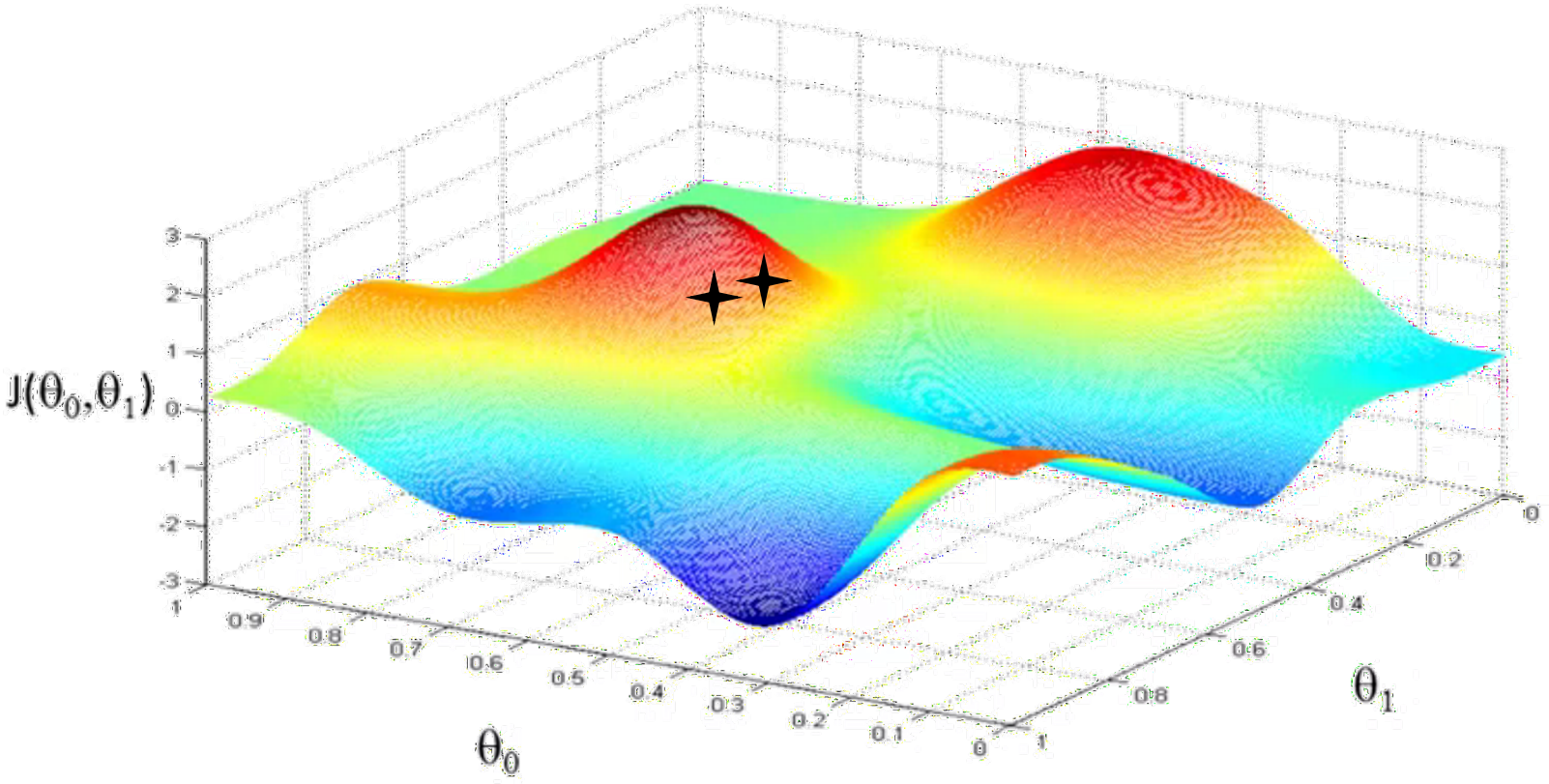
Gradient descent



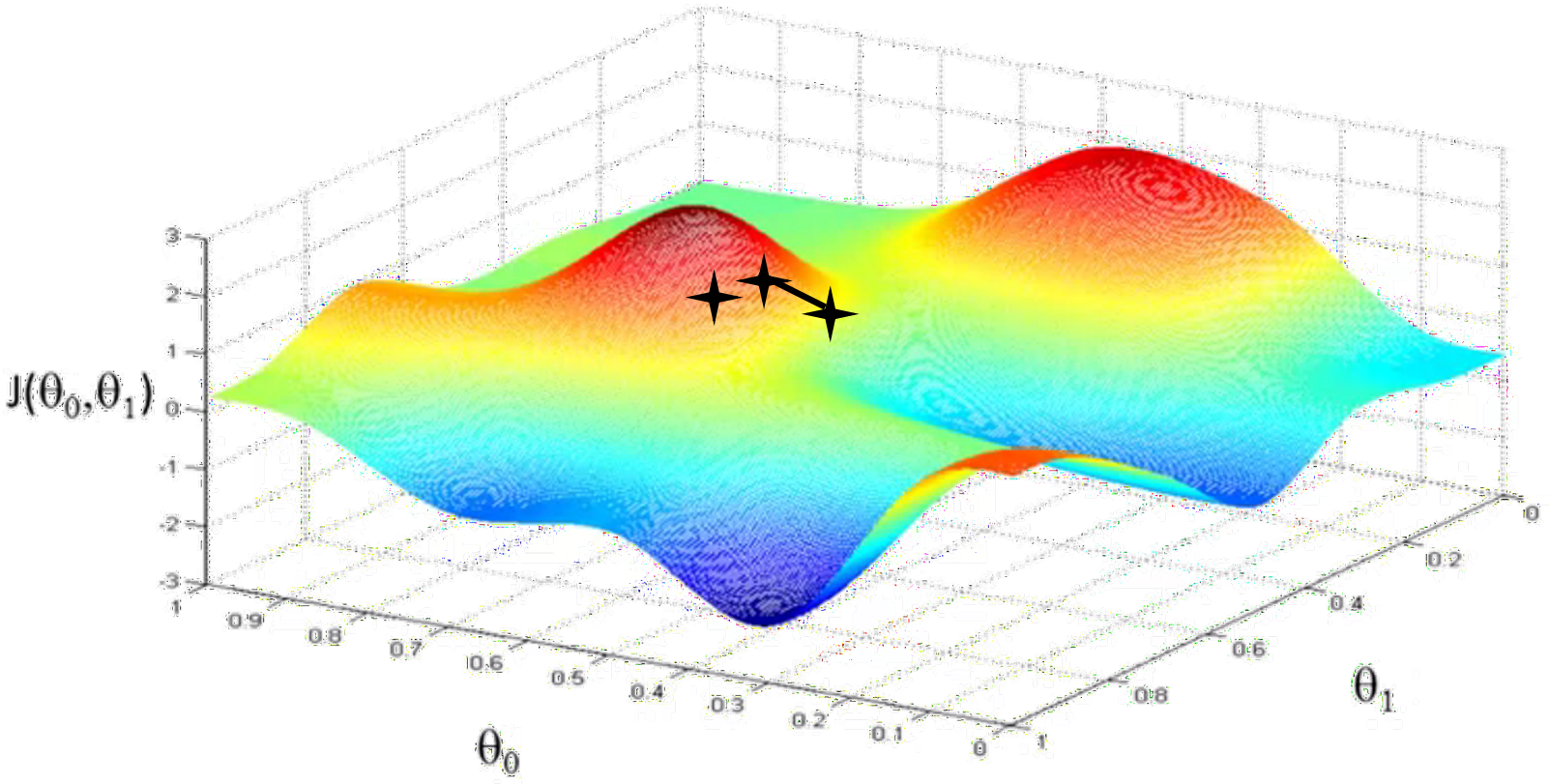
Gradient descent



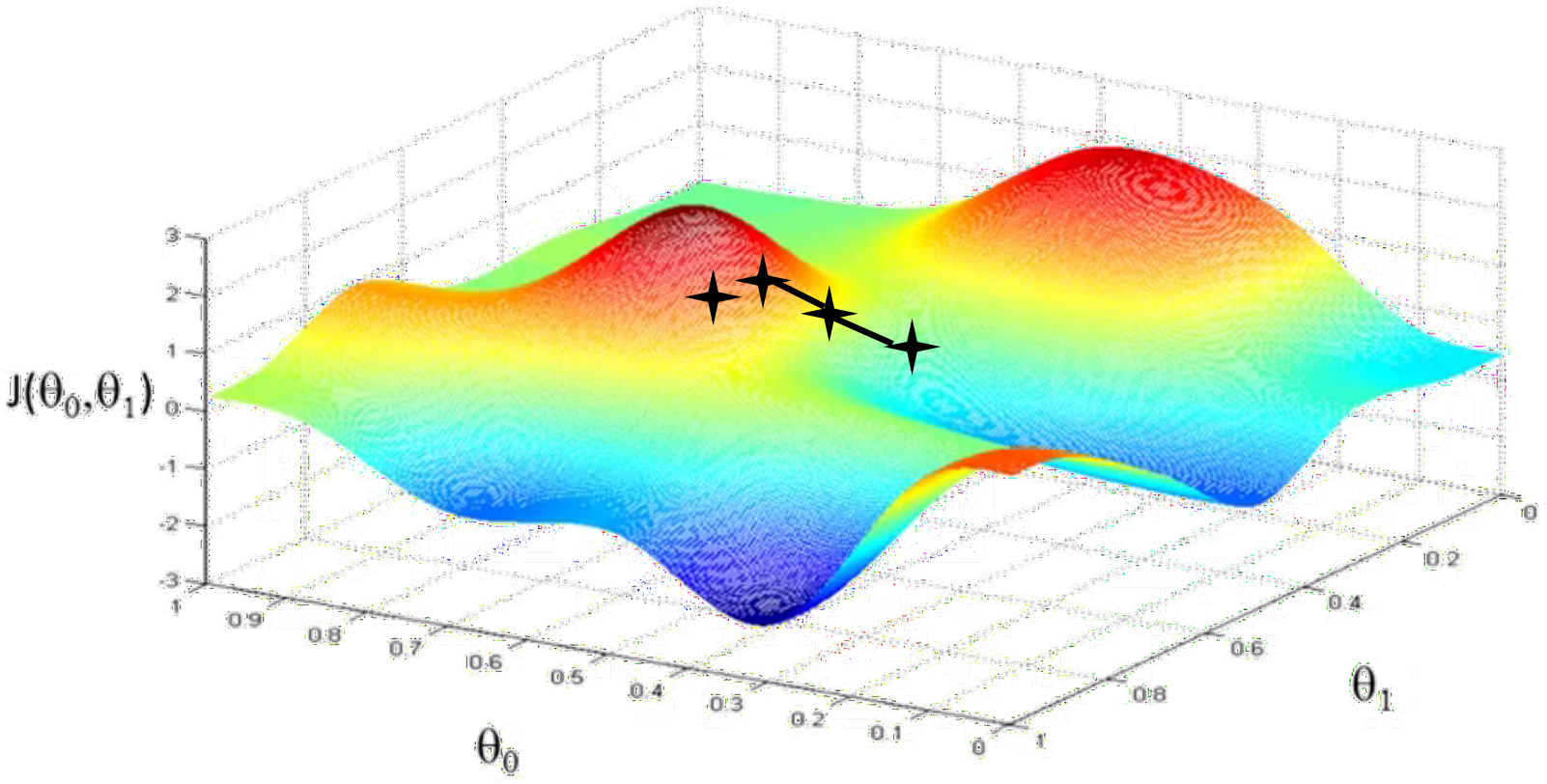
Gradient descent



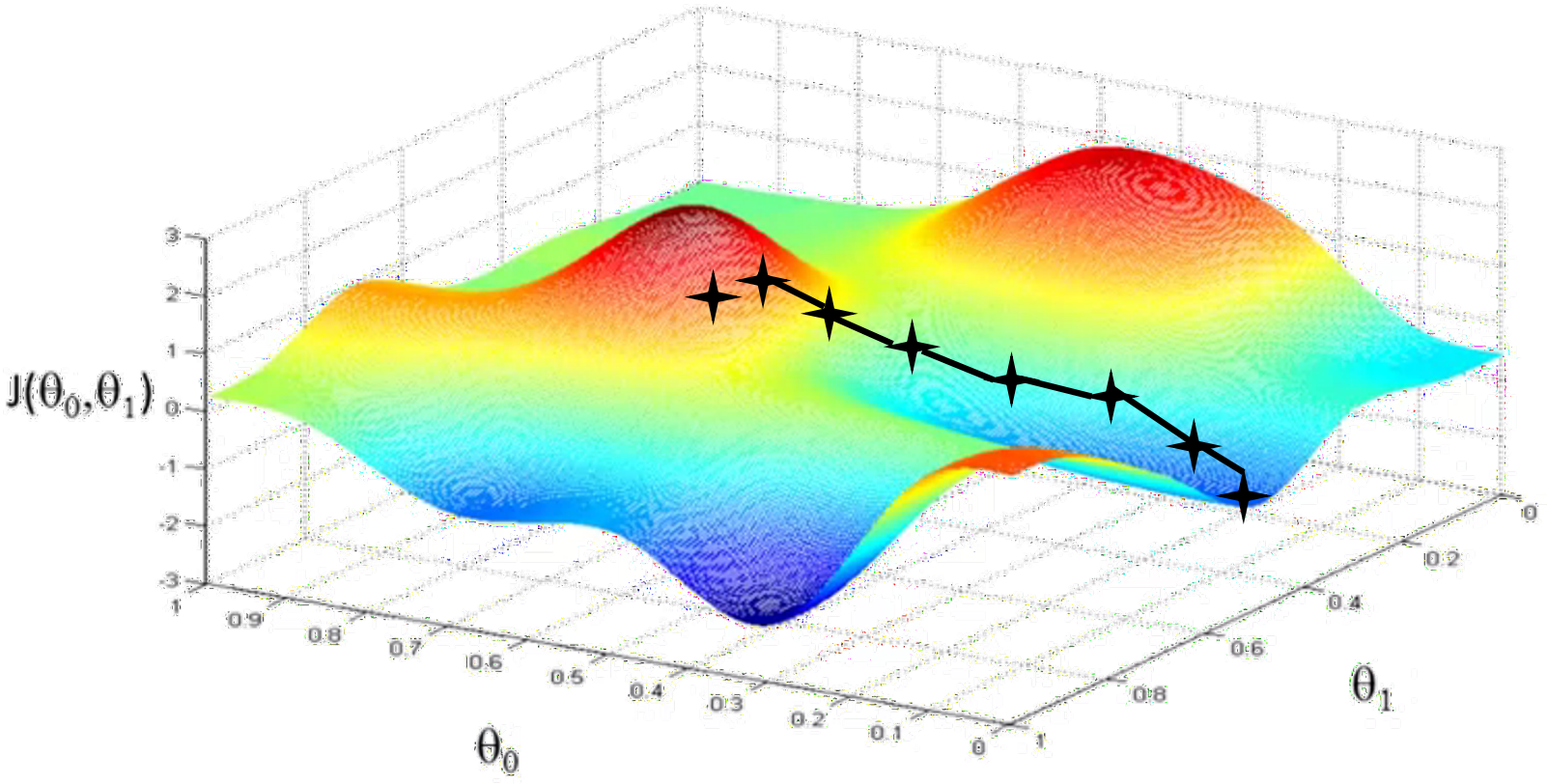
Gradient descent



Gradient descent



Gradient descent



Gradient descent algorithm

- Equation:

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

  - α: learning rate

- Updates:

$$tmp_0 := \theta_0 - \alpha . \frac{\partial}{\partial \theta_0} J(\theta_0, \theta_1)$$

$$tmp_1 := \theta_1 - \alpha . \frac{\partial}{\partial \theta_1} J(\theta_0, \theta_1)$$

$$\theta_0 := tmp_0$$

$$\theta_1 := tmp_1$$

Verifying the Gradient descent updates

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

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

Chain rule

$$[f(g(x))]' = f'(g(x)) . g(x)'$$

$f = (\,)^2$ e $g(x) = h_\theta(x^i) - y^i$

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

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

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

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

Verifying the Gradient descent updates

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

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

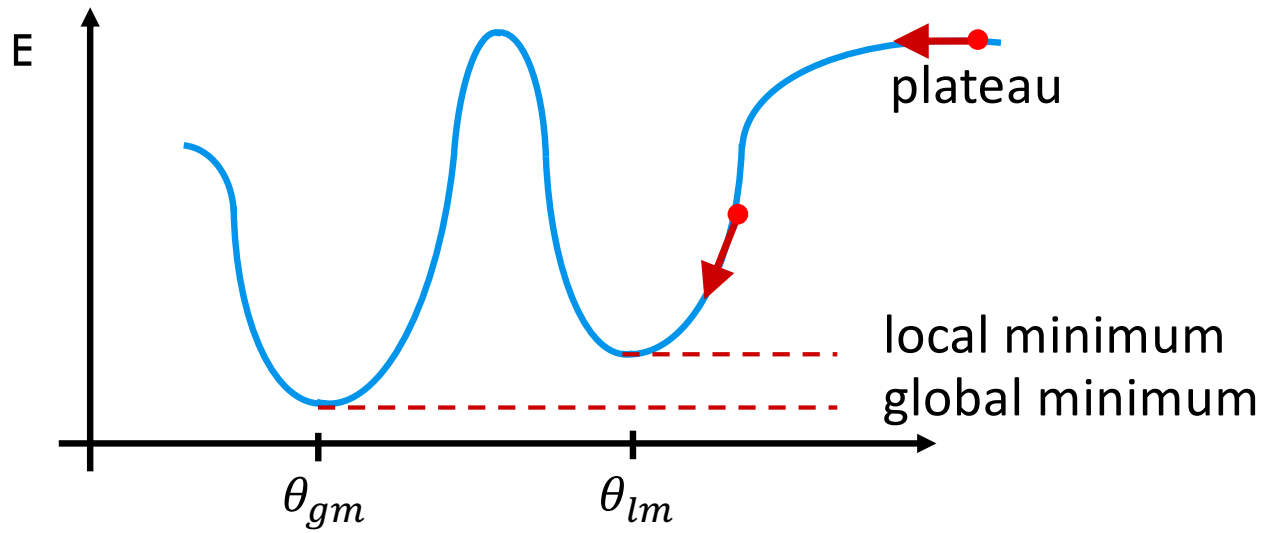
- Hence,

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

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

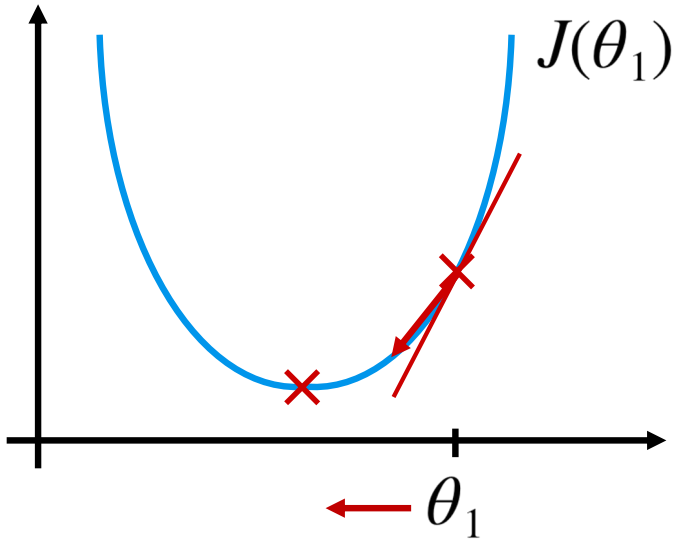
Gradient descent

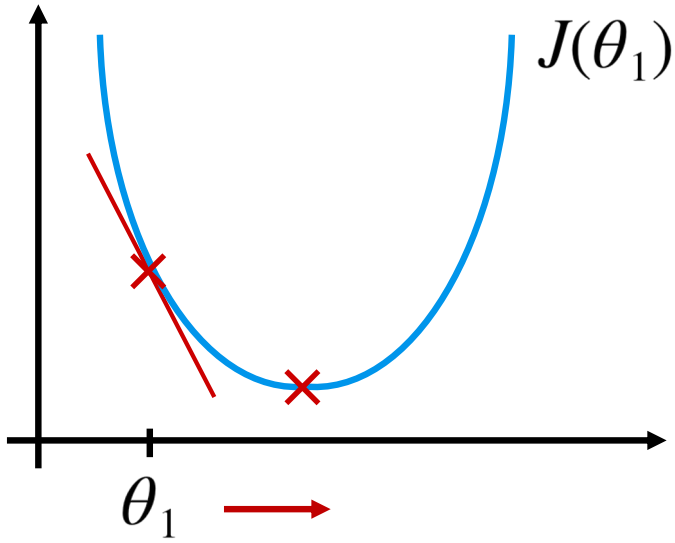
- Gradient descent approach usually presents some pittfals like **local minimum** or **plateaus**



- Fortunately, the MSE cost function for a Linear Regression is a convex function. This means that:
  - There are no local minima, just the global minimum!
  - Function never changes abruptly!

Gradient descent algorithm





$$\theta_1 = \theta_1 - \alpha . \frac{d}{d\theta_1} J(\theta_1) \quad \geq 0$$

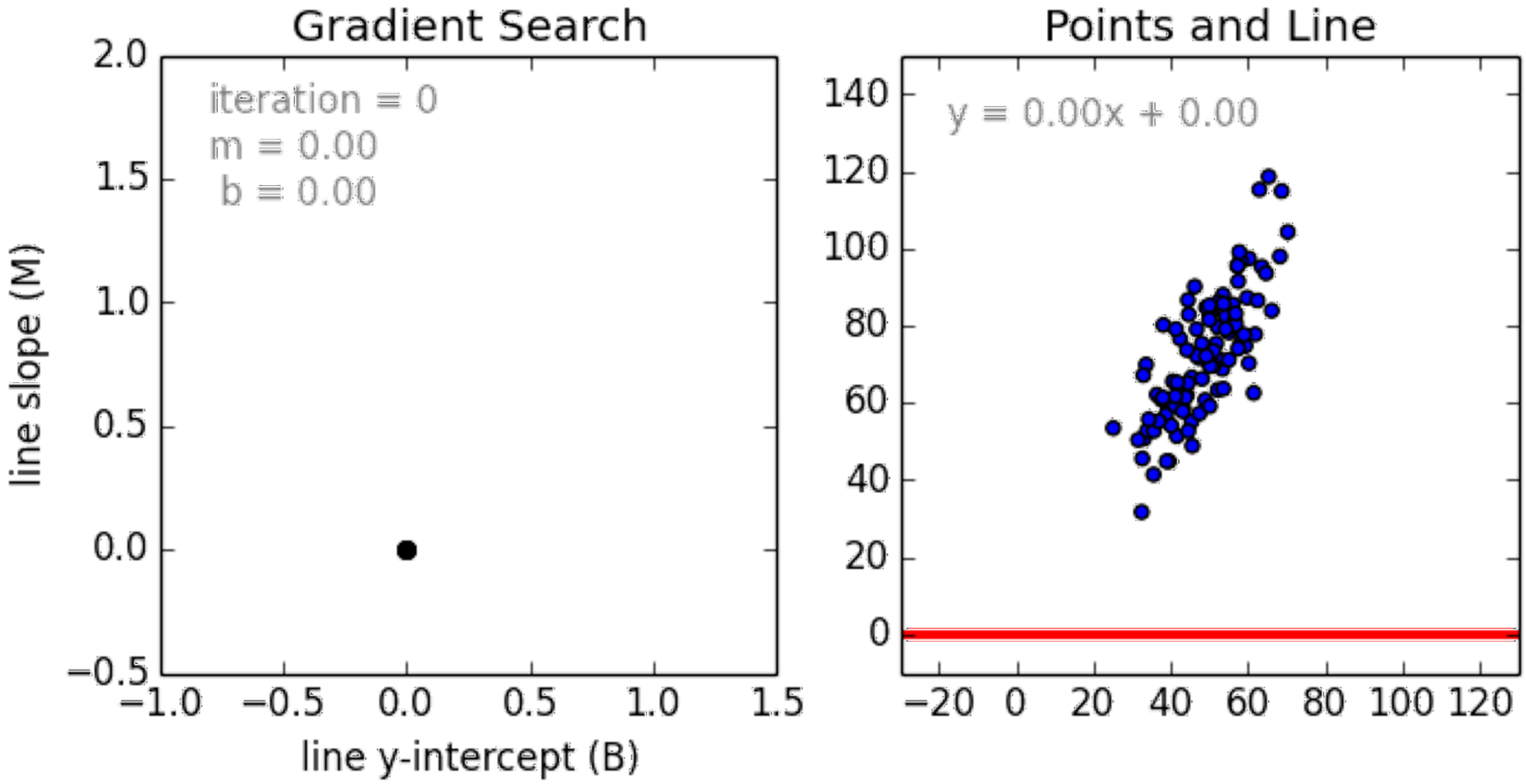
$$\theta_1 = \theta_1 - \alpha . (positive\ number)$$

$$\theta_1 = \theta_1 - \alpha . \frac{d}{d\theta_1} J(\theta_1) \quad \leq 0$$

$$\theta_1 = \theta_1 - \alpha . (negative\ number)$$

Gradient descent algorithm

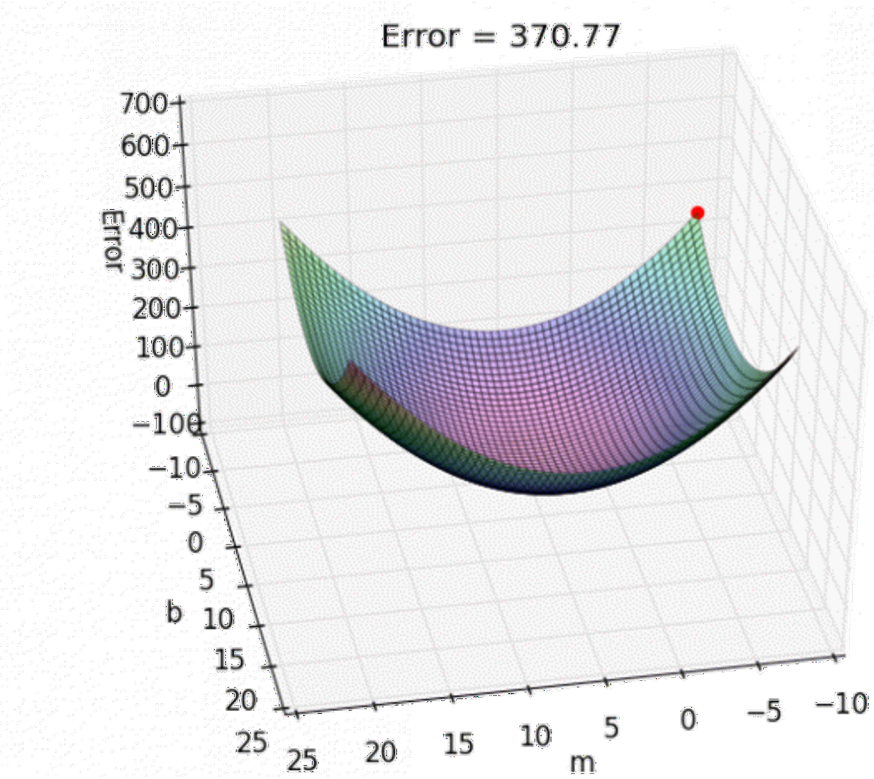
$$h_\theta(x) = \theta_0 + \theta_1 x \quad \Rightarrow \quad y = b + mx$$

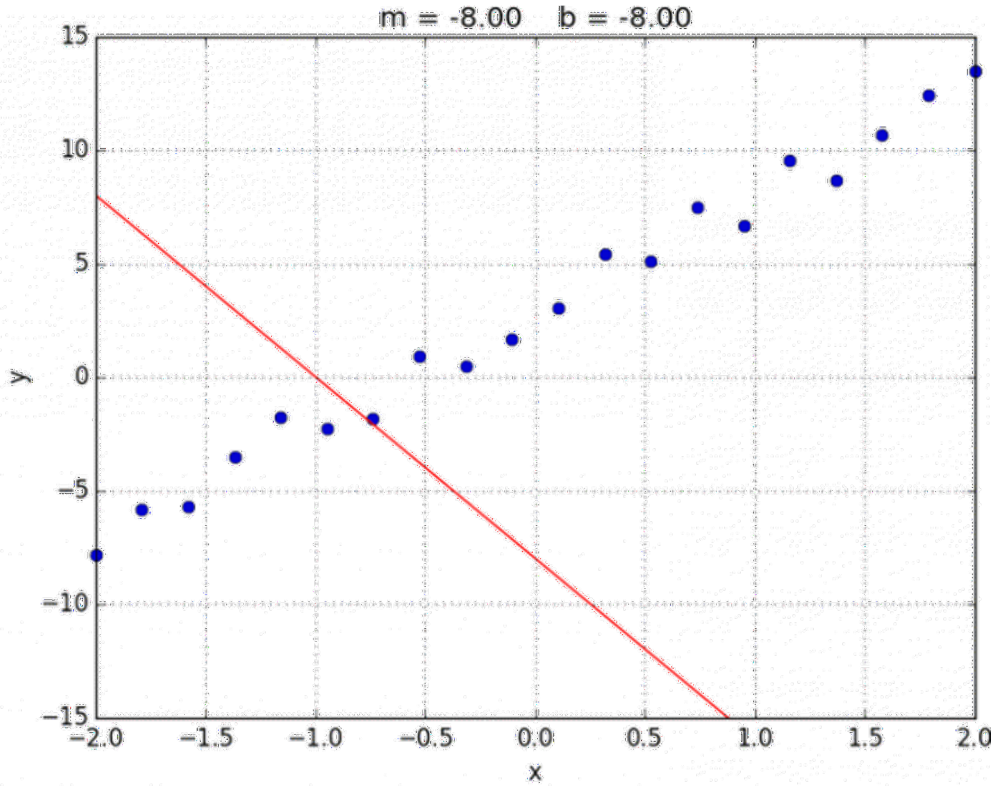


Credit: https://github.com/mattnedrich/GradientDescentExample/raw/master/gradient_descent_example.gif

Gradient descent algorithm

$$y = b + mx$$

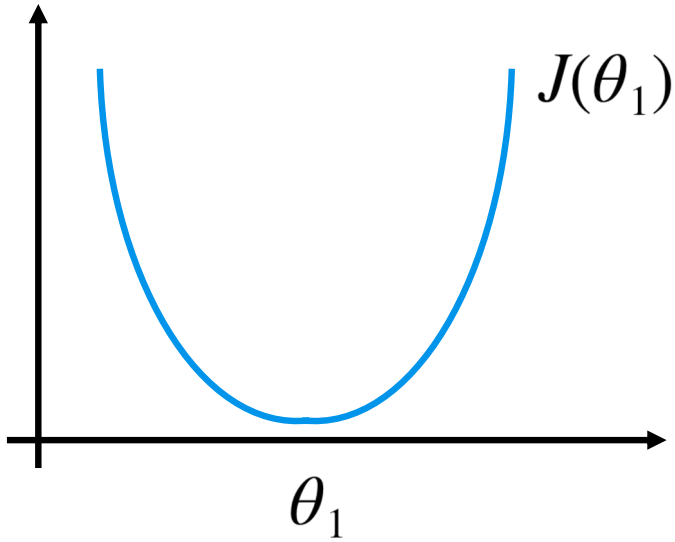




Credit: https://alykhantejani.github.io/a-brief-introduction-to-gradient-descent/

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

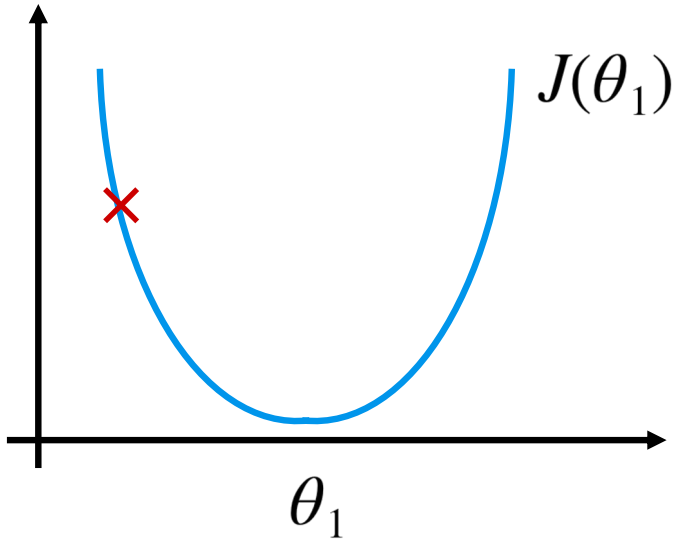
If α is too small, gradient descent can be ...



Learning rate

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

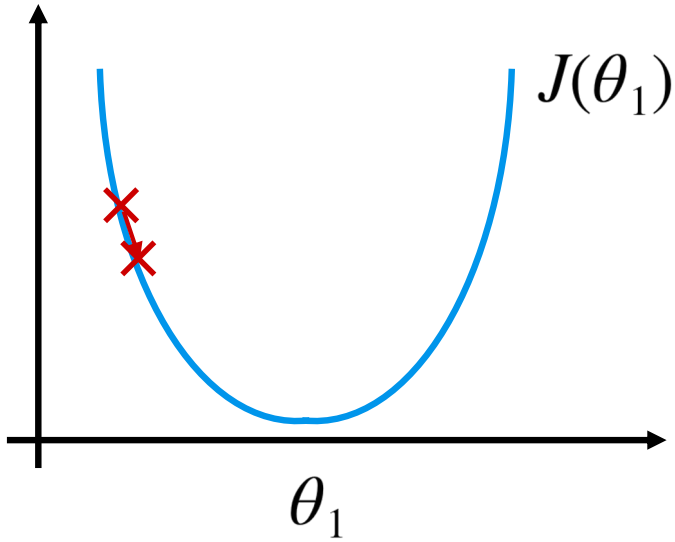
If α is too small, gradient descent can be ...



Learning rate

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

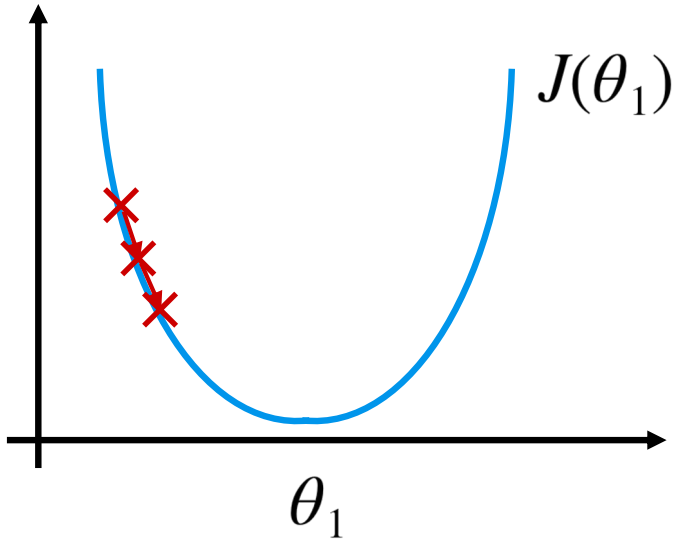
If α is too small, gradient descent can be slow.



Learning rate

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

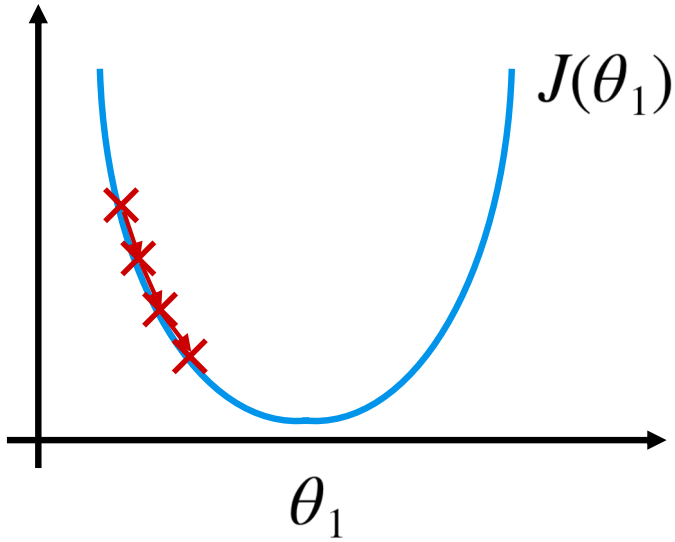
If α is too small, gradient descent can be slow.



Learning rate

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

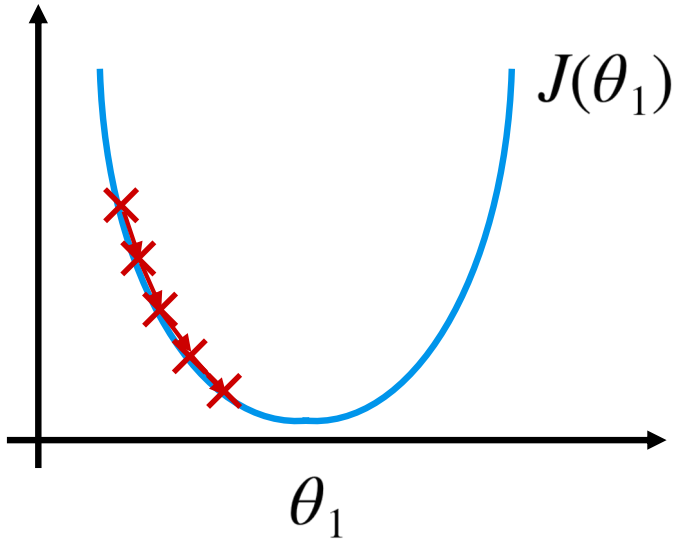
If α is too small, gradient descent can be slow.



Learning rate

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

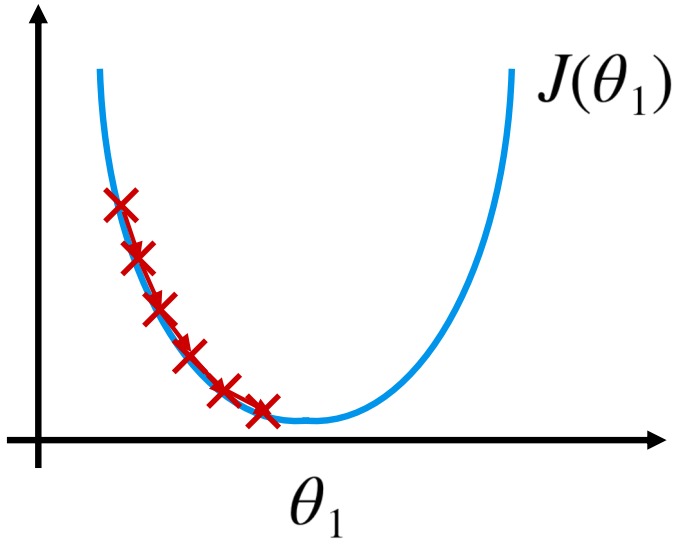
If α is too small, gradient descent can be slow.



Learning rate

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

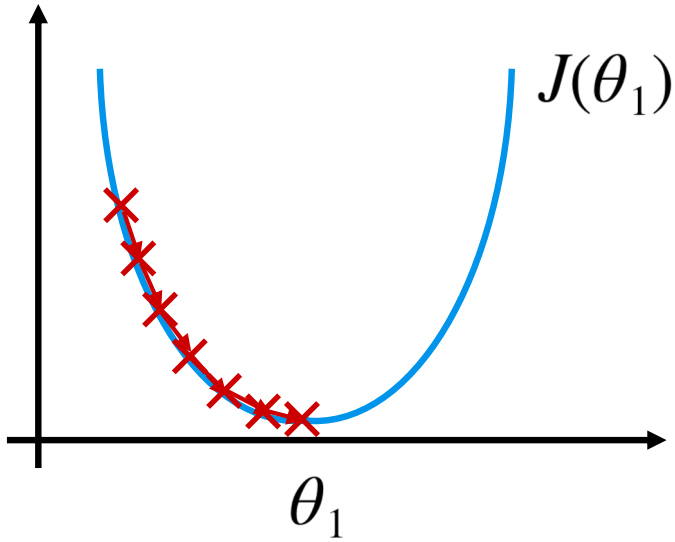
If α is too small, gradient descent can be slow.



Learning rate

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

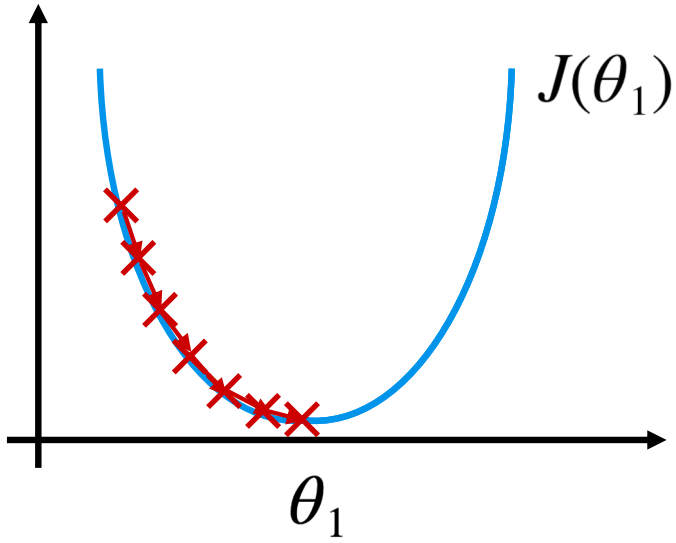
If α is too small, gradient descent can be slow.



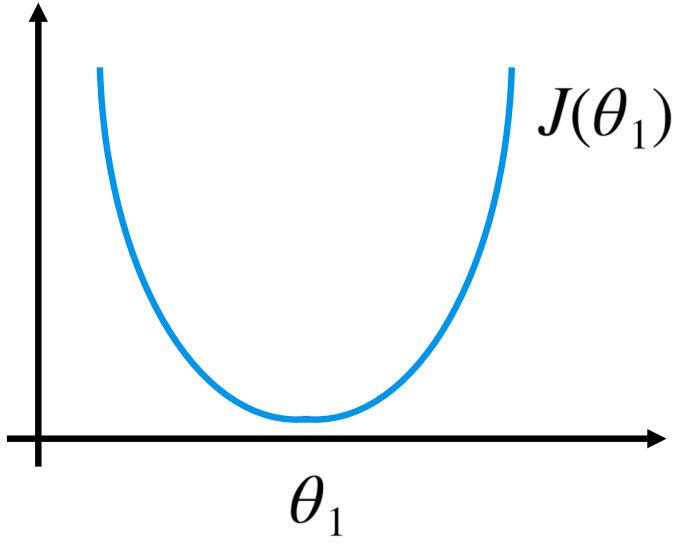
Learning rate

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

If α is too small, gradient descent can be slow.



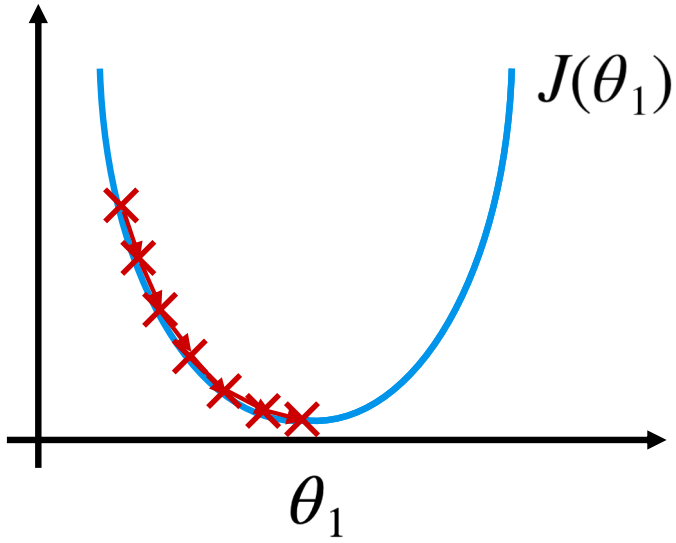
If α is too large, gradient descent can be ...



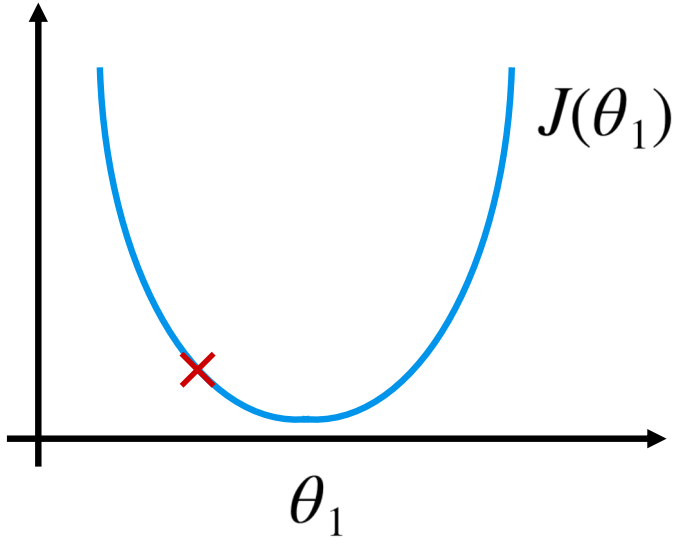
Learning rate

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

If α is too small, gradient descent can be slow.



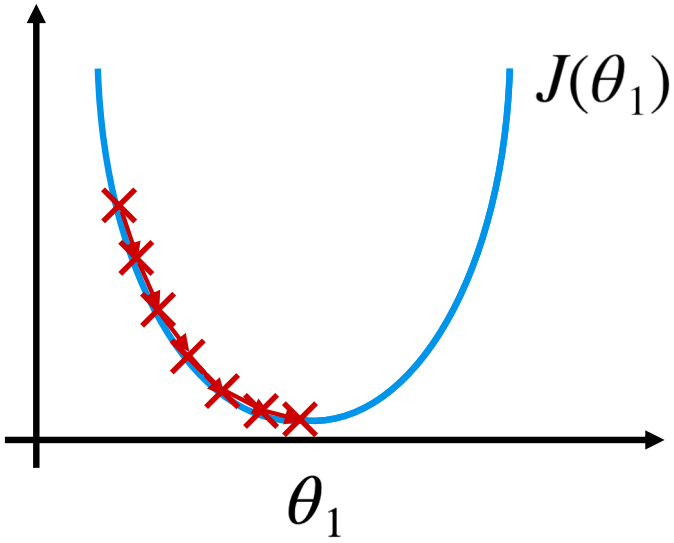
If α is too large, gradient descent can be ...



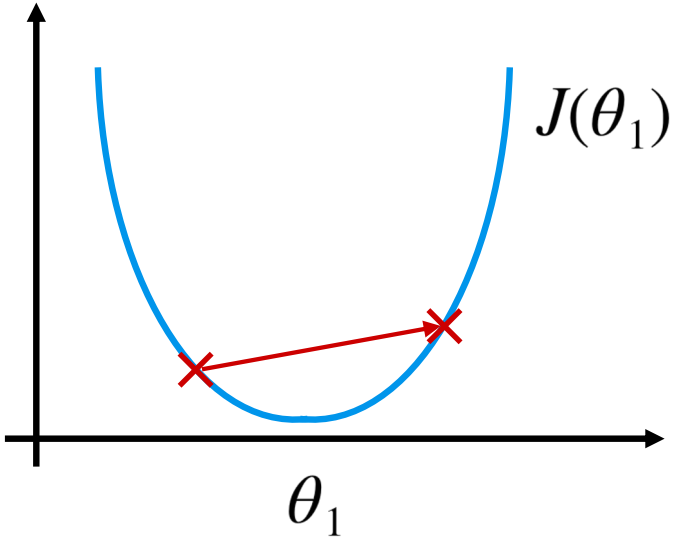
Learning rate

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

If α is too small, gradient descent can be slow.



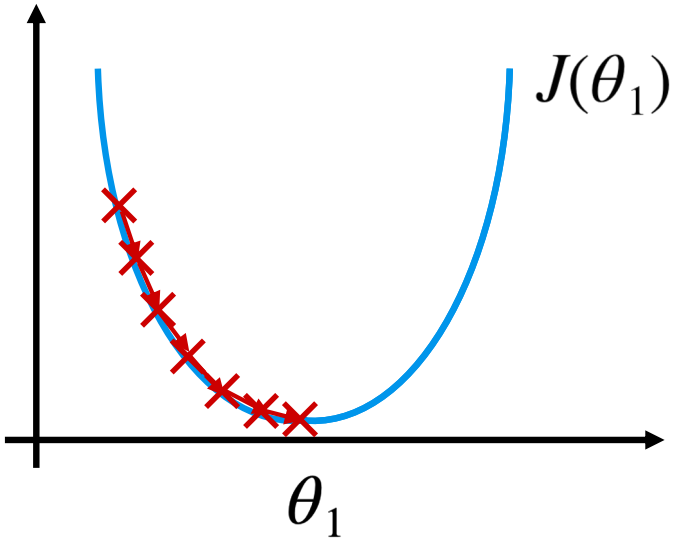
If α is too large, gradient descent can be overshoot the minimum. It may fail to converge, or even diverge.



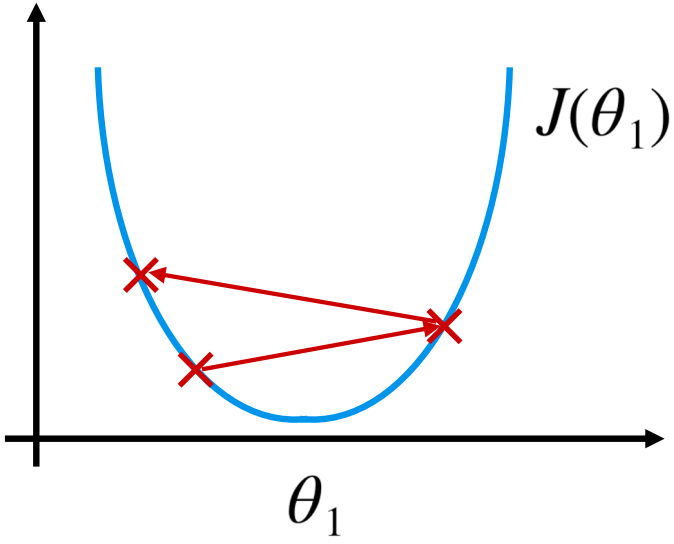
Learning rate

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

If α is too small, gradient descent can be slow.



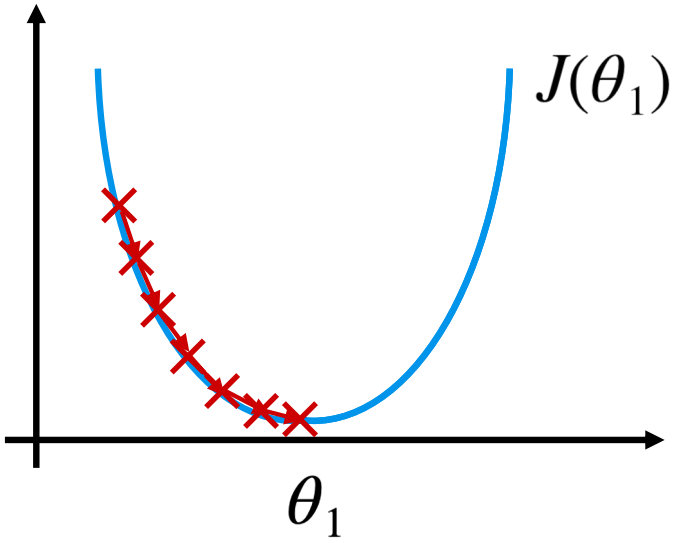
If α is too large, gradient descent can be overshoot the minimum. It may fail to converge, or even diverge.



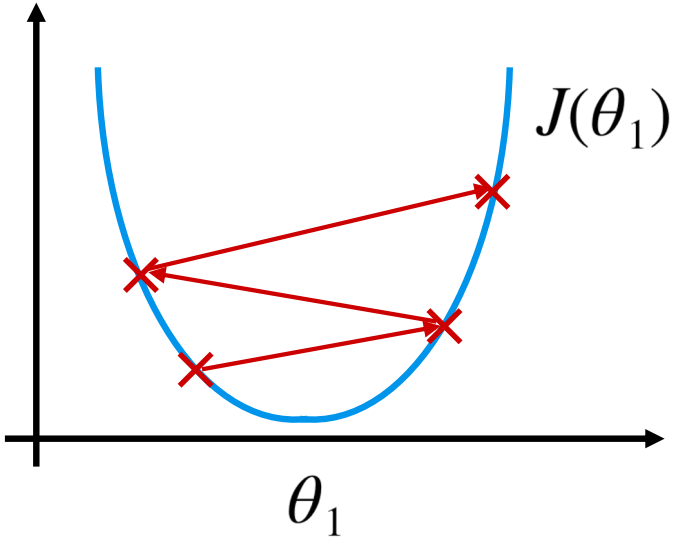
Learning rate

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

If α is too small, gradient descent can be slow.



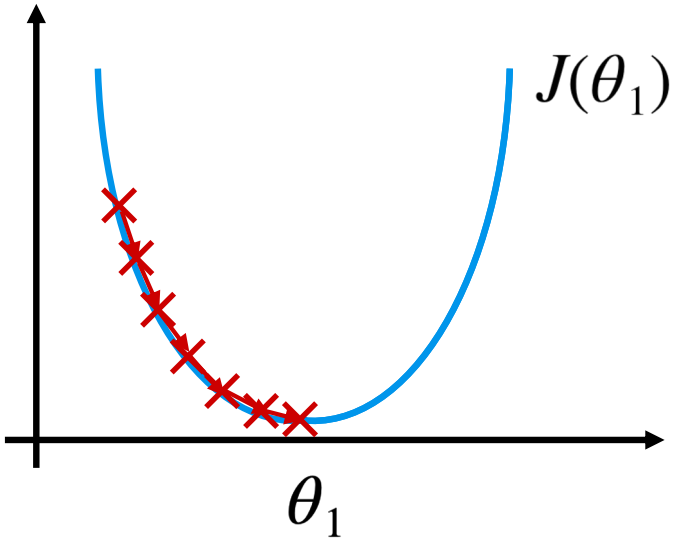
If α is too large, gradient descent can be overshoot the minimum. It may fail to converge, or even diverge.



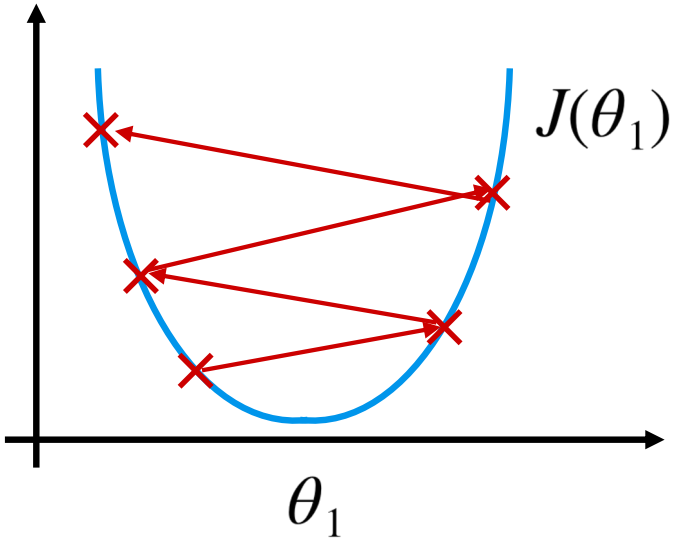
Learning rate

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

If α is too small, gradient descent can be slow.

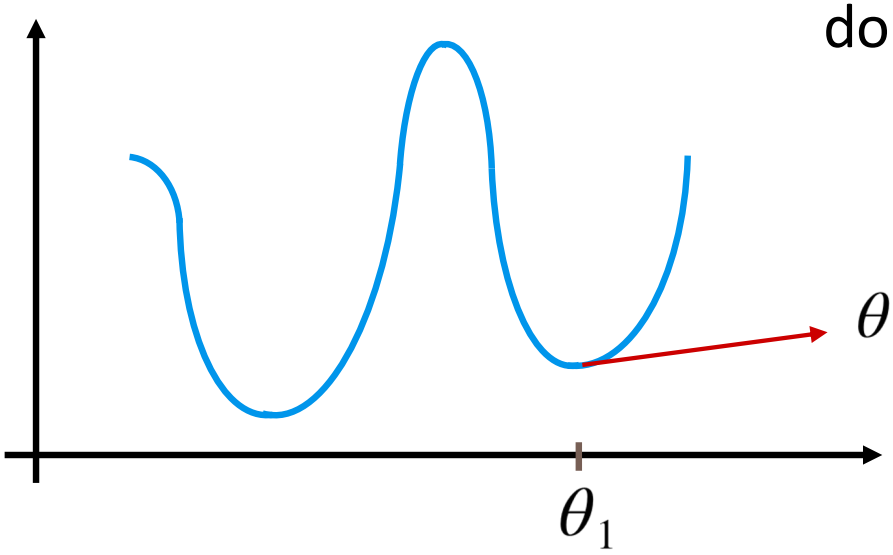


If α is too large, gradient descent can be overshoot the minimum. It may fail to converge, or even diverge.

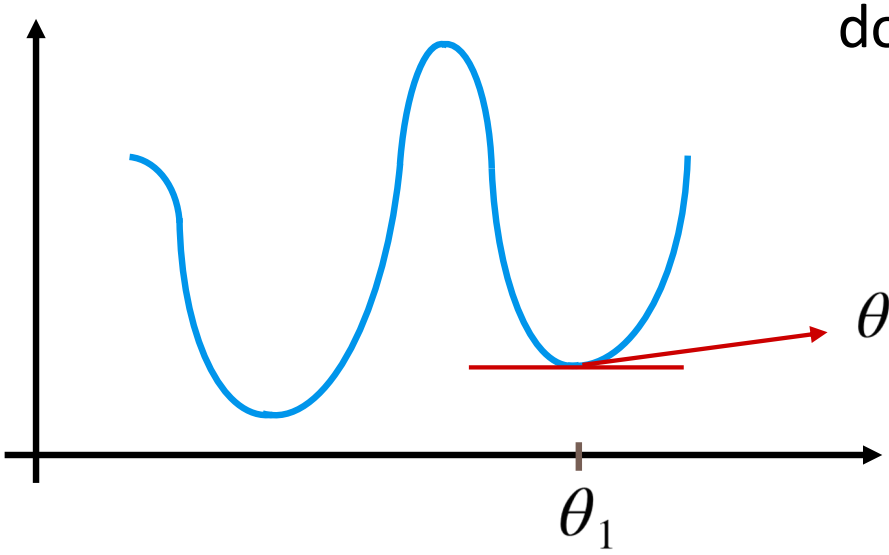


Local minimum

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



Local minimum

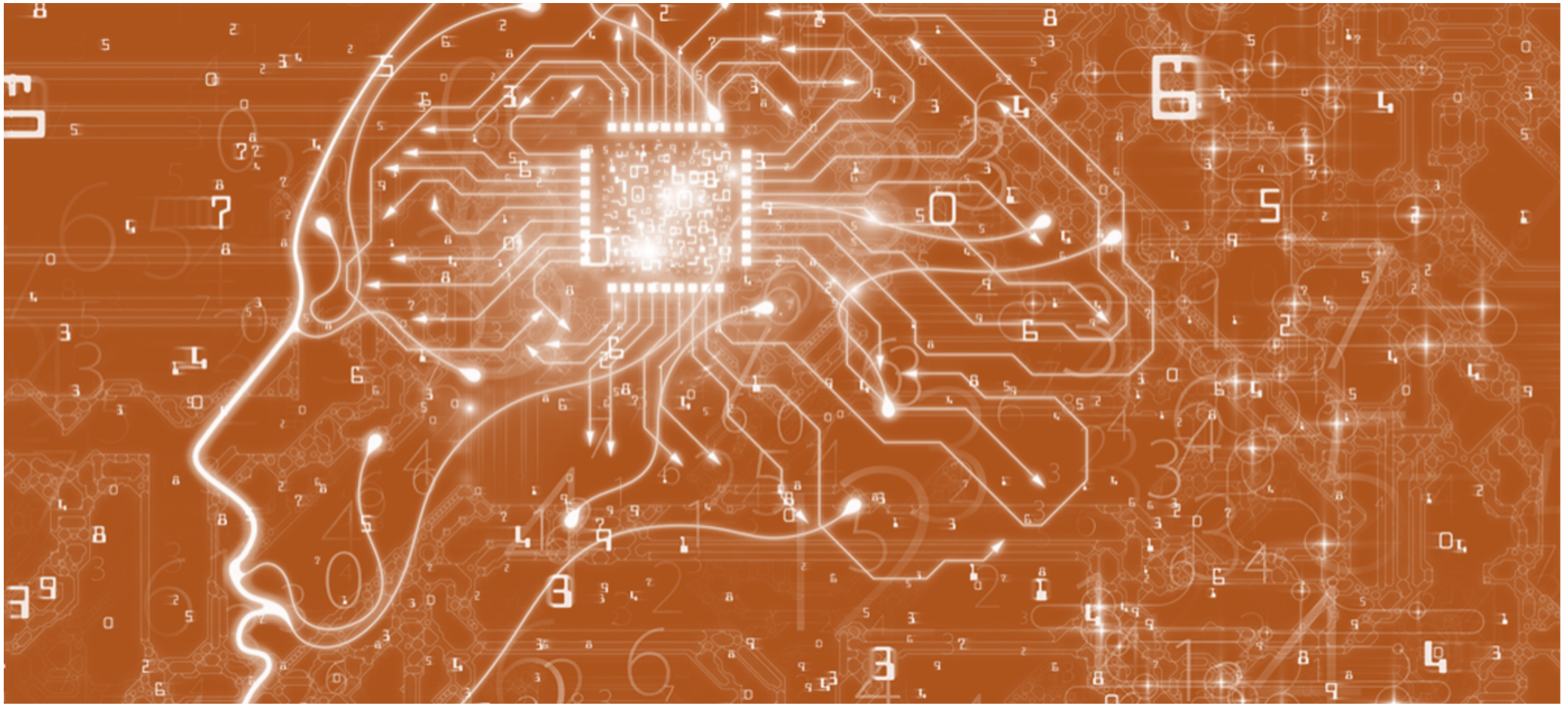


What will one step of gradient descent do?

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

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

θ_1 at local optima $= 0$



Linear regression with multiple variables

Multiple variables (features)

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

Notation:

n = number of features

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

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

Multiple variables (features)

- Hypothesis: $h_\theta(x) = \theta_0.x_0 + \theta_1.x_1 + \cdots + \theta_n.x_n$
- Parameters: $\theta_0, \theta_1, \ldots, \theta_n$
- Cost function:

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

- Gradient descent:

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

(simultaneously update for every j)

Multiple variables (features)

- Before:

$$h_\theta(x) = \theta_0 + \theta_1 x_1$$

- Now:

$$h_\theta(x) = \theta_0 + \theta_1 x_1 + \theta_2 x_2 + \theta_3 x_3 + \theta_4 x_4$$

$$h_\theta(x) = 80 + 0.1 x_1 + 10 x_2 + 3 x_3 - 2 x_4$$

Gradient Descent

Before ($n = 1$):

repeat {

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

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

}

(update simultaneously)

Now:

repeat {

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

update simultaneously θ for j $= 0,1, \ldots, n$

}

Gradient Descent

Before ($n = 1$):

repeat {

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

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

}

(update simultaneously)

Now:

repeat {

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

update simultaneously θ for j $= 0,1, \dots, n$

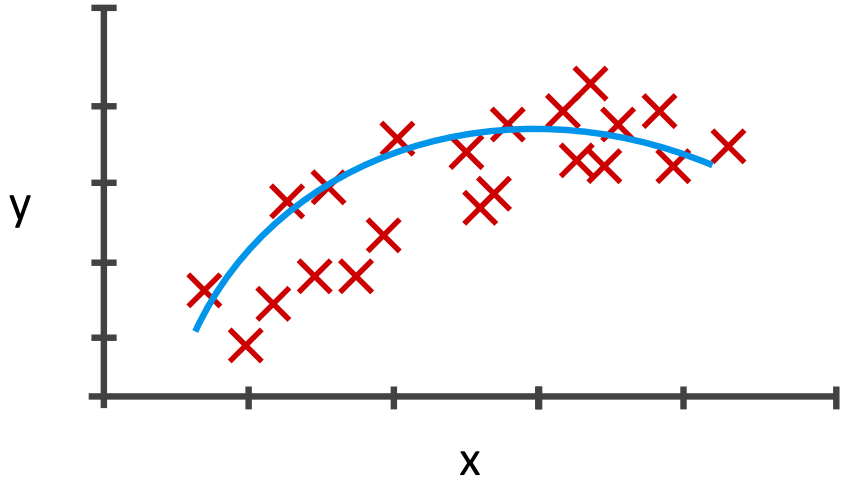
}

For $j = 0, x_j^i = 1$

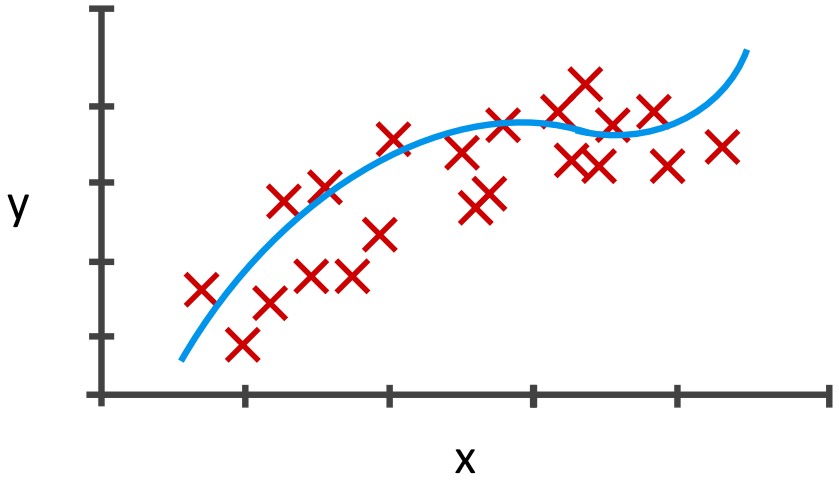
Polynomial Regression

Polynomial Regression

- What if your data is more complex that a simple straight line?
  - Consider a more complex model (polynomial of some degree)
  - Coefficients can be tunned using the same technique as in linear regressions



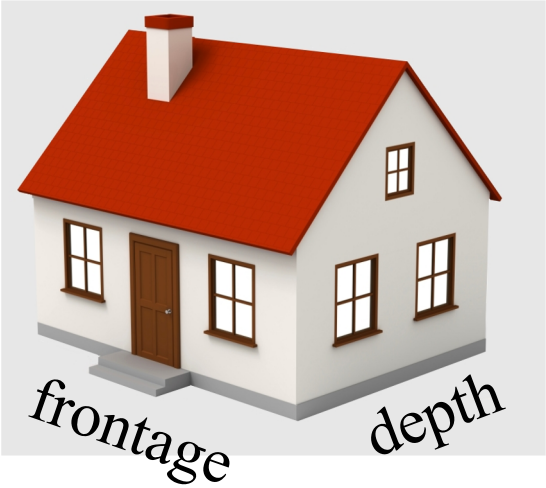
$\theta_0 + \theta_1 x + \theta_2 x^2$



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

Housing prices prediction

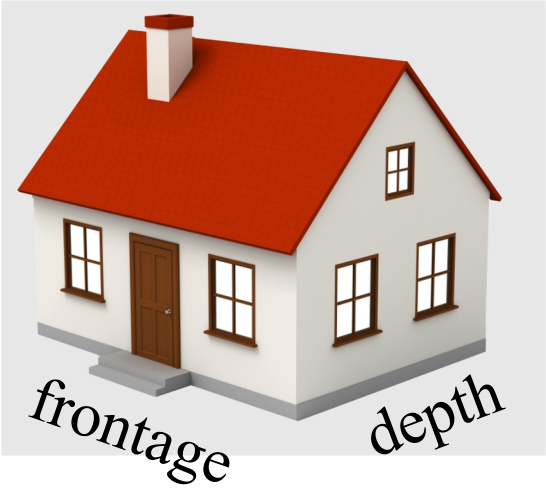
$$h_\theta(x) = \theta_0 + \theta_1 \text{frontage} + \theta_2 \text{ depth}$$



Housing prices prediction

$$h_\theta(x) = \theta_0 + \theta_1 \text{frontage} + \theta_2 \text{ depth}$$

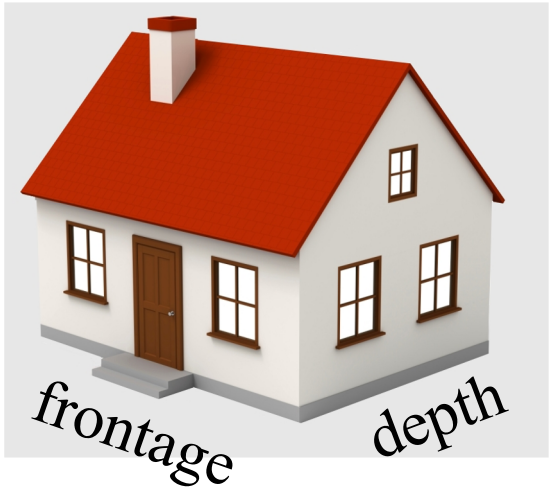
x_1 x_2



Housing prices prediction

$$h_\theta(x) = \theta_0 + \theta_1 \text{frontage} + \theta_2 \text{ depth}$$

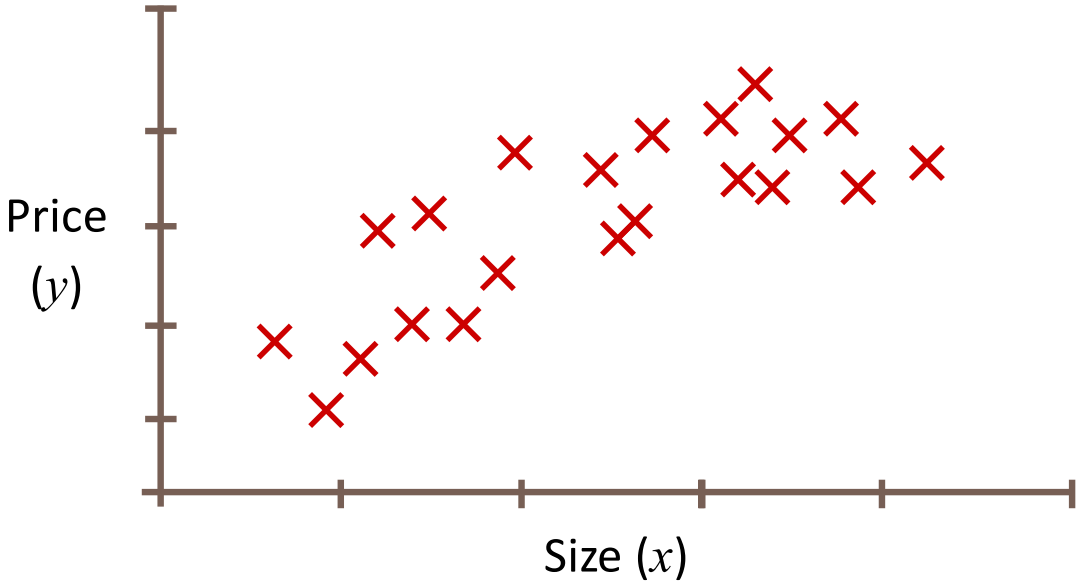
x_1 x_2



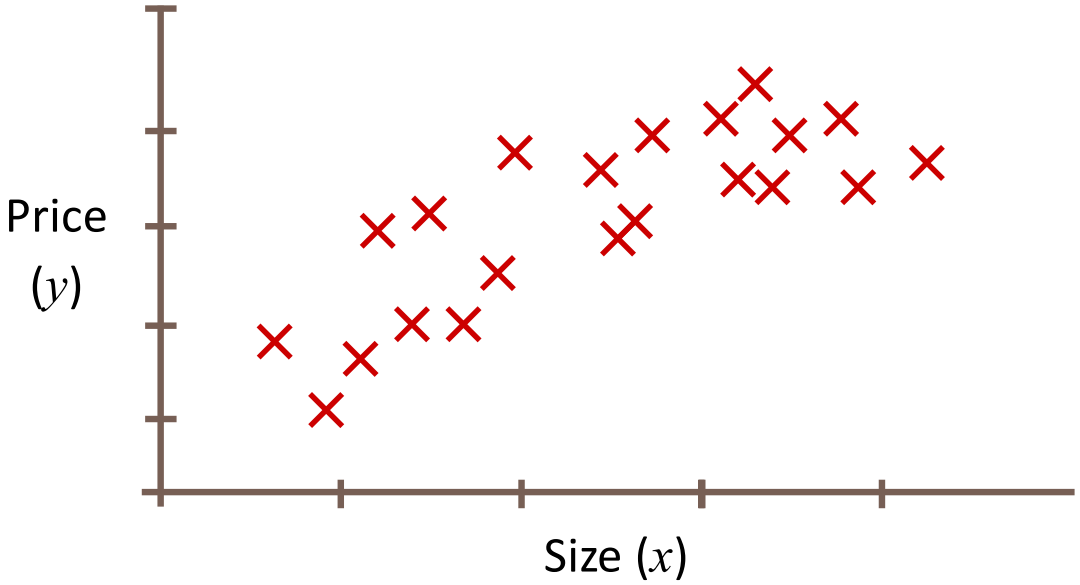
Area x = frontage x depth

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

Polynomial Regression

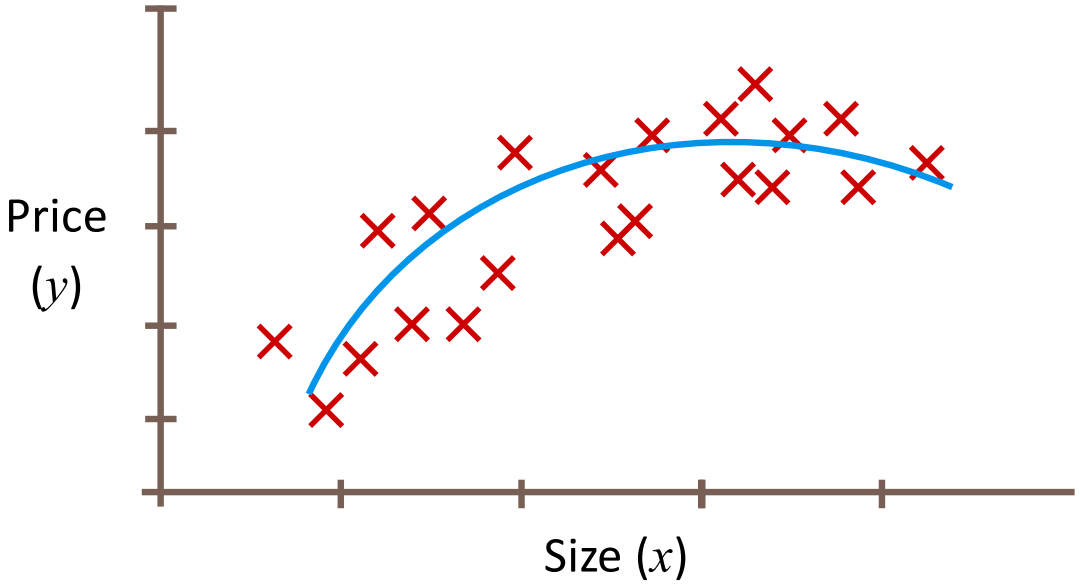


Polynomial Regression



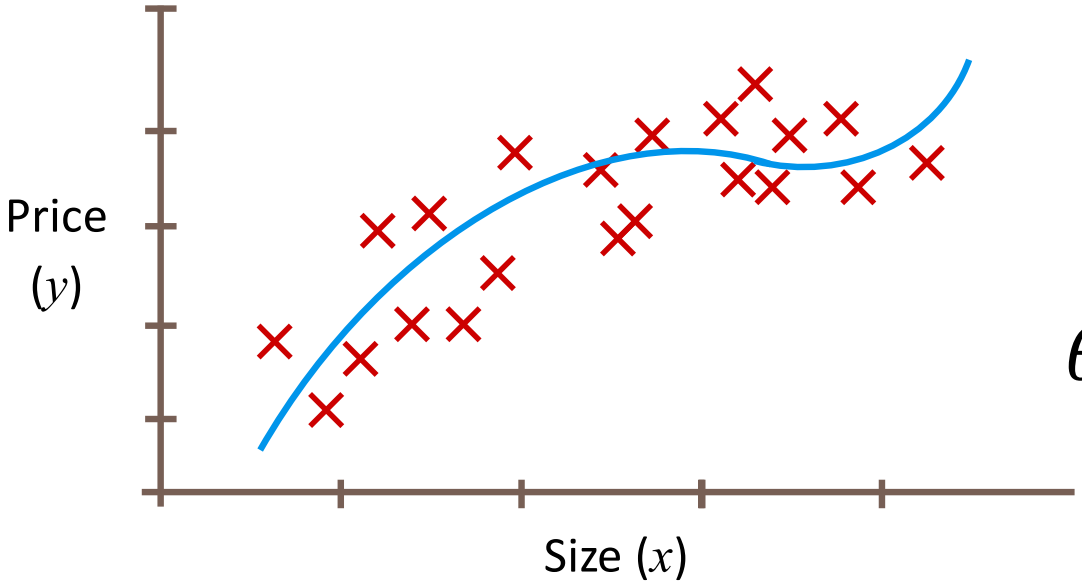
$$\theta_0 + \theta_1 x + \theta_2 x^2$$

Polynomial Regression



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

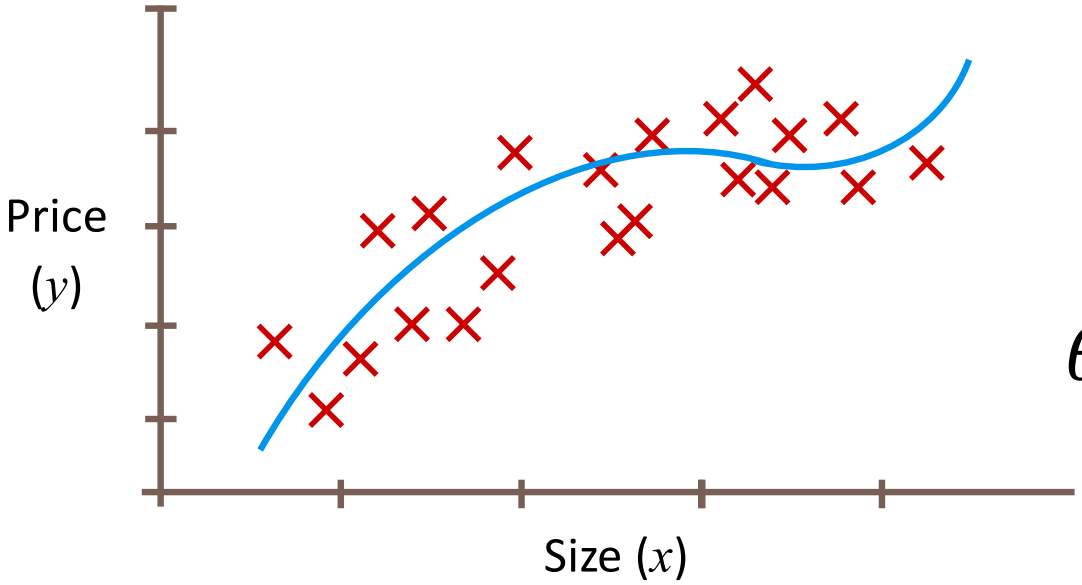
Polynomial Regression



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

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

Polynomial Regression



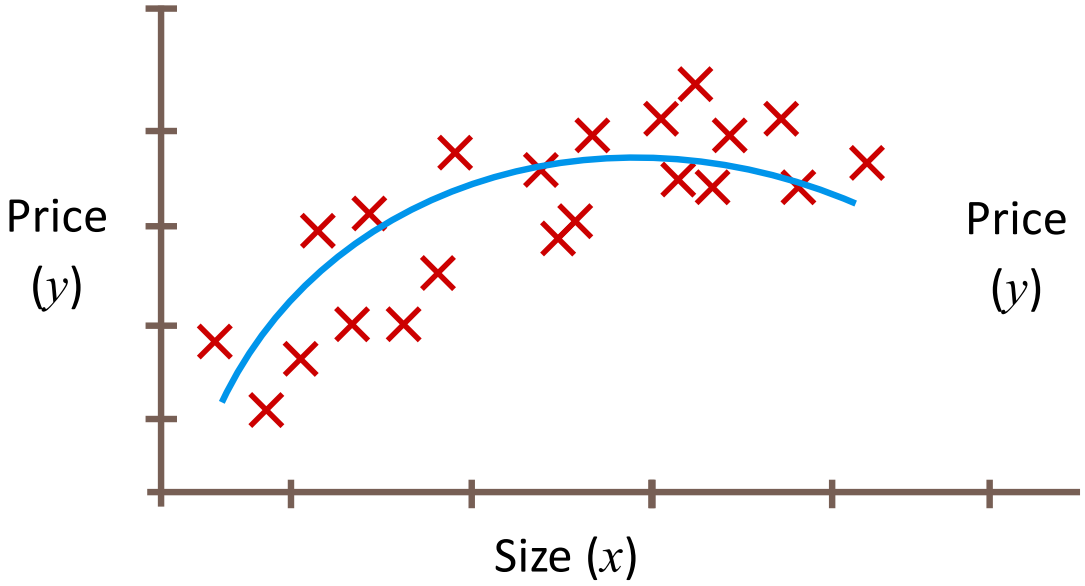
$$\theta_0 + \theta_1 x + \theta_2 x^2$$

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

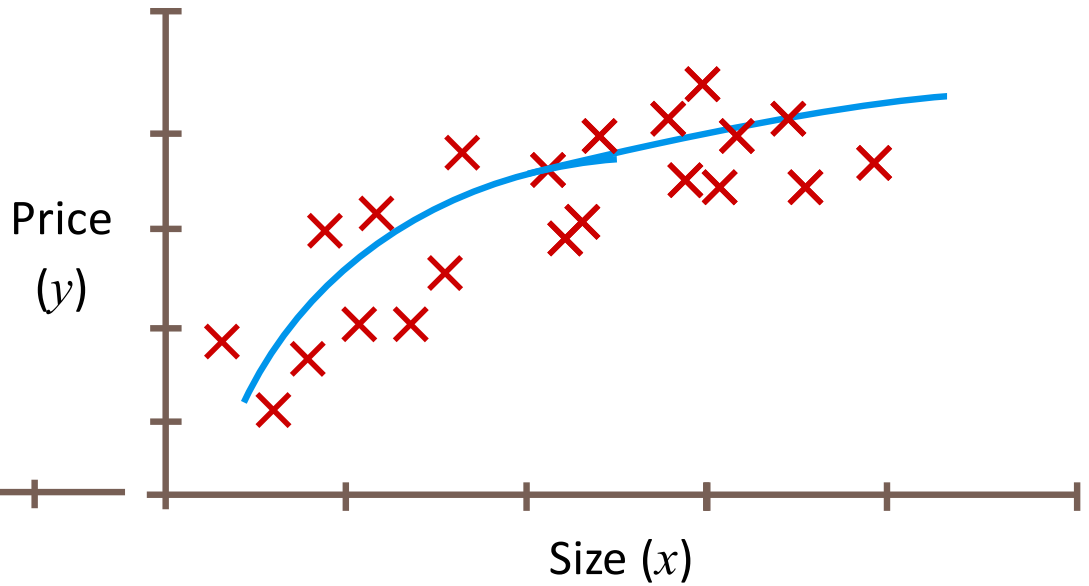
$$h_\theta(x) = \theta_0 + \theta_1 x_1 + \theta_2 x_2 + \theta_3 x_3$$
$$= \theta_0 + \theta_1(\text{size}) + \theta_2(\text{size})2 + \theta_3(\text{size})3$$

x_1 = size =1-1000
x_2 = (size)2 =1-10^6
x_3 = (size)3 =1-10^9

Polynomial Regression



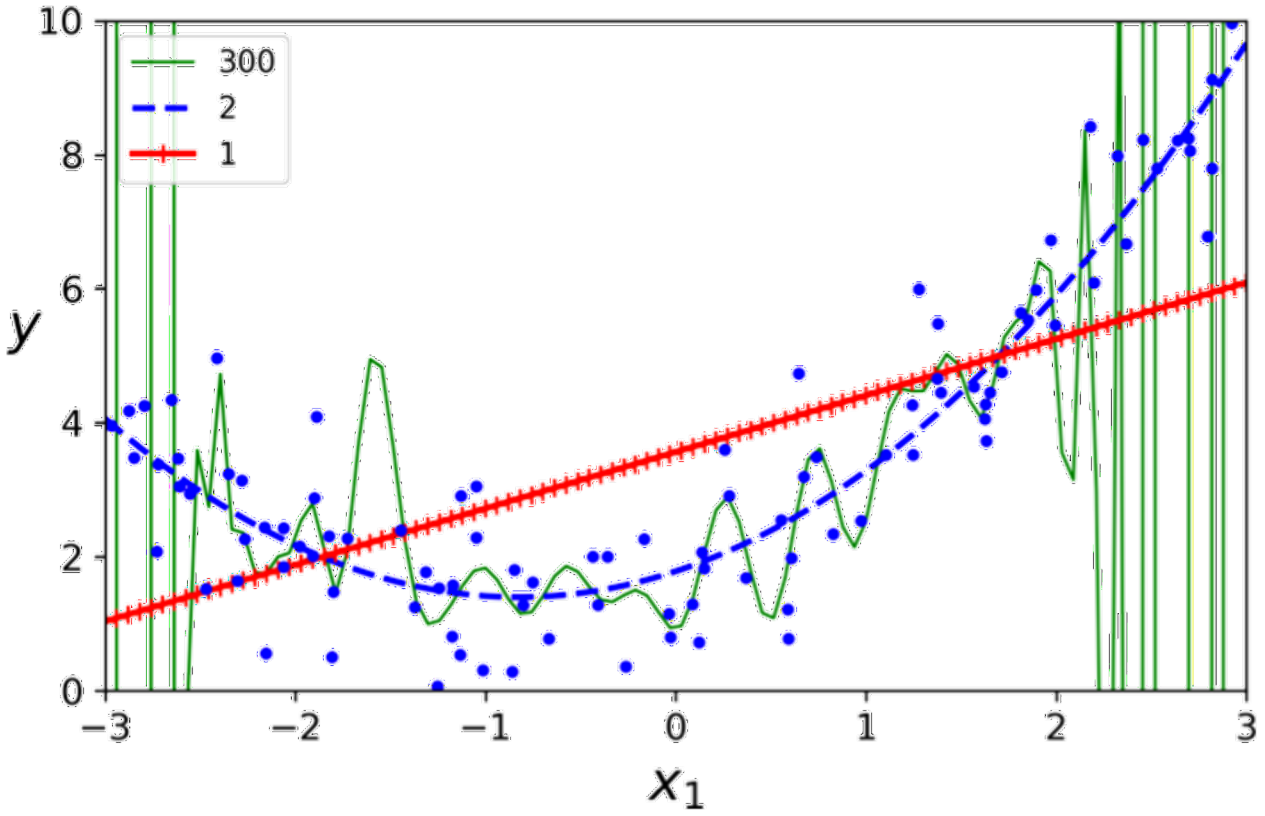
$$h_\theta(x) = \theta_0 + \theta_1(\text{size}) + \theta_2(\text{size})2$$



$$h_\theta(x) = \theta_0 + \theta_1(\text{size}) + \theta_2\sqrt{\text{size}}$$

High-degree Polynomial Regression

- We can perform high-degree PR
  - Low-degree PRs: tends to **underfit** the training data
  - High-degree PRs: tends to **overfit** the training data



Gradient Descent Variations

Algorithm variations

- **Batch:** each iteration of the Gradient Descent (GD) training uses **all examples** of the training set
  - Impractical for large datasets
- **Stochastic Gradient Descent (SGD):** each iteration of the GD training uses **only one** example of the training set
  - Large variance in Cost function
- **Mini-batch:** each iteration of the GD training uses **n examples** of the training set
  - Allows to combine advantages of batch and SGD

"Batch" Gradient Descent

- "Batch":
  - Each step of gradient descent uses **all the training examples**.

```
for i in range(nb_epochs):
  params_grad = evaluate_gradient(loss_function,data,params)
  params = params - learning_rate * params_grad
```

Epochs & Batch size & Iterations

- **Epochs:** One epoch is usually defined to be **ONE** complete run through **ALL** of the training data.

- **Batch Size:** Total number of training examples present in a **SINGLE** batch.

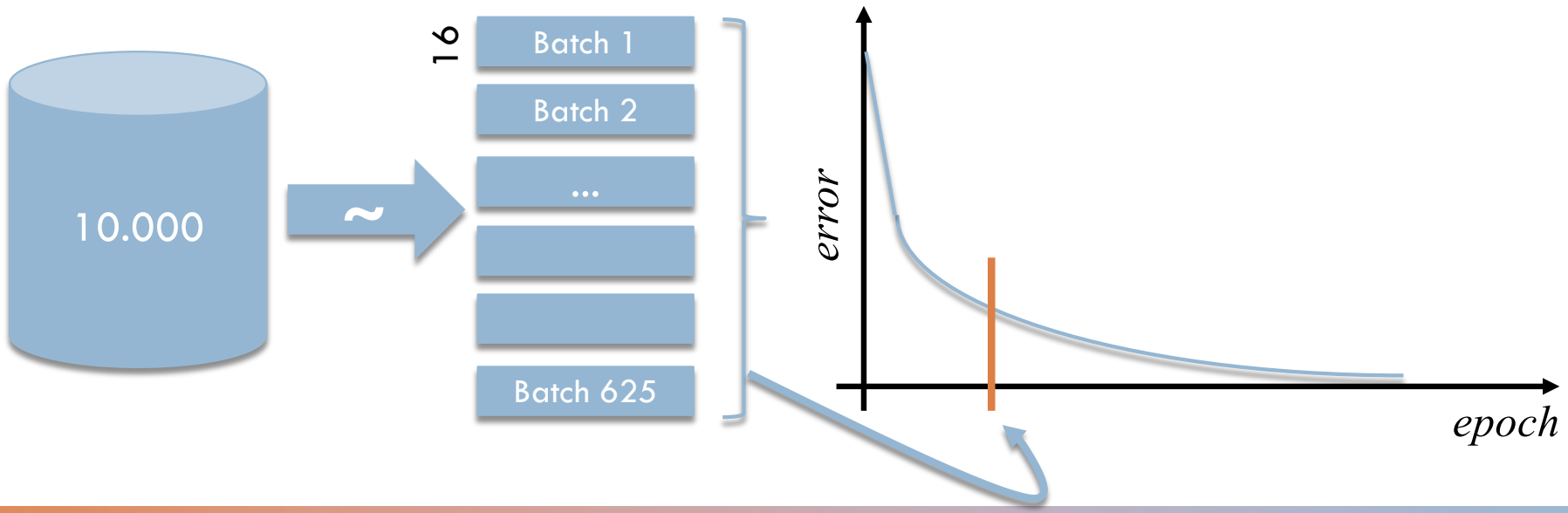
- **Iterations:** The number of batches needed to complete **ONE** epoch.

Note: The number of batches is equal to number of iterations for one epoch.

Epochs & Batch size & Iterations

- Let's say we have 10,000 training examples that we are going to use.

We can divide the dataset of 10,000 examples into **batches** of 16 then it will take 625 **iterations** to complete 1 **epoch**.



Stochastic Gradient Descent

- Stochastic:
  - Each step of gradient descent uses **one training example**.

```
for i in range(nb_epochs):
  np.random.shuffle(data)
  for example in data:
    params_grad = evaluate_gradient(loss_function,example,params)
    params = params - learning_rate * params_grad
```

Mini-batch Gradient Descent

- Mini-batch:
  - Each step of gradient descent uses ***b* training examples**.

```
for i in range(nb_epochs):
  np.random.shuffle(data)
  for batch in get_batches(data,batch_size=16):
    params_grad = evaluate_gradient(loss_function,batch,params)
    params = params - learning_rate * params_grad
```

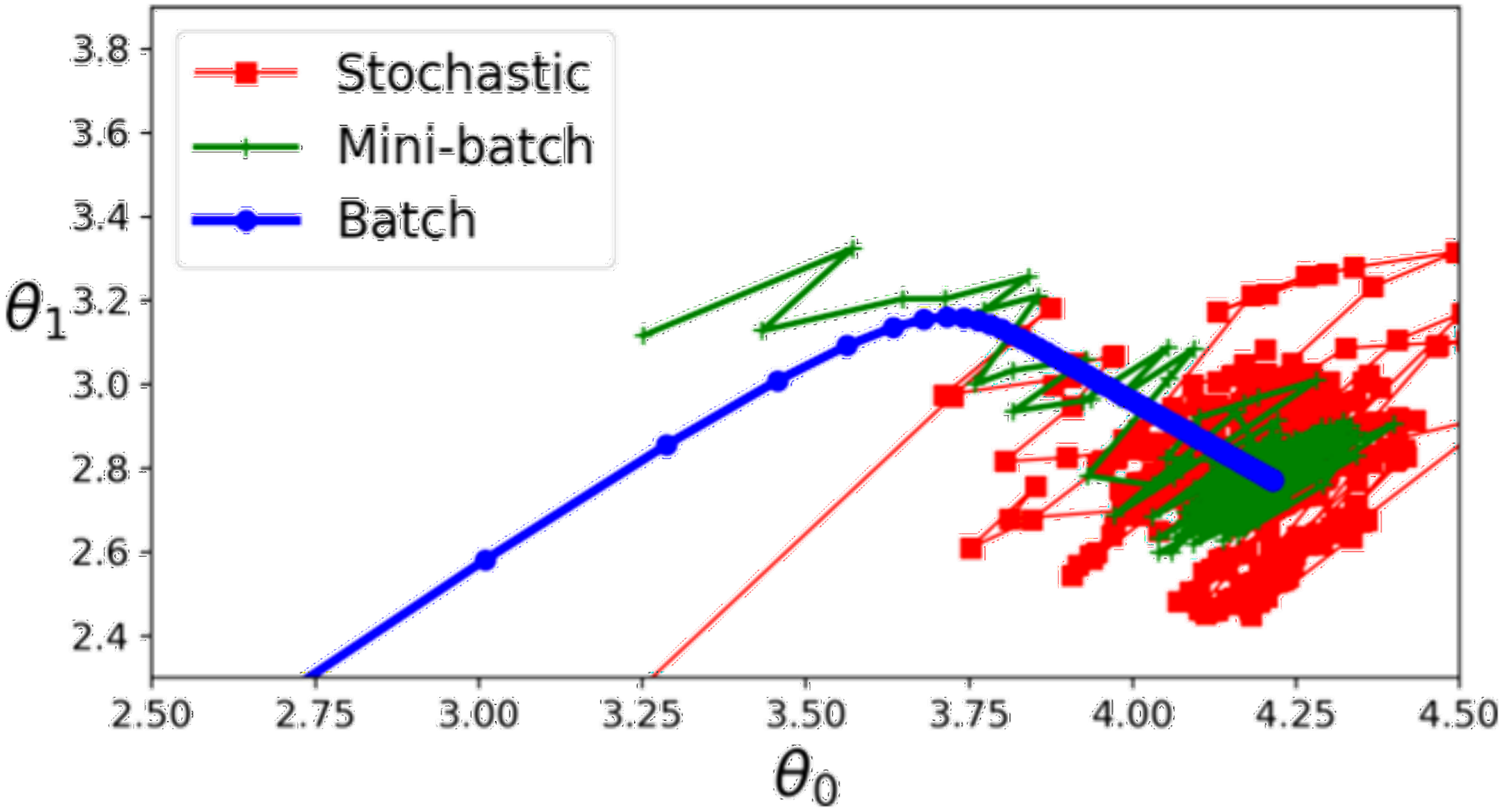
GD methods


```
for i in range(nb_epochs):
  params_grad = evaluate_gradient(loss_function,data,params)
  params = params - learning_rate * params_grad
```

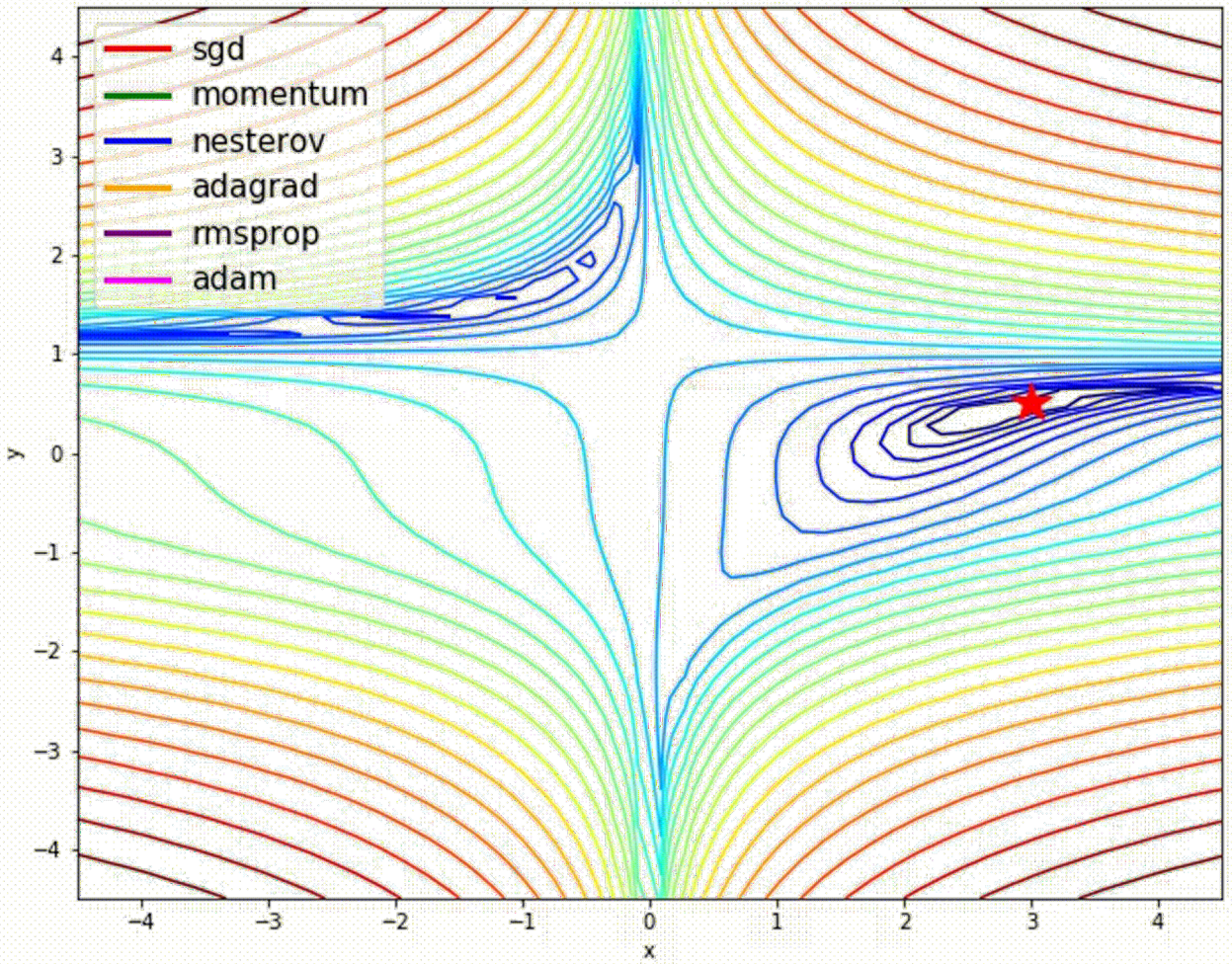
```
for i in range(nb_epochs):
  np.random.shuffle(data)
  for example in data:
    params_grad = evaluate_gradient(loss_function,example,params)
    params = params - learning_rate * params_grad
```

```
for i in range(nb_epochs):
  np.random.shuffle(data)
  for batch in get_batches(data,batch_size=16):
    params_grad = evaluate_gradient(loss_function,batch,params)
    params = params - learning_rate * params_grad
```

Algorithm variations



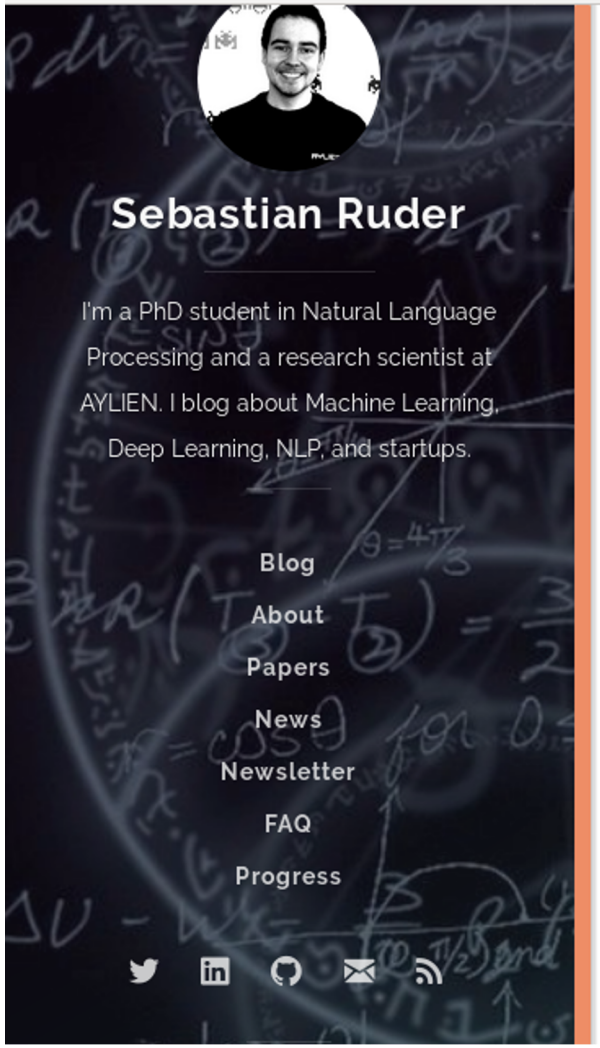
Algorithm variations

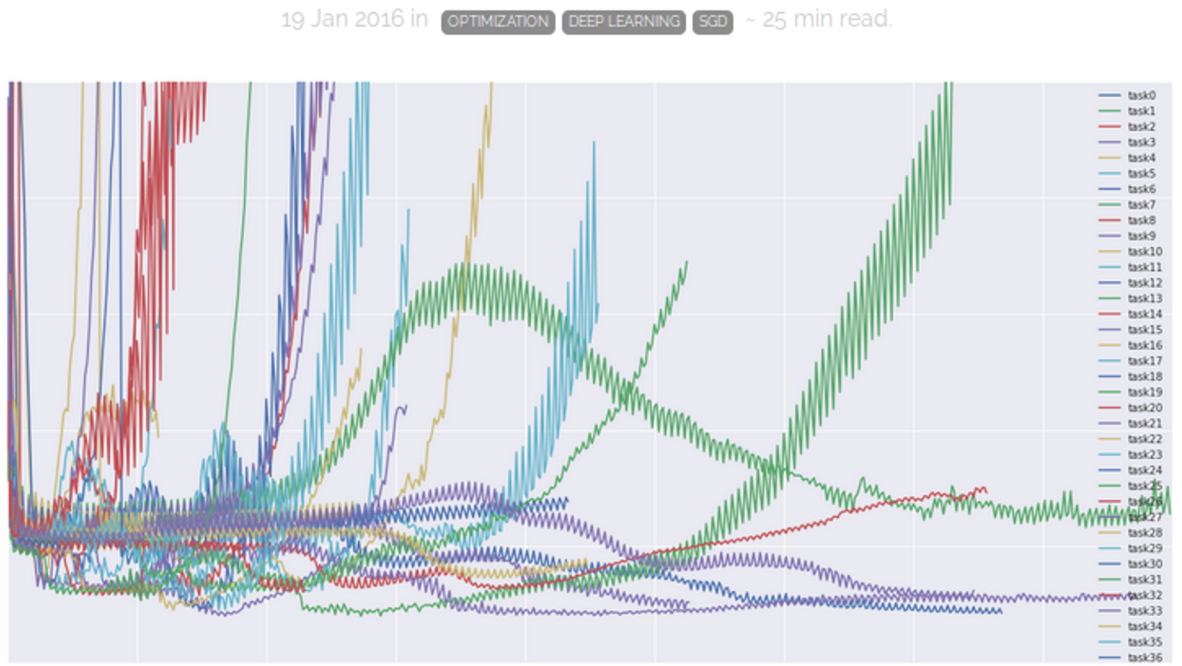


Credit: https://github.com/ilguyi/optimizers.numpy

Algorithm variations

http://ruder.io/optimizing-gradient-descent



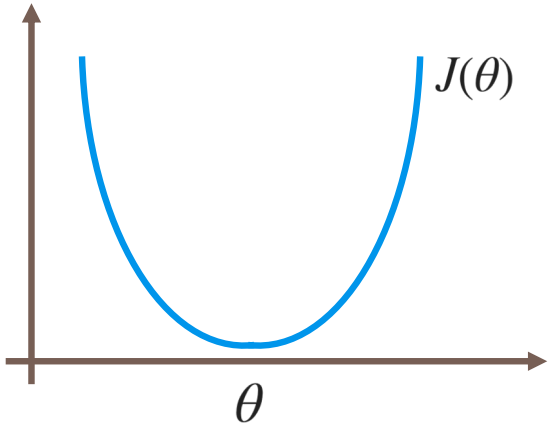


Normal Equation

- Normal equation: Method to solve θ analytically
- Intuition:
- if 1D, $\theta \in \mathbb{R}$

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

$$\frac{d}{d\theta} J(\theta) = \cdots = 0, \text{ solve for } \theta$$



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

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

$$\frac{\partial}{\partial \theta_j} J(\theta) = \cdots = 0, \text{ solve for } \theta_0, \theta_1, \ldots, \theta_n$$

Normal Equation Solution

- Examples: $m = 4$

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

Normal Equation Solution

- Examples: $m = 4$

x_0	Size (feet^2) x_1	Number of bedrooms x_2	Number of floors x_3	Age of home (years) x_4	Price ($) in 1000's 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

Normal Equation Solution

- Examples: $m = 4$

x_0	Size (feet^2) x_1	Number of bedrooms x_2	Number of floors x_3	Age of home (years) x_4	Price ($) in 1000's 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

Normal Equation Solution

- Examples: $m = 4$

x_0	Size (feet^2) x_1	Number of bedrooms x_2	Number of floors x_3	Age of home (years) x_4	Price ($) in 1000's 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

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

Normal Equation Solution

- Examples: $m = 4$

x_0	Size (feet^2) x_1	Number of bedrooms x_2	Number of floors x_3	Age of home (years) x_4	Price ($) in 1000's 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

$$X = \begin{bmatrix} 1 & 2104 & 5 & 1 & 45 \\ 1 & 1416 & 3 & 2 & 40 \\ 1 & 1534 & 3 & 2 & 30 \\ 1 & 852 & 2 & 1 & 36 \end{bmatrix}_{mx(n+1)} \qquad y = \begin{bmatrix} 460 \\ 232 \\ 315 \\ 178 \end{bmatrix}_{m}$$

Normal Equation Solution

- Examples: $m = 4$

x_0	Size (feet^2) x_1	Number of bedrooms x_2	Number of floors x_3	Age of home (years) x_4	Price ($) in 1000's 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

$$X = \begin{bmatrix} 1 & 2104 & 5 & 1 & 45 \\ 1 & 1416 & 3 & 2 & 40 \\ 1 & 1534 & 3 & 2 & 30 \\ 1 & 852 & 2 & 1 & 36 \end{bmatrix}_{mx(n+1)} \qquad y = \begin{bmatrix} 460 \\ 232 \\ 315 \\ 178 \end{bmatrix}_{m}$$

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

Normal Equation Solution

- m examples $(x^{(1)}, y^{(1)}), \ldots, (x^{(m)}, y^{(m)})$ and n features

$$x^{(i)} = \begin{bmatrix} x_0^{(i)} \\ x_1^{(i)} \\ x_2^{(i)} \\ \vdots \\ x_n^{(i)} \end{bmatrix} \in \mathbb{R}^{n+1} \qquad X = \begin{bmatrix} (x^{(1)})^T \\ (x^{(2)})^T \\ \vdots \\ (x^{(m)})^T \end{bmatrix}$$

Normal Equation Solution

- m examples $(x^{(1)}, y^{(1)}), \ldots, (x^{(m)}, y^{(m)})$ and n features

$$x^{(i)} = \begin{bmatrix} x_0^{(i)} \\ x_1^{(i)} \\ x_2^{(i)} \\ \vdots \\ x_n^{(i)} \end{bmatrix} \in \mathbb{R}^{n+1} \qquad X = \begin{bmatrix} (x^{(1)})^T \\ (x^{(2)})^T \\ \vdots \\ (x^{(m)})^T \end{bmatrix}$$

- E.g.: $x^{(i)} = \begin{bmatrix} 1 \\ x_1^{(i)} \end{bmatrix}$

$$X = \begin{bmatrix} 1 & x_1^{(1)} \\ \vdots & \vdots \\ \vdots & \vdots \\ 1 & x_m^{(1)} \end{bmatrix}_{mx2}$$

Normal Equation Solution

- m examples $(x^{(1)}, y^{(1)}), \ldots, (x^{(m)}, y^{(m)})$ and n features
- E.g.: $x^{(i)} = \begin{bmatrix} 1 \\ x_1^{(i)} \end{bmatrix}$

$$X = \begin{bmatrix} 1 & x_1^{(1)} \\ \vdots & \vdots \\ \vdots & \vdots \\ 1 & x_m^{(1)} \end{bmatrix}_{mx2}$$

- $X = \begin{bmatrix} (x^{(1)})^T \\ (x^{(2)})^T \\ \vdots \\ (x^{(m)})^T \end{bmatrix}$ $y = \begin{bmatrix} y^{(1)} \\ y^{(2)} \\ \vdots \\ y^{(m)} \end{bmatrix}$

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

Normal Equation Solution

- $\theta = (X^T X)^{-1} X^T y$
- $(X^T X)^{-1}$ is the inverse of $(X^T X)$
- Deriving the Normal Equation using matrix calculus …
  - https://ayearofai.com/rohan-3-deriving-the-normal-equation-using-matrix-calculus-1a1b16f65dda
- What if $(X^T X)^{-1}$ is noninvertible?
- The common causes might be having:
  - Redundant features, where two features are very closely related (i.e., they are linearly dependent).
  - Too many features (e.g., $m \leq n$). In this case, delete some features or use "regularization".

Gradient Descent x Normal Equation

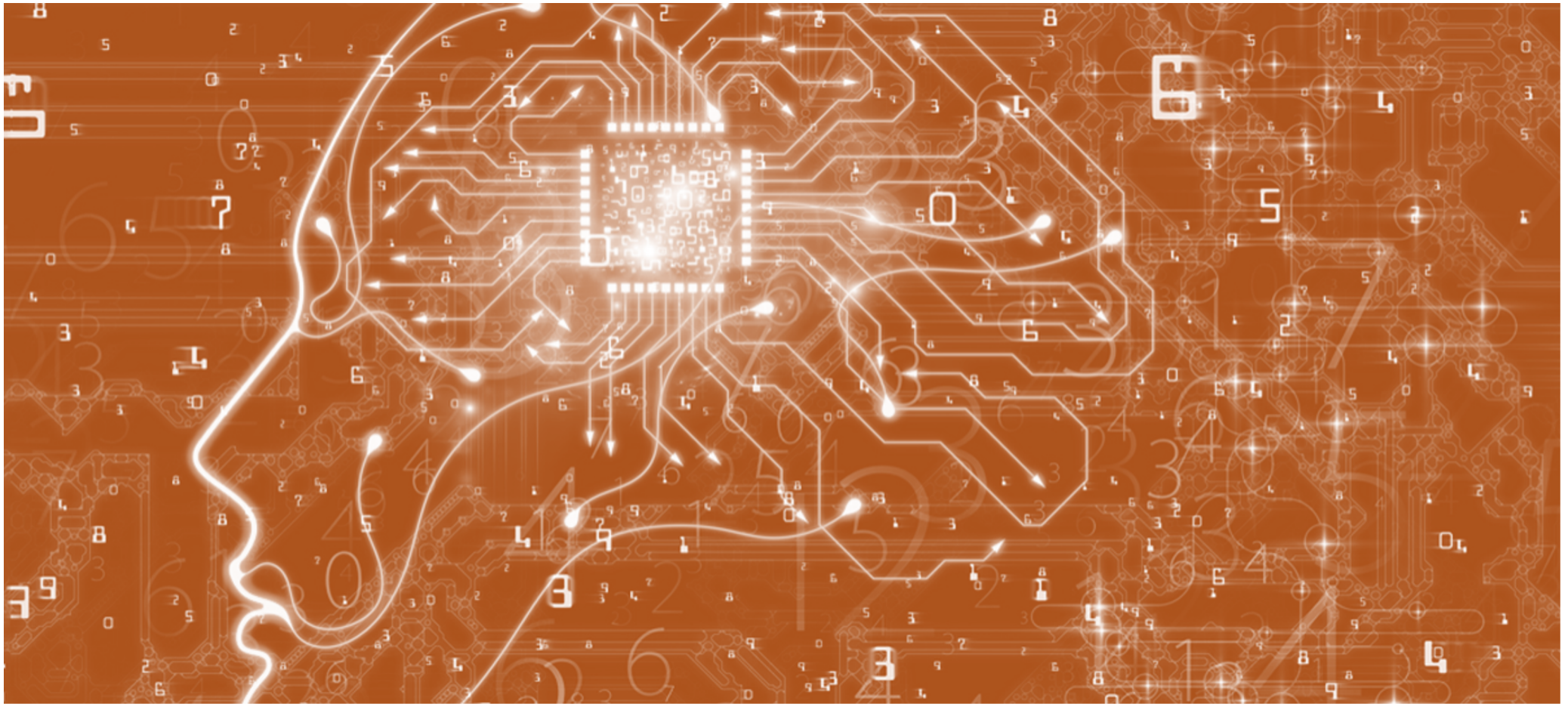
Gradient Descent

- 😞 Need to choose α.
- 😞 Needs many iterations.
- 😃 Works well even when n is large.

Normal Equation

- 😃 No need to choose α.
- 😃 Don't need to iterate.
- 😞 Need to compute $(X^T X)^{-1} \rightarrow O(n^3)$
- 😞 Slow if n is very large.

m examples and n features



Regression evaluation

Summary

- Introduction to approximation errors
  - Regression evaluation indexes:
    - Mean Absolute Error (MAE)
    - Mean Squared Error (MSE)
    - Root Mean Squared Error (RMSE)
    - Relative Absolute Error (RAE)
    - Relative Squared Error (RSE)
    - R-Squared (R^2)

Approximation errors

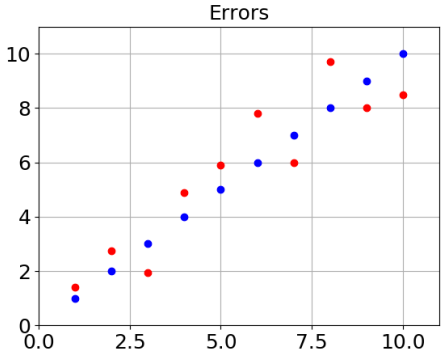
- **Approximation error** is the discrepancy between a value and some approximation to it
- Main strategies employed to measure the error:
  - **Absolute x relative error:**
    - **Absolute error**: difference between true and measured values. Typically, its range has no upper bound. In this way, it is not suitable for comparing different data or sets with different sizes
    - **Relative error**: error expressed as a ratio between true and measured values. Its typical range is from 0 to 1. Better to compare error between different data
  - **Abs (modulus):** usually error measures must be indifferent to the direction of the error (that is, positive and negative errors cannot cancel each other). In this way, the **abs** error is usually adopted (except when value is squared)
  - **Squared errors** are usually very sensitive to outliers. They are also indifferent to the error direction, and they have the advantage of being differentiable everywhere (**abs** is also not differentiable at 0)

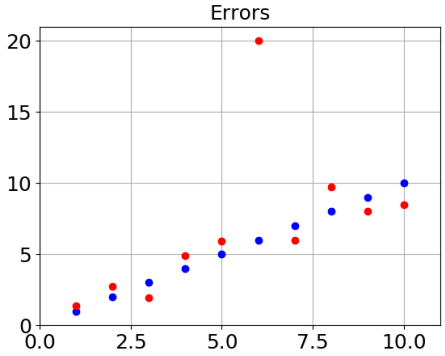
Regression evaluation

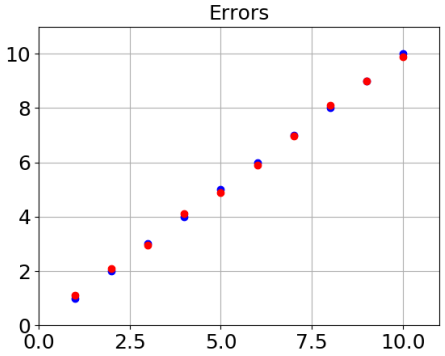
	Name	Formula	Min value	Max value	Desirable value	Notes
MAE	Mean absolute error	$\frac{1}{n}\sum_{j=1}^{n}\lvert\hat{y}_j - y_j\rvert$	0	∞	min	Measured in the same unit as the data with similar magnitude. Error cannot be compared in different data
MSE	Mean squared error	$\frac{1}{n}\sum_{j=1}^{n}(\hat{y}_j - y_j)^2$	0	∞	min	More sensible to outliers than MAE
RMSE	Root mean squared error	$\sqrt{\frac{1}{n}\sum_{j=1}^{n}(\hat{y}_j - y_j)^2}$	0	∞	min	Less sensitive than MSE to outliers
RAE	Relative absolute error	$\frac{\sum_{j=1}^{n}\lvert\hat{y}_j - y_j\rvert}{\sum_{j=1}^{n}\lvert y_j - \bar{y}\rvert}$	0	∞	min	Ratio. More suitable to compare to other data. Less sensible to outliers
RSE	Relative squared error	$\frac{\sum_{j=1}^{n}(\hat{y}_j - y_j)^2}{\sum_{j=1}^{n}(y_j - \bar{y})^2}$	0	∞	min	Ratio. More suitable to compare to other models. More sensible to outliers
R^2	Coefficient of determination	$1 - RSE$	-∞	1	1	Usually in the range [0 1]. Cannot determine whether prediction is biased

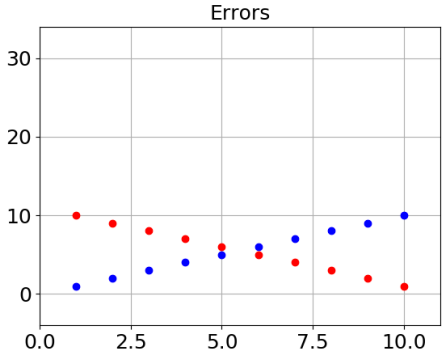
y: real value; $\hat{y}$: predicted values; $\bar{y}$: mean of real values.

Regression evaluation

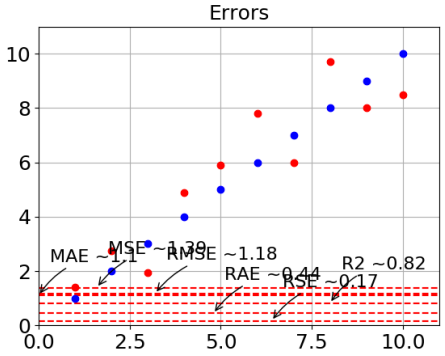




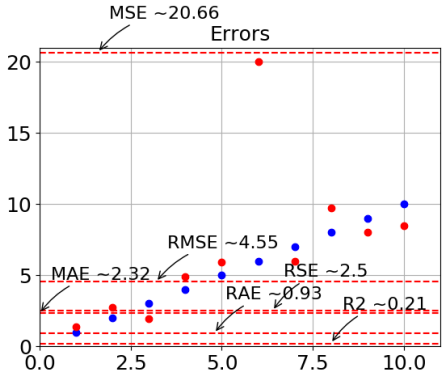




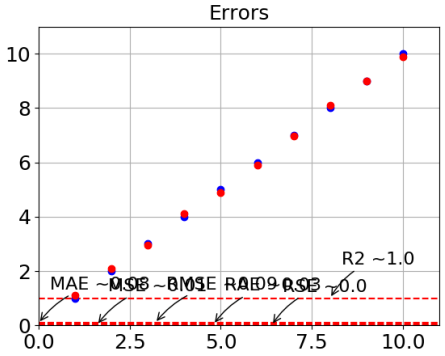
Regression evaluation



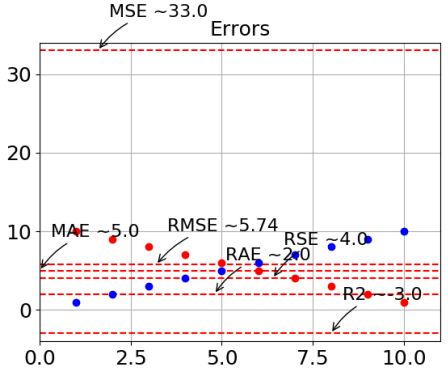
MAE: 1.1
MSE: 1.39
RMSE: 1.18
RAE: 0.44
RSE: 0.18
R2: 0.82



MAE: 2.32
MSE: 20.66
RMSE: 4.55
RAE: 0.93
RSE: 0.74
R2: 0.21



MAE: 0.08
MSE: 0.01
RMSE: 0.09
RAE: 0.03
RSE: 0.0
R2: 1.0



MAE: 5.0
MSE: 33.0
RMSE: 5.74
RAE: 2.0
RSE: 4.0
R2: -3.0

References

Lecture 6

- MARSLAND, S. Machine Learning: an algorithm perspective. CRC Press, 2nd edition, 2015.
- https://www.coursera.org/learn/machine-learning , Week 1 & 2
- NG, Y. A. https://www.coursera.org/learn/machine-learning. Logistic regression.
- AVILA, S. Linear and Logistic Regression. Institute of Computing (IC/Unicamp)
- ALPAYDIN, E. Introduction to Machine Learning. MIT Press, 3rd edition, 2014.
- SILVA, I. N.; SPATTI, D. H.; FLAUZINO, R. A. Redes Neurais Artificiais para engenharia e ciências aplicadas. Curso prático. Artliber Editora, 2010.
- SUTTON, R. S.; BARTO, A. G. Reinforcement learning: an introduction. MIT Press, Cambridge, MA, 2nd edition in progress, 2017.

This material is part of the Machine Learning Course

By Esther Colombini and Alexandre Simões