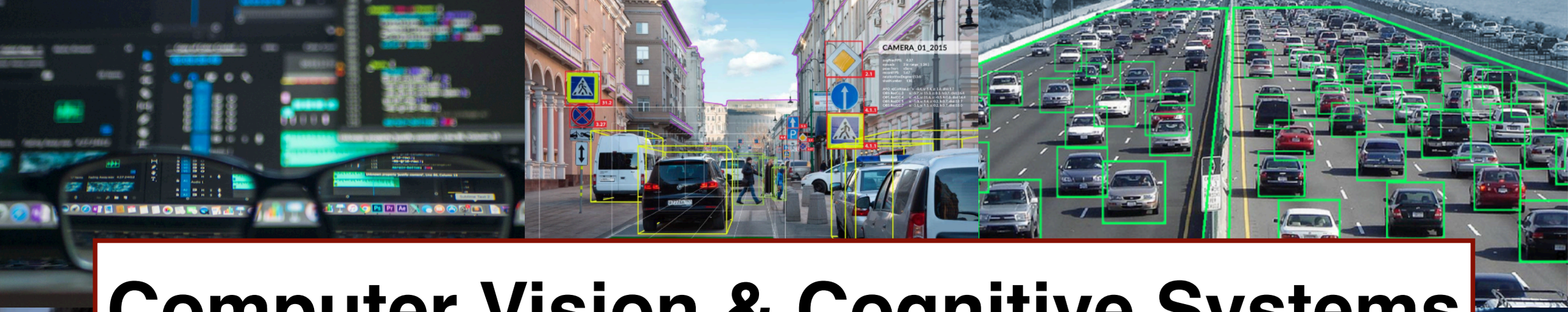


Computer Vision & Cognitive Systems

SCQ5109806 - LM CS,DS,CYB,PD

ML Basics: Linear Models and Gradient Descent

Prof. Lamberto Ballan

University of Padova - A.Y. 2025/26 October 6, 2025

DIPARTIMENTO MATEMATICA
Department of Mathematics “Tullio Levi-Civita”

Supervised Learning

Linear Regression

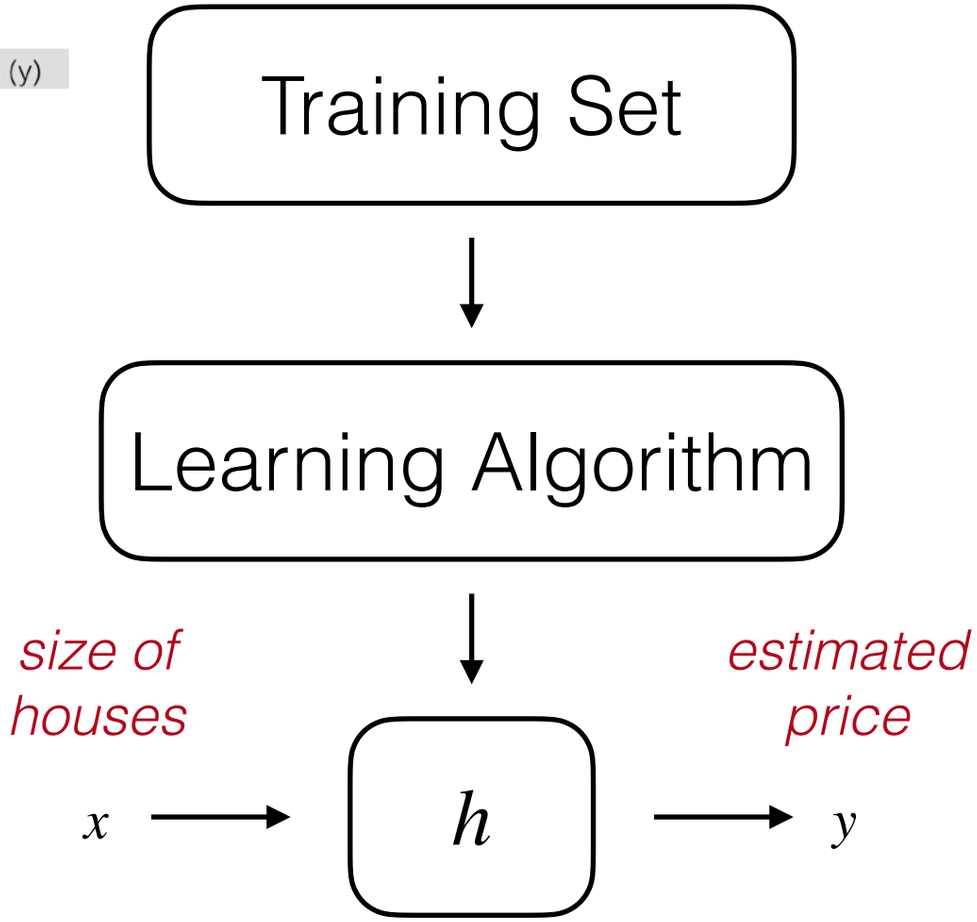
- **Example:** housing prices in Portland (OR)
  - Notation:
    - m = Number of training examples
    - x's = "input" variable / features
    - y's = "output" variable / target variable

Size in feet^2 (x)	Price ($) in 1K's (y)
2104	460
1416	232
1534	315
852	178
…	…

Linear Regression

$\{(x^{(i)}, y^{(i)})\}$

Size in feet² (x)	Price ($) in 1K's (y)
2104	460
1416	232
1534	315
852	178
…	…



h maps from x's to y's

How do we represent h ?

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

Univariate linear regression model *(i.e. one variable)*

Linear Regression

- Training set:

Size in feet^2 (x)	Price ($) in 1K's (y)
2104	460
1416	232
1534	315
852	178
…	…

- Hypothesis: $h_\theta(x) = \theta_0 + \theta_1 x$

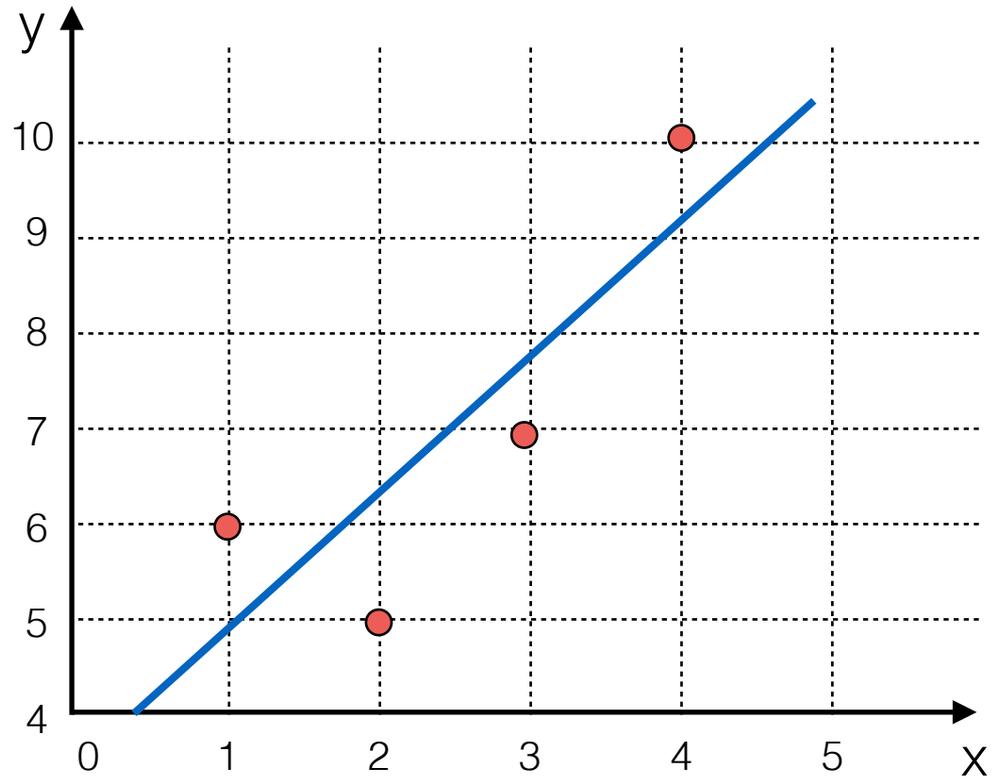
θ_i *'s parameters*

- How to chose the parameters?

Linear Regression

- Let's start with a simple example:

x	y
1	6
2	5
3	7
4	10



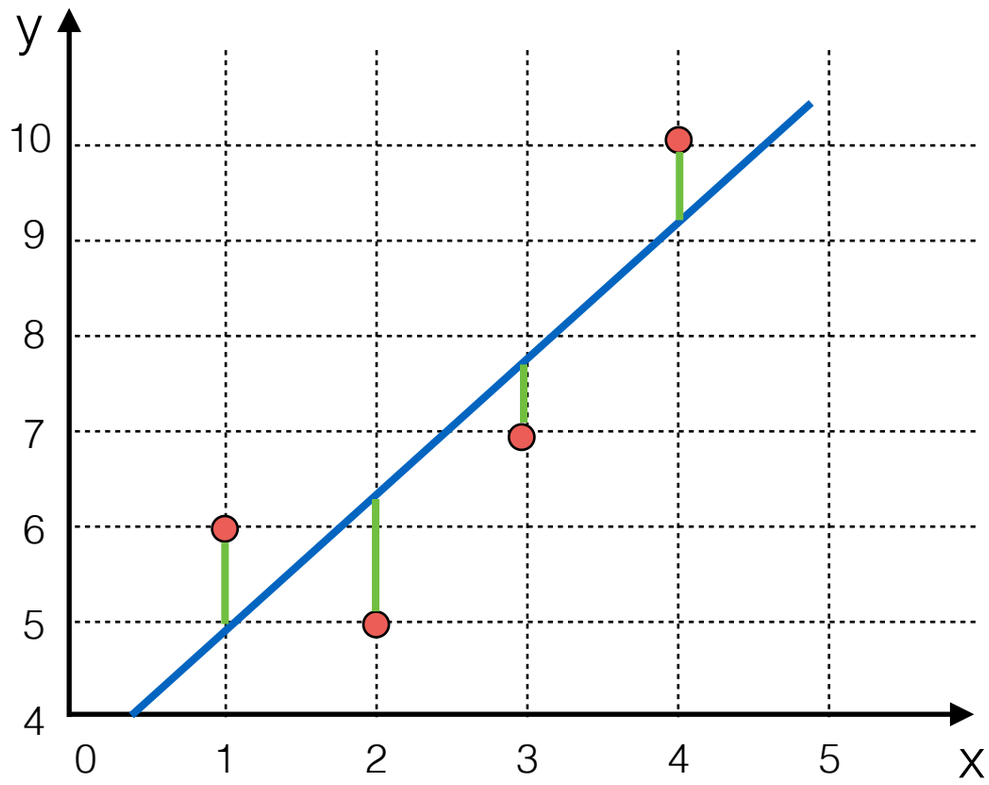
$$h_\theta(\mathrm{x}) = \theta_0 + \theta_1 \, \mathrm{x}$$

- Idea: choose θ_0, θ_1 so that $h_\theta(\mathrm{x})$ is close to y for our training examples $\{(x^{(\mathrm{i})}, y^{(\mathrm{i})})\}$

Linear Regression

m training examples

x	y
1	6
2	5
3	7
4	10



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

- Idea: choose θ_0, θ_1 so that $h_\theta(x)$ is close to y for our training examples $\{(x^{(i)}, y^{(i)})\}$

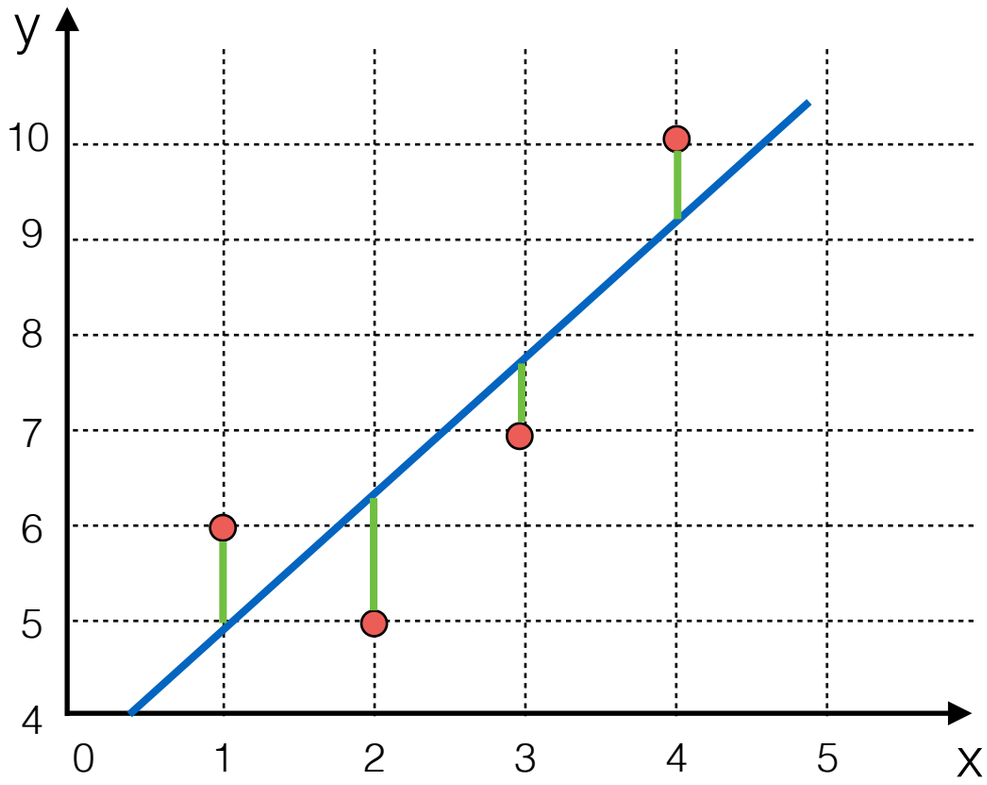
- Least squares: sum of squared distances (residuals)

Cost Function - Intuition

m training examples

x	y
1	6
2	5
3	7
4	10

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



- Residuals: $\{h_\theta(x^{(i)}) - y^{(i)}\}$

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

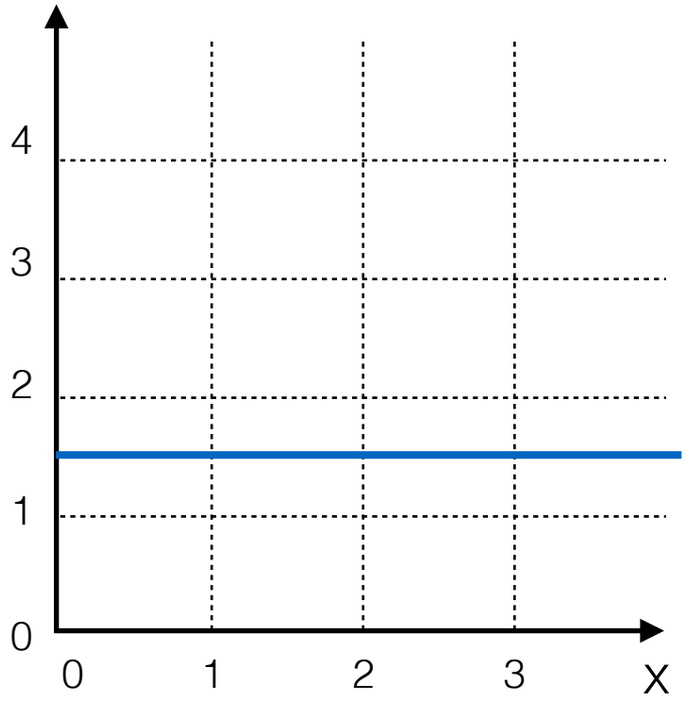
- Objective: $\underset{\theta_0,\theta_1}{\text{minimize}}\ J(\theta_0, \theta_1)$

Cost Function - Intuition

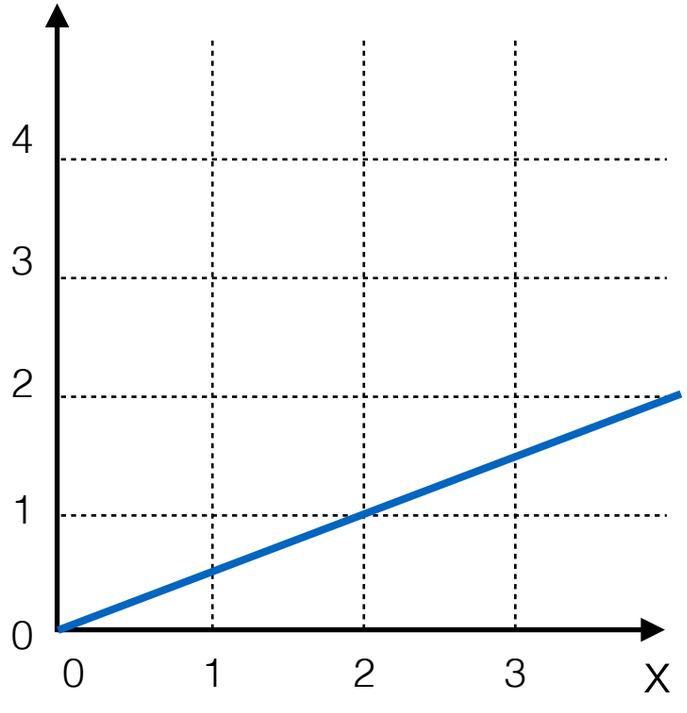
- How to choose θ_i's? *Let's see a few examples…*

$h_\theta(x) = \theta_0 + \theta_1 x$ — *hypothesis*

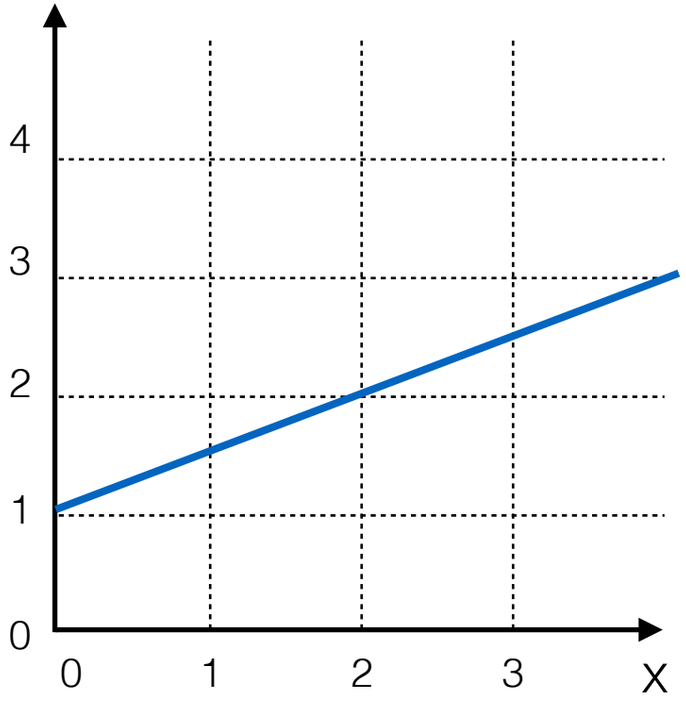
θ_0, θ_1 — *parameters*



$\theta_0=3/2, \theta_1=0$



$\theta_0=0, \theta_1=1/2$



$\theta_0=1, \theta_1=1/2$

Cost Function - Intuition

Simplified version

- Hypothesis:

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

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

- Parameters: θ_0, θ_1

$$\theta_1$$

- Cost Function:

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

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

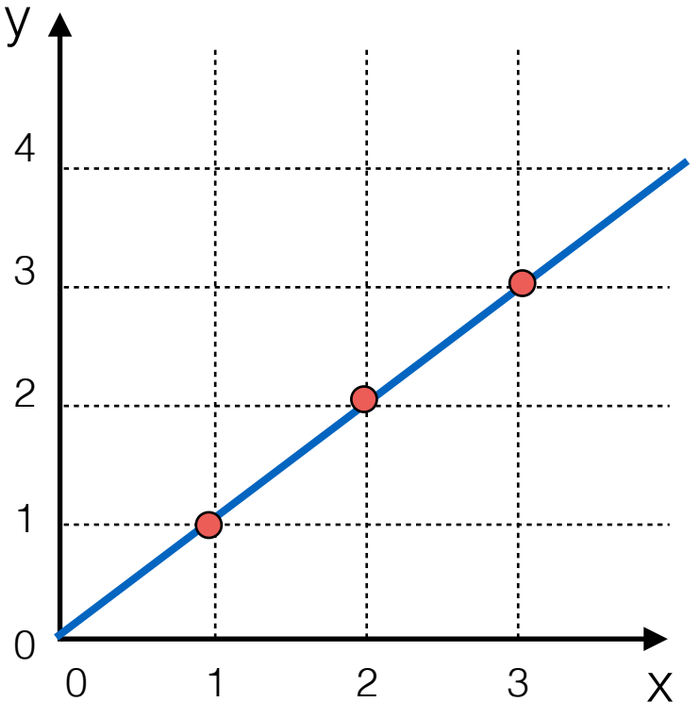
- Objective: $\underset{\theta_0,\theta_1}{\text{minimize}}\ J(\theta_0, \theta_1)$

$$\underset{\theta_1}{\text{minimize}}\ J(\theta_1)$$

Cost Function - Intuition

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

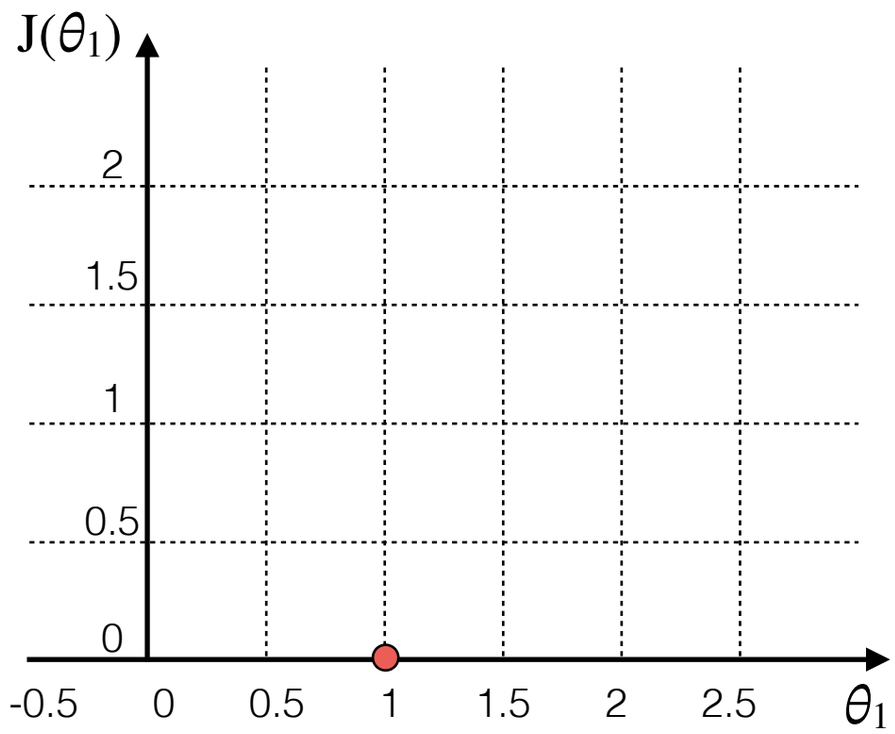
(for fixed θ_1 this is a function of x)



$\theta_1=1$

$J(\theta_1)$

(function of the parameter θ_1)



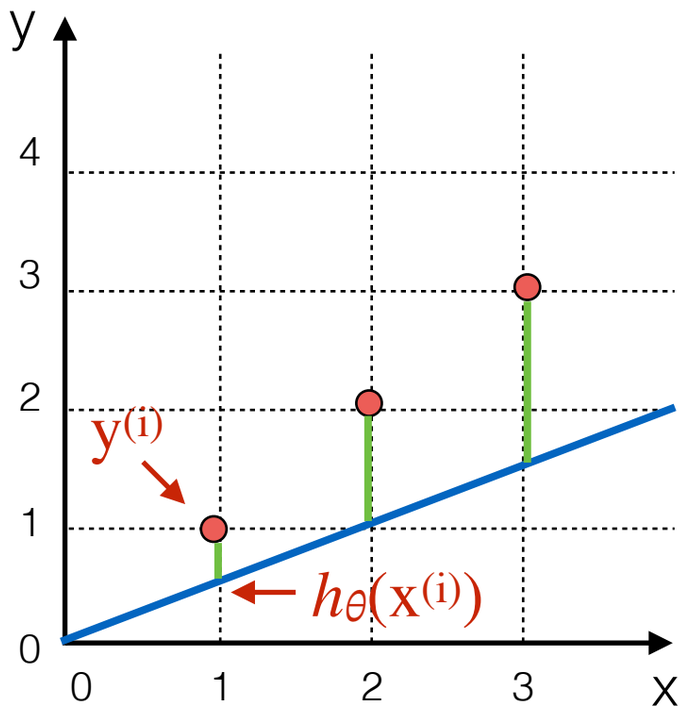
$$J(\theta_1) = \frac{1}{2m} \sum_{i=1}^{m} (h_\theta(x^{(i)}) - y^{(i)})^2 = \frac{1}{2m} \sum_{i=1}^{m} (\theta_1 \, x^{(i)} - y^{(i)})^2 = \frac{1}{2m} (0+0+0)^2$$

$\theta_1=1$

Cost Function - Intuition

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

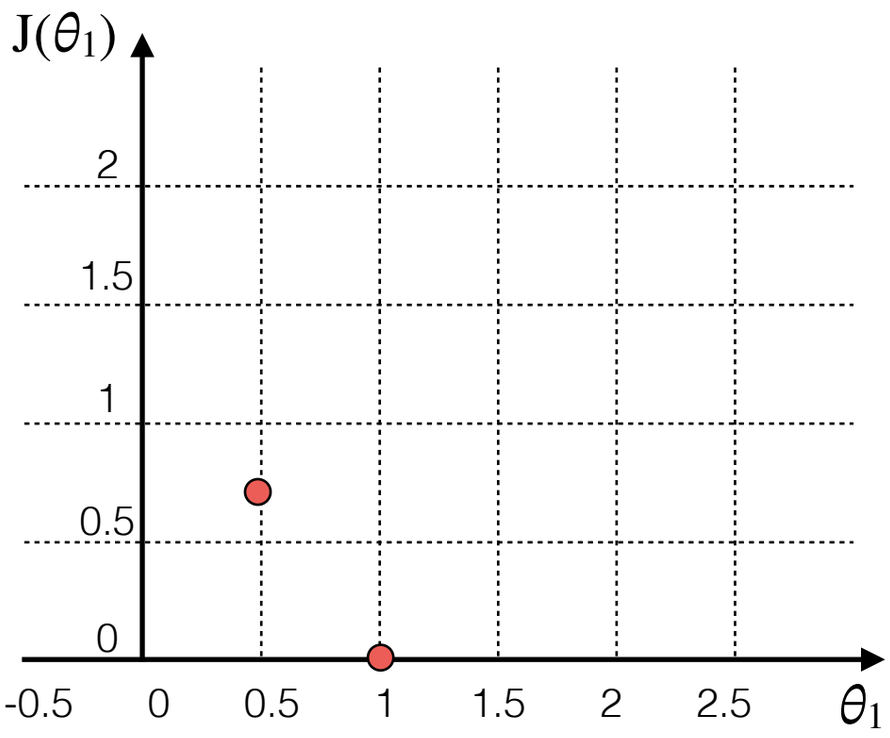
(for fixed θ_1 this is a function of x)



θ_1=1/2 ?

$J(\theta_1)$

(function of the parameter θ_1)

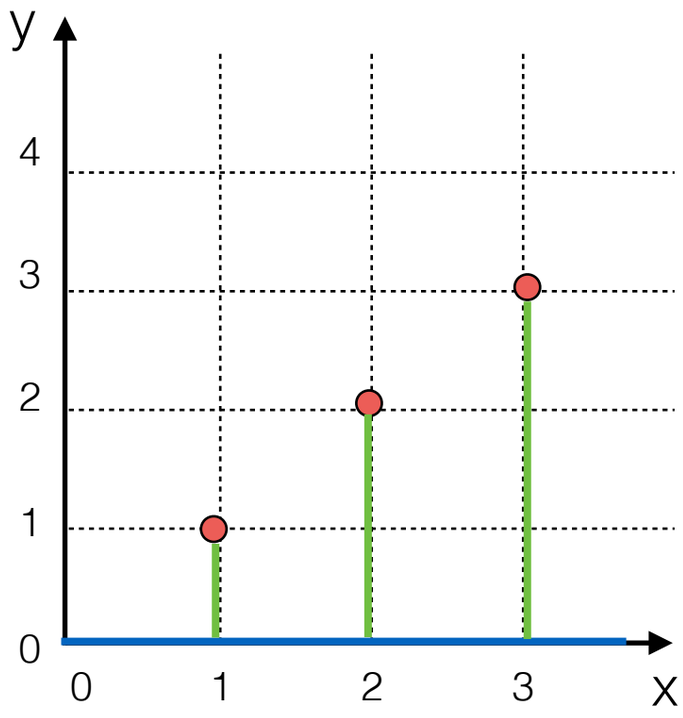


$$J(0.5) = \frac{1}{2m}[(0.5-1)^2 + (1-2)^2 + (1.5-3)^2] = \frac{1}{6}[0.25+1+2.25] \approx 0.67$$

Cost Function - Intuition

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

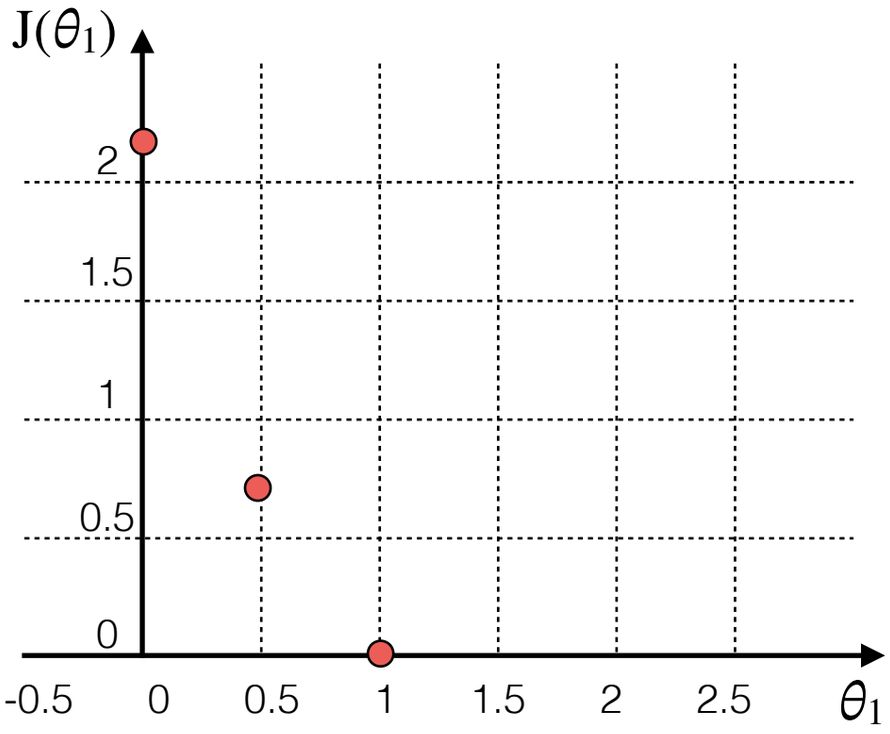
(for fixed θ_1 this is a function of x)



$\theta_1=0$?

$J(\theta_1)$

(function of the parameter θ_1)

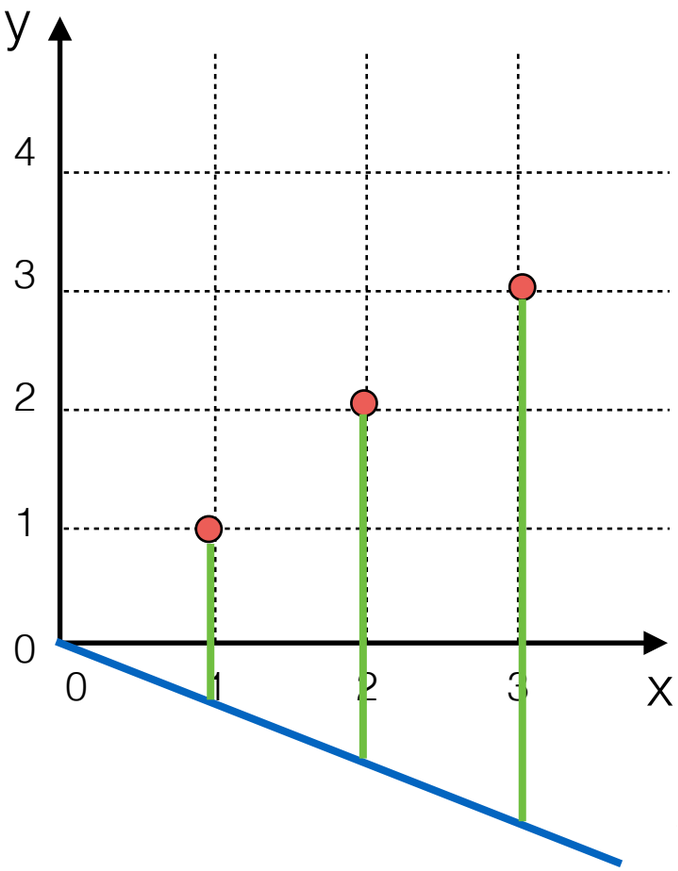


$$J(0) = \frac{1}{2m}[(0-1)^2 + (0-2)^2 + (0-3)^2] = \frac{1}{6}[1+4+9] \approx 2.33$$

Cost Function - Intuition

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

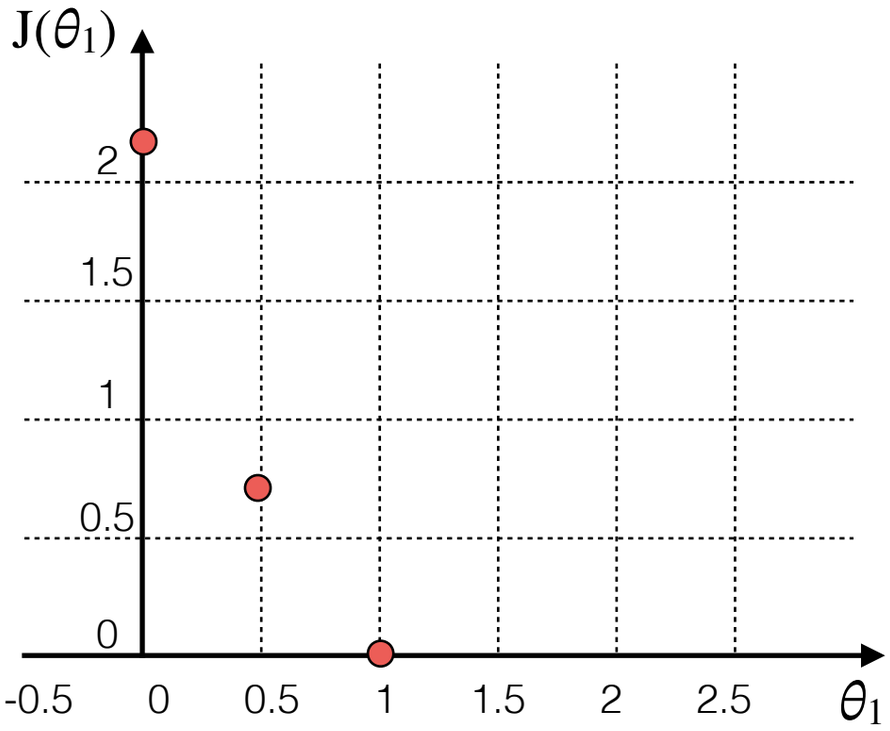
(for fixed θ_1 this is a function of x)



θ_1=-1/2 ?

$J(\theta_1)$

(function of the parameter θ_1)

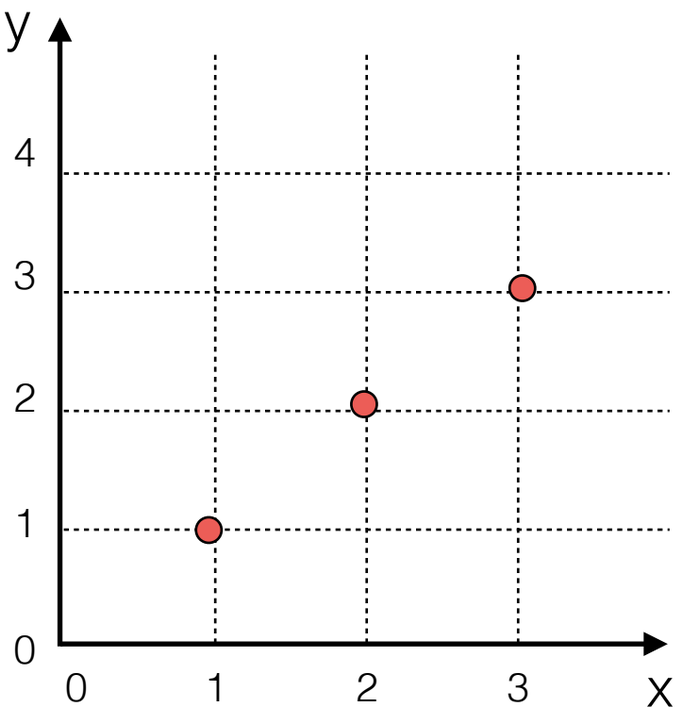


$$J(-0.5) = \frac{1}{2m}\left[(-0.5-1)^2+(-1-2)^2+(-1.5-3)^2\right] \approx 5.25$$

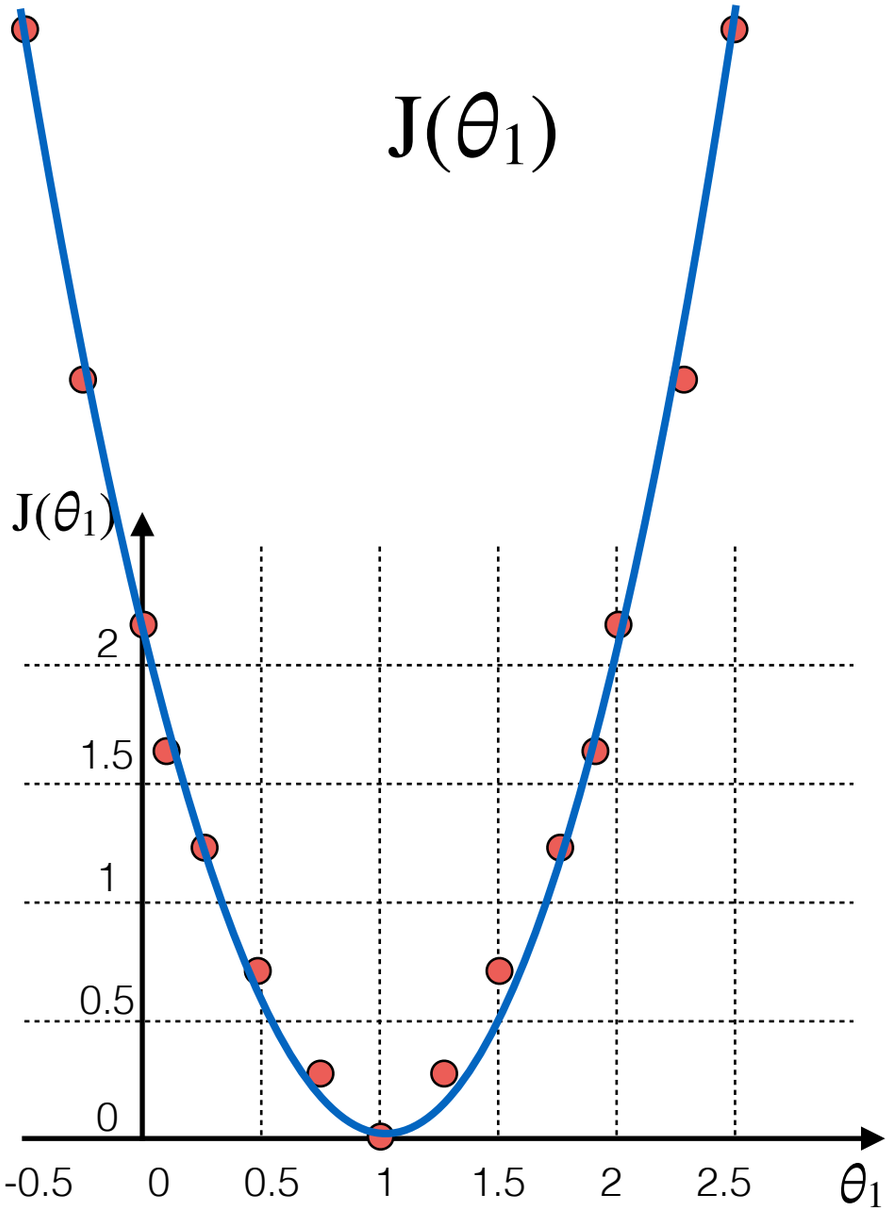
Cost Function - Intuition

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

(for fixed θ_1 this is a function of x)



θ_1= ?

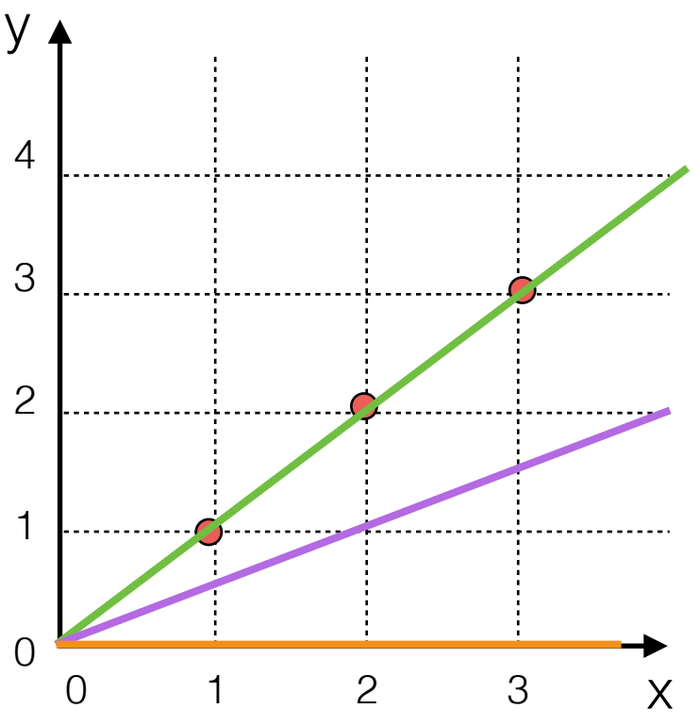


So, for different values, $J(\theta_1) = \ldots$

Cost Function - Intuition

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

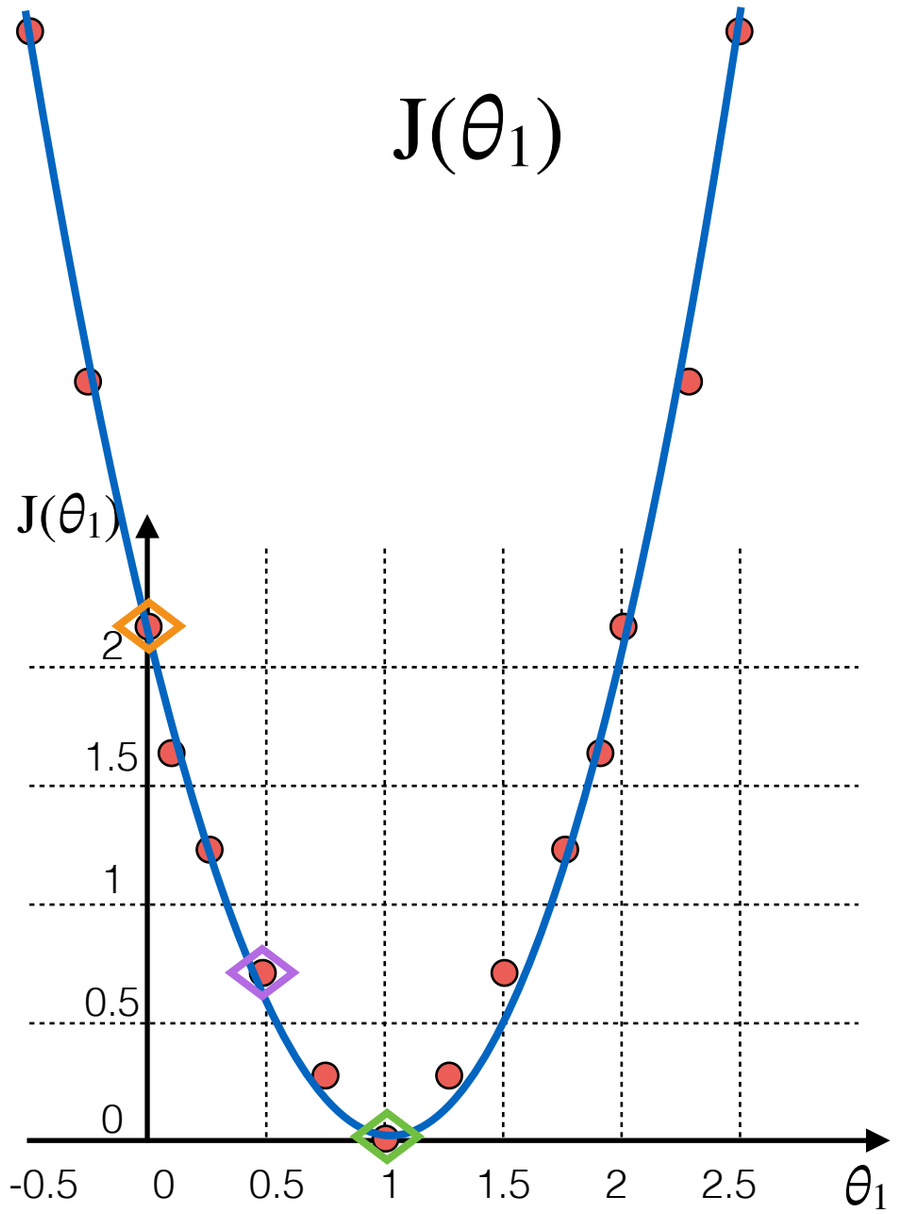
(for fixed θ_1 this is a function of x)



$\theta_1 = 1$

$\theta_1 = 1/2$

$\theta_1 = 0$



- To recap, each value of θ_1 corresponds to a different hypothesis (i.e. a different line through the data)

Cost Function - Intuition

- Hypothesis:

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

- Parameters: θ_0, θ_1

- Cost Function:

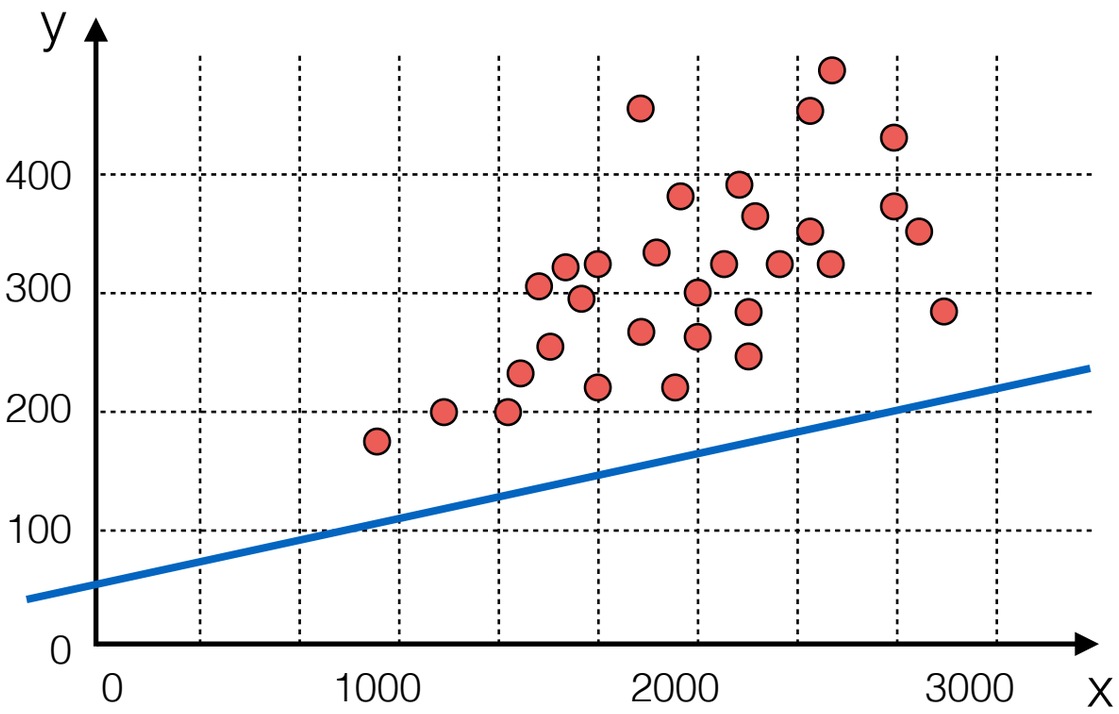
$$J(\theta_0, \theta_1) = \frac{1}{2m} \sum_{i=1}^{m} (h_\theta(x^{(i)}) - y^{(i)})^2$$

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

Cost Function - Intuition

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

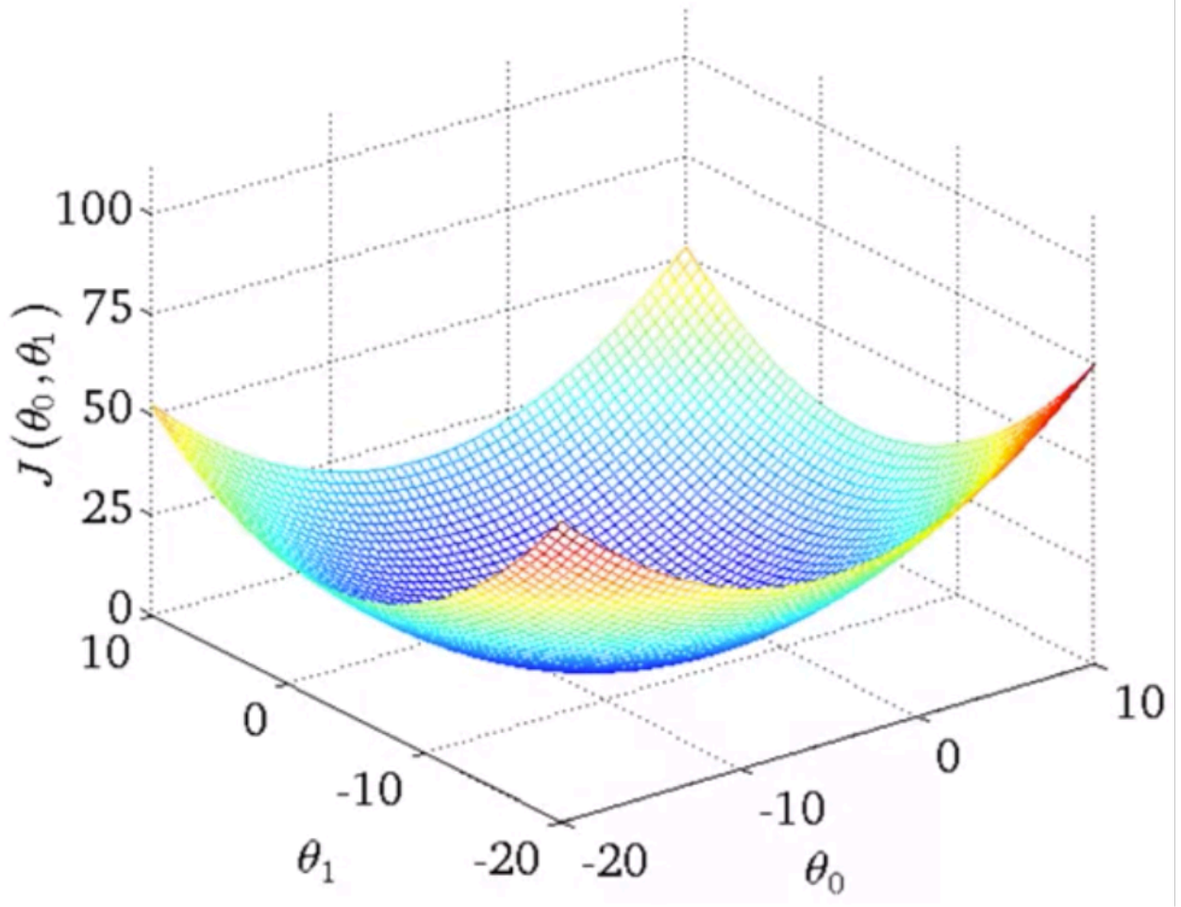
(for fixed θ_0, θ_1 this is a function of x)



$\theta_0 = 50, \theta_1 = 0.06$

$J(\theta_0, \theta_1)$

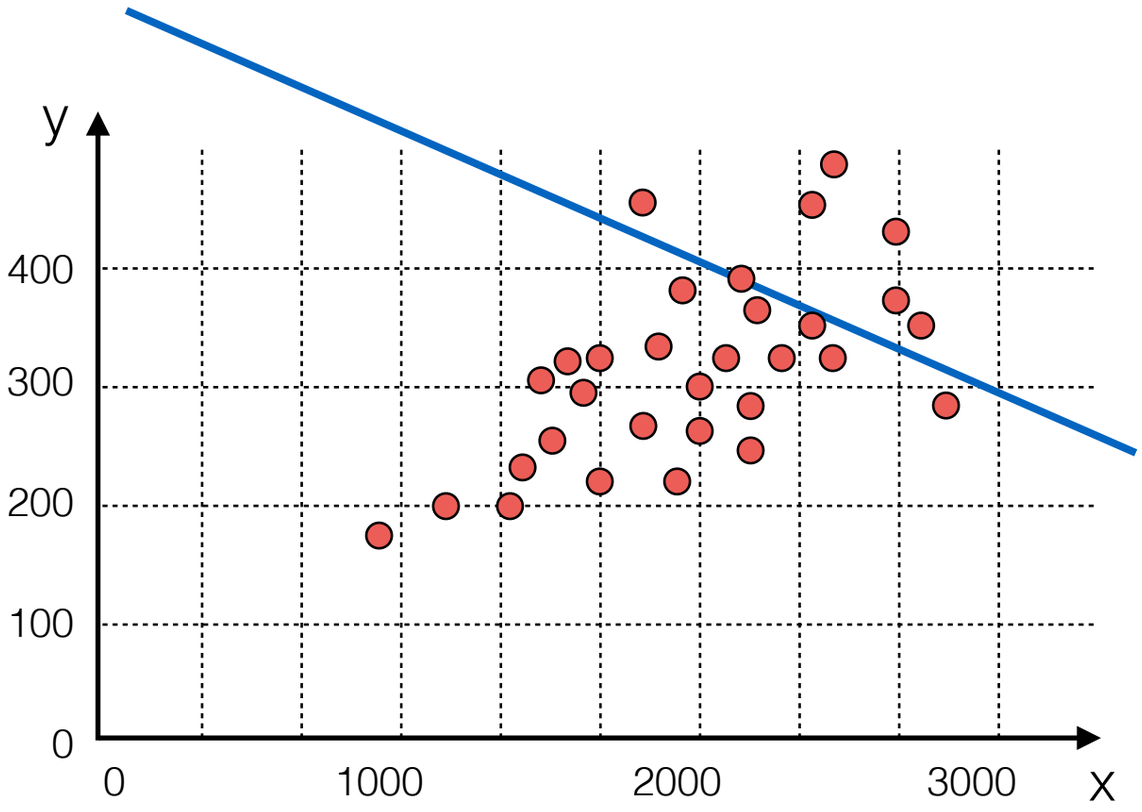
(function of the parameters θ_0, θ_1)



Cost Function - Intuition

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

(for fixed θ_0,θ_1 this is a function of x)

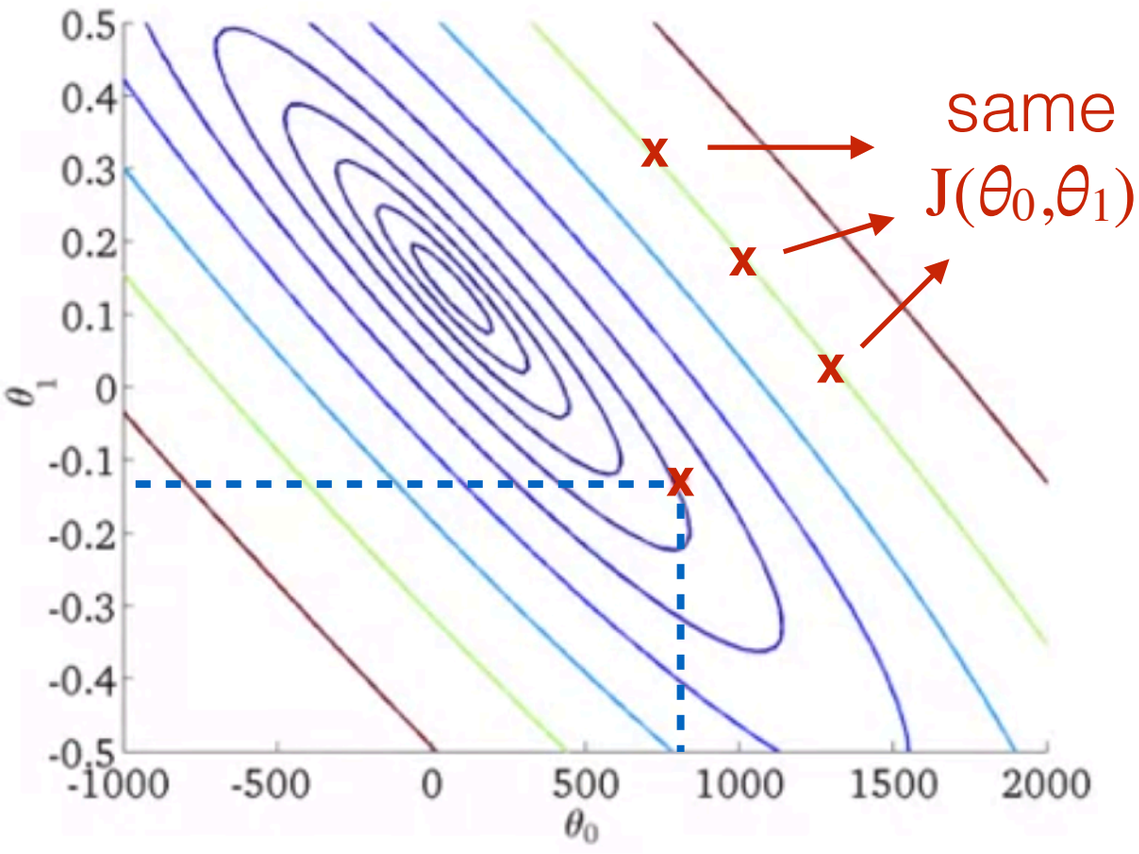


Training data

Current hypothesis

$J(\theta_0,\theta_1)$

(function of the parameters θ_0,θ_1)

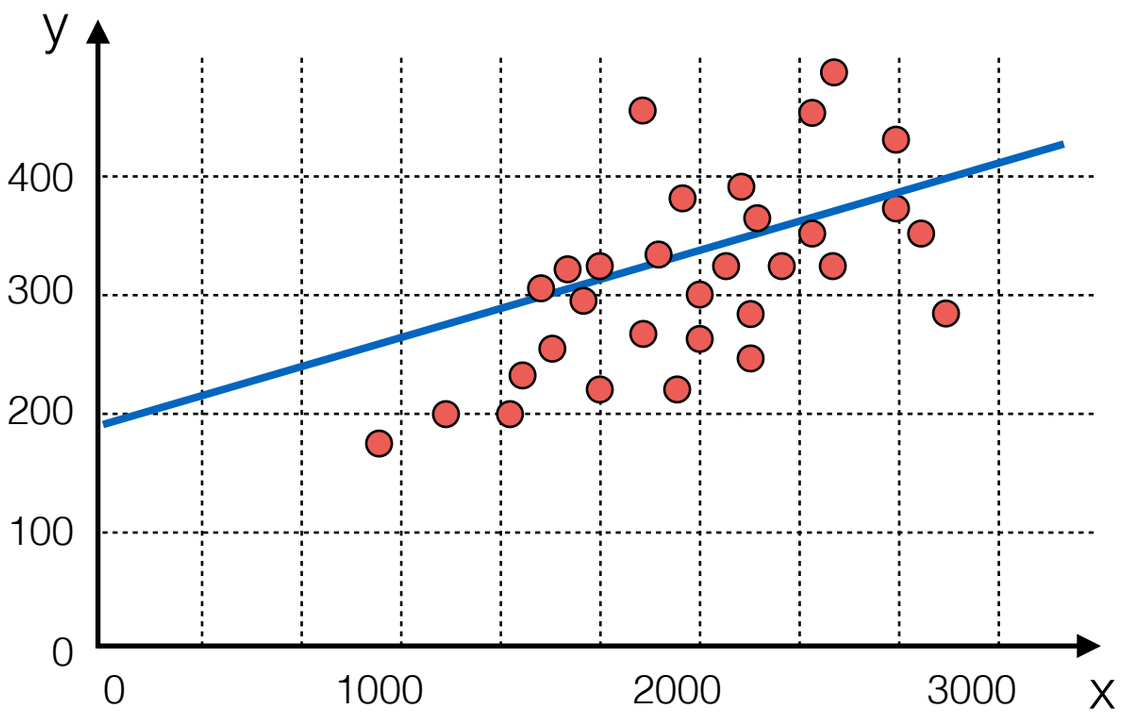


Contour plot

Cost Function - Intuition

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

(for fixed θ_0,θ_1 this is a function of x)

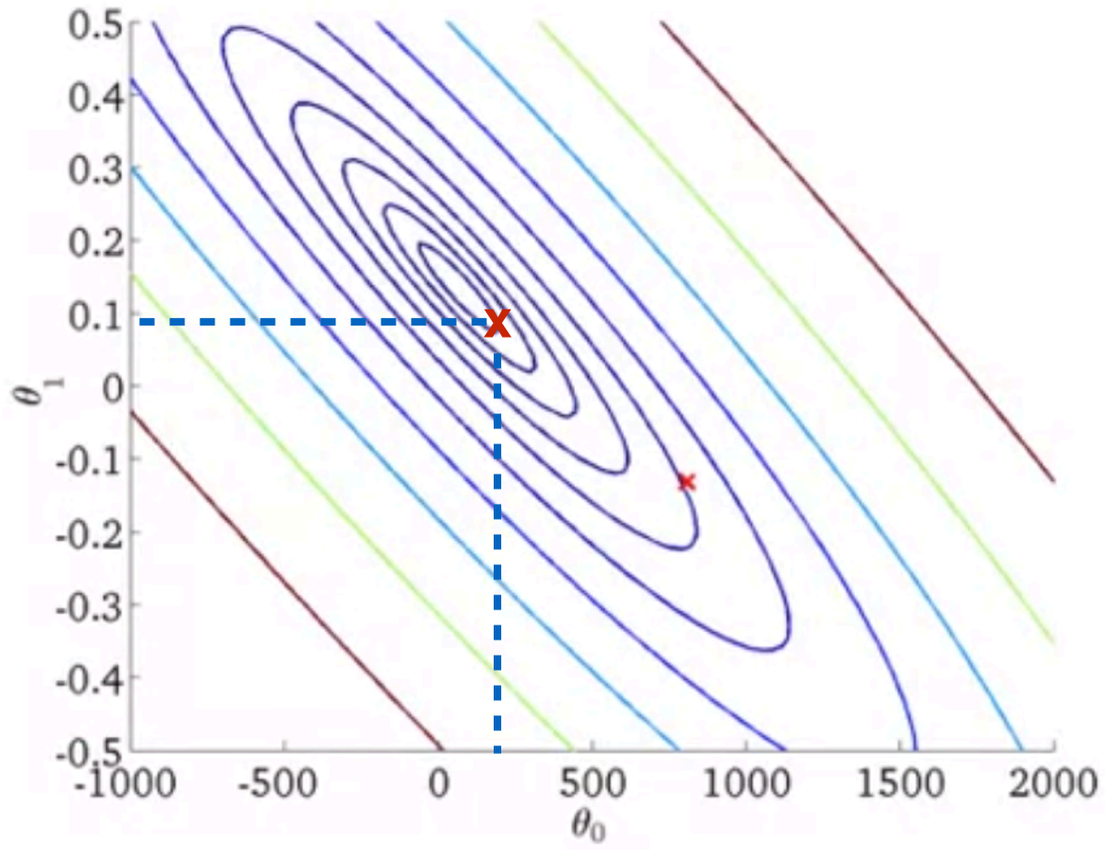


Training data

Current hypothesis

$J(\theta_0,\theta_1)$

(function of the parameters θ_0,θ_1)



Contour plot

Parameter Learning

- What we really want is an efficient algorithm for automatically finding the value of θ_0 and θ_1

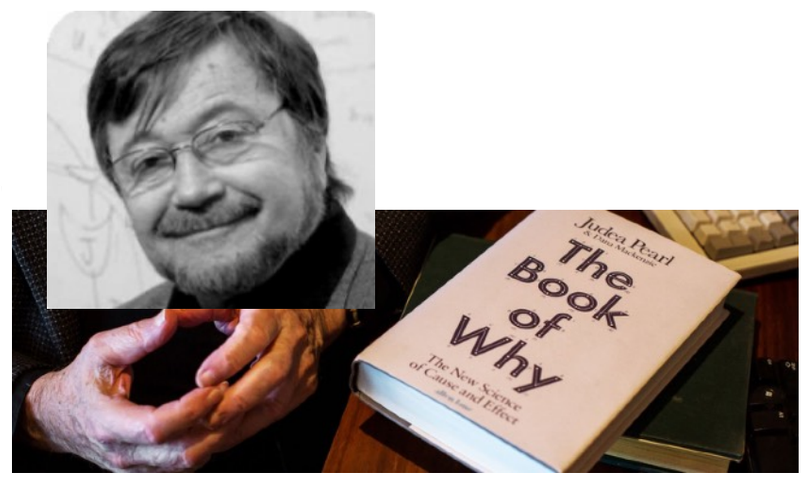
- In other words we need an efficient algorithm that minimizes the cost function $\mathbf{J}$

This is what usually machine learning is all about…

(… and not everyone is happy about that!)

"All the impressive achievements of deep learning amount to just curve fitting"

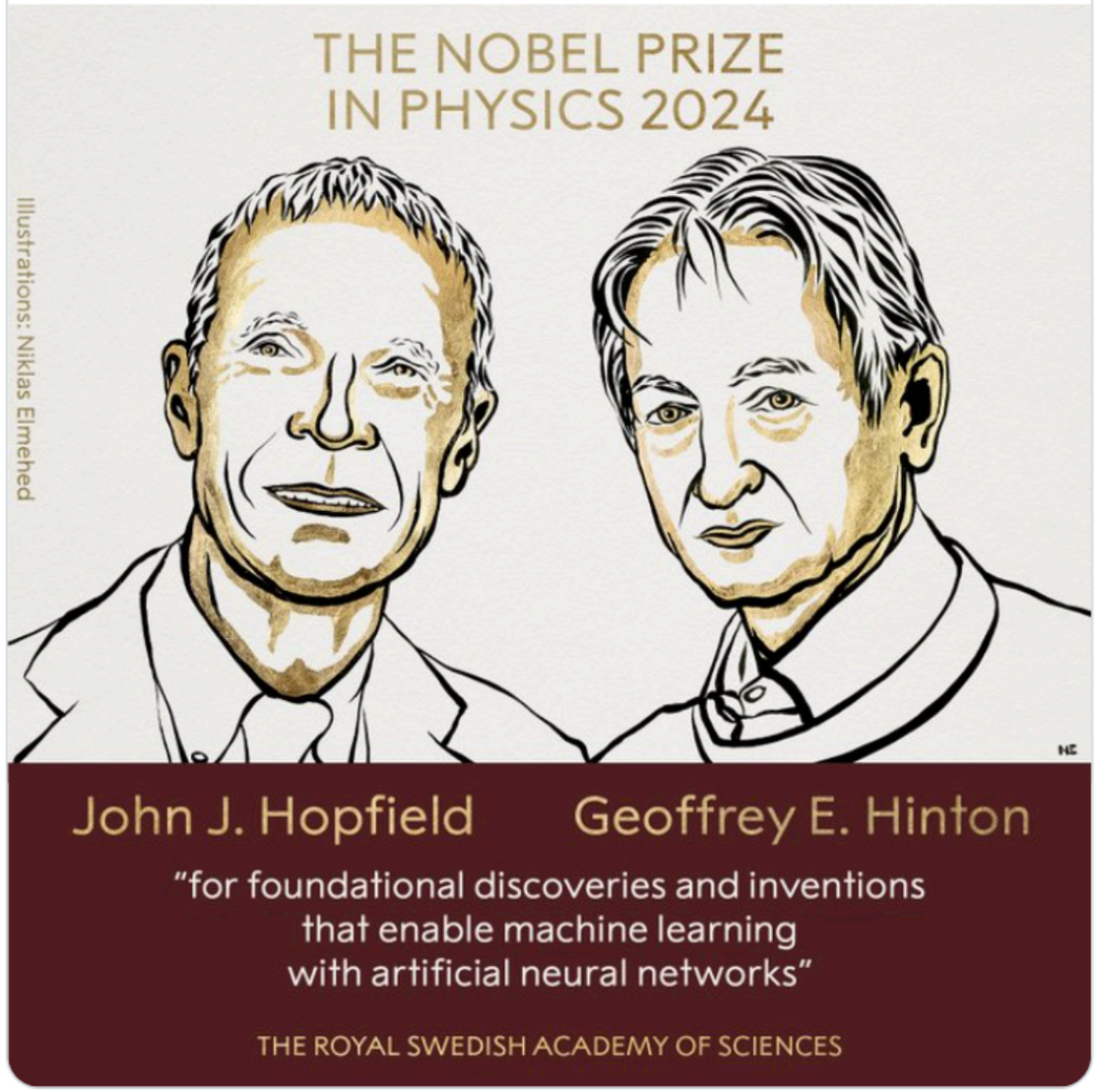
J. Pearl



Ok, but…

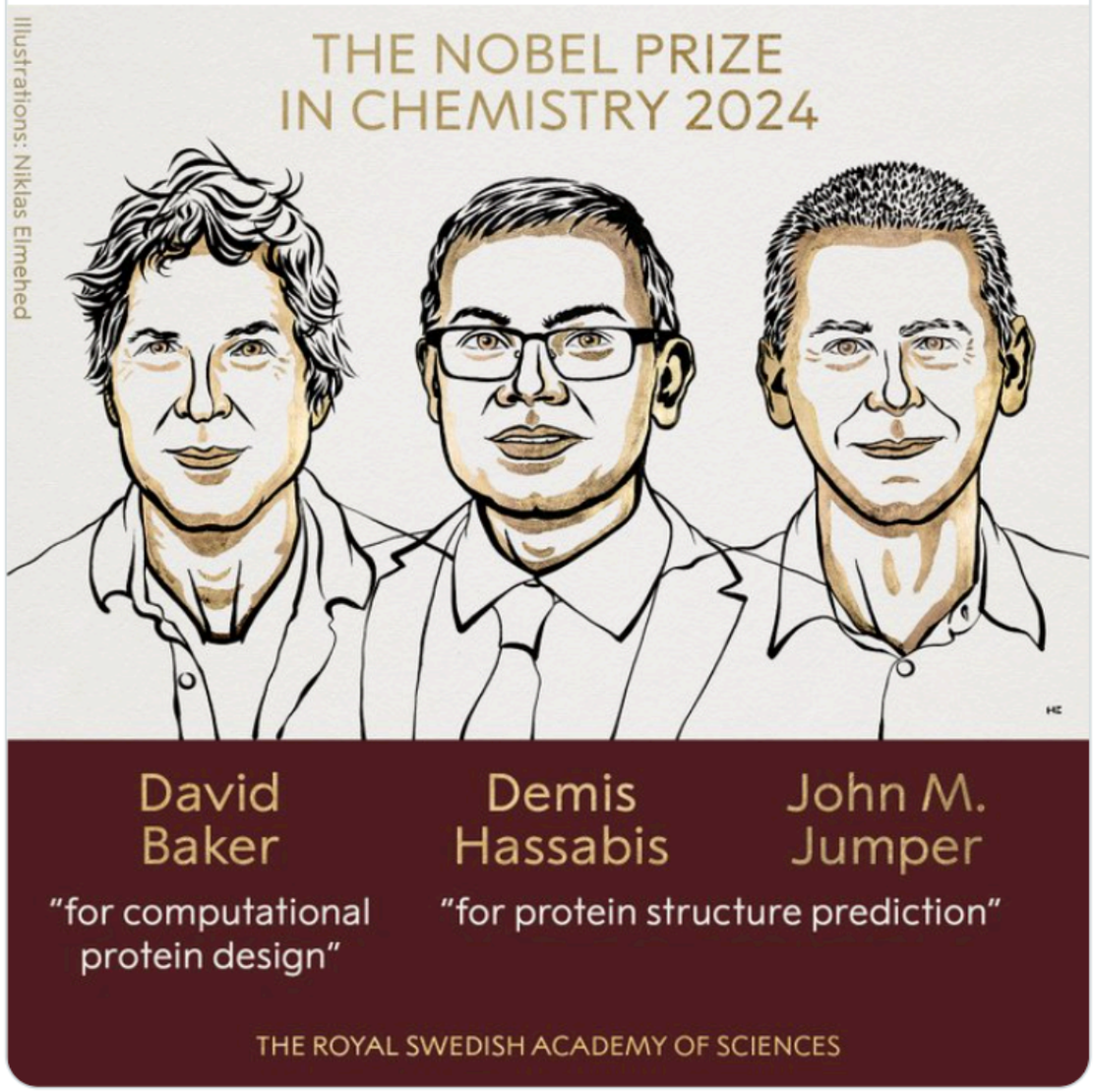
The Nobel Prize @NobelPrize · Oct 8

BREAKING NEWS
The Royal Swedish Academy of Sciences has decided to award the 2024 #NobelPrize in Physics to John J. Hopfield and Geoffrey E. Hinton "for foundational discoveries and inventions that enable machine learning with artificial neural networks."



The Nobel Prize @NobelPrize · Oct 9

BREAKING NEWS
The Royal Swedish Academy of Sciences has decided to award the 2024 #NobelPrize in Chemistry with one half to David Baker "for computational protein design" and the other half jointly to Demis Hassabis and John M. Jumper "for protein structure prediction."



. . .

Ok, but…


The Nobel Prize @NobelPrize · Oct 8
BREAKING NEWS
The Royal Swedish Academy of Sciences has
2024 #NobelPrize in Physics to John J. Hopfiel
“for foundational discoveries and inventions t
learning with artificial neural networks.”
THE NOBEL PR
IN PHYSICS 20
Illustrations: Niklas Elmehed
John J. Hopfield Geof
"for foundational discoveries a
that enable machine lea
with artificial neural net
THE ROYAL SWEDISH ACADEMY O
Bojan Tunguz @tunguz · Oct 9
OK, now they have REALLY gone too far.
THE NOBEL PRIZE
IN LITERATURE 2024
ChatGPT
"for his intricate tapestry of prose
which showcases the redundancy
of sentience in art"
THE SWEDISH ACADEMY
(Just kidding…)
…

Gradient Descent

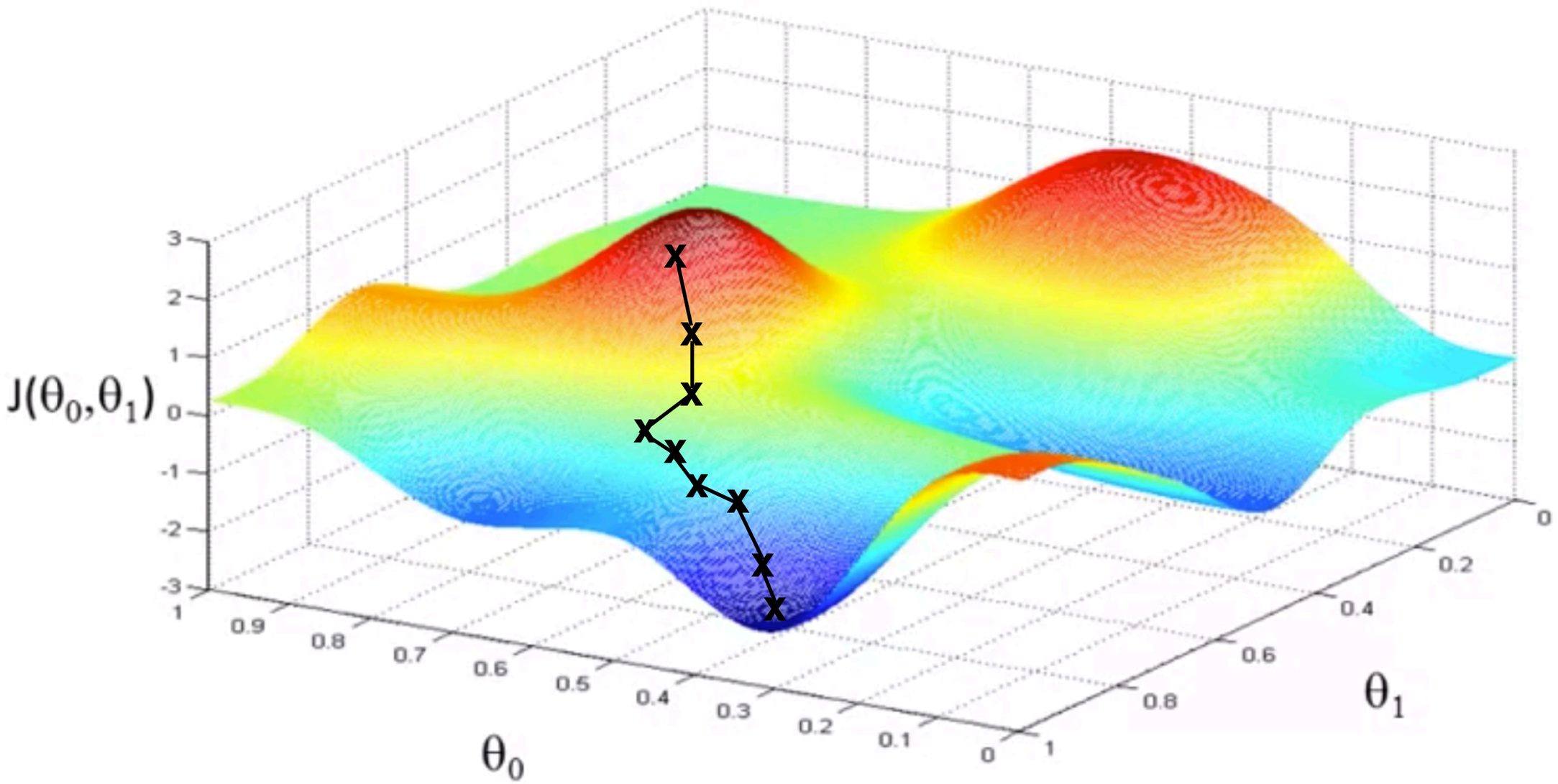
- You have a cost function $\mathbf{J}(\theta_0, \ldots, \theta_n)$
- Objective: $\underset{\theta_0, \ldots, \theta_n}{\text{minimize}}\ \mathbf{J}(\theta_0, \ldots, \theta_n)$
- Outline:
  - Start with some configuration of parameters θ_i's
  - Keep changing θ_i's parameters to reduce the cost function $\mathbf{J}$ until we (hopefully) end up at a minimum

Gradient Descent - Intuition

- “Fog in the mountains” analogy:
  - You are stuck in the mountains and visibility is extremely low
  - Look at the steepness of the hill at you current position and proceed in the direction with the steepest descent

Gradient Descent - Intuition

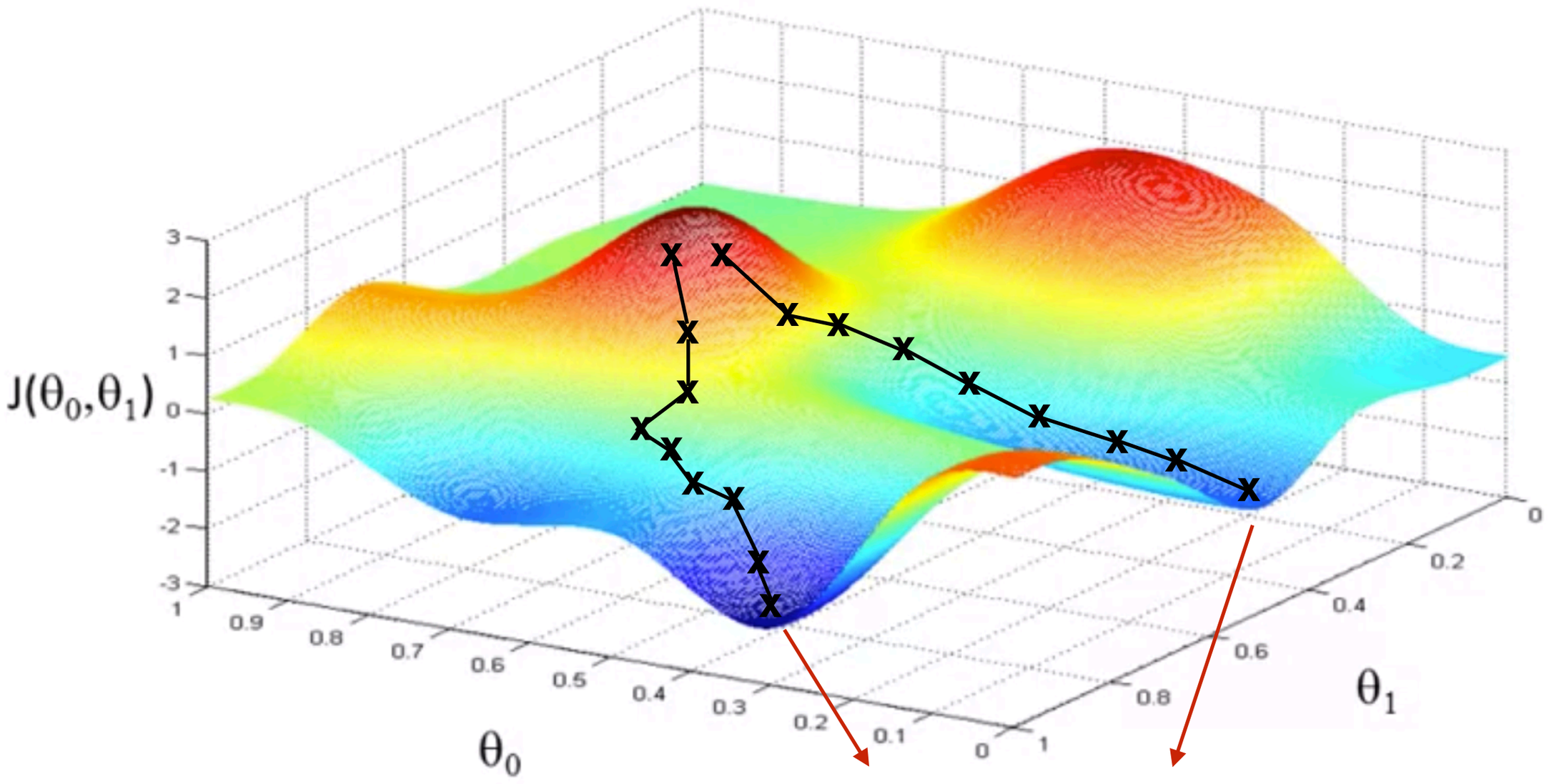
- Let's start with a visualization



Adapted from slide by A. Ng

Gradient Descent - Intuition

- Let's start with a visualization



Gradient Descent

- Given $\mathbf{J}(\theta_0, \ldots, \theta_n)$, we take a small step in the direction of the negative gradient, so that:

$$\boldsymbol{\theta}_{k+1} = \boldsymbol{\theta}_k - \eta \, \nabla \mathbf{J}(\boldsymbol{\theta}_k) \qquad \text{where} \qquad \boldsymbol{\theta}=\{\theta_0, \ldots, \theta_n\}$$

  - The parameter $\eta > 0$ is known as the *learning rate*
  - After each update, the gradient is re-evaluated for the new weight vector $\boldsymbol{\theta}_{k+1}$, and the process repeated
  - Note: the cost function is computed on the entire training set in order to evaluate $\nabla \mathbf{J}$ (i.e. *batch* method)

Gradient Descent

- A simple implementation (using our example):

repeat until convergence{

$$\underline{\theta_j} := \underline{\theta_j} - \eta \frac{\partial}{\partial \theta_j} J(\theta_0, \theta_1) \qquad \text{(for } j = 0 \text{ and } j = 1\text{)}$$

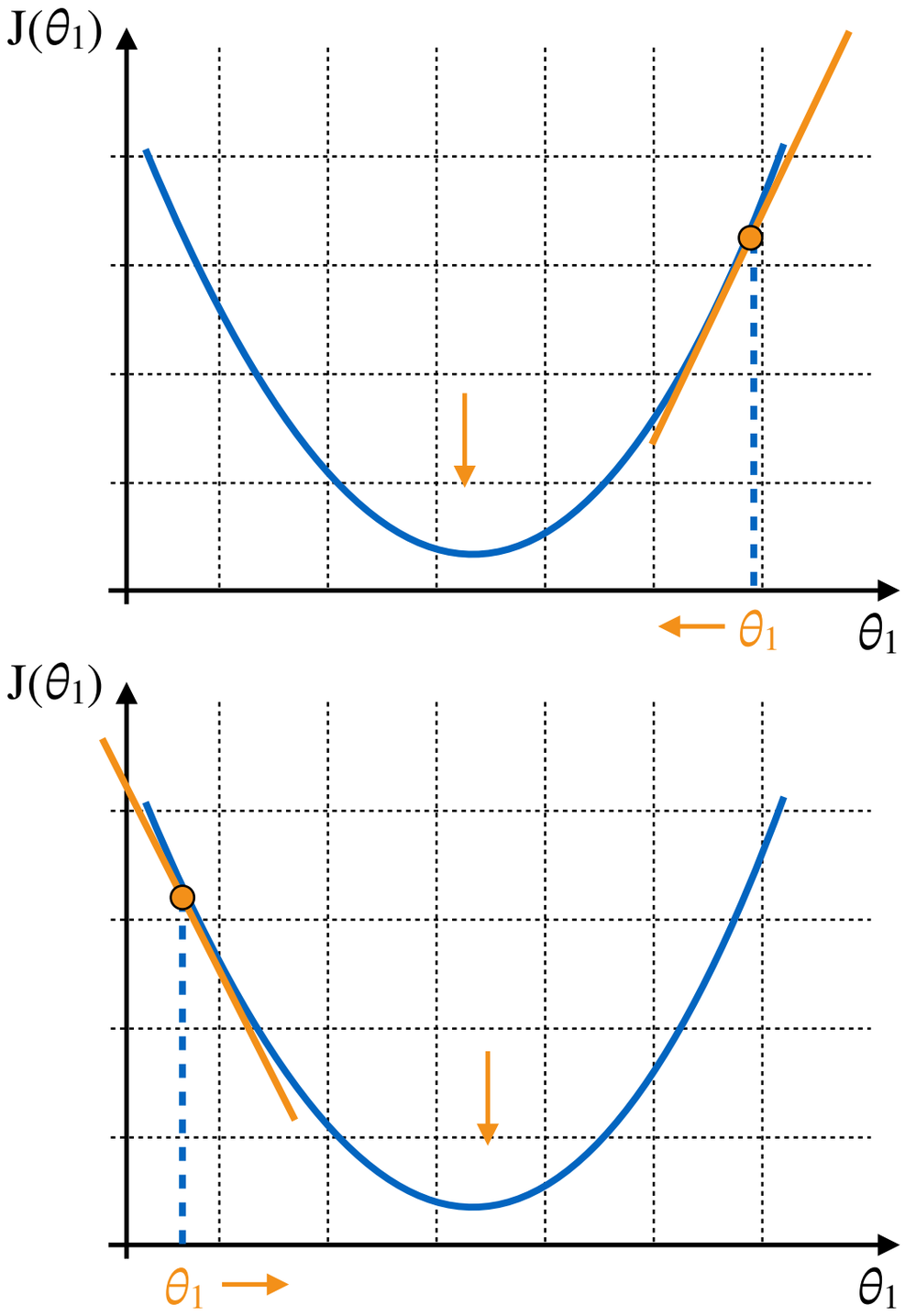
}

learning rate *derivative term*

Simplified version *(like we did a few videos back):*

$$\underset{\theta_1}{\text{minimize}}\ J(\theta_1) \quad \text{where} \quad \theta_1 \in R$$

Gradient Descent



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

≥ 0

then we are decreasing θ_1

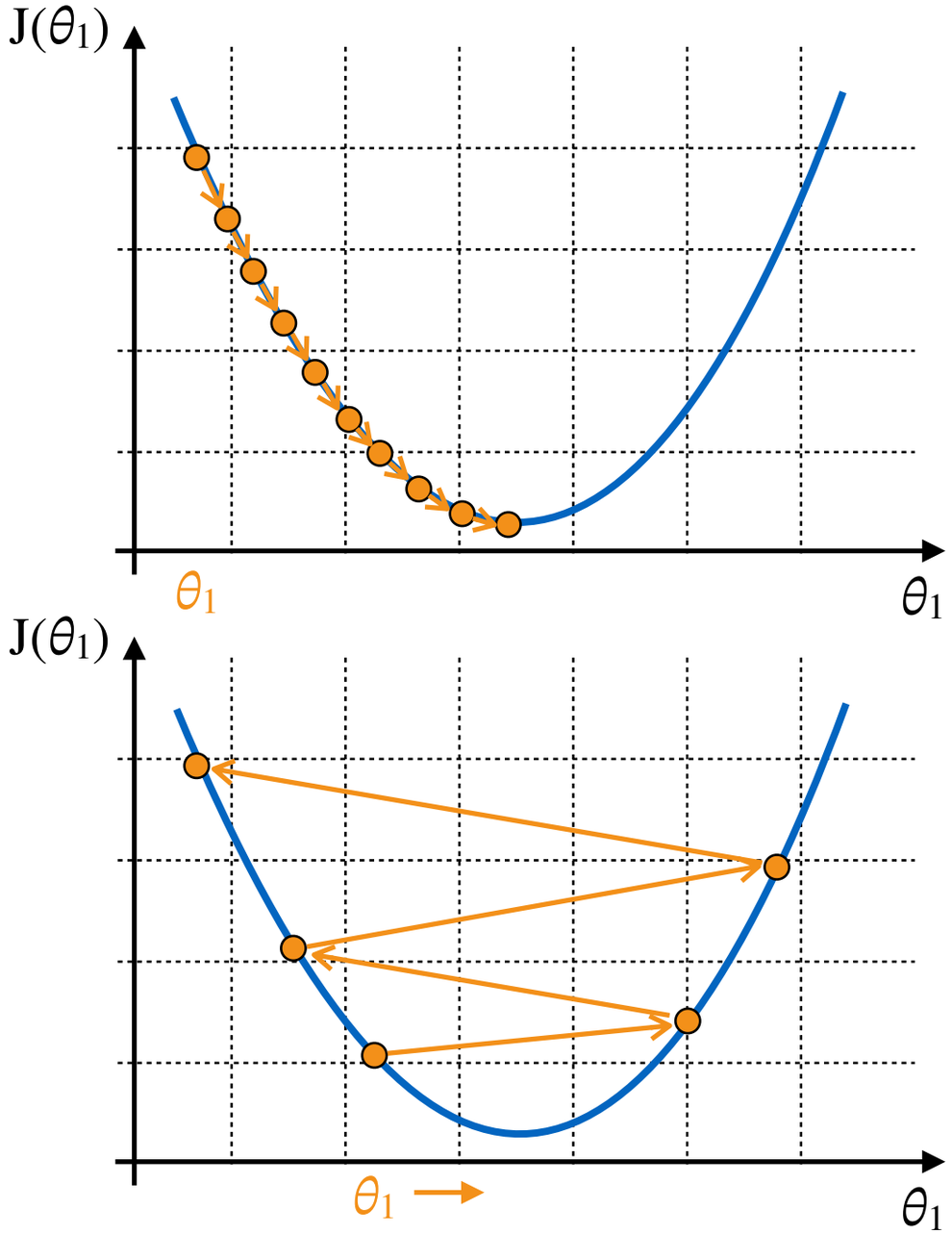
(in the "right direction")

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

≤ 0

Gradient Descent

- Let's take a look now at the learning rate term η



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

- If η is too small, gradient descent can be slow

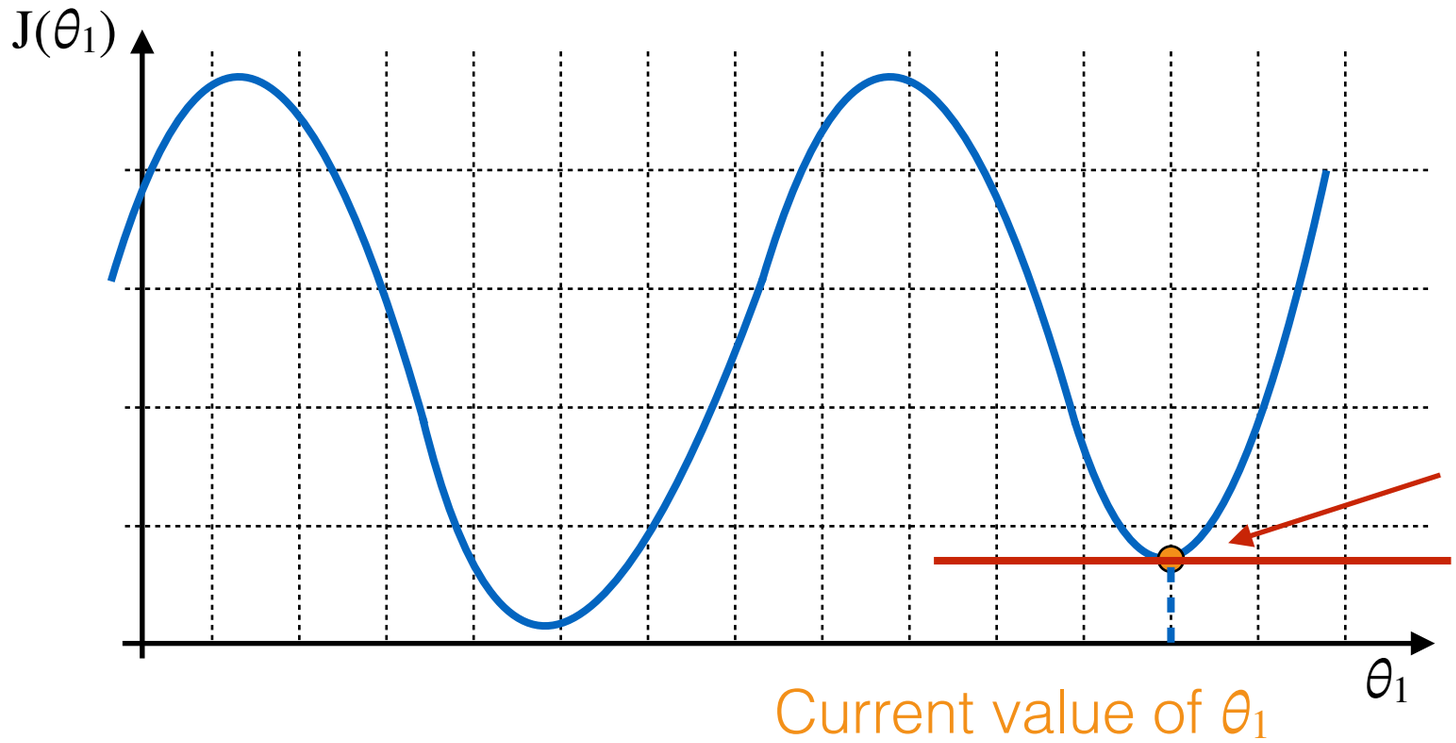
- If η is too large, it can overshoot the minimum

  (it may fail to converge or even diverge)

Adapted from slide by A. Ng

Gradient Descent

- Q: What if your parameter θ_1 is already at a local minimum?

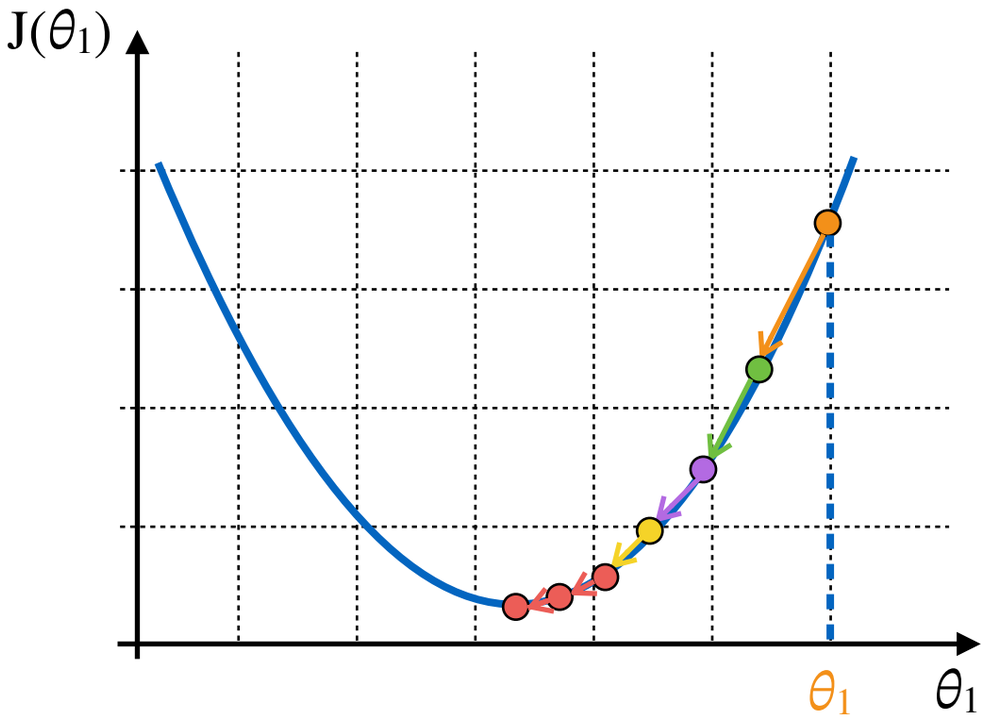


$$\theta_1 := \theta_1 - \eta \frac{\mathrm{d}}{\mathrm{d}\theta_1} J(\theta_1) = 0$$

- A: one step of gradient descent does not change θ_1

Gradient Descent

- Gradient descent can converge to a local minimum even with a fixed learning rate η



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

- As we approach a local minimum, gradient descent will “naturally” take smaller steps

Linear Regression & Gradient Descent

Linear Regression Model

- Hypothesis:

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

- Cost Function:

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

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

Gradient Descent

repeat until convergence{

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

(for j = 0 and j = 1)

}

Linear Regression & Gradient Descent

- We need to figure out what is this derivative term…

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

$$j=0 \qquad \frac{\partial}{\partial \theta_0} J(\theta_0, \theta_1) = \frac{1}{m} \sum_{i=1}^{m} (\theta_0 + \theta_1 x^{(i)} - y^{(i)})$$

$$j=1 \qquad \frac{\partial}{\partial \theta_1} J(\theta_0, \theta_1) = \frac{1}{m} \sum_{i=1}^{m} (\theta_0 + \theta_1 x^{(i)} - y^{(i)})\, x^{(i)}$$

Linear Regression & Gradient Descent

- We can now plug them back into our GD algorithm:

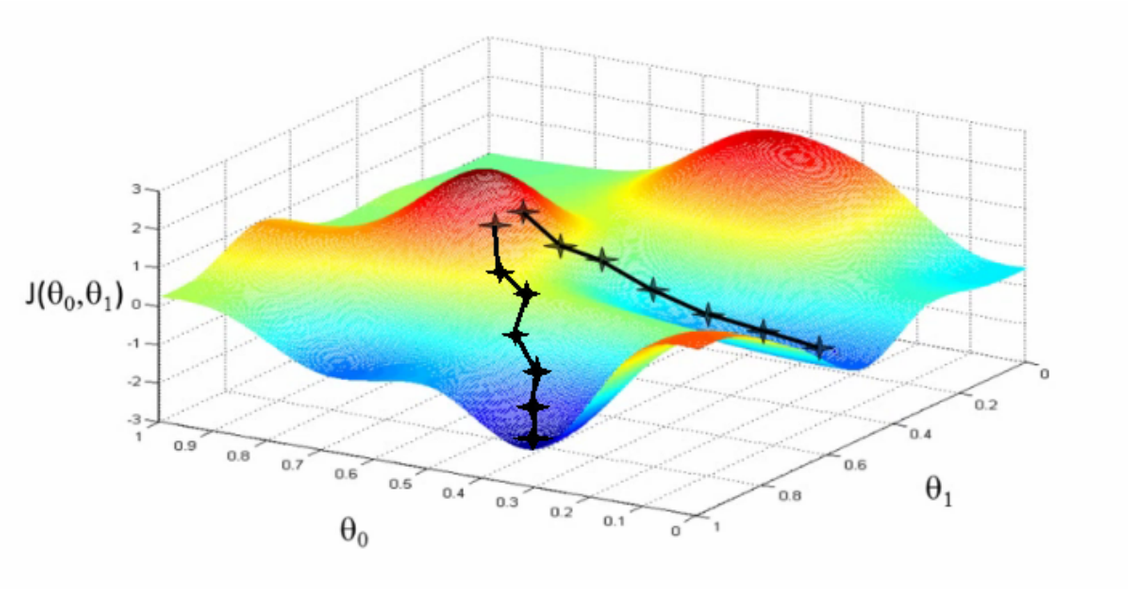
repeat until convergence{

$$\theta_0 := \theta_0 - \eta \frac{1}{m} \sum_{i=1}^{m} (\theta_0 + \theta_1 x^{(i)} - y^{(i)})$$

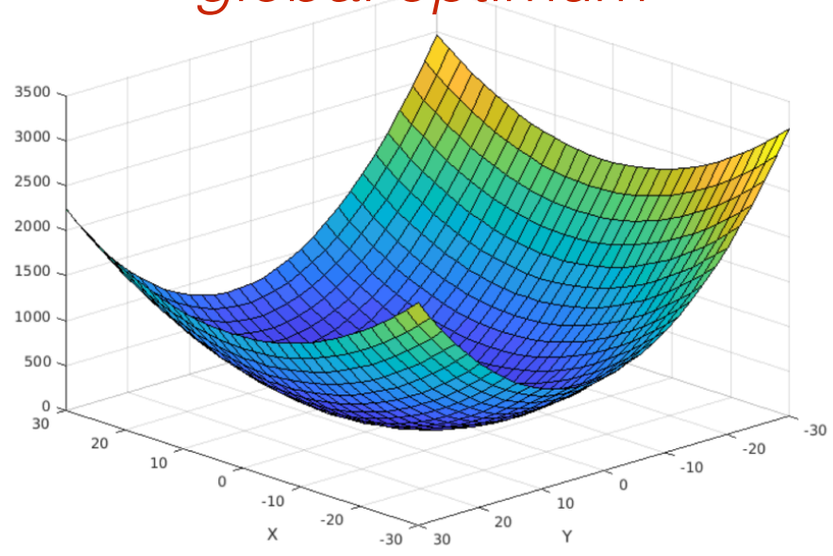
$$\theta_1 := \theta_1 - \eta \frac{1}{m} \sum_{i=1}^{m} (\theta_0 + \theta_1 x^{(i)} - y^{(i)})\, x^{(i)}$$

}

simultaneously update θ_0 and θ_1



global optimum

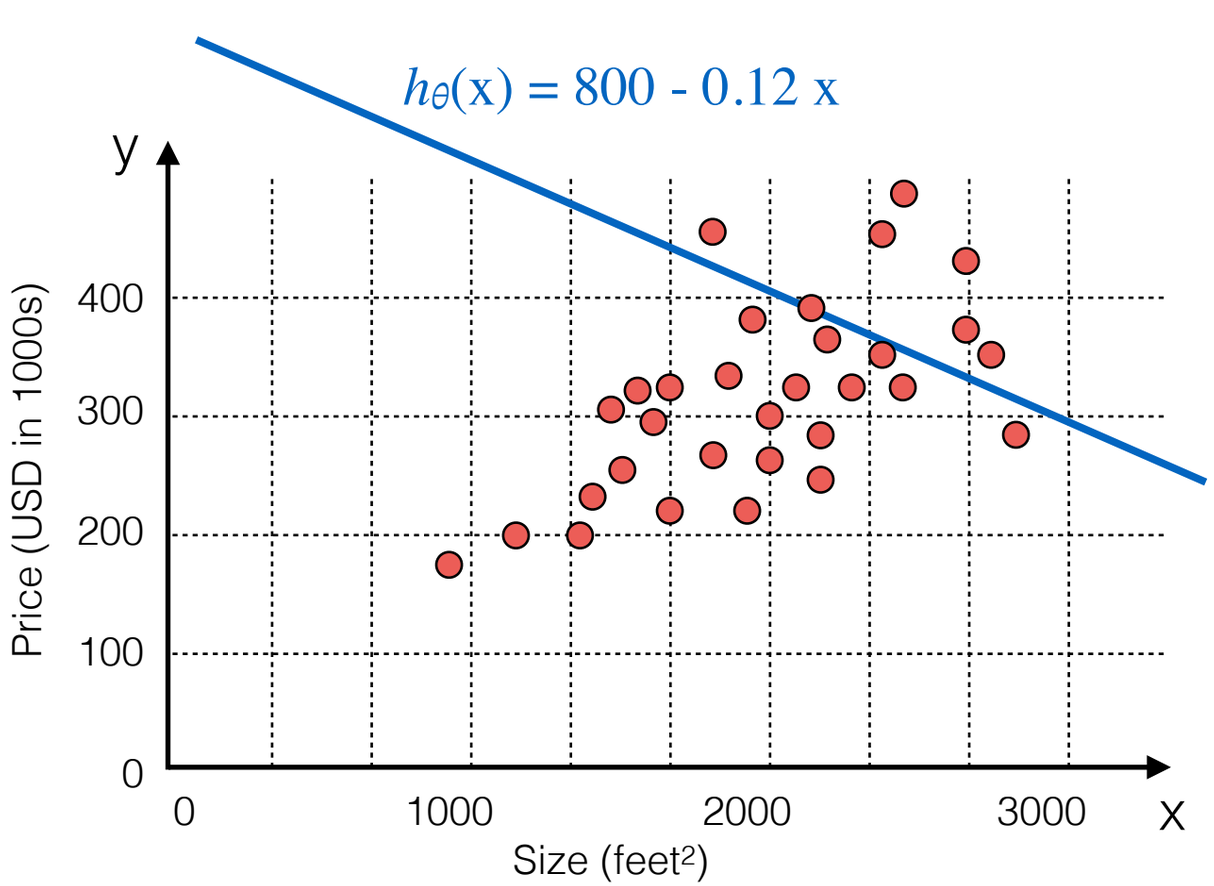


Linear Regression & Gradient Descent

- Let's see the gradient descent algorithm in action:

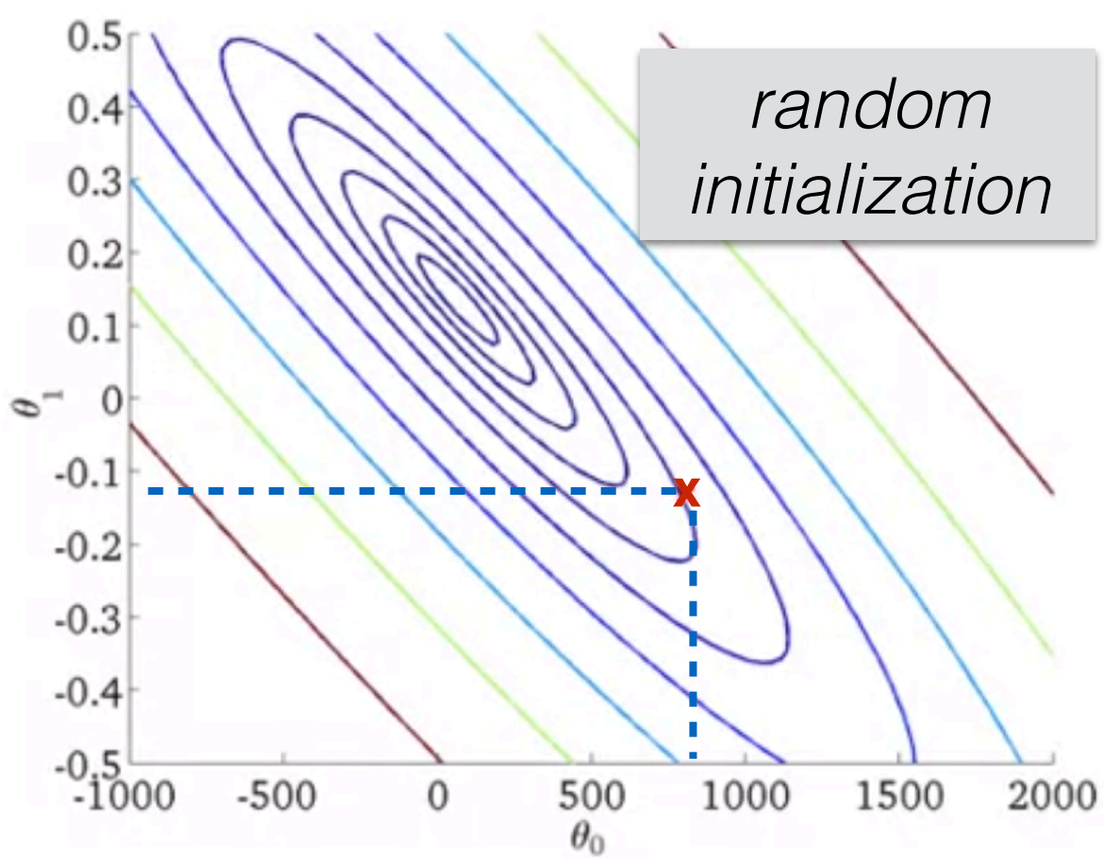
$h_\theta(x) = \theta_0 + \theta_1 x$

(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)

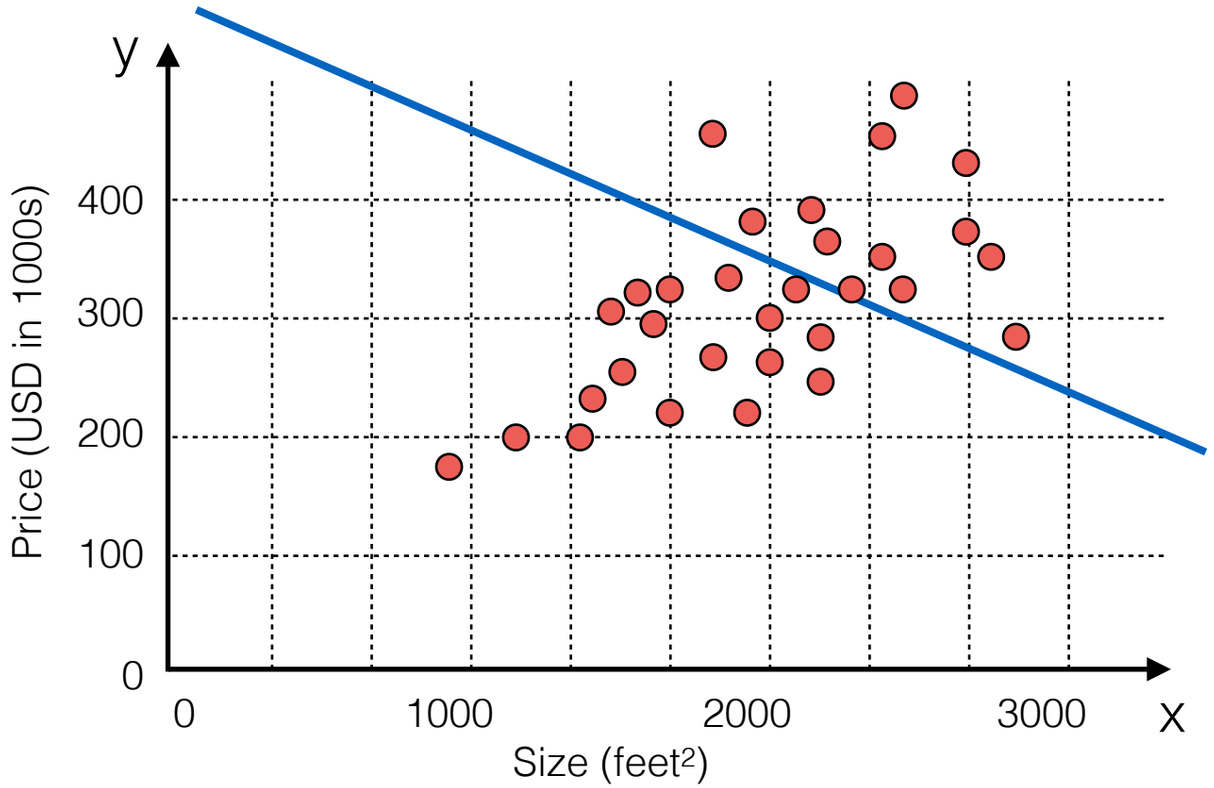


Linear Regression & Gradient Descent

- Let's see the gradient descent algorithm in action:

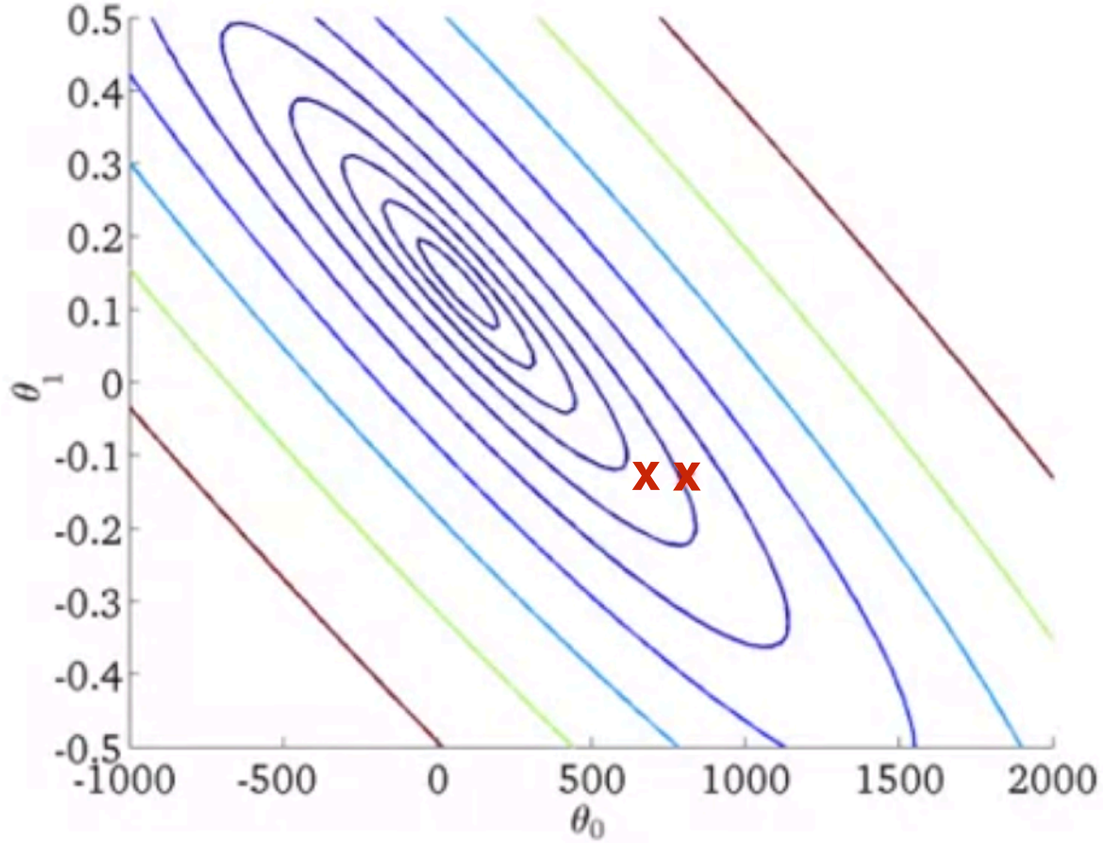
$h_\theta(x) = \theta_0 + \theta_1 x$

(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)

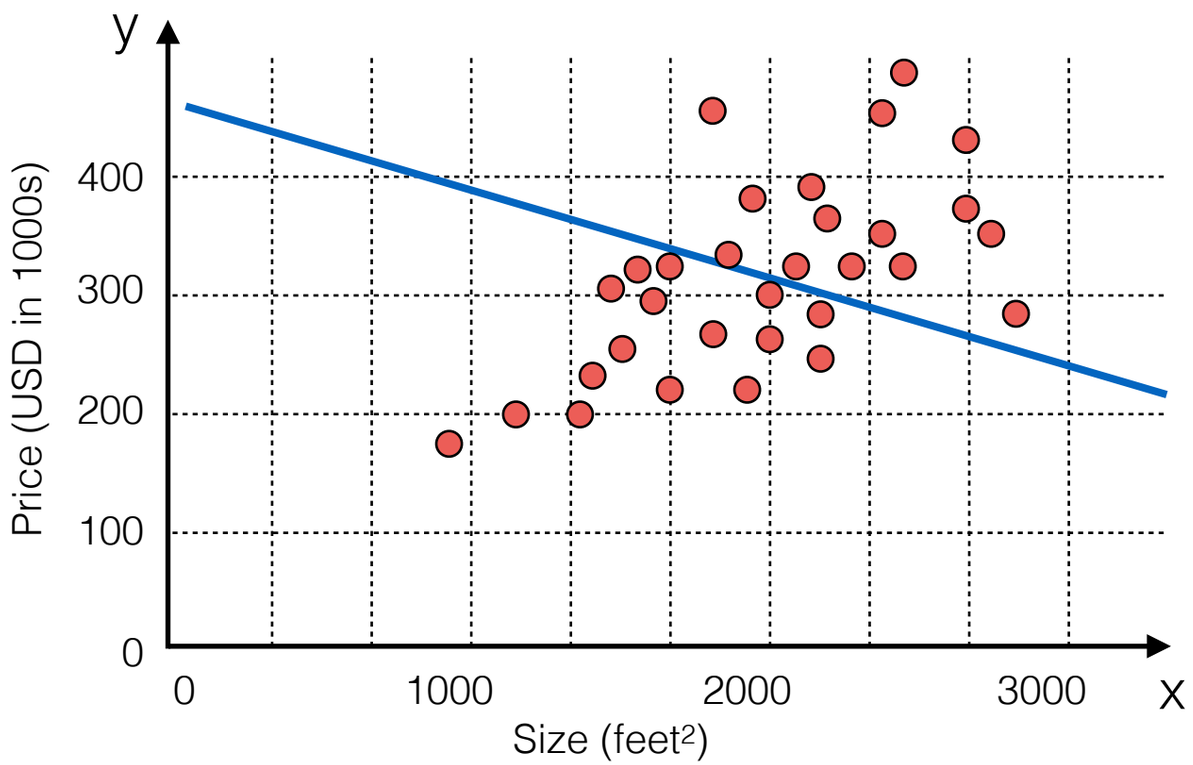


Linear Regression & Gradient Descent

- Let's see the gradient descent algorithm in action:

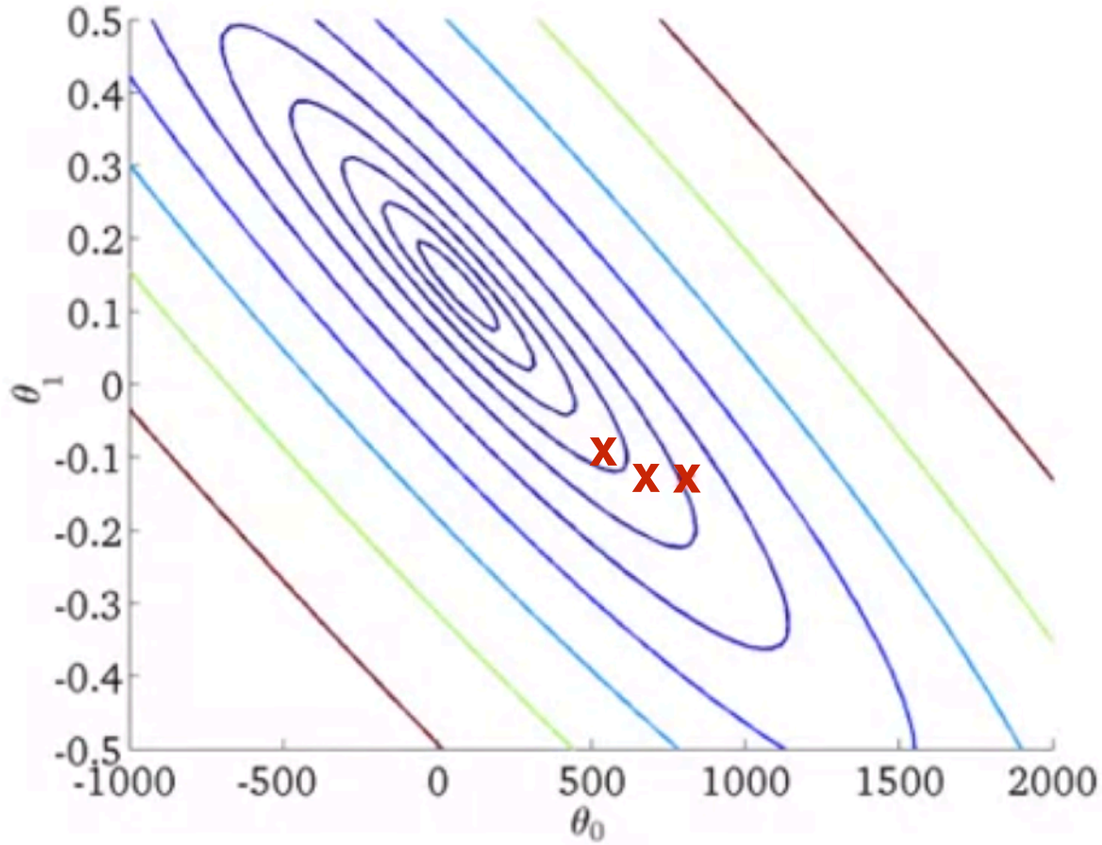
$h_\theta(x) = \theta_0 + \theta_1 x$

(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)

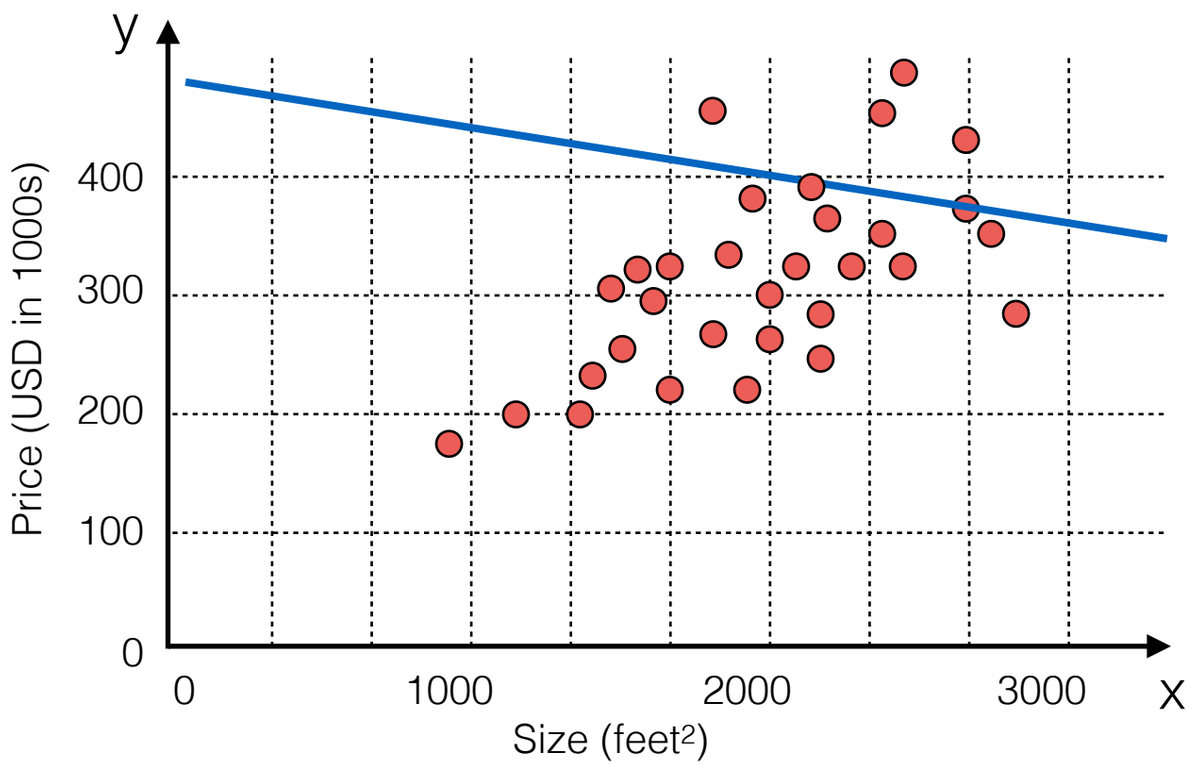


Linear Regression & Gradient Descent

- Let's see the gradient descent algorithm in action:

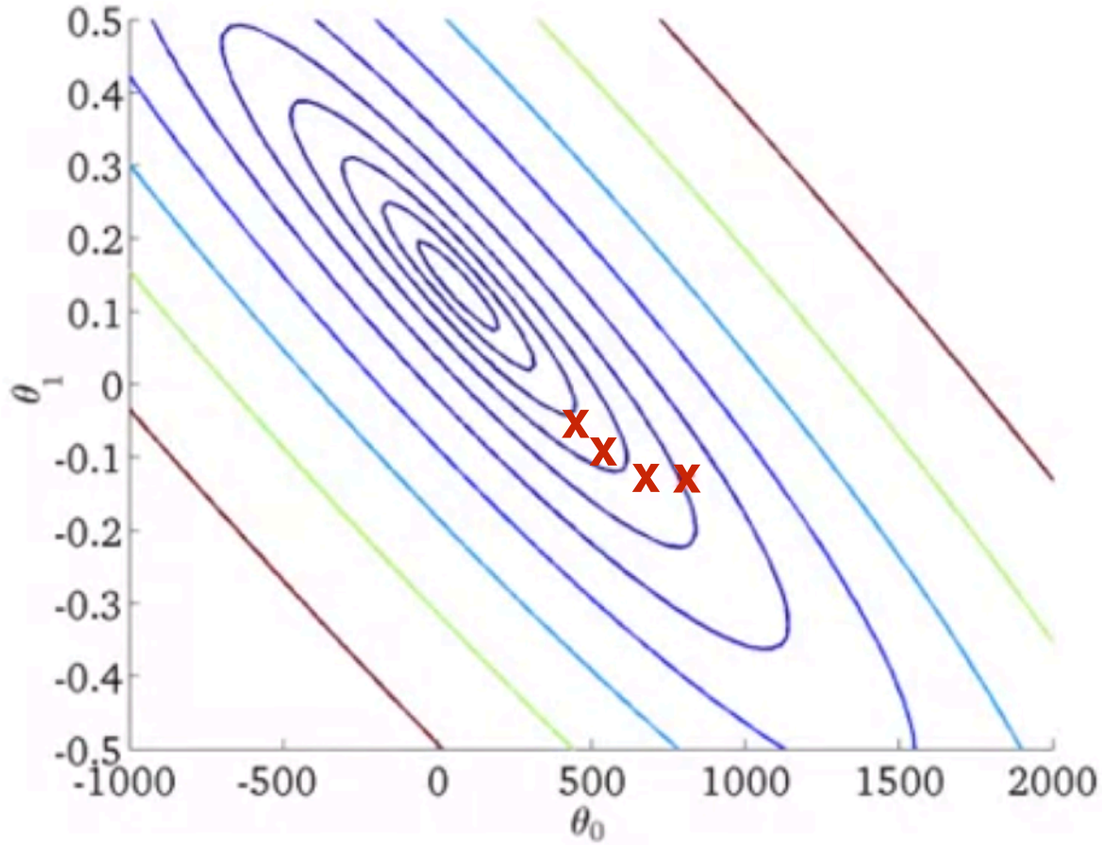
$h_\theta(x) = \theta_0 + \theta_1 x$

(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)

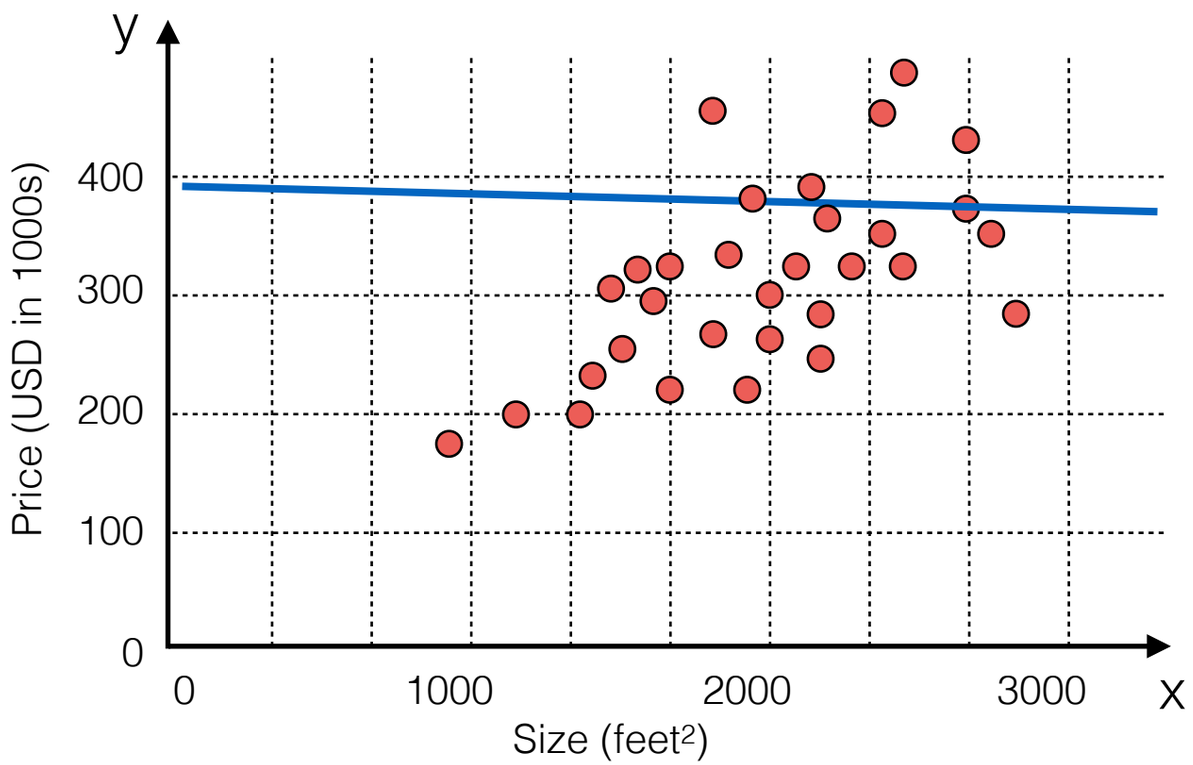


Linear Regression & Gradient Descent

- Let's see the gradient descent algorithm in action:

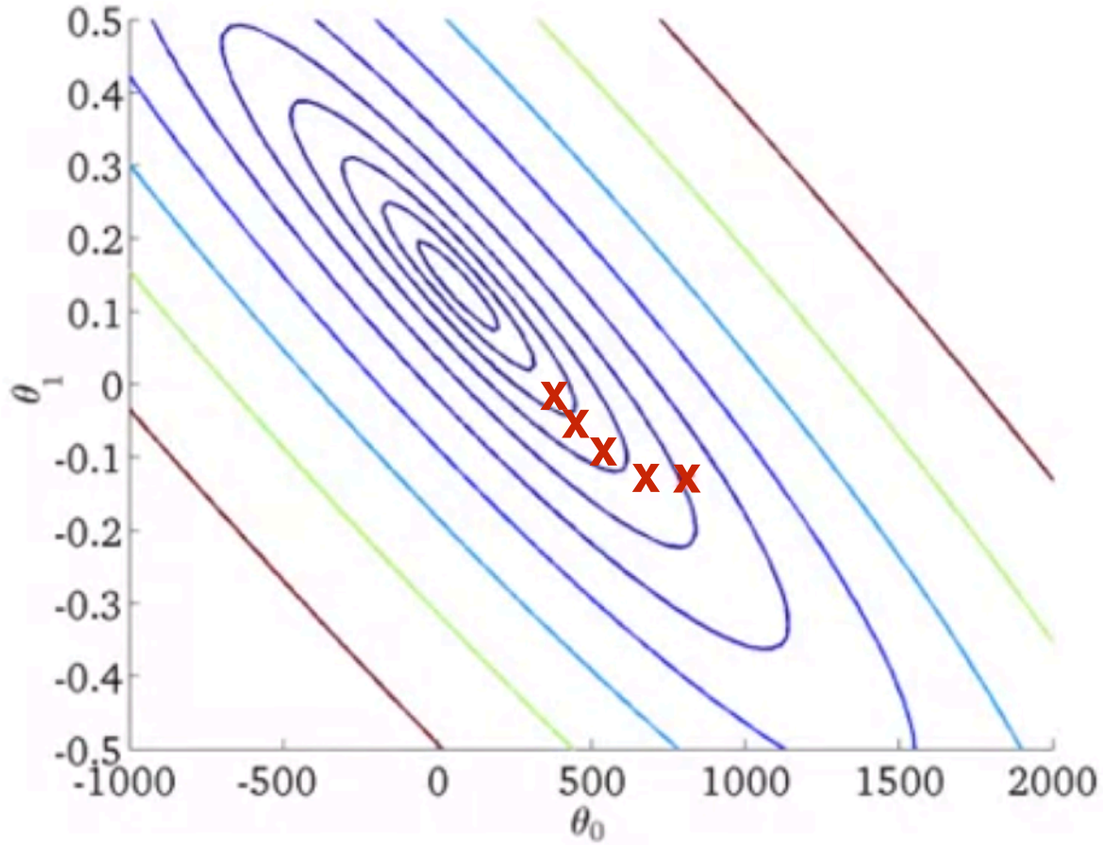
$h_\theta(\mathrm{x}) = \theta_0 + \theta_1\,\mathrm{x}$

(for fixed θ_0,θ_1 this is a function of x)



$\mathrm{J}(\theta_0,\theta_1)$

(function of the parameters θ_0,θ_1)

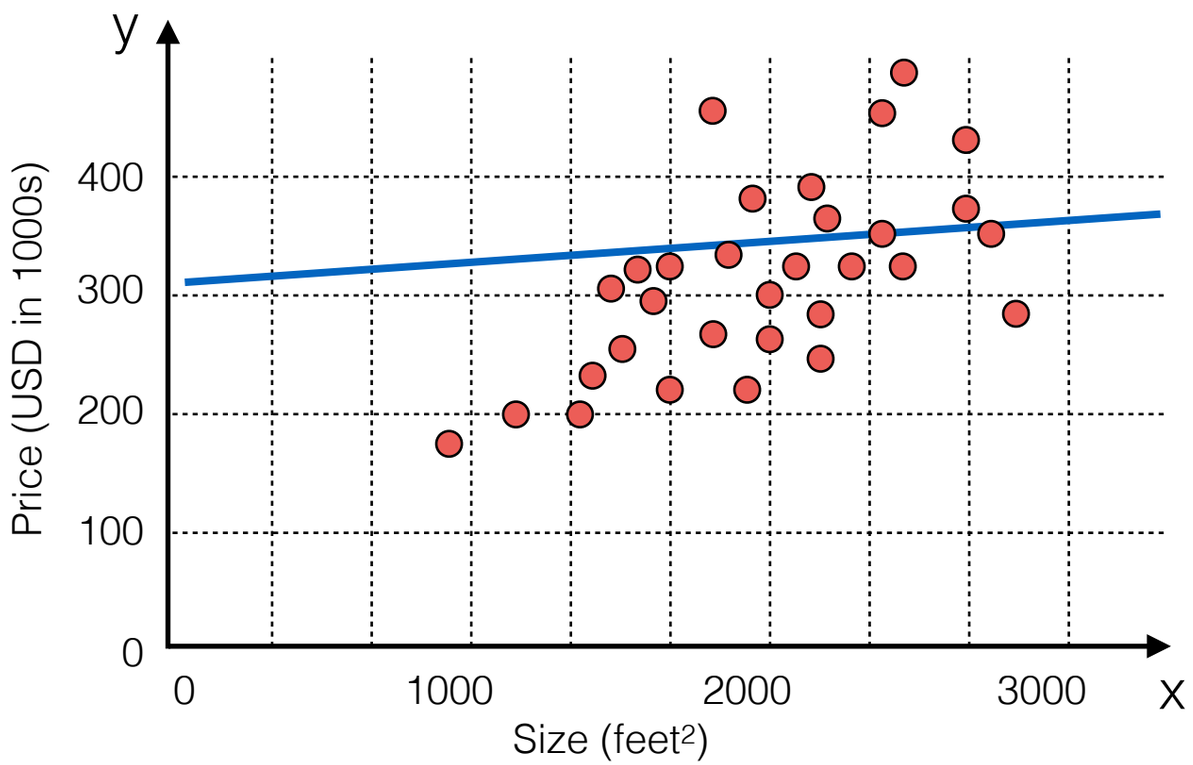


Linear Regression & Gradient Descent

- Let's see the gradient descent algorithm in action:

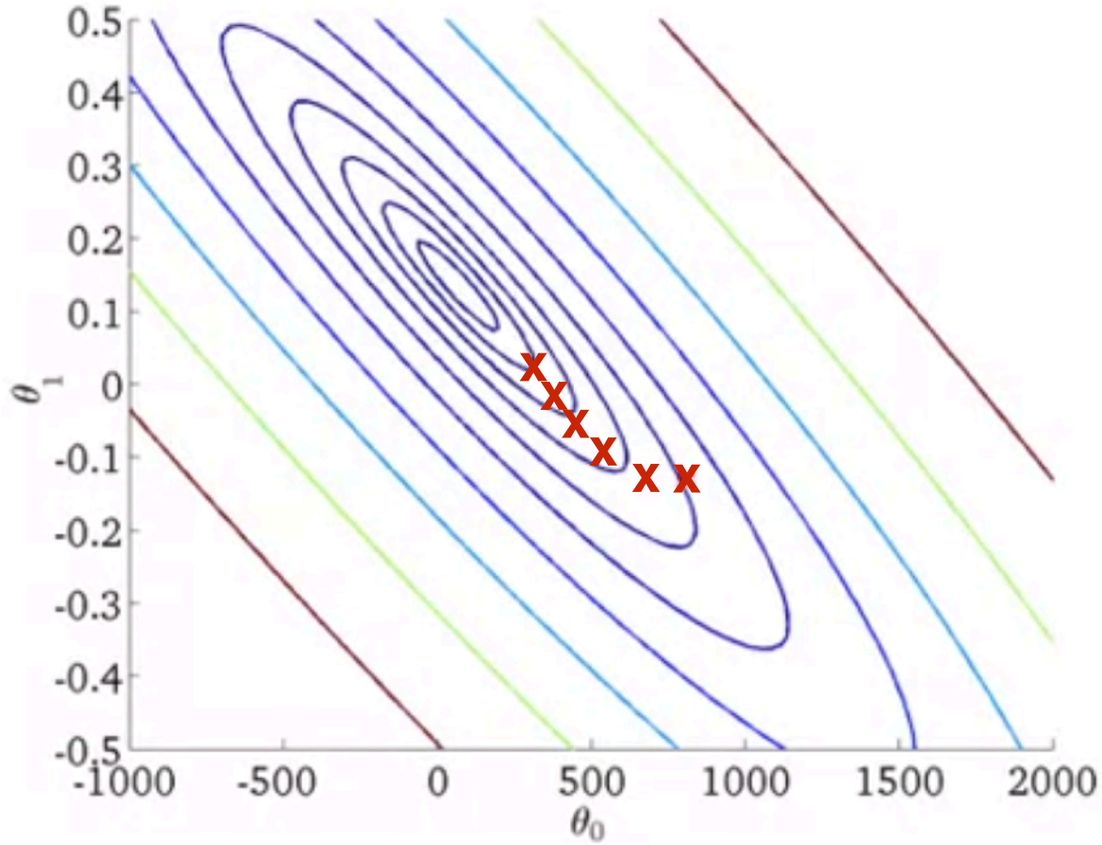
$h_\theta(x) = \theta_0 + \theta_1 x$

(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)

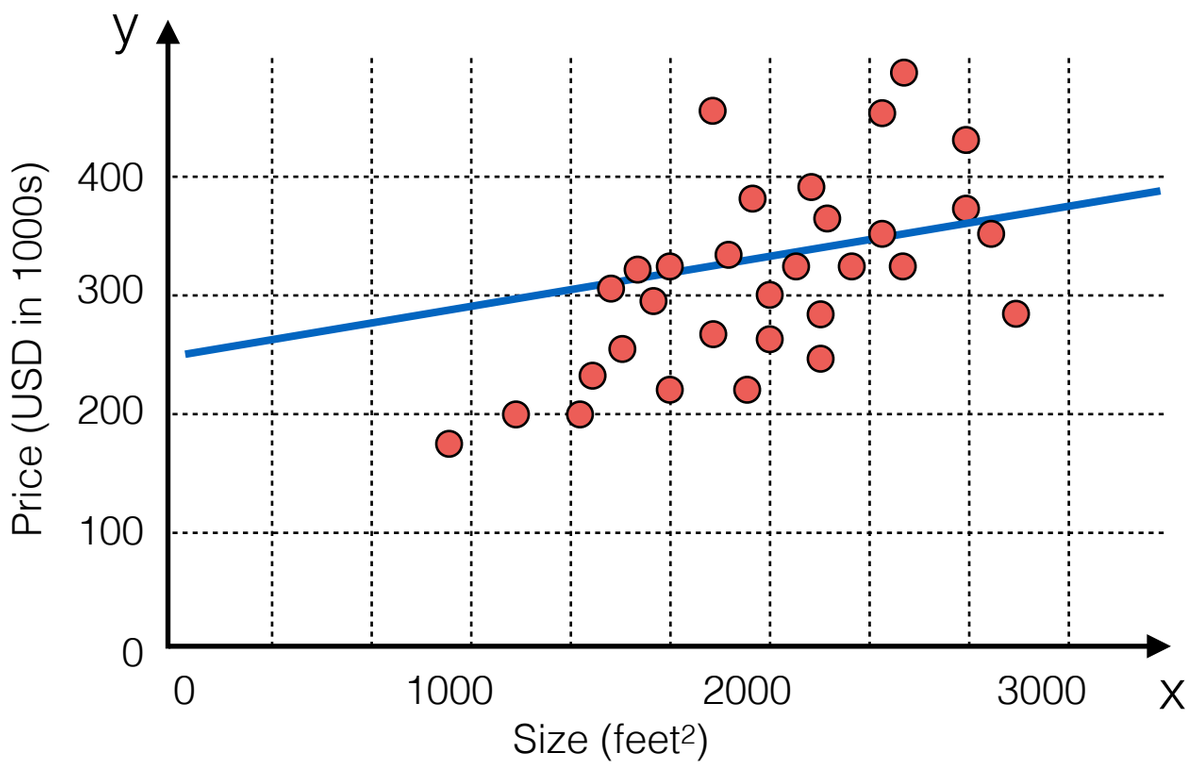


Linear Regression & Gradient Descent

- Let's see the gradient descent algorithm in action:

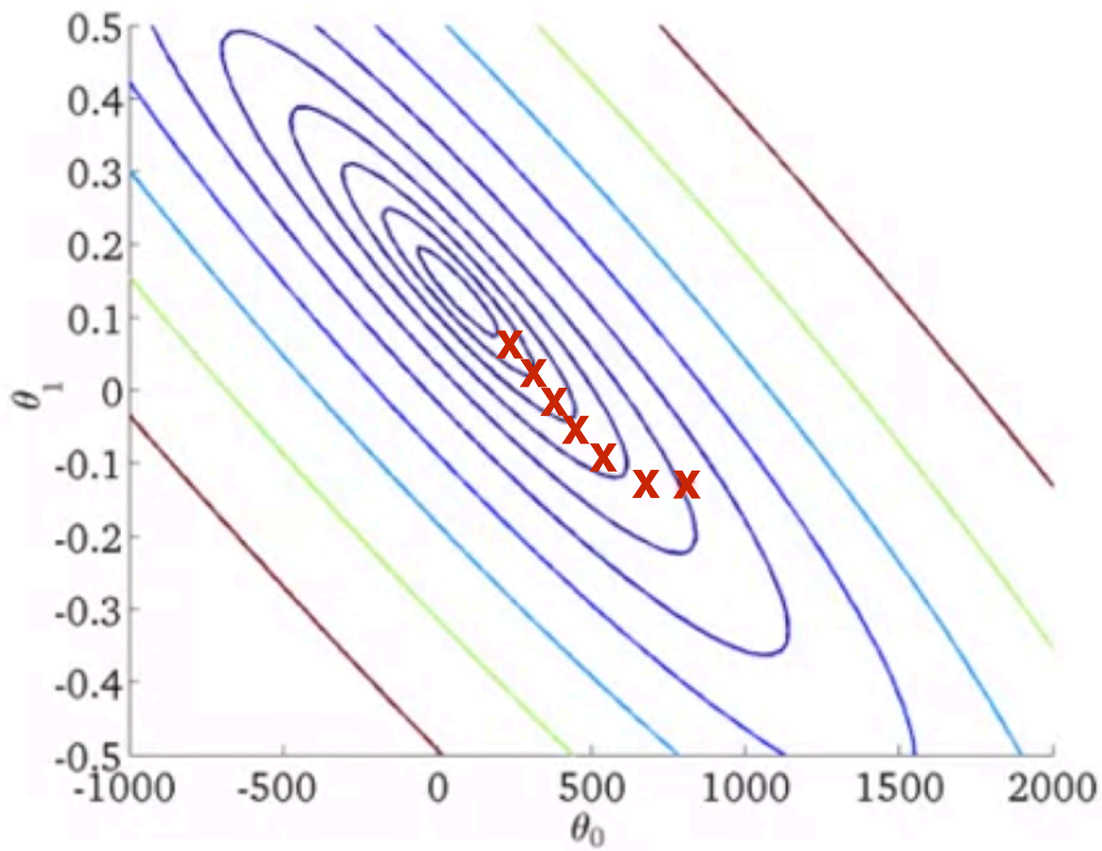
$h_\theta(x) = \theta_0 + \theta_1 x$

(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)

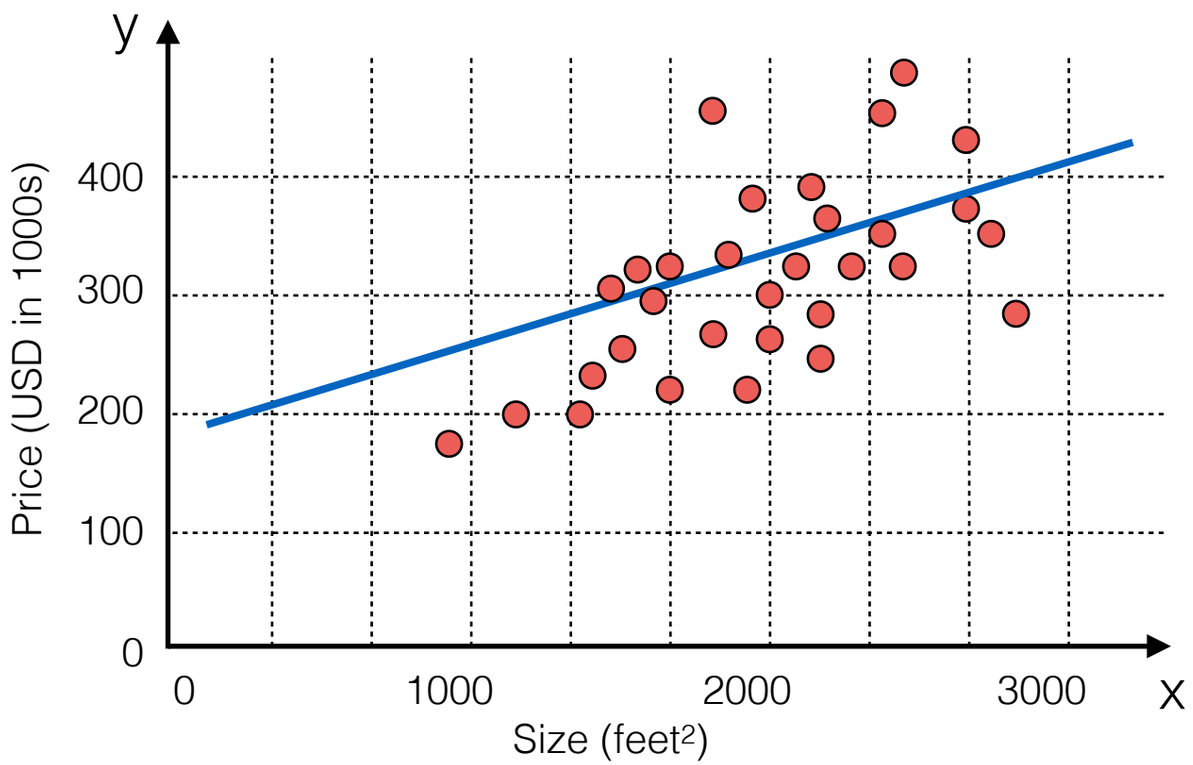


Linear Regression & Gradient Descent

- Let's see the gradient descent algorithm in action:

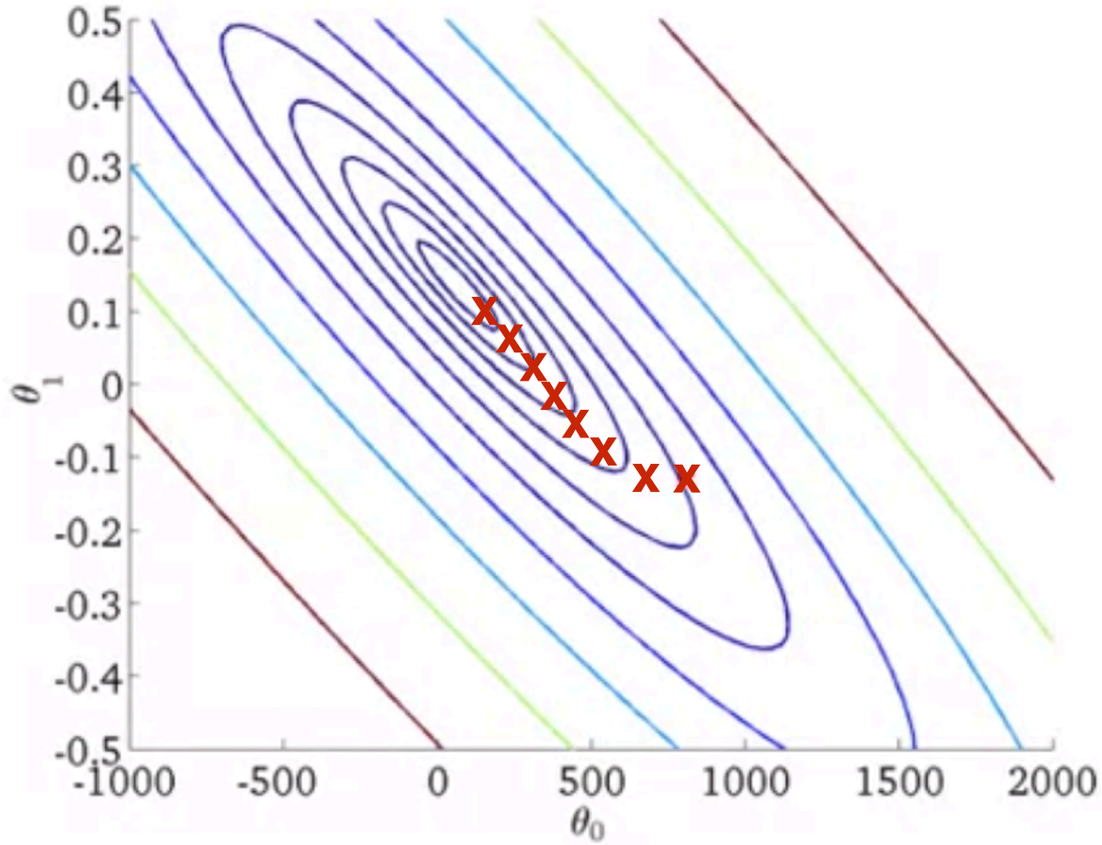
$h_\theta(x) = \theta_0 + \theta_1 x$

(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)

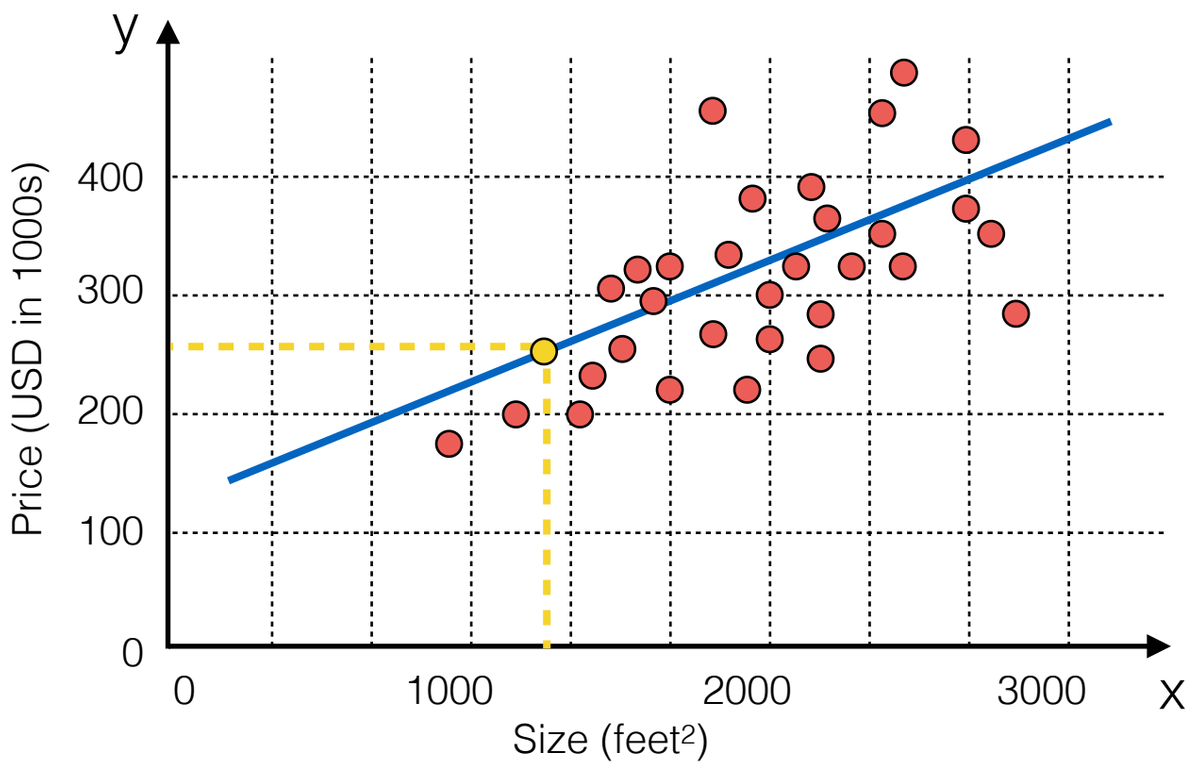


Linear Regression & Gradient Descent

- Let's see the gradient descent algorithm in action:

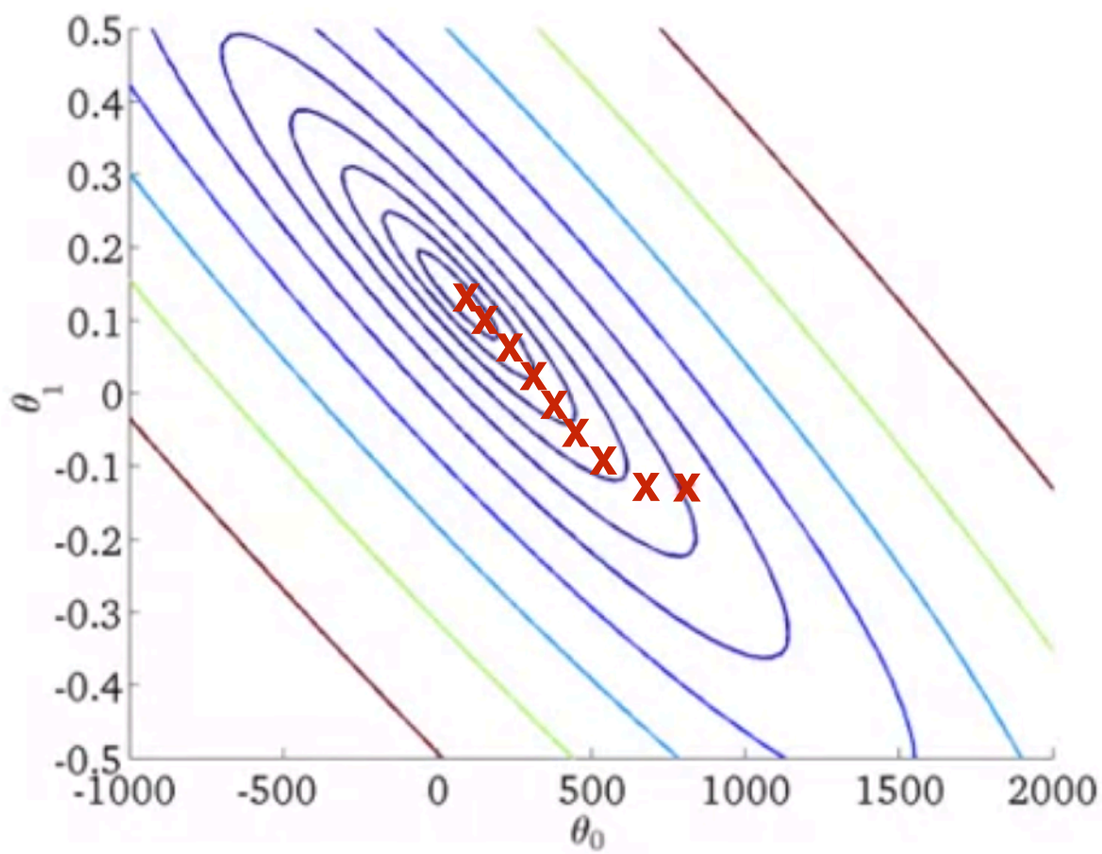
$h_\theta(x) = \theta_0 + \theta_1 x$

(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)



A bit more on Gradient Descent

- We have introduced *batch gradient descent*
  (i.e. each step of gradient descent uses all training examples)
  - There is also another way to optimize across the training set
- Stochastic Gradient Descent: update the parameters for each training case in turn, according to its own gradients

```
Randomly shuffle examples in the training set
for i=1 to m do{
```

$$\theta_0 := \theta_0 - \eta \ (\theta_0 + \theta_1 x^{(i)} - y^{(i)})$$

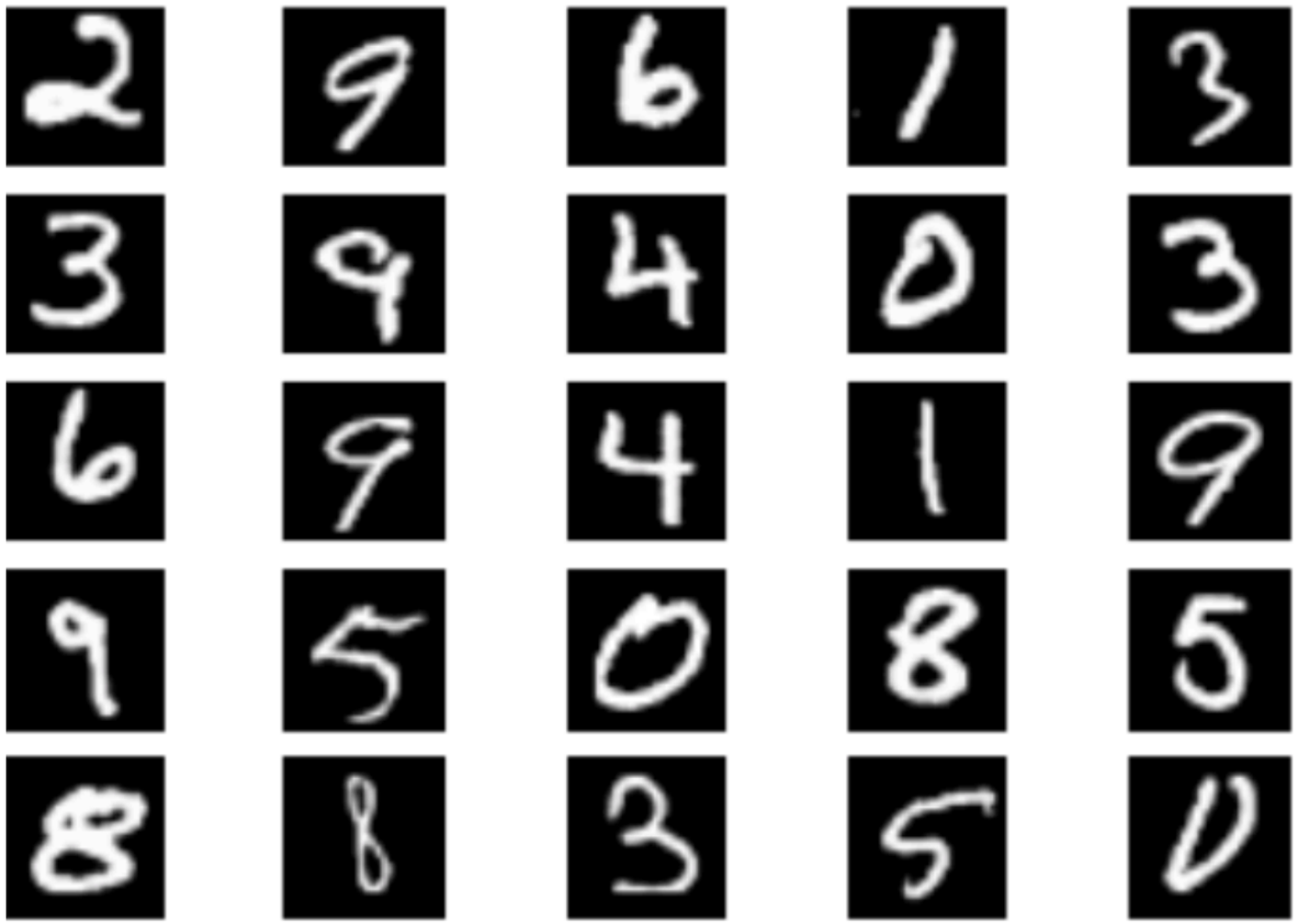
$$\theta_1 := \theta_1 - \eta \ (\theta_0 + \theta_1 x^{(i)} - y^{(i)}) \ x^{(i)} \ \}$$

Underlying assumption: samples are independent and identically distributed (i.i.d.)

- Mini-batch Gradient Descent: that's something in between

Learning is useful in many tasks

- **Classification:** determine to which discrete category a specific example belongs to



Example 1

What digit is this?

Linear classifiers

- **Classification:** determine to which discrete category a specific example belongs to

- Other examples:
  - Email: spam *vs* not spam (*ham*)
  - Online transactions: fraudulent *vs* not fraudulent
  - Tumor: malignant *vs* benign

Classification *vs* Regression

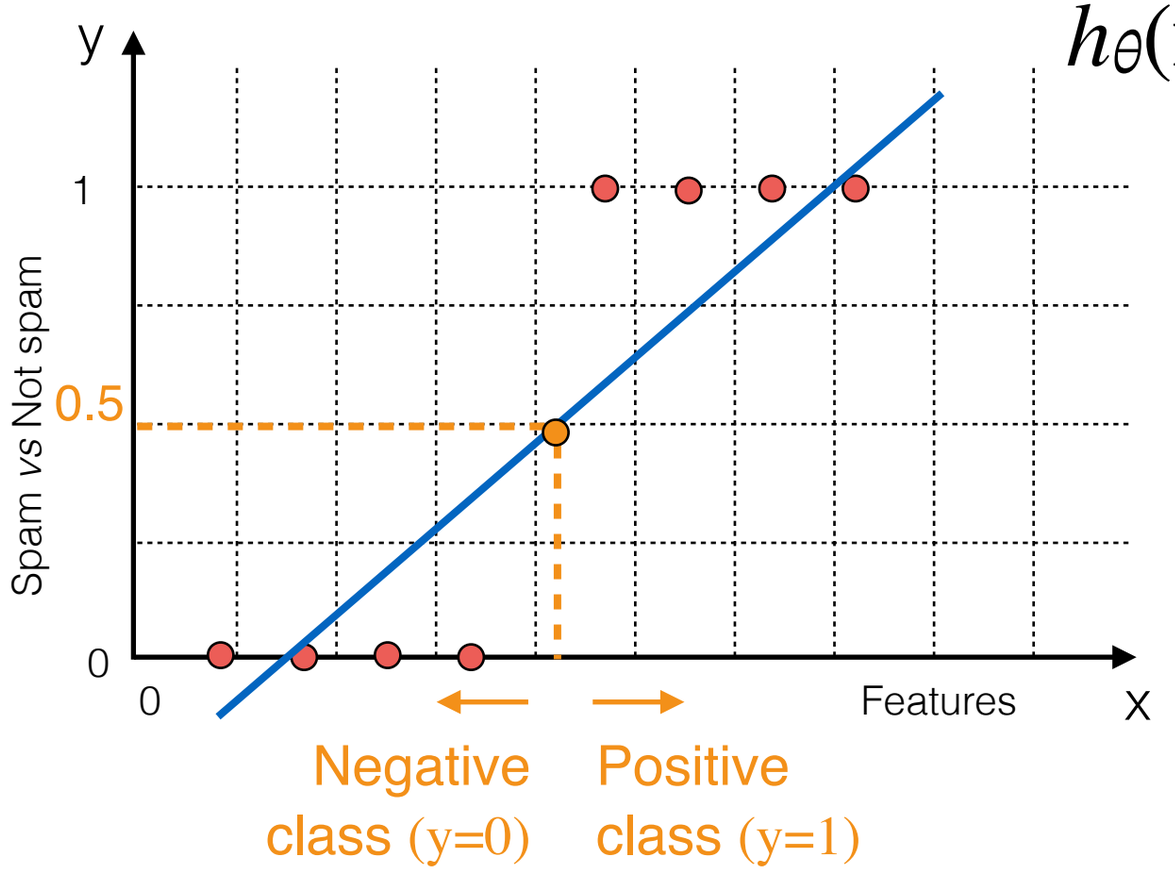
- Categorical outputs called labels (or classes)
  - e.g. yes/no, 1/2/3/…/9, cat/dog/person/…
  - Then we are interested in: $h \sim f: X \rightarrow Y$, where Y is categorical (while in regression typically $Y=R$)

- Binary classification: two possible labels
- Multi-class classification: multiple possible labels

We will first look at binary problems and then discuss multi-class problems

Classification as Regression

- Can we do (binary) classification using what we have learned until now?



$$h_\theta(\mathbf{x}) = \theta^T \mathbf{x}$$

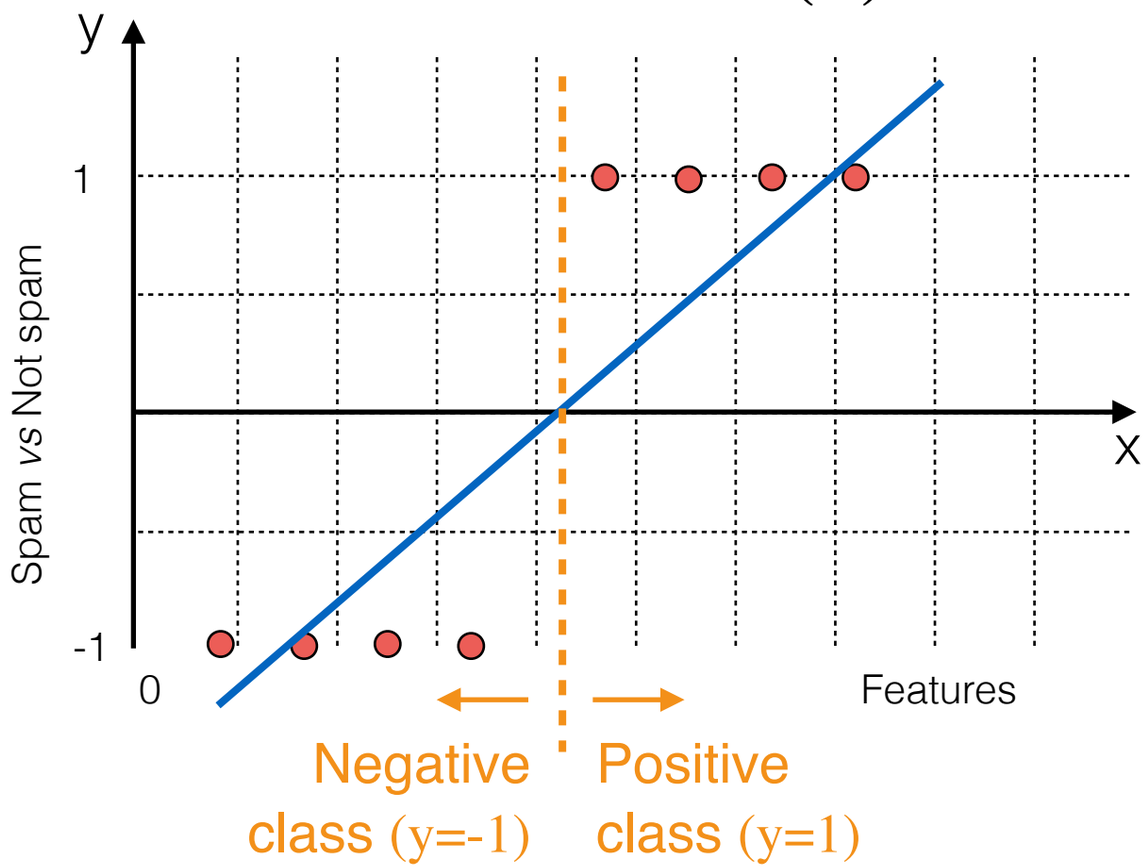
Threshold classifier:

- If $h_\theta(\mathbf{x}) \geq 0.5$, predict $\mathrm{y} = 1$
- If $h_\theta(\mathbf{x}) < 0.5$, predict $\mathrm{y} = 0$

Classification as Regression

- Let's use a slightly different notation

$$h_\theta(\mathbf{x}) = \theta^T \mathbf{x}$$



Threshold classifier:

- If $h_\theta(\mathbf{x}) \geq 0$, predict $\mathrm{y} = 1$
- If $h_\theta(\mathbf{x}) < 0$, predict $\mathrm{y} = -1$

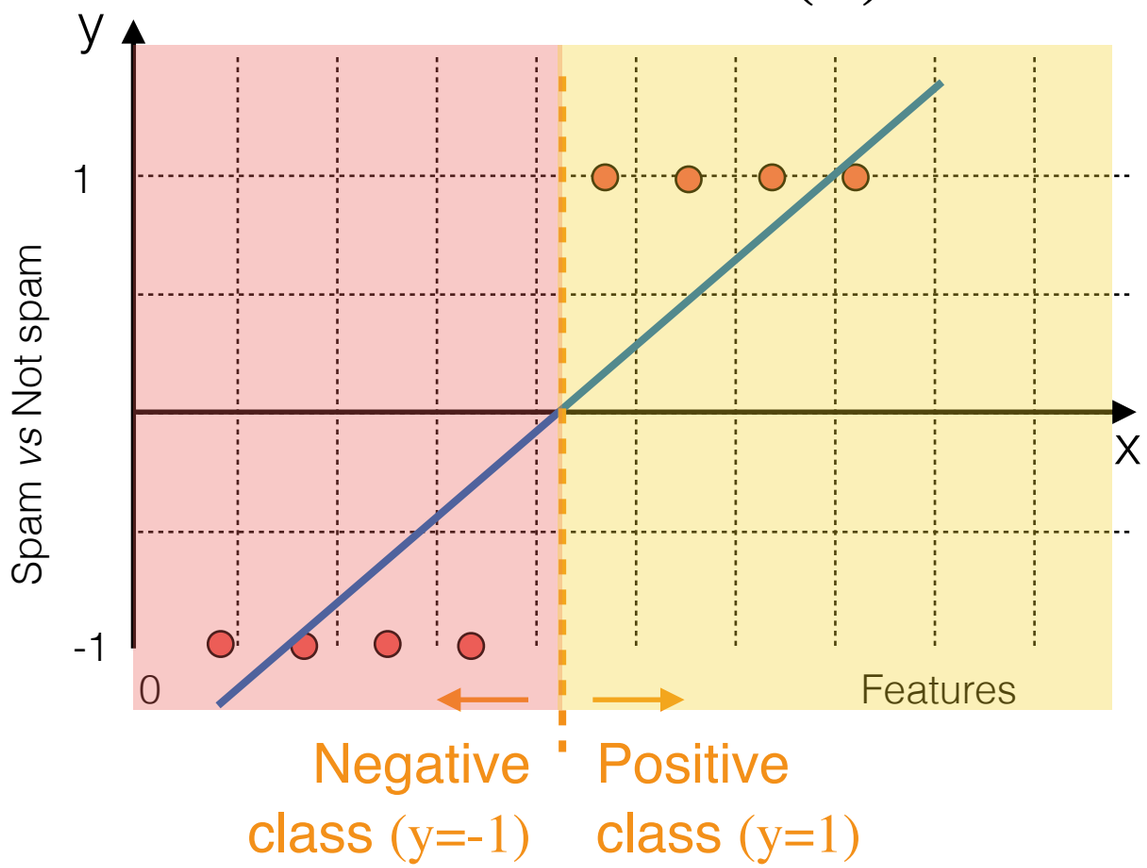
Decision rule *(mathematically)*:

- $\mathrm{y} = \mathrm{sign}(h_\theta(\mathbf{x}))$

Linear Classification

- This specifies a *linear classifier*: it has a linear boundary (hyperplane) which separates the space

$$h_\theta(\mathbf{x}) = \theta^T \mathbf{x}$$



Decision rule:

- $y = \text{sign}(h_\theta(\mathbf{x}))$

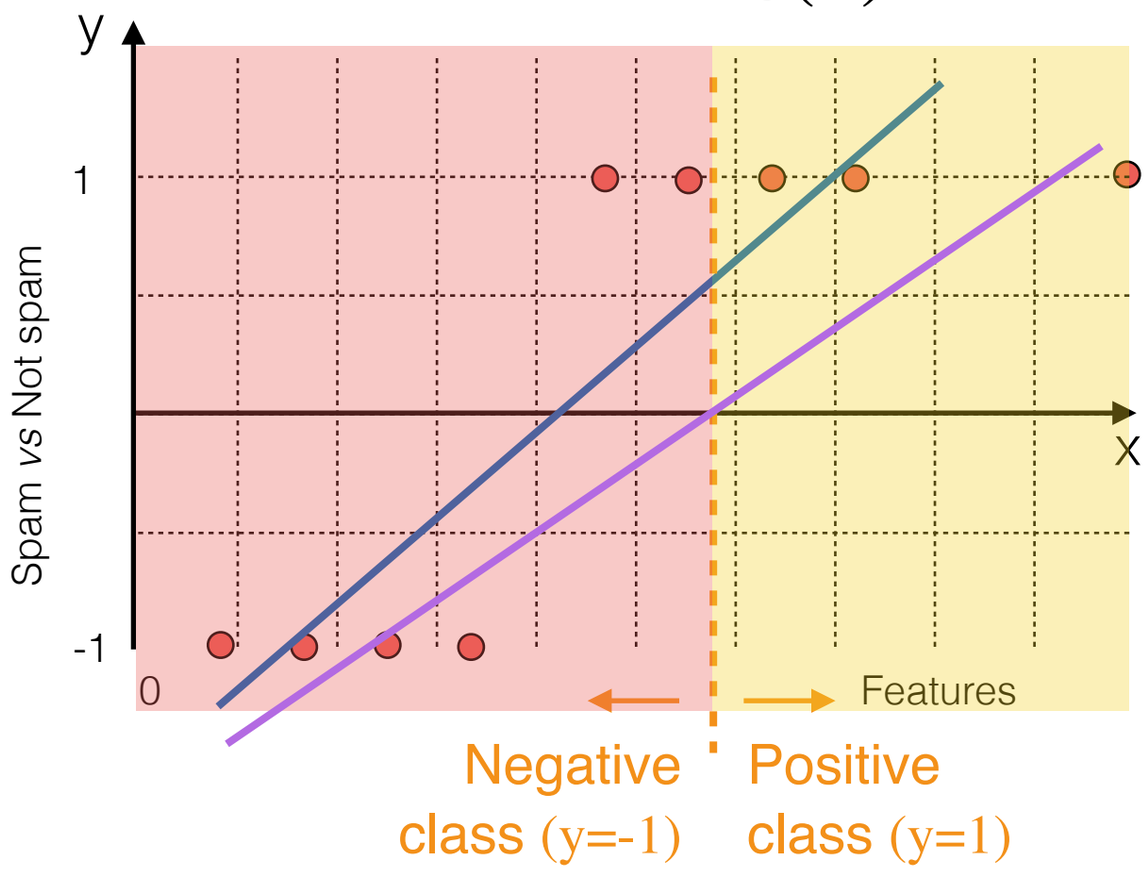
The linear boundary separates the space into two “half-spaces”

In 1D this is simply a threshold

Linear Classification

- Applying linear regression to classification tasks is not always a great idea…

$$h_\theta(\mathbf{x}) = \theta^T \mathbf{x}$$



Decision rule:

- $y = \text{sign}(h_\theta(\mathbf{x}))$

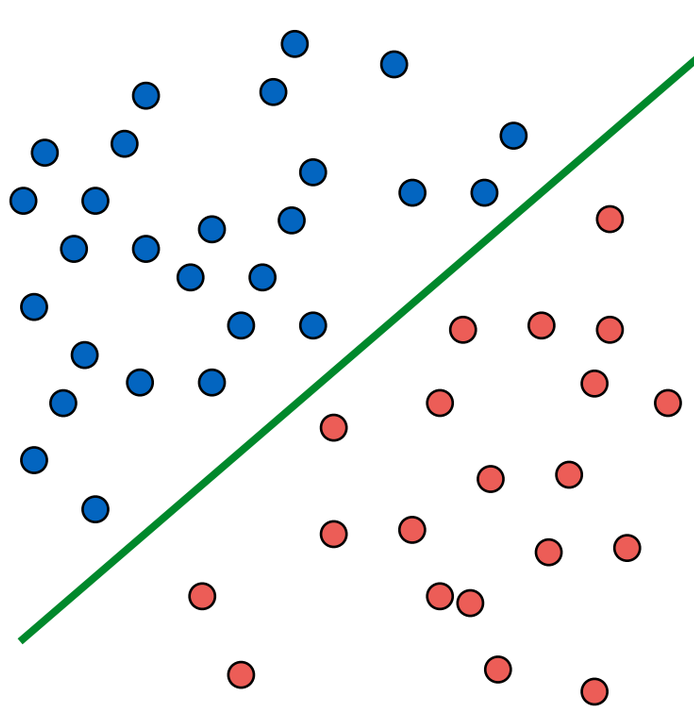
The linear boundary separates the space into two "half-spaces"

In 1D this is simply a threshold

Linear Classification

- This specifies a *linear classifier*: it has a linear boundary (hyperplane) which separates the space

$$h_\theta(\mathbf{x}) = \theta^T \mathbf{x}$$



Decision rule:

- $y = \text{sign}(h_\theta(\mathbf{x}))$

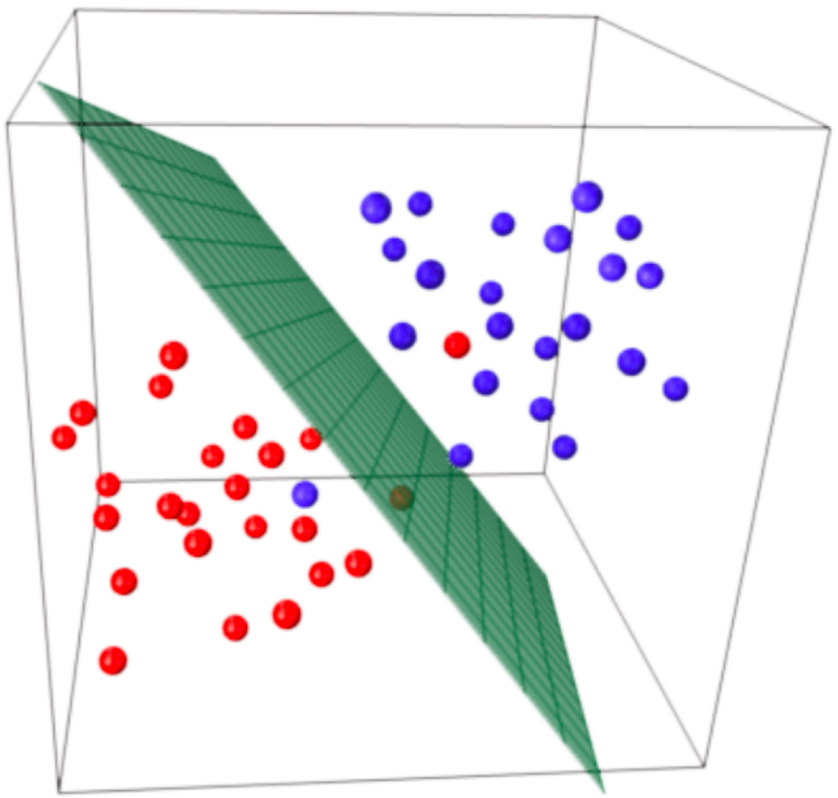
The linear boundary separates the space into two “half-spaces”

In 2D this is a line

Linear Classification

- This specifies a *linear classifier*: it has a linear boundary (hyperplane) which separates the space

$$h_\theta(\mathbf{x}) = \theta^T \mathbf{x}$$



Decision rule:

- $y = \text{sign}(h_\theta(\mathbf{x}))$

The linear boundary separates the space into two “half-spaces”

In 3D this is a plane

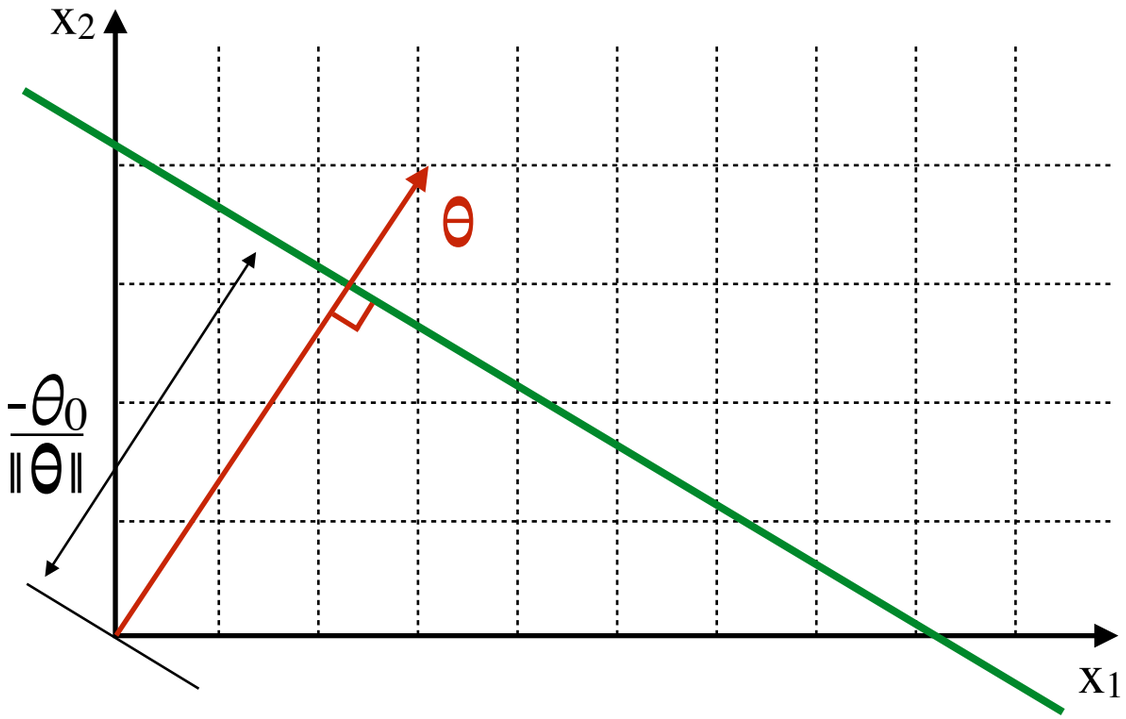
Image credit: S. Fidler

Geometric Interpretation

- What about higher-dimensional spaces?

$\theta^T \mathbf{x} = 0$ a line passing through the origin and orthogonal to θ

$\theta^T \mathbf{x} + \theta_0 = 0$ shifts it by θ_0 ← *Note: this is usually referred as to the "bias term"*



$\theta^T \mathbf{x} + \theta_0 = 0$

A bit more about the notation

We are using this trick/assumption:

$$h_\theta(\mathbf{x}) = \theta^T \mathbf{x} = \begin{bmatrix} \theta_0 \\ \theta_1 \\ \theta_2 \end{bmatrix}^T \begin{bmatrix} 1 \\ x_1 \\ x_2 \end{bmatrix}$$

Logistic Regression

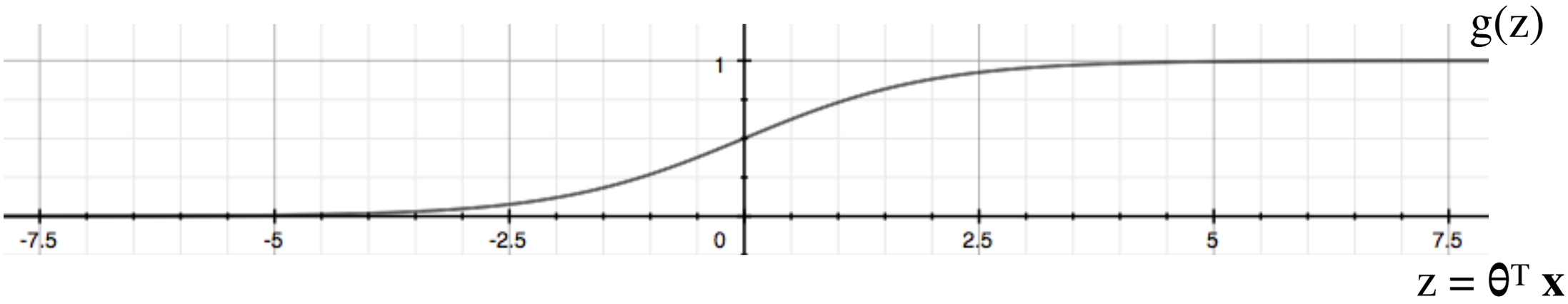
- Applying linear regression to classification tasks usually is not a great idea
- A better approach is to use *logistic regression*
  - Note: although the term regression appears in its name, logistic regression is a classification algorithm
  - It has also a nice property: $0 \leq h_\theta(\mathrm{x}) \leq 1$

Logistic Regression

- Hypothesis representation:

$$h_\theta(\mathbf{x}) = g(\theta^T \mathbf{x}) = \frac{1}{1 + e^{-\theta^T x}}$$

where $g(z) = \frac{1}{1 + e^{-z}}$ (*Sigmoid* or *Logistic* function)



Adapted from slide by A. Ng

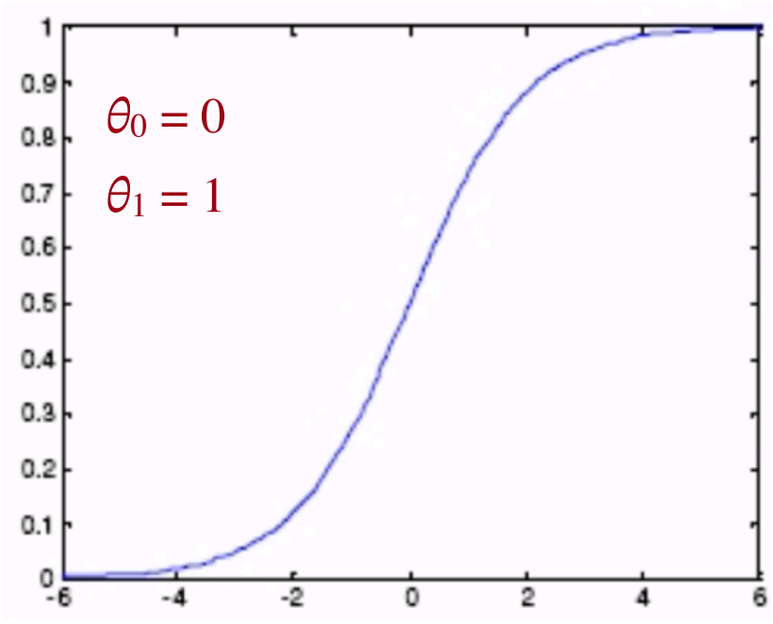
Logistic Regression

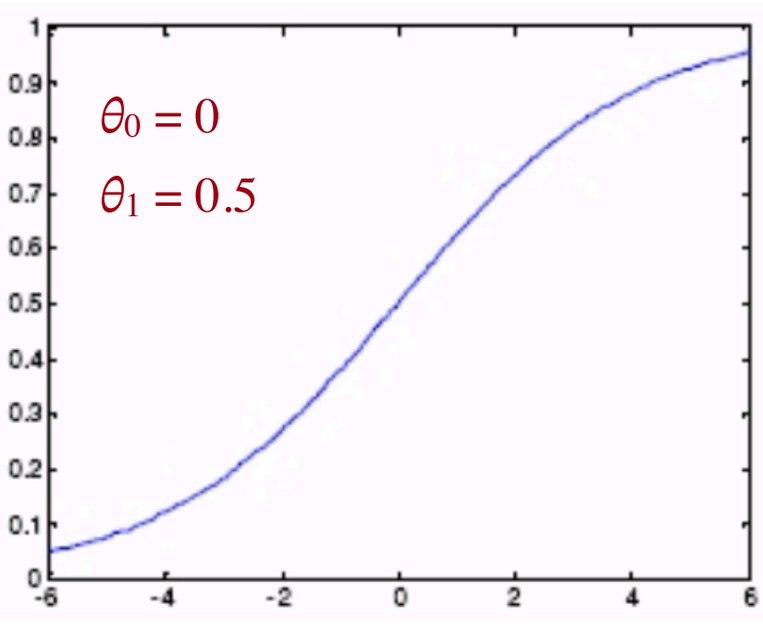
- A bit more about the shape of the logistic function:

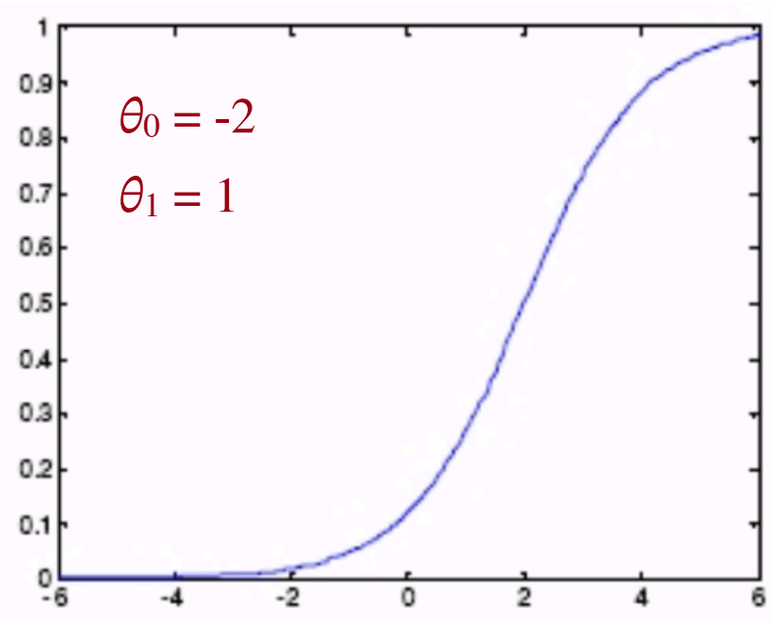
$$h_\theta(\mathbf{x}) = g(\theta^T \mathbf{x}) \qquad \text{where} \qquad g(z) = \sigma(z) = \frac{1}{1 + e^{-z}}$$

(*Sigmoid* or *Logistic* function)

1D example: $h_\theta(\mathbf{x}) = \dfrac{1}{1 + e^{-(\theta_0 + \theta_1 x)}}$





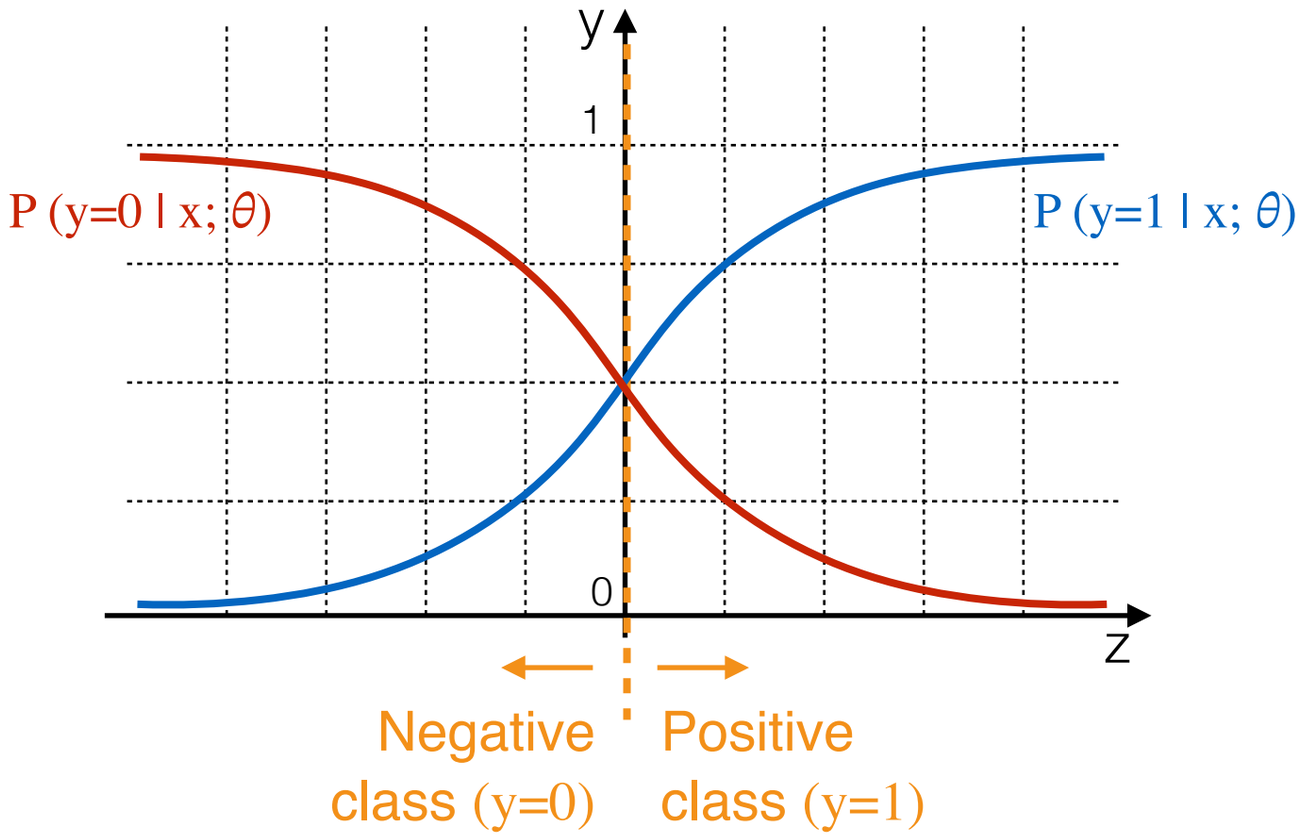


Decision Boundary

- What is the decision boundary for logistic regression?

$$h_\theta(\mathbf{x}) = g(\theta^T \mathbf{x})$$

where $g(z) = \dfrac{1}{1 + e^{-z}}$



$$h_\theta(x) = P\,(y=1 \mid x; \theta)$$

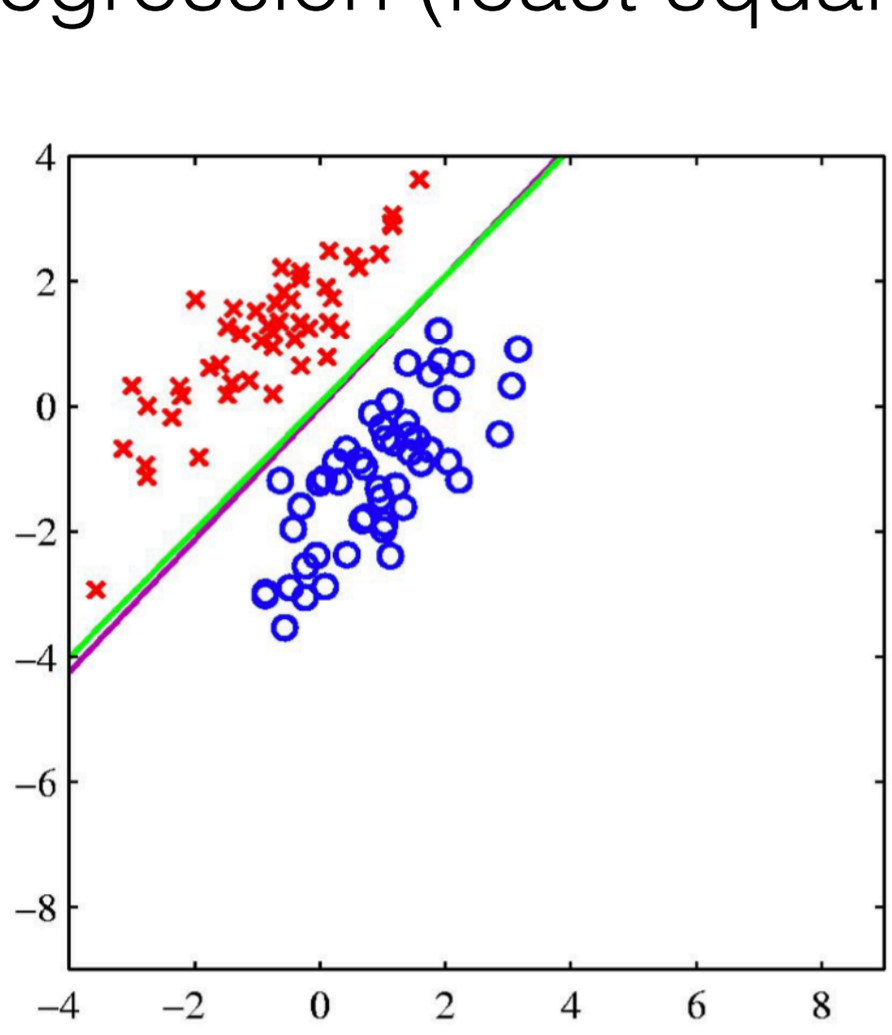
Suppose predict y=1 if $h_\theta(x) \geq 0.5$

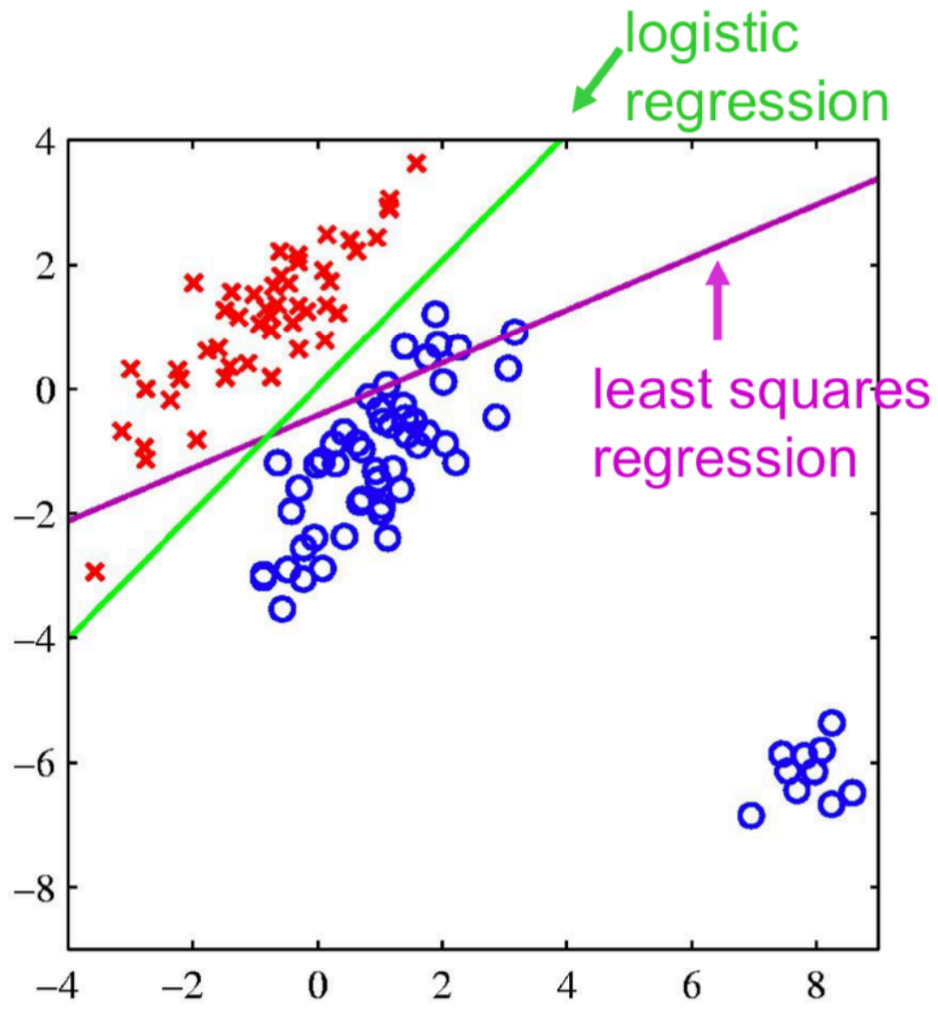
predict y=0 if $h_\theta(x) < 0.5$

Logistic Regression has a linear decision boundary

Logistic *vs* Linear Regression

- A qualitative example of logistic regression vs linear regression (least squares):





If the right answer is 1 and the model says 1.5, it loses, so it changes the boundary to avoid being "too correct" (tilts away from outliers)

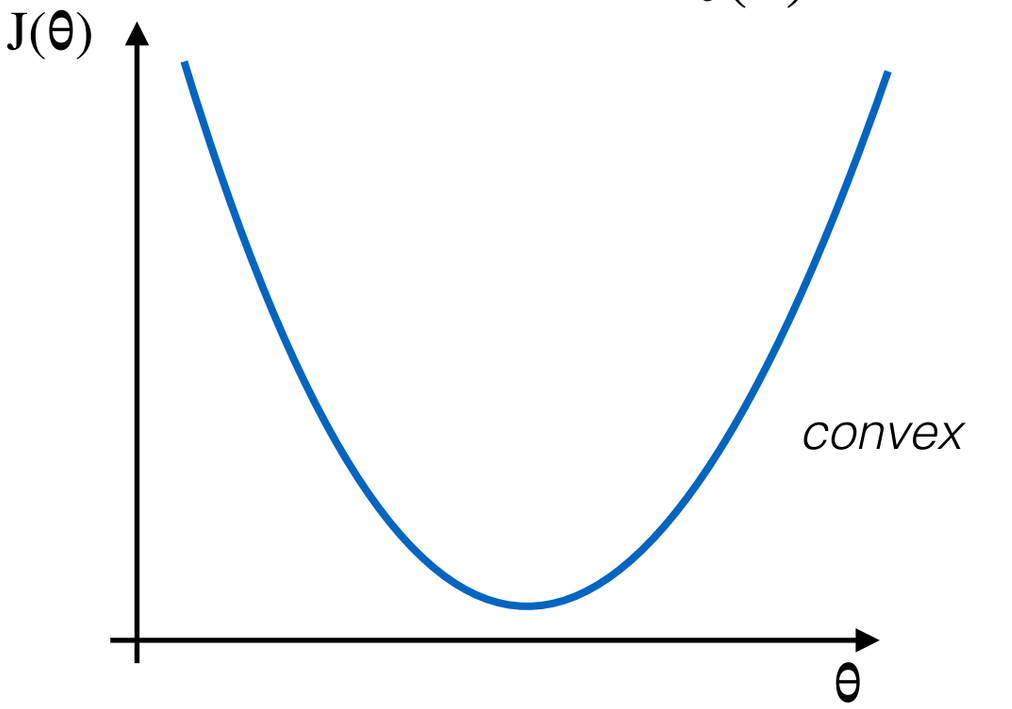
Logistic *vs* Linear Regression

- Loss function: $J(\theta) = \frac{1}{m} \sum_{i=1}^{m} cost(h_\theta(x^{(i)}), y^{(i)})$

where $cost(h_\theta(x^{(i)}), y^{(i)}) = \frac{1}{2} (h_\theta(x^{(i)}) - y^{(i)})^2$

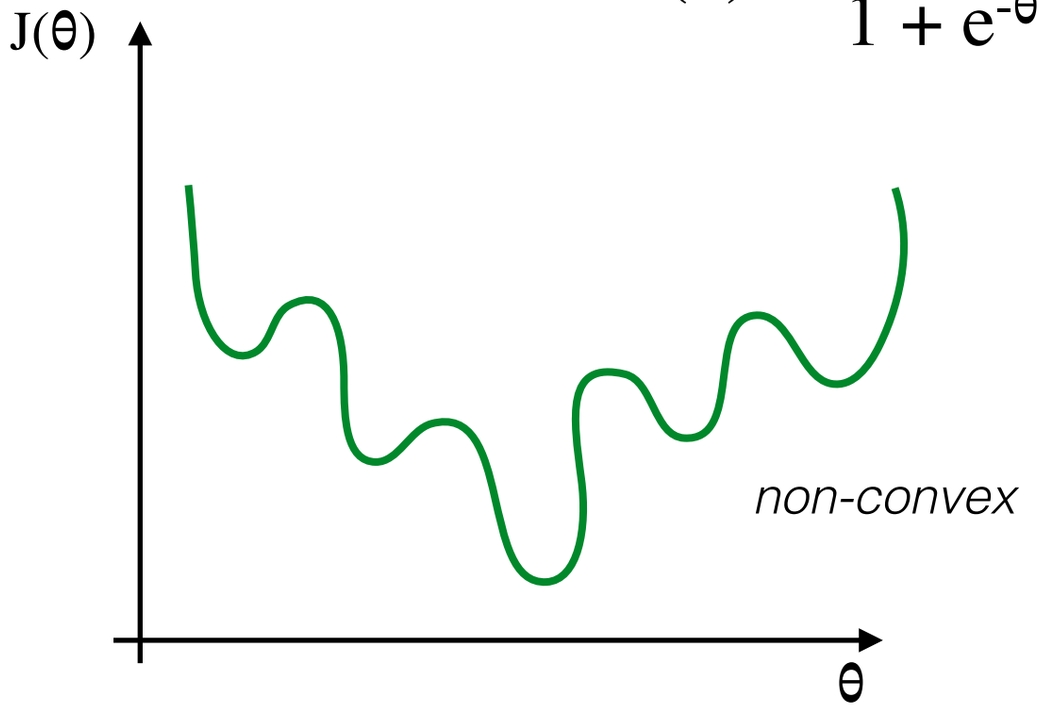
Linear Regression

$h_\theta(\mathbf{x}) = \theta^T \mathbf{x}$



Logistic Regression

$h_\theta(\mathbf{x}) = \frac{1}{1 + e^{-\theta^T \mathbf{x}}}$

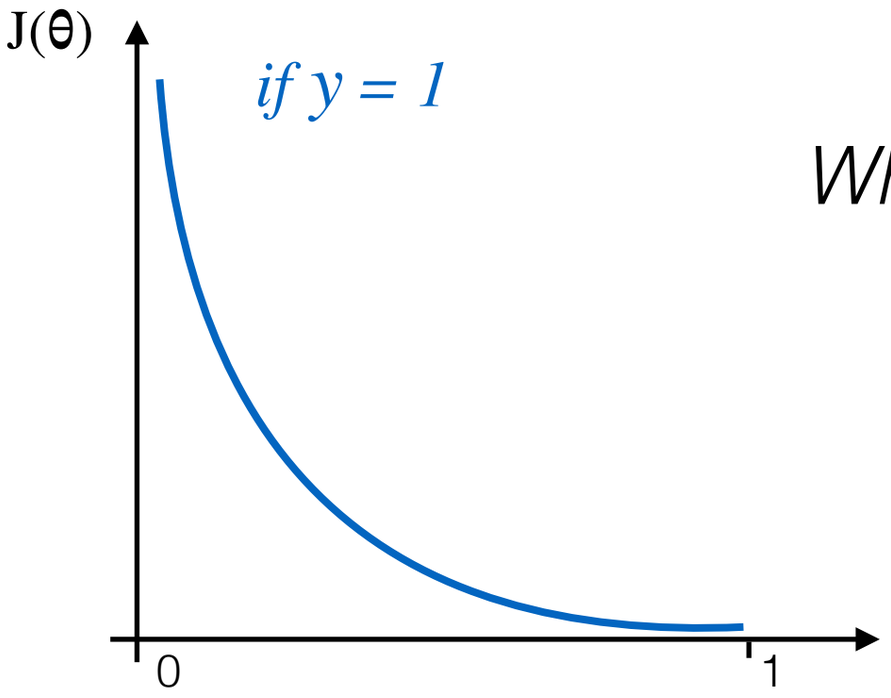


Logistic Regression Loss Function

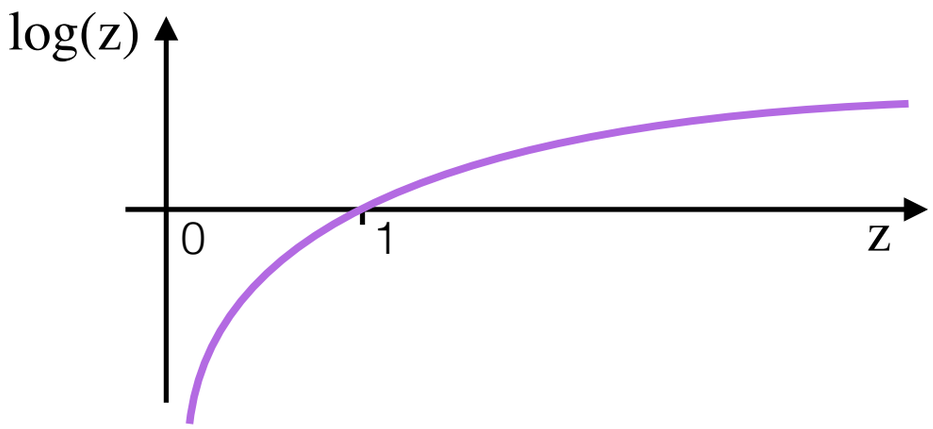
- Loss function: $J(\theta) = \frac{1}{m} \sum_{i=1}^{m} cost(h_\theta(x^{(i)}), y^{(i)})$

where $cost(h_\theta(x^{(i)}), y^{(i)}) = \begin{cases} -\log(h_\theta(x^{(i)})) & \text{if } y^{(i)} = 1 \\ -\log(1 - h_\theta(x^{(i)})) & \text{if } y^{(i)} = 0 \end{cases}$

- Intuition:



Why is that?



$cost = 0$ if $y^{(i)} = 1$ and $h_\theta(x^{(i)}) = 1$

$cost \to \infty$ if $h_\theta(x^{(i)}) \to 0$ (and $y^{(i)} = 1$)

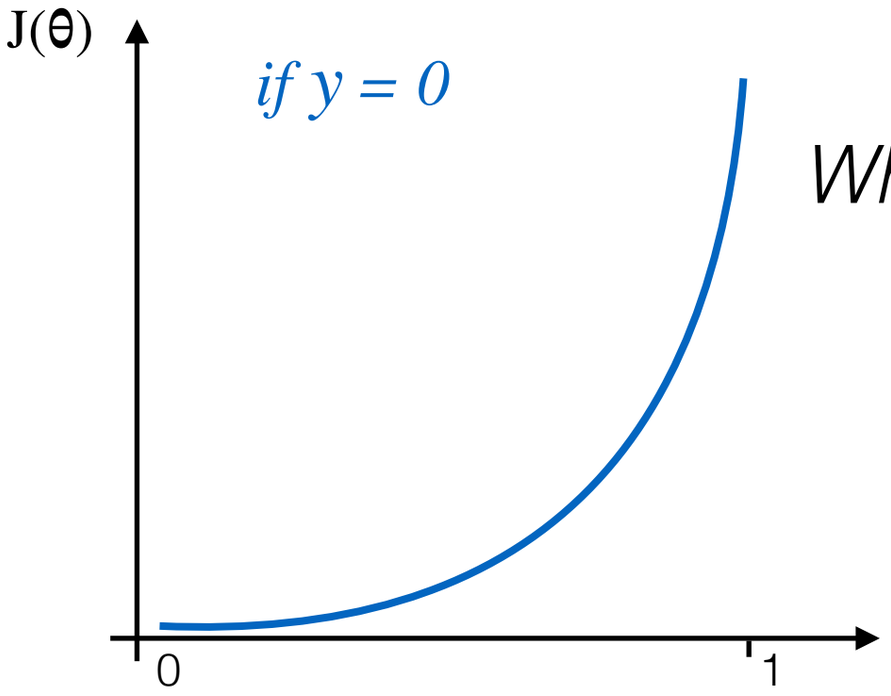
(i.e. predict $P(y=1 \mid x; \theta) = 0$ but $y=1$)

Logistic Regression Loss Function

- Loss function: $J(\theta) = \frac{1}{m} \sum_{i=1}^{m} cost(h_\theta(x^{(i)}), y^{(i)})$

$$\text{where} \quad cost(h_\theta(x^{(i)}), y^{(i)}) = \begin{cases} -\log(h_\theta(x^{(i)})) & \text{if } y^{(i)} = 1 \\ -\log(1 - h_\theta(x^{(i)})) & \text{if } y^{(i)} = 0 \end{cases}$$

- Intuition:



Why is that?

$cost = 0$ if $y^{(i)} = 0$ and $h_\theta(x^{(i)}) = 0$

$cost \to \infty$ if $h_\theta(x^{(i)}) \to 1$ (and $y^{(i)} = 0$)

(i.e. predict $P(y=0 \mid x; \theta) = 1$ but $y=0$)

Logistic Regression Loss Function

- Loss function: $J(\theta) = \frac{1}{m} \sum_{i=1}^{m} cost(h_\theta(x^{(i)}), y^{(i)})$

where $cost(h_\theta(x^{(i)}), y^{(i)}) = \begin{cases} -\log(h_\theta(x^{(i)})) & \text{if } y^{(i)} = 1 \\ -\log(1 - h_\theta(x^{(i)})) & \text{if } y^{(i)} = 0 \end{cases}$

- "Simplified notation":

$$cost(h_\theta(x^{(i)}), y^{(i)}) = -y^{(i)} \cdot \log(h_\theta(x^{(i)})) - (1 - y^{(i)}) \cdot \log(1 - h_\theta(x^{(i)}))$$

$$\Rightarrow J(\theta) = -\frac{1}{m} \sum_{i=1}^{m} y^{(i)} \cdot \log(h_\theta(x^{(i)})) + (1 - y^{(i)}) \cdot \log(1 - h_\theta(x^{(i)}))$$

This is a convex function!

Parameter Learning

- We can learn our parameters with gradient descent

$$J(\theta) = -\frac{1}{m}\sum_{i=1}^{m} y^{(i)} \cdot \log(h_\theta(x^{(i)})) + (1 - y^{(i)}) \cdot \log(1 - h_\theta(x^{(i)}))$$

$$\min_{\theta} J(\theta)$$

Note: this is usually referred as to "cross-entropy loss" or "log-loss"

```
repeat until convergence{
```

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

(simultaneously update all θ_j)

```
}
```

Coming up

- **On Moodle (only):**
  - Our first labs (“refresher”) on ML and Python NumPy

- **Next lectures:**
  - Introduction to computer vision
  - Early vision; image formation, filtering, low-level features