Machine Learning – COMS3007

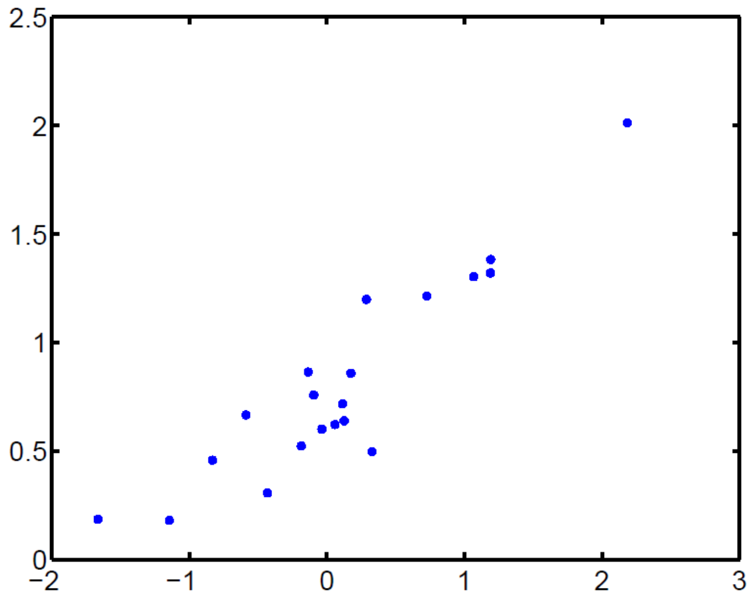
Linear Regression

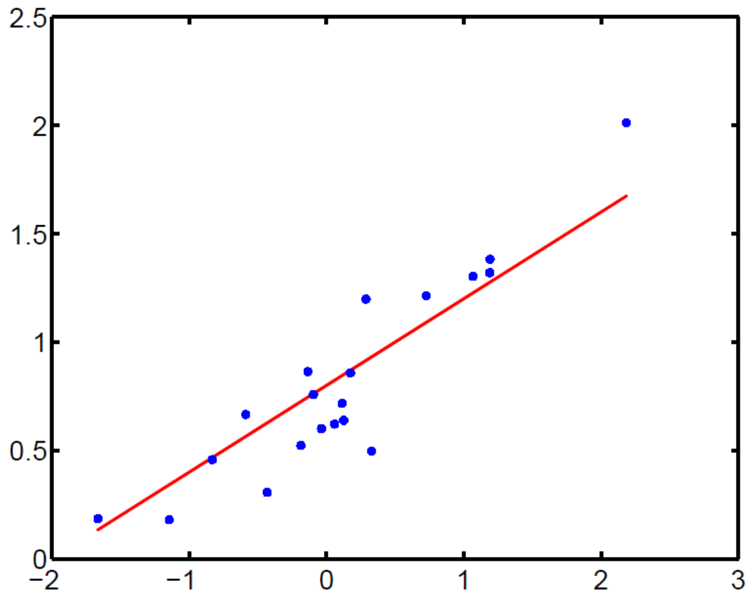
Benjamin Rosman

Based heavily on course notes by Chris Williams and Victor Lavrenko, Amos Storkey, Eric Eaton, and Clint van Alten

Regression

- Data $X = \{x^{(0)}, \ldots, x^{(n)}\}$, where $x^{(i)} \in R^d$
- Labels $\mathbf{y} = \{y^{(0)}, \ldots, y^{(n)}\}$, where $y^{(i)} \in R$
- Want to learn function $y = f(x, \theta)$ to predict y for a new x





Linear regression – model

What is this noise? Where might it come from?

- Assume model: $y = a + bx + \eta$
  - η is Gaussian noise (don't model explicitly)
- Higher dimensions:
  - $y = f(x, \theta) = \sum_{i=0}^{d} \theta_i x_i$
  - $y = \boldsymbol{\theta}^T \boldsymbol{x}$
- Note: we call them weights w and parameters θ interchangably

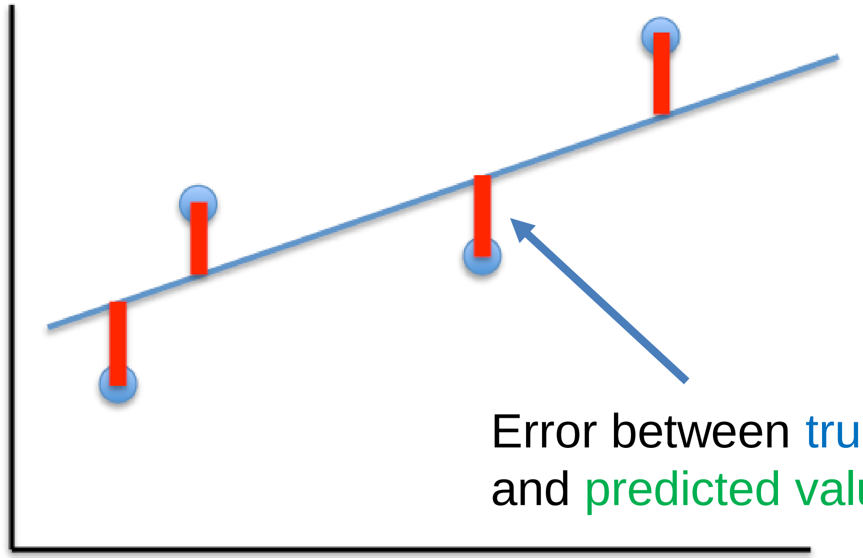
Treat 0^{th} dimension of x as 1 (i.e. the "intercept")

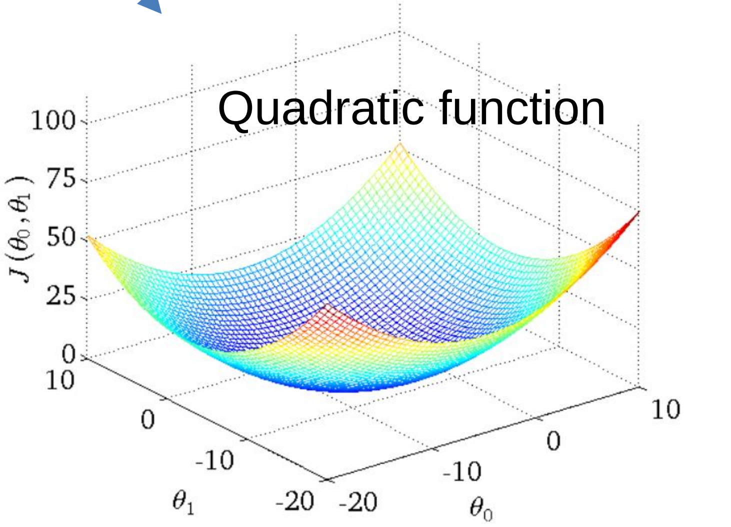
- This is linear regression: the model is **linear in the parameters**

Linear regression – cost function

- Infinitely many choices of $\boldsymbol{\theta}$
  - Learning = finding the best ones
- To choose among them, we need a cost function
- $E(\theta) = \frac{1}{2}\sum_{i=1}^{n}\left(y^{(i)} - f(x^{(i)}, \theta)\right)^2$

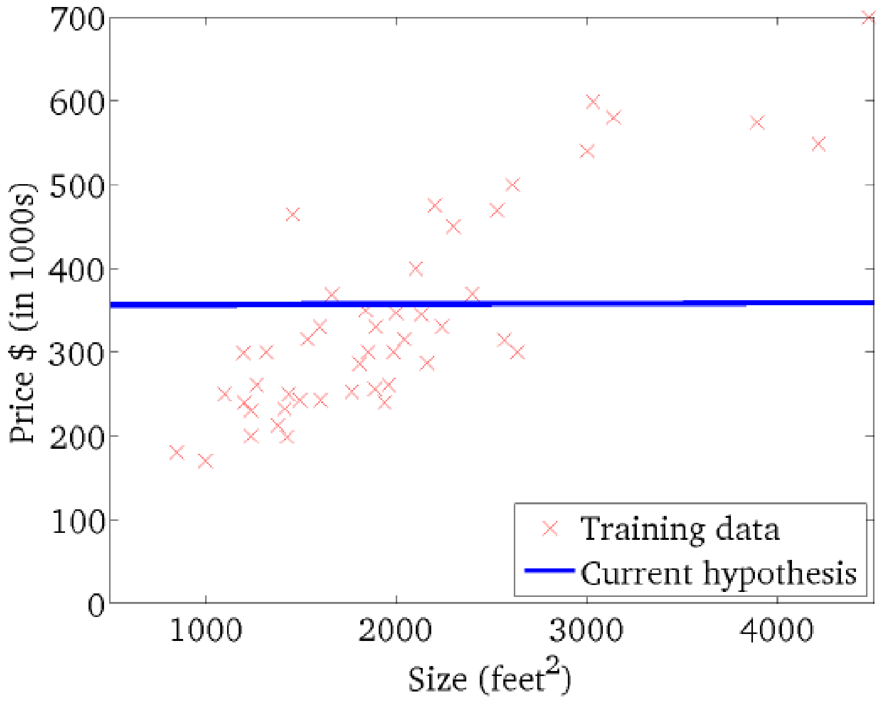
The ½ coefficient may instead be a 1, or a (1/2n). It doesn't matter, as long as it is consistent.
It can also be labelled J() instead of E()



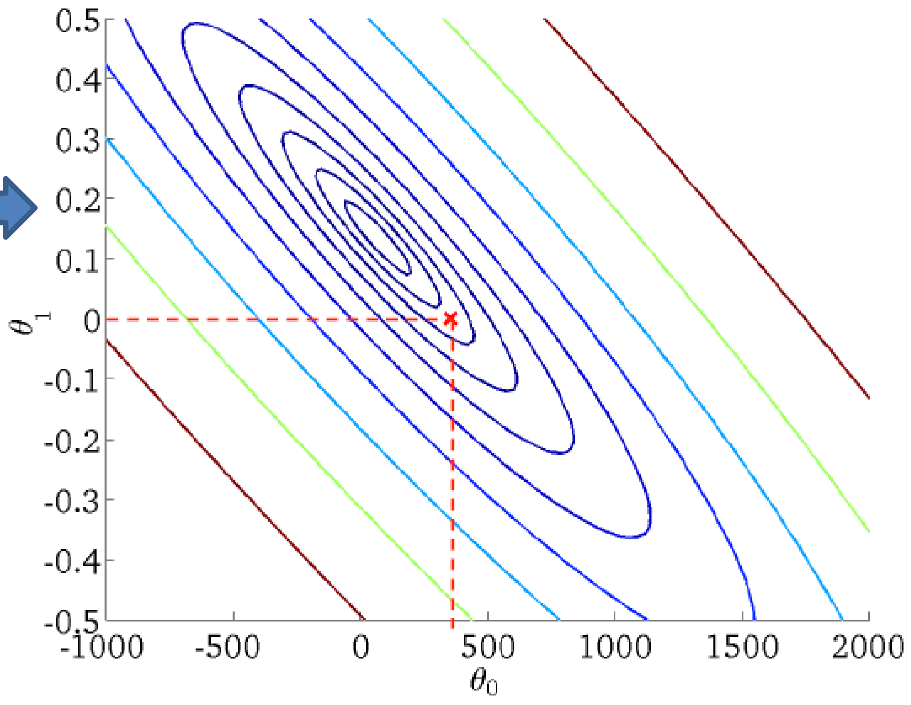


Linear regression – cost function

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



Error
(function of the parameters θ_0, θ_1)

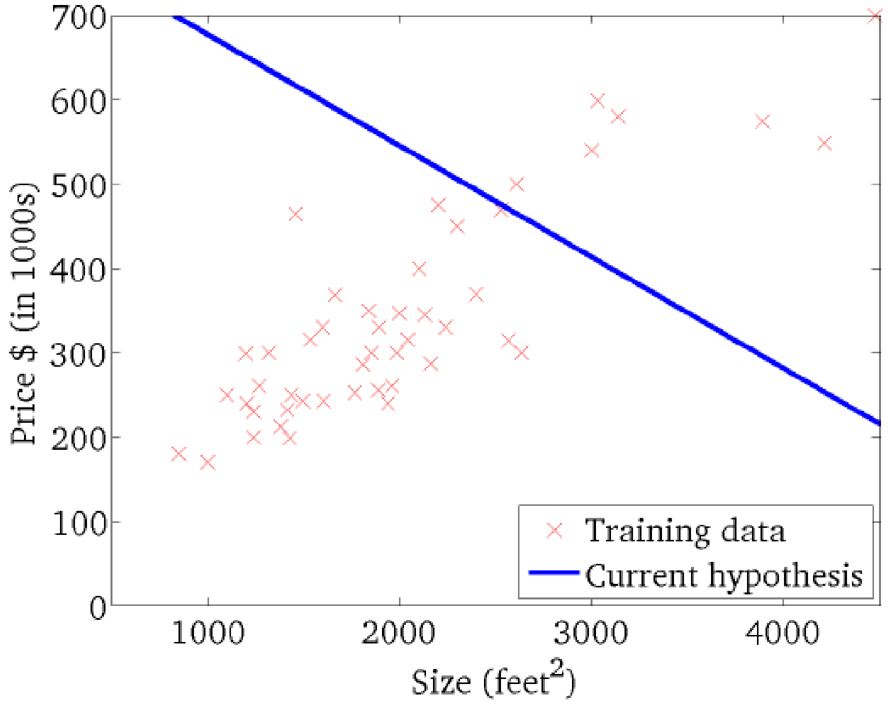


"weight space"

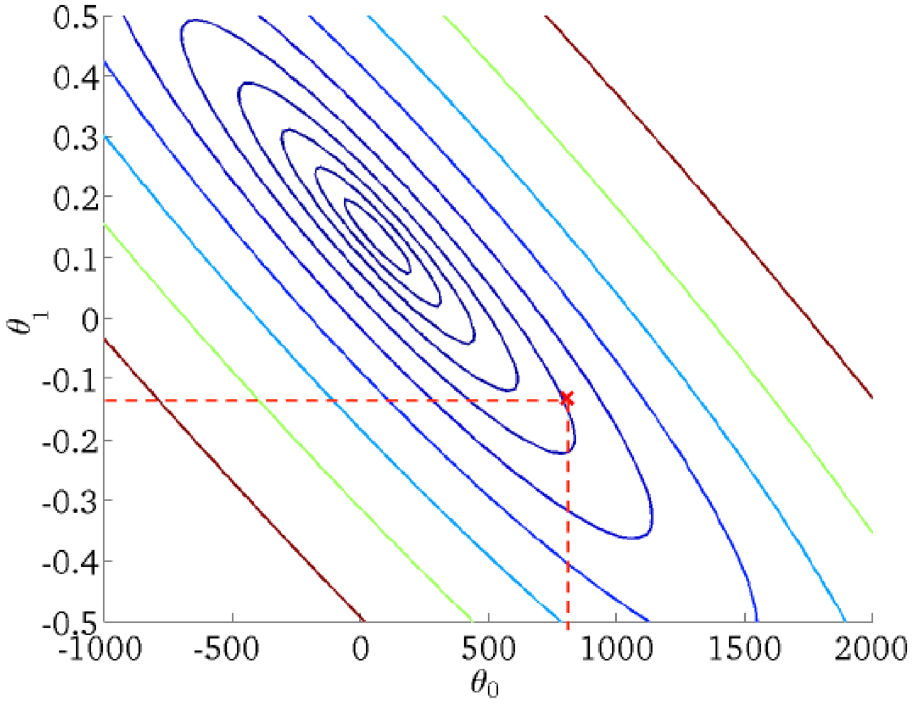
Every point here defines a regression line in the original problem

Linear regression – cost function

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

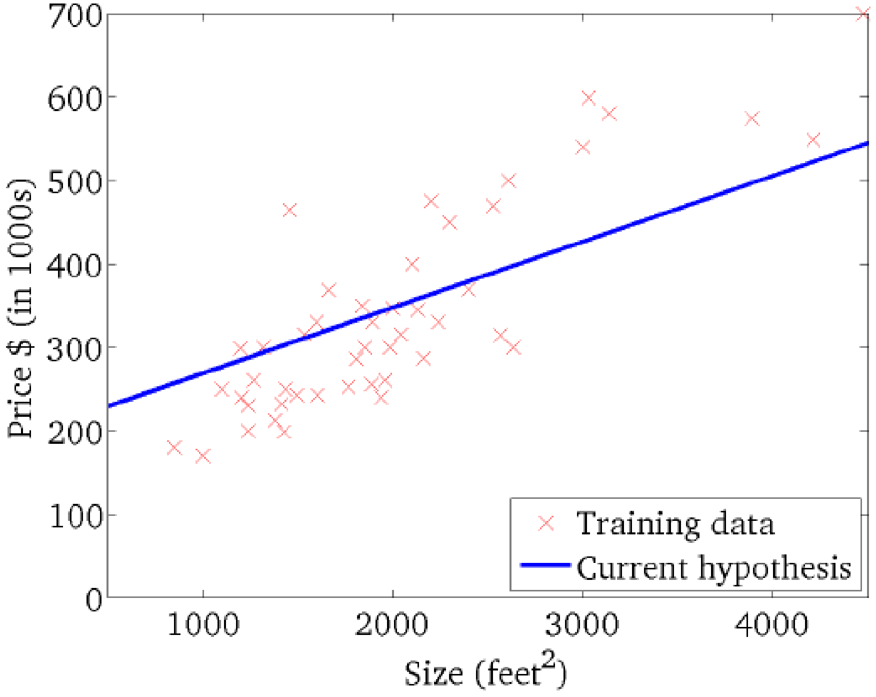


(function of the parameters θ_0, θ_1)

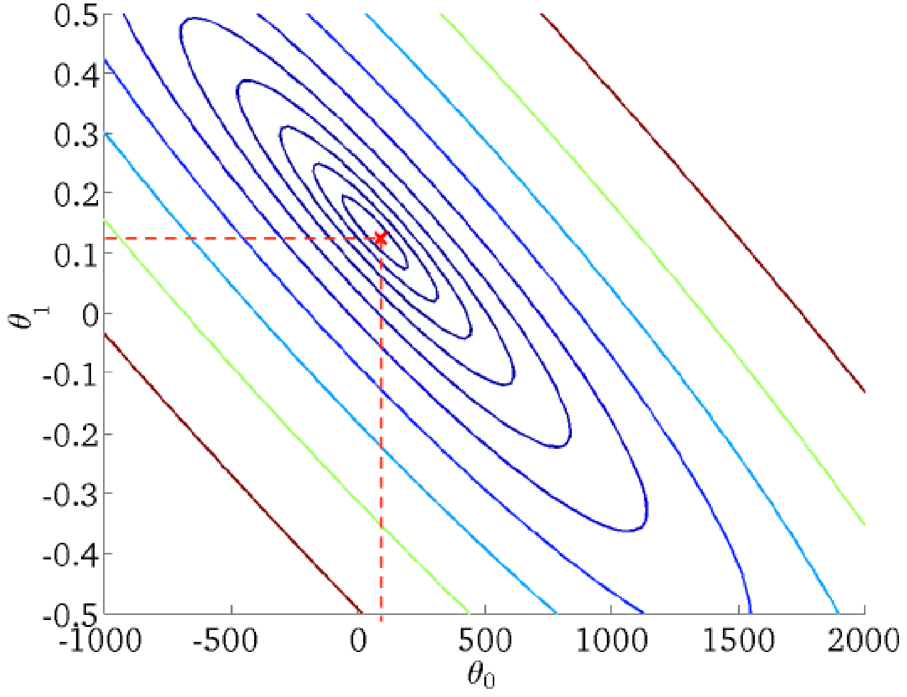


Linear regression – cost function

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



(function of the parameters θ_0, θ_1)



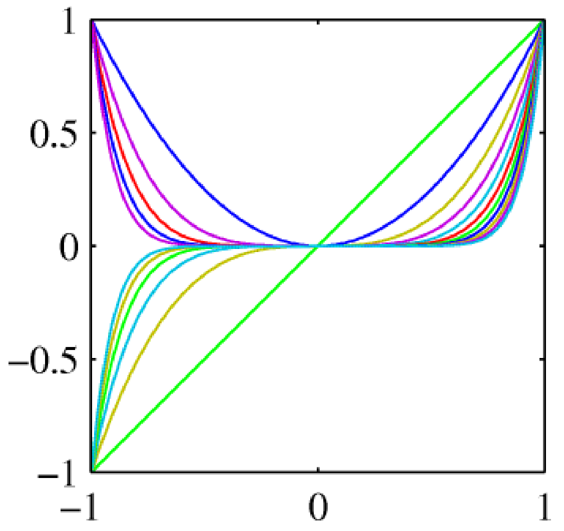
Basis functions

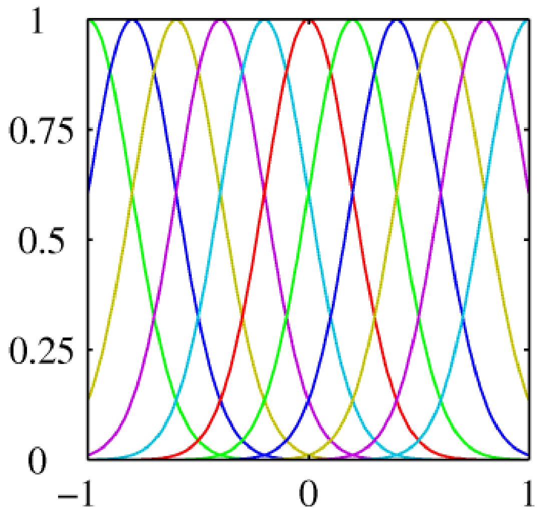
These basis functions are the features: good features are needed for regression to work!

- What makes it linear?
  - $y = f(x, \theta) = \sum_{i=0}^{d} \theta_i x_i$
  - Linear in θ, NOT x
- So: we can use different functions of x
  - "Basis functions" $\phi_j(x)$
  - Still linear regression
- Example: let $x \in R^3$, i.e. $x = (x_1, x_2, x_3)^T$
  - Possible basis functions:
  - $x_1,\ x_1^5,\ x_1 x_2,\ x_3^2 x_2,\ \sin(x_2),\ \log(x_3),\ e^{-\frac{1}{2\sigma^2}(x_1-\mu)^2}, \ldots$
- Rewrite: $y = f(x, \theta) = \sum_{i=0}^{d} \theta_i \phi_i(x)$

Basis functions

- How to choose basis functions $\phi_j(x)$?
  - Assumptions about the data
  - Try as many as possible
- Polynomial basis functions:
  - $\phi_j(x) = x^j$
  - Can include cross-terms
    - e.g. $x_3^2 x_2$
  - Global: any change in x affects all basis functions
- Gaussian basis functions:
  - Radial basis functions (RBF)
  - $\phi_j(x) = e^{-\frac{1}{2\sigma^2}(x-\mu_j)^2}$
  - Local: change in x affects nearby basis functions
  - μ_j = location, σ = scale/width





Design matrix

- Structure data in a design matrix $\mathbf{X}$
- Let there be n data points (vectors): $\boldsymbol{x}^{(1)}, \boldsymbol{x}^{(2)}, \dots, \boldsymbol{x}^{(n)}$
- And d+1 basis functions: $\phi_0, \phi_1, \dots, \phi_d$
- Design matrix $\Phi = \begin{bmatrix} \phi_0\left(\boldsymbol{x}^{(1)}\right) & \phi_1\left(\boldsymbol{x}^{(1)}\right) & \cdots & \phi_d\left(\boldsymbol{x}^{(1)}\right) \\ \phi_0\left(\boldsymbol{x}^{(2)}\right) & \phi_1\left(\boldsymbol{x}^{(2)}\right) & & \phi_d\left(\boldsymbol{x}^{(2)}\right) \\ \vdots & & \ddots & \vdots \\ \phi_0\left(\boldsymbol{x}^{(n)}\right) & \phi_1\left(\boldsymbol{x}^{(n)}\right) & \cdots & \phi_d\left(\boldsymbol{x}^{(n)}\right) \end{bmatrix}$

- So far, we've had: $\phi_0(\boldsymbol{x}) = 1, \phi_1(\boldsymbol{x}) = x_1, \phi_2(\boldsymbol{x}) = x_2,$ etc…

- Now we can work with the data in matrix form!

Closed form solution

- How do we learn the function?
- We want to minimise the error:
  - $E(\theta) = \frac{1}{2}\sum_{i=1}^{n}\left(y^{(i)} - f(x^{(i)}, \theta)\right)^2$
  - Rewrite: $E(\theta) = \frac{1}{2}(\boldsymbol{y} - \mathbf{X}\theta)^2$

Vector notation
with design matrix

  - $E(\theta) = \frac{1}{2}(\boldsymbol{y} - \mathbf{X}\theta)^T(\boldsymbol{y} - \mathbf{X}\theta)$
  - $E(\theta) = \frac{1}{2}(\boldsymbol{y}^T\boldsymbol{y} - 2\theta^T\mathbf{X}^T\boldsymbol{y} + \theta^T\mathbf{X}^T\mathbf{X}\theta)$
  - For minimum: differentiate wrt θ and set to zero

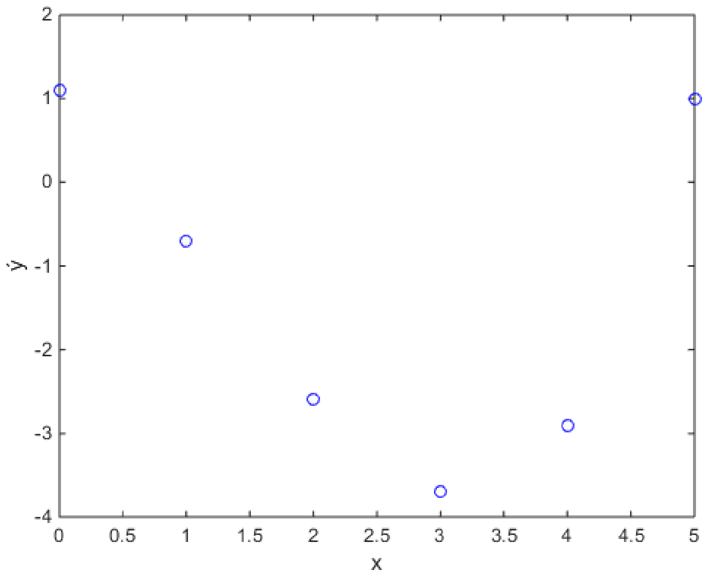
  - $\theta = \underbrace{\left(\mathbf{X}^{\mathrm{T}}\mathbf{X}\right)^{-1}\mathbf{X}^{\mathrm{T}}}_{\text{The Moore-Penrose pseudo-inverse}}\mathbf{y}$

Example

- True function: $f(x) = 0.2x^3 - 0.8x^2 - x + 1 + \eta$
  - Unknown to algorithm

- Training data:

x	y
0	1.1
1	-0.7
2	-2.6
3	-3.7
4	-2.9
5	1



Example

- Choose model: $f(x, \theta) = \theta_0 + \theta_1 x + \theta_2 x^2 + \theta_3 x^3$
- Error: $\mathrm{E}(\theta) = \frac{1}{2}\sum_{i=1}^{n}\left(y^{(i)} - f\left(x^{(i)}, \theta\right)\right)^2$

x	y
0	1.1
1	-0.7
2	-2.6
3	-3.7
4	-2.9
5	1

Design matrix $X =$

1	x	x^2	x^3
1	0	0	0
1	1	1	1
1	2	4	8
1	3	9	27
1	4	16	64
1	5	25	125

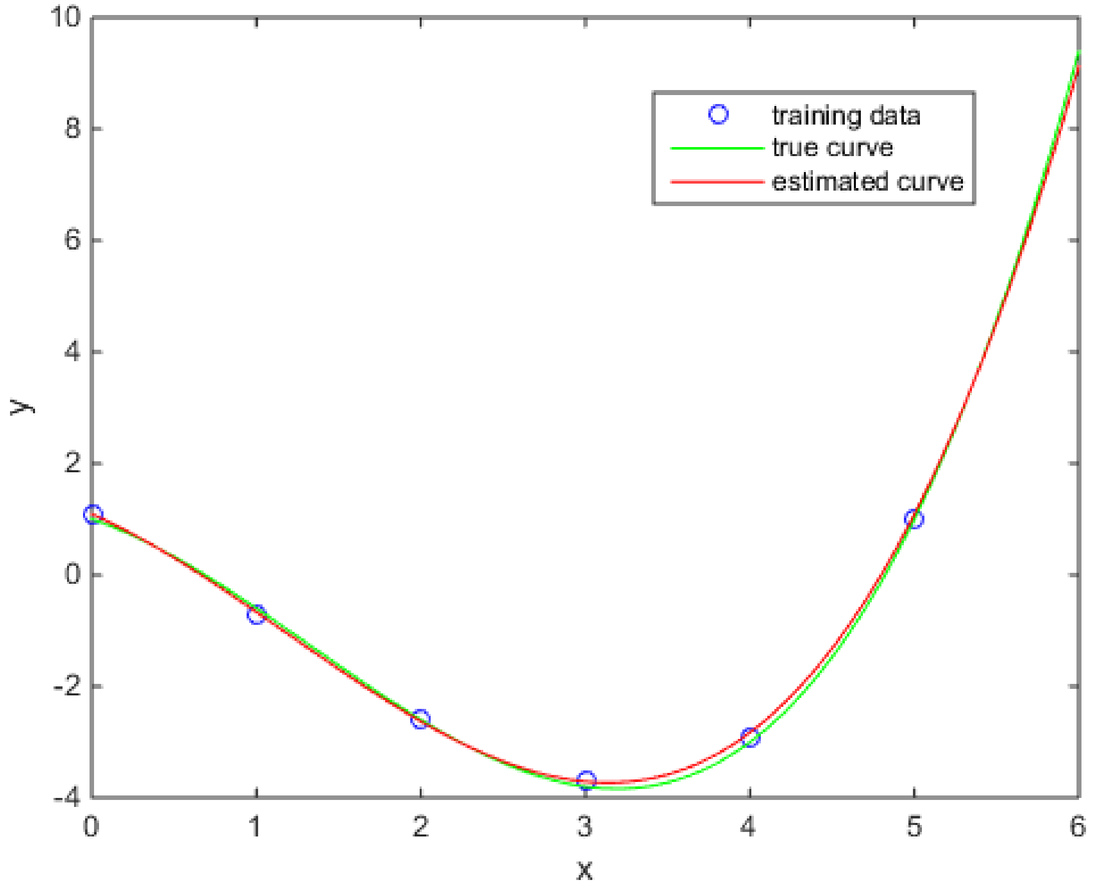
- $\theta = \left(\mathbf{X}^{\mathrm{T}}\mathbf{X}\right)^{-1}\mathbf{X}^{\mathrm{T}}\mathbf{y} = [1.09,\ -1.3,\ \ -0.64,\ \ \ 0.18]^{T}$

$f(x) = \quad 1 \qquad -x \qquad -0.8x^2 \quad +0.2x^3$

Example

$$f(x) = 1 - x - 0.8x^2 + 0.2x^3$$

- $\theta = (\mathbf{X}^{\mathrm{T}}\mathbf{X})^{-1}\mathbf{X}^{\mathrm{T}}\mathbf{y} = [1.09, -1.3, -0.64, 0.18]^{\boldsymbol{T}}$

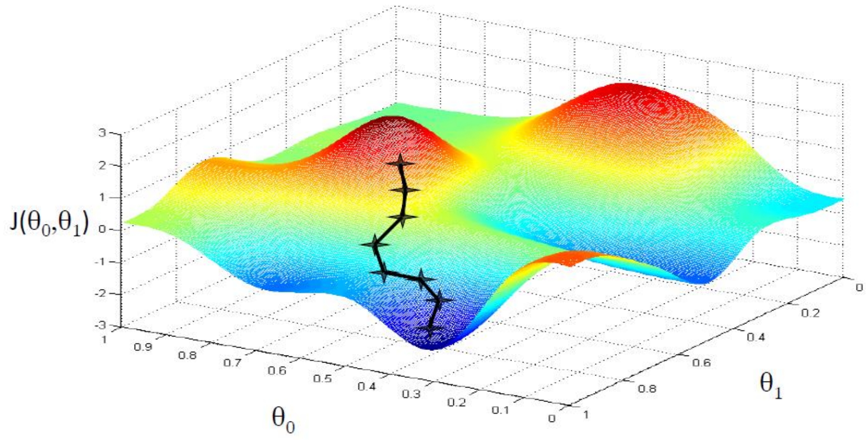


Gradient descent

- Closed form solutions are not always appropriate
  - Can be slow: computing $\left(\mathbf{X}^{\mathrm{T}}\mathbf{X}\right)^{-1}$ is roughly $O(n^3)$
  - Cannot learn incrementally (redo computation for a new data point)
  - Different error functions?

- Instead use iterative procedure: gradient descent (GD)

- Basic idea:
  - Choose initial θ
  - Until a minimum:
    - Update θ to reduce $\mathrm{J}(\theta)$

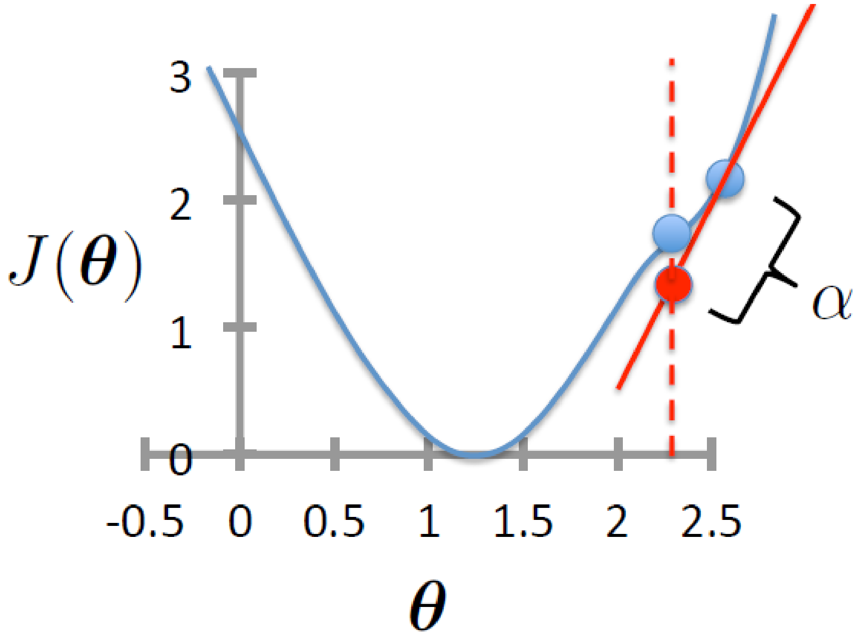


Gradient descent

Cost function = E = J

- Initialise θ
- Repeat until convergence:
  - $\theta_j \leftarrow \theta_j - \alpha \frac{\partial}{\partial \theta_j} J(\theta)$
  - Simultaneous update for $j = 0, \ldots, d$

$0 < \alpha \leq 1$ is the learning rate, usually set quite small



Take a step of size α in the "downhill" direction (negative gradient)

Gradient descent

- Consider point *i*
- $J(\theta) = E(\theta) = \frac{1}{2}\left(y^{(i)} - f(x^{(i)}, \theta)\right)^2$
- For $f\left(x^{(i)}, \theta\right) = \sum_{j=0}^{d} \theta_j x_j^{(i)}$,
  - $\frac{\partial}{\partial \theta_j} J(\theta) = (f\left(x^{(i)}, \theta\right) - y^{(i)})x_j^{(i)}$

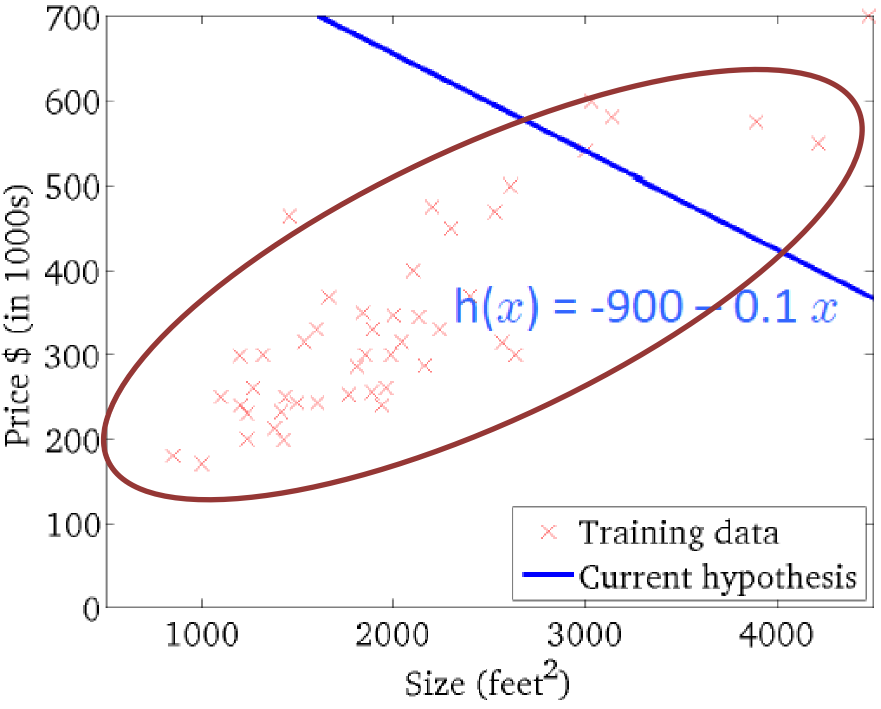
Stop when $||\theta_{new} - \theta_{old}||_2 < \epsilon$

- So:
- Initialise θ
- Repeat until convergence:
  - For each datapoint *i*:
  - $\theta_j \leftarrow \theta_j - \alpha(f\left(x^{(i)}, \theta\right) - y^{(i)})x_j^{(i)}$
  - Simultaneous update for $j = 0, \dots, d$

Gradient descent example

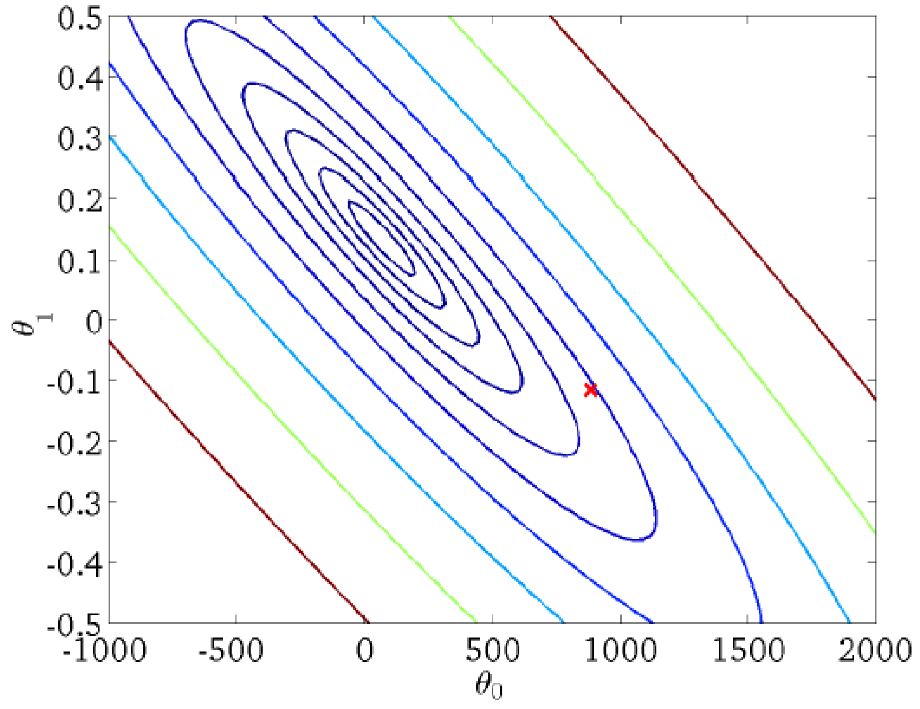
$h_\theta(x)$

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



$J(\theta_0, \theta_1)$

(function of the parameters θ_0, θ_1)

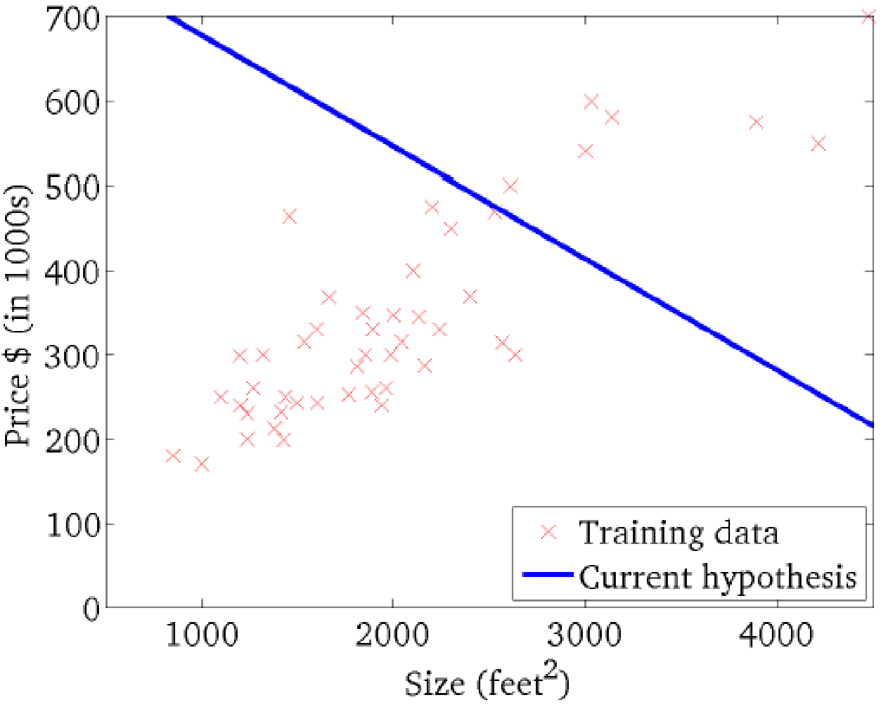


We will be taking downhill steps

Gradient descent example

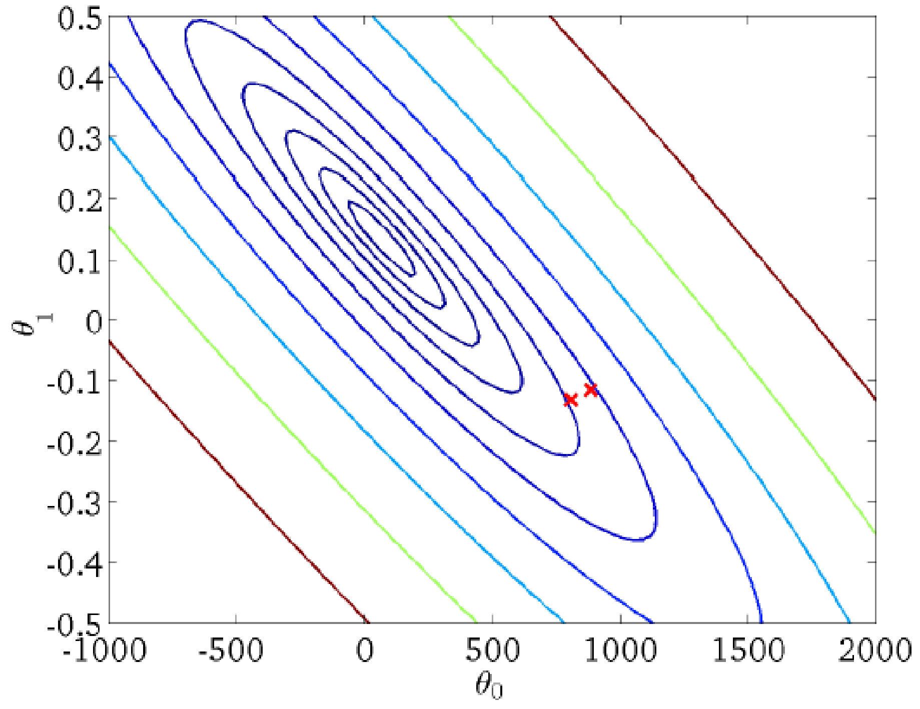
$h_\theta(x)$

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



$J(\theta_0, \theta_1)$

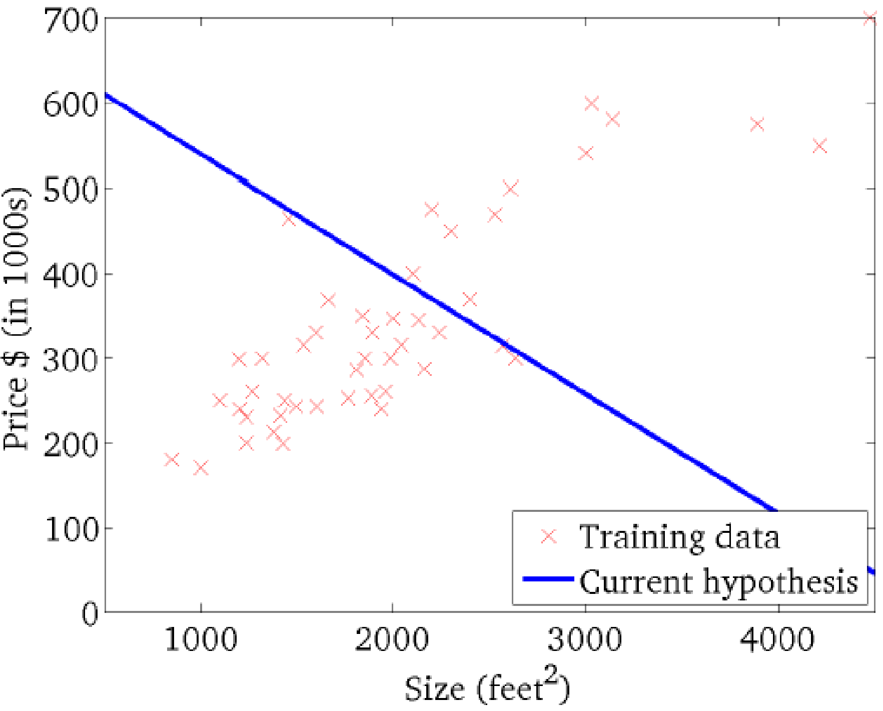
(function of the parameters θ_0, θ_1)



Gradient descent example

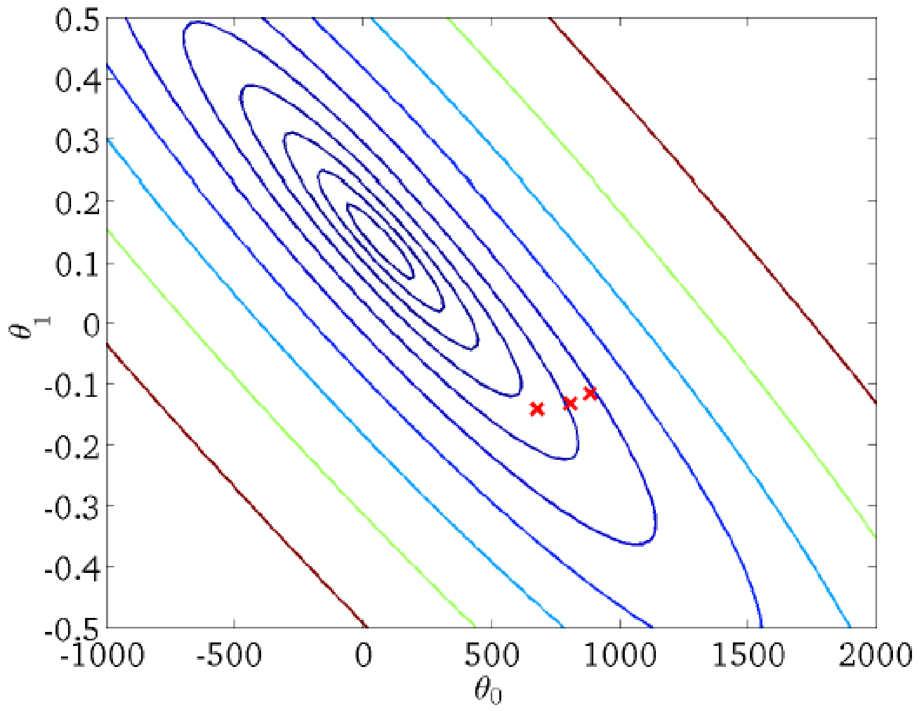
$h_\theta(x)$

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



$J(\theta_0, \theta_1)$

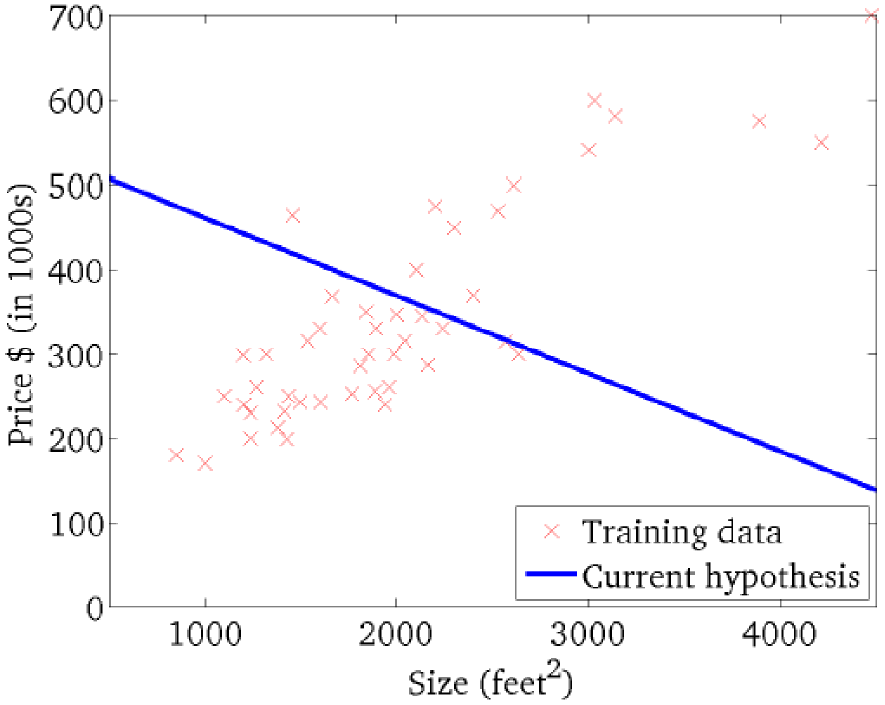
(function of the parameters θ_0, θ_1)



Gradient descent example

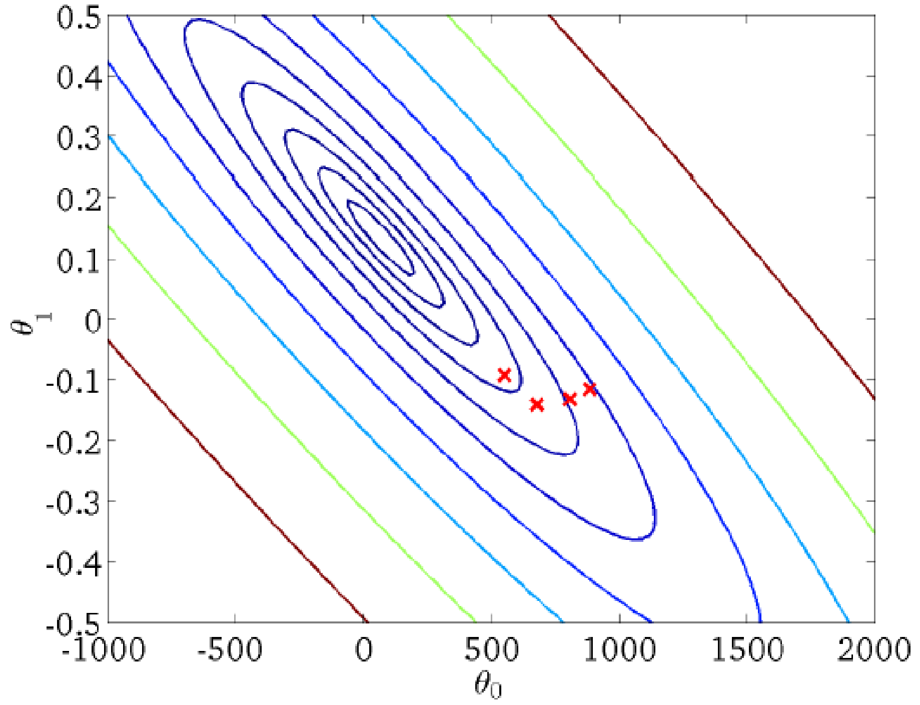
$h_\theta(x)$

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



$J(\theta_0, \theta_1)$

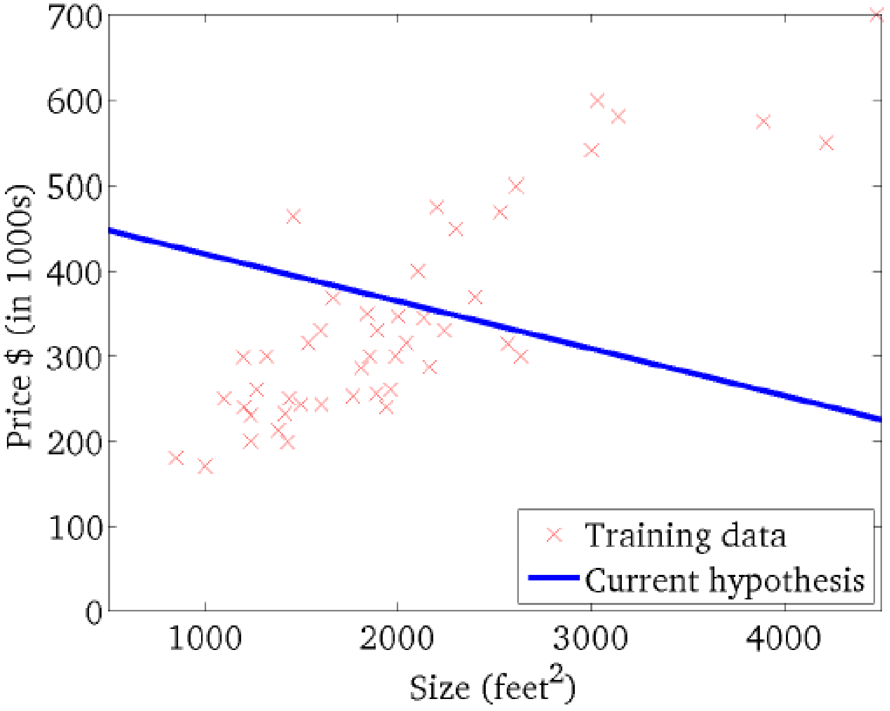
(function of the parameters θ_0, θ_1)



Gradient descent example

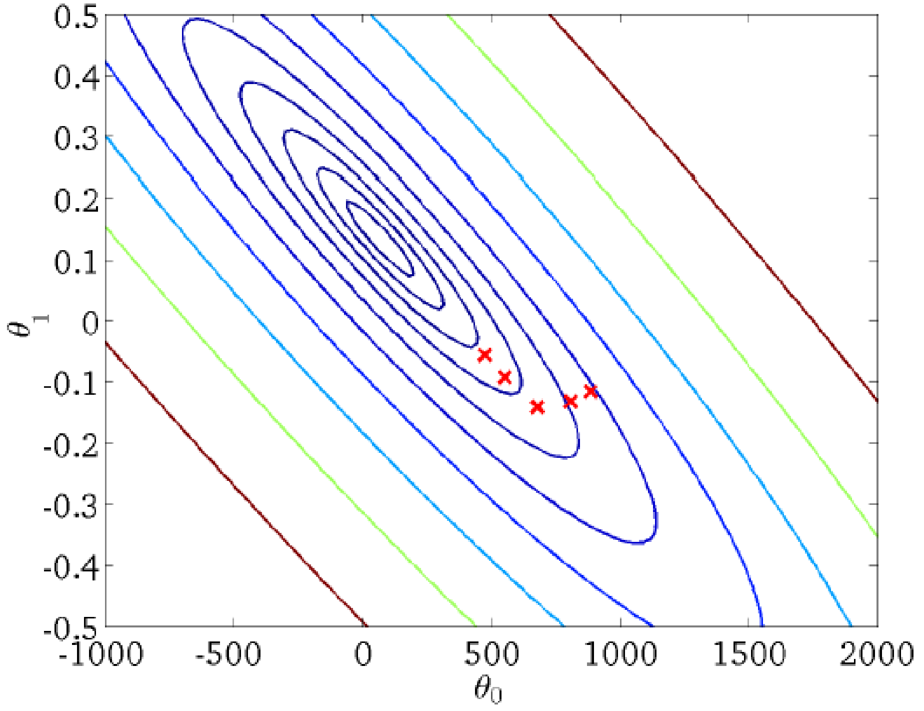
$h_\theta(x)$

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



$J(\theta_0, \theta_1)$

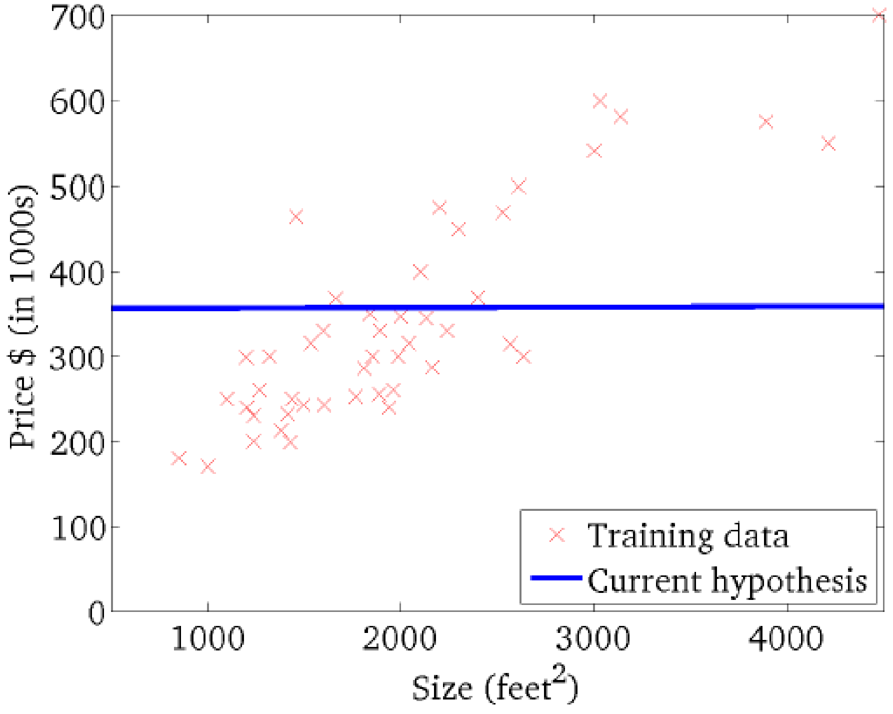
(function of the parameters θ_0, θ_1)



Gradient descent example

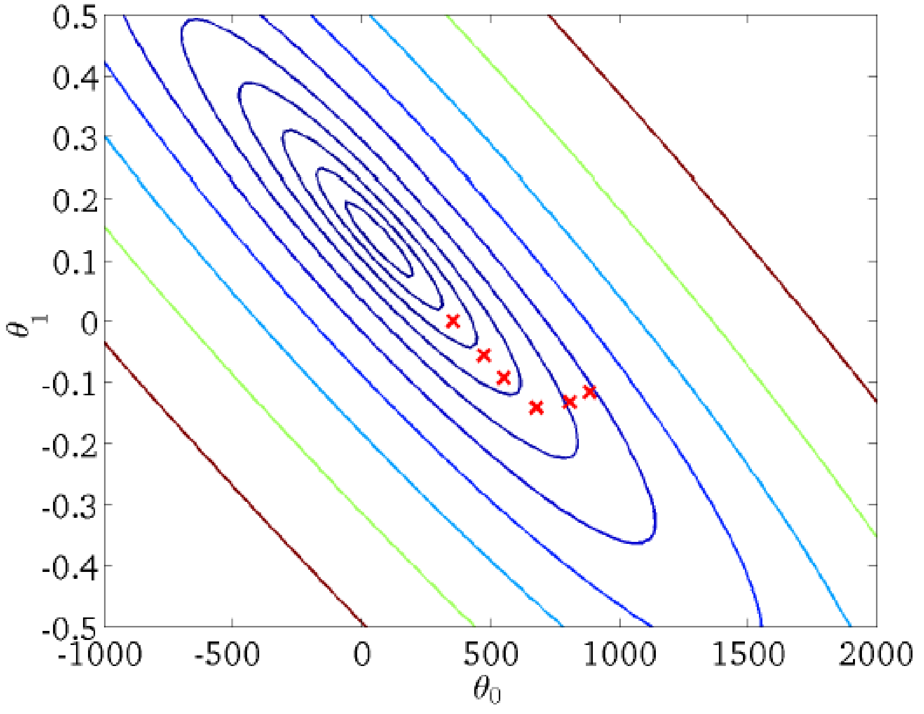
$h_\theta(x)$

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



$J(\theta_0, \theta_1)$

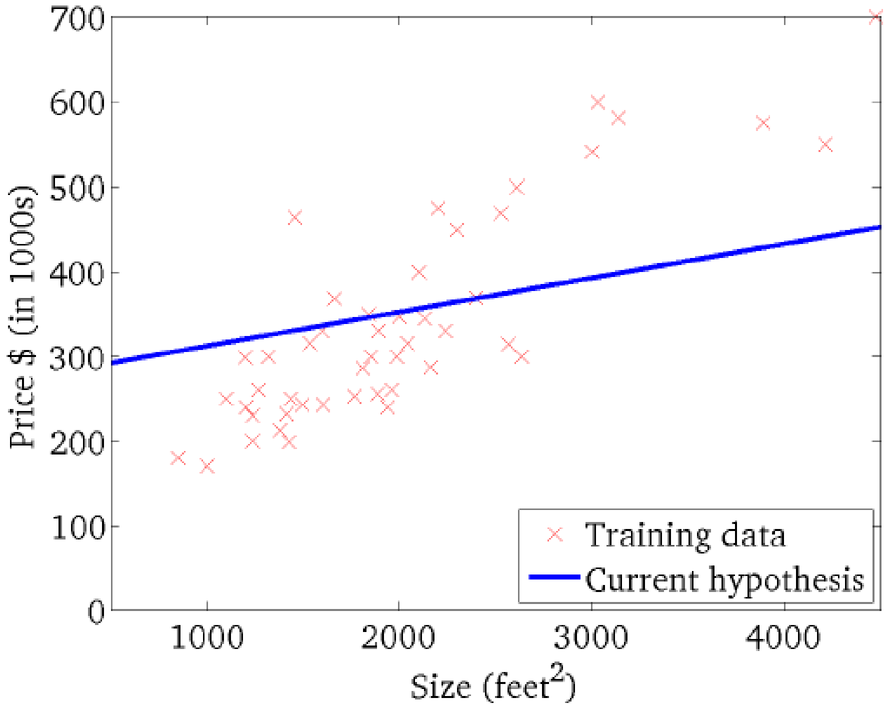
(function of the parameters θ_0, θ_1)



Gradient descent example

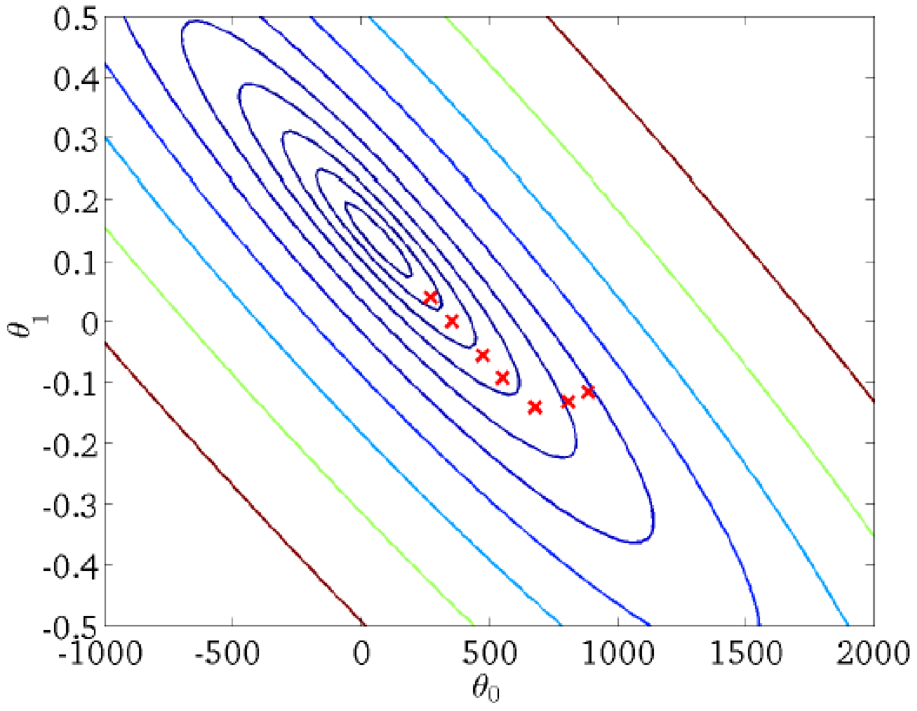
$h_\theta(x)$

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



$J(\theta_0, \theta_1)$

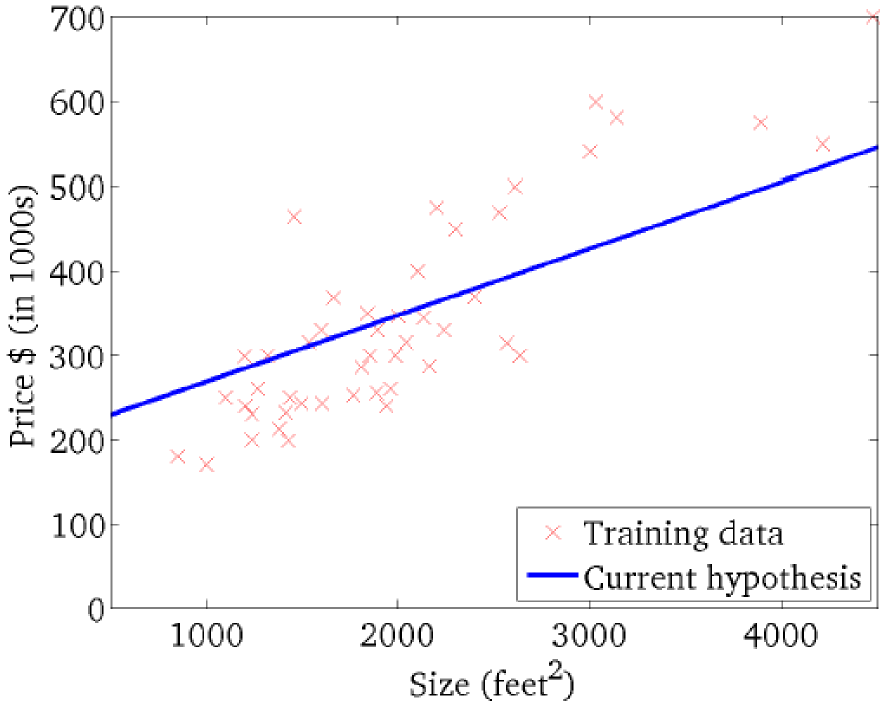
(function of the parameters θ_0, θ_1)



Gradient descent example

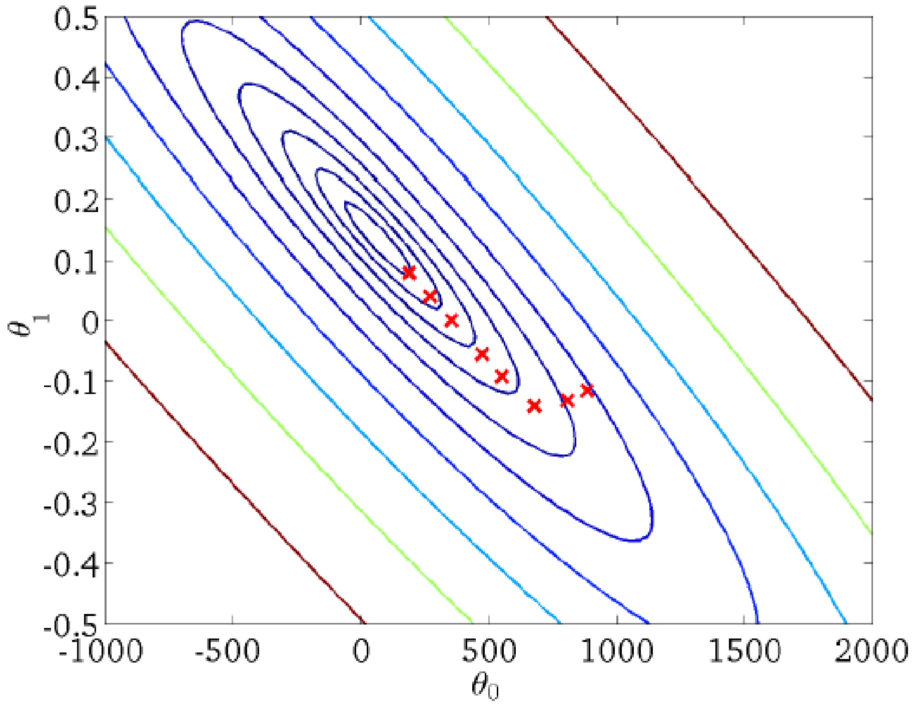
$h_\theta(x)$

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



$J(\theta_0, \theta_1)$

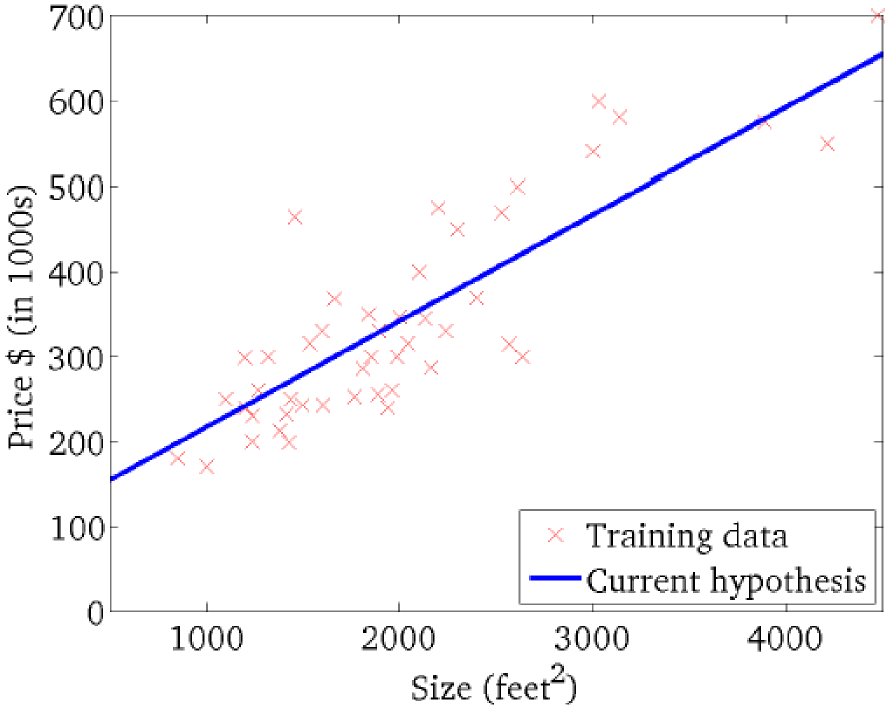
(function of the parameters θ_0, θ_1)



Gradient descent example

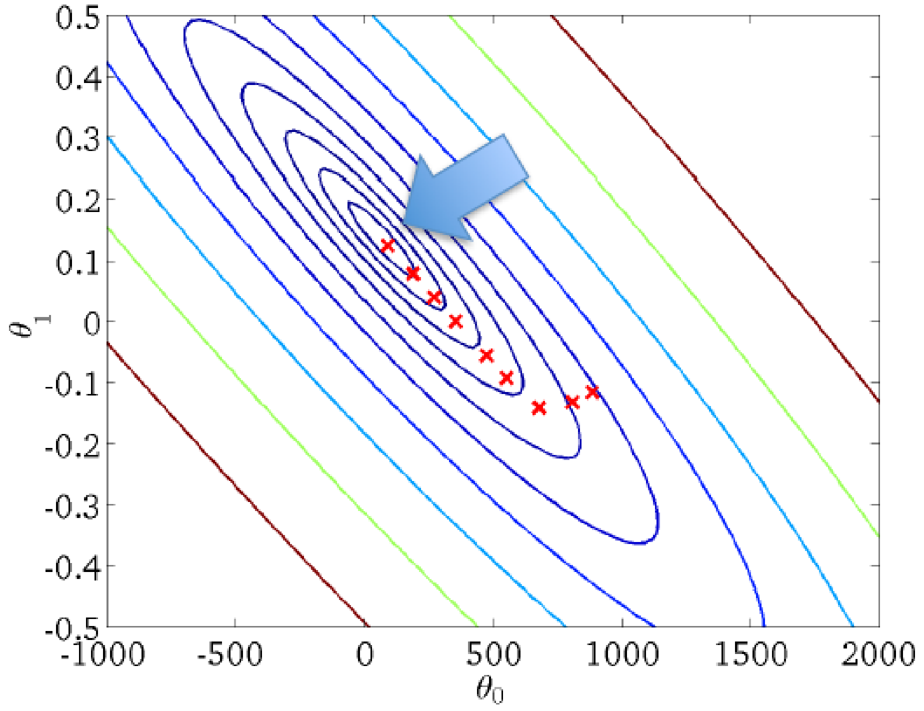
$h_\theta(x)$

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



$J(\theta_0, \theta_1)$

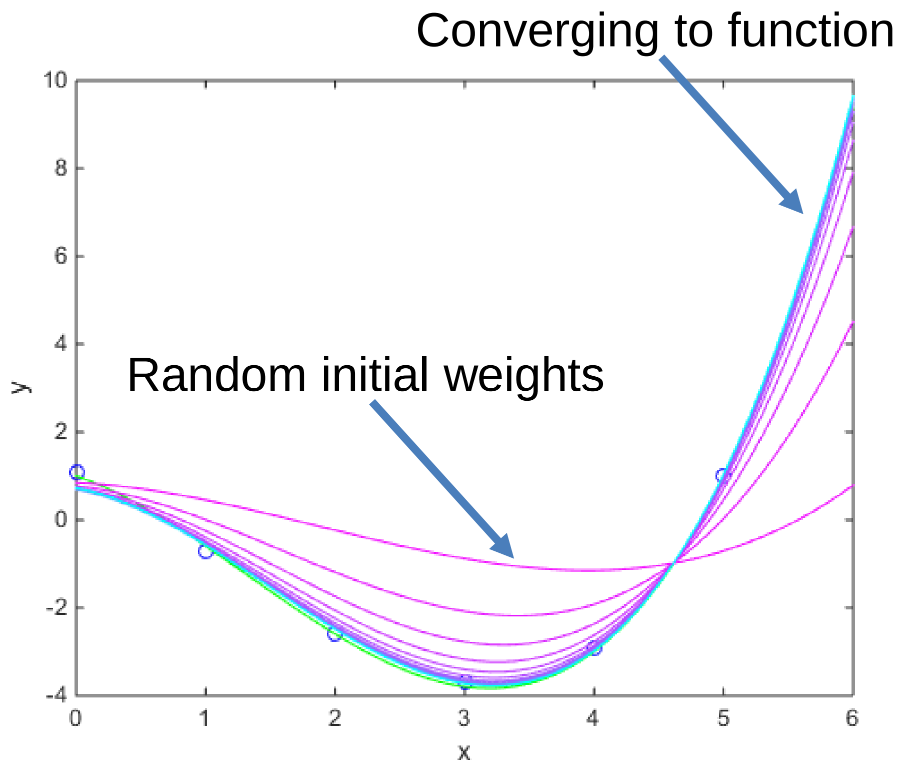
(function of the parameters θ_0, θ_1)



Example

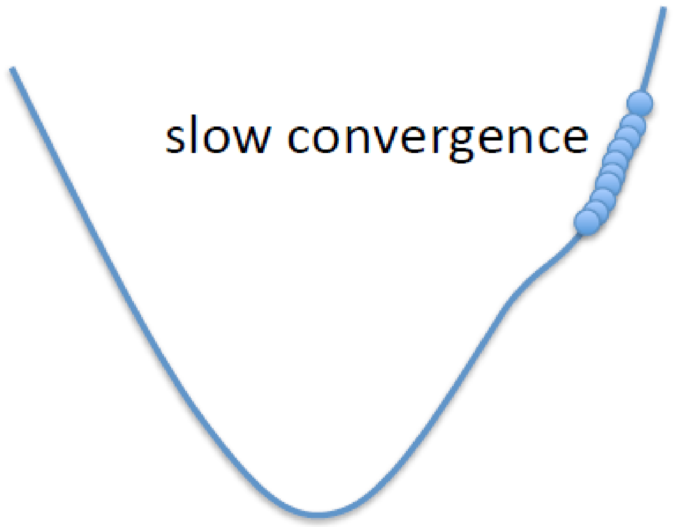
- $\theta_0 \leftarrow \theta_0 - \alpha(f(x,\theta) - y^{(i)})$
- $\theta_1 \leftarrow \theta_1 - \alpha\left(f(x,\theta) - y^{(i)}\right)x$
- $\theta_2 \leftarrow \theta_2 - \alpha\left(f(x,\theta) - y^{(i)}\right)x^2$
- $\theta_3 \leftarrow \theta_3 - \alpha\left(f(x,\theta) - y^{(i)}\right)x^3$

$$f(x,\theta) = \theta_0 + \theta_1 x + \theta_2 x^2 + \theta_3 x^3$$

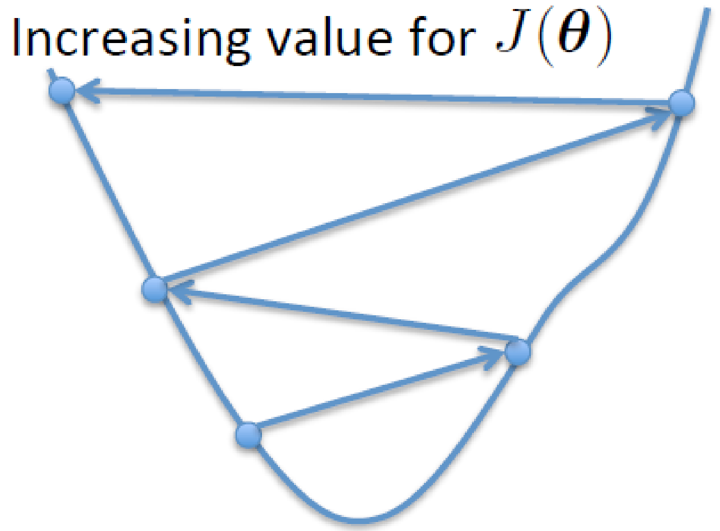


The effect of α

α too small

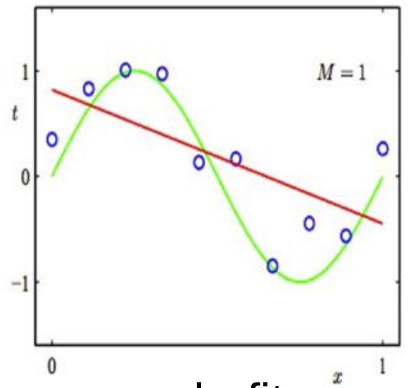


α too large

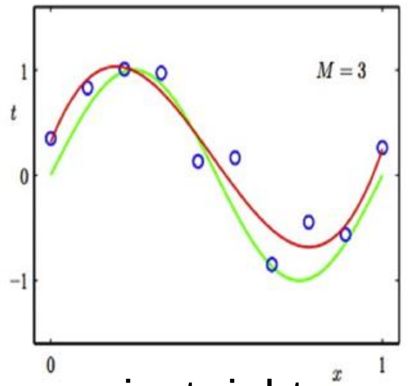


- May overshoot the minimum
- May fail to converge
- May even diverge

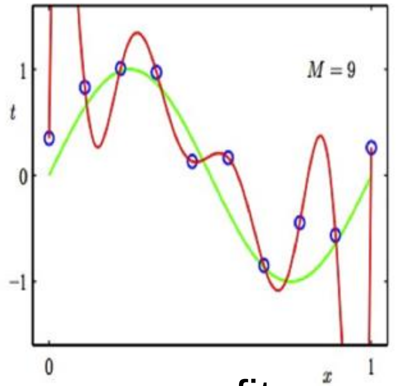
Overfitting and underfitting



underfit



just right



overfit

- We don’t just care about matching the training data, we want to generalise
- Too few (or the wrong) basis functions underfits
- But, too many overfits!
- Can we automate this?

Regularisation

- $\theta_0 + \theta_1 x + \theta_2 x^2 + \theta_3 x^3$
- $\approx \theta_0 + \theta_1 x$ if $\theta_2, \theta_3 \approx 0$

- Idea: penalise large parameter values
  - Change the cost function

- $E(\theta) = \frac{1}{2}\sum_{i=1}^{n}\left(y^{(i)} - f\left(x^{(i)}, \theta\right)\right)^2 + \lambda \sum_{j=1}^{d} \theta_j^2$

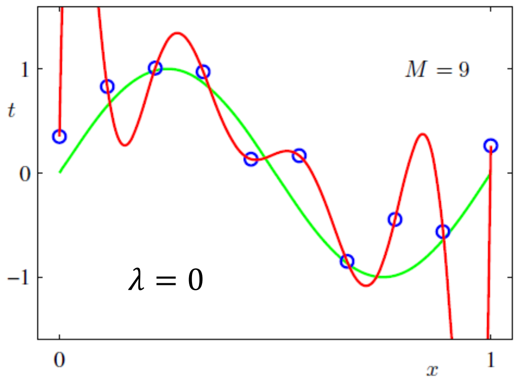
  Fit the data regularise

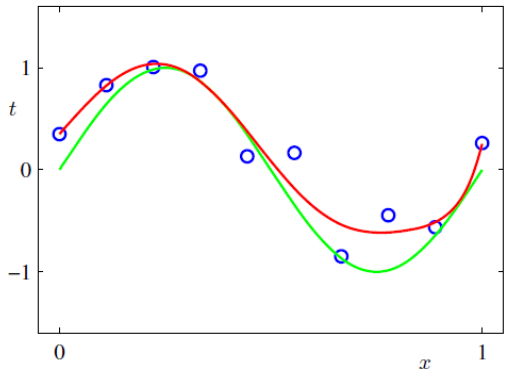
- $\lambda \geq 0$ controls amount of regularisation
- Note: do not regularise θ_0

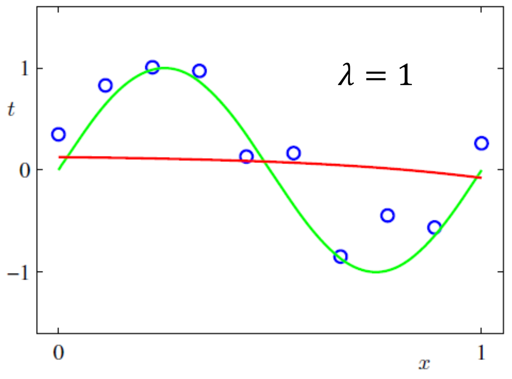
While minimising E, we want to fit the data AND not have large parameters:
So, it will only grow parameters if the fit becomes much better!

The effect of λ

- $\lambda = 0$: no regularisation
  - Linear regression as normal
- As λ increases (no upper bound):
  - It acts as an increasing force keeping parameters small unless really necessary
  - Tends to a straight line







Closed-form with regularisation

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

No regularisation on θ_0

- $\theta = \left(\mathbf{X}^{\mathrm{T}}\mathbf{X} + \lambda \begin{bmatrix} 0 & 0 & 0 & & 0 \\ 0 & 1 & 0 & \cdots & 0 \\ 0 & 0 & 1 & & 0 \\ & \vdots & & \ddots & \vdots \\ 0 & 0 & 0 & \cdots & 1 \end{bmatrix} \right)^{-1} \mathbf{X}^{\mathrm{T}}\mathbf{y}$

- Can derive this as before

GD with regularisation

- Initialise θ
- Repeat until convergence:
  - $\theta_0 \leftarrow \theta_0 - \alpha(f\left(x^{(i)}, \theta\right) - y^{(i)})$
  - $\theta_j \leftarrow \theta_j - \alpha\left[\left(f\left(x^{(i)}, \theta\right) - y^{(i)}\right)x_j^{(i)} + \lambda\theta_j\right]$
  - Simultaneous update for $j = 0, \dots, d$

No regularisation on θ_0

Recap

- Model
- Cost function
- Basis functions
- Design matrix
- Closed-form solution
- Gradient descent
- Regularisation