ENCS5341
Machine Learning and Data Science

Regression

Introduction

- Regression is a supervised learning task where the target variable that we are trying to predict is continuous. Examples: predicting houses prices based on the living area, predicting stock price based on the history of previous prices.

- When there is a single input variable (x), the method is referred to as simple linear regression. E.g.: predicting blood pressure as a function of drug dose.

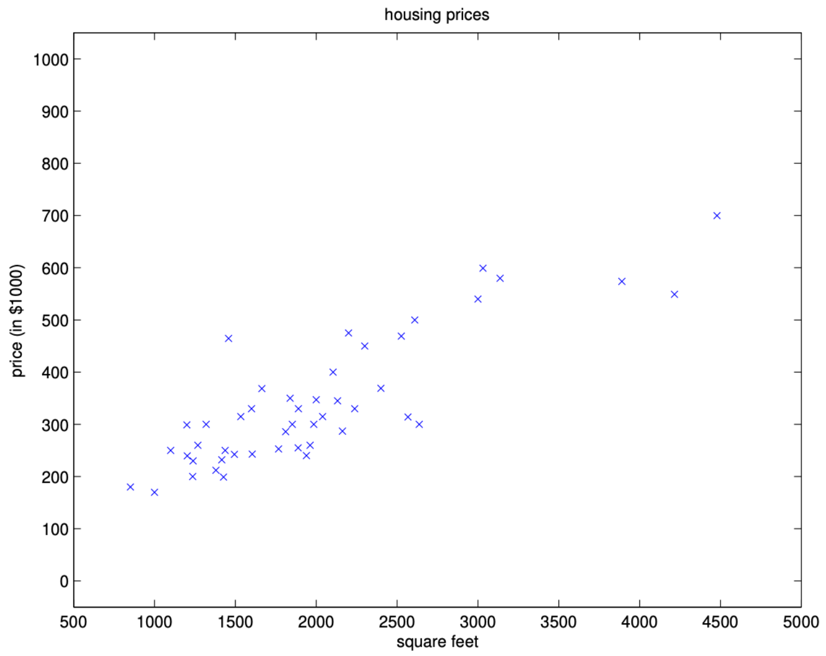
- When there are multiple input variables, literature from statistics often refers to the method as multiple linear regression. E.g.: predicting crop yields as a function of fertilizer and water.

- Linear regression is a model that assumes a linear relationship between the input variables (x) and the single output variable (y). More specifically, that y can be calculated from a linear combination of the input variables (x)

Linear regression example with one variable

- Suppose we have a dataset giving the living areas and prices of houses.

Living area (feet^2)	Price (1000\$s)
2104	400
1600	330
2400	369
1416	232
3000	540
⋮	⋮



- With linear regression we would learn a function with the form

$$y = f(x) = w_0 + w_1 x$$

where y is the predicted house price, and x is the input feature (living area in this example)

Linear regression example with more than one variable

- Now assume that we have two features: living areas, and the number of bedrooms

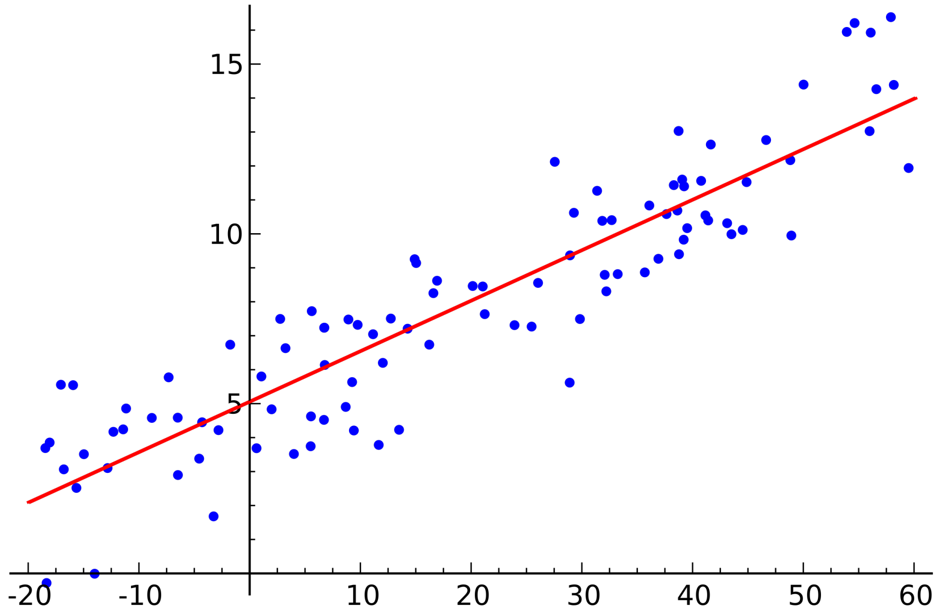
Living area (feet^2)	#bedrooms	Price (1000$s)
2104	3	400
1600	3	330
2400	3	369
1416	2	232
3000	4	540
$\vdots$	$\vdots$	$\vdots$

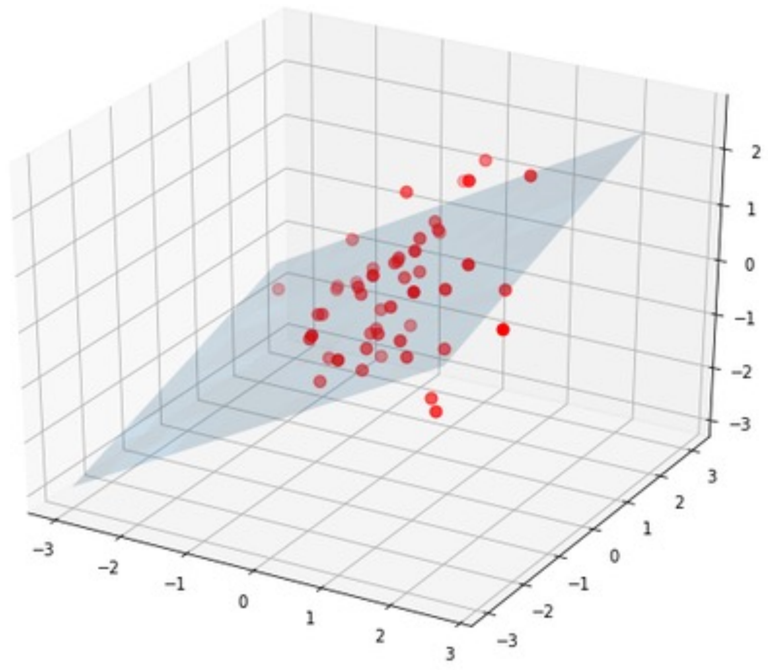
- In this case, our linear regression model will have the form

$$y = f(x) = w_0 + w_1x_1 + w_2\, x_2$$

where y is the predicted house price, x_1 is the first feature (living area), x_2 is the second feature (number of bedrooms), and $x = (x_1,x_2)^T$ is the input features

Linear regression





For x in $\mathbb{R}$, linear regression fits a line in a 2- dimensional space (simple linear regression)

For x in $\mathbb{R}^2$, linear regression fits a plane in a 3- dimensional space (multiple linear regression)

Linear regression

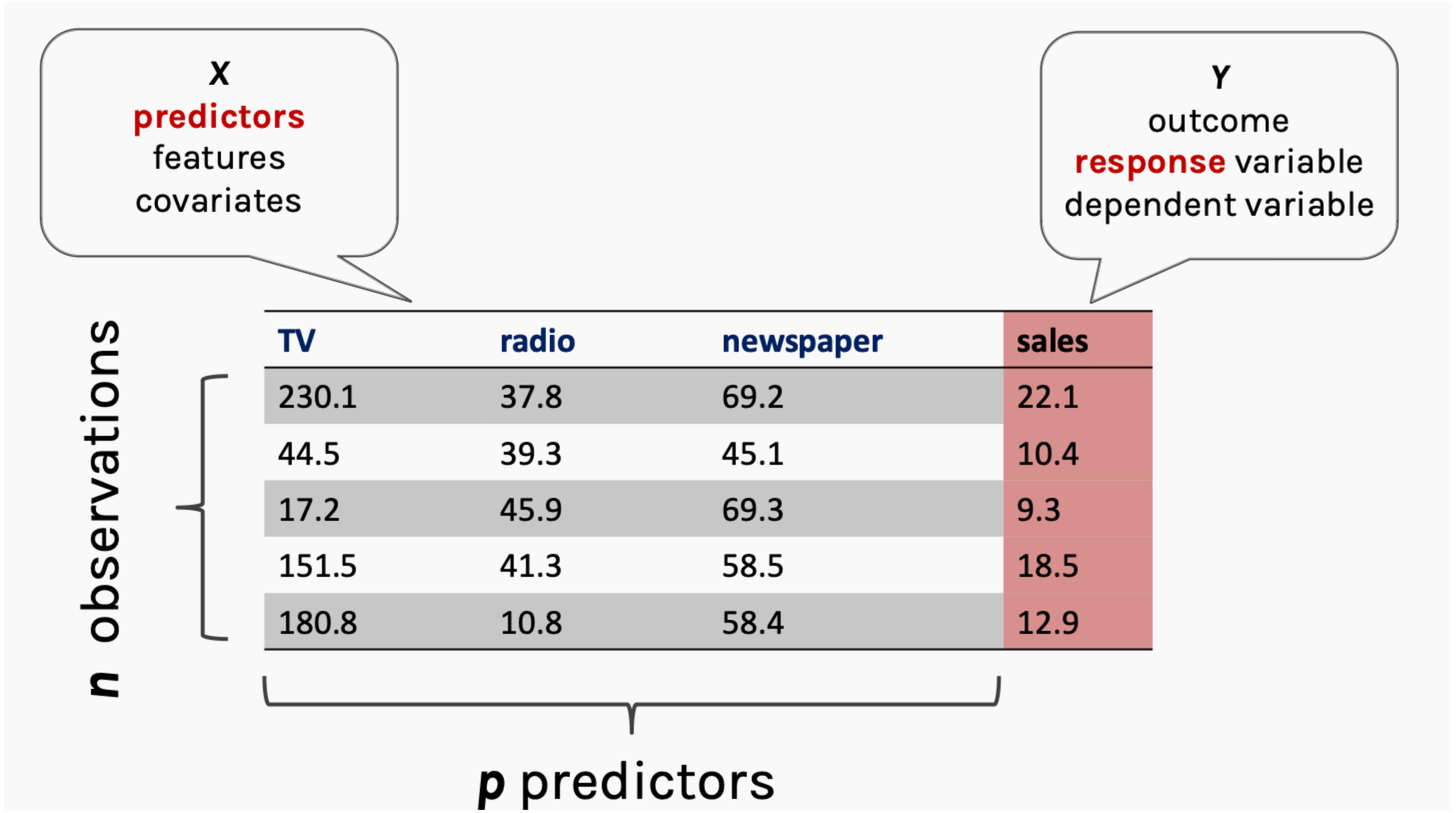
- In general, if we have d features as input $\mathbf{x} = (x_1, x_2, \ldots, x_d)^T$, then the linear regression would have the following form

$$y = f(\mathbf{x}) = w_0 + w_1x_1 + w_2x_2 + \ldots + w_dx_d$$

- The coefficients $w_0, \ldots, w_d$ are the parameters for the model. The goal of **learning** is to find the "best" values for these parameters that describe the relationship between the input features **x** and the target label y based on a set of training example (dataset).

- Once we estimated the parameters, we can use the learned model to **predict** y values for new inputs.

Response vs. Predictor Variables



TV	radio	newspaper	sales
230.1	37.8	69.2	22.1
44.5	39.3	45.1	10.4
17.2	45.9	69.3	9.3
151.5	41.3	58.5	18.5
180.8	10.8	58.4	12.9

Prediction with linear regression model

- Example: Hours studying and grades

We want to learn w_0 and w_1 such that

Predicted final grade in class =

$w_0 + w_1$*(#hours you study/week)

- Assume after learning we have:

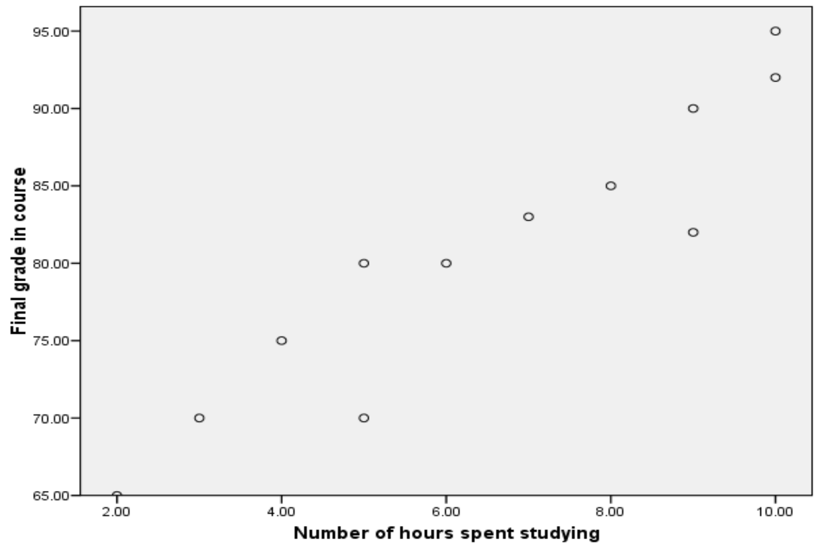
Predicted final grade in class =

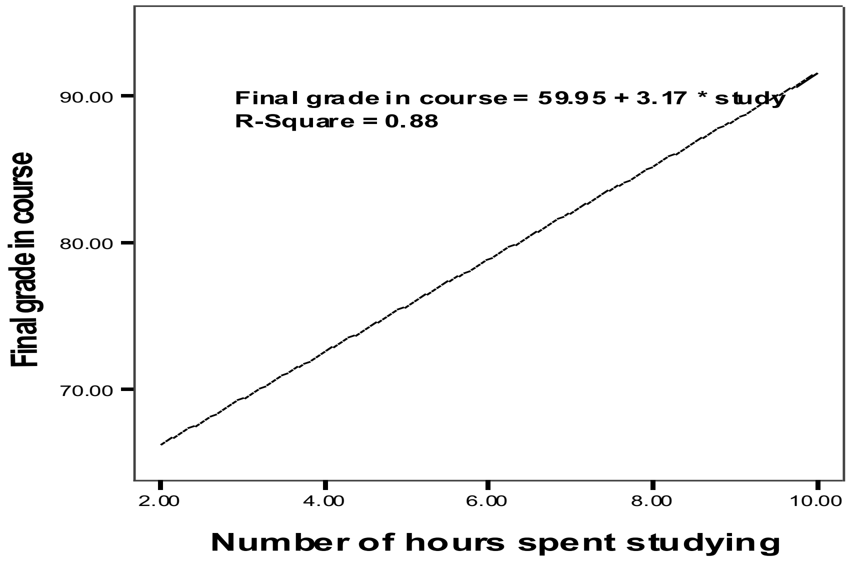
59.95 + 3.17*(# hours you study/week)

- We can now use this function to predict grades for new #hours

Ex: Someone who studies for 12 hours

Final grade = 59.95 + (3.17*12) = 97.99





Linear regression

- In general, if we have d features as input **x** = $(x_1, x_2, \ldots, x_d)^T$, then the linear regression would have the following form

$$y = f(\mathbf{x}) = w_0 + w_1x_1 + w_2x_2 + \ldots + w_dx_d$$

- To simplify notation, we can augment the input with an extra dimension that has the value 1

$$x = (x_1, \ldots, x_d) \rightarrow x = (1, x_1, \ldots, x_d)$$

- We can now write the linear regression model as follows

$$y = f(\mathbf{x}) = w_0x_0 + w_1x_1 + w_2x_2 + \ldots + w_dx_d = \sum_{i=1}^{d} x_i w_i = \mathbf{x}^T \boldsymbol{w}$$

Task Definition

Problem: Given a sample $S = \{(\mathbf{x}_1, y_1), \ldots, (\mathbf{x}_n, y_n)\} \subseteq \mathbb{R}^d \times \mathbb{R}$, find a linear function

$$f(\boldsymbol{x}) = \mathbf{x}^T \boldsymbol{w} = \sum_{i=1}^{d} x_i w_i$$

that best interpolates S.

Notion and notation:

- $\mathbf{x} \in \mathbb{R}^d$ is regarded as a column vector, its transpose $\mathbf{x}^T$ as a row vector.
- $\mathbf{X}$ is an $n \times d$ data matrix (i.e. its i-th row is $\mathbf{x}_i^T$); $\mathbf{y} = (y_1, \ldots, y_n)^T$
- Inner product of $\mathbf{x}, \mathbf{z} \in \mathbb{R}^d$: $\langle \boldsymbol{x}, \boldsymbol{z} \rangle = \mathbf{x}^T \boldsymbol{z} = \sum_{i=1}^{d} x_i z_i$
- Euclidean norm of a vector $\mathbf{x} \in \mathbb{R}^d$: $||\boldsymbol{x}|| = \sqrt{\langle \boldsymbol{x}, \boldsymbol{x} \rangle}$

Task Definition

Problem: Given a sample S = $\{(\mathbf{x}_1, y_1), \dots, (\mathbf{x}_n, y_n)\} \subseteq \mathbb{R}^d \times \mathbb{R}$, find a vector $\mathbf{w} \in \mathbb{R}^d$ such that

$$f(\boldsymbol{x}) = \langle \boldsymbol{x}, \boldsymbol{w} \rangle$$

best interpolates S.

"**best interpolates**": for ($\mathbf{x}$, y) we measure the discrepancy between f($\mathbf{x}$) and y by the square loss function

$$E(f(\mathbf{x}), y) = (f(\mathbf{x}) - y)^2$$

Linear regression solution – simple case

- Let's first consider the solution for the simple linear regression case. I.e., the input is only one variable x.

- Given a set of n training examples: (x_1,y_1), … , (x_n,y_n), we want to learn w_0 and w_1 such that

$$f(x) = y = w_0 + w_1 x$$

- The solution is found by minimizing the sum of squared errors:

$$\underset{w_0,w_1}{\operatorname{argmin}} \sum_{i=1}^{n} \left(y_i - f(x_i)\right)^2$$

$$\underset{w_0,w_1}{\operatorname{argmin}} \sum_{i=1}^{n} (y_i - w_0 - w_1 x_i)^2$$

Linear regression solution – simple case

- The solution is found by minimizing the sum of squared errors:

$$\underset{w_0,w_1}{\operatorname{argmin}} \sum_{i=1}^{n} (y_i - w_0 - w_1 x_i)^2$$

Find the derivative of the error function E w.r.t. each parameter and set it to 0

$$\frac{\partial E}{\partial w_0} = \frac{\partial}{\partial w_0}\left(\sum_{i=1}^{n}(y_i - w_0 - w_1 x_i)^2\right) = \sum_{i=1}^{n} \frac{\partial}{\partial w_0}(y_i - w_0 - w_1 x_i)^2$$

$$= \sum_{i=1}^{n} -2(y_i - w_0 - w_1 x_i) = -2\sum_{i=1}^{n} y_i + 2\sum_{i=1}^{n} w_0 + 2\sum_{i=1}^{n} w_1 x_i$$

$$\frac{\partial E}{\partial w_0} = 0 \rightarrow 0 = -2\sum_{i=1}^{n} y_i + 2\sum_{i=1}^{n} w_0 + 2\sum_{i=1}^{n} w_1 x_i \rightarrow \boldsymbol{w_0 = \frac{\sum_{i=1}^{n} y_i}{n} - w_1 \frac{\sum_{i=1}^{n} x_i}{n}}$$

$$\boldsymbol{w_0 = \bar{y} - w_1 \bar{x}}$$

Linear regression solution – simple case

- The solution is found by minimizing the sum of squared errors:

$$\underset{w_0, w_1}{\operatorname{argmin}} \sum_{i=1}^{n} (y_i - w_0 - w_1 x_i)^2$$

Find the derivative of the error function E w.r.t. each parameter and set it to 0

$$\frac{\partial E}{\partial w_1} = \frac{\partial}{\partial w_1}\left(\sum_{i=1}^{n}(y_i - w_0 - w_1 x_i)^2\right) = \sum_{i=1}^{n} \frac{\partial}{\partial w_1}(y_i - w_0 - w_1 x_i)^2$$

$$= \sum_{i=1}^{n} -2x_i(y_i - w_0 - w_1 x_i) = -2\sum_{i=1}^{n}\left(y_i x_i - w_0 x_i - w_1 x_i^2\right)$$

$$\frac{\partial E}{\partial w_1} = 0 \rightarrow 0 = \sum_{i=1}^{n}\left(y_i x_i - \left(\frac{\sum_{i=1}^{n} y_i}{n} - w_1 \frac{\sum_{i=1}^{n} x_i}{n}\right) x_i - w_1 x_i^2\right)$$

$$\rightarrow \quad w_1 = \frac{\sum_{i=1}^{n} y_i x_i - \frac{\sum_{i=1}^{n} y_i \sum_{i=1}^{n} x_i}{n}}{\sum_{i=1}^{n} x_i^2 - \frac{\sum_{i=1}^{n} x_i \sum_{i=1}^{n} x_i}{n}} \quad \rightarrow \quad w_1 = \frac{\sum_{i=1}^{n}(x_i - \bar{x})(y_i - \bar{y})}{\sum_{i=1}^{n}(x_i - \bar{x})^2}$$

Recap - Task Definition

Problem: Given a sample S = $\{(\mathbf{x}_1, y_1), \ldots, (\mathbf{x}_n, y_n)\} \subseteq \mathbb{R}^d \times \mathbb{R}$, find a vector $\mathbf{w} \in \mathbb{R}^d$ such that

$$f(\boldsymbol{x}) = \langle \boldsymbol{x}, \boldsymbol{w} \rangle$$

best interpolates S.

"**best interpolates**": for $(\mathbf{x}, y)$ we measure the discrepancy between $f(\mathbf{x})$ and y by the square loss function

$$E(f(\mathbf{x}), y) = (f(\mathbf{x}) - y)^2$$

Notion and notation:

- $\mathbf{x} \in \mathbb{R}^d$ is regarded as a column vector, its transpose $\mathbf{x}^T$ as a row vector.
- $\mathbf{X}$ is an n × d data matrix (i.e. its i-th row is $\mathbf{x}_i^T$); $\mathbf{y} = (y_1, \ldots, y_n)^T$
- Inner product of $\mathbf{x}, \mathbf{z} \in \mathbb{R}^d$: $\langle \boldsymbol{x}, \boldsymbol{z} \rangle = \mathbf{x}^T \boldsymbol{z} = \sum_{i=1}^{d} x_i z_i$
- Euclidean norm of a vector $\mathbf{x} \in \mathbb{R}^d$: $\|\boldsymbol{x}\| = \sqrt{\langle \boldsymbol{x}, \boldsymbol{x} \rangle}$

Linear Regression – The normal equations

- Empirical risk w.r.t. the square loss function:

$$E[f] = \frac{1}{n}\sum_{i=1}^{n} (f(\mathbf{x}_i) - y_i)^2$$

$$= \frac{1}{n}\sum_{i=1}^{n} (\langle \mathbf{x}_i, \mathbf{w} \rangle - y_i)^2$$

$$= \frac{1}{n} (\mathbf{X}\,\mathbf{w} - \mathbf{y})^T (\mathbf{X}\,\mathbf{w} - \mathbf{y})$$

$$= \frac{1}{n} \|\mathbf{X}\,\mathbf{w} - \mathbf{y}\|^2$$

Solve $\min_{\mathbf{w}} \frac{1}{n} \|\mathbf{X}\,\mathbf{w} - \mathbf{y}\|^2$ (known as least squares)

Linear Regression – The normal equations

- Convex minimization problem: $\min_{\mathbf{w}} E[\mathbf{w}] = \min_{\mathbf{w}} \frac{1}{n} \|\mathbf{X}\,\mathbf{w} - \mathbf{y}\|^2$
- Calculate the gradient:

$$\nabla_{\mathbf{w}} E[\mathbf{w}] = \frac{\partial}{\partial \mathbf{w}} \left(\frac{1}{n} \|\mathbf{X}\,\mathbf{w} - \mathbf{y}\|^2 \right)$$

$$= \frac{\partial}{\partial \mathbf{w}} \left(\frac{1}{n} (\mathbf{w}^T \mathbf{X}^T \mathbf{X}\,\mathbf{w} - 2\,\mathbf{w}^T \mathbf{X}^T \mathbf{y} + \mathbf{y}^T\mathbf{y}) \right)$$

$$= \frac{1}{n} (2\,\mathbf{X}^T \mathbf{X}\,\mathbf{w} - 2\,\mathbf{X}^T \mathbf{y})$$

- Set it to 0: $\mathbf{X}^T \mathbf{X}\,\mathbf{w} = \mathbf{X}^T \mathbf{y}$
- And solve the linear system of equations: $\mathbf{w} = (\mathbf{X}^T \mathbf{X})^{-1} \mathbf{X}^T \mathbf{y}$

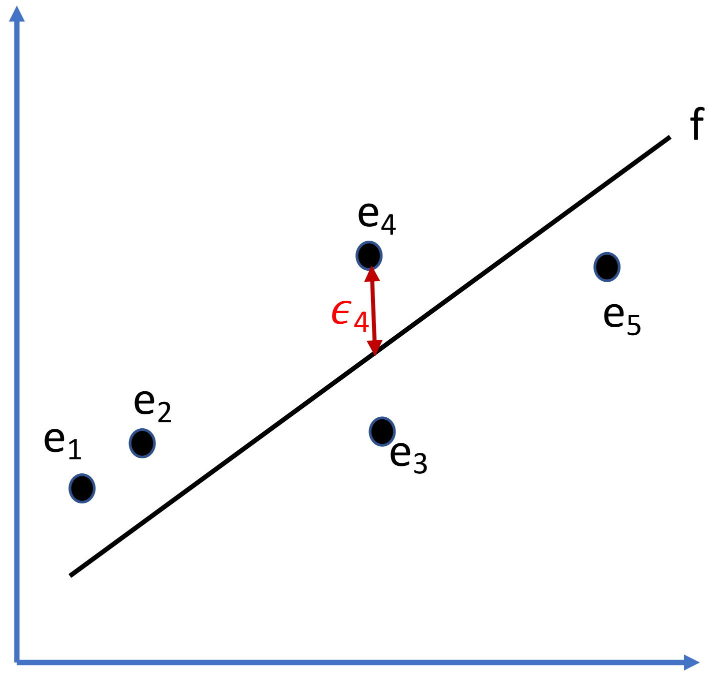
Probabilistic Interpretation of Linear Regression

- Consider real-valued target function f

- Training examples $e_i(x_i, y_i)$

- y_i is a noisy training value

  $y_i = f(x_i) + \epsilon_i$



- ϵ_i is a random variable (noise) drawn independently for

each x_i according to some Gaussian distribution with zero mean

$$p(y_i \mid f) = \frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{1}{2}\frac{(y_i - f(x_i))^2}{\sigma^2}}$$

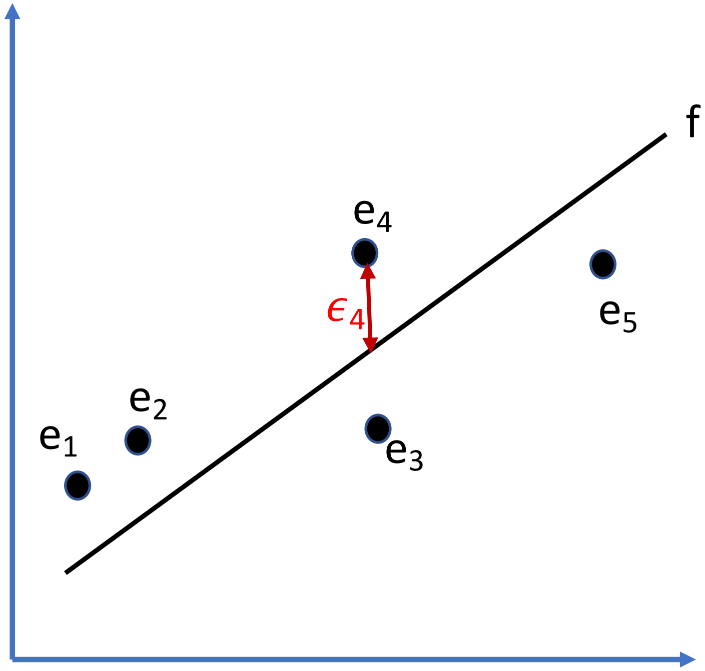
Probabilistic Interpretation of Linear Regression

- Our goal is to approximate f by f’
- The maximum likelihood estimate of f’ is

$$f'_{ML} = \underset{f'}{\operatorname{argmax}}\, p(D|f')$$

$$= \underset{f'}{\operatorname{argmax}} \prod_{i=1}^{n} p(ei|f')$$

$$= \underset{f'}{\operatorname{argmax}} \prod_{i=1}^{n} \frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{1}{2}\frac{(y_i - f'(x_i))^2}{\sigma^2}}$$



Maximize natural log of this instead ….

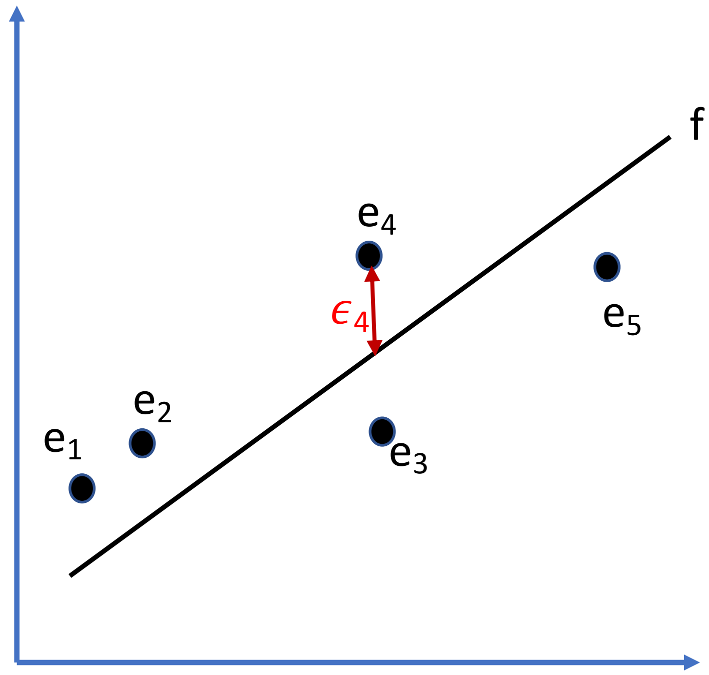
Probabilistic Interpretation of Linear Regression

$$f'_{ML} = \underset{f'}{\operatorname{argmax}} \sum_{i=1}^{n} ln \frac{1}{\sqrt{2\pi\sigma^2}} - \frac{1}{2}\frac{(y_i - f'(x_i))^2}{\sigma^2}$$

$$= \underset{f'}{\operatorname{argmax}} \sum_{i=1}^{n} - \frac{1}{2}\frac{(y_i - f'(x_i))^2}{\sigma^2}$$

$$= \underset{f'}{\operatorname{argmax}} \sum_{i=1}^{n} -(y_i - f'(x_i))^2$$

$$f'_{ML} = \underset{f'}{\operatorname{argmin}} \sum_{i=1}^{n} (y_i - f'(x_i))^2$$



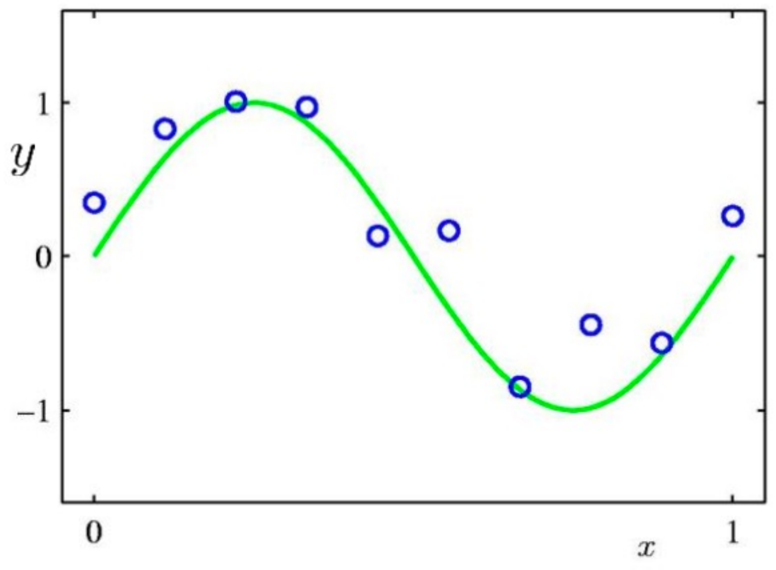
Maximum Likelihood estimate f'_{ML} minimizes the sum of squared errors

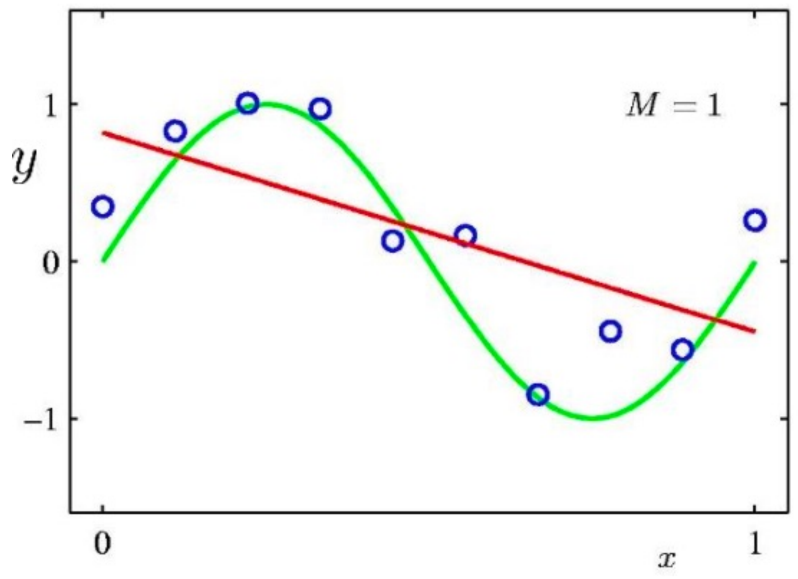
Non-Linear Regression

Non-Linear Regression

- Linear regression fits a linear model to the data.

- In real world applications, many problems are non-linear. In this case, fitting a linear model will underfit.





- Can we still use linear regression to fit a non-linear model?

Non-Linear Regression

Goal:

- Keep the math of linear regression, but extend to more general functions

Key idea:

- You can make a non-linear function from a linear weighted sum of non-linear basis functions

Non-Linear Regression

Linear regression:

$$f(\boldsymbol{x}) = \langle \boldsymbol{x}, \boldsymbol{w} \rangle$$

Non-linear regression:

$f(\boldsymbol{x}) = \langle \boldsymbol{z}, \boldsymbol{w} \rangle$, where **z** = g(**x**)

In other words, create z by evaluating x against basis functions, then linearly regress against z

Non-Linear Regression

Example of non-linear basis functions:

- Radial basis functions

$$f(x) = e^{\frac{-(x-\alpha)^2}{\lambda}}$$

- Arctan Functions

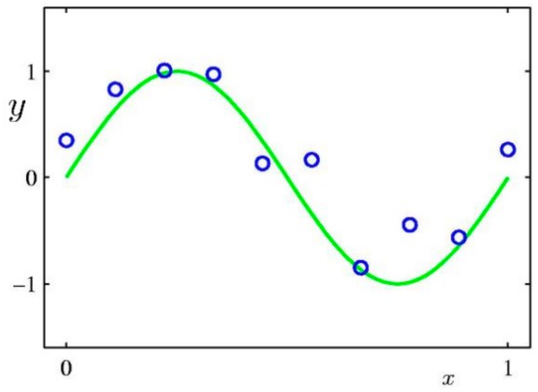
- Monomials

$$x \rightarrow x, x^2, \ldots, x^m$$

$$(x_1, x_2) \rightarrow x_1, x_2, x_1x_2, x_1^2, x_2^2$$

Example: polynomial regression

example: polynomial curve fitting



unknown target function: $\sin(2\pi x)$

training data: $S = \{(x_1, y_1), \ldots, (x_{10}, y_{10})\}$

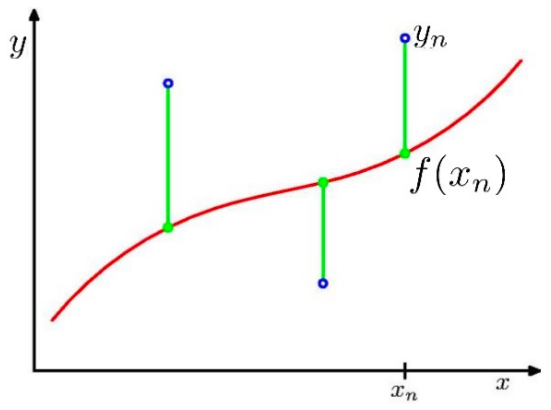
- $y_i = \sin(2\pi x_i)$ + random noise

hypothesis space: polynomials

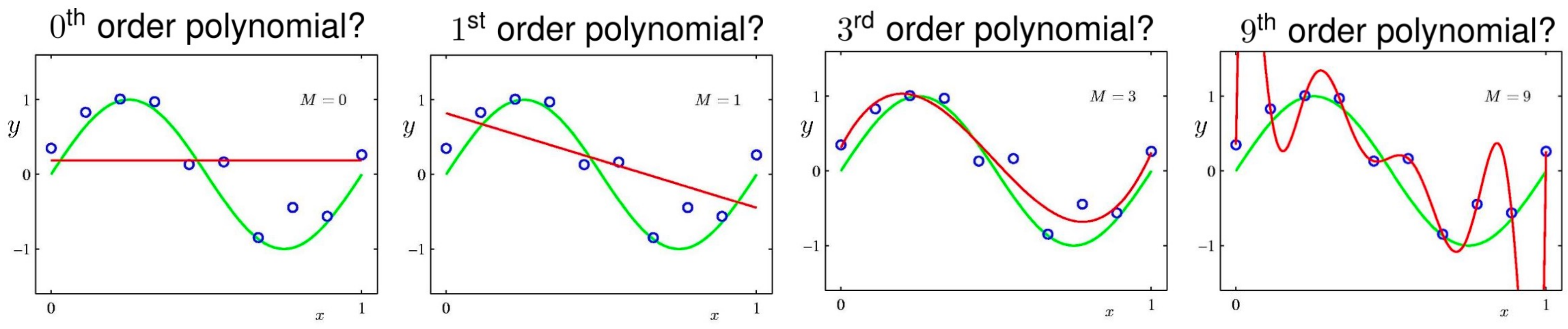
$$\left\{\sum_{i=1}^{M} w_i x^i + w_0 : M \in \mathbb{N}, w_j \in \mathbb{R}\right\}$$

- feature space embedding: $x \mapsto (x^0, x^1, \ldots, x^M)$
- patterns: hyperplanes in $\mathbb{R}^M$

loss function: square loss



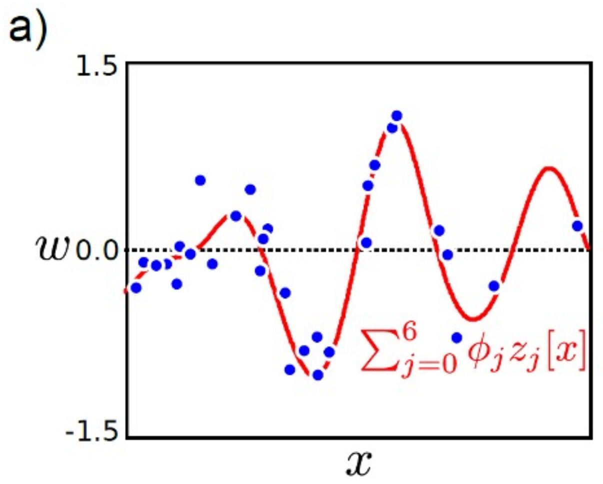
Example: polynomial regression

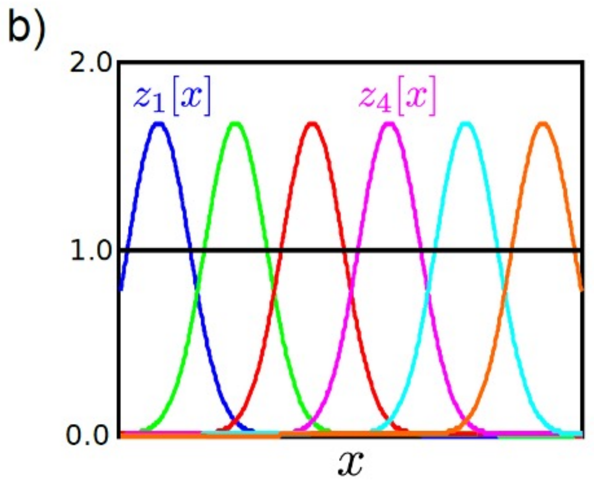


lessons:

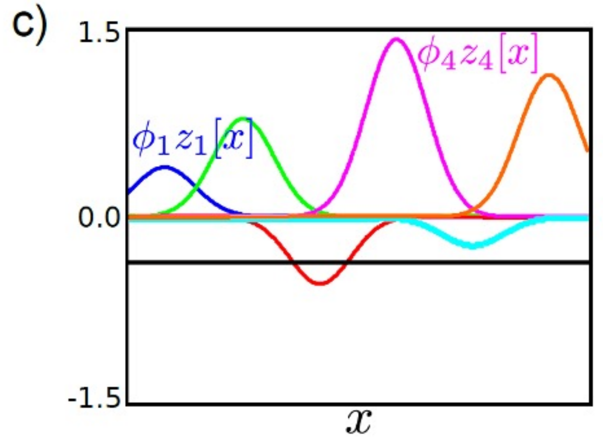
- the 0th and 1st order polynomials fit badly the data (blue points)
  - $\Rightarrow$ too simple models **underfit**
- the 9th order polynomial fits the best the data, but is a bad generalization
  - $\Rightarrow$ too complex models **overfit**
- the 3rd order polynomial is expected to **generalize** the best
  - $\Rightarrow$ model generalizing good: neither too simple nor too complex

Radial basis functions

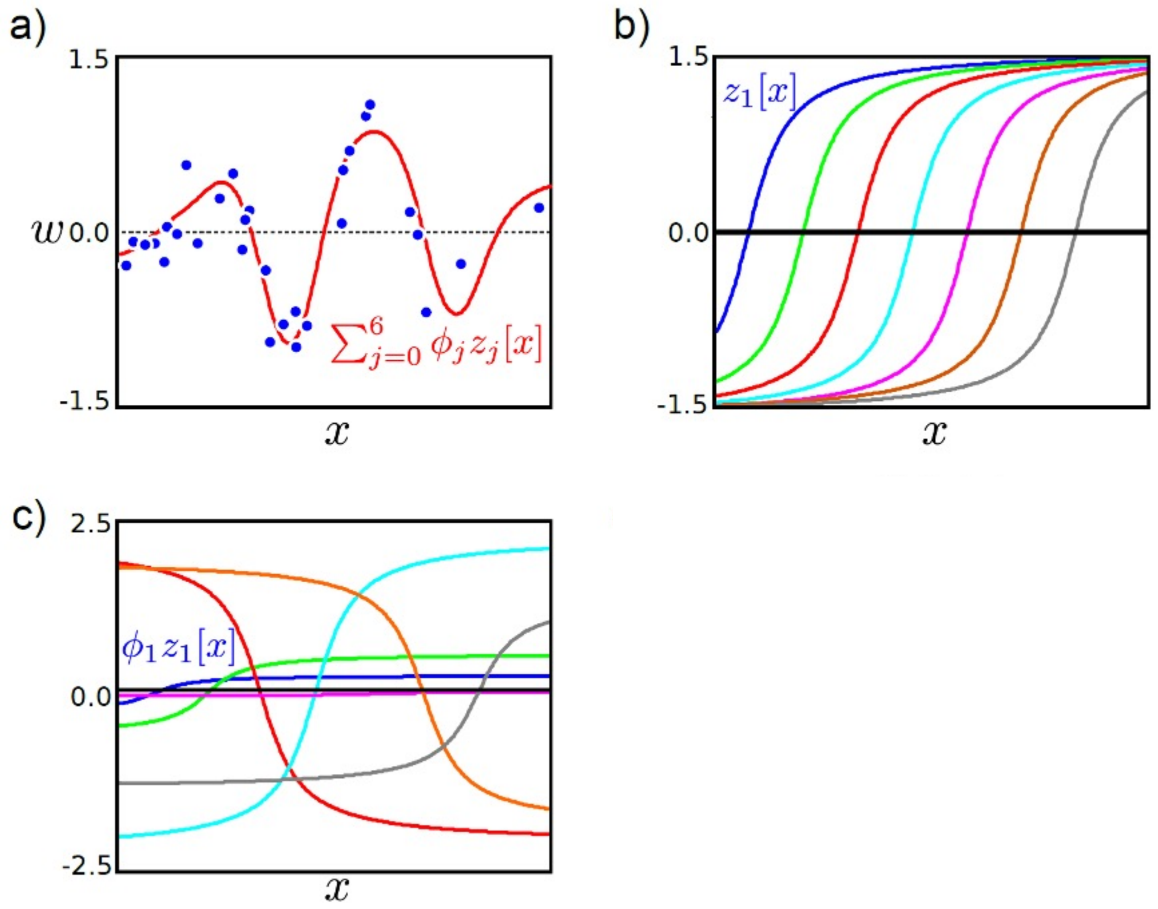




$$
\mathbf{z}_i = \begin{bmatrix} 1 \\ \exp\left[-(x_i-\alpha_1)^2/\lambda\right] \\ \exp\left[-(x_i-\alpha_2)^2/\lambda\right] \\ \exp\left[-(x_i-\alpha_3)^2/\lambda\right] \\ \exp\left[-(x_i-\alpha_4)^2/\lambda\right] \\ \exp\left[-(x_i-\alpha_5)^2/\lambda\right] \\ \exp\left[-(x_i-\alpha_6)^2/\lambda\right] \end{bmatrix}
$$



Arc Tan Functions



$$\mathbf{z}_i = \begin{bmatrix} \arctan[\lambda x_i - \alpha_1] \\ \arctan[\lambda x_i - \alpha_2] \\ \arctan[\lambda x_i - \alpha_3] \\ \arctan[\lambda x_i - \alpha_4] \\ \arctan[\lambda x_i - \alpha_5] \\ \arctan[\lambda x_i - \alpha_6] \\ \arctan[\lambda x_i - \alpha_7] \end{bmatrix}$$

Non-Linear Regression

- Evaluate x against some basis functions to create the z vector

- Apply linear regression on z

$$f(\boldsymbol{x}) = \langle \boldsymbol{z}, \boldsymbol{w} \rangle$$, where **z** = g(**x**)

- Solution:

$$\mathbf{w} = (\mathbf{Z}^T \mathbf{Z})^{-1} \mathbf{Z}^T \mathbf{y}$$

Problems of the normal equations solution

Linear regression solution: $w = (X^T X)^{-1} X^T y$

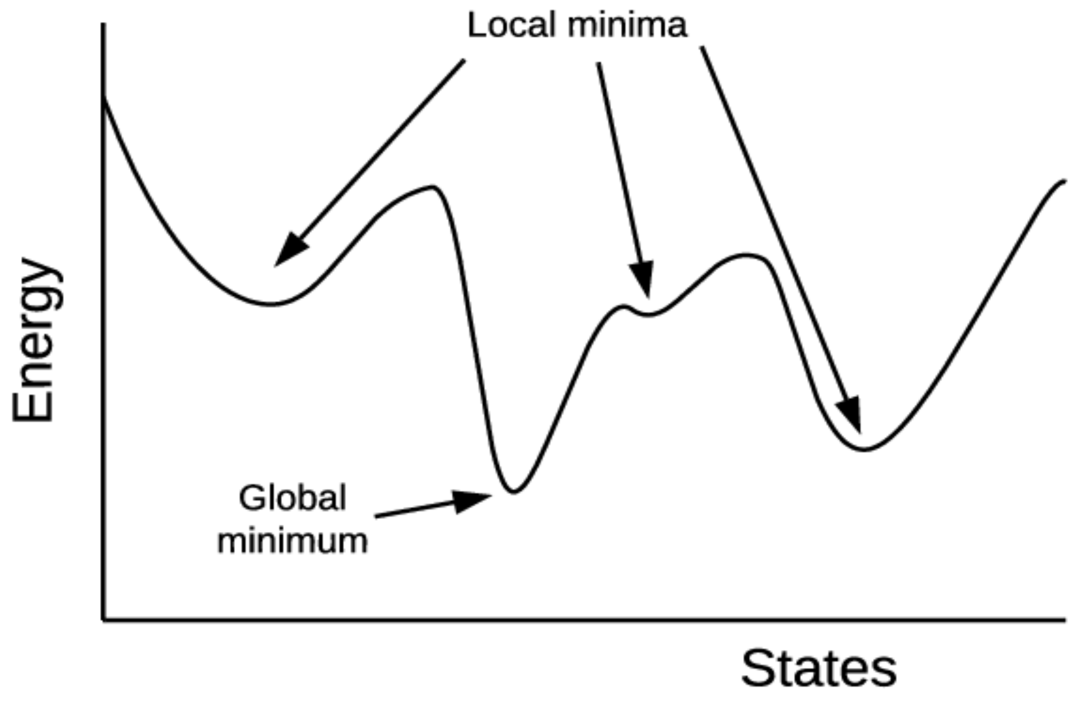
- Issues:
    - Inverse is costly $O(d^3)$, where d is the number of features.
    - Non-invertibility of the matrix.
    - The dataset could be very large.
- Solution: use iterative methods such as *gradient descent.*

Gradient Descent

Introduction

- Gradient descent (GD) is an iterative first-order optimization algorithm used to find a local minimum of a given function.

- This method is commonly used in machine learning (ML) and deep learning(DL) to minimize a cost/loss function (e.g. in a linear regression).

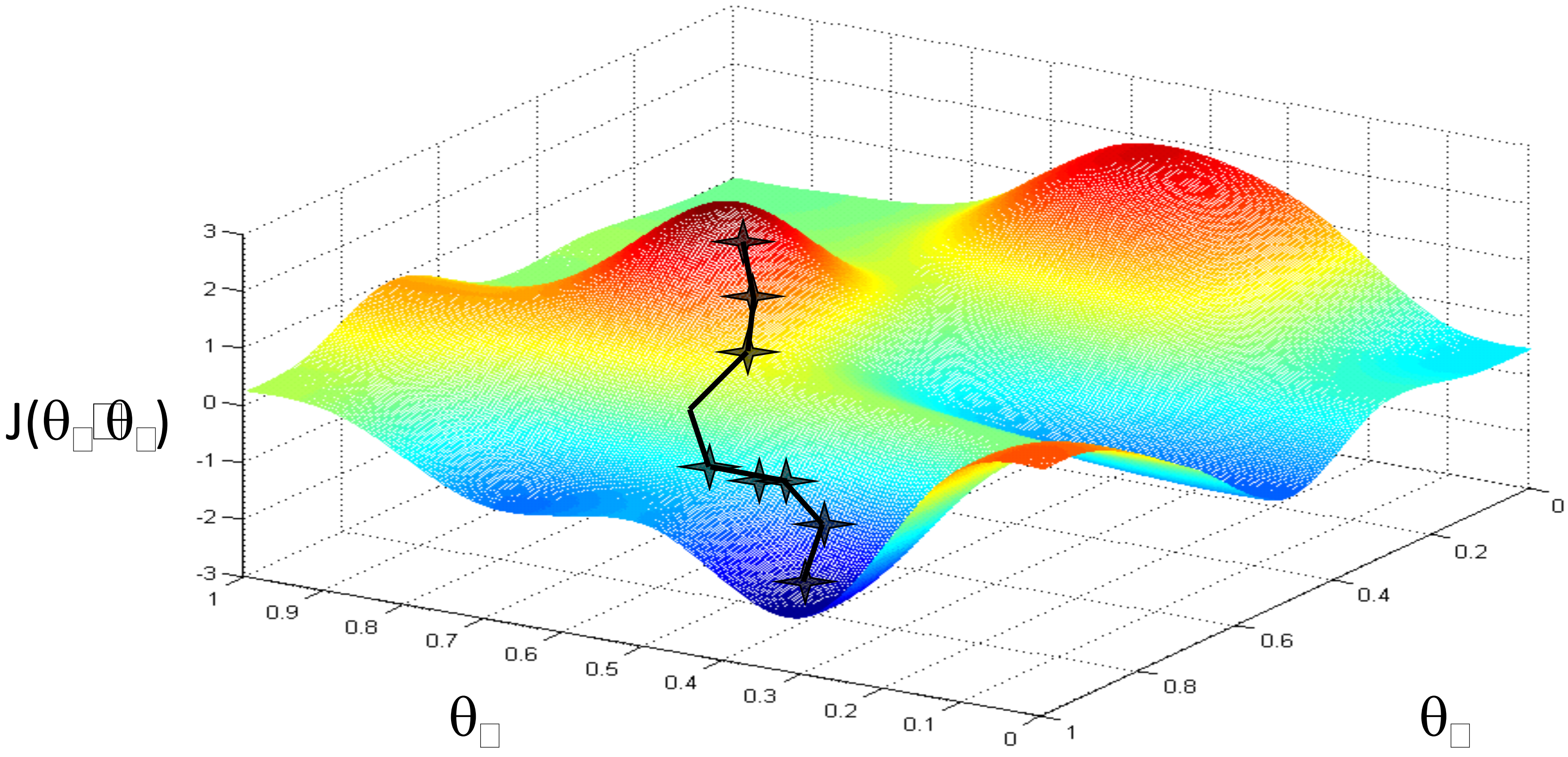


Gradient Descent Idea

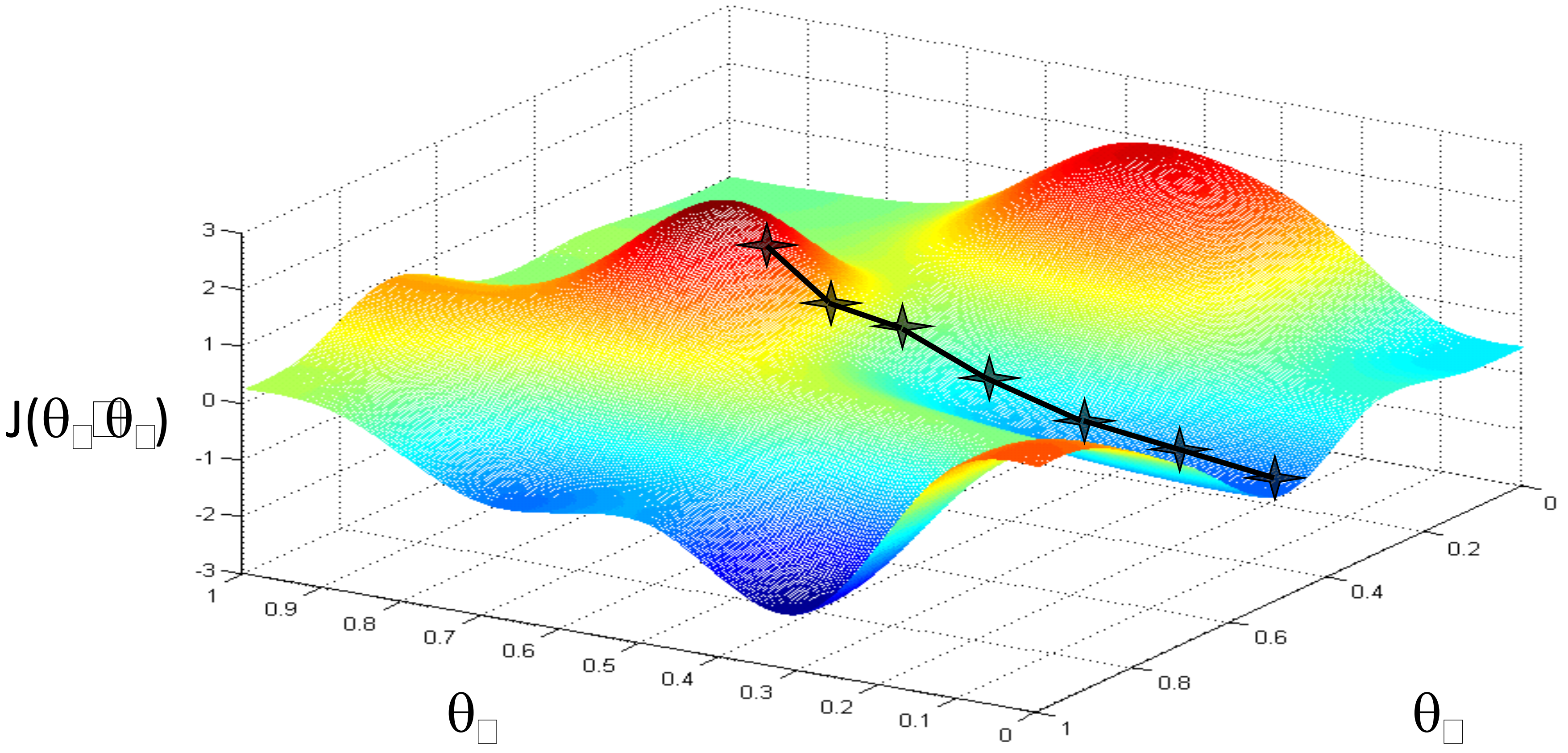
- Have some function $J(\theta_0, \theta_1)$
- Want $\min\limits_{\theta_0, \theta_1} J(\theta_0, \theta_1)$
- Outline:

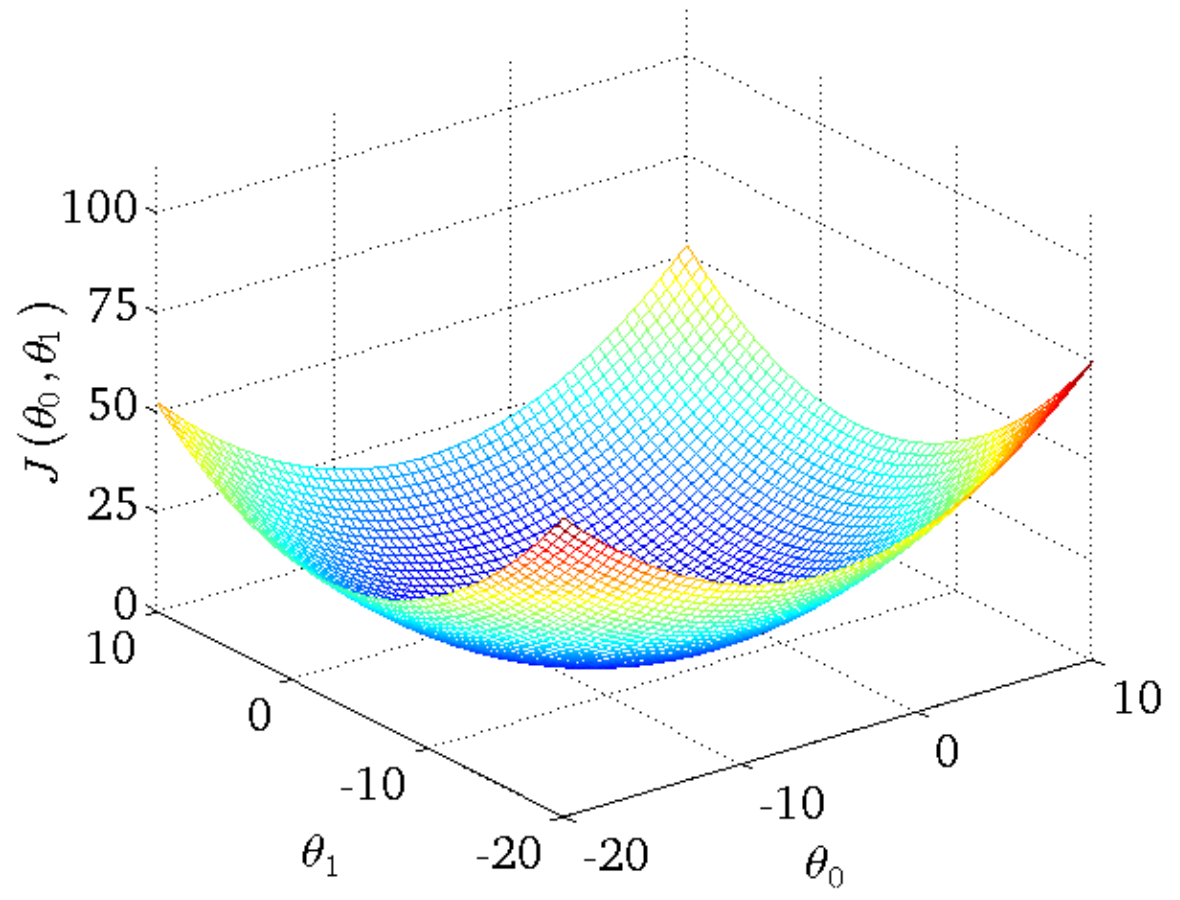
  - Start with some θ_0, θ_1

  - Keep changing θ_0, θ_1 to reduce $J(\theta_0, \theta_1)$ until we hopefully end up at a minimum



$J(\theta_0,\theta_1)$
3
2
1
0
-1
-2
-3
1
0.9
0.8
0.7
0.6
0.5
0.4
0.3
0.2
0.1
0
θ_0
0
0.2
0.4
0.6
0.8
1
θ_1

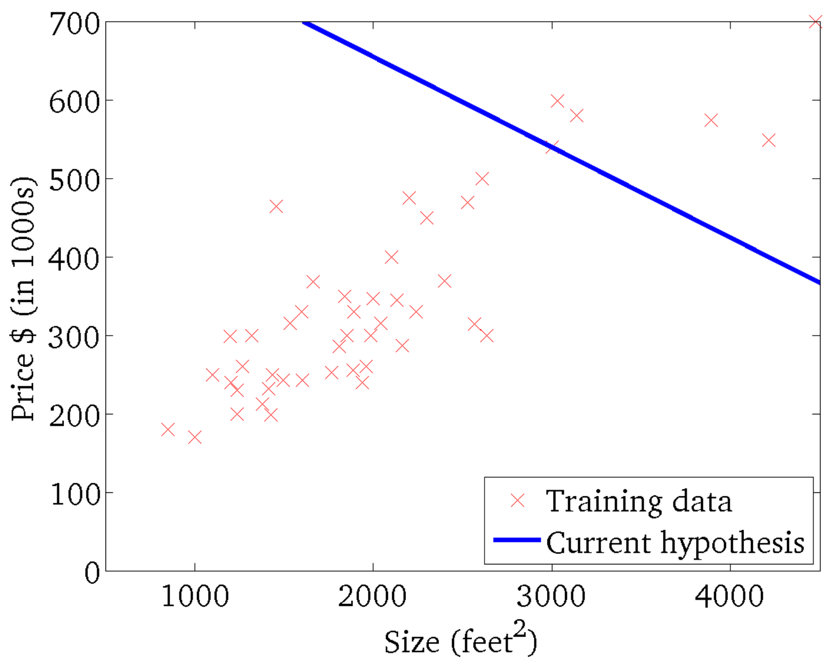

J(θ□□θ□)
θ□
θ□
3
2
1
0
-1
-2
-3
1
0.9
0.8
0.7
0.6
0.5
0.4
0.3
0.2
0.1
0
0
0.2
0.4
0.6
0.8
1



100
75
50
25
0
$J(\theta_0, \theta_1)$
10
0
-10
-20
θ_1
-20
-10
0
10
θ_0

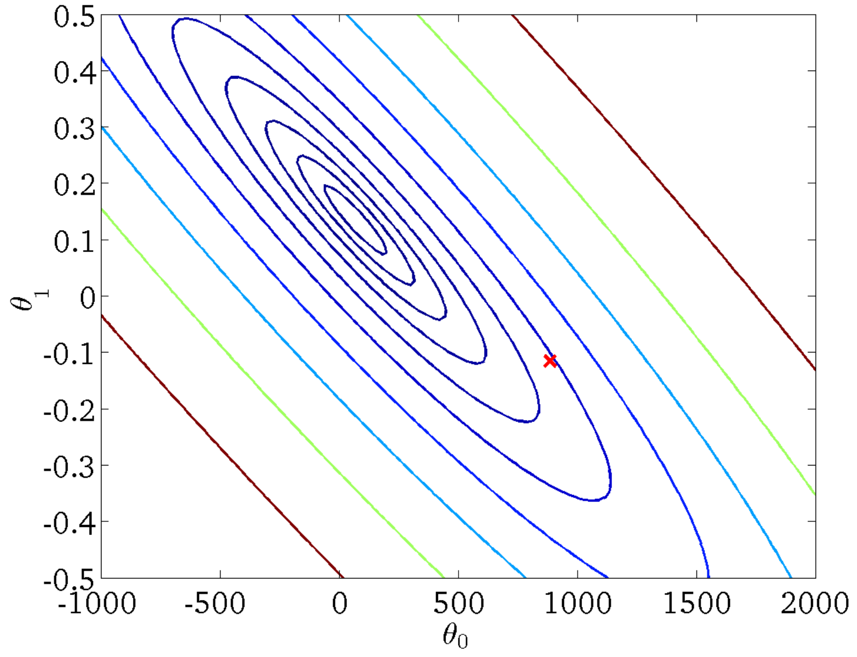
$h_\theta(x)$

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



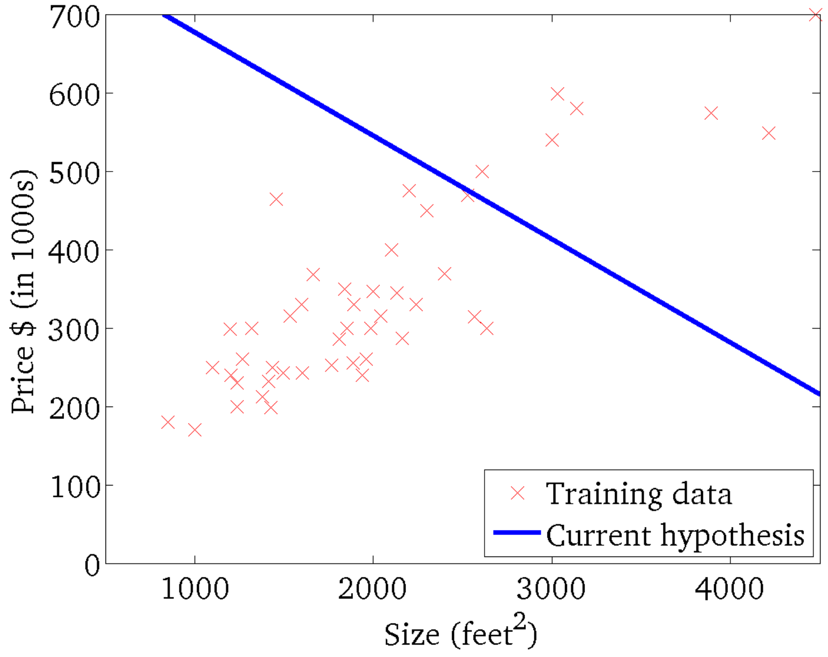
$J(\theta_0, \theta_1)$

(function of the parameters θ_0, θ_1)



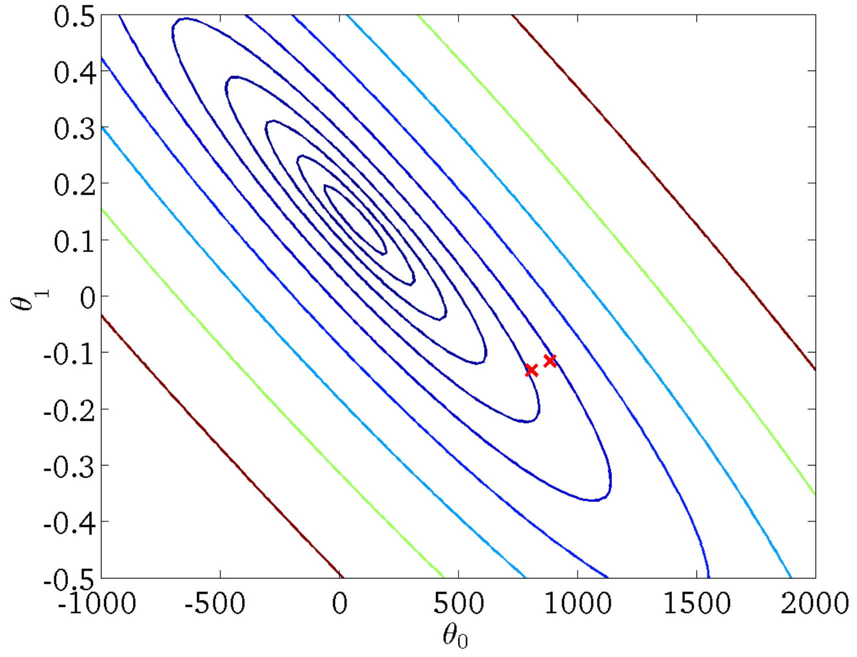
$$h_\theta(x)$$

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



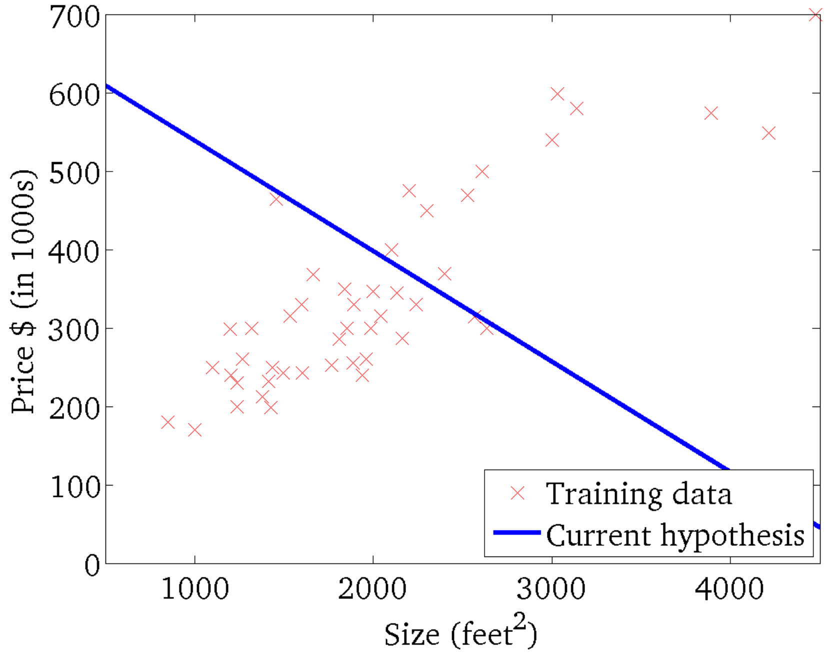
$$J(\theta_0, \theta_1)$$

(function of the parameters θ_0, θ_1)



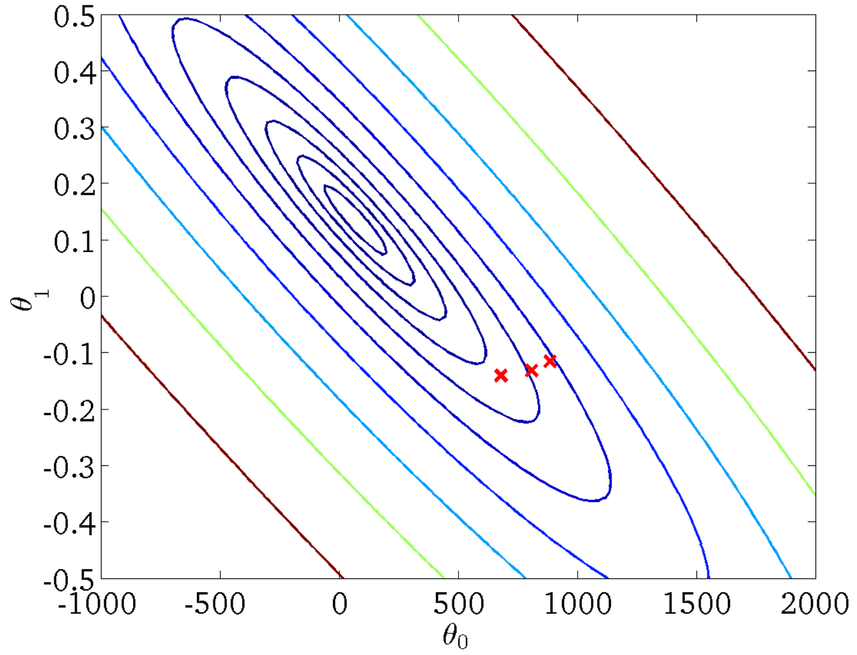
$h_\theta(x)$

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



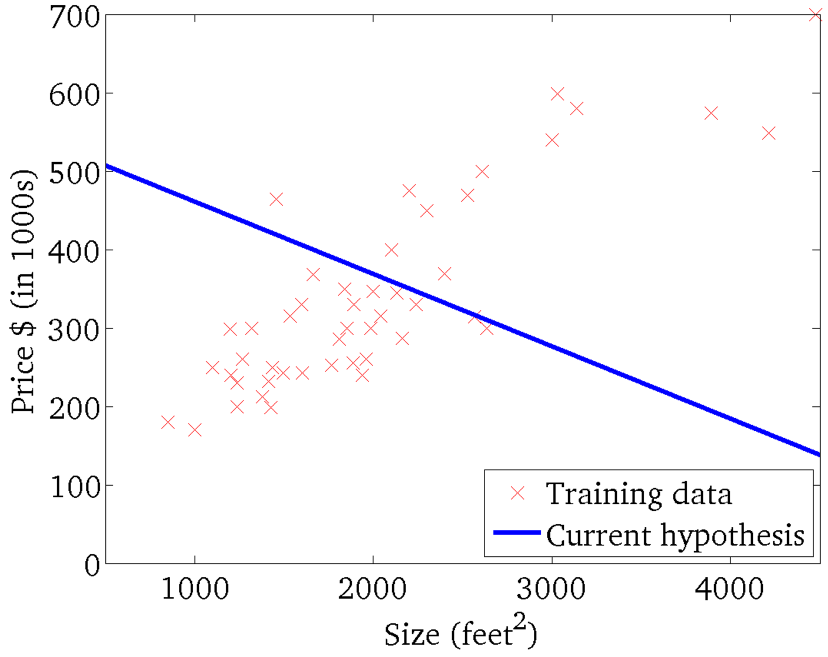
$J(\theta_0, \theta_1)$

(function of the parameters θ_0, θ_1)



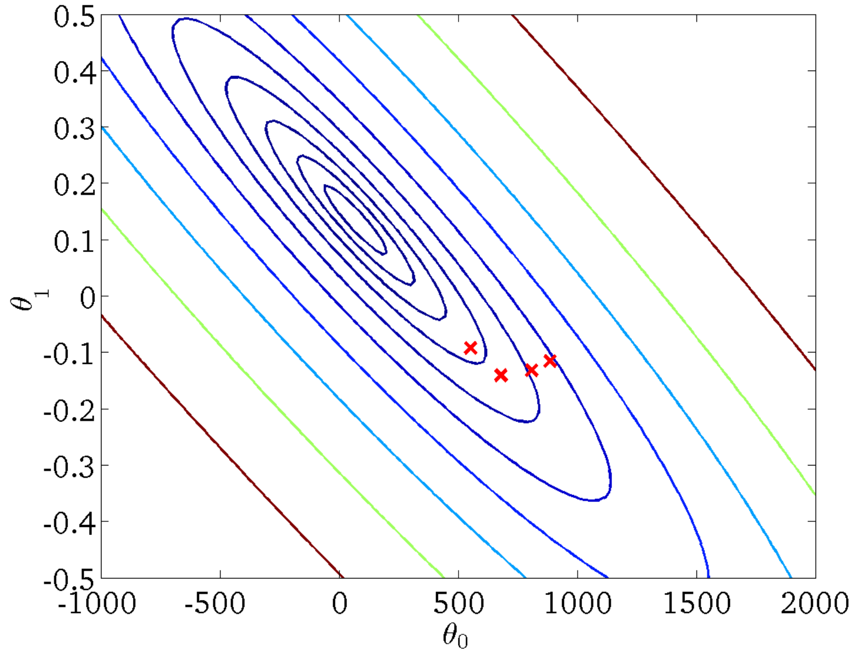
$$h_\theta(x)$$

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



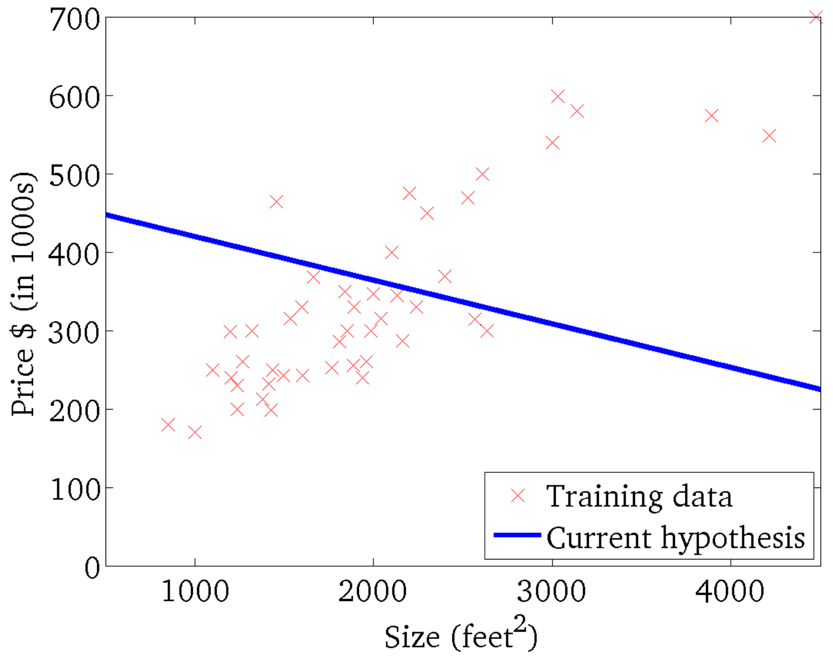
$$J(\theta_0, \theta_1)$$

(function of the parameters θ_0, θ_1)



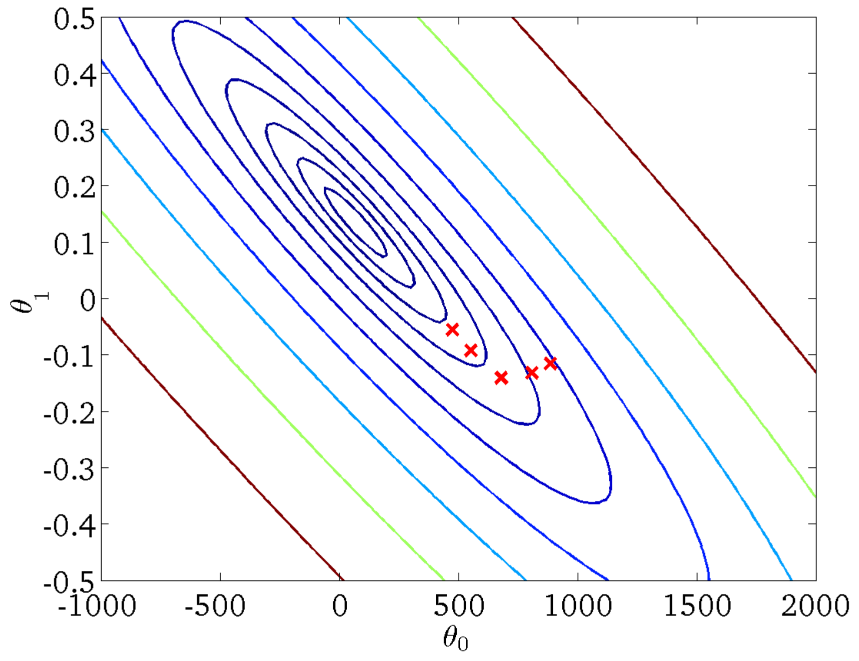
$h_\theta(x)$

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



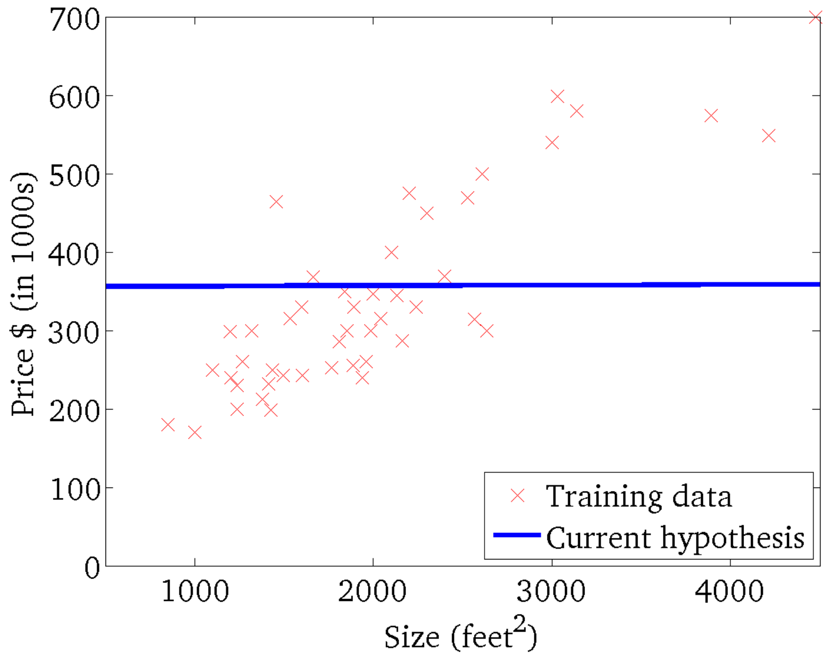
$J(\theta_0, \theta_1)$

(function of the parameters θ_0, θ_1)



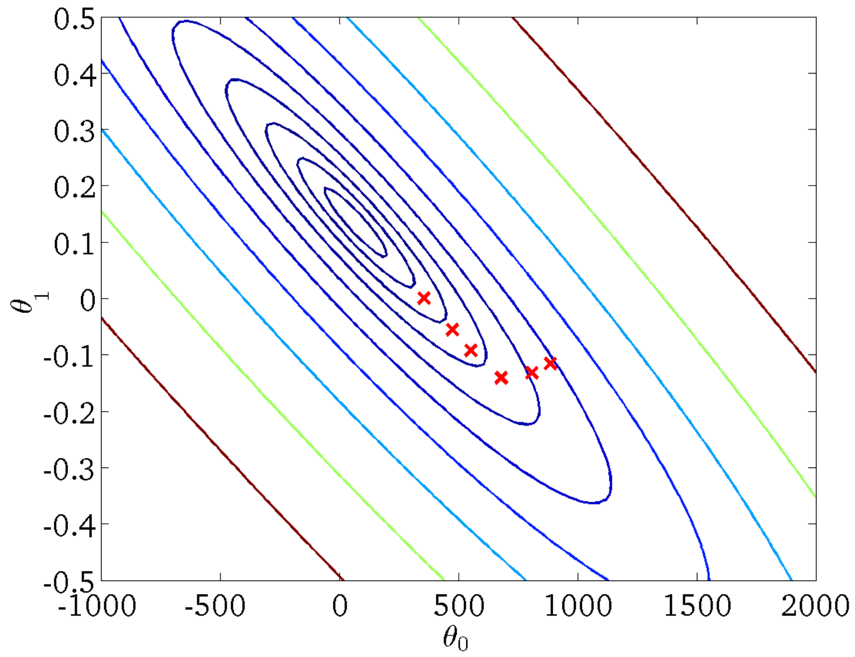
$h_\theta(x)$

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



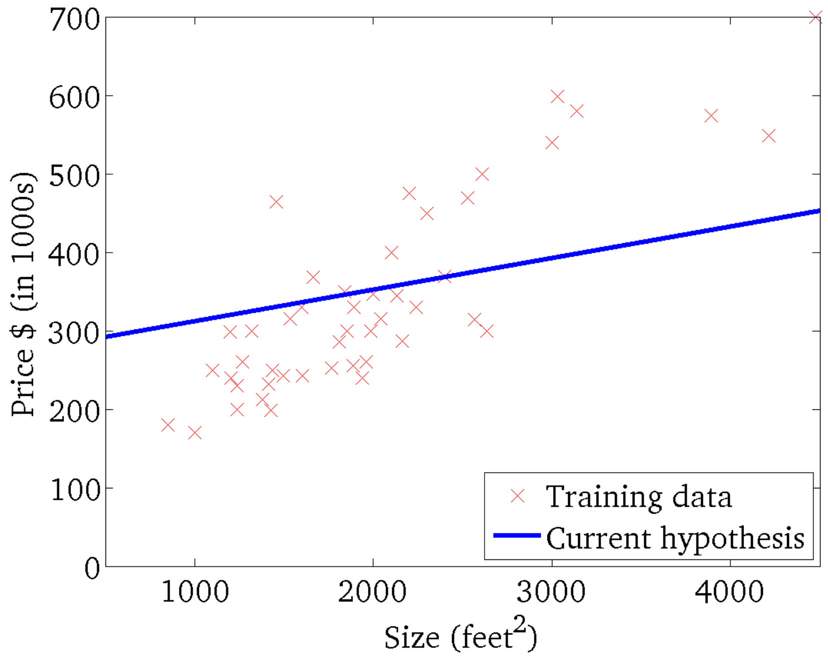
$J(\theta_0, \theta_1)$

(function of the parameters θ_0, θ_1)



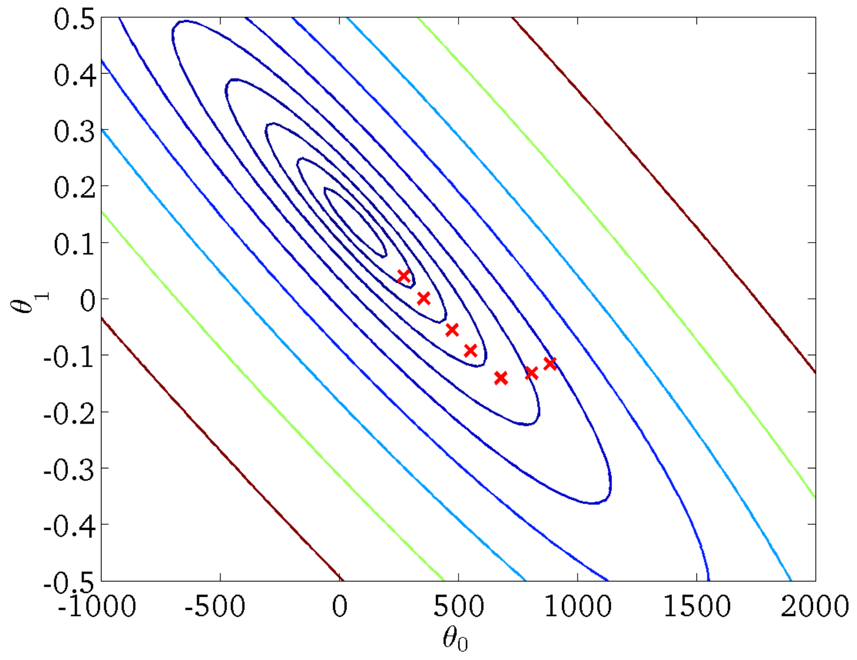
$h_\theta(x)$

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



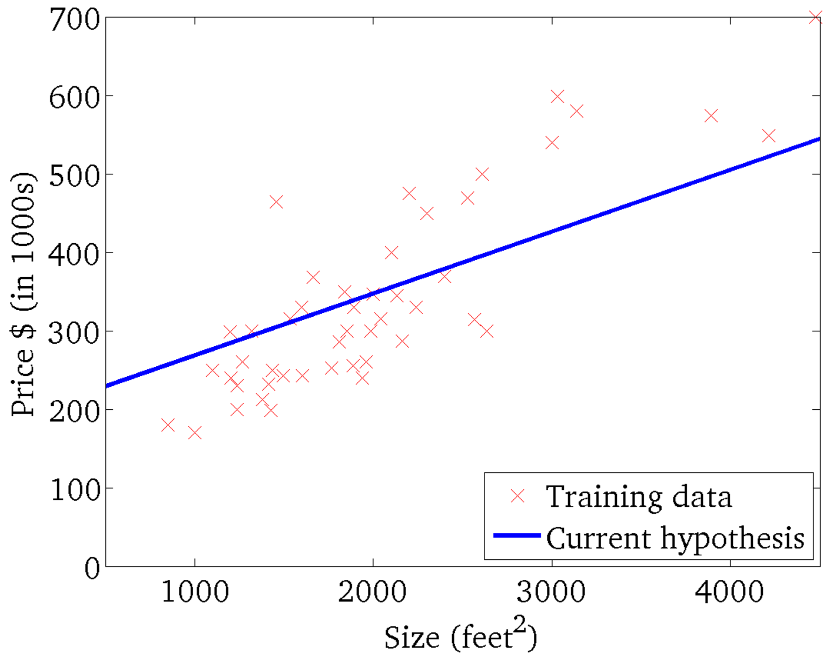
$J(\theta_0, \theta_1)$

(function of the parameters θ_0, θ_1)



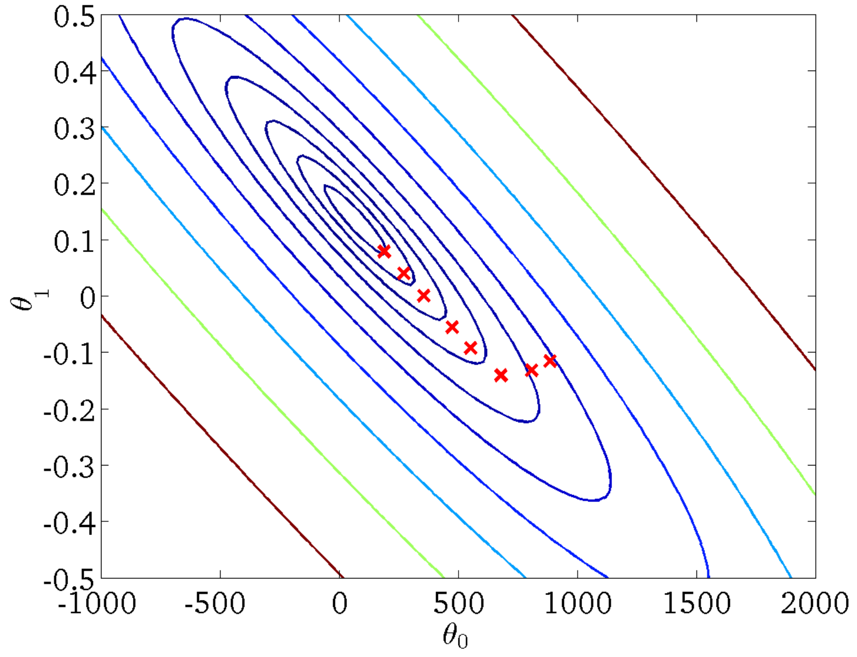
$$h_\theta(x)$$

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



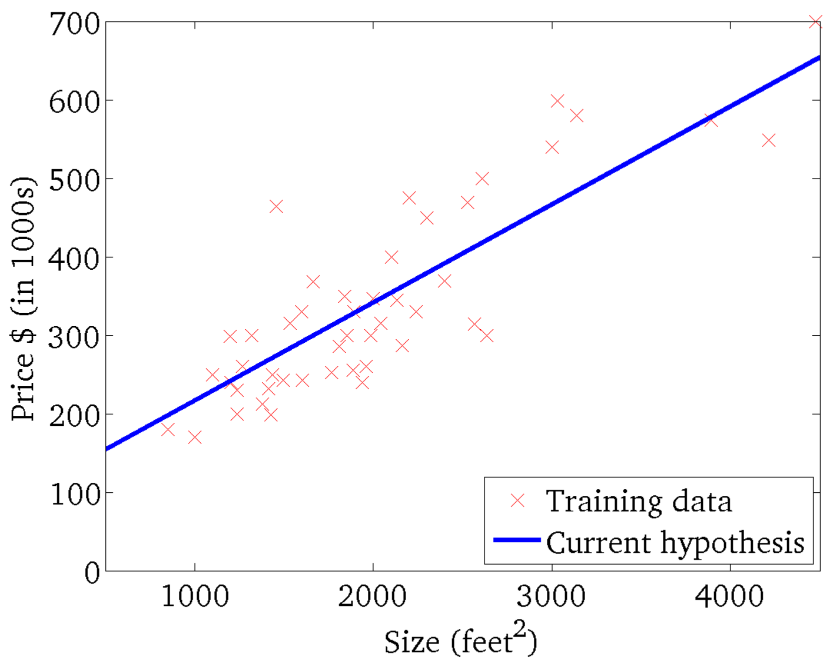
$$J(\theta_0, \theta_1)$$

(function of the parameters θ_0, θ_1)



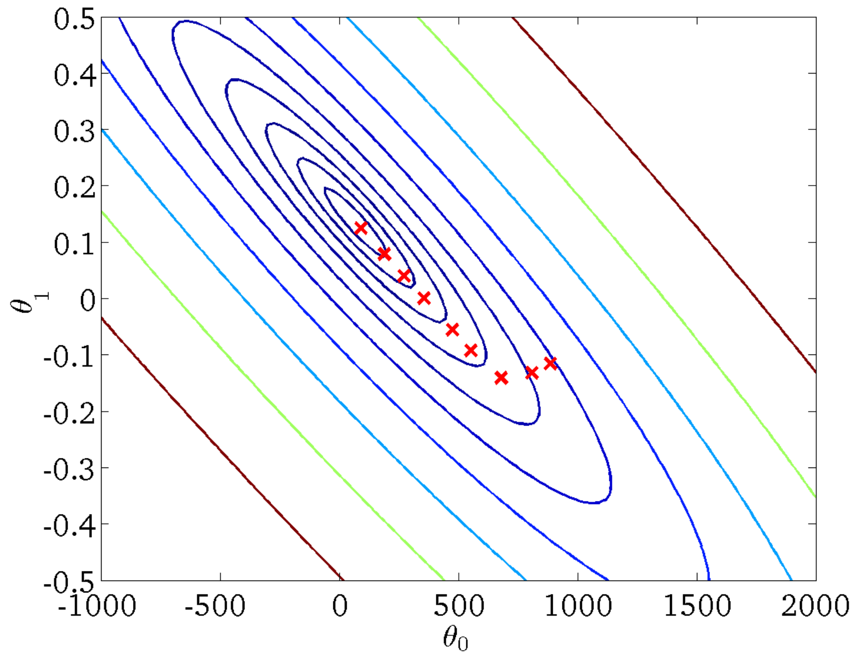
$h_\theta(x)$

(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 Idea

- Have some function $J(\theta_0, \theta_1)$
- Want $\min_{\theta_0, \theta_1} J(\theta_0, \theta_1)$
- Outline:

  - Start with some θ_0, θ_1

  - **Keep changing** θ_0, θ_1 to reduce $J(\theta_0, \theta_1)$ until we hopefully **end up at a minimum**

How to change the parameters?
How to identify the minimum?

Function Requirements

- Gradient descent algorithm does not work for all functions. There are two specific requirements. A function has to be:

  - **Differentiable**: If a function is differentiable it has a derivative for each point in its domain.

  - **Convex**: for a univariate function, this means that the line segment connecting two function's points lays on or above its curve (it does not cross it).

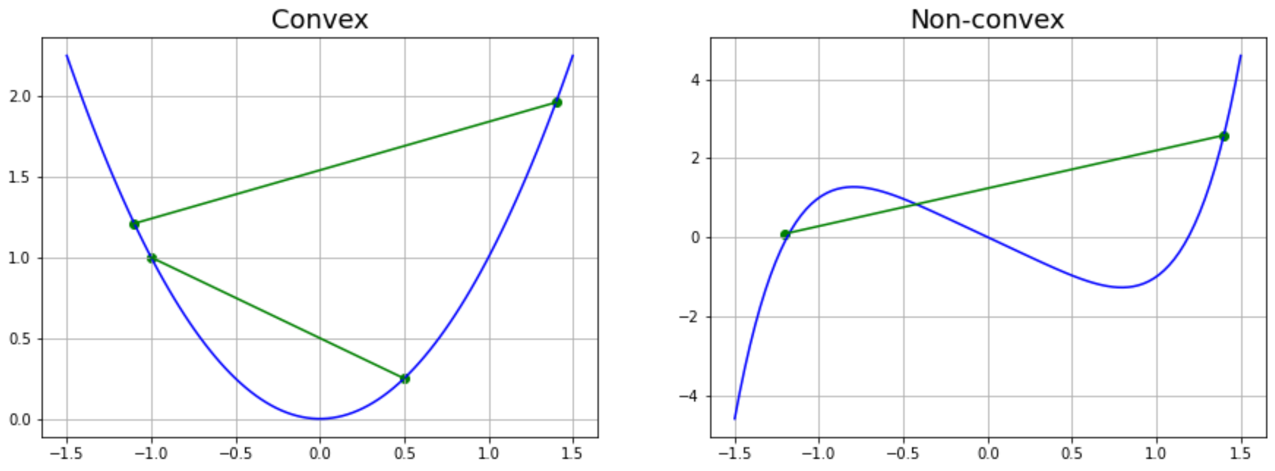
- Gradient descent is widely used with non-convex functions. However, in this case the answer is not optimal. Ex: training deep neural networks.

Convex Function

- A function f is convex if for any two points x_1, x_2 laying on the function's curve the following condition is true:

$$f(\lambda x_1 + (1-\lambda)x_2) \leq \lambda f(x_1) + (1-\lambda)f(x_2))$$

where $0 \leq \lambda \leq 1$



- Another way to check mathematically if a univariate function is convex is to calculate the second derivative and check if its value is always bigger than 0.

$$\frac{d^2 f(x)}{dx^2} > 0$$

What is Gradient

- Intuitively it is a slope of a curve at a given point in a specified direction.
- In the case of a univariate function, it is simply the first derivative at a selected point.
- In the case of a multivariate function, it is a vector of derivatives in each main direction (along variable axes)
- A gradient for a function with n parameters f(x) at a given point p is defined as follows:

$$\nabla f(p) = \begin{bmatrix} \frac{\partial f}{\partial x_1}(p) \\ \vdots \\ \frac{\partial f}{\partial x_n}(p) \end{bmatrix}$$

- Example:

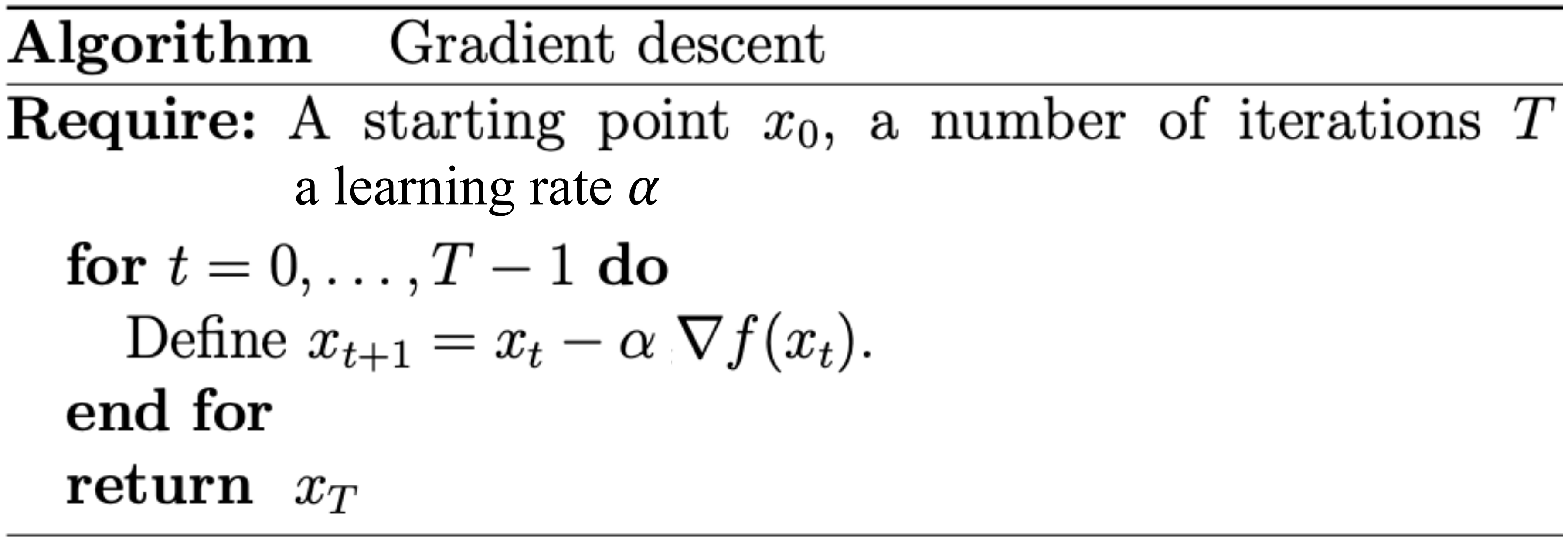
$$f(x,y) = 0.5x^2 + y^2 \qquad \nabla f(x,y) = \begin{bmatrix} x \\ 2y \end{bmatrix} \qquad \nabla f(10,10) = \begin{bmatrix} 10 \\ 20 \end{bmatrix}$$

Gradient Descent Algorithm

Gradient Descent method steps are:

- choose a starting point (initialization).
- calculate gradient at this point.
- make a scaled step in the opposite direction to the gradient (objective: minimize).
- repeat points 2 and 3 until one of the criteria is met:
  - maximum number of iterations reached
  - step size is smaller than the tolerance.

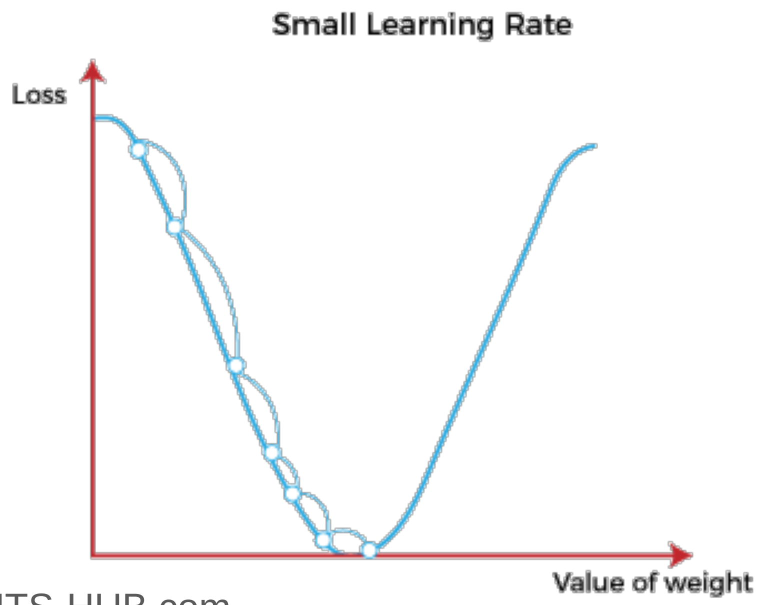
Gradient Descent Algorithm

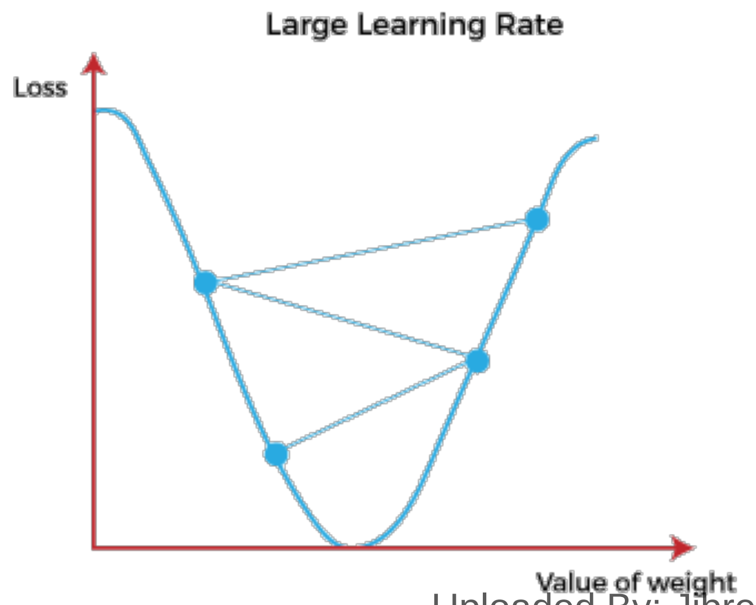


```
Algorithm   Gradient descent
Require: A starting point $x_0$, a number of iterations $T$
         a learning rate $\alpha$
  for $t = 0, \ldots, T-1$ do
    Define $x_{t+1} = x_t - \alpha \nabla f(x_t)$.
  end for
  return $x_T$
```

The learning rate α

- It is defined as the step size taken to reach the minimum or lowest point.
- This is typically a small positive value that is evaluated and updated based on the behavior of the cost function.
- If the learning rate is high, it results in larger steps but also leads to risks of overshooting the minimum.
- At the same time, a low learning rate results in small step sizes, which compromises overall efficiency but gives the advantage of more precision.





Types of Gradient Descent

- Batch gradient descent

Batch gradient descent sums the error for each point in a training set, updating the model only after all training examples have been evaluated. This process referred to as a training epoch.

- Stochastic gradient descent

Stochastic gradient descent (SGD) runs a training epoch for each example within the dataset and it updates each training example's parameters one at a time.

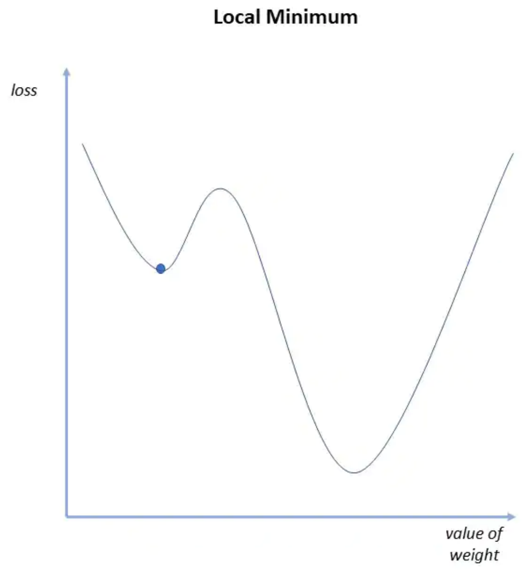
- Mini-batch gradient descent

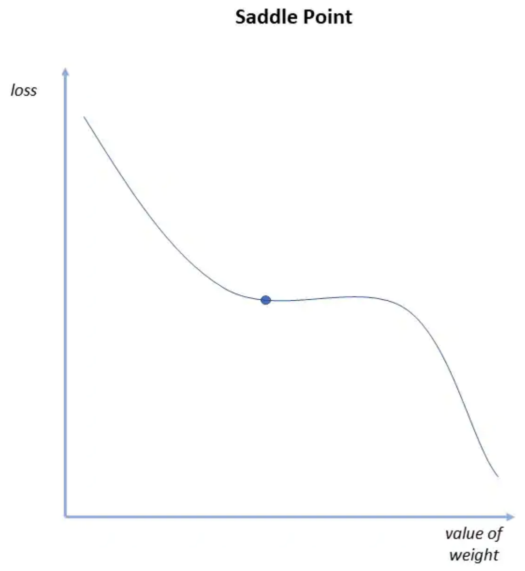
Mini-batch gradient descent combines concepts from both batch gradient descent and stochastic gradient descent. It splits the training dataset into small batch sizes and performs updates on each of those batches. This approach strikes a balance between the computational efficiency of batch gradient descent and the speed of stochastic gradient descent.

Challenges with Gradient Descent

- Local minima and saddle points

For convex problems, gradient descent can find the global minimum with ease, but as nonconvex problems emerge, gradient descent can struggle to find the global minimum, where the model achieves the best results.





- Vanishing and Exploding Gradients

Vanishing gradients occurs when the gradient is too small. Whereas exploding gradients happens when the gradient is too large, creating an unstable model. In this case, the model weights will grow too large, and they will eventually be represented as NaN.

Practical Tip: Feature Scaling

- Gradient descent converges faster if the attributes are normalized.

- Some of the techniques that are used for normalization

  - min-max

$$v' = \frac{v - min_A}{max_A - min_A}(newMax_A - newMin_A) + newMin_A$$

  - Z-score

$$v' = \frac{v - \mu_A}{\sigma_A}$$

  - Scaling

$$v' = \frac{v}{10^j}$$

Solving Linear Regression with Gradient Descent

- Remember: for linear regression the model has the following form

$$y = f(\mathbf{x}) = w_0x_0 + w_1x_1 + w_2x_2 + \dots + w_dx_d = \sum_{i=1}^{d} x_i w_i = \mathbf{x}^T \boldsymbol{w}$$

- The first step is to find the gradient vector of the error function:

$$E[\mathbf{w}] = \frac{1}{n} \sum_{i=1}^{n} (f(\mathbf{x}_i) - y_i)^2$$

$$\nabla_{\mathbf{w}} E_j = \frac{\partial}{\partial w_j} \left(\frac{1}{n} \sum_{i=1}^{n} (f(\mathbf{x}_i) - y_i)^2\right) = \frac{1}{n} \sum_{i=1}^{n} \frac{\partial}{\partial w_j} (f(\mathbf{x}_i) - y_i)^2$$

$$= \frac{2}{n} \sum_{i=1}^{n} (f(\mathbf{x}_i) - y_i) \frac{\partial}{\partial w_j} f(\mathbf{x}_i) = \frac{2}{n} \sum_{i=1}^{n} (f(\mathbf{x}_i) - y_i)\mathbf{x}_{ij}$$

- Starting with a random initial parameters $\hat{\boldsymbol{w}}$, we can iteratively update the parameters using the following update rule.

$$\hat{w}_i^{(t+1)} = \hat{w}_i^{(t)} - \alpha \frac{2}{n} \sum_{i=1}^{n} (f(\mathbf{x}_i) - y_i)\, xij$$

Example

- Perform one iteration of gradient descent to learn a linear regression model for the following dataset. Assume that the learning rate is 0.05 and the initial state is $\hat{\boldsymbol{w}}^{(0)}$ = (5, 3)^T

x_0	x_1	y
1	5	10
1	3	8
1	0	1

- Solution:

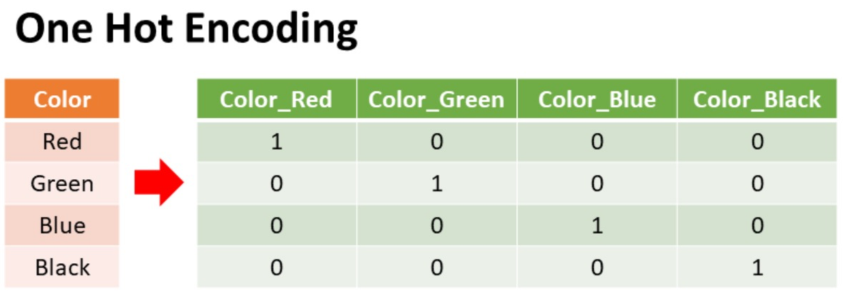
$$\nabla_w E = \begin{bmatrix} \frac{2}{n} \sum_{i=1}^{n} (f(\mathbf{x}_i) - y_i)\mathbf{x}_{i0} \\ \frac{2}{n} \sum_{i=1}^{n} (f(\mathbf{x}_i) - y_i)\mathbf{x}_{i1} \end{bmatrix} =$$

$$\begin{bmatrix} \frac{2}{3}\,[(20-10)+(14-8)+(5-1\,)] \\ \frac{2}{3}[(20-10)\cdot 5+(14-8)\cdot 3+(5-1\,)\cdot 0] \end{bmatrix} = \begin{bmatrix} 13.3 \\ 45.3 \end{bmatrix}$$

$$\hat{\boldsymbol{w}}^{(1)} = \hat{\boldsymbol{w}}^{(0)} - \alpha\,\nabla_w E = \begin{bmatrix} 5 \\ 3 \end{bmatrix} - 0.05 \begin{bmatrix} 13.3 \\ 45.3 \end{bmatrix} = \begin{bmatrix} 4.335 \\ 0.735 \end{bmatrix}$$

Handling Non-Numeric Attributes

- So far, we assumed that the features are continuous (numeric). But what if this is not the case?
- For binary attributes, we can encode them with 0 and 1.
- For categorical attributes, we can use one-hot encoding



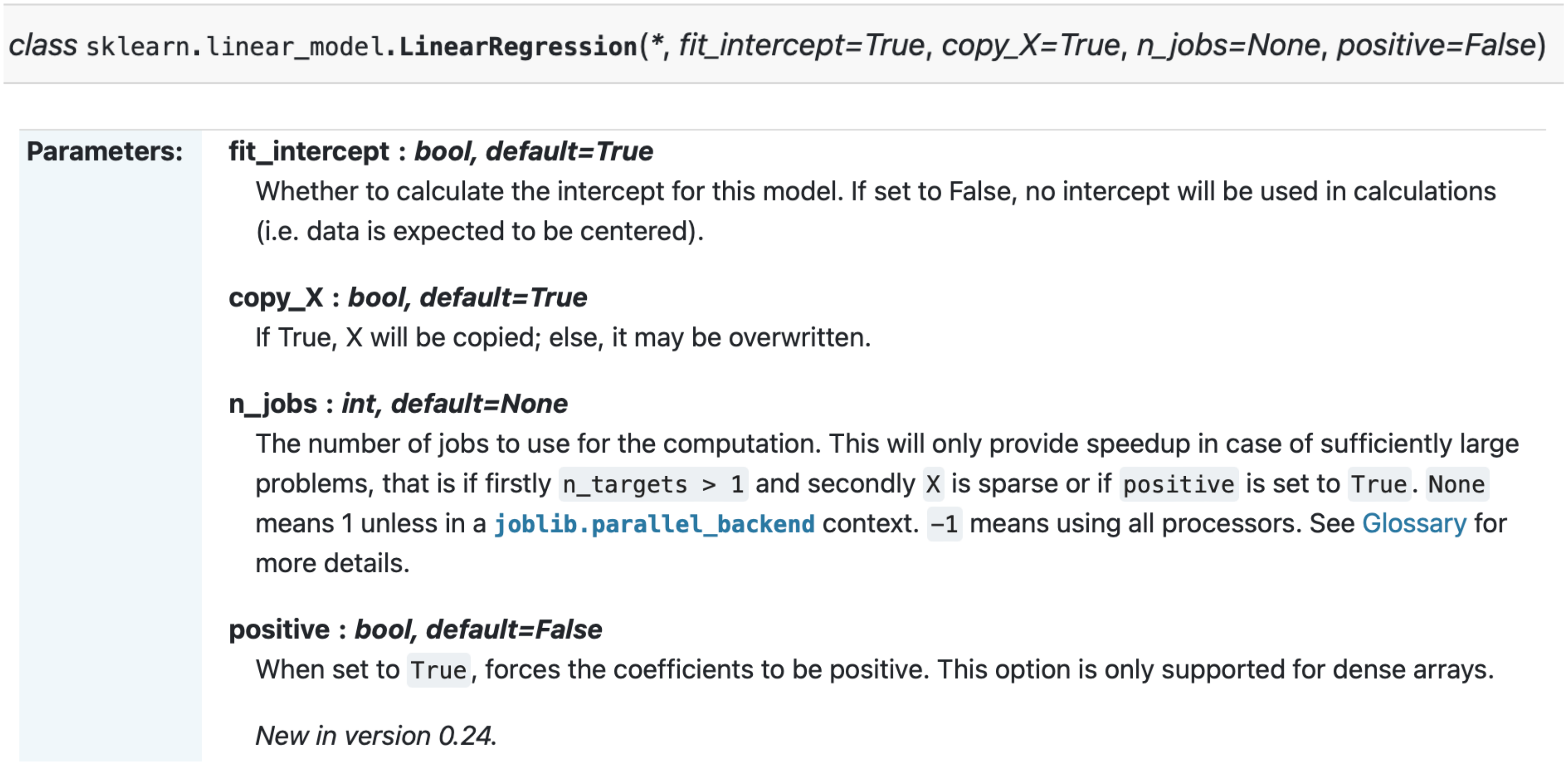
One Hot Encoding

Color
Red
Green
Blue
Black

Color_Red	Color_Green	Color_Blue	Color_Black
1	0	0	0
0	1	0	0
0	0	1	0
0	0	0	1

Linear Regression Using *scikit-learn*

- Ordinary least squares Linear Regression.



class `sklearn.linear_model.`**`LinearRegression`**(**, fit_intercept=True, copy_X=True, n_jobs=None, positive=False*)

Parameters:

fit_intercept : *bool, default=True*

Whether to calculate the intercept for this model. If set to False, no intercept will be used in calculations (i.e. data is expected to be centered).

copy_X : *bool, default=True*

If True, X will be copied; else, it may be overwritten.

n_jobs : *int, default=None*

The number of jobs to use for the computation. This will only provide speedup in case of sufficiently large problems, that is if firstly `n_targets > 1` and secondly `X` is sparse or if `positive` is set to `True`. `None` means 1 unless in a **`joblib.parallel_backend`** context. `-1` means using all processors. See Glossary for more details.

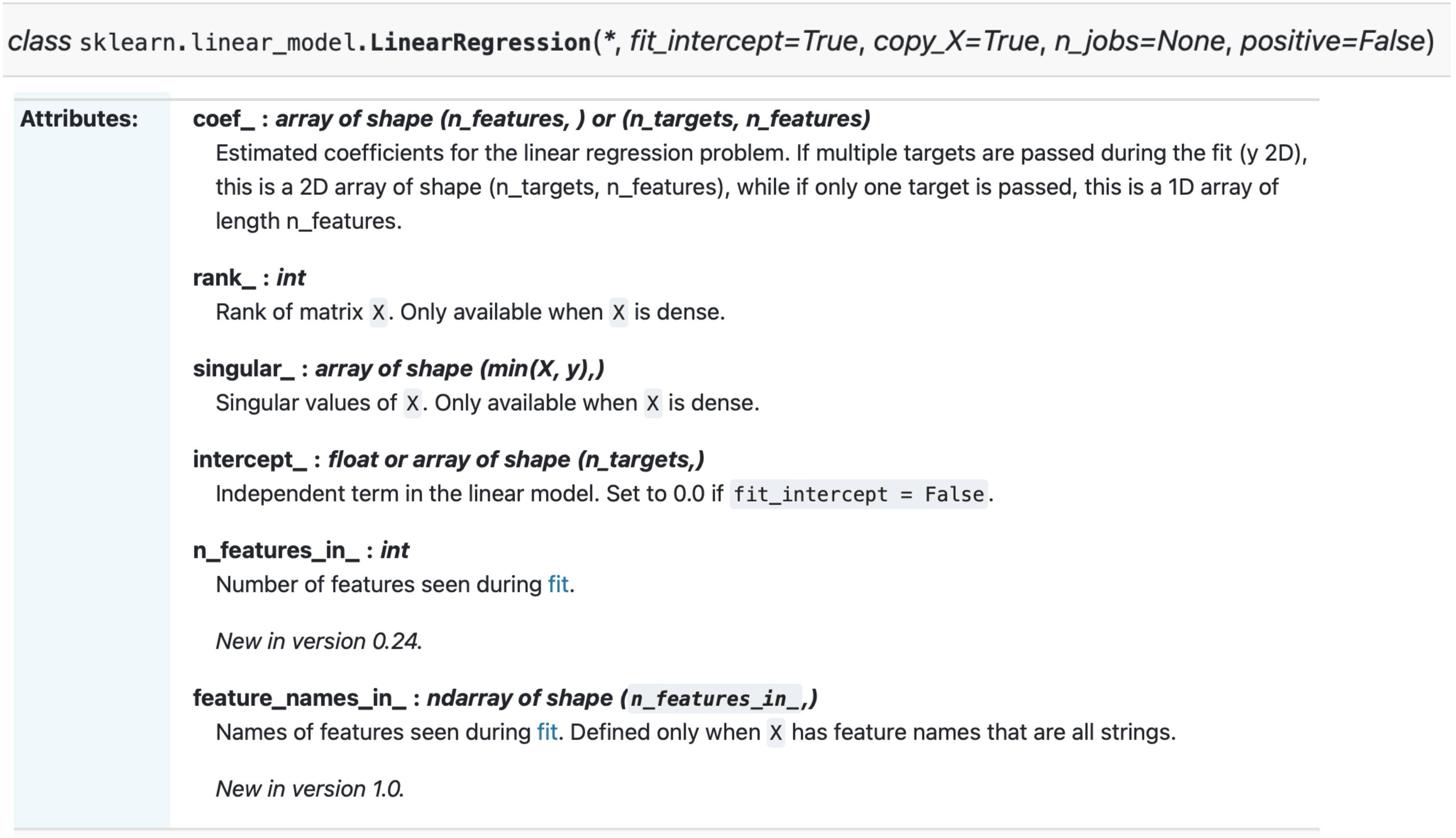
positive : *bool, default=False*

When set to `True`, forces the coefficients to be positive. This option is only supported for dense arrays.

New in version 0.24.

Linear Regression Using *scikit-learn*

- Ordinary least squares Linear Regression.



class `sklearn.linear_model.`**`LinearRegression`**(**, fit_intercept=True, copy_X=True, n_jobs=None, positive=False*)

Attributes:

coef_ : *array of shape (n_features,) or (n_targets, n_features)*
Estimated coefficients for the linear regression problem. If multiple targets are passed during the fit (y 2D), this is a 2D array of shape (n_targets, n_features), while if only one target is passed, this is a 1D array of length n_features.

rank_ : *int*
Rank of matrix `X`. Only available when `X` is dense.

singular_ : *array of shape (min(X, y),)*
Singular values of `X`. Only available when `X` is dense.

intercept_ : *float or array of shape (n_targets,)*
Independent term in the linear model. Set to 0.0 if `fit_intercept = False`.

n_features_in_ : *int*
Number of features seen during fit.

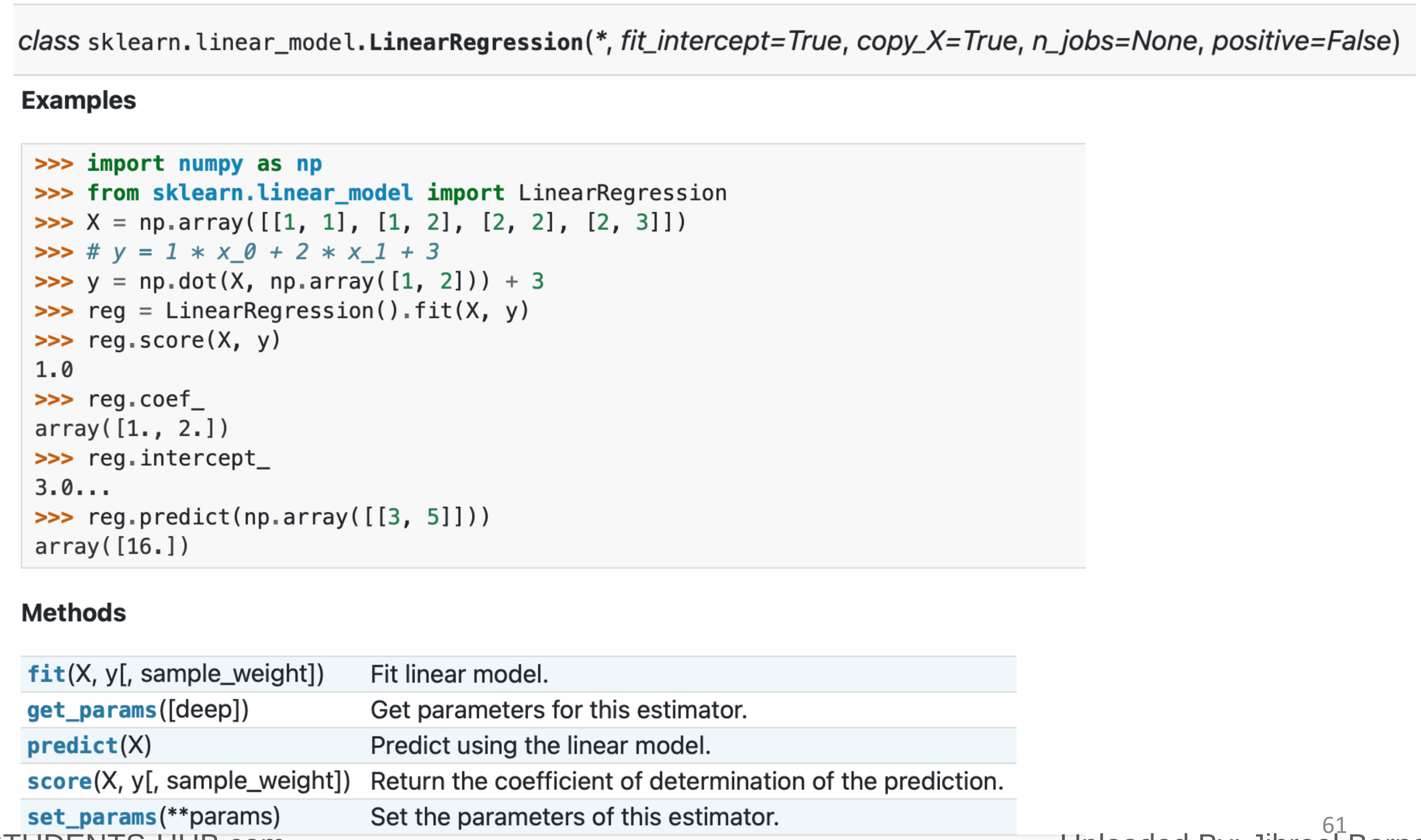
New in version 0.24.

feature_names_in_ : *ndarray of shape (`n_features_in_`,)*
Names of features seen during fit. Defined only when `X` has feature names that are all strings.

New in version 1.0.

Linear Regression Using *scikit-learn*

- Ordinary least squares Linear Regression.



class `sklearn.linear_model.`**`LinearRegression`**(**, fit_intercept=True, copy_X=True, n_jobs=None, positive=False*)

Examples

```
>>> import numpy as np
>>> from sklearn.linear_model import LinearRegression
>>> X = np.array([[1, 1], [1, 2], [2, 2], [2, 3]])
>>> # y = 1 * x_0 + 2 * x_1 + 3
>>> y = np.dot(X, np.array([1, 2])) + 3
>>> reg = LinearRegression().fit(X, y)
>>> reg.score(X, y)
1.0
>>> reg.coef_
array([1., 2.])
>>> reg.intercept_
3.0...
>>> reg.predict(np.array([[3, 5]]))
array([16.])
```

Methods

`fit`(X, y[, sample_weight])	Fit linear model.
`get_params`([deep])	Get parameters for this estimator.
`predict`(X)	Predict using the linear model.
`score`(X, y[, sample_weight])	Return the coefficient of determination of the prediction.
`set_params`(**params)	Set the parameters of this estimator.

Regression Evaluation Metrics

Regression metrics

- Most commonly used:
  - Mean squared error. $$\frac{\sum_i (y_{pred_i} - y_{actual_i})^2}{n}$$
    - Root mean squared error (RMSE) often used as well.
  - Mean absolute error. $$\frac{\sum_i \left| y_{pred_i} - y_{actual_i} \right|}{n}$$
    - Less sensitive to outliers.
- R squared $$R^2 = 1 - \frac{\sum_i (y_{pred_i} - y_{actual_i})^2}{\sum_i (y_{mean} - y_{actual_i})^2}$$
  - Between 0 and 1, but negative if the model is worse than just predicting the mean.
  - Easier to interpret (higher is better).