CE9010: Introduction to Data Science

Lecture 3: Supervised Linear Regression

Semester 2 2017/18

Xavier Bresson

School of Computer Science and Engineering
Data Science and AI Research Centre
Nanyang Technological University (NTU), Singapore

NANYANG TECHNOLOGICAL UNIVERSITY SINGAPORE

Outline

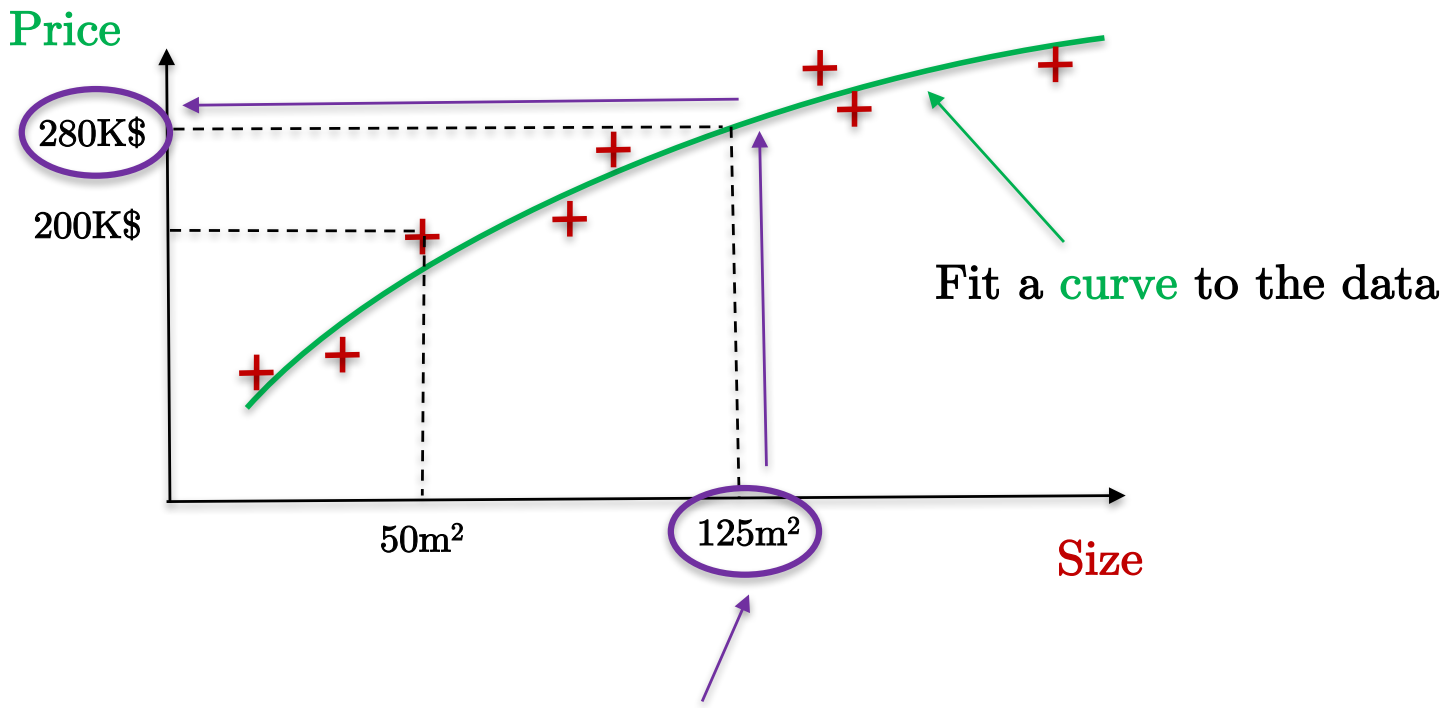
- Regression task
  - Supervised learning
  - Linear regression
  - Loss function
- Optimization with gradient descent
  - Generic case
  - Linear regression case
- Regression with multiple variables
- Non-linear regression
- Optimization with normal equations
- Conclusion

Outline

- Regression task
  - Supervised learning
  - Linear regression
  - Loss function
- Optimization with gradient descent
  - Generic case
  - Linear regression case
- Regression with multiple variables
- Non-linear regression
- Optimization with normal equations
- Conclusion

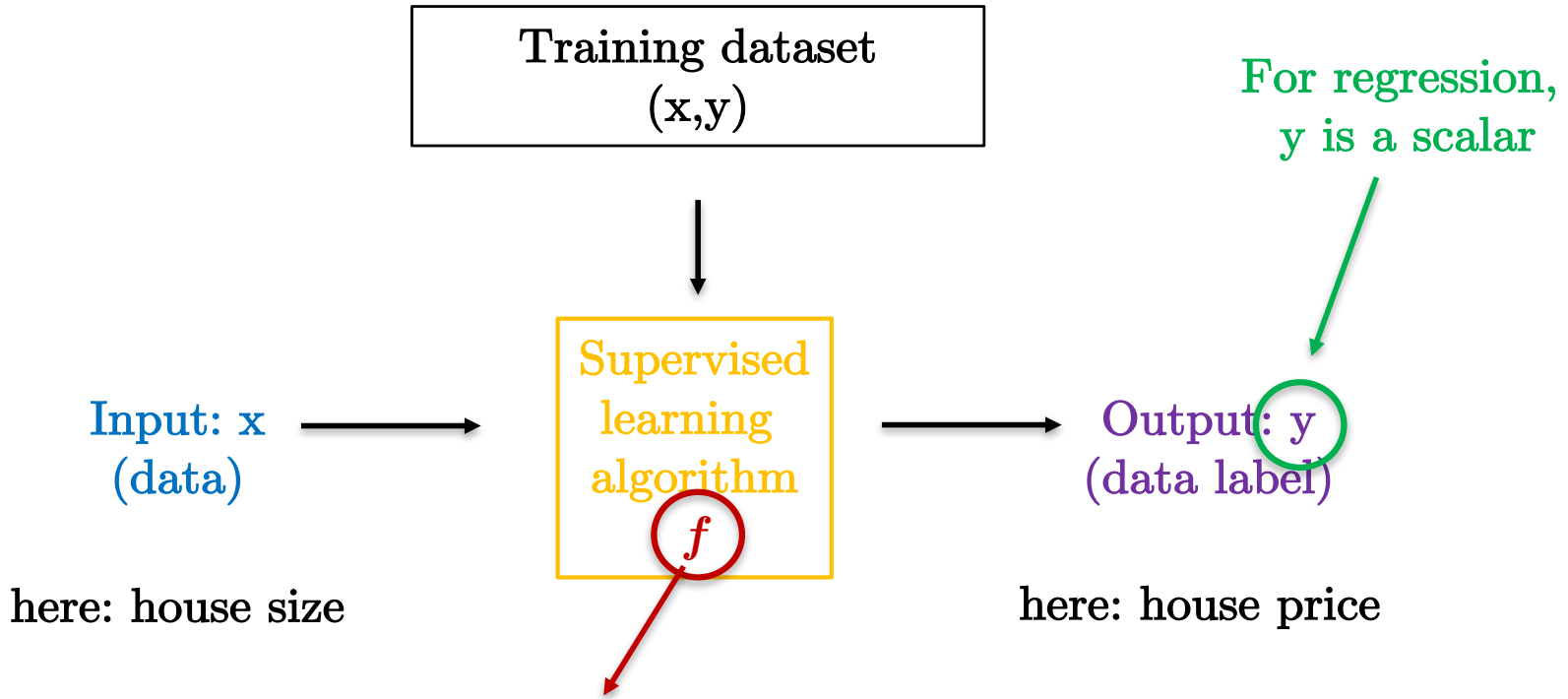
Housing price prediction

- Supervised regression problem: Predict the price (continuous value) of houses given existing data features/properties (house size).



Formalization

- Supervised regression as an example of supervised learning:

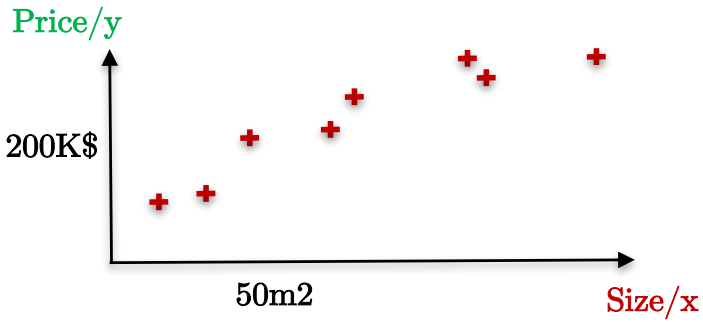


f maps data feature x to data label y

$f : x \rightarrow y$ or $y = f(x)$

Training set

- All supervised learning techniques use a training set to design the prediction function f.
- Notations:
  - n=number of training data, here n=8
  - x=input variable/feature, here x=size
  - y=output variable/label, here y=price



Size (x)	Price (y)
31	124
54	156
...	...

- One training data/sample is identified as (x_i,y_i) where i is the index of the training data. Examples: $(x_1,y_1)=(31,124)$, $(x_2,y_2)=(54,156)$.

Training set

- Performance of SL techniques depend on the size of the training sets.
- The bigger the better (super-human for some tasks):
  - Object recognition: 1.5M images
  - Self-driving cars: Millions of miles (several years)
  - Go: Millions of games
  - Medical diagnostic: Thousands of scans and radiologist reports
- The challenging issue is data collection: Time consuming, expensive, sometimes not available.
- Humans transfer biases to data. Talk by Kate Crawford at NIPS17: https://nips.cc/Conferences/2017/Schedule?showEvent=8742

Quiz

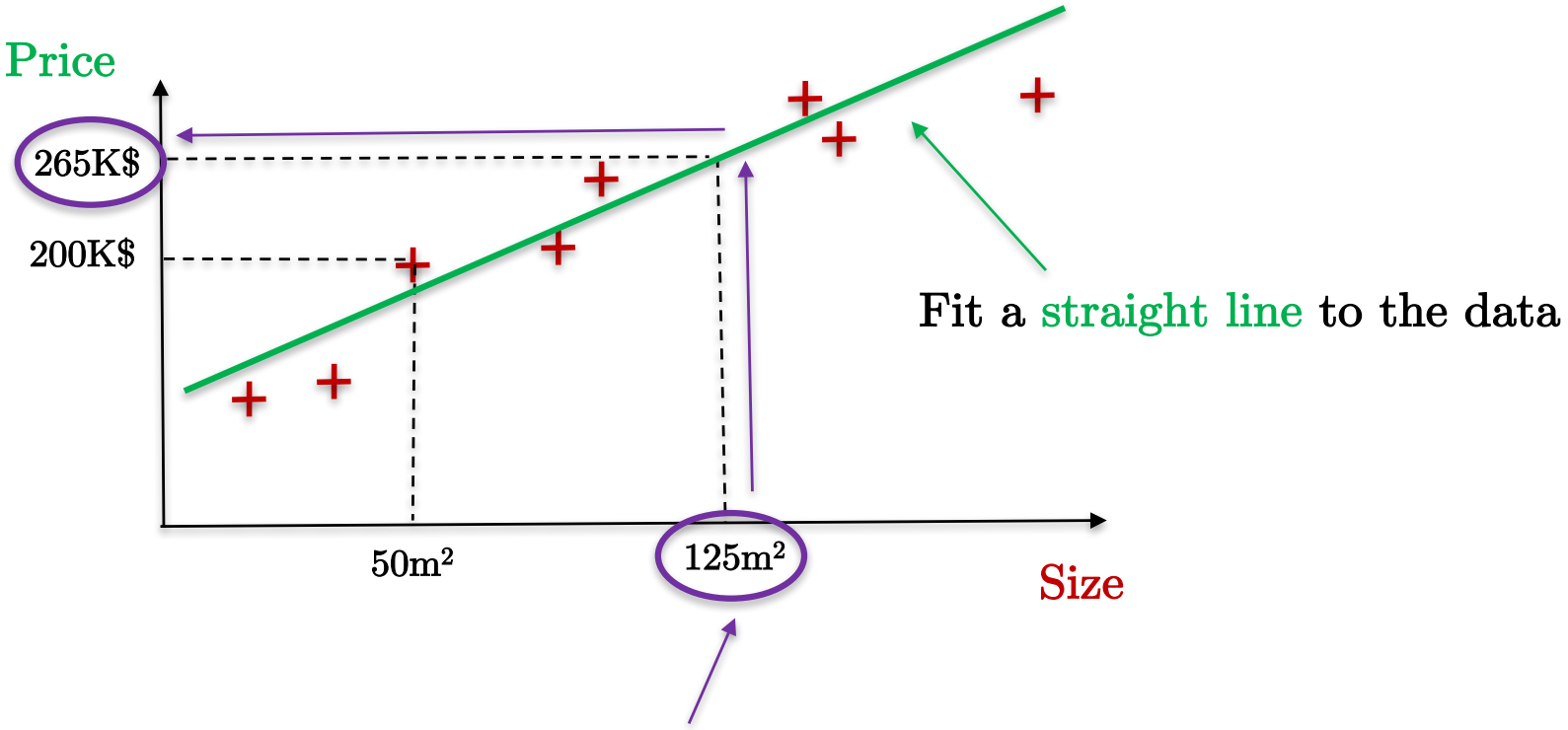
- Quality of supervised learning depends on training set. Let's take an example, we want to develop a learning algorithm that predict student marks for a particular course like algebra given students profile:
  - How much time to collect students profile?
  - How much time to collect final exams?
  - How much time to digitalize them?
  - How many profiles do we need to get a good estimation?

Outline

- Regression task
  - Supervised learning
  - Linear regression
  - Loss function
- Optimization with gradient descent
  - Generic case
  - Linear regression case
- Regression with multiple variables
- Non-linear regression
- Optimization with normal equations
- Conclusion

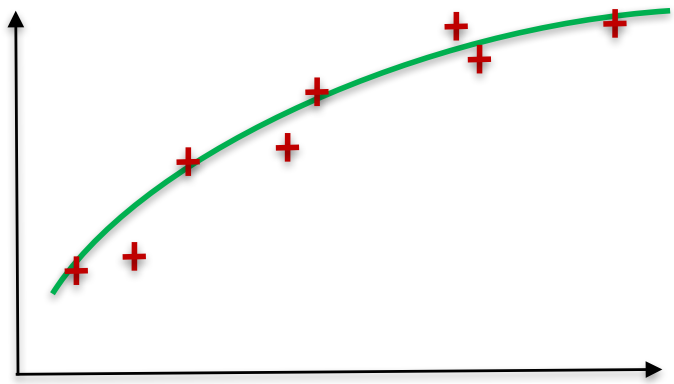
Linear regression

- What is the simplest model representation to regress the data?
  Straight line.



Model representation

- How to represent the prediction function f ?

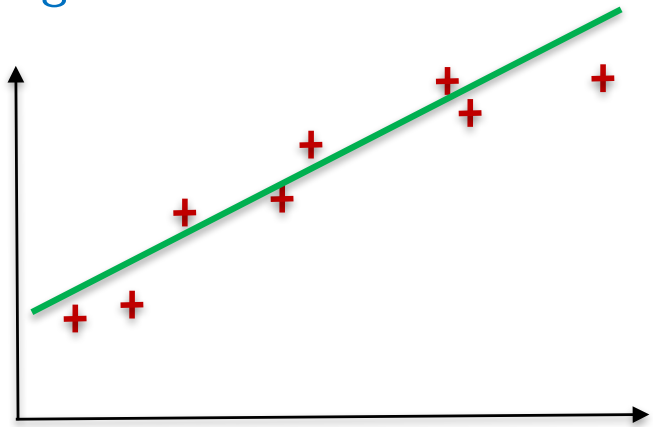


$$y = f(x)$$

Non-linear function

Example: Neural networks

- Case of linear regression:

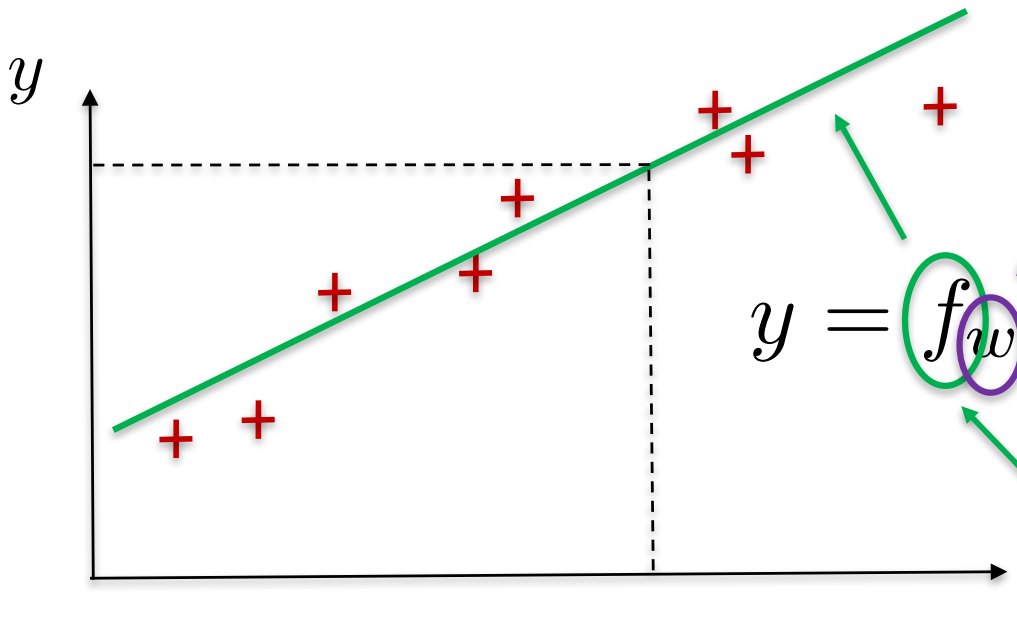


$$y = f(x)$$

Linear function

Linear representation

- Predictor function f as a straight line:



Compact representation of all variables/parameters of prediction function f

$$w = (w_0, w_1)$$

Data feature

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

Model parameters

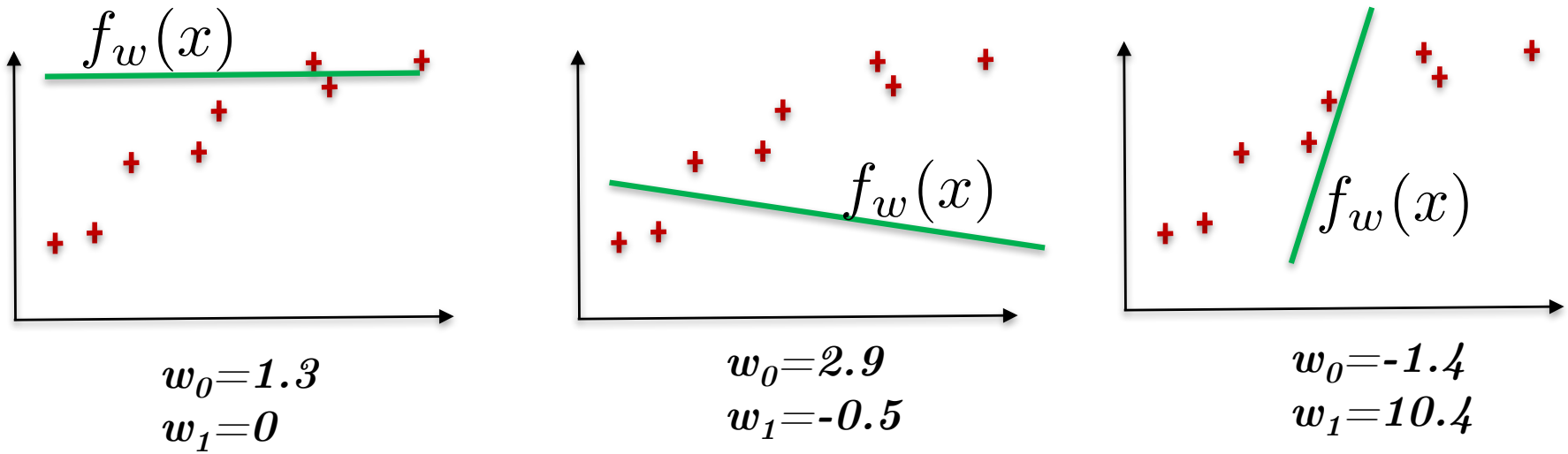
Linear predictive function/ Linear regression model with one single variable x

Prediction function parameters

- Prediction function: $f_w(x) = w_0 + w_1 x$

Model parameters

- Influence of different parameter values (w_0, w_1) on the prediction:



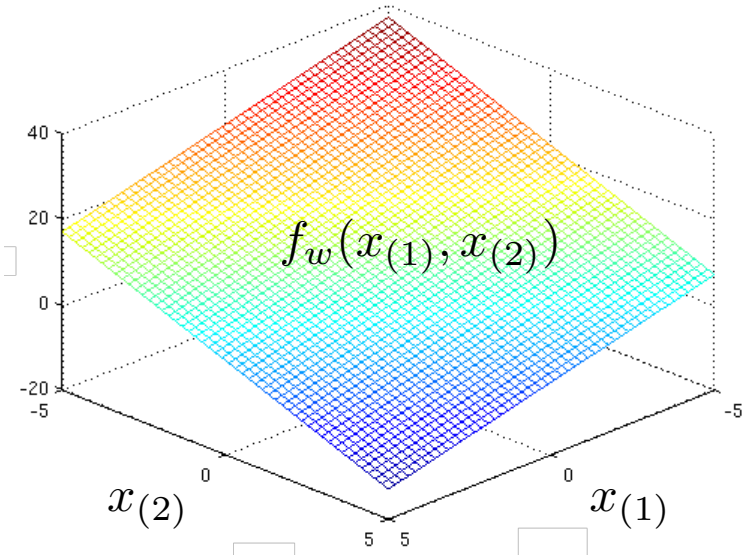
- How to find the best possible straight line (w_0, w_1) that fits all data?
  A fitness measure is required ⇒ Loss function.

Quiz

- Linear prediction function with 2 data features:

$$f_w(x_{(1)}, x_{(2)}) = w_0 + w_1 x_{(1)} + w_2 x_{(2)}$$

- What is the representation of the prediction function with 2 variables?

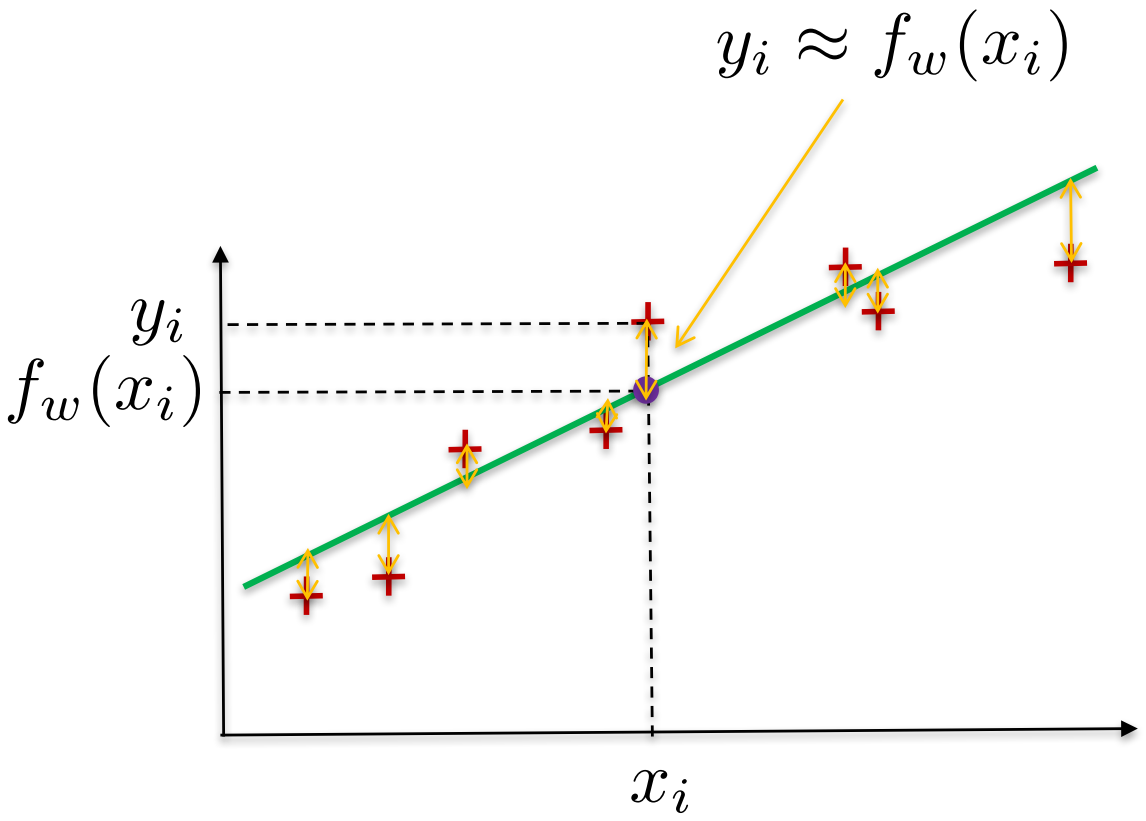


Outline

- Regression task
  - Supervised learning
  - Linear regression
  - Loss function
- Optimization with gradient descent
  - Generic case
  - Linear regression case
- Regression with multiple variables
- Non-linear regression
- Optimization with normal equations
- Conclusion

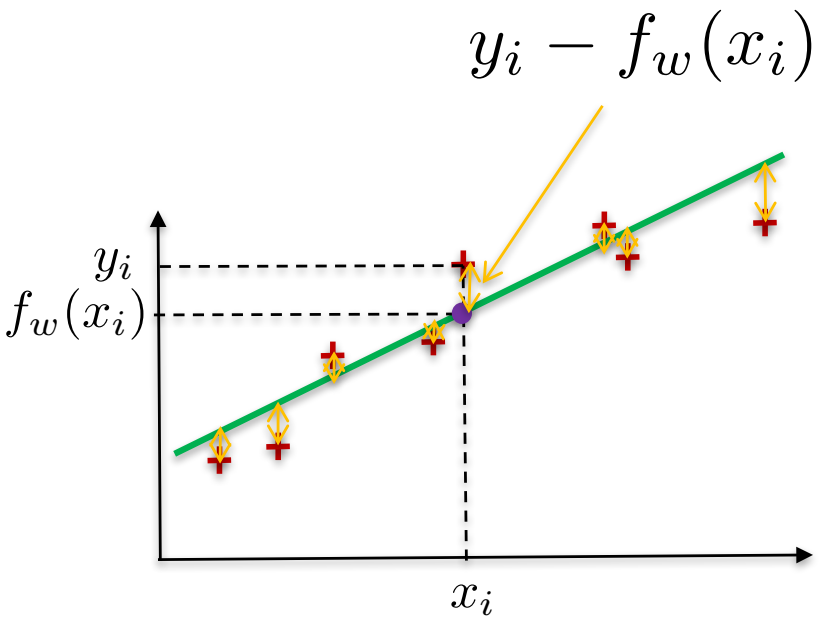
How to choose the parameters?

- Idea: Choose parameters (w_0, w_1) such that $f_w(x_i)$ is close to y_i for all training data (x_i, y_i), that is



Formalization

- Find parameters (w_0, w_1) by solving a minimization problem:



$$\min_{w=(w_0,w_1)} \underbrace{\frac{1}{n}\sum_{i=1}^{n}}_{\text{Average prediction error}} \underbrace{\left(\underbrace{f_w(x_i)}_{\text{Prediction given } x_i} - y_i\right)^2}_{\text{Prediction error}}$$

Sum over the n training data

$$L(w)$$

Loss function

- Remarks:
  - Generic optimization problem in supervised learning (f is not specified here).
  - Loss function is also called cost function.
  - This loss function is called mean square error (most used regression loss).

Linear regression loss

- From generic to linear regression task:

$$\min_{w=(w_0,w_1)} \frac{1}{n}\sum_{i=1}^{n}\Big(f_w(x_i)-y_i\Big)^2$$

$$\Downarrow \quad f_w(x_i) = w_0 + w_1 x_i$$

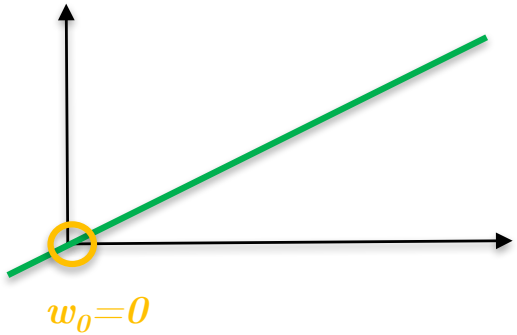
$$\min_{w=(w_0,w_1)} \underbrace{\frac{1}{n}\sum_{i=1}^{n}\Big(w_0+w_1x_i-y_i\Big)^2}_{L(w_0,w_1)}$$

Linear regression loss

Loss analysis

- **Prediction function:** $f_w(x) = w_0 + w_1 x$
- **Parameters:** w_0, w_1
- **Loss function:** $L(w_0, w_1) = \frac{1}{n} \sum_{i=1}^{n} \left(w_0 + w_1 x_i - y_i \right)^2$
- **Optimization:** $\min_{w=(w_0,w_1)} L(w_0, w_1)$

- **Let us simplify with a single parameter: $w_0=0$, $w_1=w$**



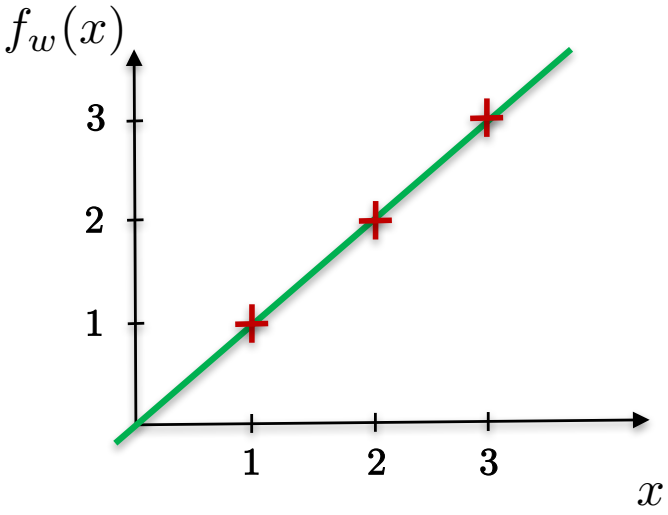
$$f_w(x) = wx$$

$$\min_w L(w) = \frac{1}{n} \sum_{i=1}^{n} \left(w x_i - y_i \right)^2$$

Loss analysis

Prediction function $f_w(x)$

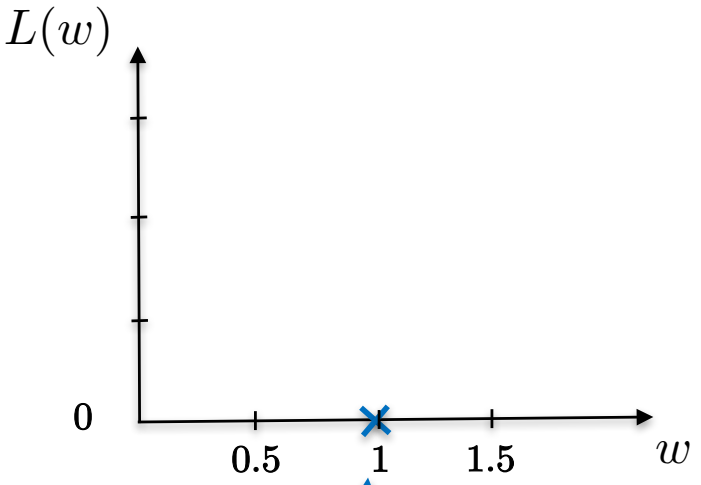
Data feature



$$f_w(x) = x, \quad w = 1$$

Loss function $L(w)$

Prediction parameter



$$L(w=1) = \frac{1}{n}\sum_{i=1}^{n}\left(x_i - y_i\right)^2$$

$$\frac{1}{3}\left((1-1)^2 + (2-2)^2 + (3-3)^2\right) = 0$$

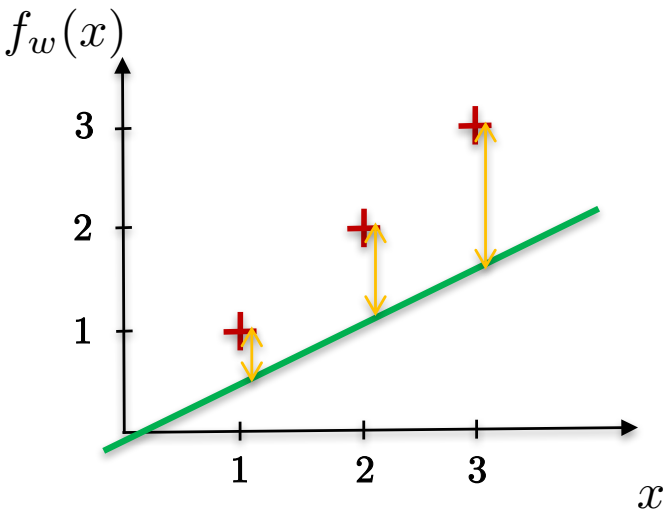
$$L(w=1) = 0$$

No error in prediction $\Rightarrow$ loss=0

Loss analysis

Prediction function $f_w(x)$

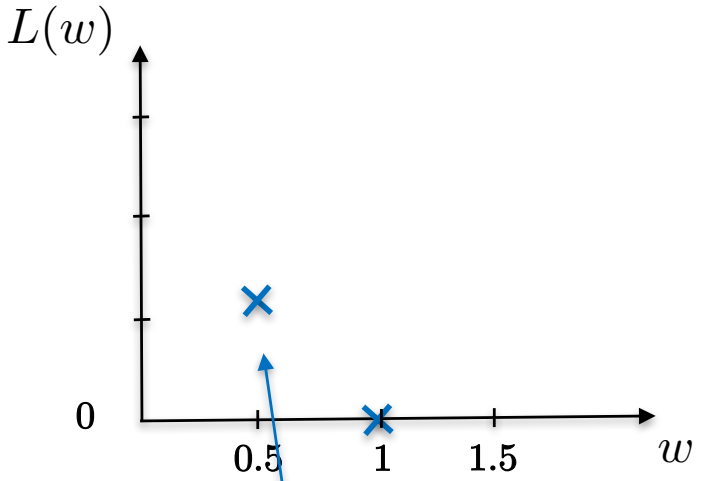
Data feature



$$f_w(x) = x, w = 0.5$$

Loss function $L(w)$

Prediction parameter



$$L(w=1) = \frac{1}{n}\sum_{i=1}^{n}\left(0.5x_i - y_i\right)^2$$

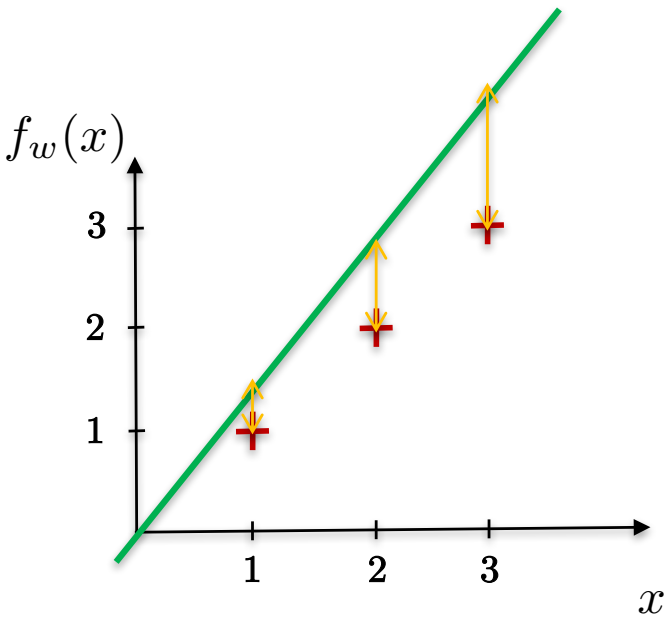
$$\frac{1}{3}\left((0.5-1)^2 + (1-2)^2 + (1.5-3)^2\right) = 1.16$$

$$L(w=0.5) = 1.16$$

Loss analysis

Prediction function $f_w(x)$

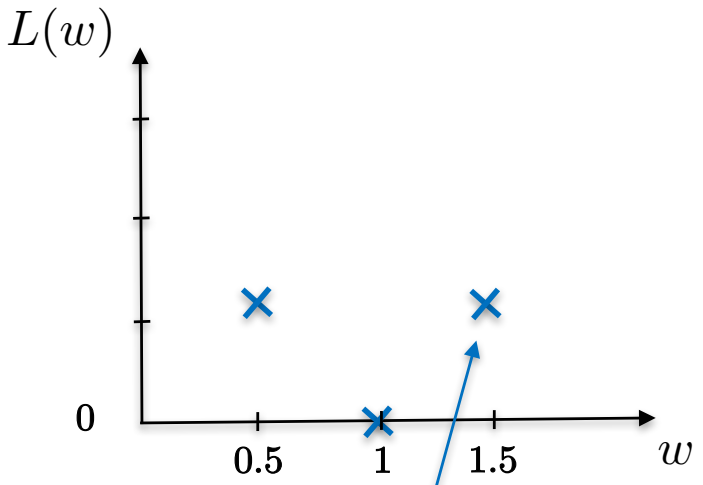
Data feature



$$f_w(x) = x, w = 1.5$$

Loss function $L(w)$

Prediction parameter



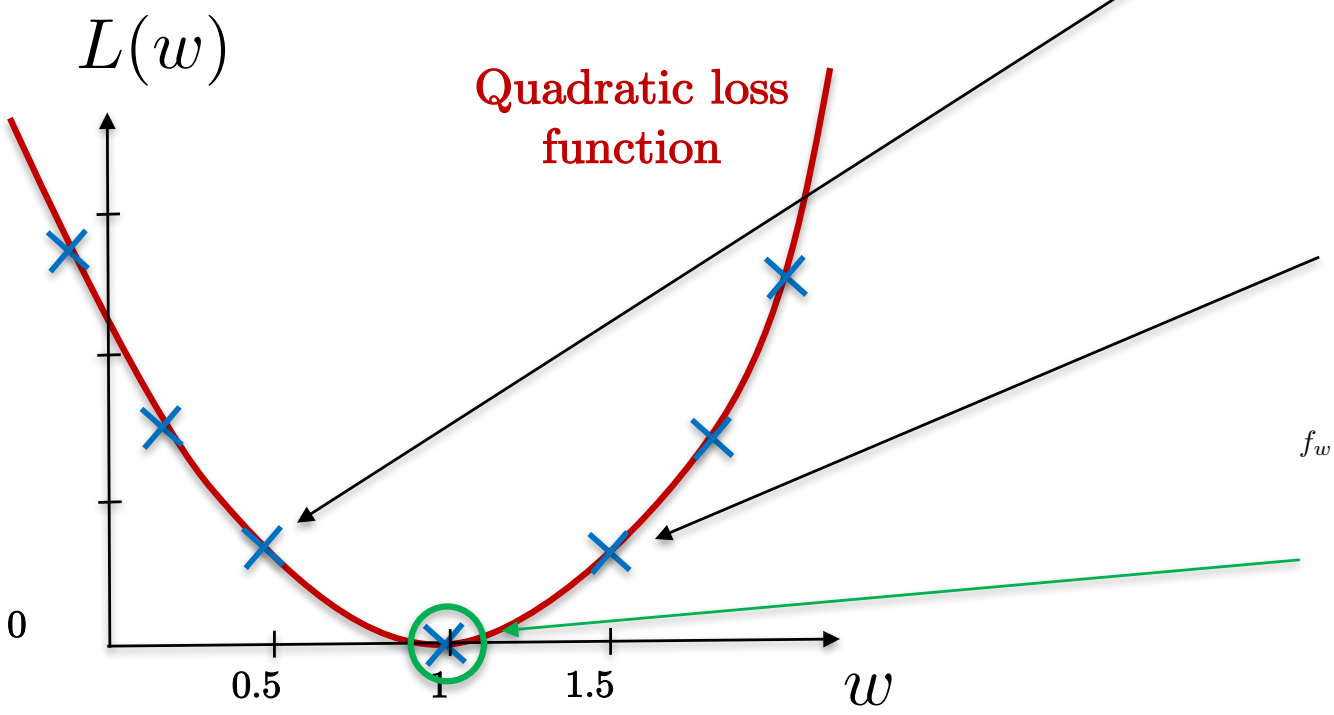
$$L(w = 1.5) = \frac{1}{n}\sum_{i=1}^{n}\left(1.5x_i - y_i\right)^2$$

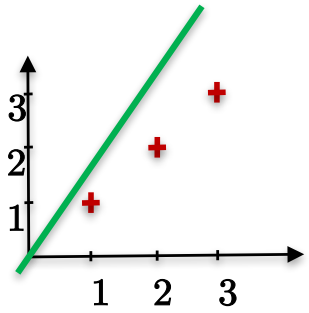
$$\frac{1}{3}\left((1.5-1)^2 + (3-2)^2 + (4.5-3)^2\right) = 1.16$$

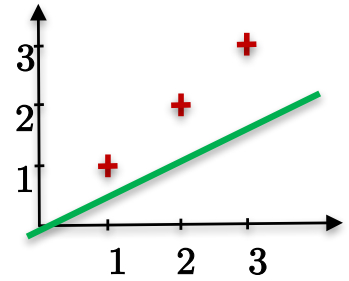
$$L(w = 1.5) = 1.16$$

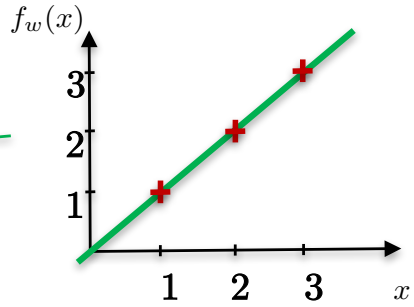
Loss analysis

- Compute $L(w)$ for multiple values w:







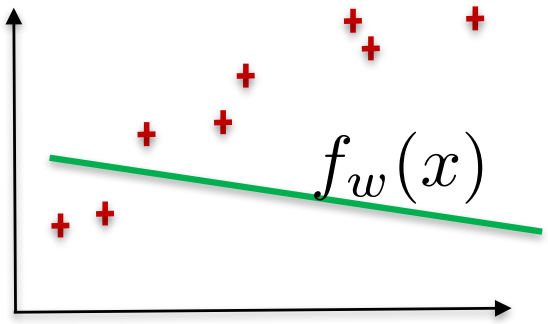


$$\min_w \; L(w) = \frac{1}{n}\sum_{i=1}^{n}\left(wx_i - y_i\right)^2 \quad \Rightarrow \quad w = 1$$

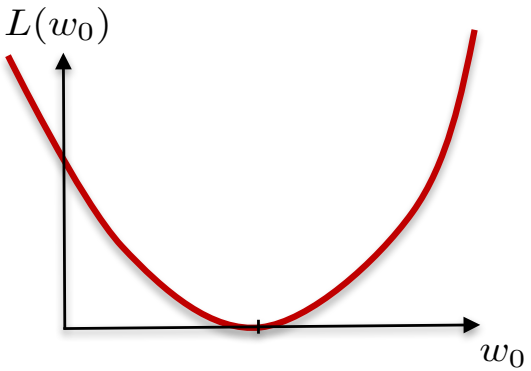
Parameter w=1 provides a perfect prediction function for all training data.

Loss function with 2 parameters

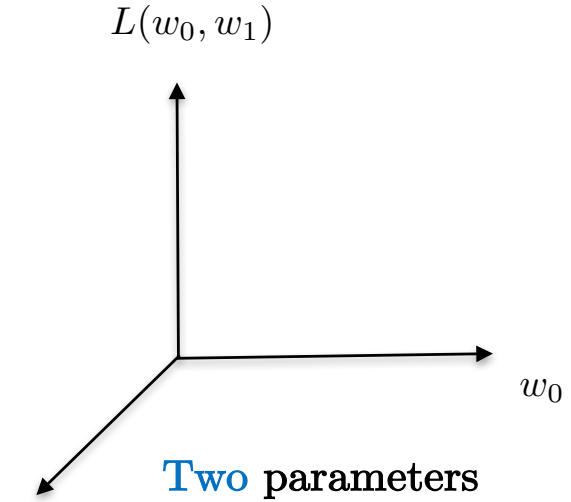
- **Prediction function:** $f_w(x) = w_0 + w_1 x$
- **Parameters:** w_0, w_1
- **Loss function:** $L(w_0, w_1) = \frac{1}{n} \sum_{i=1}^{n} \left(w_0 + w_1 x_i - y_i \right)^2$
- **Optimization:** $\min_{w=(w_0,w_1)} L(w_0, w_1)$



$w_0 = 2.9$
$w_1 = -0.5$



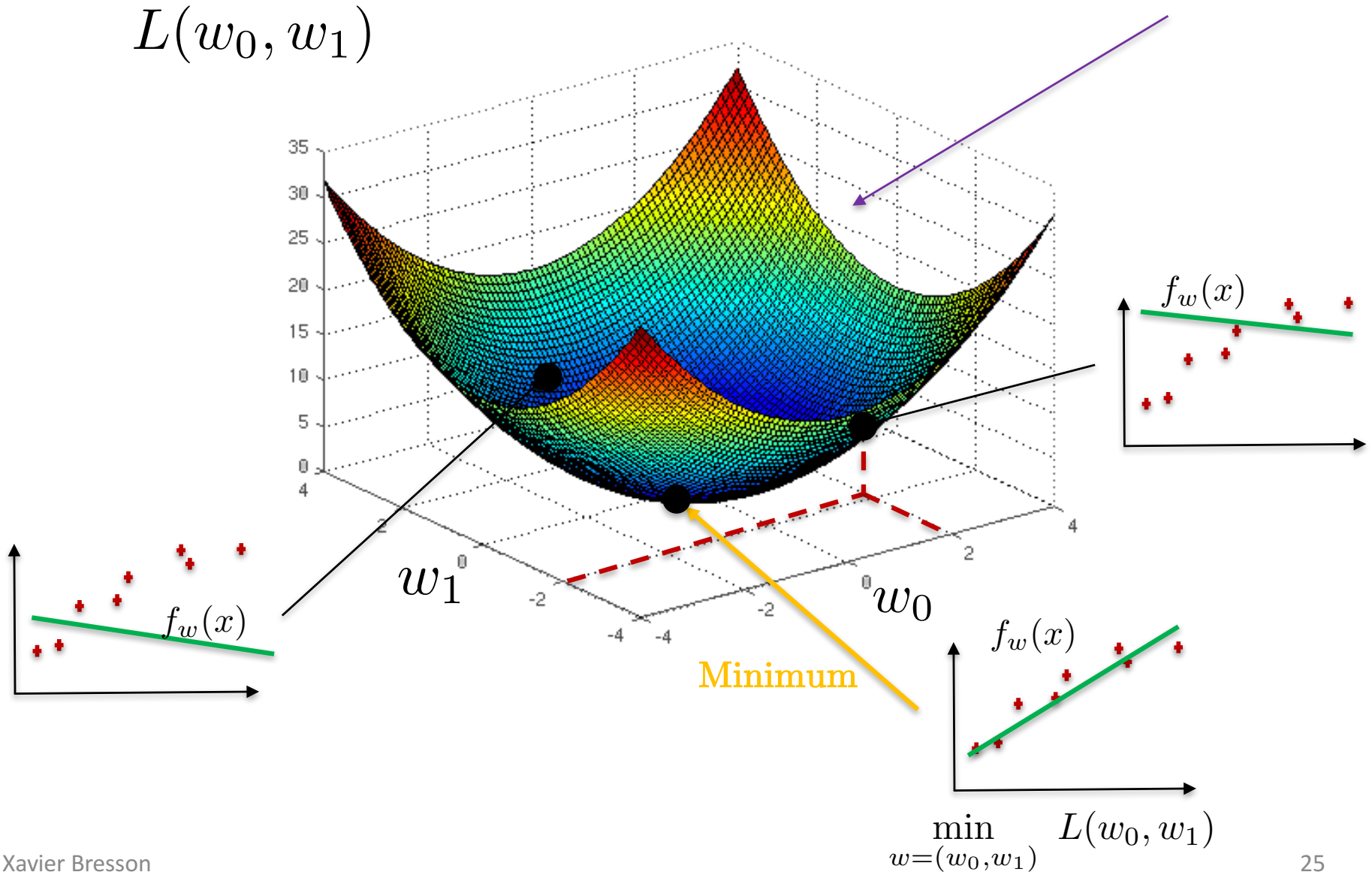
Single parameter
Loss is represented
by a curve



Two parameters
What is the loss
representation? 3D surface

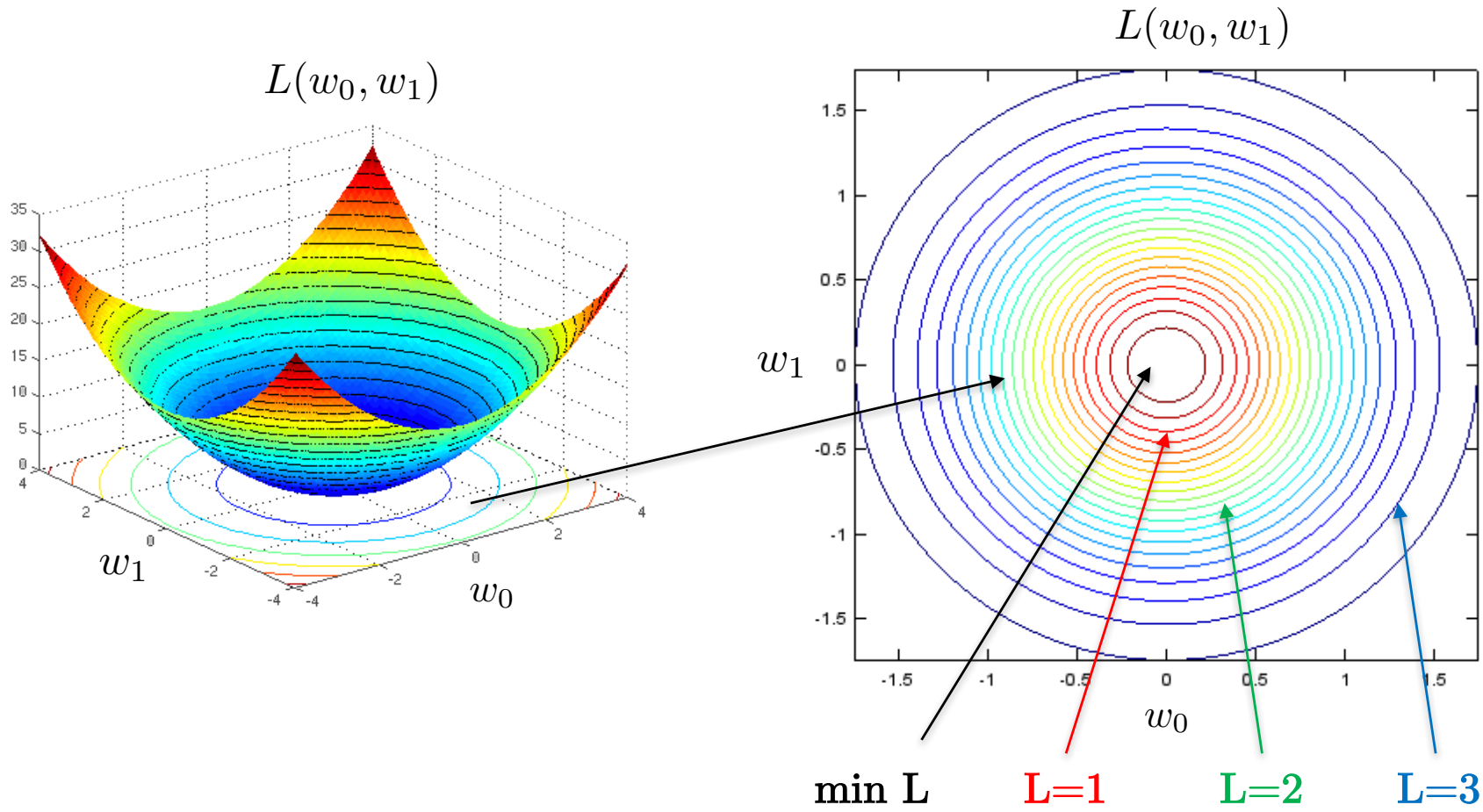
Loss function with 2 parameters

Loss is represented by a 3D surface.
It is called the landscape.

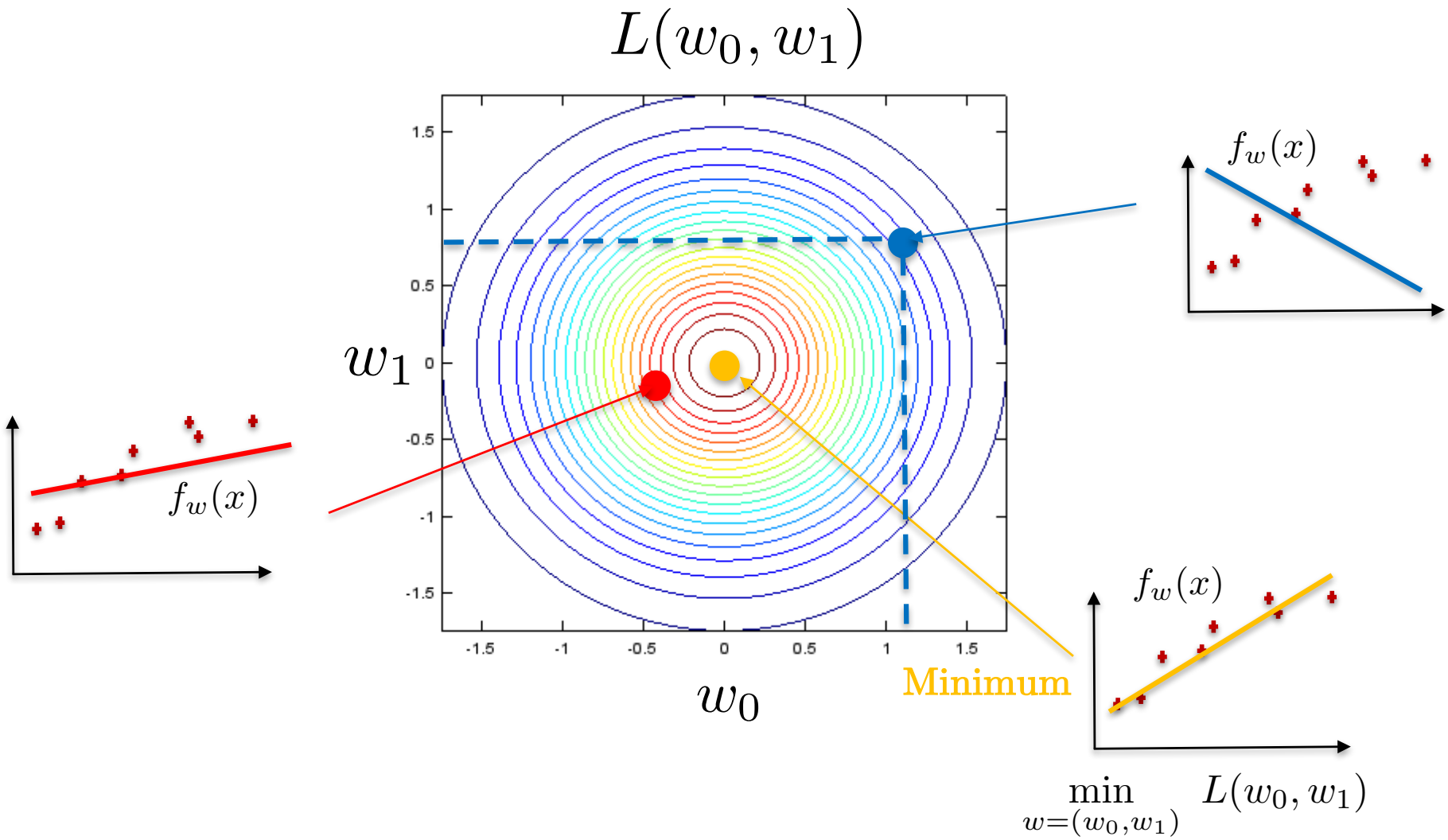


Contours of L

- Level sets/iso-contours = curves with the same loss values.
- They offer a 2D visualization of the 3D loss surface.



Loss function with 2 parameters



- How to find the value of the parameters (w_0, w_1) that minimize the loss?
  $\Rightarrow$ Gradient descent.

Quiz

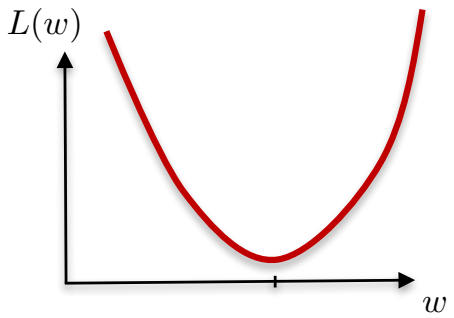
- Expression of the mean square error loss:

$$L(w) = \frac{1}{n} \sum_{i=1}^{n} \left(f_w(x_i) - y_i \right)^2$$

- Can we use alternate regression loss?
  - Mean absolute error loss:

$$L(w) = \frac{1}{n} \sum_{i=1}^{n} \left| f_w(x_i) - y_i \right|$$

  - Preferably, any convex function:

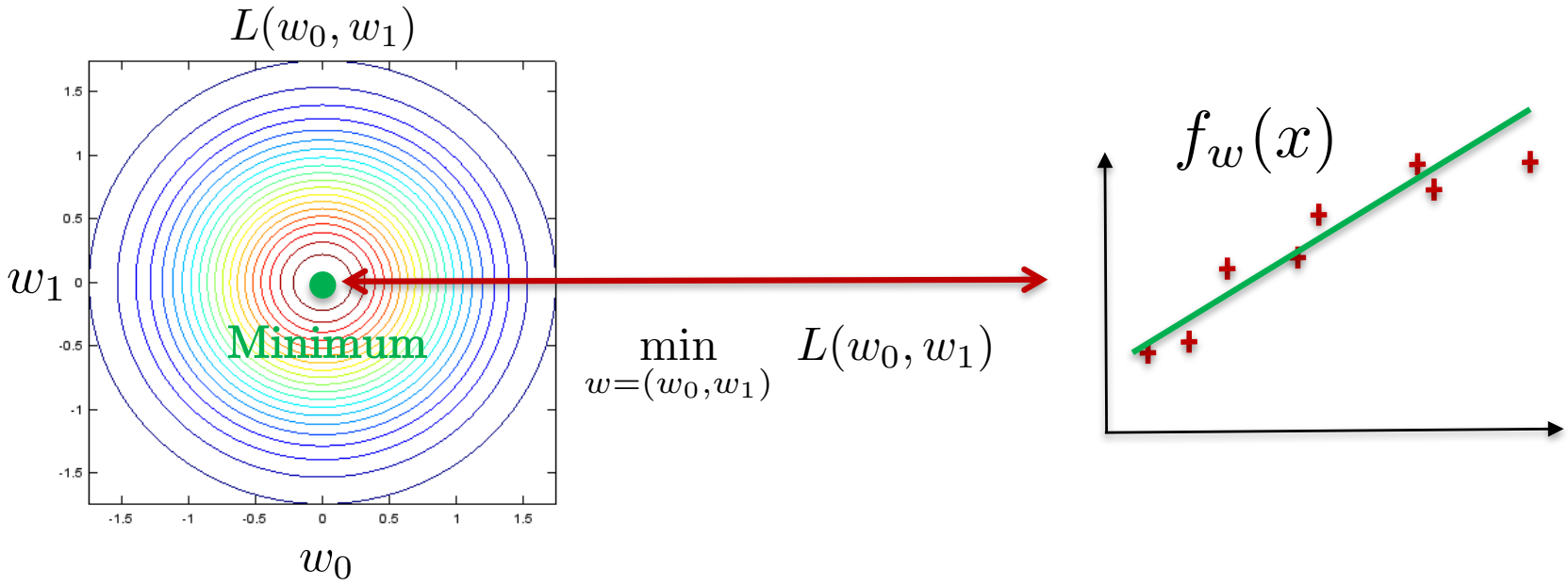


Outline

- Regression task
  - Supervised learning
  - Linear regression
  - Loss function
- **Optimization with gradient descent**
  - **Generic case**
  - Linear regression case
- Regression with multiple variables
- Non-linear regression
- Optimization with normal equations
- Conclusion

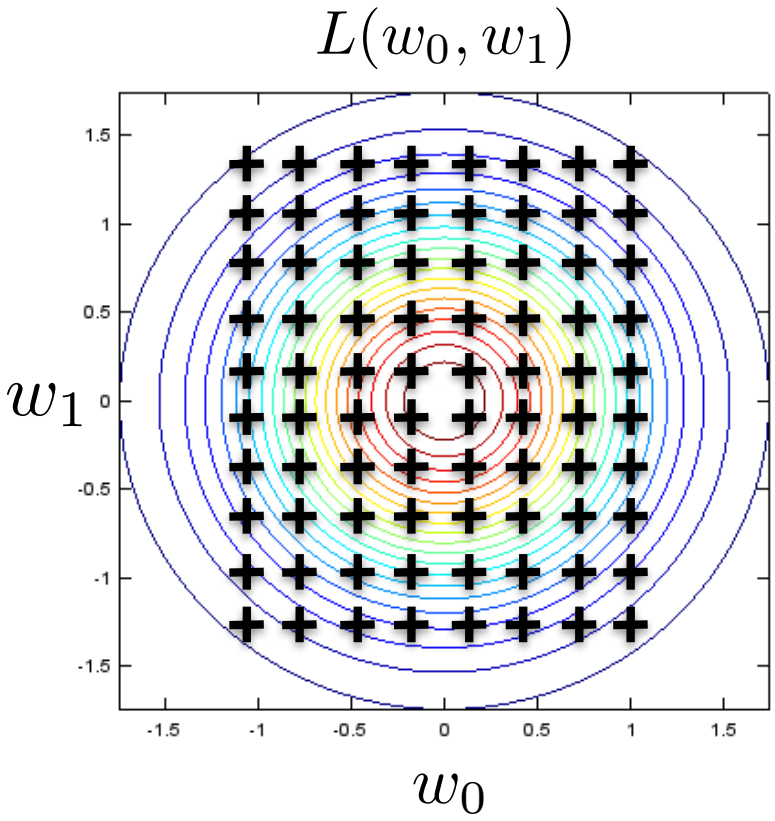
Optimization problem

- **Prediction function:** $f_w(x) = w_0 + w_1 x$
- **Parameters:** w_0, w_1
- **Loss function:** $L(w_0, w_1) = \frac{1}{n} \sum_{i=1}^{n} \left(w_0 + w_1 x_i - y_i \right)^2$
- **Optimization:** $\min_{w=(w_0,w_1)} L(w_0, w_1)$



Optimization by brute-force approach

- Try all possible parameter values (w_0, w_1) to find the minimizer:

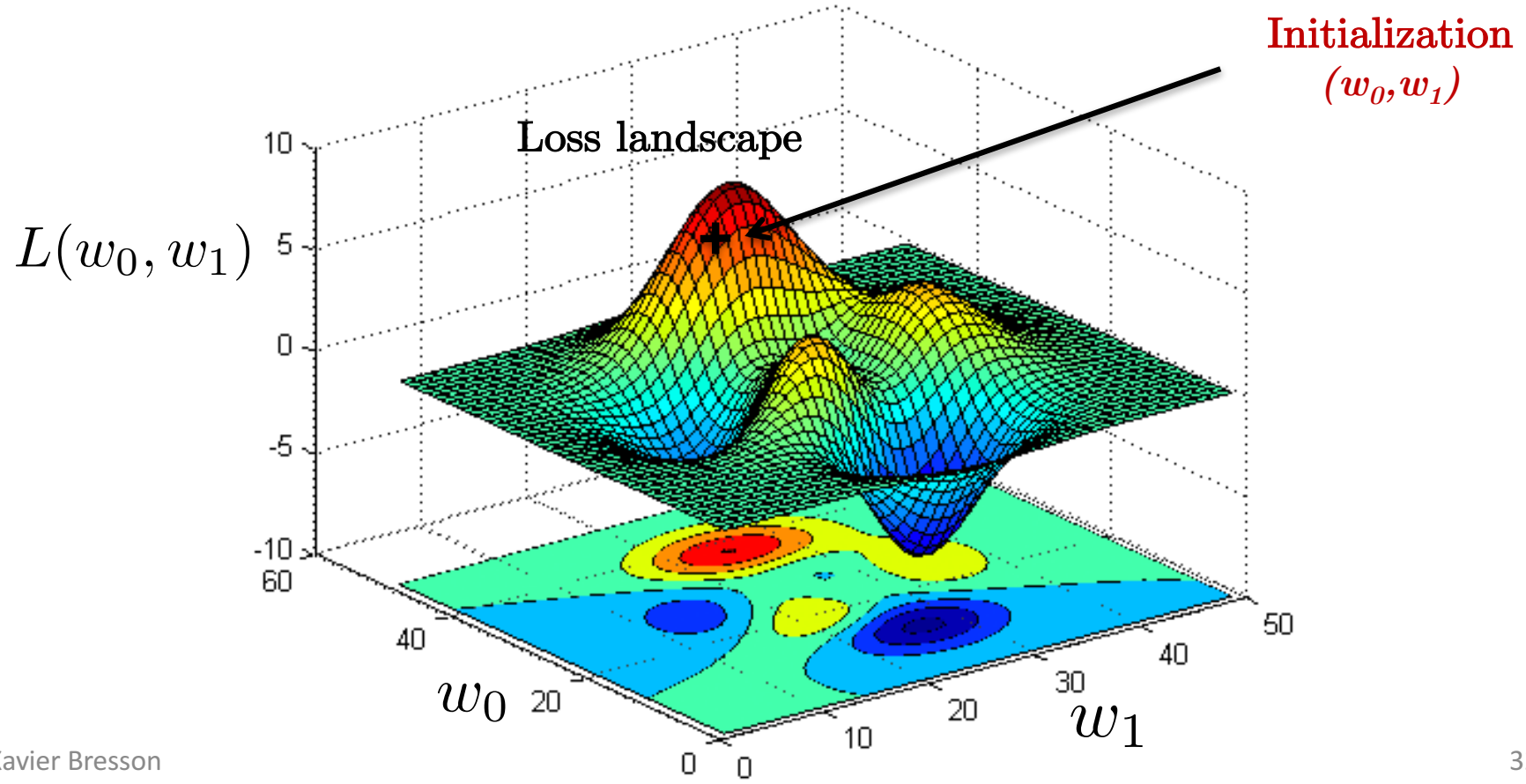


- Complexity: $N^{dim} \Rightarrow$ Exponential ☹

  Here, N=8, dim=2

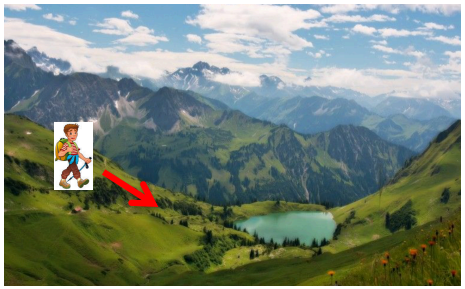
Optimization by gradient descent

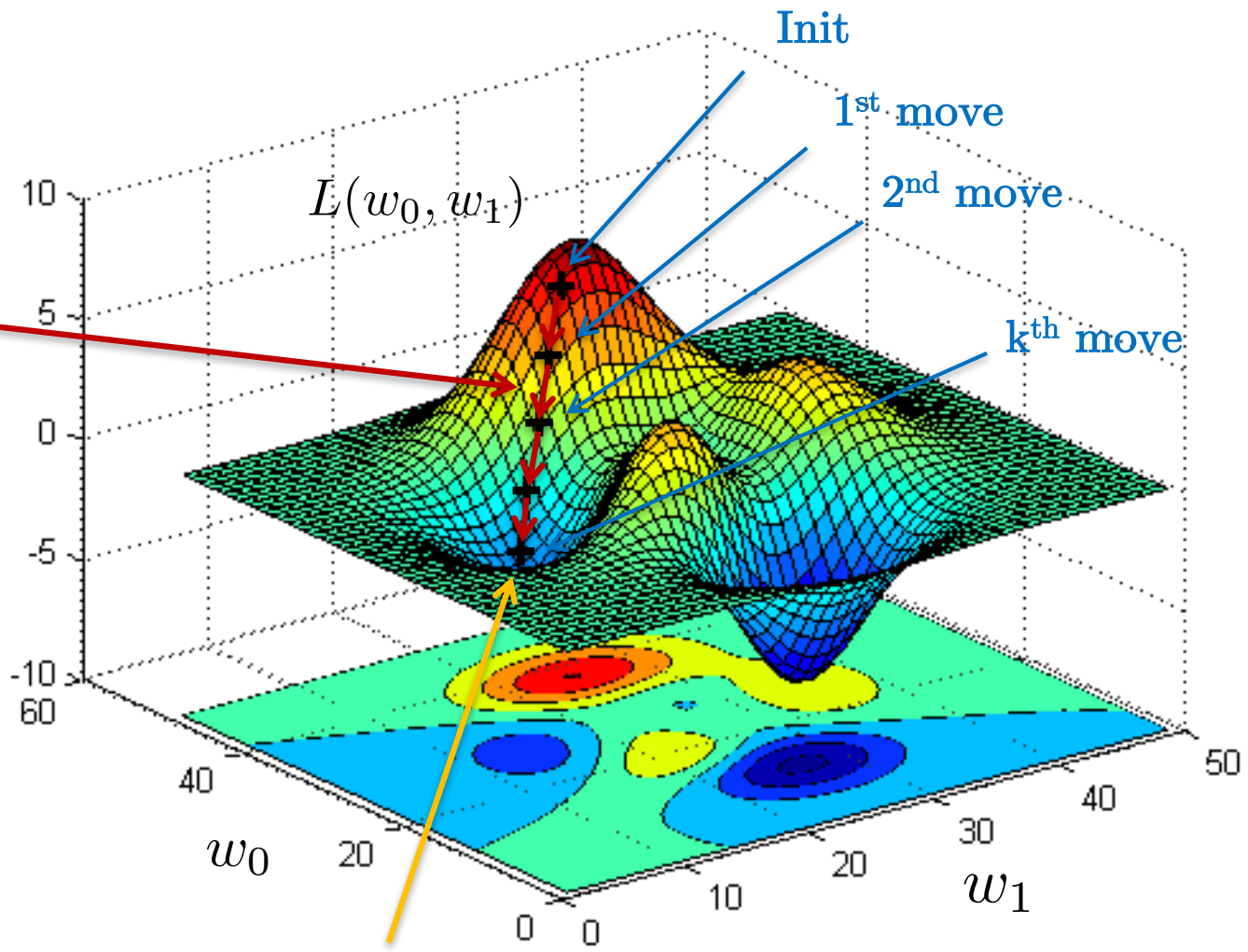
- Gradient descent technique:
  - Initialization: Start with some values (w_0, w_1).
  - Iterate (loop): Keep updating (w_0, w_1) to reduce the loss value $L(w_0, w_1)$ until a minimum is reached (when no possible update is possible).



Dynamic process

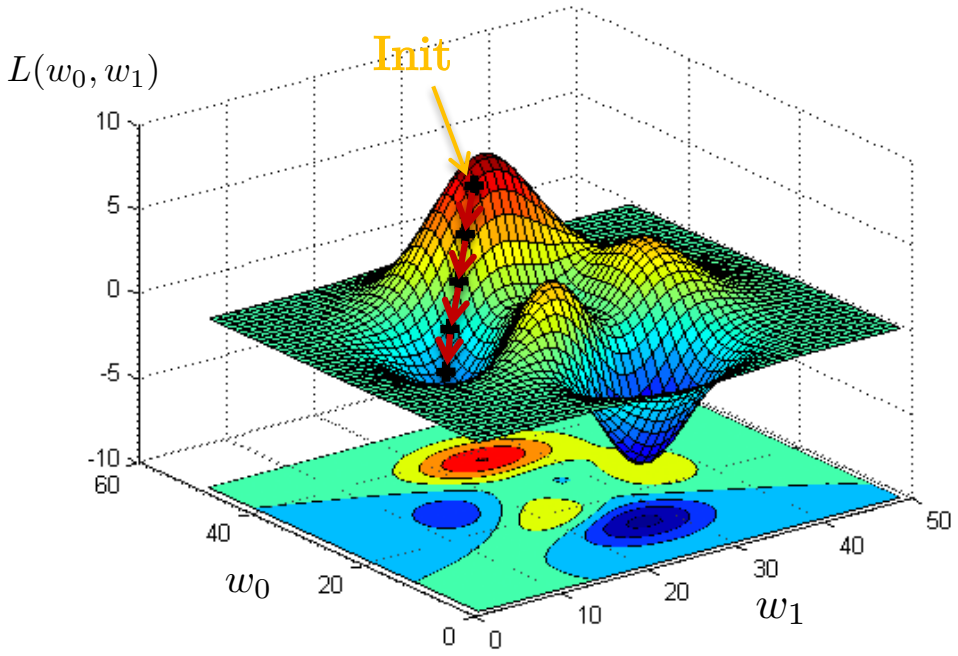
Idea: Move the values of (w_0, w_1) in the direction of the steepest descent to decrease the value of the loss L as fast as possible.



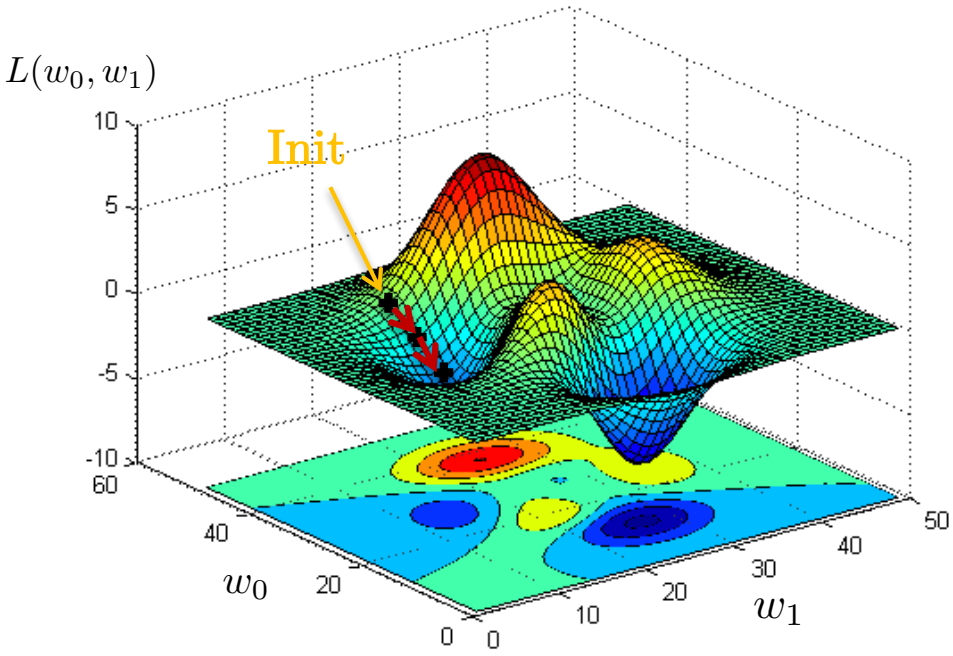


Initialization

- How to start?
  - Initialization can be random.
  - It can also be estimated if additional information on L is available.



Initialization #1

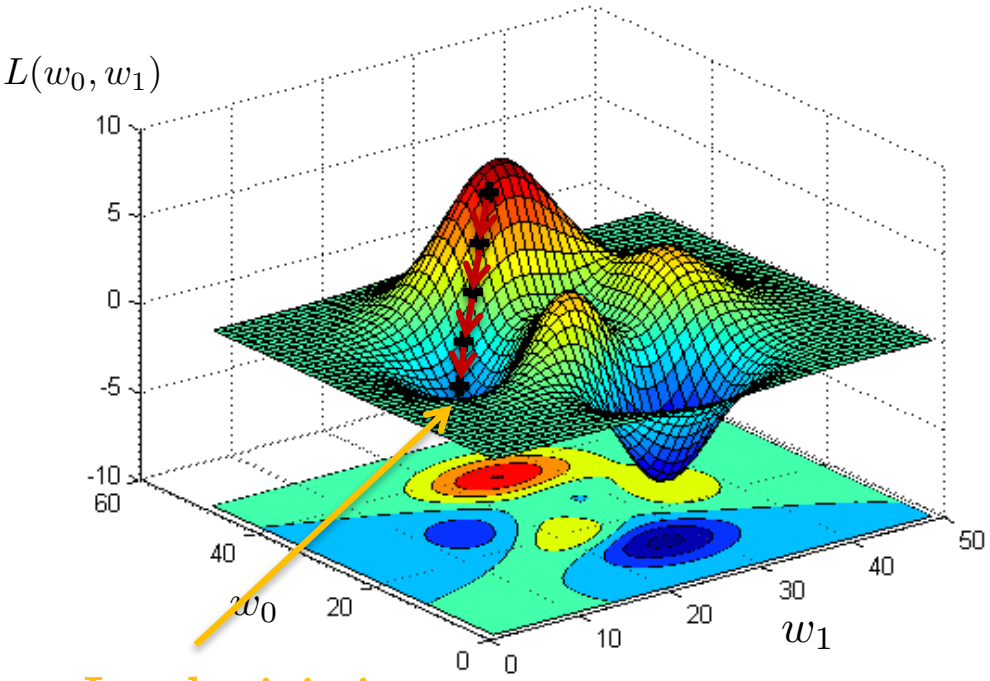


Initialization #2

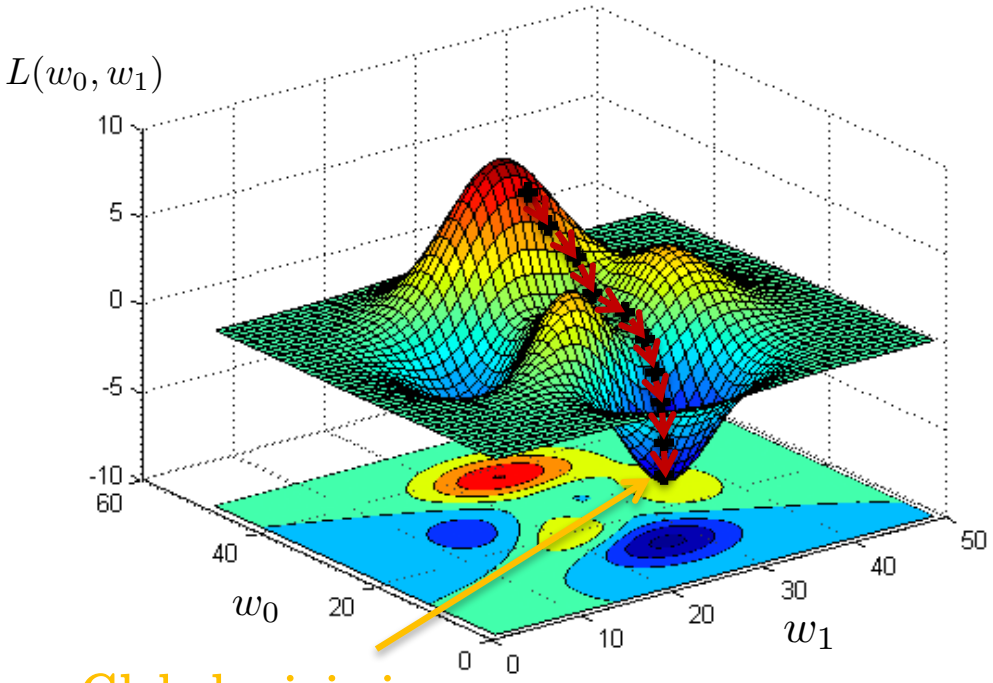
- Optimization scheme is faster with initialization #2 because the optimization starts closer to the solution.

Initialization

- Good initialization can be essential:
  - Start closer to the minimizer $\Rightarrow$ Quicker convergence.
  - If bad choice then gradient descent captures local minimizer (not as good as global minimizer).



Local minimizer

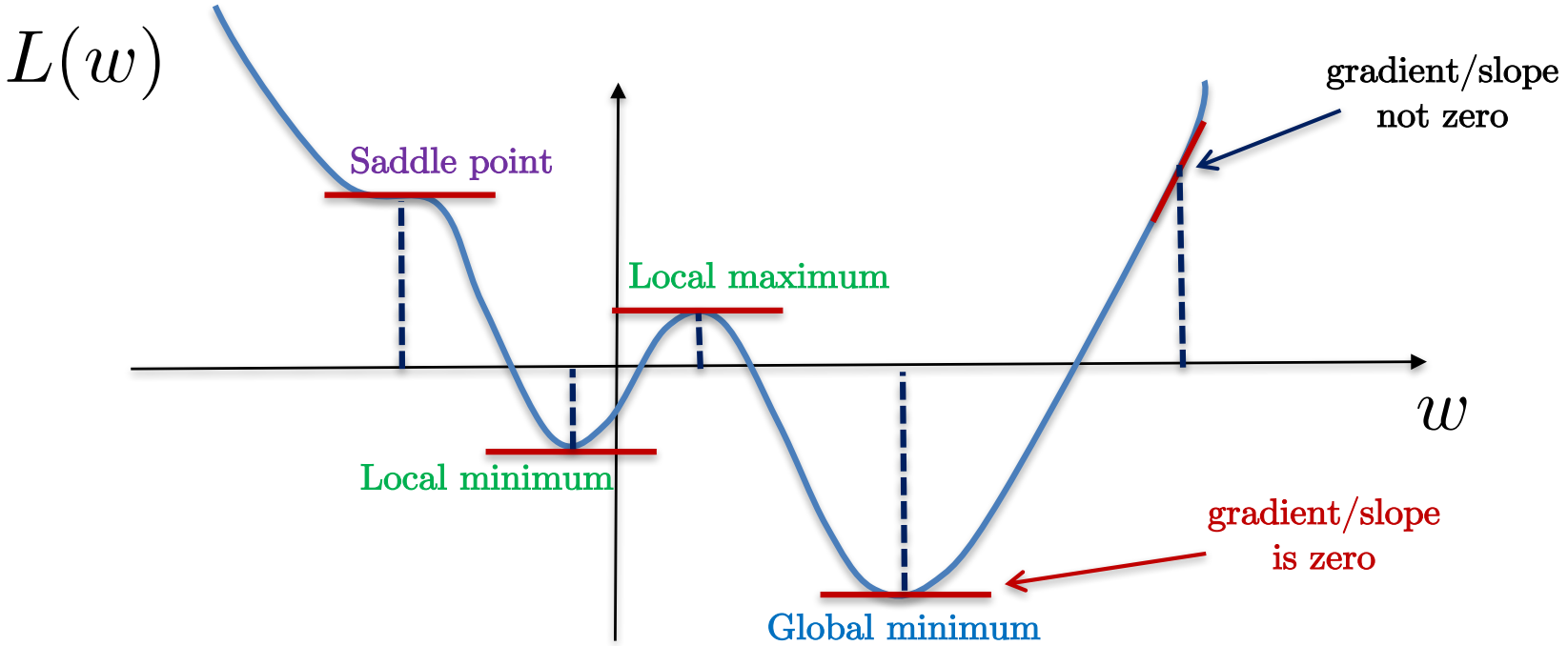


Global minimizer

$$\min_{w=(w_0,w_1)} L(w_0, w_1)$$

Minimizers

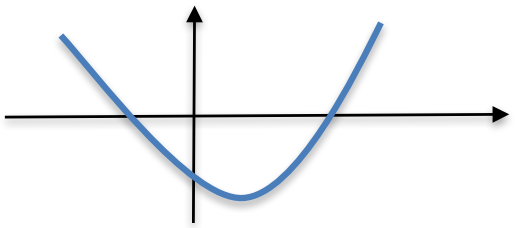
- Stationary/critical points: Points where the gradient/slope of the function is zero.
- Characterization of stationary points:
  - Global minimizers (maximizers)
  - Local minimizers (maximizers)
  - Saddle points



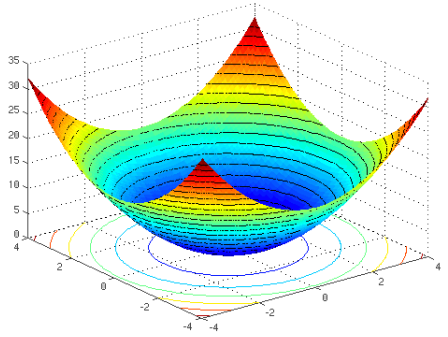
$$\min_{w=(w_0,w_1)} L(w_0, w_1)$$

Convexity vs non-convexity

- Convex losses are mathematically well studied: Fast optimization techniques (Newton, Nesterov), well understood behaviors and properties (existence of global minimum).

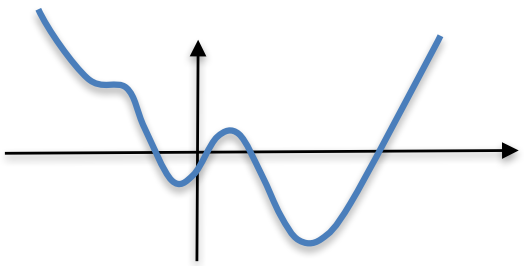


1D convex function

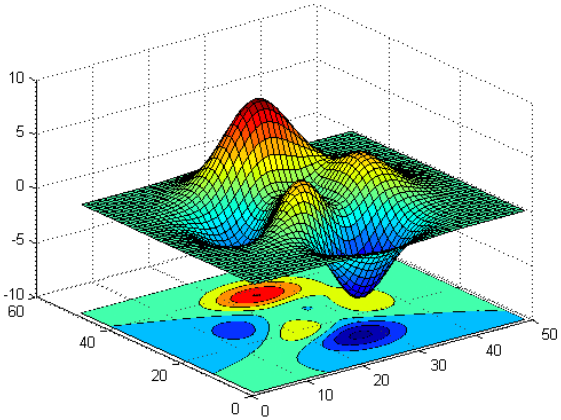


2D convex function

- Non-convex losses are in general not well understood, are slow to optimize (gradient descent), have critical points, but they offer large learning capacity.



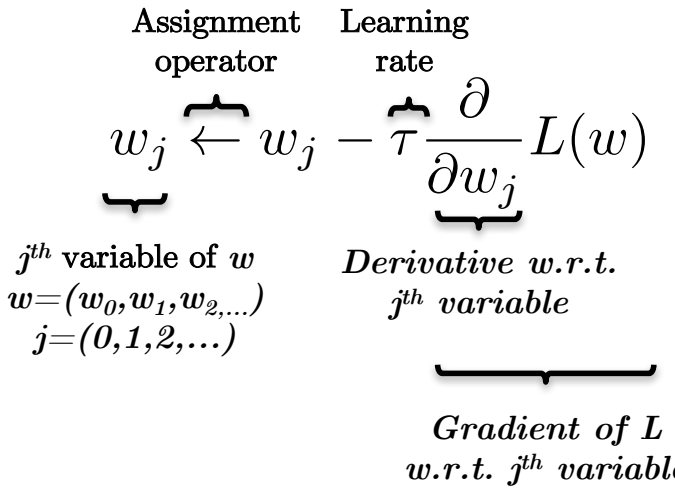
1D non-convex function



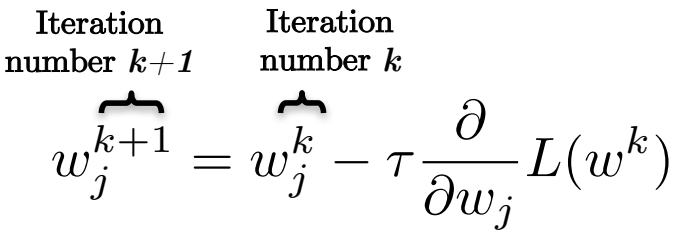
2D non-convex function

Formalization

- **Repeat until convergence** (until reaching a stationary point):



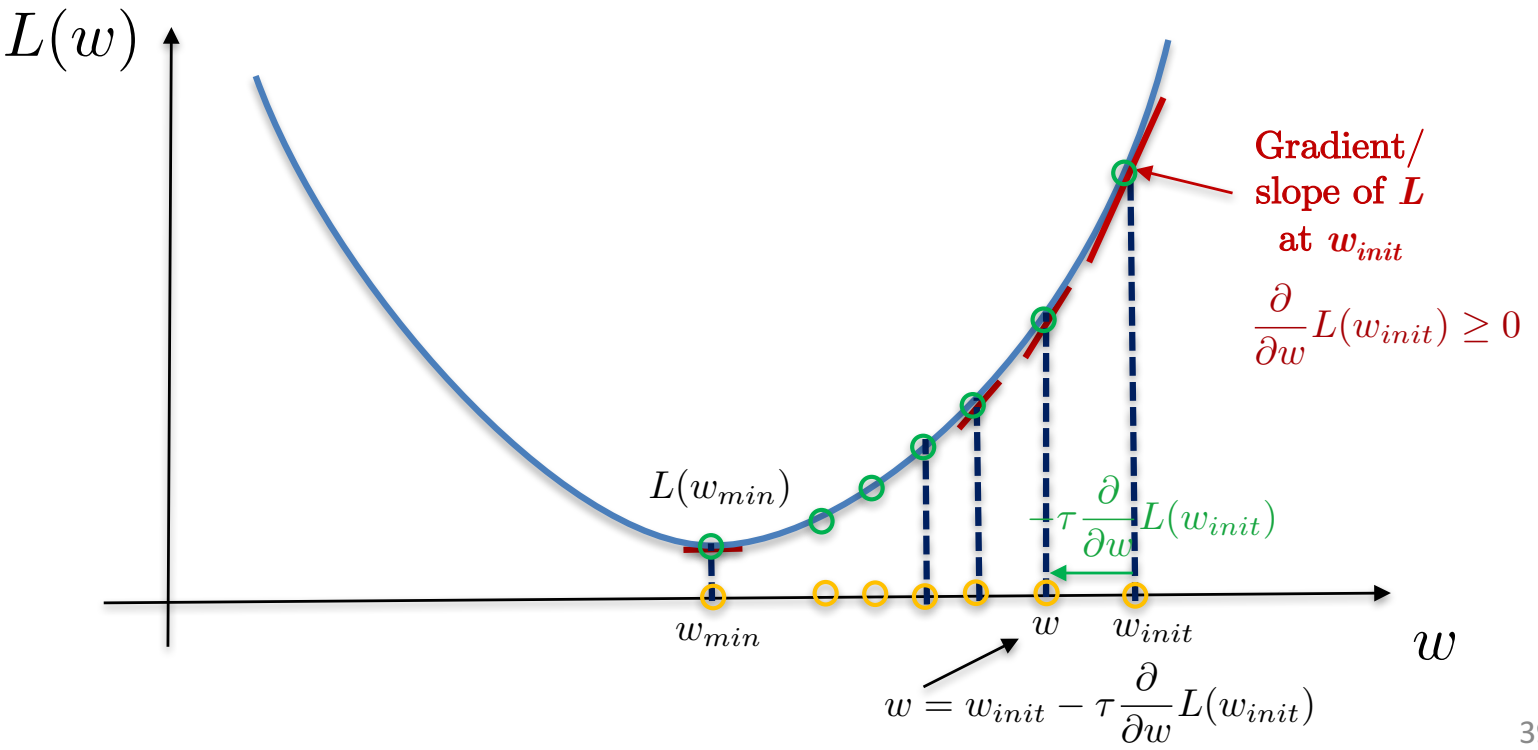
Computer notation



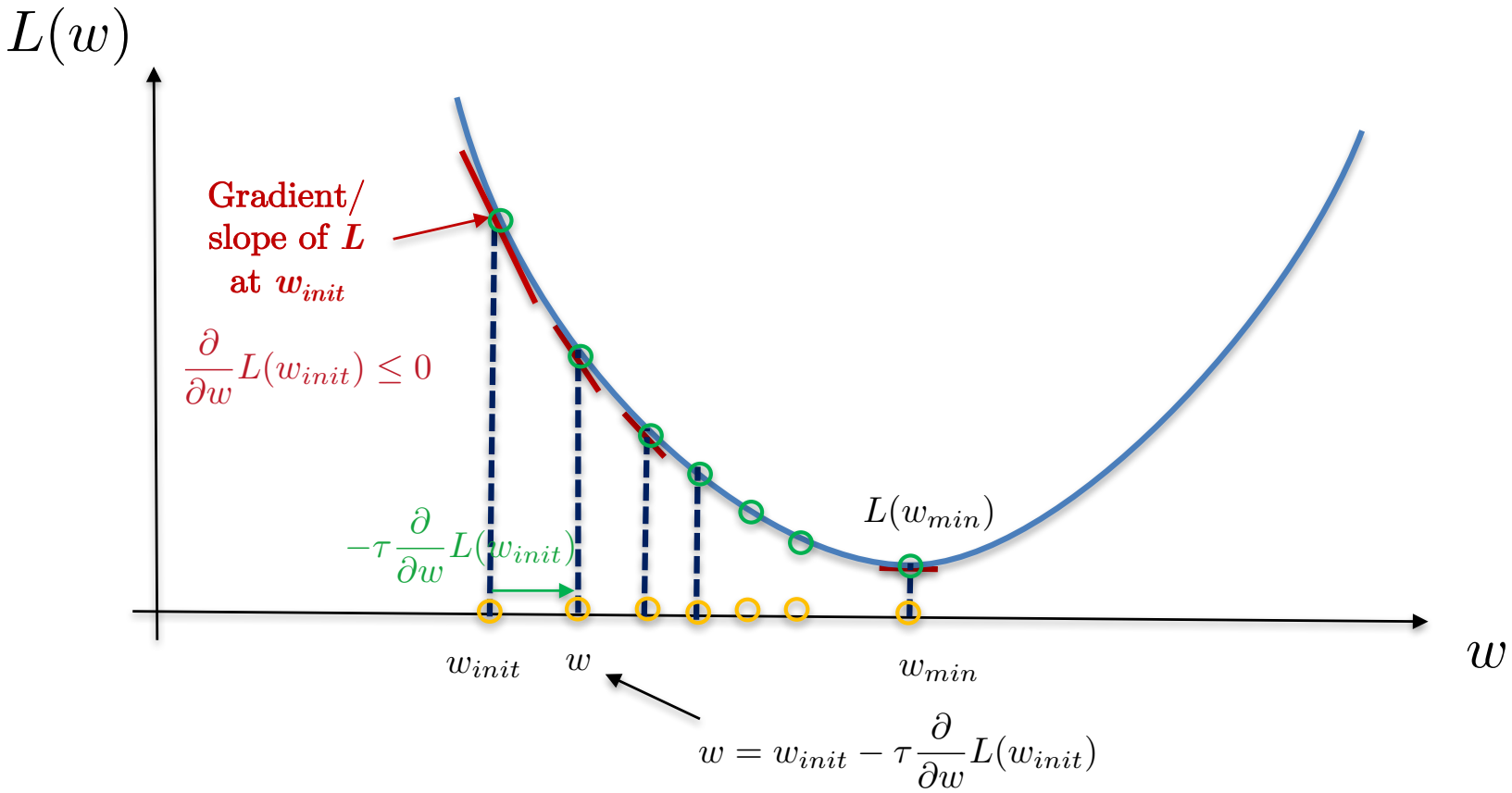
Math notation

Gradient descent with one variable

- Let us consider a single parameter w:
  - **Prediction function:** $f_w(x) = wx$
  - **Loss optimization:** $\min_w \; L(w) = \frac{1}{n} \sum_{i=1}^{n} \left(wx_i - y_i \right)^2$
  - **Gradient descent:** $w \leftarrow w - \tau \frac{\partial}{\partial w} L(w)$



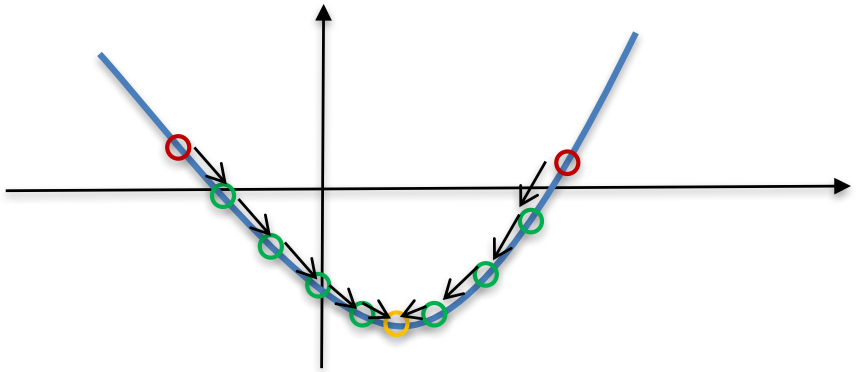
Different initialization



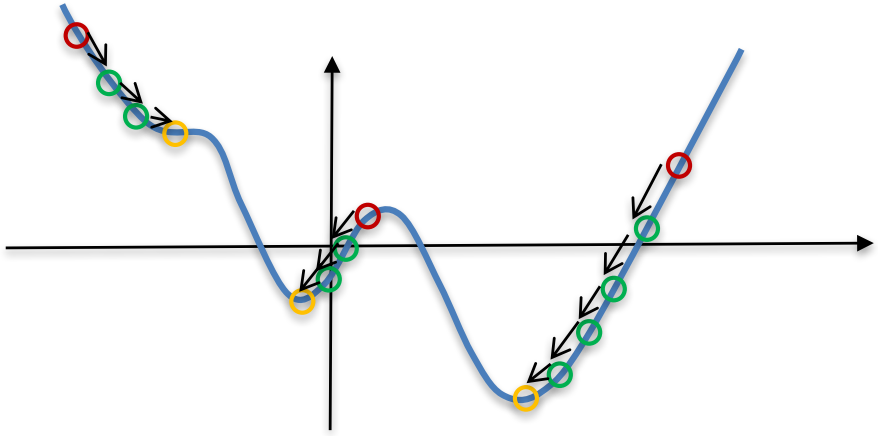
- Any initialization of the gradient descent does converge to the solution of the optimization problem. Only the optimization time is affected. Why? Loss is convex.

Convexity vs non-convexity with GD

- Gradient descent techniques with convex function: Only global minimizers, GD will converge for all initializations.



- Gradient descent techniques with non-convex function: Critical points $\Rightarrow$ Choice of initialization is critical for global minimizers, local minimizer, saddle points.

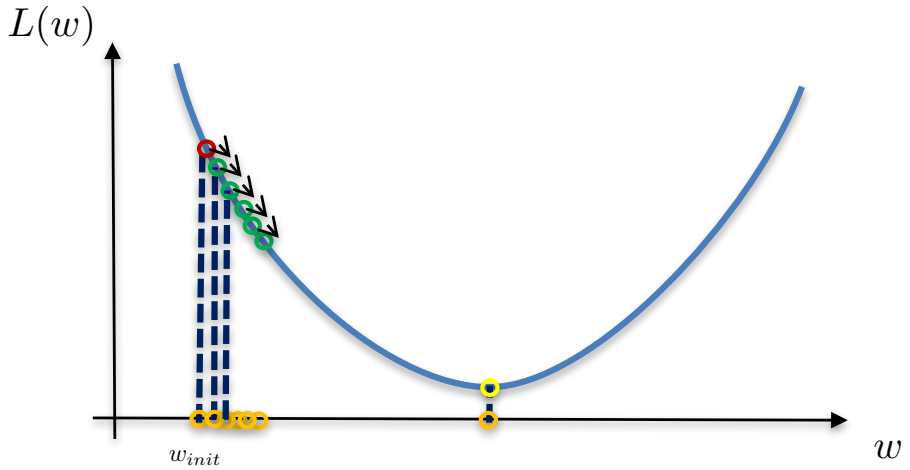


Learning rate

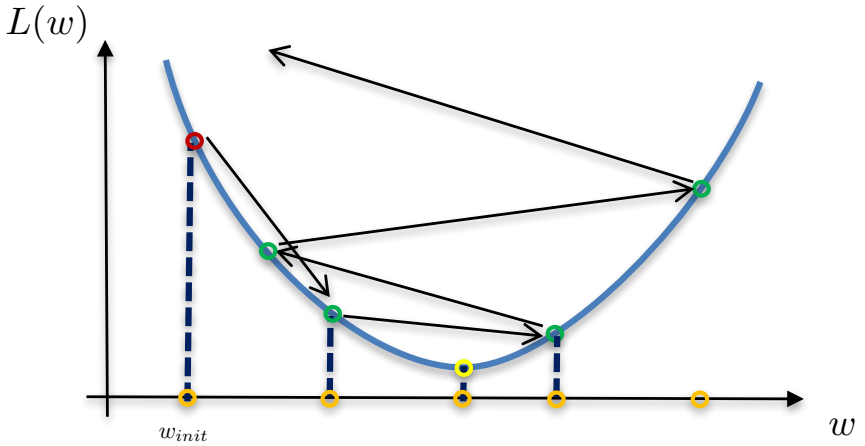
- Influence of learning rate/ time step in gradient descent:

$$w \leftarrow w - \tau \frac{\partial}{\partial w} L(w)$$

- If τ is too small, then the optimization process requires a lot of iterations to converge to the solution.

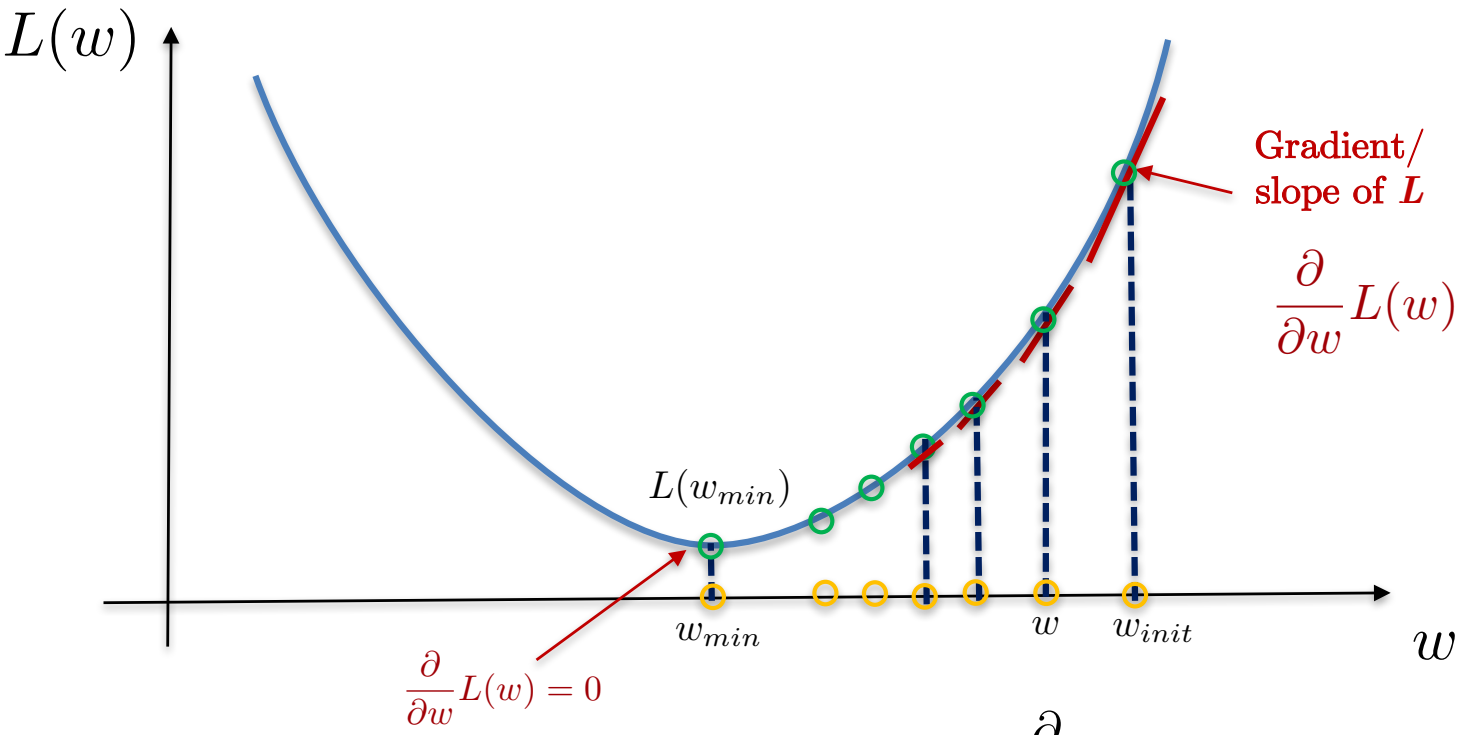


- If τ is too large, then the optimization process does not converge (diverges) and cannot capture the minimum (loss value explodes).



Convergence speed

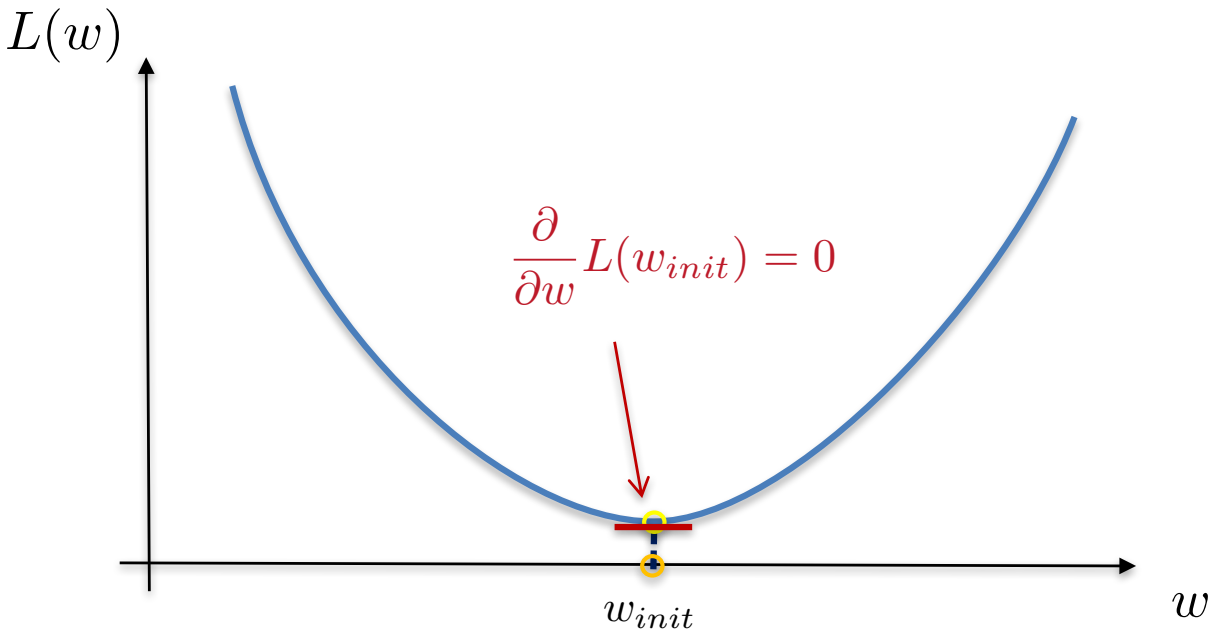
- Do we need to decrease the learning rate τ to guarantee convergence?
  No. The slope/gradient decreases its value when we get closer to the minimum.



$$w \leftarrow w - \tau \frac{\partial}{\partial w} L(w)$$

Quiz

- If you start at the solution, i.e. the minimum, what one gradient step will do? Nothing.



$$w \leftarrow w_{init} - \tau \frac{\partial}{\partial w} L(w_{init})$$

$$w \leftarrow w_{init}$$

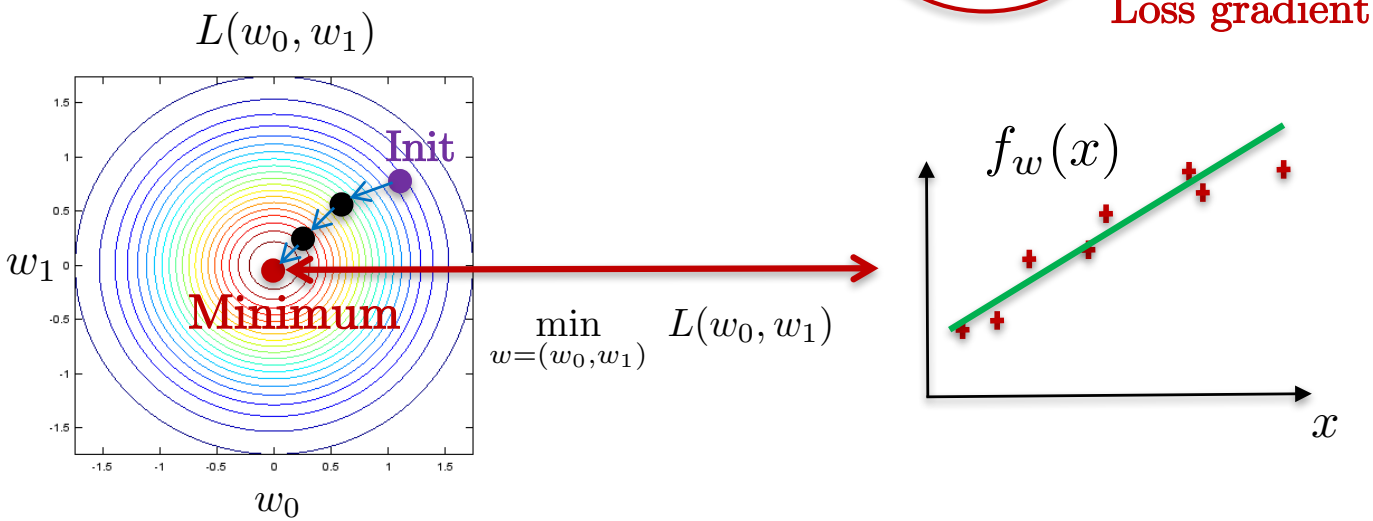
Outline

- Regression task
  - Supervised learning
  - Linear regression
  - Loss function
- **Optimization with gradient descent**
  - Generic case
  - **Linear regression case**
- Regression with multiple variables
- Non-linear regression
- Optimization with normal equations
- Conclusion

Gradient descent for linear regression

- **Prediction function:** $f_w(x) = w_0 + w_1 x$
- **Parameters:** w_0, w_1
- **Loss function:** $L(w_0, w_1) = \frac{1}{n} \sum_{i=1}^{n} \left(w_0 + w_1 x_i - y_i \right)^2$
- **Optimization:** $\min_{w=(w_0,w_1)} L(w_0, w_1)$
- **Gradient descent:** $w_j \leftarrow w_j - \tau \frac{\partial}{\partial w_j} L(w)$

Loss gradient



Loss gradient

- For any w_j:

$$\frac{\partial}{\partial w_j} L(w_0, w_1) = \frac{\partial}{\partial w_j} \left[\frac{1}{n} \sum_{i=1}^{n} \left(w_0 + w_1 x_i - y_i \right)^2 \right]$$

- For w_0:

$$\begin{aligned}
\frac{\partial}{\partial w_0} L(w_0, w_1) &= \frac{\partial}{\partial w_0} \left[\frac{1}{n} \sum_{i=1}^{n} \left(w_0 + w_1 x_i - y_i \right)^2 \right] \\
&= \frac{1}{n} \sum_{i=1}^{n} \frac{\partial}{\partial w_0} \left(w_0 + w_1 x_i - y_i \right)^2 \\
&= \frac{2}{n} \sum_{i=1}^{n} (w_0 + w_1 x_i - y_i) . \frac{\partial}{\partial w_0} (w_0 + w_1 x_i - y_i) \\
&= \frac{2}{n} \sum_{i=1}^{n} (w_0 + w_1 x_i - y_i) . 1
\end{aligned}$$

Loss gradient

- For w_1:

$$\frac{\partial}{\partial w_1} L(w_0, w_1) = \frac{\partial}{\partial w_1} \left[\frac{1}{n} \sum_{i=1}^{n} \left(w_0 + w_1 x_i - y_i \right)^2 \right]$$

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

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

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

Gradient descent equation

- **Initialize:**

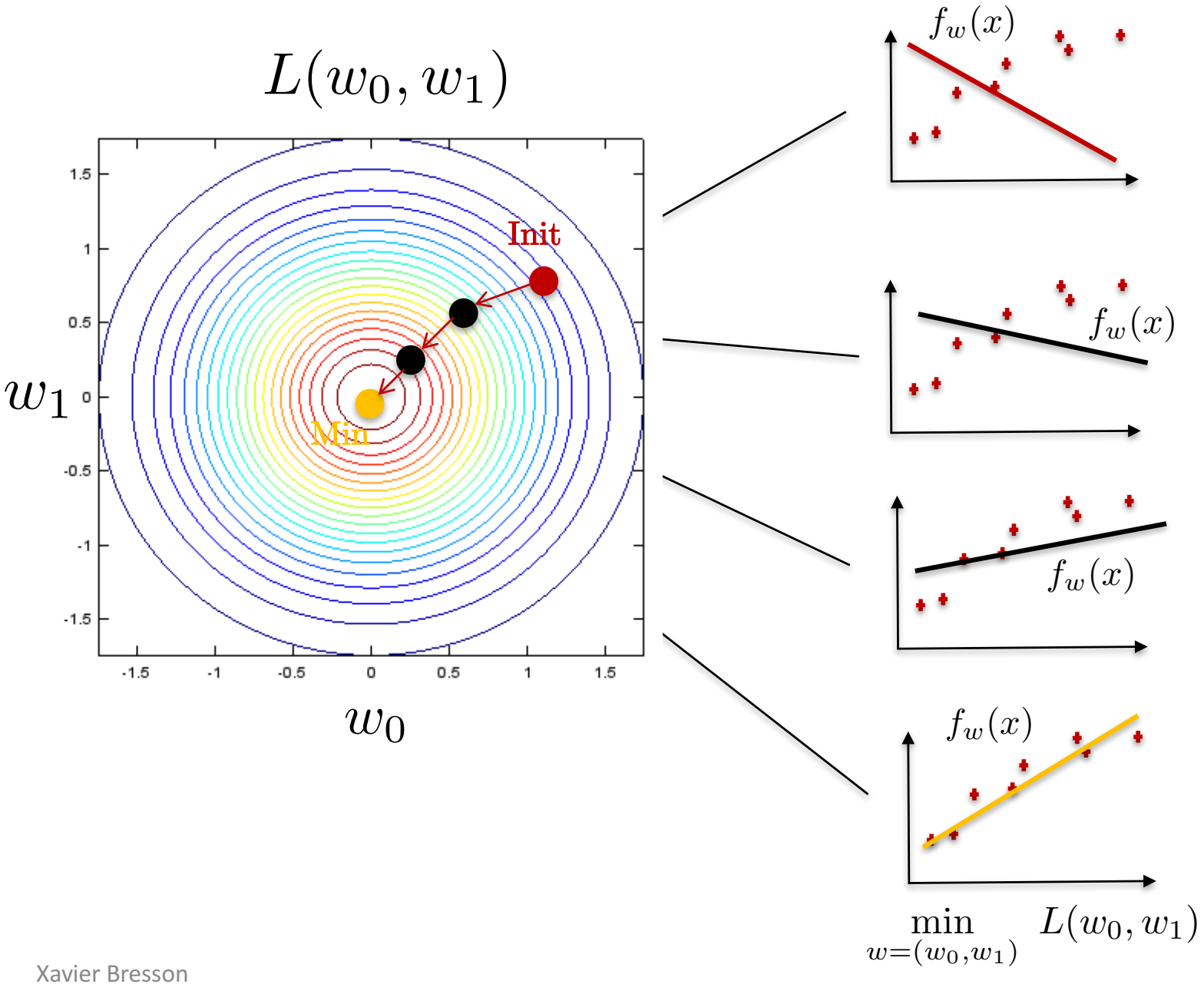
$$w_0, w_1$$

- **Iterate until convergence:**

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

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

Gradient descent in action



Stochastic vs deterministic GD

- Training set size is n:

$$w_j \leftarrow w_j - \tau \frac{\partial}{\partial w_j} L(w)$$

$$\text{with } L(w) = \frac{1}{n} \sum_{i=1}^{n} \Big(w_0 + w_1 x_i - y_i \Big)^2$$

$$\Downarrow$$

$$w_j \leftarrow w_j - \tau \frac{1}{n} \sum_{i=1}^{n} \frac{\partial}{\partial w_j} \Big(w_0 + w_1 x_i - y_i \Big)^2$$

- If n is small a few K, but if n is as large as M or B?
  $\Rightarrow$ Stochastic gradient descent (neural networks)

Quiz

- What is the gradient descent technique for the mean absolute loss?

$$L(w_0, w_1) = \frac{1}{n} \sum_{i=1}^{n} \left| w_0 + w_1 x_i - y_i \right|$$

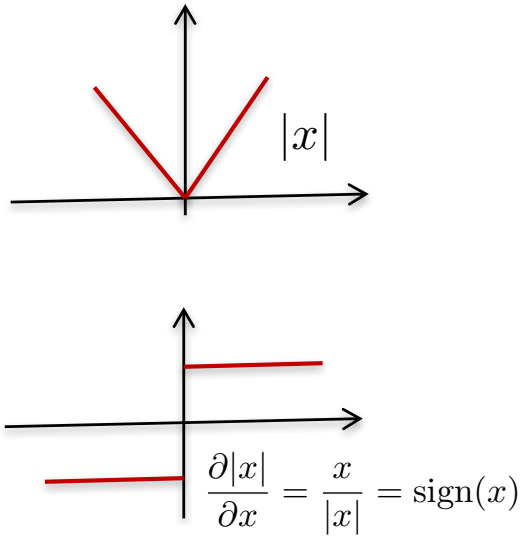
- For any w_j:

$$\frac{\partial}{\partial w_j} L(w_0, w_1) = \frac{\partial}{\partial w_j} \left[\frac{1}{n} \sum_{i=1}^{n} \left| w_0 + w_1 x_i - y_i \right| \right]$$

- Iterate until convergence:

$$w_0 \leftarrow w_0 - \tau \frac{1}{n} \sum_{i=1}^{n} \text{sign}(w_0 + w_1 x_i - y_i)$$

$$w_1 \leftarrow w_1 - \tau \frac{1}{n} \sum_{i=1}^{n} \text{sign}(w_0 + w_1 x_i - y_i) x_i$$



Outline

- Regression task
  - Supervised learning
  - Linear regression
  - Loss function
- Optimization with gradient descent
  - Generic case
  - Linear regression case
- **Regression with multiple variables**
- Non-linear regression
- Optimization with normal equations
- Conclusion

Multiple variables

- Data may have K even M features/attributes (curse of dimensionality).

d=1 variable

	Size x	Price y
n data	31	124
	54	156
	...	...

Prediction function:

$$f_w(x) = w_0 + w_1 x$$

d=3 variables

	Size $x_{(1)}$	#Rooms $x_{(2)}$	Age $x_{(3)}$	Price y
n data	31	2	6	124
	54	3	12	156
	...	...	...	...

Prediction function:

$$f_w(x_{(1)}, x_{(2)}, x_{(3)}) = w_(0) + w_1 x_{(1)} + w_2 x_{(2)} + w_3 x_{(3)}$$

Notation: $x_{(j)}$ = j^{th} variable of x data

Data matrix X

- Notation:
  - n = number of training data
  - d = number of variables/features, dim(x)
  - x_i = i^{th} training data, size(x_i) = d x 1
  - $x_{(j)}$ = j^{th} feature of training data x, scalar
  - X = data matrix, size(X)= n x d
  - X_{ij} = $x_{i(j)}$= j^{th} variable of i^{th} training data
- Example:

$$\underset{d \text{ x } 1}{x_1} = \begin{bmatrix} 2 \\ 1 \\ 7 \end{bmatrix}$$ d variables

$$\underset{1 \text{ x } d}{x_1^T} = [2 \quad 1 \quad 7]$$

d variables

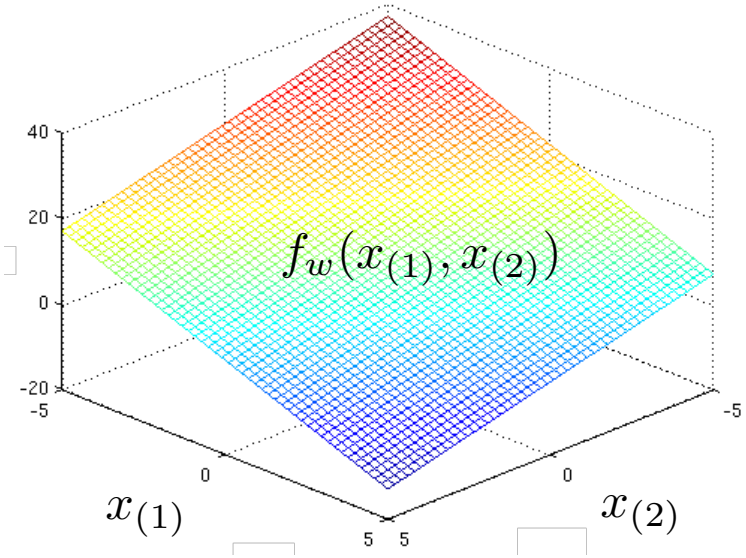
$$X = \begin{bmatrix} —x_1^T— \\ \vdots \\ —x_n^T— \end{bmatrix}$$ n data

Data matrix
n x d

All training data in a single matrix
(very efficient for linear algebra computations)

Prediction function with d variables

- ***d*=1 variable:** $f_w(x) = w_0 + w_1 x$
- ***d* variables:** $f_w(x_{(1)}, ..., x_{(d)}) = w_{(0)} + w_1 x_{(1)} + ... + w_d x_{(d)}$
- **Example: *d*=2** $f_w(x_{(1)}, x_{(2)}) = w_{(0)} + w_1 x_{(1)} + w_d x_{(2)}$



$$f_w(x_{(1)}, x_{(2)}) = -0.3 + 1.2x_{(1)} + 9.3x_{(2)}$$

Vector representation

- Define the vectors:

$$x = \begin{bmatrix} 1 \\ x_{(1)} \\ x_{(2)} \\ \vdots \\ x_{(d)} \end{bmatrix} \qquad w = \begin{bmatrix} w_0 \\ w_1 \\ w_2 \\ \vdots \\ w_d \end{bmatrix}$$

$(d+1)$ x 1 $(d+1)$ x 1

- Re-write the prediction function (as vector-vector multiplication):

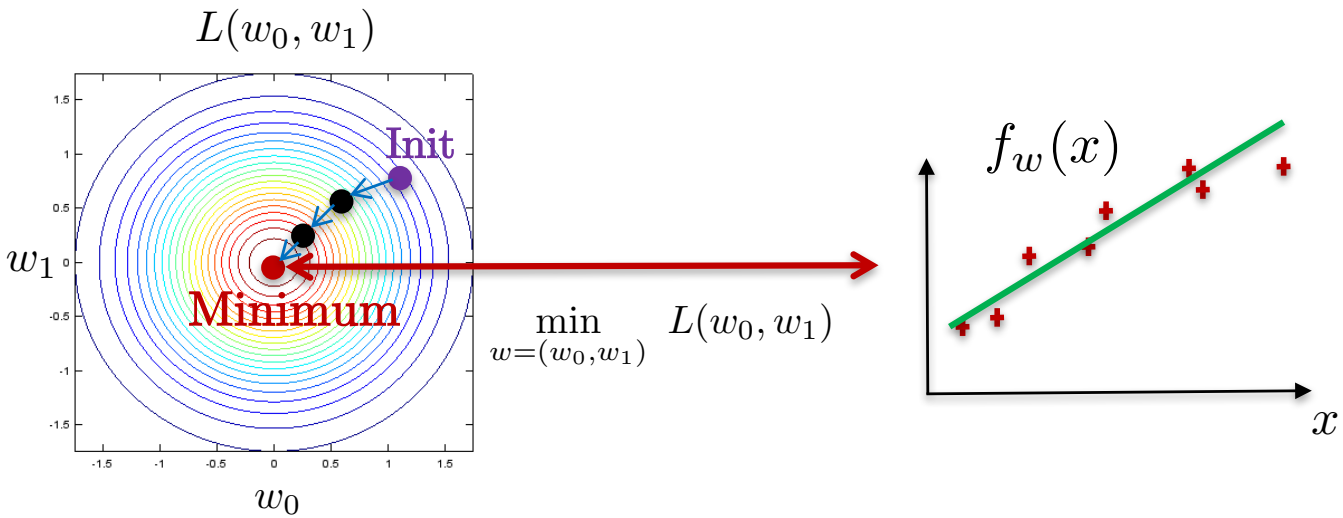
$$f_w(x) = w_0.1 + w_1 x_{(1)} + \ldots + w_d x_{(d)}$$
$$= w^T x \quad \text{One line of code}$$

1 x $(d+1)$ $(d+1)$ x 1

f is called multivariate linear regression function.

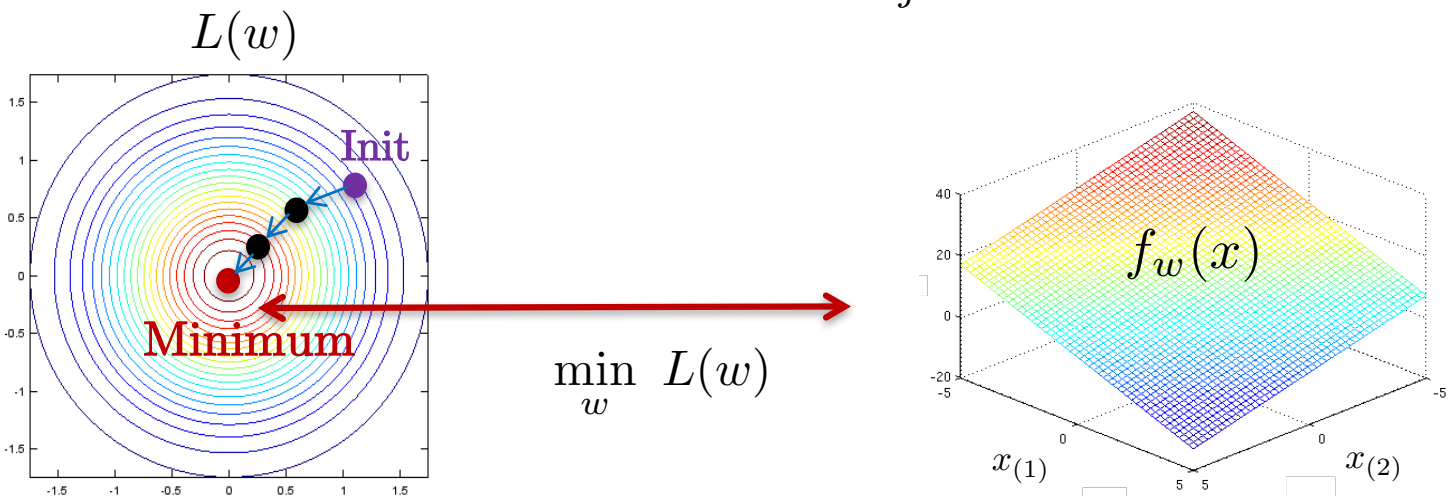
Gradient descent with d=1 variable

- **Prediction function:** $f_w(x) = w_0 + w_1 x$
- **Parameters:** w_0, w_1
- **Loss function:** $L(w_0, w_1) = \frac{1}{n} \sum_{i=1}^{n} \left(w_0 + w_1 x_i - y_i \right)^2$
- **Optimization:** $\min_{w=(w_0,w_1)} L(w_0, w_1)$
- **Gradient descent:** $w_j \leftarrow w_j - \tau \frac{\partial}{\partial w_j} L(w)$



Gradient descent with d variables

- **Prediction function:** $f_w(x) = w^T x = w_0 + w_1 x_{(1)} + ... + w_d x_{(d)}$
- **Parameters:** $w = [w_0, w_1, ..., w_d]$
- **Loss function:** $L(w) = \frac{1}{n} \sum_{i=1}^{n} \left(w^T x_i - y_i \right)^2$
- **Optimization:** $\min_w \; L(w)$
- **Gradient descent:** $w_j \leftarrow w_j - \tau \frac{\partial}{\partial w_j} L(w)$



Gradient descent equations

- ***d*=1 (one variable):**

$$\frac{\partial}{\partial w_j} L(w_0, w_1) = \frac{\partial}{\partial w_j} \left[\frac{1}{n} \sum_{i=1}^{n} \left(w_0.1 + w_1 x_i - y_i \right)^2 \right]$$

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

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

- ***d* variables:**

$$\frac{\partial}{\partial w_j} L(w) = \frac{\partial}{\partial w_j} \left[\frac{1}{n} \sum_{i=1}^{n} \left(w_0 + w_1 x_{i(1)} + ... + w_d x_{i(d)} - y_i \right)^2 \right]$$

Gradient descent equations

- **Gradient:**

$$\frac{\partial}{\partial w_j} L(w) = \frac{\partial}{\partial w_j} \Big[\frac{1}{n} \sum_{i=1}^{n} \Big(w_0 + w_1 x_{i(1)} + \ldots + w_d x_{i(d)} - y_i \Big)^2 \Big]$$

$$= \frac{1}{n} \sum_{i=1}^{n} \frac{\partial}{\partial w_j} \Big[\Big(w_0 + w_1 x_{i(1)} + \ldots + w_d x_{i(d)} - y_i \Big)^2 \Big]$$

$$= \frac{2}{n} \sum_{i=1}^{n} \Big(w_0 + w_1 x_{i(1)} + \ldots + w_d x_{i(d)} - y_i \Big).$$

$$\frac{\partial}{\partial w_j} \Big(w_0 + w_1 x_{i(1)} + \ldots + w_d x_{i(d)} - y_i \Big)$$

$$= \frac{2}{n} \sum_{i=1}^{n} \Big(w_0 + w_1 x_{i(1)} + \ldots + w_d x_{i(d)} - y_i \Big) x_{i(j)}$$

- **Gradient descent:**

$$w_j \leftarrow w_j - \tau \frac{2}{n} \sum_{i=1}^{n} \Big(w_0 + w_1 x_{i(1)} + \ldots + w_d x_{i(d)} - y_i \Big) x_{i(j)}$$

Data matrix X_{ij}

Matrix-vector representation

- **Vectorize gradient descent scheme:**

$$w_j \leftarrow w_j - \tau \frac{2}{n} \sum_{i=1}^{n} (w_0 + w_1 x_{i(1)} + \ldots + w_d x_{i(d)} - y_i).x_{i(j)}$$

$$w_j \leftarrow w_j - \tau \frac{2}{n} \sum_{i=1}^{n} X_{ij}.(x_i^T w - y_i)$$

$$w_j \leftarrow w_j - \tau \frac{2}{n} X_j^T (Xw - y)$$

$$w \leftarrow w - \tau \frac{2}{n} X^T (Xw - y)$$

$(d+1)$ x 1 — $(d+1)$ x n — n x 1

1 line of code

$$x_i = \begin{bmatrix} 1 \\ x_{i(1)} \\ \vdots \\ x_{i(d)} \end{bmatrix} \quad (d+1) \text{ x } 1$$

$$w = \begin{bmatrix} w_0 \\ w_1 \\ w_2 \\ \vdots \\ w_d \end{bmatrix} \quad (d+1) \text{ x } 1$$

for w_0

$$X = \begin{bmatrix} 1 & -x_1^T- \\ & \vdots \\ 1 & -x_n^T- \end{bmatrix} \quad n \text{ x } (d+1)$$

Data matrix

$$X_j = \begin{bmatrix} x_{1(j)} \\ \vdots \\ x_{n(j)} \end{bmatrix} \quad n \text{ x } 1$$

$$y = \begin{bmatrix} y_1 \\ \vdots \\ y_n \end{bmatrix} \quad n \text{ x } 1$$

Quiz

- How much speed up can we gain between a vectorized and un-vectorized algorithm?

$$w_j \leftarrow w_j - \tau \frac{2}{n} \sum_{i=1}^{n} (w_0 + w_1 x_{i(1)} + ... + w_d x_{i(d)} - y_i).x_{i(j)}$$

$$w \leftarrow w - \tau \frac{2}{n} X^T (Xw - y)$$

$\Rightarrow$ Up to 2-3 orders of magnitudes faster!

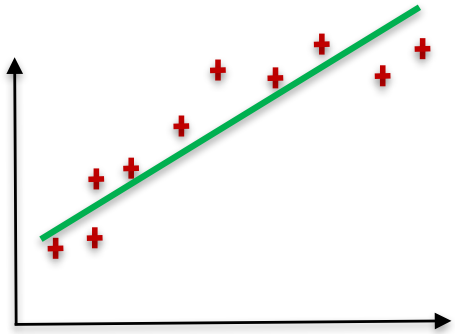
Existing computational components (CPU, multi-core CPU, GPU) are built for fast linear algebra computations.

Outline

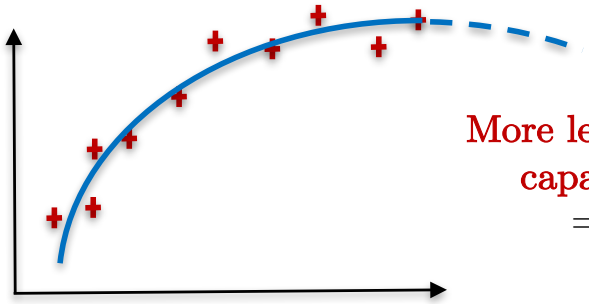
- Regression task
  - Supervised learning
  - Linear regression
  - Loss function
- Optimization with gradient descent
  - Generic case
  - Linear regression case
- Regression with multiple variables
- **Non-linear regression**
- Optimization with normal equations
- Conclusion

Beyond linear regression

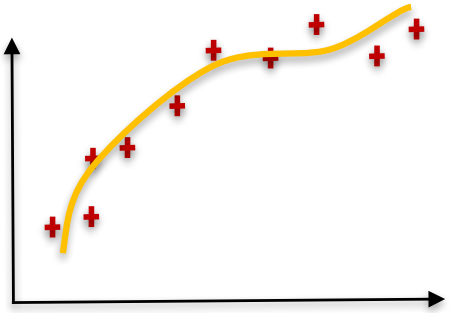
- **Example:** Housing prices prediction function



More learning capacity ⇒



More learning capacity ⇒



Linear prediction

$f_w(x) = w_0 + w_1 x$

Quadratic prediction

$f_w(x) = w_0 + w_1 x + w_2 x^2$

Polynomial prediction

$f_w(x) = w_0 + w_1 x + \ldots$
$w_2 x^2 + \ldots + w_6 x^6$

- **Handcrafted prediction function:** Domain expertise allows to define better families of predictive regression functions. Example:

$$f_w(x) = w_0 + w_1 \sqrt{x} + w_2 e^{-x}$$

- **Best non-linear regression technique:** Neural networks.

Outline

- Regression task
  - Supervised learning
  - Linear regression
  - Loss function
- Optimization with gradient descent
  - Generic case
  - Linear regression case
- Regression with multiple variables
- Non-linear regression
- **Optimization with normal equations**
- Conclusion

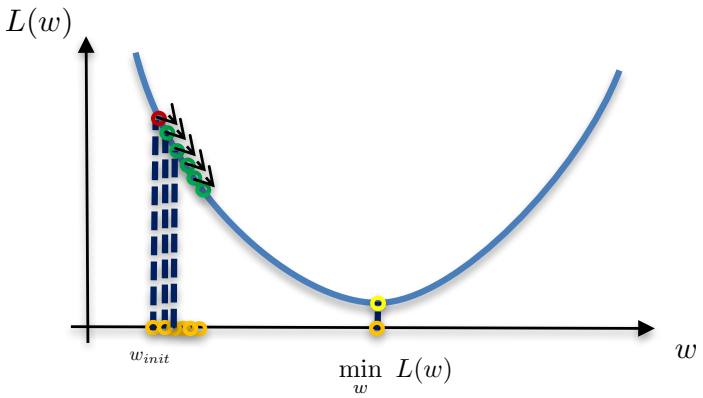
Beyond gradient descent

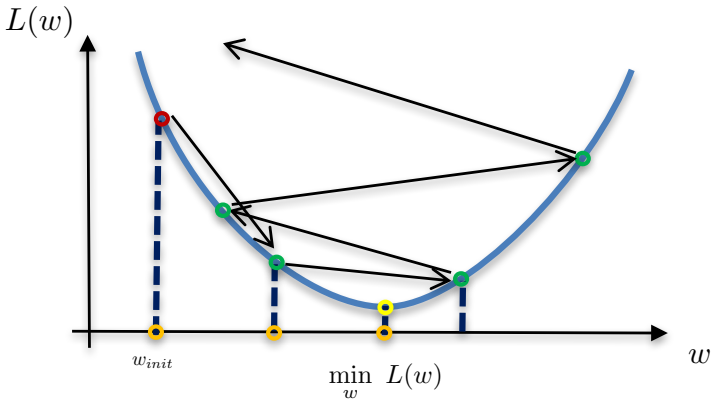
- Gradient descent is the most generic optimization technique.

$$\min_w \; L(w)$$

$$w_j \leftarrow w_j - \tau \frac{\partial}{\partial w_j} L(w)$$

- But it has limitations:
  - Choice of learning rate τ
  - Convergence speed (even with optimal τ)

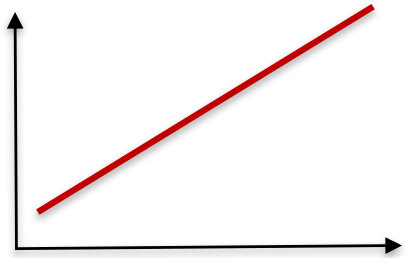




Beyond gradient descent

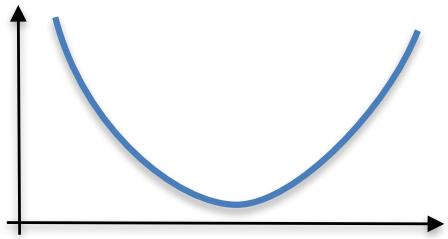
- We can leverage some mathematical properties to speed up the optimization.
  - Prediction function is linear:

$$f_w(x) = w^T x = w_0 + w_1 x_{(1)} + ... + w_d x_{(d)}$$



  - Loss function is convex (quadratic):

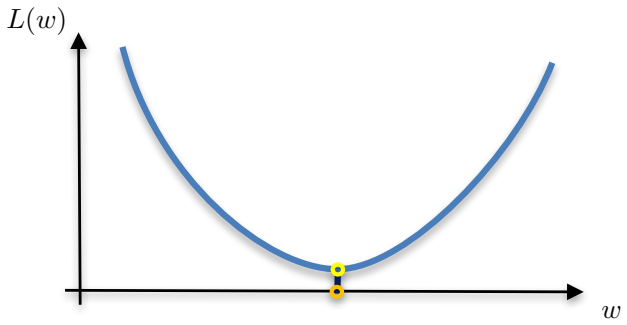
$$L(w) = \frac{1}{n} \sum_{i=1}^{n} \left(f_w(x_i) - y_i \right)^2$$



Normal equation for d=1 and n=1

- Normal equation: Solution of mean square error loss

$$\min_w \left\{ L(w) = (wx - y)^2 \right\}$$



- The minimum is obtained when the gradient/slope is zero:

$$\frac{\partial}{\partial w} L(w) = 0$$

$$\frac{\partial}{\partial w}(wx - y)^2 = 2x(wx - y) = 0 \Rightarrow w = x^{-1}y$$

Solution

One line of code

Normal equation for d=1 and n data

- Loss L:

$$L(w) = \frac{1}{n}\sum_{i=1}^{n}\left(wx_i - y_i\right)^2$$

- Gradient of the loss L w.r.t. w:

$$\frac{\partial}{\partial w}L(w) = 0 \Rightarrow \min_w L(w)$$

$$\begin{aligned}
\frac{\partial}{\partial w}\left[\frac{1}{n}\sum_{i=1}^{n}\left(wx_i - y_i\right)^2\right] &= \frac{1}{n}\sum_{i=1}^{n}\frac{\partial}{\partial w}\left[\left(wx_i - y_i\right)^2\right] \\
&= \frac{2}{n}\sum_{i=1}^{n}(wx_i - y_i)\frac{\partial}{\partial w}(wx_i - y_i) \\
&= \frac{2}{n}\sum_{i=1}^{n}(wx_i - y_i)x_i \\
&= \frac{2}{n}w\sum_{i=1}^{n}x_i^2 - \frac{2}{n}\sum_{i=1}^{n}y_i x_i \\
&= 0 \Rightarrow w = \frac{\sum_{i=1}^{n} y_i x_i}{\sum_{i=1}^{n} x_i^2}
\end{aligned}$$

Solution

Vectorization

- Loss L:

$$L(w) = \frac{1}{n}\sum_{i=1}^{n}\left(wx_i - y_i\right)^2$$

$$= \frac{1}{n}\underbrace{\left(wx-y\right)^T}_{1 \text{ x } n}\underbrace{\left(wx-y\right)}_{n \text{ x } 1}$$

$$x = \underset{n \text{ x } 1}{\begin{bmatrix} x_1 \\ \vdots \\ x_n \end{bmatrix}} \qquad y = \underset{n \text{ x } 1}{\begin{bmatrix} y_1 \\ \vdots \\ y_n \end{bmatrix}}$$

- Gradient of the loss L w.r.t. w:

$$\frac{\partial}{\partial w}L(w) = 0 \Rightarrow \min_w L(w)$$

$$\frac{\partial}{\partial w}\left[\frac{1}{n}\left(wx-y\right)^T\left(wx-y\right)\right] = \frac{1}{n}\frac{\partial}{\partial w}\left[\left(wx^T-y^T\right)\left(wx-y\right)\right]$$

$$= \frac{2}{n}x^T\left(wx-y\right)$$

$$= \frac{2}{n}\left(wx^Tx-x^Ty\right)$$

$$= 0 \Rightarrow w = \left(x^Tx\right)^{-1}x^Ty$$

One line of code

Normal equation for d features and n data

- **Loss L:** $L(w_0, ..., w_d) = \frac{1}{n} \sum_{i=1}^{n} \left(w_0 + w_1 x_{i(1)} + ... + w_d x_{i(d)} - y_i \right)^2$

- **Gradient of the loss L w.r.t. w_j:**

$$\frac{\partial}{\partial w_j} L(w_0, ..., w_d) = 0 \quad \forall j$$

$$\frac{\partial}{\partial w_j} L(w) = \frac{\partial}{\partial w_j} \left[\frac{1}{n} \sum_{i=1}^{n} \left(w_0 + w_1 x_{i(1)} + ... + w_d x_{i(d)} - y_i \right)^2 \right]$$

$$= \frac{2}{n} \sum_{i=1}^{n} \left(w_0 + w_1 x_{i(1)} + ... + w_d x_{i(d)} - y_i \right) x_{i(j)}$$

$$= 0 \Rightarrow w_j = \frac{\sum_{k \neq j} \sum_i w_k x_{i(k)} x_{i(j)}}{\sum_i x_{i(j)}^2} \qquad \text{with} \quad x_{i(0)} = 1$$

Solution

Vectorization

- Loss ***L:***

$$L(w_0, ..., w_d) = \frac{1}{n} \sum_{i=1}^{n} \left(w_0 + w_1 x_{i(1)} + ... + w_d x_{i(d)} - y_i \right)^2$$

$$L(w) = \frac{1}{n} \sum_{i=1}^{n} \left(x_i^T w - y_i \right)^2$$

$$L(w) = \frac{1}{n} \left(Xw - y \right)^T \left(Xw - y \right)$$

$$\underset{(d+1)\text{ x }1}{w} = \begin{bmatrix} w_0 \\ \vdots \\ w_d \end{bmatrix} \qquad \underset{(d+1)\text{ x }1}{x_i} = \begin{bmatrix} x_{i(0)} \\ \vdots \\ x_{i(d)} \end{bmatrix} \qquad \underset{\text{n x (d+1)}}{X} = \begin{bmatrix} —x_1^T— \\ \vdots \\ —x_n^T— \end{bmatrix} \qquad \underset{\text{n x 1}}{y} = \begin{bmatrix} y_1 \\ \vdots \\ y_n \end{bmatrix}$$

with $x_{i(0)} = 1$

Data matrix

Vectorization

- Gradient of the loss L w.r.t. w_j:

$$\begin{aligned}
\frac{\partial}{\partial w} L(w) &= \frac{1}{n} \frac{\partial}{\partial w} \left[\left(Xw - y \right)^T \left(Xw - y \right) \right] \\
&= \frac{1}{n} \frac{\partial}{\partial w} \left[\left(w^T X^T - y^T \right) \left(Xw - y \right) \right] \\
&= \frac{2}{n} X^T \left(Xw - y \right) \\
&= \frac{2}{n} \left(X^T Xw - X^T y \right) \\
&= 0 \Rightarrow w = (X^T X)^{-1} X^T y
\end{aligned}$$

One line of code

Quiz

- Gradient descent vs normal equation?

  - Advantages:
    - Works well for large n (big data). Complexity is $O(n)$.
  - Limitations:
    - Select learning rate τ.
    - May requite lots of iterations.

  - Advantages:
    - Very fast for small n (solve linear system of equations).
    - No need to select learning rate τ.
  - Limitations:
    - Works well for small n ($n<10^4$). Complexity is $O(n^3)$.
      - n=10 < 1sec
      - n=100 O(sec)
      - n=1,000 O(min)
      - n=10,000 O(hour)

Outline

- Regression task
  - Supervised learning
  - Linear regression
  - Loss function
- Optimization with gradient descent
  - Generic case
  - Linear regression case
- Regression with multiple variables
- Non-linear regression
- Optimization with normal equations
- **Conclusion**

Conclusion

- Linear regression is the most common regression technique.
- Gradient descent technique is one approach to optimize the regression loss, but there exist better optimization techniques for small, medium-scale datasets (quadratic optimization (normal equations), convex optimization).
- Code vectorization benefits from existing architectures s.a. CPU, multi-core CPU, GPU (much faster than *for* loops).
- Any learning technique, including regression, can be scaled up to billions of data with stochastic gradient descent.
- Gradient descent technique (a.k.a. backpropagation) is the only available technique to train neural networks with large-scale datasets (it works very well but it is slow).

Introduction to Python

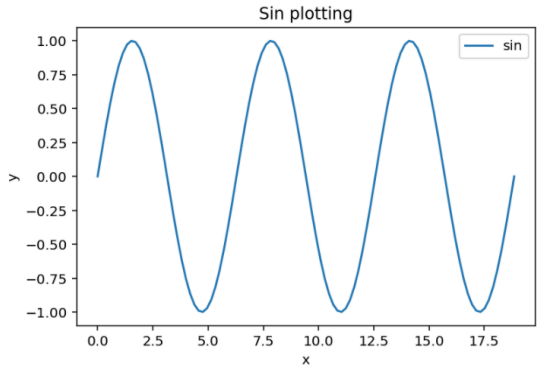
- tutorial01.ipynb

5. Plotting data

In [22]:
```
# Visualization library
%matplotlib inline
from IPython.display import set_matplotlib_formats
set_matplotlib_formats('png2x','pdf')
import matplotlib.pyplot as plt
```

In [23]:
```
x = np.linspace(0,6*np.pi,100)
#print(x)
y = np.sin(x)

plt.figure(1)
plt.plot(x, y,label='sin'.format(i=1))
plt.legend(loc='best')
plt.title('Sin plotting')
plt.xlabel('x')
plt.ylabel('y')
plt.show()
```



Linear algebra with Python

- tutorial02.ipynb

4. Matrix-matrix multiplication

4.1 Example

$$\underset{3\times 2}{\begin{bmatrix} 4.1 & 5.3 \\ -3.9 & 8.4 \\ 6.4 & -1.8 \end{bmatrix}} \times \underset{2\times 2}{\begin{bmatrix} 2.7 & 3.2 \\ 3.5 & -8.2 \end{bmatrix}} = \underset{3\times 2}{\begin{bmatrix} 4.1\times 2.7+5.3\times 3.5 & 4.1\times 3.2+5.3\times -8.2 \\ -3.9\times 2.7+8.4\times 3.5 & -3.9\times 3.2+8.4\times -8.2 \\ 6.4\times 2.7-1.8\times 3.5 & 6.4\times 3.2-1.8\times -8.2 \end{bmatrix}}$$

Dimension of the matrix-matrix multiplication operation is given by contraction of 3×2 with $2\times 2 = 3\times 2$.

4.2 Formalization

$$\underset{m\times n}{[A]} \times \underset{n\times p}{[X]} = \underset{m\times p}{[Y]}$$

Like for matrix-vector multiplication, matrix-matrix multiplication can be carried out only if A and X have the same n dimension.

4.3 Linear algebra operations can be parallelized/distributed

Column Y_i is given by multiplying matrix A with the i^{th} column of X:

$$\begin{array}{ccccc} Y_i & = & A & \times & X_i \\ 1\times 1 & = & 1\times n & \times & n\times 1 \end{array}$$

Observe that all columns X_i are independent. Consequently, all columns Y_i are also independent. This allows to vectorize/parallelize linear algebra operations on (multi-core) CPUs, GPUs, clouds, and consequently to solve all linear problems (including linear regression) very efficiently, basically with one single line of code ($Y = AX$ for millions/billions of data). With Moore's law (computers speed increases by 100x every decade), it has introduced a computational revolution in data analysis.

Question 4: Multiply the two matrices above in Python

In [4]:
```
import numpy as np

A = np.array([[4.1,5.3],[-3.9,8.4],[6.4,-1.8]]) #YOUR CODE HERE
X = np.array([[2.7,3.2],[3.5,-8.2]])
Y = A.dot(X)
print(A)
print(A.shape)
print(X)
print(X.shape)
print(Y)
print(Y.shape)
```

Coding exercise on supervised regression

- tutorial03.ipynb

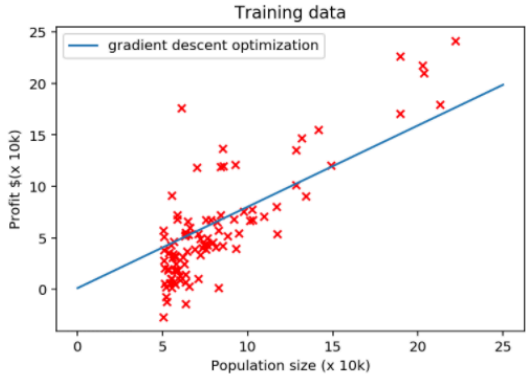
7. Plot the linear prediction function

$$f_w(x) = w_0 + w_1 x$$

Hint: You may use numpy function `linspace`.

In [8]:
```
# linear regression model
x_pred = np.linspace(0,25,100) #YOUR CODE HERE
y_pred = w[0] + w[1]* x_pred #YOUR CODE HERE

# plot
plt.figure(3)
plt.scatter(x_train, y_train, s=30, c='r', marker='x', linewidths=1)
plt.plot(x_pred, y_pred,label='gradient descent optimization'.format(i=1))
plt.legend(loc='best')
plt.title('Training data')
plt.xlabel('Population size (x 10k)')
plt.ylabel('Profit $(x 10k)')
plt.show()
```



Questions?