Regression and Gradient Descent

Prof. Alessandro Lucantonio

Aarhus University

24/10/2023

Weight-Height dataset

Task: build a model that predicts the height given the weight.

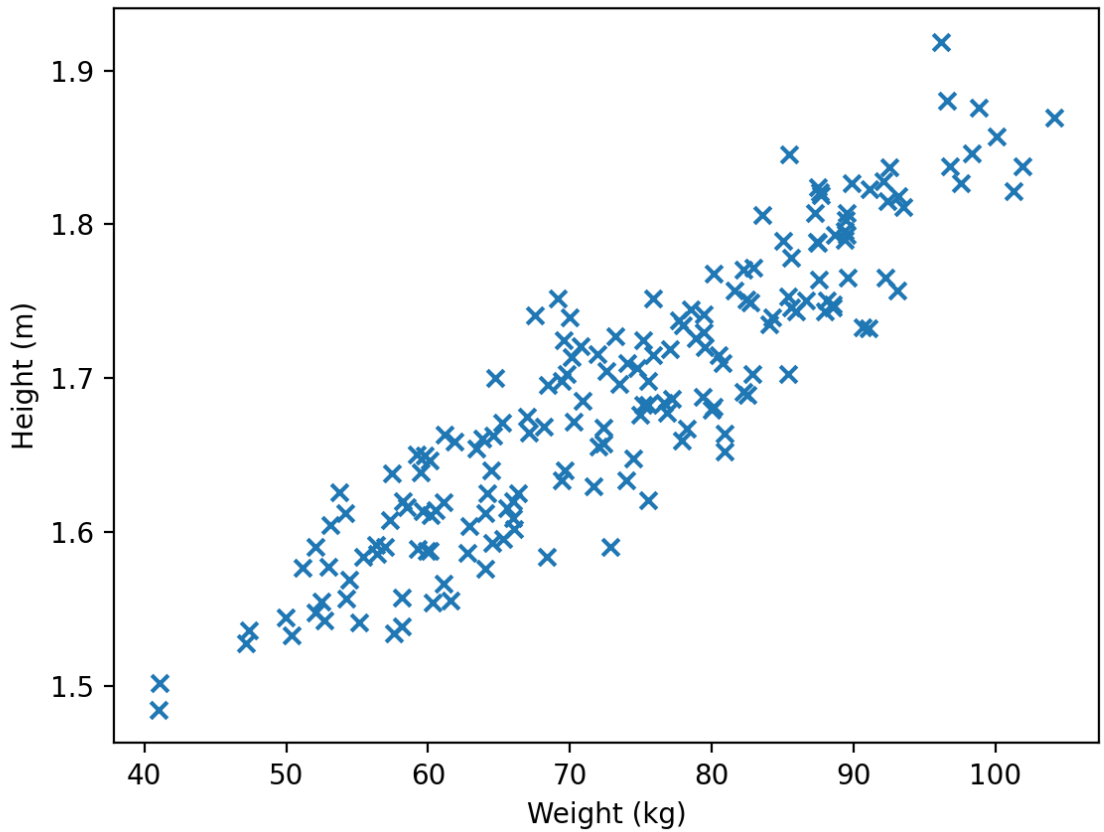


Figure: Plot of the dataset

A solution - Linear Regression model

Remarks:

- Regression problem (continuous output).
- Data with different orders of magnitude.

A possible solution to this problem is a linear model (red line in the figure below). This learning algorithm is called **Linear Regression** (LR).

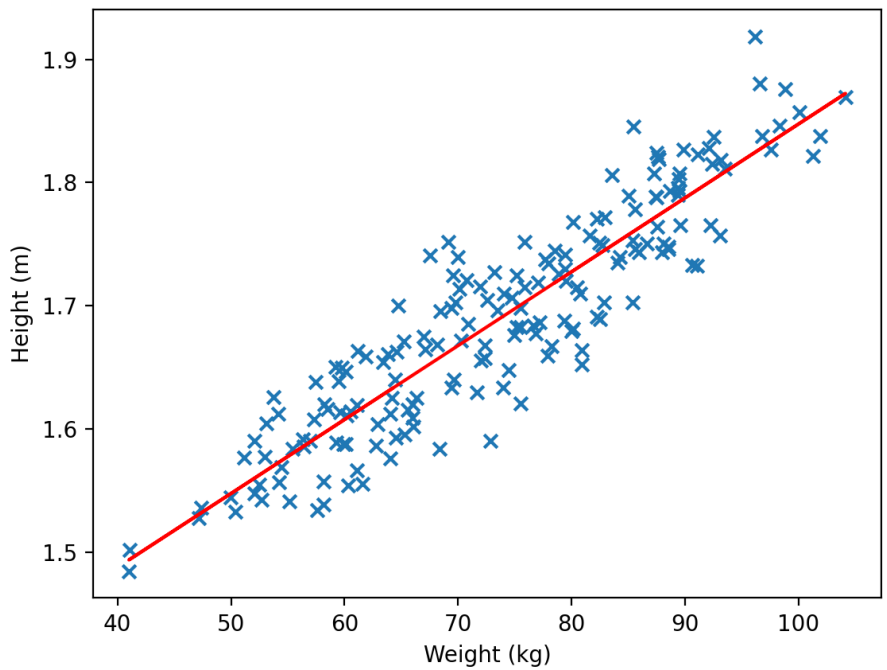


Figure: Linear regression model.

Linear Regression: main ingredients

Notation:

- $x^{(i)}$: a data sample (weight of the i-th person).
- $y^{(i)}$: the target corresponding to $x^{(i)}$ (height).
- N: number of samples.

Model/hypothesis:

$$h_{\boldsymbol{w}}(x^{(i)}) = w_1 x^{(i)} + w_0$$

where $\boldsymbol{w} = [w_0, w_1]^T$ is the vector of parameters to be learned.

The set $\mathcal{H} := \{h_{\boldsymbol{w}} | \boldsymbol{w} \in \mathbb{R}^2\}$ is called **hypothesis space**.

How to learn $\boldsymbol{w}$ from data?

Performance measure: Mean Squared Error (MSE)

To evaluate how good is the prediction we compute the **Mean Squared Error** (MSE) is:

$$E(\boldsymbol{w}) := \frac{1}{N} \sum_{i=1}^{N} (h_{\boldsymbol{w}}(x^{(i)}) - y^{(i)})^2.$$

To find the "best" set of parameters, we minimize the MSE:

$$\boldsymbol{w} \in \arg\min_{\tilde{\boldsymbol{w}} \in \mathbb{R}^2} E(\tilde{\boldsymbol{w}}).$$

Multiple-feature Linear Regression

- Dataset: $\boldsymbol{x}^{(i)} \in \mathbb{R}^n, y^{(i)} \in \mathbb{R}$, where $x_j^{(i)}$ is the j-th feature of the i-th sample and n is the number of features.
- Hypothesis:

$$\begin{aligned} h_{\boldsymbol{w}}(\boldsymbol{x}^{(i)}) &= w_n x_n^{(i)} + w_{n-1} x_{n-1}^{(i)} + \cdots + w_1 x_1^{(i)} + w_0 \\ &= \sum_{i=0}^{n} w_i \tilde{x}_i^{(i)} = \boldsymbol{w}^T \tilde{\boldsymbol{x}}^{(i)}, \end{aligned}$$

where $\boldsymbol{w} = [w_0, \ldots, w_n]^T$ and $\tilde{\boldsymbol{x}}^{(i)} = [1, x_1^{(i)}, \ldots, x_n^{(i)}]^T$.

Multiple-feature Linear Regression - MSE

MSE:

$$
\begin{aligned}
E(\boldsymbol{w}) &= \frac{1}{N}\sum_{i=1}^{N}(h_{\boldsymbol{w}}(\boldsymbol{x}^{(i)}) - y^{(i)})^2 \\
&= \frac{1}{N}(\mathrm{X}\boldsymbol{w} - \boldsymbol{y})^T(\mathrm{X}\boldsymbol{w} - \boldsymbol{y}) \\
&= \frac{1}{N}||\mathrm{X}\boldsymbol{w} - \boldsymbol{y}||^2
\end{aligned}
$$

where

$$
\mathrm{X} = \begin{bmatrix} \tilde{\boldsymbol{x}}^{(1)} \\ \vdots \\ \tilde{\boldsymbol{x}}^{(N)} \end{bmatrix} \quad \boldsymbol{y} = \begin{bmatrix} y^{(1)} \\ \vdots \\ y^{(N)} \end{bmatrix}.
$$

Notice that $\mathrm{X} \in \mathbb{R}^{N\times(n+1)}$ and $\boldsymbol{y} \in \mathbb{R}^N$.

Coefficient of determination

Idea: X and $\boldsymbol{y}$ can be thought as two random variables.

Let $\boldsymbol{\epsilon} := \boldsymbol{y} - \mathrm{X}\boldsymbol{w}$ the vector of the residuals and $\sigma_{\boldsymbol{y}} := \sqrt{\mathrm{Var}(\boldsymbol{y})}$ the standard deviation of the targets. The quantity

$$R^2 := 1 - \frac{||\boldsymbol{\epsilon}||^2}{\sigma_{\boldsymbol{y}}^2}$$

is called the *coefficient of determination*.

Best case scenario: $\sigma_{\boldsymbol{\epsilon}} = 0$, hence $R^2 = 1$.

Finding the minimum: Gradient Descent

How to find $\boldsymbol{w} \in \arg\min_{\tilde{\boldsymbol{w}} \in \mathbb{R}^2} E(\tilde{\boldsymbol{w}})$?

Main idea: geometrically, the gradient of a scalar function represents the direction of maximum slope. Hence, following the direction opposite to the gradient allows to decrease the value of the function, *i.e.*

$$E(\boldsymbol{w}^{j+1}) \leq E(\boldsymbol{w}^j)$$

Formally:

- Start with an initial guess $\boldsymbol{w}^0$.
- For $j \geq 0$, update $\boldsymbol{w}^{j+1} := \boldsymbol{w}^j + \boldsymbol{d}^j$, where $\boldsymbol{d}^j$ is such that

$$\boldsymbol{d}^j = -\alpha \nabla E(\boldsymbol{w}^j)$$

where $\alpha > 0$ is the **learning rate**.

Gradient Descent - 3D visualization

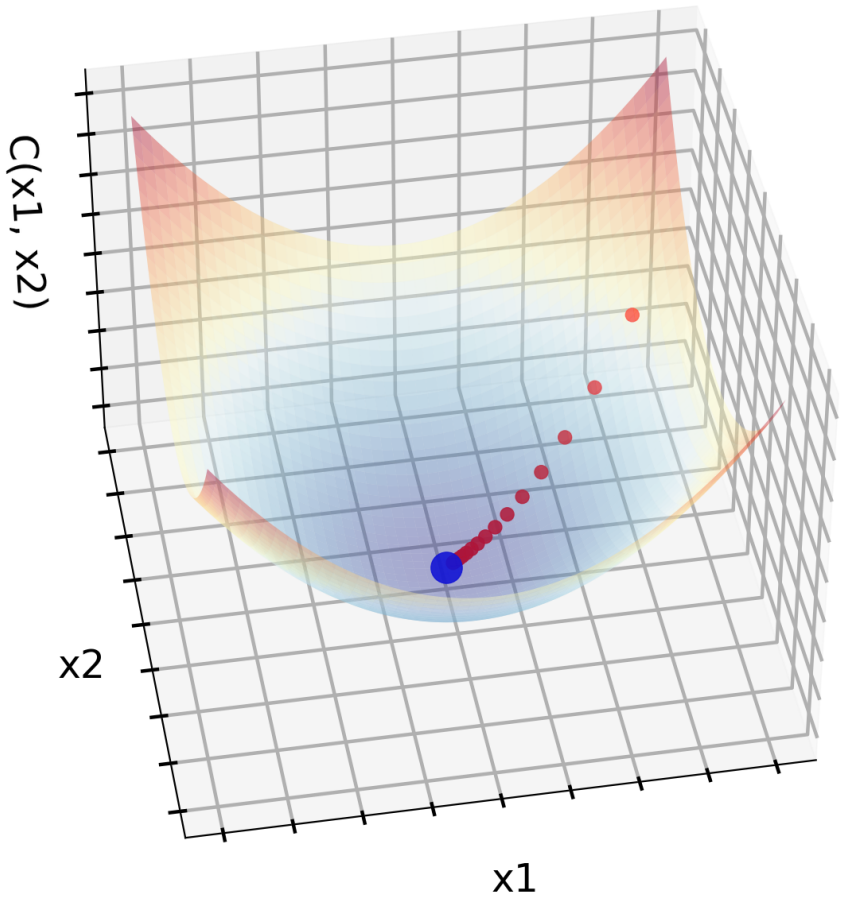


Figure: In blue the global minimum, in red the iteration points.

Gradient Descent: Effect of the learning rate/1

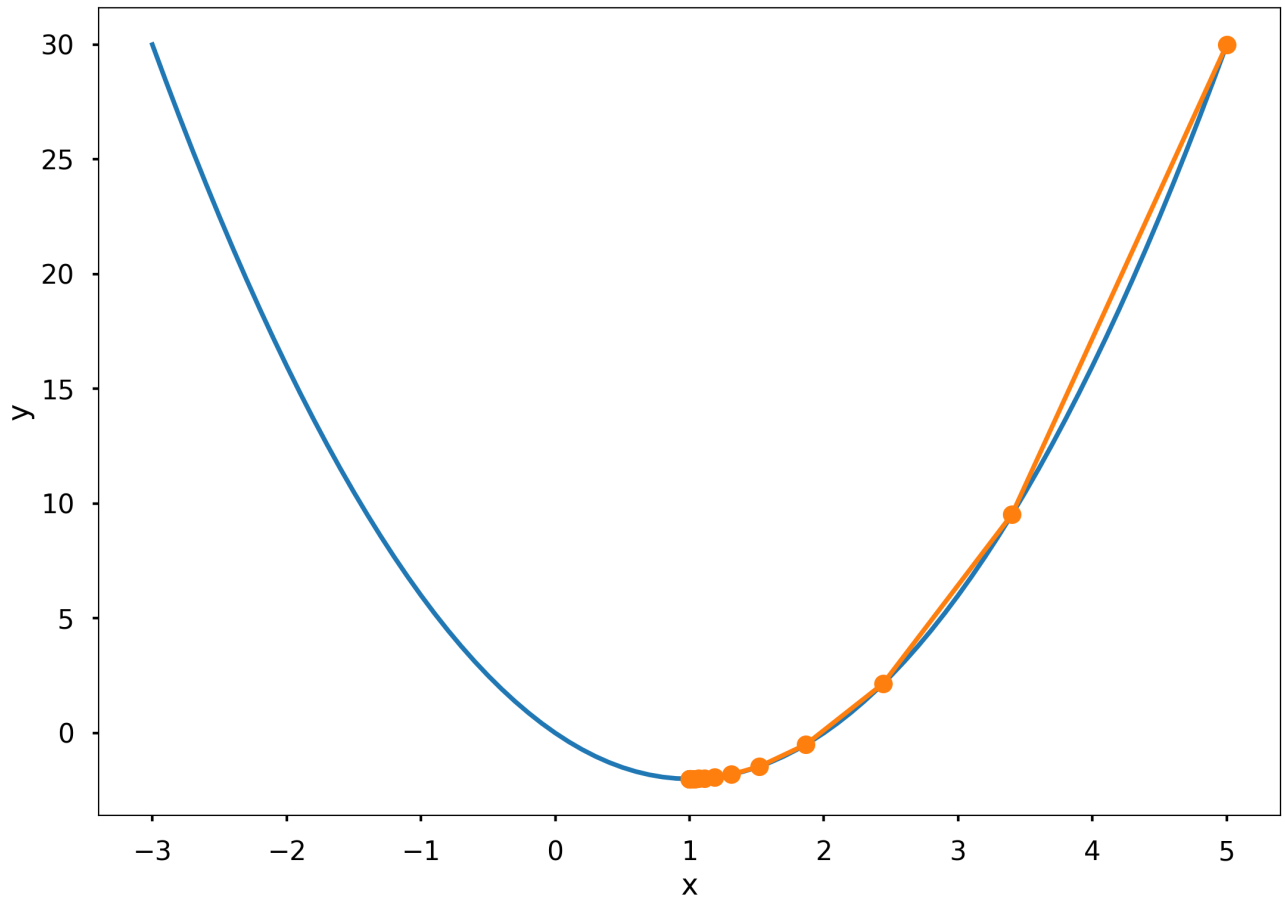


Figure: Learning rate = 0.1

Gradient Descent: Effect of the learning rate/2

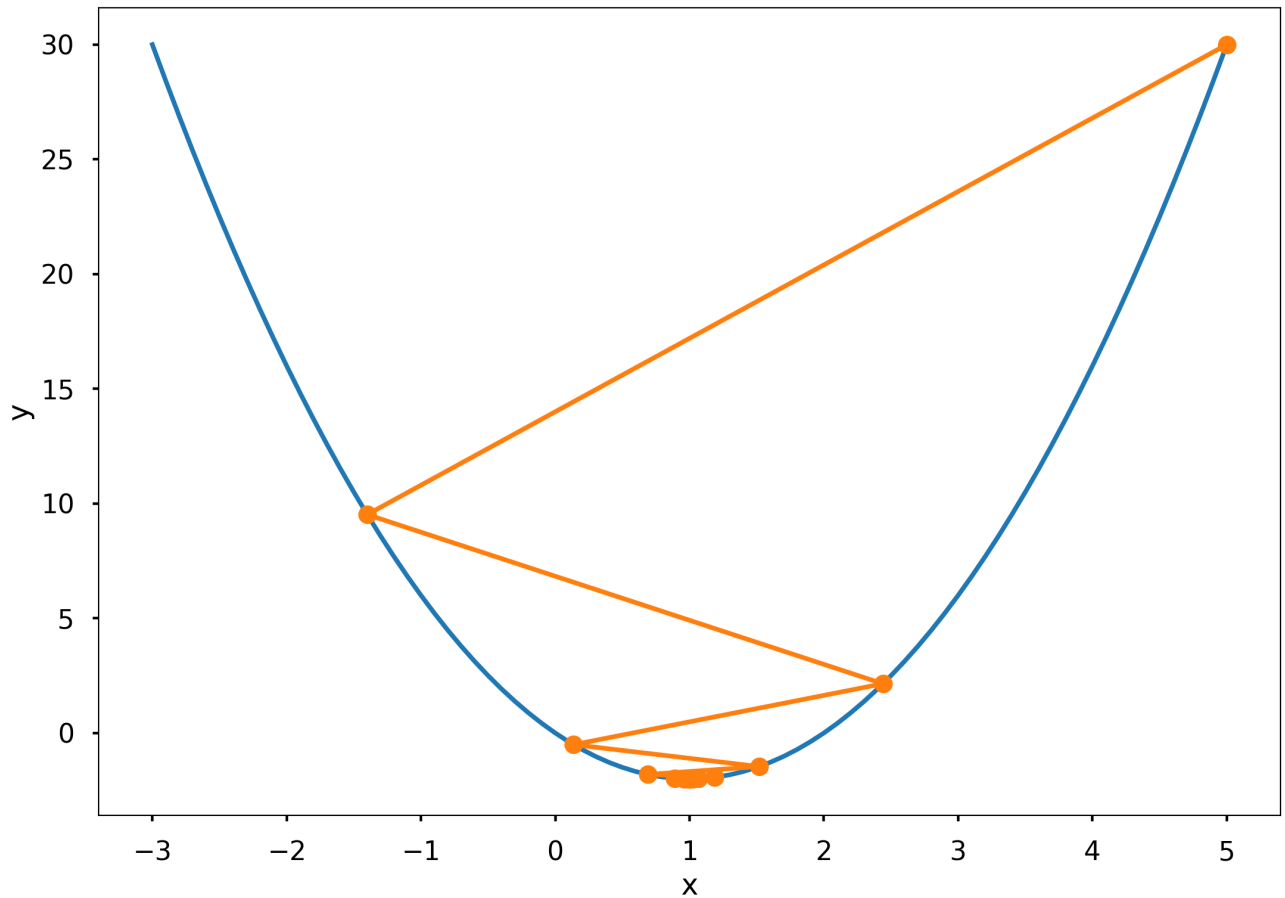


Figure: Learning rate = 0.4

Gradient Descent: Effect of the learning rate/3

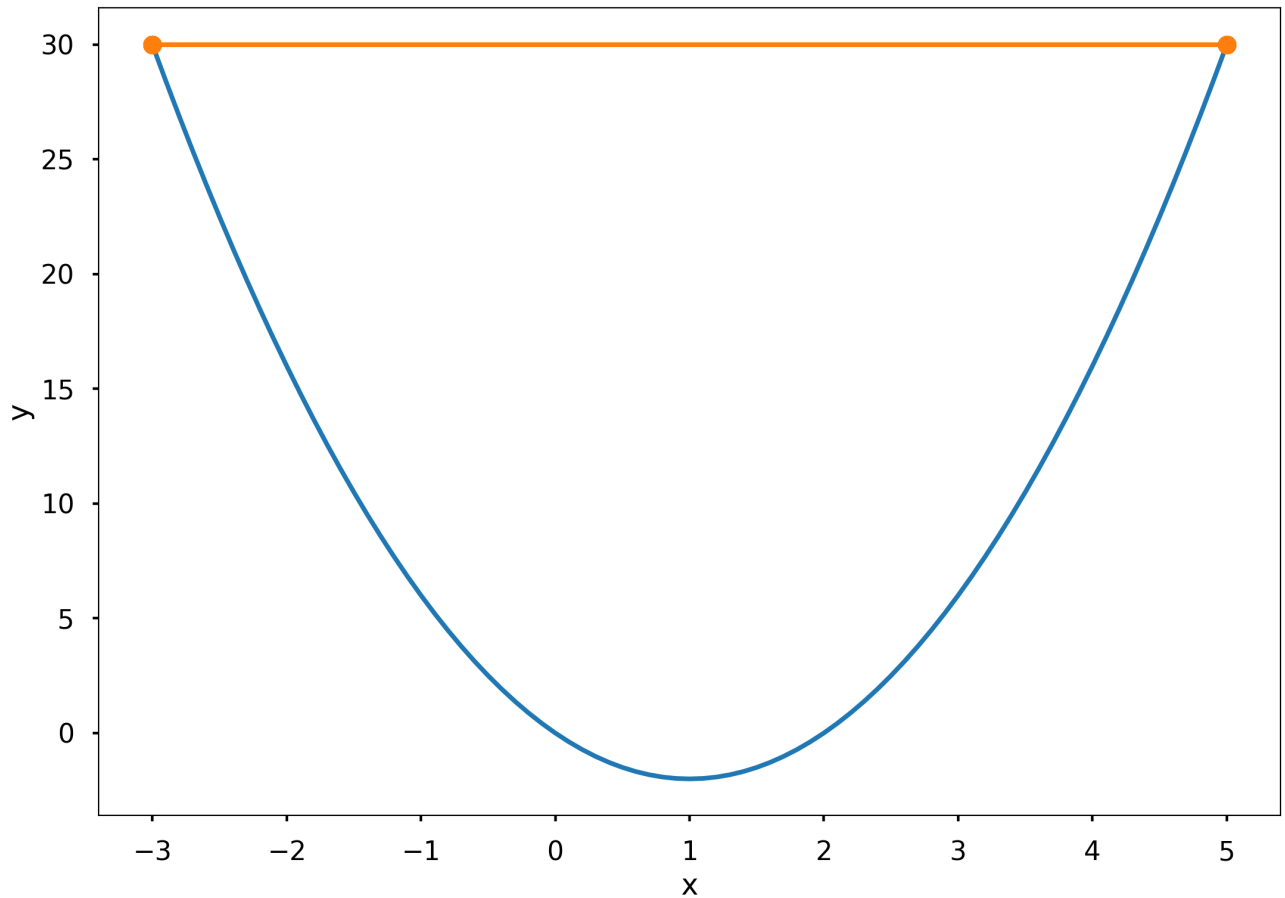


Figure: Learning rate = 0.5

Batch GD, SGD and Mini-Batch GD - Intuition

Notation: $E(\boldsymbol{w}) = 1/N \sum_{i=1}^{N} E_i(\boldsymbol{w})$
The classical gradient descent update rule, *i.e.* update the weights computing the gradient of the entire cost (error) function $E(\boldsymbol{w})$, is called **batch version**. However, for large number of samples (N) computing $\nabla E(\boldsymbol{w})$ is very time consuming.

To speed-up the update rule we approximate $\nabla E(\boldsymbol{w})$ with $\nabla E_i(\boldsymbol{w})$. This is the idea behind the so-called **Stochastic Gradient Descent** (SGD) or **online version**.

A trade-off between batch GD and SGD is called the **mini-batch** GD.

Batch GD, SGD and Mini-Batch GD - Algorithms

Batch GD

- Start with an initial guess $\boldsymbol{w}^0$.
- For $j \geq 0$, update $\boldsymbol{w}^{j+1} := \boldsymbol{w}^j - \alpha \nabla E(\boldsymbol{w}^j)$.

SGD (online)

- Start with an initial guess $\boldsymbol{w}^0$.
- For each epoch $j \geq 0$:
  - shuffle the dataset;
  - for each $1 \leq i \leq N$ update $\boldsymbol{w} := \boldsymbol{w} - \alpha \nabla E_i(\boldsymbol{w})$.

Mini-Batch GD

- Fix an integer $1 \leq \text{mb} \leq N$ (mini-batch size).
- Start with an initial guess $\boldsymbol{w}^0$.
- For each epoch $j \geq 0$:
  - shuffle the dataset;
  - for each $0 \leq i < \frac{N}{\text{mb}}$ update

$$\boldsymbol{w} := \boldsymbol{w} - \alpha \nabla \sum_{k=i \cdot \text{mb}+1}^{(i+1) \cdot \text{mb}} E_k(\boldsymbol{w}).$$

Tips and Tricks - How to choose?

- Batch: usually more stable and provide a more accurate estimation of the gradient, but slow.
- SGD: fast, stochastic approximation of the gradient implies possible instability (zig-zag effect)
- Mini-Batch GD: a trade-off between Batch GD and SGD (parallelism available).

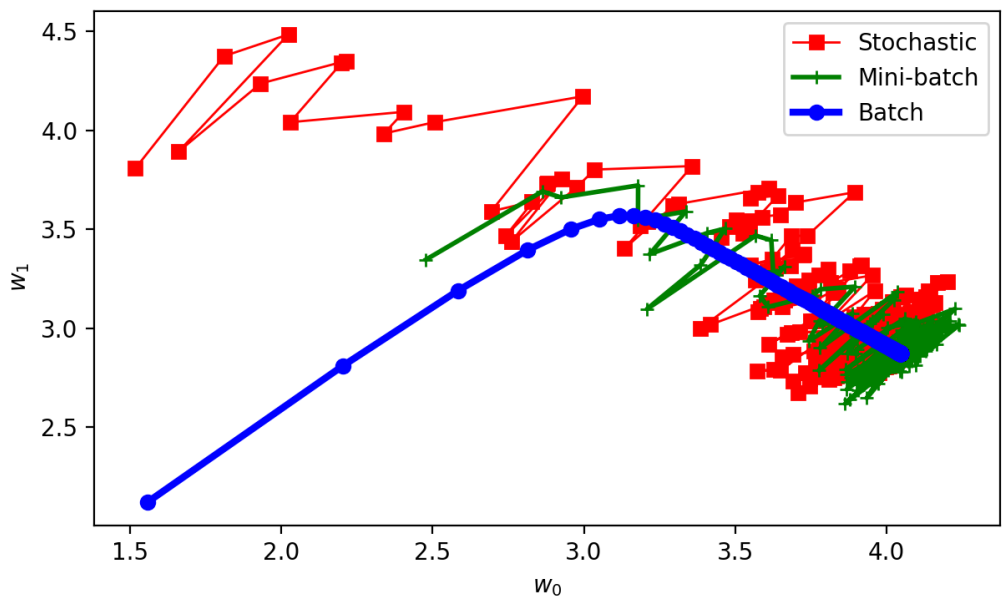


Figure: Batch GD vs SGD vs Mini-Batch GD

Normal Equation for LR and Gradient Descent

We have $E(\boldsymbol{w}) = \frac{1}{N}||\mathrm{X}\boldsymbol{w} - \boldsymbol{y}||^2$, hence

$$\nabla E(\boldsymbol{w}) = \frac{1}{N}\nabla(||\mathrm{X}\boldsymbol{w} - \boldsymbol{y}||^2) = \frac{2}{N}\mathrm{X}^T(\mathrm{X}\boldsymbol{w} - \boldsymbol{y})$$

- Normal equation ($\Longleftrightarrow$ holds if $\mathrm{X}^T\mathrm{X}$ is invertible):

$$\begin{aligned}\nabla E(\boldsymbol{w}) = 0 &\Longleftrightarrow \frac{2}{N}\mathrm{X}^T(\mathrm{X}\boldsymbol{w} - \boldsymbol{y}) = 0 \\ &\Longleftrightarrow \mathrm{X}^T\mathrm{X}\boldsymbol{w} = \mathrm{X}^T\boldsymbol{y} \\ &\Longleftrightarrow \boldsymbol{w} = (\mathrm{X}^T\mathrm{X})^{-1}\mathrm{X}^T\boldsymbol{y}\end{aligned}$$

- Gradient descent main iteration for LR:

$$\boldsymbol{w}^{j+1} := \boldsymbol{w}^j - \frac{2\alpha}{N}\mathrm{X}^T(\mathrm{X}\boldsymbol{w}^j - \boldsymbol{y})$$

Tips and Tricks - Invertibility of $X^T X$

Invertibility of $X^T X \iff$ columns of X linearly independent.

What if $X^T X$ is not invertible?

If two columns are linearly dependent, then those features are correlated (**redundant**).

Solution: discard one of those features.

Normal Equation vs Gradient Descent

Normal equation:

- No hyperparameters (explicit solution).
- No iterations.
- $\mathcal{O}(N^3)$, since this is the cost to invert a dense matrix. In particular, it is slow when N is large.

Gradient Descent:

- Need to choose the learning rate α.
- Needs many iterations.
- $\mathcal{O}(N^2)$, hence faster when N is large.

Tips and Tricks - Standardization

General (not only for LR): features must have similar magnitudes!

- Speed up the convergence of gradient descent.
- Try to have (on average) $-1 \leq \boldsymbol{x}^{(i)} \leq 1$.

Common techniques:

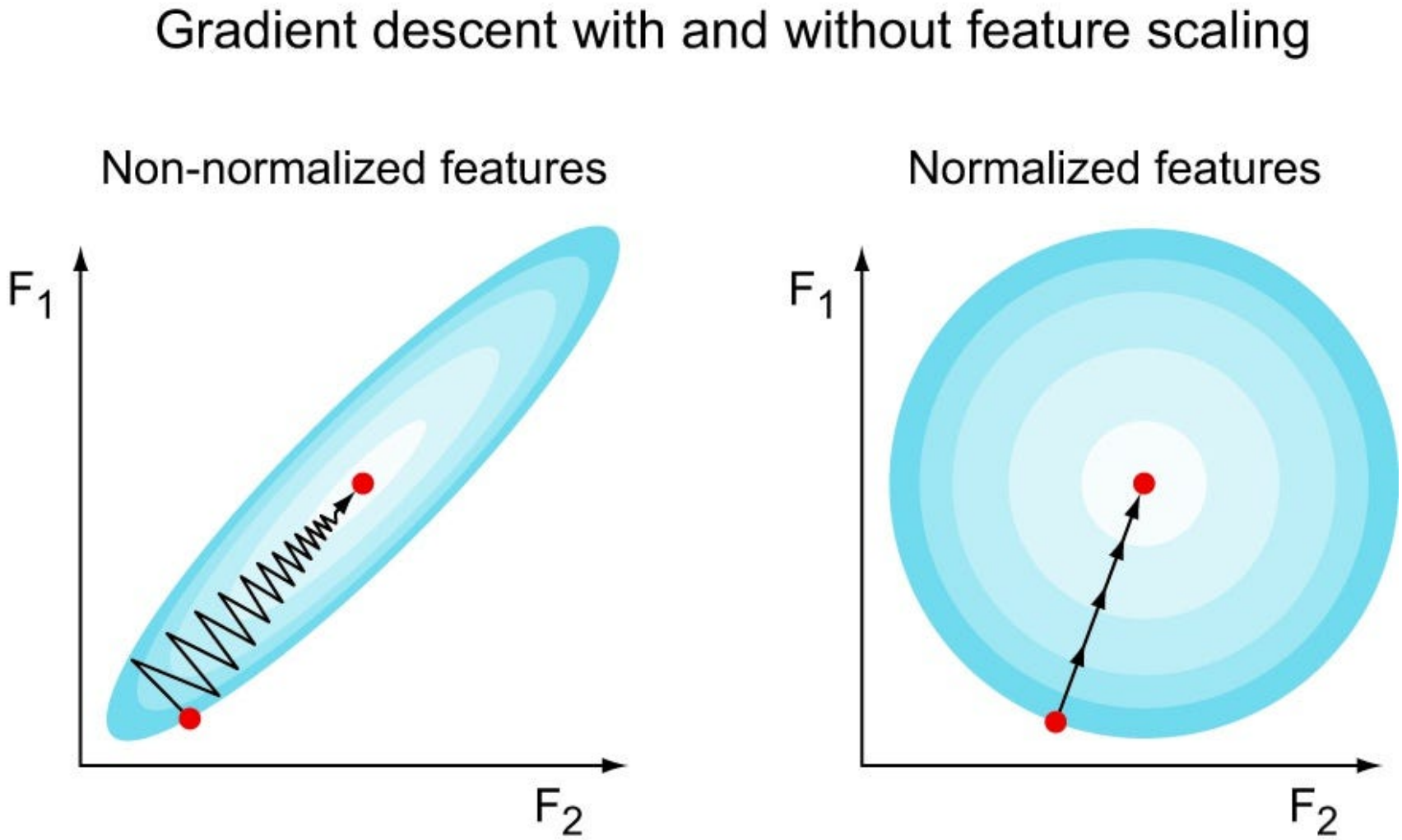
- **Feature scaling**. Compute the feature max $\boldsymbol{M} := [\max_i x_j^{(i)}]$ and the feature min $\boldsymbol{m} := [\min_i x_j^{(i)}]$ vectors. Then normalize features as follows

$$\boldsymbol{x}^i_{\text{norm}} = \frac{\boldsymbol{x}^{(i)} - \boldsymbol{m}}{\boldsymbol{M} - \boldsymbol{m}}$$

- **Mean normalization**. Compute the feature mean $\boldsymbol{\mu}$ ($\mu_j := \mathbb{E}[[x_j^{(i)}]_i]$) and the feature standard deviation $\boldsymbol{\sigma}$ ($\sigma_j := \sqrt{\text{Var}[[x_j^{(i)}]_i]}$). Then normalize features as follows

$$\boldsymbol{x}^{(i)}_{\text{norm}} = \frac{\boldsymbol{x}^{(i)} - \boldsymbol{\mu}}{\boldsymbol{\sigma}}$$

Tips and Tricks - Standardization



Polynomial regression (PR)

PR corresponds to polynomial hypothesis, i.e. of the form

$$h_{\boldsymbol{w}}(\boldsymbol{x}) = \sum_{j=0}^{n} w_j x_j^j.$$

More in general: linear basis expansion (LBE)

$$h_{\boldsymbol{w}}(\boldsymbol{x}) = \sum_{j=0}^{n} w_j \phi_j(\boldsymbol{x}),$$

where $\phi_j : \mathbb{R}^n \to \mathbb{R}$.

Underfitting and overfitting - main intuition

Imagine that you have to prepare an exam.

Doing only a few exercises lead a poor perfomance both on homeworks and on the exam exercises. This is called *underfitting*: you have a bad performance on the exam because you did not trained enough.

Moreover, brutally memorize all the homework lead to a perfect score on homeworks (trivially) but probably a bad score on the exam exercises. This is called *overfitting*: you have a bad performance on the exam because you did not captured the true essence of your homework, you have also memorize their "noise" (homework pecularities).

Underfitting and overfitting - another example

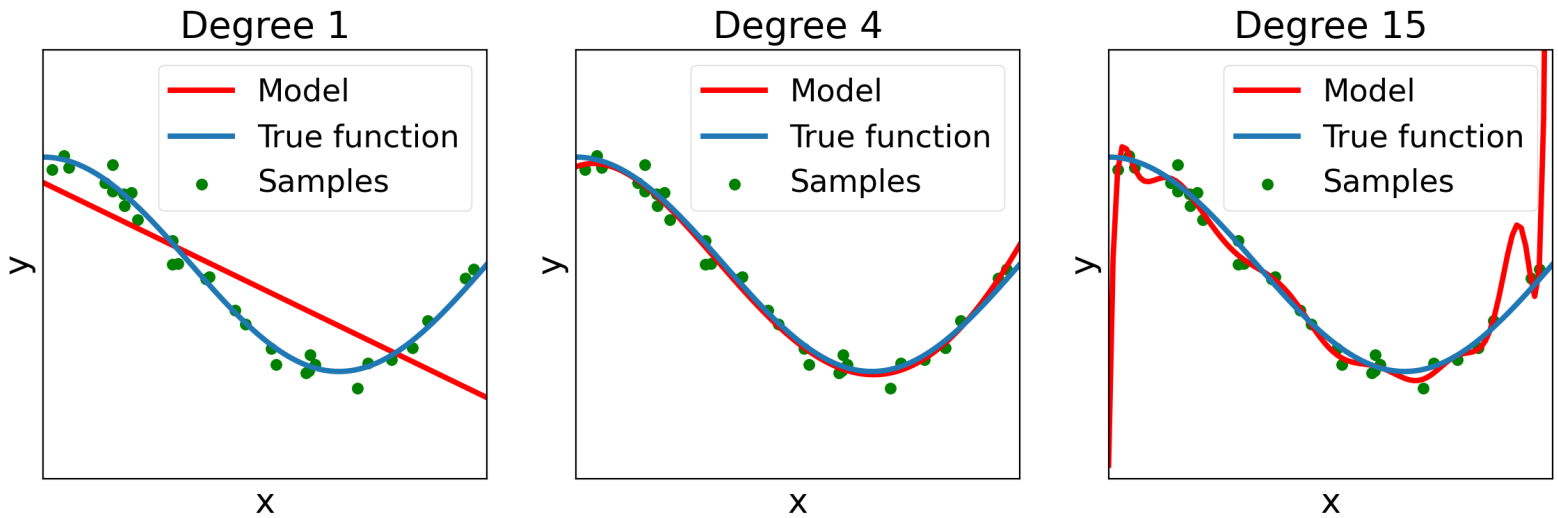


Figure: Underfitting (degree 1), good fitting (degree 4), overfitting (degree 15).

Overfitting can be countered in many ways, including:

- Validation, the true core of Machine Learning;
- Regularization.

Regularization

Overfitting is highly correlated with "complex" models. To avoid "complex" models, the idea is to penalize models with large weights. One way to do it is to consider the following cost function

$$E_r(\boldsymbol{w}) = E(\boldsymbol{w}) + R_\lambda(\boldsymbol{w}).$$

R_λ is called *regularization term*. $\lambda > 0$ is an hyperparameter that must be chosen in the model selection phase.
Most common regularization techniques:

- Tikhonov regularization: $R_\lambda(\boldsymbol{w}) = \lambda||w||_2$. Tends to bring all the weights to small values.
- Lasso: $R_\lambda(\boldsymbol{w}) = \lambda||w||_1$. Tends to bring some weights to 0 (feature selection).

LR with Tikhonov regularization

Cost function (with no regularization):

$$E(\boldsymbol{w}) = \frac{1}{N}||\mathrm{X}\boldsymbol{w} - \boldsymbol{y}||^2.$$

New cost function:

$$E_r(\boldsymbol{w}) = E(\boldsymbol{w}) + \lambda||\boldsymbol{w}||^2.$$

Gradient of the cost function:

$$\nabla E_r(\boldsymbol{w}) = \nabla E(\boldsymbol{w}) + 2\lambda\boldsymbol{w} = 2(\frac{1}{N}\mathrm{X}^T(\mathrm{X}\boldsymbol{w} - \boldsymbol{y}) + \lambda\boldsymbol{w})$$

Normal equation:

$$\boldsymbol{w} = (\mathrm{X}^T\mathrm{X} + \lambda\mathrm{I})^{-1}\mathrm{X}^T\boldsymbol{y}$$

Note that in this case $\mathrm{X}^T\mathrm{X} + \lambda\mathrm{I}$ is *always* invertible (why? :))