Department of Mathematics
ETH Zurich

CAS ETH Machine Learning in Finance and Insurance

BLOCK I. Introduction to Machine Learning. Lecture 2.

Dr. A. Ferrario, ETH Zurich and UZH

Last week we…

1	Introduced the mathematical background needed for **Block I: Introduction to Machine Learning**
2	Discussed what statistical learning is and focused on the formalism of supervised learning

Machine Learning Methods (Part 1)

- Linear regression

Linear regression

We will use slides to introduce our topics and the blackboard to deep-dive into selected items

Linear regression: ideas

I	Simplest model to predict numerical outputs (distances, temperatures, prices, time durations…). It assumes a linear relationship between a scalar output/outcome and the dependent variables, up to random contributions that we call “noise”
II	Although the linear assumption is often an oversimplification, it is rather useful to learn the basics of machine learning modelling
III	Linear models are also used in statistics. Here, we do machine learning: our main goal is to compute numerical predictions on unseen test data as accurately as possible

Linear regression
Notation

Model

$$Y = \theta_0 + \theta_1 X_1 + \cdots + \theta_d X_d + \epsilon, \quad \epsilon \sim \mathcal{N}(0, \sigma^2) \text{ independent of } (X_1, \ldots, X_d)$$

We model a numerical output r.v. with a linear comb. of *d+1* r.v.

Linear regression
Notation

Candidate $f_{\theta}(X)$ to approximate the "true" model f such that $Y = f(X) + \epsilon$

Model

$$Y = \theta_0 + \theta_1 X_1 + \cdots + \theta_d X_d + \epsilon, \quad \epsilon \sim \mathcal{N}(0, \sigma^2) \text{ independent of } (X_1, \ldots, X_d)$$

We model a numerical output r.v. with a linear comb. of *d+1* r.v.

Linear combination of *d+1* random variables

these assumptions allow to determine statistical properties of the estimators

output random variable

error term (also a random variable)

Linear regression
Notation

Model

$$Y = \theta_0 + \theta_1 X_1 + \cdots + \theta_d X_d + \epsilon, \quad \epsilon \sim \mathcal{N}(0, \sigma^2) \text{ independent of } (X_1, \ldots, X_d)$$

We model a numerical output r.v. with a linear comb. of *d+1* r.v.

Empirical setting

Let $(x_{i1}, \ldots, x_{id}, y_i, \varepsilon_i)$, $i = 1, \ldots, m$, be iid realizations of $(X_1, \ldots, X_d, Y, \varepsilon)$

Matrix notation: $y = A\boldsymbol{\theta} + \epsilon$ where

$$y = \begin{pmatrix} y_1 \\ \vdots \\ \vdots \\ y_m \end{pmatrix}, \; A = \underbrace{\begin{pmatrix} 1 & x_{11} & \ldots & x_{1d} \\ 1 & x_{21} & \ldots & x_{2d} \\ \vdots & \vdots & \ldots & \vdots \\ 1 & x_{m1} & \ldots & x_{md} \end{pmatrix}}_{\text{design matrix}}, \; \boldsymbol{\theta} = \begin{pmatrix} \theta_0 \\ \theta_1 \\ \vdots \\ \theta_d \end{pmatrix}, \; \boldsymbol{\varepsilon} = \begin{pmatrix} \varepsilon_1 \\ \vdots \\ \vdots \\ \varepsilon_m \end{pmatrix}$$

typically, $m \geq d + 1$ *(more observations than parameters)*

We can use the matrix notation to represent the (empirical) linear model in a compact form

Linear regression
Notation

Model

$$Y = \theta_0 + \theta_1 X_1 + \cdots + \theta_d X_d + \epsilon, \quad \epsilon \sim \mathcal{N}(0, \sigma^2) \text{ independent of } (X_1, \ldots, X_d)$$

We model a numerical output r.v. with a linear comb. of *d+1* r.v.

Empirical setting

Let $(x_{i1}, \ldots, x_{id}, y_i, \varepsilon_i)$, $i = 1, \ldots, m$, be iid realizations of $(X_1, \ldots, X_d, Y, \varepsilon)$

Matrix notation: $y = A\boldsymbol{\theta} + \epsilon$ where

$$y = \begin{pmatrix} y_1 \\ \vdots \\ y_m \end{pmatrix}, \quad A = \underbrace{\begin{pmatrix} 1 & x_{11} & \ldots & x_{1d} \\ 1 & x_{21} & \ldots & x_{2d} \\ \vdots & \vdots & \ldots & \vdots \\ 1 & x_{m1} & \ldots & x_{md} \end{pmatrix}}_{\text{design matrix}}, \quad \boldsymbol{\theta} = \begin{pmatrix} \theta_0 \\ \theta_1 \\ \vdots \\ \theta_d \end{pmatrix}, \quad \boldsymbol{\varepsilon} = \begin{pmatrix} \varepsilon_1 \\ \vdots \\ \varepsilon_m \end{pmatrix}$$

typically, $m \geq d + 1$ *(more observations than parameters)*

We can use the matrix notation to represent the (empirical) linear model in a compact form

Data! Realizations of the output r.v.

The design matrix depends on data! Realizations of the *d+1* r.v.

These coefficients are unknown…we need to estimate them

Again, these are data

Linear regression
Learning the model with Ordinary Least Squares (OLS)

Empirical risk with squared loss

$$E(\boldsymbol{\theta}) = \frac{1}{m}\sum_{i=1}^{m} L(f_{\boldsymbol{\theta}}(x_i), y_i) = \frac{1}{m}\sum_{i=1}^{m} (\overbrace{\theta_0 + \theta_1 x_{i1} + \cdots + \theta_d x_{id}}^{f_{\boldsymbol{\theta}}(x_i)} - y_i)^2$$

Notation

$$\min_{\boldsymbol{\theta}\in\mathbb{R}^{d+1}} E(\boldsymbol{\theta}) = \min_{\boldsymbol{\theta}\in\mathbb{R}^{d+1}} \|A\boldsymbol{\theta} - y\|_2^2 \qquad \|z\|_2^2 = \sum_{i=1}^{m} z_i^2, \quad z \in \mathbb{R}^m$$

We model a numerical output r.v. with a linear comb. of *d+1* r.v.

Lemma

$\boldsymbol{\theta} \in \mathbb{R}^{d+1}$ minimizes $E(\boldsymbol{\theta}) \Leftrightarrow A^T A\boldsymbol{\theta} = A^T y$ (normal equations)

If the columns of A are linearly independent, $A^T A$ is regular, and the unique solution of $A^T A\boldsymbol{\theta} = A^T y$ is

$$\hat{\boldsymbol{\theta}} = (A^T A)^{-1} A^T y$$

(Python computes these parameters for us)

Linear regression
Learning the model with Ordinary Least Squares (OLS)

Empirical risk with squared loss

$$E(\boldsymbol{\theta}) = \frac{1}{m}\sum_{i=1}^{m} L(f_{\boldsymbol{\theta}}(x_i), y_i) = \frac{1}{m}\sum_{i=1}^{m} (\overbrace{\theta_0 + \theta_1 x_{i1} + \cdots + \theta_d x_{id}}^{f_{\boldsymbol{\theta}}(x_i)} - y_i)^2$$

Notation

$$\min_{\boldsymbol{\theta}\in\mathbb{R}^{d+1}} E(\boldsymbol{\theta}) = \min_{\boldsymbol{\theta}\in\mathbb{R}^{d+1}} \|A\boldsymbol{\theta} - y\|_2^2 \qquad \|z\|_2^2 = \sum_{i=1}^{m} z_i^2, \quad z \in \mathbb{R}^m$$

We model a numerical output r.v. with a linear comb. of *d+1* r.v.

Lemma

$\boldsymbol{\theta} \in \mathbb{R}^{d+1}$ minimizes $E(\boldsymbol{\theta}) \Leftrightarrow A^T A\boldsymbol{\theta} = A^T y$ (normal equations)

If the columns of A are linearly independent, $A^T A$ is regular, and the unique solution of $A^T A\boldsymbol{\theta} = A$

$$\hat{\boldsymbol{\theta}} = (A^T A)^{-1} A^T y$$

(Python computes these param

We will be able to understand this equivalence when we will discuss the minima of functions in some detail

Linear regression
Learning the model with Ordinary Least Squares (OLS)

Empirical risk with squared loss

$$E(\boldsymbol{\theta}) = \frac{1}{m}\sum_{i=1}^{m} L(f_{\boldsymbol{\theta}}(x_i), y_i) = \frac{1}{m}\sum_{i=1}^{m} (\overbrace{\theta_0 + \theta_1 x_{i1} + \cdots + \theta_d x_{id}}^{f_{\boldsymbol{\theta}}(x_i)} - y_i)^2$$

Predicting with the trained/learned linear regression model

$$\text{For each data point } x: \ \hat{y} := f_{\hat{\boldsymbol{\theta}}}(x) = \hat{\theta}_0 + \hat{\theta}_1 x_1 + \cdots + \hat{\theta}_d x_d.$$

comb. of *d+1* r.v.

Lemma

$\boldsymbol{\theta} \in \mathbb{R}^{d+1}$ minimizes $E(\boldsymbol{\theta}) \Leftrightarrow A^T A\boldsymbol{\theta} = A^T y$ (normal equations)

If the columns of A are linearly independent, $A^T A$ is regular, and the unique solution of $A^T A\boldsymbol{\theta} = A^T y$ is

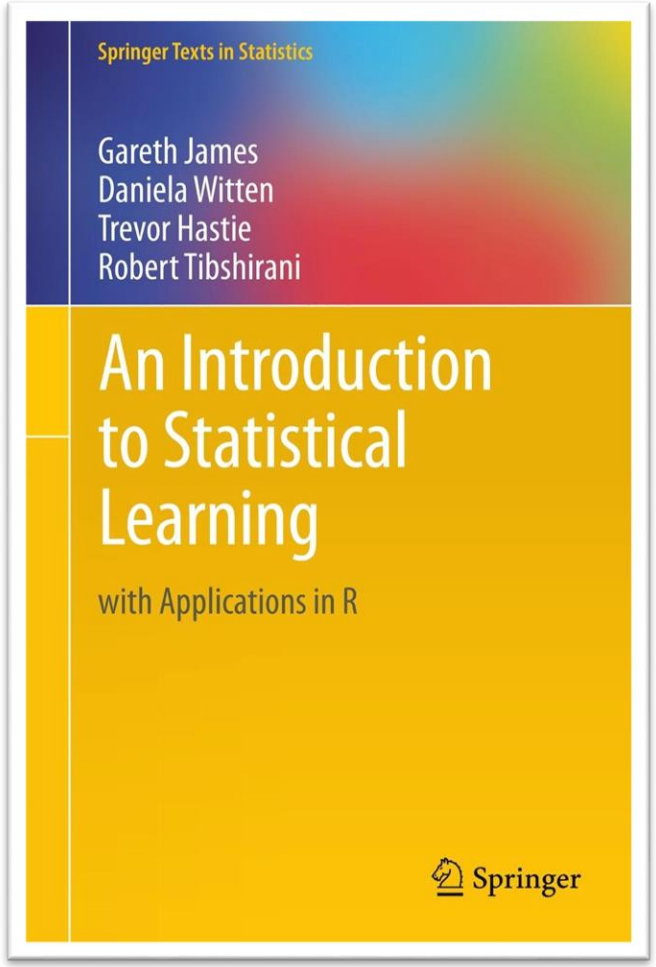
$$\hat{\boldsymbol{\theta}} = (A^T A)^{-1} A^T y$$

(Python computes these parameters for us)

Linear regression with regularization

What if we have a lot of predictors (and, possibly, few data points)?

Regularization is a methodology that is used to improve (1) the performance of the linear regression on test data by reducing its dependence on training data ("variance") and (2) the interpretability of the model by reducing the number of variables that *are associated* with the output variable

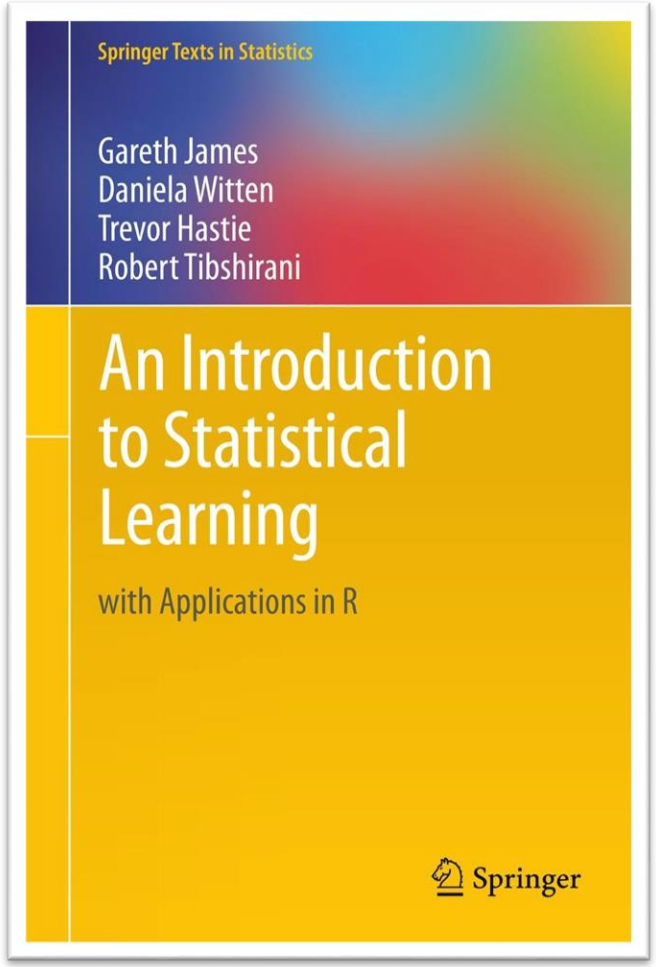


James, G., Witten, D., Hastie, T., & Tibshirani, R. (2013). An introduction to statistical learning (Vol. 112, p. 18). New York: Springer

Linear regression with regularization

What if we have a lot of predictors (and, possibly, few data points)?

"By *constraining* or *shrinking* the estimated coefficients, we can often substantially reduce the variance at the cost of a negligible increase in bias. This can lead to substantial improvements in the accuracy with which we can predict the response for observations not used in model training" (pag. 230, emphasis in original)

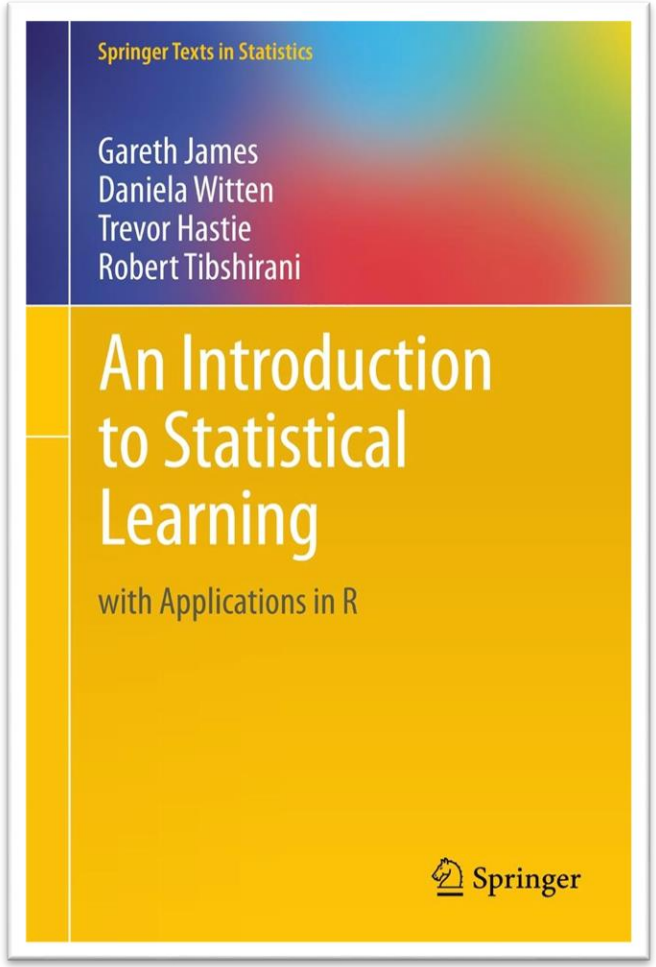


James, G., Witten, D., Hastie, T., & Tibshirani, R. (2013). An introduction to statistical learning (Vol. 112, p. 18). New York: Springer

Linear regression with regularization

What if we have a lot of predictors (and, possibly, few data points)?

"*Shrinkage*. This approach involves fitting a model involving all *p* predictors. However, the estimated coefficients are shrunken towards zero relative to the least squares estimates. This shrinkage (also known as regularization) has the effect of reducing variance. Depending on what type of shrinkage is performed, some of the coefficients may be estimated to be exactly zero. Hence, shrinkage methods can also perform variable selection" (pag. 230, emphasis in original)



James, G., Witten, D., Hastie, T., & Tibshirani, R. (2013). An introduction to statistical learning (Vol. 112, p. 18). New York: Springer

Linear regression with regularization
Ridge, LASSO

“hyperparameter”

$$\min_{\boldsymbol{\theta}\in\mathbb{R}^{d+1}} \left(\|A\boldsymbol{\theta}-y\|_2^2 + \lambda r(\boldsymbol{\theta})\right)$$

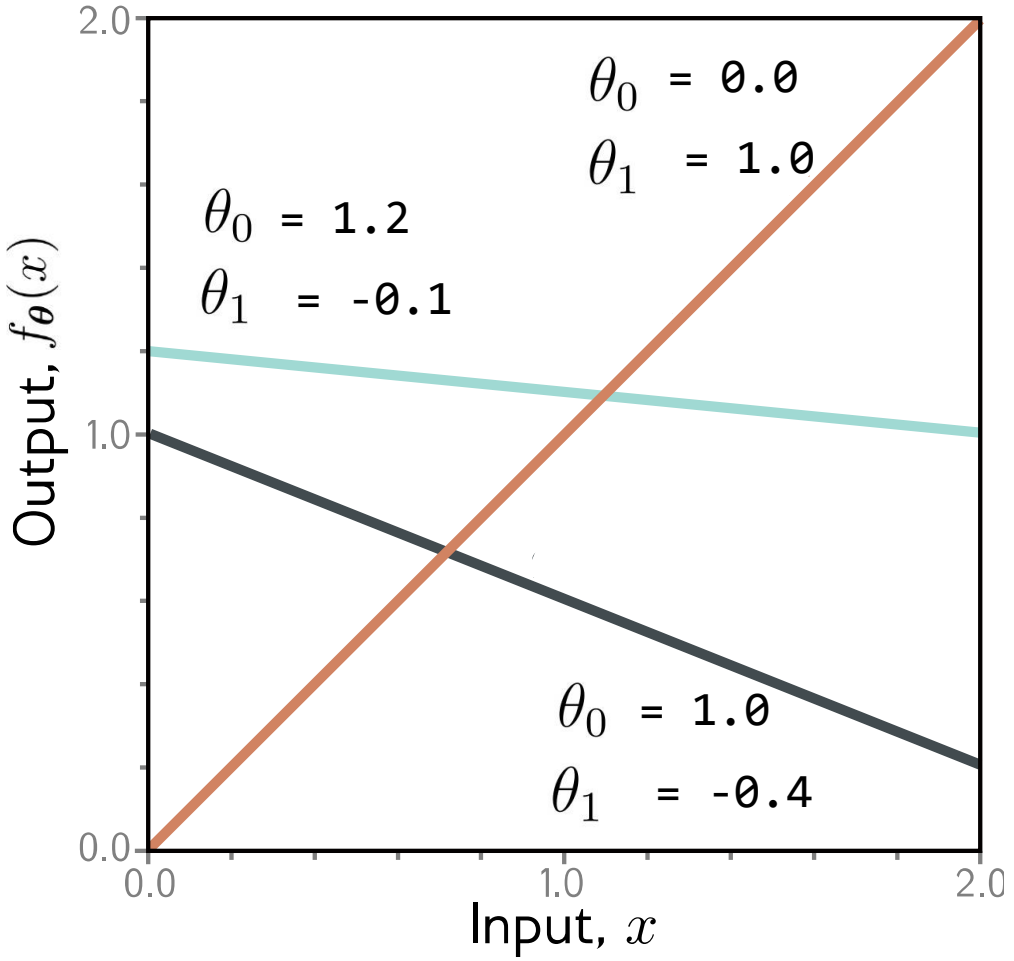
$$r(\boldsymbol{\theta}) = \sum_{i=1}^{d} \theta_i^2$$ **Ridge**

$$r(\boldsymbol{\theta}) = \sum_{i=1}^{d} |\theta_i|$$ **LASSO (Least absolute shrinkage and selection operator)**

Let us train a machine learning model
Toy example: linear regression in one variable

$$f_{\boldsymbol{\theta}}(x) = \theta_0 + \theta_1 x$$

intercept slope



From , S.J.D. Prince (2023), Understanding Deep Learning, MIT Press, Available at "http://udlbook.com"

Let us train a machine learning model
Toy example: linear regression in one variable

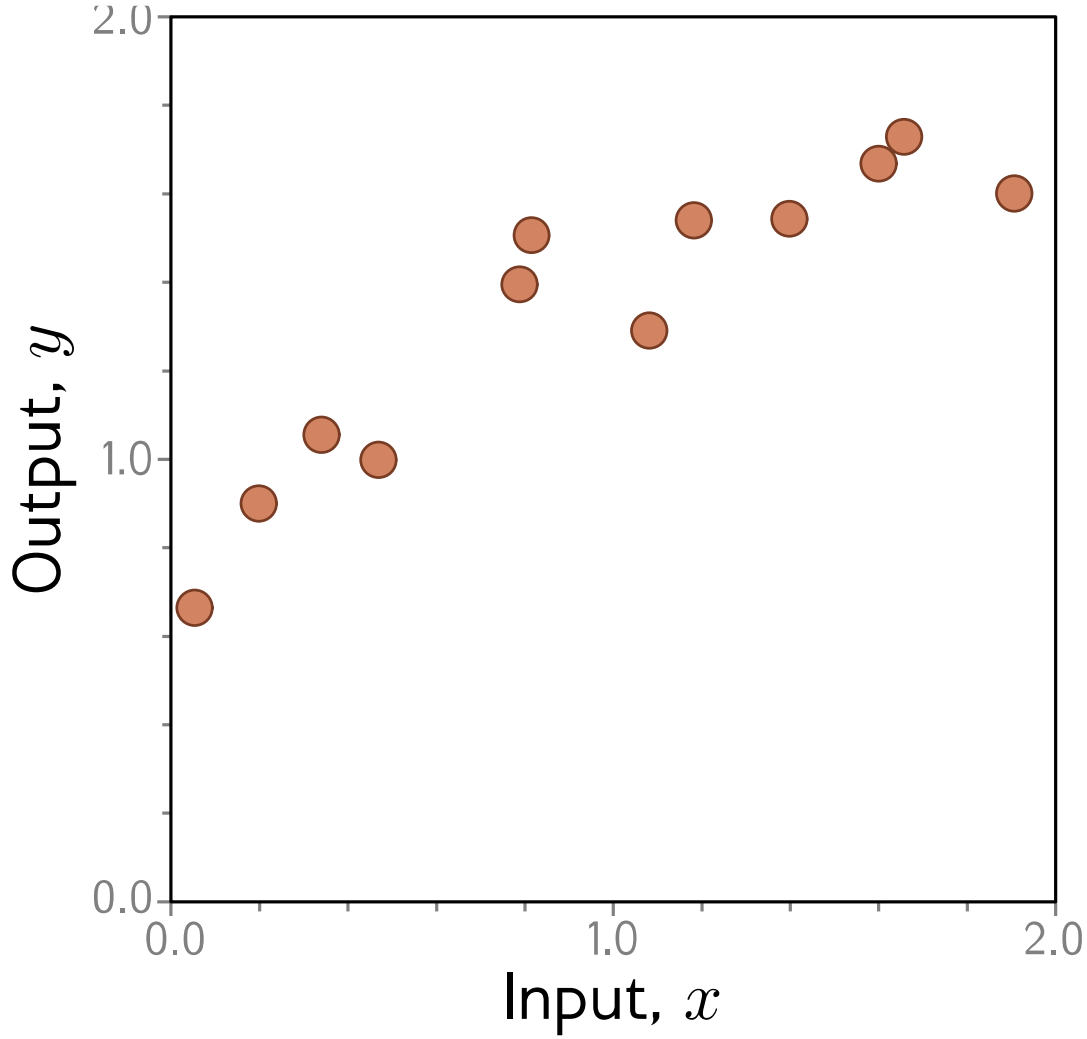
Let us simulate N samples $\{x_i, y_i\}$ (N=12)

We choose to minimize the empirical risk:

$$E(\boldsymbol{\theta}) = \frac{1}{12}\sum_{i=1}^{12} L(f_{\boldsymbol{\theta}}(x_i), y_i) =$$

$$= \frac{1}{12}\sum_{i=1}^{12} (\underbrace{\theta_0 + \theta_1 x_i}_{\text{approximation of the true value } y_i} - y_i)^2$$

true value



From , S.J.D. Prince (2023), Understanding Deep Learning, MIT Press, Available at "http://udlbook.com"

Let us train a machine learning model
Toy example: linear regression in one variable

Let us simulate N samples $\{x_i, y_i\}$ (N=12)

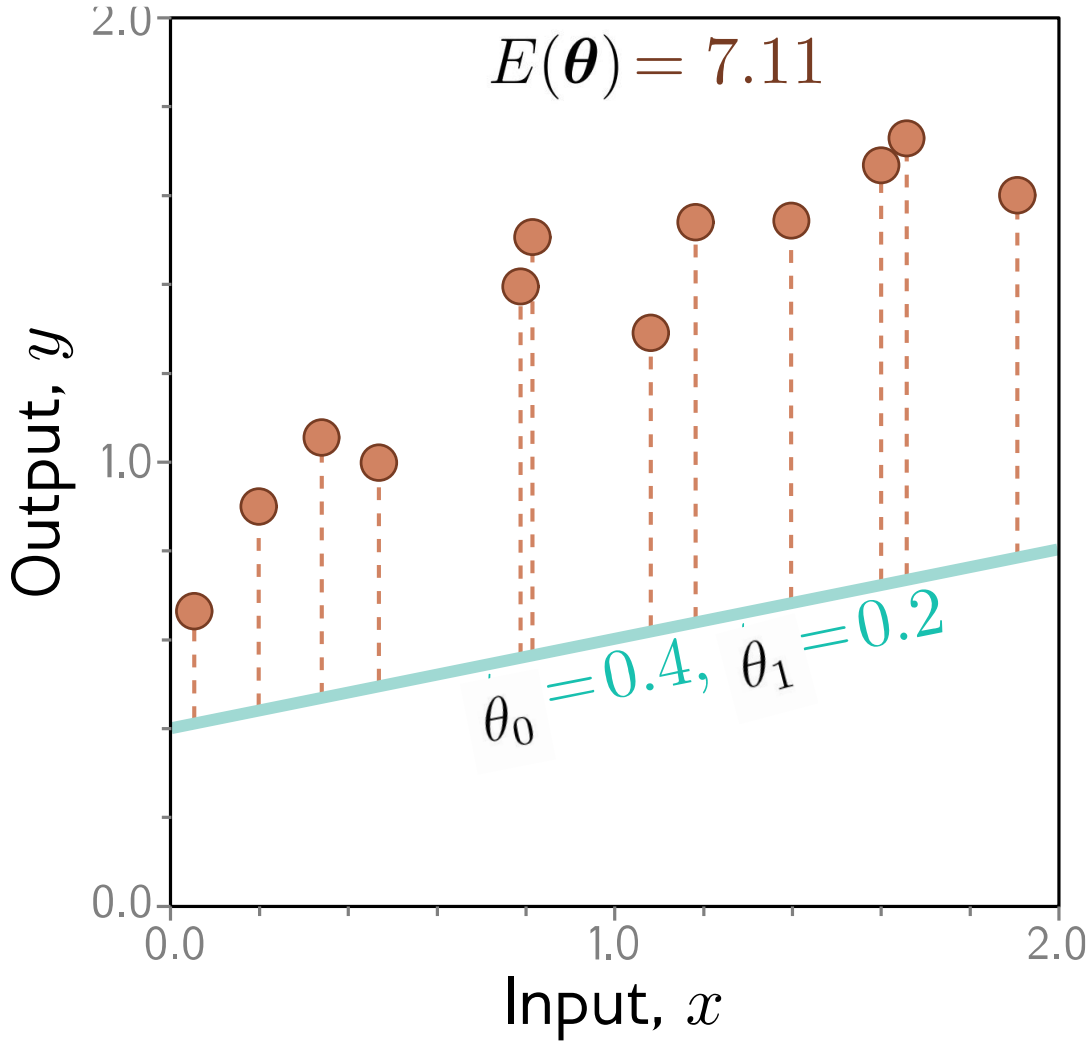
We choose to minimize the empirical risk:

$$E(\boldsymbol{\theta}) = \frac{1}{12}\sum_{i=1}^{12} L(f_{\boldsymbol{\theta}}(x_i), y_i) =$$

$$= \frac{1}{12}\sum_{i=1}^{12}(\theta_0 + \theta_1 x_i - y_i)^2$$

$\theta_0 + \theta_1 x_i$: approximation of the true value y_i

y_i: true value



From , S.J.D. Prince (2023), Understanding Deep Learning, MIT Press, Available at "http://udlbook.com"

Let us train a machine learning model
Toy example: linear regression in one variable

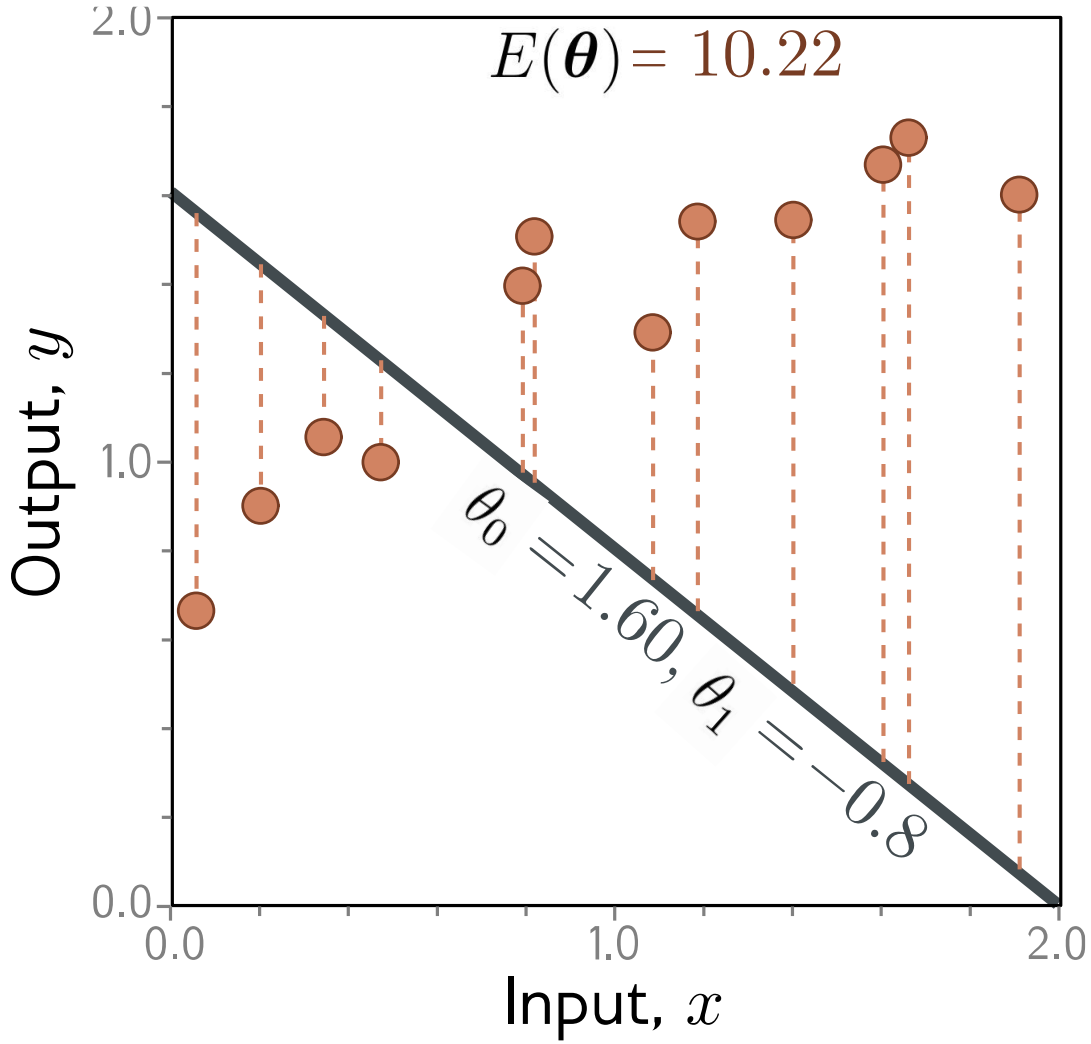
Let us simulate N samples $\{x_i, y_i\}$ (N=12)

We choose to minimize the empirical risk:

$$E(\boldsymbol{\theta}) = \frac{1}{12}\sum_{i=1}^{12} L(f_{\boldsymbol{\theta}}(x_i), y_i) =$$

$$= \frac{1}{12}\sum_{i=1}^{12} (\underbrace{\theta_0 + \theta_1 x_i}_{\text{approximation of the true value } y_i} - y_i)^2$$

true value



From , S.J.D. Prince (2023), Understanding Deep Learning, MIT Press, Available at "http://udlbook.com"

Let us train a machine learning model
Toy example: linear regression in one variable

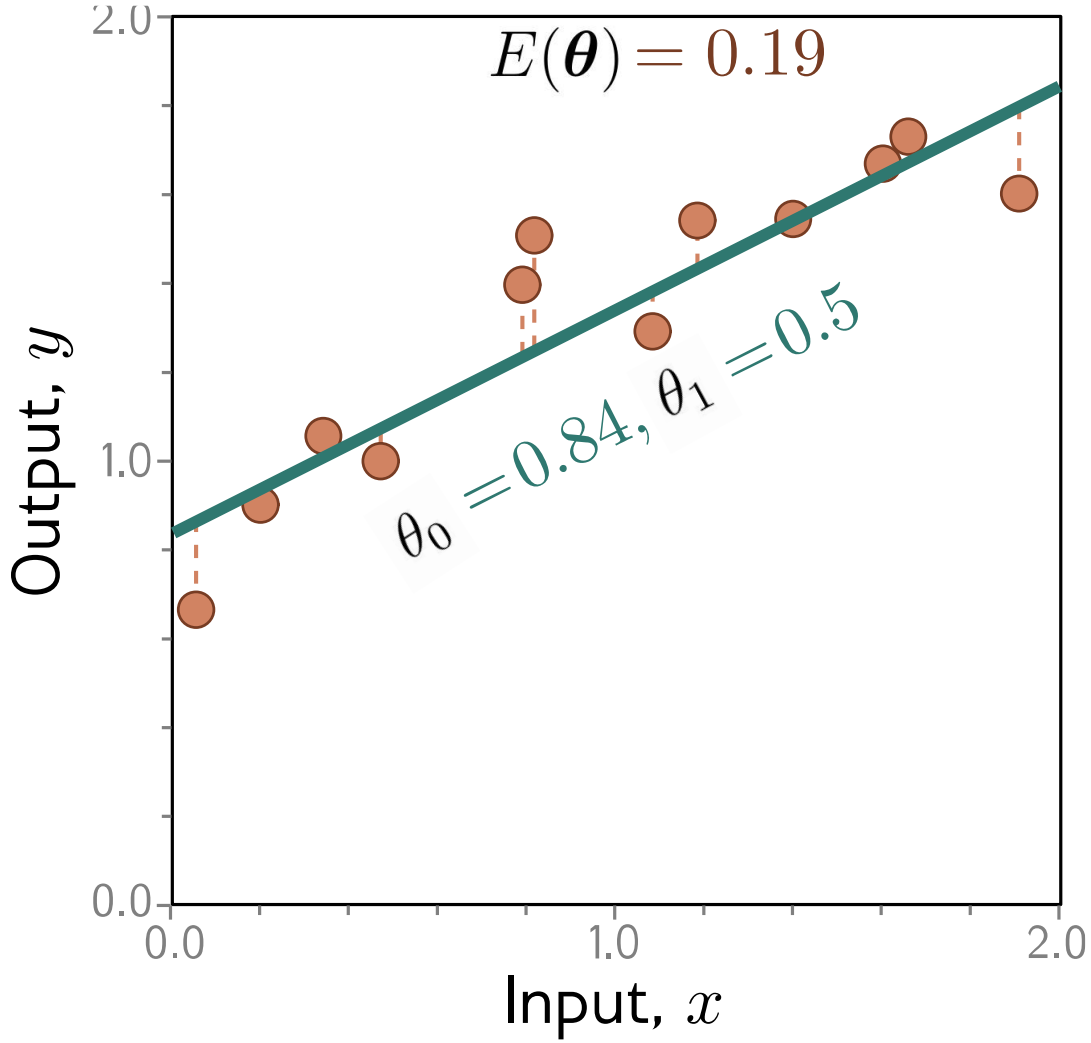
Let us simulate N samples $\{x_i, y_i\}$ (N=12)

We choose to minimize the empirical risk:

$$E(\boldsymbol{\theta}) = \frac{1}{12}\sum_{i=1}^{12} L(f_{\boldsymbol{\theta}}(x_i), y_i) =$$

$$= \frac{1}{12}\sum_{i=1}^{12} (\underbrace{\theta_0 + \theta_1 x_i}_{\text{approximation of the true value } y_i} - y_i)^2$$

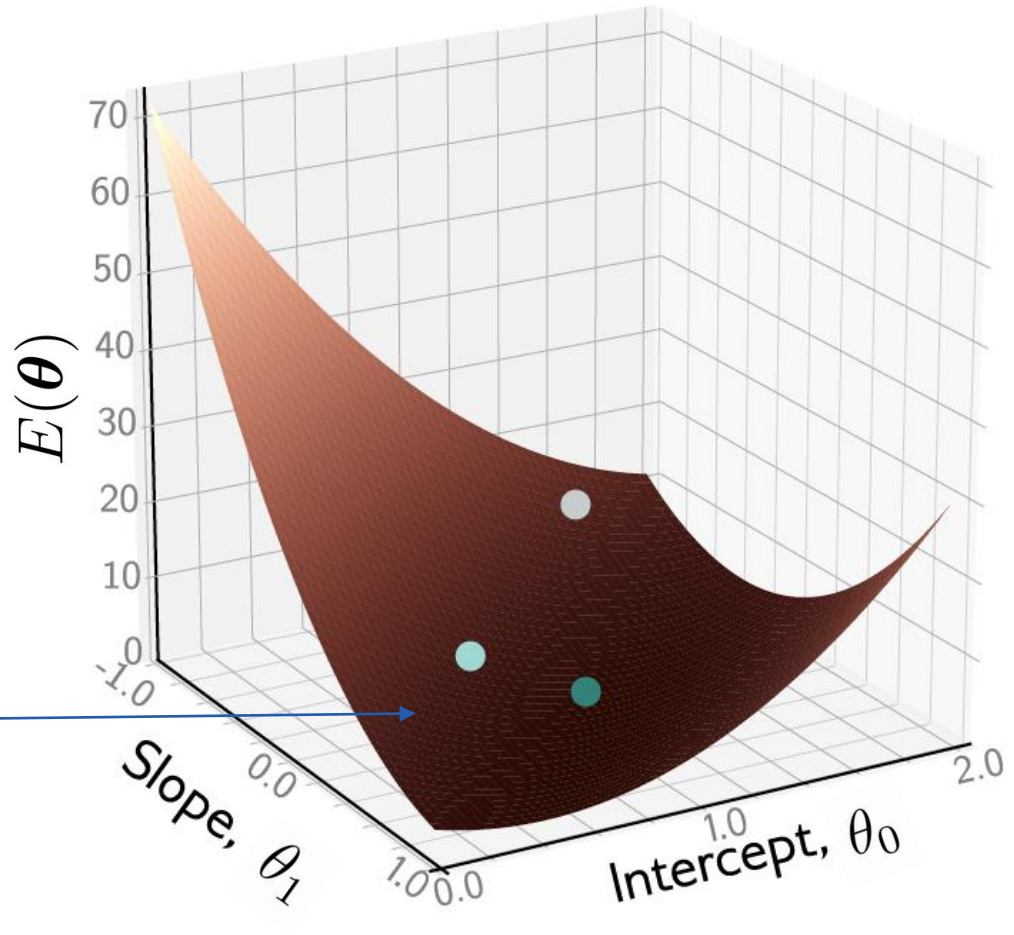
true value



From , S.J.D. Prince (2023), Understanding Deep Learning, MIT Press, Available at "http://udlbook.com"

Let us train a machine learning model
Toy example: linear regression in one variable

The darker the colour, the smaller the value of $E(\boldsymbol{\theta})$



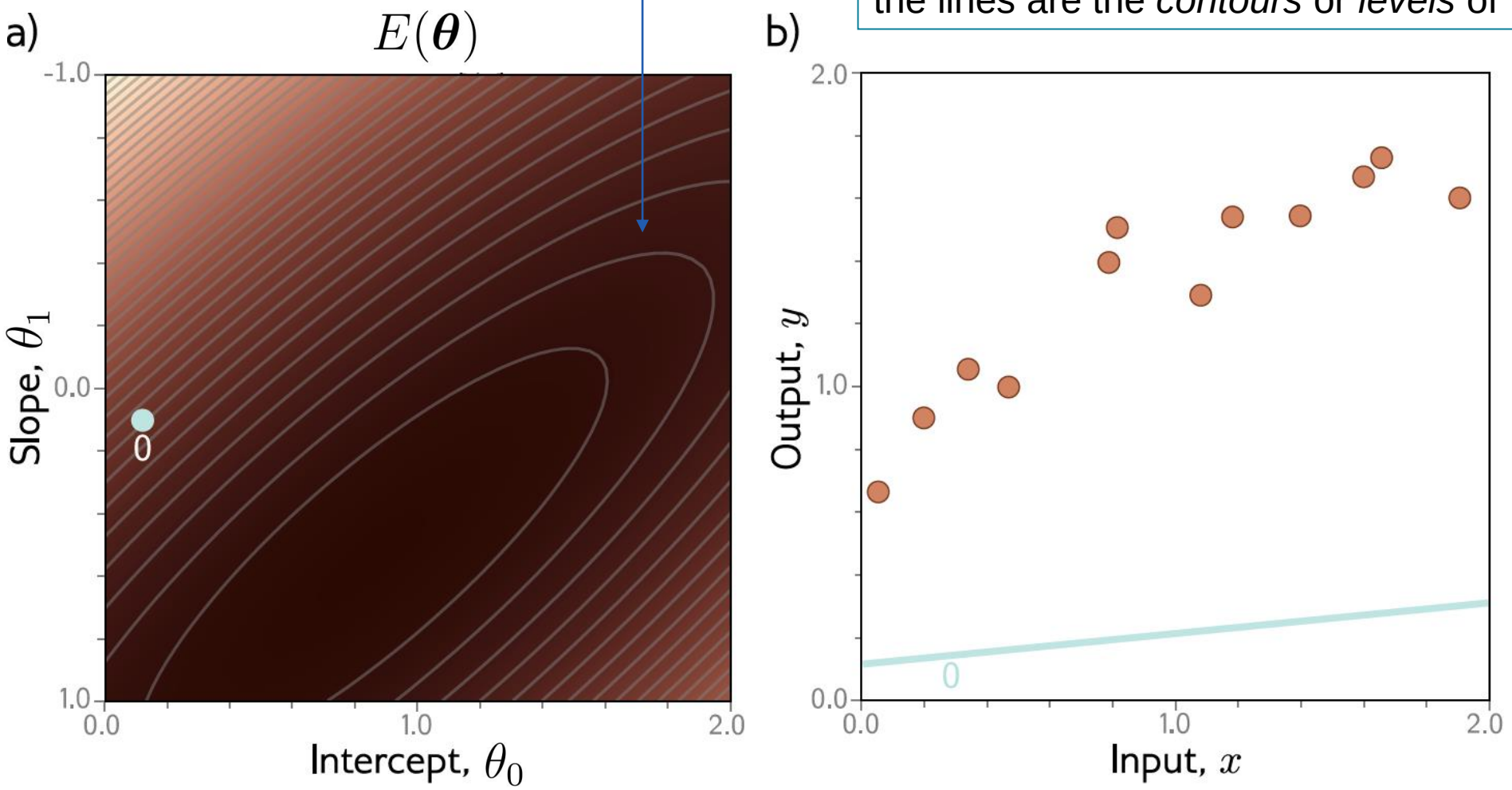
From , S.J.D. Prince (2023), Understanding Deep Learning, MIT Press, Available at "http://udlbook.com"

To minimize the empirical risk we can follow two approaches:

1. **"Brute force"** – there exists a closed solution to the minimization problem
2. **Stepwise approach** that approximates the closed solution in a finite number of steps

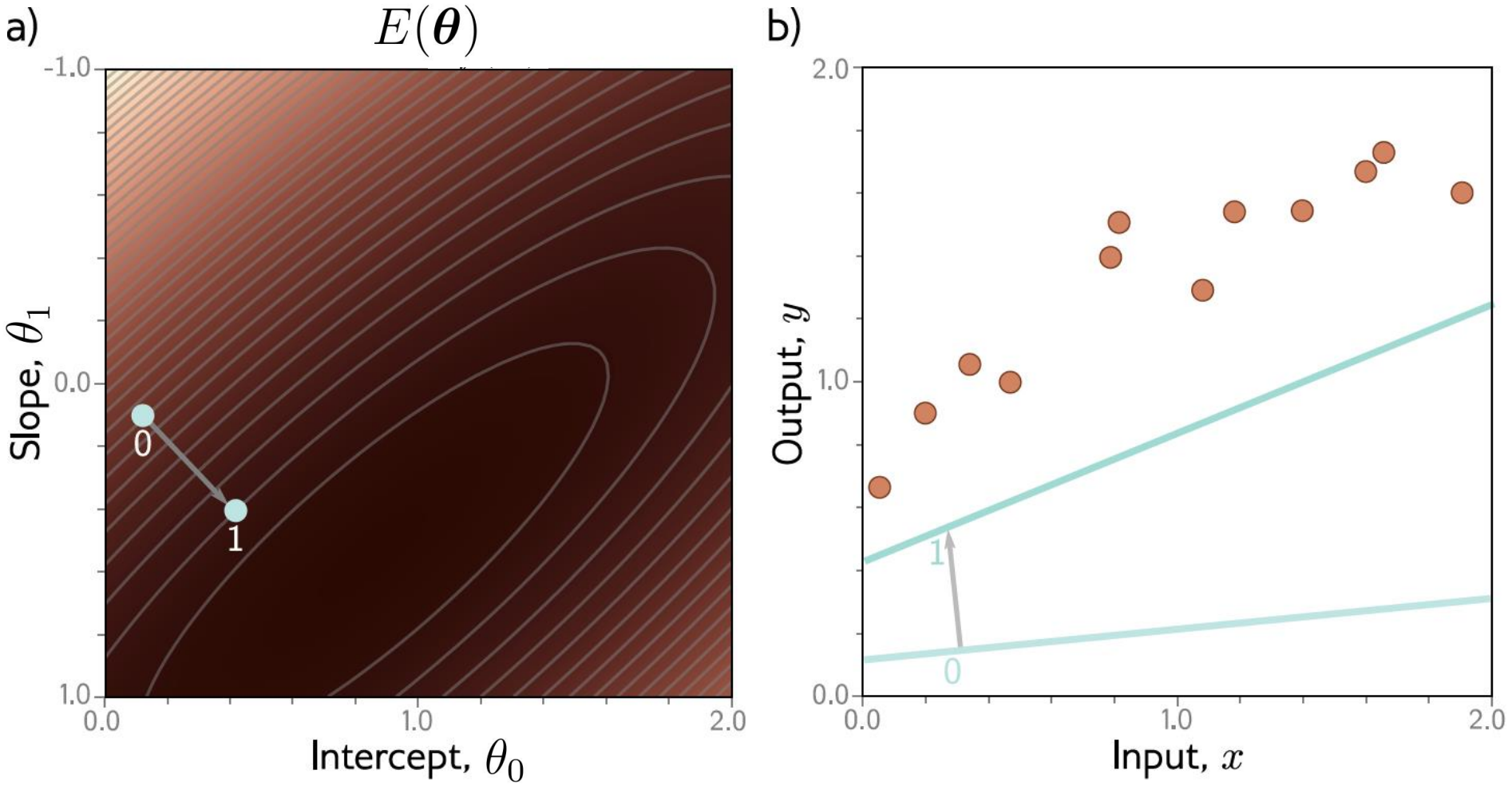
Let us train a machine learning model
Toy example: linear regression in one variable

the lines are the *contours* or *levels* of $E(\boldsymbol{\theta})$



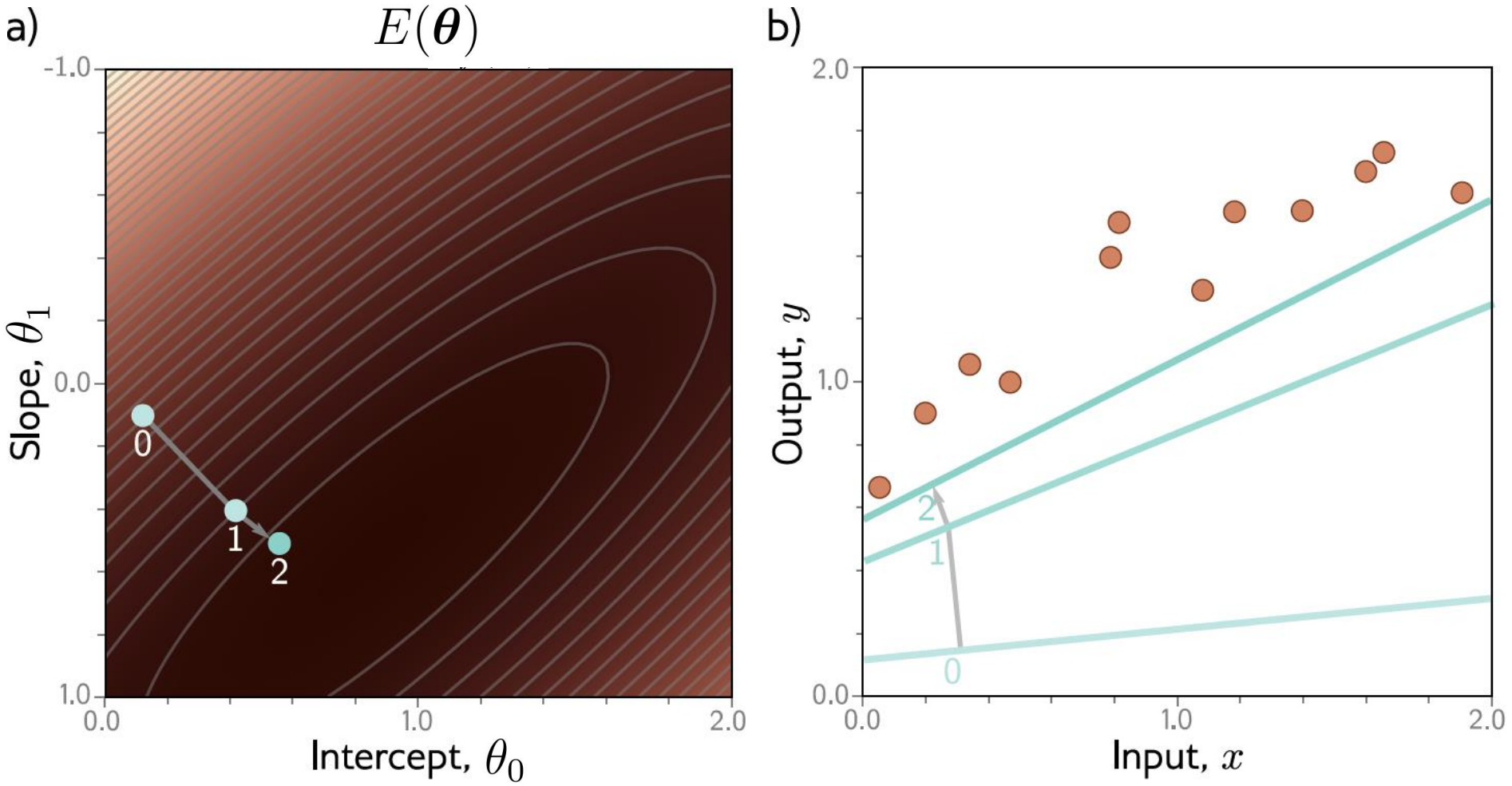
From , S.J.D. Prince (2023), Understanding Deep Learning, MIT Press, Available at "http://udlbook.com"

Let us train a machine learning model
Toy example: linear regression in one variable



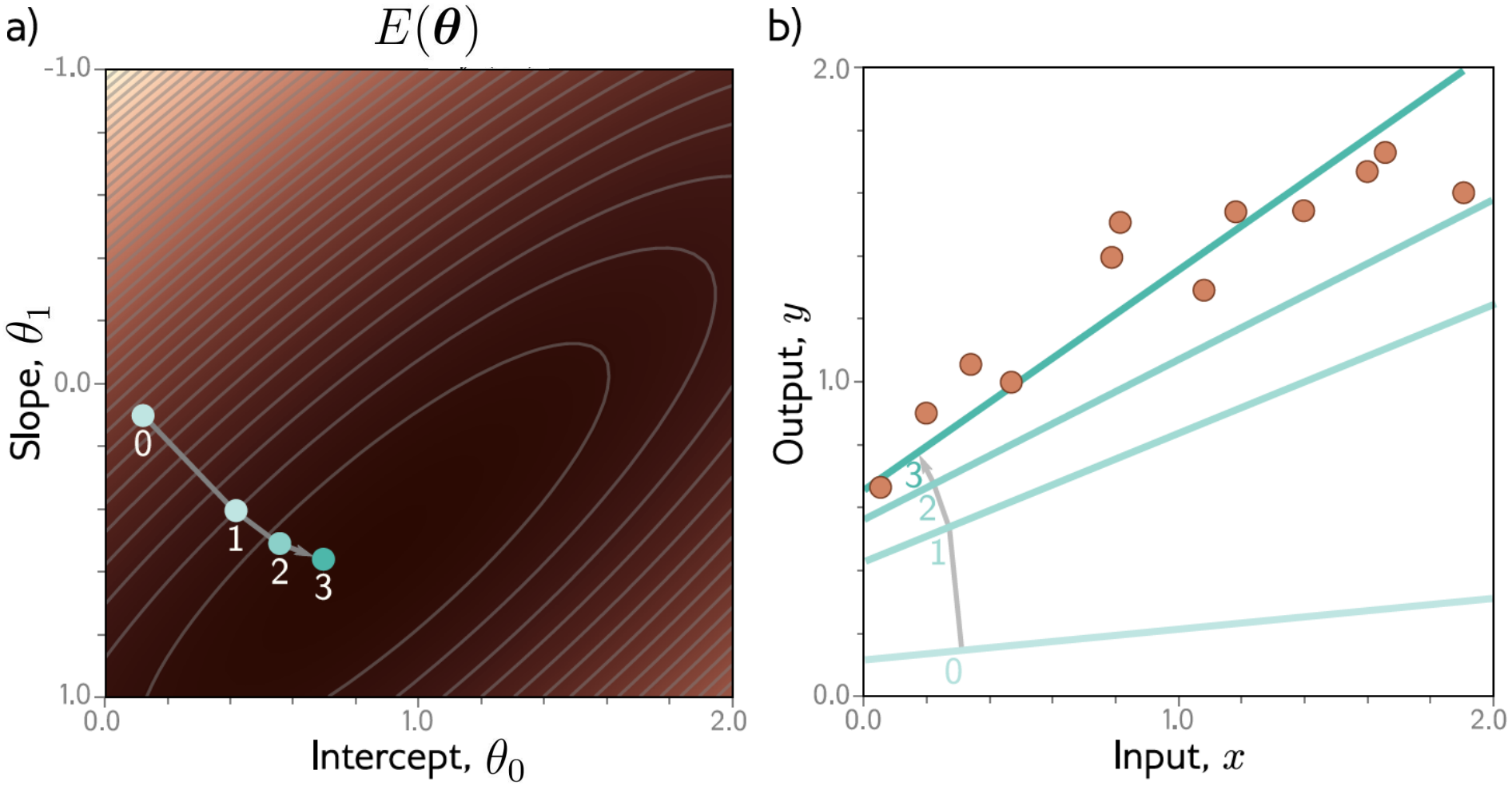
From , S.J.D. Prince (2023), Understanding Deep Learning, MIT Press, Available at "http://udlbook.com"

Let us train a machine learning model
Toy example: linear regression in one variable



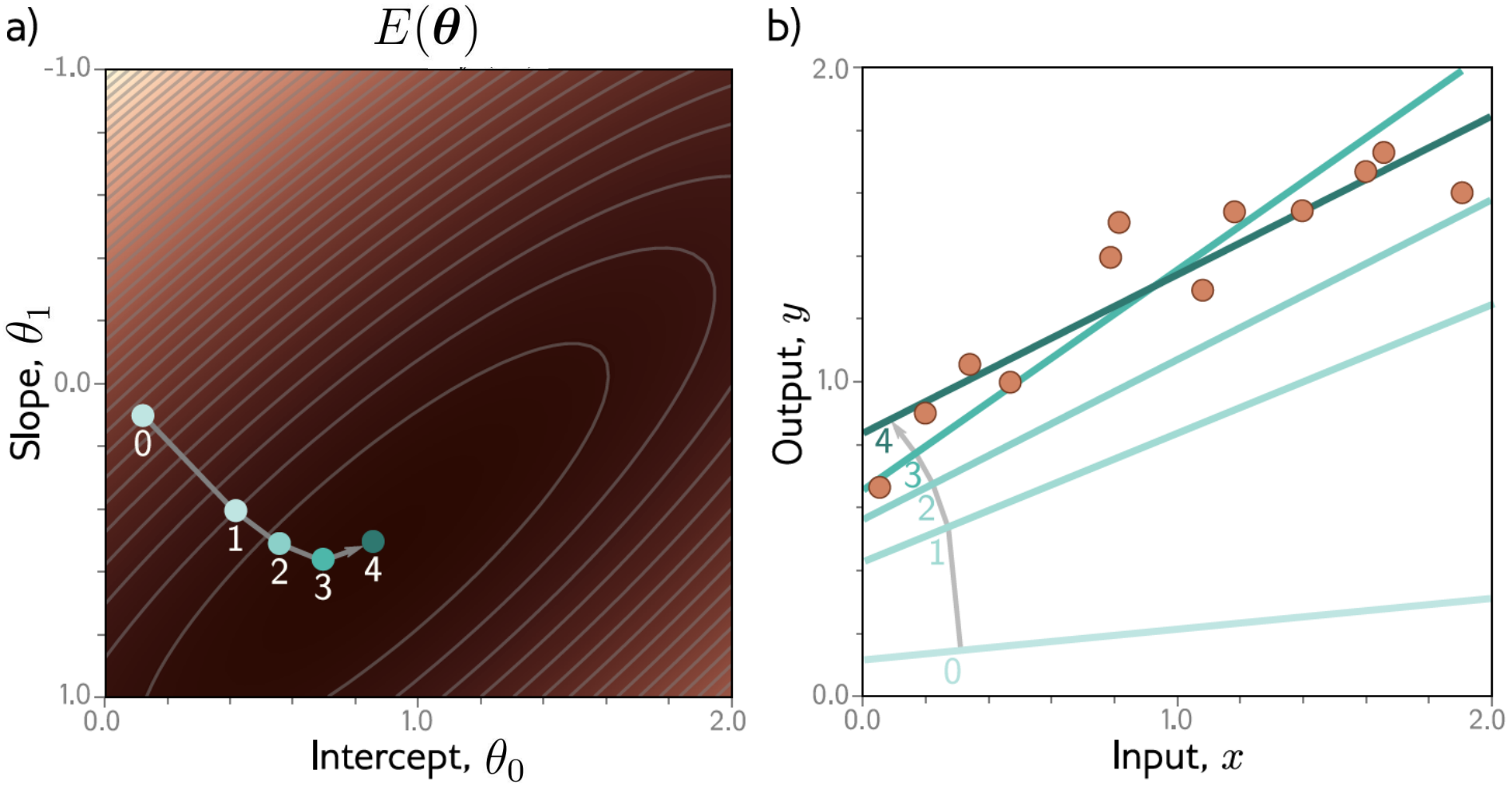
From , S.J.D. Prince (2023), Understanding Deep Learning, MIT Press, Available at "http://udlbook.com"

Let us train a machine learning model
Toy example: linear regression in one variable



From , S.J.D. Prince (2023), Understanding Deep Learning, MIT Press, Available at "http://udlbook.com"

Let us train a machine learning model
Toy example: linear regression in one variable



From , S.J.D. Prince (2023), Understanding Deep Learning, MIT Press, Available at "http://udlbook.com"

Go to our Moodle page and download the file under
Notebooks and Colab Instructions/1_Lecture_March_08_2024

Google Colab: `lin_regr.ipynb` (PART 1)

Linear regression: Key takeaways

I	**Fundamentals**: Linear regression is an ideal starting point for understanding supervised learning, loss functions, and the bias-variance tradeoff.
II	**Interpretability**: The model's coefficients provide clear insights into the relationship between features and the target variable, useful for interpretative analysis in various domains.
III	**Regularization Introduction**: Linear regression introduces regularization concepts like Ridge and Lasso, crucial for handling overfitting and feature selection.
IV	**Advanced Models Foundation**: It lays the groundwork for understanding more complex models, demonstrating basic principles applicable to advanced algorithms like neural networks.

Self-Study: Closing with linear regression

Self-Study

From Moodle download the book:

James, G., Witten, D., Hastie, T., & Tibshirani, R. (2013). *An introduction to statistical learning* (Vol. 112, p. 18). New York: Springer.

- **Section 3.1.3** Assessing the Accuracy of the Model
- **Section 3.2.2** Some Important Questions
- **Section 3.3.3** Potential Problems

Machine Learning Methods (Part 2)

- (Stochastic) gradient descent
- Logistic regression

(Stochastic) gradient descent

Supervised Learning

In applications, we use training data and we try to minimize the empirical risk

From last week

Training data

1. $\{(x_i, y_i)\}_{i=1}^m$ i.i.d. realizations of (X, Y)
2. $L : \mathcal{Y} \times \mathcal{Y} \to \mathbb{R}_+$
3. $f_{\boldsymbol{\theta}} \in \mathcal{H}$, where $f_{\boldsymbol{\theta}} : \mathcal{X} \to \mathcal{Y}$

Empirical risk minimization

Find $f_{\boldsymbol{\theta}} \in \mathcal{H}$ that minimizes the **empirical risk functional**

$$E(\boldsymbol{\theta}) = \frac{1}{m} \sum_{i=1}^{m} L(f_{\boldsymbol{\theta}}(x_i), y_i)$$

Supervised Learning

In applications, we use training data and we try to minimize the empirical risk

From last week

Training data

1. $\{(x_i, y_i)\}_{i=1}^m$ i.i.d. realizations of (X, Y)
2. $L : \mathcal{Y} \times \mathcal{Y} \to \mathbb{R}_+$
3. $f_{\boldsymbol{\theta}} \in \mathcal{H}$, where $f_{\boldsymbol{\theta}} : \mathcal{X} \to \mathcal{Y}$

1 This is a finite sum that can be computed by a machine, once we collect some training data and we choose a loss function and a hypothesis class.

Empirical risk minimization

Find $f_{\boldsymbol{\theta}} \in \mathcal{H}$ that minimizes the **empirical risk functional**

$$E(\boldsymbol{\theta}) = \frac{1}{m} \sum_{i=1}^{m} L(f_{\boldsymbol{\theta}}(x_i), y_i)$$

2 To minimize the empirical risk functional is referred to "**learning**" or "**training**" the machine learning model(s) using the **training data**. There exist algorithms that allow solving this problem and compute the "learned/trained" machine learning model.

From last week

Supervised Learning: Summary

Given training data $\{(x_i, y_i)\}_{i=1}^m$, select a family $f_{\boldsymbol{\theta}} \in \mathcal{H}$ of machine learning models and a loss function L .

Then, minimizing the empirical risk functional $E(\boldsymbol{\theta})$, we arrive at the

(learned/trained model) $\hat{f} := f_{\hat{\boldsymbol{\theta}}}$ where $\hat{\boldsymbol{\theta}}$ minimizes $E(\boldsymbol{\theta})$.

And now?

Problem	We need to solve the empirical risk minimization problem to learn the parameters of the machine learning model
Solution	Let us introduce a method than can be implemented as a algorithm to do that

To understand how to minimize the empirical risk, we start focusing on the loss function

Let us start our journey by focusing on the loss function, which is seen as a function of the parameters $\boldsymbol{\theta}$.

$$E(\boldsymbol{\theta}) = \frac{1}{m} \sum_{i=1}^{m} L(f_{\boldsymbol{\theta}}(x_i), y_i)$$

Let us introduce the concept of (global) minimum of a function

"minimizer"

1

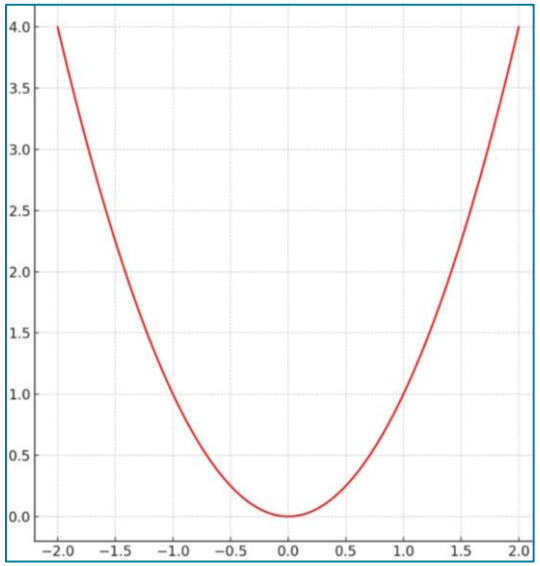
$f : X \subseteq \mathbb{R}^n \to \mathbb{R}$ has a (global) minimum point at x^* if $f(x) \geq f(x^*), \forall x \in X$

The value $f(x^*)$ is the (global) minimum of f

2

A function can have no (global) minimum point, one minimum point, or many minima points

?



?

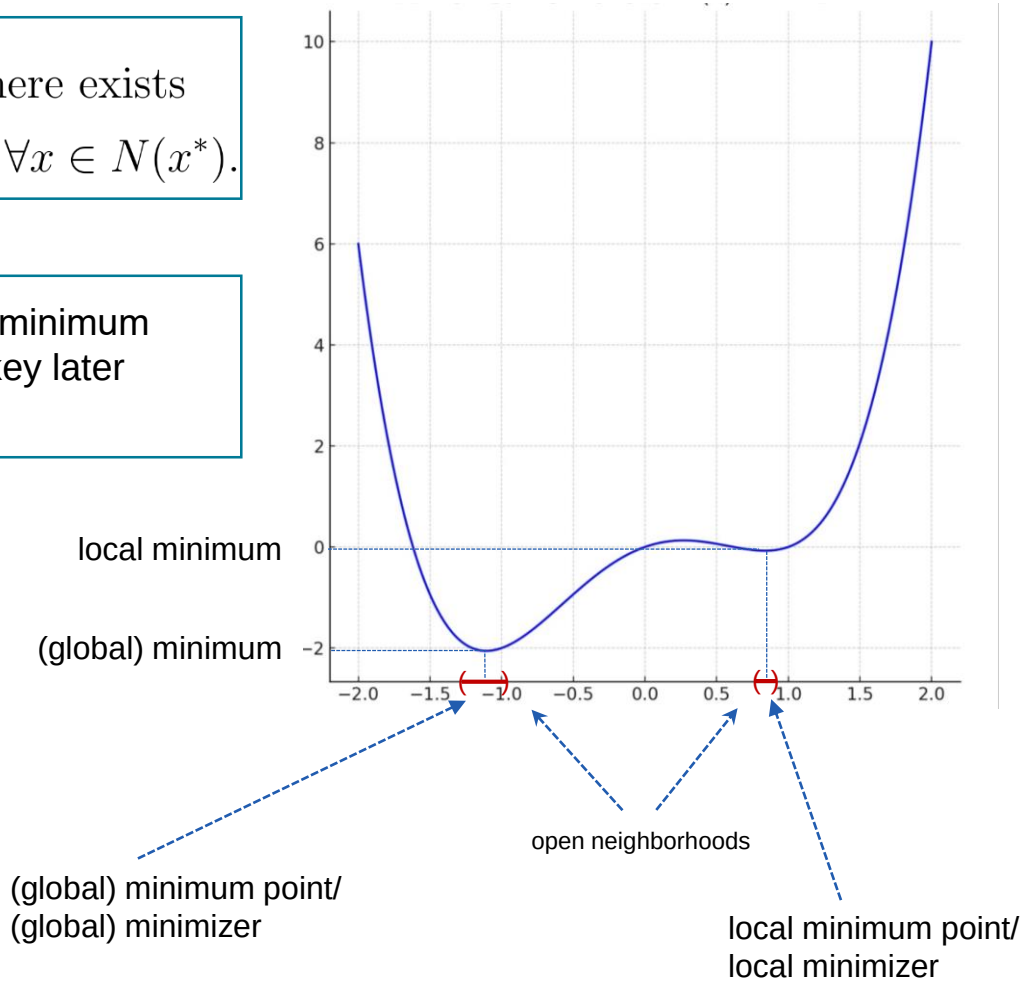
Important: there are also the local minima points

1

$f : X \subseteq \mathbb{R}^n \to \mathbb{R}$ has a local minimum point at x^* if there exists a local neighborhood $N(x^*)$ of x^* in $X : f(x) \geq f(x^*), \forall x \in N(x^*)$.

2

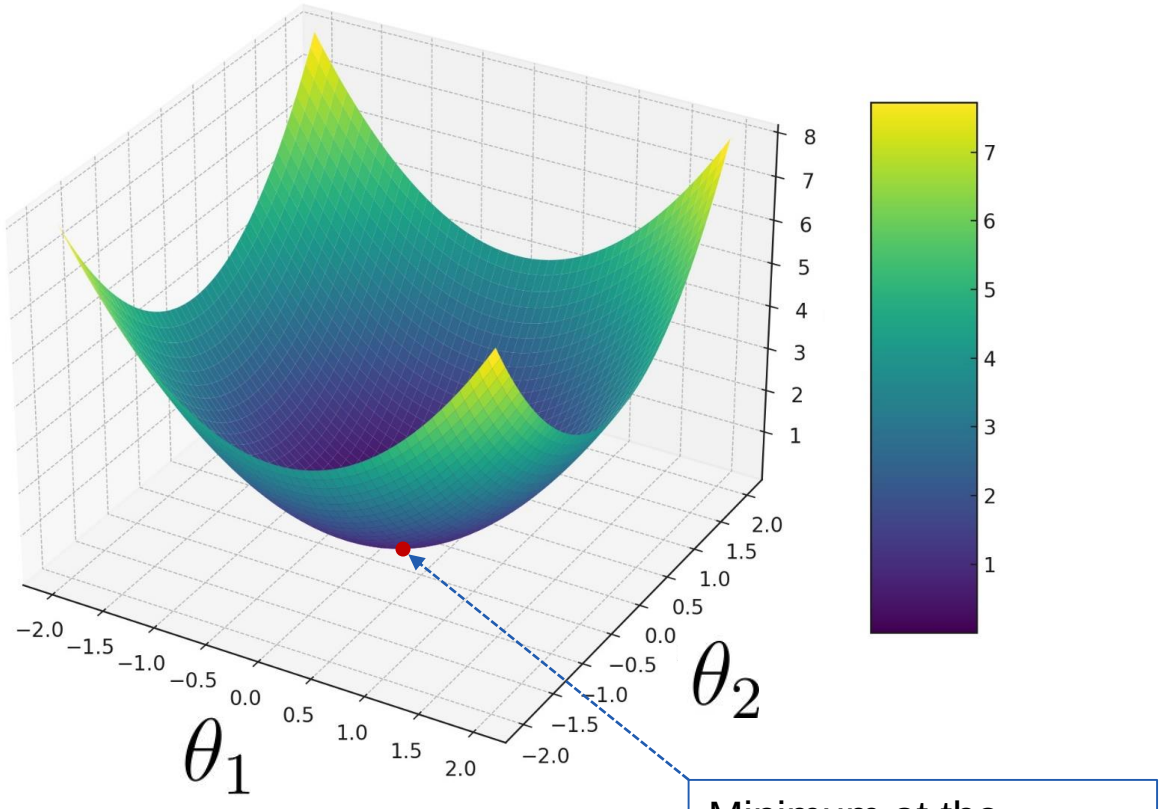
If the function “behaves well” in an open neighborhood of a local minimum point, then this point satisfies an important condition that will be key later (generalization of Fermat’s theorem)



With more parameters the situation does not improve

Two examples computed by ChatGPT 4

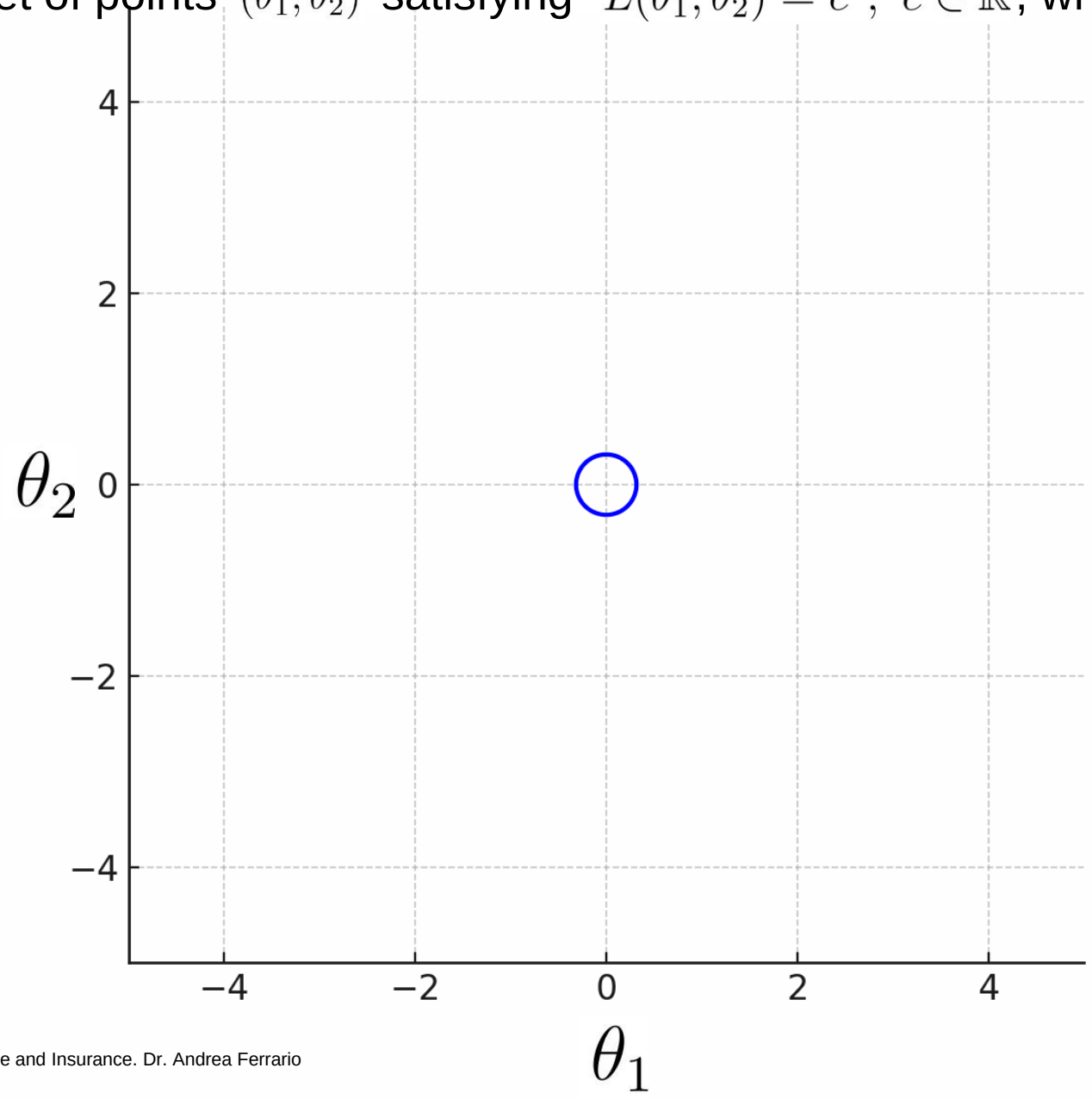
$$L(\theta_1, \theta_2) = \theta_1^2 + \theta_2^2$$



Minimum at the bottom of this “sink”

Contours or levels of a function – an animated example

Each circle is the set of points (θ_1, θ_2) satisfying $L(\theta_1, \theta_2) = c^2,\ c \in \mathbb{R}$, where $L(\theta_1, \theta_2) = \theta_1^2 + \theta_2^2$.

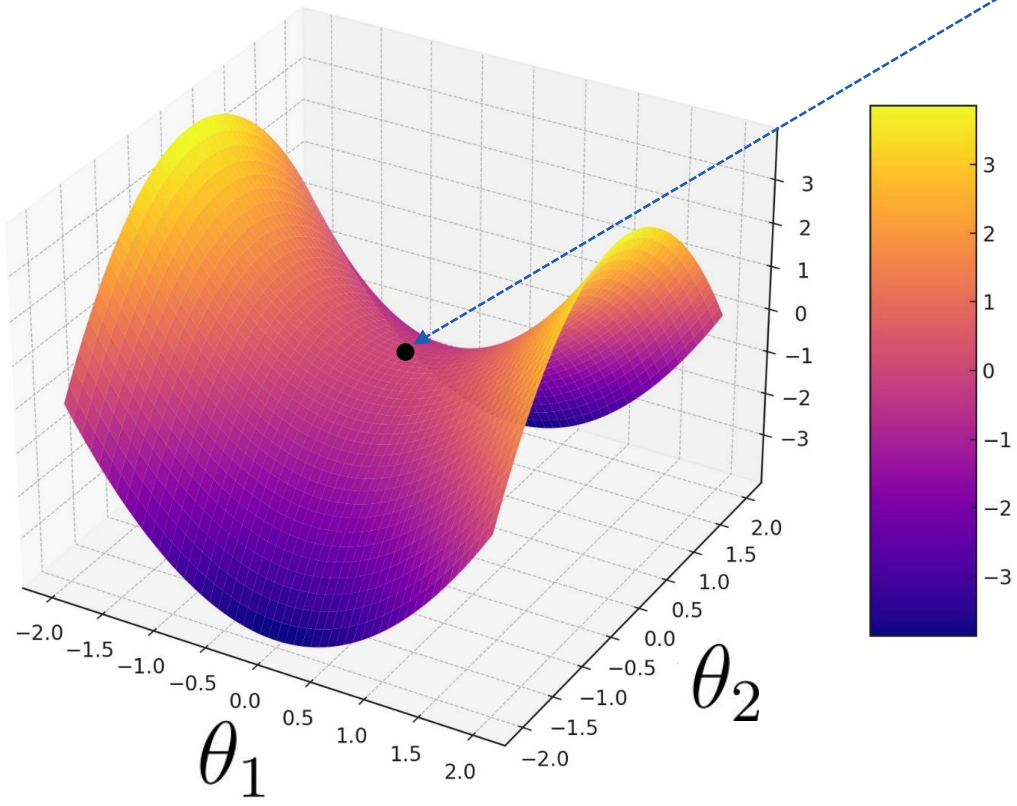


With more parameters the situation does not improve

Two examples computed by ChatGPT 4

Is this point a local minimum?

$$L(\theta_1, \theta_2) = \theta_1^2 - \theta_2^2$$



What are the contours of this function? Write a Python code to visualize them!

The problem: Minimizing a function

Idea: can we do it *analytically*?

Notation

$$\nabla L(\boldsymbol{\theta}) = \begin{pmatrix} \frac{\partial L}{\partial \theta_0} \\ \vdots \\ \frac{\partial L}{\partial \theta_d} \end{pmatrix}$$ gradient $$\boldsymbol{\theta} = \begin{pmatrix} \theta_0 \\ \vdots \\ \theta_d \end{pmatrix}$$

$$\frac{\partial L(\boldsymbol{\theta})}{\partial \theta_i} := \lim_{h \to 0} \frac{L(\theta_0, \ldots, \theta_i + h, \ldots, \theta_d) - L(\boldsymbol{\theta})}{h}$$ partial derivative

The gradient of a function is a vector of partial derivatives

The partial derivatives represent the rate of change of the function along each axis in the variable space

To compute $\frac{\partial L(\boldsymbol{\theta})}{\partial \theta_i}$ apply the rules of calculus

To compute the gradient of a function is to compute the partial derivatives

The problem: Minimizing a function

Idea: can we do it *analytically*?

Important result

If $\boldsymbol{\theta}^*$ is a local minimum point for $L(\boldsymbol{\theta})$ and $L(\boldsymbol{\theta})$ behaves well in a neighborhood of $\boldsymbol{\theta}^*$, then $\nabla L(\boldsymbol{\theta}^*) = 0$.

The result applied to empirical risk minimization

$$\nabla E(\boldsymbol{\theta}) = \frac{1}{m}\sum_{i=1}^{m} \nabla L(f_{\boldsymbol{\theta}}(x_i), y_i) = 0 \qquad (*)$$

We arrive at the set of equations (*) and then, based on the properties of the loss function, we try to find the local minima among all possible solutions of (*)

The problem: Minimizing a function

Idea: can we do it *analytically*? Now we can explain the normal equations.

Linear regression revisited

1

$$\nabla E(\boldsymbol{\theta}) = \frac{1}{m}\sum_{i=1}^{m} \nabla L(f_{\boldsymbol{\theta}}(x_i), y_i) = 0 \quad \Leftrightarrow$$

$$\frac{1}{m}\sum_{i=1}^{m} \nabla(\theta_0 + \theta_1 x_{i1} + \cdots + \theta_d x_{id} - y_i)^2 = 0 \quad \Leftrightarrow$$

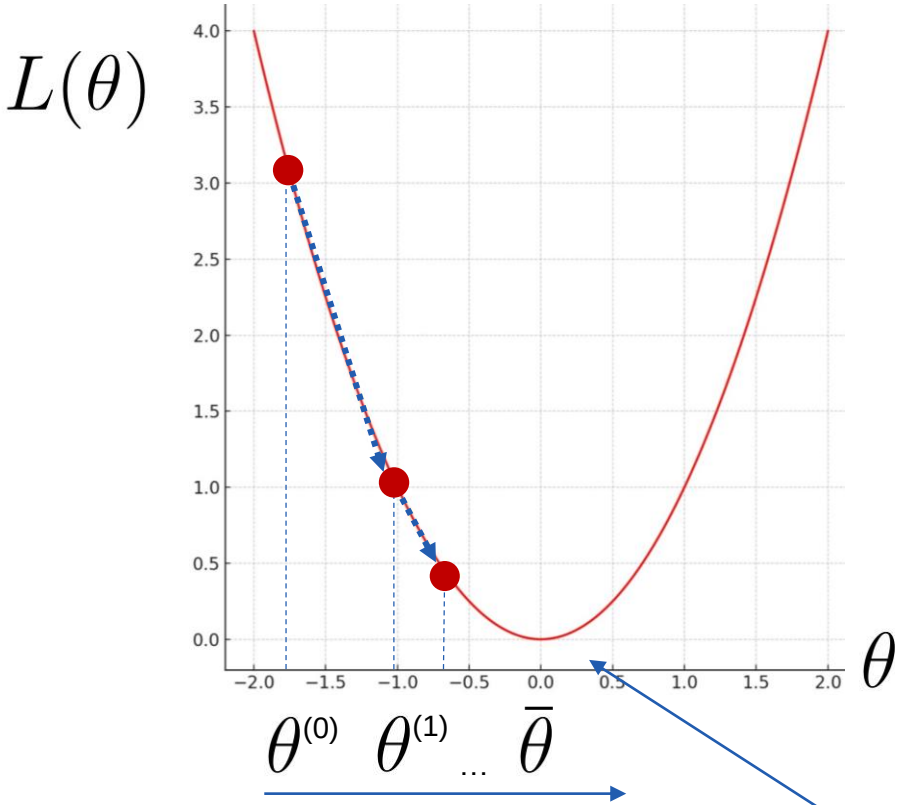
$$A^T A\boldsymbol{\theta} = A^T y \text{ (normal equations)}$$

2

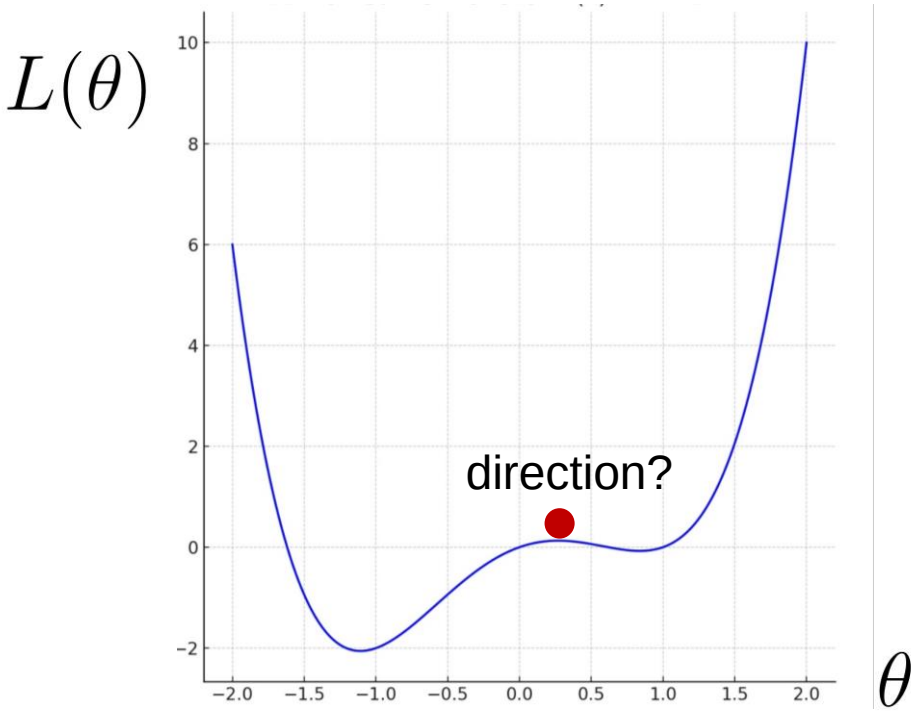
$E(\boldsymbol{\theta}) = \frac{1}{m}\|A\boldsymbol{\theta} - y\|_2^2$ is convex and behaves well (twice differentiable): then, all solutions of the normal equations are (global) minimizers of the empirical risk. On slide 12 we introduced sufficient conditions for the uniqueness of the minimum point.

The problem: Minimizing a function

Idea: what if we do it *iteratively*?



A sequence that may possibly converge to the (global) minimum



Can we introduce a method that allows approaching minimum point of this function iteratively? Can we translate it into an algorithm that a computer can execute?

Why do we even bother? The case of neural networks

We will discuss this in the forthcoming lectures

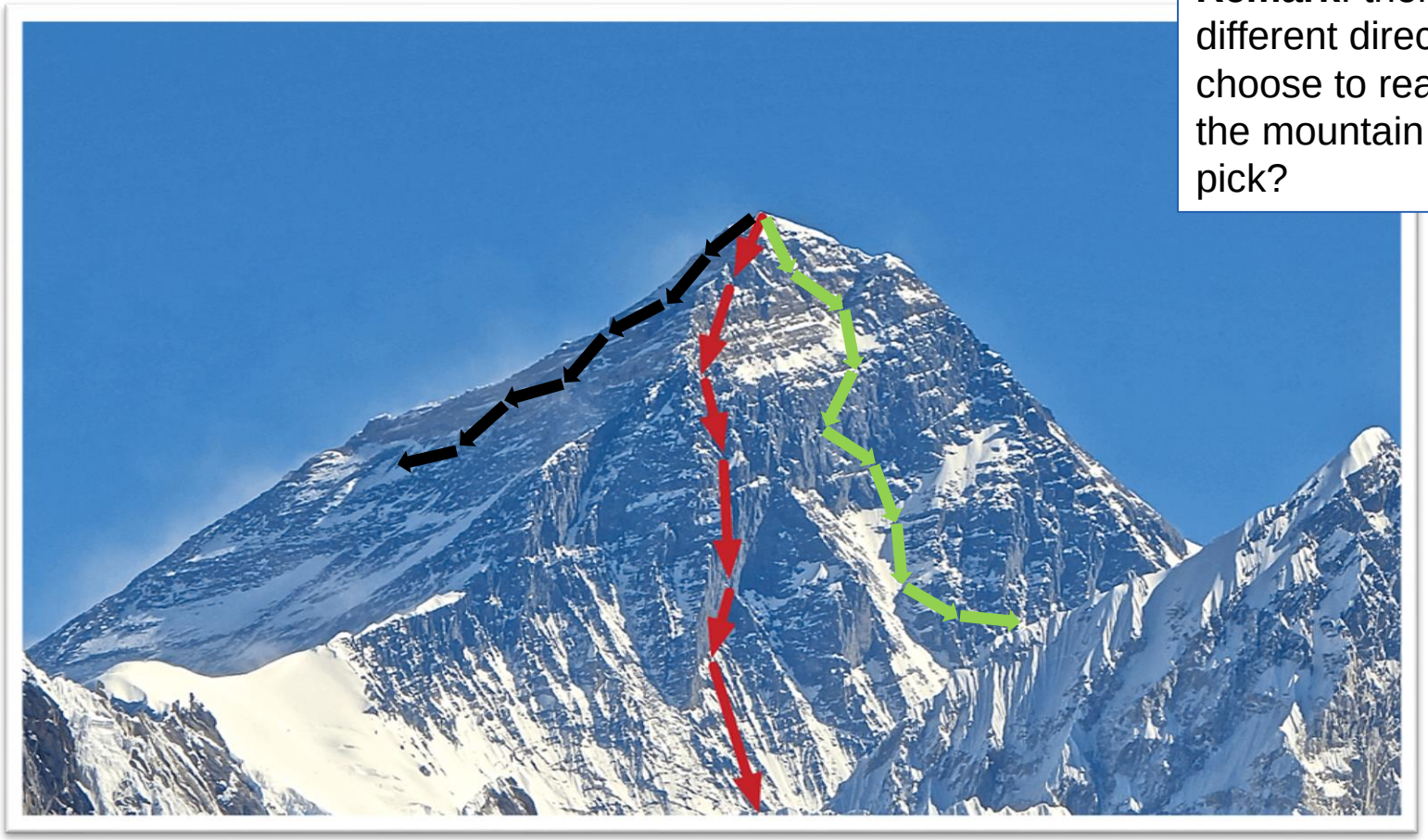
I	When utilizing artificial neural networks, the count of weights can easily reach into the millions
II	The loss functions are highly “non-convex”: vast number of local minima

Gradient descent: Ideas

I	Gradient descent is an optimization algorithm used for finding the minimum of a function iteratively
II	In machine learning, it's commonly used to minimize the empirical risk/loss, which is a function of the parameters of the machine learning model (remember the linear regression example)
III	It is rather efficient and can be used with a wide range of functions and machine learning models
IV	It is commonly implemented in computer programs (e.g., Python) and can be extended to a stochastic version, that is used in deep learning

Core idea: if the gradient of a function gives me the direction of its greatest increase, then I can use “minus the gradient” to follow the direction of greatest decrease…

Remark: there are many different directions we can choose to reach the bottom of the mountain – which one to pick?



Source: https://medium.com/@edwardpie/simplified-gradient-descent-with-heuristics-8971ee3bec52

Gradient descent algorithm for empirical risk minimization

GD algorithm

1. Choose a learning rate $\eta > 0$

2. Choose the max. number of steps $t_{\max}$

3. Initialize parameters (randomly or deterministically) $\boldsymbol{\theta}^{(0)} \in \mathbb{R}^{d+1}$

4. Update parameters at each step as follows, until $t_{\max}$

$$\boldsymbol{\theta}^{(t+1)} = \boldsymbol{\theta}^{(t)} - \eta \nabla E(\boldsymbol{\theta}) =$$

$$\boldsymbol{\theta}^{(t)} - \eta \frac{1}{m} \sum_{i=1}^{m} \underbrace{\nabla L(f_{\boldsymbol{\theta}}(x_i), y_i)}_{\nabla L(\boldsymbol{\theta})}$$

Gradient descent algorithm for empirical risk minimization

GD algorithm

1. Choose a learning rate

2. Choose the max. number of steps

3. Initialize parameters (randomly or deterministically)

4. Update parameters at each step as follows, until $t_{\max}$

$\eta > 0$

$t_{\max}$

$\boldsymbol{\theta}^{(0)} \in \mathbb{R}^{d+1}$

$$\boldsymbol{\theta}^{(t+1)} = \boldsymbol{\theta}^{(t)} - \eta \nabla E(\boldsymbol{\theta}) =$$

$$\boldsymbol{\theta}^{(t)} - \eta \frac{1}{m} \sum_{i=1}^{m} \underbrace{\nabla L(f_{\boldsymbol{\theta}}(x_i), y_i)}_{\nabla L(\boldsymbol{\theta})}$$

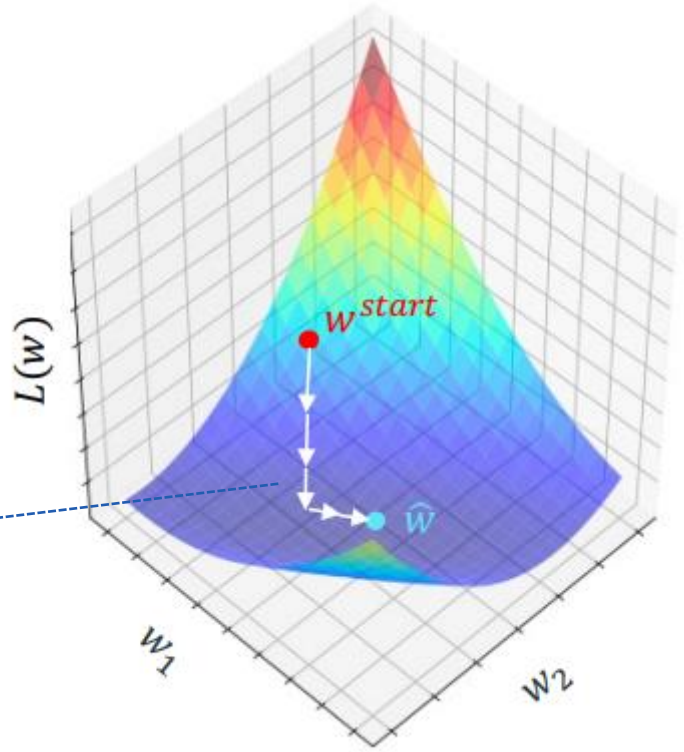
1. Too small: slow progress
Too large: oscillations
Heuristics: between 0.001 and 0.1

2. GD updates the weights/parameters in the direction of the **fastest decrease** of the empirical risk function

We have to compute the gradients of the loss function w.r.t. the *d* parameters for each of the *m* data points

Gradient descent: a visual example

A “good” example of gradient descent.

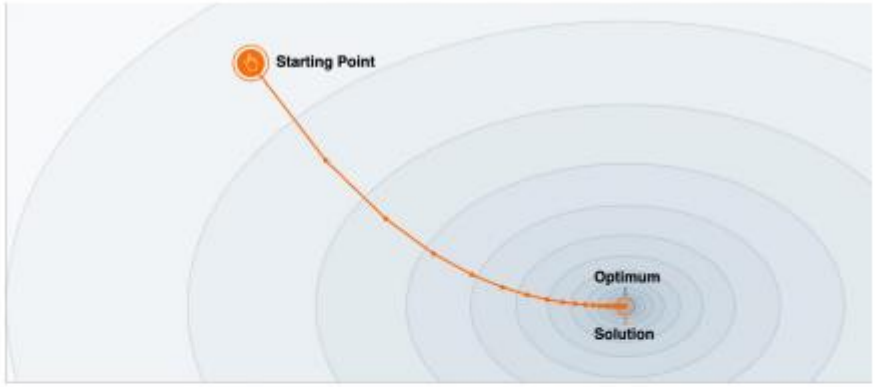


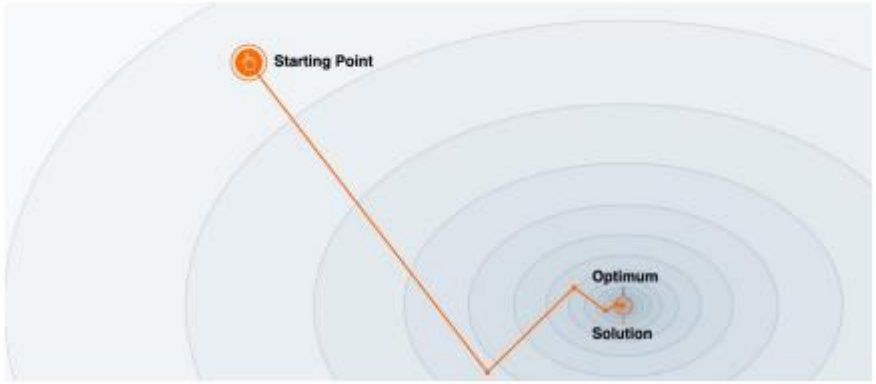
Source: A. Krause and F. Yang. Introduction to Machine Learning (2024).

In general, what could go wrong with gradient descent?

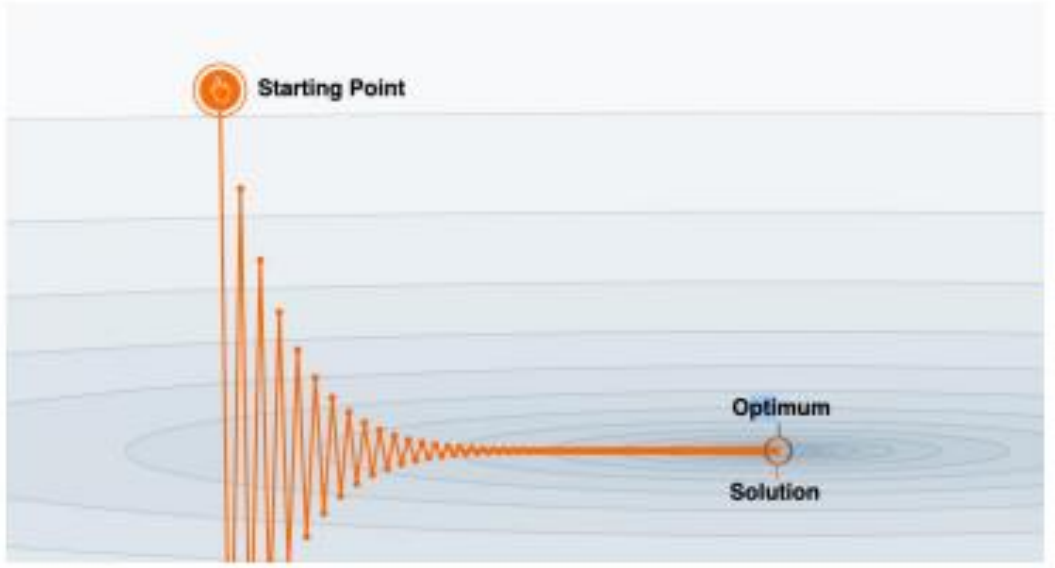
Depending on the learning rate, the GD algorithm behaves quite differently

A “good” convergence rate





Quick convergence rate



Strong oscillations

Source: A. Krause and F. Yang. Introduction to Machine Learning (2024).

Gradient descent algorithm in one dimensions: a nice animation

Check: https://github.com/jermwatt/machine_learning_refined

A nice animation

The main challenge with GD: it does not scale

I	Gradient descent requires computing the gradients w.r.t. all parameters – what if the parameters are the millions (or more)?
II	Gradient descent requires computing the gradients at each data sample – what if the training data samples are in the millions (or more)?
III	The computational cost for the operations in I and II (and others, for deep learning methods) is too high

A solution: Stochastic gradient descent (SGD)

I	SGD computes the gradient and updates parameters using only **one data point (or a small batch) at a time**
II	Much faster per iteration and more memory-efficient, especially useful for large datasets
III	Introduces randomness in the optimization process, which can help escape local minima in non-convex problems. However, it requires a good selection of learning rates.

GD vs. SGD
Overview of SGD

Algorithm 8.1 Stochastic gradient descent (SGD) update

Require: Learning rate schedule $\epsilon_1, \epsilon_2, \ldots$

Require: Initial parameter $\boldsymbol{\theta}$

$k \leftarrow 1$

while stopping criterion not met **do**

Sample a minibatch of m examples from the training set $\{\boldsymbol{x}^{(1)}, \ldots, \boldsymbol{x}^{(m)}\}$ with corresponding targets $\boldsymbol{y}^{(i)}$.

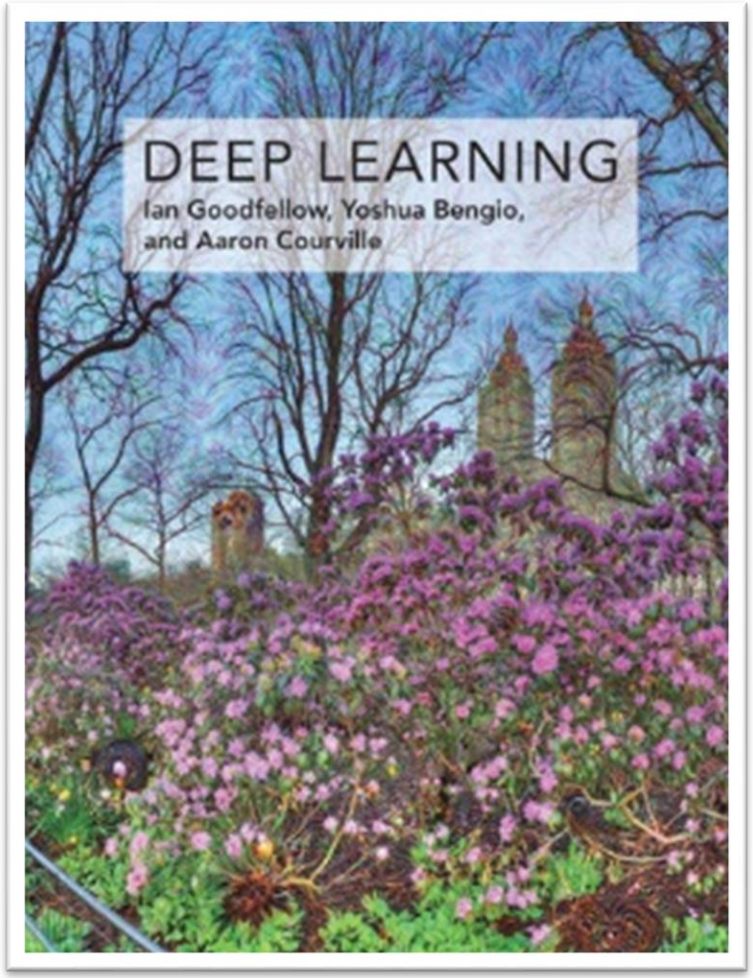
Compute gradient estimate: $\hat{\boldsymbol{g}} \leftarrow \frac{1}{m} \nabla_{\boldsymbol{\theta}} \sum_i L(f(\boldsymbol{x}^{(i)}; \boldsymbol{\theta}), \boldsymbol{y}^{(i)})$

Apply update: $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} - \epsilon_k \hat{\boldsymbol{g}}$

$k \leftarrow k + 1$

end while

Source: Goodfellow, I., Bengio, Y., & Courville, A. (2016). Deep learning. MIT press. (section 8.3.1)



Goodfellow, I., Bengio, Y., & Courville, A. (2016). Deep learning. MIT press.

GD vs. SGD
Overview of SGD

2 We can implement a criterion that makes us stop if there is no "substantial gain" from updating parameters – we will see this later

1 Goodfellow et al. suggest that "[i]n practice, it is common to decay the learning rate linearly until iteration τ . […] After iteration τ it is common to leave [the learning rate] constant." (pag. 287)

Algorithm 8.1 Stochastic gradient descent (SGD) update

Require: Learning rate schedule $\epsilon_1, \epsilon_2, .$!

Require: Initial parameter $\boldsymbol{\theta}$

$k \leftarrow 1$

while stopping criterion not met **do**

Sample a minibatch of m examples from the training set $\{\boldsymbol{x}^{(1)}, \ldots, \boldsymbol{x}^{(m)}\}$ with corresponding targets $\boldsymbol{y}^{(i)}$.

Compute gradient estimate: $\hat{\boldsymbol{g}} \leftarrow \frac{1}{m} \nabla_{\boldsymbol{\theta}} \sum_i L(f(\boldsymbol{x}^{(i)}; \boldsymbol{\theta}), \boldsymbol{y}^{(i)})$

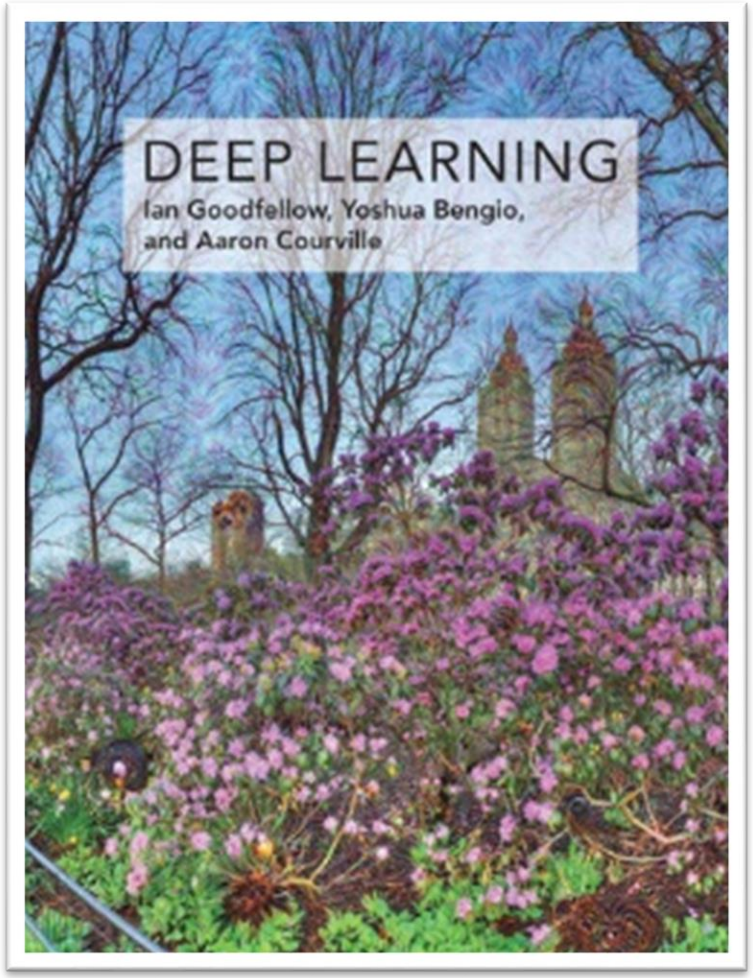
Apply update: $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} - \epsilon_k \hat{\boldsymbol{g}}$

$k \leftarrow k + 1$

end while

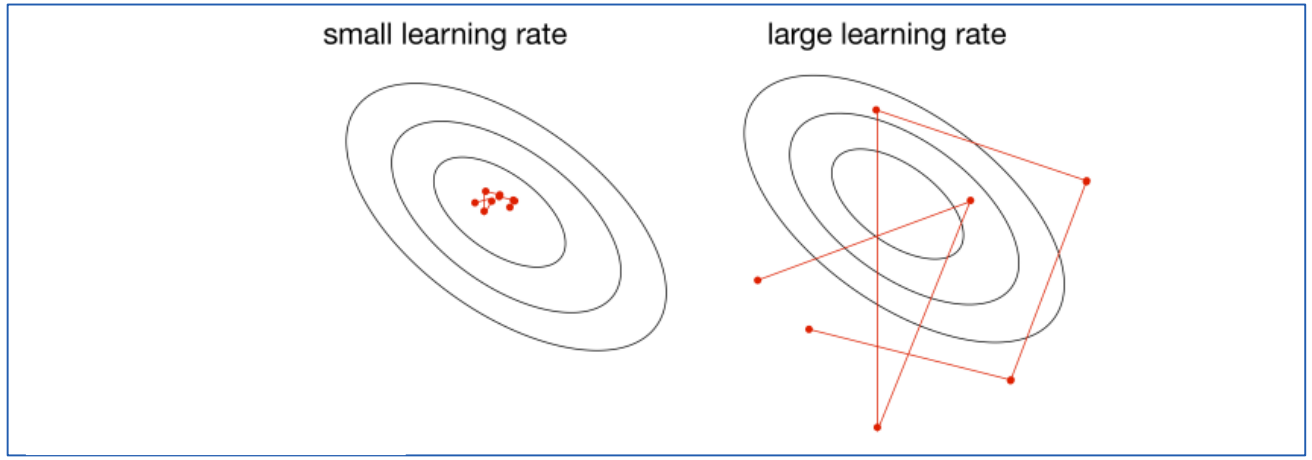
Source: Goodfellow, I., Bengio, Y., & Courville, A. (2016). Deep learning. MIT press. (section 8.3.1)

3 The sum is over the *m* examples of the minibatch

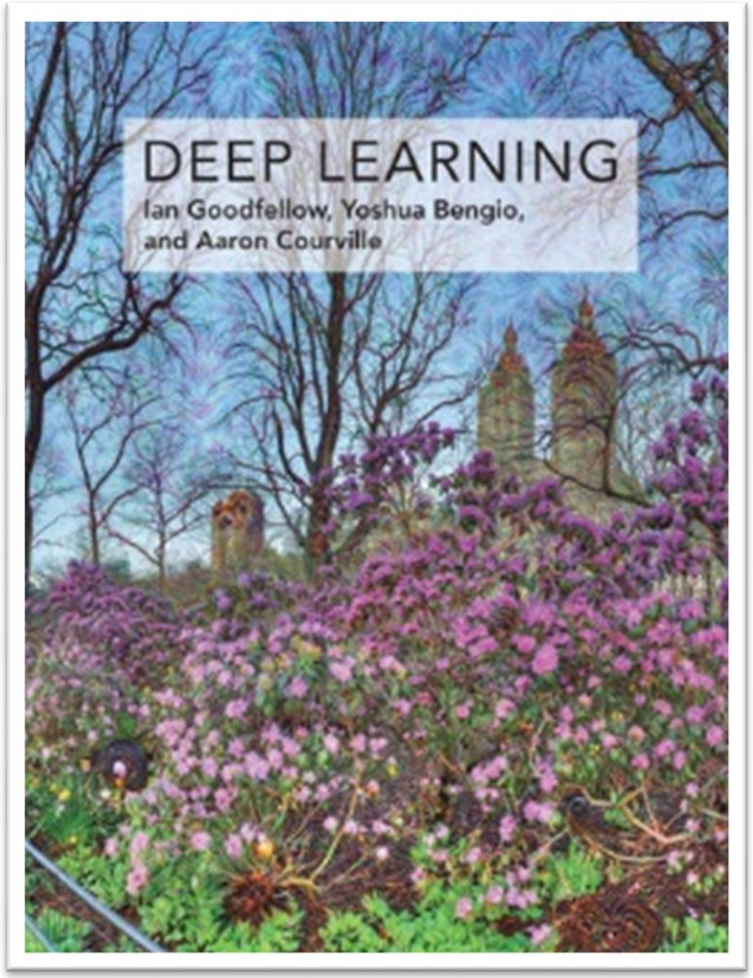


Goodfellow, I., Bengio, Y., & Courville, A. (2016). Deep learning. MIT press.

GD vs. SGD
Overview of SGD



Source: R. Grosse et al. “CSC 311: Machine Learning Lecture 2 - Linear Methods for Regression, Optimization” Available at: https://www.cs.toronto.edu/~rgrosse/courses/csc311_f20/slides/lec02.pdf



Goodfellow, I., Bengio, Y., & Courville, A. (2016). Deep learning. MIT press.

(S)GD: Key takeaways

I	**Finding minima of the empirical risk** is a key task of machine learning modeling
II	**(Stochastic) gradient** descent are first-order methods, they aim to find local minima by computing partial derivatives of the loss function for all data points, mini-batches of data points or just one data point
III	**(S)GD are easy to implement** but exhibit difficulties for nonconvex functions with multiple local minima or plateaus. They also require tuning the learning rate parameter
IV	**Several methods using (S)GD** have been implemented and are used in the context of deep learning (we will see this later)

Google Colab: `SGD.ipynb`

Self-Study: Closing with (S)GD

Self-Study

Access the online book:

Goodfellow, I., Bengio, Y., & Courville, A. (2016). Deep learning. MIT press.

(website: https://www.deeplearningbook.org/)

- **Section 4.3** Gradient-based Optimization
- **Section 8.3.1** Stochastic Gradient Descent

Self-Study: Closing with (S)GD

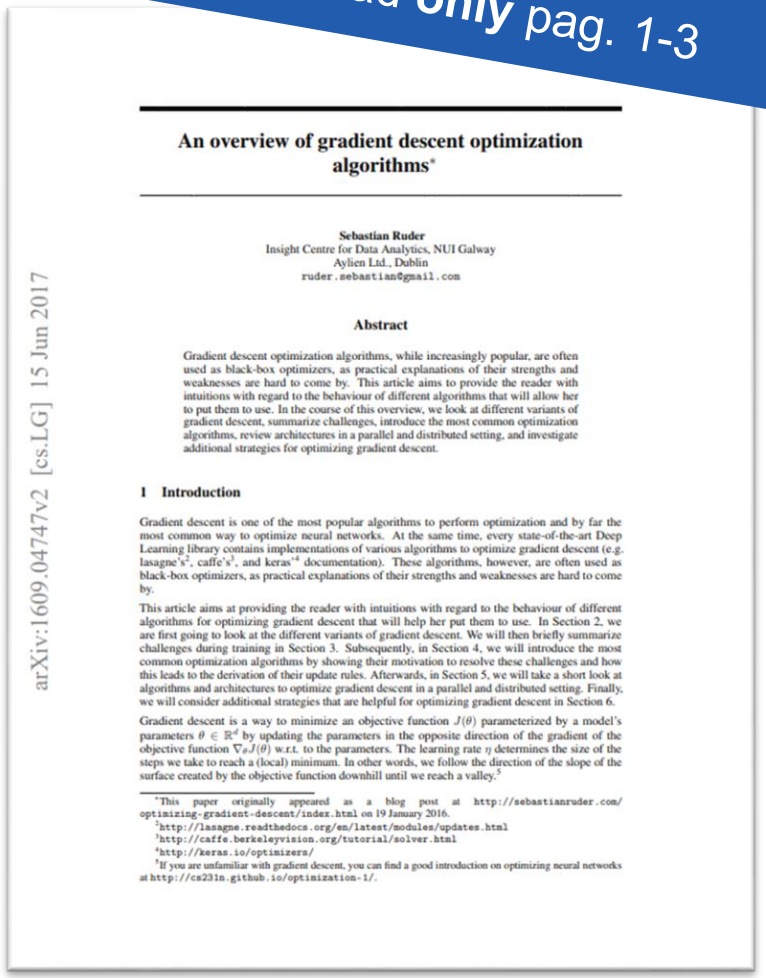
Self-Study

Access the online preprint (8996 citations as of Jan. 2024!):

https://arxiv.org/pdf/1609.04747.pdf

Ruder, S. (2016). An overview of gradient descent optimization algorithms. *arXiv preprint arXiv:1609.04747*.

Read **only** pag. 1-3



arXiv:1609.04747v2 [cs.LG] 15 Jun 2017

An overview of gradient descent optimization algorithms*

Sebastian Ruder
Insight Centre for Data Analytics, NUI Galway
Aylien Ltd., Dublin
ruder.sebastian@gmail.com

Abstract

Gradient descent optimization algorithms, while increasingly popular, are often used as black-box optimizers, as practical explanations of their strengths and weaknesses are hard to come by. This article aims to provide the reader with intuitions with regard to the behaviour of different algorithms that will allow her to put them to use. In the course of this overview, we look at different variants of gradient descent, summarize challenges, introduce the most common optimization algorithms, review architectures in a parallel and distributed setting, and investigate additional strategies for optimizing gradient descent.

1 Introduction

Gradient descent is one of the most popular algorithms to perform optimization and by far the most common way to optimize neural networks. At the same time, every state-of-the-art Deep Learning library contains implementations of various algorithms to optimize gradient descent (e.g. lasagne's[2], caffe's[3], and keras'[4] documentation). These algorithms, however, are often used as black-box optimizers, as practical explanations of their strengths and weaknesses are hard to come by.

This article aims at providing the reader with intuitions with regard to the behaviour of different algorithms for optimizing gradient descent that will help her put them to use. In Section 2, we are first going to look at the different variants of gradient descent. We will then briefly summarize challenges during training in Section 3. Subsequently, in Section 4, we will introduce the most common optimization algorithms by showing their motivation to resolve these challenges and how this leads to the derivation of their update rules. Afterwards, in Section 5, we will take a short look at algorithms and architectures to optimize gradient descent in a parallel and distributed setting. Finally, we will consider additional strategies that are helpful for optimizing gradient descent in Section 6.

Gradient descent is a way to minimize an objective function $J(\theta)$ parameterized by a model's parameters $\theta \in \mathbb{R}^d$ by updating the parameters in the opposite direction of the gradient of the objective function $\nabla_\theta J(\theta)$ w.r.t. to the parameters. The learning rate η determines the size of the steps we take to reach a (local) minimum. In other words, we follow the direction of the slope of the surface created by the objective function downhill until we reach a valley.[5]

*This paper originally appeared as a blog post at http://sebastianruder.com/optimizing-gradient-descent/index.html on 19 January 2016.
[2]http://lasagne.readthedocs.org/en/latest/modules/updates.html
[3]http://caffe.berkeleyvision.org/tutorial/solver.html
[4]http://keras.io/optimizers/
[5]If you are unfamiliar with gradient descent, you can find a good introduction on optimizing neural networks at http://cs231n.github.io/optimization-1/.

Logistic regression

Logistic regression: ideas

I	Logistic regression is specifically designed for binary classification problems, where the goal is to categorize data into two distinct groups (e.g., spam and fraud detection)
II	It provides probabilities for class membership, offering a measure of certainty about the classification, which is valuable in decision-making processes where understanding the likelihood of outcomes is crucial
III	It models the relationship between the features and the probability of a particular outcome, offering interpretable insights into how different predictors affect the odds of the target variable

Logistic regression
Notation

Model

A random variable Y is *Bernoulli distributed with parameter* $p \in [0,1]$ $(\text{Ber}(p))$ if

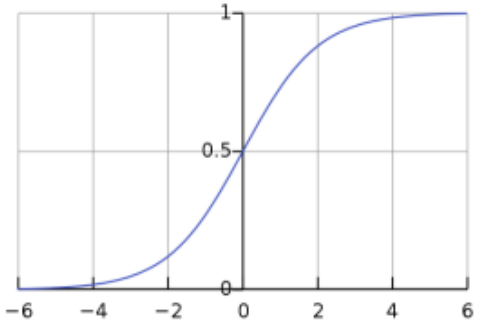
$$\begin{cases} \mathbb{P}[Y=1] = p \\ \mathbb{P}[Y=0] = 1-p \end{cases}$$

We are interested in an outcome rv with the Bernoulli distribution. The r.bv. Has two possible outcomes.

Model Let $Y \mid X \sim \text{Ber}(p(X))$, where

- $X = (X_1, \dots, X_d)$,
- $p(X) = \psi(\beta_0 + \beta_1 X_1 + \cdots + \beta_d X_d)$ and
- $\psi(x) = \dfrac{e^x}{e^x+1} = \dfrac{1}{1+e^{-x}}$

  (logistic function)



$$\begin{cases} \mathbb{P}[Y=1 \mid X] = p(X) \\ \mathbb{P}[Y=0 \mid X] = 1-p(X) \end{cases}$$

The logistic regression modeling states that the outcome r.v. conditioned on d+1 r.v. follows a Bernoulli distribution. The parameter of this distribution depends on the d+1 r.v.

Logistic regression
Notation

Empirical setting - preparation

- The *probability mass function* of a $\text{Ber}(p)$ random variable Y is given by

$$f_p(y) = \left\{ \begin{matrix} p & \text{if } y = 1 \\ 1-p & \text{if } y = 0 \end{matrix} \right\} = p^y(1-p)^{1-y}, \quad y \in \{0,1\}$$

- The *probability mass function* of m independent $\text{Ber}(p)$ random variables $Y_1, \ldots, Y_m$ is

$$\prod_{i=1}^{m} f_p(y_i) = \prod_{i=1}^{m} p^{y_i}(1-p)^{1-y_i}, \quad y_i \in \{0,1\}$$

- The *likelihood function* of $Y_1, \ldots, Y_m$ is $L(p \mid y_1, \ldots, y_m) = \prod_{i=1}^{m} f_p(y_i) = \prod_{i=1}^{m} p^{y_i}(1-p)^{1-y_i}$

- The *log-likelihood function* of $Y_1, \ldots, Y_m$ is

$$l(p \mid y_1, \ldots, y_m) = \log L(p \mid y_1, \ldots, y_m) = \sum_{i=1}^{m} \{y_i \log(p) + (1-y_i)\log(1-p)\}$$

We are interested in an outcome rv with Bernoulli distribution.

The logistic regression modeling states that the outcome r.v. conditioned on d+1 r.v. follows a Bernoulli distribution. The parameter of this distribution depends on the d+1 r.v.

Logistic regression
Empirical loss minimization

Empirical setting

- *Training Data:* let $(x_{i1}, \ldots, x_{id}, y_i)$, $i = 1, \ldots, m$, be iid realizations of $(X_1, \ldots, X_d, Y)$
- Set $x_{i0} = 1$, $x_i^T b = \sum_{j=0}^{d} x_{ij} b_j$, $p_i(b) = \psi(x_i^T b)$ for $b \in \mathbb{R}^{d+1}$ and $i = 1, \ldots, m$
- *Empirical loss minimization problem* *(conditional negative likelihood minimization)*

$$\min_{b \in \mathbb{R}^{d+1}} \sum_{i=1}^{m} \{-y_i \log(p_i(b)) + (y_i - 1) \log(1 - p_i(b))\} \qquad \textit{(cross-entropy loss} \geq 0\textit{)}$$

$$= \min_{b \in \mathbb{R}^{d+1}} \sum_{i=1}^{m} \{-y_i \log(\psi(x_i^T b)) + (y_i - 1) \log(1 - \psi(x_i^T b))\}$$

$$= \min_{b \in \mathbb{R}^{d+1}} \sum_{i=1}^{m} \left\{ y_i \log\left(\frac{1 - \psi(x_i^T b)}{\psi(x_i^T b)}\right) - \log(1 - \psi(x_i^T b)) \right\}$$

$$= \min_{b \in \mathbb{R}^{d+1}} \sum_{i=1}^{m} \left\{ \log\left(1 + e^{x_i^T b}\right) - y_i x_i^T b \right\}$$

- This is a convex minimization problem in $b \in \mathbb{R}^{d+1}$
- Can be solved with (S)GD

We are interested in an outcome rv with Bernoulli distribution.

The logistic regression modeling states that the outcome r.v. conditioned on d+1 r.v. follows a Bernoulli distribution. The parameter of this distribution depends on the d+1 r.v.

Logistic regression

Empirical loss minimization in **`sklearn`** is regularized/penalized **by default**

`sklearn.linear_model`.LogisticRegression

$$\min_{b\in\mathbb{R}^{d+1}} \sum_{i=1}^{m} \{-y_i \log(p_i(b)) + (y_i - 1)\log(1 - p_i(b))\} + r(b)$$

where

$$r(b) = 0 \qquad \textbf{No reg.}$$

$$r(b) = \|b\|_1,\ \|b\|_1 = \sum_{k=1}^{d+1} |b_k| \qquad \textbf{L1}$$

$$r(b) = \|b\|_2^2,\ \|b\|_2^2 = \sum_{k=1}^{d+1} b_k^2 \qquad \textbf{L2}$$

default

$$r(b) = \rho\|b\|^1 + \frac{1-\rho}{2}\|b\|_2^2 \qquad \textbf{Elastic Net}$$

Warning: the algorithm used to solve the minimization has to be compatible with the regularization

Is there an optimal value for ρ ? How to find it?

Logistic regression
Predicting probabilities and classes from new data points

Predictions

Solution of the empirical loss minimization problem: $\hat{\beta} = (\hat{\beta}_0, \ldots, \hat{\beta}_d)$

New data point: $x = (x_1, \ldots, x_d)$

Predicted probability that $Y = 1$ conditioned on $X = (x_1, \ldots, x_d)$: $\hat{p}(x) = \psi\left(\hat{\beta}_0 + \sum_{j=1}^{d} x_j \hat{\beta}_j\right)$

Binary Classification

- Choose a decision threshold $c \in (0, 1)$
- Predict Y to be $\hat{y}(x) = \begin{cases} 1 & \text{if } \hat{p}(x) \geq c \\ 0 & \text{if } \hat{p}(x) < c \end{cases}$

The learned/trained logistic regression model allows us computing the empirical probabilities AND a class/label, at the cost of introducing an arbitrary decision threshold

Logistic regression
Evaluating the performance of the estimated model

Performance measures for classification problems

Test Data: let $(x_{i1}, \ldots, x_{id}, y_i)$, $i = m+1, \ldots, m+n$, be iid realizations of $(X_1, \ldots, X_d, Y)$ independent of $(x_{i1}, \ldots, x_{id}, y_i)$, $i = 1, \ldots, m$

A case i is ...

- P (positive) if $y_i = 1$
- N (negative) if $y_i = 0$
- TP (true positive) if $\hat{y}_i = 1$ and $y_i = 1$
- FP (false positive) if $\hat{y}_i = 1$ and $y_i = 0$
- TN (true negative) if $\hat{y}_i = 0$ and $y_i = 0$
- FN (false negative) if $\hat{y}_i = 0$ and $y_i = 1$

Accuracy = (TP + TN)/(P + N) *% of correctly predicted cases*

Imbalanced data: often, one of the two classes P or N is much smaller than the other

- e.g. in fraud detection, P/(P+N) can be around 0.001
- in such cases, accuracy might not be the best performance measure

e.g., just predicting N has accuracy 99.9 %

For classification problems we can compute different performance measures and combine them

Logistic regression
Evaluating the performance of the estimated model

Performance measures for classification problems

TPR = TP/P *(true positive rate, recall, sensitivity)*

FPR = FP/N *(false positive rate)*

ROC *(receiving operator curve)*

shows (TPR,FPR) for $c \in [0, 1]$

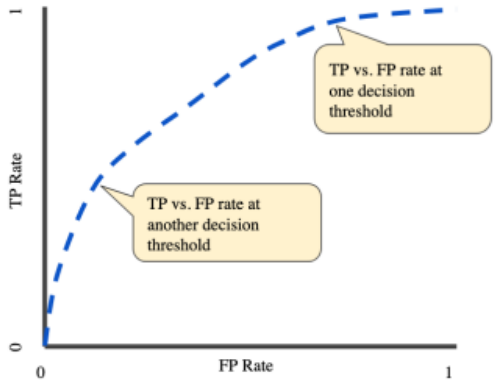
(originally from radar detection)

AUC *(area under the curve)*

(the larger the better)

(random guessing produces the diagonal, AUC = 1/2)

(AUC < 1/2 is worse than random guessing)



PPV = TP/(TP + FP) *(positive predication value, precision)*

$$F1 = 2 \times \frac{TPR \times PPV}{TPR + PPV} = \frac{2TP}{2TP + FP + FN} \qquad \textit{(F1 score)}$$

For classification problems we can compute different performance measures and combine them

Logistic regression: Key takeaways

I	**Ideal for Binary Classification**: Logistic regression excels in binary classification tasks, such as email spam detection or disease diagnosis, by categorizing data into two distinct groups
II	**Probabilistic Outputs**: Provides probability estimates for outcomes, offering insights into the likelihood of class membership, crucial for risk assessment and decision-making.
III	**Feature Influence Analysis**: Enables understanding of how different predictors affect the probability of the target class, useful for interpretative analysis in fields like healthcare and finance
IV	**Foundation for Advanced Techniques**: Serves as a stepping stone to more complex algorithms in machine learning, such as neural networks, while teaching important concepts like the sigmoid function and log-odds

Google Colab: `log_regr.ipynb`

Self-Study: Closing with logistic regression

Self-Study

From Moodle download the book:

James, G., Witten, D., Hastie, T., & Tibshirani, R. (2013). *An introduction to statistical learning* (Vol. 112, p. 18). New York: Springer.

- **Section 4.4.3** Logistic Regression

Feedback! See you on March 22nd