Federated Learning

From Theory to Practice

Alexander Jung

April 27, 2025

please cite as: A. Jung, *Federated Learning: From Theory to Practice*. Espoo, Finland: Aalto University, 2025.

Preface

This book presents a practical framework for understanding and designing federated learning (FL) systems. FL enables collaborative training of machine learning (ML) models across interconnected devices, each with access to local data and some computational power. It is particularly effective when the datasets—and the associated learning tasks—at different devices exhibit similarities. We explore FL methods that exploit these similarities to train personalized models for each device, in a privacy-friendly and distributed manner.

Our framework is built around the concept of an FL network and formulates FL as an optimization problem over such networks. An FL network is a mathematical structure that represents—or approximates— a real-world FL application. Its nodes correspond to devices, while its edges encode communication links and similarities between local datasets. Intuitively, we expect devices that are closely connected in the network to learn similar models.

The core methodological idea of this book is to train personalized models (one per device) by balancing their individual training losses with their differences across the network. We formalize this approach through *generalized total variation minimization* (GTVMin). Just as empirical risk minimization serves as a foundational principle in traditional ML, GTVMin serves as our guiding design principle in FL. A principled approach to build FL systems is then to solve GTVMin using distributed optimization techniques.

(**Main Goal**) *This book aims to train the reader in (i) modelling a*

given FL application as an FL network and (ii) designing practical FL systems by solving GTVMin using distributed optimization methods.

FL is already transforming fields such as healthcare, finance, telecommunications, and smart cities. In these areas, it enables organizations to develop accurate predictive models without ever centralizing sensitive data. Its decentralized nature not only ensures privacy but also facilitates scalability, allowing large-scale systems to train collaboratively without relying on a central repository.

We believe FL is poised to become a cornerstone of trustworthy artificial intelligence. It combines established ML principles with essential considerations for privacy, security, and robustness. By transmitting only model updates rather than raw data, FL enhances data privacy. Its decentralized structure also makes it more resilient to adversarial attacks, such as data poisoning or ????model inversion?????.

Audience. This book assumes only a solid grounding in calculus and linear algebra. Readers with a background in undergraduate-level mathematical analysis (covering concepts such as norms, convergence, and derivatives) should be able to follow all arguments and discussions. Even without deep mathematical training, readers can grasp the essential insights. While prior exposure to machine learning or optimization may be helpful, it is not required, as we build most key concepts from the ground up.

Acknowledgement. The development of this text has benefited from student feedback received in the course *CS-E4740 Federated Learning*, offered at Aalto

University during 2023–2025. The author is grateful to Bo Zheng, Olga Kuznetsova, Diana Pfau, and Shamsiiat Abdurakhmanova for their valuable comments on early drafts. Special thanks to Mikko Seesto for his careful proofreading of the manuscript. We are grateful to Konstantina Olioumtsevits for her meticulous proofreading of the glossary. This work was partially supported by:

- the Research Council of Finland (grants 331197, 363624, 349966),
- the European Union (grant 952410),
- the Jane and Aatos Erkko Foundation (grant A835), and
- Business Finland, as part of the project *FORWARD-LOOKING AI GOVERNANCE IN BANKING and INSURANCE (FLAIG).*

I hope this book inspires you to think creatively and critically about designing machine learning systems that are not only efficient but also fair, private, and robust.

Contents

Lists of Symbols

Sets and Functions

$a \in \mathcal{A}$	The object a is an element of the set $\mathcal{A}$.
$a := b$	We use a as a shorthand for b.
$\lvert\mathcal{A}\rvert$	The cardinality (i.e., number of elements) of a finite set $\mathcal{A}$.
$\mathcal{A} \subseteq \mathcal{B}$	$\mathcal{A}$ is a subset of $\mathcal{B}$.
$\mathcal{A} \subset \mathcal{B}$	$\mathcal{A}$ is a strict subset of $\mathcal{B}$.
$\mathbb{N}$	The natural numbers $1, 2, \ldots$.
$\mathbb{R}$	The real numbers x [1].
$\mathbb{R}_+$	The non-negative real numbers $x \geq 0$.
$\mathbb{R}_{++}$	The positive real numbers $x > 0$.

$\{0,1\}$	The set consisting of the two real numbers 0 and 1.
$[0,1]$	The closed interval of real numbers x with $0 \leq x \leq 1$.
$\operatorname{argmin}_{\mathbf{w}} f(\mathbf{w})$	The set of minimizers for a real-valued function $f(\mathbf{w})$.
$\mathbb{S}^{(n)}$	The set of unit-norm vectors in $\mathbb{R}^{n+1}$.
$\log a$	The logarithm of the positive number $a \in \mathbb{R}_{++}$.
$h(\cdot): \mathcal{A} \to \mathcal{B} : a \mapsto h(a)$	A function (map) that accepts any element $a \in \mathcal{A}$ from a set $\mathcal{A}$ as input and delivers a well-defined element $h(a) \in \mathcal{B}$ of a set $\mathcal{B}$. The set $\mathcal{A}$ is the domain of the function h and the set $\mathcal{B}$ is the codomain of h. Machine learning (ML) aims at finding (or learning) a function h (i.e., a hypothesis) that reads in the features $\mathbf{x}$ of a data point and delivers a prediction $h(\mathbf{x})$ for its label y.
$\nabla f(\mathbf{w})$	The gradient of a differentiable real-valued function $f : \mathbb{R}^d \to \mathbb{R}$ is the vector $\nabla f(\mathbf{w}) = \big(\frac{\partial f}{\partial w_1}, \ldots, \frac{\partial f}{\partial w_d}\big)^T \in \mathbb{R}^d$ [2, Ch. 9].

Matrices and Vectors

$\mathbf{x} = \big(x_1, \ldots, x_d\big)^T$	A vector of length d, with its j-th entry being x_j.
$\mathbb{R}^d$	The set of vectors $\mathbf{x} = \big(x_1, \ldots, x_d\big)^T$ consisting of d real-valued entries $x_1, \ldots, x_d \in \mathbb{R}$.
$\mathbf{I}_{l\times d}$	A generalized identity matrix with l rows and d columns. The entries of $\mathbf{I}_{l\times d} \in \mathbb{R}^{l\times d}$ are equal to 1 along the main diagonal and equal to 0 otherwise.
$\mathbf{I}_d, \mathbf{I}$	A square identity matrix of size $d \times d$. If the size is clear from context, we drop the subscript.
$\|\mathbf{x}\|_2$	The Euclidean (or ℓ_2) norm of the vector $\mathbf{x} = \big(x_1, \ldots, x_d\big)^T \in \mathbb{R}^d$ defined as $\|\mathbf{x}\|_2 := \sqrt{\sum_{j=1}^d x_j^2}$.
$\|\mathbf{x}\|$	Some norm of the vector $\mathbf{x} \in \mathbb{R}^d$ [3]. Unless specified otherwise, we mean the Euclidean norm $\|\mathbf{x}\|_2$.
$\mathbf{x}^T$	The transpose of a matrix that has the vector $\mathbf{x} \in \mathbb{R}^d$ as its single column.
$\mathbf{X}^T$	The transpose of a matrix $\mathbf{X} \in \mathbb{R}^{m\times d}$. A square real-valued matrix $\mathbf{X} \in \mathbb{R}^{m\times m}$ is called symmetric if $\mathbf{X} = \mathbf{X}^T$.
$\mathbf{0} = \big(0, \ldots, 0\big)^T$	The vector in $\mathbb{R}^d$ with each entry equal to zero.
$\mathbf{1} = \big(1, \ldots, 1\big)^T$	The vector in $\mathbb{R}^d$ with each entry equal to one.

$\left(\mathbf{v}^T, \mathbf{w}^T\right)^T$	The vector of length $d + d'$ obtained by concatenating the entries of vector $\mathbf{v} \in \mathbb{R}^d$ with the entries of $\mathbf{w} \in \mathbb{R}^{d'}$.
$\text{span}\{\mathbf{B}\}$	The span of a matrix $\mathbf{B} \in \mathbb{R}^{a \times b}$, which is the subspace of all linear combinations of the columns of $\mathbf{B}$, $\text{span}\{\mathbf{B}\} = \left\{\mathbf{Ba} : \mathbf{a} \in \mathbb{R}^b\right\} \subseteq \mathbb{R}^a$.
$\det(\mathbf{C})$	The determinant of the matrix $\mathbf{C}$.
$\mathbf{A} \otimes \mathbf{B}$	The Kronecker product of $\mathbf{A}$ and $\mathbf{B}$ [4].

Probability Theory

$\mathbb{E}_p\{f(\mathbf{z})\}$	The expectation of a function $f(\mathbf{z})$ of a random variable (RV) $\mathbf{z}$ whose probability distribution is $p(\mathbf{z})$. If the probability distribution is clear from context, we just write $\mathbb{E}\{f(\mathbf{z})\}$.
$p(\mathbf{x}, y)$	A (joint) probability distribution of an RV whose realizations are data points with features $\mathbf{x}$ and label y.
$p(\mathbf{x}\|y)$	A conditional probability distribution of an RV $\mathbf{x}$ given the value of another RV y [5, Sec. 3.5].
$p(\mathbf{x}; \mathbf{w})$	A parametrized probability distribution of an RV $\mathbf{x}$. The probability distribution depends on a parameter vector $\mathbf{w}$. For example, $p(\mathbf{x}; \mathbf{w})$ could be a multivariate normal distribution with the parameter vector $\mathbf{w}$ given by the entries of the mean vector $\mathbb{E}\{\mathbf{x}\}$ and the covariance matrix $\mathbb{E}\Big\{\big(\mathbf{x} - \mathbb{E}\{\mathbf{x}\}\big)\big(\mathbf{x} - \mathbb{E}\{\mathbf{x}\}\big)^T\Big\}$.
$\mathcal{N}(\mu, \sigma^2)$	The probability distribution of a Gaussian random variable (Gaussian RV) $x \in \mathbb{R}$ with mean (or expectation) $\mu = \mathbb{E}\{x\}$ and variance $\sigma^2 = \mathbb{E}\big\{(x-\mu)^2\big\}$.
$\mathcal{N}(\boldsymbol{\mu}, \mathbf{C})$	The multivariate normal distribution of a vector-valued Gaussian RV $\mathbf{x} \in \mathbb{R}^d$ with mean (or expectation) $\boldsymbol{\mu} = \mathbb{E}\{\mathbf{x}\}$ and covariance matrix $\mathbf{C} = \mathbb{E}\big\{\big(\mathbf{x} - \boldsymbol{\mu}\big)\big(\mathbf{x} - \boldsymbol{\mu}\big)^T\big\}$.

Machine Learning

r	An index $r = 1, 2, \ldots$ that enumerates data points.
m	The number of data points in (i.e., the size of) a dataset.
$\mathcal{D}$	A dataset $\mathcal{D} = \{\mathbf{z}^{(1)}, \ldots, \mathbf{z}^{(m)}\}$ is a list of individual data points $\mathbf{z}^{(r)}$, for $r = 1, \ldots, m$.
d	The number of features that characterize a data point.
x_j	The j-th feature of a data point. The first feature is denoted x_1, the second feature x_2, and so on.
$\mathbf{x}$	The feature vector $\mathbf{x} = \big(x_1, \ldots, x_d\big)^T$ of a data point whose entries are the individual features of a data point.
$\mathcal{X}$	The feature space $\mathcal{X}$ is the set of all possible values that the features $\mathbf{x}$ of a data point can take on.
$\mathbf{z}$	Instead of the symbol $\mathbf{x}$, we sometimes use $\mathbf{z}$ as another symbol to denote a vector whose entries are the individual features of a data point. We need two different symbols to distinguish between raw and learned features [6, Ch. 9].
$\mathbf{x}^{(r)}$	The feature vector of the r-th data point within a dataset.
$x_j^{(r)}$	The j-th feature of the r-th data point within a dataset.

$\mathcal{B}$	A mini-batch (or subset) of randomly chosen data points.
B	The size of (i.e., the number of data points in) a mini-batch.
y	The label (or quantity of interest) of a data point.
$y^{(r)}$	The label of the r-th data point.
$\left(\mathbf{x}^{(r)}, y^{(r)}\right)$	The features and label of the r-th data point.
$\mathcal{Y}$	The label space $\mathcal{Y}$ of an ML method consists of all potential label values that a data point can carry. The nominal label space might be larger than the set of different label values arising in a given dataset (e.g., a training set). ML problems (or methods) using a numeric label space, such as $\mathcal{Y} = \mathbb{R}$ or $\mathcal{Y} = \mathbb{R}^3$, are referred to as regression problems (or methods). ML problems (or methods) that use a discrete label space, such as $\mathcal{Y} = \{0, 1\}$ or $\mathcal{Y} = \{cat, dog, mouse\}$, are referred to as classification problems (or methods).
η	Learning rate (or step size) used by gradient-based methods.
$h(\cdot)$	A hypothesis map that reads in features $\mathbf{x}$ of a data point and delivers a prediction $\hat{y} = h(\mathbf{x})$ for its label y.
$\mathcal{Y}^{\mathcal{X}}$	Given two sets $\mathcal{X}$ and $\mathcal{Y}$, we denote by $\mathcal{Y}^{\mathcal{X}}$ the set of all possible hypothesis maps $h : \mathcal{X} \to \mathcal{Y}$.

$\mathcal{H}$	A hypothesis space or model used by an ML method. The hypothesis space consists of different hypothesis maps $h : \mathcal{X} \to \mathcal{Y}$, between which the ML method must choose.
$d_{\text{eff}}(\mathcal{H})$	The effective dimension of a hypothesis space $\mathcal{H}$.
B^2	The squared bias of a learned hypothesis $\hat{h}$ produced by an ML method. The method is trained on data points that are modeled as the realizations of RVs. Since the data is a realization of RVs, the learned hypothesis $\hat{h}$ is also the realization of an RV.
V	The variance of the learned (parameters of the) hypothesis produced by an ML method. The method is trained on data points that are modeled as the realizations of RVs. Since the data is a realization of RVs, the learned hypothesis $\hat{h}$ is also the realization of an RV.
$L((\mathbf{x}, y), h)$	The loss incurred by predicting the label y of a data point using the prediction $\hat{y} = h(\mathbf{x})$. The prediction $\hat{y}$ is obtained by evaluating the hypothesis $h \in \mathcal{H}$ for the feature vector $\mathbf{x}$ of the data point.
E_v	The validation error of a hypothesis h, which is its average loss incurred over a validation set.
$\widehat{L}\big(h\|\mathcal{D}\big)$	The empirical risk or average loss incurred by the hypothesis h on a dataset $\mathcal{D}$.

E_t	The training error of a hypothesis h, which is its average loss incurred over a training set.
t	A discrete-time index $t = 0, 1, \ldots$ used to enumerate sequential events (or time instants).
t	An index that enumerates learning tasks within a multitask learning problem.
α	A regularization parameter that controls the amount of regularization.
$\lambda_j\big(\mathbf{Q}\big)$	The j-th eigenvalue (sorted in either ascending or descending order) of a positive semi-definite (psd) matrix $\mathbf{Q}$. We also use the shorthand λ_j if the corresponding matrix is clear from context.
$\sigma(\cdot)$	The activation function used by an artificial neuron within an artificial neural network (ANN).
$\mathcal{R}_{\hat{y}}$	A decision region within a feature space.
$\mathbf{w}$	A parameter vector $\mathbf{w} = \big(w_1, \ldots, w_d\big)^T$ of a model, e.g., the weights of a linear model or in an ANN.
$h^{(\mathbf{w})}(\cdot)$	A hypothesis map that involves tunable model parameters $w_1, \ldots, w_d$ stacked into the vector $\mathbf{w} = \big(w_1, \ldots, w_d\big)^T$.
$\phi(\cdot)$	A feature map $\phi : \mathcal{X} \to \mathcal{X}' : \mathbf{x} \mapsto \mathbf{x}' := \phi\big(\mathbf{x}\big) \in \mathcal{X}'$.
$K\big(\cdot, \cdot\big)$	Given some feature space $\mathcal{X}$, a kernel is a map $K : \mathcal{X} \times \mathcal{X} \to \mathbb{C}$ that is psd.

Federated Learning

$\mathcal{G} = (\mathcal{V}, \mathcal{E})$	An undirected graph whose nodes $i \in \mathcal{V}$ represent devices within a federated learning network (FL network). The undirected weighted edges $\mathcal{E}$ represent connectivity between devices and statistical similarities between their datasets and learning tasks.
$i \in \mathcal{V}$	A node that represents some device within an FL network. The device can access a local dataset and train a local model.
$\mathcal{G}^{(\mathcal{C})}$	The induced subgraph of $\mathcal{G}$ using the nodes in $\mathcal{C} \subseteq \mathcal{V}$.
$\mathbf{L}^{(\mathcal{G})}$	The Laplacian matrix of a graph $\mathcal{G}$.
$\mathbf{L}^{(\mathcal{C})}$	The Laplacian matrix of the induced graph $\mathcal{G}^{(\mathcal{C})}$.
$\mathcal{N}^{(i)}$	The neighborhood of a node i in a graph $\mathcal{G}$.
$d^{(i)}$	The weighted degree $d^{(i)} := \sum_{i' \in \mathcal{N}^{(i)}} A_{i,i'}$ of a node i in a graph $\mathcal{G}$.
$d^{(\mathcal{G})}_{\max}$	The maximum weighted node degree of a graph $\mathcal{G}$.
$\mathcal{D}^{(i)}$	The local dataset $\mathcal{D}^{(i)}$ carried by node $i \in \mathcal{V}$ of an FL network.
m_i	The number of data points (i.e., sample size) contained in the local dataset $\mathcal{D}^{(i)}$ at node $i \in \mathcal{V}$.

$\mathbf{x}^{(i,r)}$	The features of the r-th data point in the local dataset $\mathcal{D}^{(i)}$.
$y^{(i,r)}$	The label of the r-th data point in the local dataset $\mathcal{D}^{(i)}$.
$\mathbf{w}^{(i)}$	The local model parameters of device i within an FL network.
$L_i\left(\mathbf{w}\right)$	The local loss function used by device i to measure the usefulness of some choice $\mathbf{w}$ for the local model parameters.
$L^{(\mathrm{d})}\left(\mathbf{x}, h\big(\mathbf{x}\big), h'\big(\mathbf{x}\big)\right)$	The loss incurred by a hypothesis h' on a data point with features $\mathbf{x}$ and label $h\big(\mathbf{x}\big)$ that is obtained from another hypothesis.
$\mathrm{stack}\big\{\mathbf{w}^{(i)}\big\}_{i=1}^{n}$	The vector $\left(\big(\mathbf{w}^{(1)}\big)^{T}, \ldots, \big(\mathbf{w}^{(n)}\big)^{T}\right)^{T} \in \mathbb{R}^{dn}$ that is obtained by vertically stacking the local model parameters $\mathbf{w}^{(i)} \in \mathbb{R}^{d}$.

1 Introduction

We are surrounded by devices, such as smartphones or wearables, generating decentralized collections of local datasets [7–11]. These local datasets typically have an intrinsic network structure that arises from functional constraints (connectivity between devices) or statistical similarities.

For example, the management of pandemics uses contact networks to relate local datasets generated by patients. Network medicine relates data about diseases via co-morbidity networks [12]. Social science uses notions of acquaintance to relate data collected from be-friended individuals [13]. Another example for network-structured data are the weather observations collected at Finnish Meteorological Institute (FMI) stations. The FMI stations generate local datasets which tend to have similar statistical properties for nearby stations.

Federated learning (FL) is an umbrella term for distributed optimization techniques to train machine learning (ML) models from decentralized collections of local datasets [14–18]. The idea is to carry out the model training directly at the location of data generation (such as your smartphone or a heart rate sensor). This is different from the basic ML workflow, which is to (i) collect data at a single location (computer) and (ii) then train a single ML model on this data.

It can be beneficial to train different ML models at the locations of actual data generation [19] for several reasons:

- **Privacy.** FL methods are appealing for applications involving sensitive data (such as healthcare) as they do not require the exchange of raw data

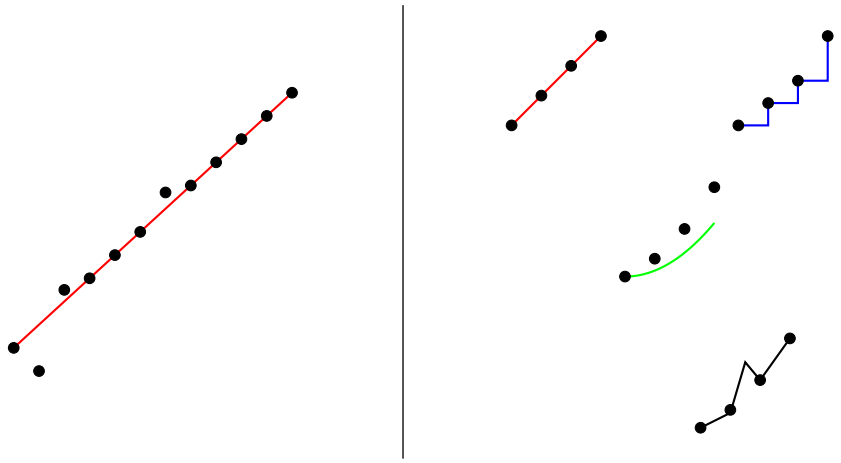


Figure 1.1: Left: A basic ML method uses a single dataset to train a single model. Right: Decentralized collection of devices (or *clients*) with the ability to generate data and train models.

but only model parameters (or their updates) [16, 17]. By exchanging only (updates of) model parameters, FL methods are considered privacy-friendly in the sense of not leaking (too much) sensitive information that is contained in the local datasets (see Chapter 9).

- **Robustness.** By relying on decentralized data and computation, FL methods offer robustness (to some extent) against hardware failures (such as *stragglers*) and cyber attacks. One type of cyber attack on FL systems is data poisoning which we discuss in Chapter 10.

- **Parallel Computing.** We can interpret a collection of interconnected devices as a parallel computer. One example of such a parallel computer is a mobile network constituted by smartphones that can communicate via radio links. This parallel computer allows to speed up computations required for the training of ML models (see Chapter 4).

- **Democratization of ML.** FL allows to combine (or pool) the computational resources of many low-cost devices to train high-dimensional models such as a large language model (LLM). Instead of using a few powerful computers we combine the contributions of many low-complexity devices [20, 21].

- **Trading Computation against Communication.** Consider a FL application where local datasets are generated by low-complexity devices at remote locations (think of a wildlife camera) that cannot be easily accessed. The cost of communicating raw local datasets to some central unit (which then trains a single global ML model) can be much higher than the computational cost incurred by using the low-complexity devices to (partially) train ML models [22].

- **Personalization.** FL trains personalized ML models for collections of devices (e.g., smartphones or wearables) with computational capabilities and can access local datasets [23]. A key challenge for ensuring personalization is the heterogeneity of local datasets [24, 25]. Indeed, the statistical properties of different local datasets can vary significantly such that they cannot be accurately modelled as independent and identically distributed (i.i.d.). Each local dataset induces a separate learning task that consists of learning useful parameter values for a local model. FL systems train personalized models for devices by combining the information carried in their local datasets (see Chapter 6).

1.1 Main Tools

Euclidean space. Our main mathematical structure for the study and design of FL systems is the Euclidean space $\mathbb{R}^d$. We expect familiarity with the algebraic and geometric structure of $\mathbb{R}^d$ [26, 27]. For example, we often use the spectral decomposition of positive semi-definite (psd) matrices that naturally arise in the formulation of FL applications. We will also use the geometric structure of $\mathbb{R}^d$, which is defined by the inner-product $\mathbf{w}^T\mathbf{w}' := \sum_{j=1}^{d} w_j w'_j$ between two vectors $\mathbf{w}, \mathbf{w}' \in \mathbb{R}^d$ and the induced norm $\|\mathbf{w}\|_2 := \sqrt{\mathbf{w}^T\mathbf{w}} = \sqrt{\sum_{j=1}^{d} w_j^2}$.

Calculus. A main toolbox for the design the FL algorithms are variants of gradient descent (GD). The common idea of these gradient-based methods is to approximate a function $f(\mathbf{w})$ locally by a linear function. This local linear approximation is determined by the gradient $\nabla f(\mathbf{w})$. We, therefore, expect some familiarity with multivariable calculus [2].

Fixed-Point Iterations. Each algorithm that we discuss in this book can be interpreted as a fixed-point iteration of some (typically non-linear) operator $\mathcal{P} : \mathbb{R}^d \to \mathbb{R}^d$. These operators depend on the local datasets and personal models used within a FL system. A prime example of such a non-linear operator is the gradient step of gradient-based methods (see Chapter 4). The computational properties (such as convergence speed) of these FL algorithms are determined by the contraction properties of the underlying operator [28].

1.2 Main Goal of the Book

The overarching goal of the book is to study FL applications and algorithms using mathematical structures from network theory and optimization theory. In particular, we develop the concept of an federated learning network (FL network) to represent a real-world FL application. We then formulate FL as an optimization problem over a given FL network. This lends naturally to a principled design of FL algorithms using distributed optimization methods.

The nodes of an FL network represent devices that train a local (personalized) model based on some local loss function. One natural construction for loss functions is via the average loss incurred on local datasets. However, the details of how to implement (the access to) loss functions are beyond the scope of this book. Our focus is on the analysis of the FL networks as a precise formulation of FL applications. This formulation assigns each node in an FL network a local model (or hypothesis space) and a local loss function.

Some devices of an FL network are connected by links which can be used to share messages (e.g., intermediate results of computations) during the FL process. We represent these links by the undirected weighted edges of an FL network. The edges of an FL network not only represent communication links but also some notion of statistical similarity between (the data generated by) different devices.

We will see later in the book (see Ch. 7.4) how to construct measures for statistical similarities between local datasets. However, we can view the edges also as a design choice that determines the behaviour of FL algorithms that built on top of an FL network.

From an engineering point of view, the main goal of the book is to establish

a flexible design principle for FL systems (see Ch. 3). This approach is inspired by empirical risk minimization as a main tool for the analysis and design of ML systems [6, 29]. Similar to empirical risk minimization, our design principle is formulated as an optimization problem, which we refer to as generalized total variation minimization (GTVMin).

GTVMin optimizes model parameters for the local models by balancing the sum of the incurred local loss with some measure for their variation across the edges of the FL network. Thus, GTVMin amounts to minimizing the local loss while also requiring the model parameters and connected nodes to be similar. We obtain different FL methods by using different measures for the variation of model parameters across edges.

Once a FL application is formulated as GTVMin, FL algorithms can be designed by applying distributed optimization methods to solve GTVMin. All optimization methods studied in this book are instances of fixed-point iterations. Different methods arise from selecting various fixed-point operators, each having the solutions of GTVMin as their fixed points. A key example of a fixed-point iteration is GD.

1.3 Outline

This book is roughly divided into three parts:

- **Part I: ML Refresher.** Chapter 2 introduces data, models and loss functions as three main components of ML. This chapter also explains how these components are combined within empirical risk minimization (ERM). We also discuss how to regularize ERM via manipulating its three main components. We then explain when and how to solve

regularized ERM via simple GD methods in Chapter 4. Overall, this part serves two main purposes: (i) to briefly recap basic concepts of ML in a simple centralized setting and (ii) to highlight ML techniques (such as regularization) that are particularly relevant for the design and analysis of FL methods.

- **Part II: FL Theory and Methods.** Chapter 3 introduces the FL network as a mathematical structure for representing collections of devices that generate local datasets and train local (*personalized*) ML models. An FL network also contains undirected weighted edges that connect some of the nodes. The edges represent statistical similarities between local datasets as well as communication links for the implementation of FL algorithms. Chapter 3 formulates FL as an instance of regularized empirical risk minimization (RERM) which we refer to as GTVMin. GTVMin uses the variation of local model parameters across edges of the FL network as regularizer. We will see that GTVMin couples the training of local ML models such that well-connected nodes (clusters) in the FL network obtain similar trained models. Chapter 4 discusses variations of gradient descent as our main algorithmic toolbox for solving GTVMin. Chapter 5 shows how FL algorithms can be obtained in a principled fashion by applying optimization methods, such as gradient-based methods, to GTVMin. We will obtain FL algorithms that can be implemented as iterative message passing methods for the distributed training of tailored (or *personalized*) models. Chapter 6 shows how some main flavours of FL can be interpreted as special cases of GTVMin. The usefulness of GTVMin crucially depends on the choice for the weighted

edges of the FL network. Chapter 7 discusses basic principles of graph learning methods. The common idea of these methods is to place edges between the nodes of an FL network that are most similar. Similarity measures can be obtained via statistical inference procedures or via learnt vector representations. For example, we can map a dataset to a vector by evaluating the gradient of the average loss incurred over this dataset.

- **Part III: Trustworthy AI.** Chapter 8 enumerates seven key requirements for trustworthy AI that have been put forward by the European Union. These key requirements include the protection of privacy as well as robustness against (intentional) perturbations of data or computation. We then discuss how FL algorithms can ensure privacy protection in Chapter 9. Chapter 10 discusses how to evaluate and ensure the robustness of FL methods against intentional perturbations (poisoning) of local dataset.

1.4 Exercises

1.1. Complexity of Matrix Inversion. Choose your favourite computer architecture (represented by a mathematical model) and think about how much computation is required - in the worst case - by the most efficient algorithm that can invert any given invertible matrix $\mathbf{Q} \in \mathbb{R}^{d \times d}$? Try also to reflect on how practical your chosen computer architecture is, i.e., is it possible to buy such a computer in your nearest electronics shop?

1.2. Vector Spaces and Euclidean Norm. Consider data points, each

characterized by a feature vector $\mathbf{x} \in \mathbb{R}^d$ with entries $x_1, x_2, \ldots, x_d$.

- Show that the set of all feature vectors forms a vector space under standard addition and scalar multiplication.
- Calculate the Euclidean norm of the vector $\mathbf{x} = (1, -2, 3)^T$.
- If $\mathbf{x}^{(1)} = (1, 2, 3)^T$ and $\mathbf{x}^{(2)} = (-1, 0, 1)^T$, compute $3\mathbf{x}^{(1)} - 2\mathbf{x}^{(2)}$.

1.3. Matrix Operations in Linear Models. Linear regression methods learn model parameters $\widehat{\mathbf{w}} \in \mathbb{R}^d$ via solving the optimization problem:

$$\widehat{\mathbf{w}} = \arg\min_{\mathbf{w} \in \mathbb{R}^d} \|\mathbf{y} - \mathbf{X}\mathbf{w}\|_2^2,$$

with some matrix $\mathbf{X} \in \mathbb{R}^{m \times d}$, and some vector $\mathbf{y} \in \mathbb{R}^m$.

- Derive a closed-form expression for $\widehat{\mathbf{w}}$ that is valid for *arbitrary* matrix $\mathbf{X}$, and vector $\mathbf{y}$.
- Discuss the conditions under which $\mathbf{X}^T\mathbf{X}$ is invertible.
- Compute $\widehat{\mathbf{w}}$ for the following dataset:

$$\mathbf{X} = \begin{pmatrix} 1 & 2 \\ 3 & 4 \\ 5 & 6 \end{pmatrix}, \quad \mathbf{y} = \begin{pmatrix} 7 \\ 8 \\ 9 \end{pmatrix}.$$

- Compute $\widehat{\mathbf{w}}$ for the following dataset: The rth row of $\mathbf{X}$, for $r = 1, \ldots, 28$, is given by the temperature recordings (with a 10-minute interval) during day r/Mar/2023 at FMI weather station *Kustavi Isokari*. The rth row of $\mathbf{y}$ is the maximum daytime temperature during day $r + 1$/Mar/2023 at the same weather station.

1.4. Eigenvalues and Positive Semi-Definiteness. The convergence properties of widely-used ML methods rely on the properties of psd matrices. Let $\mathbf{Q} = \mathbf{X}^T\mathbf{X}$, where $\mathbf{X} \in \mathbb{R}^{m \times d}$.

1. Prove that $\mathbf{Q}$ is psd.
2. Compute the eigenvalues of $\mathbf{Q}$ for $\mathbf{X} = \begin{pmatrix} 1 & 2 \\ 3 & 4 \end{pmatrix}$.
3. Compute the eigenvalues of $\mathbf{Q}$ for the matrix $\mathbf{X}$ used in Exercise 1.3 that is constituted by FMI temperature recordings.

2 ML Basics

This chapter covers basic ML techniques instrumental for FL. Content-wise, this chapter is more extensive compared to the following chapters. However, this chapter should be considerably easier to follow than the following chapters as it mainly refreshes pre-requisite knowledge.

2.1 Learning Goals

After completing this chapter, you will

- be familiar with the concept of data points (their features and labels), model and loss function,
- be familiar with ERM as a design principle for ML systems,
- know why and how validation is performed,
- be able to diagnose ML methods by comparing the training error with the validation error,
- be able to regularize ERM via modifying data, model and loss.

2.2 Three Components and a Design Principle

Machine Learning (ML) revolves around learning a hypothesis map h out of a hypothesis space $\mathcal{H}$ that allows to accurately predict the label of a data point solely from its features. One of the most crucial steps in applying ML methods to a given application domain is the definition or choice of what precisely a data point is. Coming up with a good choice or definition of data

points is not trivial as it influences the overall performance of a ML method in many different ways.

We will use weather prediction as a recurring example for an FL application. Here, data points represent the daily weather conditions around FMI weather stations. We denote a specific data point by $\mathbf{z}$. It is characterized by the following features:

- name of the FMI weather station, e.g., "TurkuRajakari"
- latitude lat and longitude lon of the weather station, e.g., $\mathsf{lat} := 60.37788$, $\mathsf{lon} := 22.0964$,
- timestamp of the measurement in the format YYYY-MM-DD HH:MM:SS, e.g., 2023-12-31 18:00:00

It is convenient to stack the features into a feature vector $\mathbf{x}$. The label $y \in \mathbb{R}$ of such a data point is the maximum daytime temperature in degree Celsius, e.g., -20. We indicate the features $\mathbf{x}$ and label y of a data point via the notation $\mathbf{z} = (\mathbf{x}, y)$.[1]

We predict the label of a data point with features $\mathbf{x}$ by the function value $h(\mathbf{x})$ of a hypothesis (map) $h(\cdot)$. The prediction will typically be not perfect, i.e., $h(\mathbf{x}) \neq y$. ML methods use a loss function $L\left((\mathbf{x}, y), h\right)$ to measure the error incurred by using the prediction $h(\mathbf{x})$ as a guess for the true label y. The choice of loss function crucially influences the statistical and computational properties of the resulting ML method (see [6, Ch. 2]).

[1] Strictly speaking, a data point $\mathbf{z}$ is not the same as the pair of features $\mathbf{x}$ and label y. Indeed, a data point can have additional properties that are neither used as features nor as label. A more precise notation would then be $\mathbf{x}(\mathbf{z})$ and $y(\mathbf{z})$, indicating that the features $\mathbf{x}$ and label y are functions of the data point $\mathbf{z}$.

It seems natural to choose (or learn) a hypothesis that minimizes the average loss (or empirical risk) on a given set of data points

$$\mathcal{D} := \left\{ \left(\mathbf{x}^{(1)}, y^{(1)}\right), \ldots, \left(\mathbf{x}^{m}, y^{(m)}\right) \right\}.$$

This is known as ERM,

$$\hat{h} \in \operatorname*{argmin}_{h \in \mathcal{H}} (1/m) \sum_{r=1}^{m} L\left(\left(\mathbf{x}^{(r)}, y^{(r)}\right), h\right). \tag{1}$$

As the notation in (1) indicates (using the symbol "$\in$" instead of ":="), there can be several different solutions to the optimization problem (1). Unless specified otherwise, $\hat{h}$ can be used to denote any hypothesis in $\mathcal{H}$ that has minimum average loss over $\mathcal{D}$.

Several important machine learning (ML) methods use a parametric model $\mathcal{H}$: Each hypothesis $h \in \mathcal{H}$ is defined by parameters $\mathbf{w} \in \mathbb{R}^d$, often indicated by the notation $h^{(\mathbf{w})}$. One important example of a parametrized model is the linear model [6, Sec. 3.1],

$$\mathcal{H}^{(d)} := \left\{ h^{(\mathbf{w})} : \mathbb{R}^d \mapsto \mathbb{R} : h^{(\mathbf{w})}(\mathbf{x}) = \mathbf{w}^T \mathbf{x} \right\}. \tag{2}$$

Linear regression learns the parameters of a linear model by minimizing the average squared error loss,

$$\widehat{\mathbf{w}}^{(\mathrm{LR})} \in \operatorname*{argmin}_{\mathbf{w} \in \mathbb{R}^d} (1/m) \sum_{r=1}^{m} \underbrace{\left(y^{(r)} - \mathbf{w}^T \mathbf{x}^{(r)}\right)^2}_{=L\left(\left(\mathbf{x}^{(r)}, y^{(r)}\right), h^{(\mathbf{w})}\right)}. \tag{3}$$

Note that (3) amounts to finding the minimum of a smooth and convex function

$$f(\mathbf{w}) := (1/m) \left[\mathbf{w}^T \mathbf{X}^T \mathbf{X} \mathbf{w} - 2 \mathbf{y}^T \mathbf{X} \mathbf{w} + \mathbf{y}^T \mathbf{y} \right]. \tag{4}$$

Here, we use the feature matrix

$$\mathbf{X} := \big(\mathbf{x}^{(1)}, \ldots, \mathbf{x}^{(m)}\big)^T \tag{5}$$

and the label vector

$$\mathbf{y} := \big(y^{(1)}, \ldots, y^{(m)}\big)^T \tag{6}$$

of the training set $\mathcal{D}$.

Inserting (4) into (3) allows to formulate linear regression as

$$\widehat{\mathbf{w}}^{(\mathrm{LR})} \in \operatorname*{argmin}_{\mathbf{w} \in \mathbb{R}^d} \mathbf{w}^T \mathbf{Q} \mathbf{w} + \mathbf{w}^T \mathbf{q} \tag{7}$$
$$\text{with } \mathbf{Q} := (1/m)\mathbf{X}^T\mathbf{X}, \mathbf{q} := -(2/m)\mathbf{X}^T\mathbf{y}.$$

The matrix $\mathbf{Q} \in \mathbb{R}^{d \times d}$ is psd with eigenvalue decomposition (EVD),

$$\mathbf{Q} = \sum_{j=1}^{d} \lambda_j \mathbf{u}^{(j)} \big(\mathbf{u}^{(j)}\big)^T. \tag{8}$$

The EVD (8) consists of orthonormal eigenvectors $\mathbf{u}^{(1)}, \ldots, \mathbf{u}^{(d)}$ and corresponding list of non-negative eigenvalues

$$0 \leq \lambda_1 \leq \ldots \leq \lambda_d, \text{ with } \mathbf{Q}\mathbf{u}^{(j)} = \lambda_j \mathbf{u}^{(j)}. \tag{9}$$

The list of eigenvalues is unique for a given psd matrix $\mathbf{Q}$. In contrast, the eigenvectors $\mathbf{u}^{(j)}$ are not unique in general.

To train a ML model $\mathcal{H}$ means to solve ERM (1) (or (3) for linear regression); the dataset $\mathcal{D}$ is therefore referred to as a training set. The trained model results in the learnt hypothesis $\hat{h}$. We obtain practical ML methods by applying optimization algorithms to solve (1). Two key questions studied in ML are:

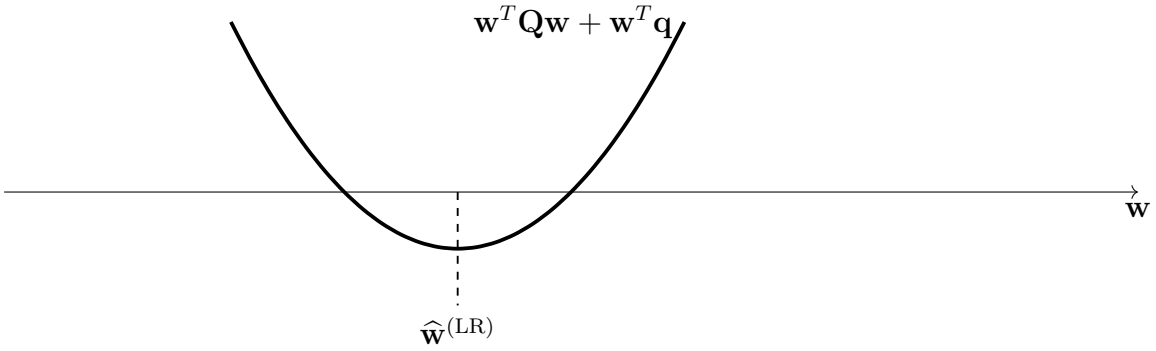


Figure 2.1: ERM (1) for linear regression minimizes a convex quadratic function $\mathbf{w}^T\mathbf{Q}\mathbf{w} + \mathbf{w}^T\mathbf{q}$.

- **Computational aspects.** How much compute do we need to solve (1)?
- **Statistical aspects.** How useful is the solution $\hat{h}$ to (1) in general, i.e., how accurate is the prediction $\hat{h}(\mathbf{x})$ for the label y of an **arbitrary** data point with features $\mathbf{x}$?

2.3 Computational Aspects of ERM

ML methods use optimization algorithms to solve (1), resulting in the learnt hypothesis $\hat{h}$. Within this book, we use optimization algorithms that are iterative methods: Starting from an initial choice $h^{(0)}$, they construct a sequence

$$h^{(0)}, h^{(1)}, h^{(2)}, \ldots,$$

which are hopefully increasingly accurate approximations to a solution $\hat{h}$ of (1). The computational complexity of such a ML method can be measured

by the number of iterations required to guarantee some prescribed level of approximation.

For a parametric model and a smooth loss function, we can solve (3) by gradient-based methods: Starting from an initial parameters $\mathbf{w}^{(0)}$, we iterate the gradient step:

$$\begin{aligned}\mathbf{w}^{(k)} &:= \mathbf{w}^{(k-1)} - \eta \nabla f\big(\mathbf{w}^{(k-1)}\big) \\ &= \mathbf{w}^{(k-1)} + (2\eta/m) \sum_{r=1}^{m} \mathbf{x}^{(r)} \big(y^{(r)} - \big(\mathbf{w}^{(k-1)}\big)^T \mathbf{x}^{(r)}\big). \end{aligned} \tag{10}$$

How much computation do we need for one iteration of (10)? How many iterations do we need? We will try to answer the latter question in Chapter 4. The first question can be answered more easily for a typical computational infrastructure (e.g., "Python running on a commercial Laptop"). The evaluation of (10) then typically requires around m arithmetic operations (addition, multiplication).

It is instructive to consider the special case of a linear model that does not use any feature, i.e., $h(\mathbf{x}) = w$. For this extreme case, the ERM (3) has a simple closed-form solution:

$$\widehat{w} = (1/m) \sum_{r=1}^{m} y^{(r)}. \tag{11}$$

Thus, for this special case of the linear model, solving (11) amounts to summing m numbers $y^{(1)}, \ldots, y^{(m)}$. The amount of computation, measured by the number of elementary arithmetic operations, required by (11) is proportional to m.

2.4 Statistical Aspects of ERM

We can train a linear model on a given training set as ERM (3). But how useful is the solution $\widehat{\mathbf{w}}$ of (3) for predicting the labels of data points outside the training set? Consider applying the learnt hypothesis $h^{(\widehat{\mathbf{w}})}$ to an arbitrary data point not contained in the training set. What can we say about the resulting prediction error $y - h^{(\widehat{\mathbf{w}})}(\mathbf{x})$ in general? In other words, how well does $h^{(\widehat{\mathbf{w}})}$ generalize beyond the training set.

The most widely used approach to study the generalization of ML methods is via probabilistic models. Here, we interpret each data point as a realization of an i.i.d. random variable (RV) with probability distribution $p(\mathbf{x}, y)$. Under this independent and identically distributed assumption (i.i.d. assumption), we can evaluate the overall performance of a hypothesis $h \in \mathcal{H}$ via the expected loss (or risk)

$$\mathbb{E}\{L\left((\mathbf{x}, y), h\right)\}. \tag{12}$$

One example of a probability distribution $p(\mathbf{x}, y)$ relates the label y with the features $\mathbf{x}$ of a data point as

$$y = \overline{\mathbf{w}}^T \mathbf{x} + \varepsilon \text{ with } \mathbf{x} \sim \mathcal{N}(\mathbf{0}, \mathbf{I}), \varepsilon \sim \mathcal{N}(0, \sigma^2), \mathbb{E}\{\varepsilon \mathbf{x}\} = \mathbf{0}. \tag{13}$$

A simple calculation reveals the expected squared error loss of a given linear hypothesis $h(\mathbf{x}) = \mathbf{x}^T \widehat{\mathbf{w}}$ as[2]

[2]Strictly speaking, the relation (14) only applies for constant (deterministic) model parameters $\widehat{\mathbf{w}}$ that do not depend on the RVs whose realizations are the observed data points (see, e.g., (13)). However, the learnt model parameters $\widehat{\mathbf{w}}$ is often the output of a ML method (such as (3)) that is applied to a dataset $\mathcal{D}$ whose generation is modelled as i.i.d. realizations from some underlying probability distribution. In this case, we need to replace the expectation on the LHS of (14) with a conditional expectation $\mathbb{E}\big\{(y - h(\mathbf{x}))^2 \big| \mathcal{D}\big\}$.

$$\mathbb{E}\{(y - h(\mathbf{x}))^2\} = \|\overline{\mathbf{w}} - \widehat{\mathbf{w}}\|^2 + \sigma^2. \quad (14)$$

The first component in (14) is the estimation error $\|\overline{\mathbf{w}} - \widehat{\mathbf{w}}\|^2$ of a ML method that reads in the training set and delivers an estimate $\widehat{\mathbf{w}}$ (e.g., via (3)) for the parameters of a linear hypothesis. The second component σ^2 in (14) can be interpreted as the intrinsic noise level of the label y. We cannot hope to find a hypothesis with an expected loss below σ^2.

We next study the estimation error $\overline{\mathbf{w}} - \widehat{\mathbf{w}}$ incurred by the specific estimate $\widehat{\mathbf{w}} = \widehat{\mathbf{w}}^{(\mathrm{LR})}$ (7) delivered by linear regression methods. To this end, we first use the probabilistic model (13) to decompose the label vector $\mathbf{y}$ in (6) as

$$\mathbf{y} = \mathbf{X}\overline{\mathbf{w}} + \mathbf{n} \text{ , with } \mathbf{n} := \big(\varepsilon^{(1)}, \ldots, \varepsilon^{(m)}\big)^T. \quad (15)$$

Inserting (15) into (7) yields

$$\widehat{\mathbf{w}}^{(\mathrm{LR})} \in \operatorname*{argmin}_{\mathbf{w} \in \mathbb{R}^d} \mathbf{w}^T \mathbf{Q} \mathbf{w} + \mathbf{w}^T \mathbf{q}' + \mathbf{w}^T \mathbf{e} \quad (16)$$

$$\text{with } \mathbf{Q} := (1/m)\mathbf{X}^T\mathbf{X}, \mathbf{q}' := -(2/m)\mathbf{X}^T\mathbf{X}\overline{\mathbf{w}}, \text{ and } \mathbf{e} := -(2/m)\mathbf{X}^T\mathbf{n}. \quad (17)$$

Figure 2.2 depicts the objective function of (16). It is a perturbation of the convex quadratic function $\mathbf{w}^T\mathbf{Q}\mathbf{w} + \mathbf{w}^T\mathbf{q}'$, which is minimized at $\mathbf{w} = \overline{\mathbf{w}}$. In general, the minimizer $\widehat{\mathbf{w}}^{(\mathrm{LR})}$ delivered by linear regression is different from $\overline{\mathbf{w}}$ due to the perturbation term $\mathbf{w}^T\mathbf{e}$ in (16).

The following result bounds the deviation between $\widehat{\mathbf{w}}^{(\mathrm{LR})}$ and $\overline{\mathbf{w}}$ under the assumption that the matrix $\mathbf{Q} = (1/m)\mathbf{X}^T\mathbf{X}$ is invertible.[3]

[3]Can you think of sufficient conditions on the feature matrix of the training set that ensure $\mathbf{Q} = (1/m)\mathbf{X}^T\mathbf{X}$ is invertible?

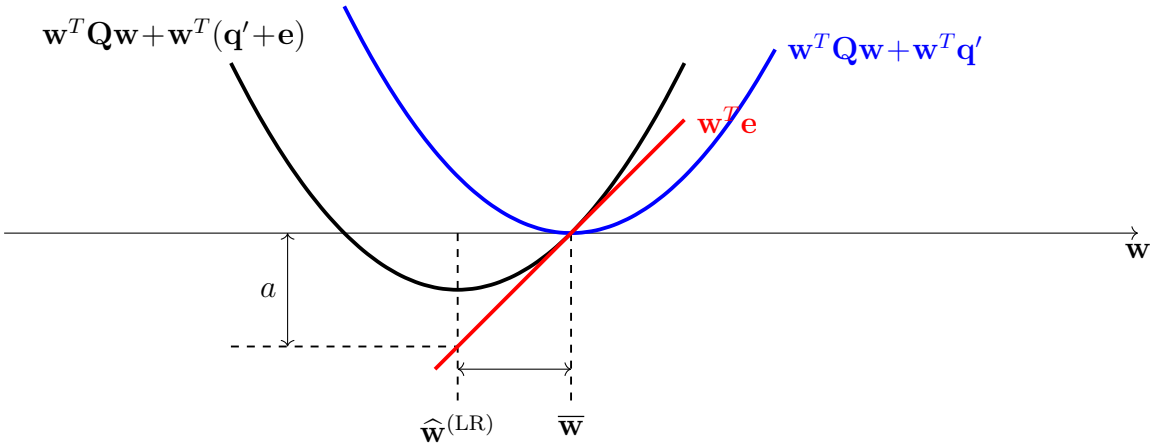


Figure 2.2: The estimation error of linear regression is determined by the effect of the perturbation term $\mathbf{w}^T\mathbf{e}$ on the minimizer of the convex quadratic function $\mathbf{w}^T\mathbf{Q}\mathbf{w} + \mathbf{w}^T\mathbf{q}'$.

Proposition 2.1. *Consider a solution* $\widehat{\mathbf{w}}^{(\mathrm{LR})}$ *to the ERM instance* (16) *that is applied to the dataset* (15)*. If the matrix* $\mathbf{Q} = (1/m)\mathbf{X}^T\mathbf{X}$ *is invertible, with minimum eigenvalue* $\lambda_1(\mathbf{Q}) > 0$*,*

$$\left\|\widehat{\mathbf{w}}^{(\mathrm{LR})} - \overline{\mathbf{w}}\right\|_2^2 \leq \frac{\|\mathbf{e}\|_2^2}{\lambda_1^2} \overset{(17)}{=} \frac{4}{m^2}\frac{\left\|\mathbf{X}^T\mathbf{n}\right\|_2^2}{\lambda_1^2}. \tag{18}$$

Proof. Let us rewrite (16) as

$$\widehat{\mathbf{w}}^{(\mathrm{LR})} \in \operatorname*{argmin}_{\mathbf{w}\in\mathbb{R}^d} f(\mathbf{w}) \text{ with } f(\mathbf{w}) := \big(\mathbf{w}-\overline{\mathbf{w}}\big)^T\mathbf{Q}\big(\mathbf{w}-\overline{\mathbf{w}}\big) + \mathbf{e}^T\big(\mathbf{w}-\overline{\mathbf{w}}\big). \tag{19}$$

Clearly $f\big(\overline{\mathbf{w}}\big) = 0$ and, in turn, $f(\widehat{\mathbf{w}}) = \min_{\mathbf{w}\in\mathbb{R}^d} f(\mathbf{w}) \leq 0$. On the other hand,

$$\begin{aligned} f(\mathbf{w}) &\overset{(19)}{=} \big(\mathbf{w}-\overline{\mathbf{w}}\big)^T\mathbf{Q}\big(\mathbf{w}-\overline{\mathbf{w}}\big) + \mathbf{e}^T\big(\mathbf{w}-\overline{\mathbf{w}}\big) \\ &\overset{(a)}{\geq} \big(\mathbf{w}-\overline{\mathbf{w}}\big)^T\mathbf{Q}\big(\mathbf{w}-\overline{\mathbf{w}}\big) - \|\mathbf{e}\|_2\,\|\mathbf{w}-\overline{\mathbf{w}}\|_2 \\ &\overset{(b)}{\geq} \lambda_1\,\|\mathbf{w}-\overline{\mathbf{w}}\|_2^2 - \|\mathbf{e}\|_2\,\|\mathbf{w}-\overline{\mathbf{w}}\|_2\,. \end{aligned} \tag{20}$$

Step (a) used Cauchy–Schwarz inequality and (b) used the EVD (8) of $\mathbf{Q}$. Evaluating (20) for $\mathbf{w}=\widehat{\mathbf{w}}$ and combining with $f\big(\widehat{\mathbf{w}}\big) \leq 0$ yields (18). $\square$

The bound (18) suggests that the estimation error $\widehat{w}^{(\mathrm{LR})} - \overline{w}$ is small if $\lambda_1(\mathbf{Q})$ is large. This smallest eigenvalue of the matrix $\mathbf{Q} = (1/m)\mathbf{X}^T\mathbf{X}$ could be controlled by a suitable choice (or transformation) of features $\mathbf{x}$ of a data point. Trivially, we can increase $\lambda_1(\mathbf{Q})$ by a factor of 100 if we scale each feature by a factor of 10. However, this approach would also scale (by a factor of 100) the error term $\big\|\mathbf{X}^T\mathbf{n}\big\|_2^2$ in (18). For some applications, we can find feature transformations ("whitening") that increases $\lambda_1(\mathbf{Q})$ but do not increase $\big\|\mathbf{X}^T\mathbf{n}\big\|_2^2$. We finally note that the error term $\big\|\mathbf{X}^T\mathbf{n}\big\|_2^2$ in (18) vanishes if the noise vector $\mathbf{n}$ is orthogonal to the columns of the feature matrix $\mathbf{X}$.

It is instructive to evaluate the bound (18) for the special case where each data point has the same feature $x = 1$. Here, the probabilistic model (15) reduces to a *signal in noise* model,

$$y^{(r)} = x^{(r)}\overline{w} + \varepsilon^{(r)} \text{ with } x^{(r)} = 1, \tag{21}$$

with some true underlying parameter $\overline{w}$. The noise terms $\varepsilon^{(r)}$, for $r = 1, \ldots, m$, are realizations of i.i.d. RVs with probability distribution $\mathcal{N}(0, \sigma^2)$. The feature matrix then becomes $\mathbf{X} = \mathbf{1}$ and, in turn, $\mathbf{Q} = 1$, $\lambda_1(\mathbf{Q}) = 1$. Inserting these values into (18) results in the bound

$$\big(\widehat{w}^{(\mathrm{LR})} - \overline{w}\big)^2 \leq 4\,\|\mathbf{n}\|_2^2\,/m^2. \tag{22}$$

For the labels and features in (21), the solution of (16) is given by

$$\widehat{w}^{(\mathrm{LR})} = (1/m)\sum_{r=1}^{m} y^{(r)} \overset{(21)}{=} \overline{w} + (1/m)\sum_{r=1}^{m} \varepsilon^{(r)}. \tag{23}$$

2.5 Validation and Diagnosis of ML

The above analysis of the generalization error started from postulating the probabilistic model (13) for the generation of data points. Strictly speaking, if the data points are not generated according to the probabilistic model the bound (18) does not apply. Thus, we might want to use a more data-driven approach for assessing the usefulness of a learnt hypothesis $\hat{h}$ obtained, e.g., from solving ERM (1).

Loosely speaking, validation tries to find out if a learnt hypothesis $\hat{h}$ performs similarly well inside and outside the training set. In its most basic form, validation amounts to computing the average loss of a learnt hypothesis $\hat{h}$ on some data points not included in the training set. We refer to these data points as the validation set.

Algorithm 1 summarizes a single iteration of a prototypical ML workflow that consists of model training and validation. The workflow starts with an initial choice of a dataset $\mathcal{D}$, model $\mathcal{H}$, and loss function $L\left(\cdot,\cdot\right)$. We then repeat Algorithm 1 several times. After each repetition, based on the resulting training error and validation error, we modify (some of) the design choices for the dataset, the model and the loss function.

We can diagnose an ERM-based ML method, such as Algorithm 1, by comparing its training error with its validation error. This diagnosis is further enabled if we know a baseline $E^{(\mathrm{ref})}$. One important source for a baseline $E^{(\mathrm{ref})}$ are probabilistic models for the data points (see Section 2.4).

Given a probabilistic model $p(\mathbf{x}, y)$, we can compute the minimum achievable risk (12). Indeed, the minimum achievable risk is precisely the expected loss of the Bayes estimator $\widehat{h}(\mathbf{x})$ of the label y, given the features $\mathbf{x}$ of a

Algorithm 1 One Iteration of ML Training and Validation

Input: dataset $\mathcal{D}$, model $\mathcal{H}$, loss function $L\left(\cdot,\cdot\right)$

1: split $\mathcal{D}$ into a training set $\mathcal{D}^{(\text{train})}$ and a validation set $\mathcal{D}^{(\text{val})}$

2: learn a hypothesis via solving ERM

$$\widehat{h} \in \operatorname*{argmin}_{h \in \mathcal{H}} \sum_{(\mathbf{x},y) \in \mathcal{D}^{(\text{train})}} L\left(\left(\mathbf{x},y\right),h\right) \tag{24}$$

3: compute resulting training error

$$E_t := (1/|\mathcal{D}^{(\text{train})}|) \sum_{(\mathbf{x},y) \in \mathcal{D}^{(\text{train})}} L\left(\left(\mathbf{x},y\right),\widehat{h}\right)$$

4: compute validation error

$$E_v := (1/|\mathcal{D}^{(\text{val})}|) \sum_{(\mathbf{x},y) \in \mathcal{D}^{(\text{val})}} L\left(\left(\mathbf{x},y\right),\widehat{h}\right)$$

Output: learnt hypothesis (or trained model) $\widehat{h}$, training error E_t and validation error E_v

data point. The Bayes estimator $\widehat{h}(\mathbf{x})$ is fully determined by the probability distribution $p(\mathbf{x}, y)$ [30, Chapter 4].

A further potential source for a baseline $E^{(\text{ref})}$ is an existing, but for some reason unsuitable, ML method. This existing ML method might be computationally too expensive to be used for the ML application at hand. However, we might still use its statistical properties as a baseline.

We can also use the performance of human experts as a baseline. For example,if we develop a ML method to detect skin cancer from images, a possible baseline is the classification accuracy achieved by experienced dermatologists [31].

We can diagnose a ML method by comparing the training error E_t with the validation error E_v and (if available) the baseline $E^{(\text{ref})}$.

- $E_t \approx E_v \approx E^{(\text{ref})}$: The training error is on the same level as the validation error and the baseline. There seems to be little point in trying to improve the method further since the validation error is already close to the baseline. Moreover, the training error is not much smaller than the validation error which indicates that there is no overfitting.

- $E_v \gg E_t$: The validation error is significantly larger than the training error, which hints at overfitting. We can address overfitting either by reducing the effective dimension of the hypothesis space or by increasing the size of the training set. To reduce the effective dimension of the hypothesis space, we can use fewer features (in a linear model), a smaller maximum depth of decision trees or fewer layers in an artificial neural network (ANN). Instead of this coarse-grained discrete model pruning, we can also reduce the effective dimension of a hypothesis

space continuously via regularization (see [6, Ch. 7]).

- $E_t \approx E_v \gg E^{(\text{ref})}$: The training error is on the same level as the validation error and both are significantly larger than the baseline. Thus, the learnt hypothesis seems to not overfit the training set. However, the training error achieved by the learnt hypothesis is significantly larger than the baseline. There can be several reasons for this to happen. First, it might be that the hypothesis space is too small, i.e., it does not include a hypothesis that provides a satisfactory approximation for the relation between the features and the label of a data point. One remedy to this situation is to use a larger hypothesis space, e.g., by including more features in a linear model, using higher polynomial degrees in polynomial regression, using deeper decision trees or ANNs with more hidden layers (deep net). Second, besides the model being too small, another reason for a large training error could be that the optimization algorithm used to solve ERM (24) is not working properly (see Chapter 4).

- $E_t \gg E_v$: The training error is significantly larger than the validation error. The idea of ERM (24) is to approximate the risk (12) of a hypothesis by its average loss on a training set $\mathcal{D} = \{(\mathbf{x}^{(r)}, y^{(r)})\}_{r=1}^{m}$. The mathematical underpinning for this approximation is the law of large numbers which characterizes the average of (realizations of) i.i.d. RVs. The accuracy of this approximation depends on the validity of two conditions: First, the data points used for computing the average loss "should behave" like realizations of i.i.d. RVs with a common probability

distribution. Second, the number of data points used for computing the average loss must be sufficiently large.

Whenever the training set or validation set differs significantly from realizations of i.i.d. RVs, the interpretation (and comparison) of the training error and the validation error of a learnt hypothesis becomes more difficult. As an extreme case, the validation set might consist of data points for which every hypothesis incurs a small average loss (see Figure 2.3). Here, we might try to increase the size of the validation set by collecting more labelled data points or by using data augmentation (see Sec. 2.6). If the size of the training set and the validation set is large but we still obtain $E_t \gg E_v$, we should verify if the data points in these sets conform to the i.i.d. assumption. There are principled statistical tests for the validity of the i.i.d. assumption for a given dataset (see [32] and references therein).

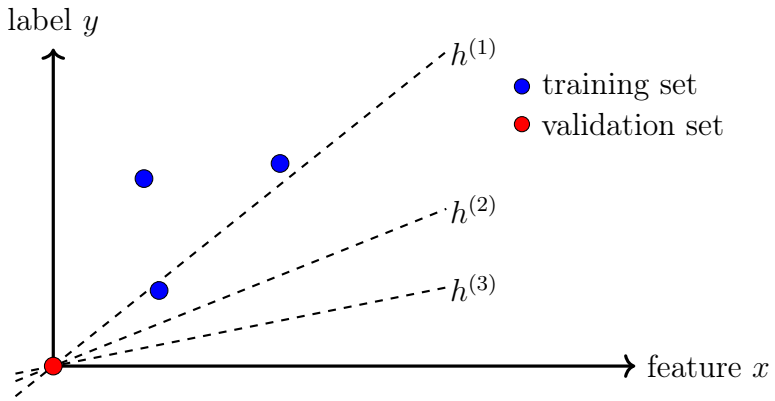


Figure 2.3: An example for an unlucky split of a dataset into a training set and a validation set for the model $\mathcal{H} := \{h^{(1)}, h^{(2)}, h^{(3)}\}$.

2.6 Regularization

Consider an ERM-based ML method using a hypothesis space $\mathcal{H}$ and dataset $\mathcal{D}$ (we assume all data points are used for training). A key parameter for such a ML method is the ratio $d_{\text{eff}}(\mathcal{H})/|\mathcal{D}|$ between the model size $d_{\text{eff}}(\mathcal{H})$ and the number $|\mathcal{D}|$ of data points. The tendency of the ML method to overfit increases with the ratio $d_{\text{eff}}(\mathcal{H})/|\mathcal{D}|$.

Regularization techniques decrease the ratio $d_{\text{eff}}(\mathcal{H})/|\mathcal{D}|$ via three (essentially equivalent) approaches:

- collect more data points, possibly via data augmentation (see Fig. 2.4),
- add penalty term $\alpha\mathcal{R}\big\{h\big\}$ to average loss in ERM (1) (see Fig. 2.4),

$$\hat{h} \in \operatorname*{argmin}_{h\in\mathcal{H}} (1/m) \sum_{r=1}^{m} L\left(\left(\mathbf{x}^{(r)}, y^{(r)}\right), h\right) + \alpha\mathcal{R}\big\{h\big\}, \tag{25}$$

- shrink the hypothesis space, e.g., by adding constraints on the model parameters such as $\|\mathbf{w}\|_2 \leq 10$.

It can be shown that these three perspectives (corresponding to the three components data, model and loss) on regularization are closely related [6, Ch. 7]. For example, the regularized ERM (25) is equivalent to ERM (1) with a pruned hypothesis space $\mathcal{H}^{(\alpha)} \subseteq \mathcal{H}$. Using a larger α typically results in a smaller $\mathcal{H}^{(\alpha)}$.

One example for regularization via adding a penalty term is ridge regression. In particular, ridge regression uses the regularizer $\mathcal{R}\big\{h\big\} := \|\mathbf{w}\|_2^2$ for a linear hypothesis $h(\mathbf{x}) := \mathbf{w}^T\mathbf{x}$. Thus, ridge regression learns the parameters of a

linear hypothesis via solving

$$\widehat{\mathbf{w}}^{(\alpha)} \in \operatorname*{argmin}_{\mathbf{w}\in\mathbb{R}^d} \left[(1/m) \sum_{r=1}^{m} \left(y^{(r)} - \mathbf{w}^T \mathbf{x}^{(r)} \right)^2 + \alpha \left\| \mathbf{w} \right\|_2^2 \right]. \tag{26}$$

The objective function in (26) can be interpreted as the objective function of linear regression applied to a modification of the training set $\mathcal{D}$: We replace each data point $(\mathbf{x}, y) \in \mathcal{D}$ by a sufficiently large number of i.i.d. realizations of

$$(\mathbf{x} + \mathbf{n}, y) \text{ , with } \mathbf{n} \sim \mathcal{N}(\mathbf{0}, \alpha \mathbf{I}). \tag{27}$$

Thus, ridge regression (26) is equivalent to linear regression applied to an augmentation $\mathcal{D}'$ of the original dataset $\mathcal{D}$. The augmentation $\mathcal{D}'$ is obtained by replacing each data point $(\mathbf{x}, y) \in \mathcal{D}$ with a sufficiently large number of noisy copies. Each copy of $(\mathbf{x}, y)$ is obtained by adding an i.i.d. realization $\mathbf{n}$ of a zero-mean Gaussian noise with covariance matrix $\alpha \mathbf{I}$ to the features $\mathbf{x}$ (see (27)). The label of each copy of $(\mathbf{x}, y)$ is equal to y, i.e., the label is not perturbed.

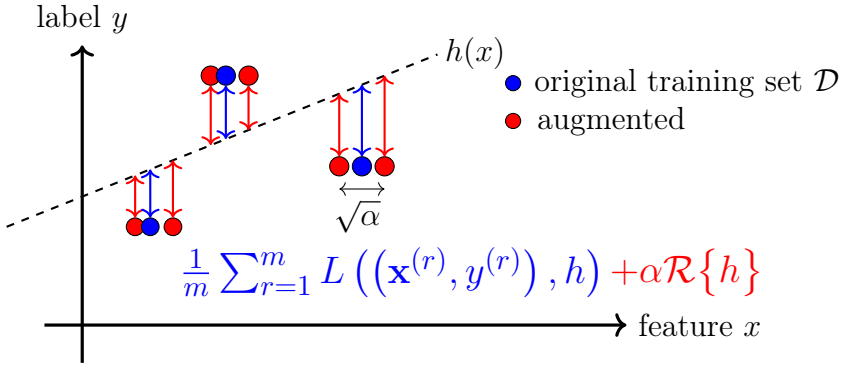


Figure 2.4: Equivalence between data augmentation and loss penalization.

To study the computational aspects of ridge regression, let us rewrite (26) as

$$\widehat{\mathbf{w}}^{(\alpha)} \in \operatorname*{argmin}_{\mathbf{w}\in\mathbb{R}^d} \mathbf{w}^T\mathbf{Q}\mathbf{w} + \mathbf{w}^T\mathbf{q}, \quad \text{with } \mathbf{Q} := (1/m)\mathbf{X}^T\mathbf{X} + \alpha\mathbf{I},\ \mathbf{q} := (-2/m)\mathbf{X}^T\mathbf{y}. \tag{28}$$

Thus, like linear regression (7), also ridge regression minimizes a convex quadratic function. A main difference between linear regression (7) and ridge regression (for $\alpha > 0$) is that the matrix $\mathbf{Q}$ in (28) is guaranteed to be invertible for any training set $\mathcal{D}$. In contrast, the matrix $\mathbf{Q}$ in (7) for linear regression might be singular for some training sets.[4]

The statistical properties of the solutions to (28) (i.e., the parameters learnt by ridge regression) crucially depend on the value of α. This choice can be guided by an error analysis using a probabilistic model for the data (see Proposition 2.1). Instead of using a probabilistic model, we can also compare the training error and validation error of the hypothesis $h(\mathbf{x}) = \big(\widehat{\mathbf{w}}^{(\alpha)}\big)^T\mathbf{x}$ learnt by ridge regression with different values of α.

2.7 Upgrading a Linear Model

This book presents FL algorithms (see Ch. 5) that are flexible in the sense of allowing to use different types of ML models. However, for ease of exposition we mainly focus on the special case of linear models. The restriction to linear models allows for a more comprehensive analysis of FL applications. On the

[4]Consider the extreme case where all features of each data point in the training set $\mathcal{D}$ are zero.

flip side, the scope of our analysis is limited to FL applications involving local models that can be well approximated by linear models.

Several important ML methods are obtained from the combination of non-linear feature learning and a linear model. For example,

- a deep net, with the hidden layers implementing a learnable feature map and the final (output) layer implementing a linear model [33], [6, Sec. 3.11.],
- a decision tree with a fixed topology (which corresponds to a specific decision boundary) but tunable predictions for each decision region [34], [6, Sec. 3.10],
- kernel methods [35], [6, Sec. 3.9].

2.8 From ML to FL via Regularization

The main theme of this book is the analysis of FL systems that consists of a network of devices, indexed by $i = 1, \ldots, n$. Each device i train a local (or personalized) model $\mathcal{H}^{(i)}$. One natural way to couple the training processes at different devices is to via regularization. Assume that each device solves a separate instance of RERM (25), for a parametric model,

$$\widehat{\mathbf{w}}^{(i)} \in \operatorname*{argmin}_{\mathbf{w}^{(i)} \in \mathbb{R}^d} (1/m) \sum_{r=1}^{m} L\left(\left(\mathbf{x}^{(r)}, y^{(r)}\right), h\right) + \alpha \mathcal{R}^{(i)}\left(\mathbf{w}^{(i)}\right). \tag{29}$$

To couple the RERM instance (29) at device i with other devices $\mathcal{N}^{(i)} \subseteq \{1, \ldots, n\} \backslash \{i\}$, we simply construct a regularizer $\mathcal{R}^{(i)}$ that penalizes deviations between the model parameters $\mathbf{w}^{(i)}$ and those at the other devices, $\mathbf{w}^{(i')}$ for $i' \in \mathcal{N}^{(i)}$. Chapter 3 discusses this construction in some detail.

2.9 Exercises

2.1. Fundamental Limits for Linear Regression. Linear regression learns model parameters of a linear model to minimize the risk $\mathbb{E}\{\big(y-\mathbf{w}^T\mathbf{x}\big)^2\}$ where $(\mathbf{x}, y)$ is a RV. In practice, we do not observe the RV $(\mathbf{x}, y)$ itself but a (realization of a) sequence of i.i.d. samples $\big(\mathbf{x}^{(t)}, y^{(t)}\big)$, for $t = 1, 2, \ldots$. The minimax risk is a lower bound on the risk obtained by any learning method [36, Ch. 15]. Determine the minimax risk in terms of the probability distribution of $(\mathbf{x}, y)$.

2.2. Uniqueness of Eigenvectors. Consider the EVD $\mathbf{Q} = \sum_{j=1}^{d} \lambda_j \mathbf{u}^{(j)} \big(\mathbf{u}^{(j)}\big)^T$ of a psd matrix $\mathbf{Q}$. The EVD consists of orthonormal eigenvectors $\mathbf{u}^{(j)}$ and non-negative eigenvalues λ_j, with $\mathbf{Q}\mathbf{u}^{(j)} = \lambda_j \mathbf{u}^{(j)}$, for $j = 1, \ldots, d$. Can you provide conditions on the eigenvalues $\lambda_1 \leq \ldots \leq \lambda_d$ such that the (unit-norm) eigenvectors are unique?

2.3. Penalty Term as Data Augmentation. Consider a ML method that trains a model with model parameters $\mathbf{w}$. The training uses ERM with squared error loss. Show that regularization of the model training via adding a penalty term $\alpha \left\|\mathbf{w}\right\|_2^2$ is equivalent to a specific form of data augmentation. What is the augmented training set?

2.4. Data Augmentation via Linear Interpolation. Consider a ML method that trains a model, with model parameters $\mathbf{w}$, from a training set $\mathcal{D}$. Each data point $\mathbf{z} \in \mathcal{D}$ is characterized by a feature vector $\mathbf{x} \in \mathbb{R}^d$ and label $y \in \mathbb{R}$, i.e., $\mathbf{z} = (\mathbf{x}, y)$. We augment the training set by adding, for each pair of two different data points $\mathbf{z}, \mathbf{z}' \in \mathcal{D}$, synthetic data points $\tilde{\mathbf{z}}^{(r)} := \mathbf{z} + (\mathbf{z}' - \mathbf{z})r/100$ and , for $r = 0, \ldots, 99$. Does this augmentation typically increase the training error?

2.5. Ridge Regression via Deterministic Data Augmentation. Ridge regression is obtained from linear regression by adding the penalty term $\alpha \left\|\mathbf{w}\right\|_2^2$ to the average squared error loss incurred by the hypothesis $h^{(\mathbf{w})}$ on the training set $\mathcal{D}$,

$$\min_{\mathbf{w}} (1/m) \sum_{r=1}^{m} \left(y^{(r)} - h\big(\mathbf{x}^{(r)}\big)\right)^2 + \alpha \left\|\mathbf{w}\right\|_2^2 . \tag{30}$$

Construct an augmented training set $\mathcal{D}'$ such that the objective function of (30) coincides with the objective function of plain linear regression using $\mathcal{D}'$ as training set. To construct $\mathcal{D}'$, add carefully chosen data points to the original training set $\mathcal{D} = \left\{ \left(y^{(1)}, \mathbf{x}^{(1)}\right), \ldots, \left(y^{(m)}, \mathbf{x}^{(m)}\right) \right\}$. Generalize the construction of $\mathcal{D}'$ to implement a generalized form of ridge regression,

$$\min_{\mathbf{w}} (1/m) \sum_{r=1}^{m} \left(y^{(r)} - h\big(\mathbf{x}^{(r)}\big)\right)^2 + \alpha \left\|\mathbf{w} - \widetilde{\mathbf{w}}\right\|_2^2 . \tag{31}$$

Here, we used some prescribed reference model parameters $\widetilde{\mathbf{w}}$. Note that (31) reduces to basic ridge regression (30) for the specific choice $\widetilde{\mathbf{w}} = \mathbf{0}$.

3 A Design Principle for FL

Chapter 2 reviewed ML methods that use numeric arrays to store data points (their features and labels) and model parameters. We have also discussed ERM (and its regularization) as the main design principle for practical ML systems. This chapter extends the basic ML concepts from a centralized *single-dataset single-model* setting to FL applications involving distributed collections of data and models.

Section 3.2 introduces the notion of an FL network as a mathematical model of a FL application. An FL network consists of nodes that represent devices generating local datasets and training local models. Some of the nodes are connected by weighted edges that represent communication links and statistical similarities between devices and their local datasets.

Section 3.3 introduces GTV as a measure for the discrepancy between the local model parameters at connected nodes. Section 3.4 uses GTV to regularize the training of parametric local models, resulting in GTVMin as our main design principle for FL algorithms. Section 3.5 generalize GTVMin from parametric local models to non-parametric local models. Section 3.6 discusses some useful interpretations of GTVMin that offer conceptual links to other fields of applied mathematics and statistics.

3.1 Learning Goals

After completing this chapter, you will

- be familiar with the concept of an FL network,
- know how to characterize the connectivity of an FL network via the

spectrum of its Laplacian matrix,

- know some measures for the variation of local models,
- be able to formulate FL as instances of GTVMin.

3.2 FL Networks

Consider a FL system that consists of devices, indexed by $i = 1, \ldots, i$, each with the ability to generate a local dataset $\mathcal{D}^{(i)}$ and to train a personalized model $\mathcal{H}^{(i)}$. These devices collaborate with each other via some communication network to learn a local hypothesis $h^{(i)} \in \mathcal{H}^{(i)}$. We measure the quality of $h^{(i)} \in \mathcal{H}^{(i)}$ via some loss function $L_i\left(h^{(i)}\right)$.

We now introduce the concept of an FL network as a mathematical model for FL applications. An FL network consists of an undirected weighted graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ with nodes $\mathcal{V} := \{1, \ldots, n\}$ and undirected edges $\mathcal{E}$ between pairs of different nodes. The nodes $\mathcal{V}$ represent devices with varying amounts of computational resources.

An undirected edge $\{i, i'\} \in \mathcal{E}$ in an FL network represents a form of similarity between device i and device i'. The amount of similarity is represented by an edge weight $A_{i,i'}$. We can collect edge weights into an *adjacency matrix* $\mathbf{A} \in \mathbb{R}^{n \times n}$, with $A_{i,i'} = A_{i',i}$. Fig. 3.1 depicts an example of an FL network.

Note that the undirected edges $\mathcal{E}$ of an FL network encode a symmetric notion of similarity between devices: If the device i is similar to the device i', i.e., $\{i, i'\} \in \mathcal{E}$, then also the device i' is similar to the device i. For some FL applications, an asymmetric notion of similarity, represented by directed edges, could be more accurate. However, the generalization of an FL network

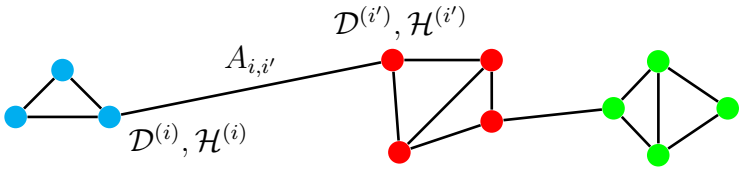


Figure 3.1: Example of an FL network whose nodes $i \in \mathcal{V}$ represent devices. Each device i generates a local dataset $\mathcal{D}^{(i)}$ and trains a local model $\mathcal{H}^{(i)}$. Some devices i, i' are connected by an undirected edge $\{i, i'\}$ with a positive edge weight $A_{i,i'}$.

to directed graphs is beyond the scope of this book.

It can be convenient to replace a given FL network $\mathcal{G}$ with an equivalent fully connected FL network $\mathcal{G}'$ (see Figure 3.2). The fully connected graph $\mathcal{G}'$ contains an edge between every pair of two different nodes i, i',

$$\mathcal{E}' = \big\{\{i, i'\} : i, i' \in \mathcal{V}, i \neq i'\big\}.$$

The edge weights are chosen $A'_{i,i'} = A_{i,i'}$ for any edge $\{i, i'\} \in \mathcal{E}$ and $A'_{i,i'} = 0$ if the original FL network $\mathcal{G}$ does not contain an edge between nodes i, i'.

Figure 3.2: Left: An FL network $\mathcal{G}$ consisting of $n = 4$ nodes. Right: Equivalent fully connected FL network $\mathcal{G}'$ with the same nodes and non-zero edge weights $A'_{i,i'} = A_{i,i'}$ for $\{i, i'\} \in \mathcal{E}$ and $A'_{i,i'} = 0$ for $\{i, i'\} \notin \mathcal{E}$.

An FL network is more than the undirected weighted graph $\mathcal{G}$: It also includes the local dataset $\mathcal{D}^{(i)}$ and the local model $\mathcal{H}^{(i)}$ (or its model parameters $\mathbf{w}^{(i)}$) for each device $i \in \mathcal{V}$. The details of the generation and the format of a local dataset will not be important in what follows. Indeed, the main purpose of a local dataset will be as a means to construct a loss function to evaluate a given choice of model parameters. However, to build intuition, we can think of a local dataset $\mathcal{D}^{(i)}$ as a labelled dataset

$$\mathcal{D}^{(i)} := \left\{ \left(\mathbf{x}^{(i,1)}, y^{(i,1)}\right), \ldots, \left(\mathbf{x}^{(i,m_i)}, y^{(i,m_i)}\right) \right\}. \tag{32}$$

Here, $\mathbf{x}^{(i,r)}$ and $y^{(i,r)}$ denote, respectively, the features and the label of the rth data point in the local dataset $\mathcal{D}^{(i)}$. Note that the size m_i of the local dataset can vary between different nodes $i \in \mathcal{V}$.

It is convenient to collect the feature vectors $\mathbf{x}^{(i,r)}$ and labels $y^{(i,r)}$ into a feature matrix $\mathbf{X}^{(i)}$ and label vector $\mathbf{y}^{(i)}$, respectively,

$$\mathbf{X}^{(i)} := \left(\mathbf{x}^{(i,1)}, \ldots, \mathbf{x}^{(i,m_i)}\right)^T, \text{ and } \mathbf{y} := \left(y^{(1)}, \ldots, y^{(m_i)}\right)^T. \tag{33}$$

The local dataset $\mathcal{D}^{(i)}$ can then be represented compactly by the feature matrix $\mathbf{X}^{(i)} \in \mathbb{R}^{m_i \times d}$ and the vector $\mathbf{y}^{(i)} \in \mathbb{R}^{m_i}$.

Besides the local dataset $\mathcal{D}^{(i)}$, each node $i \in \mathcal{G}$ also carries a local model $\mathcal{H}^{(i)}$. Our focus is on parametric local models with by model parameters $\mathbf{w}^{(i)} \in \mathbb{R}^d$, for $i = 1, \ldots, n$. The usefulness of a specific choice of the local model parameter $\mathbf{w}^{(i)}$ is then measured by a local loss function $L_i\left(\mathbf{w}^{(i)}\right)$, for $i = 1, \ldots, n$. Note that we can use different local loss functions $L_i(\cdot) \neq L_{i'}(\cdot)$ at different nodes $i, i' \in \mathcal{V}$.

We now have introduced all the components of an FL network. Strictly speaking, an FL network is a tuple $\left(\mathcal{G}, \{\mathcal{H}^{(i)}\}_{i \in \mathcal{V}}, \{L_i(\cdot)\}_{i \in \mathcal{V}}\right)$ consisting of

an undirected weighted graph $\mathcal{G}$, a local model $\mathcal{H}^{(i)}$ and local loss function $L_i\,(i)$ for each node $i \in \mathcal{V}$. In principle, all of these components are design choices that influence the computational and statistical properties of the FL algorithms presented in Chapter 5. The main focus of this book will be on the effect of the network structure, i.e., the edges $\mathcal{E}$ of the graph $\mathcal{G}$, on the resulting FL systems.

The role (or meaning) of an edge $\{i, i'\}$ in an FL network is two-fold: First, it represents a communication link that allows to exchange messages between devices i, i' (see Sec. 5.3). Second, an edge $\{i, i'\}$ indicates similar statistical properties of local datasets generated by devices i, i'. It then seems natural to learn similar hypothesis maps $h^{(i)}, h^{(i')}$. This is actually the main idea behind all the FL algorithms that we will discuss in the rest of this book. To make this idea precise, we next discuss how to obtain quantitative measures for how much local hypothesis maps $h^{(i)}$ vary across the edges $\{i, i'\} \in \mathcal{E}$ of an FL network.

3.3 Generalized Total Variation

Consider an FL network with nodes $i = 1, \ldots, n$, undirected edges $\mathcal{E}$ with edge weights $A_{i,i'} > 0$ for each $\{i, i'\} \in \mathcal{E}$. For each edge $\{i, i'\} \in \mathcal{E}$, we want to couple the training of the corresponding local models $\mathcal{H}^{(i)}, \mathcal{H}^{(i')}$. The strength of this coupling is determined by the edge weight $A_{i,i'}$. We implement the coupling by penalizing the discrepancy between the model parameters $\mathbf{w}^{(i)}, \mathbf{w}^{(i')}$.

We can measure the discrepancy between two local models $h^{(i)}, h^{(i')}$ across $\{i, i'\} \in \mathcal{E}$ in different ways. For example, we can compare their predictions

on a common test set $\mathcal{D}$ by computing $\sum_{\mathbf{x} \in \mathcal{D}} \big[h^{(i)}(\mathbf{x}) - h^{(i')}(\mathbf{x})\big]^2$. In what follows, our main focus will be on FL networks with parametrized local models $\mathcal{H}^{(i)}$, each having their own model parameters $\mathbf{w}^{(i)} \in \mathbb{R}^d$ with some fixed dimension d.

We measure the discrepancy between parametric local models using a function $\phi\big(\mathbf{w}^{(i)} - \mathbf{w}^{(i')}\big)$, which depends on the difference $\mathbf{w}^{(i)} - \mathbf{w}^{(i')}$. The function ϕ can be arbitrary, provided it is monotonically increasing with respect to some norm in the Euclidean space $\mathbb{R}^d$ [18, 37]. This requirement ensures symmetry, i.e., $\phi\big(\mathbf{w}^{(i)} - \mathbf{w}^{(i')}\big) = \phi\big(\mathbf{w}^{(i')} - \mathbf{w}^{(i)}\big)$, allowing its use as a measure of variation across an undirected edge $\{i, i'\} \in \mathcal{E}$.

By summing up the edge-wise variations (weighted by the edge weights), we obtain the GTV of a collection of local model parameters,

$$\sum_{\{i,i'\} \in \mathcal{E}} A_{i,i'} \phi\big(\mathbf{w}^{(i)} - \mathbf{w}^{(i')}\big). \tag{34}$$

Our main focus will be on the special case of (34), obtained for $\phi(\cdot) := \|\cdot\|_2^2$,

$$\sum_{\{i,i'\} \in \mathcal{E}} A_{i,i'} \Big\|\mathbf{w}^{(i)} - \mathbf{w}^{(i')}\Big\|_2^2. \tag{35}$$

The choice of penalty $\phi(\cdot)$ has a crucial impact on the computational and statistical properties of the FL algorithms presented in Ch. 5. Our main choice during the rest of this book will be the penalty function $\phi(\cdot) := \|\cdot\|_2^2$. This choice often allows to formulate FL as the minimization of a smooth convex function, which can be done via simple gradient-based methods (see Ch. 7). On the other hand, choosing ϕ to be a norm results in FL algorithms that require more computation but less training data [37].

The connectivity of an FL network $\mathcal{G}$ can be characterized locally - around a node $i \in \mathcal{V}$ - by its weighted node degree

$$d^{(i)} := \sum_{i' \in \mathcal{N}^{(i)}} A_{i,i'}. \tag{36}$$

Here, we used the neighborhood $\mathcal{N}^{(i)} := \{i' \in \mathcal{V} : \{i, i'\} \in \mathcal{E}\}$ of node $i \in \mathcal{V}$. A global characterization for the connectivity of $\mathcal{G}$ is the maximum weighted node degree

$$d^{(\mathcal{G})}_{\max} := \max_{i \in \mathcal{V}} d^{(i)} \overset{(36)}{=} \max_{i \in \mathcal{V}} \sum_{i' \in \mathcal{N}^{(i)}} A_{i,i'}. \tag{37}$$

Besides inspecting its (maximum) node degrees, we can study the connectivity of $\mathcal{G}$ also via the eigenvalues and eigenvectors of its Laplacian matrix $\mathbf{L}^{(\mathcal{G})} \in \mathbb{R}^{n \times n}$.[5] The Laplacian matrix of an undirected weighted graph $\mathcal{G}$ is defined element-wise as (see Fig. 3.3)

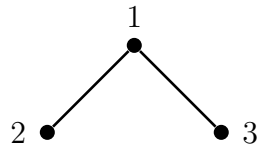
$$L^{(\mathcal{G})}_{i,i'} := \begin{cases} -A_{i,i'} & \text{for } i \neq i', \{i, i'\} \in \mathcal{E} \\ \sum_{i'' \neq i} A_{i,i''} & \text{for } i = i' \\ 0 & \text{else.} \end{cases} \tag{38}$$

The Laplacian matrix is symmetric and psd, which follows from the identity

$$\mathbf{w}^T (\mathbf{L}^{(\mathcal{G})} \otimes \mathbf{I}) \mathbf{w} = \sum_{\{i,i'\} \in \mathcal{E}} A_{i,i'} \left\| \mathbf{w}^{(i)} - \mathbf{w}^{(i')} \right\|_2^2$$

$$\text{for any } d \in \mathbb{N}, \mathbf{w} := \underbrace{\left(\left(\mathbf{w}^{(1)}\right)^T, \ldots, \left(\mathbf{w}^{(n)}\right)^T \right)^T}_{=:\text{stack}\left\{\mathbf{w}^{(i)}\right\}_{i=1}^n} \in \mathbb{R}^{d \cdot n}. \tag{39}$$

[5] The study of graphs via the eigenvalues and eigenvectors of associated matrices is the main subject of spectral graph theory [38, 39].



$$\mathbf{L}^{(\mathcal{G})} = \begin{pmatrix} 2 & -1 & -1 \\ -1 & 1 & 0 \\ -1 & 0 & 1 \end{pmatrix}$$

Figure 3.3: Left: Example of an FL network $\mathcal{G}$ with three nodes $i = 1, 2, 3$. Right: Laplacian matrix $\mathbf{L}^{(\mathcal{G})} \in \mathbb{R}^{3\times 3}$ of $\mathcal{G}$.

As a psd matrix, $\mathbf{L}^{(\mathcal{G})}$ possesses an EVD

$$\mathbf{L}^{(\mathcal{G})} = \sum_{i=1}^{n} \lambda_i \mathbf{u}^{(i)} \big(\mathbf{u}^{(i)}\big)^T, \tag{40}$$

with orthonormal eigenvectors $\mathbf{u}^{(1)}, \ldots, \mathbf{u}^{(n)}$ and corresponding list of eigenvalues

$$0 = \lambda_1\big(\mathbf{L}^{(\mathcal{G})}\big) \leq \lambda_2\big(\mathbf{L}^{(\mathcal{G})}\big) \leq \ldots \leq \lambda_n\big(\mathbf{L}^{(\mathcal{G})}\big). \tag{41}$$

We just write λ_i instead of $\lambda_i\big(\mathbf{L}^{(\mathcal{G})}\big)$ if the Laplacian matrix $\mathbf{L}^{(\mathcal{G})}$ is clear from context. The eigenvalue $\lambda_i\big(\mathbf{L}^{(\mathcal{G})}\big)$ corresponds to the eigenvector $\mathbf{u}^{(i)}$, i.e., $\mathbf{L}^{(\mathcal{G})}\mathbf{u}^{(i)} = \lambda_i\big(\mathbf{L}^{(\mathcal{G})}\big)\mathbf{u}^{(i)}$ for $i = 1, \ldots, n$.

It is important to note that the ordered list of eigenvalues (41) is uniquely determined for a given Laplacian matrix. In contrast, the eigenvectors $\mathbf{u}^{(i)}$ in (40) are not unique in general.[6]

The ordered eigenvalues $\lambda_i\big(\mathbf{L}^{(\mathcal{G})}\big)$ in (41) can be computed (or characterized) via the Courant–Fischer–Weyl min-max characterization (CFW) [3, Thm.

[6]Consider the scenario where the list (41) contains repeated entries, i.e., several ordered eigenvalues are identical.

8.1.2.]. Two important special cases of this characterization are [38, 39]

$$\lambda_n\big(\mathbf{L}^{(\mathcal{G})}\big) \overset{\text{CFW}}{=} \max_{\substack{\mathbf{v}\in\mathbb{R}^n \\ \|\mathbf{v}\|=1}} \mathbf{v}^T\mathbf{L}^{(\mathcal{G})}\mathbf{v}$$
$$\overset{(39)}{=} \max_{\substack{\mathbf{v}\in\mathbb{R}^n \\ \|\mathbf{v}\|=1}} \sum_{\{i,i'\}\in\mathcal{E}} A_{i,i'}\big(v_i - v_{i'}\big)^2 \tag{42}$$

and

$$\lambda_2\big(\mathbf{L}^{(\mathcal{G})}\big) \overset{\text{CFW}}{=} \min_{\substack{\mathbf{v}\in\mathbb{R}^n \\ \mathbf{v}^T\mathbf{1}=0 \\ \|\mathbf{v}\|=1}} \mathbf{v}^T\mathbf{L}^{(\mathcal{G})}\mathbf{v}$$
$$\overset{(39)}{=} \min_{\substack{\mathbf{v}\in\mathbb{R}^n \\ \mathbf{v}^T\mathbf{1}=0 \\ \|\mathbf{v}\|=1}} \sum_{\{i,i'\}\in\mathcal{E}} A_{i,i'}\big(v_i - v_{i'}\big)^2. \tag{43}$$

By (39), we can compute the GTV of a collection of model parameters via the quadratic form $\mathbf{w}^T\big(\mathbf{L}^{(\mathcal{G})} \otimes \mathbf{I}_{d\times d}\big)\mathbf{w}$. This quadratic form involves the vector $\mathbf{w} \in \mathbb{R}^{nd}$ which is obtained by stacking the local model parameters $\mathbf{w}^{(i)}$ for $i = 1, \ldots, n$. Another consequence of (39) is that any collection of identical local model parameters, stacked into the vector

$$\mathbf{w} = \text{stack}\{\mathbf{c}\} = \big(\mathbf{c}^T, \ldots, \mathbf{c}^T\big)^T, \text{ with some } \mathbf{c} \in \mathbb{R}^d, \tag{44}$$

is an eigenvector of $\mathbf{L}^{(\mathcal{G})} \otimes \mathbf{I}$ with corresponding eigenvalue $\lambda_1 = 0$ (see (41)). Thus, the Laplacian matrix of any FL network is singular (non-invertible).

The second eigenvalue λ_2 of the Laplacian matrix provides a great deal of information about the connectivity structure of $\mathcal{G}$.[7]

- Consider the case $\lambda_2 = 0$: Here, beside the eigenvector (44), we can find at least one additional eigenvector

$$\widetilde{\mathbf{w}} = \text{stack}\big\{\mathbf{w}^{(i)}\big\}_{i=1}^n \text{ with } \mathbf{w}^{(i)} \neq \mathbf{w}^{(i')} \text{ for some } i, i' \in \mathcal{V}, \tag{45}$$

[7]Much of spectral graph theory is devoted to the analysis of λ_2 for different graph constructions [38, 39].

of $\mathbf{L}^{(\mathcal{G})} \otimes \mathbf{I}$ with eigenvalue equal to 0. In this case, the graph $\mathcal{G}$ is not connected graph, i.e., we can find two subsets (components) of nodes that do not have any edge between them (see Fig. 3.4). For each connected component $\mathcal{C}$, we can construct the eigenvector by assigning the same (non-zero) vector $\mathbf{c} \in \mathbb{R}^d \setminus \{\mathbf{0}\}$ to all nodes $i \in \mathcal{C}$ and the zero vector $\mathbf{0}$ to the remaining nodes $i \in \mathcal{V} \setminus \mathcal{C}$.

- On the other hand, if $\lambda_2 > 0$ then $\mathcal{G}$ is connected graph. Moreover, the larger the value of λ_2, the stronger the connectivity between the nodes in $\mathcal{G}$. Indeed, adding edges to $\mathcal{G}$ can only increase the objective in (43) and, in turn, λ_2.

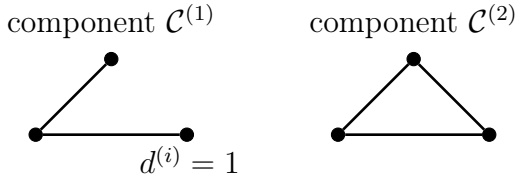


Figure 3.4: FL network with graph $\mathcal{G}$ consisting of $n=6$ nodes that form two connected components $\mathcal{C}^{(1)}, \mathcal{C}^{(2)}$.

In what follows, we will make use of the lower bound [40]

$$\sum_{\{i,i'\} \in \mathcal{E}} A_{i,i'} \left\| \mathbf{w}^{(i)} - \mathbf{w}^{(i')} \right\|_2^2 \geq \lambda_2 \sum_{i=1}^{n} \left\| \mathbf{w}^{(i)} - \text{avg}\{\mathbf{w}^{(i)}\} \right\|_2^2. \tag{46}$$

Here, $\text{avg}\{\mathbf{w}^{(i)}\} := (1/n) \sum_{i=1}^{n} \mathbf{w}^{(i)}$ is the average of all local model parameters. The bound (46) follows from (39) and the CFW for the eigenvalues of the matrix $\mathbf{L}^{(\mathcal{G})} \otimes \mathbf{I}$.

The quantity $\sum_{i=1}^{n} \left\| \mathbf{w}^{(i)} - \text{avg}\{\mathbf{w}^{(i)}\}_{i=1}^{n} \right\|_2^2$ on the right-hand side of (46) has an interesting geometric interpretation: It is the squared Euclidean norm of

the projection of the stacked local model parameters $\mathbf{w} := \left(\left(\mathbf{w}^{(1)}\right)^T, \ldots, \left(\mathbf{w}^{(n)}\right)^T \right)^T$ onto the orthogonal complement of the subspace

$$\mathcal{S} := \left\{ \mathbf{1} \otimes \mathbf{a} : \mathbf{a} \in \mathbb{R}^d \right\} = \left\{ \left(\mathbf{a}^T, \ldots, \mathbf{a}^T\right)^T, \text{ for some } \mathbf{a} \in \mathbb{R}^d \right\} \subseteq \mathbb{R}^{dn}. \quad (47)$$

The subspace $\mathcal{S}$ consists of stacked local model parameters $\mathbf{w}^{(i)}$ that are identical for all nodes $i = 1, \ldots, n$. Some FL applications involve a single (global) model whose model parameters $\mathbf{a} \in \mathbb{R}^d$ are shared among all nodes $i = 1, \ldots, n$, i.e., $\mathbf{w}^{(i)} = \mathbf{a}$ for all $i = 1, \ldots, n$ (see Sec. 6.2). We can represent this single-model setting by the condition $\left(\mathbf{w}^{(1)}, \ldots, \mathbf{w}^{(n)} \right)^T \in \mathcal{S}$.

The projection $\mathbf{P}_{\mathcal{S}}\mathbf{w}$ of $\mathbf{w} \in \mathbb{R}^{nd}$ on $\mathcal{S}$ is

$$\mathbf{P}_{\mathcal{S}}\mathbf{w} = \left(\mathbf{a}^T, \ldots, \mathbf{a}^T\right)^T, \text{ with } \mathbf{a} = \text{avg}\{\mathbf{w}^{(i)}\}_{i=1}^n. \quad (48)$$

The projection on the orthogonal complement $\mathcal{S}^{\perp}$, in turn, is

$$\mathbf{P}_{\mathcal{S}^{\perp}}\mathbf{w} = \mathbf{w} - \mathbf{P}_{\mathcal{S}}\mathbf{w} = \text{stack}\left\{ \mathbf{w}^{(i)} - \text{avg}\{\mathbf{w}^{(i)}\}_{i=1}^n \right\}_{i=1}^n. \quad (49)$$

3.4 Generalized Total Variation Minimization

Consider some FL network $\mathcal{G}$ with nodes $i \in \mathcal{V}$ representing devices that learn personalized model parameters $\mathbf{w}^{(i)}$. The usefulness of a specific choice of the model parameters $\mathbf{w}^{(i)}$ is measured by a local loss function $L_i\left(\mathbf{w}^{(i)}\right)$. We are mainly interested in FL applications where the local loss functions do not provide enough information for learning accurate model parameters.[8] We

[8] For example, the local loss function can be obtained from the training error on a local dataset that is much too small relative to the effective dimension of the local model (see Sec. 2.6).

therefore require learnt model parameters to not only incur a small local loss but also to have a small GTV (34).

GTV minimization (GTVMin) optimally balances the (average) local loss and the GTV of local model parameters $\mathbf{w}^{(i)}$,

$$\big\{\widehat{\mathbf{w}}^{(i)}\big\}_{i=1}^{n} \in \operatorname*{argmin}_{\mathrm{stack}\left\{\mathbf{w}^{(i)}\right\}_{i=1}^{n}} \sum_{i\in\mathcal{V}} L_i\left(\mathbf{w}^{(i)}\right) + \alpha \sum_{\{i,i'\}\in\mathcal{E}} A_{i,i'}\phi\big(\mathbf{w}^{(i)}-\mathbf{w}^{(i')}\big). \tag{50}$$

Our main focus will be on the special case of (50), obtained with $\phi(\cdot) := \|\cdot\|_2^2$,

$$\big\{\widehat{\mathbf{w}}^{(i)}\big\}_{i=1}^{n} \in \operatorname*{argmin}_{\mathbf{w}^{(1)},\ldots,\mathbf{w}^{(n)}} \sum_{i\in\mathcal{V}} L_i\left(\mathbf{w}^{(i)}\right) + \alpha \sum_{\{i,i'\}\in\mathcal{E}} A_{i,i'}\left\|\mathbf{w}^{(i)}-\mathbf{w}^{(i')}\right\|_2^2. \tag{51}$$

Note that GTVMin is an instance of RERM: The regularizer is the GTV of local model parameters over the weighted edges $A_{i,i'}$ of the FL network. Clearly, the FL network is an important design choice in GTVMin-based methods. This choice can be guided by computational aspects and statistical aspects of GTVMin-based FL systems.

Some application domains allow to leverage domain expertise to guess a useful choice for the FL network. If local datasets are generated at different geographic locations, we might use nearest-neighbour graphs based on geodesic distances between data generators (e.g., FMI weather stations). Chapter 7 will also discuss graph learning methods that determine the edge weights $A_{i,i'}$ in a data-driven fashion, i.e., directly from the local datasets $\mathcal{D}^{(i)}, \mathcal{D}^{(i')}$.

Let us now consider the special case of GTVMin with local models being a linear model. For each node $i \in \mathcal{V}$ of the FL network, we want to learn the parameters $\mathbf{w}^{(i)}$ of a linear hypothesis $h^{(i)}(\mathbf{x}) := \big(\mathbf{w}^{(i)}\big)^T\mathbf{x}$. We measure the

quality of the parameters via the average squared error loss

$$L_i\left(\mathbf{w}^{(i)}\right) := (1/m_i)\sum_{r=1}^{m_i}\left(y^{(i,r)} - \left(\mathbf{w}^{(i)}\right)^T\mathbf{x}^{(i,r)}\right)^2$$
$$\stackrel{(33)}{=} (1/m_i)\left\|\mathbf{y}^{(i)} - \mathbf{X}^{(i)}\mathbf{w}^{(i)}\right\|_2^2. \quad (52)$$

Inserting (52) into (51), yields the following instance of GTVMin to train local linear models,

$$\left\{\widehat{\mathbf{w}}^{(i)}\right\} \in \operatorname*{argmin}_{\{\mathbf{w}^{(i)}\}_{i=1}^n} \sum_{i\in\mathcal{V}}(1/m_i)\left\|\mathbf{y}^{(i)}-\mathbf{X}^{(i)}\mathbf{w}^{(i)}\right\|_2^2 + \alpha\sum_{\{i,i'\}\in\mathcal{E}}A_{i,i'}\left\|\mathbf{w}^{(i)}-\mathbf{w}^{(i')}\right\|_2^2. \quad (53)$$

The identity (39) allows to rewrite (53) using the Laplacian matrix $\mathbf{L}^{(\mathcal{G})}$ as

$$\widehat{\mathbf{w}}^{(i)} \in \operatorname*{argmin}_{\mathbf{w}=\mathrm{stack}\left\{\mathbf{w}^{(i)}\right\}} \sum_{i\in\mathcal{V}}(1/m_i)\left\|\mathbf{y}^{(i)}-\mathbf{X}^{(i)}\mathbf{w}^{(i)}\right\|_2^2 + \alpha\mathbf{w}^T\left(\mathbf{L}^{(\mathcal{G})}\otimes\mathbf{I}_d\right)\mathbf{w}. \quad (54)$$

Let us rewrite the objective function in (54) as

$$\mathbf{w}^T\left(\begin{pmatrix}\mathbf{Q}^{(1)} & \cdots & 0\\ \vdots & \ddots & \vdots\\ 0 & \cdots & \mathbf{Q}^{(n)}\end{pmatrix} + \alpha\mathbf{L}^{(\mathcal{G})}\otimes\mathbf{I}\right)\mathbf{w} + \left(\left(\mathbf{q}^{(1)}\right)^T,\ldots,\left(\mathbf{q}^{(n)}\right)^T\right)\mathbf{w} \quad (55)$$

with $\mathbf{Q}^{(i)} = (1/m_i)\left(\mathbf{X}^{(i)}\right)^T\mathbf{X}^{(i)}$ and $\mathbf{q}^{(i)} := (-2/m_i)\left(\mathbf{X}^{(i)}\right)^T\mathbf{y}^{(i)}$.

Thus, like linear regression (7) and ridge regression (28), GTVMin (54) (for local linear models $\mathcal{H}^{(i)}$) minimizes a convex quadratic function,

$$\left\{\widehat{\mathbf{w}}^{(i)}\right\}_{i=1}^n \in \operatorname*{argmin}_{\mathbf{w}=\mathrm{stack}\left\{\mathbf{w}^{(i)}\right\}_{i=1}^n} \mathbf{w}^T\mathbf{Q}\mathbf{w} + \mathbf{q}^T\mathbf{w}. \quad (56)$$

Here, we used the psd matrix

$$\mathbf{Q} := \begin{pmatrix}\mathbf{Q}^{(1)} & \cdots & \mathbf{0}\\ \vdots & \ddots & \vdots\\ \mathbf{0} & \cdots & \mathbf{Q}^{(n)}\end{pmatrix} + \alpha\mathbf{L}^{(\mathcal{G})}\otimes\mathbf{I} \text{ with } \mathbf{Q}^{(i)} := (1/m_i)\left(\mathbf{X}^{(i)}\right)^T\mathbf{X}^{(i)} \quad (57)$$

and the vector

$$\mathbf{q} := \big(\big(\mathbf{q}^{(1)}\big)^T, \ldots, \big(\mathbf{q}^{(n)}\big)^T\big)^T, \text{ with } \mathbf{q}^{(i)} := (-2/m_i)\big(\mathbf{X}^{(i)}\big)^T\mathbf{y}^{(i)}. \quad (58)$$

3.4.1 Computational Aspects of GTVMin

Chapter 5 will apply optimization methods to solve GTVMin, resulting in practical FL algorithms. Different instances of GTVMin favour different classes of optimization methods. For example, using a differentiable loss function allows to apply gradient-based methods (see Chapter 4) to solve GTVMin. Another important class of loss functions are those for which we can efficiently compute the proximal operator

$$\mathbf{prox}_{L,\rho}(\mathbf{w}) := \operatorname*{argmin}_{\mathbf{w}' \in \mathbb{R}^d} L(\mathbf{w}') + (\rho/2)\,\|\mathbf{w} - \mathbf{w}'\|_2^2 \text{ for some } \rho > 0.$$

We refer to functions L for which $\mathbf{prox}_{L,\rho}(\mathbf{w})$ can be computed easily as *simple* or proximable [41]. GTVMin with proximable loss functions can be solved via proximal algorithms [42].

Besides influencing the choice of optimization method, the design choices underlying GTVMin also determine the amount of computation that is required by a given optimization method.

Chapter 5 discusses FL algorithms that are obtained by applying fixed-point iterations to solve GTVMin. These fixed-point iterations repeatedly apply a fixed-point operator which is determined by the FL network (including the choice for the local loss functions, local models and edges in the FL network). The computational complexity of the resulting iterative method has two factors: (i) the amount of computation required by a single iteration (i.e., the per-iteration complexity) and (ii) the number iterations required

by the method to achieve a sufficiently accurate approximate solution of GTVMin.

The fixed-point iterations used in Chapter 5 to design FL algorithms can be implemented as message passing over the FL network. These algorithms require an amount of computation that is proportional to the number of edges of the FL network. Clearly, using an FL network with few edges (i.e., using a sparse graph) results in a smaller per-iteration complexity.

The number of iterations required by a FL algorithm employing a fixed-point operator $\mathcal{F}$ depends on the contraction properties of $\mathcal{F}$. These contraction properties can be influenced through design choices for the FL network, such as selecting local loss functions that are strongly convex. In addition to affecting the iteration count, the contraction properties of $\mathcal{F}$ also play a crucial role in determining whether the FL algorithm can tolerate asynchronous execution (see Sec. 5.9).

It is instructive to study the computational aspects of the special case of GTVMin (53) for local linear models. As discussed above, this instance is equivalent to solving (56). Any solution $\widehat{\mathbf{w}}$ of (56) (and, in turn, (53)) is characterized by the zero-gradient condition

$$\mathbf{Q}\widehat{\mathbf{w}} = -(1/2)\mathbf{q}, \tag{59}$$

with $\mathbf{Q}, \mathbf{q}$ as defined in (57) and (58). If the matrix $\mathbf{Q}$ in (59) is invertible, the solution to (59) and, in turn, to the GTVMin instance (53) is unique and given by $\widehat{\mathbf{w}} = (-1/2)\mathbf{Q}^{-1}\mathbf{q}$.

The size of the matrix $\mathbf{Q}$ (see (57)) is proportional to the number of nodes in the FL network $\mathcal{G}$ which might be in the order of millions (or even billions) for internet-scale applications. For such large systems, we typically cannot use

direct matrix inversion methods (such as Gaussian elimination) to compute $\mathbf{Q}^{-1}$.[9] Instead, we typically need to resort to iterative methods [43, 44].

One important family of such iterative methods are the gradient-based methods which we will discuss in Chapter 4. Starting from an initial choice of the local model parameters $\widehat{\mathbf{w}}_0 = \big(\widehat{\mathbf{w}}_0^{(1)}, \ldots, \widehat{\mathbf{w}}_0^{(n)}\big)$, these methods repeat (variants of) the gradient step,

$$\widehat{\mathbf{w}}_{k+1} := \widehat{\mathbf{w}}_k - \eta\big(2\mathbf{Q}\widehat{\mathbf{w}}_k + \mathbf{q}\big) \text{ for } k = 0, 1, \ldots.$$

The gradient step results in the updated local model parameters $\widehat{\mathbf{w}}^{(i)}$ which we stacked into

$$\widehat{\mathbf{w}}_{k+1} := \Big(\big(\widehat{\mathbf{w}}^{(1)}\big)^T, \ldots, \big(\widehat{\mathbf{w}}^{(n)}\big)^T\Big)^T.$$

We repeat the gradient step for a sufficient number of times, according to some stopping criterion (see Chapter 4).

3.4.2 Statistical Aspects of GTVMin

How useful are the solutions of GTVMin (51) as a choice for the local model parameters? To answer this question, we use - as for the statistical analysis of ERM in Chapter 2 - a probabilistic model for the local datasets. In particular, we use a variant of an i.i.d. assumption: Each local dataset $\mathcal{D}^{(i)}$ consists of data points whose features and labels are realizations of i.i.d. RVs

$$\mathbf{y}^{(i)} = \underbrace{\big(\mathbf{x}^{(i,1)}, \ldots, \mathbf{x}^{(i,m_i)}\big)^T}_{\text{local feature matrix } \mathbf{X}^{(i)}} \overline{\mathbf{w}}^{(i)} + \boldsymbol{\varepsilon}^{(i)} \text{ with } \mathbf{x}^{(i,r)} \overset{\text{i.i.d.}}{\sim} \mathcal{N}(\mathbf{0}, \mathbf{I}), \boldsymbol{\varepsilon}^{(i)} \sim \mathcal{N}(\mathbf{0}, \sigma^2\mathbf{I}). \quad (60)$$

[9] How many arithmetic operations (addition, multiplication) do you think are required to invert an arbitrary matrix $\mathbf{Q} \in \mathbb{R}^{d \times d}$?

In contrast to the probabilistic model (13) (which we already used for the analysis of ERM), the probabilistic model (60) allows for different node-specific parameters $\overline{\mathbf{w}}^{(i)}$, for $i \in \mathcal{V}$. In particular, the entire dataset obtained from pooling all local datasets does not conform to an i.i.d. assumption.

In what follows, we focus on the GTVMin instance (53) to learn the parameters $\mathbf{w}^{(i)}$ of a local linear model for each node $i \in \mathcal{V}$. For a reasonable choice of FL network, the parameters $\overline{\mathbf{w}}^{(i)}, \overline{\mathbf{w}}^{(i')}$ at connected nodes $\{i, i'\} \in \mathcal{E}$ should be similar. We cannot choose the edge weights based on parameters $\overline{\mathbf{w}}^{(i)}$ as they are unknown. However, we can still use estimates of $\overline{\mathbf{w}}^{(i)}$ that are computed from the available local datasets (see Chapter 7).

Consider an FL network with nodes carrying local datasets generated from the probabilistic model (60) with true model parameters $\overline{\mathbf{w}}^{(i)}$. For ease of exposition, we assume that

$$\overline{\mathbf{w}}^{(i)} = \overline{\mathbf{c}}, \text{ for some } \overline{\mathbf{c}} \in \mathbb{R}^d \text{ and all } i \in \mathcal{V}. \tag{61}$$

To study the deviation between the solutions $\widehat{\mathbf{w}}^{(i)}$ of (53) and the true underlying parameters $\overline{\mathbf{w}}^{(i)}$, we decompose it as

$$\widehat{\mathbf{w}}^{(i)} = \widetilde{\mathbf{w}}^{(i)} + \widehat{\mathbf{c}}, \text{ with } \widehat{\mathbf{c}} := (1/n) \sum_{i'=1}^{n} \widehat{\mathbf{w}}^{(i')}. \tag{62}$$

The component $\widehat{\mathbf{c}}$ is identical at all nodes $i \in \mathcal{V}$ and obtained as the orthogonal projection of $\widehat{\mathbf{w}} = \text{stack}\big\{\widehat{\mathbf{w}}^{(i)}\big\}_{i=1}^{n}$ on the subspace (47). The component $\widetilde{\mathbf{w}}^{(i)} := \widehat{\mathbf{w}}^{(i)} - (1/n)\sum_{i'=1}^{n} \widehat{\mathbf{w}}^{(i')}$ consists of the deviations, for each node i, between the GTVMin solution $\widehat{\mathbf{w}}^{(i)}$ and their average over all nodes. Trivially, the average of the deviations $\widetilde{\mathbf{w}}^{(i)}$ across all nodes is the zero vector, $(1/n)\sum_{i=1}^{n} \widetilde{\mathbf{w}}^{(i)} = \mathbf{0}$.

The decomposition (62) entails an analogous (orthogonal) decomposition of the error $\widehat{\mathbf{w}}^{(i)} - \overline{\mathbf{w}}^{(i)}$. Indeed, for identical true underlying model parameters (61) (which makes $\overline{\mathbf{w}}$ an element of the subspace (47)), we have

$$\sum_{i=1}^{n} \big\|\widehat{\mathbf{w}}^{(i)} - \overline{\mathbf{w}}^{(i)}\big\|_2^2 \overset{(61),(62)}{=} \underbrace{\sum_{i=1}^{n} \|\mathbf{c} - \widehat{\mathbf{c}}\|_2^2}_{n\|\mathbf{c}-\widehat{\mathbf{c}}\|_2^2} + \sum_{i=1}^{n} \big\|\widetilde{\mathbf{w}}^{(i)}\big\|_2^2 . \tag{63}$$

The following proposition provides an upper bound on the second error component in (63).

Proposition 3.1. *Consider a connected FL network, i.e., $\lambda_2 > 0$ (see (41)), and the solution (62) to GTVMin (53) for the local datasets (60). If the true local model parameters in (60) are identical (see (61)), we can upper bound the deviation $\widetilde{\mathbf{w}}^{(i)} := \widehat{\mathbf{w}}^{(i)} - (1/n)\sum_{i=1}^{n} \widehat{\mathbf{w}}^{(i)}$ of learnt model parameters $\widehat{\mathbf{w}}^{(i)}$ from their average, as*

$$\sum_{i=1}^{n} \big\|\widetilde{\mathbf{w}}^{(i)}\big\|_2^2 \leq \frac{1}{\lambda_2 \alpha} \sum_{i=1}^{n} (1/m_i) \big\|\boldsymbol{\varepsilon}^{(i)}\big\|_2^2 . \tag{64}$$

Proof. See Section 3.8.1. □

Note that Prop. 3.1 only applies to GTVMin over a FL network with a connected graph $\mathcal{G}$. A necessary and sufficient condition for $\mathcal{G}$ to be connected is that the second smallest eigenvalue is positive, $\lambda_2 > 0$. However, for an FL network with a graph $\mathcal{G}$ that is not connected, we can still apply Prop. 3.1 separately to each connected component of $\mathcal{G}$.

The upper bound (64) involves three components:

- the properties of local datasets, via the noise terms $\boldsymbol{\varepsilon}^{(i)}$ in (60),
- the FL network via the eigenvalue $\lambda_2\big(\mathbf{L}^{(\mathcal{G})}\big)$ (see (41)),

- the GTVMin parameter α.

According to (64), we can ensure a small error component $\widetilde{\mathbf{w}}^{(i)}$ of the GTVMin solution by choosing a large value α. Thus, by (63), for sufficiently large α, the local model parameters $\widehat{\mathbf{w}}^{(i)}$ delivered by GTVMin are approximately identical for all nodes $i \in \mathcal{V}$ of a connected FL network (where $\lambda_2\big(\mathbf{L}^{(\mathcal{G})}\big) > 0$).

Enforcing identical local model parameters at all nodes is desirable for FL applications that require to learn a common (global) model for all nodes [14]. However, some FL applications involve heterogeneous nodes that carry local datasets with significantly different statistics. For such applications it is detrimental to enforce a common model at all nodes (see Chapter 6). Rather,

3.5 How to Handle Non-Parametric Models

In its basic form (51), GTVMin can only be applied to parametric local models with model parameters belonging to the same Euclidean space $\mathbb{R}^d$. Some FL applications involve non-parametric local models (such as decision trees) or parametric local models with varying parametrizations (e.g., nodes use different deep net architectures). Here, we cannot use the difference between model parameters as a measure for the discrepancy between $h^{(i)}$ and $h^{(i')}$ across an edge $\{i, i'\} \in \mathcal{E}$.

One simple approach to measuring the discrepancy between hypothesis maps $h^{(i)}, h^{(i')}$ is to compare their predictions on a dataset

$$\mathcal{D}^{\{i,i'\}} = \Big\{\mathbf{x}^{(1)}, \ldots, \mathbf{x}^{(m')}\Big\}. \tag{65}$$

For each edge $\{i, i'\}$, the connected nodes need to agree on dataset $\mathcal{D}^{(\{i,i'\})}$. Note that the dataset $\mathcal{D}^{\{i,i'\}}$ can be different for different edges. Examples

for constructions of $\mathcal{D}^{\{i,i'\}}$ include i.i.d. realizations of some probability distribution or by using subsets of $\mathcal{D}^{(i)}$ and $\mathcal{D}^{(i')}$ (see Exercise 3.7).

We compare the predictions delivered by $h^{(i)}$ and $h^{(i')}$ on $\mathcal{D}^{\{i,i'\}}$ using some loss function L. In particular, we define the discrepancy measure

$$d^{(h^{(i)},h^{(i')})} := (1/m') \sum_{\mathbf{x} \in \mathcal{D}^{\{i,i'\}}} (1/2) \big[L\left(\big(\mathbf{x}, h^{(i)}(\mathbf{x})\big), h^{(i')} \right) + L\left(\big(\mathbf{x}, h^{(i')}(\mathbf{x})\big), h^{(i)} \right) \big]. \tag{66}$$

Different choices for the loss function in (66) result in different computational and statistical properties of the resulting FL algorithms (see Sec. 5.7). For real-valued predictions we can use the squared error loss in (66), yielding

$$d^{(h^{(i)},h^{(i')})} := (1/m') \sum_{\mathbf{x} \in \mathcal{D}^{\{i,i'\}}} \big[h^{(i)}(\mathbf{x}) - h^{(i')}(\mathbf{x}) \big]^2. \tag{67}$$

We can generalize GTVMin by replacing $\big\|\mathbf{w}^{(i)} - \mathbf{w}^{(i')}\big\|_2^2$ in (51) with the discrepancy $d^{(h^{(i)},h^{(i')})}$ (66) (or the special case (67)). This results in

$$\big\{\widehat{h}^{(i)}\big\}_{i=1}^{n} \in \operatorname*{argmin}_{\substack{h^{(i)} \in \mathcal{H}^{(i)} \\ i \in \mathcal{V}}} \sum_{i \in \mathcal{V}} L_i\left(h^{(i)}\right) + \alpha \sum_{\{i,i'\} \in \mathcal{E}} A_{i,i'} d^{(h^{(i)},h^{(i')})}. \tag{68}$$

3.6 Interpretations

We next discuss some interpretations of GTVMin (50).

Empirical Risk Minimization. GTVMin (51) is obtained as a special case of ERM (1) for specific choices for the model $\mathcal{H}$ and loss function L. The model (or hypothesis space) used by GTVMin is a product space generated by the local models at the nodes of an FL network. The loss function of GTVMin consists of two parts: the sum of loss functions at each node and a

penalty term that measures the variation of local models across the edges of the FL network.

Generalized Convex Clustering. One important special case of GTVMin (50) is convex clustering [45, 46]. Indeed, convex clustering is obtained from (50) using the local loss function

$$L_i\left(\mathbf{w}^{(i)}\right) = \|\mathbf{w}^{(i)} - \mathbf{a}^{(i)}\|^2, \text{ for all nodes } i \in \mathcal{V} \tag{69}$$

and the GTV penalty function $\phi(\mathbf{u}) = \|\mathbf{u}\|_p$ with some $p \geq 1$.[10] The vectors $\mathbf{a}^{(i)}$, for $i = 1, \ldots, n$, are the features of data points that we wish to cluster in (69). Thus, we can interpret GTVMin as a generalization of convex clustering: we replace the terms $\|\mathbf{w}^{(i)} - \mathbf{a}^{(i)}\|^2$ with a more general local loss function.

Dual of Minimum-Cost Flow Problem. The optimization variables of GTVMin (50) are the local model parameters $\mathbf{w}^{(i)}$, for each node $i \in \mathcal{V}$ in an FL network $\mathcal{G}$. The optimization of node-wise variables $\mathbf{w}^{(i)}$, for $i = 1, \ldots, n$, is naturally associated with a dual problem [47]. This dual problem optimizes edge-wise variables $\mathbf{u}^{(\{i,i'\})}$, one for each edge $\{i, i'\} \in \mathcal{E}$ of $\mathcal{G}$,

$$\max_{\substack{\mathbf{u}^{(e)}, e\in\mathcal{E} \\ \mathbf{w}^{(i)}, i\in\mathcal{V}}} -\sum_{i\in\mathcal{V}} L_i^*\left(\mathbf{w}^{(i)}\right) - \alpha \sum_{e\in\mathcal{E}} A_e \phi^*\left(\mathbf{u}^{(e)}/(\alpha A_e)\right) \tag{70}$$

$$\text{subject to } -\mathbf{w}^{(i)} = \sum_{\substack{e\in\mathcal{E} \\ e_+=i}} \mathbf{u}^{(e)} - \sum_{\substack{e\in\mathcal{E} \\ e_-=i}} \mathbf{u}^{(e)} \text{ for each } i \in \mathcal{V}. \tag{71}$$

Here, we have introduced an orientation for each edge $e := \{i, i'\}$, by defining the *head* $e_- := \min\{i, i'\}$ and the *tail* $e_+ := \max\{i, i'\}$.[11] Moreover, we used

[10] Here, we used the p-norm $\|\mathbf{u}\|_p := \left(\sum_{j=1}^d |u_j|^p\right)^{1/p}$ of a vector $\mathbf{u} \in \mathbb{R}^d$.

[11] We use this orientation only for notational convenience to formulate the dual of GTVMin. The orientation of an edge (by choosing a head and tail) has no practical

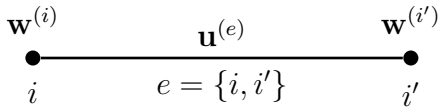


Figure 3.5: Two nodes of an FL network that are connected by an edge $e = \{i, i'\}$. GTVMin (50) optimizes local model parameters $\mathbf{w}^{(i)}$ for each node $i \in \mathcal{V}$ in the FL network. The dual (70) of GTVMin optimizes local parameters $\mathbf{u}^{(e)}$ for each edge $e \in \mathcal{E}$ in the FL network.

the convex conjugates $L_i^*(\cdot), \phi^*$ of the local loss function $L_i(\cdot)$ and GTV penalty function ϕ.[12]

The dual problem (70) generalizes the optimal flow problem [47, Sec. 1J] to vector-valued flows. The special case of (70), obtained when the GTV penalty function ϕ is a norm, is equivalent to a generalized minimum-cost flow problem [49, Sec. 1.2.1]. Indeed, the maximization problem (70) is equivalent to the minimization

$$\begin{aligned} &\min_{\substack{\mathbf{u}^{(e)}, e\in\mathcal{E} \\ \mathbf{w}^{(i)}, i\in\mathcal{V}}} \sum_{i\in\mathcal{V}} L_i^*\left(\mathbf{w}^{(i)}\right) \\ &\text{subject to } -\mathbf{w}^{(i)} = \sum_{\substack{e\in\mathcal{E} \\ e_+=i}} \mathbf{u}^{(e)} - \sum_{\substack{e\in\mathcal{E} \\ e_-=i}} \mathbf{u}^{(e)} \text{ for each node } i \in \mathcal{V} \\ &\quad \|\mathbf{u}^{(e)}\|_* \leq \alpha A_e \text{ for each edge } e \in \mathcal{E}. \end{aligned} \tag{73}$$

meaning in terms of GTVMin-based FL algorithms. After all, GTVMin (50) and its dual (70) are defined for an FL network with undirected edges $\mathcal{E}$.

[12]The convex conjugate of a function $f : \mathbb{R}^d \to \mathbb{R}$ is defined as [48]

$$f^*(\mathbf{x}) := \sup_{\mathbf{z}\in\mathbb{R}^d} \mathbf{x}^T\mathbf{z} - f(\mathbf{z}). \tag{72}$$

The optimization problem (73) reduces to the minimum-cost flow problem [49, Eq. (1.3) - (1.5)] for scalar local model parameters $\mathbf{w}^{(i)} \in \mathbb{R}$.

Locally Weighted Learning. The solution of GTVMin are local model parameters $\widehat{\mathbf{w}}^{(i)}$ that tend to be clustered: Each node $i \in \mathcal{V}$ belongs to a subset or cluster $\mathcal{C} \subseteq \mathcal{V}$. All the nodes in $\mathcal{C}$ have nearly identical local model parameters, $\widehat{\mathbf{w}}^{(i')} \approx \overline{\mathbf{w}}^{(\mathcal{C})}$ for all $i' \in \mathcal{C}$ [37]. The cluster-wise model parameters $\overline{\mathbf{w}}^{(\mathcal{C})}$ are the solutions of

$$\min_{\mathbf{w}} \sum_{i' \in \mathcal{C}} L_{i'}\left(\mathbf{w}\right), \tag{74}$$

which, in turn, is an instance of a locally weighted learning problem [50, Sec. 3.1.2]

$$\overline{\mathbf{w}}^{(\mathcal{C})} = \operatorname*{argmin}_{\mathbf{w} \in \mathbb{R}^d} \sum_{i' \in \mathcal{V}} \rho_{i'} L_{i'}\left(\mathbf{w}\right). \tag{75}$$

Indeed, we obtain (74) from (75) by setting the weights $\rho_{i'}$ equal to 1 if $i' \in \mathcal{C}$ and 0 otherwise.

3.7 Exercises

3.1. Spectral Radius of Laplacian Matrix. The spectral radius $\rho(\mathbf{Q})$ of a square matrix $\mathbf{Q}$ is the largest magnitude of an eigenvalue,

$$\rho(\mathbf{Q}) := \max\{|\lambda| : \lambda \text{ is an eigenvalue of } \mathbf{Q}\}.$$

Consider the Laplacian matrix $\mathbf{L}^{(\mathcal{G})}$ of an FL network with undirected graph $\mathcal{G}$. Show that $\rho\big(\mathbf{L}^{(\mathcal{G})}\big) = \lambda_n\big(\mathbf{L}^{(\mathcal{G})}\big)$ and verify the upper bound $\lambda_n\big(\mathbf{L}^{(\mathcal{G})}\big) \leq 2d_{\max}^{(\mathcal{G})}$. Try to find a graph $\mathcal{G}$ such that $\lambda_n\big(\mathbf{L}^{(\mathcal{G})}\big) \approx 2d_{\max}^{(\mathcal{G})}$.

3.2. Kernel of the Laplacian matrix. Consider an undirected weighted graph $\mathcal{G}$ with Laplacian matrix $\mathbf{L}^{(\mathcal{G})}$. A component of $\mathcal{G}$ is a subset $\mathcal{C} \subseteq \mathcal{V}$ of nodes that are connected but there is no edge between $\mathcal{C}$ and the rest $\mathcal{V} \setminus \mathcal{C}$. The *nullspace* (or *kernel*) of $\mathbf{L}^{(\mathcal{G})}$ is the subspace $\mathcal{K} \subseteq \mathbb{R}^n$ constituted by all vectors $\mathbf{v} \in \mathbb{R}^n$ such that $\mathbf{L}^{(\mathcal{G})}\mathbf{v} = \mathbf{0}$. Show that the dimension of $\mathcal{K}$ coincides with the number of components in $\mathcal{G}$.

3.3. Toy Example of Spectral Clustering. Consider the graph $\mathcal{G}$ depicted in Figure 3.6. The Laplacian matrix has two zero eigenvalues $\lambda_1 = \lambda_2 = 0$.

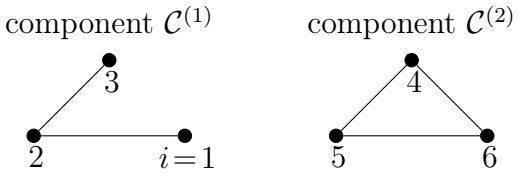


Figure 3.6: An undirected graph $\mathcal{G}$ that consists of two connected components $\mathcal{C}^{(1)}, \mathcal{C}^{(2)}$.

Can you find corresponding orthonormal eigenvectors $\mathbf{u}^{(1)}, \mathbf{u}^{(2)}$? Are they unique?

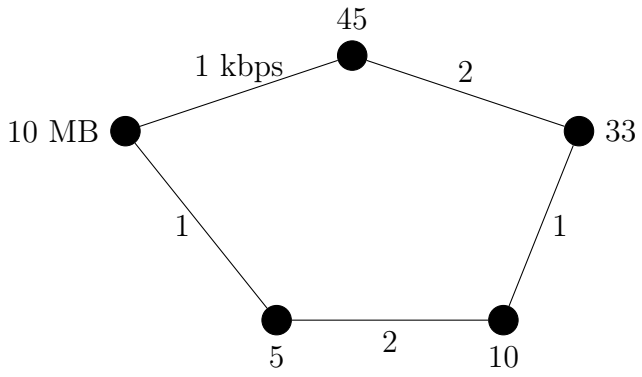


Figure 3.7: An FL network whose nodes $i = 1, \ldots, 5$ represent devices that hold local datasets whose size is indicated next to each node.

3.4. Adding an Edge Increases Connectivity. Consider an undirected weighted graph $\mathcal{G}$ with Laplacian matrix $\mathbf{L}^{(\mathcal{G})}$. We construct a new graph $\mathcal{G}'$, with Laplacian matrix $\mathbf{L}^{(\mathcal{G}')}$, by adding a new edge to $\mathcal{G}$. Show that $\lambda_2(\mathcal{G}') \geq \lambda_2(\mathcal{G})$, i.e., the second smallest eigenvalue of $\mathbf{L}^{(\mathcal{G}')}$ is at least as large as the second smallest eigenvalue of $\mathbf{L}^{(\mathcal{G})}$.

3.5. Capacity of an FL network. Consider the FL network shown in Fig. 3.7. Each node holds a local dataset, with its size indicated by the adjacent numbers. The devices (nodes) communicate over bi-directional links, whose capacities are specified by the numbers next to the edges. What is the minimum time required for the leftmost node to collect all local datasets from the other nodes?

3.6. Discrepancy Measures. Consider an FL network with nodes carrying parametric local models, each having model parameters $\mathbf{w}^{(i)} \in \mathbb{R}^d$. Is it possible to construct a dataset $\mathcal{D}^{\{i,i'\}}$ such that (67) coincides with $\left\|\mathbf{w}^{(i)} - \mathbf{w}^{(i')}\right\|_2^2$?

3.7. Privacy-Friendly Discrepancy Measures. The discrepancy measure (66) requires to choose a test-set $\mathcal{D}^{\{i,i'\}}$. One possible choice is to combine data points of the local datasets $\mathcal{D}^{(i)}$ and $\mathcal{D}^{(i')}$. However, sharing these data points might be harmful in the sense of leaking sensitive information. Could you think of a simple message passing protocol between node i and i' that allows them to evaluate (66) only by sharing the predictions $h^{(i)}(\mathbf{x}), h^{(i')}(\mathbf{x})$ for $\mathbf{x} \in \mathcal{D}^{\{i,i'\}}$?

3.8. Structure of GTVMin. What are sufficient conditions for the local datasets and the edge weights used in GTVMin such that $\mathbf{Q}$ in (57) is invertible?

3.9. Existence and Uniqueness of GTVMin Solution. Consider the GTVMin instance (51), defined over an FL network with the weighted undirected graph $\mathcal{G}$.

1. **Existence.** Can you state a sufficient condition on the local loss functions and the weighted edges of $\mathcal{G}$ such that (51) has at least one solution?

2. **Uniqueness.** Then, try to find a condition that ensures that (51) has a unique solution.

3. Finally, try to find necessary conditions for the existence and uniqueness of solutions to (51).

3.10. Computing the Average. Consider an FL network with each nodes carrying a single model parameter $w^{(i)}$ and a local dataset, consisting of a single number $y^{(i)}$. Construct an instance of GTVMin such that its solutions are given by $\widehat{w}^{(i)} \approx (1/n) \sum_{i=1}^{n} y^{(i)}$ for all $i = 1, \ldots, n$.

3.11. Computing the Average over a Star. Consider the FL network depicted in Fig. 3.8, which consists of a centre node i_0 which is connected to $n-1$ peripheral nodes $\mathcal{P} := \mathcal{V} \setminus \{i_0\}$. Each peripheral node $i \in \mathcal{P}$ carries a local dataset that consists of a single real-valued observation $y^{(i)} \in \mathbb{R}$. Construct an instance of GTVMin, using real-valued local model parameters $w^{(i)} \in \mathbb{R}$, such that the solution satisfies $\widehat{w}^{(i_0)} \approx (1/(n-1)) \sum_{i \in \mathcal{P}} y^{(i)}$.

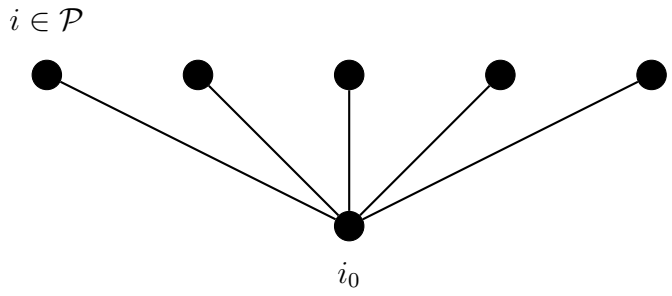


Figure 3.8: An FL network that consists of a centre node i_0 that is connected to several peripheral nodes $\mathcal{P} := \mathcal{V} \setminus \{i_0\}$.

3.12. Fundamental Limits. Consider the FL network depicted in Fig. 3.9. Each node carries a local model with single parameter $w^{(i)}$ as well as a local dataset that consists of a single number $y^{(i)}$. We use a probabilistic model for the local datasets: $y^{(i)} = \bar{w} + n^{(i)}$. Here, $\bar{w}$ is some fixed but unknown number and $n^{(i)} \sim \mathcal{N}(0,1)$ are i.i.d. Gaussian RVs. We use a message-passing FL algorithm to estimate c based on the local datasets. What is a fundamental limit on the accuracy of the estimate $\hat{c}^{(i)}$ delivered at some fixed node i by such an algorithm after two iterations? Compare this limit with the risk $\mathbb{E}\big\{\big(\hat{w}^{(i)} - \bar{w}\big)^2\big\}$ incurred by the estimate $\hat{w}^{(i)}$ delivered by running Algorithm 4 for two iterations.

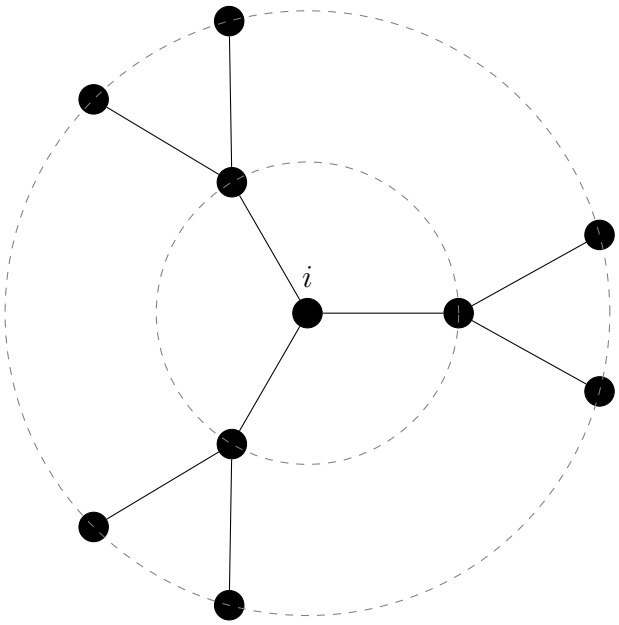


Figure 3.9: An FL network containing a node i having degree $d^{(i)} = 3$ like all its neighbours $\mathcal{N}^{(i)}$. We use an FL algorithm to learn local model parameters $w^{(i)}$. If the algorithm employs message passing, the first iteration provides access only to the local datasets of the neighbors in $\mathcal{N}^{(i)}$ (located along the inner dashed circle). In the second iteration, the algorithm gains access to the local datasets of the neighbors $\mathcal{N}^{(i')}$ of each $i' \in \mathcal{N}^{(i)}$. These *second-hop* neighbors are located along the outer dashed circle.

3.13. Counting Number of Paths. Consider an undirected graph $\mathcal{G}$ with each edge $\{i, i'\} \in \mathcal{E}$ having unit edge weight $A_{i,i'} = 1$. For $k \in \{1, 2, \ldots.\}$, a k-hop path between two nodes $i, i' \in \mathcal{V}$ is a node sequence $i^{(1)}, \ldots, i^{(k+1)}$ such that $i^{(1)} = i$, $i^{(k+1)} = i'$, and $\{i^{(r)}, i^{(r+1)}\} \in \mathcal{E}$ for reach $r = 1, \ldots, k$. Show that the number of k-hop paths between two nodes $i, i' \in \mathcal{V}$ is given by $\left(\mathbf{A}^k\right)_{i,i'}$.

3.8 Proofs

3.8.1 Proof of Proposition 3.1

Let us introduce the shorthand $f\big(\mathbf{w}^{(i)}\big)$ for the objective function of the GTVMin instance (53). We verify the bound (64) by showing that if it does not hold, the choice of the local model parameters $\mathbf{w}^{(i)} := \overline{\mathbf{w}}^{(i)}$ (see (60)) results in a smaller objective function value, $f\big(\overline{\mathbf{w}}^{(i)}\big) < f\big(\widehat{\mathbf{w}}^{(i)}\big)$. This would contradict the fact that $\widehat{\mathbf{w}}^{(i)}$ is a solution to (53).

First, note that

$$\begin{aligned} f\big(\overline{\mathbf{w}}^{(i)}\big) &= \sum_{i\in\mathcal{V}} (1/m_i) \big\|\mathbf{y}^{(i)} - \mathbf{X}^{(i)}\overline{\mathbf{w}}^{(i)}\big\|_2^2 + \alpha \sum_{\{i,i'\}\in\mathcal{E}} A_{i,i'} \Big\|\overline{\mathbf{w}}^{(i)} - \overline{\mathbf{w}}^{(i')}\Big\|_2^2 \\ &\overset{(61)}{=} \sum_{i\in\mathcal{V}} (1/m_i) \big\|\mathbf{y}^{(i)} - \mathbf{X}^{(i)}\overline{\mathbf{w}}^{(i)}\big\|_2^2 \\ &\overset{(60)}{=} \sum_{i\in\mathcal{V}} (1/m_i) \big\|\mathbf{X}^{(i)}\overline{\mathbf{w}}^{(i)} + \boldsymbol{\varepsilon}^{(i)} - \mathbf{X}^{(i)}\overline{\mathbf{w}}^{(i)}\big\|_2^2 \\ &= \sum_{i\in\mathcal{V}} (1/m_i) \big\|\boldsymbol{\varepsilon}^{(i)}\big\|_2^2 . \end{aligned} \tag{76}$$

Inserting (62) into (53),

$$\begin{aligned} f\big(\widehat{\mathbf{w}}^{(i)}\big) &= \underbrace{\sum_{i\in\mathcal{V}} (1/m_i) \big\|\mathbf{y}^{(i)} - \mathbf{X}^{(i)}\widehat{\mathbf{w}}^{(i)}\big\|_2^2}_{\geq 0} + \alpha \sum_{\{i,i'\}\in\mathcal{E}} A_{i,i'} \underbrace{\Big\|\widehat{\mathbf{w}}^{(i)} - \widehat{\mathbf{w}}^{(i')}\Big\|_2^2}_{\overset{(62)}{=}\big\|\widetilde{\mathbf{w}}^{(i)} - \widetilde{\mathbf{w}}^{(i')}\big\|_2^2} \\ &\geq \alpha \sum_{\{i,i'\}\in\mathcal{E}} A_{i,i'} \Big\|\widetilde{\mathbf{w}}^{(i)} - \widetilde{\mathbf{w}}^{(i')}\Big\|_2^2 \\ &\overset{(46)}{\geq} \alpha\lambda_2 \sum_{i=1}^{n} \big\|\widetilde{\mathbf{w}}^{(i)}\big\|_2^2 . \end{aligned} \tag{77}$$

If the bound (64) would not hold, then by (77) and (76) we would obtain $f\big(\widehat{\mathbf{w}}^{(i)}\big) > f\big(\overline{\mathbf{w}}^{(i)}\big)$. This is a contradiction to the fact that $\widehat{\mathbf{w}}^{(i)}$ solves (53).

.

4 Gradient Methods

Chapter 3 introduced GTVMin as a central design principle for FL methods. Many significant instances of GTVMin minimize a smooth objective function $f(\mathbf{w})$ over the parameter space (typically a subset of $\mathbb{R}^d$). This chapter explores gradient-based methods, a widely used family of iterative algorithms to find the minimum of a smooth function. These methods approximate the objective function locally using its gradient at the current choice of the model parameters. Chapter 5 focuses on FL algorithms obtained from applying gradient-based methods to solve GTVMin.

4.1 Learning Goals

After completing this chapter, you will

- have some intuition about the effect of a gradient step,
- understand the role of the step size (or learning rate),
- know some examples of a stopping criterion,
- be able to analyze the effect of perturbations in the gradient step,
- know about projected gradient descent (projected GD) to cope with constraints on model parameters.

4.2 Gradient Descent

Gradient-based methods are iterative algorithms for finding the minimum of a differentiable objective function $f(\mathbf{w})$ of a vector-valued argument $\mathbf{w}$ (e.g., the model parameters in a ML method). Unless stated otherwise, we consider an objective function of the form:

$$f(\mathbf{w}) := \mathbf{w}^T \mathbf{Q} \mathbf{w} + \mathbf{q}^T \mathbf{w}. \tag{78}$$

Although restricting our discussion to objective functions of the form (78) may seem limiting, this formulation allows for a straightforward analysis and generalization to larger classes of differentiable functions. Moreover, we can use the functions (78) also as an approximation for broader families of objective functions (see Section 4.6).

Note that (78) defines an entire family of convex quadratic functions $f(\mathbf{w})$. Each member of this family is specified by a psd matrix $\mathbf{Q} \in \mathbb{R}^{d \times d}$ and a vector $\mathbf{q} \in \mathbb{R}^d$. We have already encountered some ML and FL methods that minimize an objective function of the form (78): Linear regression (3) and ridge regression (28) in Chapter 2 as well as GTVMin (53) for local linear models in Chapter 3. Moreover, (78) is a useful approximation for the objective functions arising in larger classes of ML methods [51–53].

Given a current choice of model parameters $\mathbf{w}^{(k)}$, we want to update (or improve) them towards a minimum of (78). To this end, we use the gradient $\nabla f\big(\mathbf{w}^{(k)}\big)$ to locally approximate $f(\mathbf{w})$ (see Figure 4.1). The gradient $\nabla f\big(\mathbf{w}^{(k)}\big)$ indicates the direction in which the function $f(\mathbf{w})$ maximally increases. Therefore, it seems reasonable to update $\mathbf{w}^{(k)}$ in the opposite

direction of $\nabla f\big(\mathbf{w}^{(k)}\big)$,

$$\begin{aligned}\mathbf{w}^{(k+1)} &:= \mathbf{w}^{(k)} - \eta \nabla f\big(\mathbf{w}^{(k)}\big) \\ &\overset{(78)}{=} \mathbf{w}^{(k)} - \eta\big(2\mathbf{Q}\mathbf{w}^{(k)} + \mathbf{q}\big). \end{aligned} \tag{79}$$

The gradient step (79) involves the factor η which we refer to as step size or learning rate. Algorithm 2 summarizes the most basic variant of gradient-based methods, which simply iterates (79) until a predefined stopping criterion is met.

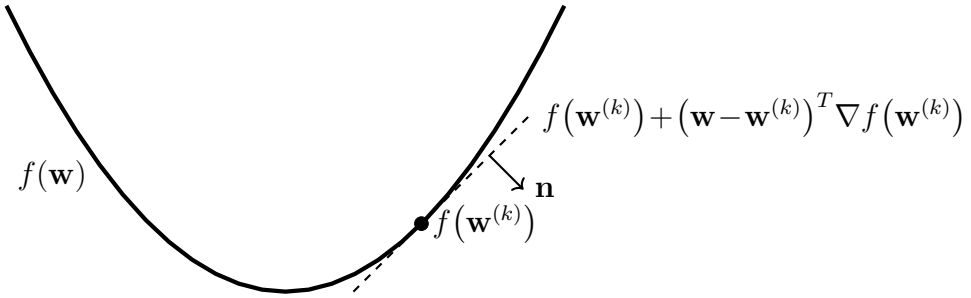


Figure 4.1: We can approximate a differentiable function $f(\mathbf{w})$ locally around a point $\mathbf{w}^{(k)} \in \mathbb{R}^d$ using the linear function $f\big(\mathbf{w}^{(k)}\big) + \big(\mathbf{w} - \mathbf{w}^{(k)}\big)^T \nabla f\big(\mathbf{w}^{(k)}\big)$. Geometrically, we approximate the graph of $f(\mathbf{w})$ by a hyperplane with normal vector $\mathbf{n} = (\nabla f\big(\mathbf{w}^{(k)}\big), -1)^T \in \mathbb{R}^{d+1}$ of this approximating hyperplane is determined by the gradient $\nabla f\big(\mathbf{w}^{(k)}\big)$ [2].

The usefulness of gradient-based methods crucially depends on the computational complexity of evaluating the gradient $\nabla f(\mathbf{w})$. Modern software libraries for automatic differentiation enable the efficient evaluation of the gradients arising in widely-used ERM-based methods [54].

Besides the actual computation of the gradient, it might already be

challenging to gather the required data points which define the objective function $f(\mathbf{w})$ (e.g., being the average loss over a large training set). Indeed, the matrix $\mathbf{Q}$ and vector $\mathbf{q}$ in (78) are constructed from the features and labels of data points in the training set. For example, the gradient of the objective function in ridge regression (28) is

$$\nabla f(\mathbf{w}) = -(2/m)\sum_{r=1}^{m} \mathbf{x}^{(r)}\big(y^{(r)} - \mathbf{w}^T\mathbf{x}^{(r)}\big) + 2\alpha\mathbf{w}.$$

Evaluating this gradient requires roughly $d \times m$ arithmetic operations (summations and multiplications).

Algorithm 2 A blueprint for gradient-based methods

Input: some objective function $f(\mathbf{w})$ (e.g., the average loss of a hypothesis $h^{(\mathbf{w})}$ on a training set); learning rate $\eta > 0$; some stopping criterion.

Initialize: set $\mathbf{w}^{(0)} := \mathbf{0}$; set iteration counter $k := 0$

1: **repeat**

2: $k := k+1$ (increase iteration counter)

3: $\mathbf{w}^{(k)} := \mathbf{w}^{(k-1)} - \eta\nabla f\big(\mathbf{w}^{(k-1)}\big)$ (do a gradient step (79))

4: **until** stopping criterion is met

Output: learnt model parameters $\widehat{\mathbf{w}} := \mathbf{w}^{(k)}$ (hopefully $f\big(\widehat{\mathbf{w}}\big) \approx \min_{\mathbf{w}} f(\mathbf{w})$)

Like most other gradient-based methods, Algorithm 2, involves two hyper-parameters: (i) the learning rate η used for the gradient step and (ii) a stopping criterion to decide when to stop repeating the gradient step. We next discuss how to choose these hyper-parameters.

Note that we can apply Algorithm 2 to find the minimum of any differentiable objective function $f(\mathbf{w})$. Indeed, Algorithm 2 only needs to be able

to access the gradient $\nabla f\big(\mathbf{w}^{(k-1)}\big)$. In particular, we an apply Algorithm 2 to objective functions that do not belong to the family of convex quadratic function (see (78)).

4.3 Learning Rate

The learning rate must not be too large to avoid moving away from the optimum by *overshooting* (see Figure 4.2-(a)). On the other hand, if the learning rate is too small, the gradient step makes too little progress towards the solutions of (78) (see Figure 4.2-(b)). In practice we can only afford to repeat the gradient step for a finite number of iterations.

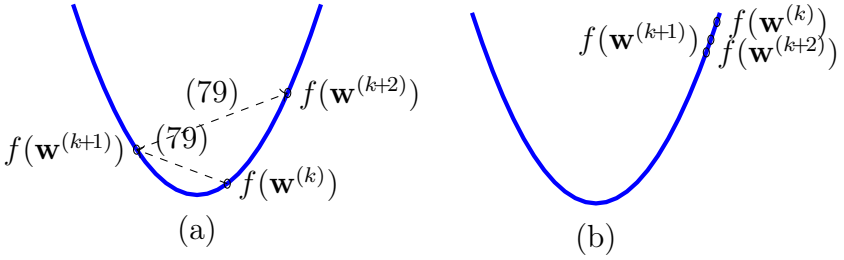


Figure 4.2: Effect of inadequate learning rates η in the gradient step (79). (a) If η is too large, the gradient steps might "overshoot" such that the iterates $\mathbf{w}^{(k)}$ might diverge from the optimum, i.e., $f(\mathbf{w}^{(k+1)}) > f(\mathbf{w}^{(k)})$! (b) If η is too small, the gradient steps make very little progress towards the optimum or even fail to reach the optimum at all.

One approach to choosing the learning rate is to start with some initial value (first guess) and monitor the decrease in the objective function. If this decrease does not agree with the decrease predicted by the (local linear approximation using the) gradient, we decrease the learning rate by a constant

factor. After we decrease the learning rate, we re-consider the decrease in the objective function. We repeat this procedure until a sufficient decrease in the objective function is achieved [55, Sec 6.1].

Alternatively, we can use a prescribed sequence (schedule) η_k, for $k = 1, 2, \ldots$, of learning rates that vary across successive gradient steps [56]. For example, we could require the learning rate η_k to satisfy a *diminishing step-size rule* [55, Sec. 6.1]

$$\lim_{k\to\infty} \eta_k = 0, \ \sum_{k=1}^{\infty} \eta_k = \infty \ , \text{ and } \ \sum_{k=1}^{\infty} \eta_k^2 < \infty. \tag{80}$$

Running the gradient step (79) with a learning rate schedule η_k that satisfies (80) ensures convergence to a minimum of $f(\mathbf{w})$ if

- the iterates $\left\|\mathbf{w}^{(k)}\right\|_2$ are bounded, i.e., $\sup_{k=1,\ldots} \left\|\mathbf{w}^{(k)}\right\|_2$ is finite, and
- the gradients $\left\|\nabla f\left(\mathbf{w}^{(k)}\right)\right\|_2$ are also bounded.

A detailed convergence proof can be found in [55, Sec. 3].

It is instructive to discuss the meanings of the individual conditions in (80). The first condition (80) requires that the learning rate eventually become sufficiently small to avoid overshooting. The third condition (80) ensures that this required decay of the learning rate does not take "forever". Note that the first and third condition in (80) could be satisfied by the trivial learning rate schedule $\eta_k = 0$ which is clearly not useful as the gradient step has no effect.

The trivial schedule $\eta_k = 0$ is ruled out by the middle condition of (80). This middle condition ensures that the learning rate η_k is large enough such that the gradient steps make sufficient progress towards a minimizer of the objective function.

We emphasize that the conditions in (80) are independent of any properties of the matrix $\mathbf{Q}$ in (78). The matrix $\mathbf{Q}$ is determined by data points (see, e.g., (3)), whose statistical properties can typically be controlled only to a limited extent, such as through data normalization.

4.4 When to Stop?

For the stopping criterion, we may use a fixed number of iterations, $k_{\max}$. This hyperparameter can be determined by constraints on computational resources. We can optimize the number of iterations also via meta-learning, i.e., trying to predict the optimal $k_{\max}$ based on key characteristics (or features) of the objective function [57].

Another stopping criterion can be obtained by monitoring the decrease in the objective function $f\big(\mathbf{w}^{(k)}\big)$. Specifically, we stop repeating the gradient step (79) when $\big|f\big(\mathbf{w}^{(k)}\big) - f\big(\mathbf{w}^{(k+1)}\big)\big| \leq \varepsilon^{(\mathrm{tol})}$ for a given tolerance $\varepsilon^{(\mathrm{tol})}$. As before, we can optimize the tolerance level $\varepsilon^{(\mathrm{tol})}$ via meta-learning techniques [57].

For an objective function of the form (78), we can use information about the psd matrix $\mathbf{Q}$ to construct a stopping criterion.[13] Indeed, the choice of the learning rate η and the stopping criterion can be guided by the eigenvalues

$$0 \leq \lambda_1(\mathbf{Q}) \leq \ldots \leq \lambda_d(\mathbf{Q}). \tag{81}$$

Even if we do not know these eigenvalues precisely, we might know (or be

[13] For linear regression (7), the matrix $\mathbf{Q}$ is determined by the features of the data points in the training set. We can influence the properties of $\mathbf{Q}$ to some extent by feature transformation methods. One important example of such a transformation is the normalization of features.

able to ensure via feature learning) some upper and lower bounds,

$$0 \leq L \leq \lambda_1(\mathbf{Q}) \leq \ldots \leq \lambda_d(\mathbf{Q}) \leq U. \tag{82}$$

In what follows, we assume that $\mathbf{Q}$ is invertible and that we know some positive lower bound $L > 0$ on its eigenvalues (see (82)). The objective function (78) has then a unique solution $\widehat{\mathbf{w}}$. A gradient step (79) reduces the distance $\left\|\mathbf{w}^{(k)} - \widehat{\mathbf{w}}\right\|_2$ to $\widehat{\mathbf{w}}$ by a constant factor [55, Ch. 6],

$$\left\|\mathbf{w}^{(k+1)} - \widehat{\mathbf{w}}\right\|_2 \leq \kappa^{(\eta_k)}(\mathbf{Q}) \left\|\mathbf{w}^{(k)} - \widehat{\mathbf{w}}\right\|_2. \tag{83}$$

Here, we used the contraction factor

$$\kappa^{(\eta)}(\mathbf{Q}) := \max\left\{|1 - \eta 2\lambda_1|, |1 - \eta 2\lambda_d|\right\}. \tag{84}$$

The contraction factor depends on the learning rate η which is a hyperparameter of gradient-based methods that we can control. However, the contraction factor also depends on the eigenvalues of the matrix $\mathbf{Q}$ in (78). In ML and FL applications, this matrix typically depends on data and can be controlled only to some extent, e.g., using feature transformation [6, Ch. 5]. To ensure $\kappa^{(\eta)}(\mathbf{Q}) < 1$, we require a positive learning rate satisfying $\eta_k < 1/U$.

Consider the gradient step (79) with fixed learning rate η and a contraction factor $\kappa^{(\eta)}(\mathbf{Q}) < 1$ (see (84)). We can then ensure an optimization error $\left\|\mathbf{w}^{(k)} - \widehat{\mathbf{w}}\right\|_2 \leq \varepsilon$ (see (83)) if the number k of gradient steps satisfies

$$k \geq \underbrace{\left\lceil \frac{\log\left(\left\|\mathbf{w}^{(0)} - \widehat{\mathbf{w}}\right\|_2 / \varepsilon\right)}{\log\left(1/\kappa^{(\eta)}(\mathbf{Q})\right)} \right\rceil}_{=:k^{(\varepsilon)}}. \tag{85}$$

According to (83), smaller values of the contraction factor $\kappa^{(\eta)}(\mathbf{Q})$ guarantee a faster convergence of (79) towards the solution of (78). Figure 4.3

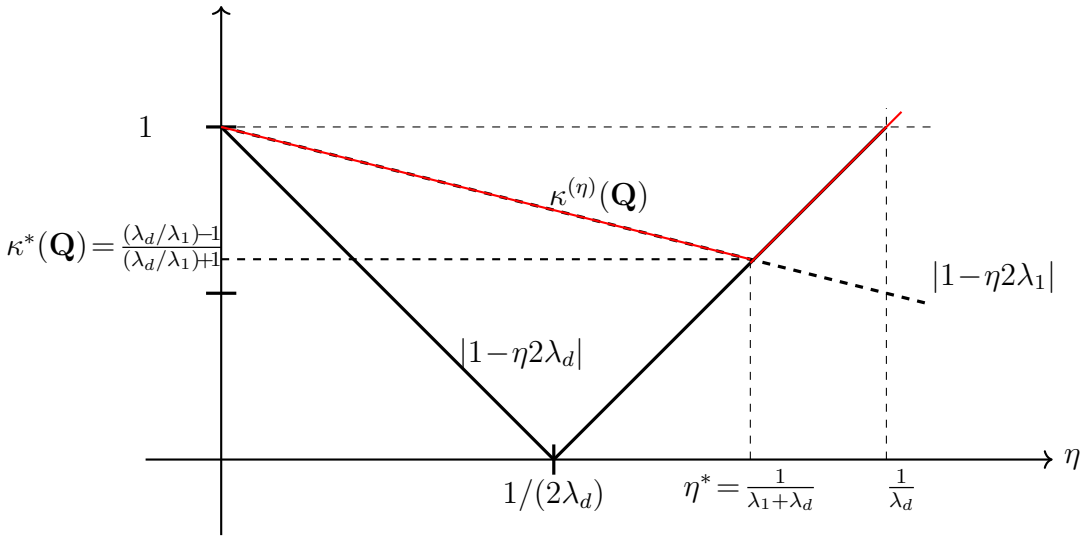


Figure 4.3: The contraction factor $\kappa^{(\eta)}(\mathbf{Q})$ (84), used in the upper bound (83), as a function of the learning rate η. Note that $\kappa^{(\eta)}(\mathbf{Q})$ also depends on the eigenvalues of the matrix $\mathbf{Q}$ in (78).

illustrates the dependence of $\kappa^{(\eta)}(\mathbf{Q})$ on the learning rate η. Thus, choosing a small η (close to 0) will typically result in a larger $\kappa^{(\eta)}(\mathbf{Q})$ and, in turn, require more iterations to ensure optimization error level $\varepsilon^{(\text{tol})}$ via (83).

We can minimize this contraction factor by choosing the learning rate (see Figure 4.3)

$$\eta^{(*)} := \frac{1}{\lambda_1 + \lambda_d}. \tag{86}$$

[Note that evaluating (86) requires to know the extremal eigenvalues λ_1, λ_d

of $\mathbf{Q}$.] Inserting the optimal learning rate (86) into (83),

$$\left\|\mathbf{w}^{(k+1)} - \widehat{\mathbf{w}}\right\|_2 \leq \underbrace{\frac{(\lambda_d/\lambda_1)-1}{(\lambda_d/\lambda_1)+1}}_{:=\kappa^*(\mathbf{Q})} \left\|\mathbf{w}^{(k)} - \widehat{\mathbf{w}}\right\|_2 . \tag{87}$$

Carefully note that the formula (87) is valid only if the matrix $\mathbf{Q}$ in (78) is invertible, i.e., if $\lambda_1 > 0$. If the matrix $\mathbf{Q}$ is singular ($\lambda_1 = 0$), the convergence of (79) towards a solution of (78) is much slower than the decrease of the bound (87). However, we can still ensure the convergence of gradient steps $\mathbf{w}^{(k)}$ by using a fixed learning rate $\eta_k = \eta$ that satisfies [58, Thm. 2.1.14]

$$0 < \eta < 1/\lambda_d(\mathbf{Q}). \tag{88}$$

It is interesting to note that for linear regression, the matrix $\mathbf{Q}$ depends only on the features $\mathbf{x}^{(r)}$ of the data points in the training set (see (17)) but not on their labels $y^{(r)}$. Thus, the convergence of gradient steps is only affected by the features, whereas the labels are irrelevant. The same is true for ridge regression and GTVMin (using local linear models).

Note that both, the optimal learning rate (86) and the optimal contraction factor

$$\kappa^*(\mathbf{Q}) := \frac{(\lambda_d/\lambda_1)-1}{(\lambda_d/\lambda_1)+1} \tag{89}$$

depend on the eigenvalues of the matrix $\mathbf{Q}$ in (78).

According to (87), the ideal case is when all eigenvalues are identical which leads, in turn, to a contraction factor $\kappa^*(\mathbf{Q}) = 0$. Here, a single gradient step arrives at the unique solution of (78).

In general, we do not have full control over the matrix $\mathbf{Q}$ and its eigenvalues. For example, the matrix $\mathbf{Q}$ arising in linear regression (7) is determined by

the features of data points in the training set. These features might be obtained from sensing devices and therefore beyond our control. However, some applications might allow for some design freedom in the choice of feature vectors. We might also use feature transformations that nudge the resulting $\mathbf{Q}$ in (7) more towards a scaled identity matrix.

4.5 Perturbed Gradient Step

Consider the gradient step (79) used to find a minimum of (78). We again assume that the matrix $\mathbf{Q}$ in (78) is invertible ($\lambda_1(\mathbf{Q}) > 0$) and, in turn, (78) has a unique solution $\widehat{\mathbf{w}}$.

In some applications, it is challenging to evaluate the gradient $\nabla f(\mathbf{w}) = 2\mathbf{Q}\mathbf{w} + \mathbf{q}$ of (78) exactly. For example, the evaluation could require to gather data points from distributed storage locations. These storage locations can become unavailable during the computation of $\nabla f(\mathbf{w})$ due to software or hardware failures (e.g., limited connectivity).

We can model imperfections during the computation of (79) as the perturbed gradient step

$$\begin{aligned}\mathbf{w}^{(k+1)} &:= \mathbf{w}^{(k)} - \eta \nabla f\big(\mathbf{w}^{(k)}\big) + \boldsymbol{\varepsilon}^{(k)} \\ &\overset{(78)}{=} \mathbf{w}^{(k)} - \eta\big(2\mathbf{Q}\mathbf{w}^{(k)} + \mathbf{q}\big) + \boldsymbol{\varepsilon}^{(k)}, \text{ for } k = 0, 1, \ldots. \end{aligned} \tag{90}$$

We can use the contraction factor $\kappa := \kappa^{(\eta)}(\mathbf{Q})$ (84) to upper bound the deviation between $\mathbf{w}^{(k)}$ and the optimum $\widehat{\mathbf{w}}$ as (see (83))

$$\big\|\mathbf{w}^{(k)} - \widehat{\mathbf{w}}\big\|_2 \leq \kappa^k \big\|\mathbf{w}^{(0)} - \widehat{\mathbf{w}}\big\|_2 + \sum_{k'=1}^{k} \kappa^{k'} \Big\|\boldsymbol{\varepsilon}^{(k-k')}\Big\|_2 . \tag{91}$$

This bound applies for any number of iterations $k = 1, 2, \ldots$ of the perturbed gradient step (90).

The perturbed gradient step (90) could also be used as a tool to analyze the (exact) gradient step for an objective function $\tilde{f}(\mathbf{w})$ which does not belong to the family (78) of convex quadratic functions. Indeed, we can write the gradient step for minimizing $\tilde{f}(\mathbf{w})$ as

$$\begin{aligned}\mathbf{w}^{(k+1)} &:= \mathbf{w}^{(k)} - \eta \nabla \tilde{f}(\mathbf{w}) \\ &= \mathbf{w}^{(k)} - \eta \nabla f(\mathbf{w}) + \underbrace{\eta\big(\nabla f(\mathbf{w}) - \nabla \tilde{f}(\mathbf{w})\big)}_{:=\boldsymbol{\varepsilon}^{(k)}}.\end{aligned}$$

The last identity is valid for any choice of surrogate function $f(\mathbf{w})$. In particular, we can choose $f(\mathbf{w})$ as a convex quadratic function (78) that approximates $\tilde{f}(\mathbf{w})$. Note that the perturbation term $\boldsymbol{\varepsilon}^{(k)}$ is scaled by the learning rate η.

4.6 Handling Constraints - Projected Gradient Descent

Many important ML and FL methods amount to the minimization of an objective function of the form (78). The optimization variable $\mathbf{w}$ in (78) represents some model parameters.

Sometimes we might require the parameters $\mathbf{w}$ to belong to a subset $\mathcal{S} \subset \mathbb{R}^d$. One example is regularization via model pruning (see Chapter 2). Another example are FL methods that learn identical local model parameters $\mathbf{w}^{(i)}$ at all nodes $i \in \mathcal{V}$ of an FL network. This can be implemented by requiring the stacked local model parameters $\mathbf{w} = \big(\mathbf{w}^{(1)}, \ldots, \mathbf{w}^{(n)}\big)^T$ to belong

to the subset

$$\mathcal{S} = \Big\{ \big(\mathbf{w}^{(1)}, \ldots, \mathbf{w}^{(n)}\big)^T : \mathbf{w}^{(1)} = \ldots = \mathbf{w}^{(n)} \Big\}.$$

Let us now show how to adapt the gradient step (79) to solve the constrained problem

$$f^* = \min_{\mathbf{w} \in \mathcal{S}} \mathbf{w}^T \mathbf{Q} \mathbf{w} + \mathbf{q}^T \mathbf{w}. \tag{92}$$

We assume that the constraint set $\mathcal{S} \subseteq \mathbb{R}^d$ is such that we can efficiently compute the projection

$$P_{\mathcal{S}}\big(\mathbf{w}\big) = \operatorname*{argmin}_{\mathbf{w}' \in \mathcal{S}} \|\mathbf{w} - \mathbf{w}'\|_2 \text{ for any } \mathbf{w} \in \mathbb{R}^d. \tag{93}$$

A suitable modification of the gradient step (79) to solve the constrained variant (92) is [55]

$$\begin{aligned} \mathbf{w}^{(k+1)} &:= P_{\mathcal{S}}\big(\mathbf{w}^{(k)} - \eta \nabla f\big(\mathbf{w}^{(k)}\big)\big) \\ &\overset{(78)}{=} P_{\mathcal{S}}\big(\mathbf{w}^{(k)} - \eta\big(2\mathbf{Q}\mathbf{w}^{(k)} + \mathbf{q}\big)\big). \end{aligned} \tag{94}$$

The projected GD step (94) amounts to

1. compute ordinary gradient step $\mathbf{w}^{(k)} \mapsto \mathbf{w}^{(k)} - \eta \nabla f\big(\mathbf{w}^{(k)}\big)$ and
2. project back to the constraint set $\mathcal{S}$.

Note that we re-obtain the basic gradient step (79) from the projected gradient step (94) for the trivial constraint set $\mathcal{S} = \mathbb{R}^d$.

The approaches for choosing the learning rate η and stopping criterion for basic gradient step (79) explained in Sections 4.3 and 4.4 work also for the projected gradient step (94). In particular, the convergence speed of the projected gradient step is also characterized by (83) [55, Ch. 6]. This follows

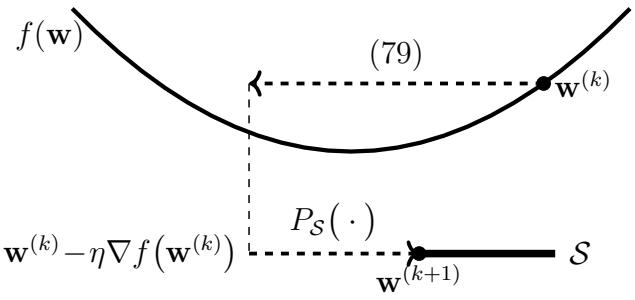


Figure 4.4: Projected GD augments a basic gradient step with a projection back onto the constraint set $\mathcal{S}$.

from the fact that the concatenation of a contraction (such as the gradient step (79) for sufficiently small η) and a projection (such as $P_{\mathcal{S}}(\,\cdot\,)$) results again in a contraction with the same contraction factor.

Thus, the convergence speed of projected GD, in terms of number of iterations required to ensure a given level of optimization error, is essentially the same as that of basic GD. However, the bound (83) is only telling about the number of projected gradient steps (94) required to achieve a guaranteed level of sub-optimality $\big|f\big(\mathbf{w}^{(k)}\big) - f^{*}\big|$. The iteration (94) of projected GD might require significantly more computation than the basic gradient step, as it requires to compute the projection (93).

4.7 Generalizing the Gradient Step

The gradient-based methods discussed so far can be used to learn a hypothesis from a parametric model. Let us now sketch one possible generalization of the gradient step (79) for a model $\mathcal{H}$ without a parametrization.

We start with rewriting the gradient step (79) as the optimization

$$\mathbf{w}^{(k+1)} = \operatorname*{argmin}_{\mathbf{w}\in\mathbb{R}^d} \left[(1/(2\eta)) \left\|\mathbf{w}-\mathbf{w}^{(k)}\right\|_2^2 + \underbrace{f\left(\mathbf{w}^{(k)}\right) + \left(\mathbf{w}-\mathbf{w}^{(k)}\right)^T \nabla f\left(\mathbf{w}^{(k)}\right)}_{\approx f(\mathbf{w})} \right]. \tag{95}$$

The objective function in (95) includes the first-order approximation

$$f(\mathbf{w}) \approx f\left(\mathbf{w}^{(k)}\right) + \left(\mathbf{w}-\mathbf{w}^{(k)}\right)^T \nabla f\left(\mathbf{w}^{(k)}\right)$$

of the function $f(\mathbf{w})$ around the location $\mathbf{w} = \mathbf{w}^{(k)}$ (see Figure 4.1).

Let us modify (95) by using $f(\mathbf{w})$ itself (instead of an approximation),

$$\mathbf{w}^{(k+1)} = \operatorname*{argmin}_{\mathbf{w}\in\mathbb{R}^d} \left[f(\mathbf{w}) + (1/(2\eta)) \left\|\mathbf{w}-\mathbf{w}^{(k)}\right\|_2^2 \right]. \tag{96}$$

Like the gradient step, also (96) maps a given vector $\mathbf{w}^{(k)}$ to an updated vector $\mathbf{w}^{(k+1)}$. Note that (96) is nothing but the proximal operator of the function $f(\mathbf{w})$ [42]. Similar to the role of the gradient step as the main building block of gradient-based methods, the proximal operator (96) is the main building block of proximal algorithms [42].

To obtain a version of (96) for a non-parametric model, we need to be able to evaluate its objective function directly in terms of a hypothesis h instead of its parameters $\mathbf{w}$. The objective function (96) consists of two components. The second component $f(\cdot)$, which is the function we want to minimize, is obtained from a training error incurred by a hypothesis, which might be parametric $h^{(\mathbf{w})}$. Thus, we can evaluate the function $f(h)$ by computing the training error for a given hypothesis.

The first component of the objective function in (96) uses $\left\|\mathbf{w}-\mathbf{w}^{(k)}\right\|_2^2$ to measure the difference between the hypothesis maps $h^{(\mathbf{w})}$ and $h^{(\mathbf{w}^{(k)})}$. Another

measure for the difference between two hypothesis maps can be obtained by using some test dataset $\mathcal{D}' = \{\mathbf{x}^{(1)}, \ldots, \mathbf{x}^{(m')}\}$: The average squared difference between their predictions,

$$(1/m') \sum_{r=1}^{m'} \Big(h\big(\mathbf{x}^{(r)}\big) - h^{(k)}\big(\mathbf{x}^{(r)}\big) \Big)^2, \tag{97}$$

is a measure for the difference between h and $h^{(k)}$. Note that (97) only requires the predictions delivered by the hypothesis maps $h, h^{(k)}$ on $\mathcal{D}'$ - no other information is needed about these maps.

It is interesting to note that (97) coincides with $\big\|\mathbf{w} - \mathbf{w}^{(k)}\big\|_2^2$ for the linear model $h^{(\mathbf{w})}(\mathbf{x}) := \mathbf{w}^T\mathbf{x}$ and a specific construction of the dataset $\mathcal{D}'$. This construction uses the realizations $\mathbf{x}^{(1)}, \mathbf{x}^{(2)}, \ldots$ of i.i.d. RVs with a common probability distribution $\mathbf{x} \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$. Indeed, by the law of large numbers

$$\begin{aligned}
&\lim_{m' \to \infty} (1/m') \sum_{r=1}^{m'} \Big(h^{(\mathbf{w})}\big(\mathbf{x}^{(r)}\big) - h^{(\mathbf{w}^{(k)})}\big(\mathbf{x}^{(r)}\big) \Big)^2 \\
&= \lim_{m' \to \infty} (1/m') \sum_{r=1}^{m'} \Big(\big(\mathbf{w} - \mathbf{w}^{(k)}\big)^T \mathbf{x}^{(r)} \Big)^2 \\
&= \lim_{m' \to \infty} (1/m') \sum_{r=1}^{m'} \big(\mathbf{w} - \mathbf{w}^{(k)}\big)^T \mathbf{x}^{(r)} \big(\mathbf{x}^{(r)}\big)^T \big(\mathbf{w} - \mathbf{w}^{(k)}\big) \\
&= \big(\mathbf{w} - \mathbf{w}^{(k)}\big)^T \underbrace{\Big[\lim_{m' \to \infty} (1/m') \sum_{r=1}^{m'} \mathbf{x}^{(r)} \big(\mathbf{x}^{(r)}\big)^T \Big]}_{=\mathbf{I}} \big(\mathbf{w} - \mathbf{w}^{(k)}\big) \\
&= \big\|\mathbf{w} - \mathbf{w}^{(k)}\big\|_2^2 .
\end{aligned} \tag{98}$$

Finally, we arrive at a generalized gradient step for the training of a non-parametric model $\mathcal{H}$ by replacing $\big\|\mathbf{w} - \mathbf{w}^{(k)}\big\|_2^2$ in (96) with (97). In

other words,

$$h^{(k+1)} = \operatorname*{argmin}_{h \in \mathcal{H}} \left[(1/(2\eta m')) \sum_{r=1}^{m'} \left(h\big(\mathbf{x}^{(r)}\big) - h^{(k)}\big(\mathbf{x}^{(r)}\big) \right)^2 + f(h) \right]. \tag{99}$$

We can turn gradient-based methods for the training of parametric models into corresponding training methods for non-parametric models by replacing the gradient step with the update (99). For example, we obtain Algorithm 3 from Algorithm 2 by modifying step 3 suitably.

Algorithm 3 A blueprint for generalized gradient-based methods

Input: some objective function $f : \mathcal{H} \to \mathbb{R}$ (e.g., the average loss of a hypothesis $h \in \mathcal{H}$ on a training set); learning rate $\eta > 0$; some stopping criterion; test dataset $\mathcal{D}' = \{\mathbf{x}^{(1)}, \ldots, \mathbf{x}^{(m')}\}$

Initialize: set $h^{(0)} := \mathbf{0}$; set iteration counter $k := 0$

1: **repeat**

2: $k := k + 1$ (increase iteration counter)

3: do a generalized gradient step (99),

$$h^{(k)} = \operatorname*{argmin}_{h \in \mathcal{H}} \left[(1/(2\eta m')) \sum_{r=1}^{m'} \left(h\big(\mathbf{x}^{(r)}\big) - h^{(k-1)}\big(\mathbf{x}^{(r)}\big) \right)^2 + f(h) \right]$$

4: **until** stopping criterion is met

Output: learnt hypothesis $\widehat{h} := h^{(k)}$ (hopefully $f\big(\widehat{h}\big) \approx \min_{h \in \mathcal{H}} f(h)$)

4.8 Gradient Methods as Fixed-Point Iterations

The iterative optimization methods discussed in the previous sections are all special cases of a fixed-point iteration,

$$\mathbf{w}^{(k)} = \mathcal{F}\mathbf{w}^{(k-1)}, \text{ for } k = 1, 2, \ldots. \tag{100}$$

In what follows, we tacitly assume that the operator $\mathcal{F} : \mathbb{R}^m \to \mathbb{R}^m$ is defined on a Euclidean space $\mathbb{R}^m$ with some fixed dimension m. We will use fixed-point iterations of the form (100) to solve GTVMin (see Ch. 3). For parametrized local models, we can learn local model parameters by the iterations (100) using the stacking $\mathbf{w}^{(k)}$ according to $\mathbf{w}^{(k)} = \big(\mathbf{w}^{(1,k)}, \ldots, \mathbf{w}^{(n,k)}\big)^T$. For a suitable choice of $\mathcal{F}$, the sequence model parameters $\mathbf{w}^{(1,i)}, \mathbf{w}^{(1,i)}, \ldots$ converges towards the optimal local model parameters (solving GTVMin) at each node $i = 1, \ldots, n$.

The FL algorithms in Ch. 5 are implementations of (100) with an operator $\mathcal{F}$ having a GTVMin-solution $\widehat{\mathbf{w}} \in \mathbb{R}^{d \cdot n}$ as a fixed point,

$$\mathcal{F}\widehat{\mathbf{w}} = \widehat{\mathbf{w}}. \tag{101}$$

Given an instance of GTVMin, there are (infinitely) many different operators $\mathcal{F}$ that satisfy (101). We obtain different FL algorithms by using different choices for $\mathcal{F}$ in (100). A useful choice of $\mathcal{F}$ should reduce the distance to a solution,

$$\underbrace{\big\|\mathbf{w}^{(k+1)} - \widehat{\mathbf{w}}\big\|_2}_{\overset{(100),(101)}{=}\big\|\mathcal{F}\mathbf{w}^{(k)} - \mathcal{F}\widehat{\mathbf{w}}\big\|_2} \leq \big\|\mathbf{w}^{(k)} - \widehat{\mathbf{w}}\big\|_2 . \tag{102}$$

Thus, we require $\mathcal{F}$ to be at least non-expansive, i.e., the iteration (100) should not result in worse model parameters (increasing the distance to the

GTVMin solution). Moreover, each iteration (100) should also make some progress, i.e., reduce the distance from a GTVMin solution. This requirement can be made precise using the notion of a contractive operator [59, 60].

The operator $\mathcal{F}$ is contractive (or a contraction mapping) if, for some $\kappa \in [0,1)$,

$$\|\mathcal{F}\mathbf{w} - \mathcal{F}\mathbf{w}'\|_2 \leq \kappa \|\mathbf{w} - \mathbf{w}'\|_2 \text{ holds for any } \mathbf{w}, \mathbf{w}' \in \mathbb{R}^{dn}. \tag{103}$$

For a contractive $\mathcal{F}$, the fixed-point iteration (100) generates a sequence $\mathbf{w}^{(k)}$ that converges to a GTVMin solution $\widehat{\mathbf{w}}$ quite rapidly. In particular [2, Theorem 9.23],

$$\left\|\mathbf{w}^{(k)} - \widehat{\mathbf{w}}\right\|_2 \leq \kappa^k \left\|\mathbf{w}^{(0)} - \widehat{\mathbf{w}}\right\|_2. \tag{104}$$

Here, $\left\|\mathbf{w}^{(0)} - \widehat{\mathbf{w}}\right\|_2$ is the distance between the initialization $\mathbf{w}^{(0)}$ and the solution $\widehat{\mathbf{w}}$.

A well-known example of a fixed-point iteration (100) using a contractive operator is GD (79) for a smooth and strongly convex objective function $f(\mathbf{w})$.[14] In particular, (79) is obtained from (100) using $\mathcal{F} := \mathcal{G}^{(\eta)}$ with the (gradient step) operator

$$\mathcal{G}^{(\eta)} : \mathbf{w} \mapsto w - \eta \nabla f(\mathbf{w}). \tag{105}$$

Note that the operator (105) is parametrized by the learning rate η.

It is instructive to study the operator $\mathcal{G}^{(\eta)}$ for an objective function of the form (78). Here,

$$\mathcal{G}^{(\eta)} : \mathbf{w} \mapsto \mathbf{w} - \eta \underbrace{\left(2\mathbf{Q}\mathbf{w} + \mathbf{q}\right)}_{\overset{(78)}{=}\nabla f(\mathbf{w})}. \tag{106}$$

[14] The objective function in (78) is convex and smooth for any choice of psd matrix $\mathbf{Q}$ and vector $\mathbf{q}$. Moreover, it is strongly convex whenever $\mathbf{Q}$ is invertible.

For $\eta := 1/(2\lambda_{\max}(\mathbf{Q}))$, the operator $\mathcal{G}^{(\eta)}$ is contractive with $\kappa = 1 - \lambda_{\min}(\mathbf{Q})/\lambda_{\max}(\mathbf{Q})$. Note that $\kappa < 1$ only when $\lambda_{\min}(\mathbf{Q}) > 0$, i.e., only when the matrix $\mathbf{Q}$ in (78) is invertible.

The gradient step operator (106) is not contractive for the objective function (78) with a singular matrix $\mathbf{Q}$ (for which $\lambda_{\min} = 0$). However, even then $\mathcal{G}^{(\eta)}$ is still firmly non-expansive [28]. We refer to an operator $\mathcal{F} : \mathbb{R}^{d \cdot n} \to \mathbb{R}^{d \cdot n}$ as firmly non-expansive if

$$\|\mathcal{F}\mathbf{w} - \mathcal{F}\mathbf{w}'\|_2^2 \leq \big(\mathcal{F}\mathbf{w} - \mathcal{F}\mathbf{w}'\big)^T \big(\mathbf{w} - \mathbf{w}'\big), \text{ for any } \mathbf{w}, \mathbf{w}' \in \mathbb{R}^{d \cdot n}. \quad (107)$$

It turns out that a fixed-point iteration (100) with a firmly non-expansive operator $\mathcal{F}$ is guaranteed to converge to a fixed-point of $\mathcal{F}$ [59, Cor. 5.16]. Fig. 4.5 depicts examples of a firmly non-expansive, a non-expansive and a contractive operator defined on the one-dimensional space $\mathbb{R}$. Another example of a firmly non-expansive operator is the proximal operator (96) of a convex function [42, 59].

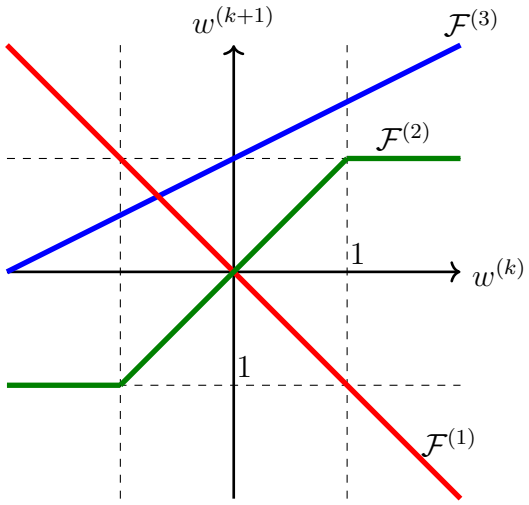


Figure 4.5: Example of a non-expansive operator $\mathcal{F}^{(1)}$, a firmly non-expansive operator $\mathcal{F}^{(2)}$ and a contractive operator $\mathcal{F}^{(3)}$.

4.9 Exercises

4.1. Learning Rate Schedule. Consider the gradient step method applied to a differentiable objective function $f(\mathbf{w})$,

$$\mathbf{w}^{(k+1)} = \mathbf{w}^{(k)} - \eta_k \nabla f\big(\mathbf{w}^{(k)}\big), \quad \text{for } k = 1, 2, \ldots.$$

where the learning rate schedule is defined as $\eta_k := \frac{1}{k}$.

1. Verify that this learning rate schedule satisfies the standard conditions in (80).

2. Construct a differentiable, convex function $f(\mathbf{w})$ and an initialization $\mathbf{w}^{(0)}$ such that the gradient step iteration fails to converge to a minimizer of $f(\mathbf{w})$.

4.2. Online Gradient Descent. Linear regression methods learn model parameters of a linear model with minimum risk $\mathbb{E}\big\{\big(y-\mathbf{w}^T\mathbf{x}\big)^2\big\}$ where $(\mathbf{x},y)$ is a RV. In practice, we do not observe the RV $(\mathbf{x},y)$ itself but a (realization of a) sequence of i.i.d. samples $\big(\mathbf{x}^{(t)},y^{(t)}\big)$, for $t=1,2,\ldots$. Online GD is an online learning method that updates the current model parameters $\mathbf{w}^{(t)}$, after observing $\big(\mathbf{x}^{(t)},y^{(t)}\big)$,

$$\mathbf{w}^{(t+1)} := \mathbf{w}^{(t)} + 2\eta_t\mathbf{x}^{(t)}\big(y-\big(\mathbf{w}^{(t)}\big)^T\mathbf{x}^{(t)}\big) \text{ at time } t=1,2,\ldots. \tag{108}$$

Starting with initialization $\mathbf{w}^{(1)}:=\mathbf{0}$, we run online GD for M time steps, resulting in the learnt model parameters $\mathbf{w}^{(M+1)}$. Develop upper bounds on the risk $\mathbb{E}\big\{\big(y-\big(\mathbf{w}^{(M)}\big)^T\mathbf{x}\big)^2\big\}$ for two choices for the learning rate schedule: $\eta_t:=1/(t+5)$ or $\eta_t:=1/\sqrt{t+5}$.

4.3. Computing the Average - I. Consider an FL network with graph $\mathcal{G}$ and its Laplacian matrix $\mathbf{L}^{(\mathcal{G})}$. Each node carries a local dataset which consists of a single measurement $y^{(i)}\in\mathbb{R}$. To compute their average $(1/n)\sum_{i=1}^n y^{(i)}$ we try an iterative method that, starting from the initialization $\mathbf{u}^{(0)}:=\big(y^{(1)},\ldots,y^{(n)}\big)^T\in\mathbb{R}^n$, repeats the update

$$\mathbf{u}^{(k+1)} = \mathbf{u}^{(k)} - \eta\mathbf{L}^{(\mathcal{G})}\mathbf{u}^{(k)} \text{ for } k=1,2,\ldots. \tag{109}$$

Can you find a choice for η such that (109) becomes a fixed-point iteration (100) with a contractive operator $\mathcal{F}$. Given such a choice of η, how is the limit $\lim_{k\to\infty}\mathbf{u}^{(k+1)}$ related to the average $(1/n)\sum_{i=1}^n y^{(i)}$?

4.4. Computing the Average - II. Consider the FL network from Problem 4.3. Try to construct an instance of GTVMin for learning scalar local model parameters $w^{(i)}$ which coincide, for each node $i=1,\ldots,n$ with the average

$(1/n)\sum_{i'=1}^{n} y^{(i')}$. If you find such an instance of GTVMin, solve it using GD (see Section 4.2).

4.5. How to Quantize the Gradients? Any ML and FL application that uses a digital computer to implement a gradient step (79) must quantize the gradient $\nabla f(\mathbf{w})$ of the objective function $f(\mathbf{w})$. The quantization process introduces perturbations to the gradient step. Given a fixed total budget of bits available for quantization, a key question arises: Should we allocate more bits (reducing quantization noise) during the initial gradient steps or during the final gradient steps in gradient-based methods?

Hint: See Section 4.5.

4.6. When is a Gradient Step (Firmly) Non-Expansive? Consider the function $f(w) = (1/2)w^2$ and the associated gradient step $\mathcal{G}^{(\eta)} : w \mapsto w - \eta\nabla f(w)$. Discuss the value ranges for the learning rate η, for which the operator $\mathcal{G}^{(\eta)}$ is non-expansive or even firmly non-expansive.

5 FL Algorithms

Chapter 3 introduced GTVMin as a flexible design principle for FL methods that arise from different design choices for the local models and edge weights of the FL network. The solutions of GTVMin are local model parameters that strike a balance between the loss incurred on local datasets and the GTV. This chapter applies the gradient-based methods from Chapter 4 to solve GTVMin. We obtain FL algorithms by implementing these optimization methods as *m*essage passing across the edges of the FL network. These messages contain intermediate results of the computations carried out by FL algorithms. The details of how this message passing is implemented physically (e.g., via short-range wireless technology) are beyond the scope of this book.

Section 5.2 studies the gradient step for the GTVMin instance obtained for training local linear models. In particular, we show how the convergence rate of the gradient step can be characterized by the properties of the local datasets and their FL network.

Section (5.3) spells out the gradient step from Section 5.2 in the form of a message passing across the edges of the FL network. This results in Algorithm 4 as a distributed FL method for parametric local models. Section 5.4 generalizes Algorithm 4 by replacing the exact gradient of local loss functions with some approximation. One possible approximation is to use a random subset (a batch) of a local dataset to estimate the gradient.

Section 5.5 discusses FL algorithms that train a single (global) model in a distributed fashion. We show how the widely-used FL algorithms FedAvg and FedProx are obtained from variations of projected GD which we discussed in Section 4.6.

Section 5.7 generalizes the gradient step, which is the core computation of FL algorithms for parametric models, to cope with non-parametric models. The idea is to compare the predictions of hypothesis maps at nodes i, i' on a common test-set to measure their variation across the edge $\{i, i'\}$.

Most of the algorithms discussed in this chapter operate in a synchronous manner: All devices must complete their local model updates (e.g., gradient steps) before exchanging updates simultaneously across the edges of the FL network. However, synchronous operation can be impractical or even infeasible for certain FL applications. Section 5.9 explores the design of FL algorithms that support asynchronous operation. In particular, the resulting algorithms allow devices to update and communicate at different times within the FL system

5.1 Learning Goals

After completing this chapter, you

- can apply gradient-based methods to GTVMin for local linear models,
- can implement a gradient step via message passing over FL networks,
- can generalize gradient-based methods by using gradient approximation,
- know how FedAvg is obtained from projected GD.
- know how FedProx is obtained from FedAvg.
- can generalize gradient-based methods to handle non-parametric models.
- can formulate asynchronous FL algorithms.

5.2 Gradient Descent for GTVMin

Consider a collection of n local datasets represented by the nodes $\mathcal{V} = \{1,\ldots,n\}$ of an FL network $\mathcal{G} = (\mathcal{V},\mathcal{E})$. Each undirected edge $\{i,i'\} \in \mathcal{E}$ in FL network $\mathcal{G}$ has a known edge weight $A_{i,i'}$. We want to learn local model parameters $\mathbf{w}^{(i)}$ of a personalized linear model for each node $i = 1,\ldots,n$. To this end, we solve the GTVMin instance

$$\big\{\widehat{\mathbf{w}}^{(i)}\big\}_{i=1}^{n} \in \operatorname*{argmin}_{\{\mathbf{w}^{(i)}\}} \underbrace{\sum_{i\in\mathcal{V}} \overbrace{(1/m_i)\left\|\mathbf{y}^{(i)}-\mathbf{X}^{(i)}\mathbf{w}^{(i)}\right\|_2^2}^{\text{local loss } L_i\left(\mathbf{w}^{(i)}\right)} + \alpha \sum_{\{i,i'\}\in\mathcal{E}} A_{i,i'}\left\|\mathbf{w}^{(i)}-\mathbf{w}^{(i')}\right\|_2^2}_{=:f(\mathbf{w})}. \tag{110}$$

As discussed in Chapter 3, the objective function in (110) - viewed as a function of the stacked local model parameters $\mathbf{w} := \text{stack}\{\mathbf{w}^{(i)}\}_{i=1}^{n}$ - is a quadratic function

$$\mathbf{w}^T\left(\begin{pmatrix}\mathbf{Q}^{(1)} & \cdots & \mathbf{0}\\ \vdots & \ddots & \vdots\\ \mathbf{0} & \cdots & \mathbf{Q}^{(n)}\end{pmatrix} + \alpha\mathbf{L}^{(\mathcal{G})}\otimes\mathbf{I}\right)\mathbf{w} + \left(\left(\mathbf{q}^{(1)}\right)^T,\ldots,\left(\mathbf{q}^{(n)}\right)^T\right)\mathbf{w} \tag{111}$$

with $\mathbf{Q}^{(i)} = (1/m_i)\left(\mathbf{X}^{(i)}\right)^T\mathbf{X}^{(i)}$ and $\mathbf{q}^{(i)} := (-2/m_i)\left(\mathbf{X}^{(i)}\right)^T\mathbf{y}^{(i)}$.

Note that (111) is a special case of the generic quadratic function (78) studied in Chapter 4. Indeed, we obtain (111) from (78) for the choices

$$\mathbf{Q} := \left(\begin{pmatrix}\mathbf{Q}^{(1)} & \cdots & \mathbf{0}\\ \vdots & \ddots & \vdots\\ \mathbf{0} & \cdots & \mathbf{Q}^{(n)}\end{pmatrix} + \alpha\mathbf{L}^{(\mathcal{G})}\otimes\mathbf{I}\right), \text{ and } \mathbf{q} := \left(\left(\mathbf{q}^{(1)}\right)^T,\ldots,\left(\mathbf{q}^{(n)}\right)^T\right)^T. \tag{112}$$

Therefore, the discussion and analysis of gradient-based methods from Chapter 4 also apply to GTVMin (110). In particular, we can use the gradient step

$$\begin{aligned}\mathbf{w}^{(k+1)} &:= \mathbf{w}^{(k)} - \eta \nabla f\big(\mathbf{w}^{(k)}\big) \\ &\overset{(111)}{=} \mathbf{w}^{(k)} - \eta\big(2\mathbf{Q}\mathbf{w}^{(k)} + \mathbf{q}\big)\end{aligned} \tag{113}$$

to iteratively compute an approximate solution $\widehat{\mathbf{w}}$ to (110). This solution consists of learnt local model parameters $\widehat{\mathbf{w}}^{(i)}$, i.e., $\widehat{\mathbf{w}} = \text{stack}\{\widehat{\mathbf{w}}^{(i)}\}$. Section 5.3 formulates the gradient step (113) directly in terms of local model parameters, resulting in a message passing over the FL network $\mathcal{G}$.

According to the convergence analysis in Chapter 4, the convergence rate of the iterations (113) is determined by the eigenvalues $\lambda_j(\mathbf{Q})$ of the matrix $\mathbf{Q}$ in (111). Clearly, these eigenvalues are related to the eigenvalues $\lambda_j\big(\mathbf{Q}^{(i)}\big)$ and to the eigenvalues $\lambda_j\big(\mathbf{L}^{(\mathcal{G})}\big)$ of the Laplacian matrix of the FL network $\mathcal{G}$. In particular, we will use the following two summary parameters

$$\lambda_{\max} := \max_{i=1,\ldots,n} \lambda_d\big(\mathbf{Q}^{(i)}\big), \text{ and } \bar{\lambda}_{\min} := \lambda_1\Big((1/n)\sum_{i=1}^{n} \mathbf{Q}^{(i)}\Big). \tag{114}$$

We first present an upper bound U (see (82)) on the eigenvalues of the matrix $\mathbf{Q}$ in (111).

Proposition 5.1. *The eigenvalues of* $\mathbf{Q}$ *in* (111) *are upper-bounded as*

$$\begin{aligned}\lambda_j(\mathbf{Q}) &\leq \lambda_{\max} + \alpha\lambda_n\big(\mathbf{L}^{(\mathcal{G})}\big) \\ &\leq \underbrace{\lambda_{\max} + 2\alpha d^{(\mathcal{G})}_{\max}}_{=:U} \text{ , for } j = 1, \ldots, dn.\end{aligned} \tag{115}$$

Proof. See Section 5.11.1. □

The next result offers a lower bound on the eigenvalues $\lambda_j(\mathbf{Q})$.

Proposition 5.2. *Consider the matrix* $\mathbf{Q}$ *in* (111). *If* $\lambda_2\big(\mathbf{L}^{(\mathcal{G})}\big) > 0$ *(i.e., the FL network in* (110) *is connected) and* $\bar{\lambda}_{\min} > 0$ *(i.e., the average of the matrices* $\mathbf{Q}^{(i)}$ *is non-singular), then the matrix* $\mathbf{Q}$ *is invertible and its smallest eigenvalue is lower bounded as*

$$\lambda_1(\mathbf{Q}) \geq \frac{1}{1+\rho^2}\min\{\lambda_2\big(\mathbf{L}^{(\mathcal{G})}\big)\alpha\rho^2, \bar{\lambda}_{\min}/2\}. \tag{116}$$

Here, we used the shorthand $\rho := \bar{\lambda}_{\min}/(4\lambda_{\max})$ *(see* (114)*).*

Proof. See Section 5.11.2. □

Prop. 5.1 and Prop. 5.2 provide some guidance for the design choices of GTVMin. According to the convergence analysis of gradient-based methods in Ch. 4, the eigenvalue $\lambda_1\big(\mathbf{Q}\big)$ should be close to $\lambda_{dn}\big(\mathbf{Q}\big)$ to ensure fast convergence. This suggests to favour FL networks $\mathcal{G}$ resulting in a small ratio between the upper bound (115) and the lower bound (116). A small ratio between these bounds, in turn, requires a large eigenvalue $\lambda_2\big(\mathbf{L}^{(\mathcal{G})}\big)$ and small node degree $d^{(\mathcal{G})}_{\max}$.[15]

The bounds in (115) and (116) also depend on the GTVMin parameter α. While these bounds might provide some guidance for the choice of α, the exact dependence of the convergence speed of (113) on α is complicated. For a fixed value of learning rate in (113), using larger values for α might slow down the convergence of (113) for some collection of local datasets but speed up the convergence of (113) for another collection of local datasets (see Exercise 5.1).

[15]The are constructions of graphs with a prescribed value of $d^{(\mathcal{G})}_{\max}$ such that $\lambda_2\big(\mathbf{L}^{(\mathcal{G})}\big)$ is maximal [61, 62].

5.3 Message Passing Implementation

We now discuss in more detail the implementation of gradient-based methods to solve the GTVMin instances with a differentiable objective function $f(\mathbf{w})$. One such instance is GTVMin for local linear models (see (110)). The core of gradient-based methods is the gradient step

$$\mathbf{w}^{(k+1)} := \mathbf{w}^{(k)} - \eta \nabla f\big(\mathbf{w}^{(k)}\big). \tag{117}$$

The iterate $\mathbf{w}^{(k)}$ contains local model parameters $\mathbf{w}^{(i,k)}$,

$$\mathbf{w}^{(k)} =: \mathrm{stack}\big\{\mathbf{w}^{(i,k)}\big\}_{i=1}^{n}. \tag{118}$$

Inserting (110) into (117), we obtain the gradient step

$$\mathbf{w}^{(i,k+1)} := \mathbf{w}^{(i,k)} - \eta \Bigg[\underbrace{(2/m_i)\big(\mathbf{X}^{(i)}\big)^T\big(\mathbf{X}^{(i)}\mathbf{w}^{(i,k)} - \mathbf{y}^{(i)}\big)}_{\text{(I)}} + \underbrace{2\alpha \sum_{i' \in \mathcal{N}^{(i)}} A_{i,i'}\big(\mathbf{w}^{(i,k)} - \mathbf{w}^{(i',k)}\big)}_{\text{(II)}} \Bigg]. \tag{119}$$

We slightly modify this gradient step by using potentially different learning rates $\eta_{k,i}$ at different nodes i and iterations k,

$$\mathbf{w}^{(i,k+1)} := \mathbf{w}^{(i,k)} - \eta_{k,i} \Bigg[\underbrace{(2/m_i)\big(\mathbf{X}^{(i)}\big)^T\big(\mathbf{X}^{(i)}\mathbf{w}^{(i,k)} - \mathbf{y}^{(i)}\big)}_{\text{(I)}} + \underbrace{2\alpha \sum_{i' \in \mathcal{N}^{(i)}} A_{i,i'}\big(\mathbf{w}^{(i,k)} - \mathbf{w}^{(i',k)}\big)}_{\text{(II)}} \Bigg]. \tag{120}$$

The update (120) consists of two components, denoted (I) and (II). The component (I) is the gradient $\nabla L_i\big(\mathbf{w}^{(i,k)}\big)$ of the local loss $L_i\big(\mathbf{w}^{(i)}\big) :=$

$(1/m_i)\left\|\mathbf{y}^{(i)} - \mathbf{X}^{(i)}\mathbf{w}^{(i)}\right\|_2^2$. Component (I) drives the updated local model parameters $\mathbf{w}^{(i,k+1)}$ towards the minimum of $L_i(\cdot)$, i.e., having a small deviation between labels $y^{(i,r)}$ and the predictions $\left(\mathbf{w}^{(i,k+1)}\right)^T\mathbf{x}^{(i,r)}$. Note that we can rewrite the component (I) in (120), as

$$(2/m_i)\sum_{r=1}^{m_i}\mathbf{x}^{(i,r)}\big(y^{(i,r)} - \left(\mathbf{x}^{(i,r)}\right)^T\mathbf{w}^{(i,k)}\big). \tag{121}$$

The purpose of component (II) in (120) is to force the local model parameters to be similar across an edge $\{i, i'\}$ with large weight $A_{i,i'}$. We control the relative importance of (II) and (I) using the GTVMin parameter α: Choosing a large value for α puts more emphasis on enforcing similar local model parameters across the edges. Using a smaller α puts more emphasis on learning local model parameters delivering accurate predictions (incurring a small loss) on the local dataset.

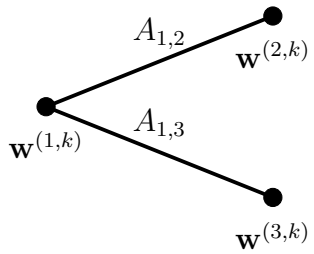


Figure 5.1: At the beginning of iteration k, node $i = 1$ collects the current local model parameters $\mathbf{w}^{(2,k)}$ and $\mathbf{w}^{(3,k)}$ from its neighbours. Then, it computes the gradient step (120) to obtain the new local model parameters $\mathbf{w}^{(1,k+1)}$. These updated parameters are then used in the next iteration for the local updates at the neighbours $i = 2, 3$.

The execution of the gradient step (120) requires only local information at node i. Indeed, the update (120) at node i depends only on its current

model parameters $\mathbf{w}^{(i,k)}$, the local loss function $L_i(\cdot)$, the neighbors' model parameters $\mathbf{w}^{(i',k)}$, for $i' \in \mathcal{N}^{(i)}$, and the corresponding edge weights $A_{i,i'}$ (see Figure 5.1). In particular, the update (120) does not depend on any properties kor edge weights) of the FL network beyond the neighbors $\mathcal{N}^{(i)}$.

We obtain Algorithm 4 by repeating the gradient step (120), simultaneously for each node $i \in \mathcal{V}$, until a stopping criterion is met. Algorithm 4 allows for potentially different learning rates $\eta_{k,i}$ at different nodes i and iterations k (see Section 4.3). It is important to note that Algorithm 4 requires a synchronous

Algorithm 4 FedGD for Local Linear Models

Input: FL network $\mathcal{G}$; GTV parameter α; learning rate $\eta_{k,i}$; local dataset $\mathcal{D}^{(i)} = \left\{\left(\mathbf{x}^{(i,1)}, y^{(i,1)}\right); \ldots, \left(\mathbf{x}^{(i,m_i)}, y^{(i,m_i)}\right)\right\}$ for each i; some stopping criterion.

Output: linear model parameters $\widehat{\mathbf{w}}^{(i)}$ for each node $i \in \mathcal{V}$

Initialize: $k := 0$; $\mathbf{w}^{(i,0)} := \mathbf{0}$

while stopping criterion is not satisfied **do**
  for all nodes $i \in \mathcal{V}$ (simultaneously) **do**
    share local model parameters $\mathbf{w}^{(i,k)}$ with neighbors $i' \in \mathcal{N}^{(i)}$
    update local model parameters via (120)
  end for
  increment iteration counter: $k := k+1$
end while
$\widehat{\mathbf{w}}^{(i)} := \mathbf{w}^{(i,k)}$ for all nodes $i \in \mathcal{V}$

(simultaneous) execution of the updates (120) at all nodes $i \in \mathcal{V}$ [20, 21]. Loosely speaking, all nodes i relies on a single global clock that maintains the current iteration counter k [63].

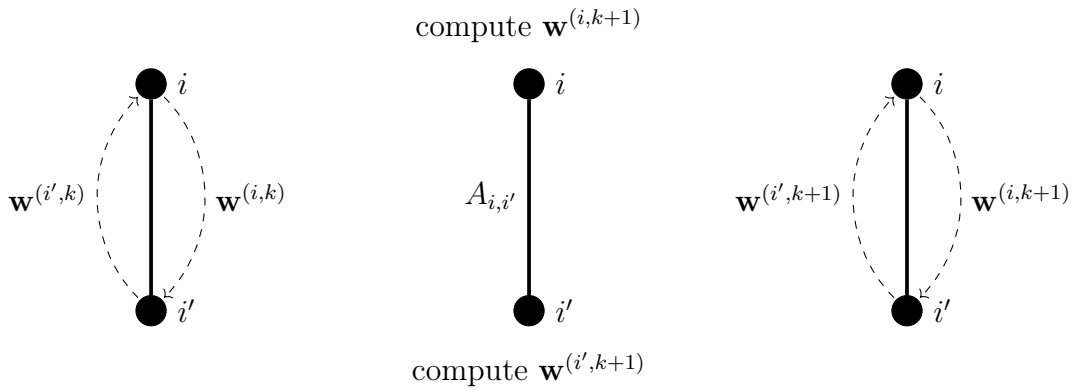


Figure 5.2: Algorithm 4 alternates between message passing across the edges of the FL network (left and right) and updates of local model parameters (centre).

At the beginning of iteration k, each node $i \in \mathcal{V}$ sends its current model parameters $\mathbf{w}^{(i,k)}$ to their neighbors $i' \in \mathcal{N}^{(i)}$. Then, each node $i \in \mathcal{V}$ updates their model parameters according to (120), resulting in the updated model parameters $\mathbf{w}^{(i,k+1)}$. As soon as these local updates are completed, the global clock increments the counter $k \mapsto k+1$ and triggers the next iteration to be executed by all nodes. Fig. 5.2 illustrates the alternating execution of message passing and local updates of Algorithm 4.

The implementation of Algorithm 4 in real-world computational infrastructures might incur deviations from the exact synchronous execution of (120) [64, Sec. 10]. This deviation can be modelled as a perturbation of the gradient step (117) and therefore analyzed using the concepts of Section 4.5 on perturbed GD. Section 8.3 will also discuss the effect of imperfect computation in the context of key requirements for trustworthy FL.

We close this section by generalizing Algorithm 4 which is limited FL networks using local linear models. This generalization, summarized in Algorithm 5, can be used to train parametric local models $\mathcal{H}^{(i)}$ with a differentiable loss function $L_i\left(\mathbf{w}^{(i)}\right)$, for $i = 1, \ldots, n$.

Algorithm 5 FedGD for Parametric Local Models

Input: FL network $\mathcal{G}$; GTV parameter α; learning rate $\eta_{k,i}$
local loss function $L_i\left(\mathbf{w}^{(i)}\right)$ for each $i = 1, \ldots, n$; some stopping criterion.

Output: linear model parameters $\widehat{\mathbf{w}}^{(i)}$ for each node $i \in \mathcal{V}$

Initialize: $k{:=}0$; $\mathbf{w}^{(i,0)}{:=}\mathbf{0}$

1: **while** stopping criterion is not satisfied **do**
2: **for** all nodes $i \in \mathcal{V}$ (simultaneously) **do**
3: share local model parameters $\mathbf{w}^{(i,k)}$ with neighbors $i' \in \mathcal{N}^{(i)}$
4: update local model parameters via

$$\mathbf{w}^{(i,k+1)} := \mathbf{w}^{(i,k)} - \eta_{k,i}\left[\nabla L_i\left(\mathbf{w}^{(i,k)}\right) + 2\alpha \sum_{i' \in \mathcal{N}^{(i)}} A_{i,i'}\left(\mathbf{w}^{(i,k)} - \mathbf{w}^{(i',k)}\right)\right].$$

5: **end for**
6: increment iteration counter: $k{:=}k{+}1$
7: **end while**
8: $\widehat{\mathbf{w}}^{(i)} := \mathbf{w}^{(i,k)}$ for all nodes $i \in \mathcal{V}$

5.4 FedSGD

Consider Algorithm 4 for training local linear models $h^{(i)}(\mathbf{x}) = \mathbf{x}^T\mathbf{w}^{(i)}$ for each node $i = 1, \ldots, n$ of an FL network. Note that step 4 of Algorithm 4 requires to compute the sum (121). It might be infeasible to compute this

sum exactly, e.g., when local datasets are generated by remote devices with limited connectivity. It is then useful to approximate the sum by

$$\underbrace{(2/B)\sum_{r\in\mathcal{B}} \mathbf{x}^{(i,r)}\big(y^{(i,r)} - \big(\mathbf{x}^{(i,r)}\big)^T \mathbf{w}^{(i,k)}\big)}_{\approx(121)}. \tag{122}$$

The approximation (122) uses a subset (so-called *batch*)

$$\mathcal{B} = \big\{\big(\mathbf{x}^{(r_1)}, y^{(r_1)}\big), \ldots, \big(\mathbf{x}^{(r_B)}, y^{(r_B)}\big)\big\}$$

of B randomly chosen data points from $\mathcal{D}^{(i)}$. While (121) requires summing over m data points, the approximation requires to sum over B (typically $B \ll m$) data points.

Inserting the approximation (122) into the gradient step (120) yields the approximate gradient step

$$\mathbf{w}^{(i,k+1)} := \mathbf{w}^{(i,k)} - \eta_{k,i}\Bigg[\underbrace{(2/B)\sum_{r\in\mathcal{B}} \mathbf{x}^{(i,r)}\Big(\big(\mathbf{x}^{(i,r)}\big)^T \mathbf{w}^{(i,k)} - y^{(i,r)}\Big)}_{\approx(121)} + 2\alpha\sum_{i'\in\mathcal{N}^{(i)}} A_{i,i'}\big(\mathbf{w}^{(i,k)} - \mathbf{w}^{(i',k)}\big)\Bigg]. \tag{123}$$

We obtain Algorithm 6 from Algorithm 4 by replacing the gradient step (120) with the approximation (123).

We close this section by generalizing Algorithm 6 which is limited FL networks using local linear models. This generalization, summarized in Algorithm 7, can be used to train parametric local models $\mathcal{H}^{(i)}$ with a differentiable loss function $L_i\big(\mathbf{w}^{(i)}\big)$, for $i = 1, \ldots, n$. Algorithm 7 does not require these local loss function themselves, but only an oracle $\mathbf{g}^{(i)}(\cdot)$ for each node $i = 1, \ldots, n$. For a given vector $\mathbf{w}^{(i)}$, the oracle at node i delivers an approximate gradient

Algorithm 6 FedSGD for Local Linear Models

Input: FL network $\mathcal{G}$; GTV parameter α; learning rate $\eta_{k,i}$; local datasets $\mathcal{D}^{(i)} = \left\{\left(\mathbf{x}^{(i,1)}, y^{(i,1)}\right), \ldots, \left(\mathbf{x}^{(i,m_i)}, y^{(i,m_i)}\right)\right\}$ for each node i; batch size B; some stopping criterion.

Output: linear model parameters $\widehat{\mathbf{w}}^{(i)}$ at each node $i \in \mathcal{V}$

Initialize: $k{:=}0$; $\mathbf{w}^{(i,0)}{:=}\mathbf{0}$

1: **while** stopping criterion is not satisfied **do**
2: **for** all nodes $i \in \mathcal{V}$ (simultaneously) **do**
3: share local model parameters $\mathbf{w}^{(i,k)}$ with all neighbors $i' \in \mathcal{N}^{(i)}$
4: draw fresh batch $\mathcal{B}^{(i)} := \{r_1, \ldots, r_B\}$
5: update local model parameters via (123)
6: **end for**
7: increment iteration counter $k{:=}k{+}1$
8: **end while**
9: $\widehat{\mathbf{w}}^{(i)} := \mathbf{w}^{(i,k)}$ for all nodes $i \in \mathcal{V}$

(or estimate) $\mathbf{g}^{(i)}(\mathbf{w}^{(i)}) \approx \nabla L_i\left(\mathbf{w}^{(i)}\right)$. The analysis of Algorithm 7 can be facilitated by a probabilistic model which interprets the oracle output $\mathbf{g}^{(i)}(\mathbf{w}^{(i)})$ as the realization of a RV. Under such a probabilistic model, we refer to an oracle as unbiased if $\mathbb{E}\big\{\mathbf{g}^{(i)}(\mathbf{w}^{(i)})\big\} = \nabla L_i\left(\mathbf{w}^{(i)}\right)$.

Algorithm 7 FedSGD for Parametric Local Models

Input: FL network $\mathcal{G}$; GTV parameter α; learning rate $\eta_{k,i}$

gradient oracle $\mathbf{g}^{(i)}\big(\,\cdot\,\big)$ for each node $i = 1, \ldots, n$; some stopping criterion.

Output: linear model parameters $\widehat{\mathbf{w}}^{(i)}$ for each node $i \in \mathcal{V}$

Initialize: $k := 0$; $\mathbf{w}^{(i,0)} := \mathbf{0}$

1: **while** stopping criterion is not satisfied **do**

2: **for** all nodes $i \in \mathcal{V}$ (simultaneously) **do**

3: share local model parameters $\mathbf{w}^{(i,k)}$ with neighbors $i' \in \mathcal{N}^{(i)}$

4: update local model parameters via

$$\mathbf{w}^{(i,k+1)} := \mathbf{w}^{(i,k)} - \eta_{k,i}\left[\mathbf{g}^{(i)}\big(\mathbf{w}^{(i,k)}\big) + 2\alpha \sum_{i' \in \mathcal{N}^{(i)}} A_{i,i'}\big(\mathbf{w}^{(i,k)} - \mathbf{w}^{(i',k)}\big)\right].$$

5: **end for**

6: increment iteration counter: $k := k+1$

7: **end while**

8: $\widehat{\mathbf{w}}^{(i)} := \mathbf{w}^{(i,k)}$ for all nodes $i \in \mathcal{V}$

5.5 FedAvg

Consider a FL method that learns model parameters $\widehat{\mathbf{w}} \in \mathbb{R}^d$ of a single (global) linear model from de-centralized collection local datasets $\mathcal{D}^{(i)}$, $i = 1, \ldots, n$.[16] How can we learn $\widehat{\mathbf{w}}$ without exchanging local datasets, but instead only exchanging updates for the model parameters?

One approach is to apply Algorithm 4 to GTVMin (110) with a sufficiently large α. According to our analysis in Chapter 3 (specifically Prop. 3.1), if α is sufficiently large, then the GTVMin solutions $\widehat{\mathbf{w}}^{(i)}$ are almost identical across all nodes $i \in \mathcal{V}$. We can interpret the local model parameters delivered by GTVMin as a local copy of the global model parameters.

Note that the bound in Prop. 3.1 only applies if the FL network (used in GTVMin) is connected. One example of a connected FL network is the star as depicted in Figure 5.3. Here, we choose one node $i = 1$ as a centre node that is connected by an edge with weight $A_{1,i}$ to the remaining nodes $i = 2, \ldots, n$. The star graph uses the minimum number of edges required to connect all n nodes [65].

Instead of using GTVMin with a connected FL network and a large value of α, we can also enforce identical local copies $\widehat{\mathbf{w}}^{(i)}$ via a constraint:

$$\widehat{\mathbf{w}} \in \arg\min_{\mathbf{w} \in \mathcal{S}} \sum_{i \in \mathcal{V}} (1/m_i) \left\| \mathbf{y}^{(i)} - \mathbf{X}^{(i)} \mathbf{w}^{(i)} \right\|_2^2$$
$$\text{with } \mathcal{S} = \big\{ \mathbf{w} = \text{stack}\{\mathbf{w}^{(i)}\}_{i=1}^n : \mathbf{w}^{(i)} = \mathbf{w}^{(i')} \text{ for any } i, i' \in \mathcal{V} \big\}. \quad (124)$$

Here, we use as constraint set the subspace $\mathcal{S}$ defined in (47). The projection of a given collection of local model parameters $\mathbf{w} = \text{stack}\{\mathbf{w}^{(i)}\}$ on $\mathcal{S}$ is given

[16] This setting if a special case of horizontal FL which we discuss in Sec. 6.4.

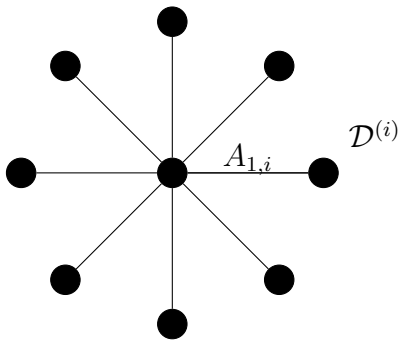


Figure 5.3: Star graph $\mathcal{G}^{(\text{star})}$ with a centre node $i = 1$ representing a server that trains a (global) model which is shared with peripheral nodes. These peripheral nodes represent *clients* generating local datasets. The training process at the server is facilitated by receiving updates on the model parameters from the clients.

by

$$P_{\mathcal{S}}\big(\mathbf{w}\big) = \big(\mathbf{v}^T, \ldots, \mathbf{v}^T\big)^T \text{ with } \mathbf{v} := (1/n) \sum_{i \in \mathcal{V}} \mathbf{w}^{(i)}. \tag{125}$$

We can solve (124) using projected GD from Chapter 4 (see Section 4.6). The resulting projected gradient step for solving (124) is

$$\widehat{\mathbf{w}}^{(i)}_{k+1/2} := \underbrace{\mathbf{w}^{(i,k)} - \eta_{i,k}(2/m_i)\big(\mathbf{X}^{(i)}\big)^T\big(\mathbf{X}^{(i)}\mathbf{w}^{(i,k)} - \mathbf{y}^{(i)}\big)}_{\text{(local gradient step)}} \tag{126}$$

$$\mathbf{w}^{(i,k+1)} := (1/n) \sum_{i' \in \mathcal{V}} \widehat{\mathbf{w}}^{(i')}_{k+1/2} \quad \text{(projection)} \ . \tag{127}$$

We can implement (127) conveniently in a server-client system with each node i being a client:

- First, each node computes the update (126), i.e., a gradient step towards a minimum of the local loss $L_i\left(\mathbf{w}^{(i)}\right) := \left\|\mathbf{y}^{(i)} - \mathbf{X}^{(i)}\mathbf{w}^{(i)}\right\|_2^2$.

- Second, each node i sends the result $\widehat{\mathbf{w}}_k^{(i)}$ of its local gradient step to a server.

- Finally, after receiving the updates $\widehat{\mathbf{w}}_k^{(i)}$ from all nodes $i \in \mathcal{V}$, the server computes the projection step (127). This projection results in the new local model parameters $\mathbf{w}^{(i,k+1)}$ that are sent back to each client i.

The averaging step (127) might take much longer to execute than the local update step (126). Indeed, (127) typically requires transmission of local model parameters from every client $i \in \mathcal{V}$ to a server or central computing unit. Thus, after the client $i \in \mathcal{V}$ has computed the local gradient step (126), it must wait until the server (i) has collected the updates $\widehat{\mathbf{w}}_k^{(i)}$ from all clients and (ii) sent back their average $\mathbf{w}^{(i,k+1)}$ to $i \in \mathcal{V}$.

Instead of using a single gradient step (126),[17] and then being forced to wait for receiving $\mathbf{w}^{(i,k+1)}$ back from the server, a client can make better use of its resources. For example, the device i could execute several local gradient steps (126) to make more progress towards the optimum,

$$\begin{aligned} \mathbf{v}^{(0)} &:= \widehat{\mathbf{w}}_k^{(i)} \\ \mathbf{v}^{(r)} &:= \mathbf{v}^{(r-1)} - \eta_{i,k}(2/m_i)\big(\mathbf{X}^{(i)}\big)^T\big(\mathbf{X}^{(i)}\mathbf{v}^{(r-1)} - \mathbf{y}^{(i)}\big), \text{ for } r = 1, \ldots, R \\ \widehat{\mathbf{w}}_{k+1/2}^{(i)} &:= \mathbf{v}^{(R)}. \end{aligned} \tag{128}$$

We obtain Algorithm 8 by iterating the combination of (128) with the projection step (127).

[17] For a large local dataset, the local gradient step (126) can become computationally too expensive and must be replaced by an approximation, e.g., using a stochastic gradient approximation (122).

Algorithm 8 Server-based FL for linear models

The Server.

Input. Some stopping criterion; list of clients $i = 1, \ldots, n$, number R of local updates.

Output. Trained model parameters $\widehat{\mathbf{w}}^{(\text{global})}$

Initialize. $k := 0$; $\mathbf{w}^{(i,k)} = \mathbf{0}$ for all $i = 1, \ldots, n$

1: **while** stopping criterion is not satisfied **do**

2: Update the global model parameters

$$\widehat{\mathbf{w}}^{(k)} := (1/n) \sum_{i=1}^{n} \mathbf{w}^{(i,k)}.$$

3: Send model parameters $\widehat{\mathbf{w}}^{(k)}$ (and k) to all clients. $i = 1, \ldots, n$

4: Gather update local model parameters $\mathbf{w}^{(i,k+1)}$ from clients $i = 1, \ldots, n$.

5: **Clock Tick.** $k := k + 1$.

6: **end while**

The Client $i \in \{1, \ldots, n\}$**.**

Input. Local dataset $\mathbf{X}^{(i)}, \mathbf{y}^{(i)}$, number of gradient steps R and learning rate (schedule) $\eta_{i,k}$.

1: Receive the current model parameters $\widehat{\mathbf{w}}^{(k)}$ from the server.

2: Update the local model parameters by R gradient steps

$$\begin{aligned} \mathbf{v}^{(0)} &:= \widehat{\mathbf{w}}^{(\text{global})} \\ \mathbf{v}^{(r)} &:= \mathbf{v}^{(r-1)} - \eta_{i,k}(2/m_i)\big(\mathbf{X}^{(i)}\big)^T\big(\mathbf{X}^{(i)}\mathbf{v}^{(r-1)} - \mathbf{y}^{(i)}\big), \text{ for } r = 1, \ldots, R \\ \mathbf{w}^{(i,k+1)} &:= \mathbf{v}^{(R)}. \end{aligned}$$

3: Send the new local model parameters $\mathbf{w}^{(i,k+1)}$ back to server.

One of the most popular server-based FL algorithms, referred to as FedAvg and summarized in Algorithm 9, is obtained by two modifications of Algorithm 8:

- replacing the updates in step 2 at the client in Algorithm 8 with $\mathbf{v}^{(r)} := \mathbf{v}^{(r-1)} - \eta_{i,k}\mathbf{g}\big(\mathbf{v}^{(r)}\big)$ using the gradient approximation $\mathbf{g}^{(i)}\big(\mathbf{v}^{(r)}\big) \approx \nabla L_i\left(\mathbf{v}^{(r)}\right)$,
- using a randomly selected subset $\mathcal{C}^{(k)}$ of clients during each global iteration k.

Algorithm 9 FedAvg [14]

The Server.

Input. List of clients $i = 1, \ldots, n$, number R of local updates

Output. Trained model parameters $\widehat{\mathbf{w}}^{(\text{global})}$

Initialize. $k := 0$; $\widehat{\mathbf{w}}^{(\text{global})} := \mathbf{0}$ for all $i = 1, \ldots, n$

1: **while** stopping criterion is not satisfied **do**

2: randomly select a subset $\mathcal{C}^{(k)}$ of clients

3: send $\widehat{\mathbf{w}}^{(\text{global})}$ to all clients $i \in \mathcal{C}^{(k)}$

4: receive updated model parameters $\mathbf{w}^{(i)}$ from clients $i \in \mathcal{C}^{(k)}$

5: update global model parameters

$$\widehat{\mathbf{w}}^{(\text{global})} := \left(1/\left|\mathcal{C}^{(k)}\right|\right) \sum_{i \in \mathcal{C}^{(k)}} \mathbf{w}^{(i)}.$$

6: increase iteration counter $k := k+1$

7: **end while**

Client $i \in \{1, \ldots, n\}$, with local loss function $L_i(\cdot)$

1: receive global model parameters $\widehat{\mathbf{w}}^{(\text{global})}$ from server

2: update local model parameters by R approximate gradient steps

$$\begin{aligned} \mathbf{v}^{(0)} &:= \widehat{\mathbf{w}}^{(\text{global})} \\ \mathbf{v}^{(r)} &:= \mathbf{v}^{(r-1)} - \eta_{i,k} \underbrace{\mathbf{g}^{(i)}\left(\mathbf{v}^{(r-1)}\right)}_{\approx \nabla L_i\left(\mathbf{v}^{(r-1)}\right)}, \text{ for } r = 1, \ldots, R \\ \mathbf{w}^{(i)} &:= \mathbf{v}^{(R)}. \end{aligned} \tag{129}$$

3: return $\mathbf{w}^{(i)}$ back to server

5.6 FedProx

A central challenge in FedAvg (Algorithm 9) is selecting an appropriate number of local updates, R, in (129). In each iteration, all clients perform exactly R approximate gradient steps. However, [66] argues that enforcing a uniform number R across clients can degrade performance in certain FL settings. To mitigate this, they propose an alternative to (129) for the local update step. This alternative is given by

$$\mathbf{w}^{(i)} := \operatorname*{argmin}_{\mathbf{v} \in \mathbb{R}^d} \left[L_i\left(\mathbf{v}\right) + (1/\eta) \left\| \mathbf{v} - \widehat{\mathbf{w}}^{(\text{global})} \right\|_2^2 \right]. \tag{130}$$

We have already encountered an update of the form (130) in Section 4.7. Indeed, (130) is the application of the proximal operator of $L_i\left(\mathbf{v}\right)$ (see (96)) to the current model parameters. We obtain Algorithm 10 from Algorithm 9 by replacing the local update step (129) with (130). Empirical studies have shown that Algorithm 10 outperforms FedAvg (Algorithm 9) for FL applications with a high-level of heterogeneity among the computational capabilities of devices $i = 1, \ldots, n$ and the statistical properties of their local datasets $\mathcal{D}^{(i)}$ [66].

As the notation in (130) indicates, the parameter η plays a role similar to the learning rate of a gradient step (79). It controls the size of the neighbourhood of $\mathbf{w}^{(i,k)}$ over which (130) optimizes the local loss function $L_i\left(\cdot\right)$. Choosing a small η forces the update (130) to not move too far from the current model parameters $\mathbf{w}^{(i,k)}$.

The core computation (131) of FedProx Algorithm 10 can be interpreted as form of regularization (see Sec. 2.6). Indeed, we obtain (131) from (26) by

- replacing the average squared error loss with the local loss function

Algorithm 10 FedProx [66]

The Server.

Input. List of clients $i = 1, \ldots, n$

Output. Trained model parameters $\widehat{\mathbf{w}}^{(\text{global})}$

Initialize. $k := 0$; $\widehat{\mathbf{w}}^{(\text{global})} := \mathbf{0}$ for all $i = 1, \ldots, n$

1: **while** stopping criterion is not satisfied **do**

2: randomly select a subset $\mathcal{C}^{(k)}$ of clients

3: send $\widehat{\mathbf{w}}^{(\text{global})}$ to all clients $i \in \mathcal{C}^{(k)}$

4: receive updated model parameters $\mathbf{w}^{(i)}$ from clients $i \in \mathcal{C}^{(k)}$

5: update global model parameters

$$\widehat{\mathbf{w}}^{(\text{global})} := \left(1/\left|\mathcal{C}^{(k)}\right|\right) \sum_{i \in \mathcal{C}^{(k)}} \mathbf{w}^{(i)}.$$

6: increase iteration counter $k := k+1$

7: **end while**

Client $i \in \{1, \ldots, n\}$, with local loss function $L_i(\cdot)$

1: receive global model parameters $\widehat{\mathbf{w}}^{(\text{global})}$ from server

2: update local model parameters by

$$\mathbf{w}^{(i)} := \underset{\mathbf{v} \in \mathbb{R}^d}{\operatorname{argmin}} \left[L_i(\mathbf{v}) + (1/\eta) \left\| \mathbf{v} - \widehat{\mathbf{w}}^{(\text{global})} \right\|_2^2 \right] \tag{131}$$

3: return $\mathbf{w}^{(i)}$ back to server

$L_i(\mathbf{v})$,

- using the regularizer

$$\mathcal{R}\{\mathbf{v}\} := \left\|\mathbf{v} - \widehat{\mathbf{w}}^{(\text{global})}\right\|_2^2, \tag{132}$$

- and the regularization parameter $\alpha := 1/\eta$.

Note that Algorithms 10 and 9 provide only an abstract description (or mathematical model) of a practical FL system. The details of their actual implementation, such as providing means for synchronous communication between the server and all clients (see steps 4 and 3 in Algorithm 10) is beyond the scope of this book. Instead, we refer the reader to relevant literature on the implementation of distributed computing systems [21, 67].

5.7 FedRelax

We now apply a simple block-coordinate minimization method [20] to solve GTVMin (51). To this end, we rewrite (51) as

$$\begin{aligned}
&\widehat{\mathbf{w}} \in \arg\min_{\mathbf{w}\in\mathbb{R}^{d\cdot n}} \underbrace{\sum_{i\in\mathcal{V}} f^{(i)}(\mathbf{w})}_{=:f^{(\text{GTV})}(\mathbf{w})} \\
&\quad \text{with } f^{(i)}(\mathbf{w}) := L_i\left(\mathbf{w}^{(i)}\right) + (\alpha/2)\sum_{i'\in\mathcal{N}^{(i)}} A_{i,i'}\left\|\mathbf{w}^{(i)} - \mathbf{w}^{(i')}\right\|_2^2, \\
&\quad \text{and the stacked model parameters } \mathbf{w} = \left(\mathbf{w}^{(1)},\ldots,\mathbf{w}^{(n)}\right)^T.
\end{aligned} \tag{133}$$

According to (133), the objective function of (51) decomposes into components $f^{(i)}(\mathbf{w})$, one for each node $\mathcal{V}$ of the FL network. Moreover, the local model parameters $\mathbf{w}^{(i)}$ influence the objective function only via the components

at the nodes $i \cup \mathcal{N}^{(i)}$. We exploit this structure of (133) to decouple the optimization of the local model parameters $\left\{\widehat{\mathbf{w}}^{(i)}\right\}_{i \in \mathcal{V}}$ as described next.

Consider some local model parameters $\widehat{\mathbf{w}}_k^{(i)}$, for $i = 1, \ldots, n$, at time k. We then update (in parallel) each $\widehat{\mathbf{w}}_k^{(i)}$ by minimizing $f^{(\mathrm{GTV})}(\cdot)$ along $\mathbf{w}^{(i)}$ with the other local model parameters $\mathbf{w}^{(i')} := \widehat{\mathbf{w}}_k^{(i')}$ held fixed for all $i' \neq i$,

$$\begin{aligned}
\widehat{\mathbf{w}}_{k+1}^{(i)} &\in \operatorname*{argmin}_{\mathbf{w}^{(i)} \in \mathbb{R}^d} f^{(\mathrm{GTV})}\left(\widehat{\mathbf{w}}_k^{(1)}, \ldots, \widehat{\mathbf{w}}_k^{(i-1)}, \mathbf{w}^{(i)}, \widehat{\mathbf{w}}_k^{(i+1)}, \ldots\right) \\
&\overset{(133)}{=} \operatorname*{argmin}_{\mathbf{w}^{(i)} \in \mathbb{R}^d} f^{(i)}\left(\widehat{\mathbf{w}}_k^{(1)}, \ldots, \widehat{\mathbf{w}}_k^{(i-1)}, \mathbf{w}^{(i)}, \widehat{\mathbf{w}}_k^{(i+1)}, \ldots\right) \\
&\overset{(133)}{=} \operatorname*{argmin}_{\mathbf{w}^{(i)} \in \mathbb{R}^d} L_i\left(\mathbf{w}^{(i)}\right) + \alpha \sum_{i' \in \mathcal{N}^{(i)}} A_{i,i'} \left\|\mathbf{w}^{(i)} - \widehat{\mathbf{w}}_k^{(i')}\right\|_2^2. \qquad (134)
\end{aligned}$$

The update (134) is an instance of the *non-linear Jacobi algorithm* (applied to (133)) [20, Sec. 3.2.4.]. Another interpretation of (134) is as a variant of block-coordinate optimization [68]. We obtain Algorithm 11 by repeating the update (134) for a sufficient number of iterations.

Algorithm 11 FedRelax for Parametric Models

Input: FL network $\mathcal{G}$ with local loss functions $L_i(\cdot)$, GTV parameter α

Initialize: $k := 0$; $\widehat{\mathbf{w}}_0^{(i)} := \mathbf{0}$

1: **while** stopping criterion is not satisfied **do**
2: **for** all nodes $i \in \mathcal{V}$ in parallel **do**
3: compute $\widehat{\mathbf{w}}_{k+1}^{(i)}$ via (134)
4: share $\widehat{\mathbf{w}}_{k+1}^{(i)}$ with neighbors $\mathcal{N}^{(i)}$
5: **end for**
6: $k := k+1$
7: **end while**

A Model Agnostic Method. The applicability of Algorithm 11 is

limited to FL networks with parametric local models (such as linear regression or ANNs with a common structure). Note that Algorithm 11 results from the application of the non-linear Jacobi method to solve GTVMin (51) for parametric local models. We can generalize Algorithm 11 to non-parametric local models by applying the non-linear Jacobi algorithm to the GTVMin variant (68). This results in the update

$$\widehat{h}_{k+1}^{(i)} \in \operatorname*{argmin}_{h^{(i)} \in \mathcal{H}^{(i)}} L_i\left(h^{(i)}\right) + \alpha \sum_{i' \in \mathcal{N}^{(i)}} A_{i,i'} \underbrace{d^{(h^{(i)}, \widehat{h}_k^{(i')})}}_{\text{see (66)}}. \tag{135}$$

We obtain Algorithm 12 as a model-agnostic variant of Algorithm 11 by replacing the update (134) in its step 3 with the update (135).

Algorithm 12 is model-agnostic in the sense of allowing the devices of an FL network to train parametric as well as non-parametric local models. The only restriction for the local models is that they allow for efficient (approximate) solution of (135). For some choices of local models and loss function, the update (135) can be implemented by basic data augmentation (see Exercise 5.3).

Algorithm 12 Model Agnostic FedRelax

Input: FL network with $\mathcal{G}$, local models $\mathcal{H}^{(i)}$, loss functions $L_i(\cdot)$, GTV parameter α, loss $L(\cdot,\cdot)$ used in (66).

Initialize: $k:=0$; $\widehat{h}_0^{(i)}:=\mathbf{0}$

1: **while** stopping criterion is not satisfied **do**
2: **for** all nodes $i \in \mathcal{V}$ in parallel **do**
3: compute $\widehat{h}_{k+1}^{(i)}$ via (135)
4: **end for**
5: $k:=k+1$
6: **end while**

5.8 A Unified Formulation

The previous sections have presented some widely-used FL algorithms. These algorithms are obtained by applying distributed optimization methods to solve GTVMin. Despite their different formulations they share a common underlying structure. In particular, they can all be expressed as synchronous fixed-point iterations:

$$\widehat{h}^{(i)}_{k+1} = \mathcal{F}^{(i)}\big(\widehat{h}^{(1)}_{k}, \ldots, \widehat{h}^{(n)}_{k}\big). \tag{136}$$

Each operator $\mathcal{F}^{(i)} : \mathcal{H}^{(1)} \times \ldots \times \mathcal{H}^{(n)} \to \mathcal{H}^{(i)}$, represents the local update rule at the $i = 1, \ldots, n$.

Clearly, any FL algorithm of the form 136 is fully specified by its fixed-point operators $\mathcal{F}^{(1)}, \ldots, \mathcal{F}^{(n)}$. This rather trivial observation implies that we can study the behaviour of FL algorithms via analyzing the properties of the operators $\mathcal{F}^{(i)}$, for $i = 1, \ldots, n$.

Some algorithms use time-varying update rules, i.e., the corresponding operator $\mathcal{F}^{(i,k)}$ can vary with k. For parametric local models (parametrized by $\mathbb{R}^d$), we can re-formulate the fixed-point iteration (136) directly in terms of the model parameters

$$\mathbf{w}^{(i,k+1)} = \mathcal{F}^{(i)}\big(\mathbf{w}^{(1,k)}, \ldots, \mathbf{w}^{(n,k)}\big), \tag{137}$$

with operators $\mathcal{F}^{(i)} : \mathbb{R}^{nd} \to \mathbb{R}^{d}$,

5.9 Asynchronous FL Algorithms

The FL algorithms presented so far require a synchronization between the devices $i = 1, \ldots, n$ of an FL network [21, Ch. 6]. Indeed, only when all participating device have completed their local updates (e.g., a variant of a gradient step) and shared the results with its neighbours, the algorithm can proceed to the next iteration [69, Sec. 10], [20, Sec. 1.4].

Using synchronous algorithms for FL can be detrimental for several reasons. Ch. 8 discusses key requirements for trustworthy FL which includes their robustness towards the failure of devices. A synchronous algorithm is prone to failure if even a single device stops to operate. Besides their lack of robustness, synchronous algorithms induce challenges for heterogeneous FL systems, consisting of devices with varying computational power. As a result, faster devices may be forced to idle while waiting for slower devices to complete their updates— a phenomenon commonly referred to as the *straggler effect* [70].

Let us now discuss a generic FL algorithm that does not require synchronization among the devices $i = 1, \ldots, n$ [20, Sec. 6]. We formulate this algorithm for parametric local models, each having its own vector $\mathbf{w}^{(i)}$ of local model parameters. The generalization of this algorithm to non-parametric local models will not pose a significant challenge.

This generic asynchronous algorithm comprises a sequence of *update events*, indexed by $k = 1, 2, \ldots$. During each event k, a subset of nodes $i \in \mathcal{V}$ update their local model parameters according to

$$\mathbf{w}^{(i,k+1)} = \mathcal{F}^{(i)}\big(\mathbf{w}^{(1,k_{i,1})}, \ldots, \mathbf{w}^{(n,k_{i,n})}\big). \tag{138}$$

The set of nodes performing the update (138) during event k is denoted as the active set $\mathcal{A}^{(k)} \subseteq \mathcal{V}$. We can summarize the resulting asynchronous algorithm as

$$\mathbf{w}^{(i,k+1)} = \begin{cases} \mathcal{F}^{(i)}\big(\mathbf{w}^{(1,k_{i,1})}, \ldots, \mathbf{w}^{(n,k_{i,n})}\big) & \text{for } k \in T^{(i)} \\ \mathbf{w}^{(i,k)} & \text{otherwise.} \end{cases} \tag{139}$$

For each node i, the set $T^{(i)}$ consists of those clock ticks during which node i is active, i.e.,

$$T^{(i)} := \big\{k \in \{0,1,\ldots,\} : i \in \mathcal{A}^{(k)}\big\} \tag{140}$$

The update (138) involves an operator $\mathcal{F}^{(i)} : \mathbb{R}^{dn} \to \mathbb{R}^d$ that determines the resulting FL algorithm. Different choices of $\mathcal{F}^{(i)}$ yield different asynchronous FL algorithms. For example, an asynchronous variant of Algorithm 4 can be obtained by setting

$$\mathcal{F}^{(i)}\big(\mathbf{w}^{(1)}, \ldots, \mathbf{w}^{(n)}\big) = \mathbf{w}^{(i)} - \eta\Big(\nabla L_i\big(\mathbf{w}^{(i)}\big) + \sum_{i' \in \mathcal{N}^{(i)}} 2A_{i,i'}\big(\mathbf{w}^{(i)} - \mathbf{w}^{(i')}\big)\Big). \tag{141}$$

Note that the choice (141) involves the local loss functions and the weighted edges of an FL network.

The update (138), at an active node $i \in \mathcal{A}^{(k)}$, involves potentially outdated local model parameters $\mathbf{w}^{(i',k_{i,i'})}$, with $k_{i,i'} \leq k$, for $i' = 1, \ldots, n$. We can interpret the difference $k - k_{i,i'}$ as a measure of the communication delay between node i' and node i.

Depending on the extent of the delays $k - k_{i,i'}$ in the update (138), we distinguish between [20]

- **totally asynchronous algorithms** where delays $k - k_{i,i'}$ can be arbitrarily large and

- **partially asynchronous algorithms** with delays $k - k_{i,i'} \leq B$ bounded by some constant B.

For some choices of $\mathcal{F}^{(i)}$ in (138), a partially asynchronous algorithm can converge for any value of B. However, there also choices of $\mathcal{F}^{(i)}$, for which a partially asynchronous algorithm will only converge if B is sufficiently small [20, Ch. 7].

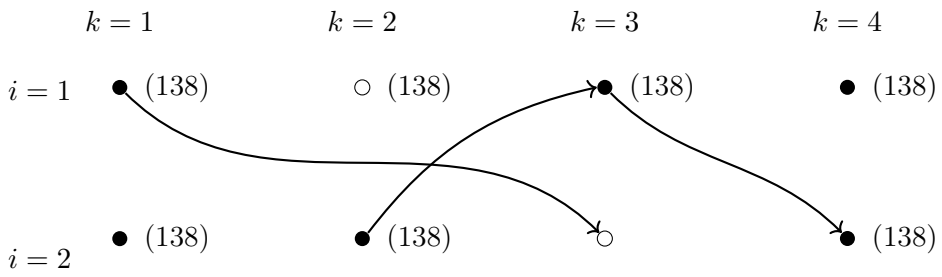


Figure 5.4: Illustration of an asynchronous FL algorithm. During each event k, the active nodes $i \in \mathcal{A}^{(k)} \subseteq \mathcal{V}$ of an FL network update their local model parameters $\mathbf{w}^{(i)}$ by computing (138). Active nodes are depicted as filled circles.

5.10 Exercises

5.1. The convergence speed of Gradient-based methods. Study the convergence speed of (113) for two different collections of local datasets assigned to the nodes of the FL network $\mathcal{G}$ with nodes $\mathcal{V} = \{1, 2\}$ and (unit weight) edges $\mathcal{E} = \{\{1, 2\}\}$. The first collection of local datasets results in the local loss functions $L_1(w) := (w + 5)^2$ and $L_2(w) := 1000(w + 5)^2$. The second collection of local datasets results in the local loss functions $L_1(w) := 1000(w + 5)^2$ and $L_2(w) := 1000(w - 5)^2$. Use a fixed learning rate $\eta := 0.5 \cdot 10^{-3}$ for the iteration (113).

5.2. Convergence Speed for Homogeneous Data. Study the convergence speed of (113) when applied to GTVMin (110) with the following FL network $\mathcal{G}$: Each node $i = 1, \ldots, n$ carries a simple local model with single parameter $w^{(i)}$ and the local loss function $L_i(w) := \left(y^{(i)} - x^{(i)} w^{(i)}\right)^2$. The local dataset consists of a constant $x^{(i)} := 1$ and some $y^{(i)} \in \mathbb{R}$. The edges $\mathcal{E}$ are obtained by connecting each node i with 4 other randomly chosen nodes. We learn model parameters $\widehat{w}^{(i)}$ by repeating (113), starting with the initializations $w^{(i,0)} := y^{(i)}$. Study the dependence of the convergence speed of (113) (towards a solution of (110)) on the value of α in (110).

5.3. Implementing FedRelax via Data Augmentation. Consider the application of Algorithm 12 to an FL network whose nodes carry regression tasks. In particular, each device $i = 1, \ldots, n$ learns a hypothesis $h^{(i)}$ to predict the numeric label $y \in \mathbb{R}$ of a data point with feature vector $\mathbf{x}$. The usefulness of a hypothesis is measured by the average squared error loss incurred on a

labelled local dataset

$$\mathcal{D}^{(i)} := \Big\{ \big(\mathbf{x}^{(1)}, y^{(1)}\big), \ldots, \big(\mathbf{x}^{(m_i)}, \mathbf{x}^{(m_i)}\big) \Big\}.$$

To compare the learnt hypothesis maps at the nodes of an edge $\{i, i'\}$, we use (66) with the squared error loss. Show that the update (135) is equivalent to plain ERM (1) using a dataset $\mathcal{D}$ that is obtained by a specific augmentation of $\mathcal{D}^{(i)}$.

5.4. FedAvg as Fixed-Point Iteration. Consider the Algorithm 8 for training the model parameters $\mathbf{w}^{(i)}$ of local linear models for each $i = 1, \ldots, n$ of an FL network. Each client uses a constant learning rate schedule $\eta_{i,k} := \eta_i$. Try to find a collection of operators $\mathcal{F}^{(i)} : \mathbb{R}^{nd} \to \mathbb{R}^d$, for each node $i = 1, \ldots, n$, such that Algorithm 8 is equivalent to the fixed-point iteration

$$\mathbf{w}^{(i,k+1)} = \mathcal{F}^{(i)}\big(\mathbf{w}^{(1,k)}, \ldots, \mathbf{w}^{(n,k)}\big). \tag{142}$$

5.5. Fixed-Points of a Pseudo-Contraction. Show that a pseudo-contraction cannot have more than one fixed-point.

5.6. FedRelax vs. FedGD Show that that the update in step (4) of FedGD Algorithm 5 is obtained from the update (134) of FedRelax by replacing the local loss function $L_i\left(\mathbf{w}^{(i)}\right)$ with a local approximation by a quadratic function, centered around $\mathbf{w}^{(i,k)}$.

5.11 Proofs

5.11.1 Proof of Proposition 5.1

The first inequality in (115) follows from well-known results on the eigenvalues of a sum of symmetric matrices (see, e.g., [3, Thm 8.1.5]). In particular,

$$\lambda_{\max}\big(\mathbf{Q}\big) \leq \max\big\{ \underbrace{\max_{i=1,\ldots,n} \lambda_d\big(\mathbf{Q}^{(i)}\big)}_{\stackrel{(114)}{=}\lambda_{\max}}, \lambda_{\max}\big(\alpha \mathbf{L}^{(\mathcal{G})} \otimes \mathbf{I}\big)\big\}. \tag{143}$$

The second inequality in (115) uses the following upper bound on the maximum eigenvalue $\lambda_n\big(\mathbf{L}^{(\mathcal{G})}\big)$ of the Laplacian matrix:

$$\begin{aligned}
\lambda_n\big(\mathbf{L}^{(\mathcal{G})}\big) &\stackrel{(a)}{=} \max_{\mathbf{v}\in\mathbb{S}^{(n-1)}} \mathbf{v}^T \mathbf{L}^{(\mathcal{G})} \mathbf{v} \\
&\stackrel{(39)}{=} \max_{\mathbf{v}\in\mathbb{S}^{(n-1)}} \sum_{\{i,i'\}\in\mathcal{E}} A_{i,i'}\big(v_i - v_{i'}\big)^2 \\
&\stackrel{(b)}{\leq} \max_{\mathbf{v}\in\mathbb{S}^{(n-1)}} \sum_{\{i,i'\}\in\mathcal{E}} 2A_{i,i'}\big(v_i^2 + v_{i'}^2\big) \\
&\stackrel{(c)}{=} \max_{\mathbf{v}\in\mathbb{S}^{(n-1)}} \sum_{i\in\mathcal{V}} 2v_i^2 \sum_{i'\in\mathcal{N}^{(i)}} A_{i,i'} \\
&\stackrel{(37)}{\leq} \max_{\mathbf{v}\in\mathbb{S}^{(n-1)}} \sum_{i\in\mathcal{V}} 2v_i^2 d^{(\mathcal{G})}_{\max} \\
&= 2d^{(\mathcal{G})}_{\max}.
\end{aligned} \tag{144}$$

Here, step (a) uses the CFW of eigenvalues [3, Thm. 8.1.2.] and step (b) uses the inequality $(u+v)^2 \leq 2(u^2+v^2)$ for any $u, v \in \mathbb{R}$. For step (c) we use the identity $\sum_{i\in\mathcal{V}} \sum_{i'\in\mathcal{N}^{(i)}} f(i,i') = \sum_{\{i,i'\}} \big(f(i,i') + f(i',i)\big)$ (see Figure 5.5). The bound (144) is essentially tight.[18]

[18]Consider an FL network being a chain (or path).

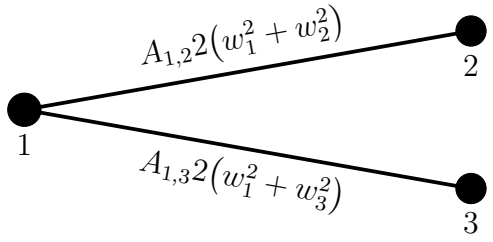


Figure 5.5: Illustration of step (c) in (144).

5.11.2 Proof of Proposition 5.2

Similar to the upper bound (144) we also start with the CFW for the eigenvalues of $\mathbf{Q}$ in (111). In particular,

$$\lambda_1 = \min_{\|\mathbf{w}\|_2^2=1} \mathbf{w}^T\mathbf{Q}\mathbf{w}. \tag{145}$$

We next analyze the right-hand side of (145) by partitioning the constraint set $\{\mathbf{w} : \|\mathbf{w}\|_2^2 = 1\}$ of (145) into two complementary regimes for the optimization variable $\mathbf{w} = \text{stack}\{\mathbf{w}^{(i)}\}$. To define these two regimes, we use the orthogonal decomposition

$$\mathbf{w} = \underbrace{\mathbf{P}_{\mathcal{S}}\mathbf{w}}_{=:\overline{\mathbf{w}}} + \underbrace{\mathbf{P}_{\mathcal{S}^\perp}\mathbf{w}}_{=:\widetilde{\mathbf{w}}} \text{ for subspace } \mathcal{S} \text{ in (47).} \tag{146}$$

Explicit expressions for the orthogonal components $\overline{\mathbf{w}}$, $\widetilde{\mathbf{w}}$ are given by (48) and (49). In particular, the component $\overline{\mathbf{w}}$ satisfies

$$\overline{\mathbf{w}} = \big((\mathbf{c})^T, \ldots, (\mathbf{c})^T\big)^T \text{ with } \mathbf{c} := \text{avg}\big\{\mathbf{w}^{(i)}\big\}_{i=1}^n. \tag{147}$$

Note that

$$\|\mathbf{w}\|_2^2 = \|\overline{\mathbf{w}}\|_2^2 + \|\widetilde{\mathbf{w}}\|_2^2. \tag{148}$$

Regime I. This regime is obtained for $\|\widetilde{\mathbf{w}}\|_2 \geq \rho \|\overline{\mathbf{w}}\|_2$. Since $\|\mathbf{w}\|_2^2 = 1$, and due to (148), we have

$$\|\widetilde{\mathbf{w}}\|_2^2 \geq \rho^2/(1+\rho^2). \tag{149}$$

This implies, in turn, via (46) that

$$\begin{aligned}
\mathbf{w}^T\mathbf{Q}\mathbf{w} &\overset{(111)}{\geq} \alpha \mathbf{w}^T\big(\mathbf{L}^{(\mathcal{G})} \otimes \mathbf{I}\big)\mathbf{w} \\
&\overset{(39),(46)}{\geq} \alpha\lambda_2\big(\mathbf{L}^{(\mathcal{G})}\big)\, \|\widetilde{\mathbf{w}}\|_2^2 \\
&\overset{(149)}{\geq} \alpha\lambda_2\big(\mathbf{L}^{(\mathcal{G})}\big)\rho^2/(1+\rho^2).
\end{aligned} \tag{150}$$

Regime II. This regime is obtained for $\|\widetilde{\mathbf{w}}\|_2 < \rho \|\overline{\mathbf{w}}\|_2$. Here we have $\|\overline{\mathbf{w}}\|_2^2 > (1/\rho^2)\big(1 - \|\overline{\mathbf{w}}\|_2^2\big)$ and, in turn,

$$n\,\|\mathbf{c}\|_2^2 = \|\overline{\mathbf{w}}\|_2^2 > 1/(1+\rho^2). \tag{151}$$

We next develop the right-hand side of (145) according to

$$\begin{aligned}
\mathbf{w}^T\mathbf{Q}\mathbf{w} &\overset{(111)}{\geq} \sum_{i=1}^{n} \big(\mathbf{w}^{(i)}\big)^T\mathbf{Q}^{(i)}\mathbf{w}^{(i)} \\
&\overset{(146)}{\geq} \sum_{i=1}^{n} \big(\mathbf{c} + \widetilde{\mathbf{w}}^{(i)}\big)^T\mathbf{Q}^{(i)}\big(\mathbf{c} + \widetilde{\mathbf{w}}^{(i)}\big) \\
&\overset{(151)}{\geq} \|\overline{\mathbf{w}}\|_2^2 \underbrace{\lambda_1\bigg((1/n)\sum_{i=1}^{n}\mathbf{Q}^{(i)}\bigg)}_{\bar{\lambda}_{\min}} + \sum_{i=1}^{n}\big[2\big(\widetilde{\mathbf{w}}^{(i)}\big)^T\mathbf{Q}^{(i)}\mathbf{c} + \underbrace{\big(\widetilde{\mathbf{w}}^{(i)}\big)^T\mathbf{Q}^{(i)}\widetilde{\mathbf{w}}^{(i)}}_{\geq 0}\big] \\
&\geq \|\overline{\mathbf{w}}\|_2^2\,\bar{\lambda}_{\min} + \sum_{i=1}^{n} 2\big(\widetilde{\mathbf{w}}^{(i)}\big)^T\mathbf{Q}^{(i)}\mathbf{c}.
\end{aligned} \tag{152}$$

To develop (152) further, we note that

$$\begin{aligned}
\bigg|\sum_{i=1}^{n} 2\big(\widetilde{\mathbf{w}}^{(i)}\big)^T\mathbf{Q}^{(i)}\mathbf{c}\bigg| &\overset{(a)}{\leq} 2\lambda_{\max}\,\|\widetilde{\mathbf{w}}\|_2\,\|\overline{\mathbf{w}}\|_2 \\
&\overset{\|\widetilde{\mathbf{w}}\|_2 < \rho\|\overline{\mathbf{w}}\|_2}{\leq} 2\lambda_{\max}\rho\,\|\overline{\mathbf{w}}\|_2^2\,.
\end{aligned} \tag{153}$$

Here, step (a) follows from $\max_{\|\mathbf{y}\|_2=1,\|\mathbf{x}\|_2=1} \mathbf{y}^T\mathbf{Q}\mathbf{x} = \lambda_{\max}$. Inserting (153) into (152) for $\rho = \bar{\lambda}_{\min}/(4\lambda_{\max})$,

$$\mathbf{w}^T\mathbf{Q}\mathbf{w} \geq \|\overline{\mathbf{w}}\|_2^2 \, \bar{\lambda}_{\min}/2 \overset{(151)}{\geq} (1/(1+\rho^2))\bar{\lambda}_{\min}/2 \tag{154}$$

For each $\mathbf{w}$ with $\|\mathbf{w}\|_2^2 = 1$, either (150) or (154) must hold.

6 Main Flavours of FL

Chapter 3 discussed GTVMin as a main design principle for FL algorithms. GTVMin learns local model parameters that optimally balance the individual local loss with their variation across the edges of an FL network. Chapter 5 discussed how to obtain practical FL algorithms. These algorithms solve GTVMin using distributed optimization methods, such as those from Chapter 4.

This chapter discusses important special cases (or "main flavours") of GTVMin obtained for specific construction of local datasets, choices of local models, measures for their variation and the weighted edges of the FL network. We next briefly summarize the resulting main flavours of FL discussed in the following sections.

Section 6.2 discusses single-model FL that learns model parameters of a single (global) model from local datasets. This single-model flavour can be obtained from GTVMin using a connected FL network with large edge weights or, equivalently, a sufficient large value for the GTVMin parameter.

Section 6.3 discusses how clustered federated learning (CFL) is obtained from GTVMin over FL networks with a clustering structure. CFL exploits the presence of clusters (subsets of local datasets) which can be approximated using an i.i.d. assumption. GTVMin captures these clusters if they are well-connected by many (large weight) edges of the FL network.

Section 6.4 discusses horizontal FL which is obtained from GTVMin over an FL network whose nodes carry different subsets of a single underlying global dataset. Loosely speaking, horizontal FL involves local datasets characterized by the same set of features but obtained from different data points from an

underlying dataset.

Section 6.5 discusses vertical federated learning (vertical FL) which is obtained from GTVMin over an FL network whose nodes the same data points but using different features. As an example, consider the local datasets at different public institutions (tax authority, social insurance institute, supermarkets) which contain different informations about the same underlying population (anybody who has a Finnish social security number).

Section 6.6 shows how personalized FL can be obtained from GTVMin by using specific measures for the GTV of local model parameters. For example, using deep neural networks as local models, we might only use the model parameters corresponding to the first few input layers to define the GTV.

6.1 Learning Goals

After this chapter, you will know particular design choices for GTVMin corresponding to some main flavours of FL:

- single-model FL
- CFL (generalization of clustering methods)
- horizontal FL (relation to semi-supervised learning)
- personalized FL/multi-task learning
- vertical FL.

6.2 Single-Model FL

Some FL use cases require to train a single (global) model $\mathcal{H}$ from a decentralized collection of local datasets $\mathcal{D}^{(i)}$, $i = 1, \ldots, n$ [15, 71]. In what follows we assume that the model $\mathcal{H}$ is parametrized by a vector $\mathbf{w} \in \mathbb{R}^d$. Figure 6.1 depicts a server-client architecture for an iterative FL algorithm that generates a sequence of (global) model parameters $\mathbf{w}^{(k)}$, $k = 1, \ldots$. After computing the new model parameters $\mathbf{w}^{(k+1)}$, the server broadcasts it to the devices $i = 1, \ldots, i$ and increments the clock $k := k + 1$. In the next iteration, each device i uses the current global model parameters $\mathbf{w}^{(k)}$ to compute a local update $\mathbf{w}^{(i,k)}$ based on its local dataset $\mathcal{D}^{(i)}$. The precise implementation of this local update step depends on the choice of the global model $\mathcal{H}$ (trained by the server). One example of such a local update has been discussed in Chapter 5 (see (130)).

Chapter 5 already hinted at an alternative to the server-based system in Figure 6.1. Indeed, we might learn local model parameters $\mathbf{w}^{(i)}$ for each client i using a distributed optimization of GTVMin. We can force the resulting model parameters $\mathbf{w}^{(i)}$ to be (approximately) identical by using a connected FL network and a sufficiently large GTVMin parameter α.

To minimize the computational complexity of the resulting single-model FL system, we prefer FL networks with a small number of edges such as the star graph in Figure 5.3 [65]. However, to increase the robustness against node/link failures we should use an FL network with more edges. This redundancy helps to ensure that the FL network is connected even after removing some of its edges [72].

Much like the server-based system from Figure 6.1, GTVMin-based meth-

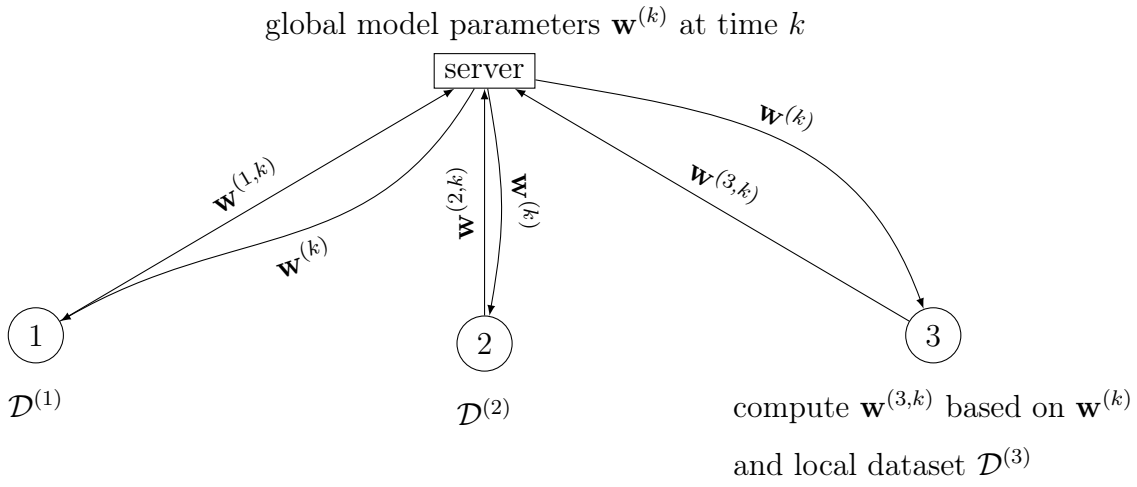


Figure 6.1: The operation of a server-based (or centralized) FL system during iteration k. First, the server broadcasts the current global model parameters $\mathbf{w}^{(k)}$ to each client $i \in \mathcal{V}$. Each device i then computes the update $\mathbf{w}^{(i,k)}$ by combining the previous model parameters $\mathbf{w}^{(k)}$ (received from the server) and its local dataset $\mathcal{D}^{(i)}$. The updates $\mathbf{w}^{(i,k)}$ are then sent back to the server who aggregates them to obtain the updated global model parameters $\mathbf{w}^{(k+1)}$.

ods using a star graph offers a single point of failure which is the server in Figure 6.1 or the centre node in Figure 5.3. Chapter 8 will discuss the robustness of GTVMin-based FL systems in slightly more detail (see Section 8.3).

6.3 Clustered FL

Single-model FL systems require the local datasets to be well approximated as i.i.d. realizations from a common underlying probability distribution. However, requiring homogeneous local datasets, generated from the same probability distribution, might be overly restrictive. Indeed, the local datasets might be heterogeneous and need to be modelled using different probability distribution [18, 37].

CFL relaxes the requirement of a common probability distribution underlying all local datasets. Instead, we approximate subsets of local datasets as i.i.d. realizations from a common probability distribution. In other words, CFL assumes that local datasets form clusters. Each cluster $\mathcal{C} \subseteq \mathcal{V}$ has a cluster-specific probability distribution $p^{(\mathcal{C})}$.

The idea of CFL is to pool the local datasets $\mathcal{D}^{(i)}$ in the same cluster $\mathcal{C}$ to obtain a training set to learn cluster-specific $\widehat{\mathbf{w}}^{(\mathcal{C})}$. Each node $i \in \mathcal{C}$ then uses these learnt model parameters $\widehat{\mathbf{w}}^{(\mathcal{C})}$. A main challenge in CFL is that the cluster assignments of the local datasets are unknown in general.

To determine a cluster $\mathcal{C}$, we could apply basic clustering methods, such as k-means or Gaussian mixture model (GMM) to vector representations for local datasets [6, Ch. 5]. We can obtain a vector representation for local dataset $\mathcal{D}^{(i)}$ via the learnt model parameters $\widehat{\mathbf{w}}$ of some parametric ML model

that is trained on $\mathcal{D}^{(i)}$.

We can also implement CFL via GTVMin with a suitably chosen FL network. In particular, the FL network should contain many edges (with large weight) between nodes in the same cluster and few edges (with a small weight) between nodes in different clusters. To fix ideas, consider the FL network in Figure 6.3, which contains a cluster $\mathcal{C} = \{1, 2, 3\}$.

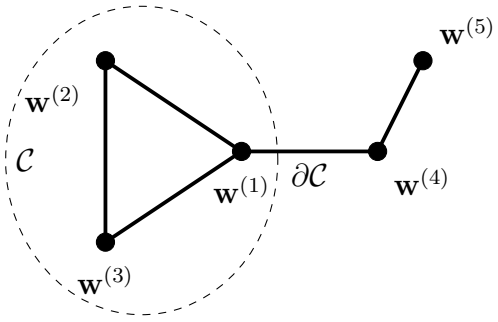


Figure 6.2: The solution of GTVMin (51) are local model parameters that are approximately identical for all nodes in a tight-knit cluster $\mathcal{C}$.

Chapter 3 discussed how the eigenvalues of the Laplacian matrix can be used to measure the connectivity of $\mathcal{G}$. Similarly, we can measure the connectivity of a cluster $\mathcal{C}$ via the eigenvalue $\lambda_2\big(\mathbf{L}^{(\mathcal{C})}\big)$ of the Laplacian matrix $\mathbf{L}^{(\mathcal{C})}$ of the induced sub-graph $\mathcal{G}^{(\mathcal{C})}$:[19] The larger $\lambda_2\big(\mathbf{L}^{(\mathcal{C})}\big)$, the better the connectivity among the nodes in $\mathcal{C}$. While $\lambda_2\big(\mathbf{L}^{(\mathcal{C})}\big)$ describes the intrinsic connectivity of a cluster $\mathcal{C}$, we also need to characterize its connectivity with the other nodes in the FL network. To this end, we use the cluster boundary

$$|\partial\mathcal{C}| := \sum_{\{i,i'\}\in\partial\mathcal{C}} A_{i,i'} \text{ with } \partial\mathcal{C} := \big\{\{i,i'\} \in \mathcal{E} : i \in \mathcal{C}, i' \notin \mathcal{C}\big\}. \tag{155}$$

[19]The graph $\mathcal{G}^{(\mathcal{C})}$ consists of the nodes in $\mathcal{C}$ and the edges $\{i,i'\} \in \mathcal{E}$ for $i,i' \in \mathcal{C}$.

Note that for a single-node cluster $\mathcal{C} = \{i\}$, the cluster boundary coincides with the node degree, $|\partial\mathcal{C}| = d^{(i)}$ (see (36)).

Intuitively, GTVMin tends to deliver (approximately) identical model parameters $\mathbf{w}^{(i)}$ for nodes $i \in \mathcal{C}$ if $\lambda_2\big(\mathbf{L}^{(\mathcal{C})}\big)$ is large and the cluster boundary $|\partial\mathcal{C}|$ is small. The following result makes this intuition more precise for the special case of GTVMin (110) for local linear models.

Proposition 6.1. *Consider an FL network $\mathcal{G}$ which contains a cluster $\mathcal{C}$ of local datasets with labels $\mathbf{y}^{(i)}$ and feature matrix $\mathbf{X}^{(i)}$ related via*

$$\mathbf{y}^{(i)} = \mathbf{X}^{(i)}\overline{\mathbf{w}}^{(\mathcal{C})} + \boldsymbol{\varepsilon}^{(i)}, \text{ for all } i \in \mathcal{C}. \tag{156}$$

We learn local model parameters $\widehat{\mathbf{w}}^{(i)}$ via solving GTVMin (110). If the cluster is connected, the error component

$$\widetilde{\mathbf{w}}^{(i)} := \widehat{\mathbf{w}}^{(i)} - (1/|\mathcal{C}|)\sum_{i\in\mathcal{C}} \widehat{\mathbf{w}}^{(i)} \tag{157}$$

is upper bounded as

$$\sum_{i\in\mathcal{C}} \big\|\widetilde{\mathbf{w}}^{(i)}\big\|_2^2 \leq \frac{1}{\alpha\lambda_2\big(\mathbf{L}^{(\mathcal{C})}\big)}\Bigg[\sum_{i\in\mathcal{C}} \frac{1}{m_i}\big\|\boldsymbol{\varepsilon}^{(i)}\big\|_2^2 + \alpha\,|\partial\mathcal{C}|\,2\Bigg(\big\|\overline{\mathbf{w}}^{(\mathcal{C})}\big\|_2^2 + R^2\Bigg)\Bigg]. \tag{158}$$

Here, we used $R := \max_{i'\in\mathcal{V}\setminus\mathcal{C}} \big\|\widehat{\mathbf{w}}^{(i')}\big\|_2$.

Proof. See Sec. 6.9.1. □

The bound (158) depends on the cluster $\mathcal{C}$ (via the eigenvalue $\lambda_2\big(\mathbf{L}^{(\mathcal{C})}\big)$ and the boundary $|\partial\mathcal{C}|$) and the GTVMin parameter α. Using a larger $\mathcal{C}$ might result in a decreased eigenvalue $\lambda_2\big(\mathbf{L}^{(\mathcal{C})}\big)$.[20] According to (158), we

[20]Consider an FL network (with uniform edge weights) that contains a fully connected cluster $\mathcal{C}$ which is connected via a single edge with another node $i' \in \mathcal{V}\setminus\mathcal{C}$ (see Figure 6.3). Compare the corresponding eigenvalues $\lambda_2\big(\mathbf{L}^{(\mathcal{C})}\big)$ and $\lambda_2\big(\mathbf{L}^{(\mathcal{C}')}\big)$ of $\mathcal{C}$ and the enlarged cluster $\mathcal{C}' := \mathcal{C}\cup\{i'\}$.

should then increase α to maintain a small deviation $\widetilde{\mathbf{w}}^{(i)}$ of the learnt local model parameters from their cluster-wise average. Thus, increasing α in (51) enforces its solutions to be approximately constant over increasingly larger subsets (clusters) of nodes (see Figure 6.3).

For a connected FL network $\mathcal{G}$ and a sufficiently large α, the solution of GTVMin consists of learnt model parameters $\mathbf{w}^{(i)}$ that are approximately identical for all $\mathcal{V} = 1, \ldots, n$. The resulting approximation error is quantified by Prop. 6.1 for the extreme case where the entire FL network forms a single cluster, i.e., $\mathcal{C} = \mathcal{V}$. Trivially, the cluster boundary is then equal to 0 and the bound (158) specializes to (64).

We hasten to add that the bound (158) only applies for local datasets that conform with the probabilistic model (156). In particular, it assumes that all cluster nodes $i \in \mathcal{C}$ have identical model parameters $\overline{\mathbf{w}}^{(\mathcal{C})}$. Trivially, this is no restriction if we allow for arbitrary error terms $\boldsymbol{\varepsilon}^{(i)}$ in the probabilistic model (158). However, as soon as we place additional assumptions on these error terms (such as being realizations of i.i.d. Gaussian RVs) we should verify their validity using principled statistical tests [32, 73]. Finally, we might replace $\left\|\overline{\mathbf{w}}^{(\mathcal{C})}\right\|_2^2$ in (158) with an upper bound for this quantity.

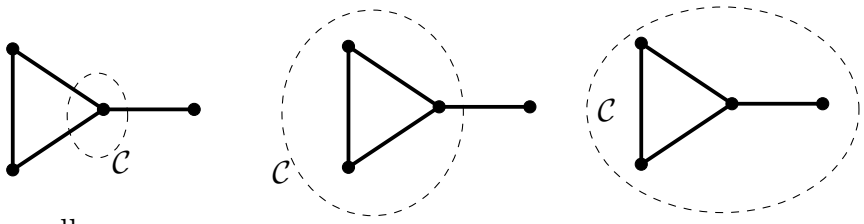


Figure 6.3: The solutions of GTVMin (51) become increasingly clustered for increasing α.

6.4 Horizontal FL

Horizontal FL uses local datasets $\mathcal{D}^{(i)}$, for $i \in \mathcal{V}$, that contain data points characterized by the same features [74]. As illustrated in Figure 6.4, we can think of each local dataset $\mathcal{D}^{(i)}$ as being a subset (or batch) of an underlying global dataset

$$\mathcal{D}^{(\text{global})} := \left\{ \left(\mathbf{x}^{(1)}, y^{(1)}\right), \ldots, \left(\mathbf{x}^{(m)}, y^{(m)}\right) \right\}.$$

In particular, local dataset $\mathcal{D}^{(i)}$ is constituted by the data points of $\mathcal{D}^{(\text{global})}$ with indices in $\{r_1, \ldots, r_{m_i}\}$,

$$\mathcal{D}^{(i)} := \left\{ \left(\mathbf{x}^{(r_1)}, y^{(r_1)}\right), \ldots, \left(\mathbf{x}^{(r_{m_i})}, y^{(r_{m_i})}\right) \right\}.$$

We can interpret horizontal FL as a generalization of semi-supervised learning (SSL) [75]: For some local datasets $i \in \mathcal{U}$ we might not have access to the label values of data points. Still, we can use the features of the data points to construct (the weighted edges of) the FL network. To implement SSL, we can solve GTVMin using a trivial loss function $L_i\left(\mathbf{w}^{(i)}\right) = 0$ for each unlabelled node $i \in \mathcal{U}$. Solving GTVMin delivers model parameters $\mathbf{w}^{(i)}$

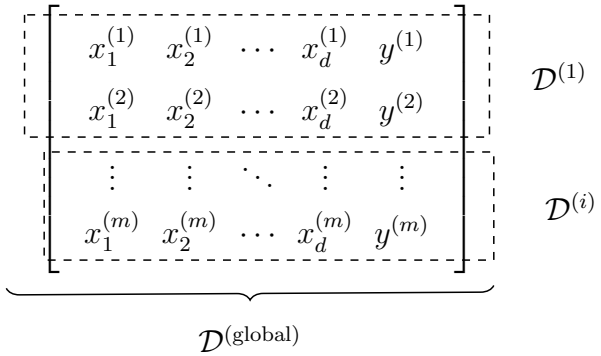


Figure 6.4: Horizontal FL uses the same features to characterize data points in different local datasets. Different local datasets are constituted by different subsets of an underlying global dataset.

for all nodes i (including the unlabelled ones $\mathcal{U}$). GTVMin-based methods combine the information in the labelled local datasets $\mathcal{D}^{(i)}$, for $i \in \mathcal{V} \setminus \mathcal{U}$ and their connections (via the edges of $\mathcal{G}$) with nodes in $\mathcal{U}$ (see Figure 6.4).

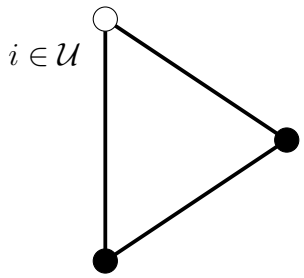


Figure 6.5: Horizontal FL includes SSL as a special case. SSL involves a subset of nodes $\mathcal{U}$, for which the local datasets do not contain labels. We can take this into account by using the trivial loss function $L_i(\cdot) = 0$ for each node $i \in \mathcal{U}$. However, we can still use the features in $\mathcal{D}^{(i)}$ to construct an FL network $\mathcal{G}$.

6.5 Vertical FL

Vertical FL uses local datasets that are constituted by the same (identical!) data points. However, each local dataset uses a different choice of features to characterize these data points [76]. Formally, vertical FL applications revolve around an underlying global dataset

$$\mathcal{D}^{(\text{global})} := \left\{ \left(\mathbf{x}^{(1)}, y^{(1)} \right), \ldots, \left(\mathbf{x}^{(m)}, y^{(m)} \right) \right\}.$$

Each data point in the global dataset is characterized by d' features $\mathbf{x}^{(r)} = \left(x_1^{(r)}, \ldots, x_{d'}^{(r)} \right)^T$. The global dataset can only be accessed indirectly via local datasets that use different subsets of the feature vectors $\mathbf{x}^{(r)}$ (see Figure 6.6).

For example, the local dataset $\mathcal{D}^{(i)}$ consists of feature vectors

$$\mathbf{x}^{(i,r)} = \left(x_{j_1}^{(r)}, \ldots, x_{j_d}^{(r)} \right)^T.$$

Here, we used a subset $\mathcal{F}^{(i)} := \{j_1, \ldots, j_d\}$ of the original d' features in $\mathbf{x}^{(r)}$. There is typically one node i' with a a local datasets that contains the label values $y^{(1)}, \ldots, y^{(m)}$.

A potential toy application for vertical FL is a national social insurance system. The global dataset comprises data points representing individuals enrolled in the system. Each individual is characterized by multiple sets of features sourced from different institutions. Healthcare providers contribute medical records, offering health-related features. Financial service providers, such as banks, supply financial features. Some individuals participate in retailer loyalty programs, which generate consumer behaviour features. Additionally, social network accounts can provide real-time data on user activities and mobility patterns, further enriching the available features. Since these diverse data sources belong to separate entities, vertical FL enables collaborative learning while preserving data privacy.

6.6 Personalized Federated Learning

Consider GTVMin (51) for learning local model parameters $\widehat{\mathbf{w}}^{(i)}$ for each local dataset $\mathcal{D}^{(i)}$. If the value of α in (51) is not too large, the local model parameters $\widehat{\mathbf{w}}^{(i)}$ can be different for each $i \in \mathcal{V}$. However, the local model parameters are still coupled via the GTV term in (51).

For some FL use-cases we should use different coupling strengths for different components of the local model parameters. For example, if local models are deep ANNs we might enforce the parameters of input layers to be identical while the parameters of the deeper layers might be different for each local dataset.

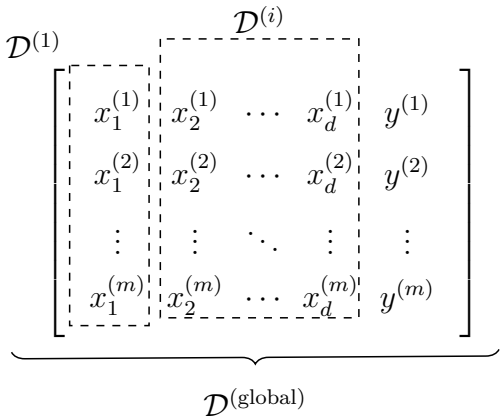


Figure 6.6: Vertical FL uses local datasets that are derived from the same data points. The local datasets differ in the choice of features used to characterize the common data points.

The partial parameter sharing for local models can be implemented in many different ways [77, Sec. 4.3.]:

- One way is to use a choice of the GTV penalty that is different from $\phi = \big\|\mathbf{w}^{(i)} - \mathbf{w}^{(i')}\big\|_2^2$. In particular, we could construct the penalty function as a combination of two terms,

$$\phi\big(\mathbf{w}^{(i)} - \mathbf{w}^{(i')}\big) := \alpha^{(1)}\phi^{(1)}\big(\mathbf{w}^{(i)} - \mathbf{w}^{(i')}\big) + \alpha^{(2)}\phi^{(2)}\big(\mathbf{w}^{(i)} - \mathbf{w}^{(i')}\big). \quad (159)$$

  The functions $\phi^{(1)}$ and $\phi^{(2)}$ measure different components of the variation $\mathbf{w}^{(i)} - \mathbf{w}^{(i')}$ across the edge $\{i, i'\} \in \mathcal{E}$. For example, we might construct $\phi^{(1)}$ and $\phi^{(2)}$ by (66) with different choices for the dataset $\mathcal{D}^{\{i,i'\}}$.

- Moreover, we might use different regularization strengths $\alpha^{(1)}$ and $\alpha^{(2)}$ for different penalty components in (159) to enforce different subsets of

the model parameters to be clustered with different granularity (cluster size).

- For local models being deep ANNs, we might want to enforce the low-level layers (closer to the input) to have the same model parameters (weights and bias terms), while deeper (closer to the output) layers can have different model parameters. Figure 6.7 illustrates this setting for local models constituted by ANNs with a single hidden layer.

- Yet another technique for partial sharing of model parameters is to train a hyper-model which, in turn, is used to initialize the training of local models [78].

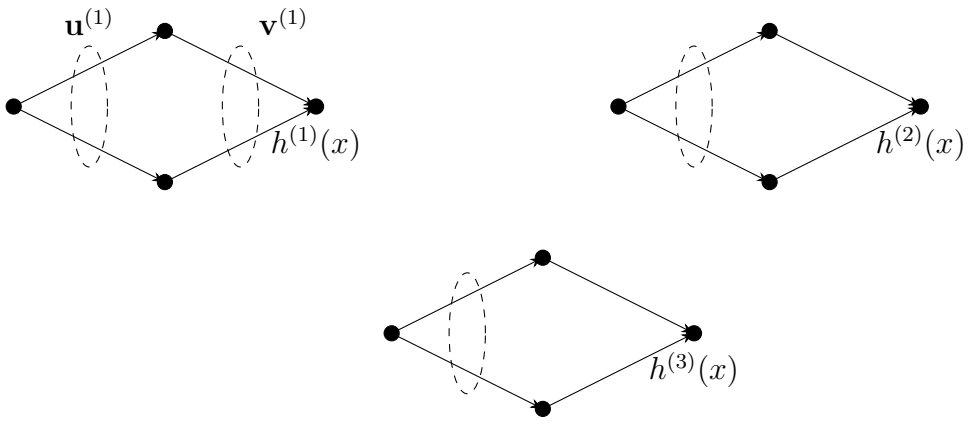


Figure 6.7: Personalized FL with local models being ANNs with a single hidden layer. The ANN $h^{(i)}$ is parametrized by the vector $\mathbf{w}^{(i)} = \left(\left(\mathbf{u}^{(i)}\right)^T, \left(\mathbf{v}^{(i)}\right)^T\right)^T$, with parameters $\mathbf{u}^{(i)}$ of the hidden layer and the parameters $\mathbf{v}^{(i)}$ of the output layer. We couple the training of $\mathbf{u}^{(i)}$ via GTVMin using the discrepancy measure $\phi = \left\|\mathbf{u}^{(i)} - \mathbf{u}^{(i')}\right\|_2^2$.

6.7 Few-Shot Learning

Some machine learning (ML) applications involve data points belonging to a large number of different categories. A prime example is the detection of a specific object in a given image [79, 80]. Here, the object category is the label $y \in \mathcal{Y}$ of a data point (image). The label space $\mathcal{Y}$ is constituted by the possible object categories and, in turn, can be quite large. Moreover, for some categories, we might only have a few example images in the training set.

Few-shot learning exploits structural similarities between object categories to accurately detect objects with limited (or even no) training examples. A principled approach to few-shot learning is via GTVMin, which leverages relational information between categories.

To formalize this approach, we define an FL network $\mathcal{G} = (\mathcal{V}, \mathcal{E}, \mathbf{A})$, where each node $i \in \mathcal{V}$ corresponds to an element of the label space $\mathcal{Y}$. The edge weights $\mathbf{A}$ encode prior knowledge about category relationships, providing a structured way to propagate information between object categories.

Each node i in $\mathcal{G}$ represents a distinct object category and corresponding object detector. Solving GTVMin yields model parameters $\widehat{\mathbf{w}}^{(i)}$ for each of these specialized object detectors. The coupling of these tailored object detectors via GTVMin enables knowledge transfer across categories, improving detection performance even in low-data regimes.

6.8 Exercises

6.1. Horizontal FL of a Linear Model [69, Sec. 8.2] Linear regression learns the model parameters of a linear model by minimizing the average

squared error loss on a given dataset $\mathcal{D}$. Consider an application where the data points are gathered by different devices. We can model such an application using an FL network with nodes i carrying different subsets of $\mathcal{D}$. Construct an instance of GTVMin such that its solutions coincide (approximately) with the solution of plain vanilla linear regression.

6.2. Vertical FL of a Linear Model [69, Sec. 8.3] Linear regression learns the model parameters of a linear model by minimizing the average squared error loss on a given dataset $\mathcal{D}$. Consider an application where the features of a data point are measured by different devices. We can model such an application using an FL network with nodes i carrying different features of the same dataset $\mathcal{D}$. In particular, node i carries the features x_j with $j \in \mathcal{F}^{(i)}$. Construct an instance of GTVMin such that its solutions coincide (approximately) with the solution of plain vanilla linear regression.

6.9 Proofs

6.9.1 Proof of Proposition 6.1

To verify (158), we follow a similar argument as used in the proof (see Section 3.8.1) of Prop. 3.1.

First, we decompose the objective function $f\big(\mathbf{w}\big)$ in (110) as follows:

$$f(\mathbf{w}) = \underbrace{\sum_{i\in\mathcal{C}}(1/m_i)\left\|\mathbf{y}^{(i)}-\mathbf{X}^{(i)}\mathbf{w}^{(i)}\right\|_2^2+\alpha\Bigg[\sum_{i,i'\in\mathcal{C}}A_{i,i'}\left\|\mathbf{w}^{(i)}-\mathbf{w}^{(i')}\right\|_2^2+\sum_{\{i,i'\}\in\partial\mathcal{C}}A_{i,i'}\left\|\mathbf{w}^{(i)}-\mathbf{w}^{(i')}\right\|_2^2\Bigg]}_{=:f'\left(\mathbf{w}\right)} + f''\big(\mathbf{w}\big). \tag{160}$$

Note that only the first component f' depends on the local model parameters $\mathbf{w}^{(i)}$ of cluster nodes $i \in \mathcal{C}$. Let us introduce the shorthand $f'\big(\mathbf{w}^{(i)}\big)$ for the function obtained from $f'(\mathbf{w})$ for varying $\mathbf{w}^{(i)}$, $i \in \mathcal{C}$, but fixing $\mathbf{w}^{(i')} := \widehat{\mathbf{w}}^{(i')}$ for $i' \notin \mathcal{C}$.

We obtain the bound (158) via a proof by contradiction: If (158) does not hold, the local model parameters $\overline{\mathbf{w}}^{(i)} := \overline{\mathbf{w}}^{(\mathcal{C})}$, for $i \in \mathcal{C}$, result in a smaller value $f'\big(\overline{\mathbf{w}}^{(i)}\big) < f'\big(\widehat{\mathbf{w}}^{(i)}\big)$ than the choice $\widehat{\mathbf{w}}^{(i)}$, for $i \in \mathcal{C}$. This would contradict the fact that $\widehat{\mathbf{w}}^{(i)}$ is a solution to (110).

First, note that

$$
\begin{aligned}
f'\big(\overline{\mathbf{w}}^{(i)}\big) &= \sum_{i\in\mathcal{C}}(1/m_i)\,\big\|\mathbf{y}^{(i)}-\mathbf{X}^{(i)}\overline{\mathbf{w}}^{(\mathcal{C})}\big\|_2^2 \\
&\qquad +\alpha\Bigg[\sum_{\substack{\{i,i'\}\in\mathcal{E}\\ i,i'\in\mathcal{C}}} A_{i,i'}\,\big\|\overline{\mathbf{w}}^{(\mathcal{C})}-\overline{\mathbf{w}}^{(\mathcal{C})}\big\|_2^2+\sum_{\substack{\{i,i'\}\in\mathcal{E}\\ i\in\mathcal{C},i'\notin\mathcal{C}}} A_{i,i'}\,\Big\|\overline{\mathbf{w}}^{(\mathcal{C})}-\widehat{\mathbf{w}}^{(i')}\Big\|_2^2\Bigg] \\
&\overset{(156)}{=} \sum_{i\in\mathcal{C}}(1/m_i)\,\big\|\boldsymbol{\varepsilon}^{(i)}\big\|_2^2+\alpha\sum_{\substack{\{i,i'\}\in\mathcal{E}\\ i\in\mathcal{C},i'\notin\mathcal{C}}} A_{i,i'}\,\Big\|\overline{\mathbf{w}}^{(\mathcal{C})}-\widehat{\mathbf{w}}^{(i')}\Big\|_2^2 \\
&\overset{(a)}{\leq} \sum_{i\in\mathcal{C}}(1/m_i)\,\big\|\boldsymbol{\varepsilon}^{(i)}\big\|_2^2+\alpha\sum_{\substack{\{i,i'\}\in\mathcal{E}\\ i\in\mathcal{C},i'\notin\mathcal{C}}} 2A_{i,i'}\Big(\,\big\|\overline{\mathbf{w}}^{(\mathcal{C})}\big\|_2^2+\Big\|\widehat{\mathbf{w}}^{(i')}\Big\|_2^2\Big) \\
&\leq \sum_{i\in\mathcal{C}}(1/m_i)\,\big\|\boldsymbol{\varepsilon}^{(i)}\big\|_2^2+\alpha\,|\partial\mathcal{C}|\,2\Big(\,\big\|\overline{\mathbf{w}}^{(\mathcal{C})}\big\|_2^2+R^2\Big). \qquad (161)
\end{aligned}
$$

Step (a) uses the inequality $\|\mathbf{u}+\mathbf{v}\|_2^2 \leq 2\big(\,\|\mathbf{u}\|_2^2+\|\mathbf{v}\|_2^2\,\big)$ which is valid for any two vectors $\mathbf{u},\mathbf{v}\in\mathbb{R}^d$.

On the other hand,

$$
\begin{aligned}
f'\big(\widehat{\mathbf{w}}^{(i)}\big) &\geq \alpha\sum_{i,i'\in\mathcal{C}} A_{i,i'}\underbrace{\Big\|\widehat{\mathbf{w}}^{(i)}-\widehat{\mathbf{w}}^{(i')}\Big\|_2^2}_{\overset{(157)}{=}\big\|\widetilde{\mathbf{w}}^{(i)}-\widetilde{\mathbf{w}}^{(i')}\big\|_2^2} \\
&\overset{(46)}{\geq} \alpha\lambda_2\big(\mathbf{L}^{(\mathcal{C})}\big)\sum_{i\in\mathcal{C}}\big\|\widetilde{\mathbf{w}}^{(i)}\big\|_2^2. \qquad (162)
\end{aligned}
$$

If the bound (158) would not hold, then by (162) and (161) we would obtain $f'\big(\widehat{\mathbf{w}}^{(i)}\big) > f'\big(\overline{\mathbf{w}}^{(i)}\big)$, which contradicts the fact that $\widehat{\mathbf{w}}^{(i)}$ solves (110).

7 Graph Learning

Chapter 3 introduced GTVMin as a flexible design principle for FL algorithms. In particular, Chapter 5 discusses FL algorithms that arise from the application of optimization methods, such as the gradient-based methods from Chapter 4, to solve GTVMin.

The computational and statistical properties of these algorithms depend crucially on the properties of the underlying FL network. For example, the amount of computation and communication required by FL systems typically grows with the number of edges of the FL network. Moreover, the connectivity of the FL network steers the pooling of local datasets into clusters that share common model parameters.

In some applications, domain expertise can guide the choice of the FL network. However, it might also be useful to learn the FL network in a more data-driven fashion. This chapter discusses methods that learn an FL network solely from a given collection of local datasets and corresponding local loss functions.

The outline of this chapter is as follows: Section 7.2 discusses how the computational and statistical properties of Algorithm 4 from Chapter 5 can guide the construction of the FL network. Section 7.3 presents some ideas for how to measure the discrepancy (lack of similarity) between two local datasets.

The discrepancy measure is an important design choice of the graph learning methods discussed in Section 7.4. We formulate these methods as the optimization of edge weights given the discrepancy measure for any pair of local datasets. The formulation as an optimization problem allows to include

connectivity constraints such as a minimum value for each node degree.

7.1 Learning Goals

After completing this chapter, you will

- have intuition for how the computational and statistical properties of GTVMin-based methods depend on the structure of the FL network,
- know some measures for (dis-)similarity (or discrepancy) between local datasets,
- be able to learn a graph from pairwise similarities and structural constraints, such as prescribed maximum node degree.[21]

7.2 Edges as Design Choice

Consider the GTVMin instance (53) for learning the model parameters of a local linear model for each local dataset $\mathcal{D}^{(i)}$. To solve (53), we use Algorithm 4 as a message-passing implementation of the basic gradient step (113). Note that GTVMin (53) is defined for a given FL network $\mathcal{G}$. Therefore, the choice of $\mathcal{G}$ is critical for the statistical and computational properties of Algorithm 4.

Statistical Properties. The statistical properties of Algorithm 4 can be assessed via a probabilistic model for the local datasets. One important example of a probabilistic model is the clustering assumption (156) of CFL

[21] Graphs with a small maximum node degree are an example of *sparse graphs* [81].

(see Section 3.4.1). For CFL, we would like to learn similar model parameters for nodes in the same cluster.

According to Prop. 6.1, the GTVMin solutions will be approximately constant over $\mathcal{C}$, if $\lambda_2\big(\mathbf{L}^{(\mathcal{C})}\big)$ is large and the cluster boundary $|\partial\mathcal{C}|$ is small. Here, $\lambda_2\big(\mathbf{L}^{(\mathcal{C})}\big)$ denotes the smallest non-zero eigenvalue of the Laplacian matrix associated with the induced sub-graph $\mathcal{G}^{(\mathcal{C})}$.

Roughly speaking, $\lambda_2\big(\mathbf{L}^{(\mathcal{C})}\big)$ will be larger if there are more edges connecting the nodes in $\mathcal{C}$. This informal statement can be made precise using a celebrated result from spectral graph theory, known as Cheeger's inequality [39, Ch. 21]. Alternatively, we can analyse $\lambda_2\big(\mathbf{L}^{(\mathcal{C})}\big)$ by interpreting (or approximating) the induced sub-graph $\mathcal{G}^{(\mathcal{C})}$ as the realization of an Erdős-Rényi (ER) graph.[22]

The ER graph with nodes $\mathcal{C}$ postulates that two nodes $i, i' \in \mathcal{C}$ are connected by an edge with probability p_e. In particular, the presence of edges is determined by the realizations $b^{(\{i,i'\})}$ of i.i.d. RVs, one for each pair $\{i, i'\}$ of nodes. As a consequence, the presence of an edge between a given pair of nodes does not depend on the presence of an edge between any other pair of nodes.

The absence of dependencies among the edges facilities the analysis of the ER graph. For example, the Laplacian matrix $\mathbf{L}^{(\mathrm{ER})}$ of an ER graph can be written as a sum of statistically independent random matrices $b^{(\{i,i'\})}\mathbf{T}^{(\{i,i'\})}$ which allows to apply matrix concentration inequalities [82, 83].

If we interpret a graph $\mathcal{G}$ as the realization of an ER graph, also the node degrees $d^{(i)}$ as well as the eigenvalue $\lambda_2\big(\mathbf{L}^{(\mathcal{C})}\big)$ become realizations of RVs. The

[22] This approximation is particularly useful if the FL network $\mathcal{G}$ itself is (close to) a typical realization of an ER graph.

expected node degree is given as $\mathbb{E}\big\{d^{(i)}\big\} = p_e(|\mathcal{C}|-1)$. With high probability, a realization of an ER graph has maximum node degree

$$d_{\max}^{(\mathcal{G})} \approx \mathbb{E}\big\{d^{(i)}\big\} = p_e(|\mathcal{C}|-1). \tag{163}$$

Thus, increasing the parameter p_e results in a larger node degree (i.e., a higher connectivity) of $\mathcal{G}^{(\mathcal{C})}$.

We can approximate $\lambda_2\big(\mathbf{L}^{(\mathcal{C})}\big)$ with the second smallest eigenvalue $\lambda_2\big(\overline{\mathbf{L}}\big)$ of the expected Laplacian matrix $\overline{\mathbf{L}} := \mathbb{E}\big\{\mathbf{L}^{(\mathcal{C})}\big\} = |\mathcal{C}|p_e\mathbf{I} - p_e\mathbf{1}\mathbf{1}^T$. A simple calculation reveals that $\lambda_2\big(\overline{\mathbf{L}}\big) = |\mathcal{C}|p_e$. Thus, we have the approximation

$$\lambda_2\big(\mathbf{L}^{(\mathcal{C})}\big) \approx \lambda_2\big(\overline{\mathbf{L}}\big) = |\mathcal{C}|p_e \overset{(163)}{\approx} d_{\max}^{(\mathcal{G})}. \tag{164}$$

The precise quantification of the approximation error in (164) is beyond the scope of this book. We refer the reader to relevant literature on the theory of random graphs [82, 84].

Computational Properties. The computational complexity of Algorithm 4 depends on the amount of computation required by a single iteration of its steps (3) and (4). Clearly, the *per-iteration* complexity of Algorithm 4 increases with increasing node degrees $d^{(i)}$. Indeed, step (3) requires the communication of local model parameters across each edge of the FL network. This can be implemented by different physical channels, such as a short-range wireless link or an optical fibre cable [85, 86].

To summarize, using an FL network with a smaller $d^{(i)}$ translates into a smaller amount of computation and communication needed during a single iteration of Algorithm 4. Trivially, the per-iteration complexity of Algorithm 4 is minimized by $d^{(i)} = 0$, i.e., an empty FL network without any edges ($\mathcal{E} = \emptyset$). However, the overall computational complexity of Algorithm 4 also

depends on the number of iterations required to achieve an approximate solution to GTVMin (53).

According to (83), the convergence speed of the gradient steps (120) used in Algorithm 4 depends on the condition number $\underbrace{\lambda_{nd}(\mathbf{Q})}_{=\lambda_{\max}(\mathbf{Q})}/\lambda_1(\mathbf{Q})$ of the matrix $\mathbf{Q}$ in (111). Algorithm 4 tends to require fewer iterations when the condition number of $\mathbf{Q}$ is small (closer to 1). This condition number, which is the ratio between the largest and smallest eigenvalues, of $\mathbf{Q}$ tends to be smaller for a smaller ratio between the maximum node degree $d^{(\mathcal{G})}_{\max}$ and the eigenvalue $\lambda_2\big(\mathbf{L}^{(\mathcal{G})}\big)$ (see (115) and (116)). Thus, for a given maximum node degree $d^{(\mathcal{G})}_{\max}$, we should place the edges of an FL network such that $\lambda_2\big(\mathbf{L}^{(\mathcal{G})}\big)$ is large and, in turn, Algorithm 4 converges faster.

Spectral graph theory also provides upper bounds on $\lambda_2\big(\mathbf{L}^{(\mathcal{G})}\big)$ in terms of the node degrees [39, 40, 87]. These upper bounds can be used as a baseline for practical constructions of the FL network: If some construction results in a value $\lambda_2\big(\mathbf{L}^{(\mathcal{G})}\big)$ close to the upper bound, there is little benefit in trying to further improve the construction (in the sense of achieving higher $\lambda_2\big(\mathbf{L}^{(\mathcal{G})}\big)$). The next result provides an example of such an upper bound.

Proposition 7.1. *Consider an FL network $\mathcal{G}$ with $n > 1$ nodes and associated Laplacian matrix $\mathbf{L}^{(\mathcal{G})}$. Then, $\lambda_2\big(\mathbf{L}^{(\mathcal{G})}\big)$ cannot exceed the node degree $d^{(i)}$ of any node by more than a factor $n/(n-1)$. In other words,*

$$\lambda_2\big(\mathbf{L}^{(\mathcal{G})}\big) \leq \frac{n}{n-1} d^{(i)}, \text{ for every } i = 1, \ldots, n. \tag{165}$$

Proof. The bound (165) follows from (43) and evaluating the quadratic form

$\mathbf{w}^T \mathbf{L}^{(\mathcal{G})} \mathbf{w}$ for the specific vector

$$\widetilde{\mathbf{w}} = \sqrt{\frac{n}{n-1}} \Big(-(1/n), \ldots, \underbrace{1-(1/n)}_{\tilde{w}^{(i)}}, \ldots, -(1/n) \Big)^T .$$

Note that the vector $\widetilde{\mathbf{w}}$ is tailored to a specific node $i \in \mathcal{V}$: its only positive entry is $\tilde{w}^{(i)} = 1-(1/n)$. A basic calculation reveals that $\|\widetilde{\mathbf{w}}\| = 1$ and $\widetilde{\mathbf{w}}^T \mathbf{1} = 0$, i.e., it is feasible for the optimization in (43). □

Alternative (and potentially tighter) upper bounds on $\lambda_2\big(\mathbf{L}^{(\mathcal{G})}\big)$ can be found in the graph theory literature [38, 39, 84, 88].

The per-iteration complexity of FL algorithms increases with increasing node degrees $d^{(i)}$ (and, in turn, total number of edges) of the FL network $\mathcal{G}$. On the other hand, the number of iterations required by Algorithm 4 will typically decrease with increasing $\lambda_2\big(\mathbf{L}^{(\mathcal{G})}\big)$. By (165), the eigenvalue $\lambda_2\big(\mathbf{L}^{(\mathcal{G})}\big)$ can only be large if the node degrees $d^{(i)}$ (and, in turn, the total number of edges) are sufficiently large.[23] Fig. 7.1 illustrates the typical dependency of the per-iteration complexity and number of iterations required by FL algorithms.

7.3 Measuring (Dis-)Similarity Between Datasets

The main idea behind GTVMin is to enforce similar model parameters at two nodes i, i' that are connected by an edge $\{i, i'\}$ with (relatively) large edge weight $A_{i,i'}$. In general, the edges (and their weights) of the FL network are a design choice. Placing an edge between two nodes i, i' is typically only

[23] Some recent work studies graph constructions that maximize $\lambda_2\big(\mathbf{L}^{(\mathcal{G})}\big)$ for a given (prescribed) maximum node degree $d^{(\mathcal{G})}_{\max} = \max_{i \in \mathcal{V}} d^{(i)}$ [61, 89].

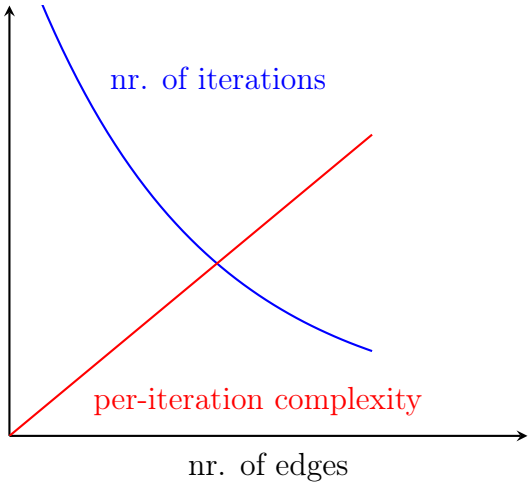


Figure 7.1: The per-iteration complexity and number of iterations required by Algorithm 4 depends on the number of edges of the underlying FL network in different manners.

useful if the local datasets $\mathcal{D}^{(i)}, \mathcal{D}^{(i')}$ (generated by devices i, i') have similar statistical properties. We next discuss different approaches to measuring the similarity or, equivalently, the discrepancy (the lack of similarity) between two local datasets.

The first approach is based on a probabilistic model, i.e., we interpret the local dataset $\mathcal{D}^{(i)}$ as realizations of RVs with some parametrized probability distribution $p^{(i)}\big(\mathcal{D}^{(i)}; \mathbf{w}^{(i)}\big)$. We can then measure the discrepancy between $\mathcal{D}^{(i)}$ and $\mathcal{D}^{(i')}$ via the Euclidean distance $\big\|\mathbf{w}^{(i)} - \mathbf{w}^{(i')}\big\|_2$ between the parameters $\mathbf{w}^{(i)}, \mathbf{w}^{(i')}$ of the probability distributions.

In most FL applications, we do not know the parameters of the probability distribution $p^{(i)}\big(\mathcal{D}^{(i)}; \mathbf{w}^{(i)}\big)$ underlying a local dataset.[24] We might still be

[24] One exception is when we generate the local dataset by drawing i.i.d. realizations from $p^{(i)}\big(\mathcal{D}^{(i)}; \mathbf{w}^{(i)}\big)$.

able to estimate these parameters using established statistical techniques such as ML [6, Ch. 3]. Given the estimates $\widehat{\mathbf{w}}^{(i)}, \widehat{\mathbf{w}}^{(i')}$ for the model parameters, we can then compute the discrepancy measure $d^{(i,i')} := \left\|\widehat{\mathbf{w}}^{(i)} - \widehat{\mathbf{w}}^{(i')}\right\|_2$.

Example. Consider local datasets, each consisting of a single number $y^{(i)} = w^{(i)} + n^{(i)}$ with $n^{(i)} \sim \mathcal{N}(0,1)$ and model parameter $w^{(i)}$, for $i = 1, \ldots, n$. The ML estimator for $w^{(i)}$ is then obtained as $\hat{w}^{(i)} = y^{(i)}$ [30, 90] and, in turn, the resulting discrepancy measure $d^{(i,i')} := \left|y^{(i)} - y^{(i')}\right|$ [91].

Example. Consider an FL network with nodes $i \in \mathcal{V}$ that carry local datasets $\mathcal{D}^{(i)}$. Each $\mathcal{D}^{(i)}$ consists of data points with labels from the label space $\mathcal{Y}^{(i)}$. We can measure the similarity between i, i' by the fraction of data points in $\mathcal{D}^{(i)} \bigcup \mathcal{D}^{(i')}$ with label values in $\mathcal{Y}^{(i)} \cap \mathcal{Y}^{(i')}$ [92].

Example. Consider local datasets $\mathcal{D}^{(i)}$ constituted by images of handwritten digits $0, 1, \ldots, 9$. We model a local dataset using a hierarchical probabilistic model: Each node $i \in \mathcal{V}$ is assigned a deterministic but unknown distribution $\boldsymbol{\alpha}^{(i)} = \left(\alpha_1^{(i)}, \ldots, \alpha_9^{(i)}\right)$. The entry $\alpha_j^{(i)}$ is the fraction of images at node i that show digit j. We interpret the labels $y^{(i,1)}, \ldots, y^{(i,m_i)}$ as realizations of i.i.d. RVs, with values in $\{0, 1, \ldots, 9\}$ and distributed according to $\boldsymbol{\alpha}^{(i)}$. The features are interpreted as realizations of RVs with conditional distribution $p(\mathbf{x}|y)$ which is the same for all nodes $i \in \mathcal{V}$. We can then estimate the dis-similarity between nodes i, i' via the distance between (estimations of) the parameters $\boldsymbol{\alpha}^{(i)}$ and $\boldsymbol{\alpha}^{(i')}$.

The above discrepancy measure construction (using estimates for the parameters of a probabilistic model) is a special case of a more generic two-step approach:

- First, we determine a vector representation $\mathbf{z}^{(i)} \in \mathbb{R}^{m'}$ for each node

$i \in \mathcal{V}$ [6, 93].

- Second, we construct the discrepancy $d^{(i,i')}$ between nodes i, i' via the distance between the representation vectors $\mathbf{z}^{(i)}, \mathbf{z}^{(i')} \in \mathbb{R}^{m'}$, e.g., using $d^{(i,i')} := \left\|\mathbf{z}^{(i)} - \mathbf{z}^{(i')}\right\|$.

Let us next discuss three implementations of the first step to obtain the vector for each node i.

Parametrized Probabilistic Models. If we use a parametrized probabilistic model $p\big(\mathcal{D}^{(i)}; \mathbf{w}^{(i)}\big)$ for the local dataset $\mathcal{D}^{(i)}$, we can use an estimator $\widehat{\mathbf{w}}^{(i)}$ as $\mathbf{z}^{(i)}$. One popular approach for estimating the model parameters of a probabilistic model is the ML principle [6].

Gradients. Let us next discuss a construction for the vector representation $\mathbf{z}^{(i)} \in \mathbb{R}^{m'}$ that is motivated by stochastic gradient descent (SGD) (see Section 5.4). In particular, we could define the discrepancy between two local datasets by interpreting them as two potential batches used by SGD to train a model. If these two batches have similar statistical properties, then their corresponding gradient approximations (122) should be aligned. This suggests to use the gradient $\nabla f(\mathbf{w}')$ of the average loss $f(\mathbf{w}) := (1/|\mathcal{D}^{(i)}|) \sum_{(\mathbf{x},y) \in \mathcal{D}^{(i)}} L\left((\mathbf{x}, y), h^{(\mathbf{w})}\right)$ as a vector representation $\mathbf{z}^{(i)}$ for $\mathcal{D}^{(i)}$. We can generalize this construction, for parametric local models $\mathcal{H}^{(i)}$, by using the gradient of the local loss function,

$$\mathbf{z}^{(i)} := \nabla L_i\left(\mathbf{v}\right). \tag{166}$$

Note that the construction (166) requires a choice of the model parameters $\mathbf{v}$ at which the gradient is evaluated.

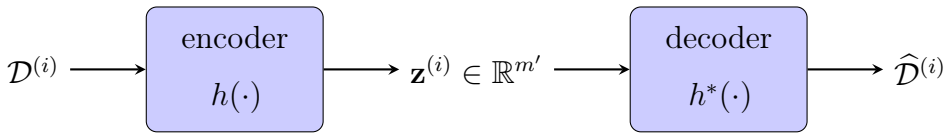


Figure 7.2: An autoencoder consists of an encoder, mapping the input to a latent vector, and a decoder which tries to reconstruct the input as accurately as possible. The encoder and the decoder are trained jointly by minimizing some quantitative measure (a loss) of the reconstruction error (see [6, Ch. 9]). When using a local dataset as input, we can use the latent vector as its vector representation.

Feature Learning. We can also use an autoencoder [93, Ch. 14] to learn a vector representation for a local dataset. In particular, we fed it into an encoder ANN which has been trained jointly with a decoder ANN on some learning task. Figure 7.2 illustrates a generic autoencoder setup.

7.4 Graph Learning Methods

Assume we have constructed a useful measure $d^{(i,i')} \in \mathbb{R}_{+}$ for the discrepancy between any two local datasets $\mathcal{D}^{(i)}, \mathcal{D}^{(i')}$. We could then construct an FL network by connecting each node i with its nearest neighbours. The nearest neighbors of i are those other nodes $i' \in \mathcal{V} \setminus \{i\}$ with the smallest discrepancy $d^{(i,i')}$. We next discuss an alternative to the nearest-neighbour graph construction. This alternative approach formulates graph learning as a constrained linear optimization problem.

Let us measure the usefulness of a given choice of the edge weights

$A_{i,i'} \in \mathbb{R}_+$ via

$$\sum_{i,i' \in \mathcal{V}} A_{i,i'} d^{(i,i')}. \tag{167}$$

The objective function (167) penalizes having a large edge weight $A_{i,i'}$ between two nodes i, i' with a large discrepancy $d^{(i,i')}$. Note that the objective function (167) is minimized by the trivial choice $A_{i,i'} = 0$, i.e., the empty graph without any edges $\mathcal{E} = \emptyset$.

As discussed in Section 7.2, the FL network should have a sufficient amount of edges to ensure that the GTVMin solutions are useful model parameters. Indeed, the desired pooling effect of GTVMin requires the eigenvalue $\lambda_2\big(\mathbf{L}^{(\mathcal{G})}\big)$ to be sufficiently large. Ensuring a large $\lambda_2\big(\mathbf{L}^{(\mathcal{G})}\big)$ requires, in turn, that the FL network $\mathcal{G}$ contains a sufficiently large number of edges and corresponding node degrees (see (164)).

We can enforce the presence of edges (with positive weight) by adding constraints to (167). For example, we might require

$$A_{i,i} = 0 \ , \ \sum_{i' \neq i} A_{i,i'} = d^{(\mathcal{G})}_{\max} \text{ for all } i \in \mathcal{V}, \ A_{i,i'} \in [0,1] \text{ for all } i, i' \in \mathcal{V}. \tag{168}$$

The constraints (168) require that each node i is connected with other nodes using total edge weight (weighted node degree)

$$d^{(\mathcal{G})}_{\max} = \sum_{i' \neq i} A_{i,i'}.$$

Combining the constraints (168) with the objective function (167) results

in the following graph learning principle,

$$\widehat{A}_{i,i'} \in \operatorname*{argmin}_{A_{i,i'}=A_{i',i}} \sum_{i,i' \in \mathcal{V}} A_{i,i'} d^{(i,i')} \tag{169}$$

$$A_{i,i'} \in [0,1] \text{ for all } i, i' \in \mathcal{V},$$

$$A_{i,i} = 0 \text{ for all } i \in \mathcal{V},$$

$$\sum_{i' \neq i} A_{i,i'} = d^{(\mathcal{G})}_{\max} \text{ for all } i \in \mathcal{V}.$$

Note that (169) is a special case of the constrained quadratic minimization problem (92). In fact, (169) is equivalent to a linear program [48, Sec. 4.3]. We can therefore use projected GD from Section 4.6 to compute (approximate) solutions to (169).

The first constraint in (169) requires each edge weight to belong to the interval $[0,1]$. The second constraint prohibits any self-loops in the resulting FL network. Note that adding self-loops to an FL network has no effect on the resulting GTVMin-based method (see (51)). The last constraint of the learning principle (169) enforces a regular FL network: Each node i having the same weighted node degree $d^{(i)} = \sum_{i' \neq i} A_{i,i'} = d^{(\mathcal{G})}_{\max}$.

For some FL applications, it might be detrimental to insist on identical node degrees of the FL network. Instead, we might prefer other structural properties such as a small total number of edges or the presence of a few *hub nodes* with exceptionally large node degrees [13, 91].

We can enforce an upper bound on the total number $E_{\max}$ of edges by

modifying the last constraint in (169),

$$\widehat{A}_{i,i'} \in \operatorname*{argmin}_{A_{i,i'}=A_{i',i}} \sum_{i,i' \in \mathcal{V}} A_{i,i'} d^{(i,i')} \tag{170}$$

$$A_{i,i'} \in [0,1] \text{ for all } i, i' \in \mathcal{V},$$

$$A_{i,i} = 0 \text{ for all } i \in \mathcal{V},$$

$$\sum_{i',i \in \mathcal{V}} A_{i,i'} = E_{\max}.$$

The problem has a closed-form solution as explained in [91]: It is obtained by placing the edges between those pairs $i, i' \in \mathcal{V}$ that result in the smallest discrepancy $d^{(i,i')}$. However, it might still be useful to solve (170) via iterative optimization methods such as the gradient-based methods discussed in Chapter 4. These methods can be implemented in a fully distributed fashion as message passing over an underlying communication network [69]. This communication network might be significantly different from the learnt FL network.[25]

7.5 Exercises

7.1. A Simple Ranking Approach. Consider a collection of devices $i = 1, \ldots, n = 100$, each carrying a local dataset that consists of a single vector $\mathbf{x} \in \mathbb{R}^{(m_i)}$. The vectors $\mathbf{x} \in \mathbb{R}^{m_i}$ can be modelled as statistically independent (across nodes) RVs. Moreover, the vector $\mathbf{x} \in \mathbb{R}^{m_i}$ is a realization of a multivariate normal distribution $\mathcal{N}(c_i \mathbf{1}, \mathbf{I})$ with given (fixed) quantities

[25] Can you think of FL application domains where the connectivity (e.g., via short-range wireless links) of two clients $i, i' \in \mathcal{V}$ might also reflect the pair-wise similarities between probability distributions of local datasets $\mathcal{D}^{(i)}, \mathcal{D}^{(i')}$?

$c_i \in \{-1, 1\}$. We construct an FL network by determining for each node i its neighbors $\mathcal{N}^{(i)}$ as follows

- we randomly select a fraction $\mathcal{B}^{(i)}$ of 10 percent from all other nodes
- we define $\mathcal{N}^{(i)}$ as those $i' \in \mathcal{B}^{(i)}$ whose corresponding values $|(1/m_i)\mathbf{1}^T\mathbf{x}^{(i)} - (1/m_{i'})\mathbf{1}^T\mathbf{x}^{(i')}|$ are among the 3 smallest.

Analyze the probability that some neighborhood $\mathcal{N}^{(i)}$ contains a node i' such that $c_i \neq c_{i'}$.

8 Trustworthy FL

The Story So Far. We have introduced GTVMin as a main design principle for FL in Chapter 3. Chapter 5 applied the gradient-based methods from Chapter 4 to solve GTVMin, resulting in practical FL systems. Our focus has been on the computational and statistical properties of these FL systems. In this and the following chapters, we shift the focus from technical properties to the trustworthiness of FL systems.

Section 8.2 reviews key requirements for trustworthy AI, which includes FL systems, that have been put forward by the European Union [94, 95]. We will also discuss how these requirements guide the design choices for GTVMin-based methods. Our focus will be on the three design criteria for trustworthy FL: privacy, robustness and explainability. This chapter covers robustness and explainability, the leakage and protection of privacy in FL systems is the subject of Chapter 9.

Section 8.3 discusses the robustness of FL systems against perturbations of local datasets and computations. A special type of perturbation is the intentional modification or poisoning of local datasets (see Chapter 10). Section 8.4 introduces a measure for the (subjective) explainability of the personalized models trained by FL systems.

8.1 Learning Goals

After completing this chapter, you will

- know some key requirements for trustworthy artificial intelligence (trustworthy AI),

- be familiar with quantitative measures of robustness, and explainability
- have some intuition about how to ensure robustness, privacy protection and explainability via suitable design choices for local models, loss functions, and FL network in GTVMin.

8.2 Seven Key Requirements by the EU

As part of its AI strategy, the European Commission set up the High-Level Expert Group on Artificial Intelligence (AI HLEG) in 2018. This group put forward seven key requirements for trustworthy AI [94, 95]. We next discuss in a step-by-step fashion how these requirements guide the design choices for GTVMin-based FL systems.

8.2.1 KR1 - Human Agency and Oversight.

"*..AI systems should support human autonomy and decision-making, as prescribed by the principle of respect for human autonomy. This requires that AI systems should both act as enablers to a democratic, flourishing and equitable society by supporting the user's agency and foster fundamental rights, and allow for human oversight...*" [95, p.15]

Human Dignity. Learning personalized model parameters for recommender systems allows to boost addiction or widespread emotional manipulation resulting in genocide [96–98]. KR1 rules out certain design choices for the labels of data points. In particular, we might not use the mental and psychological characteristics of a user as the label. We should avoid loss functions that can be used to train predictors of psychological characteristics.

Using personalized ML models to predict user preferences for products or susceptibility towards propaganda is also referred to as *micro-targeting* [99].

Simple is Good. Human oversight can be facilitated by relying on simple local models. Examples include linear models with few features or decision trees with a small tree depth. However, we are unaware of a widely accepted definition of when a model is simple. Loosely speaking, a simple model results in a learnt hypothesis that allows humans to understand how features of a data point relate to the prediction $h(\mathbf{x})$. This notion of simplicity is closely related to the concept of explainability which we discuss in more detail in Section 8.4.

Continuous Monitoring. In its simplest form, GTVMin-based methods involve a single training phase, i.e., learning local model parameters by solving GTVMin. However, this approach is only useful if the data can be well approximated by an i.i.d. assumption. In particular, this approach works only if the statistical properties of local datasets do not change over time. For many FL applications, this assumption is unrealistic (consider a social network which is exposed to constant change of memberships and user behaviour). It is then important to continuously compute a validation error on a timely validation set which is then used, in turn, to diagnose the overall FL system (see [6, Sec. 6.6]).

8.2.2 KR2 - Technical Robustness and Safety.

"*...Technical robustness requires that AI systems be developed with a preventative approach to risks and in a manner such that they reliably behave as intended while minimising unintentional and unexpected harm, and preventing*

unacceptable harm. ...' [95, p.16].

Practical FL systems are obtained by implementing FL algorithms in physical distributed computers [20, 21]. One example of a distributed computer is a collection of smartphones that are connected either by short-range wireless links or by a cellular network. Most distributed computers will incur imperfections, such as a temporary lack of connectivity or mobile devices becoming inactive due to running out of battery. Moreover, the data generation processes can be subject to perturbations such as statistical anomalies (outliers) or intentional modifications (see Chapter 10). Section 8.3 studies in some detail the robustness of GTVMin-based systems against different perturbations of data sources and imperfections of computational infrastructure.

8.2.3 KR3 - Privacy and Data Governance.

"*..privacy, a fundamental right particularly affected by AI systems. Prevention of harm to privacy also necessitates adequate data governance that covers the quality and integrity of the data used...*" [95, p.17].

We have introduced GTVMin and FL networks as abstract mathematical structures for the study of FL systems. However, to obtain actual FL systems we need to implement these mathematical concepts in a given physical hardware. These implementations incur deviations from the (idealized) GTVMin formulation (51) and the gradient-based methods (such as Algorithm 4) used to solve it. For example, using quantized label values results in a quantization error. Moreover, the local datasets can deviate significantly from a typical realization of i.i.d. RVs, which is referred to as statistical bias [100, Sec. 3.3.])

Data processing regulations limit the choice of the features of a data

point [101–103]. In particular, the general data protection regulation (GDPR) includes a data minimization principle which requires to use only features that are relevant for predicting the label.

Data Governance. Some FL applications involve local datasets that are generated by human users, i.e., personal data. Whenever personal data is used by a FL method, special care must be dedicated towards data protection regulations [103]. It is useful (or even compulsory) to designate a data protection officer and conduct a data protection impact assessment [95].

Privacy. The operation of a FL system must not violate the fundamental human right to privacy [104]. Chapter 9 discusses quantitative measures and methods for privacy protection in GTVMin-based FL systems.

8.2.4 KR4 - Transparency.

Traceability. This key requirement includes the documentation of design choices (and underlying business models) for a GTVMin-based FL system. This includes the source for the local datasets, the local models, the local loss function as well as the construction of the FL network. Moreover, the documentation should also cover the details of the implemented optimization method used to solve GTVMin. This documentation might also require the periodic storing of the model parameters along with a time stamp (*logging*).

Communication. Depending on the use case, FL systems need to communicate the capabilities and limitations to their end users (e.g., of a digital health app running on a smartphone). For example, we can indicate a measure of uncertainty about the predictions delivered by the trained local models. Such an uncertainty measure can be obtained naturally from a

probabilistic model for the data generation. For example, the conditional variance of the label y, given the features $\mathbf{x}$ of a random data point. Another example of an uncertainty measure is the validation error of a trained local model.

Explainability. The transparency of a GTVMin-based FL system also includes the explainability of the trained local models. Section 8.4 discusses quantitative measures for the subjective explainability of a learnt hypothesis. We will also use this measure as a regularizer to obtain GTVMin-based systems that guarantee subjective explainability "by design".

8.2.5 KR5 - Diversity, Non-Discrimination and Fairness.

"*...we must enable inclusion and diversity throughout the entire AI system's life cycle...this also entails ensuring equal access through inclusive design processes as well as equal treatment.*" [95, p.18].

The local datasets used for the training of local models should be carefully selected to not enforce existing discrimination. In a healthcare application, there might be significantly more training data for patients of a specific gender, resulting in models that perform best for that specific gender at the cost of worse performance for the minority [100, Sec. 3.3.].

Fairness is also important for ML methods used to determine credit score and, in turn, if a loan should be granted or not [105]. Here, we must ensure that ML methods do not discriminate against customers based on ethnicity or race. To this end, we could augment data points by modifying any features that mainly reflect the ethnicity or race of a customer (see Figure 8.1).

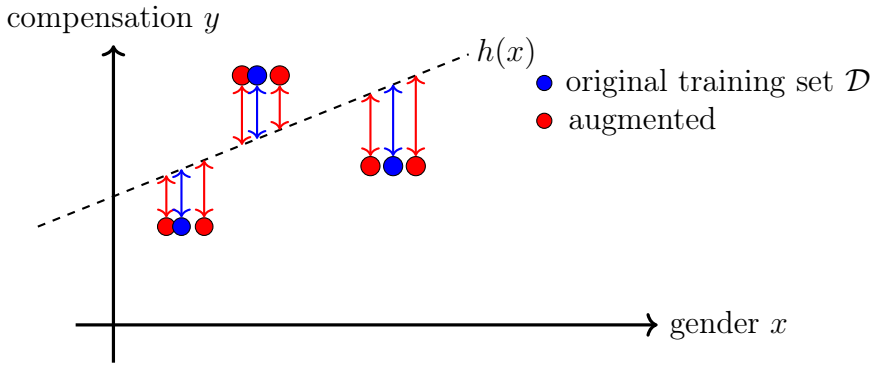


Figure 8.1: We can improve the fairness of a ML method by augmenting the training set using perturbations of an irrelevant feature such as the gender of a person for which we want to predict the adequate compensation as the label.

8.2.6 KR6 - Societal and Environmental Well-Being.

"*...Sustainability and ecological responsibility of AI systems should be encouraged, and research should be fostered into AI solutions addressing areas of global concern, such as for instance the Sustainable Development Goals.*" [95, p.19].

Society. FL systems might be used to deliver personalized recommendations to users within a social media application (social network). These recommendations might be (fake) news used to boost polarization and, in the extreme case, social unrest [106].

Environment. Chapter 5 discussed FL algorithms that were obtained by applying gradient-based methods to solve GTVMin. These methods require computational resources to compute local updates for model parameters

and to share them across the edges of the FL network. Computation and communication require energy which should be generated in an environmental-friendly fashion [107].

8.2.7 KR7 - Accountability.

"*...mechanisms be put in place to ensure responsibility and accountability for AI systems and their outcomes, both before and after their development, deployment and use.*" [95, p. 19].

8.3 Technical Robustness of FL Systems

Consider a GTVMin-based FL system that trains a single (global) linear model in a distributed fashion from a collection of local datasets $\mathcal{D}^{(i)}$, for $i = 1, \ldots, n$. As discussed in Section 6.2, this single-model FL setting amounts to using GTVMin (53) over a connected FL network with a sufficiently large choice of α.

To ensure **KR2** we need to understand the effect of perturbations on a GTVMin-based FL system. These perturbations might be intentional or non-intentional and affect the local datasets used to evaluate the loss of local model parameters or the computational infrastructure used to implement a GTVMin-based method (see Chapter 5). We next explain how to use some of the theoretic tools from previous chapters to quantify the robustness of GTVMin-based FL systems.

8.3.1 Sensitivity Analysis

As pointed out in Chapter 3, GTVMin (53) can be rewritten as the minimization of a quadratic function,

$$\min_{\mathbf{w}=\text{stack}\{\mathbf{w}^{(i)}\}} \mathbf{w}^T\mathbf{Q}\mathbf{w} + \mathbf{q}^T\mathbf{w}. \tag{171}$$

The matrix $\mathbf{Q}$ and vector $\mathbf{q}$ are determined by the feature matrices $\mathbf{X}^{(i)}$ and label vector $\mathbf{y}^{(i)}$ at nodes $i \in \mathcal{V}$ (see (28)). We next study the sensitivity of (the solutions of) (171) towards external perturbations of the label vector.[26]

Consider an additive perturbation $\widetilde{\mathbf{y}}^{(i)} := \mathbf{y}^{(i)} + \boldsymbol{\varepsilon}^{(i)}$ of the label vector $\mathbf{y}^{(i)}$. Using the perturbed label vector $\widetilde{\mathbf{y}}^{(i)}$ results also in a "perturbation" of GTVMin (171),

$$\min_{\mathbf{w}=\text{stack}\{\mathbf{w}^{(i)}\}} \mathbf{w}^T\mathbf{Q}\mathbf{w} + \mathbf{q}^T\mathbf{w} + \mathbf{n}^T\mathbf{w} + c. \tag{172}$$

An inspection of (28) yields that $\mathbf{n} = \Big(\big(\boldsymbol{\varepsilon}^{(1)}\big)^T\mathbf{X}^{(1)}, \ldots, \big(\boldsymbol{\varepsilon}^{(n)}\big)^T\mathbf{X}^{(n)}\Big)^T$. The next result provides an upper bound on the deviation between the solutions of (171) and (172).

Proposition 8.1. *Consider the GTVMin instance* (171) *for learning local model parameters of a linear model for each node $i \in \mathcal{V}$ of an FL network $\mathcal{G}$. We assume that the FL network is connected, i.e., $\lambda_2\big(\mathbf{L}^{(\mathcal{G})}\big) > 0$ and the local datasets are such that $\bar{\lambda}_{\min} > 0$ (see* (114)*). Then, the deviation between the solution $\widehat{\mathbf{w}}^{(i)}$ to* (171) *and the solution $\widetilde{\mathbf{w}}^{(i)}$ to the perturbed problem* (172) *is upper bounded as*

$$\sum_{i=1}^{n} \big\|\widehat{\mathbf{w}}^{(i)} - \widetilde{\mathbf{w}}^{(i)}\big\|_2^2 \leq \frac{\lambda_{\max}(1+\rho^2)^2}{\big[\min\{\lambda_2\big(\mathbf{L}^{(\mathcal{G})}\big)\alpha\rho^2, \bar{\lambda}_{\min}/2\}\big]^2} \sum_{i=1}^{n} \big\|\boldsymbol{\varepsilon}^{(n)}\big\|_2^2. \tag{173}$$

[26] Our study can be generalized to also take into account perturbations of the feature matrices $\mathbf{X}^{(i)}$, for $i = 1, \ldots, n$.

Here, we used the shorthand $\rho := \bar{\lambda}_{\min}/(4\lambda_{\max})$ *(see* (114)*).*

Proof. The assumptions of Prop. 8.1 allow to apply the lower bound (**??**) on the eigenvalues of the matrix $\mathbf{Q}$ in (171). □

8.3.2 Estimation Error Analysis

Prop. 8.1 characterizes the sensitivity of GTVMin solutions against *external* perturbations of the local datasets. While this notion of robustness is important, it might not suffice for a comprehensive assessment of a FL system. For example, we can trivially achieve perfect robustness (in the sense of minimum sensitivity) by delivering constant model parameters, e.g., $\widehat{\mathbf{w}}^{(i)} = \mathbf{0}$.

Another form of robustness is to ensure a small estimation error of (53). To study this form of robustness, we use a variant of the probabilistic model (60): We assume that the labels and features of data points of each local dataset $\mathcal{D}^{(i)}$, for $i = 1, \ldots, n$, are related via

$$\mathbf{y}^{(i)} = \mathbf{X}^{(i)}\overline{\mathbf{w}} + \boldsymbol{\varepsilon}^{(i)}. \tag{174}$$

In contrast to Sec. 3.4.2, we assume that all components of (174) are deterministic. In particular, the noise term $\boldsymbol{\varepsilon}^{(i)}$ is a deterministic but unknown quantity. This term accommodates any perturbation that might arise from technical imperfections or intrinsic label noise due to random fluctuations in the labelling process.[27]

In the ideal case of no perturbation, we would have $\boldsymbol{\varepsilon}^{(i)} = \mathbf{0}$. However, in general might only know some upper bound measure for the size of the

[27] Consider labels obtained from physical sensing devices which are typically subject to uncertainties [108].

perturbation, e.g., $\left\|\boldsymbol{\varepsilon}^{(i)}\right\|_2^2$. We next present upper bounds on the estimation error $\widehat{\mathbf{w}}^{(i)} - \overline{\mathbf{w}}$ incurred by the GTVMin solutions $\widehat{\mathbf{w}}^{(i)}$.

This estimation error consists of two components, the first component being $\mathrm{avg}\big\{\widehat{\mathbf{w}}^{(i')}\big\} - \overline{\mathbf{w}}$ for each node $i \in \mathcal{V}$. Note that this error component is identical for all nodes $i \in \mathcal{V}$. The second component of the estimation error is the deviation $\widetilde{\mathbf{w}}^{(i)} := \widehat{\mathbf{w}}^{(i)} - \mathrm{avg}\big\{\widehat{\mathbf{w}}^{(i')}\big\}$ of the learnt local model parameters $\widehat{\mathbf{w}}^{(i')}$, for $i' = 1, \ldots, n$, from their average $\mathrm{avg}\big\{\widehat{\mathbf{w}}^{(i')}\big\} = (1/n)\sum_{i'=1}^{n} \widehat{\mathbf{w}}^{(i')}$. As discussed in Section 3.4.2, these two components correspond to two orthogonal subspaces of $\mathbb{R}^{d \cdot n}$.

According to Prop. 3.1, the second error component is upper bounded as

$$\sum_{i=1}^{n} \left\|\widetilde{\mathbf{w}}^{(i)}\right\|_2^2 \leq \frac{1}{\lambda_2 \alpha} \sum_{i=1}^{n} (1/m_i) \left\|\boldsymbol{\varepsilon}^{(i)}\right\|_2^2. \tag{175}$$

To bound the first error component $\bar{\mathbf{c}} - \overline{\mathbf{w}}$, using the shorthand $\bar{\mathbf{c}} := \mathrm{avg}\big\{\widehat{\mathbf{w}}^{(i)}\big\}$, we first note that (see (53))

$$\bar{\mathbf{c}} = \operatorname*{argmin}_{\mathbf{w} \in \mathbb{R}^d} \sum_{i \in \mathcal{V}} (1/m_i) \left\|\mathbf{y}^{(i)} - \mathbf{X}^{(i)}\big(\mathbf{w} - \widetilde{\mathbf{w}}^{(i)}\big)\right\|_2^2 + \alpha \sum_{\{i,i'\} \in \mathcal{E}} A_{i,i'} \left\|\widetilde{\mathbf{w}}^{(i)} - \widetilde{\mathbf{w}}^{(i')}\right\|_2^2. \tag{176}$$

Using a similar argument as in the proof for Prop. 2.1, we obtain

$$\|\bar{\mathbf{c}} - \overline{\mathbf{w}}\|_2^2 \leq \left\|\sum_{i=1}^{n} (1/m_i)\big(\mathbf{X}^{(i)}\big)^T\big(\boldsymbol{\varepsilon}^{(i)} + \mathbf{X}^{(i)}\widetilde{\mathbf{w}}^{(i)}\big)\right\|_2^2 / (n\bar{\lambda}_{\min})^2. \tag{177}$$

Here, $\bar{\lambda}_{\min}$ is the smallest eigenvalue of $(1/n)\sum_{i=1}^{n} \mathbf{Q}^{(i)}$, i.e., the average of the matrices $\mathbf{Q}^{(i)} = (1/m_i)\big(\mathbf{X}^{(i)}\big)^T\mathbf{X}^{(i)}$ over all nodes $i \in \mathcal{V}$.[28] Note that the bound (177) is only valid if $\bar{\lambda}_{\min} > 0$ which, in turn, implies that the solution to (176) is unique.

[28] We encountered the quantity $\bar{\lambda}_{\min}$ already during our discussion of gradient-based methods for solving the GTVMin instance (53) (see (114)).

We can develop (177) further using

$$\begin{aligned}
&\left\|\sum_{i=1}^{n}(1/m_i)\big(\mathbf{X}^{(i)}\big)^T\big(\boldsymbol{\varepsilon}^{(i)}+\mathbf{X}^{(i)}\widetilde{\mathbf{w}}^{(i)}\big)\right\|_2 \\
&\overset{(a)}{\leq} \sum_{i=1}^{n}\left\|(1/m_i)\big(\mathbf{X}^{(i)}\big)^T\big(\boldsymbol{\varepsilon}^{(i)}+\mathbf{X}^{(i)}\widetilde{\mathbf{w}}^{(i)}\big)\right\|_2 \\
&\overset{(b)}{\leq} \sqrt{n}\sqrt{\sum_{i=1}^{n}\left\|(1/m_i)\big(\mathbf{X}^{(i)}\big)^T\big(\boldsymbol{\varepsilon}^{(i)}+\mathbf{X}^{(i)}\widetilde{\mathbf{w}}^{(i)}\big)\right\|_2^2} \\
&\overset{(c)}{\leq} \sqrt{n}\sqrt{\sum_{i=1}^{n}2\left\|(1/m_i)\big(\mathbf{X}^{(i)}\big)^T\boldsymbol{\varepsilon}^{(i)}\right\|_2^2+2\left\|(1/m_i)\big(\mathbf{X}^{(i)}\big)^T\mathbf{X}^{(i)}\widetilde{\mathbf{w}}^{(i)}\right\|_2^2} \\
&\overset{(d)}{\leq} \sqrt{n}\sqrt{\sum_{i=1}^{n}(2/m_i)\lambda_{\max}\left\|\boldsymbol{\varepsilon}^{(i)}\right\|_2^2+2\lambda_{\max}^2\left\|\widetilde{\mathbf{w}}^{(i)}\right\|_2^2}. \qquad (178)
\end{aligned}$$

Here, step (a) uses the triangle inequality of norms, step (b) uses the Cauchy-Schwarz inequality, step (c) uses the inequality $\|\mathbf{a}+\mathbf{b}\|_2^2 \leq 2\Big(\|\mathbf{a}\|_2^2+\|\mathbf{b}\|_2^2\Big)$, and step (d) uses the maximum eigenvalue $\lambda_{\max} := \max_{i\in\mathcal{V}}\lambda_d\big(\mathbf{Q}^{(i)}\big)$ of the matrices $\mathbf{Q}^{(i)} = (1/m_i)\big(\mathbf{X}^{(i)}\big)^T\mathbf{X}^{(i)}$ (see (114)).

Inserting (178) into (177) results in the upper bound

$$\begin{aligned}
\|\bar{\mathbf{c}}-\overline{\mathbf{w}}\|_2^2 &\leq 2\sum_{i=1}^{n}\left[(1/m_i)\lambda_{\max}\left\|\boldsymbol{\varepsilon}^{(i)}\right\|_2^2+\lambda_{\max}^2\left\|\widetilde{\mathbf{w}}^{(i)}\right\|_2^2\right]/(n\bar{\lambda}_{\min}^2) \\
&\overset{(175)}{\leq} 2\big(\lambda_{\max}+(\lambda_{\max}^2/(\lambda_2\alpha))\big)\sum_{i=1}^{n}(1/m_i)\left\|\boldsymbol{\varepsilon}^{(i)}\right\|_2^2/(n\bar{\lambda}_{\min}^2). \qquad (179)
\end{aligned}$$

The upper bound (179) on the estimation error of GTVMin-based methods depends on both, the FL network $\mathcal{G}$ via the eigenvalue λ_2 of $\mathbf{L}^{(\mathcal{G})}$, and the feature matrices $\mathbf{X}^{(i)}$ of the local datasets (via the quantities $\lambda_{\max}$ and $\bar{\lambda}_{\min}$ as defined in (114)). Let us next discuss how the upper bound (179) might guide the choice of the FL network $\mathcal{G}$ and the features of data points in the

local datasets.

According to (179), we should use an FL network $\mathcal{G}$ with large $\lambda_2\big(\mathbf{L}^{(\mathcal{G})}\big)$ to ensure a small estimation error for GTVMin-based methods. Note that we came across the same design criterion already when discussing graph learning methods in Chapter 7. In particular, using an FL network with large $\lambda_2\big(\mathbf{L}^{(\mathcal{G})}\big)$ also tends to speed up the convergence of gradient-based methods for solving GTVMin (such as Algorithm 4).

The upper bound (179) suggests using features that result in a small ratio $\lambda_{\max}/\bar{\lambda}_{\min}$ between the quantities $\lambda_{\max}$ and $\bar{\lambda}_{\min}$ (see (114)). Some feature learning methods have been proposed in order to minimize this ratio [6, 109].

8.3.3 Network Resilience

The previous sections studied the robustness of GTVMin-based methods against perturbations of local datasets (see Exercise 8.1) and in terms of ensuring a small estimation error (see (179)). We also need to ensure that FL systems are robust against imperfections of the computational infrastructure used to solve GTVMin. These imperfections include hardware failures, running out of battery or lack of wireless connectivity.

Chapter 5 showed how to design FL algorithms by applying gradient-based methods to solve GTVMin (53). We obtain practical FL systems by implementing these algorithms, such as Algorithm 4, in a particular computational infrastructure. Two important examples of such an infrastructure are mobile networks and wireless sensor networks [22, 110].

The effect of imperfections in the implementation of the GD based Algorithm 4 can be modelled as perturbed GD (90) from Chapter 4. We can

then analyze the robustness of the resulting FL system via the convergence analysis of perturbed GD (see Section 4.5).

According to (91), the performance of the decentralized Algorithm 4 degrades gracefully in the presence of imperfections such as missing or faulty communication links. In contrast, the server-based implementation of FedAvg Algorithm 9 offers a single point of failure (the server).

8.3.4 Stragglers

Using perturbed GD to model imperfections in the actual implementation of Algorithm 4 is quite flexible. The flexibility in allowing for a wide range of imperfections might come at the cost of a coarse-grained analysis. In particular, the upper bound (91) can be too loose (pessimistic) to be of any use. We then need to take into account the specific nature of the imperfection and the resulting perturbation $\boldsymbol{\varepsilon}^{(i)}$. Let us next focus on a particular type of imperfection that arises from the asynchronous implementation of Algorithm 4 at different nodes $i \in \mathcal{V}$.

Note that Algorithm 4 requires the nodes to operate synchronously: during each iteration, the nodes need to exchange their current model parameters $\mathbf{w}^{(i,k)}$ simultaneously with their neighbours in the FL network. This requires, in turn, that the update in step 4 of Algorithm 4 is completed at every node i before the global clock ticks (triggering the next iteration).

Some of the nodes i might have limited computational resources and therefore require much longer for the update step 4 of Algorithm 4. The literature refers to such slower nodes sometimes as stragglers [111]. Instead of forcing the other (faster) nodes to wait until also the slower ones are ready,

we could instead let them continue with their local updates. This results in an asynchronous variant of Algorithm 4 which we summarize in Algorithm 13.

Note that, much like the synchronous Algorithm 4, also the asynchronous Algorithm 13 uses an iteration counter k. However, the practical meaning of k in the asynchronous variant is fundamentally different from the synchronous variant. Instead of indexing a global iteration that corresponds to the synchronous execution of local updates at all nodes $\mathcal{V}$, the counter k now denotes an event during which some nodes are active, i.e., compute local updates. We denote the set of active nodes (or devices) during event k by $\mathcal{A}^{(k)} \subseteq \mathcal{V}$. The remaining *inactive* nodes $i \notin \mathcal{A}^{(k)}$ leave their current model parameters unchanged, $\mathbf{w}^{(i,k+1)} := \mathbf{w}^{(i,k)}$.

For each *active* node $i \in \mathcal{A}^{(k)}$, the local update (180) uses potentially *outdated* model parameters $\mathbf{w}^{(i',k_{i,i'})}$ from its neighbours $i' \in \mathcal{N}^{(i)}$. Indeed, some of the neighbours might have not been in the active sets $\mathcal{A}^{(k-1)}, \mathcal{A}^{(k-2)}, \ldots$ of the most recent iterations. In this case, the update (180) does not have access to $\mathbf{w}^{(i',k)}$ at *time instant* k. Instead, we can only use $\mathbf{w}^{(i',k_{i,i'})}$ that has been produced obtained during some previous iteration $k_{i,i'} < k$.

We can interpret the quantity $k_{i,i'}$ as the most recent time instant during which node i' has shared its updated local model parameters with node i. We can interpret the difference $k - k_{i,i'}$ as a measure for the communication delay from node i' to node i. The robustness of (the convergence of) Algorithm 13 against these communication delays is studied in-depth in [20, Ch. 6 and 7].

Algorithm 13 Asynchronous FedGD for Local Linear Models

Input: FL network $\mathcal{G}$; GTV parameter α; learning rate η

local dataset $\mathcal{D}^{(i)} = \left\{\left(\mathbf{x}^{(i,1)}, y^{(i,1)}\right), \ldots, \left(\mathbf{x}^{(i,m_i)}, y^{(i,m_i)}\right)\right\}$ for each i; some stopping criterion.

Output. linear model parameters $\widehat{\mathbf{w}}^{(i)}$ for each node $i \in \mathcal{V}$

Initialize. $k{:=}0$; $\mathbf{w}^{(i,0)}{:=}\mathbf{0}$ for all nodes $i \in \mathcal{V}$.

1: **while** stopping criterion is not satisfied **do**

2: **for** all active nodes $i \in \mathcal{A}^{(k)}$: **do**

3: update local model parameters via

$$\mathbf{w}^{(i,k+1)}{:=}\mathbf{w}^{(i,k)} + \eta\Bigg[(2/m_i)\big(\mathbf{X}^{(i)}\big)^T\big(\mathbf{y}^{(i)} - \mathbf{X}^{(i)}\mathbf{w}^{(i,k)}\big) + 2\alpha\sum_{i' \in \mathcal{N}^{(i)}} A_{i,i'}\big(\mathbf{w}^{(i',k_{i,i'})} - \mathbf{w}^{(i,k)}\big)\Bigg]. \tag{180}$$

4: share local model parameters $\mathbf{w}^{(i,k+1)}$ with neighbors $i' \in \mathcal{N}^{(i)}$

5: **end for**

6: **for** all inactive nodes $i \notin \mathcal{A}^{(k)}$: **do**

7: keep model parameters unchanged $\mathbf{w}^{(i,k+1)} := \mathbf{w}^{(i,k)}$

8: **end for**

9: increment iteration counter: $k{:=}k{+}1$

10: **end while**

11: $\widehat{\mathbf{w}}^{(i)} := \mathbf{w}^{(i,k)}$ for all nodes $i \in \mathcal{V}$

8.4 Subjective Explainability of FL Systems

Let us now discuss how to ensure key requirement **KR4 - Transparency** in GTVMin-based FL systems. This key requirement includes also the explainability of a trained personalized model $\hat{h}^{(i)} \in \mathcal{H}^{(i)}$ (and their predictions). It is important to note that the explainability of $\hat{h}^{(i)}$ is subjective: A given learnt hypothesis $\hat{h}^{(i)}$ might offer a high degree of explainability to one user (a graduate student at a university) but a low degree of explainability to another user (a high-school student). We must ensure explainability or the trained models $\hat{h}^{(i)}$ for potentially different users of the devices $i = 1, \ldots, n$.

The explainability of trained ML models is closely related to its simulatability [112–114]: How well can a user anticipate (or guess) the prediction $\hat{y} = \hat{h}^{(i)}(\mathbf{x})$ delivered by $\hat{h}^{(i)}$ for a data point with features $\mathbf{x}$. We can then measure the explainability of $\hat{h}^{(i)}(\mathbf{x})$ to the user at node i by comparing the prediction $\hat{h}^{(i)}(\mathbf{x})$ with the corresponding *guess* (or *simulation*) $u^{(i)}(\mathbf{x})$.

We can enforce (subjective) explainability of FL systems by modifying the local loss functions in GTVMin. For ease of exposition, we focus on the GTVMin instance (110) for training local (personalized) linear models. For each node $i \in \mathcal{V}$, we construct a test-set $\mathcal{D}_t^{(i)}$ and ask user i to deliver a guess $u^{(i)}(\mathbf{x})$ for each data point in $\mathcal{D}_t^{(i)}$.[29]

We measure the (subjective) explainability of a linear hypothesis with model parameters $\mathbf{w}^{(i)}$ by

$$(1/|\mathcal{D}_t^{(i)}|) \sum_{\mathbf{x} \in \mathcal{D}_t^{(i)}} \left(u^{(i)}(\mathbf{x}) - \mathbf{x}^T \mathbf{w}^{(i)} \right)^2. \tag{181}$$

[29] We only use the features of the data points in $\mathcal{D}_t^{(i)}$, i.e., this dataset can be constructed from unlabeled data.

It seems natural to add this measure as a penalty term to the local loss function in (110), resulting in the new loss function

$$L_i\left(\mathbf{w}^{(i)}\right) := \underbrace{(1/m_i)\left\|\mathbf{y}^{(i)}-\mathbf{X}^{(i)}\mathbf{w}^{(i)}\right\|_2^2}_{\text{training error}} + \underbrace{\rho\,(1/\left|\mathcal{D}_t^{(i)}\right|)\sum_{\mathbf{x}\in\mathcal{D}_t^{(i)}}\left(u^{(i)}\left(\mathbf{x}\right)-\mathbf{x}^T\mathbf{w}^{(i)}\right)^2}_{\text{subjective explainability}}. \tag{182}$$

The regularization parameter ρ controls the preference for a high subjective explainability of the hypothesis $h^{(i)}(\mathbf{x}) = \left(\mathbf{w}^{(i)}\right)^T\mathbf{x}$ over a small training error [114]. It can be shown that (182) is the average weighted squared error loss of $h^{(i)}(\mathbf{x})$ on an augmented version of $\mathcal{D}^{(i)}$. This augmented version includes the data point $\left(\mathbf{x}, u^{(i)}(\mathbf{x})\right)$ for each data point $\mathbf{x}$ in the test-set $\mathcal{D}_t^{(i)}$.

So far, we have focused on the problem of explaining (the predictions of) a trained personalized model to some user. The general idea is to provide partial information, in the form of some explanation, about the learnt hypothesis map $\hat{h}$. Explanations should help the user to anticipate the prediction $\hat{h}(\mathbf{x})$ for any given data point. Instead of explaining a given trained model $\hat{h}$, it might be more useful to explain an entire FL algorithm.

Mathematically, we can interpret an FL algorithm as a map $\mathcal{A}$ that reads in local datasets and delivers learnt hypothesis maps $\hat{h}^{(i)}$. Explaining an FL algorithm amounts to providing partial information about this map $\mathcal{A}$. Thus, mathematically speaking, the problem of explaining a learnt hypothesis is essentially the same as explaining an entire FL algorithm: Provide partial information (an explanation) about a map such that the user can anticipate the results of applying the map to arbitrary arguments. However, the map $\mathcal{A}$ might be much more complicated than a learnt hypothesis map (which could be a linear map for linear models).

The different complexity levels of maps to be explained requires different forms of explanation. For example, we might explain a FL algorithm using a pseudo-code such as Algorithm 4. Another form of explanation could be a Python code snippet that illustrates a potential implementation of the algorithm (see Fig. 8.2).

```python
from sklearn.datasets import load_iris
from sklearn.model_selection import train_test_split
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import accuracy_score

# Load the Iris dataset
data = load_iris()
X = data.data
y = data.target

# Split the dataset into training and test sets
X_train, X_test, y_train, y_test = train_test_split(X, y,
    test_size=0.3, random_state=42)

# Create a Decision Tree classifier
clf = DecisionTreeClassifier(random_state=42)

# Train the classifier
clf.fit(X_train, y_train)

# Make predictions on the test data
y_pred = clf.predict(X_test)

# Calculate accuracy
accuracy = accuracy_score(y_test, y_pred)
accuracy
```

Figure 8.2: Python code for fitting a decision tree model on the Iris dataset.

8.5 Exercises

8.1. Robustness of GTVMin. Discuss the robustness of GTVMin (53) for training local linear models. In particular, which attack is more effective (detrimental): perturbing the labels, the features of data points in the local datasets or perturbing the FL network, e.g., by removing or adding edges.

8.2. Subjectively Explainable FL. Consider GTVMin (53) to train local linear models with model parameters $\mathbf{w}^{(i)}$. The local datasets are modelled as (60). Each local model has a user that is characterized by the user signal $u(\mathbf{x}) := \mathbf{x}^T \mathbf{u}^{(i)}$. To ensure subjective explainability of local model with model parameters $\mathbf{w}^{(i)}$ we require the deviation $(1/m_i) \left\| \widetilde{\mathbf{X}}^{(i)} \big(\mathbf{w}^{(i)} - \mathbf{u}^{(i)} \big) \right\|_2^2$ to be sufficiently small. Here, we used the feature matrix $\widetilde{\mathbf{X}}^{(i)}$ obtained from the realization of m_i i.i.d. RVs with common probability distribution $\mathcal{N}(\mathbf{0}, \mathbf{I})$. We then add this deviation to the local loss functions resulting in using the augmented loss function (182) used in (53). Study, either analytically or by numerical experiments, the effect of varying levels of explainability (via the parameter ρ in (182)) on the estimation error $\widehat{\mathbf{w}}^{(i)} - \overline{\mathbf{w}}^{(i)}$.

9 Privacy Protection in FL

The core idea of FL is to share information contained in collections of local datasets to improve the training of (personalized) ML models. Chapter 5 discussed FL algorithms that share information in the form of model parameters that are computed from the local loss function. Each node $i \in \mathcal{V}$ receives the current model parameters of other nodes and, after executing a local update, shares its new model parameters with other nodes.

Depending on the design choices for GTVMin-based methods, sharing model parameters allows to reconstruct local loss functions and, in turn, to estimate private information about individual data points such as healthcare customers (patients) [115]. Thus, the bad news is that FL systems will almost inevitably incur some leakage of private information. The good news is, however, that the extent of privacy leakage can be controlled by (i) careful design choices for GTVMin and (ii) applying slight modifications of basic FL algorithms (such as those from Chapter 5).

This chapter revolves around two main questions:

- How can we measure the privacy leakage of a FL system?
- How can we control (minimize) privacy leakage of a FL system?

Section 9.2 addresses the first question while Sections 9.3 and 9.4 address the second question.

9.1 Learning Goals

After completing this chapter, you will

- be aware of threats to privacy in FL and the need to protect it,
- know some quantitative measures for privacy leakage,
- understand how GTVMin can facilitate privacy protection,
- be able to implement FL algorithms with guaranteed levels of privacy protection.

9.2 Measuring Privacy Leakage

Consider a FL system that trains a personalized model for the users, indexed by $i = 1, \ldots, n$, of heart rate sensors. Each user i generates a local dataset $\mathcal{D}^{(i)}$ that consists of time-stamped heart rate measurements. We define a single data point as a single continuous activity, e.g. as a 50-minute long run. The features of such a data point (activity) might include the trajectory in the form of a time series of GPS coordinates (e.g., measured every 30 seconds). The label of a data point (activity) could be the average heart rate during the activity. Let us assume that this average heart rate is private information that should not be shared with anybody.[30]

Our FL system also exploits the information provided by a fitness expert that determines pair-wise similarities $A_{i,i'}$ between users i, i' (e.g., due to body weight and height). We then use some FL algorithm (e.g., Algorithm 4) to learn, for each user i, the model parameters $\mathbf{w}^{(i)}$ of an AI healthcare assistant [116]. We model this algorithm as a map $\mathcal{A}(\cdot)$ (see Figure 9.1) that reads in the

[30] In particular, we might not want to share our heart rate profiles with a potential future employer who prefers candidates with a long life expectation.

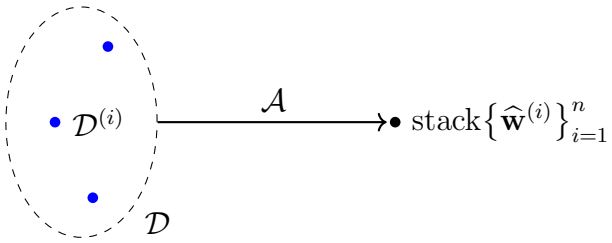


Figure 9.1: A FL algorithm maps the local datasets $\mathcal{D}^{(i)}$ to the learnt model parameters $\widehat{\mathbf{w}}^{(i)}$, for $i = 1, \ldots, n$. These learnt model parameters are (approximate) solutions to the GTVMin instance (53).

dataset $\mathcal{D} := \big\{\mathcal{D}^{(i)}\big\}_{i=1}^{n}$ (constituted by the local datasets $\mathcal{D}^{(i)}$ for $i = 1, \ldots, n$) and delivers the learnt local model parameters $\mathcal{A}\big(\mathcal{D}\big) := \underbrace{\text{stack}\big\{\widehat{\mathbf{w}}^{(i)}\big\}_{i=1}^{n}}_{\widehat{\mathbf{w}}}$.

A privacy-preserving FL system should not allow to infer, solely from the learnt model parameters, the average heart rate $y^{(i,r)}$ during a specific single activity r of a specific user i. Mathematically, we must ensure that the map $\mathcal{A}$ is not invertible: The learnt model parameters should not change if we were to apply the FL algorithm to a perturbed dataset that includes a different value for the average heart rate $y^{(i,r)}$. Figure 9.2 illustrates an algorithm that is partially invertible in the sense of allowing to infer the label of some data points used in the training set.

The sole requirement for a FL algorithm $\mathcal{A}$ to be not invertible is not useful in general. Indeed, we can easily make any algorithm $\mathcal{A}$ by simple pre- or post-processing techniques whose effect is limited to irrelevant regions of the input space (which is the space of all possible datasets). The level of privacy protection offered by $\mathcal{A}$ can be characterized by a measure of its *non-invertibility*.

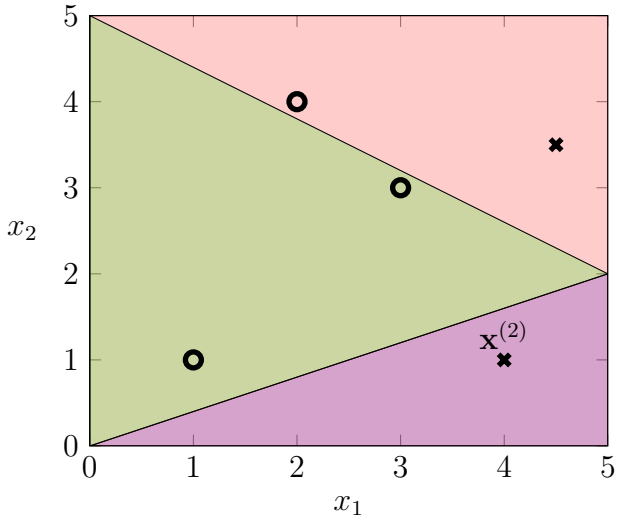


Figure 9.2: Scatterplot of a training set consisting of data points, each characterized by features $\mathbf{x}^{(r)} = \left(x_1^{(r)}, x_2^{(r)}\right)^2$ and a binary label $y^{(r)} \in \{\circ, \times\}$. The plot also indicates the decision regions of a hypothesis $\widehat{h}$ that has been learnt via ERM. Would you be able to infer the label of data point $\mathbf{x}^{(2)}$ if you knew the decision regions?

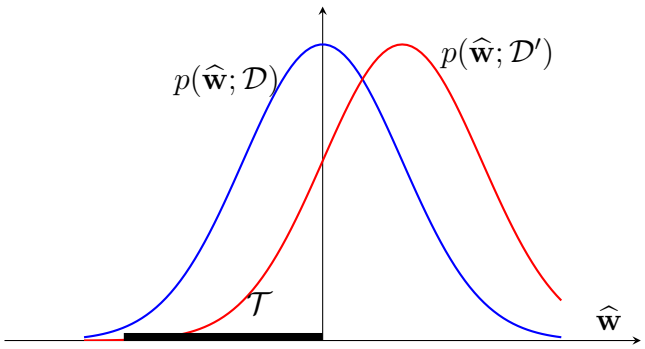


Figure 9.3: Two probability distributions of the learnt model parameters $\widehat{\mathbf{w}} = \text{stack}\big\{\widehat{\mathbf{w}}^{(i)}\big\}_{i=1}^{n}$ delivered by some FL algorithm (such as Algorithm 4). These two probability distributions correspond to two different choices for the input dataset, denoted by $\mathcal{D}'$ and $\mathcal{D}$. For example $\mathcal{D}'$ might be obtained from $\mathcal{D}$ by changing the value of a private feature of some data point in $\mathcal{D}$. We also indicate an "acceptance region" $\mathcal{T}$ that is used to detect if $\mathcal{D}$ (or a neighbouring dataset $\mathcal{D}'$) has been fed into the algorithm.

A simple measure of non-invertibility is the sensitivity of the output $\mathcal{A}(\mathcal{D})$ against varying the heart rate value $y^{(i,r)}$,

$$\frac{\left\|\mathcal{A}(\mathcal{D}) - \mathcal{A}(\mathcal{D}')\right\|_2}{\varepsilon}. \tag{183}$$

Here, $\mathcal{D}$ denotes some given collection of local datasets and $\mathcal{D}'$ is a modified dataset. In particular, $\mathcal{D}'$ is obtained by replacing the actual average heart rate $y^{(i,r)}$ with the modified value $y^{(i,r)} + \varepsilon$. The privacy protection of $\mathcal{A}$ is higher for smaller values (183), i.e., the output changes only a little when varying the value of the average heart rate.

Another measure for the non-invertibility of $\mathcal{A}$ is referred to as DP. This measure is particularly useful for stochastic algorithms that use some random mechanism. One example of such a random mechanism is the selection of a random subset of data points (a batch) within Algorithm 4. Section 9.3 discusses another example of a random mechanism: add the realization of a RV to the (intermediate) results of an algorithm.

A stochastic algorithm $\mathcal{A}$ can be described by a probability distribution $p(\widehat{\mathbf{w}}; \mathcal{D})$ over a measurable space that is constituted by the possible values of the learnt model parameters $\widehat{\mathbf{w}}$ (see Figure 9.3).[31] This probability distribution is parametrized by the dataset $\mathcal{D}$ that is fed as input to the algorithm $\mathcal{A}$.

DP measures the non-invertibility of a stochastic algorithm $\mathcal{A}$ via the similarity of the probability distributions obtained for two datasets $\mathcal{D}, \mathcal{D}'$ that are considered neighbouring or adjacent [100, 120]. Typically, we consider $\mathcal{D}'$ as adjacent to $\mathcal{D}$ if it is obtained by modifying the features or label of a single data point in $\mathcal{D}$. As a case in point, consider data points representing

[31] For more details about the concept of a measurable space, we refer to the literature on probability and measure theory [117–119].

physical activities which are characterized by a binary feature $x_j \in \{0,1\}$ that indicates an excessively high average heart rate during the activity. We could then define neighbouring datasets by flipping the feature x_j of a single data point. In general, the notion of neighbouring datasets is a design choice used in the formal definition of DP.

Definition 1. *(from [120]) A stochastic algorithm $\mathcal{A}$ is (ε,δ)-DP if for any measurable set $\mathcal{S}$ and any two neighbouring datasets $\mathcal{D},\mathcal{D}'$*

$$\text{Prob}\big\{\mathcal{A}(\mathcal{D}) \in \mathcal{S}\big\} \leq \exp(\varepsilon)\text{Prob}\big\{\mathcal{A}(\mathcal{D}') \in \mathcal{S}\big\} + \delta. \tag{184}$$

It appears that Definition 1 is the current de facto standard for measuring the (lack of) privacy protection in FL systems [100, 120]. Nevertheless, there are also other measures for the similarity between probability distributions $p(\widehat{\mathbf{w}};\mathcal{D})$ and $p(\widehat{\mathbf{w}};\mathcal{D}')$ that might be more useful for practical or theoretical reasons [121]. One such alternative measure is the Rényi divergence of order $\alpha > 1$,

$$D_\alpha\Big(p(\widehat{\mathbf{w}};\mathcal{D})\big\|p(\widehat{\mathbf{w}};\mathcal{D}')\Big) := \frac{1}{\alpha-1}\mathbb{E}_{p(\widehat{\mathbf{w}};\mathcal{D}')}\left[\left(\frac{dp(\widehat{\mathbf{w}};\mathcal{D})}{dp(\widehat{\mathbf{w}};\mathcal{D}')}\right)^\alpha\right]. \tag{185}$$

The Rényi divergence allows to define the following variant of DP [121, 122].

Definition 2. *(from [120]) A stochastic algorithm $\mathcal{A}$ is (α,γ)-RDP if, for any two neighbouring datasets $\mathcal{D},\mathcal{D}'$,*

$$D_\alpha\Big(p(\widehat{\mathbf{w}};\mathcal{D})\big\|p(\widehat{\mathbf{w}};\mathcal{D}')\Big) \leq \gamma. \tag{186}$$

A recent use-case of (α,γ)-RDP is the analysis of DP guarantees offered by variants of SGD [121]. This analysis uses the fact that (α,γ)-RDP implies (ε,δ)-DP for suitable choices of ε,δ [121].

One important property of the DP notions in Definition 1 and Definition 2 is that they are preserved by post-processing:

Proposition 9.1. *Consider a FL system $\mathcal{A}$ that is applied to some dataset $\mathcal{D}$ and some (possibly stochastic) map $\mathcal{B}$ that does not depend on $\mathcal{D}$. If $\mathcal{A}$ is (ε, δ)-DP (or (α, γ)-RDP), then so is also the composition $\mathcal{B} \circ \mathcal{A}$.*

Proof. See, e.g., [120, Sec. 2.3]. □

By Prop. (9.1), we cannot reduce the (degree of) DP of $\mathcal{A}$ by any post-processing method $\mathcal{B}$ that has no access to the raw data itself. It seems almost natural to make this immunity against post-processing a defining property of any useful notion of DP [122]. However, due to Prop. (9.1), this property is already "built-in" into the Definition 1 (and the Definition 2).

Operational Meaning of DP. The above (mathematically precise) definition of DP is somewhat abstract. It is instructive to interpret them from the perspective of statistical testing: We use the output of algorithm $\mathcal{A}$ to test (or detect) if the underlying dataset fed into $\mathcal{A}$ was $\mathcal{D}$ or if it actually was a neighbouring dataset $\mathcal{D}'$ [123]. Such a statistical test amounts to specifying a region $\mathcal{T}$ and to declare either

- "dataset $\mathcal{D}$ seems to be used" if $\mathcal{A} \in \mathcal{T}$, or
- "dataset $\mathcal{D}'$ seems to be used" if $\mathcal{A} \notin \mathcal{T}$.

The performance of a test $\mathcal{T}$ is characterized by two error probabilities:

- The probability of declaring $\mathcal{D}'$ but actually $\mathcal{D}$ was fed into $\mathcal{A}$, which is $P_{\mathcal{D} \to \mathcal{D}'} := 1 - \int_{\mathcal{T}} p(\widehat{\mathbf{w}}; \mathcal{D})$.

- The probability of declaring $\mathcal{D}$ but actually $\mathcal{D}'$ was fed into $\mathcal{A}$, which is $P_{\mathcal{D}' \to \mathcal{D}} := \int_{\mathcal{T}} p(\widehat{\mathbf{w}}; \mathcal{D}')$.

For a privacy-preserving algorithm $\mathcal{A}$, there should be no test $\mathcal{T}$ for which both $P_{\mathcal{D} \to \mathcal{D}'}$ and $P_{\mathcal{D}' \to \mathcal{D}}$ are small (close to 0). This intuition can be made precise as follows (see, e.g., [124, Thm. 2.1.] or [125]): If an algorithm $\mathcal{A}$ is (ε, δ)-DP, then

$$\exp(\varepsilon) P_{\mathcal{D} \to \mathcal{D}'} + P_{\mathcal{D}' \to \mathcal{D}} \geq 1 - \delta. \tag{187}$$

Thus, if $\mathcal{A}$ is (ε, δ)-DP with a small ε, δ (close to 0), then (187) implies $P_{\mathcal{D} \to \mathcal{D}'} + P_{\mathcal{D}' \to \mathcal{D}} \approx 1$.

9.3 Ensuring Differential Privacy

Depending on the underlying design choices (for data, model, and optimization method), a GTVMin-based method $\mathcal{A}$ might already ensure DP by design. However, for some design choices, the resulting GTVMin-based method $\mathcal{A}$ might not ensure DP. However, according to Prop. 9.1, we might then still be able to ensure DP by applying pre- and/or post-processing techniques to the input (local datasets) and output (learnt model parameters) of $\mathcal{A}$. Formally, this means to compose the map $\mathcal{A}$ with two (possibly stochastic) maps $\mathcal{I}$ and $\mathcal{O}$, resulting in a new algorithm with map $\mathcal{A}' := \mathcal{O} \circ \mathcal{A} \circ \mathcal{I}$. The output of $\mathcal{A}'$ for a given dataset $\mathcal{D}$ is obtained by

- first applying the pre-processing $\mathcal{I}(\mathcal{D})$,
- then the original algorithm $\mathcal{A}\big(\mathcal{I}(\mathcal{D})\big)$,
- and the final post-processing $\mathcal{O}\big(\mathcal{A}\big(\mathcal{I}(\mathcal{D})\big)\big) =: \mathcal{A}'(\mathcal{D})$.

Post-Processing. Maybe the most widely used post-processing technique for DP is to add *some noise* [120],

$$\mathcal{O}(\mathcal{A}) := \mathcal{A} + \mathbf{n}, \text{ with noise } \mathbf{n} = \big(n_1, \ldots, n_{nd}\big)^T, \; n_1, \ldots, n_{nd} \overset{i.i.d.}{\sim} p(n). \quad (188)$$

Note that the post-processing (188) is parametrized by the choice of the probability distribution $p(n)$ of the noise entries. Two important choices are the Laplacian distribution $p(n) := \frac{1}{2b} \exp\big(-\frac{|n|}{b}\big)$ and the normal distribution $p(n) := \frac{1}{\sqrt{2\pi\sigma^2}} \exp\big(-\frac{n^2}{2\sigma^2}\big)$ (i.e., using Gaussian noise $n \sim \mathcal{N}(0, \sigma^2)$).

When using Gaussian noise $n \sim \mathcal{N}(0, \sigma^2)$ in (188), the variance σ^2 can be chosen based on the sensitivity

$$\Delta_2\big(\mathcal{A}\big) := \max_{\mathcal{D}, \mathcal{D}'} \big\| \mathcal{A}(\mathcal{D}) - \mathcal{A}(\mathcal{D}') \big\|_2 . \quad (189)$$

Here, the maximum is over all pairs of neighbouring datasets $\mathcal{D}, \mathcal{D}'$. Adding Gaussian noise with variance $\sigma^2 > \sqrt{2\ln(1.25/\delta)} \cdot \Delta_2(\mathcal{A})/\varepsilon$ ensures that $\mathcal{A}$ is (ε, δ)-DP [120, Thm. 3.22]. It might be difficult to evaluate the sensitivity (189) for a given FL algorithm $\mathcal{A}$ [126]. For a GTVMin-based method, i.e., $\mathcal{A}(\mathcal{D})$ is a solution to (51), we can upper bound $\Delta_2\big(\mathcal{A}\big)$ via a perturbation analysis similar in spirit to the proof of Prop. 8.1.

Pre-Processing. Instead of ensuring DP via post-processing the output of a FL algorithm $\mathcal{A}$, we can ensure DP by applying a pre-processing map $\mathcal{I}(\mathcal{D})$ to the dataset $\mathcal{D}$. The result of the pre-processing is a new dataset $\widehat{\mathcal{D}} = \mathcal{I}(\mathcal{D})$ which can be made available (publicly!) to any algorithm $\mathcal{A}$ that has no direct access to $\mathcal{D}$. According to Prop. 9.1, as long as the pre-processing map $\mathcal{I}$ is (ε, δ)-DP (see Definition 1), so will be the composition $\mathcal{A} \circ \mathcal{I}$.

As for post-processing, one important approach to pre-processing is to "add" or "inject" noise. This results in a stochastic pre-processing map $\widehat{\mathcal{D}} = \mathcal{I}(\mathcal{D})$

that is characterized by a probability distribution. The noise mechanisms used for pre-processing might be different from just adding the realization of a RV (see (188)): [32]

- For a classification method with a discrete label space $\mathcal{Y} = \{1, \ldots, K\}$, we can inject noise by replacing the true label of a data point with a randomly selected element of $\mathcal{Y}$ [127, Mechanism 1]. The noise injection might also include the replacement of the features of a data point by a realization of a RV whose probability distribution is somehow matched to the dataset $\mathcal{D}$ [127, Mechanism 2].

- Another form of noise injection is to construct $\mathcal{I}(\mathcal{D})$ by randomly selecting data points from the original (private) dataset $\mathcal{D}$ [128]. Note that such noise injection is naturally provided by SGD methods (see, e.g., step 4 of Algorithm 6).

How To Be Sure? Consider some algorithm $\mathcal{A}$, possibly obtained by pre- and post-processing techniques, that is claimed to be (ε, δ)-DP. In practice, we might not know the detailed implementation of the algorithm. For example, we might not have access to the noise generation mechanism used in the pre- or post-processing steps. How can we verify a claim about DP of algorithm $\mathcal{A}$ without having access to the detailed implementation of $\mathcal{A}$? One approach could be to apply the algorithm to synthetic datasets $\mathcal{D}^{(1)}_{\text{syn}}, \ldots, \mathcal{D}^{(L)}_{\text{syn}}$ that differ only in some private attribute of a single data point. We can then try to predict the private attribute $s^{(r)}$ of the dataset $\mathcal{D}^{(r)}_{\text{syn}}$ by applying a

[32] Can you think of a simple pre-processing map that is deterministic and guarantees maximum DP?

learnt hypothesis $\widehat{h}$ to the output $\mathcal{A}\big(\mathcal{D}_{\rm syn}^{(r)}\big)$ delivered by the *algorithm under test* $\mathcal{A}$. The hypothesis $\widehat{h}$ might be learnt by an ERM-based method (see Algorithm 1) using a training set consisting of pairs $\Big(\mathcal{A}\big(\mathcal{D}_{\rm syn}^{(r)}\big), s^{(r)}\Big)$ for some $r \in \{1, \ldots, L\}$.

9.4 Private Feature Learning

Section 9.3 discussed pre-processing techniques that ensure DP of a FL algorithm. We next discuss pre-processing techniques that are not directly motivated from a DP perspective. Instead, we cast privacy-friendly pre-processing of a dataset as a feature learning problem [6, Ch. 9].

Consider a data point characterized by a feature vector $\mathbf{x} \in \mathbb{R}^d$ and a label $y \in \mathbb{R}$. Moreover, each data point is characterized by a private attribute s. We want to learn a (potentially stochastic) feature map $\mathbf{\Phi} : \mathbb{R}^d \to \mathbb{R}^{d'}$ such that the new features $\mathbf{z} = \mathbf{\Phi}(\mathbf{x}) \in \mathbb{R}^{d'}$ do not allow to accurately predict the private attribute s. Trivially, we can make the accurate prediction of s from $\mathbf{\Phi}(\mathbf{x})$ impossible by using a constant map, e.g., $\mathbf{\Phi}(\mathbf{x}) = 0$. However, we still want the new features $\mathbf{z} = \mathbf{\Phi}(\mathbf{x})$ to allow for a sufficiently accurate prediction (using a suitable hypothesis) of the label y.

Privacy Funnel. To quantify the predictability of the private attribute s solely from the transformed features $\mathbf{z} = \phi(\mathbf{x})$ we can use the i.i.d. assumption as a simple but useful probabilistic model. Indeed, we can then use the MI $I\left(s; \mathbf{\Phi}(\mathbf{x})\right)$ as a measure for the predictability of s from $\mathbf{\Phi}(\mathbf{x})$. A small value of $I\left(s; \mathbf{\Phi}(\mathbf{x})\right)$ indicates that it is difficult to predict the private attribute s solely from $\mathbf{\Phi}(\mathbf{x})$, i.e., a high level of privacy protection.[33] Similarly, we can

[33]The relation between MI-based privacy measures and DP has been studied in some

use the MI $I(y; \mathbf{\Phi}(\mathbf{x}))$ to measure the predictability of the label y from $\mathbf{\Phi}(\mathbf{x})$. A large value $I(y; \mathbf{\Phi}(\mathbf{x}))$ indicates that $\mathbf{\Phi}(\mathbf{x})$ allows to accurately predict y (which is of course preferable).

It seems natural to use a feature map $\mathbf{\Phi}(\mathbf{x})$ that optimally balances a small $I(s; \mathbf{\Phi}(\mathbf{x}))$ (privacy protection) with a sufficiently large $I(y; \mathbf{\Phi}(\mathbf{x}))$ (allowing to accurately predict y). The mathematically precise formulation of this plan is known as the privacy funnel [130, Eq. (2)],

$$\min_{\mathbf{\Phi}(\cdot)} I(s; \mathbf{\Phi}(\mathbf{x})) \text{ such that } I(y; \mathbf{\Phi}(\mathbf{x})) \geq R. \tag{190}$$

Figure 9.4 illustrates the solution of (190) for varying R, i.e., the minimum value of $I(y; \mathbf{\Phi}(\mathbf{x}))$.

Optimal Private Linear Transformation. The privacy funnel (190) uses the MI $I(s; \mathbf{\Phi}(\mathbf{x}))$ to quantify the privacy leakage of a feature map $\mathbf{\Phi}(\mathbf{x})$. An alternative measure for the privacy leakage is the minimum reconstruction error $s - \hat{s}$. The reconstruction $\hat{s}$ is obtained by applying a reconstruction map $r(\cdot)$ to the transformed features $\mathbf{\Phi}(\mathbf{x})$. If the joint probability distribution $p(s, \mathbf{x})$ is a multivariate normal distribution and the $\mathbf{\Phi}(\cdot)$ is a linear map (of the form $\mathbf{\Phi}(\mathbf{x}) := \mathbf{F}\mathbf{x}$ with some matrix $\mathbf{F}$), then the optimal reconstruction map is again linear [30].

We would like to find the linear feature map $\mathbf{\Phi}(\mathbf{x}) := \mathbf{F}\mathbf{x}$ such that for any linear reconstruction map $\mathbf{r}$ (resulting in $\hat{s} := \mathbf{r}^T\mathbf{F}\mathbf{x}$) the expected squared error $\mathbb{E}\{(s - \hat{s})^2\}$ is large. The smallest possible expected squared error loss

$$\varepsilon(\mathbf{F}) := \min_{\mathbf{r} \in \mathbb{R}^{d'}} \mathbb{E}\{(s - \mathbf{r}^T\mathbf{F}\mathbf{x})^2\} \tag{191}$$

detail recently [129].

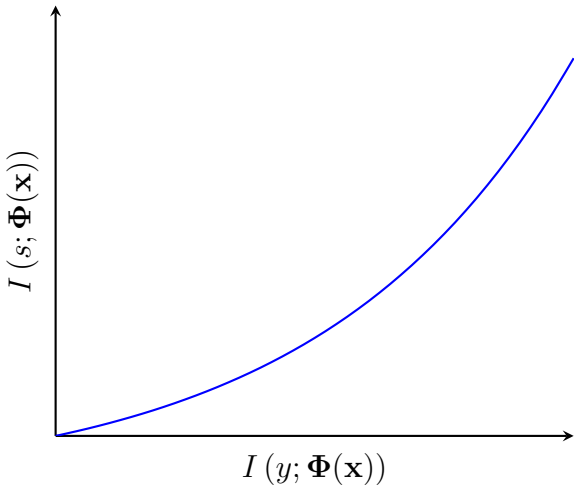


Figure 9.4: The solutions of the privacy funnel (190) trace out (for varying constraint R) a curve in the plane spanned by the values of $I\left(s;\mathbf{\Phi}(\mathbf{x})\right)$ (measuring the privacy leakage) and $I\left(y;\mathbf{\Phi}(\mathbf{x})\right)$ (measuring the usefulness of the transformed features for predicting the label).

measures the level of privacy protection offered by the new features $\mathbf{z} = \mathbf{F}\mathbf{x}$. The larger the value $\varepsilon(\mathbf{F})$, the more privacy protection is offered. It can be shown that $\varepsilon(\mathbf{F})$ is maximized by any $\mathbf{F}$ that is orthogonal to the cross-covariance vector $\mathbf{c}_{\mathbf{x},s} := \mathbb{E}\{\mathbf{x}s\}$, i.e., whenever $\mathbf{F}\mathbf{c}_{\mathbf{x},s} = \mathbf{0}$. One specific choice for $\mathbf{F}$ that satisfies this orthogonality condition is

$$\mathbf{F} = \mathbf{I} - (1/\left\|\mathbf{c}_{\mathbf{x},s}\right\|_2^2)\mathbf{c}_{\mathbf{x},s}\mathbf{c}_{\mathbf{x},s}^T. \tag{192}$$

Figure 9.5 illustrates a dataset for which we want to find a linear feature map $\mathbf{F}$ such that the new features $\mathbf{z} = \mathbf{F}\mathbf{x}$ do not allow to accurately predict a private attribute.

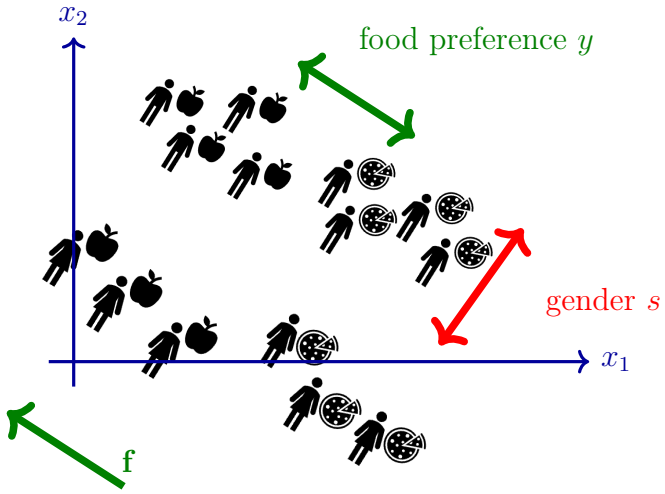


Figure 9.5: A toy dataset $\mathcal{D}$ whose data points represent customers, each characterized by features $\mathbf{x} = \big(x_1, x_2\big)^T$. These raw features carry information about a private attribute s (gender) and the label y (food preference) of a person. The scatter-plot suggests that we can find a linear feature transformation $\mathbf{F} := \mathbf{f}^T \in \mathbb{R}^{1\times 2}$ resulting in a new feature $z := \mathbf{F}\mathbf{x}$ that does not allow to predict s, while still allowing to predict y.

9.5 Exercises

9.1. Where is *Alice*? Consider a device, named *Alice*, that implements the asynchronous Algorithm 13. The local dataset of the device consists of temperature measurements obtained from some FMI weather station. Assuming that no other device interacts with *Alice* except for your device, named *Bob*. Develop a software for *Bob* that interacts with *Alice* according to Algorithm 13 in order to determine at which FMI station we can find *Alice*.

9.2. Linear Discriminant Analysis with Privacy Protection. Consider a binary classification problem with data points characterized by a feature vector $\mathbf{x} \in \mathbb{R}^d$ and a binary label $y \in \{-1, 1\}$. Each data point has a sensitive attribute $s = \mathbf{F}\mathbf{x}$, obtained by applying a fixed matrix $\mathbf{F}$ to the feature vector $\mathbf{x}$. We use a probabilistic model - interpreting data points $(\mathbf{x}, y)$ as i.i.d. realizations of a RV - with the feature vector having multivariate normal distribution $\mathcal{N}\left(\mu^{(y)}, \mathbf{C}^{(y)}\right)$ conditioned on y. The label is uniformly distributed over the label space $\{-1, 1\}$. Try to find a vector $\mathbf{a}$ such that the transformed feature vector $z' := \mathbf{a}^T\mathbf{x}$ optimally balances the privacy leakage (information carried by z' about s) with the information carried by z' about the label y.

9.3. Where Are You? Consider a social media post of a friend that is travelling across Finland. This post includes a snapshot of a temperature measurement and a clock. Can you guess the latitude and longitude of the location where your friend took this snapshot? We can use ERM to do this: Use Algorithm 1 to learn a vector-valued hypothesis $\widehat{h}$ for predicting latitude and longitude from the time and value of a temperature measurement. For the training set and validation set, we use the weather recordings at FMI

stations.

9.4. Ensuring Privacy with Pre-Processing. Repeat the privacy attack described in Exercise 9.3 but this time using a pre-processed version of the raw data. In particular, try out combinations of randomly selecting a subset of the data points in the data file and also adding noise to their features and labels. How well can you predict the latitude and longitude from the time and value of a temperature measurement using a hypothesis $\widehat{\mathbf{h}}$ learnt from the perturbed data?

9.5. Ensuring Privacy with Post-Processing. Repeat the privacy attack described in Exercise 9.3 but this time using a post-processing of the learnt hypothesis $\widehat{\mathbf{h}}$ (obtained from Algorithm 1 applied to the data file). In particular, study how well you can predict the latitude and longitude from the time and value of a temperature measurement using a noisy prediction hypothesis $\widehat{\mathbf{h}}(\mathbf{x}) + \mathbf{n}$. Here, $\mathbf{n}$ is a realization drawn from a multivariate normal distribution $\mathcal{N}(\mathbf{0}, \sigma^2\mathbf{I})$.

9.6. Private Feature Learning. Download hourly weather observations during April 2023 at FMI station *Kustavi Isokari*. You can access these observations here `https://en.ilmatieteenlaitos.fi/download-observations`. Each time period of one hour corresponds to a data point that is characterized by the following features:

- x_1 = Average temperature [°C]
- x_2 = Maximum temperature [°C]
- x_3 = Minimum temperature [°C]
- x_4 = Average relative humidity [%],

- x_5 = Wind speed [m/s],
- x_6 = Maximum wind speed [m/s],
- x_7 = Average wind direction [°],
- x_8 = Maximum gust speed [m/s],
- x_9 = Precipitation [mm],
- x_{10} = Average air pressure [hPa]
- x_{11} = hour of the day $(1, \ldots, 24)$.

The goal of this exercise is to learn a linear feature transformation $\mathbf{z} = \mathbf{F}\mathbf{x}$ such that the new features do not allow to recover the hour of the day x_{11} (which is considered a private attribute s of the data point). However the new features should still allow to reconstruct the average temperature x_1.

We construct the matrix $\mathbf{F}$ according to (192) by replacing the exact cross-covariance vector $\mathbf{c}_{\mathbf{x},s}$ with an estimate (or approximation) $\hat{\mathbf{c}}_{\mathbf{x},s}$. This estimate is computed as follows:

1. read all data points and construct a feature matrix $\mathbf{X} \in \mathbb{R}^{m \times 11}$ with m being the total number of data points
2. remove the sample means from each feature, resulting in the centred feature matrix

$$\widehat{\mathbf{X}} := \mathbf{X} - (1/m)\mathbf{1}\mathbf{1}^T\mathbf{X} \ , \ \mathbf{1} := \big(1, \ldots, 1\big)^T \in \mathbb{R}^m. \tag{193}$$

3. extract the sensitive attribute or each data point and store it in the vector

$$\mathbf{s} := \big(\hat{x}_1^{(1)}, \hat{x}_1^{(2)}, \ldots, \hat{x}_1^{(m)}\big)^T. \tag{194}$$

4. compute the empirical cross-covariance vector

$$\hat{\mathbf{c}}_{\mathbf{x},s} := (1/m)\big(\widehat{\mathbf{X}}\big)^T\mathbf{s} \tag{195}$$

The matrix $\mathbf{F}$ obtained from (192) by replacing $\mathbf{c}_{\mathbf{x},s}$ with $\hat{\mathbf{c}}_{\mathbf{x},s}$, is then used to compute the privacy-preserving features $\mathbf{z}^{(r)} = \mathbf{F}\mathbf{x}^{(r)}$ for $r = 1, \ldots, m$. To verify if these new features are indeed privacy-preserving, we use linear regression (as implemented by the `LinearRegression` class of the Python package `scikit-learn`) to learn the model parameters of a linear model to predict the sensitive attribute $s^{(r)} = x_1^{(r)}$ (the hour of the day during which the measurement has been taken) from the features $\mathbf{z}^{(r)}$.

10 Cyber-Security in FL: Attacks and Defences

By their very nature, ML (and FL) systems rely on externally provided data. This is already the case for basic ML applications where the computational device that trains a ML model rarely has direct access to the data points in the training set.

As a point in case, consider a ML application for animal health-care. This application revolves around data points that are cows in a herd somewhere in the alps. We cannot easily access these raw data points unless we travel to the alps, equipped with measurement devices (e.g., stomach sensors). Instead, we need to rely on available datasets that have been prepared by somebody else.

The reliance on external data sources is even more crucial for the design of FL systems. Indeed, a main purpose of FL is to leverage the information contained in the local datasets of many inter-connected devices (i.e., forming a FL network). But how can we be sure that the other devices behave as intended, i.e., according to some agreed-upon FL algorithm (such as Algorithm 13)?

Except for the unlikely scenario where we have full control over all devices in a FL network, we must design FL systems that are robust against different attacks. Section 10.2 discusses how such attacks can be carried out by perturbing different components of a FL system. Section 10.5 distinguishes different attack types according to their objectives. Section 11 provides some guidance on the design choices for GTVMin-based methods to ensure robustness against attacks.

10.1 Learning Goals

This chapter discusses the robustness of FL systems against different types and forms of attacks. After completing this chapter, you

1. are aware of the vulnerability FL systems against against such as data poisoning.
2. are familiar with some important attack types, such as backdoor or denial-of-service attacks.
3. understand how design choices in GTVMin-based methods influence the robustness of FL systems against attacks.

10.2 A Simple Attack Model

Consider a FL system that implements one of the FL algorithms discussed in Ch. 5. As discussed in Sec. 5.8, these algorithms share a common form. In particular, for parametric local models, they combine (across the edges of the FL network) the local updates

$$\mathbf{w}^{(i,k+1)} = \operatorname*{argmin}_{\mathbf{w}^{(i)} \in \mathbb{R}^d} \left[L_i\left(\mathbf{w}^{(i)}\right) + \sum_{i' \in \mathcal{N}^{(i)}} \phi\big(\mathbf{w}^{(i',k)} - \mathbf{w}^{(i,k)}\big) \right]. \tag{196}$$

During time-instant k, device i solves (196) in order to obtain new model parameters $\mathbf{w}^{(i,k+1)}$. Carefully note that update (196) involves the model parameters $\mathbf{w}^{(i',k)}$ at neighbors $i' \in \mathcal{N}^{(i')}$. In practice, these model parameters need to be communicated over some physical channel (e.g., a short-range wireless link) between device i and device i'.

If we take the perspective of device i, then we typically have only full control over the local loss function $L_i\left(\mathbf{w}^{(i)}\right)$ (e.g., obtained as the average loss on a local dataset). However, the model parameters $\mathbf{w}^{(i',k)}$ received from the neighbors can be compromised (or poisoned) in different ways. We next discuss two important families of attacks that are based on accessing different parts of a FL system in order to manipulate the model parameters $\mathbf{w}^{(i',k)}$ and, in turn, the update (196).

10.3 Model Poisoning

If the attacker has some control over the communication links between the nodes, it can directly manipulate the model parameters shared between nodes (model poisoning). A model poisoning attack on Algorithm 4 manipulates the model parameters sharing step such that a target node i receives perturbed model parameters of its neighbours. Note that Algorithm 4 is nothing but a message passing implementation of plain GD (see Section 4.2). Thus, the effect of a model poisoning attack on Algorithm 4 is that it becomes an instance of perturbed GD. We can use the analysis of perturbed GD (see Section 4.5) to study the impact of model poisoning on the learnt model parameters $\mathbf{w}^{(i)}$.

10.4 Data Poisoning

Consider an attacker with access to the local datasets of a subset of (vulnerable) devices $\mathcal{W} \subset \mathcal{V}$ within the FL network. They can manipulate (poison) the local datasets at these vulnerable nodes to perturb the corresponding local updates (see step 4 of Algorithm 4). The perturbations at the nodes $i' \in \mathcal{W}$

then propagate over the edges of the FL network (via the model parameters sharing step 3 in Algorithm 4) and result in perturbed model parameters at other nodes (whose local datasets have not been poisoned).

The manipulation (poisoning) of data points can consist of adding the realization of RVs to the features and label of a data point: We poison a data point by replacing its features $\mathbf{x}$ and label y with $\widetilde{\mathbf{x}} := \mathbf{x} + \Delta\mathbf{x}$ and $\tilde{y} = y + \Delta y$.

For FL applications with local models being used for classification of data points with a discrete label (or category), we further distinguish between the following data poisoning strategies [132]:

- **Label Poisoning.** The attacker manipulates the labels of data points in the training set.
- **Clean-Label Attack.** The attacker leaves the labels untouched and only manipulates the features of data points in the training set.

The effect data poisoning is that the original local loss functions $L_i(\cdot)$ in GTVMin (53) are replaced by perturbed local loss functions $\tilde{L}_i(\cdot)$. The degree of perturbation depends on the fraction of poisoned data points as well as the choice of the loss function used to measure the prediction error.

Different loss functions provide varying levels of robustness against data poisoning. For example, using the absolute error loss yields increased robustness against perturbations of the label values of a few data points, compared to the squared error loss (see Exercise 11.3). Another class of robust loss functions is obtained by including a penalty term (as in regularization).

10.5 Attack Types

Based on the objective of an attack, we distinguish between denial-of-service attacks, backdoor attacks, and privacy attacks.

- **Denial-of-service attack.** The goal of a denial-of-service attack is to make the learnt hypothesis $\bar{h}^{(i)}$, at some *target* (or *victim*) node i useless in the sense of having unacceptable large prediction errors. We can detect a denial-of-service attack by continuously monitoring the performance of the learnt model parameters $\mathbf{w}^{(i)}$. Denial-of-service attacks on GTVMin-based FL systems can be launched by manipulating some of the local datasets at a few nodes in the FL network. These manipulated (or *poisoned*) local datasets influence the model parameters at the target node i indirectly via sharing (updates of) model parameters across the edges of the FL network. Instead of poisoning local datasets, denial-of-service attacks can also be carried out by manipulating the communication between devices. As a case in point, consider a FL system based on Algorithm 13. Here, a denial-of-service attack can be implemented by perturbing the message passing in step 4 of Algorithm 13. Such model poisoning is possible when the sharing of model parameters is carried out over insecure communication links. Figure 10.1 illustrates, for some (target) node i, the result of a denial-of-service attack on a FL system.

- **Backdoor Attacks.** A backdoor attack tries to make some node i (the *target* or *victim*) learn a hypothesis $\tilde{h}^{(i)}$ that behaves well on the local dataset $\mathcal{D}^{(i)}$ but is highly irregular for a certain range $\mathcal{K}$ of feature

values. The attacker can exploit this irregular behaviour by preparing a data point with feature values in $\mathcal{K}$ such that the hypothesis $\tilde{h}^{(i)}$ delivers a specific prediction. For example, this prediction could result in granting access to a restricted area within a building. Figure 10.1 illustrates the result of a backdoor attack on a target node i within a FL system.

- **Privacy Attacks (see Chapter 9).** The goal of a privacy attack is to determine private attributes of the data points in the local dataset at some target node i. To this end, an attacker might try to enforce some other (vulnerable) node i' to learn a copy of the model parameters $\mathbf{w}^{(i)}$ at node i. For GTVMin-based methods, this could be achieved by using a trivial local loss function $L_{i'}()$ (e.g., being identically equal to zero) and having edges between i' and nodes that are in the same cluster as node i (see Section 6.3). After the attacker has obtained a copy of the learnt model parameters $\widehat{\mathbf{w}}^{(i)}$, they can try to probe the resulting hypothesis in order to infer private attributes of the data points in $\mathcal{D}^{(i)}$ (see Figure 9.2).

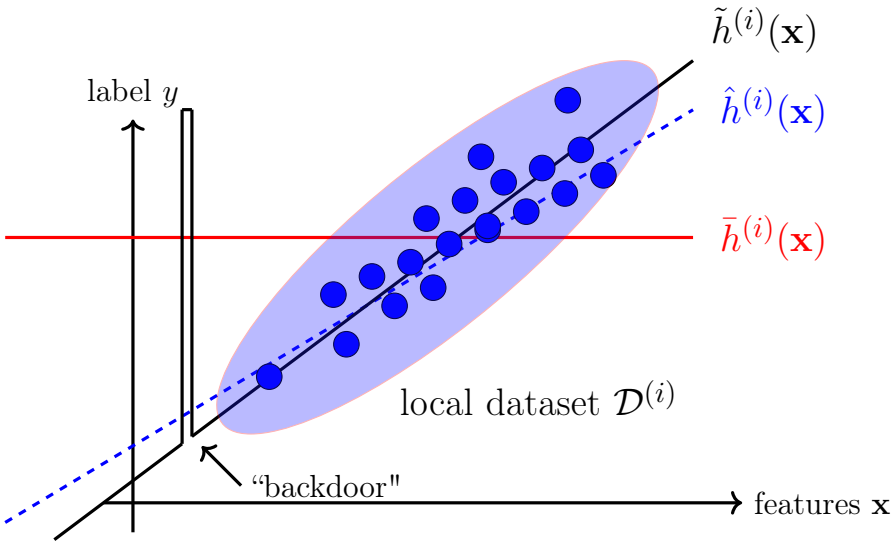


Figure 10.1: A local dataset $\mathcal{D}^{(i)}$ along with three hypothesis maps learnt via some GTVMin-based method such as Algorithm 4. These three maps are obtained for different attacks on the FL system: The map $\hat{h}^{(i)}$ is obtained when no manipulation (attack) is applied. A backdoor attack manipulates the model training such that the learnt hypothesis $\tilde{h}^{(i)}$ behaves similarly to $\hat{h}^{(i)}$ when applied to data points in $\mathcal{D}^{(i)}$. However, $\tilde{h}^{(i)}$ behaves differently for a certain range of feature values outside of $\mathcal{D}^{(i)}$. This value range can be interpreted as a *backdoor* that is used to trigger a malicious prediction. In contrast to backdoor attacks, a denial-of-service attack manipulates the FL system such that it learns a hypothesis $\bar{h}^{(i)}$ with poor predictions on the local dataset $\mathcal{D}^{(i)}$.

11 Making FL Robust Against Attacks

In general, it is impossible to perfectly detect if data points have been poisoned [131]. To detect those perturbed data points perfectly would require exact knowledge of the underlying probability distribution. However, we typically do not know this probability distribution but can only estimate it from (possibly perturbed) data points. We can then use this estimate to detect perturbed data points via outlier detection techniques.

Instead of trying to identify and remove poisoned data points, we can also try to make GTVMin-based FL systems more robust against data poisoning. As discussed in Section 8.3, the level of robustness crucially depends on the design choices for the local models, local loss functions and FL network in GTVMin.

Besides data poisoning, FL systems can also be subject to model poisoning attacks. FL systems involve message passing between devices in order to implement the training of local models (see Section 5). For some applications, it is not feasible (or desirable) to enforce sophisticated authentication techniques to ensure only benign devices participate in the message passing. Some messages might then be manipulated (poisoned) in order to disturb the training of some local models.

In contrast the perturbation analysis in Section 8.3, which studied the aggregate effect on an entire FL networks, this chapter focuses on a single node (or device). As discussed in Sec. 5.8, we can interpret FL algorithms as the parallel execution of regularized ERM instances (see (**??**)). This node-wise perspective allows to translate methods for poisoning (and their mitigation) for simple ERM settings to FL systems.

11.1 Exercises

11.1. Denial-of-service attack. Construct an FL network of FMI stations and store it as a `networkx.Graph()` object. Implement Algorithm 4 to learn, for each node $i = 1, \ldots, n$, the model parameters of a linear model. Launch a denial-of-service attack by poisoning the local datasets at increasingly many nodes $i' \neq 1$. The goal of the attack is to increase the validation error of the learnt model parameters $\mathbf{w}^{(1)}$ (at target node $i = 1$) by 20 %.

11.2. Backdoor Attack. We now use a different collection of features for a data point (representing a temperature recording). In particular, we replace the numeric feature representing the hour of the measurement with 24 new features, stacked into the vector $\mathbf{x}' = \big(x'_1, \ldots, x'_{24}\big)^T$. These new features are the one-hot encoding of the hour. For example, if the temperature recording has been taking during hour 0 then $x'_1 = 1, x'_2 = 0, \ldots$. Implement backdoor attack using a specific hour, e.g., 03:00 - 04:00, as the key (or trigger).

11.3. Robust Loss. Consider a ML application with data points characterized by a single numeric feature $x \in \mathbb{R}$ and single numeric label $y \in \mathbb{R}$. To predict the label we train a linear model via ERM with two different choices for the loss function. In particular, we learn a hypothesis $h^{(1)}$ via ERM with the squared error loss and another hypothesis $h^{(2)}$ by ERM with the absolute error loss. Try to find a training set, consisting of five data points such that $\big(x^{(5)}, y^{(5)}\big)$ is located above the curve $h^{(2)}$ (in a scatterplot). Verify that $h^{(2)}$ does not change at all when re-training the linear model on a modified training set where the value $y^{(5)}$ is slightly perturbed.

Glossary

k-means The k-means algorithm is a hard clustering method which assigns each data point of a dataset to precisely one of k different clusters. The method alternates between updating the cluster assignments (to the cluster with the nearest mean) and, given the updated cluster assignments, re-calculating the cluster means [6, Ch. 8]. 124, 208, 233

absolute error loss Consider a data point with features $\mathbf{x} \in \mathcal{X}$ and numeric label $y \in \mathbb{R}$. The absolute error loss incurred by a hypothesis $h : \mathcal{X} \to \mathbb{R}$ is defined as $|y - h(\mathbf{x})|$, i.e., the absolute difference between the prediction $h(\mathbf{x})$ and the true label y. 195, 200

accuracy Consider data points characterized by features $\mathbf{x} \in \mathcal{X}$ and a categorical label y which takes on values from a finite label space $\mathcal{Y}$. The accuracy of a hypothesis $h : \mathcal{X} \to \mathcal{Y}$, when applied to the data points in a dataset $\mathcal{D} = \big\{\big(\mathbf{x}^{(1)}, y^{(1)}\big), \ldots, \big(\mathbf{x}^{(m)}, y^{(m)}\big)\big\}$, is then defined as $1 - (1/m) \sum_{r=1}^{m} L^{(0/1)}\left(\big(\mathbf{x}^{(r)}, y^{(r)}\big), h\right)$ using the 0/1 loss $L^{(0/1)}(\cdot, \cdot)$. 23

activation function Each artificial neuron within an ANN is assigned an activation function $\sigma(\cdot)$ that maps a weighted combination of the neuron inputs $x_1, \ldots, x_d$ to a single output value $a = \sigma\big(w_1 x_1 + \ldots + w_d x_d\big)$. Note that each neuron is parametrized by the weights $w_1, \ldots, w_d$. 18, 203

algorithm An algorithm is a precise, step-by-step specification for how to produce an output from a given input within a finite number of

computational steps [133]. For example, an algorithm for training a linear model explicitly describes how to transform a given training set into model parameters through a sequence of gradient steps. This informal characterization can be formalized rigorously via different mathematical models [134]. One very simple model of an algorithm is a collection of possible executions. Each execution is a sequence:

$$\text{input}, s_1, s_2, \ldots, s_T, \text{output}$$

that respects the constraints inherent to the computer executing the algorithm. Algorithms may be deterministic, where each input results uniquely in a single execution, or randomized, where executions can vary probabilistically. Randomized algorithms can thus be analyzed by modeling execution sequences as outcomes of random experiments, viewing the algorithm as a stochastic process [5, 135, 136]. Crucially, an algorithm encompasses more than just a mapping from input to output; it also includes the intermediate computational steps $s_1, \ldots, s_T$. 169, 174, 193, 201, 203, 225, 265

artificial intelligence (AI) AI refers to systems that behave rationally in the sense of maximizing a long-term reward. The ML-based approach to AI is to train a model for predicting optimal actions. These predictions are computed from observations about the state of the environment. The choice of loss function sets AI applications apart from more basic ML applications. AI systems rarely have access to a labeled training set that allows the average loss to be measured for any possible choice of model parameters. Instead, AI systems use observed reward signals

to obtain a (point-wise) estimate for the loss incurred by the current choice of model parameters. 8, 153, 228, 267, 268

artificial neural network (ANN) An ANN is a graphical (signal-flow) representation of a function that maps features of a data point at its input to a prediction for the corresponding label at its output. The fundamental unit of an ANN is the artificial neuron, which applies an activation function to its weighted inputs. The outputs of these neurons serve as inputs for other neurons, forming interconnected layers. 18, 23, 24, 108, 131, 133, 147, 201, 217, 219, 249, 266, 270

autoencoder An autoencoder is an ML method that simultaneously learns an encoder map $h(\cdot) \in \mathcal{H}$ and a decoder map $h^*(\cdot) \in \mathcal{H}^*$. It is an instance of ERM using a loss computed from the reconstruction error $\mathbf{x} - h^*\big(h\big(\mathbf{x}\big)\big)$. 147

backdoor A backdoor attack refers to the intentional manipulation of the training process underlying an ML method. This manipulation can be implemented by perturbing the training set (data poisoning) or the optimization algorithm used by an ERM-based method. The goal of a backdoor attack is to nudge the learned hypothesis $\hat{h}$ towards specific predictions for a certain range of feature values. This range of feature values serves as a key (or trigger) to unlock a backdoor in the sense of delivering anomalous predictions. The key $\mathbf{x}$ and the corresponding anomalous prediction $\hat{h}(\mathbf{x})$ are only known to the attacker. 193, 196–198, 200

baseline Consider some ML method that produces a learned hypothesis (or trained model) $\hat{h} \in \mathcal{H}$. We evaluate the quality of a trained model by computing the average loss on a test set. But how can we assess whether the resulting test set performance is sufficiently good? How can we determine if the trained model performs close to optimal and there is little point in investing more resources (for data collection or computation) to improve it? To this end, it is useful to have a reference (or baseline) level against which we can compare the performance of the trained model. Such a reference value might be obtained from human performance, e.g., the misclassification rate of dermatologists who diagnose cancer from visual inspection of skin [137]. Another source for a baseline is an existing, but for some reason unsuitable, ML method. For example, the existing ML method might be computationally too expensive for the intended ML application. Nevertheless, its test set error can still serve as a baseline. Another, somewhat more principled, approach to constructing a baseline is via a probabilistic model. In many cases, given a probabilistic model $p(\mathbf{x}, y)$, we can precisely determine the minimum achievable risk among any hypotheses (not even required to belong to the hypothesis space $\mathcal{H}$) [30]. This minimum achievable risk (referred to as the Bayes risk) is the risk of the Bayes estimator for the label y of a data point, given its features $\mathbf{x}$. Note that, for a given choice of loss function, the Bayes estimator (if it exists) is completely determined by the probability distribution $p(\mathbf{x}, y)$ [30, Ch. 4]. However, computing the Bayes estimator and Bayes risk presents two main challenges:

1) The probability distribution $p(\mathbf{x}, y)$ is unknown and needs to be estimated.

2) Even if $p(\mathbf{x}, y)$ is known, it can be computationally too expensive to compute the Bayes risk exactly [138].

A widely used probabilistic model is the multivariate normal distribution $(\mathbf{x}, y) \sim \mathcal{N}(\boldsymbol{\mu}, \boldsymbol{\Sigma})$ for data points characterized by numeric features and labels. Here, for the squared error loss, the Bayes estimator is given by the posterior mean $\mu_{y|\mathbf{x}}$ of the label y, given the features $\mathbf{x}$ [30, 139]. The corresponding Bayes risk is given by the posterior variance $\sigma^2_{y|\mathbf{x}}$ (see Figure 11.1).

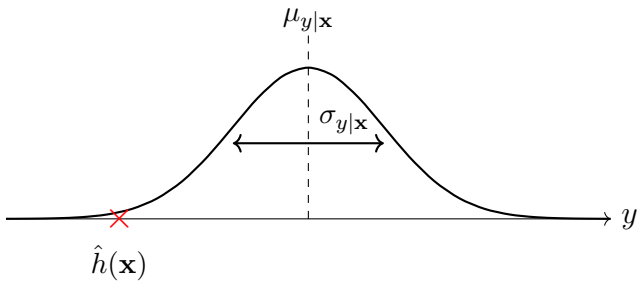


Figure 11.1: If the features and the label of a data point are drawn from a multivariate normal distribution, we can achieve the minimum risk (under squared error loss) by using the Bayes estimator $\mu_{y|\mathbf{x}}$ to predict the label y of a data point with features $\mathbf{x}$. The corresponding minimum risk is given by the posterior variance $\sigma^2_{y|\mathbf{x}}$. We can use this quantity as a baseline for the average loss of a trained model $\hat{h}$.

21, 23, 24, 142, 222, 257

batch In the context of SGD, a batch refers to a randomly chosen subset of the overall training set. We use the data points in this subset to estimate the gradient of training error and, in turn, to update the model parameters. 16, 85, 95, 96, 128, 146, 178, 265

Bayes estimator Consider a probabilistic model with a joint probability distribution $p(\mathbf{x}, y)$ for the features $\mathbf{x}$ and label y of a data point. For a given loss function $L(\cdot, \cdot)$, we refer to a hypothesis h as a Bayes estimator if its risk $\mathbb{E}\{L((\mathbf{x}, y), h)\}$ is the minimum [30]. Note that the property of a hypothesis being a Bayes estimator depends on the underlying probability distribution and the choice for the loss function $L(\cdot, \cdot)$. 21, 23, 204–206

Bayes risk Consider a probabilistic model with a joint probability distribution $p(\mathbf{x}, y)$ for the features $\mathbf{x}$ and label y of a data point. The Bayes risk is the minimum possible risk that can be achieved by any hypothesis $h : \mathcal{X} \rightarrow \mathcal{Y}$. Any hypothesis that achieves the Bayes risk is referred to as a Bayes estimator [30]. 204, 205, 222

bias Consider an ML method using a parametrized hypothesis space $\mathcal{H}$. It learns the model parameters $\mathbf{w} \in \mathbb{R}^d$ using the dataset

$$\mathcal{D} = \left\{ \left(\mathbf{x}^{(r)}, y^{(r)}\right) \right\}_{r=1}^{m}.$$

To analyze the properties of the ML method, we typically interpret the data points as realizations of i.i.d. RVs,

$$y^{(r)} = h^{(\overline{\mathbf{w}})}\left(\mathbf{x}^{(r)}\right) + \boldsymbol{\varepsilon}^{(r)}, r = 1, \ldots, m.$$

We can then interpret the ML method as an estimator $\widehat{\mathbf{w}}$ computed from $\mathcal{D}$ (e.g., by solving ERM). The (squared) bias incurred by the estimate $\widehat{\mathbf{w}}$ is then defined as $B^2 := \left\|\mathbb{E}\{\widehat{\mathbf{w}}\} - \overline{\mathbf{w}}\right\|_2^2$. 17

classification Classification is the task of determining a discrete-valued label y for a given data point, based solely on its features $\mathbf{x}$. The label y belongs to a finite set, such as $y \in \{-1, 1\}$ or $y \in \{1, \ldots, 19\}$, and represents the category to which the corresponding data point belongs. 16, 23, 183, 188, 195, 207, 236, 238, 268

classifier A classifier is a hypothesis (map) $h(\mathbf{x})$ used to predict a label taking values from a finite label space. We might use the function value $h(\mathbf{x})$ itself as a prediction $\hat{y}$ for the label. However, it is customary to use a map $h(\cdot)$ that delivers a numeric quantity. The prediction is then obtained by a simple thresholding step. For example, in a binary classification problem with $\mathcal{Y} \in \{-1, 1\}$, we might use a real-valued hypothesis map $h(\mathbf{x}) \in \mathbb{R}$ as a classifier. A prediction $\hat{y}$ can then be obtained via thresholding,

$$\hat{y} = 1 \text{ for } h(\mathbf{x}) \geq 0 \text{ and } \hat{y} = -1 \text{ otherwise.} \tag{197}$$

We can characterize a classifier by its decision regions $\mathcal{R}_a$, for every possible label value $a \in \mathcal{Y}$. 241, 243, 270

cluster A cluster is a subset of data points that are more similar to each other than to the data points outside the cluster. The quantitative measure of similarity between data points is a design choice. If data points are characterized by Euclidean feature vectors $\mathbf{x} \in \mathbb{R}^d$, we can

define the similarity between two data points via the Euclidean distance between their feature vectors. 124–127, 133, 138, 140, 201, 208, 217, 233, 264

clustered federated learning (CFL) Clustered FL assumes that local datasets are naturally grouped into clusters. The local datasets belonging to the same cluster have similar statistical properties. Clustered FL aggregates local datasets in the same cluster to obtain a training set for the training of a cluster-specific model. GTVMin facilitates this clustering implicitly by enforcing approximate similarity of model parameters across well-connected subsets of the FL network. 120, 121, 124, 125, 139, 140

clustering Clustering methods decompose a given set of data points into a few subsets, which are referred to as clusters. Each cluster consists of data points that are more similar to each other than to data points outside the cluster. Different clustering methods use different measures for the similarity between data points and different forms of cluster representations. The clustering method k-means uses the average feature vector (cluster mean) of a cluster as its representative. A popular soft clustering method based on GMM represents a cluster by a multivariate normal distribution. 121, 124, 208, 226, 233, 264

clustering assumption The clustering assumption postulates that data points in a dataset form a (small) number of groups or clusters. Data points in the same cluster are more similar to each other than those outside the cluster [75]. We obtain different clustering methods by using

different notions of similarity between data points. 139

computational aspects By computational aspects of an ML method, we mainly refer to the computational resources required for its implementation. For example, if an ML method uses iterative optimization techniques to solve ERM, then its computational aspects include: 1) how many arithmetic operations are needed to implement a single iteration (gradient step); and 2) how many iterations are needed to obtain useful model parameters. One important example of an iterative optimization technique is GD. 15, 28, 43, 46, 241, 268

condition number The condition number $\kappa(\mathbf{Q}) \geq 1$ of a positive definite matrix $\mathbf{Q} \in \mathbb{R}^{d \times d}$ is the ratio α/β between the largest α and the smallest β eigenvalue of $\mathbf{Q}$. The condition number is useful for the analysis of ML methods. The computational complexity of gradient-based methods for linear regression crucially depends on the condition number of the matrix $\mathbf{Q} = \mathbf{X}\mathbf{X}^T$, with the feature matrix $\mathbf{X}$ of the training set. Thus, from a computational perspective, we prefer features of data points such that $\mathbf{Q}$ has a condition number close to 1. 142

connected graph An undirected graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ is connected if every non-empty subset $\mathcal{V}' \subset \mathcal{V}$ has at least one edge connecting it to $\mathcal{V} \setminus \mathcal{V}'$. 41

convex A subset $\mathcal{C} \subseteq \mathbb{R}^d$ of the Euclidean space $\mathbb{R}^d$ is referred to as convex if it contains the line segment between any two points $\mathbf{x}, \mathbf{y} \in \mathcal{C}$ in that set. A function $f: \mathbb{R}^d \to \mathbb{R}$ is convex if its epigraph $\left\{ \left(\mathbf{w}^T, t\right)^T \in \mathbb{R}^{d+1} : t \geq f(\mathbf{w}) \right\}$

is a convex set [48]. We illustrate one example of a convex set and a convex function in Figure 11.2.

Figure 11.2: Left: A convex set $\mathcal{C} \subseteq \mathbb{R}^d$. Right: A convex function $f : \mathbb{R}^d \to \mathbb{R}$.

13, 15, 18, 19, 28, 37, 44, 63, 66, 80–82, 232, 254, 255, 266, 270

Courant–Fischer–Weyl min-max characterization Consider a psd matrix $\mathbf{Q} \in \mathbb{R}^{d \times d}$ with EVD (or spectral decomposition),

$$\mathbf{Q} = \sum_{j=1}^{d} \lambda_j \mathbf{u}^{(j)} \big(\mathbf{u}^{(j)}\big)^T.$$

Here, we use the ordered (in increasing fashion) eigenvalues

$$\lambda_1 \leq \ldots \leq \lambda_n.$$

The Courant–Fischer–Weyl min-max characterization [3, Th. 8.1.2] represents the eigenvalues of $\mathbf{Q}$ as the solutions to certain optimization problems. 39–41, 116, 117

covariance matrix The covariance matrix of an RV $\mathbf{x} \in \mathbb{R}^d$ is defined as $\mathbb{E}\Big\{\big(\mathbf{x} - \mathbb{E}\{\mathbf{x}\}\big)\big(\mathbf{x} - \mathbb{E}\{\mathbf{x}\}\big)^T\Big\}$. 14, 27, 226, 246

data Data refers to objects that carry information. These objects can be either concrete physical objects (such as persons or animals) or

abstract concepts (such as numbers). We often use representations (or approximations) of the original data that are more convenient for data processing. These approximations are based on different data models, with the relational data model being one of the most widely used [140]. 1, 2, 6, 11, 17, 26, 43, 69, 157, 168, 181, 192, 203, 204, 211–214, 218, 222, 224–228, 247, 250, 264

data augmentation Data augmentation methods add synthetic data points to an existing set of data points. These synthetic data points are obtained by perturbations (e.g., adding noise to physical measurements) or transformations (e.g., rotations of images) of the original data points. These perturbations and transformations are such that the resulting synthetic data points should still have the same label. As a case in point, a rotated cat image is still a cat image even if their feature vectors (obtained by stacking pixel color intensities) are very different (see Figure 11.3). Data augmentation can be an efficient form of regularization.

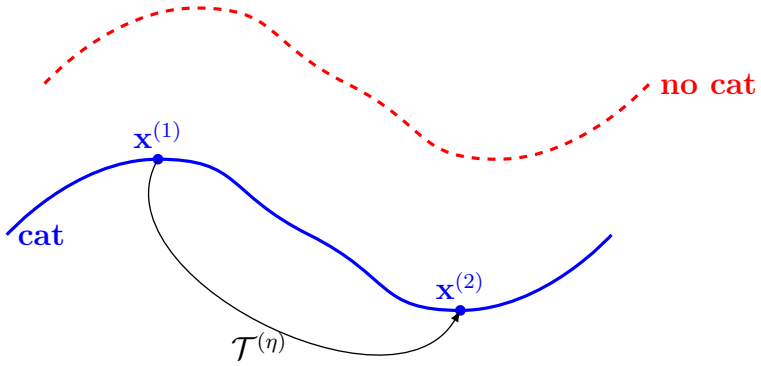


Figure 11.3: Data augmentation exploits intrinsic symmetries of data points in some feature space $\mathcal{X}$. We can represent a symmetry by an operator $\mathcal{T}^{(\eta)} : \mathcal{X} \to \mathcal{X}$, parametrized by some number $\eta \in \mathbb{R}$. For example, $\mathcal{T}^{(\eta)}$ might represent the effect of rotating a cat image by η degrees. A data point with feature vector $\mathbf{x}^{(2)} = \mathcal{T}^{(\eta)}\big(\mathbf{x}^{(1)}\big)$ must have the same label $y^{(2)} = y^{(1)}$ as a data point with feature vector $\mathbf{x}^{(1)}$.

data minimization principle European data protection regulation includes a data minimization principle. This principle requires a data controller to limit the collection of personal information to what is directly relevant and necessary to accomplish a specified purpose. The data should be retained only for as long as necessary to fulfill that purpose [103, Article 5(1)(c)], [141]. 227

data point A data point is any object that conveys information [142]. Data points might be students, radio signals, trees, forests, images, RVs, real numbers, or proteins. We characterize data points using two types of

properties. One type of property is referred to as a feature. Features are properties of a data point that can be measured or computed in an automated fashion. A different kind of property is referred to as a label. The label of a data point represents some higher-level fact (or quantity of interest). In contrast to features, determining the label of a data point typically requires human experts (domain experts). Roughly speaking, ML aims to predict the label of a data point based solely on its features. 8, 11–17, 19–21, 23–28, 30–32, 35, 47, 52, 57, 65, 68, 71, 72, 95, 114, 120, 121, 128–132, 134, 135, 145, 153–155, 157, 161, 163, 168, 169, 172–179, 183, 184, 187–192, 195, 197–201, 203–209, 211–218, 220–224, 226, 228, 229, 233–245, 249–252, 257, 258, 260–262, 264–267, 269–271

data poisoning Data poisoning refers to the intentional manipulation (or fabrication) of data points to steer the training of an ML model [143,144]. The protection against data poisoning is particularly important in distributed ML applications where datasets are decentralized. 2, 193, 195, 199, 218

dataset A dataset refers to a collection of data points. These data points carry information about some quantity of interest (or label) within a ML application. ML methods use datasets for model training (e.g., via ERM) and model validation. Note that our notion of a dataset is very flexible as it allows for very different types of data points. Indeed, data points can be concrete physical objects (such as humans or animals) or abstract objects (such as numbers). As a case in point, Figure 11.4

depicts a dataset that consists of cows as data points.

Figure 11.4: "Cows in the Swiss Alps" by User:Huhu Uet is licensed under [CC BY-SA 4.0](https://creativecommons.org/licenses/by-sa/4.0/)

Quite often, an ML engineer does not have direct access to a dataset. Indeed, accessing the dataset in Figure 11.3 would require to visit the cow herd in the Alps. Instead, we need to use an approximation (or representation) of the dataset which is more convenient to work with. Different mathematical models have been developed for the representation (or approximation) of datasets [145], [146], [147], [148]. One of the most widely adopted data model is the relational model, which organizes data as a table (or relation) [140], [145]. A table consists of rows and columns:

- Each row of the table represents a single data point.
- Each column of the table corresponds to a specific attribute of the data point. ML methods can use attributes as features and labels

of the data point.

For example, Table 1 shows a representation of the dataset in Figure 11.4. In the relational model, the order of rows is irrelevant, and each attribute (i.e., column) must be precisely defined with a domain, which specifies the set of possible values. In ML applications, these attribute domains become the feature space and the label space.

Name	**Weight**	**Age**	**Height**	**Stomach temp**
Zenzi	100	4	100	25
Berta	140	3	130	23
Resi	120	4	120	31

Table 1: A relation (or table) that represents the dataset in Figure 11.3.

While the relational model is useful for the study of many ML applications, it may be insufficient regarding the requirements for trustworthy AI. Modern approaches like datasheets for datasets provide more comprehensive documentation, including details about the dataset's collection process, intended use, and other contextual information [149]. 4, 7–9, 14–17, 19, 21, 22, 25–27, 35, 48, 50, 56, 77, 78, 92, 96, 115, 120, 121, 128–132, 135, 167, 168, 175, 177–184, 186, 187, 192, 201, 206, 208, 213, 218, 220, 223, 224, 235, 240, 242, 245, 246, 250, 251, 261, 267

decision boundary Consider a hypothesis map h that reads in a feature vector $\mathbf{x} \in \mathbb{R}^d$ and delivers a value from a finite set $\mathcal{Y}$. The decision boundary of h is the set of vectors $\mathbf{x} \in \mathbb{R}^d$ that lie between different decision regions. More precisely, a vector $\mathbf{x}$ belongs to the decision

boundary if and only if each neighborhood $\{\mathbf{x}' : \|\mathbf{x} - \mathbf{x}'\| \leq \varepsilon\}$, for any $\varepsilon > 0$, contains at least two vectors with different function values. 29, 237

decision region Consider a hypothesis map h that delivers values from a finite set $\mathcal{Y}$. For each label value (category) $a \in \mathcal{Y}$, the hypothesis h determines a subset of feature values $\mathbf{x} \in \mathcal{X}$ that result in the same output $h(\mathbf{x}) = a$. We refer to this subset as a decision region of the hypothesis h. 18, 29, 176, 207, 215, 217, 241

decision tree A decision tree is a flow-chart-like representation of a hypothesis map h. More formally, a decision tree is a directed graph containing a root node that reads in the feature vector $\mathbf{x}$ of a data point. The root node then forwards the data point to one of its children nodes based on some elementary test on the features $\mathbf{x}$. If the receiving child node is not a leaf node, i.e., it has itself children nodes, it represents another test. Based on the test result, the data point is forwarded to one of its descendants. This testing and forwarding of the data point is continued until the data point ends up in a leaf node (having no children nodes).

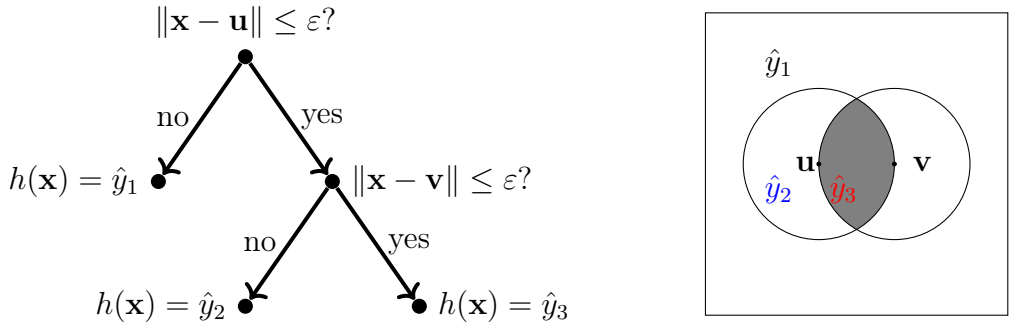


Figure 11.5: Left: A decision tree is a flow-chart-like representation of a piece-wise constant hypothesis $h : \mathcal{X} \to \mathbb{R}$. Each piece is a decision region $\mathcal{R}_{\hat{y}} := \big\{\mathbf{x} \in \mathcal{X} : h(\mathbf{x}) = \hat{y}\big\}$. The depicted decision tree can be applied to numeric feature vectors, i.e., $\mathcal{X} \subseteq \mathbb{R}^d$. It is parametrized by the threshold $\varepsilon > 0$ and the vectors $\mathbf{u}, \mathbf{v} \in \mathbb{R}^d$. Right: A decision tree partitions the feature space $\mathcal{X}$ into decision regions. Each decision region $\mathcal{R}_{\hat{y}} \subseteq \mathcal{X}$ corresponds to a specific leaf node in the decision tree.

deep net A deep net is an ANN with a (relatively) large number of hidden layers. Deep learning is an umbrella term for ML methods that use a deep net as their model [93]. 24, 29, 50, 224, 239, 246, 248

degree of belonging Degree of belonging is a number that indicates the extent to which a data point belongs to a cluster [6, Ch. 8]. The degree of belonging can be interpreted as a soft cluster assignment. Soft clustering methods can encode the degree of belonging by a real number in the interval $[0, 1]$. Hard clustering is obtained as the extreme case when the degree of belonging only takes on values 0 or 1. 264

denial-of-service attack A denial-of-service attack aims (e.g., via data poisoning) to steer the training of a model such that it performs poorly for typical data points. 193, 196, 198, 200

device Any physical system that can be used to store and process data. In the context of ML, we typically mean a computer that is able to read in data points from different sources and, in turn, to train an ML model using these data points. 1, 3, 19, 20, 29, 114, 122, 123, 150, 188, 192–194, 199, 225, 247

differentiable A real-valued function $f : \mathbb{R}^d \to \mathbb{R}$ is differentiable if it can, at any point, be approximated locally by a linear function. The local linear approximation at the point $\mathbf{x}$ is determined by the gradient $\nabla f(\mathbf{x})$ [2]. 11, 45, 63–65, 82, 94, 95, 230, 231, 233, 262, 266

differential privacy (DP) Consider some ML method $\mathcal{A}$ that reads in a dataset (e.g., the training set used for ERM) and delivers some output $\mathcal{A}(\mathcal{D})$. The output could be either the learned model parameters or the predictions for specific data points. DP is a precise measure of privacy leakage incurred by revealing the output. Roughly speaking, an ML method is differentially private if the probability distribution of the output $\mathcal{A}(\mathcal{D})$ does not change too much if the sensitive attribute of one data point in the training set is changed. Note that DP builds on a probabilistic model for an ML method, i.e., we interpret its output $\mathcal{A}(\mathcal{D})$ as the realization of an RV. The randomness in the output can be ensured by intentionally adding the realization of an auxiliary RV (noise) to the output of the ML method. 178–184, 251

discrepancy Consider an FL application with networked data represented by an FL network. FL methods use a discrepancy measure to compare hypothesis maps from local models at nodes i, i' connected by an edge in the FL network. 32, 36, 51, 133, 138, 139, 144–148, 150, 230

edge weight Each edge $\{i, i'\}$ of an FL network is assigned a non-negative edge weight $A_{i,i'} \geq 0$. A zero edge weight $A_{i,i'} = 0$ indicates the absence of an edge between nodes $i, i' \in \mathcal{V}$. 34, 36, 37, 43, 48, 57, 85, 87, 92, 120, 134, 138, 143, 147–149, 238

effective dimension The effective dimension $d_{\text{eff}}(\mathcal{H})$ of an infinite hypothesis space $\mathcal{H}$ is a measure of its size. Loosely speaking, the effective dimension is equal to the effective number of independent tunable model parameters. These parameters might be the coefficients used in a linear map or the weights and bias terms of an ANN. 17, 23, 42, 257

eigenvalue We refer to a number $\lambda \in \mathbb{R}$ as an eigenvalue of a square matrix $\mathbf{A} \in \mathbb{R}^{d \times d}$ if there is a non-zero vector $\mathbf{x} \in \mathbb{R}^d \setminus \{\mathbf{0}\}$ such that $\mathbf{A}\mathbf{x} = \lambda \mathbf{x}$. 10, 14, 18–20, 30, 38–41, 49, 55, 56, 68–71, 88, 89, 116, 117, 125, 126, 140–143, 148, 161–163, 209, 210, 219, 220

eigenvalue decomposition (EVD) The eigenvalue decomposition for a square matrix $\mathbf{A} \in \mathbb{R}^{d \times d}$ is a factorization of the form

$$\mathbf{A} = \mathbf{V} \mathbf{\Lambda} \mathbf{V}^{-1}.$$

The columns of the matrix $\mathbf{V} = \big(\mathbf{v}^{(1)}, \ldots, \mathbf{v}^{(d)}\big)$ are the eigenvectors of the matrix $\mathbf{V}$. The diagonal matrix $\mathbf{\Lambda} = \operatorname{diag}\big\{\lambda_1, \ldots, \lambda_d\big\}$ contains the

eigenvalues λ_j corresponding to the eigenvectors $\mathbf{v}^{(j)}$. Note that the above decomposition exists only if the matrix $\mathbf{A}$ is diagonalizable. 14, 20, 30, 39, 210

eigenvector An eigenvector of a matrix $\mathbf{A} \in \mathbb{R}^{d \times d}$ is a non-zero vector $\mathbf{x} \in \mathbb{R}^d \setminus \{\mathbf{0}\}$ such that $\mathbf{A}\mathbf{x} = \lambda \mathbf{x}$ with some eigenvalue λ. 14, 30, 38–41, 55, 219, 220

empirical risk The empirical risk $\widehat{L}\big(h|\mathcal{D}\big)$ of a hypothesis on a dataset $\mathcal{D}$ is the average loss incurred by h when applied to the data points in $\mathcal{D}$. 13, 17, 220, 228, 248, 265, 266

empirical risk minimization (ERM) Empirical risk minimization is the optimization problem of finding a hypothesis (out of a model) with the minimum average loss (or empirical risk) on a given dataset $\mathcal{D}$ (i.e., the training set). Many ML methods are obtained from empirical risk via specific design choices for the dataset, model, and loss [6, Ch. 3]. 6, 7, 11, 13, 14, 16, 17, 19, 21, 22, 24, 26, 30, 32, 47, 48, 51, 64, 115, 176, 184, 188, 199, 200, 203, 207, 209, 213, 218, 221, 228, 229, 240, 244, 247, 248, 253, 257, 265–267, 269

Erdős-Rényi (ER) graph An Erdős-Rényi (ER) graph is a probabilistic model for graphs defined over a given node set $i = 1, \ldots, n$. One way to define the ER graph is via collection of i.i.d. binary RVs $b^{(\{i,i'\})} \in \{0, 1\}$, for each pair of different nodes i, i'. A specific realization of an ER graph contains an edge $\{i, i'\}$ if and only if $b^{(\{i,i'\})} = 1$. The ER graph is parametrized by the number n of nodes and the edge probability $p(b^{(\{i,i'\})} = 1)$. 140, 141

estimation error Consider data points, each with feature vector $\mathbf{x}$ and label y. In some applications, we can model the relation between the feature vector and the label of a data point as $y = \bar{h}(\mathbf{x}) + \varepsilon$. Here, we use some true underlying hypothesis $\bar{h}$ and a noise term ε which summarizes any modeling or labeling errors. The estimation error incurred by an ML method that learns a hypothesis $\widehat{h}$, e.g., using ERM, is defined as $\widehat{h}(\mathbf{x}) - \bar{h}(\mathbf{x})$, for some feature vector. For a parametric hypothesis space, which consists of hypothesis maps determined by model parameters $\mathbf{w}$, we can define the estimation error as $\Delta\mathbf{w} = \widehat{\mathbf{w}} - \overline{\mathbf{w}}$ [90, 150]. 18–20, 161, 163, 164

Euclidean space The Euclidean space $\mathbb{R}^d$ of dimension $d \in \mathbb{N}$ consists of vectors $\mathbf{x} = \big(x_1, \ldots, x_d\big)$, with d real-valued entries $x_1, \ldots, x_d \in \mathbb{R}$. Such an Euclidean space is equipped with a geometric structure defined by the inner product $\mathbf{x}^T\mathbf{x}' = \sum_{j=1}^{d} x_j x'_j$ between any two vectors $\mathbf{x}, \mathbf{x}' \in \mathbb{R}^d$ [2]. 4, 37, 50, 79, 209, 224, 234, 248, 254, 256

expectation Consider a numeric feature vector $\mathbf{x} \in \mathbb{R}^d$ which we interpret as the realization of an RV with a probability distribution $p(\mathbf{x})$. The expectation of $\mathbf{x}$ is defined as the integral $\mathbb{E}\{\mathbf{x}\} := \int \mathbf{x} p(\mathbf{x})$ [2, 118, 119]. Note that the expectation is only defined if this integral exists, i.e., if the RV is integrable. 14, 245, 269

expert ML aims to learn a hypothesis h that accurately predicts the label of a data point based on its features. We measure the prediction error using some loss function. Ideally, we want to find a hypothesis that incurs minimal loss on any data point. We can make this informal goal

precise via the i.i.d. assumption and by using the Bayes risk as the baseline for the (average) loss of a hypothesis. An alternative approach to obtaining a baseline is to use the hypothesis h' learned by an existing ML method. We refer to this hypothesis h' as an expert [151]. Regret minimization methods learn a hypothesis that incurs a loss comparable to the best expert [151, 152]. 23, 257

explainability We define the (subjective) explainability of an ML method as the level of simulatability [112] of the predictions delivered by an ML system to a human user. Quantitative measures for the (subjective) explainability of a trained model can be constructed by comparing its predictions with the predictions provided by a user on a test set [112,114]. Alternatively, we can use probabilistic models for data and measure the explainability of a trained ML model via the conditional (differential) entropy of its predictions, given the user predictions [113,153]. 152–154, 157, 168, 169, 172, 227, 267

feature A feature of a data point is one of its properties that can be measured or computed easily without the need for human supervision. For example, if a data point is a digital image (e.g., stored as a `.jpeg` file), then we could use the red-green-blue intensities of its pixels as features. Domain-specific synonyms for the term feature are "covariate," "explanatory variable," "independent variable," "input (variable)," "predictor (variable)," or "regressor" [154], [155], [156]. 11, 12, 14–17, 20, 21, 23–25, 27, 28, 32, 35, 47, 52, 65, 68, 69, 71, 72, 120, 121, 128–132, 135, 145, 154–158, 161, 163, 164, 168, 172, 174, 176–179, 183–187, 189–191,

feature learning Consider an ML application with data points characterized by raw features $\mathbf{x} \in \mathcal{X}$. Feature learning refers to the task of learning a map

$$\mathbf{\Phi} : \mathcal{X} \rightarrow \mathcal{X}' : \mathbf{x} \mapsto \mathbf{x}',$$

that reads in raw features $\mathbf{x} \in \mathcal{X}$ of a data point and delivers new features $\mathbf{x}' \in \mathcal{X}'$ from a new feature space $\mathcal{X}'$. Different feature learning methods are obtained for different design choices of $\mathcal{X}, \mathcal{X}'$, for a hypothesis space $\mathcal{H}$ of potential maps $\mathbf{\Phi}$, and for a quantitative measure of the usefulness of a specific $\mathbf{\Phi} \in \mathcal{H}$. For example, principal component analysis (PCA) uses $\mathcal{X} := \mathbb{R}^d$, $\mathcal{X}' := \mathbb{R}^{d'}$ with $d' < d$, and a hypothesis space

$$\mathcal{H} := \big\{ \mathbf{\Phi} : \mathbb{R}^d \rightarrow \mathbb{R}^{d'} : \mathbf{x}' := \mathbf{F}\mathbf{x} \text{ with some } \mathbf{F} \in \mathbb{R}^{d' \times d} \big\}.$$

PCA measures the usefulness of a specific map $\mathbf{\Phi}(\mathbf{x}) = \mathbf{F}\mathbf{x}$ by the minimum linear reconstruction error incurred on a dataset,

$$\min_{\mathbf{G} \in \mathbb{R}^{d \times d'}} \sum_{r=1}^{m} \big\| \mathbf{G}\mathbf{F}\mathbf{x}^{(r)} - \mathbf{x}^{(r)} \big\|_2^2 .$$

29

feature map Feature map refers to a map that transforms the original features of a data point into new features. The so-obtained new features might be preferable over the original features for several reasons. For example, the arrangement of data points might become simpler (or more linear) in the new feature space, allowing the use of linear models

in the new features. This idea is a main driver for the development of kernel methods [35]. Moreover, the hidden layers of a deep net can be interpreted as a trainable feature map followed by a linear model in the form of the output layer. Another reason for learning a feature map could be that learning a small number of new features helps to avoid overfitting and ensures interpretability [157]. The special case of a feature map delivering two numeric features is particularly useful for data visualization. Indeed, we can depict data points in a scatterplot by using two features as the coordinates of a data point. 18, 29, 185, 186, 250

feature matrix Consider a dataset $\mathcal{D}$ with m data points with feature vectors $\mathbf{x}^{(1)}, \ldots, \mathbf{x}^{(m)} \in \mathbb{R}^d$. It is convenient to collect the individual feature vectors into a feature matrix $\mathbf{X} := \big(\mathbf{x}^{(1)}, \ldots, \mathbf{x}^{(m)}\big)^T$ of size $m \times d$. 14, 18, 20, 35, 47, 126, 172, 190, 209

feature space The feature space of a given ML application or method is constituted by all potential values that the feature vector of a data point can take on. A widely used choice for the feature space is the Euclidean space $\mathbb{R}^d$, with the dimension d being the number of individual features of a data point. 15, 18, 212, 215, 217, 223, 234–236

feature vector Feature vector refers to a vector $\mathbf{x} = \big(x_1, \ldots, x_d\big)^T$ whose entries are individual features $x_1, \ldots, x_d$. Many ML methods use feature vectors that belong to some finite-dimensional Euclidean space $\mathbb{R}^d$. For some ML methods, however, it can be more convenient to work with feature vectors that belong to an infinite-dimensional vector space (e.g.,

parameters. In contrast to FedAvg, which uses SGD to train local models, FedProx uses a proximal operator for the training [66]. 85, 86

Finnish Meteorological Institute (FMI) The FMI is a government agency responsible for gathering and reporting weather data in Finland. 1, 9, 10, 12, 43, 188, 200, 262

Gaussian mixture model (GMM) A GMM is a particular type of probabilistic model for a numeric vector $\mathbf{x}$ (e.g., the features of a data point). Within a GMM, the vector $\mathbf{x}$ is drawn from a randomly selected multivariate normal distribution $p^{(c)} = \mathcal{N}\left(\boldsymbol{\mu}^{(c)}, \mathbf{C}^{(c)}\right)$ with $c = I$. The index $I \in \{1, \ldots, k\}$ is an RV with probabilities $p(I = c) = p_c$. Note that a GMM is parametrized by the probability p_c, the mean vector $\boldsymbol{\mu}^{(c)}$, and the covariance matrix $\boldsymbol{\Sigma}^{(\mathbf{c})}$ for each $c = 1, \ldots, k$. GMMs are widely used for clustering, density estimation, and as a generative model. 124, 208, 264

Gaussian random variable (Gaussian RV) A standard Gaussian RV is a real-valued RV x with probability density function (pdf) [5, 139, 158]

$$p(x) = \frac{1}{\sqrt{2\pi}} \exp^{-x^2/2}.$$

Given a standard Gaussian RV x, we can construct a general Gaussian RV x' with mean μ and variance σ^2 via $x' := \sigma(x + \mu)$. The probability distribution of a Gaussian RV is referred to as normal distribution, denoted $\mathcal{N}(\mu, \sigma)$.

A Gaussian random vector $\mathbf{x} \in \mathbb{R}^d$ with covariance matrix $\mathbf{C}$ and mean $\boldsymbol{\mu}$ can be constructed via $\mathbf{x} := \mathbf{A}\big(\mathbf{z} + \boldsymbol{\mu}\big)$. Here, $\mathbf{A}$ is any matrix that

satisfies $\mathbf{A}\mathbf{A}^T = \mathbf{C}$ and $\mathbf{z} := \big(z_1, \ldots, z_d\big)^T$ is a vector whose entries are i.i.d. standard Gaussian RVs $z_1, \ldots, z_d$. Gaussian random processes generalize Gaussian random vectors by applying linear transformations to infinite sequences of standard Gaussian RVs [159].

Gaussian RVs are widely used probabilistic models for the statistical analysis of ML methods. Their significance arises partly from the central limit theorem, which states that the average of an increasing number of independent RVs (not necessarily Gaussian themselves) converges to a Gaussian RV [160]. 14, 258

general data protection regulation (GDPR) The GDPR was enacted by the European Union (EU), effective from May 25, 2018 [103]. It safeguards the privacy and data rights of individuals in the EU. The GDPR has significant implications for how data is collected, stored, and used in ML applications. Key provisions include the following:

- Data minimization principle: ML systems should only use the necessary amount of personal data for their purpose.
- Transparency and explainability: ML systems should enable their users to understand how the systems make decisions that impact the users.
- Data subject rights: Users should get an opportunity to access, rectify, and delete their personal data, as well as to object to automated decision-making and profiling.
- Accountability: Organizations must ensure robust data security and demonstrate compliance through documentation and regular

audits.

156

generalization Many current ML (and AI) systems are based on ERM: At their core, they train a model (i.e., learn a hypothesis $\hat{h} \in \mathcal{H}$) by minimizing the average loss (or empirical risk) on some data points $\mathbf{z}^{(1)}, \ldots, \mathbf{z}^{(m)}$, which serve as a training set $\mathcal{D}^{(\text{train})}$. Generalization refers to an ML method's ability to perform well outside the training set. Any mathematical theory of generalization needs some mathematical concept for the "outside the training set." For example, statistical learning theory uses a probabilistic model such as the i.i.d. assumption for data generation: the data points in the training set are i.i.d. realizations of some underlying probability distribution $p(\mathbf{z})$. A probabilistic model allows us to explore the outside of the training set by drawing additional i.i.d. realizations from $p(\mathbf{z})$. Moreover, using the i.i.d. assumption allows us to define the risk of a trained model $\hat{h} \in \mathcal{H}$ as the expected loss $\bar{L}\big(\hat{h}\big)$. What is more, we can use concentration bounds or convergence results for sequences of i.i.d. RVs to bound the deviation between the empirical risk $\widehat{L}\big(\hat{h}|\mathcal{D}^{(\text{train})}\big)$ of a trained model and its risk [29]. It is possible to study generalization also without using probabilistic models. For example, we could use (deterministic) perturbations of the data points in the training set to study its outside. In general, we would like the trained model to be robust, i.e., its predictions should not change too much for small perturbations of a data point. Consider a trained model for detecting an object in a smartphone snapshot. The detection

result should not change if we mask a small number of randomly chosen pixels in the image [161].

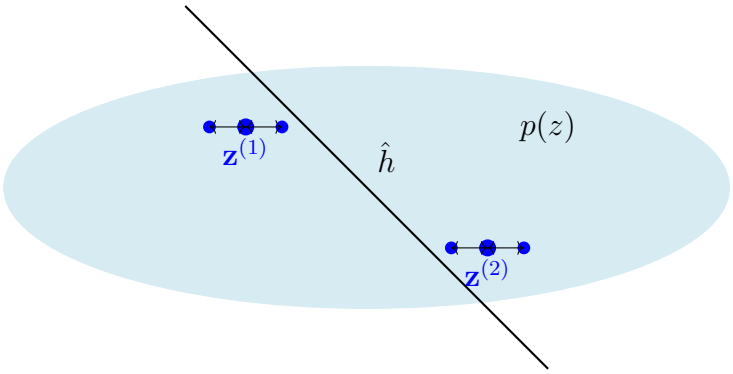


Figure 11.6: Two data points $\mathbf{z}^{(1)}, \mathbf{z}^{(2)}$ that are used as a training set to learn a hypothesis $\hat{h}$ via ERM. We can evaluate $\hat{h}$ outside $\mathcal{D}^{(\text{train})}$ either by an i.i.d. assumption with some underlying probability distribution $p(\mathbf{z})$ or by perturbing the data points.

vector $\mathbf{w}^{(k+1)}$ that has a smaller objective value $f(\mathbf{w}^{(k+1)}) < f\big(\mathbf{w}^{(k)}\big)$ than the current guess $\mathbf{w}^{(k)}$. We can achieve this typically by using a gradient step

$$\mathbf{w}^{(k+1)} = \mathbf{w}^{(k)} - \eta \nabla f(\mathbf{w}^{(k)}) \tag{198}$$

with a sufficiently small step size $\eta > 0$. Figure 11.7 illustrates the effect of a single gradient descent step (198).

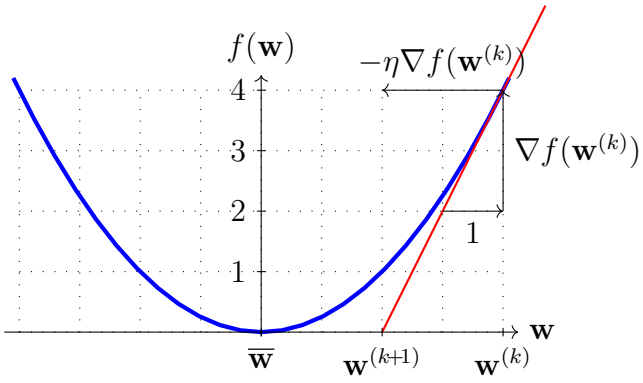


Figure 11.7: A single gradient step (198) towards the minimizer $\overline{\mathbf{w}}$ of $f(\mathbf{w})$.

gradient step Given a differentiable real-valued function $f(\cdot) : \mathbb{R}^d \to \mathbb{R}$ and a vector $\mathbf{w} \in \mathbb{R}^d$, the gradient step updates $\mathbf{w}$ by adding the scaled negative gradient $\nabla f(\mathbf{w})$ to obtain the new vector (see Figure 11.8)

$$\widehat{\mathbf{w}} := \mathbf{w} - \eta \nabla f(\mathbf{w}). \tag{199}$$

Mathematically, the gradient step is a (typically non-linear) operator $\mathcal{T}^{(f,\eta)}$ that is parametrized by the function f and the step size η.

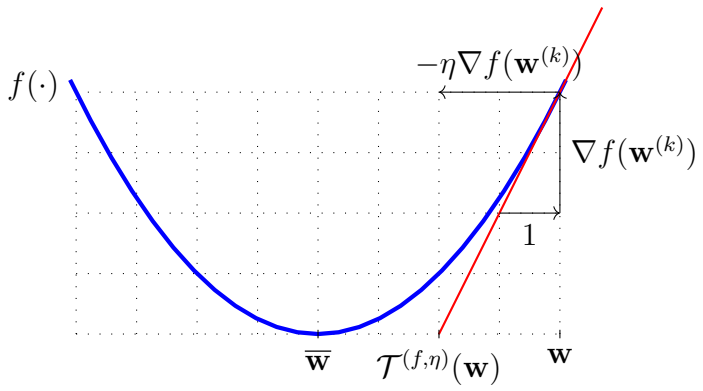


Figure 11.8: The basic gradient step (199) maps a given vector $\mathbf{w}$ to the updated vector $\mathbf{w}'$. It defines an operator $\mathcal{T}^{(f,\eta)}(\cdot) : \mathbb{R}^d \rightarrow \mathbb{R}^d : \mathbf{w} \mapsto \widehat{\mathbf{w}}$.

Note that the gradient step (199) optimizes locally - in a neighborhood whose size is determined by the step size η - a linear approximation to the function $f(\cdot)$. A natural generalization of (199) is to locally optimize the function itself - instead of its linear approximation - such that

$$\widehat{\mathbf{w}} = \operatorname*{argmin}_{\mathbf{w}' \in \mathbb{R}^d} f(\mathbf{w}') + (1/\eta) \left\| \mathbf{w} - \mathbf{w}' \right\|_2^2 . \tag{200}$$

We intentionally use the same symbol η for the parameter in (200) as we used for the step size in (199). The larger the η we choose in (200), the more progress the update will make towards reducing the function value $f(\widehat{\mathbf{w}})$. Note that, much like the gradient step (199), also the update (200) defines a (typically non-linear) operator that is parametrized by the function $f(\cdot)$ and the parameter η. For a convex function $f(\cdot)$, this operator is known as the proximal operator of $f(\cdot)$ [42]. 4, 16, 47, 62,

64–69, 71–78, 80–82, 84–86, 88, 90–93, 95, 99–101, 103, 104, 111, 139, 142, 202, 209, 231, 253–255, 263

gradient-based methods Gradient-based methods are iterative techniques for finding the minimum (or maximum) of a differentiable objective function of the model parameters. These methods construct a sequence of approximations to an optimal choice for model parameters that results in a minimum (or maximum) value of the objective function. As their name indicates, gradient-based methods use the gradients of the objective function evaluated during previous iterations to construct new, (hopefully) improved model parameters. One important example of a gradient-based method is GD. 4, 7, 16, 37, 45, 47, 62–65, 69, 75, 76, 78, 84–86, 88–90, 114, 138, 150, 152, 155, 158, 162, 164, 209, 263

graph A graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ is a pair that consists of a node set $\mathcal{V}$ and an edge set $\mathcal{E}$. In its most general form, a graph is specified by a map that assigns each edge $e \in \mathcal{E}$ a pair of nodes [47]. One important family of graphs is simple undirected graphs. A simple undirected graph is obtained by identifying each edge $e \in \mathcal{E}$ with two different nodes $\{i, i'\}$. Weighted graphs also specify numeric weights A_e for each edge $e \in \mathcal{E}$. 19, 38, 43, 89, 139, 148, 209, 216, 220, 225, 230, 238, 247

hard clustering Hard clustering refers to the task of partitioning a given set of data points into (a few) non-overlapping clusters. The most widely used hard clustering method is k-means. 201, 217

Hilbert space A Hilbert space is a linear vector space equipped with an

inner product between pairs of vectors. One important example of a Hilbert space is the Euclidean space $\mathbb{R}^d$, for some dimension d, which consists of Euclidean vectors $\mathbf{u} = \big(u_1, \ldots, u_d\big)^T$ along with the inner product $\mathbf{u}^T\mathbf{v}$. 236

horizontal federated learning (horizontal FL) Horizontal FL uses local datasets constituted by different data points but uses the same features to characterize them [74]. For example, weather forecasting uses a network of spatially distributed weather (observation) stations. Each weather station measures the same quantities, such as daily temperature, air pressure, and precipitation. However, different weather stations measure the characteristics or features of different spatiotemporal regions. Each spatiotemporal region represents an individual data point, each characterized by the same features (e.g., daily temperature or air pressure). 98, 120, 121, 128–130

hypothesis A hypothesis refers to a map (or function) $h : \mathcal{X} \to \mathcal{Y}$ from the feature space $\mathcal{X}$ to the label space $\mathcal{Y}$. Given a data point with features $\mathbf{x}$, we use a hypothesis map h to estimate (or approximate) the label y using the prediction $\hat{y} = h(\mathbf{x})$. ML is all about learning (or finding) a hypothesis map h such that $y \approx h(\mathbf{x})$ for any data point (having features $\mathbf{x}$ and label y). 11–18, 20–22, 24–28, 31, 33, 36, 43, 50, 65, 75–78, 86, 114, 115, 154, 157, 168, 169, 176, 184, 188, 189, 196–198, 200, 201, 203, 204, 206, 207, 215–217, 219–222, 225, 228–230, 234, 235, 239, 241–245, 247–250, 257, 260–262, 264, 266, 267, 269

hypothesis space Every practical ML method uses a hypothesis space (or

model) $\mathcal{H}$. The hypothesis space of an ML method is a subset of all possible maps from the feature space to the label space. The design choice of the hypothesis space should take into account available computational resources and statistical aspects. If the computational infrastructure allows for efficient matrix operations, and there is an (approximately) linear relation between a set of features and a label, a useful choice for the hypothesis space might be the linear model. 5, 11, 17, 23, 24, 26, 51, 204, 206, 219, 221, 223, 241, 242, 244, 245, 249, 260, 267, 269, 270

independent and identically distributed (i.i.d.) It can be useful to interpret data points $\mathbf{z}^{(1)}, \ldots, \mathbf{z}^{(m)}$ as realizations of i.i.d. RVs with a common probability distribution. If these RVs are continuous-valued, their joint pdf is $p\big(\mathbf{z}^{(1)}, \ldots, \mathbf{z}^{(m)}\big) = \prod_{r=1}^{m} p\big(\mathbf{z}^{(r)}\big)$, with $p(\mathbf{z})$ being the common marginal pdf of the underlying RVs. 3, 17, 20, 24, 25, 27, 30, 47, 51, 58, 77, 83, 124, 127, 140, 144, 145, 155, 172, 182, 188, 206, 220, 227, 228, 235, 239, 245, 252, 261, 264

independent and identically distributed assumption (i.i.d. assumption) The i.i.d. assumption interprets data points of a dataset as the realizations of i.i.d. RVs. 17, 25, 47, 48, 120, 154, 184, 222, 228, 229, 261

interpretability An ML method is interpretable for a specific user if they can well anticipate the predictions delivered by the method. The notion of interpretability can be made precise using quantitative measures of the uncertainty about the predictions [113]. 224, 241

kernel Consider data points characterized by a feature vector $\mathbf{x} \in \mathcal{X}$ with a generic feature space $\mathcal{X}$. A (real-valued) kernel $K : \mathcal{X} \times \mathcal{X} \rightarrow \mathbb{R}$ assigns each pair of feature vectors $\mathbf{x}, \mathbf{x}' \in \mathcal{X}$ a real number $K\big(\mathbf{x}, \mathbf{x}'\big)$. The value $K\big(\mathbf{x}, \mathbf{x}'\big)$ is often interpreted as a measure for the similarity between $\mathbf{x}$ and $\mathbf{x}'$. Kernel methods use a kernel to transform the feature vector $\mathbf{x}$ to a new feature vector $\mathbf{z} = K\big(\mathbf{x}, \cdot\big)$. This new feature vector belongs to a linear feature space $\mathcal{X}'$ which is (in general) different from the original feature space $\mathcal{X}$. The feature space $\mathcal{X}'$ has a specific mathematical structure, i.e., it is a reproducing kernel Hilbert space [35, 162]. 18, 236, 250

kernel method A kernel method is an ML method that uses a kernel K to map the original (raw) feature vector $\mathbf{x}$ of a data point to a new (transformed) feature vector $\mathbf{z} = K\big(\mathbf{x}, \cdot\big)$ [35, 162]. The motivation for transforming the feature vectors is that, by using a suitable kernel, the data points have a "more pleasant" geometry in the transformed feature space. For example, in a binary classification problem, using transformed feature vectors $\mathbf{z}$ might allow us to use linear models, even if the data points are not linearly separable in the original feature space (see Figure 11.9).

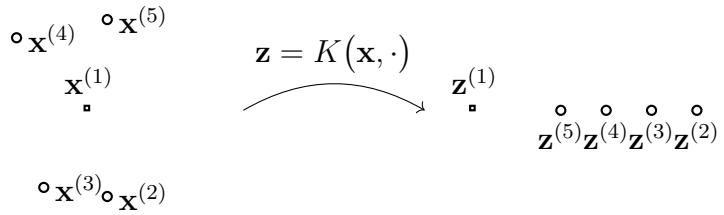


Figure 11.9: Five data points characterized by feature vectors $\mathbf{x}^{(r)}$ and labels $y^{(r)} \in \{\circ, \square\}$, for $r = 1, \ldots, 5$. With these feature vectors, there is no way to separate the two classes by a straight line (representing the decision boundary of a linear classifier). In contrast, the transformed feature vectors $\mathbf{z}^{(r)} = K\big(\mathbf{x}^{(r)}, \cdot\big)$ allow us to separate the data points using a linear classifier.

29, 224, 225, 236

label A higher-level fact or quantity of interest associated with a data point. For example, if the data point is an image, the label could indicate whether the image contains a cat or not. Synonyms for label, commonly used in specific domains, include "response variable," "output variable," and "target" [154], [155], [156]. 11, 12, 14, 16–18, 20, 24, 27, 30, 32, 35, 47, 65, 71, 91, 114, 128, 130, 131, 134, 145, 153, 155, 156, 158, 160, 161, 172, 175, 176, 178, 184, 188, 189, 195, 200, 201, 203–207, 211–214, 216, 221, 234, 235, 237–244, 249–251, 257, 259, 261, 262, 264, 267, 271

label space Consider an ML application that involves data points characterized by features and labels. The label space is constituted by all potential values that the label of a data point can take on. Regression methods, aiming at predicting numeric labels, often use the label

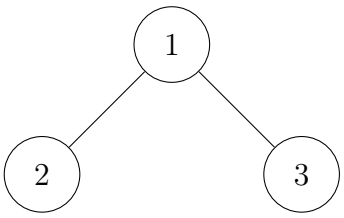
space $\mathcal{Y} = \mathbb{R}$. Binary classification methods use a label space that consists of two different elements, e.g., $\mathcal{Y} = \{-1, 1\}$, $\mathcal{Y} = \{0, 1\}$, or $\mathcal{Y} = \{$"cat image","no cat image"$\}$. 16, 134, 145, 183, 188, 201, 207, 215, 234, 235, 240, 241, 243

labeled datapoint A data point whose label is known or has been determined by some means which might require human labor. 239, 241, 243, 250, 261, 262

Laplacian matrix The structure of a graph $\mathcal{G}$, with nodes $i = 1, \ldots, n$, can be analyzed using the properties of special matrices that are associated with $\mathcal{G}$. One such matrix is the graph Laplacian matrix $\mathbf{L}^{(\mathcal{G})} \in \mathbb{R}^{n \times n}$, which is defined for an undirected and weighted graph [163, 164]. It is defined element-wise as (see Figure 11.10)

$$L_{i,i'}^{(\mathcal{G})} := \begin{cases} -A_{i,i'} & \text{for } i \neq i', \{i, i'\} \in \mathcal{E}, \\ \sum_{i'' \neq i} A_{i,i''} & \text{for } i = i', \\ 0 & \text{else.} \end{cases} \tag{201}$$

Here, $A_{i,i'}$ denotes the edge weight of an edge $\{i, i'\} \in \mathcal{E}$.



$$\mathbf{L}^{(\mathcal{G})} = \begin{pmatrix} 2 & -1 & -1 \\ -1 & 1 & 0 \\ -1 & 0 & 1 \end{pmatrix}$$

Figure 11.10: Left: Some undirected graph $\mathcal{G}$ with three nodes $i = 1, 2, 3$. Right: The Laplacian matrix $\mathbf{L}^{(\mathcal{G})} \in \mathbb{R}^{3 \times 3}$ of $\mathcal{G}$.

large language model (LLM) Large language models is an umbrella term for ML methods that process and generate human-like text. These methods typically use deep nets with billions (or even trillions) of parameters. A widely used choice for the network architecture is referred to as Transformers [165]. The training of large language models is often based on the task of predicting a few words that are intentionally removed from a large text corpus. Thus, we can construct labeled datapoints simply by selecting some words of a text as labels and the remaining words as features of data points. This construction requires very little human supervision and allows for generating sufficiently large training sets for large language models. 3

law of large numbers The law of large numbers refers to the convergence of the average of an increasing (large) number of i.i.d. RVs to the mean of their common probability distribution. Different instances of the law of large numbers are obtained by using different notions of convergence [158]. 24, 77

learning rate Consider an iterative ML method for finding or learning a useful hypothesis $h \in \mathcal{H}$. Such an iterative method repeats similar computational (update) steps that adjust or modify the current hypothesis to obtain an improved hypothesis. One well-known example of such an iterative learning method is GD and its variants, SGD and projected GD. A key parameter of an iterative method is the learning rate. The learning rate controls the extent to which the current hypothesis can

be modified during a single iteration. A well-known example of such a parameter is the step size used in GD [6, Ch. 5]. 16, 62, 64–71, 73, 74, 78, 80, 82–84, 89, 90, 92, 94, 96, 97, 101, 104, 114, 115, 167, 264

learning task Consider a dataset $\mathcal{D}$ constituted by several data points, each of them characterized by features $\mathbf{x}$. For example, the dataset $\mathcal{D}$ might be constituted by the images of a particular database. Sometimes it might be useful to represent a dataset $\mathcal{D}$, along with the choice of features, by a probability distribution $p(\mathbf{x})$. A learning task associated with $\mathcal{D}$ consists of a specific choice for the label of a data point and the corresponding label space. Given a choice for the loss function and model, a learning task gives rise to an instance of ERM. Thus, we could define a learning task also via an instance of ERM, i.e., via an objective function. Note that, for the same dataset, we obtain different learning tasks by using different choices for the features and label of a data point. These learning tasks are related, as they are based on the same dataset, and solving them jointly (via multitask learning methods) is typically preferable over solving them separately [166], [167], [168]. 3, 18, 19, 147, 246

least absolute shrinkage and selection operator (Lasso) The Lasso is an instance of structural risk minimization (SRM). It learns the weights $\mathbf{w}$ of a linear map $h(\mathbf{x}) = \mathbf{w}^T\mathbf{x}$ based on a training set. Lasso is obtained from linear regression by adding the scaled ℓ_1-norm $\alpha \left\|\mathbf{w}\right\|_1$ to the average squared error loss incurred on the training set. 260

linear classifier Consider data points characterized by numeric features

$\mathbf{x} \in \mathbb{R}^d$ and a label $y \in \mathcal{Y}$ from some finite label space $\mathcal{Y}$. A linear classifier is characterized by having decision regions that are separated by hyperplanes in $\mathbb{R}^d$ [6, Ch. 2]. 237

linear model Consider data points, each characterized by a numeric feature vector $\mathbf{x} \in \mathbb{R}^d$. A linear model is a hypothesis space which consists of all linear maps,

$$\mathcal{H}^{(d)} := \left\{ h(\mathbf{x}) = \mathbf{w}^T \mathbf{x} : \mathbf{w} \in \mathbb{R}^d \right\}. \tag{202}$$

Note that (202) defines an entire family of hypothesis spaces, which is parametrized by the number d of features that are linearly combined to form the prediction $h(\mathbf{x})$. The design choice of d is guided by computational aspects (e.g., reducing d means less computation), statistical aspects (e.g., increasing d might reduce prediction error), and interpretability. A linear model using few carefully chosen features tends to be considered more interpretable [157, 169]. 13, 16–18, 23, 24, 28–30, 43, 44, 46, 48, 63, 71, 77, 83, 85–87, 90, 92, 94–98, 101, 115, 126, 134, 135, 139, 154, 159, 160, 167–169, 172, 191, 200, 202, 223, 224, 235, 236, 248, 249, 270

linear regression Linear regression aims to learn a linear hypothesis map to predict a numeric label based on the numeric features of a data point. The quality of a linear hypothesis map is measured using the average squared error loss incurred on a set of labeled datapoints, which we refer to as the training set. 9, 13–15, 18, 19, 27, 28, 30, 31, 44, 63, 68, 71, 83, 108, 134, 135, 191, 209, 240, 257, 258

local dataset The concept of a local dataset is in between the concept of a data point and a dataset. A local dataset consists of several individual data points, which are characterized by features and labels. In contrast to a single dataset used in basic ML methods, a local dataset is also related to other local datasets via different notions of similarity. These similarities might arise from probabilistic models or communication infrastructure and are encoded in the edges of an FL network. 1–3, 5, 7, 8, 19, 20, 32–36, 42, 43, 47–50, 56–59, 83, 85, 87, 89, 91, 95, 98–101, 104, 114, 115, 120–124, 126–132, 138, 139, 144–147, 150, 152, 154–157, 159–161, 163, 164, 169, 172–175, 178, 181, 188, 192, 194–198, 200, 208, 225, 234, 242, 247, 270

local model Consider a collection of local datasets that are assigned to the nodes of an FL network. A local model $\mathcal{H}^{(i)}$ is a hypothesis space assigned to a node $i \in \mathcal{V}$. Different nodes might be assigned different hypothesis spaces, i.e., in general $\mathcal{H}^{(i)} \neq \mathcal{H}^{(i')}$ for different nodes $i, i' \in \mathcal{V}$. 3, 6, 19, 29, 32–37, 41–43, 45, 47, 50–52, 56, 58, 61, 73, 79, 85, 87, 88, 90–92, 94, 95, 97, 98, 100, 108–111, 114, 120–122, 125, 126, 131–133, 136, 146, 154, 156, 157, 167, 172, 193, 195, 199, 219, 225, 226, 230

logistic loss Consider a data point characterized by the features $\mathbf{x}$ and a binary label $y \in \{-1, 1\}$. We use a real-valued hypothesis h to predict the label y from the features $\mathbf{x}$. The logistic loss incurred by this prediction is defined as

$$L\left((\mathbf{x}, y), h\right) := \log(1 + \exp(-yh(\mathbf{x}))). \tag{203}$$

Carefully note that the expression (203) for the logistic loss applies only

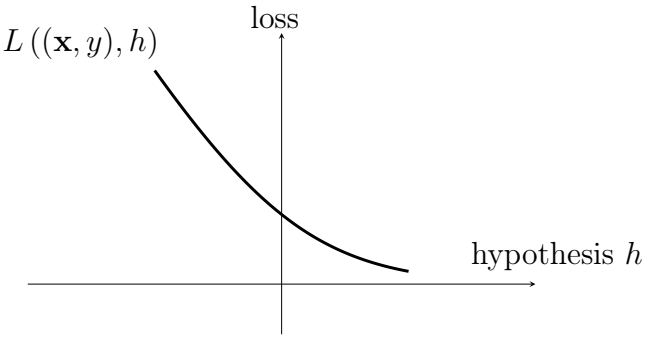


Figure 11.11: Some loss function $L\left((\mathbf{x}, y), h\right)$ for a fixed data point, with feature vector $\mathbf{x}$ and label y, and a varying hypothesis h. ML methods try to find (or learn) a hypothesis that incurs minimal loss.

machine learning (ML) ML aims to predict a label from the features of a data point. ML methods achieve this by learning a hypothesis from a hypothesis space (or model) through the minimization of a loss function [6, 170]. One precise formulation of this principle is ERM. Different ML methods are obtained from different design choices for data points (their features and label), model, and loss function [6, Ch. 3].

maximum The maximum of a set $\mathcal{A} \subseteq \mathbb{R}$ of real numbers is the greatest element in that set, if such an element exists. A set $\mathcal{A}$ has a maximum if

$\mathbf{x}$. This prediction could be obtained by a hypothesis learned by an ERM-based ML method. 184, 185, 251

neighborhood The neighborhood of a node $i \in \mathcal{V}$ is the subset of nodes constituted by the neighbors of i. 19, 38, 151, 216, 232

neighbors The neighbors of a node $i \in \mathcal{V}$ within an FL network are those nodes $i' \in \mathcal{V} \setminus \{i\}$ that are connected (via an edge) to node i. 59, 92–94, 96, 97, 107, 147, 151, 167, 193, 194, 247

networked data Networked data consists of local datasets that are related by some notion of pairwise similarity. We can represent networked data using a graph whose nodes carry local datasets and edges encode pairwise similarities. One example of networked data arises in FL applications where local datasets are generated by spatially distributed devices. 219

node degree The degree $d^{(i)}$ of a node $i \in \mathcal{V}$ in an undirected graph is the number of its neighbors, i.e., $d^{(i)} := \left|\mathcal{N}^{(i)}\right|$. 38, 89, 126, 139–143, 148, 149

norm A norm is a function that maps each (vector) element of a linear vector space to a non-negative real number. This function must be homogeneous and definite, and it must satisfy the triangle inequality [172]. 11, 12, 240, 248, 261

objective function An objective function is a map that assigns each value of an optimization variable, such as the model parameters $\mathbf{w}$ of a

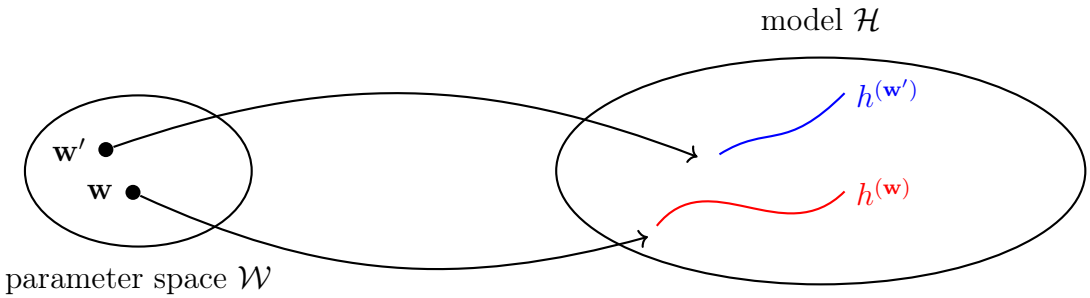


Figure 11.12: The parameter space $\mathcal{W}$ of an ML model $\mathcal{H}$ consists of all feasible choices for the model parameters. Each choice $\mathbf{w}$ for the model parameters selects a hypothesis map $h^{(\mathbf{w})} \in \mathcal{H}$.

253

parameters The parameters of an ML model are tunable (i.e., learnable or adjustable) quantities that allow us to choose between different hypothesis maps. For example, the linear model $\mathcal{H} := \{h^{(\mathbf{w})} : h^{(\mathbf{w})}(x) = w_1 x + w_2\}$ consists of all hypothesis maps $h^{(\mathbf{w})}(x) = w_1 x + w_2$ with a particular choice for the parameters $\mathbf{w} = \big(w_1, w_2\big)^T \in \mathbb{R}^2$. Another example of parameters is the weights assigned to the connections between neurons of an ANN. 13, 16–18, 26, 28, 41–44, 47, 48, 61, 73, 76, 85, 87, 88, 90–92, 94, 96–98, 100, 122, 125, 126, 131, 133, 136, 144, 145, 167, 219, 239, 245, 251

polynomial regression Polynomial regression aims at learning a polynomial hypothesis map to predict a numeric label based on the numeric features of a data point. For data points characterized by a single numeric feature, polynomial regression uses the hypothesis space

$\mathcal{H}_d^{(\text{poly})} := \{h(x) = \sum_{j=0}^{d-1} x^j w_j\}$. The quality of a polynomial hypothesis map is measured using the average squared error loss incurred on a set of labeled datapoints (which we refer to as the training set). 24

positive semi-definite (psd) A (real-valued) symmetric matrix $\mathbf{Q} = \mathbf{Q}^T \in \mathbb{R}^{d \times d}$ is referred to as psd if $\mathbf{x}^T \mathbf{Q} \mathbf{x} \geq 0$ for every vector $\mathbf{x} \in \mathbb{R}^d$. The property of being psd can be extended from matrices to (real-valued) symmetric kernel maps $K : \mathcal{X} \times \mathcal{X} \to \mathbb{R}$ (with $K(\mathbf{x}, \mathbf{x}') = K(\mathbf{x}', \mathbf{x})$) as follows: For any finite set of feature vectors $\mathbf{x}^{(1)}, \ldots, \mathbf{x}^{(m)}$, the resulting matrix $\mathbf{Q} \in \mathbb{R}^{m \times m}$ with entries $Q_{r,r'} = K\big(\mathbf{x}^{(r)}, \mathbf{x}^{(r')}\big)$ is psd [35]. 4, 10, 14, 18, 30, 38, 39, 44, 63, 68, 80, 210

prediction A prediction is an estimate or approximation for some quantity of interest. ML revolves around learning or finding a hypothesis map h that reads in the features $\mathbf{x}$ of a data point and delivers a prediction $\widehat{y} := h(\mathbf{x})$ for its label y. 11, 12, 16, 17, 29, 36, 50, 51, 57, 77, 91, 154, 156, 168, 169, 184, 195–198, 201, 203, 207, 218, 221, 222, 228, 234, 235, 241–243, 247, 250, 251, 260, 261, 264, 267, 270, 271

principal component analysis (PCA) PCA determines a linear feature map such that the new features allow us to reconstruct the original features with the minimum reconstruction error [6]. 223

privacy leakage Consider an ML application that processes a dataset $\mathcal{D}$ and delivers some output, such as the predictions obtained for new data points. Privacy leakage arises if the output carries information about a private (or sensitive) feature of a data point (which might be a human) of $\mathcal{D}$. Based on a probabilistic model for the data generation, we can

measure the privacy leakage via the MI between the output and the senstive feature. Another quantitative measure of privacy leakage is DP. The relations between different measures of privacy leakage have been studied in the literature (see [173]). 173, 174, 188, 218

privacy protection Consider some ML method $\mathcal{A}$ that reads in a dataset $\mathcal{D}$ and delivers some output $\mathcal{A}(\mathcal{D})$. The output could be the learned model parameters $\widehat{\mathbf{w}}$ or the prediction $\hat{h}(\mathbf{x})$ obtained for a specific data point with features $\mathbf{x}$. Many important ML applications involve data points representing humans. Each data point is characterized by features $\mathbf{x}$, potentially a label y, and a sensitive attribute s (e.g., a recent medical diagnosis). Roughly speaking, privacy protection means that it should be impossible to infer, from the output $\mathcal{A}(\mathcal{D})$, any of the sensitive attributes of data points in $\mathcal{D}$. Mathematically, privacy protection requires non-invertibility of the map $\mathcal{A}(\mathcal{D})$. In general, just making $\mathcal{A}(\mathcal{D})$ non-invertible is typically insufficient for privacy protection. We need to make $\mathcal{A}(\mathcal{D})$ sufficiently non-invertible. 153, 174

probabilistic model A probabilistic model interprets data points as realizations of RVs with a joint probability distribution. This joint probability distribution typically involves parameters which have to be manually chosen or learned via statistical inference methods such as maximum likelihood estimation [30]. 17, 18, 20, 21, 28, 47, 48, 58, 97, 127, 139, 144–146, 157, 161, 184, 188, 204–206, 218, 220, 222, 226–228, 242, 250, 264

probability We assign a probability value, typically chosen in the interval

$[0,1]$, to each event that might occur in a random experiment [5, 118, 119, 174]. 252, 253, 256, 264

probability density function (pdf) The probability density function $p(x)$ of a real-valued RV $x \in \mathbb{R}$ is a particular representation of its probability distribution. If the probability density function exists, it can be used to compute the probability that x takes on a value from a (measurable) set $\mathcal{B} \subseteq \mathbb{R}$ via $p(x \in \mathcal{B}) = \int_{\mathcal{B}} p(x')dx'$ [5, Ch. 3]. The probability density function of a vector-valued RV $\mathbf{x} \in \mathbb{R}^d$ (if it exists) allows us to compute the probability of $\mathbf{x}$ belonging to a (measurable) region $\mathcal{R}$ via $p(\mathbf{x} \in \mathcal{R}) = \int_{\mathcal{R}} p(\mathbf{x}')dx'_1 \ldots dx'_d$ [5, Ch. 3]. 226, 235, 252

probability distribution To analyze ML methods, it can be useful to interpret data points as i.i.d. realizations of an RV. The typical properties of such data points are then governed by the probability distribution of this RV. The probability distribution of a binary RV $y \in \{0,1\}$ is fully specified by the probabilities $p(y=0)$ and $p(y=1)=1-p(y=0)$. The probability distribution of a real-valued RV $x \in \mathbb{R}$ might be specified by a pdf $p(x)$ such that $p(x \in [a,b]) \approx p(a)|b-a|$. In the most general case, a probability distribution is defined by a probability measure [119, 139]. 14, 17, 20, 23, 24, 30, 51, 77, 124, 144, 150, 172, 177–179, 182, 183, 185, 199, 204–206, 218, 221, 226, 228, 229, 235, 239, 240, 245, 246, 251, 252, 260, 261, 264

probability space A probability space is a mathematical model of a physical process (a random experiment) with an uncertain outcome. Formally, a probability space $\mathcal{P}$ is a triplet $(\Omega, \mathcal{F}, P)$ where

- Ω is a sample space containing all possible elementary outcomes of a random experiment;
- $\mathcal{F}$ is a sigma-algebra, a collection of subsets of Ω (called events) that satisfies certain closure properties under set operations;
- P is a probability measure, a function that assigns a probability $P(\mathcal{A}) \in [0,1]$ to each event $\mathcal{A} \in \mathcal{F}$. The function must satisfy $P(\Omega) = 1$ and $P\left(\bigcup_{i=1}^{\infty} \mathcal{A}_i\right) = \sum_{i=1}^{\infty} P(\mathcal{A}_i)$ for any countable sequence of pairwise disjoint events $\mathcal{A}_1, \mathcal{A}_2, \ldots$ in $\mathcal{F}$.

Probability spaces provide the foundation for defining RVs and to reason about uncertainty in ML applications [119, 139, 160]. 246, 256

projected gradient descent (projected GD) Consider an ERM-based method that uses a parametrized model with parameter space $\mathcal{W} \subseteq \mathbb{R}^d$. Even if the objective function of ERM is smooth, we cannot use basic GD, as it does not take into account contraints on the optimization variable (i.e., the model parameters). Projected GD extends basic GD to handle constraints on the optimization variable (i.e., the model parameters). A single iteration of projected GD consists of first taking a gradient step and then projecting the result back onto the parameter space.

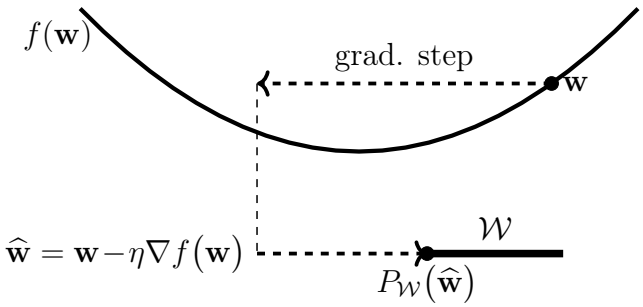


Figure 11.13: Projected GD augments a basic gradient step with a projection back onto the constraint set $\mathcal{W}$.

62, 75, 85, 86, 99, 149, 239

projection Consider a subset $\mathcal{W} \subseteq \mathbb{R}^d$ of the d-dimensional Euclidean space. We define the projection $P_{\mathcal{W}}(\mathbf{w})$ of a vector $\mathbf{w} \in \mathbb{R}^d$ onto $\mathcal{W}$ as

$$P_{\mathcal{W}}(\mathbf{w}) = \operatorname*{argmin}_{\mathbf{w}' \in \mathcal{W}} \|\mathbf{w} - \mathbf{w}'\|_2 . \tag{204}$$

In other words, $P_{\mathcal{W}}(\mathbf{w})$ is the vector in $\mathcal{W}$ which is closest to $\mathbf{w}$. The projection is only well-defined for subsets $\mathcal{W}$ for which the above minimum exists [48]. 254

proximable A convex function for which the proximal operator can be computed efficiently is sometimes referred to as proximable or simple [41]. 45

proximal operator Given a convex function $f(\mathbf{w}')$, we define its proximal operator as [42, 59]

$$\mathbf{prox}_{f(\cdot),\rho}(\mathbf{w}) := \operatorname*{argmin}_{\mathbf{w}' \in \mathbb{R}^d} \left[f(\mathbf{w}') + (\rho/2) \|\mathbf{w} - \mathbf{w}'\|_2^2 \right] \text{ with } \rho > 0.$$

As illustrated in Figure 11.14, evaluating the proximal operator amounts to minimizing a penalized variant of $f(\mathbf{w}')$. The penalty term is the scaled squared Euclidean distance to a given vector $\mathbf{w}$ (which is the input to the proximal operator). The proximal operator can be interpreted as a generalization of the gradient step, which is defined for a smooth convex function $f(\mathbf{w}')$. Indeed, taking a gradient step with step size η at the current vector $\mathbf{w}$ is the same as applying the proximal operator of the function $\tilde{f}(\mathbf{w}') = \big(\nabla f(\mathbf{w})\big)^T(\mathbf{w}' - \mathbf{w})$ and using $\rho = 1/\eta$.

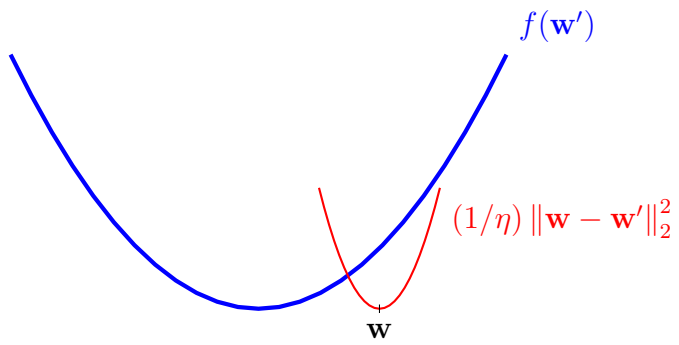


Figure 11.14: A generalized gradient step updates a vector $\mathbf{w}$ by minimizing a penalized version of the function $f(\cdot)$. The penalty term is the scaled squared Euclidean distance between the optimization variable $\mathbf{w}'$ and the given vector $\mathbf{w}$.

45, 76, 81, 104, 226, 232, 254, 255

quadratic function A function $f : \mathbb{R}^d \to \mathbb{R}$ of the form

$$f(\mathbf{w}) = \mathbf{w}^T\mathbf{Q}\mathbf{w} + \mathbf{q}^T\mathbf{w} + a,$$

with some matrix $\mathbf{Q} \in \mathbb{R}^{d \times d}$, vector $\mathbf{q} \in \mathbb{R}^d$, and scalar $a \in \mathbb{R}$. 15, 18, 19, 28, 44, 63, 66, 87, 115, 160

random variable (RV) An RV is a function that maps from a probability space $\mathcal{P}$ to a value space [119, 139]. The probability space consists of elementary events and is equipped with a probability measure that assigns probabilities to subsets of $\mathcal{P}$. Different types of RVs include

- binary RVs, which map elementary events to a set of two distinct values, such as $\{-1, 1\}$ or $\{\text{cat}, \text{no cat}\}$;
- real-valued RVs, which take values in the real numbers $\mathbb{R}$;
- vector-valued RVs, which map elementary events to the Euclidean space $\mathbb{R}^d$.

Probability theory uses the concept of measurable spaces to rigorously define and study the properties of (large) collections of RVs [119]. 14, 17, 20, 24, 25, 30, 47, 58, 77, 83, 97, 127, 140, 144, 145, 150, 155, 172, 178, 183, 188, 195, 206, 210, 212, 218, 220, 221, 226–228, 235, 239, 245, 246, 251–253, 256, 258, 261, 269

realization Consider an RV x which maps each element (i.e., outcome or elementary event) $\omega \in \mathcal{P}$ of a probability space $\mathcal{P}$ to an element a of a measurable space $\mathcal{N}$ [2, 118, 119]. A realization of x is any element $a' \in \mathcal{N}$ such that there is an element $\omega' \in \mathcal{P}$ with $x(\omega') = a'$. 14, 17, 20, 24, 25, 27, 30, 47, 51, 77, 83, 97, 124, 127, 140, 141, 144, 145, 150, 155, 172, 178, 183, 188, 189, 195, 206, 218, 220, 221, 228, 235, 245, 251, 252, 258, 261, 264

regression Regression problems revolve around the prediction of a numeric label solely from the features of a data point [6, Ch. 2]. 16, 114, 237, 241, 243, 249, 260

regret The regret of a hypothesis h relative to another hypothesis h', which serves as a baseline, is the difference between the loss incurred by h and the loss incurred by h' [151]. The baseline hypothesis h' is also referred to as an expert. 222

regularization A key challenge of modern ML applications is that they often use large models, which have an effective dimension in the order of billions. Training a high-dimensional model using basic ERM-based methods is prone to overfitting: the learned hypothesis performs well on the training set but poorly outside the training set. Regularization refers to modifications of a given instance of ERM in order to avoid overfitting, i.e., to ensure that the learned hypothesis performs not much worse outside the training set. There are three routes for implementing regularization:

1) Model pruning: We prune the original model $\mathcal{H}$ to obtain a smaller model $\mathcal{H}'$. For a parametric model, the pruning can be implemented via constraints on the model parameters (such as $w_1 \in [0.4, 0.6]$ for the weight of feature x_1 in linear regression).

2) Loss penalization: We modify the objective function of ERM by adding a penalty term to the training error. The penalty term estimates how much larger the expected loss (or risk) is compared to the average loss on the training set.

3) Data augmentation: We can enlarge the training set $\mathcal{D}$ by adding perturbed copies of the original data points in $\mathcal{D}$. One example for such a perturbation is to add the realization of an RV to the feature vector of a data point.

Figure 11.15 illustrates the above three routes to regularization. These routes are closely related and sometimes fully equivalent: data augmentation using Gaussian random variable (Gaussian RV)s to perturb the feature vectors in the training set of linear regression has the same effect as adding the penalty $\lambda \|\mathbf{w}\|_2^2$ to the training error (which is nothing but ridge regression). The decision on which route to use for regularization can be based on the available computational infrastructure. For example, it might be much easier to implement data augmentation than model pruning.

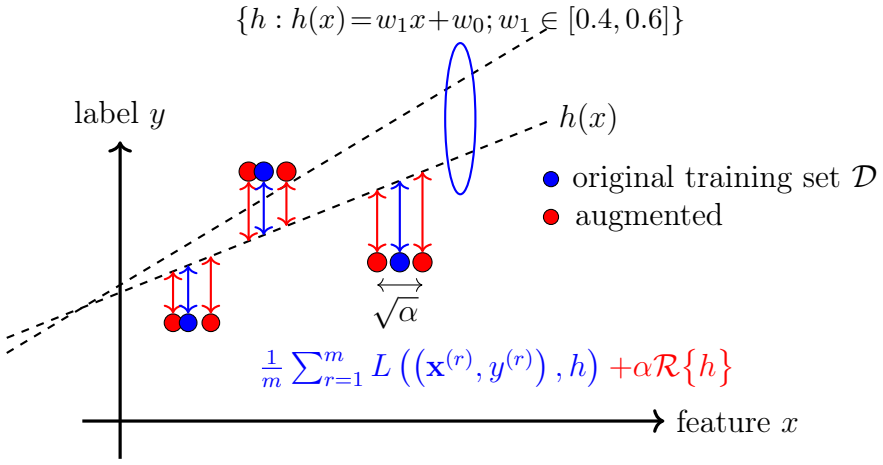


Figure 11.15: Three approaches to regularization: 1) data augmentation; 2) loss penalization; and 3) model pruning (via constraints on model parameters).

7, 18, 24, 26, 29, 30, 32, 73, 104, 106, 132, 169, 195, 211, 261, 266

regularized empirical risk minimization (RERM) Synonym for SRM. 7, 29, 43, 230

regularizer A regularizer assigns each hypothesis h from a hypothesis space $\mathcal{H}$ a quantitative measure $\mathcal{R}\big\{h\big\}$ for how much its prediction error on a training set might differ from its prediction errors on data points outside the training set. Ridge regression uses the regularizer $\mathcal{R}\big\{h\big\} := \|\mathbf{w}\|_2^2$ for linear hypothesis maps $h^{(\mathbf{w})}(\mathbf{x}) := \mathbf{w}^T\mathbf{x}$ [6, Ch. 3]. Lasso uses the regularizer $\mathcal{R}\big\{h\big\} := \|\mathbf{w}\|_1$ for linear hypothesis maps $h^{(\mathbf{w})}(\mathbf{x}) := \mathbf{w}^T\mathbf{x}$ [6, Ch. 3]. 7, 26, 29, 43, 106, 157, 230

Rényi divergence The Rényi divergence measures the (dis)similarity between two probability distributions [175]. 179

reward A reward refers to some observed (or measured) quantity that allows us to estimate the loss incurred by the prediction (or decision) of a hypothesis $h(\mathbf{x})$. For example, in an ML application to self-driving vehicles, $h(\mathbf{x})$ could represent the current steering direction of a vehicle. We could construct a reward from the measurements of a collision sensor that indicate if the vehicle is moving towards an obstacle. We define a low reward for the steering direction $h(\mathbf{x})$ if the vehicle moves dangerously towards an obstacle. 202

ridge regression Ridge regression learns the weights $\mathbf{w}$ of a linear hypothesis map $h^{(\mathbf{w})}(\mathbf{x}) = \mathbf{w}^T\mathbf{x}$. The quality of a particular choice for the model parameters $\mathbf{w}$ is measured by the sum of two components. The first

component is the average squared error loss incurred by $h^{(\mathbf{w})}$ on a set of labeled datapoints (i.e., the training set). The second component is the scaled squared Euclidean norm $\alpha\|\mathbf{w}\|_2^2$ with a regularization parameter $\alpha > 0$. Adding $\alpha\|\mathbf{w}\|_2^2$ to the average squared error loss is equivalent to replacing each original data points by the realization of (infinitely many) i.i.d. RVs centered around these data points (see regularization). 26–28, 31, 44, 63, 65, 71, 258, 260

risk Consider a hypothesis h used to predict the label y of a data point based on its features $\mathbf{x}$. We measure the quality of a particular prediction using a loss function $L\left((\mathbf{x}, y), h\right)$. If we interpret data points as the realizations of i.i.d. RVs, also the $L\left((\mathbf{x}, y), h\right)$ becomes the realization of an RV. The i.i.d. assumption allows us to define the risk of a hypothesis as the expected loss $\mathbb{E}\big\{L\left((\mathbf{x}, y), h\right)\big\}$. Note that the risk of h depends on both the specific choice for the loss function and the probability distribution of the data points. 17, 21, 204–206, 220, 228, 248, 257, 266, 269

sample size The number of individual data points contained in a dataset. 19

scatterplot A visualization technique that depicts data points by markers in a two-dimensional plane. Figure 11.16 depicts an example of a scatterplot.

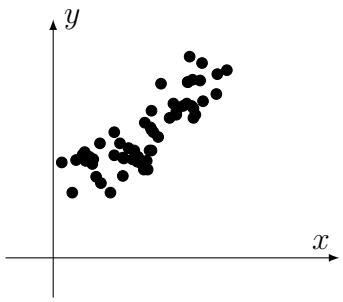


Figure 11.16: A scatterplot of some data points representing daily weather conditions in Finland. Each data point is characterized by its minimum daytime temperature x as the feature and its maximum daytime temperature y as the label. The temperatures have been measured at the FMI weather station Helsinki Kaisaniemi during 1.9.2024 - 28.10.2024.

semi-supervised learning (SSL) SSL methods use unlabeled datapoints to support the learning of a hypothesis from labeled datapoints [75]. This approach is particularly useful for ML applications that offer a large amount of unlabeled datapoints, but only a limited number of labeled datapoints.

sensitive attribute ML revolves around learning a hypothesis map that allows us to predict the label of a data point from its features. In some applications, we must ensure that the output delivered by an ML system does not allow us to infer sensitive attributes of a data point. Which part of a data point is considered a sensitive attribute is a design choice that varies across different application domains.

smooth A real-valued function $f : \mathbb{R}^d \to \mathbb{R}$ is smooth if it is differentiable

and its gradient $\nabla f(\mathbf{w})$ is continuous at all $\mathbf{w} \in \mathbb{R}^d$ [58, 176]. A smooth function f is referred to as β-smooth if the gradient $\nabla f(\mathbf{w})$ is Lipschitz continuous with Lipschitz constant β, i.e.,

$$\|\nabla f(\mathbf{w}) - \nabla f(\mathbf{w}')\| \leq \beta \|\mathbf{w} - \mathbf{w}'\|, \text{ for any } \mathbf{w}, \mathbf{w}' \in \mathbb{R}^d.$$

The constant β quantifies the amount of smoothness of the function f: the smaller the β, the smoother f is. Optimization problems with a smooth objective function can be solved effectively by gradient-based methods. Indeed, gradient-based methods approximate the objective function locally around a current choice $\mathbf{w}$ using its gradient. This approximation works well if the gradient does not change too rapidly. We can make this informal claim precise by studying the effect of a single gradient step with step size $\eta = 1/\beta$ (see Figure 11.17).

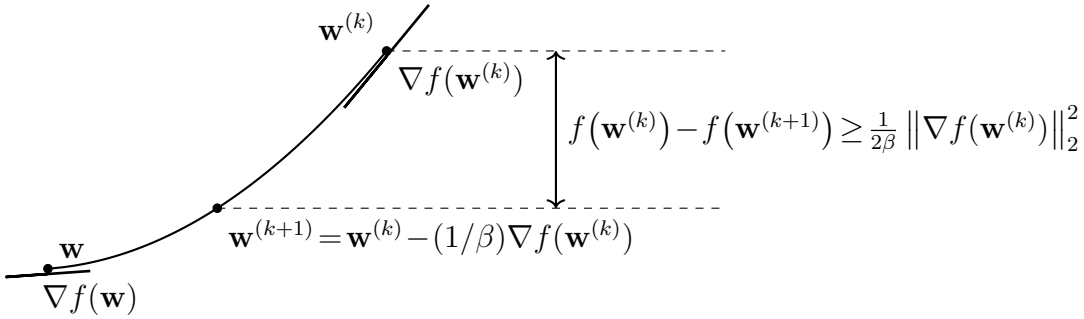


Figure 11.17: Consider an objective function $f(\mathbf{w})$ that is β-smooth. Taking a gradient step, with step size $\eta = 1/\beta$, decreases the objective by at least $\frac{1}{2\beta} \left\|\nabla f(\mathbf{w}^{(k)})\right\|_2^2$ [55, 58, 176]. Note that the step size $\eta = 1/\beta$ becomes larger for smaller β. Thus, for smoother objective functions (i.e., those with smaller β), we can take larger steps.

13, 16, 37, 62, 80, 253, 255, 270

soft clustering Soft clustering refers to the task of partitioning a given set of data points into (a few) overlapping clusters. Each data point is assigned to several different clusters with varying degrees of belonging. Soft clustering methods determine the degree of belonging (or soft cluster assignment) for each data point and each cluster. A principled approach to soft clustering is by interpreting data points as i.i.d. realizations of a GMM. We then obtain a natural choice for the degree of belonging as the conditional probability of a data point belonging to a specific mixture component. 208, 217

squared error loss The squared error loss measures the prediction error of a hypothesis h when predicting a numeric label $y \in \mathbb{R}$ from the features $\mathbf{x}$ of a data point. It is defined as

$$L\left((\mathbf{x}, y), h\right) := \big(y - \underbrace{h(\mathbf{x})}_{=\hat{y}}\big)^2.$$

13, 17, 30, 31, 44, 51, 104, 114, 115, 135, 169, 185, 195, 200, 205, 240, 241, 250, 261

statistical aspects By statistical aspects of an ML method, we refer to (properties of) the probability distribution of its output under a probabilistic model for the data fed into the method. 15, 43, 235, 241, 268

step size See learning rate. 16, 62, 64, 231, 232, 240, 255, 263

stochastic gradient descent (SGD) Stochastic GD is obtained from GD by replacing the gradient of the objective function with a stochastic

approximation. A main application of stochastic GD is to train a parametrized model via ERM on a training set $\mathcal{D}$ that is either very large or not readily available (e.g., when data points are stored in a database distributed all over the planet). To evaluate the gradient of the empirical risk (as a function of the model parameters $\mathbf{w}$), we need to compute a sum $\sum_{r=1}^{m} \nabla_{\mathbf{w}} L\left(\mathbf{z}^{(r)}, \mathbf{w}\right)$ over all data points in the training set. We obtain a stochastic approximation to the gradient by replacing the sum $\sum_{r=1}^{m} \nabla_{\mathbf{w}} L\left(\mathbf{z}^{(r)}, \mathbf{w}\right)$ with a sum $\sum_{r \in \mathcal{B}} \nabla_{\mathbf{w}} L\left(\mathbf{z}^{(r)}, \mathbf{w}\right)$ over a randomly chosen subset $\mathcal{B} \subseteq \{1, \ldots, m\}$ (see Figure 11.18). We often refer to these randomly chosen data points as a batch. The batch size $|\mathcal{B}|$ is an important parameter of stochastic GD. Stochastic GD with $|\mathcal{B}| > 1$ is referred to as mini-batch stochastic GD [177].

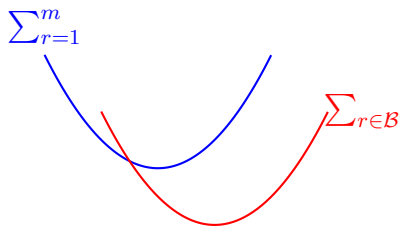


Figure 11.18: Stochastic GD for ERM approximates the gradient $\sum_{r=1}^{m} \nabla_{\mathbf{w}} L\left(\mathbf{z}^{(r)}, \mathbf{w}\right)$ by replacing the sum over all data points in the training set (indexed by $r = 1, \ldots, m$) with a sum over a randomly chosen subset $\mathcal{B} \subseteq \{1, \ldots, m\}$.

146, 179, 183, 206, 225, 226, 239

stopping criterion Many ML methods use iterative algorithms that construct a sequence of model parameters (such as the weights of a linear

map or the weights of an ANN). These parameters (hopefully) converge to an optimal choice for the model parameters. In practice, given finite computational resources, we need to stop iterating after a finite number of repetitions. A stopping criterion is any well-defined condition required for stopping the iteration. 47, 62, 64, 65, 68, 74, 78, 92, 94, 96, 97, 101, 103, 105, 167

strongly convex A continuously differentiable real-valued function $f(\mathbf{x})$ is strongly convex with coefficient σ if $f(\mathbf{y}) \geq f(\mathbf{x}) + \nabla f(\mathbf{x})^T(\mathbf{y} - \mathbf{x}) + (\sigma/2)\left\|\mathbf{y} - \mathbf{x}\right\|_2^2$ [58], [55, Sec. B.1.1]. 46, 80

structural risk minimization (SRM) Structural risk minimization is the problem of finding the hypothesis that optimally balances the average loss (or empirical risk) on a training set with a regularization term. The regularization term penalizes a hypothesis that is not robust against (small) perturbations of the data points in the training set. 240, 260

supremum (or least upper bound) The supremum of a set of real numbers is the smallest number that is greater than or equal to every element in the set. More formally, a real number a is the supremum of a set $\mathcal{A} \subseteq \mathbb{R}$ if: 1) a is an upper bound of $\mathcal{A}$; and 2) no number smaller than a is an upper bound of $\mathcal{A}$. Every non-empty set of real numbers that is bounded above has a supremum, even if it does not contain its supremum as an element [2, Sec. 1.4]. 245

test set A set of data points that have been used neither to train a model (e.g., via ERM) nor in a validation set to choose between different

models. 37, 204, 222

training error The average loss of a hypothesis when predicting the labels of the data points in a training set. We sometimes refer by training error also to minimal average loss which is achieved by a solution of ERM. 11, 18, 21–25, 28, 30, 42, 76, 169, 206, 257, 258, 267, 269

training set A training set is a dataset $\mathcal{D}$ which consists of some data points used in ERM to learn a hypothesis $\hat{h}$. The average loss of $\hat{h}$ on the training set is referred to as the training error. The comparison of the training error with the validation error of $\hat{h}$ allows us to diagnose the ML method and informs how to improve the validation error (e.g., using a different hypothesis space or collecting more data points) [6, Sec. 6.6]. 14, 16–18, 21–25, 27, 28, 30, 31, 65, 68, 71, 72, 78, 124, 134, 158, 175, 176, 184, 188, 192, 195, 200, 202, 203, 206, 208, 209, 218, 220, 228, 229, 239–241, 243, 248, 250, 257–261, 265–267, 269

transparency Transparency is a fundamental requirement for trustworthy AI [95]. In the context of ML methods, transparency is often used interchangeably with explainability [113, 178]. However, in the broader scope of AI systems, transparency extends beyond explainability and includes providing information about the system's limitations, reliability, and intended use. In medical diagnosis systems, transparency requires disclosing the confidence level for the predictions delivered by a trained model. In credit scoring, AI-based lending decisions should be accompanied by explanations of contributing factors, such as income level or credit history. These explanations allow humans (e.g., a loan applicant)

to understand and contest automated decisions. Some ML methods inherently offer transparency. For example, logistic regression provides a quantitative measure of classification reliability through the value $|h(\mathbf{x})|$. Decision trees are another example, as they allow human-readable decision rules [169]. Transparency also requires a clear indication when a user is engaging with an AI system. For example, AI-powered chatbots should notify users that they are interacting with an automated system rather than a human. Furthermore, transparency encompasses comprehensive documentation detailing the purpose and design choices underlying the AI system. For instance, model datasheets [149] and AI system cards [179] help practitioners understand the intended use cases and limitations of an AI system [180]. 227

trustworthy artificial intelligence (trustworthy AI) Besides the computational aspects and statistical aspects, a third main design aspect of ML methods is their trustworthiness [181]. The EU has put forward seven key requirements (KRs) for trustworthy AI (that typically build on ML methods) [94]:

1) KR1 - Human agency and oversight;
2) KR2 - Technical robustness and safety;
3) KR3 - Privacy and data governance;
4) KR4 - Transparency;
5) KR5 - Diversity, non-discrimination and fairness;
6) KR6 - Societal and environmental well-being;

7) KR7 - Accountability.

152, 215, 267

validation Consider a hypothesis $\hat{h}$ that has been learned via some ML method, e.g., by solving ERM on a training set $\mathcal{D}$. Validation refers to the practice of evaluating the loss incurred by the hypothesis $\hat{h}$ on a set of data points that are not contained in the training set $\mathcal{D}$. 11, 21, 213, 269

validation error Consider a hypothesis $\hat{h}$ which is obtained by some ML method, e.g., using ERM on a training set. The average loss of $\hat{h}$ on a validation set, which is different from the training set, is referred to as the validation error. 11, 17, 21–25, 28, 154, 157, 200, 267, 269

validation set A set of data points used to estimate the risk of a hypothesis $\hat{h}$ that has been learned by some ML method (e.g., solving ERM). The average loss of $\hat{h}$ on the validation set is referred to as the validation error and can be used to diagnose an ML method (see [6, Sec. 6.6]). The comparison between training error and validation error can inform directions for improvement of the ML method (such as using a different hypothesis space). 17, 21, 22, 25, 154, 188, 266, 269

variance The variance of a real-valued RV x is defined as the expectation $\mathbb{E}\big\{\big(x-\mathbb{E}\{x\}\big)^2\big\}$ of the squared difference between x and its expectation $\mathbb{E}\{x\}$. We extend this definition to vector-valued RVs $\mathbf{x}$ as $\mathbb{E}\big\{\big\|\mathbf{x}-\mathbb{E}\{\mathbf{x}\}\big\|_2^2\big\}$. 14, 17, 205, 226

vertical federated learning (vertical FL) Vertical FL uses local datasets that are constituted by the same data points but characterizing them with different features [76]. For example, different healthcare providers might all contain information about the same population of patients. However, different healthcare providers collect different measurements (e.g., blood values, electrocardiography, lung X-ray) for the same patients. 121, 130–132

weights Consider a parametrized hypothesis space $\mathcal{H}$. We use the term weights for numeric model parameters that are used to scale features or their transformations in order to compute $h^{(\mathbf{w})} \in \mathcal{H}$. A linear model uses weights $\mathbf{w} = \big(w_1, \ldots, w_d\big)^T$ to compute the linear combination $h^{(\mathbf{w})}(\mathbf{x}) = \mathbf{w}^T\mathbf{x}$. Weights are also used in ANNs to form linear combinations of features or the outputs of neurons in hidden layers. 18, 201, 219, 233, 240, 246, 249, 260, 265, 266

zero-gradient condition Consider the unconstrained optimization problem $\min_{\mathbf{w}\in\mathbb{R}^d} f(\mathbf{w})$ with a smooth and convex objective function $f(\mathbf{w})$. A necessary and sufficient condition for a vector $\widehat{\mathbf{w}} \in \mathbb{R}^d$ to solve this problem is that the gradient $\nabla f\big(\widehat{\mathbf{w}}\big)$ is the zero vector,

$$\nabla f\big(\widehat{\mathbf{w}}\big) = \mathbf{0} \Leftrightarrow f\big(\widehat{\mathbf{w}}\big) = \min_{\mathbf{w}\in\mathbb{R}^d} f(\mathbf{w}).$$

46

0/1 loss The 0/1 loss $L^{(0/1)}\left((\mathbf{x}, y), h\right)$ measures the quality of a classifier $h(\mathbf{x})$ that delivers a prediction $\hat{y}$ (e.g., via thresholding (197)) for the

label y of a data point with features $\mathbf{x}$. It is equal to 0 if the prediction is correct, i.e., $L^{(0/1)}\left(\left(\mathbf{x}, y\right), h\right) = 0$ when $\hat{y} = y$. It is equal to 1 if the prediction is wrong, i.e., $L^{(0/1)}\left(\left(\mathbf{x}, y\right), h\right) = 1$ when $\hat{y} \neq y$. 201

References

[1] W. Rudin, *Real and Complex Analysis*, 3rd ed., New York, 1987.

[2] W. Rudin, *Principles of Mathematical Analysis*, 3rd ed., New York, 1976.

[3] G. H. Golub and C. F. Van Loan, *Matrix Computations*, 4th ed. Baltimore, MD: Johns Hopkins University Press, 2013.

[4] G. Golub and C. van Loan, "An analysis of the total least squares problem," *SIAM J. Numerical Analysis*, vol. 17, no. 6, pp. 883–893, Dec. 1980.

[5] D. Bertsekas and J. Tsitsiklis, *Introduction to Probability*, 2nd ed. Athena Scientific, 2008.

[6] A. Jung, *Machine Learning: The Basics*, 1st ed. Springer Singapore, Feb. 2022.

[7] M. Wollschlaeger, T. Sauter, and J. Jasperneite, "The future of industrial communication: Automation networks in the era of the internet of things and industry 4.0," *IEEE Industrial Electronics Magazine*, vol. 11, no. 1, pp. 17–27, 2017.

[8] M. Satyanarayanan, "The emergence of edge computing," *Computer*, vol. 50, no. 1, pp. 30–39, Jan. 2017. [Online]. Available: https://doi.org/10.1109/MC.2017.9

[9] H. Ates, A. Yetisen, F. Güder, and C. Dincer, "Wearable devices for the detection of covid-19," *Nature Electronics*, vol. 4, no. 1, pp. 13–14, 2021. [Online]. Available: https://doi.org/10.1038/s41928-020-00533-1

[10] H. Boyes, B. Hallaq, J. Cunningham, and T. Watson, "The industrial internet of things (iiot): An analysis framework," *Computers in Industry*, vol. 101, pp. 1–12, 2018. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0166361517307285

[11] S. Cui, A. Hero, Z.-Q. Luo, and J. Moura, Eds., *Big Data over Networks*, 2016.

[12] A. Barabási, N. Gulbahce, and J. Loscalzo, "Network medicine: a network-based approach to human disease," *Nature Reviews Genetics*, vol. 12, no. 56, 2011.

[13] M. E. J. Newman, *Networks: An Introduction*, 2010.

[14] B. McMahan, E. Moore, D. Ramage, S. Hampson, and B. A. y. Arcas, "Communication-Efficient Learning of Deep Networks from Decentralized Data," in *Proceedings of the 20th International Conference on Artificial Intelligence and Statistics*, ser. Proceedings of Machine Learning Research, A. Singh and J. Zhu, Eds., vol. 54. PMLR, 20–22 Apr 2017, pp. 1273–1282. [Online]. Available: https://proceedings.mlr.press/v54/mcmahan17a.html

[15] T. Li, A. K. Sahu, A. Talwalkar, and V. Smith, "Federated learning: Challenges, methods, and future directions," *IEEE Signal Processing Magazine*, vol. 37, no. 3, pp. 50–60, May 2020.

[16] Y. Cheng, Y. Liu, T. Chen, and Q. Yang, "Federated learning for privacy-preserving ai," *Communications of the ACM*, vol. 63, no. 12, pp. 33–36, Dec. 2020.

[17] N. Agarwal, A. Suresh, F. Yu, S. Kumar, and H. McMahan, "cpSGD: Communication-efficient and differentially-private distributed sgd," in *Proc. Neural Inf. Proc. Syst. (NIPS)*, 2018.

[18] V. Smith, C.-K. Chiang, M. Sanjabi, and A. Talwalkar, "Federated Multi-Task Learning," in *Advances in Neural Information Processing Systems*, vol. 30, 2017. [Online]. Available: https://proceedings.neurips.cc/paper/2017/file/6211080fa89981f66b1a0c9d55c61d0f-Paper.pdf

[19] J. You, J. Wu, X. Jin, and M. Chowdhury, "Ship compute or ship data? why not both?" in *18th USENIX Symposium on Networked Systems Design and Implementation (NSDI 21)*. USENIX Association, Apr. 2021, pp. 633–651. [Online]. Available: https://www.usenix.org/conference/nsdi21/presentation/you

[20] D. P. Bertsekas and J. Tsitsiklis, *Parallel and Distributed Computation: Numerical Methods*. Athena Scientific, 2015.

[21] M. van Steen and A. Tanenbaum, *Distributed Systems*, 3rd ed., Feb. 2017, self-published, open publication.

[22] D. Tse and P. Viswanath, *Fundamentals of Wireless Communication*. Cambridge University Press, 2005.

[23] T. Yang, G. Andrew, H. Eichner, H. Sun, W. Li, N. Kong, D. Ramage, and F. Beaufays, "Applied federated learning: Improving google keyboard query suggestions," 2018. [Online]. Available: https://arxiv.org/abs/1812.02903

[24] A. Ghosh, J. Chung, D. Yin, and K. Ramchandran, "An efficient framework for clustered federated learning," in *34th Conference on Neural Information Processing Systems (NeurIPS 2020)*, Vancouver, Canada, 2020.

[25] F. Sattler, K. Müller, and W. Samek, "Clustered federated learning: Model-agnostic distributed multitask optimization under privacy constraints," *IEEE Transactions on Neural Networks and Learning Systems*, 2020.

[26] G. Strang, *Computational Science and Engineering*. Wellesley-Cambridge Press, MA, 2007.

[27] G. Strang, *Introduction to Linear Algebra*, 5th ed. Wellesley-Cambridge Press, MA, 2016.

[28] H. H. Bauschke and P. L. Combettes, *Convex Analysis and Monotone Operator Theory in Hilbert Spaces*. New York: Springer, 2011.

[29] S. Shalev-Shwartz and S. Ben-David, *Understanding Machine Learning – from Theory to Algorithms*. Cambridge University Press, 2014.

[30] E. L. Lehmann and G. Casella, *Theory of Point Estimation*, 2nd ed., New York, 1998.

[31] A. Esteva, B. Kuprel, R. A. Novoa, J. Ko, S. M. Swetter, H. M. Blau, and S. Thrun, "Dermatologist-level classification of skin cancer with deep neural networks," *Nature*, vol. 542, 2017.

[32] H. Lütkepohl, *New Introduction to Multiple Time Series Analysis*, New York, 2005.

[33] N. Goodall, "Can you program ethics into a self-driving car?" *IEEE Spectrum*, vol. 53, no. 6, pp. 28–58, June 2016.

[34] J. R. Quinlan, "Induction of decision trees," *Machine Learning*, vol. 1, no. 1, pp. 81–106.

[35] B. Schölkopf and A. Smola, *Learning with Kernels: Support Vector Machines, Regularization, Optimization, and Beyond.* Cambridge, MA, USA: MIT Press, Dec. 2002.

[36] M. Wainwright, *High-Dimensional Statistics: A Non-Asymptotic Viewpoint.* Cambridge: Cambridge University Press, 2019.

[37] Y. SarcheshmehPour, Y. Tian, L. Zhang, and A. Jung, "Clustered federated learning via generalized total variation minimization," *IEEE Transactions on Signal Processing*, vol. 71, pp. 4240–4256, 2023.

[38] F. Chung, "Spectral graph theory," in *Regional Conference Series in Mathematics*, 1997, no. 92.

[39] D. Spielman, "Spectral and algebraic graph theory," 2019.

[40] D. Spielman, "Spectral graph theory," in *Combinatorial Scientific Computing*, U. Naumann and O. Schenk, Eds. Chapman and Hall/CRC, 2012.

[41] L. Condat, "A primal–dual splitting method for convex optimization involving lipschitzian, proximable and linear composite terms," *Journal of Opt. Th. and App.*, vol. 158, no. 2, pp. 460–479, Aug. 2013.

[42] N. Parikh and S. Boyd, "Proximal algorithms," *Foundations and Trends in Optimization*, vol. 1, no. 3, pp. 123–231, 2013.

[43] R. Peng and D. A. Spielman, "An efficient parallel solver for SDD linear systems," in *Proc. ACM Symposium on Theory of Computing*, New York, NY, 2014, pp. 333–342.

[44] N. K. Vishnoi, "$Lx = b$ — Laplacian solvers and their algorithmic applications," *Foundations and Trends in Theoretical Computer Science*, vol. 8, no. 1–2, pp. 1–141, 2012. [Online]. Available: http://dx.doi.org/10.1561/0400000054

[45] D. Sun, K.-C. Toh, and Y. Yuan, "Convex clustering: Model, theoretical guarantee and efficient algorithm," *Journal of Machine Learning Research*, vol. 22, no. 9, pp. 1–32, 2021. [Online]. Available: http://jmlr.org/papers/v22/18-694.html

[46] K. Pelckmans, J. D. Brabanter, J. Suykens, and B. D. Moor, "Convex clustering shrinkage," in *PASCAL Workshop on Statistics and Optimization of Clustering Workshop*, 2005.

[47] R. T. Rockafellar, *Network Flows and Monotropic Optimization*. Athena Scientific, Jul. 1998.

[48] S. Boyd and L. Vandenberghe, *Convex Optimization*, Cambridge, UK, 2004.

[49] D. P. Bertsekas, *Network Optimization: Continuous and Discrete Models*. Athena Scientific, 1998.

[50] C. G. Atkeson, A. W. Moore, and S. Schaal, "Locally weighted learning," *Artif. Intell. Rev.*, vol. 11, no. 1–5, pp. 11–73, Feb. 1997. [Online]. Available: https://doi.org/10.1023/A:1006559212014

[51] A. Jacot, F. Gabriel, and C. Hongler, "Neural tangent kernel: Convergence and generalization in neural networks," in *Advances in Neural Information Processing Systems*, S. Bengio, H. Wallach, H. Larochelle, K. Grauman, N. Cesa-Bianchi, and R. Garnett, Eds., vol. 31. Curran Associates, Inc., 2018. [Online]. Available: https://proceedings.neurips.cc/paper_files/paper/2018/file/5a4be1fa34e62bb8a6ec6b91d2462f5a-Paper.pdf

[52] S. S. Du, X. Zhai, B. Póczos, and A. Singh, "Gradient descent provably optimizes over-parameterized neural networks," in *7th International Conference on Learning Representations, ICLR 2019, New Orleans, LA, USA, May 6-9, 2019*. OpenReview.net, 2019. [Online]. Available: https://openreview.net/forum?id=S1eK3i09YQ

[53] W. E, C. Ma, and L. Wu, "A comparative analysis of optimization and generalization properties of two-layer neural network and random feature models under gradient descent dynamics," *Science China Mathematics*, vol. 63, no. 7, pp. 1235–1258, 2020. [Online]. Available: https://doi.org/10.1007/s11425-019-1628-5

[54] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga, A. Desmaison, A. Köpf,

E. Yang, Z. DeVito, M. Raison, A. Tejani, S. Chilamkurthy, B. Steiner, L. Fang, J. Bai, and S. Chintala, *PyTorch: An Imperative Style, High-Performance Deep Learning Library.* Red Hook, NY, USA: Curran Associates Inc., 2019.

[55] D. P. Bertsekas, *Convex Optimization Algorithms.* Athena Scientific, 2015.

[56] T. Schaul, X. Zhang, and Y. LeCun, "No more pesky learning rates," in *Proc. of the 30th International Conference on Machine Learning, PMLR 28(3)*, vol. 28, Atlanta, Georgia, June 2013, pp. 343–351.

[57] M. Andrychowicz, M. Denil, S. G. Colmenarejo, M. W. Hoffman, D. Pfau, T. Schaul, B. Shillingford, and N. de Freitas, "Learning to learn by gradient descent by gradient descent," in *Proceedings of the 30th International Conference on Neural Information Processing Systems*, ser. NIPS'16. Red Hook, NY, USA: Curran Associates Inc., 2016, pp. 3988–3996.

[58] Y. Nesterov, *Introductory lectures on convex optimization*, ser. Applied Optimization. Kluwer Academic Publishers, Boston, MA, 2004, vol. 87, a basic course. [Online]. Available: http://dx.doi.org/10.1007/978-1-4419-8853-9

[59] H. Bauschke and P. Combettes, *Convex Analysis and Monotone Operator Theory in Hilbert Spaces*, 2nd ed., New York, 2017.

[60] V. Istrățescu, *Fixed point theory: An Introduction*, ser. Mathematics and its applications ; 7. Dordrecht: Reidel, 1981.

[61] B. Ying, K. Yuan, Y. Chen, H. Hu, P. PAN, and W. Yin, "Exponential graph is provably efficient for decentralized deep training," in *Advances in Neural Information Processing Systems*, M. Ranzato, A. Beygelzimer, Y. Dauphin, P. Liang, and J. W. Vaughan, Eds., vol. 34. Curran Associates, Inc., 2021, pp. 13 975–13 987. [Online]. Available: https://proceedings.neurips.cc/paper_files/paper/2021/file/74e1ed8b55ea44fd7dbb685c412568a4-Paper.pdf

[62] S. Boyd, P. Diaconis, and L. Xiao, "Fastest mixing markov chain on a graph," *SIAM Review*, vol. 46, no. 4, pp. 667–689, 2004.

[63] D. Mills, "Internet time synchronization: the network time protocol," *IEEE Transactions on Communications*, vol. 39, no. 10, pp. 1482–1493, 1991.

[64] J. Hirvonen and J. Suomela. (2023) Distributed algorithms 2020.

[65] R. Diestel, *Graph Theory*. Springer Berlin Heidelberg, 2005.

[66] T. Li, A. K. Sahu, M. Zaheer, M. Sanjabi, A. Talwalkar, and V. Smith, "Federated optimization in heterogeneous networks," in *Proceedings of the Third Conference on Machine Learning and Systems, MLSys 2020, Austin, TX, USA, March 2-4, 2020*, I. S. Dhillon, D. S. Papailiopoulos, and V. Sze, Eds. mlsys.org, 2020. [Online]. Available: https://proceedings.mlsys.org/paper_files/paper/2020/hash/1f5fe83998a09396ebe6477d9475ba0c-Abstract.html

[67] A. Tanenbaum and D. Wetherall, *Computer Networks*, 5th ed. USA: Prentice Hall Press, 2010.

[68] P. Tseng, "Convergence of a block coordinate descent method for nondifferentiable minimization," *Journal of Optimization Theory and Applications*, vol. 109, no. 3, pp. 475–494, 2001. [Online]. Available: https://doi.org/10.1023/A:1017501703105

[69] S. Boyd, N. Parikh, E. Chu, B. Peleato, and J. Eckstein, *Distributed Optimization and Statistical Learning via the Alternating Direction Method of Multipliers*. Hanover, MA: Now Publishers, 2010, vol. 3, no. 1.

[70] J. Liu and C. Zhang, "Distributed learning systems with first-order methods," *Foundations and Trends in Databases*, vol. 9, no. 1, p. 100.

[71] Q. Yang, Y. Liu, Y. Cheng, Y. Kang, T. Chen, and H. Yu, *Federated Learning*, 1st ed. Springer, 2022.

[72] S. Iyer, T. Killingback, B. Sundaram, and Z. Wang, "Attack robustness and centrality of complex networks." *PLoS One*, vol. 8, no. 4, p. e59613, 2013.

[73] D. J. Spiegelhalter, "An omnibus test for normality for small samples," *Biometrika*, vol. 67, no. 2, pp. 493–496, 2024/03/25/ 1980. [Online]. Available: http://www.jstor.org/stable/2335498

[74] Q. Yang, Y. Liu, Y. Cheng, Y. Kang, T. Chen, and H. Yu, *Horizontal Federated Learning*. Cham: Springer International Publishing, 2020, pp. 49–67. [Online]. Available: https://doi.org/10.1007/978-3-031-01585-4_4

[75] O. Chapelle, B. Schölkopf, and A. Zien, Eds., *Semi-Supervised Learning*. Cambridge, Massachusetts: The MIT Press, 2006.

[76] Q. Yang, Y. Liu, Y. Cheng, Y. Kang, T. Chen, and H. Yu, *Vertical Federated Learning*. Cham: Springer International Publishing, 2020, pp. 69–81. [Online]. Available: https://doi.org/10.1007/978-3-031-01585-4_5

[77] H. Ludwig and N. Baracaldo, Eds., *Federated Learning: A Comprehensive Overview of Methods and Applications*. Springer, 2022.

[78] A. Shamsian, A. Navon, E. Fetaya, and G. Chechik, "Personalized federated learning using hypernetworks," in *Proceedings of the 38th International Conference on Machine Learning*, ser. Proceedings of Machine Learning Research, M. Meila and T. Zhang, Eds., vol. 139. PMLR, 18–24 Jul 2021, pp. 9489–9502. [Online]. Available: https://proceedings.mlr.press/v139/shamsian21a.html

[79] F. Sung, Y. Yang, L. Zhang, T. Xiang, P. H. Torr, and T. M. Hospedales, "Learning to compare: Relation network for few-shot learning," in *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2018, pp. 1199–1208.

[80] V. Satorras and J. Bruna, "Few-shot learning with graph neural networks." in *ICLR (Poster)*. OpenReview.net, 2018. [Online]. Available: http://dblp.uni-trier.de/db/conf/iclr/iclr2018.html#SatorrasE18

[81] J. Nešetřil and P. O. de Mendez, *Sparsity: Graphs, Structures, and Algorithms*, ser. Algorithms and Combinatorics. Springer, 2012, vol. 28.

[82] J. Tropp, "An introduction to matrix concentration inequalities," *Found. Trends Mach. Learn.*, May 2015.

[83] A. Jung, "Clustering in partially labeled stochastic block models via total variation minimization," in *Proc. 54th Asilomar Conf. Signals, Systems, Computers*, Pacific Grove, CA, Nov. 2020.

[84] B. Bollobas, W. Fulton, A. Katok, F. Kirwan, and P. Sarnak, *Random graphs*. Cambridge studies in advanced mathematics., 2001, vol. 73.

[85] G. Keiser, *Optical Fiber Communication*, 4th ed. New Delhi: Mc-Graw Hill, 2011.

[86] D. Tse and P. Viswanath, *Fundamentals of wireless communication*. USA: Cambridge University Press, 2005.

[87] M. Fiedler, "Algebraic connectivity of graphs," *Czechoslovak Mathematical Journal,*, vol. 23, no. 2, pp. 298–305, 1973.

[88] S. Hoory, N. Linial, and A. Wigderson, "Expander graphs and their applications," *Bull. Amer. Math. Soc.*, vol. 43, no. 04, pp. 439–562, Aug. 2006.

[89] Y.-T. Chow, W. Shi, T. Wu, and W. Yin, "Expander graph and communication-efficient decentralized optimization," in *2016 50th Asilomar Conference on Signals, Systems and Computers*, 2016, pp. 1715–1720.

[90] S. M. Kay, *Fundamentals of Statistical Signal Processing: Estimation Theory*. Englewood Cliffs, NJ: Prentice Hall, 1993.

[91] S. Chepuri, S. Liu, G. Leus, and A. Hero, "Learning sparse graphs under smoothness prior," in *Proc. of the IEEE Int. Conf. on Acoustics, Speech and Signal Processing*, 2017, pp. 6508–6512.

[92] J. Tan, Y. Zhou, G. Liu, J. H. Wang, and S. Yu, "pFedSim: Similarity-Aware Model Aggregation Towards Personalized Federated Learning," *arXiv e-prints*, p. arXiv:2305.15706, May 2023.

[93] I. Goodfellow, Y. Bengio, and A. Courville, *Deep Learning*. MIT Press, 2016.

[94] E. Commission, C. Directorate-General for Communications Networks, and Technology, *The Assessment List for Trustworthy Artificial Intelligence (ALTAI) for self assessment*. Publications Office, 2020.

[95] H.-L. E. G. on Artificial Intelligence, "Ethics guidelines for trustworthy AI," European Commission, Tech. Rep., April 2019.

[96] D. Kuss and O. Lopez-Fernandez, "Internet addiction and problematic internet use: A systematic review of clinical research." *World J Psychiatry*, vol. 6, no. 1, pp. 143–176, Mar 2016.

[97] L. Munn, "Angry by design: toxic communication and technical architectures," *Humanities and Social Sciences Communications*, vol. 7, no. 1, p. 53, 2020. [Online]. Available: https://doi.org/10.1057/s41599-020-00550-7

[98] P. Mozur, "A genocide incited on facebook, with posts from myanmar's military," *The New York Times*, 2018.

[99] A. Simchon, M. Edwards, and S. Lewandowsky, "The persuasive effects of political microtargeting in the age of generative artificial intelligence." *PNAS Nexus*, vol. 3, no. 2, p. pgae035, Feb 2024.

[100] J. Near and D. Darais, "Guidelines for evaluating differential privacy guarantees," National Institute of Standards and Technology, Gaithersburg, MD, Tech. Rep., 2023.

[101] S. Wachter, "Data protection in the age of big data," *Nature Electronics*, vol. 2, no. 1, pp. 6–7, 2019. [Online]. Available: https://doi.org/10.1038/s41928-018-0193-y

[102] P. Samarati, "Protecting respondents identities in microdata release," *IEEE Transactions on Knowledge and Data Engineering*, vol. 13, no. 6, pp. 1010–1027, 2001.

[103] E. Comission, "Regulation (eu) 2016/679 of the european parliament and of the council of 27 april 2016 on the protection of natural persons with regard to the processing of personal data and on the free movement of such data, and repealing directive 95/46/ec (general data protection regulation) (text with eea relevance)," no. 119, pp. 1–88, May 2016.

[104] U. N. G. Assembly, *The Universal Declaration of Human Rights (UDHR)*, New York, 1948.

[105] N. Kozodoi, J. Jacob, and S. Lessmann, "Fairness in credit scoring: Assessment, implementation and profit implications," *European Journal of Operational Research*, vol. 297, no. 3, pp. 1083–1094, 2022. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0377221721005385

[106] J. Gonçalves-Sá and F. Pinheiro, *Societal Implications of Recommendation Systems: A Technical Perspective*. Cham: Springer

International Publishing, 2024, pp. 47–63. [Online]. Available: https://doi.org/10.1007/978-3-031-41264-6_3

[107] A. Abrol and R. Jha, "Power optimization in 5g networks: A step towards green communication," *IEEE Access*, vol. 4, pp. 1355–1374, 2016.

[108] J. R. Taylor, *An Introduction to Error Analysis: The study of uncertainties in physical measurements*, second edition. ed. Sausalito, Calif: University Science Books, 1997.

[109] A. Jung, "A fixed-point of view on gradient methods for big data," *Frontiers in Applied Mathematics and Statistics*, vol. 3, 2017. [Online]. Available: https://www.frontiersin.org/article/10.3389/fams.2017.00018

[110] I. Akyildiz, W. Su, Y. Sankarasubramaniam, and E. Cayirci, "A survey on sensor networks," *IEEE Communications Magazine*, vol. 40, no. 8, pp. 102–114, 2002.

[111] C. Wang, Y. Yang, and P. Zhou, "Towards efficient scheduling of federated mobile devices under computational and statistical heterogeneity," *IEEE Transactions on Parallel and Distributed Systems*, vol. 32, no. 2, pp. 394–410, 2021.

[112] J. Colin, T. Fel, R. Cadène, and T. Serre, "What I Cannot Predict, I Do Not Understand: A Human-Centered Evaluation Framework for Explainability Methods." *Advances in Neural Information Processing Systems*, vol. 35, pp. 2832–2845, 2022.

[113] A. Jung and P. Nardelli, "An information-theoretic approach to personalized explainable machine learning," *IEEE Sig. Proc. Lett.*, vol. 27, pp. 825–829, 2020.

[114] L. Zhang, G. Karakasidis, A. Odnoblyudova, L. Dogruel, Y. Tian, and A. Jung, "Explainable empirical risk minimization," *Neural Computing and Applications*, vol. 36, no. 8, pp. 3983–3996, 2024. [Online]. Available: https://doi.org/10.1007/s00521-023-09269-3

[115] M. J. Sheller, B. Edwards, G. A. Reina, J. Martin, S. Pati, A. Kotrotsou, M. Milchenko, W. Xu, D. Marcus, R. R. Colen, and S. Bakas, "Federated learning in medicine: facilitating multi-institutional collaborations without sharing patient data," *Scientific Reports*, vol. 10, no. 1, p. 12598, 2020. [Online]. Available: https://doi.org/10.1038/s41598-020-69250-1

[116] P. Amin, N. R. Anikireddypally, S. Khurana, S. Vadakkemadathil, and W. Wu, "Personalized health monitoring using predictive analytics," in *2019 IEEE Fifth International Conference on Big Data Computing Service and Applications (BigDataService)*, 2019, pp. 271–278.

[117] R. B. Ash, *Probability and Measure Theory*, 2nd ed. New York: Academic Press, 2000.

[118] P. R. Halmos, *Measure Theory*, New York, 1974.

[119] P. Billingsley, *Probability and Measure*, 3rd ed., New York, 1995.

[120] C. Dwork and A. Roth, "The algorithmic foundations of differential privacy," *Foundations and Trends® in Theoretical Computer*

Science, vol. 9, no. 3–4, pp. 211–407, 2014. [Online]. Available: http://dx.doi.org/10.1561/0400000042

[121] S. Asoodeh, J. Liao, F. P. Calmon, O. Kosut, and L. Sankar, "A Better Bound Gives a Hundred Rounds: Enhanced Privacy Guarantees via f-Divergences," *arXiv e-prints*, p. arXiv:2001.05990, Jan. 2020.

[122] I. Mironov, "Rényi differential privacy," in *2017 IEEE 30th Computer Security Foundations Symposium (CSF)*, 2017, pp. 263–275.

[123] S. M. Kay, *Fundamentals of statistical signal processing. Vol. 2., Detection theory*, ser. Prentice-Hall signal processing series. Upper Saddle River, NJ: Prentice-Hall PTR, 1998.

[124] P. Kairouz, S. Oh, and P. Viswanath, "The composition theorem for differential privacy," in *Proceedings of the 32nd International Conference on Machine Learning*, ser. Proceedings of Machine Learning Research, F. Bach and D. Blei, Eds., vol. 37. Lille, France: PMLR, 07–09 Jul 2015, pp. 1376–1385. [Online]. Available: https://proceedings.mlr.press/v37/kairouz15.html

[125] Q. Geng and P. Viswanath, "The optimal noise-adding mechanism in differential privacy," *IEEE Transactions on Information Theory*, vol. 62, no. 2, pp. 925–951, 2016.

[126] H. Shu and H. Zhu, "Sensitivity analysis of deep neural networks," in *Proceedings of the Thirty-Third AAAI Conference on Artificial Intelligence*, ser. AAAI'19/IAAI'19/EAAI'19. AAAI Press, 2019. [Online]. Available: https://doi.org/10.1609/aaai.v33i01.33014943

[127] R. Busa-Fekete, A. Munoz-Medina, U. Syed, and S. Vassilvitskii, "Label differential privacy and private training data release," in *Proceedings of the 40th International Conference on Machine Learning*, ser. Proceedings of Machine Learning Research, vol. 202. PMLR, 23–29 Jul 2023, pp. 3233–3251. [Online]. Available: https://proceedings.mlr.press/v202/busa-fekete23a.html

[128] B. Balle, G. Barthe, and M. Gaboardi, "Privacy amplification by subsampling: tight analyses via couplings and divergences," in *Proceedings of the 32nd International Conference on Neural Information Processing Systems*, ser. NIPS'18. Red Hook, NY, USA: Curran Associates Inc., 2018, pp. 6280–6290.

[129] P. Cuff and L. Yu, "Differential privacy as a mutual information constraint," in *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*, ser. CCS '16. New York, NY, USA: Association for Computing Machinery, 2016, pp. 43–54. [Online]. Available: https://doi.org/10.1145/2976749.2978308

[130] A. Makhdoumi, S. Salamatian, N. Fawaz, and M. Médard, "From the information bottleneck to the privacy funnel," in *2014 IEEE Information Theory Workshop (ITW 2014)*, 2014, pp. 501–505.

[131] G. Lugosi and S. Mendelson, "Robust multivariate mean estimation: The optimality of trimmed mean," *Annals of Statistics*, vol. 49, no. 1, pp. 393–410, Feb. 2021, publisher Copyright: © Institute of Mathematical Statistics, 2021.

[132] A. Turner, D. Tsipras, and A. Madry, "Clean-label backdoor attacks," 2019. [Online]. Available: https://openreview.net/forum?id=HJg6e2CcK7

[133] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein, *Introduction to Algorithms*. Cambridge, MA, USA: MIT Press, 2022. [Online]. Available: http://ebookcentral.proquest.com/lib/aalto-ebooks/detail.action?docID=6925615

[134] M. Sipser, *Introduction to the Theory of Computation*, 3rd ed. Andover, U.K.: Cengage Learning, 2013.

[135] R. Motwani and P. Raghavan, *Randomized Algorithms*. Cambridge, U.K.: Cambridge Univ. Press, 1995.

[136] R. G. Gallager, *Stochastic Processes: Theory for Applications*. Cambridge University Press, 2013.

[137] M. P. Salinas et al., "A systematic review and meta-analysis of artificial intelligence versus clinicians for skin cancer diagnosis," *npj Digit. Med.*, vol. 7, no. 1, May 2024, Art. no. 125, doi: 10.1038/s41746-024-01103-x.

[138] G. F. Cooper, "The computational complexity of probabilistic inference using bayesian belief networks," *Artif. Intell.*, vol. 42, no. 2–3, pp. 393–405, Mar. 1990, doi: 10.1016/0004-3702(90)90060-D.

[139] R. Gray, *Probability, Random Processes, and Ergodic Properties*, 2nd ed. New York: Springer, 2009.

[140] E. F. Codd, "A relational model of data for large shared data banks," *Commun. ACM*, vol. 13, no. 6, pp. 377–387, Jun. 1970, doi: 10.1145/362384.362685.

[141] European Union, "Regulation (EU) 2018/1725 of the European Parliament and of the Council of 23 October 2018 on the protection of natural persons with regard to the processing of personal data by the Union institutions, bodies, offices and agencies and on the free movement of such data, and repealing Regulation (EC) No 45/2001 and Decision No 1247/2002/EC (Text with EEA relevance)," L 295/39, Nov. 21, 2018. [Online]. Available: https://eur-lex.europa.eu/eli/reg/2018/1725/oj

[142] T. M. Cover and J. A. Thomas, *Elements of Information Theory*, 2nd ed., New Jersey, 2006.

[143] X. Liu, H. Li, G. Xu, Z. Chen, X. Huang, and R. Lu, "Privacy-enhanced federated learning against poisoning adversaries," *IEEE Transactions on Information Forensics and Security*, vol. 16, pp. 4574–4588, 2021.

[144] J. Zhang, B. Chen, X. Cheng, H. T. T. Binh, and S. Yu, "Poisongan: Generative poisoning attacks against federated learning in edge computing systems," *IEEE Internet of Things Journal*, vol. 8, no. 5, pp. 3310–3322, 2021.

[145] A. Silberschatz, H. F. Korth, and S. Sudarshan, *Database System Concepts*, 7th ed. New York, NY, USA: McGraw-Hill Education, 2019. [Online]. Available: https://db-book.com/

[146] S. Abiteboul, R. Hull, and V. Vianu, *Foundations of Databases*. Reading, MA, USA: Addison-Wesley Publishing Company, 1995.

[147] S. Hoberman, *Data Modeling Made Simple: A Practical Guide for Business and IT Professionals*, 2nd ed. Basking Ridge, NJ, USA: Technics Publications, 2009.

[148] R. Ramakrishnan and J. Gehrke, *Database Management Systems*, 3rd ed. New York, NY, USA: McGraw-Hill, 2002.

[149] T. Gebru, J. Morgenstern, B. Vecchione, J. Vaughan, H. Wallach, H. Daumé, and K. Crawford, "Datasheets for datasets," *Commun. ACM*, vol. 64, no. 12, pp. 86–92, nov 2021. [Online]. Available: https://doi.org/10.1145/3458723

[150] T. Hastie, R. Tibshirani, and J. Friedman, *The Elements of Statistical Learning*, ser. Springer Series in Statistics. New York, NY, USA: Springer, 2001.

[151] N. Cesa-Bianchi and G. Lugosi, *Prediction, Learning, and Games*. New York, NY, USA: Cambridge University Press, 2006.

[152] E. Hazan, *Introduction to Online Convex Optimization*. Now Publishers Inc., 2016.

[153] J. Chen, L. Song, M. Wainwright, and M. Jordan, "Learning to explain: An information-theoretic perspective on model interpretation," in *Proc. 35th Int. Conf. on Mach. Learning*, Stockholm, Sweden, 2018.

[154] D. Gujarati and D. Porter, *Basic Econometrics*. Mc-Graw Hill, 2009.

[155] Y. Dodge, *The Oxford Dictionary of Statistical Terms*. Oxford University Press, 2003.

[156] B. Everitt, *Cambridge Dictionary of Statistics*. Cambridge University Press, 2002.

[157] M. Ribeiro, S. Singh, and C. Guestrin, ""Why should i trust you?": Explaining the predictions of any classifier," in *Proc. 22nd ACM SIGKDD*, Aug. 2016, pp. 1135–1144.

[158] A. Papoulis and S. U. Pillai, *Probability, Random Variables, and Stochastic Processes*, 4th ed. New York: Mc-Graw Hill, 2002.

[159] C. E. Rasmussen and C. K. I. Williams, *Gaussian Processes for Machine Learning*. Cambridge, MA, USA: MIT Press, 2006.

[160] S. Ross, *A First Course in Probability*, 9th ed. Boston, MA, USA: Pearson Education, 2014.

[161] J. Su, D. V. Vargas, and K. Sakurai, "One pixel attack for fooling deep neural networks," *IEEE Trans. Evol. Comput.*, vol. 23, no. 5, pp. 828–841, Oct. 2019, doi: 10.1109/TEVC.2019.2890858.

[162] C. Lampert, "Kernel methods in computer vision," *Foundations and Trends in Computer Graphics and Vision*, 2009.

[163] U. von Luxburg, "A tutorial on spectral clustering," *Statistics and Computing*, vol. 17, no. 4, pp. 395–416, Dec. 2007.

[164] A. Y. Ng, M. I. Jordan, and Y. Weiss, "On spectral clustering: Analysis and an algorithm," in *Adv. Neur. Inf. Proc. Syst.*, 2001.

[165] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. Gomez, Ł. Kaiser, and I. Polosukhin, "Attention is all you need," in *Advances in neural information processing systems*, vol. 30, 2017, pp. 5998–6008.

[166] R. Caruana, "Multitask learning," *Machine Learning*, vol. 28, no. 1, pp. 41–75, 1997. [Online]. Available: https://doi.org/10.1023/A:1007379606734

[167] A. Jung, G. Hannak, and N. Görtz, "Graphical LASSO Based Model Selection for Time Series," *IEEE Sig. Proc. Letters*, vol. 22, no. 10, Oct. 2015.

[168] A. Jung, "Learning the conditional independence structure of stationary time series: A multitask learning approach," vol. 63, no. 21, Nov. 2015.

[169] C. Rudin, "Stop explaining black box machine learning models for high-stakes decisions and use interpretable models instead," *Nature Machine Intelligence*, vol. 1, no. 5, pp. 206–215, 2019.

[170] T. Hastie, R. Tibshirani, and M. Wainwright, *Statistical Learning with Sparsity. The Lasso and its Generalizations*, 2015.

[171] A. Lapidoth, *A Foundation in Digital Communication.* New York: Cambridge University Press, 2009.

[172] R. A. Horn and C. R. Johnson, *Matrix Analysis*, 2nd ed., Cambridge, UK, 2013.

[173] A. Ünsal and M. Önen, "Information-theoretic approaches to differential

privacy," *ACM Comput. Surv.*, vol. 56, no. 3, Oct. 2023, Art. no. 76, doi: 10.1145/3604904.

[174] O. Kallenberg, *Foundations of modern probability.* New York: Springer, 1997.

[175] I. Csiszar, "Generalized cutoff rates and Renyi's information measures," *IEEE Transactions on Information Theory*, vol. 41, no. 1, pp. 26–34, 1995.

[176] S. Bubeck, "Convex optimization. algorithms and complexity." in *Foundations and Trends in Machine Learning.* Now Publishers, 2015, vol. 8.

[177] L. Bottou, "On-line learning and stochastic approximations," in *On-Line Learning in Neural Networks*, D. Saad, Ed. New York, NY, USA: Cambridge Univ. Press, 1999, ch. 2, pp. 9–42.

[178] C. Gallese, "The AI act proposal: A new right to technical interpretability?," *SSRN Electron. J.*, Feb. 2023. [Online]. Available: https://ssrn.com/abstract=4398206

[179] M. M. et.al., "Model cards for model reporting," in *Proceedings of the Conference on Fairness, Accountability, and Transparency*, ser. FAT* '19. New York, NY, USA: Association for Computing Machinery, 2019, pp. 220–229. [Online]. Available: https://doi.org/10.1145/3287560.3287596

[180] K. Shahriari and M. Shahriari, "IEEE standard review — Ethically aligned design: A vision for prioritizing human wellbeing with artificial

intelligence and autonomous systems," in *2017 IEEE Canada International Humanitarian Technology Conference*, pp. 197–201, doi: 10.1109/IHTC.2017.8058187.

[181] D. Pfau and A. Jung, "Engineering trustworthy AI: A developer guide for empirical risk minimization," Nov. 2024. [Online]. Available: https://arxiv.org/abs/2410.19361