Federated Learning

From Theory to Practice

Alexander Jung

November 29, 2025

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

Preface

This book offers a hands-on introduction to building and understanding federated learning (FL) systems. FL enables multiple devices – such as smartphones, sensors, or local computers – to collaboratively train machine learning (ML) models, while keeping their data private and local. It is a powerful solution when data cannot or should not be centralized due to privacy, regulatory, or technical reasons.

The book is designed for students, engineers, and researchers who want to learn how to design scalable, privacy-preserving FL systems. Our main focus is on personalization: enabling each device to train its own model while still benefiting from collaboration with relevant devices. This is achieved by leveraging similarities between the learning tasks associated with devices. We represent these similarities as weighted edges of a federated learning network (FL network).

The key idea is to represent real-world FL systems as networks of devices, where nodes correspond to device and edges represent communication links and data similarities between them. The training of personalized models for these devices can be naturally framed as a distributed optimization problem. This optimization problem is referred to as generalized total variation minimization (GTVMin) and ensures that devices with similar learning tasks learn similar model parameters.

Our approach is both mathematically principled and practically motivated. While we introduce some advanced ideas from optimization theory and graph-based learning, we aim to keep the book accessible. Readers are guided through the core ideas step-by-step, with intuitive explanations. Throughout,

we maintain a focus of building FL systems that are trustworthy—robust against attacks, privacy-friendly, and secure.

Audience. We assume a basic background in undergraduate-level mathematics, including calculus and linear algebra. Familiarity with concepts such as convergence, derivatives, and norms will be helpful but not strictly necessary. No prior experience with ML or optimization is required, as we build up most concepts from first principles.

The book is intended for advanced undergraduates, graduate students, and practitioners who are looking for a practical, principled, and privacy-friendly approach to decentralized ML.

Structure. The book begins by introducing the key motivations and challenges of FL. We then move on to introduce the notion of an FL network and explain how they capture the structure of distributed ML applications. The core chapters develop the GTVMin formulation and explore how to solve it using various distributed optimization techniques. Later chapters focus on practical concerns such as robustness and privacy protection of GTVMin-based systems. A comprehensive glossary is also included to better support the reader.

Acknowledgements. The development of this book has greatly benefited from feedback and insights gathered during the course *CS-E4740 Federated Learning* at Aalto University, taught between 2023 and 2025. I am grateful to Bo Zheng, Olga Kuznetsova, Diana Pfau, and Shamsiiat Abdurakhmanova for their thoughtful comments on early drafts. Special thanks go to Ekkehard Schnoor and Mikko Seesto for their careful proofreading of the manuscript and to Konstantina Olioumtsevits for her meticulous revision of the glossary.

Some of the figures in the glossary have been prepared with the help of Salvatore Rastelli and Juliette Gronier.

This work was 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).*

Contents

Lists of Symbols

Sets and Functions

$a \in \mathcal{A}$	The object a is an element of the set $\mathcal{A}$.
$a := b$	Depending on the context, we use the symbol $:=$ either to mean a definition or to mean an assignment (e.g., within a pseudocode for an algorithm.
$\mathcal{A} \subseteq \mathcal{B}$	$\mathcal{A}$ 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 $x > 0$.
$\lvert x \rvert$	The absolute value of a real number $x \in \mathbb{R}$.

$\{0,1\}$	The set consisting of the two real numbers 0 and 1.
$[0,1]$	The closed interval of real numbers x with $0 \leq x \leq 1$.
$\arg\min_{\mathbf{w}} f(\mathbf{w})$	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 co-domain of h. 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

$\mathbf{a} = \big(a_1, \ldots, a_d\big)^T$	A vector of length d, with its j-th entry being a_j.
$\mathbb{R}^d$	The set of vectors $\mathbf{a} = \big(a_1, \ldots, a_d\big)^T$ consisting of d real-valued entries $a_1, \ldots, a_d \in \mathbb{R}$.
$\mathbf{I}_d, \mathbf{I}$	A square identity matrix of size $d \times d$. If the size is clear from context, we drop the subscript.
$\|\mathbf{a}\|_2$	The Euclidean (or ℓ_2) norm of the vector $\mathbf{a} = \big(a_1, \ldots, a_d\big)^T \in \mathbb{R}^d$ defined as $\|\mathbf{a}\|_2 := \sqrt{\sum_{j=1}^d a_j^2}$.
$\|\mathbf{a}\|$	Some norm of the vector $\mathbf{a} \in \mathbb{R}^d$ [3]. Unless specified otherwise, we mean the Euclidean norm $\|\mathbf{a}\|_2$.
$\mathbf{a}^T$	The transpose of a matrix that has the vector $\mathbf{a} \in \mathbb{R}^d$ as its single column.
$\mathbf{A}^T$	The transpose of a matrix $\mathbf{A} \in \mathbb{R}^{m \times d}$. A square real-valued matrix $\mathbf{A} \in \mathbb{R}^{m \times m}$ is called symmetric if $\mathbf{A} = \mathbf{A}^T$.
$\mathbf{0} = \big(0, \ldots, 0\big)^T$	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.
$\lambda_j\big(\mathbf{Q}\big)$	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.

$\left(\mathbf{v}^T, \mathbf{w}^T\right)^T$	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 $\mathbf{B}$, $\text{span}\{\mathbf{B}\} = \left\{\mathbf{Ba} : \mathbf{a} \in \mathbb{R}^b\right\} \subseteq \mathbb{R}^a$.
$\mathbf{A} \otimes \mathbf{B}$	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 $\mathbb{P}(\mathbf{z})$. If the probability distribution is clear from context, we just write $\mathbb{E}\{f(\mathbf{z})\}$.
$\mathbb{P}(\mathbf{x};\mathbf{w})$	A parametrized probability distribution of an RV $\mathbf{x}$. The probability distribution depends on a parameter vector $\mathbf{w}$. For example, $\mathbb{P}(\mathbf{x};\mathbf{w})$ 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.
$\mathcal{B}$	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.
$\mathbf{x}^{(r)}$	The feature vector of the r-th data point within a dataset.

$\left(\mathbf{x}^{(r)}, y^{(r)}\right)$	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.
$\mathcal{H}$	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.
$d_{\text{eff}}\left(\mathcal{H}\right)$	The effective dimension of a hypothesis space $\mathcal{H}$.
$L\left((\mathbf{x}, y), h\right)$	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.

$\mathcal{R}\{h\}$	A regularizer that assigns a hypothesis h a measure for the anticipated increase in average loss when h is applied to data points outside the training set.
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).
α	A regularization parameter that controls the amount of regularization.
$\mathbf{w}$	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 artificial neural network (ANN).
$h^{(\mathbf{w})}(\cdot)$	A hypothesis map that involves tunable model parameters $w_1, \ldots, w_d$ stacked into the vector $\mathbf{w} = \big(w_1, \ldots, w_d\big)^T$.
$\mathbf{\Phi}(\cdot)$	A feature map $\mathbf{\Phi} : \mathcal{X} \to \mathcal{X}' : \mathbf{x} \mapsto \mathbf{x}' := \phi\big(\mathbf{x}\big) \in \mathcal{X}'$.

$\mathbf{X}$	The feature matrix $\mathbf{X} = \left(\mathbf{x}^{(1)}, \ldots, \mathbf{x}^{(m)}\right)^T \in \mathbb{R}^{m \times d}$ of a dataset, consisting of m data points each characterized by a feature vector $\mathbf{x}^{(r)}$, for $r = 1, \ldots, m$.
$\mathbf{y}$	The label vector $\mathbf{y} = \left(y^{(1)}, \ldots, y^{(m)}\right)^T \in \mathbb{R}^m$ of a dataset, consisting of m data points each characterized by a label $y^{(r)}$, for $r = 1, \ldots, m$.

Federated Learning

$\mathcal{G} = (\mathcal{V}, \mathcal{E})$	An undirected graph whose nodes $i \in \mathcal{V}$ represent devices within an FL network. The undirected edges $\{i, i'\} \in \mathcal{E}$, each having a positive weight $A_{i,i'}$, represent either some form of connectivity between devices or statistical similarities between their local datasets.
$i \in \mathcal{V}$	A node that represents some device within an FL network. The device can access a local dataset and train a local model.
$\mathcal{C}$	Given a graph we denote by $\mathcal{C} \subseteq \mathcal{V}$ a subset (or cluster) of nodes which are connected by many edges with large weights.
$\lvert\partial\mathcal{C}\rvert$	The boundary of a cluster, which is the sum $\sum_{i \in \mathcal{C}, i' \notin \mathcal{C}} A_{i,i'}$.
$\mathcal{G}^{(\mathcal{C})}$	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.
$\mathcal{R}^{(i)}\big\{h\big\}$	A regularizer used for model training by device i within a FL network. This regularizer typically depends on the model parameters of other devices $i' \in \mathcal{V} \setminus \{i\}$.
$d^{(i,i')}$	A quantitative measure for the variation (or discrepancy) between trained local models at nodes i, i'.
$\text{stack}\big\{\mathbf{w}^{(i)}\big\}_{i=1}^{n}$	The vector $\Big(\big(\mathbf{w}^{(1)}\big)^T, \ldots, \big(\mathbf{w}^{(n)}\big)^T\Big)^T \in \mathbb{R}^{dn}$ that is obtained by vertically stacking the local model parameters $\mathbf{w}^{(i)} \in \mathbb{R}^d$.

1 Introduction to Federated Learning

We are surrounded by devices, such as smartphones and wearables, that generate decentralized collections of local datasets [5–9]. These local datasets often exhibit an intrinsic network structure, arising from functional dependencies or statistical similarities (see Chapter 7.3).

For example, contact networks underpin pandemic modeling, network medicine maps disease relationships via co-morbidities [10], and social sciences leverage social graphs to relate the data of connected individuals [11]. Similarly, weather stations of the Finnish Meteorological Institute (FMI) produce local datasets with statistical properties influenced by geographic proximity.

Federated learning (FL) is an umbrella term for distributed optimization methods that train machine learning (ML) models directly at the locations of data generation [12–16]. Unlike traditional ML workflows that centralize data before training, FL leverages in-situ computations. Figure 1.1 contrasts these approaches.

From an engineering perspective, this book is about building federated learning systems by formulating them as network-based optimization problems. The core idea is to represent a real-world FL setup via an federated learning network (FL network), where nodes correspond to devices with local datasets and models, and edges reflect communication capabilities or statistical similarity. We then pose FL as an optimization problem over this FL network, which we call generalized total variation minimization (GTVMin). GTVMin balances local model performance with smoothness of model parameters across connected nodes.

Different choices for how to measure model variation across the FL network

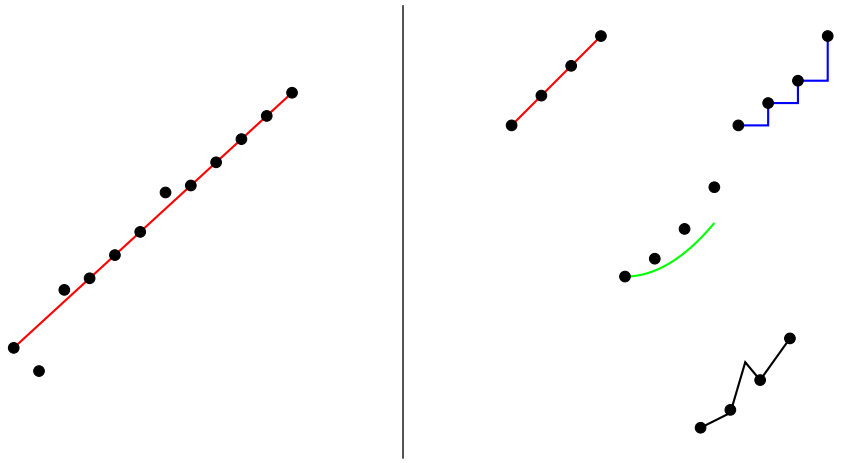


Fig. 1.1. Left: A basic ML method uses a single dataset to train a single model. Right: Decentralized collection of devices with the ability to access data and train models locally.

lead to different flavors of FL methods. The overarching goal is to derive these methods in a principled way by applying distributed optimization methods. All FL algorithms we study can be seen as fixed-point iterations for solving an instance of GTVMin.

Beyond methodology, FL is also driven by several practical motivations:

- **Privacy.** By exchanging only updates to model parameters, FL avoids raw data transmission and thus mitigates privacy risks (see Chapter 9).
- **Robustness.** FL systems can tolerate stragglers and are more resilient to cyber-attacks, such as data poisoning (see Chapter 10).
- **Parallelism.** We can interpret the interconnected devices of a FL network 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.** FL enables collective learning using low-cost, widely available devices – rather than relying on centralized high-end hardware [17, 18].
- **Communication Efficiency.** In remote or bandwidth-limited scenarios, training locally can be cheaper than transmitting raw datasets [19].
- **Personalization.** FL naturally supports training personalized models that adapt to device-specific data distributions (see Chapter 6).

1.1 Core Techniques in Federated Learning

To build and analyze FL algorithms, this book draws on core mathematical concepts:

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 $\mathbb{R}^d$ [20, 21]. 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 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 operator $\mathcal{P} : \mathbb{R}^d \to \mathbb{R}^d$. These operators depend on the local datasets and personal models used within an FL system. A prime example of such an operator is the gradient step of gradient-based methods (see Chapter 4). The computational properties of these FL algorithms are determined by the contraction properties of the underlying operator [22].

1.2 Book Structure and Roadmap

This book is organized into three parts:

- **Part I: ML Refresher.** Chapters 2 and 4 review basic ML concepts and optimization methods. These chapters serve both to refresh prerequisite knowledge and to highlight techniques like regularization and gradient descent that underpin FL.

- **Part II: FL Theory and Methods.** Chapter 3 introduces the FL network and formulates the core optimization principle, i.e., GTVMin. Chapters 4 and 5 show how to apply optimization methods to derive scalable and personalized FL algorithms. Chapter 6 explores main FL variants as special cases of GTVMin, and Chapter 7 discusses methods for constructing meaningful edge structures in FL networks.

- **Part III: trustworthy artificial intelligence (trustworthy AI).** Chapters 8–10 explore key requirements for trustworthy AI systems,

including privacy protection and robustness against data poisoning. These chapters link FL methodology to emerging ethical and regulatory demands in AI deployment.

1.3 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 Machine Learning Foundations for FL

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.

In Section 2.3, we begin with the basic components of any ML method: data, a model and a loss functions. We also describe how these components are combined through empirical risk minimization (ERM) which is a main design principle for ML.

Section 2.2 then explores the computational aspects of ERM, focusing on gradient-based methods for parametric models. Section 2.3 discusses the statistical properties of ML methods and their analysis via probabilistic models. Section 2.4 introduces the idea of model validation and discusses simple rules for the diagnosis of ML methods.

Section 2.5 explains three fundamental forms of regularization: data augmentation, model pruning and loss penalization. We will then show in Section 2.6 how to use regularization to couple the training of local models at different devices, resulting in FL.

2.1 Components of ML Systems: A Design Framework

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 of 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 $\mathbf{z} = (\mathbf{x}, y)$.

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 $\mathbf{z}$.

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 [23, Ch. 2].

It seems natural to learn a hypothesis by minimizing 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,

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

As the notation in (1) indicates, there can be several different solutions to the optimization problem (1). We denote by $\hat{h}$ one of these solutions, i.e., $\hat{h}$ is an element of the solution set for (1).

Several important 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 $h^{(\mathbf{w})}$. A prime example of a parametric model is the linear model [23, Sec. 3.1],

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

This book presents FL algorithms (see Chapter 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 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 representing a trainable feature map and the output layer implements a linear model [24], [23, Sec. 3.11.],
- a decision tree with a fixed topology that corresponds to a specific decision boundary and trainable predictions for each decision region [25], [23, Sec. 3.10],
- kernel methods [26], [23, Sec. 3.9].

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

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

Note that (2) minimizes a smooth and convex function

$$f(\mathbf{w}) := (1/m)\Big[\mathbf{w}^T\mathbf{X}^T\mathbf{X}\mathbf{w} - 2\mathbf{y}^T\mathbf{X}\mathbf{w} + \mathbf{y}^T\mathbf{y}\Big]. \tag{3}$$

Here, we use the feature matrix

$$\mathbf{X} := \big(\mathbf{x}^{(1)},\ldots,\mathbf{x}^{(m)}\big)^T \in \mathbb{R}^{m\times d} \tag{4}$$

and the label vector

$$\mathbf{y} := \big(y^{(1)},\ldots,y^{(m)}\big)^T \in \mathbb{R}^{m} \tag{5}$$

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

Inserting (3) into (2) allows to formulate linear regression as

$$\widehat{\mathbf{w}}^{(\mathrm{LR})} \in \operatorname*{arg\,min}_{\mathbf{w}\in\mathbb{R}^d} \mathbf{w}^T\mathbf{Q}\mathbf{w} + \mathbf{w}^T\mathbf{q} \tag{6}$$
$$\text{with } \mathbf{Q} := (1/m)\mathbf{X}^T\mathbf{X}, \mathbf{q} := -(2/m)\mathbf{X}^T\mathbf{y}.$$

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

$$\mathbf{Q} = \sum_{j=1}^{d} \lambda_j \mathbf{u}^{(j)} \big(\mathbf{u}^{(j)}\big)^T. \tag{7}$$

The EVD (7) consists of orthonormal eigenvectors $\mathbf{u}^{(1)}, \ldots, \mathbf{u}^{(d)}$ and corresponding list of non-negative eigenvalues

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

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.

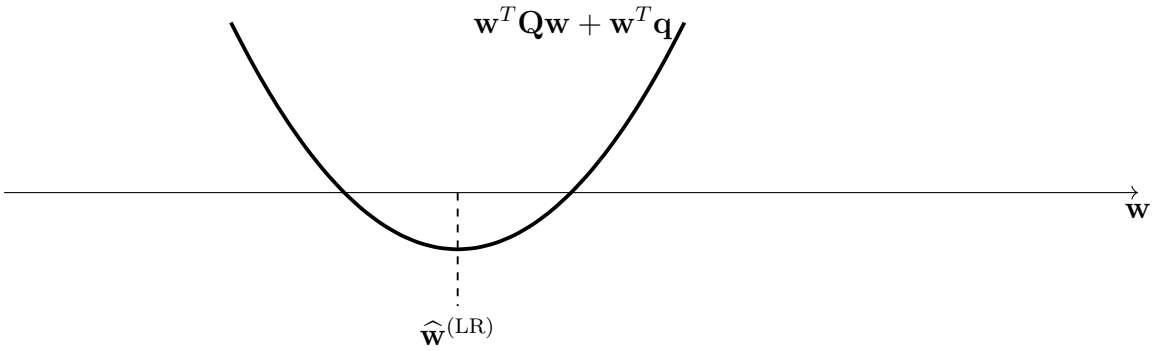


Fig. 2.1. ERM (1) for linear regression minimizes a convex quadratic function $\mathbf{w}^T\mathbf{Q}\mathbf{w} + \mathbf{w}^T\mathbf{q}$.

To train a ML model $\mathcal{H}$ means to solve ERM (1) (or (2) for linear regression); the dataset $\mathcal{D}$ is therefore referred to as a training set. The trained model results in the learned hypothesis $\hat{h}$. Two key questions for the analysis of a given ML method are:

- **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 $\mathbf{x}$?

2.2 Computational Aspects of ERM

A principled approach to design ML methods is to apply some optimization method to solve (1) [27]. Most of these optimization methods operate in an iterative fashion: 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 (2) by gradient-based methods: Starting from an initial parameters $\mathbf{w}^{(0)}$, we iterate the gradient step:

$$\begin{aligned} \mathbf{w}^{(t)} &:= \mathbf{w}^{(t-1)} - \eta \nabla f\big(\mathbf{w}^{(t-1)}\big) \\ &\overset{(3)}{=} \mathbf{w}^{(t-1)} + (2\eta/m) \sum_{r=1}^{m} \mathbf{x}^{(r)} \big(y^{(r)} - \big(\mathbf{w}^{(t-1)}\big)^T \mathbf{x}^{(r)}\big). \end{aligned} \tag{8}$$

This gradient update can be compactly expressed using the feature matrix

(4) and label vector (5) as[1]

$$\mathbf{w}^{(t)} = \mathbf{w}^{(t-1)} + \eta \cdot \frac{2}{m} \mathbf{X}^{\mathrm{T}} \big(\mathbf{y} - \mathbf{X}\mathbf{w}^{(t-1)}\big).$$

How much computation do we need for one iteration of (8)? 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 (8) then typically requires around $m \cdot d$ arithmetic operations (additions and multiplications).

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 (2) has a simple closed-form solution:

$$\widehat{w} = (1/m) \sum_{r=1}^{m} y^{(r)}. \tag{9}$$

Thus, for this special case of the linear model, solving (9) is to sum m numbers $y^{(1)}, \ldots, y^{(m)}$. The amount of computation, measured by the number of elementary arithmetic operations, required by (9) is proportional to m.

2.3 Statistical Aspects of ERM

We can train a linear model on a given training set as ERM (2). But how useful is the solution $\widehat{\mathbf{w}}$ of (2) for predicting the labels of data points outside

[1]The gradient of the objective function (3) can be expressed as

$$\nabla f(\mathbf{w}) = -\frac{2}{m} \mathbf{X}^{\mathrm{T}} (\mathbf{y} - \mathbf{X}\mathbf{w}).$$

the training set? Consider applying the learned 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.

A widely used approach to study the generalization of ML methods uses a simple probabilistic models: The idea is to interpret data points as independent and identically distributed (i.i.d.) random variables (RVs) with common 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)

$$\mathbb{E}\{L\left((\mathbf{x}, y), h\right)\}. \tag{10}$$

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

$$y = \overline{\mathbf{w}}^T \mathbf{x} + \varepsilon \text{ with } \mathbf{x} \sim \mathcal{N}(\mathbf{0}, \mathbf{I}), \varepsilon \sim \mathcal{N}(0, \sigma^2), \mathbb{E}\{\varepsilon \mathbf{x}\} = \mathbf{0}. \tag{11}$$

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

$$\mathbb{E}\{(y - h(\mathbf{x}))^2\} = \|\overline{\mathbf{w}} - \widehat{\mathbf{w}}\|^2 + \sigma^2. \tag{12}$$

Strictly speaking, (12) only holds for constant model parameters $\widehat{\mathbf{w}}$. However, the learned model parameters $\widehat{\mathbf{w}}$ are often the output of a ML method that is applied to a dataset $\mathcal{D}$. If we interpret the data points in $\mathcal{D}$ as i.i.d. realizations from some underlying probability distribution, we can replace the expectation on the LHS of (12) with the conditional expectation $\mathbb{E}\big\{(y - h(\mathbf{x}))^2 \big| \mathcal{D}\big\}$ [28].

The first component in (12) 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 (2)) for the parameters of a linear hypothesis. The second component σ^2 in (12) 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 $\widehat{\mathbf{w}} = \widehat{\mathbf{w}}^{(\mathrm{LR})}$ (6) delivered by linear regression methods. To this end, we first use the probabilistic model (11) to decompose the label vector $\mathbf{y}$ in (5) as

$$\mathbf{y} = \mathbf{X}\overline{\mathbf{w}} + \mathbf{n} \text{ , with } \mathbf{n} := \big(\varepsilon^{(1)}, \ldots, \varepsilon^{(m)}\big)^T. \tag{13}$$

Inserting (13) into (6) yields

$$\widehat{\mathbf{w}}^{(\mathrm{LR})} \in \operatorname*{arg\,min}_{\mathbf{w} \in \mathbb{R}^d} \mathbf{w}^T \mathbf{Q} \mathbf{w} + \mathbf{w}^T \mathbf{q}' + \mathbf{w}^T \mathbf{e} \tag{14}$$

$$\text{with } \mathbf{Q} := (1/m)\mathbf{X}^T\mathbf{X}, \mathbf{q}' := -(2/m)\mathbf{X}^T\mathbf{X}\overline{\mathbf{w}}, \text{ and } \mathbf{e} := -(2/m)\mathbf{X}^T\mathbf{n}. \tag{15}$$

Figure 2.2 depicts the objective function of (14). 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 (14).

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.[2]

Proposition 2.1. *Consider a solution $\widehat{\mathbf{w}}^{(\mathrm{LR})}$ to the ERM instance* (14) *that is applied to the dataset* (13)*. If the matrix $\mathbf{Q} = (1/m)\mathbf{X}^T\mathbf{X}$ is invertible,*

[2]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?

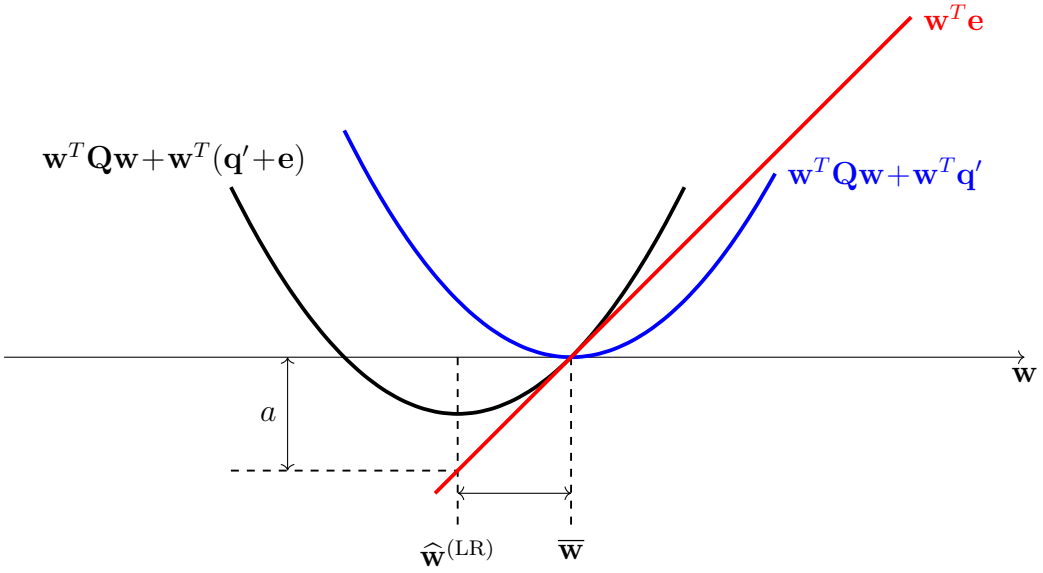


Fig. 2.2. The estimation error of linear regression is determined by the effect of the perturbation term $\mathbf{w}^T\mathbf{e}$ on the minimizer of the convex quadratic function $\mathbf{w}^T\mathbf{Q}\mathbf{w} + \mathbf{w}^T\mathbf{q}'$.

with minimum eigenvalue $\lambda_1(\mathbf{Q}) > 0$,

$$\big\|\widehat{\mathbf{w}}^{(\mathrm{LR})} - \overline{\mathbf{w}}\big\|_2^2 \leq \frac{\|\mathbf{e}\|_2^2}{\lambda_1^2} \overset{(15)}{=} \frac{4}{m^2}\frac{\big\|\mathbf{X}^T\mathbf{n}\big\|_2^2}{\lambda_1^2}. \tag{16}$$

Proof. Let us rewrite (14) as

$$\widehat{\mathbf{w}}^{(\mathrm{LR})} \in \operatorname*{arg\,min}_{\mathbf{w}\in\mathbb{R}^d} f(\mathbf{w}) \text{ with } f(\mathbf{w}) := \big(\mathbf{w}-\overline{\mathbf{w}}\big)^T\mathbf{Q}\big(\mathbf{w}-\overline{\mathbf{w}}\big)+\mathbf{e}^T\big(\mathbf{w}-\overline{\mathbf{w}}\big). \tag{17}$$

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,

$$\begin{aligned} f(\mathbf{w}) &\overset{(17)}{=} \big(\mathbf{w}-\overline{\mathbf{w}}\big)^T\mathbf{Q}\big(\mathbf{w}-\overline{\mathbf{w}}\big) + \mathbf{e}^T\big(\mathbf{w}-\overline{\mathbf{w}}\big) \\ &\overset{(a)}{\geq} \big(\mathbf{w}-\overline{\mathbf{w}}\big)^T\mathbf{Q}\big(\mathbf{w}-\overline{\mathbf{w}}\big) - \|\mathbf{e}\|_2\|\mathbf{w}-\overline{\mathbf{w}}\|_2 \\ &\overset{(b)}{\geq} \lambda_1\|\mathbf{w}-\overline{\mathbf{w}}\|_2^2 - \|\mathbf{e}\|_2\|\mathbf{w}-\overline{\mathbf{w}}\|_2. \end{aligned} \tag{18}$$

Step (a) used Cauchy–Schwarz inequality and (b) used the EVD (7) of $\mathbf{Q}$. Evaluating (18) for $\mathbf{w}=\widehat{\mathbf{w}}$ and combining with $f\big(\widehat{\mathbf{w}}\big) \leq 0$ yields (16). □

The bound (16) 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 the error term $\big\|\mathbf{X}^T\mathbf{n}\big\|_2^2$ in (16) by a factor of 100. For some applications, we can find feature transformations that increase $\lambda_1(\mathbf{Q})$ 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 (16) 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 (16) for the special case where each data point has the same feature value $x = 1$. Here, the probabilistic model (13) reduces to a "signal in noise" model [29],

$$y^{(r)} = x^{(r)}\overline{w} + \varepsilon^{(r)} \text{ with } x^{(r)} = 1, \tag{19}$$

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 (16) results in the bound

$$\left(\widehat{w}^{(\mathrm{LR})} - \overline{w}\right)^2 \leq 4\|\mathbf{n}\|_2^2/m^2.$$

For the labels and features in (19), the solution of (14) is given by

$$\widehat{w}^{(\mathrm{LR})} = (1/m)\sum_{r=1}^{m} y^{(r)} \overset{(19)}{=} \overline{w} + (1/m)\sum_{r=1}^{m} \varepsilon^{(r)}.$$

2.4 Validation and Diagnosis of ML

The above analysis of the generalization error started from postulating the probabilistic model (11) for the generation of data points. Strictly speaking, if the data points are not generated according to the probabilistic model the bound (16) does not apply. Thus, we might want to use a more data-driven approach for assessing the usefulness of a learned hypothesis $\hat{h}$ obtained, e.g., from solving ERM (1).

Loosely speaking, validation tries to find out if a learned hypothesis $\hat{h}$ performs similarly well inside and outside the training set. A basic form of validation is to compute the average loss of a learned 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 the some of the design choices for the dataset, the model and the loss function.

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

$$\widehat{h} \in \arg\min_{h \in \mathcal{H}} \sum_{(\mathbf{x},y) \in \mathcal{D}^{(\text{train})}} L\left(\left(\mathbf{x},y\right),h\right) \tag{20}$$

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

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 $E^{(\text{ref})}$. One important source for a baseline $E^{(\text{ref})}$ are probabilistic models for the data points.

Given a probabilistic model $p(\mathbf{x}, y)$, we can compute the minimum achievable risk (10). 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 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 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 [23, Ch. 7]).

- $E_t \approx E_v \gg E^{(\mathrm{ref})}$: The training error is on the same level as the validation error and both are significantly larger than the baseline. Thus, the learned hypothesis seems to not overfit the training set. However, the training error achieved by the learned 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 (20) 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 (20) is to approximate the risk (10) 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 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 learned hypothesis becomes more difficult. Figure 2.3) illustrates an extreme case of a validation set consisting of data points for which every hypothesis incurs a small average loss. Here, we might try to increase the size of the validation set by collecting more labelled data points or by using data augmentation. 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.

2.5 Regularization

Consider an ERM-based method with hypothesis space $\mathcal{H}$ and training set $\mathcal{D}$. A key indicator for the performance of such a ML method is the ratio

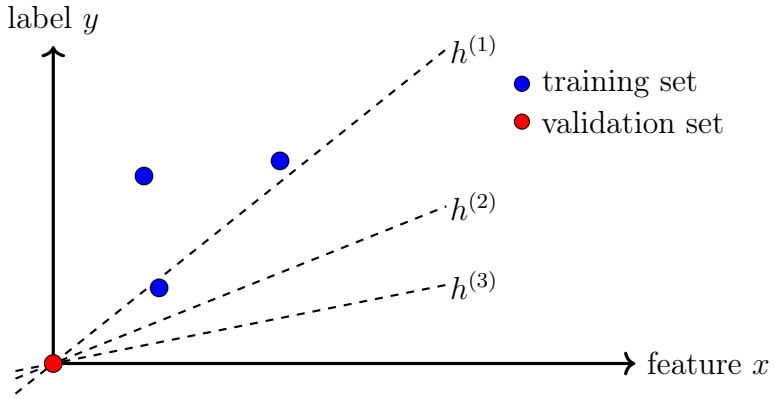


Fig. 2.3. An example of an unlucky split of a dataset into a training set and a validation set for the model $\mathcal{H} := \{h^{(1)}, h^{(2)}, h^{(3)}\}$.

$d_{\text{eff}}(\mathcal{H})/|\mathcal{D}|$ 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 approaches:

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

$$\hat{h} \in \arg\min_{h \in \mathcal{H}} (1/m) \sum_{r=1}^{m} L\left(\left(\mathbf{x}^{(r)}, y^{(r)}\right), h\right) + \alpha\mathcal{R}\{h\}, \tag{21}$$

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

As illustrated in Figure 2.4, these three forms of regularization are closely related [23, Ch. 7]. For example, the regularized ERM (21) 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 of regularization by 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

$$\widehat{\mathbf{w}}^{(\alpha)} \in \operatorname*{arg\,min}_{\mathbf{w}\in\mathbb{R}^d} \Big[(1/m)\sum_{r=1}^{m}\big(y^{(r)} - \mathbf{w}^T\mathbf{x}^{(r)}\big)^2 + \alpha\|\mathbf{w}\|_2^2\Big]. \tag{22}$$

The objective function in (22) 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

$$(\mathbf{x} + \mathbf{n}, y)\,, \text{ with } \mathbf{n} \sim \mathcal{N}(\mathbf{0}, \alpha\mathbf{I}). \tag{23}$$

Thus, ridge regression (22) 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 (23)). The label of each copy of $(\mathbf{x}, y)$ is equal to y, i.e., the label is not perturbed.

To study the computational aspects of ridge regression, we rewrite (22) as

$$\widehat{\mathbf{w}}^{(\alpha)} \in \operatorname*{arg\,min}_{\mathbf{w}\in\mathbb{R}^d} \mathbf{w}^T\mathbf{Q}\mathbf{w} + \mathbf{w}^T\mathbf{q},$$
$$\text{with } \mathbf{Q} := (1/m)\mathbf{X}^T\mathbf{X} + \alpha\mathbf{I},\ \mathbf{q} := (-2/m)\mathbf{X}^T\mathbf{y}. \tag{24}$$

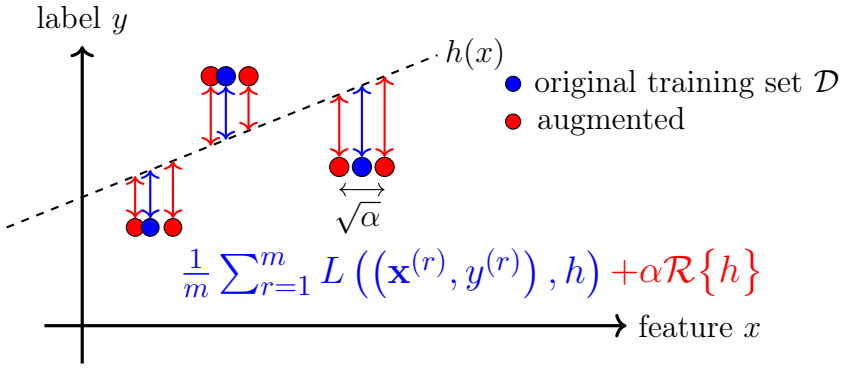


Fig. 2.4. Equivalence between data augmentation and loss penalization.

Thus, like linear regression (6), also ridge regression minimizes a convex quadratic function. A main difference between linear regression (6) and ridge regression (for $\alpha > 0$) is that the matrix $\mathbf{Q}$ in (24) is guaranteed to be invertible for any training set $\mathcal{D}$. In contrast, the matrix $\mathbf{Q}$ in (6) for linear regression might be singular for some training sets.[3]

The statistical properties of the solutions to (24) 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 $h(\mathbf{x}) = \left(\widehat{\mathbf{w}}^{(\alpha)}\right)^{T}\mathbf{x}$ learned by ridge regression with different values of α.

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

2.6 From ML to FL via Regularization

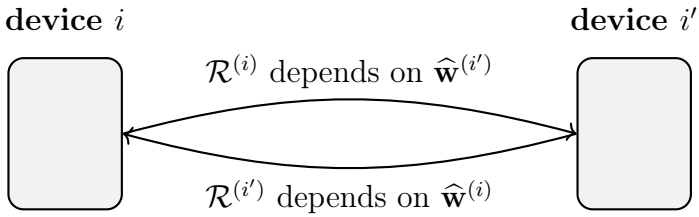
The main theme of this book is the analysis of FL systems that consists of a network of devices, indexed by $i = 1, \ldots, n$. Each device i trains a local (or personalized) model $\mathcal{H}^{(i)}$. One natural way to couple the model training of different devices is via regularization.

Assuming parametric models for ease of exposition, each device $i = 1, \ldots, n$ solves a separate instance of regularized empirical risk minimization (RERM) (21),[4]

$$\widehat{\mathbf{w}}^{(i)} \in \operatorname*{arg\,min}_{\mathbf{w}^{(i)} \in \mathbb{R}^d} \underbrace{(1/m) \sum_{r=1}^{m} L\left(\left(\mathbf{x}^{(r)}, y^{(r)}\right), h\right)}_{=: L_i\left(\mathbf{w}^{(i)}\right)} + \alpha \mathcal{R}^{(i)}\left\{\mathbf{w}^{(i)}\right\}. \tag{25}$$

We can couple the instances of (25) at device i with other devices $i' \in \mathcal{V} \setminus \{i\}$ by using a regularizer $\mathcal{R}^{(i)}\left\{\mathbf{w}^{(i)}\right\}$ that depends on their model parameters $\mathbf{w}^{(i')}$. For example, we will study constructions for $\mathcal{R}^{(i)}\left\{\mathbf{w}^{(i)}\right\}$ that penalize deviations between the model parameters $\mathbf{w}^{(i)}$ and those at other devices, $\mathbf{w}^{(i')}$ for $i' \in \mathcal{N}^{(i)}$. Figure 2.5 illustrates a simple network of two devices, each training a personalized model via (25). Chapter 3 will discuss in more detail how to construct a useful regularizer $\mathcal{R}^{(i)}\left\{\mathbf{w}^{(i)}\right\}$.

[4] It will be convenient to avoid explicit reference to the local dataset of device i and instead work with the local loss function $L_i\left(\mathbf{w}^{(i)}\right)$. The FL algorithms studied in this book require only access to $L_i\left(\mathbf{w}^{(i)}\right)$.



$$\widehat{\mathbf{w}}^{(i)} \in \arg\min_{\mathbf{w}^{(i)}} L_i\left(\mathbf{w}^{(i)}\right) + \alpha \mathcal{R}^{(i)}(\mathbf{w}^{(i)}) \qquad \widehat{\mathbf{w}}^{(i')} \in \arg\min_{\mathbf{w}^{(i')}} L_{i'}\left(\mathbf{w}^{(i')}\right) + \alpha \mathcal{R}^{(i')}(\mathbf{w}^{(i')})$$

Fig. 2.5. Two devices i and i' learn personalized model parameters. Each device executes a separate instance of regularized ERM with the regularizer depending on the model parameters of the other device.

2.7 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. RVs $\big(\mathbf{x}^{(t)}, y^{(t)}\big)$, for $t = 1, 2, \ldots$. The minimax risk is a lower bound on the risk achievable by any learning method [33, 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 $\alpha \|\mathbf{w}\|_2^2$ 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 $\alpha\|\mathbf{w}\|_2^2$ to the average squared error loss incurred by the hypothesis $h^{(\mathbf{w})}$ on the training set $\mathcal{D}$,

$$\min_{\mathbf{w}}(1/m)\sum_{r=1}^{m}\left(y^{(r)}-h\big(\mathbf{x}^{(r)}\big)\right)^2+\alpha\|\mathbf{w}\|_2^2. \tag{26}$$

Construct an augmented training set $\mathcal{D}'$ such that the objective function of (26) 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,

$$\min_{\mathbf{w}}(1/m)\sum_{r=1}^{m}\left(y^{(r)}-h\big(\mathbf{x}^{(r)}\big)\right)^2+\alpha\|\mathbf{w}-\widetilde{\mathbf{w}}\|_2^2. \tag{27}$$

Here, we used some prescribed reference model parameters $\widetilde{\mathbf{w}}$. Note that (27) reduces to basic ridge regression (26) for the specific choice $\widetilde{\mathbf{w}}=\mathbf{0}$.

3 A Design Principle for FL

Chapter 2 reviewed ERM as a central design principle for traditional, centralized ML systems that rely on a single dataset to train a single model. This chapter extends these foundations to the distributed setting of FL, where learning takes place over a network of devices, each having their own datasets and models.

We begin in Section 3.1 by introducing the notion of an FL network – a mathematical abstraction for FL systems. Each node of an FL network represents a device that collects a local dataset and trains a local model, while the edges encode communication links and statistical similarities between local datasets.

Section 3.2 introduces the concept of generalized total variation (GTV) as a measure of discrepancy between local model parameters at connected nodes. This notion leads directly to Section 3.3, where we develop GTVMin as a principled regularization framework for training parametric local models in a federated setting. We then generalize this approach in Section 3.4 to accommodate non-parametric local models, broadening its applicability. Finally, Section 3.5 offers several interpretations of GTVMin that connect it to broader themes in applied mathematics and statistics, highlighting its conceptual and practical significance in FL design.

3.1 FL Networks

Consider a FL system consisting of a collection of devices, indexed by $i = 1, \ldots, n$. The number n of devices can be arbitrarily large—potentially on

the order of billions—as encountered in internet-scale FL applications.Each device i can access a local dataset $\mathcal{D}^{(i)}$ and train a personalized model $\mathcal{H}^{(i)}$. These devices collaborate over a communication network to learn a local hypothesis $h^{(i)} \in \mathcal{H}^{(i)}$. The quality of each local hypothesis is assessed using a 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 $A_{i,i'} = A_{i',i}$. Figure 3.1 depicts an example of an FL network.

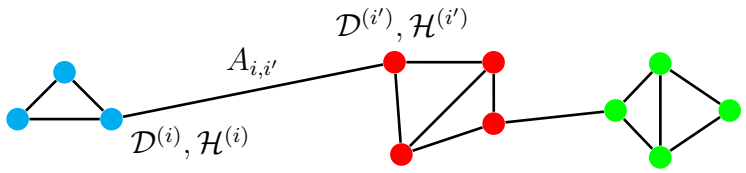


Fig. 3.1. Example of an FL network whose nodes $i \in \mathcal{V}$ represent different devices. Each device i generates a local dataset $\mathcal{D}^{(i)}$ and trains a local model $\mathcal{H}^{(i)}$. Some devices i, i' are connected by an undirected edge $\{i, i'\}$ with a positive edge weight $A_{i,i'}$.

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 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'.

Fig. 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. A local dataset is just one possible means to construct a loss function in order to evaluate model parameters. However, to build intuition, we can think of a local dataset $\mathcal{D}^{(i)}$

as a labelled dataset

$$\mathcal{D}^{(i)} := \left\{ \left(\mathbf{x}^{(i,1)}, y^{(i,1)}\right), \ldots, \left(\mathbf{x}^{(i,m_i)}, y^{(i,m_i)}\right) \right\}. \tag{28}$$

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,

$$\mathbf{X}^{(i)} := \left(\mathbf{x}^{(i,1)}, \ldots, \mathbf{x}^{(i,m_i)}\right)^T, \text{ and } \mathbf{y}^{(i)} := \left(y^{(i,1)}, \ldots, y^{(i,m_i)}\right)^T. \tag{29}$$

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 $L_i(\cdot)$ 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. To some extend, also the edges $\mathcal{E}$ in the FL network are a design choice.

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'. 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.

3.2 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 variation (or discrepancy)between the model parameters $\mathbf{w}^{(i)}, \mathbf{w}^{(i')}$.

We can measure the variation between two trained local models $h^{(i)}, h^{(i')}$ across an edge $\{i, i'\} \in \mathcal{E}$ in different ways. For example, we can compare their predictions on a common test set $\mathcal{D}$ by computing

$$d^{(i,i')} := (1/|\mathcal{D}|) \sum_{\mathbf{x} \in \mathcal{D}} \big[h^{(i)}(\mathbf{x}) - h^{(i')}(\mathbf{x})\big]^2. \tag{30}$$

In principle, we can use a different test set in (30) for each edge $\{i, i'\}$ of $\mathcal{G}$. For example, the test set could be obtained by merging randomly selected data points from each local dataset $\mathcal{D}^{(i)}$, $\mathcal{D}^{(i')}$.

Our main focus will be FL applications that use parametric local models,

i.e., each node learns local model parameters $\mathbf{w}^{(i)} \in \mathbb{R}^d$, for $i = 1, \ldots, n$. Here, we can measure the variation between $h^{(\mathbf{w}^{(i)})}$ and $h^{(\mathbf{w}^{(i')})}$ directly in terms of the model parameters $\mathbf{w}^{(i)}, \mathbf{w}^{(i')}$ at the nodes of an edge $\{i, i'\}$. In particular, we use a regularizer $\phi : \mathbb{R}^d \to \mathbb{R}$ that measures the difference between the model parameters,

$$d^{(i,i')} := \phi\big(\mathbf{w}^{(i)} - \mathbf{w}^{(i')}\big). \tag{31}$$

The penalty function ϕ will be mainly a design choice. Our main requirement is that ϕ is monotonically increasing[5] with respect to some norm in the Euclidean space $\mathbb{R}^d$ [16, 34]. 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 $\{i, i'\} \in \mathcal{E}$.

Summing up the edge-wise variations (weighted by the edge weights) yields the GTV of a collection of local model parameters,

$$\sum_{\{i,i'\} \in \mathcal{E}} A_{i,i'} \phi\big(\mathbf{w}^{(i)} - \mathbf{w}^{(i')}\big). \tag{32}$$

Our main focus will be on the special case of (32), obtained for $\phi(\cdot) := \|\cdot\|_2^2$,

$$\sum_{\{i,i'\} \in \mathcal{E}} A_{i,i'} \Big\|\mathbf{w}^{(i)} - \mathbf{w}^{(i')}\Big\|_2^2. \tag{33}$$

The choice of penalty $\phi(\cdot)$ has a crucial impact on the computational and statistical properties of the FL algorithms presented in Chapter 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

[5] A function $f : \mathbb{R} \to \mathbb{R}$ is monotonically increasing if $f(x) \leq f(y)$ whenever $x \leq y$. This means that larger argument values never result in smaller function values.

Chapter 7). On the other hand, choosing ϕ to be a norm results in FL algorithms that require more computation but less training data [34].

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

$$d^{(i)} := \sum_{i' \in \mathcal{N}^{(i)}} A_{i,i'}. \tag{34}$$

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 node degree

$$d^{(\mathcal{G})}_{\max} := \max_{i \in \mathcal{V}} d^{(i)} \overset{(34)}{=} \max_{i \in \mathcal{V}} \sum_{i' \in \mathcal{N}^{(i)}} A_{i,i'}. \tag{35}$$

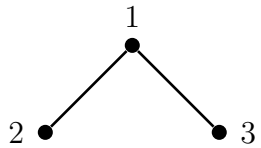
Besides inspecting the node degrees, we can study the connectivity of $\mathcal{G}$ also via the eigenvalues and eigenvectors of its Laplacian matrix $\mathbf{L}^{(\mathcal{G})} \in \mathbb{R}^{n \times n}$.[6] The Laplacian matrix of an undirected weighted graph $\mathcal{G}$ is defined element-wise as

$$L^{(\mathcal{G})}_{i,i'} := \begin{cases} -A_{i,i'} & \text{for } i \neq i', \{i, i'\} \in \mathcal{E} \\ \sum_{i'' \neq i} A_{i,i''} & \text{for } i = i' \\ 0 & \text{else.} \end{cases} \tag{36}$$

Figure 3.3 illustrates the Laplacian matrix of a small graph.

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

[6] The study of graphs via the eigenvalues and eigenvectors of associated matrices is the main subject of spectral graph theory [35, 36].



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

Fig. 3.3. Left: Example of an FL network $\mathcal{G}$ with three nodes $i = 1, 2, 3$ that are connected via two edges with unit weight $A_{1,2} = A_{1,3} = 1$. Right: Laplacian matrix $\mathbf{L}^{(\mathcal{G})} \in \mathbb{R}^{3\times 3}$ of $\mathcal{G}$.

identity

$$\mathbf{w}^T(\mathbf{L}^{(\mathcal{G})} \otimes \mathbf{I})\mathbf{w} = \sum_{\{i,i'\}\in\mathcal{E}} A_{i,i'} \left\| \mathbf{w}^{(i)} - \mathbf{w}^{(i')} \right\|_2^2$$
$$\text{for any } d \in \mathbb{N}, \mathbf{w} := \underbrace{\left(\left(\mathbf{w}^{(1)}\right)^T, \ldots, \left(\mathbf{w}^{(n)}\right)^T \right)^T}_{=:\mathrm{stack}\left\{\mathbf{w}^{(i)}\right\}_{i=1}^n} \in \mathbb{R}^{dn}. \quad (37)$$

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

$$\mathbf{L}^{(\mathcal{G})} = \sum_{i=1}^{n} \lambda_i \mathbf{u}^{(i)} \left(\mathbf{u}^{(i)}\right)^T, \quad (38)$$

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

$$0 = \lambda_1\big(\mathbf{L}^{(\mathcal{G})}\big) \leq \lambda_2\big(\mathbf{L}^{(\mathcal{G})}\big) \leq \ldots \leq \lambda_n\big(\mathbf{L}^{(\mathcal{G})}\big). \quad (39)$$

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 (39) is uniquely

determined for a given Laplacian matrix. In contrast, the eigenvectors $\mathbf{u}^{(i)}$ in (38) are not unique in general.[7]

The ordered eigenvalues $\lambda_i\big(\mathbf{L}^{(\mathcal{G})}\big)$ in (39) can be computed (or characterized) via the Courant–Fischer–Weyl min–max characterization (CFW) [3, Thm. 8.1.2.]. Two important special cases of this characterization are [35, 36]

$$\begin{aligned}\lambda_n\big(\mathbf{L}^{(\mathcal{G})}\big) &\overset{\text{CFW}}{=} \max_{\substack{\mathbf{v}\in\mathbb{R}^n \\ \|\mathbf{v}\|=1}} \mathbf{v}^T\mathbf{L}^{(\mathcal{G})}\mathbf{v} \\ &\overset{(37)}{=} \max_{\substack{\mathbf{v}\in\mathbb{R}^n \\ \|\mathbf{v}\|=1}} \sum_{\{i,i'\}\in\mathcal{E}} A_{i,i'}\big(v_i - v_{i'}\big)^2 \end{aligned} \tag{40}$$

and

$$\begin{aligned}\lambda_2\big(\mathbf{L}^{(\mathcal{G})}\big) &\overset{\text{CFW}}{=} \min_{\substack{\mathbf{v}\in\mathbb{R}^n \\ \mathbf{v}^T\mathbf{1}=0 \\ \|\mathbf{v}\|=1}} \mathbf{v}^T\mathbf{L}^{(\mathcal{G})}\mathbf{v} \\ &\overset{(37)}{=} \min_{\substack{\mathbf{v}\in\mathbb{R}^n \\ \mathbf{v}^T\mathbf{1}=0 \\ \|\mathbf{v}\|=1}} \sum_{\{i,i'\}\in\mathcal{E}} A_{i,i'}\big(v_i - v_{i'}\big)^2. \end{aligned} \tag{41}$$

By (37), 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 (37) is that any collection of identical local model parameters, stacked into the vector

$$\mathbf{w} = \text{stack}\{\mathbf{c}\} = \big(\mathbf{c}^T, \ldots, \mathbf{c}^T\big)^T, \text{ with some } \mathbf{c} \in \mathbb{R}^d \setminus \{\mathbf{0}\}, \tag{42}$$

is an eigenvector of $\mathbf{L}^{(\mathcal{G})} \otimes \mathbf{I}$ with corresponding eigenvalue $\lambda_1 = 0$ (see (39)). Thus, the Laplacian matrix of any FL network is singular (non-invertible).

[7] Consider the scenario where the list (39) contains repeated entries, i.e., some of the eigenvectors have identical eigenvalues.

The second eigenvalue λ_2 of $\mathbf{L}^{(\mathcal{G})}$ provides a great deal of information about the connectivity structure of $\mathcal{G}$.[8] Indeed, much of spectral graph theory is devoted to the analysis of λ_2, which is also referred to as algebraic connectivity, for different graph constructions [35, 36].

- Consider the case $\lambda_2 = 0$: Here, beside the eigenvector (42), we can find at least one additional eigenvector

$$\widetilde{\mathbf{w}} = \text{stack}\big\{\mathbf{w}^{(i)}\big\}_{i=1}^{n} \text{ with } \mathbf{w}^{(i)} \neq \mathbf{w}^{(i')} \text{ for some } i, i' \in \mathcal{V}, \tag{43}$$

  of $\mathbf{L}^{(\mathcal{G})} \otimes \mathbf{I}$ with eigenvalue equal to 0. In this case, the graph $\mathcal{G}$ is not connected, i.e., we can find two subsets (components) of nodes that do not have any edge between them (see Figure 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. 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 (41) and, in turn, λ_2.

In what follows, we will make use of the lower bound [36, Thm. 2.0.1]

$$\sum_{\{i,i'\} \in \mathcal{E}} A_{i,i'} \Big\| \mathbf{w}^{(i)} - \mathbf{w}^{(i')} \Big\|_2^2 \geq \lambda_2 \sum_{i=1}^{n} \big\| \mathbf{w}^{(i)} - \text{avg}\{\mathbf{w}^{(i)}\} \big\|_2^2. \tag{44}$$

[8] With slight abuse of language, we will sometimes speak about the eigenvalues of a FL network $\mathcal{G}$. However, we actually mean the eigenvalues of the Laplacian matrix (36) naturally associated with $\mathcal{G}$.

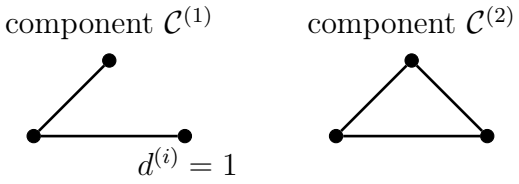


Fig. 3.4. An FL network $\mathcal{G}$ that consists of $n=6$ nodes forming two connected components $\mathcal{C}^{(1)}, \mathcal{C}^{(2)}$.

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 (44) follows from (37) and the CFW for the eigenvalues of the matrix $\mathbf{L}^{(\mathcal{G})} \otimes \mathbf{I}$.

The quantity $\sum_{i=1}^{n} \big\|\mathbf{w}^{(i)} - \text{avg}\{\mathbf{w}^{(i)}\}_{i=1}^{n}\big\|_2^2$ on the right-hand side of (44) 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

$$\mathcal{S} := \left\{ \mathbf{1} \otimes \mathbf{a} : \mathbf{a} \in \mathbb{R}^d \right\} = \left\{ \left(\mathbf{a}^T, \ldots, \mathbf{a}^T\right)^T, \text{ for some } \mathbf{a} \in \mathbb{R}^d \right\} \subseteq \mathbb{R}^{dn}. \quad (45)$$

The subspace $\mathcal{S}$ consists of stacked local model parameters $\mathbf{w}^{(i)}$ that are identical for all nodes $i = 1, \ldots, n$. Such a structure arises in certain FL settings where a single global model is shared among all devices. In this setting, the local model parameters satisfy $\mathbf{w}^{(i)} = \mathbf{a}$ for all $i = 1, \ldots, n$ and some common vector $\mathbf{a} \in \mathbb{R}^d$ (see Section 6.1). Equivalently, the condition

$$\left(\mathbf{w}^{(1)}, \ldots, \mathbf{w}^{(n)}\right)^T \in \mathcal{S}$$

characterizes this Single-model setting as membership in the subspace $\mathcal{S}$.

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

$$\mathbf{P}_{\mathcal{S}}\mathbf{w} = \big(\mathbf{a}^T, \ldots, \mathbf{a}^T\big)^T, \text{ with } \mathbf{a} = \text{avg}\{\mathbf{w}^{(i)}\}_{i=1}^{n}. \tag{46}$$

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

$$\mathbf{P}_{\mathcal{S}^{\perp}}\mathbf{w} = \mathbf{w} - \mathbf{P}_{\mathcal{S}}\mathbf{w} = \text{stack}\big\{\mathbf{w}^{(i)} - \text{avg}\{\mathbf{w}^{(i)}\}_{i=1}^{n}\big\}_{i=1}^{n}. \tag{47}$$

3.3 Generalized Total Variation Minimization

Consider an FL network $\mathcal{G}$ whose nodes $i \in \mathcal{V}$ represent individual devices, each learning personalized model parameters $\mathbf{w}^{(i)}$. The quality of a specific choice of model parameters is assessed via a local loss function $L_i\left(\mathbf{w}^{(i)}\right)$, typically derived from a training error applied to a local dataset.

Our focus is on FL applications where these local loss functions alone do not suffice to reliably train a high-dimensional local model. For instance, the local dataset $\mathcal{D}^{(i)}$ used to compute the local loss function at some node i may be too small relative to the effective dimension of the underlying local model $\mathcal{H}^{(i)}$, making the training process prone to overfitting (see Section 2.5).[9]

In such settings, collaboration between the devices across the edges of the FL network becomes essential. To address this, we seek local model parameters that not only minimize the local loss functions but also exhibit small GTV (32). Requiring a small GTV couples the training among neighboring devices and results in an implicit (and privacy-friendly) pooling of local datasets (see Section 6.2).

[9] As a rule of thumb, the number of data points used to train a model should be proportional to the (effective) number of its parameters.

GTV minimization (GTVMin) optimally balances the (average) local loss and the GTV (32) of local model parameters $\mathbf{w}^{(i)}$,

$$\big\{\widehat{\mathbf{w}}^{(i)}\big\}_{i=1}^{n} \in \operatorname*{arg\,min}_{\mathbf{w}^{(1)},\ldots,\mathbf{w}^{(n)}} \sum_{i\in\mathcal{V}} L_i\left(\mathbf{w}^{(i)}\right) + \alpha \sum_{\{i,i'\}\in\mathcal{E}} A_{i,i'}\phi\big(\mathbf{w}^{(i)}-\mathbf{w}^{(i')}\big). \tag{48}$$

Note that (48) is parametrized by the choice for the penalty function $\phi(\cdot)$. We discuss the effect of different choices for $\phi(\cdot)$ in Section 3.3.1 and 3.3.2. Our main focus will be on the special case of (48), obtained with $\phi(\cdot) := \|\cdot\|_2^2$,

$$\big\{\widehat{\mathbf{w}}^{(i)}\big\}_{i=1}^{n} \in \operatorname*{arg\,min}_{\mathbf{w}^{(1)},\ldots,\mathbf{w}^{(n)}} \sum_{i\in\mathcal{V}} L_i\left(\mathbf{w}^{(i)}\right) + \alpha \sum_{\{i,i'\}\in\mathcal{E}} A_{i,i'}\Big\|\mathbf{w}^{(i)}-\mathbf{w}^{(i')}\Big\|_2^2. \tag{49}$$

The GTVMin parameter $\alpha>0$ in (48) steers the preference for learning local model parameters $\mathbf{w}^{(i)}$ with small GTV versus incurring small local loss $\sum_{i\in\mathcal{V}} L_i\left(\mathbf{w}^{(i)}\right)$. For $\alpha=0$, GTVMin decomposes into fully independent local ERM instances $\min_{\mathbf{w}^{(i)}} L_i\left(\cdot\right)$, for $i=1,\ldots,n$. On the other hand, increasing the value of α makes the solutions of (48) increasingly clustered: the local model parameters $\widehat{\mathbf{w}}^{(i)}$ become approximately constant over increasingly large subsets of nodes. This behavior is appealing for clustered federated learning (CFL) which we discuss in Section 6.2.

Choosing α beyond a critical value - that depends on the shape of the local loss functions and the edges $\mathcal{E}$ - results in $\widehat{\mathbf{w}}^{(i)}$ being (nearly) constant over all nodes $i\in\mathcal{V}$. In practice, the choice of α can be guided by validation [37] or by a probabilistic analysis of the solutions of (48). Section 3.3.2 presents an example of such an analysis.

Note that GTVMin (48) 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. Loosely speaking, GTVMin couples the training of local models

by requiring them to be similar across the edges of the FL network. For the extreme case of an FL network without any edges, GTVMin decomposes into independent ERM instances

$$\arg\min_{\mathbf{w}^{(i)}} L_i\left(\mathbf{w}^{(i)}\right), \text{ for each } i=1,\ldots,n.$$

The connectivity (i.e., the edges $\mathcal{E}$) of 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 discusses graph learning methods that determine the edge weights $A_{i,i'}$ in a data-driven fashion, i.e., directly from the local datasets $\mathcal{D}^{(i)}, \mathcal{D}^{(i')}$.

GTVMin for linear models. 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 $h^{(i)}(\mathbf{x}) := \left(\mathbf{w}^{(i)}\right)^T \mathbf{x}$. We measure the quality of the parameters via the average squared error loss

$$\begin{aligned} L_i\left(\mathbf{w}^{(i)}\right) &:= (1/m_i)\sum_{r=1}^{m_i}\left(y^{(i,r)} - \left(\mathbf{w}^{(i)}\right)^T \mathbf{x}^{(i,r)}\right)^2 \\ &\overset{(29)}{=} (1/m_i)\left\|\mathbf{y}^{(i)} - \mathbf{X}^{(i)}\mathbf{w}^{(i)}\right\|_2^2. \end{aligned} \tag{50}$$

Inserting (50) into (49), yields the following instance of GTVMin to train local linear models,

$$\left\{\widehat{\mathbf{w}}^{(i)}\right\} \in \arg\min_{\{\mathbf{w}^{(i)}\}_{i=1}^n} \sum_{i\in\mathcal{V}} (1/m_i)\left\|\mathbf{y}^{(i)} - \mathbf{X}^{(i)}\mathbf{w}^{(i)}\right\|_2^2 + \alpha \sum_{\{i,i'\}\in\mathcal{E}} A_{i,i'}\left\|\mathbf{w}^{(i)} - \mathbf{w}^{(i')}\right\|_2^2. \tag{51}$$

The identity (37) allows to rewrite (51) using the Laplacian matrix $\mathbf{L}^{(\mathcal{G})}$ as

$$\widehat{\mathbf{w}}^{(i)} \in \operatorname*{arg\,min}_{\mathbf{w}=\mathrm{stack}\left\{\mathbf{w}^{(i)}\right\}} \sum_{i\in\mathcal{V}} (1/m_i)\left\|\mathbf{y}^{(i)}-\mathbf{X}^{(i)}\mathbf{w}^{(i)}\right\|_2^2 + \alpha \mathbf{w}^T\big(\mathbf{L}^{(\mathcal{G})}\otimes \mathbf{I}_d\big)\mathbf{w}. \quad (52)$$

Let us rewrite the objective function in (52) as

$$\mathbf{w}^T\left(\begin{pmatrix} \mathbf{Q}^{(1)} & \cdots & 0 \\ \vdots & \ddots & \vdots \\ 0 & \cdots & \mathbf{Q}^{(n)} \end{pmatrix} + \alpha \mathbf{L}^{(\mathcal{G})}\otimes \mathbf{I}\right)\mathbf{w} + \big(\big(\mathbf{q}^{(1)}\big)^T,\ldots,\big(\mathbf{q}^{(n)}\big)^T\big)\mathbf{w} \quad (53)$$

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

Thus, like linear regression (6) and ridge regression (24), GTVMin (52) (for local linear models $\mathcal{H}^{(i)}$) minimizes a convex quadratic function,

$$\big\{\widehat{\mathbf{w}}^{(i)}\big\}_{i=1}^n \in \operatorname*{arg\,min}_{\mathbf{w}=\mathrm{stack}\left\{\mathbf{w}^{(i)}\right\}_{i=1}^n} \mathbf{w}^T\mathbf{Q}\mathbf{w} + \mathbf{q}^T\mathbf{w}. \quad (54)$$

Here, we used the psd matrix

$$\mathbf{Q} := \begin{pmatrix} \mathbf{Q}^{(1)} & \cdots & \mathbf{0} \\ \vdots & \ddots & \vdots \\ \mathbf{0} & \cdots & \mathbf{Q}^{(n)} \end{pmatrix} + \alpha \mathbf{L}^{(\mathcal{G})}\otimes \mathbf{I} \text{ with } \mathbf{Q}^{(i)} := (1/m_i)\big(\mathbf{X}^{(i)}\big)^T\mathbf{X}^{(i)} \quad (55)$$

and the vector

$$\mathbf{q} := \big(\big(\mathbf{q}^{(1)}\big)^T,\ldots,\big(\mathbf{q}^{(n)}\big)^T\big)^T, \text{ with } \mathbf{q}^{(i)} := (-2/m_i)\big(\mathbf{X}^{(i)}\big)^T\mathbf{y}^{(i)}. \quad (56)$$

3.3.1 Computational Aspects of GTVMin

Chapter 5 will apply optimization methods to solve GTVMin (48), resulting in practical FL algorithms. Different instances of GTVMin favour different

classes of optimization methods. For example, using a differentiable loss function $L_i(\cdot)$ and penalty function $\phi(\cdot)$ 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 [38, 39]

$$\mathbf{prox}_{L_i(\cdot),\rho}(\mathbf{w}) := \operatorname*{arg\,min}_{\mathbf{w}' \in \mathbb{R}^d} L_i(\mathbf{w}') + (\rho/2)\|\mathbf{w} - \mathbf{w}'\|_2^2 \text{ for some } \rho > 0. \quad (57)$$

We refer to functions $L_i(\cdot)$ for which (57) can be computed easily as *simple* or proximable [40]. GTVMin (49) with proximable loss functions can be solved via proximal algorithms [39]. 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 an 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.

It is instructive to study the computational aspects of the special case of GTVMin (51) for local linear models. As discussed above, this instance is equivalent to solving (54). Any solution $\widehat{\mathbf{w}}$ of (54) (and, in turn, (51)) is characterized by the zero-gradient condition

$$\mathbf{Q}\widehat{\mathbf{w}} = -(1/2)\mathbf{q}, \tag{58}$$

with $\mathbf{Q}, \mathbf{q}$ as defined in (55) and (56). If the matrix $\mathbf{Q}$ in (58) is invertible, the solution to (58) and, in turn, to the GTVMin instance (51) is unique and given by $\widehat{\mathbf{w}} = (-1/2)\mathbf{Q}^{-1}\mathbf{q}$.

The size of the matrix $\mathbf{Q}$ (see (55)) 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 $\mathbf{Q}^{-1}$.[10] Instead, we typically need to resort to iterative methods [41, 42].

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

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

variants of a gradient step,

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

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

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

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

3.3.2 Statistical Aspects of GTVMin

How useful are the solutions of GTVMin (49) 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

$$\begin{aligned} \mathbf{y}^{(i)} =& \underbrace{\big(\mathbf{x}^{(i,1)}, \ldots, \mathbf{x}^{(i,m_i)}\big)^T}_{\text{local feature matrix } \mathbf{X}^{(i)}} \overline{\mathbf{w}}^{(i)} + \boldsymbol{\varepsilon}^{(i)} \\ & \text{with } \mathbf{x}^{(i,r)} \overset{\text{i.i.d.}}{\sim} \mathcal{N}(\mathbf{0}, \mathbf{I}), \text{ for } r = 1, \ldots, m_i, i = 1, \ldots, n, \\ & \text{and } \boldsymbol{\varepsilon}^{(i)} \sim \mathcal{N}(\mathbf{0}, \sigma^2\mathbf{I}), \text{ for } i = 1, \ldots, n. \end{aligned} \tag{59}$$

In contrast to the probabilistic model (11) (which we used for the analysis of ERM), the probabilistic model (59) 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 (51) 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 (59) with true model parameters $\overline{\mathbf{w}}^{(i)}$. For ease of exposition, we assume that

$$\overline{\mathbf{w}}^{(i)} = \overline{\mathbf{c}}, \text{ for some } \overline{\mathbf{c}} \in \mathbb{R}^d \text{ and all } i \in \mathcal{V}. \tag{60}$$

To study the deviation between the solutions $\widehat{\mathbf{w}}^{(i)}$ of (51) and the true underlying parameters $\overline{\mathbf{w}}^{(i)}$, we decompose it as

$$\widehat{\mathbf{w}}^{(i)} = \widetilde{\mathbf{w}}^{(i)} + \widehat{\mathbf{c}}, \text{ with } \widehat{\mathbf{c}} := (1/n) \sum_{i'=1}^{n} \widehat{\mathbf{w}}^{(i')}. \tag{61}$$

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 (45). 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 (61) entails an analogous (orthogonal) decomposition of the error $\widehat{\mathbf{w}}^{(i)} - \overline{\mathbf{w}}^{(i)}$. Indeed, for identical true underlying model parameters (60) (which makes $\overline{\mathbf{w}}$ an element of the subspace (45)), we have

$$\sum_{i=1}^{n} \big\|\widehat{\mathbf{w}}^{(i)} - \overline{\mathbf{w}}^{(i)}\big\|_2^2 \overset{(60),(61)}{=} \underbrace{\sum_{i=1}^{n} \|\mathbf{c} - \widehat{\mathbf{c}}\|_2^2}_{n\|\mathbf{c}-\widehat{\mathbf{c}}\|_2^2} + \sum_{i=1}^{n} \big\|\widetilde{\mathbf{w}}^{(i)}\big\|_2^2. \tag{62}$$

The following proposition provides an upper bound on the second error component in (62).

Proposition 3.1. *Consider a connected FL network, i.e., $\lambda_2 > 0$ (see (39)), and the solution (61) to GTVMin (51) for the local datasets (59). If the true local model parameters in (59) are identical (see (60)), 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 learned model parameters $\widehat{\mathbf{w}}^{(i)}$ from their average, as*

$$\sum_{i=1}^{n} \big\|\widetilde{\mathbf{w}}^{(i)}\big\|_2^2 \leq \frac{1}{\lambda_2 \alpha} \sum_{i=1}^{n} (1/m_i) \big\|\boldsymbol{\varepsilon}^{(i)}\big\|_2^2. \tag{63}$$

Proof. See Section 3.7.1. □

Note that Proposition 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 Proposition 3.1 separately to each connected component of $\mathcal{G}$.

The upper bound (63) involves three components:

- the properties of local datasets, via the noise terms $\boldsymbol{\varepsilon}^{(i)}$ in (59),
- the FL network via the eigenvalue $\lambda_2\big(\mathbf{L}^{(\mathcal{G})}\big)$ (see (39)),
- the GTVMin parameter α.

According to (63), we can ensure a small error component $\widetilde{\mathbf{w}}^{(i)}$ of the GTVMin solution by choosing a large value α. Thus, by (62), 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 of a FL network is desirable for FL applications that require to learn a common (global) model parameters for all nodes [12]. However, some FL applications involve heterogeneous devices that generate local datasets with significantly different statistics [34]. For such applications it is detrimental to enforce common model parameters at all nodes (see Chapter 6). Instead, we should enforce common model parameters only for nodes with local datasets having similar statistical properties. This is exactly the objective of clustered FL which we discuss in Section 6.2.

3.4 Non-Parametric Models in FL Networks

In its basic form (49), 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 way to measure the discrepancy between two hypothesis maps $h^{(i)}, h^{(i')}$ is to compare their predictions on a dataset

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

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 3.8).

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

$$d^{(i,i')} := (1/m') \sum_{\mathbf{x} \in \mathcal{D}^{\{i,i'\}}} (1/2) \big[L\left(\left(\mathbf{x}, h^{(i)}(\mathbf{x}) \right), h^{(i')} \right) + L\left(\left(\mathbf{x}, h^{(i')}(\mathbf{x}) \right), h^{(i)} \right) \big]. \tag{64}$$

Different choices for the loss function in (64) result in different computational and statistical properties of the resulting FL algorithms. For real-valued predictions we can use the squared error loss in (64), yielding

$$d^{(i,i')} := (1/m') \sum_{\mathbf{x} \in \mathcal{D}^{\{i,i'\}}} \big[h^{(i)}(\mathbf{x}) - h^{(i')}(\mathbf{x}) \big]^2. \tag{65}$$

We can generalize GTVMin by replacing $\left\| \mathbf{w}^{(i)} - \mathbf{w}^{(i')} \right\|_2^2$ in (49) with the discrepancy $d^{(h^{(i)}, h^{(i')})}$ (64) (or the special case (65)). This results in

$$\big\{ \widehat{h}^{(i)} \big\}_{i=1}^{n} \in \operatorname*{arg\,min}_{\substack{h^{(i)} \in \mathcal{H}^{(i)} \\ i \in \mathcal{V}}} \sum_{i \in \mathcal{V}} L_i \left(h^{(i)} \right) + \alpha \sum_{\{i,i'\} \in \mathcal{E}} A_{i,i'} d^{(h^{(i)}, h^{(i')})}. \tag{66}$$

3.5 Interpretations

We next discuss some interpretations of GTVMin (48).

Empirical Risk Minimization. GTVMin (49) 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 (48) is convex clustering [43, 44]. Indeed, convex clustering is obtained from (48) using the local loss function

$$L_i\left(\mathbf{w}^{(i)}\right) = \|\mathbf{w}^{(i)} - \mathbf{a}^{(i)}\|^2, \text{ for all nodes } i \in \mathcal{V} \tag{67}$$

and the GTV penalty function $\phi(\mathbf{u}) = \|\mathbf{u}\|_p$ with some $p \geq 1$.[11] The vectors $\mathbf{a}^{(i)}$, for $i = 1, \ldots, n$, are the features of data points that we wish to cluster in (67). 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 (48) 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 [45]. This dual problem optimizes edge-wise variables $\mathbf{u}^{(\{i,i'\})}$, one for each edge $\{i, i'\} \in \mathcal{E}$ of $\mathcal{G}$,

$$\max_{\substack{\mathbf{u}^{(e)}, e\in\mathcal{E} \\ \mathbf{w}^{(i)}, i\in\mathcal{V}}} -\sum_{i\in\mathcal{V}} L_i^*\left(\mathbf{w}^{(i)}\right) - \alpha \sum_{e\in\mathcal{E}} A_e \phi^*\left(\mathbf{u}^{(e)}/(\alpha A_e)\right) \tag{68}$$

$$\text{subject to } -\mathbf{w}^{(i)} = \sum_{\substack{e\in\mathcal{E} \\ e_+=i}} \mathbf{u}^{(e)} - \sum_{\substack{e\in\mathcal{E} \\ e_-=i}} \mathbf{u}^{(e)} \text{ for each } i \in \mathcal{V}. \tag{69}$$

Here, we have introduced an orientation for each edge $e := \{i, i'\}$, by defining the *head* $e_- := \min\{i, i'\}$ and the *tail* $e_+ := \max\{i, i'\}$.[12] Moreover, we used

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

[12] 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 meaning in terms of GTVMin-based FL algorithms. After all, GTVMin (48) and its dual (68) are defined for an FL network with undirected edges $\mathcal{E}$.

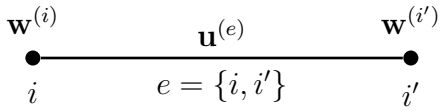


Fig. 3.5. Two nodes of an FL network that are connected by an edge $e = \{i, i'\}$. GTVMin (48) optimizes local model parameters $\mathbf{w}^{(i)}$ for each node $i \in \mathcal{V}$ in the FL network. The dual (68) of GTVMin optimizes local model parameter $\mathbf{u}^{(e)}$ for each edge $e \in \mathcal{E}$ in the FL network.

the convex conjugates $L_i^*(\cdot), \phi^*$ of the local loss function $L_i(\cdot)$ and GTV penalty function ϕ.[13]

The dual optimization problem (68) generalizes the optimal flow problem [45, Sec. 1J] to vector-valued flows. The special case of (68), obtained when the GTV penalty function ϕ is a norm, is equivalent to a generalized minimum-cost flow problem [47, Sec. 1.2.1]. Indeed, the maximization problem (68) is equivalent to the minimization

$$\begin{aligned} &\min_{\substack{\mathbf{u}^{(e)}, e\in\mathcal{E} \\ \mathbf{w}^{(i)}, i\in\mathcal{V}}} \sum_{i\in\mathcal{V}} L_i^*\left(\mathbf{w}^{(i)}\right) \\ &\text{subject to } -\mathbf{w}^{(i)} = \sum_{\substack{e\in\mathcal{E} \\ e_+=i}} \mathbf{u}^{(e)} - \sum_{\substack{e\in\mathcal{E} \\ e_-=i}} \mathbf{u}^{(e)} \text{ for each node } i \in \mathcal{V} \\ &\quad \|\mathbf{u}^{(e)}\|_* \leq \alpha A_e \text{ for each edge } e \in \mathcal{E}. \end{aligned} \tag{71}$$

The optimization problem (71) reduces to the minimum-cost flow problem [47,

[13]The convex conjugate of a function $f : \mathbb{R}^d \to \mathbb{R}$ is defined as [46]

$$f^*(\mathbf{x}) := \sup_{\mathbf{z}\in\mathbb{R}^d} \mathbf{x}^T\mathbf{z} - f(\mathbf{z}). \tag{70}$$

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 $i' \in \mathcal{C}$ [34]. The cluster-wise model parameters $\overline{\mathbf{w}}^{(\mathcal{C})}$ are the solutions of

$$\min_{\mathbf{w}} \sum_{i' \in \mathcal{C}} L_{i'}\left(\mathbf{w}\right), \tag{72}$$

which, in turn, is an instance of a locally weighted learning problem [48, Sec. 3.1.2]

$$\overline{\mathbf{w}}^{(\mathcal{C})} = \operatorname*{arg\,min}_{\mathbf{w} \in \mathbb{R}^d} \sum_{i' \in \mathcal{V}} \rho_{i'} L_{i'}\left(\mathbf{w}\right). \tag{73}$$

Indeed, we obtain (72) from (73) by setting the weights $\rho_{i'}$ equal to 1 if $i' \in \mathcal{C}$ and 0 otherwise.

3.6 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 $\lambda_n\big(\mathbf{L}^{(\mathcal{G})}\big) \leq 2d^{(\mathcal{G})}_{\max}$. Try to find a graph $\mathcal{G}$ such that $\lambda_n\big(\mathbf{L}^{(\mathcal{G})}\big) \approx 2d^{(\mathcal{G})}_{\max}$.

3.2. Null Space of the Laplacian matrix. Consider an undirected weighted graph $\mathcal{G}$ with n nodes, indexed by $i = 1, \ldots, n$. A component of $\mathcal{G}$ is a subset $\mathcal{C} \subseteq \mathcal{V}$ of nodes such that every pair of nodes in $\mathcal{C}$ is connected by a path within $\mathcal{C}$, and there are no edges connecting nodes in $\mathcal{C}$ to nodes in $\mathcal{V} \setminus \mathcal{C}$. If $\mathcal{G}$ is connected, it has a single component, namely the entire node set $\mathcal{V}$. Let $\mathbf{L}^{(\mathcal{G})}$ denote the Laplacian matrix of $\mathcal{G}$. The null space of $\mathbf{L}^{(\mathcal{G})}$ is the subspace $\mathcal{K} \subseteq \mathbb{R}^n$ defined as

$$\mathcal{K} := \left\{\mathbf{v} \in \mathbb{R}^n \,\middle|\, \mathbf{L}^{(\mathcal{G})}\mathbf{v} = \mathbf{0}\right\}.$$

Show that the dimension of $\mathcal{K}$ is equal to the number of components in $\mathcal{G}$.

3.3. Null-Space Visualization. Consider an FL network with $n = 2$ nodes. Each node $i \in \{1, 2\}$ carries a scalar local model parameter $w^{(i)} \in \mathbb{R}$. We can conveniently represent the local model parameters by stacking them into the vector

$$\mathbf{w} = \big(w^{(1)}, w^{(2)}\big)^T \in \mathbb{R}^2.$$

This exercise requires you to visualize the subspace $\mathcal{S}$ (45) of $\mathbb{R}^2$.

1. Draw the subspace $\mathcal{S}$ as a line in the plane $\mathbb{R}^2$. Label your axes as $w^{(1)}$ and $w^{(2)}$. Indicate at least two vectors that belong to $\mathcal{S}$.

2. Determine and draw the orthogonal complement $\mathcal{S}^{\perp}$ of $\mathcal{S}$. Label it clearly in your drawing.

3. Give a specific example of a non-zero vector $\mathbf{b} \in \mathcal{S}^{\perp}$, and verify that it is orthogonal to an arbitrary vector $\mathbf{a} \in \mathcal{S}$ by computing the inner product $\mathbf{b}^{\top}\mathbf{a}$.

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

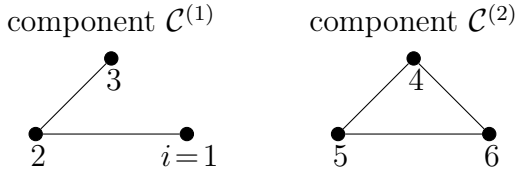


Fig. 3.6. An undirected graph $\mathcal{G}$ that consists of two connected components $\mathcal{C}^{(1)}, \mathcal{C}^{(2)}$.

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

3.5. 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 $\mathbf{L}^{(\mathcal{G})}$.

3.6. Capacity of an FL network. Consider the FL network shown in Figure 3.7. Each node holds a local dataset, with its size indicated by the

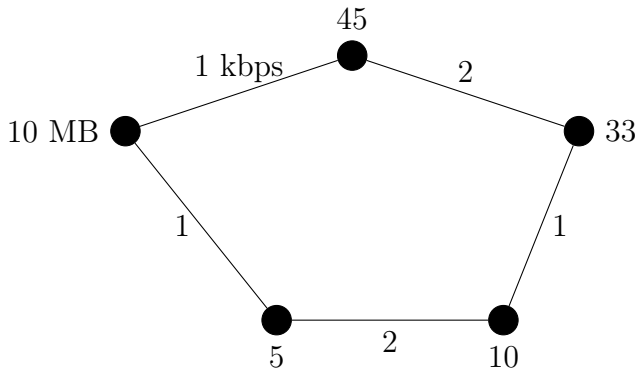


Fig. 3.7. An FL network whose nodes $i = 1, \ldots, 5$ represent devices that hold local datasets whose size is indicated next to each node.

adjacent numbers. The devices 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?

3.7. 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 (65) coincides with $\left\|\mathbf{w}^{(i)} - \mathbf{w}^{(i')}\right\|_2^2$?

3.8. Privacy-Friendly Discrepancy Measures. The discrepancy measure (64) 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 can be harmful as they potentially leak sensitive information. Could you think of a simple message passing protocol between node i and i' that allows them to evaluate (64) only by sharing the predictions $h^{(i)}(\mathbf{x}), h^{(i')}(\mathbf{x})$ for $\mathbf{x} \in \mathcal{D}^{\{i,i'\}}$?

3.9. Structure of GTVMin. What are sufficient conditions for the local datasets and the edge weights used in GTVMin such that the matrix $\mathbf{Q}$ in (55) is invertible?

3.10. Existence and Uniqueness of GTVMin Solution. Consider the GTVMin instance (49), 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 (49) has at least one solution?

2. **Uniqueness.** Then, try to find a condition that ensures that (49) has a unique solution.

3. Finally, try to find necessary conditions for the existence and uniqueness of solutions to (49).

3.11. 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 $i = 1, \ldots, n$.

3.12. Computing the Average over a Star. Consider the FL network depicted in Figure 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 $\widehat{w}^{(i_0)} \approx (1/(n-1)) \sum_{i \in \mathcal{P}} y^{(i)}$.

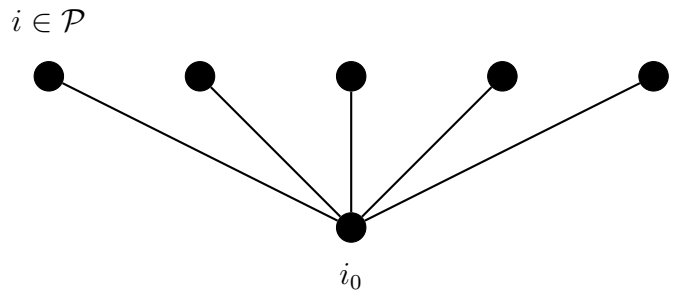


Fig. 3.8. An FL network that consists of a centre node i_0 that is connected to several peripheral nodes $\mathcal{P} := \mathcal{V} \setminus \{i_0\}$.

3.13. Fundamental Limits. Consider the FL network depicted in Figure 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.

3.14. Counting Number of Paths. Consider an undirected graph $\mathcal{G}$ with each edge $\{i, i'\} \in \mathcal{E}$ having unit edge weight $A_{i,i'} = 1$. A k-hop path, for some $k \in \{1, 2, \ldots.\}$, 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 $\big(\mathbf{A}^k\big)_{i,i'}$.

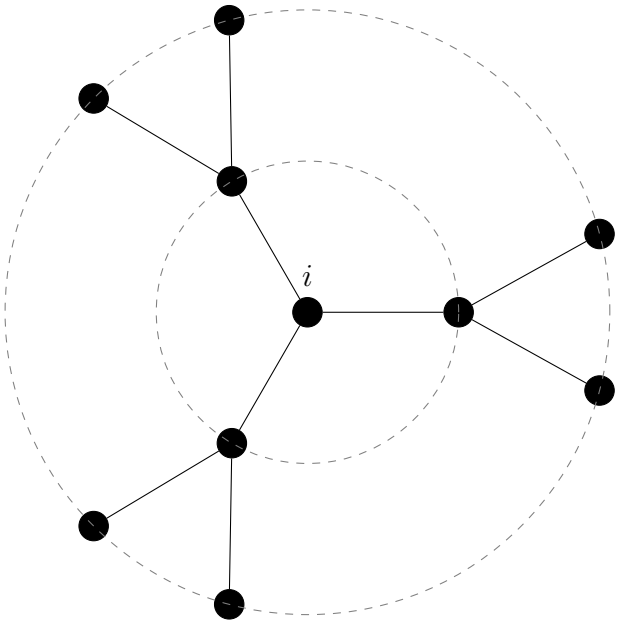


Fig. 3.9. An FL network containing a node i with node degree $d^{(i)} = 3$, like all its neighbors $i' \in \mathcal{N}^{(i)}$. We use an FL algorithm to learn local model parameters $w^{(i)}$. If the algorithm employs message passing, the first iteration provides access only to the local datasets of 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.

3.15. Proximal operator of a quadratic function. Study the proximal operator (57) for a quadratic function,

$$L_i\left(\mathbf{w}^{(i)}\right) = \left(\mathbf{w}^{(i)}\right)^T \mathbf{Q}\mathbf{w}^{(i)} + \mathbf{q}^T\mathbf{w}^{(i)} + q,$$

with some matrix $\mathbf{Q} \in \mathbb{R}^{d\times d}$, vector $\mathbf{q} \in \mathbb{R}^d$ and number $q \in \mathbb{R}$.

3.7 Proofs

3.7.1 Proof of Proposition 3.1

Let us introduce the shorthand $f\big(\mathbf{w}^{(i)}\big)$ for the objective function of the GTVMin instance (51). We verify the bound (63) by showing that if it does not hold, the choice of the local model parameters $\mathbf{w}^{(i)} := \overline{\mathbf{w}}^{(i)}$ (see (59)) 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 (51).

First, note that

$$\begin{aligned} f\big(\overline{\mathbf{w}}^{(i)}\big) &= \sum_{i\in\mathcal{V}} (1/m_i)\big\|\mathbf{y}^{(i)} - \mathbf{X}^{(i)}\overline{\mathbf{w}}^{(i)}\big\|_2^2 + \alpha \sum_{\{i,i'\}\in\mathcal{E}} A_{i,i'}\Big\|\overline{\mathbf{w}}^{(i)} - \overline{\mathbf{w}}^{(i')}\Big\|_2^2 \\ &\overset{(60)}{=} \sum_{i\in\mathcal{V}} (1/m_i)\big\|\mathbf{y}^{(i)} - \mathbf{X}^{(i)}\overline{\mathbf{w}}^{(i)}\big\|_2^2 \\ &\overset{(59)}{=} \sum_{i\in\mathcal{V}} (1/m_i)\big\|\mathbf{X}^{(i)}\overline{\mathbf{w}}^{(i)} + \boldsymbol{\varepsilon}^{(i)} - \mathbf{X}^{(i)}\overline{\mathbf{w}}^{(i)}\big\|_2^2 \\ &= \sum_{i\in\mathcal{V}} (1/m_i)\big\|\boldsymbol{\varepsilon}^{(i)}\big\|_2^2. \end{aligned} \tag{74}$$

Inserting (61) into (51),

$$\begin{aligned} f\big(\widehat{\mathbf{w}}^{(i)}\big) &= \underbrace{\sum_{i\in\mathcal{V}} (1/m_i)\big\|\mathbf{y}^{(i)} - \mathbf{X}^{(i)}\widehat{\mathbf{w}}^{(i)}\big\|_2^2}_{\geq 0} + \alpha \sum_{\{i,i'\}\in\mathcal{E}} A_{i,i'} \underbrace{\Big\|\widehat{\mathbf{w}}^{(i)} - \widehat{\mathbf{w}}^{(i')}\Big\|_2^2}_{\overset{(61)}{=}\big\|\widetilde{\mathbf{w}}^{(i)} - \widetilde{\mathbf{w}}^{(i')}\big\|_2^2} \\ &\geq \alpha \sum_{\{i,i'\}\in\mathcal{E}} A_{i,i'}\Big\|\widetilde{\mathbf{w}}^{(i)} - \widetilde{\mathbf{w}}^{(i')}\Big\|_2^2 \\ &\overset{(44)}{\geq} \alpha\lambda_2 \sum_{i=1}^{n} \big\|\widetilde{\mathbf{w}}^{(i)}\big\|_2^2. \end{aligned} \tag{75}$$

If the bound (63) would not hold, then by (75) and (74) 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 (51).

.

4 Gradient Methods for Federated Optimization

Chapter 3 introduced GTVMin as a central design principle for FL algorithms. Many important instances of GTVMin require the minimization of a smooth objective function $f(\mathbf{w})$ over a continuous parameter space. This chapter investigates how gradient-based methods –a broadly used family of iterative optimization methods – can be employed to solve such problems. These methods rely on local approximations of $f(\mathbf{w})$ using its gradient.

Section 4.1 introduces the basic gradient step and explains how it updates model parameters in the direction of steepest descent. Key considerations such as the choice of the learning rate are discussed in Section 4.2, along with stopping criteria in Section 4.3 that help determine when to terminate the optimization process. Section 4.4 studies how perturbations affect the convergence of gradient steps, which is particularly relevant in FL applications that involve unreliable communication or partial data access.

When optimization problems include explicit constraints on the model parameters, projected gradient descent (projected GD) presented in Section 4.5 provides a principled solution. Section 4.6 then extends gradient-based methods to non-parametric models, using proximal operators and test datasets to generalize the notion of a gradient step. Finally, Section 4.7 interprets gradient-based methods as a special case of fixed-point iterations. This perspective allows for a unified understanding of FL algorithms as convergent

processes driven by contraction operators.

4.1 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 $\mathbf{w}$. One example of such an optimization problem is the ERM instance (2). Unless stated otherwise, we consider an objective function of the form:

$$f(\mathbf{w}) := \mathbf{w}^T \mathbf{Q} \mathbf{w} + \mathbf{q}^T \mathbf{w}. \tag{76}$$

Although restricting our discussion to objective functions of the form (76) may seem limiting, this formulation allows for a straightforward analysis and generalization to larger classes of differentiable functions. Moreover, we can use (76) also as an approximation for broader families of objective functions.

Note that (76) 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 (76): Linear regression (2) and ridge regression (24) in Chapter 2 as well as GTVMin (51) for local linear models in Chapter 3. Moreover, (76) is a useful approximation for the objective functions arising in other ML methods [49–51].

Given a current choice of model parameters $\mathbf{w}^{(t)}$, we want to update them towards a minimum of (76). To this end, we use the gradient $\nabla f\big(\mathbf{w}^{(t)}\big)$ to locally approximate $f(\mathbf{w})$ (see Figure 4.1). The gradient $\nabla f\big(\mathbf{w}^{(t)}\big)$ indicates the direction in which the function $f(\mathbf{w})$ maximally increases. Therefore, it

seems reasonable to update $\mathbf{w}^{(t)}$ in the opposite direction of $\nabla f\big(\mathbf{w}^{(t)}\big)$,

$$\begin{aligned}\mathbf{w}^{(t+1)} &:= \mathbf{w}^{(t)} - \eta \nabla f\big(\mathbf{w}^{(t)}\big) \\ &\overset{(76)}{=} \mathbf{w}^{(t)} - \eta\big(2\mathbf{Q}\mathbf{w}^{(t)} + \mathbf{q}\big). \end{aligned} \tag{77}$$

The gradient step (77) involves the positive factor $\eta > 0$ which we refer to as step size or learning rate. Algorithm 2 summarizes the most basic variant of gradient-based methods, which simply iterates (77) until a predefined stopping criterion is met.

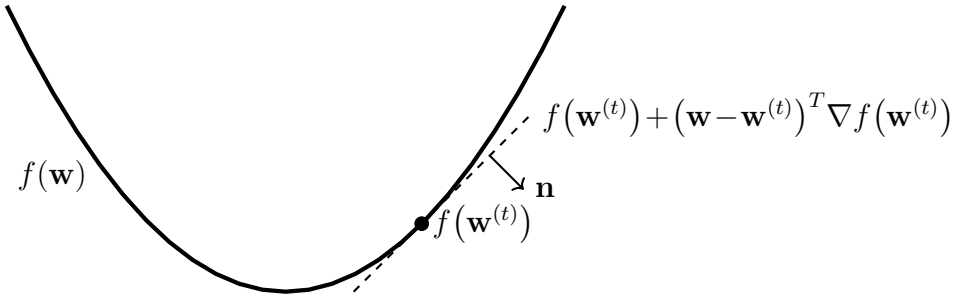


Fig. 4.1. We can approximate a differentiable function $f(\mathbf{w})$ locally around a point $\mathbf{w}^{(t)} \in \mathbb{R}^d$ using the linear function $f\big(\mathbf{w}^{(t)}\big) + \big(\mathbf{w} - \mathbf{w}^{(t)}\big)^T \nabla f\big(\mathbf{w}^{(t)}\big)$. Geometrically, we approximate the graph of $f(\mathbf{w})$ by a hyperplane with normal vector $\mathbf{n} = (\nabla f\big(\mathbf{w}^{(t)}\big), -1)^T \in \mathbb{R}^{d+1}$ of this approximating hyperplane is determined by the gradient $\nabla f\big(\mathbf{w}^{(t)}\big)$ [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 [52].

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 (76) 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 (24) 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 such as adding and multiplying numbers.

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 $t := 0$

1: **repeat**

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

3: $\mathbf{w}^{(t)} := \mathbf{w}^{(t-1)} - \eta\nabla f\big(\mathbf{w}^{(t-1)}\big)$ (do a gradient step (77))

4: **until** stopping criterion is met

Output: learned model parameters $\widehat{\mathbf{w}} := \mathbf{w}^{(t)}$ (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 $\nabla f\big(\mathbf{w}^{(t-1)}\big)$. In particular, we an apply Algorithm 2 to objective functions that do not belong to the family of convex quadratic functions (76).

4.2 How to Choose the Learning Rate

The learning rate must be chosen carefully: if it is too large, the gradient step may overshoot and diverge from the solution of (76); if it is too small, each step makes only negligible progress. Note that practical FL systems can only afford to compute a finite number gradient steps. Therefore, we must ensure that each gradient step makes a sufficiently large progress towards the optimum of the objective function. Figure 4.2 illustrates both extremes.

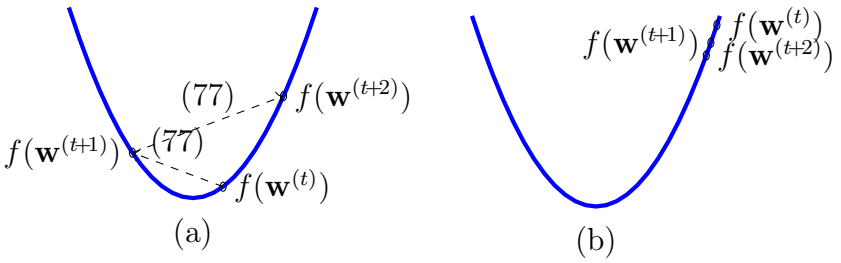


Fig. 4.2. Effect of using an inadequate learning rates η in the gradient step (77). (a) If η is too large, the gradient steps can "overshoot" such that the iterates $\mathbf{w}^{(t)}$ diverge away from the optimum, i.e., $f(\mathbf{w}^{(t+1)}) > f(\mathbf{w}^{(t)})$! (b) If η is too small, the gradient steps make too little progress towards the optimum within the available number of iterations (due to limited computational budget).

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 [53, Sec 6.1].

Alternatively, we can use a prescribed sequence (schedule) η_t, for $t = 1, 2, \ldots,$ of learning rates that vary across successive gradient steps [54]. For example, we could require the learning rate η_t to satisfy the following conditions [53, Sec. 6.1], [55]

$$\lim_{t\to\infty} \eta_t = 0, \ \sum_{t=1}^{\infty} \eta_t = \infty \ , \text{ and } \ \sum_{t=1}^{\infty} \eta_t^2 < \infty. \tag{78}$$

Running the gradient step (77) with a learning rate schedule η_t that satisfies (78) ensures convergence to a minimum of $f(\mathbf{w})$ if

- the iterates $\left\|\mathbf{w}^{(t)}\right\|_2$ are bounded, i.e., $\sup_{t=1,\ldots}\left\|\mathbf{w}^{(t)}\right\|_2$ is finite, and
- the gradients $\left\|\nabla f\left(\mathbf{w}^{(t)}\right)\right\|_2$, for $t = 1, 2, \ldots,$ are also bounded.

A detailed convergence proof can be found in [53, Sec. 3].

It is instructive to discuss the meanings of the individual conditions in (78). The first condition (78) requires that the learning rate eventually become sufficiently small to avoid overshooting. The third condition (78) ensures that this required decay of the learning rate does not take "forever". Note that the first and third condition in (78) could be satisfied by the trivial learning rate schedule $\eta_t = 0$ which is clearly not useful as the gradient step has no effect.

The trivial schedule $\eta_t = 0$ is ruled out by the middle condition of (78). This middle condition ensures that the learning rate η_t is large enough such

that the gradient steps make sufficient progress towards a minimizer of the objective function.

We emphasize that the conditions in (78) are independent of any properties of the matrix $\mathbf{Q}$ in (76). The matrix $\mathbf{Q}$ is determined by data points (see, e.g., (2)), whose statistical properties can typically be controlled only to a limited extent, such as through data normalization.

4.3 When to Stop?

For the stopping criterion, we may use a fixed number of iterations, $t_{\max}$. This hyper-parameter 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 $t_{\max}$ based on key characteristics (or features) of the objective function [56].

Another stopping criterion can be obtained by monitoring the decrease in the objective function $f\big(\mathbf{w}^{(t)}\big)$. Specifically, we stop repeating the gradient step (77) when $\big|f\big(\mathbf{w}^{(t)}\big) - f\big(\mathbf{w}^{(t+1)}\big)\big| \leq \varepsilon^{(\mathrm{tol})}$ for a given tolerance $\varepsilon^{(\mathrm{tol})}$. As before, we can optimize the tolerance level $\varepsilon^{(\mathrm{tol})}$ via meta-learning techniques [56].

For an objective function of the form (76), we can use information about the psd matrix $\mathbf{Q}$ to construct a stopping criterion.[14] Indeed, the choice of the learning rate η and the stopping criterion can be guided by the eigenvalues

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

[14] For linear regression (6), 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.

Even if we do not know these eigenvalues precisely, we might know (or be able to ensure via feature learning) some upper and lower bounds,

$$0 \leq L \leq \lambda_1(\mathbf{Q}) \leq \ldots \leq \lambda_d(\mathbf{Q}) \leq U. \tag{79}$$

In what follows, we assume that $\mathbf{Q}$ is invertible and that we know some positive lower bound $L > 0$ on its eigenvalues (see (79)). The objective function (76) has then a unique solution $\widehat{\mathbf{w}}$. A gradient step (77) reduces the distance $\big\|\mathbf{w}^{(t)} - \widehat{\mathbf{w}}\big\|_2$ to $\widehat{\mathbf{w}}$ by a constant factor [53, Ch. 6],

$$\big\|\mathbf{w}^{(t+1)} - \widehat{\mathbf{w}}\big\|_2 \leq \kappa^{(\eta_t)}\left(\mathbf{Q}\right) \big\|\mathbf{w}^{(t)} - \widehat{\mathbf{w}}\big\|_2. \tag{80}$$

Here, we used the contraction factor

$$\kappa^{(\eta)}(\mathbf{Q}) := \max\big\{|1 - \eta 2\lambda_1|, |1 - \eta 2\lambda_d|\big\}. \tag{81}$$

The contraction factor depends on the learning rate η which is a hyper-parameter of gradient-based methods that we can control. However, the contraction factor also depends on the eigenvalues of the matrix $\mathbf{Q}$ in (76). In ML and FL applications, this matrix typically depends on data and can be controlled only to some extent, e.g., using feature transformation [23, Ch. 5]. To ensure $\kappa^{(\eta)}(\mathbf{Q}) < 1$, we require a positive learning rate satisfying $\eta_t < 1/U$.

Consider the gradient step (77) with fixed learning rate η and a contraction factor $\kappa^{(\eta)}(\mathbf{Q}) < 1$ (see (81)). We can then ensure an optimization error $\big\|\mathbf{w}^{(t)} - \widehat{\mathbf{w}}\big\|_2 \leq \varepsilon$ (see (80)) if the number t of gradient steps satisfies

$$t \geq \underbrace{\left\lceil \frac{\log\left(\big\|\mathbf{w}^{(0)} - \widehat{\mathbf{w}}\big\|_2/\varepsilon\right)}{\log\left(1/\kappa^{(\eta)}(\mathbf{Q})\right)} \right\rceil}_{=:t^{(\varepsilon)}}. \tag{82}$$

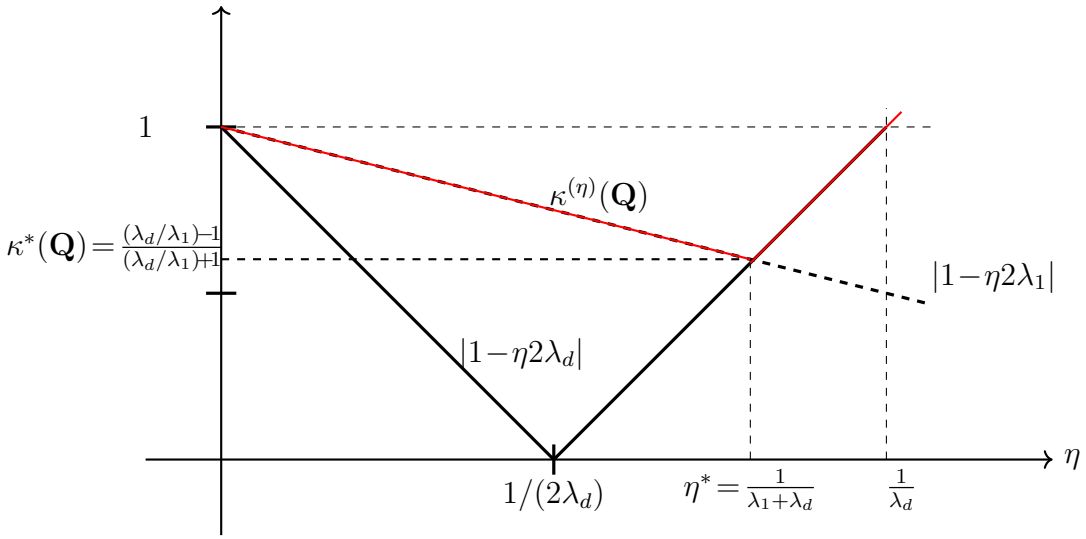


Fig. 4.3. The contraction factor $\kappa^{(\eta)}(\mathbf{Q})$ (81), used in the upper bound (80), as a function of the learning rate η. Note that $\kappa^{(\eta)}(\mathbf{Q})$ also depends on the eigenvalues of the matrix $\mathbf{Q}$ in (76).

According to (80), smaller values of the contraction factor $\kappa^{(\eta)}(\mathbf{Q})$ guarantee a faster convergence of (77) towards the solution of (76). Figure 4.3 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 $\varepsilon^{(\mathrm{tol})}$ via (80).

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

$$\eta^{(*)} := \frac{1}{\lambda_1 + \lambda_d}. \tag{83}$$

[Note that evaluating (83) requires to know the extremal eigenvalues λ_1, λ_d

of $\mathbf{Q}$.] Inserting the optimal learning rate (83) into (80),

$$\big\|\mathbf{w}^{(t+1)} - \widehat{\mathbf{w}}\big\|_2 \leq \underbrace{\frac{(\lambda_d/\lambda_1)-1}{(\lambda_d/\lambda_1)+1}}_{=:\kappa^*(\mathbf{Q})} \big\|\mathbf{w}^{(t)} - \widehat{\mathbf{w}}\big\|_2. \tag{84}$$

Carefully note that the formula (84) is valid only if the matrix $\mathbf{Q}$ in (76) is invertible, i.e., if $\lambda_1 > 0$. If the matrix $\mathbf{Q}$ is singular ($\lambda_1 = 0$), the convergence of (77) towards a solution of (76) is much slower than the decrease of the bound (84). However, we can still ensure the convergence of gradient steps $\mathbf{w}^{(t)}$ by using a fixed learning rate $\eta_t = \eta$ that satisfies [57, Thm. 2.1.14]

$$0 < \eta < 1/\lambda_d(\mathbf{Q}). \tag{85}$$

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 (15)) 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 (83) and the optimal contraction factor

$$\kappa^*(\mathbf{Q}) := \frac{(\lambda_d/\lambda_1)-1}{(\lambda_d/\lambda_1)+1} \tag{86}$$

depend on the eigenvalues of the matrix $\mathbf{Q}$ in (76).

According to (84), 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 (76).

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 (6) 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 (6) more towards a scaled identity matrix.

4.4 Perturbed Gradient Step

Consider the gradient step (77) used to find a minimum of (76). We again assume that the matrix $\mathbf{Q}$ in (76) is invertible ($\lambda_1(\mathbf{Q}) > 0$) and, in turn, (76) 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 (76) 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). Another source for imperfections can be stochastic approximation techniques where exact computations are replaced by noisy approximations that require less resources.[15]

We can model imperfections during the computation of (77) as the perturbed gradient step

$$\begin{aligned}\mathbf{w}^{(t+1)} &:= \mathbf{w}^{(t)} - \eta \nabla f\big(\mathbf{w}^{(t)}\big) + \boldsymbol{\varepsilon}^{(t)} \\ &\overset{(76)}{=} \mathbf{w}^{(t)} - \eta\big(2\mathbf{Q}\mathbf{w}^{(t)} + \mathbf{q}\big) + \boldsymbol{\varepsilon}^{(t)}, \text{ for } t = 0, 1, \ldots. \end{aligned} \tag{87}$$

We can use the contraction factor $\kappa := \kappa^{(\eta)}(\mathbf{Q})$ (81) to upper bound the

[15] A prime example for such a stochastic approximation is stochastic gradient descent (SGD) which we discuss in Section 5.3.

deviation between $\mathbf{w}^{(t)}$ and the optimum $\widehat{\mathbf{w}}$ as (see (80))

$$\big\|\mathbf{w}^{(t)}-\widehat{\mathbf{w}}\big\|_2 \leq \kappa^t \big\|\mathbf{w}^{(0)}-\widehat{\mathbf{w}}\big\|_2 + \sum_{t'=1}^{t} \kappa^{t'} \Big\|\boldsymbol{\varepsilon}^{(t-t')}\Big\|_2. \tag{88}$$

This bound applies for any number of iterations $t = 1, 2, \ldots$ of the perturbed gradient step (87).

The perturbed gradient step (87) 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 (76) of convex quadratic functions. Indeed, we can write the gradient step for minimizing $\tilde{f}(\mathbf{w})$ as

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

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 (76) that approximates $\tilde{f}(\mathbf{w})$. Note that the perturbation term $\boldsymbol{\varepsilon}^{(t)}$ is scaled by the learning rate η.

4.5 Handling Constraints - Projected Gradient Descent

Many important ML and FL methods amount to the minimization of an objective function of the form (76). The optimization variable $\mathbf{w}$ in (76) 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 (77) to solve the constrained problem

$$f^* = \min_{\mathbf{w} \in \mathcal{S}} \mathbf{w}^T \mathbf{Q} \mathbf{w} + \mathbf{q}^T \mathbf{w}. \tag{89}$$

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

$$P_{\mathcal{S}}\big(\mathbf{w}\big) = \arg\min_{\mathbf{w}' \in \mathcal{S}} \|\mathbf{w} - \mathbf{w}'\|_2 \text{ for any } \mathbf{w} \in \mathbb{R}^d. \tag{90}$$

A suitable modification of the gradient step (77) to solve the constrained variant (89) is [53]

$$\begin{aligned} \mathbf{w}^{(t+1)} &:= P_{\mathcal{S}}\big(\mathbf{w}^{(t)} - \eta \nabla f\big(\mathbf{w}^{(t)}\big)\big) \\ &\stackrel{(76)}{=} P_{\mathcal{S}}\big(\mathbf{w}^{(t)} - \eta\big(2\mathbf{Q}\mathbf{w}^{(t)} + \mathbf{q}\big)\big). \end{aligned} \tag{91}$$

The projected GD step (91) consists of:

1. computing an ordinary gradient step $\mathbf{w}^{(t)} \mapsto \mathbf{w}^{(t)} - \eta \nabla f\big(\mathbf{w}^{(t)}\big)$ and then
2. projecting the result back to the constraint set $\mathcal{S}$.

Note that we re-obtain the basic gradient step (77) from the projected gradient step (91) 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 (77) explained in Sections 4.2 and 4.3 work also for

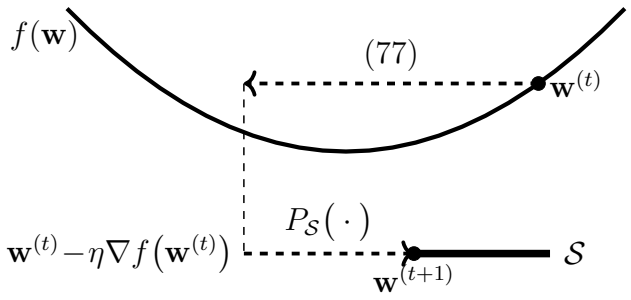


Fig. 4.4. Projected GD augments a basic gradient step with a projection back onto the constraint set $\mathcal{S}$.

the projected gradient step (91). In particular, the convergence speed of the projected gradient step is also characterized by (80) [53, Ch. 6]. This follows from the fact that the concatenation of a contraction (such as the gradient step (77) 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 (80) is only telling about the number of projected gradient steps (91) required to achieve a guaranteed level of sub-optimality $\big|f\big(\mathbf{w}^{(t)}\big) - f^{*}\big|$. The iteration (91) of projected GD might require significantly more computation than the basic gradient step, as it requires to compute the projection (90).

4.6 Extended Gradient Methods for Federated Optimization

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 (77) for a model $\mathcal{H}$ without a parametrization.

We start with rewriting the gradient step (77) as the optimization

$$\mathbf{w}^{(t+1)} = \operatorname*{arg\,min}_{\mathbf{w}\in\mathbb{R}^d} \left[(1/(2\eta))\big\|\mathbf{w}-\mathbf{w}^{(t)}\big\|_2^2 + \underbrace{f\big(\mathbf{w}^{(t)}\big)+\big(\mathbf{w}-\mathbf{w}^{(t)}\big)^T\nabla f\big(\mathbf{w}^{(t)}\big)}_{\approx f(\mathbf{w})} \right]. \tag{92}$$

The objective function in (92) includes the first-order approximation

$$f(\mathbf{w}) \approx f\big(\mathbf{w}^{(t)}\big) + \big(\mathbf{w}-\mathbf{w}^{(t)}\big)^T\nabla f\big(\mathbf{w}^{(t)}\big)$$

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

Let us modify (92) by using $f(\mathbf{w})$ itself (instead of an approximation),

$$\mathbf{w}^{(t+1)} = \operatorname*{arg\,min}_{\mathbf{w}\in\mathbb{R}^d} \left[f(\mathbf{w}) + (1/(2\eta))\big\|\mathbf{w}-\mathbf{w}^{(t)}\big\|_2^2 \right]. \tag{93}$$

Like the gradient step, also (93) maps a given vector $\mathbf{w}^{(t)}$ to an updated vector $\mathbf{w}^{(t+1)}$. Note that (93) is nothing but the proximal operator of the function $f(\mathbf{w})$ [39]. Similar to the role of the gradient step as the main building block of gradient-based methods, the proximal operator (93) is the main building block of proximal algorithms [39].

To obtain a version of (93) 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 (93) consists of two components.

The first 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 second component of the objective function in (93) uses $\left\|\mathbf{w}-\mathbf{w}^{(t)}\right\|_2^2$ to measure the difference between the hypothesis maps $h^{(\mathbf{w})}$ and $h^{(\mathbf{w}^{(t)})}$. Another measure for the difference between two hypothesis maps can be obtained by using some test dataset $\mathcal{D}' = \big\{\mathbf{x}^{(1)},\ldots,\mathbf{x}^{(m')}\big\}$: The average squared difference between their predictions,

$$(1/m')\sum_{r=1}^{m'}\Big(h\big(\mathbf{x}^{(r)}\big)-h^{(t)}\big(\mathbf{x}^{(r)}\big)\Big)^2, \tag{94}$$

is a measure for the difference between h and $h^{(t)}$. Note that (94) only requires the predictions delivered by the hypothesis maps $h, h^{(t)}$ on $\mathcal{D}'$ - no other information is needed about these maps.

It is interesting to note that (94) coincides with $\left\|\mathbf{w}-\mathbf{w}^{(t)}\right\|_2^2$ 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

$$\begin{aligned}
&\lim_{m' \to \infty} (1/m') \sum_{r=1}^{m'} \left(h^{(\mathbf{w})}\big(\mathbf{x}^{(r)}\big) - h^{(\mathbf{w}^{(t)})}\big(\mathbf{x}^{(r)}\big) \right)^2 \\
&= \lim_{m' \to \infty} (1/m') \sum_{r=1}^{m'} \left(\big(\mathbf{w} - \mathbf{w}^{(t)}\big)^T \mathbf{x}^{(r)} \right)^2 \\
&= \lim_{m' \to \infty} (1/m') \sum_{r=1}^{m'} \big(\mathbf{w} - \mathbf{w}^{(t)}\big)^T \mathbf{x}^{(r)} \big(\mathbf{x}^{(r)}\big)^T \big(\mathbf{w} - \mathbf{w}^{(t)}\big) \\
&= \big(\mathbf{w} - \mathbf{w}^{(t)}\big)^T \underbrace{\left[\lim_{m' \to \infty} (1/m') \sum_{r=1}^{m'} \mathbf{x}^{(r)} \big(\mathbf{x}^{(r)}\big)^T \right]}_{=\mathbf{I}} \big(\mathbf{w} - \mathbf{w}^{(t)}\big) \\
&= \big\|\mathbf{w} - \mathbf{w}^{(t)}\big\|_2^2. \qquad (95)
\end{aligned}$$

Finally, we arrive at a generalized gradient step for the training of a non-parametric model $\mathcal{H}$ by replacing $\big\|\mathbf{w} - \mathbf{w}^{(t)}\big\|_2^2$ in (93) with (94). In other words,

$$h^{(t+1)} = \operatorname*{arg\,min}_{h \in \mathcal{H}} \left[(1/(2\eta m')) \sum_{r=1}^{m'} \left(h\big(\mathbf{x}^{(r)}\big) - h^{(t)}\big(\mathbf{x}^{(r)}\big) \right)^2 + f(h) \right]. \qquad (96)$$

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 (96). 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 $t := 0$

1: **repeat**

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

3: do a generalized gradient step (96),

$$h^{(t)} = \arg\min_{h \in \mathcal{H}} \left[(1/(2\eta m')) \sum_{r=1}^{m'} \left(h\big(\mathbf{x}^{(r)}\big) - h^{(t-1)}\big(\mathbf{x}^{(r)}\big) \right)^2 + f(h) \right]$$

4: **until** stopping criterion is met

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

4.7 Gradient Methods as Fixed-Point Iterations

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

$$\mathbf{w}^{(t)} = \mathcal{F}\mathbf{w}^{(t-1)}, \text{ for } t = 1, 2, \ldots. \tag{97}$$

Different optimization methods use different choices for the operator $\mathcal{F}$ whose fixed points are solutions of the underlying optimization problem. For example, the gradient step (77) is obtained from (97) with the operator $\mathcal{F}^{(\mathrm{GD})} : \mathbf{w} \mapsto \mathbf{w} - \eta \nabla f(\mathbf{w})$. For a differentiable and convex objective function $f(\mathbf{w})$, every minimizer $\widehat{\mathbf{w}}$ is a fixed point of $\mathcal{F}^{(\mathrm{GD})}$.

The fixed-point iteration (97) will be the core computational step of every FL algorithm discussed in Chapter 5. These algorithms use (97) with an

operator $\mathcal{F}$ determined by an instance of GTVMin. More precisely, any fixed point of $\mathcal{F}$ must be an GTVMin-solution $\widehat{\mathbf{w}} \in \mathbb{R}^{dn}$,

$$\mathcal{F}\widehat{\mathbf{w}} = \widehat{\mathbf{w}}. \tag{98}$$

Given an instance of GTVMin, there are many different operators $\mathcal{F}$ that satisfy (98). We obtain different FL algorithms by using different choices for $\mathcal{F}$ in (97). Clearly, we should use an operator $\mathcal{F}$ in (97) that reduces the distance to a solution,

$$\underbrace{\big\|\mathbf{w}^{(t+1)} - \widehat{\mathbf{w}}\big\|_2}_{\overset{(97),(98)}{=}\big\|\mathcal{F}\mathbf{w}^{(t)} - \mathcal{F}\widehat{\mathbf{w}}\big\|_2} \leq \big\|\mathbf{w}^{(t)} - \widehat{\mathbf{w}}\big\|_2.$$

Thus, we require $\mathcal{F}$ to be at least non-expansive, i.e., the iteration (97) should not result in worse model parameters that have a larger distance to the GTVMin solution. Moreover, each iteration (97) 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 [58, 59].

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

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

For a contractive operator $\mathcal{F}$, the fixed-point iteration (97) generates a sequence $\mathbf{w}^{(t)}$ that converges to a GTVMin solution $\widehat{\mathbf{w}}$ quite rapidly. In particular [2, Theorem 9.23],

$$\big\|\mathbf{w}^{(t)} - \widehat{\mathbf{w}}\big\|_2 \leq \kappa^t \big\|\mathbf{w}^{(0)} - \widehat{\mathbf{w}}\big\|_2.$$

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

A well-known example of a fixed-point iteration (97) using a contractive operator is GD (77) for a smooth and strongly convex objective function $f(\mathbf{w})$.[16] In particular, (77) is obtained from (97) using $\mathcal{F} := \mathcal{G}^{(\eta)}$ with the "gradient step operator"

$$\mathcal{G}^{(\eta)} : \mathbf{w} \mapsto \mathbf{w} - \eta \nabla f(\mathbf{w}). \tag{99}$$

Note that the operator (99) is parametrized by the learning rate η.

It is instructive to study the operator $\mathcal{G}^{(\eta)}$ for an objective function of the form (76). Here,

$$\mathcal{G}^{(\eta)} : \mathbf{w} \mapsto \mathbf{w} - \eta \underbrace{\big(2\mathbf{Q}\mathbf{w} + \mathbf{q}\big)}_{\overset{(76)}{=}\nabla f(\mathbf{w})}. \tag{100}$$

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 (76) is invertible.

The gradient step operator (100) is not contractive for the objective function (76) with a singular matrix $\mathbf{Q}$ (for which $\lambda_{\min} = 0$). However, even then $\mathcal{G}^{(\eta)}$ is still firmly non-expansive [22]. We refer to an operator $\mathcal{F} : \mathbb{R}^{dn} \to \mathbb{R}^{dn}$ as firmly non-expansive if

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

It turns out that a fixed-point iteration (97) with a firmly non-expansive operator $\mathcal{F}$ is guaranteed to converge to a fixed-point of $\mathcal{F}$ [58, Cor. 5.16]. Figure 4.5 depicts examples of a firmly non-expansive operator, a non-expansive

[16] The objective function in (76) 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.

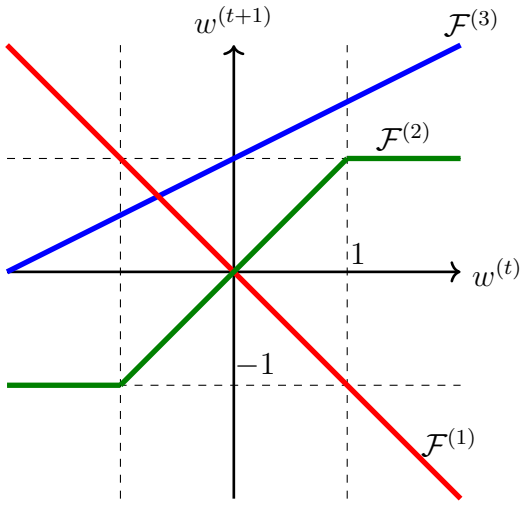


Fig. 4.5. Example of a non-expansive operator $\mathcal{F}^{(1)}$, a firmly non-expansive operator $\mathcal{F}^{(2)}$ and a contractive operator $\mathcal{F}^{(3)}$.

operator and a contractive operator. All these operators are defined on the one-dimensional space $\mathbb{R}$. Another example of a firmly non-expansive operator is the proximal operator (93) of a convex function [39, 58].

4.8 Exercises

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

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

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

1. Verify that this learning rate schedule satisfies the standard conditions in (78).

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. Learning Rate Schedule II. Consider the generic gradient step

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

with a learning rate schedule of the form $\eta_t := \frac{1}{t^p}$ with some $p > 0$. For which values of $p > 0$ does this schedule satisfy the conditions in (78)?

4.3. 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 $\big(\mathbf{x}^{(t)}, y^{(t)}\big)$,

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

Starting with initialization $\mathbf{w}^{(1)} := \mathbf{0}$, we run online gradient descent (online GD) for M time steps, resulting in the learned 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 $\eta_t := 1/\sqrt{t+5}$.

4.4. 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

$$\mathbf{u}^{(t+1)} = \mathbf{u}^{(t)} - \eta \mathbf{L}^{(\mathcal{G})} \mathbf{u}^{(t)} \text{ for } t = 1, 2, \ldots. \tag{101}$$

Can you find a choice for η such that (101) becomes a fixed-point iteration (97) with a contractive operator $\mathcal{F}$. Given such a choice of η, how is the limit $\lim_{t\to\infty} \mathbf{u}^{(t+1)}$ related to the average $(1/n)\sum_{i=1}^{n} y^{(i)}$?

4.5. Computing the Average - II. Consider the FL network from Problem 4.4. 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.

4.6. How to Quantize the Gradients? Any ML and FL application that uses a digital computer to implement a gradient step (77) 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 4.4.

4.7. 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 message 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 5.1 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 (5.2) spells out the gradient step from Section 5.1 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 5.3 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 5.4 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 have discussed in Section 4.5.

Section 5.6 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 the local models at two nodes i and i' on a common test-set. By comparing their predictions, we can 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 5.8 explores the design of FL algorithms that support asynchronous operation. These algorithms allow devices to update and communicate at different times within the FL system

5.1 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

$$\big\{\widehat{\mathbf{w}}^{(i)}\big\}_{i=1}^{n} \in \operatorname*{arg\,min}_{\{\mathbf{w}^{(i)}\}} \underbrace{\sum_{i \in \mathcal{V}} \overbrace{(1/m_i)\big\|\mathbf{y}^{(i)} - \mathbf{X}^{(i)}\mathbf{w}^{(i)}\big\|_2^2}^{\text{local loss } L_i\left(\mathbf{w}^{(i)}\right)} + \alpha \sum_{\{i,i'\} \in \mathcal{E}} A_{i,i'} \Big\|\mathbf{w}^{(i)} - \mathbf{w}^{(i')}\Big\|_2^2}_{=: f(\mathbf{w})}. \tag{102}$$

As discussed in Chapter 3, the objective function in (102) - viewed as a

function of the stacked local model parameters $\mathbf{w} := \mathrm{stack}\{\mathbf{w}^{(i)}\}_{i=1}^{n}$ - is a quadratic function

$$\mathbf{w}^T \left(\begin{pmatrix} \mathbf{Q}^{(1)} & \cdots & \mathbf{0} \\ \vdots & \ddots & \vdots \\ \mathbf{0} & \cdots & \mathbf{Q}^{(n)} \end{pmatrix} + \alpha \mathbf{L}^{(\mathcal{G})} \otimes \mathbf{I} \right) \mathbf{w} + \big(\big(\mathbf{q}^{(1)}\big)^T, \ldots, \big(\mathbf{q}^{(n)}\big)^T \big) \mathbf{w} \quad (103)$$

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

Note that (103) is a special case of the generic quadratic function (76) studied in Chapter 4. Indeed, we obtain (103) from (76) for the choices

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

Therefore, the discussion and analysis of gradient-based methods from Chapter 4 also apply to GTVMin (102). In particular, we can use the gradient step

$$\begin{aligned} \mathbf{w}^{(t+1)} &:= \mathbf{w}^{(t)} - \eta \nabla f\big(\mathbf{w}^{(t)}\big) \\ &\overset{(103)}{=} \mathbf{w}^{(t)} - \eta\big(2\mathbf{Q}\mathbf{w}^{(t)} + \mathbf{q}\big) \end{aligned} \quad (104)$$

to iteratively compute an approximate solution $\widehat{\mathbf{w}}$ to (102). This solution consists of learned local model parameters $\widehat{\mathbf{w}}^{(i)}$, i.e., $\widehat{\mathbf{w}} = \mathrm{stack}\{\widehat{\mathbf{w}}^{(i)}\}$. Section 5.2 formulates the gradient step (104) 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 (104) is determined by the eigenvalues $\lambda_j(\mathbf{Q})$ of the matrix $\mathbf{Q}$ in (103). In general, these eigenvalues depend on the eigenvalues $\lambda_j\big(\mathbf{Q}^{(i)}\big)$

as well as the eigenvalues $\lambda_j\big(\mathbf{L}^{(\mathcal{G})}\big)$ of the Laplacian matrix $\mathbf{L}^{(\mathcal{G})}$. In particular, we will use the following two summary parameters

$$\lambda_{\max} := \max_{i=1,\ldots,n} \lambda_d\big(\mathbf{Q}^{(i)}\big), \text{ and } \bar{\lambda}_{\min} := \lambda_1\bigg((1/n)\sum_{i=1}^{n}\mathbf{Q}^{(i)}\bigg). \tag{105}$$

We first present an upper bound U (see (79)) on the eigenvalues of the matrix $\mathbf{Q}$ in (103).

Proposition 5.1. *The eigenvalues of* $\mathbf{Q}$ *in* (103) *are upper-bounded as*

$$\begin{aligned}\lambda_j(\mathbf{Q}) &\leq \lambda_{\max} + \alpha\lambda_n\big(\mathbf{L}^{(\mathcal{G})}\big) \\ &\leq \underbrace{\lambda_{\max} + 2\alpha d_{\max}^{(\mathcal{G})}}_{=:U} \text{ , for } j = 1,\ldots,dn. \end{aligned} \tag{106}$$

Proof. See Section 5.10.1. □

Note how the upper bound (106) involves properties of

- the local datasets, via $\lambda_{\max}$ (see (105)),
- the FL network, via the maximum node degree $d_{\max}^{(\mathcal{G})}$ (see (35)), and
- the GTVMin parameter α.

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

Proposition 5.2. *Consider the matrix* $\mathbf{Q}$ *in* (103). *If* $\lambda_2\big(\mathbf{L}^{(\mathcal{G})}\big) > 0$ *(i.e., the FL network in* (102) *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*

$$\lambda_1(\mathbf{Q}) \geq \frac{1}{1+\rho^2}\min\{\lambda_2\big(\mathbf{L}^{(\mathcal{G})}\big)\alpha\rho^2, \bar{\lambda}_{\min}/2\}. \tag{107}$$

Here, we used the shorthand $\rho := \bar{\lambda}_{\min}/(4\lambda_{\max})$ *(see* (105)*).*

Proof. See Section 5.10.2. □

Proposition 5.1 and Proposition 5.2 provide some guidance for the design choices of GTVMin. According to the convergence analysis of gradient-based methods in Chapter 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 (106) and the lower bound (107). A small ratio between these bounds, in turn, requires a large eigenvalue $\lambda_2\big(\mathbf{L}^{(\mathcal{G})}\big)$ and small node degree $d^{(\mathcal{G})}_{\max}$.[17]

The bounds in (106) and (107) 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 (104) on α is complicated. For a fixed value of learning rate in (104), using larger values for α might slow down the convergence of (104) for some collection of local datasets but speed up the convergence of (104) for another collection of local datasets (see Exercise 5.1).

5.2 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 (102)). The core of gradient-based methods is the gradient step

$$\mathbf{w}^{(t+1)} := \mathbf{w}^{(t)} - \eta \nabla f\big(\mathbf{w}^{(t)}\big). \tag{108}$$

[17]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 [60, 61].

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

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

Inserting (102) into (108), we obtain the gradient step

$$\mathbf{w}^{(i,t+1)} := \mathbf{w}^{(i,t)} - \eta\Bigg[\underbrace{(2/m_i)\big(\mathbf{X}^{(i)}\big)^T\big(\mathbf{X}^{(i)}\mathbf{w}^{(i,t)} - \mathbf{y}^{(i)}\big)}_{\text{(I)}} + \underbrace{2\alpha\sum_{i'\in\mathcal{N}^{(i)}} A_{i,i'}\big(\mathbf{w}^{(i,t)} - \mathbf{w}^{(i',t)}\big)}_{\text{(II)}}\Bigg]. \tag{109}$$

We slightly modify this gradient step by allowing for different learning rates $\eta_{t,i}$ at different nodes i and iterations t,

$$\mathbf{w}^{(i,t+1)} := \mathbf{w}^{(i,t)} - \eta_{t,i}\Bigg[\underbrace{(2/m_i)\big(\mathbf{X}^{(i)}\big)^T\big(\mathbf{X}^{(i)}\mathbf{w}^{(i,t)} - \mathbf{y}^{(i)}\big)}_{\text{(I)}} + \underbrace{2\alpha\sum_{i'\in\mathcal{N}^{(i)}} A_{i,i'}\big(\mathbf{w}^{(i,t)} - \mathbf{w}^{(i',t)}\big)}_{\text{(II)}}\Bigg]. \tag{110}$$

The update (110) consists of two components, denoted (I) and (II). Component (I) reflects the local loss function at node i while component (II) couples node i with its neighbors $i' \in \mathcal{N}^{(i)}$. In particular, component (I) is the gradient $\nabla L_i\left(\mathbf{w}^{(i,t)}\right)$ of the local loss $L_i\left(\mathbf{w}^{(i)}\right) := (1/m_i)\big\|\mathbf{y}^{(i)} - \mathbf{X}^{(i)}\mathbf{w}^{(i)}\big\|_2^2$. Component (I) drives the updated local model parameters $\mathbf{w}^{(i,t+1)}$ towards the minimum of $L_i(\cdot)$, i.e., having a small deviation between labels $y^{(i,r)}$ and the predictions $\big(\mathbf{w}^{(i,t+1)}\big)^T\mathbf{x}^{(i,r)}$. Note that we can rewrite the component (I) in (110), as

$$(2/m_i)\sum_{r=1}^{m_i}\mathbf{x}^{(i,r)}\big(y^{(i,r)} - \big(\mathbf{x}^{(i,r)}\big)^T\mathbf{w}^{(i,t)}\big). \tag{111}$$

The component (II) in (110) The purpose of component (II) in (110) 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.

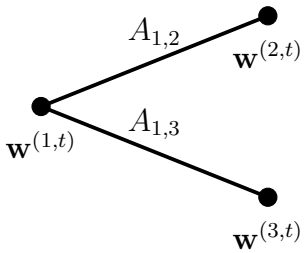


Fig. 5.1. At the beginning of iteration t, node $i = 1$ collects the current local model parameters $\mathbf{w}^{(2,t)}$ and $\mathbf{w}^{(3,t)}$ from its neighbors. Then, it computes the gradient step (110) to obtain the new local model parameters $\mathbf{w}^{(1,t+1)}$. These updated parameters are then used in the next iteration for the local updates at the neighbors $i = 2, 3$.

The execution of the gradient step (110) requires only local information at node i. Indeed, the update (110) at node i depends only on its current model parameters $\mathbf{w}^{(i,t)}$, the local loss function $L_i(\cdot)$, the neighbors' model parameters $\mathbf{w}^{(i',t)}$, for $i' \in \mathcal{N}^{(i)}$, and the corresponding edge weights $A_{i,i'}$ (see Figure 5.1). In particular, the update (110) does not depend on any properties tor edge weights) of the FL network beyond the neighbors $\mathcal{N}^{(i)}$.

We obtain Algorithm 4 by repeating the gradient step (110), simultaneously for each node $i \in \mathcal{V}$, until a stopping criterion is met. Algorithm 4 allows for

potentially different learning rates $\eta_{t,i}$ at different nodes i and iterations t. It

Algorithm 4 FedGD for Local Linear Models

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

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

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

1: **while** stopping criterion is not satisfied **do**
2: **for** all nodes $i \in \mathcal{V}$ (simultaneously) **do**
3: share local model parameters $\mathbf{w}^{(i,t)}$ with neighbors $i' \in \mathcal{N}^{(i)}$
4: update local model parameters via (110)
5: **end for**
6: increment iteration counter: $t := t+1$
7: **end while**
8: $\widehat{\mathbf{w}}^{(i)} := \mathbf{w}^{(i,t)}$ for all nodes $i \in \mathcal{V}$

is important to note that Algorithm 4 requires a synchronous (simultaneous) execution of the updates (110) at all nodes $i \in \mathcal{V}$ [17, 18]. Loosely speaking, all nodes i rely on a single global clock that maintains the current iteration counter t [62].

At the beginning of iteration t, each node $i \in \mathcal{V}$ sends its current model parameters $\mathbf{w}^{(i,t)}$ to their neighbors $i' \in \mathcal{N}^{(i)}$. Then, each node $i \in \mathcal{V}$ updates their model parameters according to (110), resulting in the updated model parameters $\mathbf{w}^{(i,t+1)}$. As soon as these local updates are completed, the global clock increments the counter $t \mapsto t+1$ and triggers the next iteration to be executed by all nodes. Figure 5.2 illustrates the alternating execution of

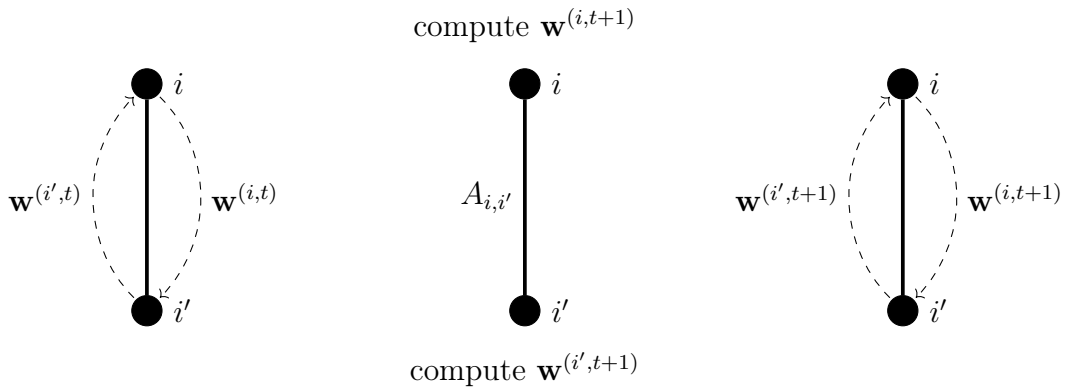


Fig. 5.2. Algorithm 4 alternates between message passing across the edges of the FL network (left and right) and updates of local model parameters (centre).

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 (110) [63, Sec. 10]. This deviation can be modelled as a perturbation of the gradient step (108) and therefore analyzed using the concepts of Section 4.4 on perturbed GD. Section 8.2 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 to 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 $\eta_{t,i}$

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

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

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

3: share local model parameters $\mathbf{w}^{(i,t)}$ with neighbors $i' \in \mathcal{N}^{(i)}$

4: update local model parameters via

$$\mathbf{w}^{(i,t+1)} := \mathbf{w}^{(i,t)} - \eta_{t,i}\left[\nabla L_i\left(\mathbf{w}^{(i,t)}\right) + 2\alpha \sum_{i' \in \mathcal{N}^{(i)}} A_{i,i'}\left(\mathbf{w}^{(i,t)} - \mathbf{w}^{(i',t)}\right)\right].$$

5: **end for**

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

7: **end while**

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

5.3 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 (111). 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

$$\underbrace{(2/B) \sum_{r \in \mathcal{B}} \mathbf{x}^{(i,r)} \big(y^{(i,r)} - \big(\mathbf{x}^{(i,r)} \big)^T \mathbf{w}^{(i,t)} \big)}_{\approx (111)}. \tag{112}$$

The approximation (112) 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 (111) requires summing over m data points, the approximation requires to sum over B (typically $B \ll m$) data points.

Inserting the approximation (112) into the gradient step (110) yields the approximate gradient step

$$\mathbf{w}^{(i,t+1)} := \mathbf{w}^{(i,t)} - \eta_{t,i} \Bigg[\underbrace{(2/B) \sum_{r \in \mathcal{B}} \mathbf{x}^{(i,r)} \Big(\big(\mathbf{x}^{(i,r)} \big)^T \mathbf{w}^{(i,t)} - y^{(i,r)} \Big)}_{\approx (111)} + 2\alpha \sum_{i' \in \mathcal{N}^{(i)}} A_{i,i'} \big(\mathbf{w}^{(i,t)} - \mathbf{w}^{(i',t)} \big) \Bigg]. \tag{113}$$

We obtain Algorithm 6 from Algorithm 4 by replacing the gradient step (110) with the approximation (113).

We close this section by generalizing Algorithm 6 which is limited FL networks using local linear models. This generalization, summarized in Algorithm

Algorithm 6 FedSGD for Local Linear Models

Input: FL network $\mathcal{G}$; GTV parameter α; learning rate $\eta_{t,i}$; 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: $t{:=}0$; $\mathbf{w}^{(i,0)}{:=}\mathbf{0}$

while stopping criterion is not satisfied **do**
  for all nodes $i \in \mathcal{V}$ (simultaneously) **do**
    share local model parameters $\mathbf{w}^{(i,t)}$ with all neighbors $i' \in \mathcal{N}^{(i)}$
    draw fresh batch $\mathcal{B}^{(i)} := \{r_1, \ldots, r_B\}$
    update local model parameters via (113)
  end for
  increment iteration counter $t{:=}t{+}1$
end while
$\widehat{\mathbf{w}}^{(i)} := \mathbf{w}^{(i,t)}$ for all nodes $i \in \mathcal{V}$

7, 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 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 (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 $\eta_{t,i}$
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: $t:=0$; $\mathbf{w}^{(i,0)}:=\mathbf{0}$

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

$$\mathbf{w}^{(i,t+1)} := \mathbf{w}^{(i,t)} - \eta_{t,i}\left[\mathbf{g}^{(i)}\big(\mathbf{w}^{(i,t)}\big) + 2\alpha \sum_{i' \in \mathcal{N}^{(i)}} A_{i,i'}\big(\mathbf{w}^{(i,t)} - \mathbf{w}^{(i',t)}\big)\right].$$

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

5.4 FedAvg

Consider an FL method that learns model parameters $\widehat{\mathbf{w}} \in \mathbb{R}^d$ of a single (global) linear model from a de-centralized collection of local datasets $\mathcal{D}^{(i)}$, $i = 1, \ldots, n$. [18] 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 (102) with a sufficiently large α. According to our analysis in Chapter 3 (specifically Proposition 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 Proposition 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 [64].

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:

$$\widehat{\mathbf{w}} \in \arg\min_{\mathbf{w} \in \mathcal{S}} \sum_{i \in \mathcal{V}} (1/m_i) \big\| \mathbf{y}^{(i)} - \mathbf{X}^{(i)} \mathbf{w}^{(i)} \big\|_2^2$$
$$\text{with } \mathcal{S} = \big\{ \mathbf{w} = \text{stack}\{\mathbf{w}^{(i)}\}_{i=1}^n : \mathbf{w}^{(i)} = \mathbf{w}^{(i')} \text{ for any } i, i' \in \mathcal{V} \big\}. \quad (114)$$

Here, we use as constraint set the subspace $\mathcal{S}$ defined in (45). The projection of a given collection of local model parameters $\mathbf{w} = \text{stack}\{\mathbf{w}^{(i)}\}$ on $\mathcal{S}$ is given

[18] This setting is a special case of horizontal federated learning (HFL) which we discuss in Section 6.3.

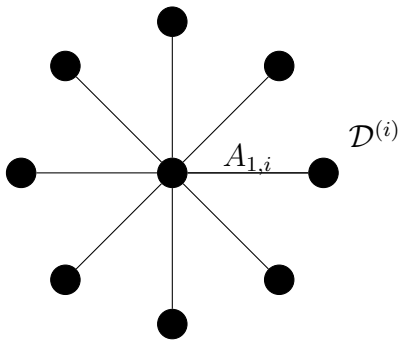


Fig. 5.3. Star-shaped graph $\mathcal{G}^{(\text{star})}$ with a centre node $i = 1$ representing a server that trains a (global) model which is shared with peripheral nodes. These peripheral nodes represent *clients* generating local datasets. The training process at the server is facilitated by receiving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

by

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

We can solve (114) using projected GD from Chapter 4. The resulting projected gradient step for solving (114) is

$$\widehat{\mathbf{w}}^{(i)}_{t+1/2} := \underbrace{\mathbf{w}^{(i,t)} - \eta_{i,t}(2/m_i)\big(\mathbf{X}^{(i)}\big)^T\big(\mathbf{X}^{(i)}\mathbf{w}^{(i,t)} - \mathbf{y}^{(i)}\big)}_{\text{(local gradient step)}} \tag{115}$$

$$\mathbf{w}^{(i,t+1)} := (1/n) \sum_{i' \in \mathcal{V}} \widehat{\mathbf{w}}^{(i')}_{t+1/2} \quad \text{(projection)} \ . \tag{116}$$

We can implement (116) conveniently in a server-client system with each node i being a client:

- First, each node computes the update (115), 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 $\widehat{\mathbf{w}}_t^{(i)}$ of its local gradient step to a server.

- Finally, after receiving the updates $\widehat{\mathbf{w}}_t^{(i)}$ from all nodes $i \in \mathcal{V}$, the server computes the projection step (116). This projection results in the new local model parameters $\mathbf{w}^{(i,t+1)}$ that are sent back to each client i.

The averaging step (116) might take much longer to execute than the local update step (115). Indeed, (116) 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 (115), it must wait until the server (i) has collected the updates $\widehat{\mathbf{w}}_t^{(i)}$ from all clients and (ii) sent back their average $\mathbf{w}^{(i,t+1)}$ to $i \in \mathcal{V}$.

Instead of using a single gradient step (115),[19] and then being forced to wait for receiving $\mathbf{w}^{(i,t+1)}$ back from the server, a client can make better use of its resources. For example, the device i could execute several local gradient steps (115) to make more progress towards the optimum,

$$\begin{aligned} \mathbf{v}^{(0)} &:= \widehat{\mathbf{w}}_t^{(i)} \\ \mathbf{v}^{(r)} &:= \mathbf{v}^{(r-1)} - \eta_{i,t}(2/m_i)\big(\mathbf{X}^{(i)}\big)^T\big(\mathbf{X}^{(i)}\mathbf{v}^{(r-1)} - \mathbf{y}^{(i)}\big), \text{ for } r = 1, \ldots, R \\ \widehat{\mathbf{w}}_{t+1/2}^{(i)} &:= \mathbf{v}^{(R)}. \end{aligned} \tag{117}$$

We obtain Algorithm 8 by iterating the combination of (117) with the projection step (116).

[19] For a large local dataset, the local gradient step (115) can become computationally too expensive and must be replaced by an approximation, e.g., using a stochastic gradient approximation (112).

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

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

2: Update the global model parameters

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

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

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

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

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) $\eta_{i,t}$.

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

2: Update the local model parameters by R gradient steps

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

3: Send the new local model parameters $\mathbf{w}^{(i,t+1)}$ 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 $\mathbf{v}^{(r)} := \mathbf{v}^{(r-1)} - \eta_{i,t}\mathbf{g}\big(\mathbf{v}^{(r)}\big)$ 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 $\mathcal{C}^{(t)}$ of clients during each global iteration t.

Algorithm 9 FedAvg [12]

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

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

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

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

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

5: update global model parameters

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

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

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

$$\begin{aligned} \mathbf{v}^{(0)} &:= \widehat{\mathbf{w}}^{(\text{global})} \\ \mathbf{v}^{(r)} &:= \mathbf{v}^{(r-1)} - \eta_{i,t} \underbrace{\mathbf{g}^{(i)}\left(\mathbf{v}^{(r-1)}\right)}_{\approx \nabla L_i\left(\mathbf{v}^{(r-1)}\right)}, \text{ for } r = 1, \ldots, R \\ \mathbf{w}^{(i)} &:= \mathbf{v}^{(R)}. \end{aligned} \tag{118}$$

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

5.5 FedProx

A central challenge in FedAvg (Algorithm 9) is selecting an appropriate number of local updates, R, in (118). In each iteration, all clients perform exactly R approximate gradient steps. However, [65] argues that enforcing a uniform number R across clients can degrade performance in certain FL settings. To mitigate this, they propose an alternative to (118) for the local update step. This alternative is given by

$$\mathbf{w}^{(i)} := \operatorname*{arg\,min}_{\mathbf{v} \in \mathbb{R}^d} \left[L_i\left(\mathbf{v}\right) + (1/\eta) \left\| \mathbf{v} - \widehat{\mathbf{w}}^{(\text{global})} \right\|_2^2 \right]. \tag{119}$$

We have already encountered an update of the form (119) in Section 4.6. Indeed, (119) is the application of the proximal operator of $L_i\left(\mathbf{v}\right)$ (see (93)) to the current model parameters. We obtain Algorithm 10 from Algorithm 9 by replacing the local update step (118) with (119). 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 $\mathcal{D}^{(i)}$ [65].

As the notation in (119) indicates, the parameter η plays a role similar to the learning rate of a gradient step (77). It controls the size of the neighbourhood of $\mathbf{w}^{(i,t)}$ over which (119) optimizes the local loss function $L_i\left(\cdot\right)$. Choosing a small η forces the update (119) to not move too far from the current model parameters $\mathbf{w}^{(i,t)}$.

The core computation (120) of FedProx Algorithm 10 can be interpreted as form of regularization. Indeed, we obtain (120) from (22) by

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

Algorithm 10 FedProx [65]

The Server.

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

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

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

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

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

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

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

5: update global model parameters

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

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

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

$$\mathbf{w}^{(i)} := \underset{\mathbf{v} \in \mathbb{R}^d}{\arg\min} \left[L_i(\mathbf{v}) + (1/\eta) \left\| \mathbf{v} - \widehat{\mathbf{w}}^{(\text{global})} \right\|_2^2 \right] \tag{120}$$

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

$L_i(\mathbf{v})$,

- using the regularizer

$$\mathcal{R}\{\mathbf{v}\} := \left\|\mathbf{v} - \widehat{\mathbf{w}}^{(\text{global})}\right\|_2^2, \tag{121}$$

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

Note that Algorithms 10 and 9 provide only an abstract description of a practical FL system. The details of their actual implementation, such as the synchronization 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 [18, 66].

5.6 FedRelax

We now apply a simple block-coordinate minimization method [17] to solve GTVMin (49). To this end, we rewrite (49) as

$$\begin{aligned}\widehat{\mathbf{w}} \in \arg\min_{\mathbf{w}\in\mathbb{R}^{dn}} & \underbrace{\sum_{i\in\mathcal{V}} f^{(i)}(\mathbf{w})}_{=:f^{(\mathrm{GTV})}(\mathbf{w})} \\ \text{with } f^{(i)}(\mathbf{w}) &:= L_i\left(\mathbf{w}^{(i)}\right) + (\alpha/2)\sum_{i'\in\mathcal{N}^{(i)}} A_{i,i'}\left\|\mathbf{w}^{(i)} - \mathbf{w}^{(i')}\right\|_2^2, \\ \text{and the stacked model parameters } \mathbf{w} &= \left(\mathbf{w}^{(1)},\ldots,\mathbf{w}^{(n)}\right)^T. \end{aligned} \tag{122}$$

According to (122), the objective function of (49) 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 (122) 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 $\mathbf{w}^{(i,t)}$, for $i = 1, \ldots, n$, at time t. We then update (in parallel) each $\mathbf{w}^{(i,t)}$ by minimizing $f^{(\mathrm{GTV})}(\cdot)$ along $\mathbf{w}^{(i)}$ with the other local model parameters $\mathbf{w}^{(i')} := \mathbf{w}^{(i',t)}$ held fixed for all $i' \neq i$,

$$\begin{aligned}
\mathbf{w}^{(i,t+1)} &\in \operatorname*{arg\,min}_{\mathbf{w}^{(i)} \in \mathbb{R}^d} f^{(\mathrm{GTV})}\Big(\mathbf{w}^{(1,t)}, \ldots, \mathbf{w}^{(i-1,t)}, \mathbf{w}^{(i)}, \mathbf{w}^{(i+1,t)}, \ldots\Big) \\
&\overset{(122)}{=} \operatorname*{arg\,min}_{\mathbf{w}^{(i)} \in \mathbb{R}^d} f^{(i)}\left(\mathbf{w}^{(1,t)}, \ldots, \mathbf{w}^{(i-1,t)}, \mathbf{w}^{(i)}, \mathbf{w}^{(i+1,t)}, \ldots\right) \\
&\overset{(122)}{=} \operatorname*{arg\,min}_{\mathbf{w}^{(i)} \in \mathbb{R}^d} L_i\left(\mathbf{w}^{(i)}\right) + \alpha \sum_{i' \in \mathcal{N}^{(i)}} A_{i,i'} \left\|\mathbf{w}^{(i)} - \mathbf{w}^{(i',t)}\right\|_2^2. \qquad (123)
\end{aligned}$$

The update rule in (123) can be viewed as a non-linear Jacobi method applied to (122) [17, Sec. 3.2.4]. It also admits an interpretation as a form of block-coordinate optimization [67]. By iterating this update sufficiently many times, we arrive at Algorithm 11. There is an interesting connection between the

Algorithm 11 FedRelax for Parametric Models

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

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

1: **while** stopping criterion is not satisfied **do**
2: **for** all nodes $i \in \mathcal{V}$ in parallel **do**
3: compute $\mathbf{w}^{(i,t+1)}$ via (123)
4: share $\mathbf{w}^{(i,t+1)}$ with neighbors $\mathcal{N}^{(i)}$
5: **end for**
6: $t := t+1$
7: **end while**

update (123) and the basic gradient steps used by FedGD and FedSGD (see

Algorithm 5 and 7). Indeed, we obtain step 4 in Algorithm 5 from (123) by replacing the loss function $L_i\left(\mathbf{w}^{(i)}\right)$ with the approximation

$$L_i\left(\mathbf{w}^{(i,t)}\right) + \left(\nabla L_i\left(\mathbf{w}^{(i,t)}\right)\right)\left(\mathbf{w}^{(i)} - \mathbf{w}^{(i,t)}\right) + (1/(2\eta))\left\|\mathbf{w}^{(i)} - \mathbf{w}^{(i,t)}\right\|_2^2.$$

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). We can generalize Algorithm 11 to non-parametric local models by applying the non-linear Jacobi method to the GTVMin variant (66). This results in the update

$$\widehat{h}_{t+1}^{(i)} \in \operatorname*{arg\,min}_{h^{(i)} \in \mathcal{H}^{(i)}} L_i\left(h^{(i)}\right) + \alpha \sum_{i' \in \mathcal{N}^{(i)}} A_{i,i'} \underbrace{d^{(h^{(i)}, \widehat{h}_t^{(i')})}}_{\text{see (64)}}. \tag{124}$$

We obtain Algorithm 12 as a model-agnostic variant of Algorithm 11 by replacing the update (123) in its step 3 with the update (124).

Algorithm 12 is model-agnostic as it allows devices of an FL network to train different types of local models. The only restriction for the local models is that the update (124) can be computed efficiently. For some choices of local models and loss function, the update (124) 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 (64).

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

1: **while** stopping criterion is not satisfied **do**
2: **for** all nodes $i \in \mathcal{V}$ in parallel **do**
3: compute $\widehat{h}_{t+1}^{(i)}$ via (124)
4: **end for**
5: $t:=t+1$
6: **end while**

5.7 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:

$$\widehat{h}_{t+1}^{(i)} = \mathcal{F}^{(i)}\big(\widehat{h}_t^{(1)}, \ldots, \widehat{h}_t^{(n)}\big), \text{ for } i = 1, \ldots, n. \tag{125}$$

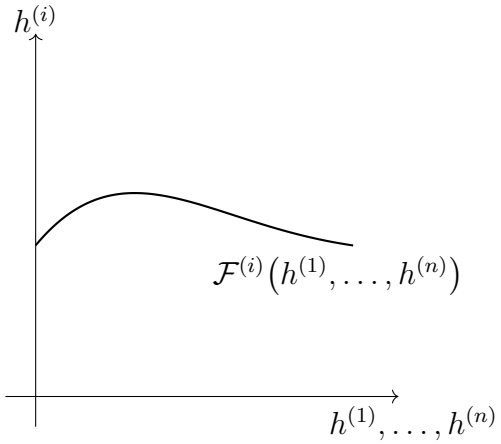


Fig. 5.4. A key computational step in many FL algorithms is the evaluation of an operator $\mathcal{F}^{(i)}$ at each node $i = 1, \ldots, n$ of the FL network.

Each operator $\mathcal{F}^{(i)} : \mathcal{H}^{(1)} \times \ldots \times \mathcal{H}^{(n)} \to \mathcal{H}^{(i)}$ represents a local update rule at the $i = 1, \ldots, n$ (see Figure 5.4). Some algorithms use time-varying update rules,

$$\widehat{h}_{t+1}^{(i)} = \mathcal{F}^{(i)}\big(\widehat{h}_t^{(1)}, \ldots, \widehat{h}_t^{(n)}\big). \tag{126}$$

with operators $\mathcal{F}^{(i,t)}$ that can vary across nodes $i = 1, \ldots, n$ and time instants $t = 1, 2, \ldots$. One example of (126) is used in Algorithm 5 for a time-varying learning rate.

Clearly, any FL algorithm of the form 125 is fully specified by the 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 $i = 1, \ldots, n$. In particular, the robustness of FL algorithms crucially depends on the shape of $\mathcal{F}^{(i)}$.

For parametric local models, we can re-formulate the fixed-point iteration (125) directly in terms of the model parameters

$$\mathbf{w}^{(i,t+1)} = \mathcal{F}^{(i)}\big(\mathbf{w}^{(1,t)}, \ldots, \mathbf{w}^{(n,t)}\big), \text{ for } t = 0, 1, \ldots, \tag{127}$$

with operators $\mathcal{F}^{(i)} : \mathbb{R}^{nd} \to \mathbb{R}^{d}$, for $i = 1, \ldots, n$. One example of (127) is the update 123 used by FedRelax (see Algorithm 11).

5.8 Asynchronous FL Algorithms

The FL algorithms presented so far rely on synchronous coordination among devices $i = 1, \ldots, n$ within an FL network [18, Ch. 6]. A new iteration is only initiated once all devices have completed their local updates (125) and communicated them to their neighbors [68, Sec. 10], [17, Sec. 1.4].

The implementation of synchronous FL algorithms can be difficult (or impossible) in practice. As highlighted in Chapter 8, trustworthy FL systems should tolerate unreliable or failing devices. Synchronous methods lack this robustness—any device failure or dropout can cause the entire algorithm execution to stall. Moreover, synchronous execution is inefficient in heterogeneous FL systems. Devices often vary in computational power or communication bandwidth, leading to the *straggler problem*: faster devices are forced to wait idly for slower ones [69, 70]. Having devices to wait idly for slower devices results in a waste of their computational resources.

To address the limitations of synchronous FL algorithms, we now show how to build asynchronous variants of the FL algorithms discussed in Section 5.7. We focus here on parametric local models, each represented by their own model parameters $\mathbf{w}^{(i)}$. The basic idea is to let each device $i = 1, \ldots, n$ execute the update (127) independently, using potentially out-dated updates from its neighbors $\mathcal{N}^{(i)}$.

An asynchronous FL algorithm consists of a sequence of update events, which we index by $t = 0, 1, 2, \ldots$ (see Figure 5.5). During each event t, a subset $\mathcal{A}^{(t)} \subseteq \mathcal{V}$ of devices performs updates:

$$\mathbf{w}^{(i,t+1)} = \mathcal{F}^{(i)}\big(\mathbf{w}^{(1,t_{i,1})}, \ldots, \mathbf{w}^{(n,t_{i,n})}\big). \tag{128}$$

Here, $t_{i,i'} \leq t$ is event index of the latest available model parameters of device i' at device i.

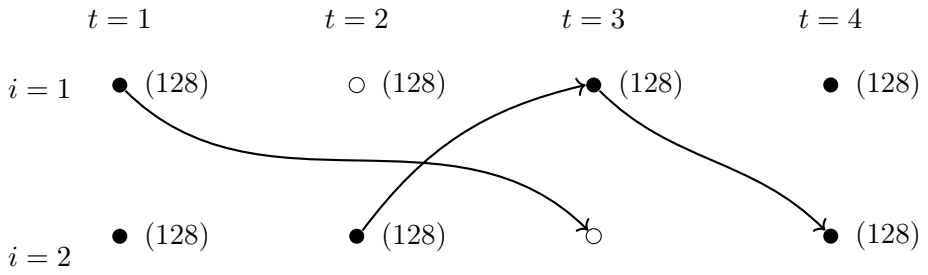


Fig. 5.5. The execution of an asynchronous FL algorithm consists of a sequence of update events, indexed by $t = 0, 1, 2, \ldots$. During each event t, the active nodes $i \in \mathcal{A}^{(t)} \subseteq \mathcal{V}$ of an FL network update their local model parameters $\mathbf{w}^{(i)}$ by computing (128). Active nodes are depicted as filled circles.

The set of nodes performing the update (128) during event t is denoted as the active set $\mathcal{A}^{(t)} \subseteq \mathcal{V}$. It is convenient to summarize the resulting asynchronous algorithm as

$$\mathbf{w}^{(i,t+1)} = \begin{cases} \mathcal{F}^{(i)}\big(\mathbf{w}^{(1,t_{i,1})}, \ldots, \mathbf{w}^{(n,t_{i,n})}\big) & \text{for } t \in T^{(i)} \\ \mathbf{w}^{(i,t)} & \text{otherwise.} \end{cases} \tag{129}$$

Here, we used the set

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

which consists, for each $i = 1, \ldots, n$, of those clock ticks during which node i is active. Note that (129) reduces to the synchronous algorithm (127) for the extreme case when $T^{(i)} = 0, 1, 2, \ldots$, for all $i = 1, \ldots, n$.

Like the synchronous algorithm (127), also the asynchronous variant 129 uses an iteration counter t. However, the practical meaning of t in the

asynchronous variant is fundamentally different: Instead of representing a global clock tick (or wall-clock time), the counter t in (129) indexes some update event during which at least one node is active and computes a local update. We denote the set of active nodes (or devices) during event t by $\mathcal{A}^{(t)} \subseteq \mathcal{V}$. The inactive nodes $i \notin \mathcal{A}^{(t)}$ leave their current model parameters unchanged, i.e., $\mathbf{w}^{(i,t+1)} = \mathbf{w}^{(i,t)}$.

For each active node $i \in \mathcal{A}^{(t)}$, the local update (129) uses potentially outdated model parameters $\mathbf{w}^{(i',t_{i,i'})}$ from its neighbors $i' \in \mathcal{N}^{(i)}$. Indeed, some of the neighbors might have not been in the active sets $\mathcal{A}^{(t-1)}, \mathcal{A}^{(t-2)}, \ldots$ of the most recent iterations. In this case, the update (129) does not have access to $\mathbf{w}^{(i',t)}$. Instead, we can only use $\mathbf{w}^{(i',t_{i,i'})}$ that has been produced obtained during some previous iteration $t_{i,i'} < t$.

The update (128) involves an operator $\mathcal{F}^{(i)} : \mathbb{R}^{dn} \to \mathbb{R}^d$ that determines the resulting FL algorithm. We can interpret (128) as an asynchronous variant of the synchronous algorithm (127) obtained for the same $\mathcal{F}^{(i)}$. For example, an asynchronous variant of Algorithm 5 (with a fixed learning rate) can be obtained for the choice

$$\mathcal{F}^{(i)}\big(\mathbf{w}^{(1)}, \ldots, \mathbf{w}^{(n)}\big) = \mathbf{w}^{(i)} - \eta\Bigg(\nabla L_i\left(\mathbf{w}^{(i)}\right) + \sum_{i' \in \mathcal{N}^{(i)}} 2A_{i,i'}\big(\mathbf{w}^{(i)} - \mathbf{w}^{(i')}\big)\Bigg). \quad (130)$$

Note that the choice (130) involves the local loss functions and the weighted edges of an FL network.

The update (128), at an active node $i \in \mathcal{A}^{(t)}$, involves potentially outdated local model parameters $\mathbf{w}^{(i',t_{i,i'})}$, with $t_{i,i'} \leq t$, for $i' = 1, \ldots, n$. The quantity $t_{i,i'}$ represents the most recent update event during which node i' has shared its updated local model parameters with node i. We can, in turn,

interpret the difference $t - t_{i,i'}$ as a measure of the communication delay between node i' and node i.

Depending on the extent of the delays $t - t_{i,i'}$ in the update (128), we distinguish between [17]

- **Totally asynchronous algorithms.** These are algorithms of the form (129) with unbounded delays $t–t_{i,i'}$, i.e., they can can become arbitrarily large. Moreover, we require that no device stops updating, i.e., the set $T^{(i)}$ is infinite for each $i = 1, \ldots, n$.

- **Partially asynchronous algorithms.** These are algorithms of the form (129) with bounded delays $t–t_{i,i'} \leq B$, with some fixed (but possibly unknown) maximum delay $B \in \mathbb{N}$. Moreover, each device updates at least once during B consecutive clock ticks, i.e., $T^{(i)} \cap \{t, t+1, t+B-1\} \neq \emptyset$ for each $t = 1, 2, \ldots,$ and $i = 1, \ldots, n$.

For some choices of $\mathcal{F}^{(i)}$ in (128), 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 [17, Ch. 7].

Convergence Guarantees. There is an elegant characterization of the convergence of totally and partially asynchronous FL algorithms of the form (129). This characterization applies whenever the operators $\mathcal{F}^{(i)}$, for $i = 1, \ldots, n$, in (129) form a pseudo-contraction [71]

$$\max_{i=1,\ldots,n} \left\| \mathcal{F}^{(i)}\big(\mathbf{w}^{(1)}, \ldots, \mathbf{w}^{(n)}\big) - \mathcal{F}^{(i)}\big(\widehat{\mathbf{w}}^{(1)}, \ldots, \widehat{\mathbf{w}}^{(n)}\big) \right\| \leq \kappa \cdot \max_{i=1,\ldots,n} \left\| \mathbf{w}^{(i)} - \widehat{\mathbf{w}}^{(i)} \right\|, \tag{131}$$

with some contraction rate $\kappa \in [0, 1)$ and some fixed-point $\widehat{\mathbf{w}}^{(1)}, \ldots, \widehat{\mathbf{w}}^{(n)}$.

The operators $\mathcal{F}^{(i)}$, for $i = 1, \ldots, n$, underlying GTVMin-based algorithms are determined by the design choices of the GTVMin building blocks. These include the choices of local loss functions $L_i(\cdot)$, for $i = 1, \ldots, n$ and edge weights $A_{i,i'}$, for $\{i, i'\} \in \mathcal{E}$. Let us next discuss specific design choices which yield operators that form a pseudo-contraction (131).

Consider the operator $\mathcal{F}^{(i)}$ defined by the update (123) of FedRelax (see Algorithm 11). If the local loss functions $L_i(\cdot)$ are strongly convex,[20] we can decompose $\mathcal{F}^{(i)}$ as

$$\mathcal{F}^{(i)} = \mathbf{prox}_{L_i(\cdot), 2\alpha d^{(i)}}(\cdot) \circ \mathcal{T}^{(i)}. \tag{132}$$

Here, we used the proximal operator as defined in (57) as well as the averaging-neighbors-operator

$$\mathcal{T}^{(i)} : \underbrace{\mathbb{R}^d \times \ldots \times \mathbb{R}^d}_{n \text{ times}} \to \mathbb{R}^d : \mathbf{w}^{(1)}, \ldots, \mathbf{w}^{(n)} \mapsto (1/d^{(i)}) \sum_{i' \in \mathcal{N}^{(i)}} A_{i,i'} \mathbf{w}^{(i')}.$$

It can be easily verified that the operators $\mathcal{T}^{(1)}, \ldots, \mathcal{T}^{(n)}$ are non-expansive. Moreover, by the basic properties of proximal operators (see, e.g., [72, Sec. 6]), the operators

$$\mathbf{prox}_{L_1(\cdot), 2\alpha d^{(1)}}(\cdot), \ldots, \mathbf{prox}_{L_n(\cdot), 2\alpha d^{(n)}}(\cdot)$$

also form a pseudo-contraction with $\kappa = \frac{1}{1+(\sigma/(2\alpha d^{(i)}))}$. Combining these facts with (132) yields that the operators $\mathcal{F}^{(i)}$, for $i = 1, \ldots, n$, form a pseudo-contraction with

$$\kappa = \frac{1}{1 + (\sigma/(2\alpha d^{(i)}))}. \tag{133}$$

[20] Strictly speaking, we also need to require that the epigraph of each $L_i(\cdot)$, for $i = 1, \ldots, n$ is non-empty and closed [39].

For any FL algorithm (129) such that (131) is satisfied, the following holds:

- A totally asynchronous algorithm of the form (129) converges to $\widehat{\mathbf{w}}^{(1)}, \ldots, \widehat{\mathbf{w}}^{(n)}$ [71, Thm. 23].
- In the partially asynchronous case with maximum delay B [71, Thm. 24],

$$\max_{i=1,\ldots,n} \left\| \mathbf{w}^{(i,t)} - \widehat{\mathbf{w}}^{(i)} \right\| \leq \kappa^{t/(2B+1)} \cdot \max_{i=1,\ldots,n} \left\| \mathbf{w}^{(i,0)} - \widehat{\mathbf{w}}^{(i)} \right\|. \tag{134}$$

The bound (134) is quite intuitive: smaller contraction factors κ and smaller delay bounds B lead to faster convergence of the algorithm (129). Figure 5.6 illustrates the factor $\kappa^{t/(2B+1)}$ for different values of κ and maximum delay B.

The contraction factor κ of the operators $\mathcal{F}^{(i)}$, for $i = 1, \ldots, n$, arising in GTVMin-based methods depends on the properties of local loss functions and the connectivity of the FL network. According to (133), the operators underlying FedRelax (see (123) and Algorithm 11), have a small contraction factor if we use

- local loss functions that are strongly convex with large coefficient σ,
- a FL network with small weighted node degrees $d^{(i)}$, for $i = 1, \ldots, n$.

Moreover, the contraction factor (133) decreases with decreasing GTVMin parameter α. In the extreme case of $\alpha = 0$ - where GTVMin decomposes into fully independent local instances of ERM $\min_{\mathbf{w}^{(i)}} L_i(\cdot)$ - the contraction factor becomes $\kappa = 0$. This makes sense as in this extreme case, there is no information sharing required among the nodes of an FL network. Clearly, the delays $t - t_{i,i'}$ are then irrelevant for the performance of FL algorithms.

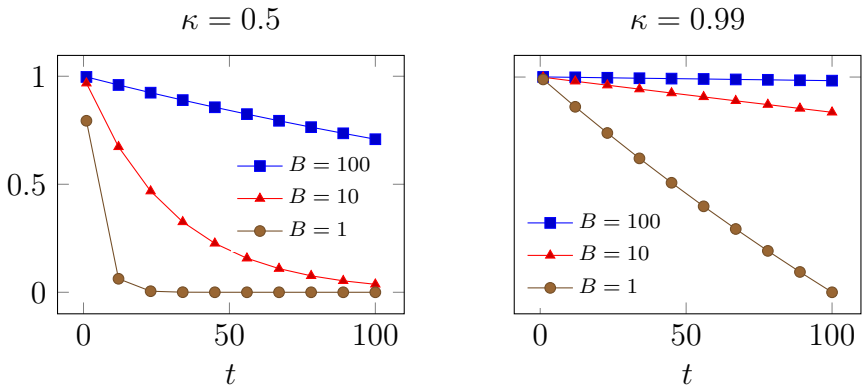


Fig. 5.6. Illustration of the factor $\kappa^{t/(2B+1)}$ in the convergence bound (134) for a partially asynchronous FL algorithm (129) using a pseudo-contraction (see (131)).

5.9 Exercises

5.1. The convergence speed of gradient-based methods. Study the convergence speed of (104) 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 (104).

5.2. Convergence speed for homogeneous data. Study the convergence speed of (104) when applied to GTVMin (102) 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 (104), starting with the initializations $w^{(i,0)} := y^{(i)}$. Study the dependence of the convergence speed of (104) (towards a solution of (102)) on the value of α in (102).

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 learned hypothesis maps at the nodes of an edge $\{i, i'\}$, we use (64) with the squared error loss. Show that the update (124) 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 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 $\eta_{i,t} := \eta_i$. 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

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

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

5.6. FedRelax update. Show that the update (123) of FedRelax for parametric local models can be rewritten as

$$\operatorname*{arg\,min}_{\mathbf{w}^{(i)} \in \mathbb{R}^d} L_i\left(\mathbf{w}^{(i)}\right) + \alpha d^{(i)} \Big\| \mathbf{w}^{(i)} - \widehat{\mathbf{w}}^{(\mathcal{N}^{(i)})} \Big\|_2^2.$$

Here, we used $\widehat{\mathbf{w}}^{(\mathcal{N}^{(i)})} := (1/d^{(i)}) \sum_{i' \in \mathcal{N}^{(i)}} A_{i,i'} \mathbf{w}^{(i',t)}$ and the weighted node degree $d^{(i)} = \sum_{i' \in \mathcal{N}^{(i)}} A_{i,i'}$ (see (34)).

5.7. FedRelax vs. FedSGD Show that the update in step (4) of Algorithm 5 is obtained from the update (123) of FedRelax by replacing the local loss function $L_i\left(\mathbf{w}^{(i)}\right)$ with a local approximation by a quadratic function, centred around $\mathbf{w}^{(i,t)}$.

5.8. FedSGD as fixed-point iteration. Show that Algorithm 6 can be written as the distributed fixed-point iteration (127). Try to find an elegant characterization of the resulting operators $\mathcal{F}^{(i)}$, for $i = 1, \ldots, n$.

5.9. FedRelax as fixed-point iteration. Show that Algorithm 12 can be written as the distributed fixed-point iteration (125). Try to find an elegant characterization of the resulting operators $\mathcal{F}^{(i)}$, for $i = 1, \ldots, n$. .

5.10 Proofs

5.10.1 Proof of Proposition 5.1

The first inequality in (106) follows from well-known results on the eigenvalues of a sum of symmetric matrices (see, e.g., [3, Thm 8.1.5]). In particular,

$$\lambda_{\max}\big(\mathbf{Q}\big) \leq \max\big\{ \underbrace{\max_{i=1,\ldots,n} \lambda_d\big(\mathbf{Q}^{(i)}\big)}_{\stackrel{(105)}{=}\lambda_{\max}}, \lambda_{\max}\big(\alpha \mathbf{L}^{(\mathcal{G})} \otimes \mathbf{I}\big)\big\}. \tag{135}$$

The second inequality in (106) uses the following upper bound on the maximum eigenvalue $\lambda_n\big(\mathbf{L}^{(\mathcal{G})}\big)$ of the Laplacian matrix:

$$\begin{aligned}
\lambda_n\big(\mathbf{L}^{(\mathcal{G})}\big) &\stackrel{(a)}{=} \max_{\mathbf{v}\in\mathbb{S}^{(n-1)}} \mathbf{v}^T \mathbf{L}^{(\mathcal{G})} \mathbf{v} \\
&\stackrel{(37)}{=} \max_{\mathbf{v}\in\mathbb{S}^{(n-1)}} \sum_{\{i,i'\}\in\mathcal{E}} A_{i,i'}\big(v_i - v_{i'}\big)^2 \\
&\stackrel{(b)}{\leq} \max_{\mathbf{v}\in\mathbb{S}^{(n-1)}} \sum_{\{i,i'\}\in\mathcal{E}} 2A_{i,i'}\big(v_i^2 + v_{i'}^2\big) \\
&\stackrel{(c)}{=} \max_{\mathbf{v}\in\mathbb{S}^{(n-1)}} \sum_{i\in\mathcal{V}} 2v_i^2 \sum_{i'\in\mathcal{N}^{(i)}} A_{i,i'} \\
&\stackrel{(35)}{\leq} \max_{\mathbf{v}\in\mathbb{S}^{(n-1)}} \sum_{i\in\mathcal{V}} 2v_i^2 d^{(\mathcal{G})}_{\max} \\
&= 2d^{(\mathcal{G})}_{\max}.
\end{aligned} \tag{136}$$

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 5.7). The bound (136) is essentially tight.[21]

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

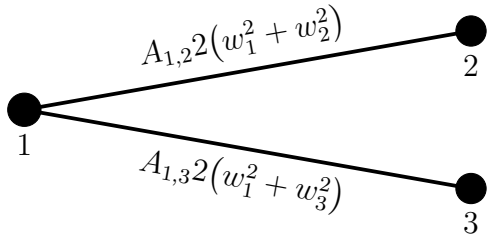


Fig. 5.7. Illustration of step (c) in (136).

5.10.2 Proof of Proposition 5.2

Similar to the upper bound (136) we also start with the CFW for the eigenvalues of $\mathbf{Q}$ in (103). In particular,

$$\lambda_1 = \min_{\|\mathbf{w}\|_2^2=1} \mathbf{w}^T\mathbf{Q}\mathbf{w}. \tag{137}$$

We next analyze the right-hand side of (137) by partitioning the constraint set $\{\mathbf{w} : \|\mathbf{w}\|_2^2 = 1\}$ of (137) 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

$$\mathbf{w} = \underbrace{\mathbf{P}_{\mathcal{S}}\mathbf{w}}_{=:\overline{\mathbf{w}}} + \underbrace{\mathbf{P}_{\mathcal{S}^\perp}\mathbf{w}}_{=:\widetilde{\mathbf{w}}} \text{ for subspace } \mathcal{S} \text{ in (45).} \tag{138}$$

Explicit expressions for the orthogonal components $\overline{\mathbf{w}}$, $\widetilde{\mathbf{w}}$ are given by (46) and (47). In particular, the component $\overline{\mathbf{w}}$ satisfies

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

Note that

$$\|\mathbf{w}\|_2^2 = \|\overline{\mathbf{w}}\|_2^2 + \|\widetilde{\mathbf{w}}\|_2^2. \tag{139}$$

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 (139), we have

$$\|\widetilde{\mathbf{w}}\|_2^2 \geq \rho^2/(1+\rho^2). \tag{140}$$

This implies, in turn, via (44) that

$$\begin{aligned}\mathbf{w}^T\mathbf{Q}\mathbf{w} &\overset{(103)}{\geq} \alpha\mathbf{w}^T\big(\mathbf{L}^{(\mathcal{G})}\otimes\mathbf{I}\big)\mathbf{w}\\ &\overset{(37),(44)}{\geq} \alpha\lambda_2\big(\mathbf{L}^{(\mathcal{G})}\big)\|\widetilde{\mathbf{w}}\|_2^2\\ &\overset{(140)}{\geq} \alpha\lambda_2\big(\mathbf{L}^{(\mathcal{G})}\big)\rho^2/(1+\rho^2).\end{aligned} \tag{141}$$

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,

$$n\|\mathbf{c}\|_2^2 = \|\overline{\mathbf{w}}\|_2^2 > 1/(1+\rho^2). \tag{142}$$

We next develop the right-hand side of (137) according to

$$\begin{aligned}\mathbf{w}^T\mathbf{Q}\mathbf{w} &\overset{(103)}{\geq} \sum_{i=1}^n \big(\mathbf{w}^{(i)}\big)^T\mathbf{Q}^{(i)}\mathbf{w}^{(i)}\\ &\overset{(138)}{\geq} \sum_{i=1}^n \big(\mathbf{c}+\widetilde{\mathbf{w}}^{(i)}\big)^T\mathbf{Q}^{(i)}\big(\mathbf{c}+\widetilde{\mathbf{w}}^{(i)}\big)\\ &\overset{(142)}{\geq} \|\overline{\mathbf{w}}\|_2^2\underbrace{\lambda_1\bigg((1/n)\sum_{i=1}^n\mathbf{Q}^{(i)}\bigg)}_{\bar{\lambda}_{\min}} + \sum_{i=1}^n\Big[2\big(\widetilde{\mathbf{w}}^{(i)}\big)^T\mathbf{Q}^{(i)}\mathbf{c} + \underbrace{\big(\widetilde{\mathbf{w}}^{(i)}\big)^T\mathbf{Q}^{(i)}\widetilde{\mathbf{w}}^{(i)}}_{\geq 0}\Big]\\ &\geq \|\overline{\mathbf{w}}\|_2^2\bar{\lambda}_{\min} + \sum_{i=1}^n 2\big(\widetilde{\mathbf{w}}^{(i)}\big)^T\mathbf{Q}^{(i)}\mathbf{c}.\end{aligned} \tag{143}$$

To develop (143) further, we note that

$$\begin{aligned}\bigg|\sum_{i=1}^n 2\big(\widetilde{\mathbf{w}}^{(i)}\big)^T\mathbf{Q}^{(i)}\mathbf{c}\bigg| &\overset{(a)}{\leq} 2\lambda_{\max}\|\widetilde{\mathbf{w}}\|_2\|\overline{\mathbf{w}}\|_2\\ &\overset{\|\widetilde{\mathbf{w}}\|_2<\rho\|\overline{\mathbf{w}}\|_2}{\leq} 2\lambda_{\max}\rho\|\overline{\mathbf{w}}\|_2^2.\end{aligned} \tag{144}$$

Here, step (a) follows from $\max_{\|\mathbf{y}\|_2=1,\|\mathbf{x}\|_2=1} \mathbf{y}^T\mathbf{Q}\mathbf{x} = \lambda_{\max}$. Inserting (144) into (143) for $\rho = \bar{\lambda}_{\min}/(4\lambda_{\max})$,

$$\mathbf{w}^T\mathbf{Q}\mathbf{w} \geq \|\overline{\mathbf{w}}\|_2^2\bar{\lambda}_{\min}/2 \overset{(142)}{\geq} (1/(1+\rho^2))\bar{\lambda}_{\min}/2 \tag{145}$$

For each $\mathbf{w}$ with $\|\mathbf{w}\|_2^2 = 1$, either (141) or (145) must hold.

6 Key Variants of Federated Learning

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 flavors") 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 flavors of FL discussed in the following sections.

Section 6.1 discusses single-model FL that learns model parameters of a single (global) model from local datasets. This single-model flavor 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 6.2 discusses how CFL is obtained from GTVMin over FL networks with a cluster 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 6.3 discusses horizontal federated learning (HFL) which is obtained from GTVMin over an FL network whose nodes carry different subsets of a single underlying global dataset. Loosely speaking, HFL involves local datasets characterized by the same set of features but obtained from different

data points from an underlying dataset.

Section 6.4 discusses vertical federated learning (VFL), which arises from applying GTVMin to a FL network where each node holds data on the same individuals but with different sets of features. A representative example involves public institutions such as tax authorities, social insurance agencies, and healthcare providers. While these organizations each collect distinct types of information, they all refer to the same underlying population, e.g., individuals identified by a Finnish social security number.

Section 6.5 shows how personalized FL can be obtained from GTVMin by using specific measures for the GTV of local model parameters. For example, using deep ANNs as local models, we might only use the model parameters corresponding to the first few input layers to define the GTV.

6.1 Single-Model FL

Some FL use cases require to train a single (global) model $\mathcal{H}$ from a decentralized collection of local datasets $\mathcal{D}^{(i)}$, $i = 1, \ldots, n$ [13, 73]. 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 $\mathbf{w}^{(t)}$, $t = 1, \ldots$.

After computing the new model parameters $\mathbf{w}^{(t+1)}$, the server broadcasts it to the devices $i = 1, \ldots, i$ and increments the clock $t := t + 1$. In the next iteration, each device i uses the current global model parameters $\mathbf{w}^{(t)}$ to compute a local update $\mathbf{w}^{(i,t)}$ 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 (119)).

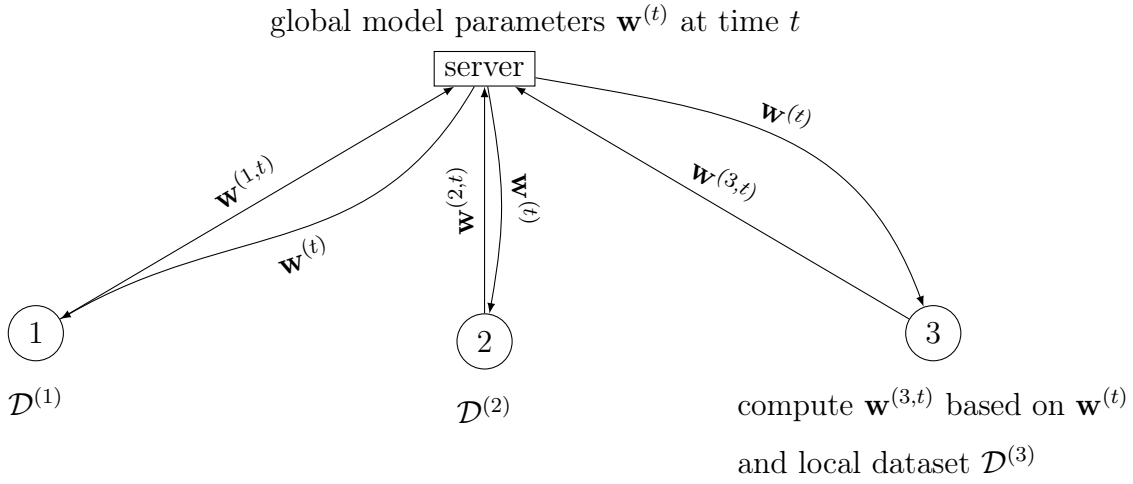


Fig. 6.1. Illustration of a server-based (centralized) FL system during iteration t. The server begins by broadcasting the current global model parameters $\mathbf{w}^{(t)}$ to each device $i \in \mathcal{V}$. Each device i then computes an update $\mathbf{w}^{(i,t)}$ based on its local dataset $\mathcal{D}^{(i)}$ and the received model parameters $\mathbf{w}^{(t)}$. These local updates $\mathbf{w}^{(i,t)}$ are sent back to the server, which aggregates them to obtain the updated global model parameters $\mathbf{w}^{(t+1)}$.

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 [64]. 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 [74].

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

6.2 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 [16, 34].

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 learned 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 can apply standard clustering techniques (such

as k-means or Gaussian mixture model (GMM)) to a vector representation of the local datasets [23, Ch. 5]. These vector representations can be constructed in various ways. One option is to use the model parameters $\widehat{\mathbf{w}}$ of a parametric ML model trained on $\mathcal{D}^{(i)}$. Alternatively, we can represent each local dataset $\mathcal{D}^{(i)}$ using the gradient of its local loss function $L_i\left(\mathbf{w}^{(i)}\right)$ (see Section 7.2).

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 6.2, which contains a cluster $\mathcal{C} = \{1, 2, 3\}$.

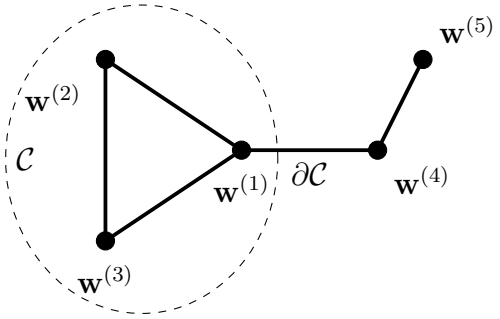


Fig. 6.2. The solution of GTVMin (49) are local model parameters that are approximately identical for all nodes in a tight-knit cluster $\mathcal{C}$.

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 $\mathcal{G}^{(\mathcal{C})}$:[22]

The larger $\lambda_2\big(\mathbf{L}^{(\mathcal{C})}\big)$, the better the connectivity among the nodes in $\mathcal{C}$.

[22]The graph $\mathcal{G}^{(\mathcal{C})}$ consists of the nodes in $\mathcal{C}$ and the edges $\{i, i'\} \in \mathcal{E}$ for $i, i' \in \mathcal{C}$.

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

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

Note that for a single-node cluster $\mathcal{C}=\{i\}$, the cluster boundary coincides with the node degree, $|\partial\mathcal{C}| = d^{(i)}$ (see (34)).

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 (102) 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*

$$\mathbf{y}^{(i)} = \mathbf{X}^{(i)}\overline{\mathbf{w}}^{(\mathcal{C})} + \boldsymbol{\varepsilon}^{(i)}, \text{ for all } i\in\mathcal{C}. \tag{146}$$

We learn local model parameters $\widehat{\mathbf{w}}^{(i)}$ via solving GTVMin (102). If the cluster is connected, the error component

$$\widetilde{\mathbf{w}}^{(i)} := \widehat{\mathbf{w}}^{(i)} - (1/|\mathcal{C}|)\sum_{i\in\mathcal{C}}\widehat{\mathbf{w}}^{(i)} \tag{147}$$

is upper bounded as

$$\sum_{i\in\mathcal{C}}\big\|\widetilde{\mathbf{w}}^{(i)}\big\|_2^2 \leq \frac{1}{\alpha\lambda_2\big(\mathbf{L}^{(\mathcal{C})}\big)}\Bigg[\sum_{i\in\mathcal{C}}\frac{1}{m_i}\big\|\boldsymbol{\varepsilon}^{(i)}\big\|_2^2 + \alpha\,|\partial\mathcal{C}|\,2\Big(\big\|\overline{\mathbf{w}}^{(\mathcal{C})}\big\|_2^2 + R^2\Big)\Bigg]. \tag{148}$$

Here, we used $R := \max_{i'\in\mathcal{V}\setminus\mathcal{C}}\big\|\widehat{\mathbf{w}}^{(i')}\big\|_2$.

Proof. See Section 6.8.1. □

The bound (148) 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 $\mathcal{C}$ typically results in a decreased eigenvalue $\lambda_2\big(\mathbf{L}^{(\mathcal{C})}\big)$.[23] According to (148), we should then increase α to maintain a small deviation $\widetilde{\mathbf{w}}^{(i)}$ of the learned local model parameters from their cluster-wise average. Thus, increasing α in (49) 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 learned model parameters $\mathbf{w}^{(i)}$ that are approximately identical for all $\mathcal{V} = 1, \ldots, n$. The resulting approximation error is quantified by Proposition 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 (148) specializes to (63).

We hasten to add that the bound (148) only applies for local datasets that conform with the probabilistic model (146). 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 (148). 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, 75]. Finally, we might replace $\big\|\overline{\mathbf{w}}^{(\mathcal{C})}\big\|_2^2$ in (148) with an upper bound for this quantity.

[23] 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 6.2). 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 $\mathcal{C}' := \mathcal{C} \cup \{i'\}$.

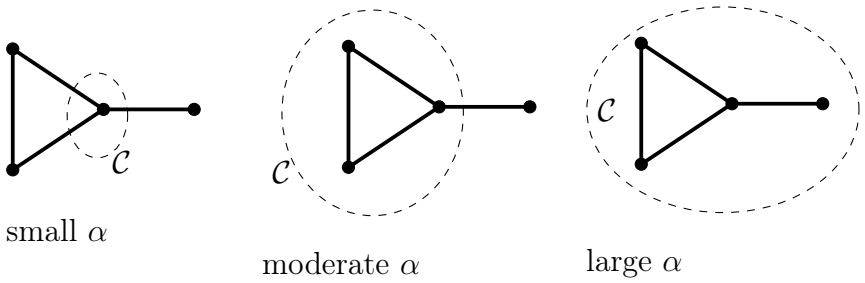


Fig. 6.3. As the regularization parameter α increases, the solutions of the GTVMin (49) become approximately constant over larger subsets of nodes, i.e., they exhibit stronger clustering.

6.3 Horizontal FL

HFL uses local datasets $\mathcal{D}^{(i)}$, for $i \in \mathcal{V}$, that contain data points characterized by the same features [76]. 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 HFL as a generalization of semi-supervised learning (SSL) [77]: 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 $\mathbf{w}^{(i)}$

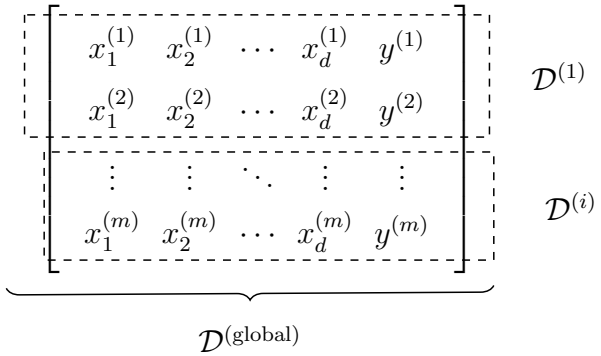


Fig. 6.4. HFL uses the same features to characterize data points in different local datasets. Different local datasets are constituted by different subsets of data points out of an underlying global dataset.

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 6.5).

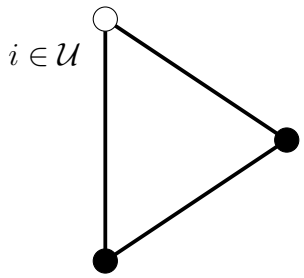


Fig. 6.5. HFL includes SSL as a special case. SSL involves a subset of nodes $\mathcal{U}$, for which the local datasets do not contain labels. We can take this into account by using the trivial loss function $L_i(\cdot) = 0$ for each node $i \in \mathcal{U}$. However, we can still use the features in $\mathcal{D}^{(i)}$ to construct an FL network $\mathcal{G}$.

6.4 Vertical FL

VFL 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 [78]. Formally, VFL 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 Fig. 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 $\mathcal{J}^{(i)} := \{j_1, \ldots, j_d\}$ of the original d' features (entries

of $\mathbf{x}^{(r)}$). At least one node i' must have a local dataset $\mathcal{D}^{(i')}$ 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 features that characterize the economic situation of the individual. Some individuals participate in retailer loyalty programs, which generate consumer behaviour features. Additionally, social network 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, VFL enables collaborative learning while preserving data privacy.

6.5 Personalized Federated Learning

Consider GTVMin (49) for learning local model parameters $\widehat{\mathbf{w}}^{(i)}$ for each local dataset $\mathcal{D}^{(i)}$. If the value of α in (49) 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 (49).

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.

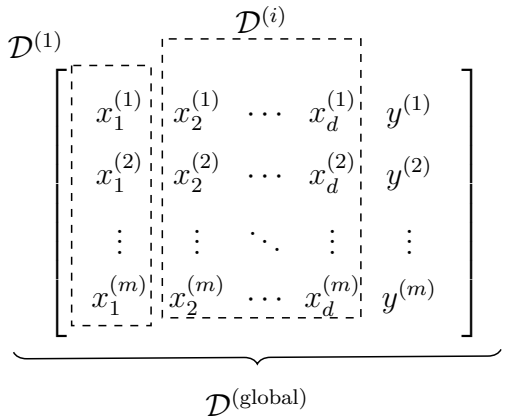


Fig. 6.6. VFL uses local datasets that are derived from the same data points. The local datasets differ in the choice of features used to characterize the common data points.

The partial parameter sharing for local models can be implemented in many different ways [79, 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,

$$\phi\big(\mathbf{w}^{(i)} - \mathbf{w}^{(i')}\big) := \alpha^{(1)}\phi^{(1)}\big(\mathbf{w}^{(i)} - \mathbf{w}^{(i')}\big) + \alpha^{(2)}\phi^{(2)}\big(\mathbf{w}^{(i)} - \mathbf{w}^{(i')}\big). \quad (149)$$

  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 (64) 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 (149) to enforce different subsets

of the model parameters to be clustered with different granularity, i.e., enforcing some of the model parameters to be constant across larger subsets of nodes.

- For local models being deep ANNs, we enforce identical model parameters in the layers closer to the input. In contrast, the layers closer to the output are allowed to have different parameters across devices. Figure 6.7 illustrates this concept 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 [80].

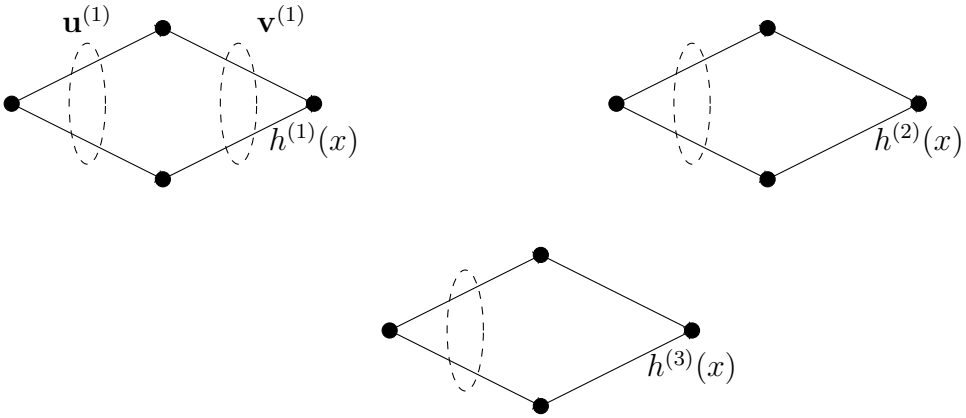


Fig. 6.7. Personalized FL with local models being ANNs with a single hidden layer. The ANN $h^{(i)}$ is parametrized by the vector $\mathbf{w}^{(i)} = \left(\left(\mathbf{u}^{(i)}\right)^T, \left(\mathbf{v}^{(i)}\right)^T \right)^T$, with parameters $\mathbf{u}^{(i)}$ of the hidden layer and the parameters $\mathbf{v}^{(i)}$ of the output layer. We couple the training of $\mathbf{u}^{(i)}$ via GTVMin using the discrepancy measure $\phi = \left\| \mathbf{u}^{(i)} - \mathbf{u}^{(i')} \right\|_2^2$.

6.6 Few-Shot Learning

Some 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 [81, 82]. 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 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.

6.7 Exercises

6.1. Horizontal FL of a Linear Model [68, 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 [68, 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 $j \in \mathcal{J}^{(i)}$. Construct an instance of GTVMin such that its solutions coincide (approximately) with the solution of plain vanilla linear regression.

6.8 Proofs

6.8.1 Proof of Proposition 6.1

To verify (148), we follow a similar argument as used in the proof of Proposition 3.1.

First, we decompose the objective function $f\big(\mathbf{w}\big)$ in (102) as follows:

$$f(\mathbf{w}) = \underbrace{\sum_{i\in\mathcal{C}}(1/m_i)\big\|\mathbf{y}^{(i)}-\mathbf{X}^{(i)}\mathbf{w}^{(i)}\big\|_2^2+\alpha\Big[\sum_{i,i'\in\mathcal{C}}A_{i,i'}\Big\|\mathbf{w}^{(i)}-\mathbf{w}^{(i')}\Big\|_2^2+\sum_{\{i,i'\}\in\partial\mathcal{C}}A_{i,i'}\Big\|\mathbf{w}^{(i)}-\mathbf{w}^{(i')}\Big\|_2^2\Big]}_{=:f'\left(\mathbf{w}\right)} + f''\big(\mathbf{w}\big). \tag{150}$$

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 (148) via a proof by contradiction: If (148) 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 (102).

First, note that

$$
\begin{aligned}
f'\big(\overline{\mathbf{w}}^{(i)}\big) &= \sum_{i\in\mathcal{C}}(1/m_i)\big\|\mathbf{y}^{(i)}-\mathbf{X}^{(i)}\overline{\mathbf{w}}^{(\mathcal{C})}\big\|_2^2 \\
&\qquad +\alpha\Bigg[\sum_{\substack{\{i,i'\}\in\mathcal{E}\\ i,i'\in\mathcal{C}}} A_{i,i'}\big\|\overline{\mathbf{w}}^{(\mathcal{C})}-\overline{\mathbf{w}}^{(\mathcal{C})}\big\|_2^2+\sum_{\substack{\{i,i'\}\in\mathcal{E}\\ i\in\mathcal{C},i'\notin\mathcal{C}}} A_{i,i'}\Big\|\overline{\mathbf{w}}^{(\mathcal{C})}-\widehat{\mathbf{w}}^{(i')}\Big\|_2^2\Bigg] \\
&\overset{(146)}{=} \sum_{i\in\mathcal{C}}(1/m_i)\big\|\boldsymbol{\varepsilon}^{(i)}\big\|_2^2+\alpha\sum_{\substack{\{i,i'\}\in\mathcal{E}\\ i\in\mathcal{C},i'\notin\mathcal{C}}} A_{i,i'}\Big\|\overline{\mathbf{w}}^{(\mathcal{C})}-\widehat{\mathbf{w}}^{(i')}\Big\|_2^2 \\
&\overset{(a)}{\leq} \sum_{i\in\mathcal{C}}(1/m_i)\big\|\boldsymbol{\varepsilon}^{(i)}\big\|_2^2+\alpha\sum_{\substack{\{i,i'\}\in\mathcal{E}\\ i\in\mathcal{C},i'\notin\mathcal{C}}} 2A_{i,i'}\left(\big\|\overline{\mathbf{w}}^{(\mathcal{C})}\big\|_2^2+\Big\|\widehat{\mathbf{w}}^{(i')}\Big\|_2^2\right) \\
&\leq \sum_{i\in\mathcal{C}}(1/m_i)\big\|\boldsymbol{\varepsilon}^{(i)}\big\|_2^2+\alpha\,|\partial\mathcal{C}|\,2\left(\big\|\overline{\mathbf{w}}^{(\mathcal{C})}\big\|_2^2+R^2\right). \qquad (151)
\end{aligned}
$$

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,

$$
\begin{aligned}
f'\big(\widehat{\mathbf{w}}^{(i)}\big) &\geq \alpha\sum_{i,i'\in\mathcal{C}} A_{i,i'}\underbrace{\Big\|\widehat{\mathbf{w}}^{(i)}-\widehat{\mathbf{w}}^{(i')}\Big\|_2^2}_{\overset{(147)}{=}\big\|\widetilde{\mathbf{w}}^{(i)}-\widetilde{\mathbf{w}}^{(i')}\big\|_2^2} \\
&\overset{(44)}{\geq} \alpha\lambda_2\big(\mathbf{L}^{(\mathcal{C})}\big)\sum_{i\in\mathcal{C}}\big\|\widetilde{\mathbf{w}}^{(i)}\big\|_2^2. \qquad (152)
\end{aligned}
$$

If the bound (148) would not hold, then by (152) and (151) 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 (102).

7 Graph Learning for FL Networks

Chapter 3 introduced GTVMin as a flexible design principle for FL algorithms. Chapter 5 explores how algorithms can be obtained by applying optimization methods - such as the gradient-based methods from Chapter 4 - to solve GTVMin instances.

The computational and statistical properties of such algorithms depend crucially on the structure of the underlying FL network. For example, both the computational and communication costs of FL systems typically increase with the number of edges in the FL network. Moreover, the graph topology governs how local datasets are pooled into clusters with shared model parameters.

In some settings, domain expertise can guide the construction of the FL network. For instance, in health-care, known clinical similarities between disease types are used to define edges connecting patients or diseases [83]. In sensor networks, physical proximity and hardware connectivity constraints naturally shape the graph structure [84, 85]. However, other applications lack strong prior structure and require to learn the graph from data [86–89]. This chapter presents techniques to infer FL networks from local datasets and associated local loss functions.

This chapter is organized as follows. Section 7.1 discusses how the analysis of FL algorithms can inform the design of the FL network. Section 7.2 presents methods to quantify discrepancies between local datasets. Section 7.3 formulates graph learning as an optimization problem that minimizes the discrepancy between datasets stored at nodes that are connected by an edge. The structure of the resulting graph can be influenced by imposing connectivity constraints, such as a minimum required node degree.

7.1 Edges as Design Choice

Consider the GTVMin instance (51), which aims to learn local model parameters for each linear model associated with a local dataset $\mathcal{D}^{(i)}$. To solve (51), we use Algorithm 4, which implements the gradient step (104) in a message-passing fashion.

The GTVMin formulation (51) is defined for a fixed FL network $\mathcal{G}$. Hence, the structure of $\mathcal{G}$ significantly impacts both the statistical and computational properties of Algorithm 4.

Statistical Properties. These can be assessed using a probabilistic model for the local datasets. An important example is the clustering assumption (146), discussed in the context of CFL in Section 3.3.1. Under the CFL assumption, nodes in the same cluster should learn similar model parameters.

According to Proposition 6.1, the solution to GTVMin will be approximately constant across a cluster $\mathcal{C}$ if the second smallest eigenvalue $\lambda_2\big(\mathbf{L}^{(\mathcal{C})}\big)$ is large and the cluster boundary $|\partial\mathcal{C}|$ is small. Here, $\lambda_2\big(\mathbf{L}^{(\mathcal{C})}\big)$ refers to the smallest nonzero eigenvalue of the Laplacian matrix of the induced subgraph $\mathcal{G}^{(\mathcal{C})}$.

Intuitively, $\lambda_2\big(\mathbf{L}^{(\mathcal{C})}\big)$ increases with the number of internal edges in $\mathcal{C}$. This can be made precise via Cheeger's inequality [36, Ch. 21]. Alternatively, we can approximate $\mathcal{G}^{(\mathcal{C})}$ as a realization of an Erdős–Rényi (ER) graph, a useful assumption especially if $\mathcal{G}$ itself resembles a typical realization of an ER graph.

In an ER graph over $\mathcal{C}$, each pair of nodes $i, i' \in \mathcal{C}$, with $i \neq i'$, is connected independently with probability p_e. Formally, the presence of edges are i.i.d. RVs $b^{(i,i')}$, one for each unordered pair $\{i, i'\} \subseteq \mathcal{C}$, indicating whether an edge

exists between the two nodes. As a result, the presence of edges between different pairs of nodes are statistically independent events.

This independence greatly simplifies analysis. For instance, the Laplacian matrix $\mathbf{L}^{(\mathrm{ER})}$ of an ER graph can be expressed as a sum of statistically independent random matrices:

$$\mathbf{L}^{(\mathrm{ER})} = \sum_{\{i,i'\}} b^{(i,i')} \mathbf{T}^{(i,i')}. \tag{153}$$

This decomposition involves, for each pair of different nodes $i, i' \in \mathcal{V}$, the deterministic matrix

$$\mathbf{T}^{(i,i')} = \big(\mathbf{e}^{(i)} - \mathbf{e}^{(i')}\big)\big(\mathbf{e}^{(i)} - \mathbf{e}^{(i')}\big)^T.$$

Here, $\mathbf{e}^{(i)}$ denotes the vector obtained from extracting the i-th column of the identity matrix $\mathbf{I}_n$. The decomposition (153) is useful for the analysis of the eigenvalues of $\mathbf{L}^{(\mathrm{ER})}$, e.g., via matrix concentration inequalities [90, 91].

Interpreting a graph $\mathcal{G}$ as (the realization of) an ER graph turns quantities such as node degrees $d^{(i)}$ and eigenvalues like $\lambda_2\big(\mathbf{L}^{(\mathcal{C})}\big)$ into (realizations of) RVs. The expected node degree is

$$\mathbb{E}\{d^{(i)}\} = p_e(|\mathcal{C}| - 1).$$

With high probability,

$$d^{(\mathcal{G})}_{\max} \approx p_e(|\mathcal{C}| - 1). \tag{154}$$

Increasing p_e results in a larger expected node degree and, thus, a higher connectivity of $\mathcal{G}^{(\mathcal{C})}$.

We can approximate $\lambda_2\big(\mathbf{L}^{(\mathcal{C})}\big)$ by the second smallest eigenvalue of the expected Laplacian matrix

$$\overline{\mathbf{L}} := \mathbb{E}\{\mathbf{L}^{(\mathcal{C})}\} = |\mathcal{C}| p_e \mathbf{I} - p_e \mathbf{1}\mathbf{1}^T.$$

A straightforward calculation yields

$$\lambda_2(\overline{\mathbf{L}}) = |\mathcal{C}| p_e.$$

Thus, we arrive at the approximation

$$\lambda_2\big(\mathbf{L}^{(\mathcal{C})}\big) \approx \lambda_2(\overline{\mathbf{L}}) = |\mathcal{C}| p_e \overset{(154)}{\approx} d^{(\mathcal{G})}_{\max}. \tag{155}$$

The precise quantification of the approximation error in (155) is beyond our scope. We refer interested readers to [90, 92] for further analysis of random graphs.

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 to communicate local model parameters across each edge of the FL network. This communication can be implemented using different physical channels, such as short-range wireless links or optical fibre connections [93, 94].

To summarize, using an FL network with smaller $d^{(i)}$ results in less computation and communication per iteration of Algorithm 4. Trivially, the lowest per-iteration cost occurs when $d^{(i)} = 0$, i.e., an empty FL network with $\mathcal{E} = \emptyset$. However, the overall computational cost also depends on the number of iterations required to approximate the GTVMin solution (51).

According to (80), the convergence speed of the gradient steps (110) used in Algorithm 4 depends on the condition number of the matrix $\mathbf{Q}$ in (103),

$$\text{condition number} = \frac{\lambda_{nd}(\mathbf{Q})}{\lambda_1(\mathbf{Q})}.$$

Faster convergence is achieved when this ratio is close to one (see (84)).

The condition number of $\mathbf{Q}$ tends to be smaller when the ratio between the maximum node degree $d_{\max}^{(\mathcal{G})}$ and the second smallest eigenvalue $\lambda_2\big(\mathbf{L}^{(\mathcal{G})}\big)$ is small (see (106) and (107)).

Thus, for a given maximum node degree $d_{\max}^{(\mathcal{G})}$, we should place the edges of an FL network so that $\lambda_2\big(\mathbf{L}^{(\mathcal{G})}\big)$ is large - leading to faster convergence of Algorithm 4 without increasing per-iteration complexity.

Spectral graph theory also provides upper bounds on $\lambda_2\big(\mathbf{L}^{(\mathcal{G})}\big)$ in terms of the node degrees [36, 95, 96]. These upper bounds can serve as a baseline for evaluating practical constructions of the FL network: If the resulting value $\lambda_2\big(\mathbf{L}^{(\mathcal{G})}\big)$ is close to its upper bound, then further attempts to improve connectivity (in terms of spectral properties) are unlikely to yield significant gains.

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,*

$$\lambda_2\big(\mathbf{L}^{(\mathcal{G})}\big) \leq \frac{n}{n-1} d^{(i)}, \quad \text{for every } i=1,\ldots,n. \tag{156}$$

Proof. The bound (156) follows from the variational characterization (41) by evaluating the quadratic form $\mathbf{w}^T\mathbf{L}^{(\mathcal{G})}\mathbf{w}$ for the specific vector

$$\widetilde{\mathbf{w}} = \sqrt{\frac{n}{n-1}}\Big(-\frac{1}{n},\ldots,\underbrace{1-\frac{1}{n}}_{\tilde{w}^{(i)}},\ldots,-\frac{1}{n}\Big)^T.$$

This "test" vector is tailored to a particular node $i \in \mathcal{V}$; its only positive entry is $\tilde{w}^{(i)} = 1-(1/n)$. It satisfies $\|\widetilde{\mathbf{w}}\| = 1$ and $\widetilde{\mathbf{w}}^T\mathbf{1} = 0$, making it a feasible vector for the optimization in (41). □

Alternative – and potentially tighter – upper bounds for $\lambda_2\big(\mathbf{L}^{(\mathcal{G})}\big)$ can be found in the graph theory literature [35, 36, 92, 97].

The per-iteration complexity of FL algorithms increases with the node degrees $d^{(i)}$ (and thus the total number of edges) in the FL network $\mathcal{G}$. On the other hand, the number of iterations required by Algorithm 4 typically decreases as the second smallest eigenvalue $\lambda_2\big(\mathbf{L}^{(\mathcal{G})}\big)$ increases.

According to the upper bound in (156), a large value of $\lambda_2\big(\mathbf{L}^{(\mathcal{G})}\big)$ is only possible if the node degrees $d^{(i)}$ - and hence the total number of edges - are sufficiently large. Recent work has focused on constructing graphs that maximize $\lambda_2\big(\mathbf{L}^{(\mathcal{G})}\big)$ given a fixed maximum node degree $d^{(\mathcal{G})}_{\max} = \max_{i \in \mathcal{V}} d^{(i)}$ [60, 98].

Figure 7.1 illustrates this trade-off between per-iteration complexity and the number of iterations required by FL algorithms.

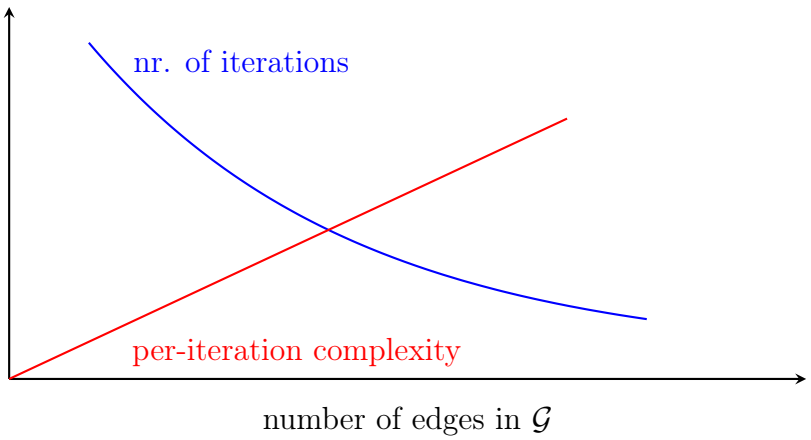


Fig. 7.1. Computational trade-off in GTVMin-based methods such as Algorithm 4: Increasing the number of edges in the FL network $\mathcal{G}$ raises the per-iteration complexity, but typically reduces the total number of iterations required for convergence.

7.2 Measuring (Dis-)Similarity Between Datasets

The main idea behind GTVMin is to enforce similar model parameters at two different nodes i and 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 design choices. Placing an edge between two nodes i, i' is typically only 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 for measuring the similarity – or, equivalently, the discrepancy (i.e., 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 parametric 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 $\mathbf{w}^{(i)}$ and $\mathbf{w}^{(i')}$ of the corresponding probability distributions.

In most FL applications, the parameters of the probability distribution $p^{(i)}\big(\mathcal{D}^{(i)}; \mathbf{w}^{(i)}\big)$ underlying a local dataset are unknown.[24] However, it is often possible to estimate these parameters using statistical inference techniques such as maximum likelihood estimation [23, Ch. 3], [30]. Given the estimates $\widehat{\mathbf{w}}^{(i)}$ and $\widehat{\mathbf{w}}^{(i')}$ for the model parameters, we can then compute the discrepancy measure $d^{(i,i')} := \big\|\widehat{\mathbf{w}}^{(i)} - \widehat{\mathbf{w}}^{(i')}\big\|_2$.

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 maximum likelihood estimator for $w^{(i)}$ is then given by $\hat{w}^{(i)} = y^{(i)}$ [30, 99].

[24] One exception is when the local dataset is generated by drawing i.i.d. realizations from $p^{(i)}\big(\mathcal{D}^{(i)}; \mathbf{w}^{(i)}\big)$.

Accordingly, the resulting discrepancy measure is [100].

$$d^{(i,i')} := \left|y^{(i)} - y^{(i')}\right|.$$

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 in the label space $\mathcal{Y}^{(i)}$. We can measure the similarity between nodes i and i' by the fraction of data points in $\mathcal{D}^{(i)} \bigcup \mathcal{D}^{(i')}$ with labels lying in $\mathcal{Y}^{(i)} \cap \mathcal{Y}^{(i')}$ [101].

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 probability distribution $\boldsymbol{\alpha}^{(i)} = \left(\alpha_0^{(i)}, \ldots, \alpha_9^{(i)}\right)$. 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 $\boldsymbol{\alpha}^{(i)}$. We also interpret the features as realizations of RVs having conditional probability 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 and i' via the distance between (estimations of) the parameters $\boldsymbol{\alpha}^{(i)}$ and $\boldsymbol{\alpha}^{(i')}$.

The above examples of a discrepancy measure – based on parameter estimates of a probabilistic model – are all special cases of a more general two-step approach:

- First, we assign a vector representation $\mathbf{z}^{(i)} \in \mathbb{R}^{m'}$ to each node $i \in \mathcal{V}$ [23, 102].
- Second, we define the discrepancy $d^{(i,i')}$ between nodes i and i' as the distance between the representation vectors $\mathbf{z}^{(i)}$ and $\mathbf{z}^{(i')}$, e.g.,

$$d^{(i,i')} := \left\|\mathbf{z}^{(i)} - \mathbf{z}^{(i')}\right\|.$$

We next discuss three specific implementations of the first step to obtain the representation vector for each node i.

Parametric Probabilistic Models. If we use a parametric probabilistic model $p\big(\mathcal{D}^{(i)}; \mathbf{w}^{(i)}\big)$ for the local dataset $\mathcal{D}^{(i)}$, we can use an estimator $\widehat{\mathbf{w}}^{(i)}$ to obtain $\mathbf{z}^{(i)}$. One popular approach for estimating the model parameters of a probabilistic model is the ML principle [23].

Gradients. We now discuss a construction for the vector representation $\mathbf{z}^{(i)} \in \mathbb{R}^{m'}$ that is inspired by the update structure of SGD. In particular, we define the discrepancy between two local datasets by treating them as two batches used by SGD to train a model. If these two batches consist of data points generated from similar probability distributions, their corresponding gradient approximations (112) are close. This suggests to use the gradient $\nabla f(\mathbf{w}')$ of the average loss (or empirical risk) $f(\mathbf{w}) := (1/|\mathcal{D}^{(i)}|) \sum_{(\mathbf{x},y) \in \mathcal{D}^{(i)}} L\left((\mathbf{x}, y), h^{(\mathbf{w})}\right)$ 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,

$$\mathbf{z}^{(i)} := \nabla L_i(\mathbf{v}). \tag{157}$$

Note that the construction (157) requires to specify the model parameters $\mathbf{v}$ at which the gradient is evaluated.

Feature learning. Another approach is to use an autoencoder [102, Ch. 14] to learn an embedding of a local dataset. In particular, we feed the dataset into an encoder ANN that has been jointly trained with a decoder ANN on a suitable learning task. The encoder maps the dataset to a latent vector, or embedding, which serves as its vector representation. A generic setup is illustrated in Figure 7.2.

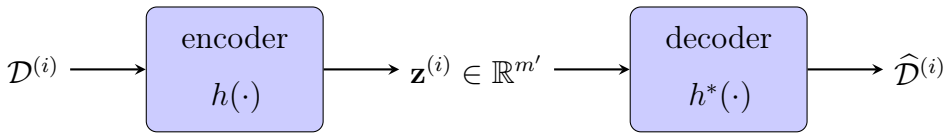


Fig. 7.2. A generic autoencoder consists of an encoder that maps the input to a latent representation, and a decoder that attempts to reconstruct the original input. Both components are trained jointly by minimizing a reconstruction loss (see [23, Ch. 9]). When a local dataset is used as input, its latent representation can serve as a compact vector embedding.

7.3 Graph Learning Methods

Assume we have constructed a discrepancy measure $d^{(i,i')} \in \mathbb{R}_+$ that quantifies the dissimilarity between any two local datasets $\mathcal{D}^{(i)}$ and $\mathcal{D}^{(i')}$. One way to construct an FL network is by connecting each node i to its nearest neighbors, i.e., the nodes $i' \in \mathcal{V} \setminus \{i\}$ with the smallest values of $d^{(i,i')}$.

An alternative to this nearest-neighbour construction is to formulate graph learning as a constrained linear optimization problem. Let us measure the quality of a candidate edge-weight assignment $A_{i,i'} \in \mathbb{R}_+$ using the objective function

$$\sum_{i,i' \in \mathcal{V}} A_{i,i'} d^{(i,i')}. \tag{158}$$

This function penalizes large weights between nodes that are dissimilar. Without any constraints, the minimum of (158) is trivially achieved by setting $A_{i,i'} = 0$ for all pairs, i.e., resulting in an empty graph.

As discussed in Section 7.1, however, a useful FL network must contain a sufficient number of edges to ensure that GTVMin produces meaningful

model parameters. In particular, the pooling effect of GTVMin depends on the second smallest eigenvalue $\lambda_2(\mathbf{L}^{(\mathcal{G})})$ of the Laplacian matrix being sufficiently large, which in turn requires that the graph is sufficiently well connected (see (155)).

To enforce the presence of edges, we introduce the following constraints:

$$A_{i,i} = 0, \quad \sum_{i' \neq i} A_{i,i'} = d^{(\mathcal{G})}_{\max} \quad \text{for all } i \in \mathcal{V}, \tag{159}$$
$$A_{i,i'} \in [0,1] \quad \text{for all } i, i' \in \mathcal{V}.$$

These constraints ensure that each node i has (weighted) node degree $\sum_{i' \neq i} A_{i,i'}$ equal to $d^{(\mathcal{G})}_{\max}$, and that edge weights are bounded and symmetric.

Combining the objective function (158) with the constraints (159), we arrive at the following graph learning principle:

$$\big\{\widehat{A}_{i,i'}\big\}_{i,i' \in \mathcal{V}} \in \operatorname*{arg\,min}_{A_{i,i'} = A_{i',i}} \sum_{i,i' \in \mathcal{V}} A_{i,i'} d^{(i,i')} \tag{160}$$
$$\text{s.t.} \quad A_{i,i'} \in [0,1] \quad \forall i, i' \in \mathcal{V},$$
$$A_{i,i} = 0 \quad \forall i \in \mathcal{V},$$
$$\sum_{i' \neq i} A_{i,i'} = d^{(\mathcal{G})}_{\max} \quad \forall i \in \mathcal{V}.$$

This constrained minimization problem is a special case of the general quadratic program introduced in (89). Because the objective is linear, (160) is equivalent to a linear program [46, Sec. 4.3]. Approximate solutions to (160) can be efficiently computed using projected GD, as discussed in Section 4.5.

The first constraint in (160) bounds edge weights between 0 and 1. The second prohibits self-loops, which have no effect on the outcome of GTVMin

(see (49)). The final constraint enforces regularity: every node has the same node degree $d^{(i)} = d^{(\mathcal{G})}_{\max}$.

While regular graphs simplify the analysis of GTVMin, they may not always be desirable in practice. In some FL applications, it may be advantageous to allow varying node degrees – such as graphs with a small number of "hub" nodes with high node degree [11, 100], or to minimize the total number of edges.

We can enforce an upper bound on the total number $E_{\max}$ of edges by modifying the last constraint in (160),

$$\begin{aligned} \widehat{A}_{i,i'} \in & \operatorname*{arg\,min}_{A_{i,i'}=A_{i',i}} \sum_{i,i' \in \mathcal{V}} A_{i,i'} d^{(i,i')} \\ & A_{i,i'} \in [0,1] \text{ for all } i, i' \in \mathcal{V}, \\ & A_{i,i} = 0 \text{ for all } i \in \mathcal{V}, \\ & \sum_{i',i \in \mathcal{V}} A_{i,i'} = E_{\max}. \end{aligned} \tag{161}$$

The problem has a closed-form solution as explained in [100]: 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 (161) 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 [68]. This communication network might be significantly different from the learned FL network. For some FL applications, the functional connectivity of two devices i and i' reflects also a similarity between probability distributions of local datasets $\mathcal{D}^{(i)}$ and $\mathcal{D}^{(i')}$ [103].

7.4 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 $\mathbf{x} \in \mathbb{R}^{(m_i)}$. We interpret the vectors $\mathbf{x} \in \mathbb{R}^{m_i}$, for $i = 1, \ldots, n$, as statistically independent 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 $c_i \in \{-1, 1\}$. We construct an FL network by determining for each node i its neighborhood $\mathcal{N}^{(i)}$ 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

This chapter examines how regulatory frameworks for trustworthy AI inform the design and implementation of GTVMin-based methods. Our discussion is primarily guided by the key requirements for trustworthy AI as formulated by the *European Union's High-Level Expert Group on AI* [104]. Comparable ethical frameworks have emerged globally, including *Australia's AI Ethics Principles* [105], the *OECD AI Principles* [106], China's governance efforts [107–109], and U.S. developments such as the *NIST AI Risk Management Framework* [110], the *Blueprint for an AI Bill of Rights* [111], and *Executive Order 14110 on the Safe, Secure, and Trustworthy Development and Use of Artificial Intelligence* [112].

Section 8.1 examines how FL systems can support human agency and oversight, as required by the principle of respect for human autonomy within the broader framework of trustworthy AI.

Section 8.2 investigates the robustness of FL systems against different forms of perturbations. Perturbations can arise from the intrinsic variability of local datasets that are obtained from stochastic data generation processes. Another source for perturbations are imperfections of the communication links between devices. We devote Chapter 10 to perturbations that are intentional (or adversarial) during so-called cyber attacks.

Section 8.3 addresses the need for privacy protection and data governance. This includes regulatory constraints on data processing, the data minimization principle, and the organizational structures needed to enforce compliance. We devote Chapter 9 to a detailed treatment of quantitative measures for privacy leakage and techniques to mitigate it in GTVMin-based FL systems.

Section 8.4 focuses on the transparency and the explainability of GTVMin-based FL systems. We introduce quantitative metrics for subjective explainability that reflect how well personalized models align with individual users' expectations. We can incorporate these metrics into GTVMin-based methods to ensure tailored explainability for heterogeneous populations of device users.

8.1 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...*" [104, p.15]

Human Dignity. Learning personalized model parameters for recommender systems allows to boost addiction or widespread emotional manipulation resulting in genocide [113–115]. 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* [116].

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 learned 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 which is then used, in turn, to diagnose the overall FL system (see [23, Sec. 6.6]).

8.2 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. ...*' [104, p.16].

Practical FL systems are obtained by implementing FL algorithms in physical distributed computers [17,18]. 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.

Distributed computers (as physical objects) typically incur imperfections, such as a temporary lack of connectivity or a mobile devices that run out of battery and therefore become inactive. Moreover, the data generation

processes can be subject to perturbations such as statistical anomalies or outliers. Section 8.2 studies in some detail the robustness of GTVMin-based systems against different perturbations of data sources and imperfections of computational infrastructure.

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 6.1, this single-model FL setting uses GTVMin (51) 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 adversarial) 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.

8.2.1 Sensitivity Analysis

As pointed out in Chapter 3, GTVMin (51) can be rewritten as the minimization of a quadratic function,

$$\min_{\mathbf{w}=\mathrm{stack}\{\mathbf{w}^{(i)}\}_{i=1}^{n}} \mathbf{w}^T \mathbf{Q} \mathbf{w} + \mathbf{q}^T \mathbf{w}. \tag{162}$$

The matrix $\mathbf{Q}$ and vector $\mathbf{q}$ are determined by the feature matrices $\mathbf{X}^{(i)}$ and label vectors $\mathbf{y}^{(i)}$ at the nodes $i \in \mathcal{V}$ (see (24)). We next study the sensitivity

of (the solutions of) (162) towards external perturbations of the label vector.[25]

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 (162),

$$\min_{\mathbf{w}=\mathrm{stack}\{\mathbf{w}^{(i)}\}} \mathbf{w}^T\mathbf{Q}\mathbf{w} + \mathbf{q}^T\mathbf{w} + \mathbf{n}^T\mathbf{w} + c. \tag{163}$$

An inspection of (24) yields that $\mathbf{n} = \begin{pmatrix} \big(\boldsymbol{\varepsilon}^{(1)}\big)^T\mathbf{X}^{(1)}, \ldots, \big(\boldsymbol{\varepsilon}^{(n)}\big)^T\mathbf{X}^{(n)} \end{pmatrix}^T$. The next result provides an upper bound on the deviation between the solutions of (162) and (163).

Proposition 8.1. *Consider the GTVMin instance* (162) *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* (105)*). Then, the deviation between the solution $\widehat{\mathbf{w}}^{(i)}$ to* (162) *and the solution $\widetilde{\mathbf{w}}^{(i)}$ to the perturbed problem* (163) *is upper bounded as*

$$\sum_{i=1}^{n} \big\|\widehat{\mathbf{w}}^{(i)} - \widetilde{\mathbf{w}}^{(i)}\big\|_2^2 \leq \frac{\lambda_{\max}(1+\rho^2)^2}{\big[\min\{\lambda_2\big(\mathbf{L}^{(\mathcal{G})}\big)\alpha\rho^2, \bar{\lambda}_{\min}/2\}\big]^2} \sum_{i=1}^{n} \big\|\boldsymbol{\varepsilon}^{(n)}\big\|_2^2.$$

Here, we used the shorthand $\rho := \bar{\lambda}_{\min}/(4\lambda_{\max})$ (see (105)*).*

Proof. The assumptions of Proposition 8.1 allow to apply the lower bound (107) on the eigenvalues of the matrix $\mathbf{Q}$ in (162). □

[25] Our study can be generalized to also take into account perturbations of the feature matrices $\mathbf{X}^{(i)}$, for $i = 1, \ldots, n$.

8.2.2 Estimation Error Analysis

Proposition 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 an FL system. For example, we can trivially achieve perfect robustness by delivering constant model parameters, e.g., $\widehat{\mathbf{w}}^{(i)} = \mathbf{0}$. Clearly, such a FL system is not very useful.

Another form of robustness is to ensure a small estimation error of (51). To study this form of robustness, we use a variant of the probabilistic model (59): 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

$$\mathbf{y}^{(i)} = \mathbf{X}^{(i)}\overline{\mathbf{w}} + \boldsymbol{\varepsilon}^{(i)}. \tag{164}$$

In contrast to Section 3.3.2, we assume that all components of (164) 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.[26]

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 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

[26] Consider labels obtained from physical sensing devices which are typically subject to measurement errors [117].

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 learned 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 3.3.2, these two components correspond to two orthogonal subspaces of $\mathbb{R}^{dn}$.

According to Proposition 3.1, the second error component is upper bounded as

$$\sum_{i=1}^{n} \big\|\widetilde{\mathbf{w}}^{(i)}\big\|_2^2 \leq \frac{1}{\lambda_2 \alpha} \sum_{i=1}^{n} (1/m_i) \big\|\boldsymbol{\varepsilon}^{(i)}\big\|_2^2. \tag{165}$$

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 (51))

$$\bar{\mathbf{c}} = \operatorname*{arg\,min}_{\mathbf{w} \in \mathbb{R}^d} \sum_{i \in \mathcal{V}} (1/m_i) \big\|\mathbf{y}^{(i)} - \mathbf{X}^{(i)}\big(\mathbf{w} - \widetilde{\mathbf{w}}^{(i)}\big)\big\|_2^2 + \alpha \sum_{\{i,i'\} \in \mathcal{E}} A_{i,i'} \Big\|\widetilde{\mathbf{w}}^{(i)} - \widetilde{\mathbf{w}}^{(i')}\Big\|_2^2. \tag{166}$$

Using a similar argument as in the proof for Proposition 2.1, we obtain

$$\|\bar{\mathbf{c}} - \overline{\mathbf{w}}\|_2^2 \leq \Big\|\sum_{i=1}^{n} (1/m_i)\big(\mathbf{X}^{(i)}\big)^T\big(\boldsymbol{\varepsilon}^{(i)} + \mathbf{X}^{(i)}\widetilde{\mathbf{w}}^{(i)}\big)\Big\|_2^2 / (n\bar{\lambda}_{\min})^2. \tag{167}$$

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 $i \in \mathcal{V}$.[27] Note that the bound (167) is only valid if $\bar{\lambda}_{\min} > 0$ which, in turn, implies that the solution to (166) is unique.

[27] We encountered the quantity $\bar{\lambda}_{\min}$ already during our discussion of gradient-based methods for solving the GTVMin instance (51) (see (105)).

We can develop (167) further using

$$\begin{aligned}
&\left\|\sum_{i=1}^{n}(1/m_i)\big(\mathbf{X}^{(i)}\big)^T\big(\boldsymbol{\varepsilon}^{(i)}+\mathbf{X}^{(i)}\widetilde{\mathbf{w}}^{(i)}\big)\right\|_2 \\
&\overset{(a)}{\leq} \sum_{i=1}^{n}\left\|(1/m_i)\big(\mathbf{X}^{(i)}\big)^T\big(\boldsymbol{\varepsilon}^{(i)}+\mathbf{X}^{(i)}\widetilde{\mathbf{w}}^{(i)}\big)\right\|_2 \\
&\overset{(b)}{\leq} \sqrt{n}\sqrt{\sum_{i=1}^{n}\left\|(1/m_i)\big(\mathbf{X}^{(i)}\big)^T\big(\boldsymbol{\varepsilon}^{(i)}+\mathbf{X}^{(i)}\widetilde{\mathbf{w}}^{(i)}\big)\right\|_2^2} \\
&\overset{(c)}{\leq} \sqrt{n}\sqrt{\sum_{i=1}^{n}2\left\|(1/m_i)\big(\mathbf{X}^{(i)}\big)^T\boldsymbol{\varepsilon}^{(i)}\right\|_2^2+2\left\|(1/m_i)\big(\mathbf{X}^{(i)}\big)^T\mathbf{X}^{(i)}\widetilde{\mathbf{w}}^{(i)}\right\|_2^2} \\
&\overset{(d)}{\leq} \sqrt{n}\sqrt{\sum_{i=1}^{n}(2/m_i)\lambda_{\max}\|\boldsymbol{\varepsilon}^{(i)}\|_2^2+2\lambda_{\max}^2\|\widetilde{\mathbf{w}}^{(i)}\|_2^2}. \qquad (168)
\end{aligned}$$

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 (105)).

Inserting (168) into (167) results in the upper bound

$$\begin{aligned}
\|\bar{\mathbf{c}}-\overline{\mathbf{w}}\|_2^2 &\leq 2\sum_{i=1}^{n}\left[(1/m_i)\lambda_{\max}\big\|\boldsymbol{\varepsilon}^{(i)}\big\|_2^2+\lambda_{\max}^2\big\|\widetilde{\mathbf{w}}^{(i)}\big\|_2^2\right]/(n\bar{\lambda}_{\min}^2) \\
&\overset{(165)}{\leq} 2\big(\lambda_{\max}+(\lambda_{\max}^2/(\lambda_2\alpha))\big)\sum_{i=1}^{n}(1/m_i)\big\|\boldsymbol{\varepsilon}^{(i)}\big\|_2^2/(n\bar{\lambda}_{\min}^2). \qquad (169)
\end{aligned}$$

The upper bound (169) 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 (105)). Let us next discuss how the upper bound (169) might guide the choice of the FL network $\mathcal{G}$ and the features of data points in the

local datasets.

According to (169), 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 (169) suggests using features that result in a small ratio $\lambda_{\max}/\bar{\lambda}_{\min}$ between the quantities $\lambda_{\max}$ and $\bar{\lambda}_{\min}$ (see (105)). Some feature learning methods have been proposed in order to minimize this ratio [23, 118].

8.2.3 Robustness of FL Algorithms

The previous sub-sections studied the robustness of GTVMin solutions against perturbations of local datasets. Ensuring trustworthy FL systems also requires robustness of FL algorithms against perturbations of their executions. It turns out that our design choices (e.g., the shape of local loss functions) for GTVMin crucially affect the robustness of the FL algorithms discussed in Section 5.

For ease of exposition, we will focus on FL algorithms for parametric local models that are based on the update

$$\mathbf{w}^{(i,t+1)} \in \operatorname*{arg\,min}_{\mathbf{w}^{(i)} \in \mathbb{R}^d} \left[L_i\left(\mathbf{w}^{(i)}\right) + \sum_{i' \in \mathcal{N}^{(i)}} A_{i,i'} \phi\big(\mathbf{w}^{(i',t)} - \mathbf{w}^{(i,t)}\big) \right]. \tag{170}$$

Note that Algorithm 5 and Algorithm 11 use (170) as their core computational step. We next discuss the robustness of (170) against perturbations of the

model parameters $\mathbf{w}^{(i',t)}$ that device receives from its neighbors $i' \in \mathcal{N}^{(i)}$. We focus on two specific choices for the GTV penalty function ϕ.

GTV penalty $\phi(\cdot) = \|\cdot\|_2^2$. For the penalty function $\phi(\mathbf{w}^{(i)} - \mathbf{w}^{(i')}) = \left\|\mathbf{w}^{(i)} - \mathbf{w}^{(i')}\right\|_2^2$, we can rewrite (170) as (see Exercise 5.6)

$$\mathbf{w}^{(i,t+1)} \in \operatorname*{arg\,min}_{\mathbf{w}^{(i)} \in \mathbb{R}^d} L_i\left(\mathbf{w}^{(i)}\right) + \alpha d^{(i)} \left\|\mathbf{w}^{(i)} - \widehat{\mathbf{w}}^{(\mathcal{N}^{(i)})}\right\|_2^2. \tag{171}$$

Here, we used $\widehat{\mathbf{w}}^{(\mathcal{N}^{(i)})} := (1/d^{(i)}) \sum_{i' \in \mathcal{N}^{(i)}} A_{i,i'} \mathbf{w}^{(i',t)}$ and the weighted node degree $d^{(i)} = \sum_{i' \in \mathcal{N}^{(i)}} A_{i,i'}$ (see (34)).

If the local loss function $L_i(\cdot)$ is convex, and under some mild technical conditions,[28] the update (171) is well-defined, i.e., the minimization has a unique solution [38, Ch. 6]. Moreover, the update (171) then coincides with an application of the proximal operator $\mathbf{prox}_{L_i(\cdot),\rho}(\cdot)$ (see (57)) of $L_i(\cdot)$ [39],

$$\mathbf{w}^{(i,t+1)} = \mathbf{prox}_{L_i(\cdot),\rho}(\widehat{\mathbf{w}}^{(\mathcal{N}^{(i)})}) \text{ with } \rho = 2\alpha d^{(i)}. \tag{172}$$

Figure 8.1 illustrates the update (172) as a straight line. The slope of this line indicates the robustness of (172) against perturbations of the received model parameters $\mathbf{w}^{(i',t)}$, for $i' \in \mathcal{N}^{(i)}$. These perturbations result in a modified input $\widetilde{\mathbf{w}}^{(i)}$ (instead of $\widehat{\mathbf{w}}^{(\mathcal{N}^{(i)})}$) for the proximal operator $\mathbf{prox}_{L_i(\cdot),\rho}(\widehat{\mathbf{w}}^{(\mathcal{N}^{(i)})})$. A natural quantitative measure for the robustness (or stability) of (172) is

$$\frac{\left\|\mathbf{prox}_{L_i(\cdot),\rho}(\widetilde{\mathbf{w}}^{(i)}) - \mathbf{prox}_{L_i(\cdot),\rho}(\widehat{\mathbf{w}}^{(\mathcal{N}^{(i)})})\right\|_2}{\left\|\widetilde{\mathbf{w}}^{(i)} - \widehat{\mathbf{w}}^{(\mathcal{N}^{(i)})}\right\|_2}. \tag{173}$$

[28] Strictly speaking, we need to require loss function $L_i(\cdot)$ to have a non-empty and closed epigraph which does not contain any non-horizontal lines [39].

It turns out that if the local loss function $L_i(\cdot)$ is strongly convex with coefficient σ, then (173) is upper bounded by [72, Sec. 6]

$$\frac{1}{1+(\sigma/\rho)} = \frac{1}{1+(\sigma/(2\alpha d^{(i)}))}. \tag{174}$$

We can interpret the quantity (174) as a measure for the robustness of the update (171). The smaller this quantity, the more robust are FL systems based on (171).

Note how the robustness measure (174) can guide the design choices for the components of GTVMin. In particular, to ensure a small value (174) (ensuring robustness), we should use

- a local loss function that is strongly convex with a large coefficient σ,
- a FL network with small node degrees $d^{(i)}$,
- a small value α for GTVMin parameter.

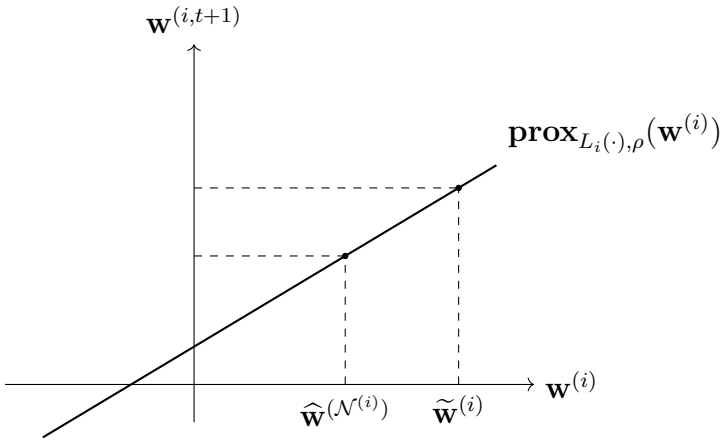


Fig. 8.1. For a convex local loss function $L_i(\cdot)$, the update (171) becomes the evaluation of the proximal operator $\mathbf{prox}_{L_i(\cdot),\rho}(\cdot)$ with $\rho = 2\alpha d^{(i)}$. We can measure the robustness of (171) by the slope of $\mathbf{prox}_{L_i(\cdot),\rho}(\cdot)$ (see (173)).

GTV penalty $\phi(\cdot) = \|\cdot\|_2$. Let us now study (170) for the centre node $i = 1$ of a star-shaped FL network (see Figure 5.3). This uses the trivial local loss function $L_i(\cdot) \equiv 0$ and is connected via unit-weight edges to the peripheral nodes $i' = 2, \ldots, n$. The variation of local model parameters is measured with the penalty function $\phi(\mathbf{w}^{(i)} - \mathbf{w}^{(i')}) = \left\|\mathbf{w}^{(i)} - \mathbf{w}^{(i')}\right\|_2$. This special case of (170) can be written as

$$\mathbf{w}^{(1,t+1)} \in \operatorname*{arg\,min}_{\mathbf{w}^{(i)} \in \mathbb{R}^d} \sum_{i'=2}^{n} \left\|\mathbf{w}^{(i',t)} - \mathbf{w}^{(i)}\right\|_2. \tag{175}$$

Note that (175) is nothing but the geometric median of the model parameters $\mathbf{w}^{(i',t)}$, for $i' \in \mathcal{N}^{(i)}$. The usefulness of the geometric median for robust FL has been studied recently [119].

The update (175) defines a non-smooth convex optimization problem. Any solution $\mathbf{w}^{(1,t+1)}$ to this problem must satisfy the subgradient optimality

condition

$$\sum_{i'=2}^{n} \mathbf{g}^{(i')} = \mathbf{0} \quad \text{, with } \mathbf{g}^{(i')} = \begin{cases} \frac{\mathbf{w}^{(i',t)} - \mathbf{w}^{(1,t+1)}}{\left\|\mathbf{w}^{(i',t)} - \mathbf{w}^{(1,t+1)}\right\|_2} & \text{if } \mathbf{w}^{(i',t)} \neq \mathbf{w}^{(1,t+1)} \\ \mathbf{u} \in \mathcal{B}(1) & \text{otherwise,} \end{cases} \tag{176}$$

where $\mathcal{B}(1) := \{\mathbf{u} \in \mathbb{R}^d : \|\mathbf{u}\|_2 \leq 1\}$ denotes the unit Euclidean ball. Each $\mathbf{g}^{(i')}$ is a subgradient of the convex non-smooth function $f(\mathbf{w}^{(i)}) := \left\|\mathbf{w}^{(i',t)} - \mathbf{w}^{(i)}\right\|_2$.

Figure 8.2 illustrates the optimality condition (176) for the case where node $i = 1$ has three neighbors, two of which are trustworthy. The third neighbour is not trustworthy and may send arbitrarily corrupted model parameters. Despite such adversarial perturbations, the solution $\mathbf{w}^{(1,t+1)}$ of (176) cannot be arbitrarily far from the model parameters of the trustworthy neighbors, provided they form the majority.

Intuitively, if the solution were far from the honest models, then the corresponding subgradients $\mathbf{g}^{(i')}$ for the trustworthy neighbors $i' \in \mathcal{N}^{(i)}$ would point in nearly the same direction, and their sum would have a norm close to the number of honest neighbors. However, the subgradients from the non-trustworthy nodes—being unit vectors—cannot cancel this sum unless they are sufficiently numerous, which contradicts the majority assumption. For a more detailed robustness analysis of (176), we refer to [120, Thm. 2.2].

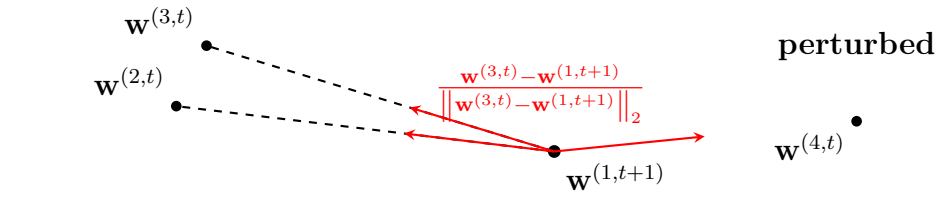


Fig. 8.2. Illustration of the (zero-subgradient) optimality condition (176) for the update (175). The arrows represent unit-norm subgradients arising from the components $\left\|\mathbf{w}^{(i',t)} - \mathbf{w}^{(i)}\right\|_2$ for $i' = 2, \ldots, n$.

8.2.4 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 (169)). 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 (51). 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 [19, 121].

The effect of imperfections in the implementation of the GD based Algorithm 4 can be modelled as perturbed GD (87) from Chapter 4. We can then analyze the robustness of the resulting FL system via the convergence analysis of perturbed GD discussed in Section 4.4.

According to (88), 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).

Instead of modelling the effect of network failures as perturbed GD, we can instead interpret it as exact GD applied to a perturbed instance of GTVMin. This perturbed instance uses a pruned FL network $\widetilde{\mathcal{G}}$, consisting of edges that are still active (i.e., corresponding to active communication links).

The effectiveness of GTVMin crucially depends on the second-smallest

eigenvalue λ_2 of the Laplacian matrix (36) associated with the FL network $\widetilde{\mathcal{G}}$ (see Section 3.3.2). As discussed in Section 7.1, λ_2 reflects how well-connected the FL network $\widetilde{\mathcal{G}}$ is. A larger λ_2 means better connectivity, which is required by GTVMin to combine the information provided by devices that work on similar learning tasks (see Section 6.2).

To make GTVMin robust against communication link failures, we need to design the original FL network $\mathcal{G}$ so that even if some edges are removed, the resulting $\widetilde{\mathcal{G}}$ still stays well connected—that is, λ_2 remains large enough. This idea is related to resilient network design, which studies how to build networks that stay connected even when some parts fail [122, 123].

8.3 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...*" [104, 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 (49) 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 [124, Sec. 3.3.])

Data processing regulations limit the choice of the features of a data point [125–127]. 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 [127]. It is useful (or even compulsory) to designate a data protection officer and conduct impact assessments [104].

Privacy. The operation of an FL system must not violate the fundamental human right to privacy [128]. One of most important characteristics of FL, and distinguishing from distributed optimization, is the privacy friendly exchange of information among the system components. We dedicate the entire Chapter 9 to the discussion of quantitative measures and methods for privacy protection in GTVMin-based FL systems.

8.4 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 an FL system can be facilitated by a sufficient level of explainability of the trained personalized model $\hat{h}^{(i)} \in \mathcal{H}^{(i)}$. It is important to note that the explainability of $\hat{h}^{(i)}$ is subjective: A given learned 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 [129–131]: 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 (102) 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]

[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.

We measure the (subjective) explainability of a linear hypothesis with model parameters $\mathbf{w}^{(i)}$ by

$$(1/|\mathcal{D}_t^{(i)}|) \sum_{\mathbf{x} \in \mathcal{D}_t^{(i)}} \left(u^{(i)}(\mathbf{x}) - \mathbf{x}^T \mathbf{w}^{(i)} \right)^2 . \tag{177}$$

It seems natural to add this measure as a penalty term to the local loss function in (102), resulting in the new loss function

$$L_i\left(\mathbf{w}^{(i)}\right) := \underbrace{(1/m_i)\left\|\mathbf{y}^{(i)} - \mathbf{X}^{(i)}\mathbf{w}^{(i)}\right\|_2^2}_{\text{training error}} + \rho \underbrace{(1/|\mathcal{D}_t^{(i)}|) \sum_{\mathbf{x} \in \mathcal{D}_t^{(i)}} \left(u^{(i)}(\mathbf{x}) - \mathbf{x}^T \mathbf{w}^{(i)}\right)^2}_{\text{subjective explainability}} . \tag{178}$$

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 [131]. It can be shown that (178) 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 learned 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 learned hypothesis maps $\hat{h}^{(i)}$. We can explain an FL algorithm by providing partial information about this map $\mathcal{A}$. Thus, mathematically speaking, the problem of explaining a learned hypothesis is

essentially the same as the problem of explaining an entire FL algorithm: Provide partial information about a map such that the user can anticipate the results of applying the map to arbitrary arguments. However, a description of the map $\mathcal{A}$ is typically more complex, in a quantitative sense, than a learned hypothesis map.

The different complexity levels of maps to be explained requires different forms of explanation. For example, we could explain an FL algorithm using a pseudo-code such as Algorithm 4. Fig. 8.3 illustrates another form of explanation, i.e., a code fragment written in the programming language Python.

```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
```

Fig. 8.3. Python code for a ML method that trains a decision tree on the *Iris* dataset.

8.5 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.*" [104, p.18].

The local datasets used for the training of local models should be carefully selected to not enforce existing discrimination. In a health-care 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 [124, 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 [132]. 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 Fig. 8.4).

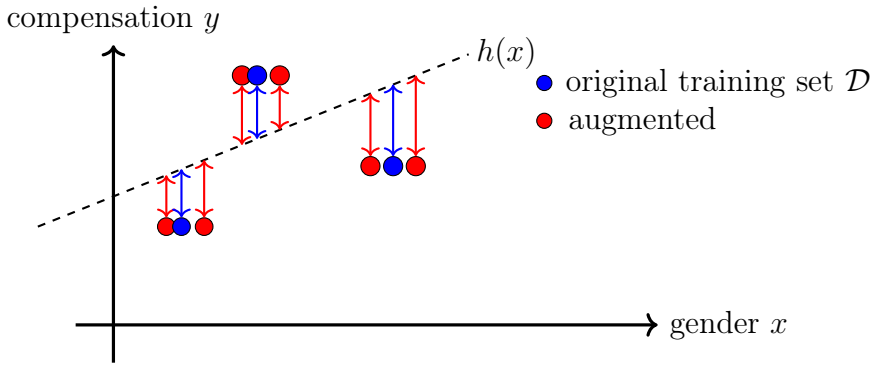


Fig. 8.4. We can improve the fairness of a ML method by augmenting the training set using perturbations of an irrelevant feature such as the gender of a person for which we want to predict the adequate compensation as the label.

8.6 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.*" [104, 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 [133].

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 [134].

8.7 Exercises

8.1. Robustness of GTVMin. Discuss the robustness of GTVMin (51) 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 (51) to train local linear models with model parameters $\mathbf{w}^{(i)}$. The local datasets are modelled as (59). 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 (178) used in (51). Study, either analytically or by numerical experiments, the effect of varying levels of explainability (via the parameter ρ in (178)) 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 which represent human patients [135]. 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 modifications to basic FL algorithms from Chapter 5.

This chapter revolves around two main questions:

- How can we measure privacy leakage in an FL system?
- How can we control (minimize) privacy leakage of an FL system?

Section 9.1 addresses the first question while Sections 9.2 and 9.3 address the second question.

9.1 Measuring Privacy Leakage

Consider an FL system designed to train personalized models for users indexed by $i = 1, \ldots, n$, each equipped with a heart rate sensor. Every user i generates

a local dataset $\mathcal{D}^{(i)}$, consisting of time-stamped heart rate measurements. A single data point corresponds to one physical activity, such as a 50-minute run. The features of such a data point include a time series of GPS coordinates, while the label may be the average heart rate recorded during the activity. We assume that this average heart rate is private and should not be disclosed to third parties.[30]

To enhance learning, the FL system incorporates expert-provided information in the form of pairwise similarity scores $A_{i,i'}$ between users i and i', based on characteristics such as body weight and height. These similarity scores are used to regularize the learning process.

Using an FL algorithm—such as Algorithm 4—we aim to learn, for each user i, personalized model parameters $\mathbf{w}^{(i)}$ for an AI-powered healthcare assistant [136]. This algorithm can be represented as a map $\mathcal{A}(\cdot)$ that takes as input the collection of local datasets

$$\mathcal{D} := \left\{\mathcal{D}^{(i)}\right\}_{i=1}^{n}$$

and delivers learned model parameters

$$\mathcal{A}(\mathcal{D}) := \left(\widehat{\mathbf{w}}^{(1)}, \ldots, \widehat{\mathbf{w}}^{(n)}\right).$$

Figure 9.1 illustrates the mapping from local datasets to learned model parameters that is implemented by an FL algorithm.

A privacy-preserving FL system should not allow to infer, solely from the learned 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

[30] For instance, individuals may not wish to share heart rate profiles with potential employers.

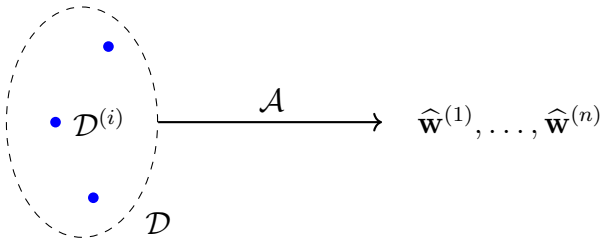


Fig. 9.1. A FL algorithm maps the local datasets $\mathcal{D}^{(i)}$ to the learned model parameters $\widehat{\mathbf{w}}^{(i)}$, for $i = 1, \ldots, n$.

$\mathcal{A}$ is not invertible: The learned model parameters (or hypothesis) 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 depicts the decision regions of a decision tree. This decision tree has been trained by (approximately) solving ERM with a training set that consists of four data points. Each data point is characterized by a feature vector $\mathbf{x}^{(r)} = \big(x_1^{(r)}, x_2^{(r)}\big)^T$ and a binary label $y^{(r)} \in \{\circ, \times\}$, for $r = 1, \ldots, 5$. If an attacker would know the label values of $\mathbf{x}^{(1)}, \mathbf{x}^{(4)}$, it could infer the label of $\mathbf{x}^{(2)}$ based on the decision regions.

The sole requirement for an FL algorithm $\mathcal{A}$ to be not invertible is not sufficient 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. Note that the input space 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 (or non-injectivity).

A simple measure of non-invertibility is the sensitivity of the output $\mathcal{A}\big(\mathcal{D}\big)$

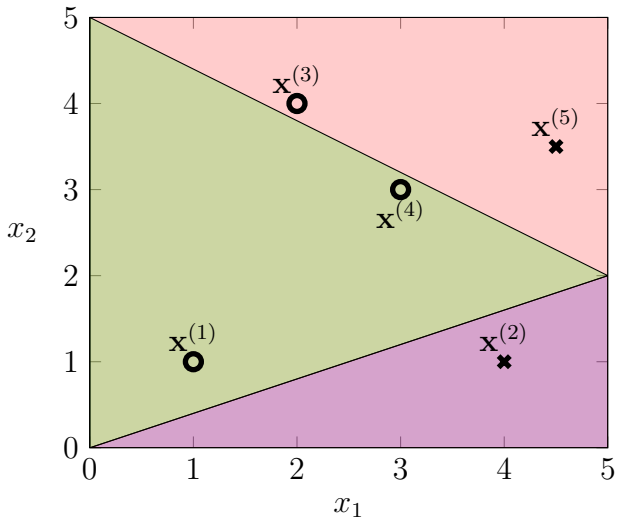


Fig. 9.2. Scatterplot of a dataset used to train a decision tree. We indicate the decision regions along with the labels of data points (via their markers).

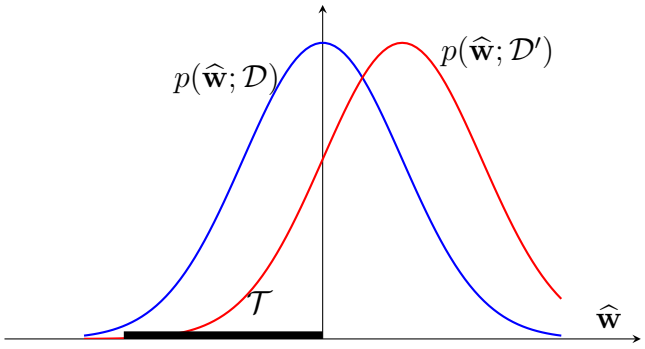


Fig. 9.3. Probability distributions of the learned model parameters $\widehat{\mathbf{w}} = \left(\widehat{\mathbf{w}}^{(1)}, \ldots, \widehat{\mathbf{w}}^{(n)}\right)$ delivered by some FL algorithm for two different input datasets, denoted by $\mathcal{D}'$ and $\mathcal{D}$.

against varying the heart rate value $y^{(i,r)}$,

$$\frac{\left\|\mathcal{A}(\mathcal{D})-\mathcal{A}(\mathcal{D}')\right\|_2}{\varepsilon}. \tag{179}$$

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 offered by $\mathcal{A}$ is higher for smaller values (179), 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 for learning model parameters. One example of such a mechanism is the random selection of a subset data points that form a batch within one iteration of FedSGD (see Algorithm 7). Section 9.2 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 the possible values of the learned model parameters $\widehat{\mathbf{w}}$. Figure 9.3 illustrates a stochastic algorithm along with the associated probability distribution $p(\widehat{\mathbf{w}};\mathcal{D})$.[31] This probability distribution is parametrized by the dataset $\mathcal{D}$ that is fed as input to the algorithm $\mathcal{A}$. Figure 9.3 depicts the probability distributions of an algorithm for two different choices $\mathcal{D},\mathcal{D}'$ of the input dataset.

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

[31] For more details about the concept of a measurable space, we refer to the literature [28, 137, 138].

are considered as adjacent (or neighbouring) [124, 139]. 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 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 changing the feature x_j of a single data point. In general, the notion of neighbouring datasets is a design choice used in the definition of quantitative measures for privacy protection. A FL algorithm ensures privacy protection if there is no statistical test that allows to reliably distinguish between neighbouring input datasets. Figure 9.3 illustrates the acceptance region $\mathcal{T}$ that defines a statistical test.

The de-facto standard for quantifying privacy leakage in ML and FL systems is the following definition.

Definition 1. *(from [139]) A stochastic algorithm $\mathcal{A}$ is (ε, δ)-DP if, for any two neighbouring datasets $\mathcal{D}, \mathcal{D}'$,*

$$\text{Prob}\big\{\mathcal{A}(\mathcal{D}) \in \mathcal{S}\big\} \leq \exp(\varepsilon)\text{Prob}\big\{\mathcal{A}(\mathcal{D}') \in \mathcal{S}\big\} + \delta$$

holds for every measurable set $\mathcal{S}$.

Definition 1 formalizes the notion that the presence or absence of an individual data point (representing, e.g., human individual) in a dataset $\mathcal{D}$ should not significantly affect the probability distribution of the output $\mathcal{A}(\mathcal{D})$. The notion of (ε, δ)-DP is widely adopted in FL applications [139–142]. The U.S. Census Bureau adopted (ε, δ)-DP for the 2020 census [142]. The National Institute of Standards and Technology (NIST) has published some guidance

for evaluating and implementing DP mechanisms in government and industry settings [143].

Besides (ε, δ)-DP, there are also other measures for privacy leakage. These measures differ by how they quantify precisely the similarity between probability distributions $p(\widehat{\mathbf{w}}; \mathcal{D})$ and $p(\widehat{\mathbf{w}}; \mathcal{D}')$ induced by neighbouring datasets [144]. One such alternative measure is the Rényi divergence of order $\alpha > 1$,

$$D_\alpha\Big(p(\widehat{\mathbf{w}}; \mathcal{D})\Big\|p(\widehat{\mathbf{w}}; \mathcal{D}')\Big) := \frac{1}{\alpha - 1}\mathbb{E}_{p(\widehat{\mathbf{w}}; \mathcal{D}')}\left[\left(\frac{dp(\widehat{\mathbf{w}}; \mathcal{D})}{dp(\widehat{\mathbf{w}}; \mathcal{D}')}\right)^\alpha\right].$$

The Rényi divergence allows to define the following variant of DP [144, 145].

Definition 2. *(from [139]) An algorithm $\mathcal{A}$ is (α, γ)-RDP if, for any two neighbouring datasets $\mathcal{D}$ and $\mathcal{D}'$,*

$$D_\alpha\Big(p(\widehat{\mathbf{w}}; \mathcal{D})\Big\|p(\widehat{\mathbf{w}}; \mathcal{D}')\Big) \leq \gamma.$$

A recent use-case of (α, γ)-RDP is the analysis of DP guarantees offered by variants of SGD [144]. This analysis uses the fact that (α, γ)-RDP implies (ε, δ)-DP for suitable choices of ε, δ [144].

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 an 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., [139, Sec. 2.3]. □

According to Proposition (9.1), the level of DP offered by an algorithm $\mathcal{A}$ does not deteriorate by any post-processing of its output. It seems almost

natural to make this immunity against post-processing a defining property of any useful notion of DP [145]. However, due to Proposition (9.1), this property is already "built-in" into the Definition 1 and Definition 2.

Operational Meaning of DP. The mathematically precise formulation of DP in Definition 1 is somewhat abstract. It is instructive to interpret (ε, δ)-DP from the perspective of hypothesis testing [143]: We use the output $\widehat{\mathbf{w}} \in \mathbb{R}^d$ of algorithm $\mathcal{A}$ to test if the underlying dataset fed into $\mathcal{A}$ was $\mathcal{D}$ or if it was a neighbouring dataset $\mathcal{D}'$ [146]. Such a statistical test uses a region $\mathcal{T} \subseteq \mathbb{R}^d$ and to declare

- "dataset $\mathcal{D}$ seems to be used" if $\widehat{\mathbf{w}} \in \mathcal{T}$, or
- "dataset $\mathcal{D}'$ seems to be used" if $\widehat{\mathbf{w}} \notin \mathcal{T}$.

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 simultaneously small (close to 0). This intuition can be made precise as follows (see [147, Thm. 2.1.], [143] or [148]): If an algorithm $\mathcal{A}$ is (ε, δ)-DP, then

$$\exp(\varepsilon) P_{\mathcal{D}\to\mathcal{D}'} + P_{\mathcal{D}'\to\mathcal{D}} \geq 1 - \delta. \tag{180}$$

Thus, if $\mathcal{A}$ is (ε, δ)-DP with a small ε, δ (close to 0), then (180) implies $P_{\mathcal{D}\to\mathcal{D}'} + P_{\mathcal{D}'\to\mathcal{D}} \approx 1$.

9.2 Ensuring Differential Privacy

Depending on the underlying local datasets, local models, and optimization method, a GTVMin-based method $\mathcal{A}$ might already ensure DP by design. A basic means of ensuring DP is through careful feature selection (or learning) for the local datasets (see Figure 9.5. Random sampling used by SGD-based algorithms can also provide a certain level of DP [149, 150].

According to Proposition 9.1, DP can also be actively ensured by applying pre- and post-processing techniques to the input and output of an FL algorithm $\mathcal{A}$. Mathematically, the map $\mathcal{A}$ is concatenated with the maps $\mathcal{I}$ and $\mathcal{O}$. These maps represent the pre- and post-processing and are typically stochastic, i.e., defined by a conditional probability distribution. The concatenation results in a new algorithm $\mathcal{A}' := \mathcal{O} \circ \mathcal{A} \circ \mathcal{I}$. In summary, for a given dataset $\mathcal{D}$, the new (privacy-enhanced) algorithm $\mathcal{A}'$ produces learned model parameters by:

- apply the pre-processing $\mathcal{I}(\mathcal{D})$,
- compute $\mathcal{A}\big(\mathcal{I}(\mathcal{D})\big)$ using the original algorithm,
- and finally apply the post-processing $\mathcal{O}\big(\mathcal{A}\big(\mathcal{I}(\mathcal{D})\big)\big)$, yielding $\mathcal{A}'(\mathcal{D})$.

Post-Processing. Maybe the most widely used post-processing technique for DP is to add *some noise* [139],

$$\mathcal{O}(\mathcal{A}) := \mathcal{A} + \mathbf{n}, \text{ with noise } \mathbf{n} = \big(n_1, \ldots, n_{nd}\big)^T, \; n_1, \ldots, n_{nd} \overset{i.i.d.}{\sim} p(n). \quad (181)$$

Note that the post-processing (181) 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 (181), the variance σ^2 can be chosen based on the sensitivity

$$\Delta_2\big(\mathcal{A}\big) := \max_{\mathcal{D},\mathcal{D}'} \|\mathcal{A}(\mathcal{D}) - \mathcal{A}(\mathcal{D}')\|_2. \tag{182}$$

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 [139, Thm. 3.22]. It might be difficult to evaluate the sensitivity (182) for a given FL algorithm $\mathcal{A}$ [151]. For a GTVMin-based method, i.e., $\mathcal{A}(\mathcal{D})$ is a solution to (49), we can upper bound $\Delta_2\big(\mathcal{A}\big)$ via a perturbation analysis similar in spirit to the proof of Proposition 8.1.

Pre-Processing. Instead of ensuring DP via post-processing the output of an 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 Proposition 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 (181)): [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

[32] Can you think of a simple pre-processing map that is deterministic and guarantees maximum DP?

randomly selected element of $\mathcal{Y}$ [152, 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 $\mathcal{D}$ [152, Mechanism 2].

- Another form of noise injection is to construct $\mathcal{I}(\mathcal{D})$ by randomly selecting data points from the original (private) dataset $\mathcal{D}$ [153]. 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 $\mathcal{D}^{(1)}_{\rm syn}, \ldots, \mathcal{D}^{(L)}_{\rm syn}$ 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 $\mathcal{D}^{(r)}_{\rm syn}$ by applying a learned hypothesis $\widehat{h}$ to the output $\mathcal{A}\big(\mathcal{D}^{(r)}_{\rm syn}\big)$ delivered by the *algorithm under test* $\mathcal{A}$. The hypothesis $\widehat{h}$ might be learned by an ERM-based method (see Algorithm 1) using a training set consisting of pairs $\Big(\mathcal{A}\big(\mathcal{D}^{(r)}_{\rm syn}\big), s^{(r)}\Big)$ for some $r \in \{1, \ldots, L\}$.

9.3 Private Feature Learning

Section 9.2 discussed pre-processing techniques that ensure DP of an 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 [23, 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 use the MI $I\left(y; \mathbf{\Phi}(\mathbf{x})\right)$ to measure the predictability of the label y from $\mathbf{\Phi}(\mathbf{x})$. A large value $I\left(y; \mathbf{\Phi}(\mathbf{x})\right)$ 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

[33] The relation between MI-based privacy measures and DP has been studied in some detail recently [154].

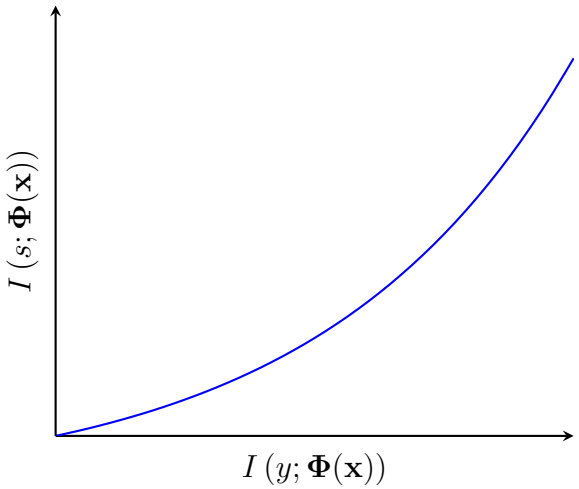


Fig. 9.4. The solutions of the privacy funnel (183) trace out (for varying constraint R) a curve in the plane spanned by the values of $I\left(s;\mathbf{\Phi}(\mathbf{x})\right)$ (measuring the privacy leakage) and $I\left(y;\mathbf{\Phi}(\mathbf{x})\right)$ (measuring the usefulness of the transformed features for predicting the label).

small $I\left(s;\mathbf{\Phi}(\mathbf{x})\right)$, i.e., a sufficiently large privacy protection, with a sufficiently large $I\left(y;\mathbf{\Phi}(\mathbf{x})\right)$ to allow for an accurate prediction of y. The mathematically precise formulation of this plan is known as the privacy funnel [155, Eq. (2)],

$$\min_{\mathbf{\Phi}(\cdot)} I\left(s;\mathbf{\Phi}(\mathbf{x})\right) \text{ such that } I\left(y;\mathbf{\Phi}(\mathbf{x})\right) \geq R. \tag{183}$$

Figure 9.4 illustrates the solution of (183) for varying R, i.e., the minimum value of $I\left(y;\mathbf{\Phi}(\mathbf{x})\right)$.

Optimal Private Linear Transformation. The privacy funnel (183) uses the MI $I\left(s;\mathbf{\Phi}(\mathbf{x})\right)$ 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

$$\varepsilon(\mathbf{F}) := \min_{\mathbf{r}\in\mathbb{R}^{d'}} \mathbb{E}\{(s - \mathbf{r}^T\mathbf{F}\mathbf{x})^2\}$$

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

$$\mathbf{F} = \mathbf{I} - (1/\|\mathbf{c}_{\mathbf{x},s}\|_2^2)\mathbf{c}_{\mathbf{x},s}\mathbf{c}_{\mathbf{x},s}^T. \tag{184}$$

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 sensitive attribute.

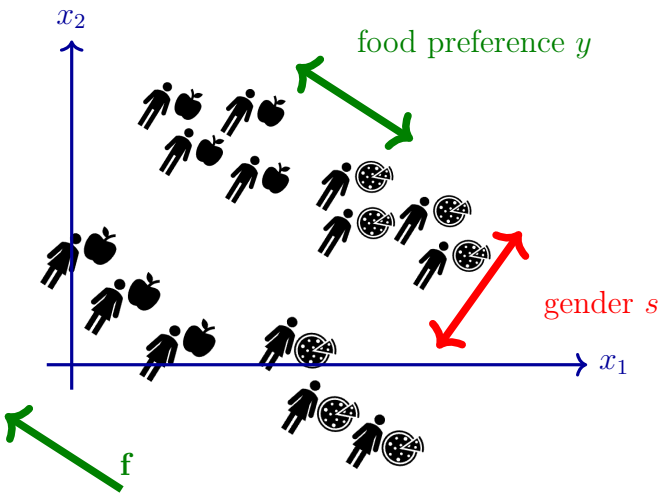


Fig. 9.5. A toy dataset $\mathcal{D}$ whose data points represent customers, each characterized by features $\mathbf{x} = \big(x_1, x_2\big)^T$. These raw features carry information about a sensitive attribute s (gender) and the label y (food preference) of a person. The scatterplot that we can find 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.

9.4 Exercises

9.1. Where is *Alice*? Consider a device, named *Alice*, that implements an asynchronous variant of Algorithm 5 (see (129) and (130)). 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 (129), 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. Use

the weather recordings at FMI stations to construct a training set and a validation set.

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. The pre-processing can be implemented either via randomly selecting a subset of data points in the raw dataset or by adding noise to their features and labels. How well can one predict the latitude and longitude from the time and value of a temperature measurement using a hypothesis $\widehat{\mathbf{h}}$ learned from the perturbed data?

9.5. 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 (184) 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

$$\widehat{\mathbf{X}} := \mathbf{X} - (1/m)\mathbf{1}\mathbf{1}^T\mathbf{X} \ , \ \mathbf{1} := \big(1, \ldots, 1\big)^T \in \mathbb{R}^m.$$

3. extract the sensitive attribute or each data point and store it in the vector

$$\mathbf{s} := \big(\hat{x}_1^{(1)}, \hat{x}_1^{(2)}, \ldots, \hat{x}_1^{(m)}\big)^T.$$

4. compute the empirical cross-covariance vector

$$\hat{\mathbf{c}}_{\mathbf{x},s} := (1/m)\big(\widehat{\mathbf{X}}\big)^T\mathbf{s}$$

The matrix $\mathbf{F}$ obtained from (184) 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 Cybersecurity in FL: Attacks and Defenses

FL, like ML more broadly, fundamentally relies on externally provided data. In most ML applications, the computational device that trains a model rarely has direct access to the raw data points of the training set. Instead, training often proceeds on pre-processed data supplied by external sources or curated databases.

As a case in point, consider an ML application for animal health-care based on monitoring livestock in remote regions. Direct access to raw data points, such as those depicted in Figure 10.1, would require physically visiting distant pastures with specialized measurement equipment such as stomach sensors. Instead, developers typically rely on external databases assembled by researchers or veterinarians who collected the data on-site.

Fig. 10.1. In many ML applications, such as monitoring livestock in remote regions, direct access to raw data points is impractical. ML methods often rely on external databases curated by third parties, introducing potential vulnerabilities.

This reliance on external data is even more pronounced in FL systems. One of the primary purposes of FL is to leverage the information contained in

the local datasets of many interconnected devices, which form a FL network. However, this raises a critical question: *How can we be confident that every device behaves as intended and faithfully follows the agreed-upon FL algorithm?*

Except in the rare case where we have full control over every device in the FL network, it is essential to design FL systems that are robust against potential attacks. Here, an attack refers to the intentional perturbation (or manipulation) of FL system parts.

This chapter is structured as follows: Section 10.1 discusses how such attacks can be carried out by perturbing different components of an FL system. Section 10.2 distinguishes different attack types according to their objectives. Section 10.3 provides some guidance on the design choices for GTVMin-based methods to ensure robustness against attacks.

10.1 A Simple Attack Model

Consider an FL system that implements one of the FL algorithms discussed in Chapter 5. As discussed in Section 5.7, these algorithms share a common form. Many widely-used FL algorithms for parametric local models (with model parameters belonging to $\mathbb{R}^d$) compute and share the results (across the edges of the FL network) of local updates

$$\mathbf{w}^{(i,t+1)} = \operatorname*{arg\,min}_{\mathbf{w}^{(i)} \in \mathbb{R}^d} \left[L_i \left(\mathbf{w}^{(i)} \right) + \sum_{i' \in \mathcal{N}^{(i)}} A_{i,i'} \phi\big(\mathbf{w}^{(i',t)} - \mathbf{w}^{(i,t)} \big) \right]. \tag{185}$$

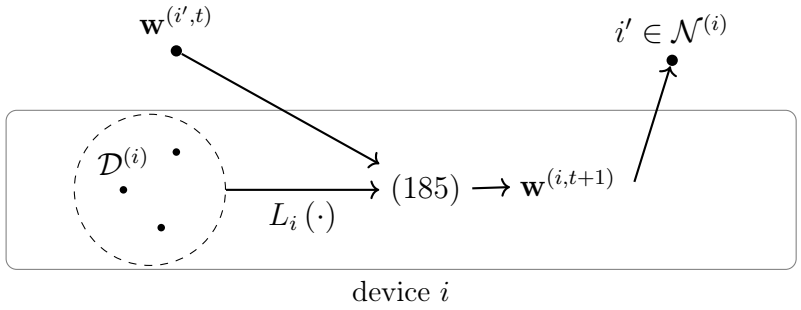


Fig. 10.2. A GTVMin-based FL system from the perspective of a specific device i.

During time-instant t, device i solves (185) in order to obtain new model parameters $\mathbf{w}^{(i,t+1)}$. Carefully note that update (185) involves the model parameters $\mathbf{w}^{(i',t)}$ 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'. Figure 10.2 illustrates the information flow during the local update (185).

From the viewpoint of a specific device i, control is typically limited to the local loss function $L_i\left(\mathbf{w}^{(i)}\right)$, which is often computed as the average loss over the local dataset.[34] In contrast, the model parameters $\mathbf{w}^{(i',t)}$ received from neighbouring devices may be unreliable: they can be intentionally perturbed (or poisoned). In what follows, we describe two major classes of attacks that exploit different parts of the FL system to manipulate the shared model parameters $\mathbf{w}^{(i',t)}$ and thereby influence the local update step (185).

[34]This assumption may not always hold in practice—for instance, the FL application might not be granted full access to the operating system of device i (e.g., a smartphone).

10.1.1 Model Poisoning

If an attacker has control over some of the communication links within an FL system, it can directly manipulate the model parameters shared between nodes. A model poisoning attack on the update (185) replaces the vector $\mathbf{w}^{(i',t)}$, for some $i' \in \mathcal{N}^{(i)}$ with a perturbed vector $\widetilde{\mathbf{w}}^{(i',t)}$. We have already discussed the robustness of the update (185), for specific choices of ϕ, against perturbations in Section 8.2.3.

10.1.2 Data Poisoning

Consider an attacker with access to the local datasets of a subset of devices $\mathcal{W} \subset \mathcal{V}$ in the FL network. By poisoning the local datasets at these compromised nodes, the attacker can manipulate the corresponding local updates (185). Protecting a given device i from such poisoning is non-trivial, especially when the attacker can exploit software vulnerabilities, such as those in smartphone operating systems [156].

The impact of the poisoned updates propagates from nodes $i' \in \mathcal{W}$ through the edges of the FL network during successive update steps. As a result, even nodes whose local datasets remain clean can eventually be affected – provided they are connected to $\mathcal{W}$. In fact, if the FL network $\mathcal{G}$ is connected, the influence of poisoned updates can reach all nodes within a number of steps proportional to the graph's diameter.

Figure 10.3 illustrates this phenomenon in a chain-structured FL network with three nodes $i = 1, 2, 3$ connected by unit-weight edges $\mathcal{E} = \{1, 2\}, \{2, 3\}$. The attacker poisons the local dataset $\mathcal{D}^{(1)}$ at node $i = 1$ at time $t - 1$, resulting in a perturbed update at time t. This perturbation influences node

$i = 2$ at time $t + 1$, and subsequently node $i = 3$ at time $t + 2$. The affected updates are marked by a red star ($*$) in Figure 10.3.

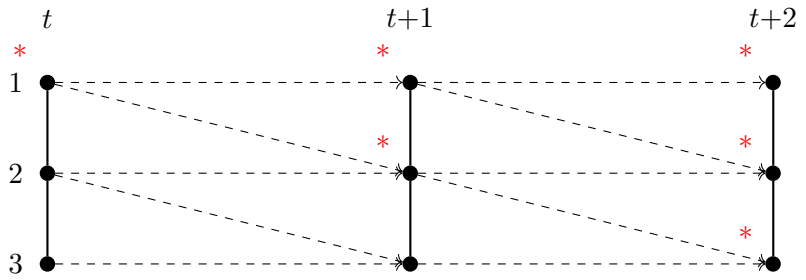


Fig. 10.3. Propagation of the effect of a data poisoning attack that perturbs the update (185) of $i = 1$ during time t.

Data poisoning 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 [157]:

- **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 (51) 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 10.4). Another class of robust loss functions is obtained by including a penalty term (as in regularization).

10.2 Attack Types

Based on their objective, we distinguish the following attacks on FL systems: denial-of-service attacks, backdoor attacks and privacy (or model inversion) attacks [158].

- **Denial-of-service attack.** A denial-of-service attack manipulates $\mathbf{w}^{(i',t)}$ in (185) to nudge the updates $\mathbf{w}^{(i,t+1)}$ towards model parameters $\overline{\mathbf{w}}^{(i)}$ with a large local loss. In other words, the resulting hypothesis $\bar{h}^{(i)}$ delivers poor predictions for the data points in the local dataset $\mathcal{D}^{(i)}$ (see Figure 10.4) [159].

- **Backdoor attack.** This attack manipulates $\mathbf{w}^{(i',t)}$ in (185) to nudge the updates $\mathbf{w}^{(i,t+1)}$ towards model parameters $\widetilde{\mathbf{w}}^{(i)}$ with a small loss on the local dataset but highly irregular predictions for specific feature vectors. In other words, the hypothesis $\tilde{h}^{(i)}$ "behaves well" on $\mathcal{D}^{(i)}$ but delivers pre-specified predictions on a subset $\mathcal{K} \subseteq \mathcal{X}$ of the feature space. We can interpret the subset $\mathcal{K}$ as a backdoor which is opened by any data point with a feature vector $\mathbf{x} \in \mathcal{K}$ (see Figure 10.4) [160].

- **Privacy (or model inversion) attack.** This attack manipulates $\mathbf{w}^{(i',t)}$ in (185) such that the updates $\mathbf{w}^{(i,t+1)}$ maximally leak information about

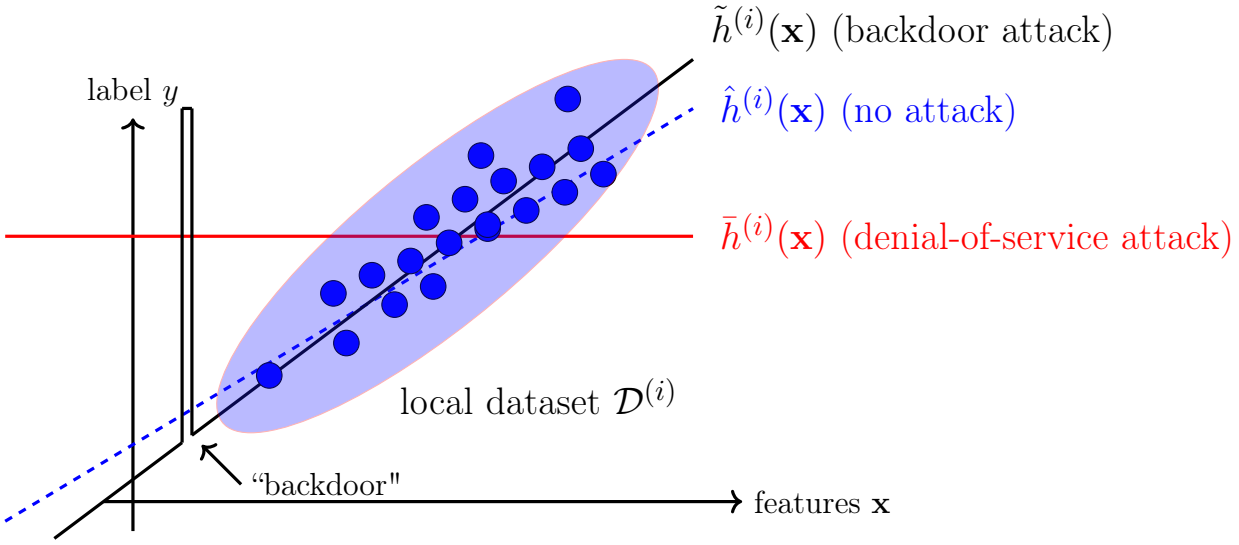


Fig. 10.4. A local dataset $\mathcal{D}^{(i)}$ along with three hypothesis maps learned via iterating (185) under three attack scenarios.

sensitive attributes of data points stored at device i. One approach is to force another device i' to learn a copy of model parameters $\mathbf{w}^{(i)}$ by designing trivial local loss functions and manipulating the structure of the FL network (see Exercise 9.1). Once obtained, the copied model parameters can be probed to reveal private information. A notable class of privacy attacks is model inversion, where an attacker tries to reconstruct feature vectors of data points [161].

10.3 Making FL Robust Against Attacks

We next discuss how to make the update (185) more robust against the attacks discussed in Section 10.2. Our focus will be on GTVMin-based methods using the GTV penalty $\phi(\cdot) = \|\cdot\|_2^2$. For this choice, (185) can be written as (see (171))

$$\mathbf{w}^{(i,t+1)} \in \arg\min_{\mathbf{w}^{(i)} \in \mathbb{R}^d} L_i\left(\mathbf{w}^{(i)}\right) + \alpha d^{(i)} \left\| \mathbf{w}^{(i)} - \widehat{\mathbf{w}}^{(\mathcal{N}^{(i)})} \right\|_2^2, \quad \text{with } \widehat{\mathbf{w}}^{(\mathcal{N}^{(i)})} := (1/d^{(i)}) \sum_{i' \in \mathcal{N}^{(i)}} A_{i,i'} \mathbf{w}^{(i',t)}. \tag{186}$$

Here, we used the weighted node degree $d^{(i)} = \sum_{i' \in \mathcal{N}^{(i)}} A_{i,i'}$ (see (34)).

The update (186) can be attacked via manipulating the model parameters $\mathbf{w}^{(i',t)}$ and, in turn, their average $\widehat{\mathbf{w}}^{(\mathcal{N}^{(i)})}$. Consider an attack that perturbs up to $\eta \cdot |\mathcal{N}^{(i)}|$ of these model parameters (see Figure 10.5). It turns out that an effective defense against these perturbations is to replace the average by [162]

$$(1/d^{(i)}) \sum_{i' \in \mathcal{N}^{(i)}} A_{i,i'} \tau(\mathbf{w}^{(i',t)}), \tag{187}$$

with some generalized threshold (or clipping) function τ. The literature on robust FL has studied different constructions of τ [162, 163]. Intuitively, the threshold function τ should not change clean model parameters $\mathbf{w}^{(i',t)}$ but also limit the impact of perturbed $\mathbf{w}^{(i',t)}$.

For the special case of model dimension, i.e., each local model is parametrized by a single number $w \in \mathbb{R}$, one useful choice for τ in (187) is

$$\tau(w) = \begin{cases} \tau_u & \text{for } w \geq \tau_u \\ w & \text{for } w \in [\tau_l, \tau_u] \\ a & \text{for } w \leq \tau_l. \end{cases} \tag{188}$$

A natural choice for the thresholds τ_l, τ_u is to use order statistic of the values $\mathbf{w}^{(i',t)}$, for $i' \in \mathcal{N}^{(i)}$. In particular, the upper threshold τ_u in (188) is chosen such that it is exceeded by $\mathbf{w}^{(i',t)}$ only for a small number of neighbors $i' \in \mathcal{N}^{(i)}$. The lower threshold τ_l in (188) is chosen analogously (see Figure 10.5). The robustness of using (188) in the averaging step (187) has been studied recently in [162].

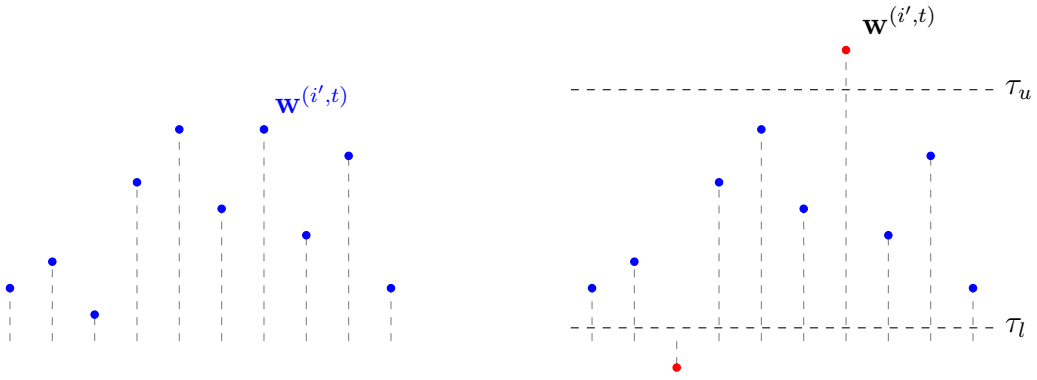
Another important choice for the threshold function τ in (187) is

$$\tau(w) = c \begin{cases} w & \text{if } w \in \mathcal{T} \\ 0 & \text{otherwise,} \end{cases}$$
$$\text{with } c = \frac{|\mathcal{N}^{(i)}|}{|\{i' \in \mathcal{N}^{(i)} : \mathbf{w}^{(i',t)} \in \mathcal{T}\}|}. \tag{189}$$

Inserting (189) into (187) yields the trimmed mean [164]. Indeed, the effect of (189) is that the average (187) is computed over a subset (or trimmed version) $\mathcal{T}$ of $\mathbf{w}^{(i',t)}$, for $i' \in \mathcal{N}^{(i)}$. Different constructions for the subset $\mathcal{T}$ in (189) have been studied in the literature on robust FL [165–167]. One such construction is based on the order statistic of $\mathbf{w}^{(i',t)}$, for $i' \in \mathcal{N}^{(i)}$, by excluding the most extreme values [168].

Note that (189) is defined for scalar model parameters $\mathbf{w}^{(i',t)} \in \mathbb{R}$ (i.e., for local models with dimension $d = 1$). We can generalize (189) to higher dimensions $d > 1$ by applying it separately to each entry $w_1^{(i,t)}, \ldots, w_d^{(i,t)}$ of the model parameters $\mathbf{w}^{(i,t)}$. The robustness of GTVMin-based methods using the averaging step (187) has been studied in [168].

So far, our discussion focused on protecting the update (186) (which is the core step of GTVMin-based methods that use the GTV penalty $\phi(\cdot) = \|\cdot\|_2^2$) against denial-of-service attacks and backdoor attacks. We now discuss how



(a) Original ("clean") model parameters. (b) Poisoned model parameters.

Fig. 10.5. An attack on (185) perturbs (adversarially) a fraction η of the received model parameters $\mathbf{w}^{(i',t)}$.

to protect (186) against privacy attacks.

For a fixed time t, we can ensure a prescribed level of DP by replacing the update (186) with a noisy version

$$\mathbf{w}^{(i,t+1)} + \sigma \cdot \mathbf{n}^{(t)}, \text{ with scaled noise } \sigma \cdot \mathbf{n}^{(t)}. \tag{190}$$

This noisy update (190) is then shared with the neighbors $i' \in \mathcal{N}^{(i)}$. Any (or even each) of these neighbors could be involved in a privacy attack that aims to learn a sensitive attribute of the local dataset $\mathcal{D}^{(i)}$.

The noise term $\mathbf{n}^{(t)}$ in (190) is drawn independently for each time t from a prescribed probability distribution, such as the Laplace distribution or the normal distribution [139]. A key challenge for implementing (190) is to find a useful choice for the noise strength σ. Increasing σ results in stronger privacy protection but typically degrades the accuracy of the trained local models [143]. However, choosing σ too small can result in insufficient privacy

protection.

The minimum value σ required to ensure (ε, δ)-DP (see Definition 1) with prescribed values $\varepsilon, \delta \geq 0$ depends on

- how the shape of the local loss function $L_i(\cdot)$ changes when data points are added to (or removed from) the local dataset $\mathcal{D}^{(i)}$ (see [169]),
- the value of the GTVMin parameter α,
- the number of time instants t during which the update (186) is executed and the noisy result (190) shared with the neighbors [147, 170].

10.4 Exercises

10.1. Model inversion for linear regression. Consider an ERM-based method for training a linear model using plain GD. Assume that the model parameters are initialized to zero, $\mathbf{w}^{(0)} = \mathbf{0} \in \mathbb{R}^d$, and that the training error $\widehat{L}(\mathbf{w})$ is the average squared error loss on a training set,

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

Suppose an attacker can observe the sequence of gradients, $\nabla \widehat{L}\big(\mathbf{w}^{(t)}\big)$ computed during the first few iterations $t = 0, 1, \ldots$. To what extent is it possible, based solely on the observed gradients and the knowledge of zero initialization, to reconstruct the training set?

10.2. 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 learned model parameters $\mathbf{w}^{(1)}$ (at target node $i = 1$) by 20 %.

10.3. A 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).

10.4. 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 $\left(x^{(5)}, y^{(5)}\right)$ 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.

References

[1] W. Rudin, *Real and Complex Analysis*, 3rd ed. New York: McGraw-Hill, 1987.

[2] W. Rudin, *Principles of Mathematical Analysis*, 3rd ed. New York: McGraw-Hill, 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] 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.

[6] 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

[7] 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

[8] 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

[9] S. Cui, A. Hero, Z.-Q. Luo, and J. Moura, Eds., *Big Data over Networks*, 2016.

[10] A. Barabási, N. Gulbahce, and J. Loscalzo, "Network medicine: a network-based approach to human disease," *Nature Reviews Genetics*, vol. 12, no. 56, 2011.

[11] M. E. J. Newman, *Networks: An Introduction.* Oxford Univ. Press, 2010.

[12] 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

[13] 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.

[14] 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.

[15] 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.

[16] 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

[17] D. P. Bertsekas and J. Tsitsiklis, *Parallel and Distributed Computation: Numerical Methods*. Athena Scientific, 2015.

[18] M. van Steen and A. Tanenbaum, *Distributed Systems*, 3rd ed., Feb. 2017, self-published, open publication.

[19] D. Tse and P. Viswanath, *Fundamentals of Wireless Communication*. Cambridge University Press, 2005.

[20] G. Strang, *Computational Science and Engineering*. Wellesley-Cambridge Press, MA, 2007.

[21] G. Strang, *Introduction to Linear Algebra*, 5th ed. Wellesley-Cambridge Press, MA, 2016.

[22] H. H. Bauschke and P. L. Combettes, *Convex Analysis and Monotone Operator Theory in Hilbert Spaces*. New York: Springer, 2011.

[23] A. Jung, *Machine Learning: The Basics*, 1st ed. Springer Singapore, Feb. 2022.

[24] N. Goodall, "Can you program ethics into a self-driving car?" *IEEE Spectrum*, vol. 53, no. 6, pp. 28–58, June 2016.

[25] J. R. Quinlan, "Induction of decision trees," *Machine Learning*, vol. 1, no. 1, pp. 81–106.

[26] B. Schölkopf and A. Smola, *Learning with Kernels: Support Vector Machines, Regularization, Optimization, and Beyond.* Cambridge, MA, USA: MIT Press, Dec. 2002.

[27] A. Juditsky and A. Nemirovski, "First-order methods for nonsmooth convex large-scale optimization, I: General purpose methods," in *Optimization for Machine Learning*, S. Sra, S. Nowozin, and S. Wright, Eds. MIT press, 2011, pp. 121–147.

[28] P. Billingsley, *Probability and Measure*, 3rd ed. New York: Wiley, 1995.

[29] A. Jung, "An RKHS Approach to Estimation with Sparsity Constraints," Ph.D. dissertation, Vienna University of Technology, 2011, available online: arXiv:1311.5768.

[30] E. L. Lehmann and G. Casella, *Theory of Point Estimation*, 2nd ed. New York: Springer, 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: Springer, 2005.

[33] M. Wainwright, *High-Dimensional Statistics: A Non-Asymptotic Viewpoint*. Cambridge: Cambridge University Press, 2019.

[34] 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.

[35] F. Chung, "Spectral graph theory," in *Regional Conference Series in Mathematics*, 1997, no. 92.

[36] D. A. Spielman, "Spectral and algebraic graph theory (incomplete draft)," 2025, version dated April 2, 2025. Available at http://cs-www.cs.yale.edu/homes/spielman/sagt.

[37] T. Hastie, R. Tibshirani, and J. Friedman, *The Elements of Statistical Learning*, ser. Springer Series in Statistics. New York, NY, USA: Springer, 2001.

[38] A. Beck, *First-Order Methods in Optimization*. Philadelphia, PA, USA: SIAM-Society for Industrial and Applied Mathematics, 2017.

[39] N. Parikh and S. Boyd, "Proximal algorithms," *Foundations and Trends in Optimization*, vol. 1, no. 3, pp. 123–231, 2013.

[40] 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.

[41] 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.

[42] 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

[43] 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

[44] 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.

[45] R. T. Rockafellar, *Network Flows and Monotropic Optimization.* Athena Scientific, Jul. 1998.

[46] S. Boyd and L. Vandenberghe, *Convex Optimization*, Cambridge, UK, 2004.

[47] D. P. Bertsekas, *Network Optimization: Continuous and Discrete Models.* Athena Scientific, 1998.

[48] 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

[49] 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

[50] 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

[51] 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

[52] 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.

[53] D. P. Bertsekas, *Convex Optimization Algorithms*. Athena Scientific, 2015.

[54] 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.

[55] H. Robbins and S. Monro, "A stochastic approximation method," *The Annals of Mathematical Statistics*, vol. 22, no. 3, pp. 400–407, 1951.

[56] 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.

[57] 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

[58] H. Bauschke and P. Combettes, *Convex Analysis and Monotone Operator Theory in Hilbert Spaces*, 2nd ed. New York: Springer, 2017.

[59] V. Istrățescu, *Fixed point theory: An Introduction*, ser. Mathematics and its applications ; 7. Dordrecht: Reidel, 1981.

[60] 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

[61] S. Boyd, P. Diaconis, and L. Xiao, "Fastest mixing markov chain on a graph," *SIAM Review*, vol. 46, no. 4, pp. 667–689, 2004.

[62] D. Mills, "Internet time synchronization: the network time protocol," *IEEE Transactions on Communications*, vol. 39, no. 10, pp. 1482–1493, 1991.

[63] J. Hirvonen and J. Suomela. (2023) Distributed algorithms 2020.

[64] R. Diestel, *Graph Theory*. Springer Berlin Heidelberg, 2005.

[65] 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

[66] A. Tanenbaum and D. Wetherall, *Computer Networks*, 5th ed. USA: Prentice Hall Press, 2010.

[67] 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

[68] 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.

[69] J. Liu and C. Zhang, "Distributed learning systems with first-order methods," *Foundations and Trends in Databases*, vol. 9, no. 1, p. 100.

[70] 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.

[71] H. Feyzmahdavian and M. Johansson, "Asynchronous iterations in optimization: new sequence results and sharper algorithmic guarantees," *J. Mach. Learn. Res.*, vol. 24, no. 1, Jan. 2023.

[72] E. K. Ryu and S. Boyd, "A primer on monotone operator methods," *Applied and Computational Mathematics*, vol. 15, no. 1, pp. 3–43, 2016, survey.

[73] Q. Yang, Y. Liu, Y. Cheng, Y. Kang, T. Chen, and H. Yu, *Federated Learning*, 1st ed. Springer, 2022.

[74] 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.

[75] 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

[76] 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

[77] O. Chapelle, B. Schölkopf, and A. Zien, Eds., *Semi-Supervised Learning*. Cambridge, Massachusetts: The MIT Press, 2006.

[78] 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

[79] H. Ludwig and N. Baracaldo, Eds., *Federated Learning: A Comprehensive Overview of Methods and Applications*. Springer, 2022.

[80] 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

[81] 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.

[82] 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

[83] A.-L. Barabási, N. Gulbahce, and J. Loscalzo, "Network medicine: a network-based approach to human disease," *Nature Reviews Genetics*, vol. 12, no. 1, pp. 56–68, 2011. [Online]. Available: https://doi.org/10.1038/nrg2918

[84] A. Jung and N. Tran, "Localized linear regression in networked data," *IEEE Sig. Proc. Lett.*, vol. 26, no. 7, Jul. 2019.

[85] D. Hallac, J. Leskovec, and S. Boyd, "Network lasso: Clustering and optimization in large graphs," in *Proc. SIGKDD*, 2015, pp. 387–396.

[86] 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.

[87] A. Jung, "Learning the conditional independence structure of stationary time series: A multitask learning approach," *IEEE Trans. Signal Processing*, vol. 63, no. 21, Nov. 2015.

[88] V. Kalofolias, "How to learn a graph from smooth signals," in *Proceedings of the 19th International Conference on Artificial Intelligence and Statistics*, ser. Proceedings of Machine Learning Research, A. Gretton and C. C. Robert, Eds., vol. 51. Cadiz, Spain: PMLR, 09–11 May 2016, pp. 920–929.

[89] X. Dong, D. Thanou, M. Rabbat, and P. Frossard, "Learning graphs from data: A signal representation perspective," *IEEE Signal Processing Magazine*, vol. 36, no. 3, pp. 44–63, 2019.

[90] J. Tropp, "An introduction to matrix concentration inequalities," *Found. Trends Mach. Learn.*, May 2015.

[91] 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.

[92] B. Bollobas, W. Fulton, A. Katok, F. Kirwan, and P. Sarnak, *Random graphs*. Cambridge studies in advanced mathematics., 2001, vol. 73.

[93] G. Keiser, *Optical Fiber Communication*, 4th ed. New Delhi: Mc-Graw Hill, 2011.

[94] D. Tse and P. Viswanath, *Fundamentals of wireless communication*. USA: Cambridge University Press, 2005.

[95] D. Spielman, "Spectral graph theory," in *Combinatorial Scientific Computing*, U. Naumann and O. Schenk, Eds. Chapman and Hall/CRC, 2012.

[96] M. Fiedler, "Algebraic connectivity of graphs," *Czechoslovak Mathematical Journal,*, vol. 23, no. 2, pp. 298–305, 1973.

[97] 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.

[98] 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.

[99] S. M. Kay, *Fundamentals of Statistical Signal Processing: Estimation Theory*. Englewood Cliffs, NJ: Prentice Hall, 1993.

[100] 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.

[101] 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.

[102] I. Goodfellow, Y. Bengio, and A. Courville, *Deep Learning*. MIT Press, 2016.

[103] A. Ortega, P. Frossard, J. Kovačević, J. M. F. Moura, and P. Vandergheynst, "Graph signal processing: Overview, challenges, and applications," *Proceedings of the IEEE*, vol. 106, no. 5, pp. 808–828, 2018.

[104] H.-L. E. G. on Artificial Intelligence, "Ethics guidelines for trustworthy AI," European Commission, Tech. Rep., April 2019.

[105] Department of Industry, Science, Energy and Resources, "Australia's AI Ethics Principles," Government of Australia, 2024, accessed: 2024-09-30. [Online]. Available: https://www.industry.gov.au/publications/australias-artificial-intelligence-ethics-framework/australias-ai-ethics-principles

[106] OECD, "Oecd ai principles: Recommendation of the council on artificial intelligence," https://oecd.ai/en/ai-principles, 2019, accessed: 2024-09-30.

[107] Cyberspace Administration of China, "Interim measures for the management of generative artificial intelligence services," https://www.chinalawtranslate.com/en/generative-ai/, 2023, accessed: 2025-05-02.

[108] China Academy of Information and Communications Technology (CAICT), "Artificial intelligence security governance framework," https://www.haynesboone.com/-/media/project/haynesboone/haynesboone/pdfs/alert-pdfs/2024/china-alert---china-publishes-the-ai-security-governance-framework.pdf, 2024, accessed: 2025-05-02.

[109] Ministry of Science and Technology of China, "New generation artificial intelligence ethics code," https://www.chinalawvision.com/2025/01/digital-economy-ai/ai-ethics-overview-china/, 2021, accessed: 2025-05-02.

[110] National Institute of Standards and Technology, "Artificial intelligence risk management framework (ai rmf 1.0)," https://nvlpubs.nist.gov/nistpubs/ai/nist.ai.100-1.pdf, 2023, accessed: 2025-05-02.

[111] White House Office of Science and Technology Policy, "Blueprint for an ai bill of rights," https://bidenwhitehouse.archives.gov/ostp/ai-bill-of-rights/, 2022, accessed: 2025-05-02.

[112] The White House, "Executive order 14110: Safe, secure, and trustworthy development and use of artificial intelligence," https://www.federalregister.gov/documents/2023/11/01/2023-24283/

safe-secure-and-trustworthy-development-and-use-of-artificial-intelligence, 2023, accessed: 2025-05-02.

[113] 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.

[114] 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

[115] P. Mozur, "A genocide incited on facebook, with posts from myanmar's military," *The New York Times*, 2018.

[116] 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.

[117] J. R. Taylor, *An Introduction to Error Analysis: The study of uncertainties in physical measurements*, second edition. ed. Sausalito, Calif: University Science Books, 1997.

[118] 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

[119] K. Pillutla, S. M. Kakade, and Z. Harchaoui, "Robust aggregation for federated learning," *IEEE Transactions on Signal Processing*, vol. 70, pp. 1142–1154, 2022.

[120] H. P. Lopuhaä and P. J. Rousseeuw, "Breakdown points of affine equivariant estimators of multivariate location and covariance matrices," *The Annals of Statistics*, vol. 19, no. 1, pp. 229–248, 1991. [Online]. Available: http://www.jstor.org/stable/2241852

[121] 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.

[122] M. Parter, "Small Cuts and Connectivity Certificates: A Fault Tolerant Approach," in *33rd International Symposium on Distributed Computing (DISC 2019)*, ser. Leibniz International Proceedings in Informatics (LIPIcs), J. Suomela, Ed., vol. 146. Dagstuhl, Germany: Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2019, pp. 30:1–30:16. [Online]. Available: https://drops.dagstuhl.de/entities/document/10.4230/LIPIcs.DISC.2019.30

[123] S. Chechik, M. Langberg, D. Peleg, and L. Roditty, "Fault-tolerant spanners for general graphs," in *Proceedings of the Forty-First Annual ACM Symposium on Theory of Computing*, ser. STOC '09. New York, NY, USA: Association for Computing Machinery, 2009, p. 435–444. [Online]. Available: https://doi.org/10.1145/1536414.1536475

[124] J. Near and D. Darais, "Guidelines for evaluating differential privacy guarantees," National Institute of Standards and Technology, Gaithersburg, MD, Tech. Rep., 2023.

[125] 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

[126] P. Samarati, "Protecting respondents identities in microdata release," *IEEE Transactions on Knowledge and Data Engineering*, vol. 13, no. 6, pp. 1010–1027, 2001.

[127] 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.

[128] U. N. G. Assembly, *The Universal Declaration of Human Rights (UDHR)*, New York, 1948.

[129] 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.

[130] A. Jung and P. Nardelli, "An information-theoretic approach to personalized explainable machine learning," *IEEE Sig. Proc. Lett.*, vol. 27, pp. 825–829, 2020.

[131] 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

[132] 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

[133] 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

[134] A. Abrol and R. Jha, "Power optimization in 5g networks: A step towards green communication," *IEEE Access*, vol. 4, pp. 1355–1374, 2016.

[135] 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

[136] 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.

[137] R. B. Ash, *Probability and Measure Theory*, 2nd ed. New York: Academic Press, 2000.

[138] P. R. Halmos, *Measure Theory*. New York: Springer, 1974.

[139] 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

[140] U. Erlingsson, V. Pihur, and A. Korolova, "Rappor: Randomized aggregatable privacy-preserving ordinal response," in *Proceedings of the 2014 ACM SIGSAC Conference on Computer and Communications Security*, ser. CCS '14. New York, NY, USA: Association for Computing Machinery, 2014, p. 1054–1067. [Online]. Available: https://doi.org/10.1145/2660267.2660348

[141] Apple Machine Learning Research, "Understanding aggregate trends for apple intelligence using differential privacy," https://machinelearning.apple.com/research/differential-privacy-aggregate-trends, April 2025, accessed: 2025-05-20.

[142] J. M. Abowd, "The u.s. census bureau adopts differential privacy," in *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, ser. KDD '18. New York, NY, USA: Association for Computing Machinery, 2018, p. 2867. [Online]. Available: https://doi.org/10.1145/3219819.3226070

[143] J. P. Near, D. Darais, N. Lefkovitz, and G. S. Howarth, "Guidelines for evaluating differential privacy guarantees," National Institute of Standards and Technology, Gaithersburg, MD, NIST

Special Publication NIST SP 800-226, 2025. [Online]. Available: https://doi.org/10.6028/NIST.SP.800-226

[144] 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.

[145] I. Mironov, "Rényi differential privacy," in *2017 IEEE 30th Computer Security Foundations Symposium (CSF)*, 2017, pp. 263–275.

[146] 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.

[147] 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

[148] 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.

[149] M. Abadi, A. Chu, I. Goodfellow, H. B. McMahan, I. Mironov, K. Talwar, and L. Zhang, "Deep learning with differential privacy," ser. CCS '16. New York, NY, USA: Association for Computing Machinery, 2016, p. 308–318. [Online]. Available: https://doi.org/10.1145/2976749.2978318

[150] L. Chua, B. Ghazi, P. Kamath, R. Kumar, P. Manurangsi, A. Sinha, and C. Zhang, "How private are DP-SGD implementations?" in *Proceedings of the 41st International Conference on Machine Learning*, ser. Proceedings of Machine Learning Research, R. Salakhutdinov, Z. Kolter, K. Heller, A. Weller, N. Oliver, J. Scarlett, and F. Berkenkamp, Eds., vol. 235. PMLR, 21–27 Jul 2024, pp. 8904–8918. [Online]. Available: https://proceedings.mlr.press/v235/chua24a.html

[151] 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

[152] 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

[153] 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.

[154] 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

[155] 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.

[156] M. Mohamed, B. Shrestha, and N. Saxena, "Smashed: Sniffing and manipulating android sensor data for offensive purposes," *IEEE Transactions on Information Forensics and Security*, vol. 12, no. 4, pp. 901–913, 2017.

[157] A. Turner, D. Tsipras, and A. Madry, "Clean-label backdoor attacks," 2019. [Online]. Available: https://openreview.net/forum?id=HJg6e2CcK7

[158] A. Vassilev, A. Oprea, A. Fordyce, and H. Anderson, "Adversarial machine learning: A taxonomy and terminology of attacks and mitigations," National Institute of Standards and Technology, Gaithersburg, MD, NIST Artificial Intelligence (AI) Report NIST AI 100-2e2023, 2024. [Online]. Available: https://doi.org/10.6028/NIST.AI.100-2e2023

[159] P. Blanchard, E. M. El Mhamdi, R. Guerraoui, and J. Stainer, "Machine learning with adversaries: Byzantine tolerant gradient descent," in *Advances in Neural Information Processing Systems*, I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and

R. Garnett, Eds., vol. 30. Curran Associates, Inc., 2017. [Online]. Available: https://proceedings.neurips.cc/paper_files/paper/2017/file/f4b9ec30ad9f68f89b29639786cb62ef-Paper.pdf

[160] E. Bagdasaryan, A. Veit, Y. Hua, D. Estrin, and V. Shmatikov, "How to backdoor federated learning," in *Proceedings of the Twenty Third International Conference on Artificial Intelligence and Statistics*, ser. Proceedings of Machine Learning Research, S. Chiappa and R. Calandra, Eds., vol. 108. PMLR, 26–28 Aug 2020, pp. 2938–2948. [Online]. Available: https://proceedings.mlr.press/v108/bagdasaryan20a.html

[161] M. Fredrikson, S. Jha, and T. Ristenpart, "Model inversion attacks that exploit confidence information and basic countermeasures," in *Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security*, ser. CCS '15. New York, NY, USA: Association for Computing Machinery, 2015, p. 1322–1333. [Online]. Available: https://doi.org/10.1145/2810103.2813677

[162] 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.

[163] X. Cao, M. Fang, J. Liu, and N. Gong, "Fltrust: Byzantine-robust federated learning via trust bootstrapping," in *Network and Distributed Systems Security (NDSS) Symposium 2021*, 01 2021.

[164] S. M. Stigler, "The Asymptotic Distribution of the Trimmed Mean,"

The Annals of Statistics, vol. 1, no. 3, pp. 472 – 477, 1973. [Online]. Available: https://doi.org/10.1214/aos/1176342412

[165] S. Shen, S. Tople, and P. Saxena, "Auror: defending against poisoning attacks in collaborative deep learning systems," in *Proceedings of the 32nd Annual Conference on Computer Security Applications*, ser. ACSAC '16. New York, NY, USA: Association for Computing Machinery, 2016, p. 508–519. [Online]. Available: https://doi.org/10.1145/2991079.2991125

[166] N. Wang, Y. Xiao, Y. Chen, Y. Hu, W. Lou, and Y. T. Hou, "Flare: Defending federated learning against model poisoning attacks via latent space representations," in *Proceedings of the 2022 ACM on Asia Conference on Computer and Communications Security*, ser. ASIA CCS '22. New York, NY, USA: Association for Computing Machinery, 2022, p. 946–958. [Online]. Available: https://doi.org/10.1145/3488932.3517395

[167] M. Fang, Z. Zhang, Hairi, P. Khanduri, J. Liu, S. Lu, Y. Liu, and N. Gong, "Byzantine-robust decentralized federated learning," in *Proceedings of the 2024 on ACM SIGSAC Conference on Computer and Communications Security*, ser. CCS '24. New York, NY, USA: Association for Computing Machinery, 2024, p. 2874–2888. [Online]. Available: https://doi.org/10.1145/3658644.3670307

[168] D. Yin, Y. Chen, R. Kannan, and P. Bartlett, "Byzantine-robust distributed learning: Towards optimal statistical rates," in *Proceedings of the 35th International Conference on Machine Learning*, ser. Proceedings of Machine Learning Research, vol. 80. PMLR, 10–15 Jul 2018, pp. 5650–5659. [Online]. Available: https://proceedings.mlr.press/v80/yin18a.html

[169] K. Chaudhuri, C. Monteleoni, and A. Sarwate, "Differentially private empirical risk minimization," *J. Mach. Learn. Res.*, vol. 12, pp. 1069–1109, Mar. 2011.

[170] C. Dwork, G. N. Rothblum, and S. Vadhan, "Boosting and differential privacy," in *2010 IEEE 51st Annual Symposium on Foundations of Computer Science*, 2010, pp. 51–60.