

# Lectures Notes

## CS-E4740 Federated Learning

Dipl.-Ing. Dr.techn. Alexander Jung\*

February 27, 2024

### Abstract

This course discusses theory and algorithms for federated learning (FL). FL exploits similarities between local datasets to train local (or personalized) models collaboratively. Two core mathematical objects of this course are empirical graphs and generalized total variation minimization (GTVMin). We use empirical graphs to store and process local datasets and parameters of local models. GTVMin formulates FL as an instance of regularized empirical risk minimization. As the regularizer, we use quantitative measures for the variation of local models across the edges of the empirical graph. We can obtain practical FL systems by applying distributed optimization methods to solve GTVMin.

---

\*AJ is currently Associate Professor for Machine Learning at Aalto University (Finland). This work has been partially funded by the Academy of Finland (decision numbers 331197, 331197) and the European Union (grant number 952410).

# Contents

<b>1</b>	<b>Lecture - “Welcome and Intro”</b>	<b>1</b>
1.1	Learning Goals . . . . .	1
1.2	Introduction . . . . .	1
1.3	Prerequisites . . . . .	4
1.4	Related Courses . . . . .	5
1.5	Main Goal of the Course . . . . .	6
1.6	Outline of the Course . . . . .	7
1.7	Assignments and Grading . . . . .	9
1.8	Student Project . . . . .	9
1.9	Schedule . . . . .	10
1.10	Ground Rules . . . . .	11
<b>2</b>	<b>Lecture - “ML Basics”</b>	<b>1</b>
2.1	Learning Goals . . . . .	1
2.2	Three Components and a Design Principle . . . . .	1
2.3	Computational Aspects of ERM . . . . .	5
2.4	Statistical Aspects of ERM . . . . .	6
2.5	Validation and Diagnosis of ML . . . . .	8
2.6	Regularization . . . . .	13
2.7	Overview of Coding Assignment . . . . .	16
<b>3</b>	<b>Lecture - “FL Design Principle”</b>	<b>1</b>
3.1	Learning Goals . . . . .	1
3.2	Empirical Graphs and Their Laplacian . . . . .	1
3.3	Generalized Total Variation Minimization . . . . .	7

3.3.1	Computational Aspects of GTVMin . . . . .	9
3.3.2	Statistical Aspects of GTVMin . . . . .	11
3.4	Overview of Coding Assignment . . . . .	15
3.5	Proofs . . . . .	17
3.5.1	Proof of Proposition 3.1 . . . . .	17
<b>4</b>	<b>Lecture - “Gradient Methods”</b>	<b>1</b>
4.1	Learning Goals . . . . .	1
4.2	Gradient Descent . . . . .	2
4.3	Learning Rate . . . . .	4
4.4	When to Stop? . . . . .	6
4.5	Perturbed Gradient Step . . . . .	9
4.6	Handling Constraints - Projected Gradient Descent . . . . .	10
4.7	Generalizing the Gradient Step . . . . .	13
4.8	Overview of Coding Assignment . . . . .	16
<b>5</b>	<b>Lecture - “FL Algorithms”</b>	<b>1</b>
5.1	Learning Goals . . . . .	1
5.2	Gradient Step for GTVMin . . . . .	1
5.3	Message Passing Implementation . . . . .	4
5.4	FedSGD . . . . .	7
5.5	FedAvg . . . . .	9
5.6	Overview of Coding Assignment . . . . .	14
5.7	Proofs . . . . .	16
5.7.1	Proof of Proposition 5.1 . . . . .	16
5.7.2	Proof of Proposition 5.2 . . . . .	17

<b>6</b>	<b>Lecture - “FL Main Flavours”</b>	<b>1</b>
6.1	Learning Goals . . . . .	1
6.2	Single Model FL . . . . .	1
6.3	Clustered FL . . . . .	3
6.4	Horizontal FL . . . . .	6
6.5	Vertical FL . . . . .	8
6.6	Personalized Federated Learning . . . . .	8
6.7	Overview of Coding Assignment . . . . .	11
6.8	Proofs . . . . .	13
6.8.1	Proof of Proposition 6.1 . . . . .	13
<b>7</b>	<b>Lecture - “Graph Learning”</b>	<b>1</b>
7.1	Learning Goals . . . . .	1
7.2	Empirical Graph is a Design Choice . . . . .	2
7.3	Measuring (Dis-)Similarity Between Datasets . . . . .	4
7.4	Graph Learning Methods . . . . .	6
7.5	Overview of Coding Assignment . . . . .	8
<b>8</b>	<b>Lecture - “Trustworthy FL”</b>	<b>1</b>
8.1	Learning Goals . . . . .	1
<b>9</b>	<b>Lecture - “Privacy-Protection in FL”</b>	<b>1</b>
9.1	Learning Goals . . . . .	1
9.2	Assignment . . . . .	1
<b>10</b>	<b>Lecture - “Data and Model Poisoning in FL”</b>	<b>1</b>
10.1	Learning Goals . . . . .	1

10.2 Data Poisoning . . . . .	1
10.3 Model Poisoning . . . . .	1
10.4 Assignment . . . . .	1
<b>Glossary</b>	<b>1</b>

# Lists of Symbols

## Sets and Functions

$a \in \mathcal{A}$	This statement indicates that the object $a$ is an element of the set $\mathcal{A}$ .
$a := b$	This statement defines $a$ to be shorthand for $b$ .
$ \mathcal{A} $	The cardinality (number of elements) of a finite set $\mathcal{A}$ .
$\mathcal{A} \subseteq \mathcal{B}$	$\mathcal{A}$ is a subset of $\mathcal{B}$ .
$\mathcal{A} \subset \mathcal{B}$	$\mathcal{A}$ is a strict subset of $\mathcal{B}$ .
$\mathbb{N}$	The set of natural numbers $1, 2, \dots$
$\mathbb{R}$	The set of real numbers $x$ [1].
$\mathbb{R}_+$	The set of non-negative real numbers $x \geq 0$ .
$\mathbb{R}_{++}$	The set of positive real numbers $x > 0$ .
$h(\cdot): \mathcal{A} \rightarrow \mathcal{B} : a \mapsto h(a)$	A function (map) that accepts any element $a \in \mathcal{A}$ from a set $\mathcal{A}$ as input and delivers a well-defined element $h(a) \in \mathcal{B}$ of a set $\mathcal{B}$ . The set $\mathcal{A}$ is the domain of the function $h$ and the set $\mathcal{B}$ is the codomain of $h$ . ML aims at finding (or learning) a function $h$ (“hypothesis”) that reads in the features $\mathbf{x}$ of a data point and delivers a prediction $h(\mathbf{x})$ for its label $y$ .

$\{0, 1\}$	The binary set that consists of the two real numbers 0 and 1.
$[0, 1]$	The closed interval of real numbers $x$ with $0 \leq x \leq 1$ .
$\operatorname{argmin} f(\mathbf{w})$	The set of minimizers for a real-valued function $f(\mathbf{w})$ .
$\log a$	The logarithm of the positive number $a \in \mathbb{R}_{++}$ .

## Matrices and Vectors

$\mathbf{I}_{l \times d}$	A generalized identity matrix with $l$ rows and $d$ columns. The entries of $\mathbf{I}_{l \times d} \in \mathbb{R}^{l \times d}$ are equal to 1 along the main diagonal and equal to 0 otherwise.
$\mathbf{I}$	A square identity matrix whose shape should be clear from the context.
$\mathbb{R}^d$	The set of vectors $\mathbf{x} = (x_1, \dots, x_d)^T$ consisting of $d$ real-valued entries $x_1, \dots, x_d \in \mathbb{R}$ .
$\mathbf{x} = (x_1, \dots, x_d)^T$	A vector of length $d$ . The $j$ th entry of the vector is denoted $x_j$ .
$\ \mathbf{x}\ _2$	The Euclidean (or “ $\ell_2$ ”) norm of the vector $\mathbf{x} = (x_1, \dots, x_d)^T \in \mathbb{R}^d$ given as $\ \mathbf{x}\ _2 := \sqrt{\sum_{j=1}^d x_j^2}$ .
$\ \mathbf{x}\ $	Some norm of the vector $\mathbf{x} \in \mathbb{R}^d$ [?]. Unless specified otherwise, we mean the Euclidean norm $\ \mathbf{x}\ _2$ .
$\mathbf{x}^T$	The transpose of a vector $\mathbf{x}$ that is considered a single column matrix. The transpose is a single-row matrix $(x_1, \dots, x_d)$ .
$\mathbf{X}^T$	The transpose of a matrix $\mathbf{X} \in \mathbb{R}^{m \times d}$ . A square real-valued matrix $\mathbf{X} \in \mathbb{R}^{m \times m}$ is called symmetric if $\mathbf{X} = \mathbf{X}^T$ .
$\mathbf{0} = (0, \dots, 0)^T$	The vector in $\mathbb{R}^d$ with each entry equal to zero.
$\mathbf{1} = (1, \dots, 1)^T$	The vector in $\mathbb{R}^d$ with each entry equal to one.



$(\mathbf{v}^T, \mathbf{w}^T)^T$	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 columns of $\mathbf{B}$ , $\text{span}\{\mathbf{B}\} = \{\mathbf{B}\mathbf{a} : \mathbf{a} \in \mathbb{R}^b\} \subseteq \mathbb{R}^a$ .
$\mathbb{S}_+^d$	The set of all positive semi-definite (psd) matrices of size $d \times d$ .
$\det(\mathbf{C})$	The determinant of the matrix $\mathbf{C}$ .
$\mathbf{A} \otimes \mathbf{B}$	The Kronecker product of $\mathbf{A}$ and $\mathbf{B}$ [2].

# Probability Theory

$\mathbb{E}_p\{f(\mathbf{z})\}$  The expectation of a function  $f(\mathbf{z})$  of a RV  $\mathbf{z}$  whose probability distribution is  $p(\mathbf{z})$ . If the probability distribution is clear from context we just write  $\mathbb{E}\{f(\mathbf{z})\}$ .

$p(\mathbf{x}, y)$  A (joint) probability distribution of a RV whose realizations are data points with features  $\mathbf{x}$  and label  $y$ .

$p(\mathbf{x}|y)$  A conditional probability distribution of a RV  $\mathbf{x}$  given the value of another RV  $y$  [3, Sec. 3.5].

$p(\mathbf{x}; \mathbf{w})$  A parametrized probability distribution of a RV  $\mathbf{x}$ . The probability distribution depends on a parameter vector  $\mathbf{w}$ . For example,  $p(\mathbf{x}; \mathbf{w})$  could be a multivariate normal distribution with the parameter vector  $\mathbf{w}$  given by the entries of the mean vector  $\mathbb{E}\{\mathbf{x}\}$  and the covariance matrix  $\mathbb{E}\left\{(\mathbf{x} - \mathbb{E}\{\mathbf{x}\})(\mathbf{x} - \mathbb{E}\{\mathbf{x}\})^T\right\}$ .

$\mathcal{N}(\mu, \sigma^2)$  The probability distribution of a scalar normal (“Gaussian”) RV  $x \in \mathbb{R}$  with mean (or expectation)  $\mu = \mathbb{E}\{x\}$  and variance  $\sigma^2 = \mathbb{E}\{(x - \mu)^2\}$ .

$\mathcal{N}(\boldsymbol{\mu}, \mathbf{C})$  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  $\mathbf{C} = \mathbb{E}\{(\mathbf{x} - \boldsymbol{\mu})(\mathbf{x} - \boldsymbol{\mu})^T\}$ .

# Machine Learning

$r$	An index $r = 1, 2, \dots$ , that enumerates data points.
$m$	The number of data points in (the size of) a dataset.
$\mathcal{D}$	A dataset $\mathcal{D} = \{\mathbf{z}^{(1)}, \dots, \mathbf{z}^{(m)}\}$ is a list of individual data points $\mathbf{z}^{(r)}$ , for $r = 1, \dots, m$ .
$d$	Number of features that characterize a data point.
$x_j$	The $j$ th feature of a data point. The first feature of a given data point is denoted $x_1$ , the second feature $x_2$ and so on.
$\mathbf{x}$	The feature vector $\mathbf{x} = (x_1, \dots, x_d)^T$ of a data point whose entries are the individual features of a data point.
$\mathcal{X}$	The feature space $\mathcal{X}$ is the set of all possible values that the features $\mathbf{x}$ of a data point can take on.
$\mathbf{z}$	Beside the symbol $\mathbf{x}$ , we sometimes use $\mathbf{z}$ as another symbol to denote a vector whose entries are features of a data point. We need two different symbols to distinguish between “raw” or “original” and learnt features [4, Ch. 9].
$\mathbf{x}^{(r)}$	The feature vector of the $r$ th data point within a dataset.
$x_j^{(r)}$	The $j$ th feature of the $r$ th data point within a dataset.
$\mathcal{B}$	A mini-batch (subset) of randomly chosen data points.
$B$	The size of (the number of data points in) a mini-batch.

$y$  The label (quantity of interest) of a data point.

$y^{(r)}$  The label of the  $r$ th data point.

$(\mathbf{x}^{(r)}, y^{(r)})$  The features and label of the  $r$ th data point.

The label space  $\mathcal{Y}$  of a ML method consists of all potential label values that a data point can have. We often use label spaces that are larger than the set of different label values arising in a give dataset (e.g., a training set). We refer to ML problems (methods) using a numeric label space, such as  $\mathcal{Y} = \mathbb{R}$  or  $\mathcal{Y} = \mathbb{R}^3$ , as regression problems (methods). ML problems (methods) that use a discrete label space, such as  $\mathcal{Y} = \{0, 1\}$  or  $\mathcal{Y} = \{\text{“cat”}, \text{“dog”}, \text{“mouse”}\}$  are referred to as classification problems (methods).

$\eta$  learning rate (step-size) used by gradient-based methods.

$h(\cdot)$  A hypothesis map that reads in features  $\mathbf{x}$  of a data point and delivers a prediction  $\hat{y} = h(\mathbf{x})$  for its label  $y$ .

$\mathcal{Y}^{\mathcal{X}}$  Given two sets  $\mathcal{X}$  and  $\mathcal{Y}$ , we denote by  $\mathcal{Y}^{\mathcal{X}}$  the set of all possible hypothesis maps  $h : \mathcal{X} \rightarrow \mathcal{Y}$ .

$\mathcal{H}$  A hypothesis space or model used by a ML method. The hypothesis space consists of different hypothesis maps  $h : \mathcal{X} \rightarrow \mathcal{Y}$  between which the ML method has to choose .

$d_{\text{eff}}(\mathcal{H})$  The effective dimension of a hypothesis space  $\mathcal{H}$ .

$B^2$	<p>The squared bias of a learnt hypothesis <math>\hat{h}</math> delivered by a ML algorithm that is fed with data points which are modelled as realizations of RVs. If data is modelled as realizations of RVs, also the delivered hypothesis <math>\hat{h}</math> is the realization of a RV.</p>
$V$	<p>The variance of the (parameters of the) hypothesis delivered by a ML algorithm. If the input data for this algorithm is interpreted as realizations of RVs, so is the delivered hypothesis a realization of a RV.</p>
$L((\mathbf{x}, y), h)$	<p>The loss incurred by predicting the label <math>y</math> of a data point using the prediction <math>\hat{y} = h(\mathbf{x})</math>. The prediction <math>\hat{y}</math> is obtained from evaluating the hypothesis <math>h \in \mathcal{H}</math> for the feature vector <math>\mathbf{x}</math> of the data point.</p>
$E_v$	<p>The validation error of a hypothesis <math>h</math>, which is its average loss incurred over a validation set.</p>
$\hat{L}(h \mathcal{D})$	<p>The empirical risk or average loss incurred by the predictions of hypothesis <math>h</math> for the data points in the dataset <math>\mathcal{D}</math>.</p>
$E_t$	<p>The training error of a hypothesis <math>h</math>, which is its average loss incurred over a training set.</p>
$t$	<p>A discrete-time index <math>t = 0, 1, \dots</math> used to enumerate a sequence to sequential events (“time instants”).</p>
$t$	<p>An index that enumerates learning tasks within a multi-task learning problem.</p>

$\alpha$	A regularization parameter that controls the amount of regularization.
$\lambda_j(\mathbf{Q})$	The $j$ th eigenvalue (sorted either ascending or descending) of a psd matrix $\mathbf{Q}$ . We also use the shorthand $\lambda_j$ if the corresponding matrix is clear from context.
$\sigma(\cdot)$	The activation function used by an artificial neuron within an artificial neural network (ANN).
$\mathcal{R}_{\hat{y}}$	A decision region within a feature space.
$\mathbf{w}$	A parameter vector $\mathbf{w} = (w_1, \dots, w_d)^T$ whose entries are parameters of a model. These parameters could be feature weights in linear maps, the weights in ANNs or the thresholds used for splits in decision trees.
$h^{(\mathbf{w})}(\cdot)$	A hypothesis map that involves tunable model parameters $w_1, \dots, w_d$ , stacked into the vector $\mathbf{w} = (w_1, \dots, w_d)^T$ .
$\nabla f(\mathbf{w})$	The gradient of a differentiable real-valued function $f : \mathbb{R}^d \rightarrow \mathbb{R}$ is the vector $\nabla f(\mathbf{w}) = (\frac{\partial f}{\partial w_1}, \dots, \frac{\partial f}{\partial w_d})^T \in \mathbb{R}^d$ [5, Ch. 9].
$\phi(\cdot)$	A feature map $\phi : \mathcal{X} \rightarrow \mathcal{X}' : \mathbf{x} \mapsto \mathbf{x}' := \phi(\mathbf{x}) \in \mathcal{X}'$ .

## Federated Learning

$\mathcal{G} = (\mathcal{V}, \mathcal{E})$	Empirical graph whose nodes $i \in \mathcal{V}$ carry local datasets and local models.
$i \in \mathcal{V}$	A node in the empirical graph that represents a local dataset and a corresponding local model. It might also be useful to think of node $i$ as a small computer that can collect data and execute computations to train ML models.
$\mathcal{G}^{(\mathcal{C})}$	The induced sub-graph 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 $\mathcal{G}^{(\mathcal{C})}$ .
$\mathcal{D}^{(i)}$	The local dataset $\mathcal{D}^{(i)}$ at node $i \in \mathcal{V}$ of an empirical graph.
$m_i$	The number of data points (sample size) contained in the local dataset $\mathcal{D}^{(i)}$ at node $i \in \mathcal{V}$ .
$\mathcal{N}^{(i)}$	The neighbourhood of the node $i$ in an empirical graph.
$\mathbf{x}^{(i,r)}$	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 $\mathcal{D}^{(i)}$ .

$L^{(d)}(\mathbf{x}, h(\mathbf{x}), h'(\mathbf{x}))$  The loss incurred by a “external” hypothesis  $h'$  on a data point with features  $\mathbf{x}$  and predicted label  $h(\mathbf{x})$  that is obtained from some local hypothesis.

$\text{stack}\{\mathbf{w}^{(i)}\}_{i=1}^n$  The vector  $\left((\mathbf{w}^{(1)})^T, \dots, (\mathbf{w}^{(n)})^T\right)^T \in \mathbb{R}^{dn}$  that is obtained by vertically stacking the local model parameters  $\mathbf{w}^{(i)} \in \mathbb{R}^d$ .



# 1 Lecture - “Welcome and Intro”

February 27, 2024

Welcome to the course CS-E4740 Federated Learning. While the course can be completed fully remote, we strongly encourage you to attend the on-site events for your convenience. Any on-site event will be recorded and made available to students via this YouTube channel.

The basic variant (5 credits) of this course consists of lectures (schedule here) and corresponding coding assignments (schedule here). We test your completion of the coding assignments via quizzes (implemented on the MyCourses page). You can upgrade the course to an extended variant (10 credits) by completing a student project (see Section 1.8).

## 1.1 Learning Goals

This lecture offers

- an introduction of course topic and positioning in wider curricula,
- a discussion of learning goals, assignments and student project,
- an overview of course schedule.

## 1.2 Introduction

We are surrounded by devices, such as smartphones or wearables, generating decentralized collections of local datasets [6–10]. These local datasets typically have an intrinsic network structure that arises from functional constraints (connectivity between devices) or statistical similarities.

For example, the high-precision management of pandemics uses contact networks to relate local datasets generated by patients. Network medicine relates data about diseases via co-morbidity networks [11]. Social science uses notions of acquaintance to relate data collected from be-friended individuals [12]. Another example for network-structured are observations collected at Finnish Meteorological Institute (FMI) weather stations. Each FMI station generates a local dataset which tend have similar statistical properties for nearby stations.

Federated learning (FL) is an umbrella term for distributed optimization techniques to train machine learning (ML) models from decentralized collections of local datasets [13–17]. The idea is to carry out the computations, such as gradient steps (see Lecture 4), arising during ML model training at the location of data generation (such as your smartphone or heart rate sensor). This design philosophy is different from the basic ML workflow, which is first to collect all local datasets at a single location (computer) and then train a single ML model on this pooled data.

It can be beneficial to train different ML models at the locations of actual data generation [18] for several reasons:

- **Privacy.** FL methods are appealing for applications involving sensitive data (such as healthcare) as they do not require the exchange of raw data but only (updates of) model parameters [15,16]. By exchanging only model parameters, FL methods are considered privacy-friendly in the sense of not leaking (too much) sensitive information that is contained in the local datasets (see Lecture 9).
- **Robustness.** By relying on decentralized data and computation, FL

methods offer robustness (to some extent) against hardware failures (such as “stragglers”) and cyber attacks (such data poisoning discussed Lecture 10).

- **Parallel Computing.** Many ML systems are constituted by humans equipped with smartphones. We can interpret a mobile network as a parallel computer which is constituted by smartphones that can communicate via radio links. This parallel computer allows to speed up computational tasks such as the computation of gradients required to train ML models (see Lecture 4).
- **Trading Computation against Communication.** Consider a FL application where local datasets are generated by low-complexity devices at remote locations (think of a wildlife camera) that cannot be easily accessed. The cost of communicating raw local datasets to some central unit (which then trains a single global ML model) might be much higher than the computational cost incurred by using the low-complexity devices to (partially) train ML models [19].
- **Personalization.** FL can be used to train personalized ML models for collections of local datasets, which might be generated by smartphones (and their users) [20]. A key challenge for ensuring personalization is the heterogeneity of local datasets [21,22]. Indeed, the statistical properties of different local datasets might vary significantly such they cannot be well modelled as independent and identically distributed (i.i.d.). Each local dataset induces a separate learning task that consists of learning useful parameter values for a local model. This course discusses FL

methods to train personalized models via combining the information carried in decentralized and heterogeneous data (see Lecture 6).

### 1.3 Prerequisites

The main mathematical structure used to study and design FL algorithms is the Euclidean space  $\mathbb{R}^d$ . We, therefore, expect some familiarity with the algebraic and geometric structure of  $\mathbb{R}^d$ . By algebraic structure, we mean the (real) vector space obtained from the elements (“vectors”) in  $\mathbb{R}^d$ , along with the usual definitions of vector addition and multiplication by scalars in  $\mathbb{R}$  [23, 24]. We will heavily use concepts from linear algebra to represent and manipulate data and ML models.

The metric structure of  $\mathbb{R}^d$  is an excellent analysis tool for the (convergence) behaviour of FL algorithms. In particular, we will study FL algorithms that are obtained as fixed-point iterations of some non-linear operator on  $\mathbb{R}^d$ . These operators are defined by the data (distribution) and ML models used within a FL system. A prime example of such a non-linear operator is the gradient step of gradient-based methods (see Lecture 4). The computational properties (such as convergence speed) of these FL algorithms can then be characterized via the contraction properties of the underlying operator [25].

A main tool for the design the FL algorithms are variants of gradient descent (GD). The common idea of these gradient-based methods is to approximate a differential privacy (DP) function  $f(\mathbf{x})$  locally by a linear function. This local linear approximation is determined by the gradient  $\nabla f(\mathbf{x})$ . We, therefore, expect some familiarity with multivariable calculus [5].

## 1.4 Related Courses

In what follows we briefly explain how CS-E4740 relates to selected courses at Aalto University and University of Helsinki.

- **CS-EJ3211 - Machine Learning with Python.** Teaches the application of basic ML methods using the Python package (library) `scikit-learn` [26]. CS-E4740 couples a network of basic ML methods using regularization techniques to obtain tailored (personalized) ML models for local datasets. This coupling is required to adaptively pool local datasets into sufficiently large training set for the personalized ML model.
- **Data Analysis with Python.** Can be considered as a substitute for CS-EJ3211.
- **CS-E4510 - Distributed Algorithms.** Teaches basic mathematical tools for the study and design of distributed algorithms that are implemented via distributed systems (computers) [27]. FL is enabled by distributed algorithms to train ML models from decentralized data (see Lecture 5).
- **CS-C3240 - Machine Learning (spring 2022 edition).** Teaches basic theory of ML models and methods [4]. CS-E4740 combines the components of basic ML methods, such as data representation and models, with network models. In particular, instead of a single dataset and a single model (such as a decision tree), we will study networks of local datasets and local models.

- **ABL-E2606 - Data Protection.** This course discusses important legal constraints (“laws”), including the European general data protection regulation (GDPR), for the use of data and, in turn, for the design of trustworthy FL methods.
- **MS-C2105 - Introduction to Optimization.** This course teaches basic optimisation theory and how to model applications as (linear, integer, and non-linear) optimization problems. CS-E4740 uses optimization theory and methods to formulate FL problems (see Lecture 3) and design FL methods (see Lecture 5).
- **ELEC-E5424 - Convex Optimization.** This course teaches advanced optimisation theory for the important class of convex optimization problems [28]. Convex optimization theory and methods can be used for the study and design of FL algorithms.

## 1.5 Main Goal of the Course

The overarching goal of the course is to demonstrate how to apply concepts from graph theory and mathematical optimization to analyze and design FL algorithms. Students will learn to formulate a given FL application as an optimization problem over an undirected empirical graph  $\mathcal{G} = (\mathcal{V}, \mathcal{E})$  whose nodes  $i \in \mathcal{V}$  represent individual local datasets. We refer to this graph as the empirical graph of a collection of local datasets (see Lecture 3).

This course uses only undirected empirical graphs with a finite number  $n$  of nodes, which we identify with the first  $n$  positive integers:

$$\mathcal{V} := \{1, \dots, n\}.$$

An edge  $\{i, i'\} \in \mathcal{E}$  in the empirical graph  $\mathcal{G}$  connects two different local datasets if they have similar statistical properties. We quantify the amount of similarity by the positive edge weight  $A_{i,i'} > 0$ .

We can formalize a FL application as an optimization problem associated with an empirical graph,

$$\min_{\mathbf{w}^{(i)}} \sum_{i \in \mathcal{V}} L_i(\mathbf{w}^{(i)}) + \alpha \sum_{\{i,i'\} \in \mathcal{E}} A_{i,i'} d(\mathbf{w}^{(i)}, \mathbf{w}^{(i')}). \quad (1.1)$$

We refer to this problem as GTV minimization (GTVMin) and devote much of the course to its computational and statistical properties. The optimization variables  $\mathbf{w}^{(i)}$  in (1.1) are local model parameters at the nodes  $i \in \mathcal{V}$  of an empirical graph. The objective function in (1.1) consists of two components: The first component is a sum over all nodes of the loss values  $L_i(\mathbf{w}^{(i)})$  incurred by local model parameters at each node  $i$ . The second component is the sum of local model parameters discrepancies  $d(\mathbf{w}^{(i)}, \mathbf{w}^{(i')})$  across the edges  $\{i, i'\}$  of the empirical graph.

## 1.6 Outline of the Course

Our course is roughly divided into three parts:

- **Part I: ML Refresher.** Lecture 2 introduces data, models and loss functions as three main components of ML. This lecture also explains how these components are combined within empirical risk minimization (ERM). We also discuss how regularization of ERM can be achieved via manipulating its three main components. We then explain when and how to solve regularized ERM via simple GD methods in Lecture

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

- **Part II: FL Theory and Methods.** Lecture 3 introduces the empirical graph as our main mathematical structure for representing collections of local datasets and corresponding tailored models. The undirected and weighted edges of the empirical graph represent statistical similarities between local datasets. Lecture 3 also formulates FL as an instance of regularized empirical risk minimization (RERM) which we refer to as GTVMin. GTVMin uses the variation of personalized model parameters across edges in the empirical graph as regularizer. We will see that GTVMin couples the training of tailored (or “personalized”) ML models such that well-connected nodes (clusters) in the empirical graph will obtain similar trained models. Lecture 4 discusses variations of gradient descent as our main algorithmic toolbox for solving GTVMin. Lecture 5 shows how FL algorithms can be obtained in a principled fashion by applying optimization methods, such as gradient-based methods, to GTVMin. We will obtain FL algorithms that can be implemented as iterative message passing methods for the distributed training of tailored (“personalized”) models. Lecture 6 derives some main flavours of FL as special cases of GTVMin. The usefulness of GTVMin crucially depends on the choice for the weighted edges in the empirical graph. Lecture 7 discusses graph learning methods that determine a useful empirical graph via different notions of statistical similarity between



local datasets.

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

## 1.7 Assignments and Grading

The course includes coding assignments that require you to implement concepts from the lecture in a Python notebook. We will use MyCourses quizzes to test your understanding of lecture contents and solutions to the coding assignments. For each quiz, you can earn around 10 points. We will sum up the points collected during the quizzes (no minimum requirements for any individual quiz) and determine your grade according to: **grade 1 for 50-59 points; grade 2 for 60-69; grade 3 for 70-79; grade 4 for 80-89 and top grade 5 for at least 90 points.** Students will be able to review their assignments' grading at the end of the course.

## 1.8 Student Project

You can extend the basic variant (which is worth 5 credits) to 10 credits by completing a student project and peer review. This project requires you to

formulate an application of your choice as a FL problem using the concepts from this course. You then have to solve this FL problem using the FL algorithms taught in this course. The main deliverable will be a project report which must follow the structure indicated in the template. You will then peer-review the reports of your fellow students by answering a detailed questionnaire.

## 1.9 Schedule

The course lectures are held on Mo. and Wed. at 16.15, during 28-Feb-2024 until 30-Apr-2024. You can find the detailed schedule and lecture halls following this link. As the course can be completed fully remote, we will record each lecture and add the recording to the YouTube playlist here.

After each lecture, we release the corresponding assignment at this site. You have few days to work on the assignment before we open the corresponding quiz on the MyCourses page of the course (click me).

The assignments and quizzes are somewhat fast paced to encourage students to actively work on the course. We will also be strict about the deadlines for the quizzes. **However, students have the possibility to cover up for lost quiz points during a review meeting at the end of course.** Moreover, active participation such as contribution to the discussion forum or providing feedback on the lecture notes will be taken into account.

## 1.10 Ground Rules

Note that as a student following this course, you must act according to the Code of Conduct of Aalto University (see [here](#)). Two main ground rules for this course are:

- **BE HONEST.** This course includes many tasks that require independent work, including the coding assignments, the working on student projects and the peer review of student projects. You must not use other's work inappropriately. For example, it is not allowed to copy other's solutions to coding assignments. We will randomly choose students who have to explain their solutions (and corresponding answers to quiz questions).
- **BE RESPECTFUL.** My personal wish is that this course provides a safe space for an enjoyable learning experience. Any form of disrespectful behaviour, including any course-related communication platforms, will be sanctioned rigorously (including reporting to university authorities).

## 2 Lecture - “ML Basics”

February 27, 2024

This lecture covers basic ML techniques that are instrumental for FL. This lecture is significantly more extensive content-wise compared to the following lectures. However, this lecture should be considerably easier to follow than the following lectures as it mainly refreshes pre-requisite knowledge.

### 2.1 Learning Goals

After this lecture, you should

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

### 2.2 Three Components and a Design Principle

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

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

This course will focus mainly on one specific choice for the data points. In particular, we will consider data points that 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., `lat := 60.37788`, `lon := 22.0964`,
- date of the day in format DDMMYYYY, e.g., 01022022
- minimum daytime temperature.

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.

We predict the label by the function value hypothesis  $h(\mathbf{x})$ . The prediction will typically be not perfect, i.e.,  $h(\mathbf{x}) \neq y$ . We measure the prediction error by a loss function such as the squared error loss  $L(\mathbf{z}, h) := (y - h(\mathbf{x}))^2$ . It seems natural to choose (or learn) a hypothesis that incurs minimum average loss (or empirical risk) on a given set of data points  $\mathcal{D} := \{(\mathbf{x}^{(1)}, y^{(1)}), \dots, (\mathbf{x}^{(m)}, y^{(m)})\}$ . This is known as ERM,

$$\hat{h} \in \operatorname{argmin}_{h \in \mathcal{H}} (1/m) \sum_{r=1}^m (y^{(r)} - h(\mathbf{x}^{(r)}))^2 \quad (2.1)$$

As our notation indicates (using the symbol “ $\in$ ” instead of “ $:=$ ”), there might be several different solutions to the optimization problem (2.1). Unless

specified otherwise,  $\hat{h}$  can be used to denote any hypothesis in  $\mathcal{H}$  that has minimum average loss over  $\mathcal{D}$ .

Several important machine learning (ML) methods use a parametrized 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 prominent instance of such a parametrized model is the linear model [4, Sec. 3.1],

$$\mathcal{H}^{(d)} := \{h(\mathbf{x}) := \mathbf{w}^T \mathbf{x}\}. \quad (2.2)$$

Linear regression, for example, determines the parameters of a linear model by minimizing the average squared error loss. For linear regression, ERM becomes an optimization over the parameter space  $\mathbb{R}^d$ ,

$$\hat{\mathbf{w}}^{(\text{LR})} \in \underset{\mathbf{w} \in \mathbb{R}^d}{\operatorname{argmin}} \underbrace{(1/m) \sum_{r=1}^m (y^{(r)} - \mathbf{w}^T \mathbf{x}^{(r)})^2}_{:=f(\mathbf{w})}. \quad (2.3)$$

Note that (2.3) amounts to finding the minimum of a smooth and convex function

$$f(\mathbf{w}) = (1/m) \left[ \mathbf{w}^T \mathbf{X}^T \mathbf{X} \mathbf{w} - 2 \mathbf{y}^T \mathbf{X} \mathbf{w} + \mathbf{y}^T \mathbf{y} \right] \quad (2.4)$$

$$\text{with the feature matrix } \mathbf{X} := (\mathbf{x}^{(1)}, \dots, \mathbf{x}^{(m)})^T \quad (2.5)$$

$$\text{and the label vector } \mathbf{y} := (y^{(1)}, \dots, y^{(m)})^T \text{ of the training set } \mathcal{D}. \quad (2.6)$$

Inserting (2.4) into (2.3) allows to formulate linear regression as

$$\hat{\mathbf{w}}^{(\text{LR})} \in \underset{\mathbf{w} \in \mathbb{R}^d}{\operatorname{argmin}} \mathbf{w}^T \mathbf{Q} \mathbf{w} + \mathbf{w}^T \mathbf{q} \quad (2.7)$$

$$\text{with } \mathbf{Q} := (1/m) \mathbf{X}^T \mathbf{X}, \mathbf{q} := -(2/m) \mathbf{X}^T \mathbf{y}.$$

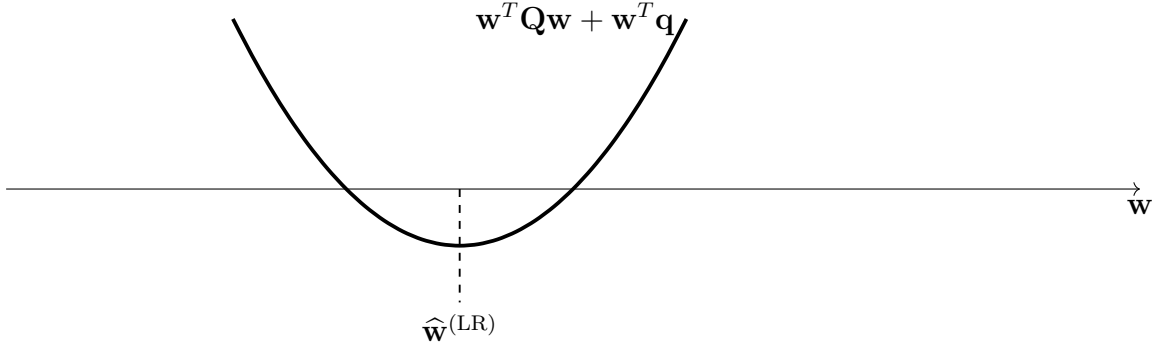


Figure 2.1: ERM (2.1) for linear regression minimizes a convex quadratic function  $\mathbf{w}^T \mathbf{Q} \mathbf{w} + \mathbf{w}^T \mathbf{q}$ .

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

$$\mathbf{Q} = \sum_{j=1}^d \lambda_j(\mathbf{Q}) \mathbf{u}^{(j)} (\mathbf{u}^{(j)})^T. \quad (2.8)$$

The EVD (2.8) involves non-negative eigenvalues

$$0 \leq \lambda_1(\mathbf{Q}) \leq \dots \leq \lambda_d(\mathbf{Q}). \quad (2.9)$$

To train a ML model  $\mathcal{H}$  means to solve ERM (2.1) (or (2.3) for linear regression); the dataset  $\mathcal{D}$  is therefore referred to as a training set. The trained model results in the learnt hypothesis  $\hat{h}$ . We obtain practical ML methods by applying optimization algorithms to solve (2.1). Two key questions studied in ML are:

- **computational aspects** How much computation do we need to solve (2.1) ?
- **statistical aspects** How useful is the solution  $\hat{h}$  to (2.1) in general,

i.e., how accurate is the prediction  $\hat{h}(\mathbf{x})$  for the label  $y$  of an **arbitrary** data point with features  $\mathbf{x}$ ?

## 2.3 Computational Aspects of ERM

ML methods use optimization algorithms to solve (2.1) in order to learn a hypothesis  $\hat{h}$ . Within this course, we use optimization algorithms that are iterative methods: Starting from an initial choice  $h^{(0)}$ , they construct a sequence

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

which are hopefully increasingly accurate approximations to a solution  $\hat{h}$  of (2.1). 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 parametrized model and a smooth loss function, we can solve (2.3) by gradient-based methods: Starting from an initial parameters  $\mathbf{w}^{(0)}$ , we iterate the gradient step:

$$\begin{aligned} \mathbf{w}^{(k)} &:= \mathbf{w}^{(k-1)} - \eta \nabla f(\mathbf{w}^{(k-1)}) \\ &= \mathbf{w}^{(k-1)} + (2\eta/m) \sum_{r=1}^m \mathbf{x}^{(r)} (y^{(r)} - (\mathbf{w}^{(k-1)})^T \mathbf{x}^{(r)}). \end{aligned} \quad (2.10)$$

How much computation do we need for one iteration of (2.10)? How many iterations do we need? We will try to answer the latter question in Lecture 4. The first question can be answered more easily for typical computational infrastructure (e.g., “Python running on a commercial Laptop”). Indeed, a naive evaluation of (2.10) requires around  $m$  arithmetic operations (addition, multiplication).



It is instructive to consider the special case of a linear model which does not use any feature, i.e.,  $h(\mathbf{x}) = w$ . For this extreme case, the ERM (2.3) has a simple closed-form solution:

$$\hat{w} = (1/m) \sum_{r=1}^m x^{(r)}. \quad (2.11)$$

Thus, for this special case of the linear model, solving (2.11) amounts to summing  $m$  numbers  $x^{(1)}, \dots, x^{(m)}$ . It seems reasonable to assume that the amount of computation required to compute (2.11) is proportional to  $m$ .

## 2.4 Statistical Aspects of ERM

We can train a linear model on a given training set as ERM (2.3). But how useful is the solution  $\hat{\mathbf{w}}$  of (2.3) for predicting the labels of data points outside the training set? Consider applying the learnt hypothesis  $h^{(\hat{\mathbf{w}})}$  to an arbitrary data point not contained in the training set. What can we say about the resulting prediction error  $y - h^{(\hat{\mathbf{w}})}(\mathbf{x})$  in general? In other words, how well does  $h^{(\hat{\mathbf{w}})}$  generalize beyond the training set.

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

$$\mathbb{E}\{L((\mathbf{x}, y), h)\}. \quad (2.12)$$

One example for a probability distribution  $p(\mathbf{x}, y)$  is obtained via relating

the label  $y$  with the features  $\mathbf{x}$  of a data point as

$$y = \bar{\mathbf{w}}^T \mathbf{x} + \varepsilon \text{ with } \mathbf{x} \sim \mathcal{N}(\mathbf{0}, \mathbf{I}), \varepsilon \sim \mathcal{N}(0, \sigma^2). \quad (2.13)$$

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

$$\mathbb{E}\{(y - h(\mathbf{x}))^2\} = \|\bar{\mathbf{w}} - \hat{\mathbf{w}}\|^2 + \sigma^2. \quad (2.14)$$

The component  $\sigma^2$  can be interpreted as intrinsic noise level of the label  $y$ . We cannot hope to find a hypothesis with expected loss smaller than this level. The first component of the RHS in (2.14) is the estimation error  $\|\bar{\mathbf{w}} - \hat{\mathbf{w}}\|^2$  of a ML method that reads in the training set and delivers an estimate  $\hat{\mathbf{w}}$  (e.g., via (2.3)) for the parameters of a linear hypothesis.

We next study the estimation error  $\bar{\mathbf{w}} - \hat{\mathbf{w}}$  incurred by the specific estimate  $\hat{\mathbf{w}} = \hat{\mathbf{w}}^{(\text{LR})}$  (2.7) delivered by linear regression methods. To this end, we first use the probabilistic model (2.13) to decompose the label vector  $\mathbf{y}$  in (2.6) as

$$\mathbf{y} = \mathbf{X}\bar{\mathbf{w}} + \mathbf{n}, \text{ with } \mathbf{n} := (\varepsilon^{(1)}, \dots, \varepsilon^{(m)})^T. \quad (2.15)$$

Inserting (2.15) into (2.7) yields

$$\hat{\mathbf{w}}^{(\text{LR})} \in \underset{\mathbf{w} \in \mathbb{R}^d}{\operatorname{argmin}} \mathbf{w}^T \mathbf{Q} \mathbf{w} + \mathbf{w}^T \mathbf{q}' + \mathbf{w}^T \mathbf{e} \quad (2.16)$$

$$\text{with } \mathbf{Q} := (1/m) \mathbf{X}^T \mathbf{X}, \mathbf{q}' := -(2/m) \mathbf{X}^T \mathbf{X} \bar{\mathbf{w}}, \text{ and } \mathbf{e} := -(2/m) \mathbf{X}^T \mathbf{n}. \quad (2.17)$$

Figure 2.2 depicts the objective function of (2.16). It is a perturbation of the convex quadratic function  $\mathbf{w}^T \mathbf{Q} \mathbf{w} + \mathbf{w}^T \mathbf{q}'$ , which is minimized at  $\mathbf{w} = \bar{\mathbf{w}}$ . In general, the minimizer  $\hat{\mathbf{w}}^{(\text{LR})}$  delivered by linear regression is different from  $\bar{\mathbf{w}}$  due the perturbation term  $\mathbf{w}^T \mathbf{e}$  in (2.16). The following result bounds

the deviation between  $\widehat{\mathbf{w}}^{(\text{LR})}$  and  $\overline{\mathbf{w}}$  under the assumption that the matrix  $\mathbf{Q} = (1/m)\mathbf{X}^T\mathbf{X}$  is invertible.<sup>1</sup>

**Proposition 2.1.** *Consider a solution  $\widehat{\mathbf{w}}^{(\text{LR})}$  to the ERM instance (2.16) that is applied to the dataset (2.15). We can then upper bound the difference  $\widehat{\mathbf{w}}^{(\text{LR})} - \overline{\mathbf{w}}$  as*

$$\|\widehat{\mathbf{w}}^{(\text{LR})} - \overline{\mathbf{w}}\|_2^2 \leq \|\mathbf{e}\|_2^2 / \lambda_1^2 \stackrel{(2.17)}{=} (4/m^2) \|\mathbf{X}^T \mathbf{n}\|_2^2 / \lambda_1^2. \quad (2.18)$$

*Proof.* Let us rewrite (2.16) as

$$\widehat{\mathbf{w}}^{(\text{LR})} \in \underset{\mathbf{w} \in \mathbb{R}^d}{\operatorname{argmin}} f(\mathbf{w}) \text{ with } f(\mathbf{w}) := (\mathbf{w} - \overline{\mathbf{w}})^T \mathbf{Q}(\mathbf{w} - \overline{\mathbf{w}}) + \mathbf{e}^T(\mathbf{w} - \overline{\mathbf{w}}). \quad (2.19)$$

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

$$\begin{aligned} f(\mathbf{w}) &\stackrel{(2.19)}{=} (\mathbf{w} - \overline{\mathbf{w}})^T \mathbf{Q}(\mathbf{w} - \overline{\mathbf{w}}) + \mathbf{e}^T(\mathbf{w} - \overline{\mathbf{w}}) \\ &\stackrel{(a)}{\geq} (\mathbf{w} - \overline{\mathbf{w}})^T \mathbf{Q}(\mathbf{w} - \overline{\mathbf{w}}) - \|\mathbf{e}\|_2 \|\mathbf{w} - \overline{\mathbf{w}}\|_2 \\ &\stackrel{(b)}{\geq} \lambda_1 \|\mathbf{w} - \overline{\mathbf{w}}\|_2^2 - \|\mathbf{e}\|_2 \|\mathbf{w} - \overline{\mathbf{w}}\|_2. \end{aligned} \quad (2.20)$$

Step (a) used Cauchy–Schwarz inequality and (b) used the EVD (2.8) of  $\mathbf{Q}$ . Evaluating (2.20) for  $\mathbf{w} = \widehat{\mathbf{w}}$  and combining with  $f(\widehat{\mathbf{w}}) \leq 0$  yields (2.18).  $\square$

## 2.5 Validation and Diagnosis of ML

The above analysis of the generalization error started from postulating a probabilistic model for the generation of data points. However, this probabilistic

---

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

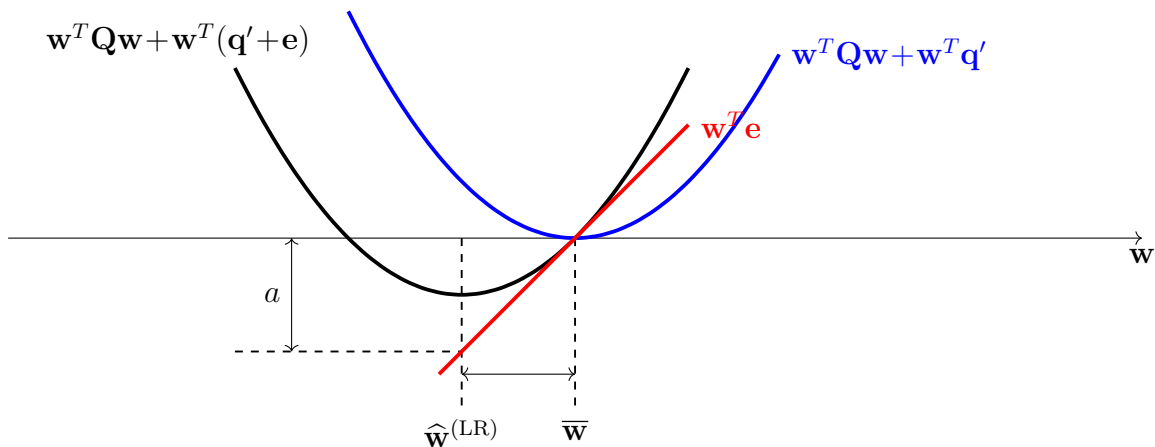


Figure 2.2: The estimation error of linear regression is determined by the effect of the perturbation term  $w^T e$  on the minimizer of the convex quadratic function  $w^T Q w + w^T q'$ .

model might be wrong and the bound (2.18) does not apply. Thus, we might want to use a more data-driven approach for assessing the usefulness of a learnt hypothesis.

Validation methods try to find out if a learnt hypothesis  $\hat{h}$ , does well outside the training set. In its most basic form, model validation amounts to computing the average loss of a learnt hypothesis  $\hat{h}$  on some data points not included in the training set. We refer to these data points as the validation set.

The most basic workflow of ML model training and validation can be summarized as follows:

1. gather a dataset and choose a model  $\mathcal{H}$
2. split dataset into a training set  $\mathcal{D}^{(\text{train})}$  and a validation set  $\mathcal{D}^{(\text{val})}$

3. learn a hypothesis via solving ERM

$$\hat{h} \in \operatorname{argmin}_{h \in \mathcal{H}} \sum_{(\mathbf{x}, y) \in \mathcal{D}^{(\text{train})}} L((\mathbf{x}, y), h) \quad (2.21)$$

4. compute resulting training error

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

5. compute validation error

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

We can diagnose a ERM-based ML method 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 (see Section 2.4).

Given a probabilistic model  $p(\mathbf{x}, y)$ , we can compute the minimum achievable risk (2.12). Indeed, the minimum achievable risk is precisely the expected loss of the Bayes estimator  $\hat{h}(\mathbf{x})$  of the label  $y$ , given the features  $\mathbf{x}$  of a data point. The Bayes estimator  $\hat{h}(\mathbf{x})$  is fully determined by the probability distribution  $p(\mathbf{x}, y)$  [29, 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 end. However, we might still use its statistical properties as a benchmark.

We can also use the performance of human experts as a baseline. If we want to develop a ML method that detects certain type of skin cancers from

images of the skin, a benchmark might be the classification accuracy achieved by experienced dermatologists [30].

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

- $E_t \approx E_v \approx E^{(\text{ref})}$ : The training error is on the same level as the validation error and the baseline. There is not much to improve here 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. This is an indicator for overfitting. We can address overfitting by reducing the effective dimension of the hypothesis space or by increasing the size of the training set. We can reduce the effective dimension of the hypothesis space by using fewer features, a smaller maximum depth of decision trees or fewer layers in an ANN. An alternative to this discrete model selection, we can also reduce the effective dimension of a hypothesis space via regularization.
- $E_t \approx E_v \gg E^{(\text{ref})}$ : The training error is on the same level as the validation error and both are significantly larger than the baseline. Thus, the learnt hypothesis seems to not overfit the training set. However, the training error achieved by the learnt hypothesis is significantly larger than the baseline. There can be several reasons for this to happen. First, it might be that the hypothesis space is too small, i.e., it does not include a hypothesis that provides a good approximation for the

relation between features and 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 (deep ANN (deep net)s). Second, besides the model being too small, another reason for a large training error could be that the optimization algorithm used to solve ERM (2.21) is not working properly (see Lecture 4).

- $E_t \gg E_v$ : The training error is significantly larger than the validation error. The idea of ERM (2.21) is to approximate the risk (2.12) of a hypothesis by its average loss on a training set  $\mathcal{D} = \{(\mathbf{x}^{(r)}, y^{(r)})\}_{r=1}^m$ . The mathematical underpinning for this approximation is the law of large numbers which characterizes the average of (realizations of) i.i.d. RVs. The accuracy of this approximation depends on the validity of two conditions: First, the data points used for computing the average loss “should behave” like realizations of i.i.d. RVs with a common probability distribution. Second, the number of data points used for computing the average loss must be sufficiently large.

Whenever the training set or validation set differs significantly from realizations of i.i.d. RVs, the interpretation (and comparison) of the training error and the validation error of a learnt hypothesis becomes more difficult. As an extreme case, the validation set might consist of data points for which every hypothesis incurs small average loss (see Figure 2.3). Here, we might try to increase the size of the validation set by collecting more labeled data points or by using data augmentation (see Section 2.6). If the size of both training set and validation set

is large but we still obtain  $E_t \gg E_v$ , one should verify if data points in these sets conform to the i.i.d. assumption. There are principled statistical test for the validity of the i.i.d. assumption for a given dataset (see [31] and references therein).

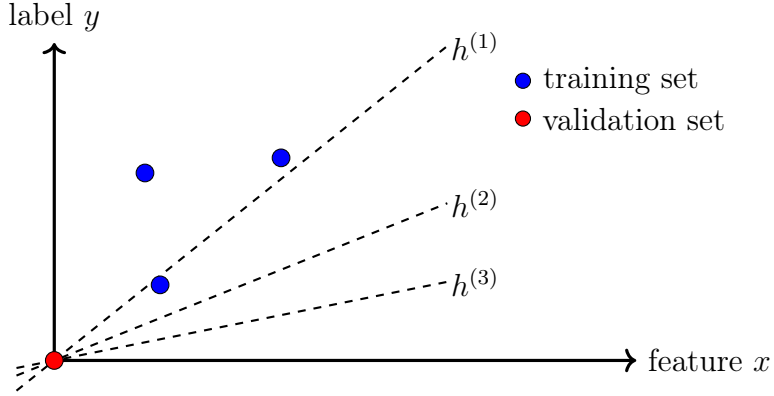


Figure 2.3: Example for an unlucky split into training set and validation set for the model  $\mathcal{H} := \{h^{(1)}, h^{(2)}, h^{(3)}\}$ .

## 2.6 Regularization

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

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



- collect more data points, possibly via data augmentation (see Fig. 2.4),
- add penalty term  $\alpha \mathcal{R}\{h\}$  to average loss in ERM (2.1) (see Fig. 2.4),
- shrink the hypothesis space, e.g., by adding constraints on the model parameters such as  $\|\mathbf{w}\|_2 \leq 10$ .

[4, Ch. 7] discusses the equivalence between these three perspectives on regularization in somewhat more detail.

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

$$\hat{\mathbf{w}}^{(\alpha)} \in \operatorname{argmin}_{\mathbf{w} \in \mathbb{R}^d} \left[ (1/m) \sum_{r=1}^m (y^{(r)} - \mathbf{w}^T \mathbf{x}^{(r)})^2 + \alpha \|\mathbf{w}\|_2^2 \right]. \quad (2.22)$$

The objective function in (2.22) 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 sufficient large number of i.i.d. realizations of

$$(\mathbf{x} + \mathbf{n}, y) \text{ with } \mathbf{n} \sim \mathcal{N}(\mathbf{0}, \alpha \mathbf{I}). \quad (2.23)$$

Thus, ridge regression (2.22) is equivalent to linear regression applied to an augmented variant  $\mathcal{D}'$  of  $\mathcal{D}$ . 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 is obtained by adding a i.i.d. realization  $\mathbf{n}$  of a zero-mean Gaussian noise with covariance matrix  $\alpha \mathbf{I}$  to the features  $\mathbf{x}$  (see (2.23)). The label of each copy is equal to  $y$ , i.e., the label is not perturbed.

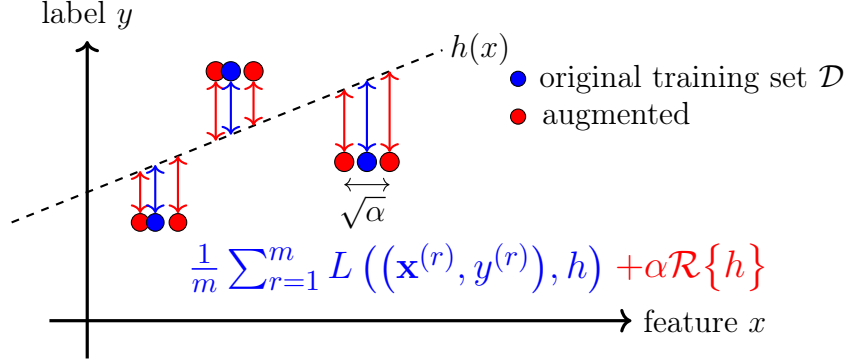


Figure 2.4: Equivalence between data augmentation and loss penalization.

To study the computational aspects of ridge regression, let us rewrite (2.22) as

$$\begin{aligned} \hat{\mathbf{w}}^{(\alpha)} &\in \operatorname{argmin}_{\mathbf{w} \in \mathbb{R}^d} \mathbf{w}^T \mathbf{Q} \mathbf{w} + \mathbf{w}^T \mathbf{q} \\ \text{with } \mathbf{Q} &:= (1/m) \mathbf{X}^T \mathbf{X} + \alpha \mathbf{I}, \mathbf{q} := (-2/m) \mathbf{X}^T \mathbf{y}. \end{aligned} \quad (2.24)$$

Thus, like linear regression (2.7), also ridge regression minimizes a convex quadratic function. A main difference between linear regression (2.7) and ridge regression (for  $\alpha > 0$ ) is that the matrix  $\mathbf{Q}$  in (2.24) is guaranteed to be invertible for any training set  $\mathcal{D}$ . In contrast, the matrix  $\mathbf{Q}$  in (2.7) for linear regression might be singular for some training sets.<sup>2</sup>

The statistical aspects of the solutions to (2.24) (i.e., the parameters learnt by ridge regression) crucially depends on the value of  $\alpha$ . 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

---

<sup>2</sup>Consider the extreme case where all features of each data point in the training set  $\mathcal{D}$  are zero.

the training error and validation error of the hypothesis  $h(\mathbf{x}) = (\widehat{\mathbf{w}}^{(\alpha)})^T \mathbf{x}$  learnt by ridge regression with different values of  $\alpha$ .

## 2.7 Overview of Coding Assignment

**Python Notebook.** `MLBasics_CodingAssignment.ipynb`

**Data File.** `Assignment_MLBasicsData.csv`

**Description.** The coding assignment revolves around weather data collected by the FMI and stored in a csv file. This file contains temperature measurements at different locations in Finland.

Each temperature measurement is a data point, characterized by  $d = 7$  features  $\mathbf{x} = (x_1, \dots, x_7)^T$  and a label  $y$  which is the temperature measurement itself. The features are (normalized) values of the latitude and longitude of the FMI station as well as the (normalized) year, month, day, hour, minute during which this measurements has been taken. Your tasks include:

- Generate `numpy` arrays  $\mathbf{X}, \mathbf{y}$ , whose  $r$ -th row hold the features  $\mathbf{x}^{(r)}$  and label  $y^{(r)}$ , respectively, of the  $r$ -th data point in the csv file.
- Split the dataset into a training set and validation set. The size of the training set should be 100.
- Train a linear model, using the `LinearRegression` class of the `scikit-learn` package, on the training set and determine the resulting training error and validation error
- Train and validate a linear model using feature augmentation via polynomial combinations (see `PolynomialFeatures` class). Try out different

different choices of the maximal degree of these polynomial combinations.

- Using a fixed value for the polynomial degree for the feature augmentation step, train and validate a linear model using ridge regression (2.22) via the `Ridge` class.
- **CAUTION!** The input parameter `alpha` of the `Ridge` class has a different meaning than  $\alpha$  in (2.22) In particular, the parameter  $\alpha$  in (2.22) corresponds to using the value  $m\alpha$  for `alpha`.
- Determine the resulting training error and validation error or the model parameters obtained for different values of  $\alpha$  in (2.22).

## 3 Lecture - “FL Design Principle”

February 27, 2024

Lecture 2 reviewed ML methods that use numeric arrays to store data point and model parameters. We have also discussed ERM as a design principle or practical ML systems. This lecture will extend these concepts to FL applications. Section 3.2 introduces empirical graphs to store collections of local datasets and corresponding parameters of local models. Section 3.3 presents our main design principle for FL systems. This principle uses the variation of local model parameters across the edges of an empirical graph for the coupling (or regularization) of the individual local models.

### 3.1 Learning Goals

After this lecture, you should

- be familiar with the concept of an empirical graph,
- know how connectivity is related to spectrum of Laplacian matrix,
- know some measures for the variation of local models,
- be familiar with the concept of GTVMin.

### 3.2 Empirical Graphs and Their Laplacian

Consider a FL application that involves a collection of local datasets  $\mathcal{D}^{(1)}, \dots, \mathcal{D}^{(2)}$ . Our goal is to train a personalized model  $\mathcal{H}^{(i)}$  for each local dataset  $\mathcal{D}^{(i)}$ , with  $i = 1, \dots, n$ . We represent such a collection of local datasets and (personal)

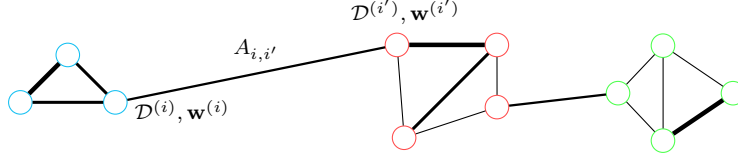


Figure 3.1: Example of an empirical graph whose nodes  $i \in \mathcal{V}$  carry local datasets  $\mathcal{D}^{(i)}$  and local models that are parametrized by local model parameters  $\mathbf{w}^{(i)}$ .

local models, along with their relations, by an empirical graph. Figure 3.1 depicts an example for an empirical graph.

We model the local datasets generated in a FL application using an empirical graph. The empirical graph is an undirected weighted graph  $\mathcal{G} = (\mathcal{V}, \mathcal{E})$  with nodes  $\mathcal{V} := \{1, \dots, n\}$ . Each node  $i \in \mathcal{V}$  of the empirical graph  $\mathcal{G}$  carries a local dataset

$$\mathcal{D}^{(i)} := \{(\mathbf{x}^{(i,1)}, y^{(i,1)}), \dots, (\mathbf{x}^{(i,m_i)}, y^{(i,m_i)})\}. \quad (3.1)$$

Here,  $\mathbf{x}^{(i,r)}$  and  $y^{(i,r)}$  denote, respectively, the features and the label of the  $r$ th data point in the local dataset  $\mathcal{D}^{(i)}$ . Note that the size  $m_i$  of the local dataset might 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,

$$\mathbf{X}^{(i)} := (\mathbf{x}^{(i,1)}, \dots, \mathbf{x}^{(i,m_i)})^T, \text{ and } \mathbf{y} := (y^{(1)}, \dots, y^{(m_i)})^T. \quad (3.2)$$

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 its local dataset  $\mathcal{D}^{(i)}$ , each node  $i \in \mathcal{G}$  also carries a local model  $\mathcal{H}^{(i)}$ . Our focus is on local models that are parametrized by local model parameters  $\mathbf{w}^{(i)} \in \mathbb{R}^d$ , for  $i = 1, \dots, n$ . The usefulness of a specific choice for the local model parameter  $\mathbf{w}^{(i)}$  is measured by a local loss function  $L_i(\mathbf{w}^{(i)})$ , for  $i = 1, \dots, n$ .

An undirected edge  $\{i, i'\} \in \mathcal{E}$  between two different nodes  $i, i' \in \mathcal{V}$  couples the training of the corresponding local models  $\mathcal{H}^{(i)}, \mathcal{H}^{(i')}$ . We quantify the strength of this coupling by a positive edge weight  $A_{i,i'} > 0$ . The coupling will be implemented by penalizing the discrepancy between local model parameters  $\mathbf{w}^{(i)}$  and  $\mathbf{w}^{(i')}$  (see Section 3.3).

There are different ways to measure the discrepancy between the local model parameters  $\mathbf{w}^{(i)}, \mathbf{w}^{(i')}$  at connected nodes  $\{i, i'\} \in \mathcal{E}$ . We could use some cost or penalty function  $\phi(\mathbf{w}^{(i)} - \mathbf{w}^{(i')})$  that satisfies basic requirements such as being a (semi-)norm [32].

The choice of penalty  $\phi(\cdot)$  has crucial impact on the computational and statistical properties of the FL methods developed in Section 3.3. Unless noted otherwise, we use the choice  $\phi(\cdot) := \|\cdot\|_2^2$ , i.e., we measure this discrepancy by the squared Euclidean distance  $\|\mathbf{w}^{(i)} - \mathbf{w}^{(i')}\|_2^2$ . Summing the discrepancies (weighted by the edge weights) over all edges in the empirical graph yields the total variation of local model parameters

$$\sum_{i, i' \in \mathcal{V}} A_{i, i'} \left\| \mathbf{w}^{(i)} - \mathbf{w}^{(i')} \right\|_2^2. \quad (3.3)$$

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

$$d^{(i)} := \sum_{i' \in \mathcal{N}^{(i)}} A_{i, i'}. \quad (3.4)$$

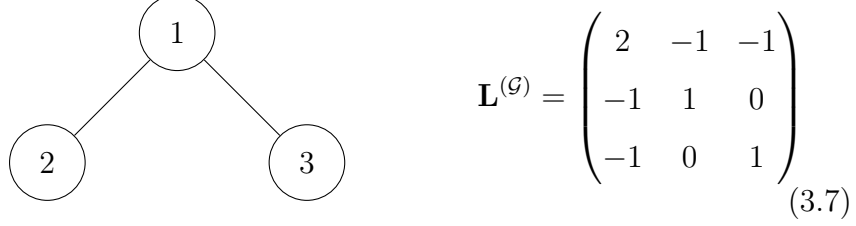


Figure 3.2: Left: Example for an empirical graph  $\mathcal{G}$  with three nodes  $i = 1, 2, 3$ . Right: Laplacian matrix  $\mathbf{L} \in \mathbb{R}^{3 \times 3}$  of  $\mathcal{G}$ .

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

$$d_{\max} := \max_{i \in \mathcal{V}} d^{(i)} \stackrel{(3.4)}{=} \max_{i \in \mathcal{V}} \sum_{i' \in \mathcal{N}^{(i)}} A_{i, i'}. \quad (3.5)$$

Besides the (maximum) node degrees, we can also analyze the connectivity of  $\mathcal{G}$  via the eigenvalues and eigenvectors of its Laplacian matrix  $\mathbf{L}^{(\mathcal{G})} \in \mathbb{R}^{n \times n}$ . The Laplacian matrix is defined element-wise as (see Fig. 3.2)

$$L_{i, i'}^{(\mathcal{G})} := \begin{cases} -A_{i, i'} & \text{for } i \neq i', \{i, i'\} \in \mathcal{E} \\ \sum_{i'' \neq i} A_{i, i''} & \text{for } i = i' \\ 0 & \text{else.} \end{cases} \quad (3.6)$$

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

$$\begin{aligned} \mathbf{w}^T (\mathbf{L} \otimes \mathbf{I}) \mathbf{w} &= \sum_{\{i, i'\} \in \mathcal{E}} A_{i, i'} \left\| \mathbf{w}^{(i)} - \mathbf{w}^{(i')} \right\|_2^2 \\ \text{for any } \mathbf{w} &:= \underbrace{\left( (\mathbf{w}^{(1)})^T, \dots, (\mathbf{w}^{(n)})^T \right)^T}_{=:\text{stack}\{\mathbf{w}^{(i)}\}_{i=1}^n}. \end{aligned} \quad (3.8)$$



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

$$\mathbf{L}^{(\mathcal{G})} = \sum_{j=1}^n \lambda_j \mathbf{u}^{(j)} (\mathbf{u}^{(j)})^T, \quad (3.9)$$

with increasingly ordered eigenvalues

$$0 = \lambda_1(\mathbf{L}^{(\mathcal{G})}) \leq \lambda_2(\mathbf{L}^{(\mathcal{G})}) \leq \dots \leq \lambda_n(\mathbf{L}^{(\mathcal{G})}). \quad (3.10)$$

According to (3.8), we can measure the total variation of local model parameters by stacking them into a single vector  $\mathbf{w} \in \mathbb{R}^{nd}$  and computing the quadratic form  $\mathbf{w}^T (\mathbf{L}^{(\mathcal{G})} \otimes \mathbf{I}_{d \times d}) \mathbf{w}$ .

One immediate consequence of (3.8) is that any collection of identical local model parameters,  $\mathbf{w}^{(i)} = \mathbf{w}^{(i')}$  results in an eigenvector

$$\mathbf{c} = \left( (\mathbf{w}^{(1)})^T, \dots, (\mathbf{w}^{(n)})^T \right)^T. \quad (3.11)$$

with corresponding eigenvalue  $\lambda_1 = 0$  (see (3.10)). Thus, the Laplacian matrix of an empirical graph is always singular (non-invertible).

The second eigenvalue  $\lambda_2$  of the Laplacian matrix provides a great deal of information about the connectivity structure of  $\mathcal{G}$ . Consider the case  $\lambda_2 = 0$ , i.e., besides  $\mathbf{c}$  there is another eigenvector (not belonging to the  $\{a\mathbf{c} : a \in \mathbb{R}\}$ ) with zero eigenvalue. 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.

On the other hand, if  $\lambda_2 > 0$  then  $\mathcal{G}$  is connected. Moreover, the larger the value of  $\lambda_2$ , the stronger the connectivity between the nodes in  $\mathcal{G}$ . We next show how to make this vague statement more precise via the identity (3.8).

The total variation on the RHS (3.8) is a measure for the connectivity. Indeed, if we assume that the local model parameters  $\mathbf{w}^{(i)}$  are different, then adding an edge will increase the total variation. We can lower bound the total variation as [33]

$$\sum_{\{i,i'\} \in \mathcal{E}} A_{i,i'} \left\| \mathbf{w}^{(i)} - \mathbf{w}^{(i')} \right\|_2^2 \geq \lambda_2 \sum_{i=1}^n \left\| \mathbf{w}^{(i)} - \text{avg}\{\mathbf{w}^{(i)}\} \right\|_2^2. \quad (3.12)$$

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 (3.12) follows from (3.8) and the Courant–Fischer–Weyl min-max characterization [34, Thm. 8.1.2.] for the eigenvalues of the matrix  $\mathbf{L} \otimes \mathbf{I}$ .

The quantity  $\sum_{i=1}^n \left\| \mathbf{w}^{(i)} - \text{avg}\{\mathbf{w}^{(i)}\} \right\|_2^2$  on the RHS of (3.12) has an interesting geometric interpretation: It is the squared Euclidean norm of the projection of the stacked local model parameters  $\mathbf{w} := \left( (\mathbf{w}^{(1)})^T, \dots, (\mathbf{w}^{(n)})^T \right)^T$  on the orthogonal complement of the subspace

$$\left\{ \mathbf{1} \otimes \mathbf{a} : \mathbf{a} \in \mathbb{R}^d \right\} = \left\{ (\mathbf{a}^T, \dots, \mathbf{a}^T)^T, \text{ for some } \mathbf{a} \in \mathbb{R}^d \right\} \subseteq \mathbb{R}^{dn}. \quad (3.13)$$

It might be convenient to replace a given empirical graph  $\mathcal{G}$  with an equivalent fully connected empirical graph  $\mathcal{G}'$  (see Figure 3.3). The graph  $\mathcal{G}'$  has an edge between each pair of different nodes  $i, i'$ ,

$$\mathcal{E}' = \left\{ \{i, i'\} \text{ , with some } i, i' \in \mathcal{V}, i \neq i' \right\}.$$

The edge weights are chosen  $A'_{i,i'} = A_{i,i'}$  for any edge  $\{i, i'\} \in \mathcal{E}$  and  $A'_{i,i'} = 0$  otherwise.

Note that the undirected edges  $\mathcal{E}$  of an empirical graph encode a symmetric notion of similarity between local datasets: If the local dataset  $\mathcal{D}^{(i)}$  at node  $i$  is similar to the local dataset  $\mathcal{D}^{(i')}$  at node  $i'$ , i.e.,  $\{i, i'\} \in \mathcal{E}$ , then also the local dataset  $\mathcal{D}^{(i')}$  is similar to the local dataset  $\mathcal{D}^{(i)}$ .



Figure 3.3: Left: Some empirical graph  $\mathcal{G}$  consisting of  $n = 4$  nodes. Right: Equivalent fully connected empirical graph  $\mathcal{G}'$  with the same nodes and non-zero edge weights  $A'_{i,i'} = A_{i,i'}$  and  $A'_{i,i'} = 0$  for  $\{i, i'\} \notin \mathcal{E}$ .

### 3.3 Generalized Total Variation Minimization

Consider data with empirical graph  $\mathcal{G}$  whose nodes  $i \in \mathcal{V}$  carry local datasets  $\mathcal{D}^{(i)}$  and local model parametrized by the vector  $\mathbf{w}^{(i)}$ . We learn these local model parameters by minimizing their local loss and at the same time enforce a small total variation and, in turn, similar local model parameters at well-connected nodes. To optimally balance local loss and total variation, we solve generalized total variation (GTV) minimization,

$$\{\widehat{\mathbf{w}}^{(i)}\}_{i=1}^n \in \underset{\text{rank}\{\mathbf{w}^{(i)}\}_{i=1}^n}{\text{argmin}} \sum_{i \in \mathcal{V}} L_i(\mathbf{w}^{(i)}) + \alpha \sum_{i, i' \in \mathcal{V}} A_{i, i'} \left\| \mathbf{w}^{(i)} - \mathbf{w}^{(i')} \right\|_2^2 \quad (\text{GTVMin}). \quad (3.14)$$

Note that GTVMin is an instance of RERM: The regularizer is the total variation of local model parameters over weighted edges  $A_{i, i'}$  of the empirical graph. Clearly, the empirical graph is an important design choice for 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 empirical graph. 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). Lecture 7 will also discuss empirical graph learning methods that determine the edge weights  $A_{i,i'}$  in a fully data-driven fashion.

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 empirical graph, we want to learn the parameters  $\mathbf{w}^{(i)}$  of a linear hypothesis  $h^{(i)}(\mathbf{x}) := (\mathbf{w}^{(i)})^T \mathbf{x}$ . We measure the quality of the parameters via the average squared error loss

$$\begin{aligned} L_i(\mathbf{w}^{(i)}) &:= (1/m_i) \sum_{r=1}^{m_i} \left( y^{(i,r)} - (\mathbf{w}^{(i)})^T \mathbf{x}^{(i,r)} \right)^2 \\ &\stackrel{(3.2)}{=} (1/m_i) \|\mathbf{y}^{(i)} - \mathbf{X}^{(i)} \mathbf{w}^{(i)}\|_2^2. \end{aligned} \quad (3.15)$$

Inserting (3.15) into (3.14), yields the following instance of GTVMin to train local linear models,

$$\{\widehat{\mathbf{w}}^{(i)}\} \in \operatorname{argmin}_{\{\mathbf{w}^{(i)}\}} \sum_{i \in \mathcal{V}} (1/m_i) \|\mathbf{y}^{(i)} - \mathbf{X}^{(i)} \mathbf{w}^{(i)}\|_2^2 + \alpha \sum_{i,i' \in \mathcal{V}} A_{i,i'} \|\mathbf{w}^{(i)} - \mathbf{w}^{(i')}\|_2^2. \quad (3.16)$$

The identity (3.8) allows to rewrite the GTVMin instance (3.16) as

$$\{\widehat{\mathbf{w}}^{(i)}\}_{i=1}^n \in \operatorname{argmin}_{\mathbf{w} = \operatorname{stack}\{\mathbf{w}^{(i)}\}_{i=1}^n} \sum_{i \in \mathcal{V}} (1/m_i) \|\mathbf{y}^{(i)} - \mathbf{X}^{(i)} \mathbf{w}^{(i)}\|_2^2 + \alpha \mathbf{w}^T \mathbf{L}^{(\mathcal{G})} \mathbf{w}. \quad (3.17)$$

Let us rewrite the objective function in (3.17) as

$$\mathbf{w}^T \left( \begin{pmatrix} \mathbf{Q}^{(1)} & 0 & \dots & 0 \\ 0 & \mathbf{Q}^{(2)} & \dots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \dots & \mathbf{Q}^{(n)} \end{pmatrix} + \alpha \mathbf{L}^{(\mathcal{G})} \otimes \mathbf{I} \right) \mathbf{w} + ((\mathbf{q}^{(1)})^T, \dots, (\mathbf{q}^{(n)})^T) \mathbf{w} \quad (3.18)$$

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

Thus, like linear regression (2.7) and ridge regression (2.24), also GTVMin (3.17) (for local linear models  $\mathcal{H}^{(i)}$ ) minimizes a convex quadratic function,

$$\{\widehat{\mathbf{w}}^{(i)}\}_{i=1}^n \in \underset{\mathbf{w} = \text{stack}\{\mathbf{w}^{(i)}\}_{i=1}^n}{\text{argmin}} \quad \mathbf{w}^T \mathbf{Q} \mathbf{w} + \mathbf{q}^T \mathbf{w}. \quad (3.19)$$

Here, we used the psd matrix

$$\mathbf{Q} := \begin{pmatrix} \mathbf{Q}^{(1)} & 0 & \dots & 0 \\ 0 & \mathbf{Q}^{(2)} & \dots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \dots & \mathbf{Q}^{(n)} \end{pmatrix} + \alpha \mathbf{L} \otimes \mathbf{I} \text{ with } \mathbf{Q}^{(i)} := (1/m_i)(\mathbf{X}^{(i)})^T \mathbf{X}^{(i)} \quad (3.20)$$

and the vector

$$\mathbf{q} := ((\mathbf{q}^{(1)})^T, \dots, (\mathbf{q}^{(n)})^T)^T, \text{ with } \mathbf{q}^{(i)} := (-2/m_i)(\mathbf{X}^{(i)})^T \mathbf{y}^{(i)}. \quad (3.21)$$

### 3.3.1 Computational Aspects of GTVMin

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

classes of optimization methods. For example, using a differentiable loss function allows to apply gradient-based methods (see Lecture 4) to solve GTVMin.

Another important class of loss functions are those for which we can efficiently compute the proximity operator

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

Some authors refer to functions  $L$  for which  $\mathbf{prox}_{L,\rho}(\mathbf{w})$  can be computed easily as *simple* or *proximable* [35]. GTVMin with proximable loss functions can be solved quite efficiently via proximal algorithms [36].

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. For example, using an empirical graph with relatively few edges (“sparse graphs”) typically results in a smaller computational complexity. Indeed, Lecture 5 discusses GTVMin-based algorithms requiring an amount of computation that is proportional to the number of edges in the empirical graph.

Let us now consider the computational aspects of GTVMin (3.16) to train local linear models. As discussed above, this instance is equivalent to solving (3.19). Any solution  $\hat{\mathbf{w}}$  of (3.19) (and, in turn, (3.16)) is characterized by the zero-gradient condition

$$\mathbf{Q}\hat{\mathbf{w}} = -(1/2)\mathbf{q}, \tag{3.22}$$

with  $\mathbf{Q}, \mathbf{q}$  as defined in (3.20) and (3.21). If the matrix  $\mathbf{Q}$  in (3.22) is invertible, the solution to (3.22) and, in turn, to the GTVMin instance (3.16) is unique and given by  $\hat{\mathbf{w}} = (-1/2)\mathbf{Q}^{-1}\mathbf{q}$ .

The size of the matrix  $\mathbf{Q}$  (see (3.20)) is proportional to the number of nodes in the empirical graph  $\mathcal{G}$ , which might be in the order of millions in some applications (such as social networks). For such large systems, we typically cannot use direct matrix inversion methods (e.g., based on Gaussian elimination) to compute  $\mathbf{Q}^{-1}$ .<sup>3</sup> Instead, we typically need to resort to iterative methods [37, 38].

Instead of using (3.22) to compute solutions to (3.19), we can also directly apply gradient-based methods (see Lecture 4) to solve (3.19). Starting from an initial choice for the local model parameters  $\widehat{\mathbf{w}}_0 = (\widehat{\mathbf{w}}_0^{(1)}, \dots, \widehat{\mathbf{w}}_0^{(n)})$ , we iterate the gradient step,

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

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

$$\widehat{\mathbf{w}}_{k+1} := \left( (\widehat{\mathbf{w}}^{(1)})^T, \dots, (\widehat{\mathbf{w}}^{(n)})^T \right)^T.$$

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

### 3.3.2 Statistical Aspects of GTVMin

What can we say about FL system that use the solutions of GTVMin (3.14) as local model parameters? To answer such a question, we use - as for the statistical analysis of ERM in Lecture 2 - a probabilistic model for the local datasets.

---

<sup>3</sup>How many arithmetic operations (addition, multiplication) do you think are required to invert an arbitrary matrix  $\mathbf{Q}^{d \times d}$ ?

In what follows we focus on the GTVMin instance (3.16) to learn the parameters  $\mathbf{w}^{(i)}$  of a local linear model for each node  $i \in \mathcal{V}$ . For each local dataset  $\mathcal{D}^{(i)}$ , we assume that features and labels of its data points are realizations of i.i.d. RVs

$$\mathbf{y}^{(i)} = \underbrace{(\mathbf{x}^{(i,1)}, \dots, \mathbf{x}^{(i,m_i)})^T}_{\text{local feature matrix } \mathbf{X}^{(i)}} \bar{\mathbf{w}}^{(i)} + \boldsymbol{\varepsilon}^{(i)} \text{ with } \mathbf{x}^{(i,r)} \stackrel{\text{i.i.d.}}{\sim} \mathcal{N}(\mathbf{0}, \mathbf{I}), \boldsymbol{\varepsilon}^{(i)} \sim \mathcal{N}(\mathbf{0}, \sigma^2 \mathbf{I}). \quad (3.23)$$

In contrast to the probabilistic model (2.13) for the analysis of ERM, the probabilistic model involves node-specific parameters  $\bar{\mathbf{w}}^{(i)}$ , for  $i \in \mathcal{V}$ .

For a reasonable choice for empirical graph, the parameters  $\bar{\mathbf{w}}^{(i)}, \bar{\mathbf{w}}^{(i')}$  at connected nodes  $\{i, i'\} \in \mathcal{E}$  should be similar. However, we cannot choose the edge weights based on parameters  $\bar{\mathbf{w}}^{(i)}$  as they are unknown. We can only use (noisy) estimates for  $\bar{\mathbf{w}}^{(i)}$  the features and labels of the data points in the local datasets (see Lecture 7).

Consider a given (design) choice for the empirical graph (and its edge weights  $A_{i,i'}$ ) for local datasets that conform to the probabilistic model (3.23) with true underlying parameters  $\bar{\mathbf{w}}^{(i)}$ . For ease of exposition, we assume that

$$\bar{\mathbf{w}}^{(i)} = \bar{\mathbf{c}}, \text{ for some } \bar{\mathbf{c}} \in \mathbb{R}^d \text{ and all } i \in \mathcal{V}. \quad (3.24)$$

To study the deviation between the solutions  $\hat{\mathbf{w}}^{(i)}$  of (3.16) and the true underlying parameters  $\bar{\mathbf{w}}^{(i)}$ , we decompose it as

$$\hat{\mathbf{w}}^{(i)} =: \tilde{\mathbf{w}}^{(i)} + \hat{\mathbf{c}}, \text{ with } \hat{\mathbf{c}} := (1/n) \sum_{i'=1}^n \hat{\mathbf{w}}^{(i')}. \quad (3.25)$$

The component  $\hat{\mathbf{c}}$  is identical at all nodes  $i \in \mathcal{V}$  and obtained as the orthogonal projection of  $\hat{\mathbf{w}} = \text{stack}\{\hat{\mathbf{w}}^{(i)}\}_{i=1}^n$  on the subspace (3.13). The component



$\tilde{\mathbf{w}}^{(i)} := \hat{\mathbf{w}}^{(i)} - (1/n) \sum_{i'=1}^n \hat{\mathbf{w}}^{(i')}$  averages to the zero vector,  $(1/n) \sum_{i=1}^n \tilde{\mathbf{w}}^{(i)} = \mathbf{0}$ .

The decomposition (3.25) entails an analogous (orthogonal) decomposition of the error  $\hat{\mathbf{w}}^{(i)} - \bar{\mathbf{w}}^{(i)}$ . Indeed, for identical true underlying model parameters (3.24) (which makes  $\bar{\mathbf{w}}$  a member of the subspace (3.13)), we have

$$\sum_{i=1}^n \|\tilde{\mathbf{w}}^{(i)} - \bar{\mathbf{w}}^{(i)}\|_2^2 \stackrel{(3.24), (3.25)}{=} \underbrace{\sum_{i=1}^n \|\mathbf{c} - \hat{\mathbf{c}}\|_2^2}_{n\|\mathbf{c} - \hat{\mathbf{c}}\|_2^2} + \sum_{i=1}^n \|\tilde{\mathbf{w}}^{(i)}\|_2^2 \quad (3.26)$$

The following proposition provides an upper bound on the second error component in (3.26).

**Proposition 3.1.** *Consider a connected empirical graph, i.e.,  $\lambda_2 > 0$  (see (3.10)), and the solution (3.25) to GTVMin (3.16) for the local datasets (3.23). If the true local model parameters in (3.23) are identical (see (3.24)), we can upper bound the component  $\tilde{\mathbf{w}}^{(i)} := \hat{\mathbf{w}}^{(i)} - (1/n) \sum_{i=1}^n \hat{\mathbf{w}}^{(i)}$  as*

$$\sum_{i=1}^n \|\tilde{\mathbf{w}}^{(i)}\|_2^2 \leq \frac{1}{\lambda_2 \alpha} \sum_{i=1}^n (1/m_i) \|\boldsymbol{\epsilon}^{(i)}\|_2^2. \quad (3.27)$$

*Proof.* See Section 3.5.1. □

The upper bound (3.27) involves three components:

- properties of the local datasets, via the noise term  $\boldsymbol{\epsilon}^{(i)}$  in (3.23),
- the empirical graph via the eigenvalue  $\lambda_2(\mathbf{L})$  (see (3.10)),
- the GTVMin parameter  $\alpha$ .

According to (3.27), we can ensure a small error component  $\tilde{\mathbf{w}}^{(i)}$  of the GTVMin solution by choosing a large value  $\alpha$ . In other words, for sufficiently

large  $\alpha$  and a connected empirical graph (where  $\lambda_2 > 0$ ), the solution of GTVMin  $\hat{\mathbf{w}}^{(i)}$  are approximately identical for all nodes  $i \in \mathcal{V}$ . This might be desirable for some FL applications where the goal is to train a common model for all nodes [13]. However, some FL applications involve heterogeneous local datasets for which it is detrimental to force all nodes agreeing on a common model (see Lecture 6)

c

### 3.4 Overview of Coding Assignment

**Python Notebook.** `FLDesignPrinciple_CodingAssignment.ipynb`

**Data File.** `Assignment_MLBasicsData.csv`

This assignment revolves around a collection of temperature measurements that we store in the empirical graph  $\mathcal{G}^{(\text{FMI})}$ . Each node  $i \in \mathcal{V}$  represents a FMI weather station at latitude  $\text{lat}^{(i)}$  and longitude  $\text{lon}^{(i)}$ . The local dataset  $\mathcal{D}^{(i)}$  contains  $m_i$  temperature measurements  $y^{(i,1)}, \dots, y^{(i,m_i)}$  at the station  $i$ .

The edges of  $\mathcal{G}^{(\text{FMI})}$  are obtained using the **Python** function `add_edges()`. Each FMI station  $i$  is connected to its nearest neighbours  $i'$ , using the Euclidean distance between the corresponding vectors  $(\text{lat}^{(i)}, \text{lon}^{(i)})^T \in \mathbb{R}^2$ . The number of neighbours is controlled by the input parameter `numneighbors`. All edges  $\{i, i'\} \in \mathcal{E}$  have the same edge weight  $A_{i,i'} = 1$ .

For each station  $i \in \mathcal{V}$ , you need to learn the single parameter  $w^{(i)} \in \mathbb{R}$  of a hypothesis  $h(x) = w^{(i)}$  that predicts the temperature. We measure the quality of a hypothesis by the average squared error loss  $L_i(w^{(i)}) = (1/m_i) \sum_{r=1}^{m_i} (y^{(i,r)} - w^{(i)})^2$ . You should learn the parameters  $w^{(i)}$  via balancing the local loss with the total variation of  $w^{(i)}$ ,

$$(\hat{w}^{(1)}, \dots, \hat{w}^{(n)})^T \in \underset{(w^{(1)}, \dots, w^{(n)})^T}{\operatorname{argmin}} \sum_{i=1}^n L_i(w^{(i)}) + \alpha \sum_{\{i, i'\} \in \mathcal{E}} (w^{(i)} - w^{(i')})^2. \quad (3.28)$$

Your tasks involve

- Reformulate (3.28) as (3.16) using a suitable choice for features  $\mathbf{x}^{(i,r)}$ . Carefully note that this choice needs to be different from the raw features (longitude, latitude, year, month, day, hour, minute) used in the assignment “ML Basics”.

- The computation of solutions  $\hat{w}^{(i)}$  to (3.28) via (3.22) and using the Python function `numpy.linalg.inv`. To this end, you should determine the matrix  $\mathbf{Q}$  and vector  $\mathbf{q}$  in terms of local datasets  $\mathcal{D}^{(i)}$  and the Laplacian matrix  $\mathbf{L}^{(\text{FMI})}$  of the empirical graph  $\mathcal{G}^{(\text{FMI})}$ .
- Studying the effect of varying values of  $\alpha$  in (3.28) on the local loss and total variation of the corresponding solutions  $\hat{w}^{(i)}$ .

## 3.5 Proofs

### 3.5.1 Proof of Proposition 3.1

Let us introduce the shorthand  $f(\mathbf{w}^{(i)})$  for the objective function of the GTVMin instance (3.16). We verify the bound (3.27) by showing that if it does not hold, the choice for the local model parameters  $\mathbf{w}^{(i)} := \bar{\mathbf{w}}^{(i)}$  (see (3.23)) results in a smaller objective function value,  $f(\bar{\mathbf{w}}^{(i)}) < f(\hat{\mathbf{w}}^{(i)})$ . This would contradict the fact that  $\hat{\mathbf{w}}^{(i)}$  is a solution to (3.16).

First, note that

$$\begin{aligned}
f(\bar{\mathbf{w}}^{(i)}) &= \sum_{i \in \mathcal{V}} (1/m_i) \|\mathbf{y}^{(i)} - \mathbf{X}^{(i)} \bar{\mathbf{w}}^{(i)}\|_2^2 + \alpha \sum_{i, i' \in \mathcal{V}} A_{i, i'} \|\bar{\mathbf{w}}^{(i)} - \bar{\mathbf{w}}^{(i')}\|_2^2 \\
&\stackrel{(3.24)}{=} \sum_{i \in \mathcal{V}} (1/m_i) \|\mathbf{y}^{(i)} - \mathbf{X}^{(i)} \bar{\mathbf{w}}^{(i)}\|_2^2 \\
&\stackrel{(3.23)}{=} \sum_{i \in \mathcal{V}} (1/m_i) \|\mathbf{X}^{(i)} \bar{\mathbf{w}}^{(i)} + \boldsymbol{\varepsilon}^{(i)} - \mathbf{X}^{(i)} \bar{\mathbf{w}}^{(i)}\|_2^2 \\
&= \sum_{i \in \mathcal{V}} (1/m_i) \|\boldsymbol{\varepsilon}^{(i)}\|_2^2. \tag{3.29}
\end{aligned}$$

Inserting (3.25) into (3.16),

$$\begin{aligned}
f(\hat{\mathbf{w}}^{(i)}) &= \underbrace{\sum_{i \in \mathcal{V}} (1/m_i) \|\mathbf{y}^{(i)} - \mathbf{X}^{(i)} \hat{\mathbf{w}}^{(i)}\|_2^2}_{\geq 0} + \alpha \sum_{i, i' \in \mathcal{V}} A_{i, i'} \underbrace{\|\hat{\mathbf{w}}^{(i)} - \hat{\mathbf{w}}^{(i')}\|_2^2}_{\stackrel{(3.25)}{=} \|\tilde{\mathbf{w}}^{(i)} - \tilde{\mathbf{w}}^{(i')}\|_2^2} \\
&\geq \alpha \sum_{i, i' \in \mathcal{V}} A_{i, i'} \|\tilde{\mathbf{w}}^{(i)} - \tilde{\mathbf{w}}^{(i')}\|_2^2 \\
&\stackrel{(3.12)}{\geq} \alpha \lambda_2 \sum_{i=1}^n \|\tilde{\mathbf{w}}^{(i)}\|_2^2. \tag{3.30}
\end{aligned}$$

If the bound (3.27) would not hold, then by (3.30) and (3.29) we would obtain  $f(\hat{\mathbf{w}}^{(i)}) > f(\bar{\mathbf{w}}^{(i)})$ . This is a contradiction to the fact that  $\hat{\mathbf{w}}^{(i)}$  solves (3.16).

## 4 Lecture - “Gradient Methods”

February 27, 2024

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

### 4.1 Learning Goals

After this lecture, you should

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

## 4.2 Gradient Descent

Unless stated otherwise, we consider an objective function of form

$$f(\mathbf{w}) = \mathbf{w}^T \mathbf{Q} \mathbf{w} + \mathbf{q}^T \mathbf{w}. \quad (4.1)$$

Note that (4.1) defines a family of convex quadratic functions. 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 involve an objective function of the form (4.1): Linear regression (2.3) and ridge regression (2.24) in Lecture 2 as well as GTVMin (3.16) for local linear models in Lecture 3. Moreover, (4.1) might serve as a useful approximation for the objective functions arising for larger classes of ML methods [39–41].

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

$$\begin{aligned} \mathbf{w}^{(k+1)} &:= \mathbf{w}^{(k)} - \eta \nabla f(\mathbf{w}^{(k)}) \\ &\stackrel{(4.1)}{=} \mathbf{w}^{(k)} - \eta (2\mathbf{Q}\mathbf{w}^{(k)} + \mathbf{q}). \end{aligned} \quad (4.2)$$

The gradient step (4.2) involves the factor  $\eta$  which we refer to as step-size or learning rate. Algorithm 1 summarizes the most basic instance of gradient-based methods which simply repeats (iterates) (4.2) until some stopping criterion is met.

The usefulness of gradient-based methods crucially depends on the ability to efficiently evaluate the gradient  $\nabla f(\mathbf{w})$ . Evaluating the gradient of a fully

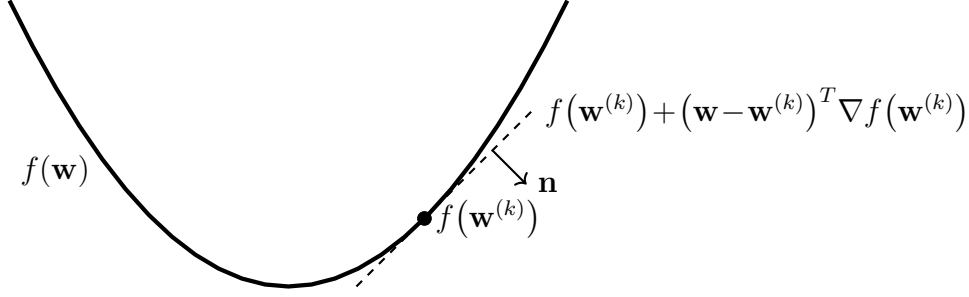


Figure 4.1: We can approximate a differentiable function  $f(\mathbf{w})$  locally around a point  $\mathbf{w}^{(k)} \in \mathbb{R}^d$  using the linear function  $f(\mathbf{w}^{(k)}) + (\mathbf{w} - \mathbf{w}^{(k)})^T \nabla f(\mathbf{w}^{(k)})$ . Geometrically, we approximate the graph of  $f(\mathbf{w})$  by a hyperplane with normal vector  $\mathbf{n} = (\nabla f(\mathbf{w}^{(k)}), -1)^T \in \mathbb{R}^{d+1}$  of this approximating hyperplane is determined by the gradient  $\nabla f(\mathbf{w}^{(k)})$  [5].

specified function has been made convenient by modern software libraries for automatic differentiation [42]. However, besides the actual computation of the gradient, it might be already challenging to gather the required data points which define the objective function  $f(\mathbf{w})$ . Indeed, the matrix  $\mathbf{Q}$  and vector  $\mathbf{q}$  in (4.1) are constructed from the features and labels of data points in the training set (see, e.g., (2.24)).

Note that Algorithm 1, as most other gradient-based methods, involves two hyper-parameters: (i) the learning rate  $\eta$  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.



---

**Algorithm 1** A blueprint for gradient-based methods

---

**Input:** function  $f(\mathbf{w})$  ; learning rate  $\eta > 0$ ; some stopping criterion;**Initialize:** set  $\mathbf{w}^{(0)} := \mathbf{0}$ ; set iteration counter  $k := 0$ 1: **repeat**2:      $k := k + 1$  (increase iteration counter)3:      $\mathbf{w}^{(k)} := \mathbf{w}^{(k-1)} - \eta \nabla f(\mathbf{w}^{(k-1)})$  (do a gradient step (4.2))4: **until** stopping criterion is met**Output:**  $\hat{\mathbf{w}} := \mathbf{w}^{(k)}$  (hopefully  $f(\hat{\mathbf{w}}) \approx \min_{\mathbf{w}} f(\mathbf{w})$ )

---

### 4.3 Learning Rate

The learning rate must not be too large to avoid overshooting or moving away from the optimum (see Figure 4.2-(b)). On the other hand, if the learning rate is chosen too small, the gradient step makes too little progress towards the solutions of (4.1) (see Figure 4.2-(a)). Note that in practice we can only afford to repeat the gradient step for a finite number of iterations.

One approach to choose the learning rate is to start with some initial value (first guess) and monitor the decrease of 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 of the objective function. We repeat this procedure until a sufficient decrease of the objective function is achieved [43, Sec 6.1].

Alternatively, we can use a prescribed sequence (schedule)  $\eta_k$ , for  $k = 1, 2, \dots$ , of learning rates that vary across successive gradient steps [44]. We can guarantee convergence of (4.2) (under mild assumptions on the objective

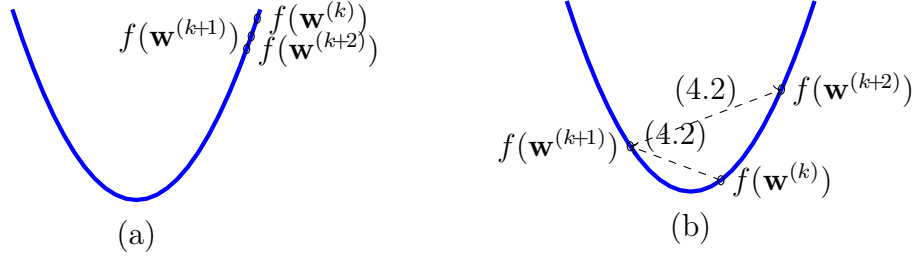


Figure 4.2: Effect of inadequate learning rates  $\eta$  in the gradient step (4.2). (a) If  $\eta$  is too small, the gradient steps make very little progress towards the optimum or even fail to reach the optimum at all. (b) If  $\eta$  is too large, the gradient steps might “overshoot” such that the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

function  $f(\mathbf{w})$ ) by using a learning rate schedule that satisfies [43, Sec. 6.1]

$$\lim_{k \rightarrow \infty} \eta_k = 0, \quad \sum_{k=1}^{\infty} \eta_k = \infty, \quad \text{and} \quad \sum_{k=1}^{\infty} \eta_k^2 < \infty. \quad (4.3)$$

The first condition (4.3) requires that the learning rate eventually become sufficiently small to avoid overshooting. The third condition (4.3) requires that the learning rate decays sufficiently fast. Note that the first and third condition in (4.3) could be satisfied by the trivial learning rate schedule  $\eta_k = 0$  which is clearly not useful as the gradient step has no effect. This trivial schedule is prohibited by the middle condition of (4.3) which requires that the learning rate  $\eta_k$  is large enough such that the gradient step makes sufficient progress.

## 4.4 When to Stop?

For the stopping criterion we might use a fixed number  $k_{\max}$  of iterations. This hyper-parameter  $k_{\max}$  might be dictated by limited resources (such as computational time) for implementing GD or tuned via validation techniques.

We can obtain another stopping criterion from monitoring the decrease in the objective function  $f(\mathbf{w}^{(k)})$ : we stop repeating the gradient step (4.2) whenever  $|f(\mathbf{w}^{(k)}) - f(\mathbf{w}^{(k+1)})| \leq \varepsilon^{(\text{tol})}$  for a given tolerance  $\varepsilon^{(\text{tol})}$ . The tolerance level  $\varepsilon^{(\text{tol})}$  is a hyper-parameter of the resulting ML method which could be chosen via validation techniques (see Lecture 2).

The above techniques for choosing the learning rate  $\eta$  and deciding when to stop the gradient steps might be useful in practice. However, they might result in sub-optimal use of computational resources for situations where we have some information about the psd matrix  $\mathbf{Q}$  in (4.1). Indeed, the choice for the learning rate  $\eta$  and the stopping criterion can be guided by the eigenvalues

$$0 \leq \lambda_1(\mathbf{Q}) \leq \dots \leq \lambda_d(\mathbf{Q}). \quad (4.4)$$

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,

$$0 \leq L \leq \lambda_1(\mathbf{Q}) \leq \dots \leq \lambda_d(\mathbf{Q}) \leq U. \quad (4.5)$$

In what follows, we assume that  $\mathbf{Q}$  is invertible and we know some positive lower bound  $L > 0$  (see (4.5)).<sup>4</sup> The objective function (4.1) has then a unique solution  $\hat{\mathbf{w}}$ . It turns out that a gradient step (4.2) reduces the distance

---

<sup>4</sup>What are sufficient conditions for the local datasets and the edge weights used in GTVMin such that  $\mathbf{Q}$  in (3.20) is invertible?

$\|\mathbf{w}^{(k)} - \widehat{\mathbf{w}}\|_2$  to  $\widehat{\mathbf{w}}$  by a constant factor [43, Ch. 6],

$$\|\mathbf{w}^{(k+1)} - \widehat{\mathbf{w}}\|_2 \leq \kappa^{(\eta_k)}(\mathbf{Q}) \|\mathbf{w}^{(k)} - \widehat{\mathbf{w}}\|_2. \quad (4.6)$$

Here, we used the contraction factor

$$\kappa^{(\eta)}(\mathbf{Q}) := \max\{|1 - \eta 2\lambda_1|, |1 - \eta 2\lambda_d|\}. \quad (4.7)$$

The contraction factor depends on the learning rate  $\eta$  which is a hyperparameter of gradient-based methods that we can control. However, the contraction factor also depends on the eigenvalues of the matrix  $\mathbf{Q}$  in (4.1). In ML and FL applications, this matrix typically depends on data and can be controlled only to some extent, e.g., using feature transformation [4, Ch. 5]. We can ensure  $\kappa^{(\eta)}(\mathbf{Q}) < 1$  if we use a positive learning rate  $\eta_k < 1/U$ .

For a contraction factor  $\kappa^{(\eta)}(\mathbf{Q}) < 1$ , a sufficient condition on the number  $k^{(\varepsilon^{(\text{tol})})}$  of gradient steps required to ensure an optimization error  $\|\mathbf{w}^{(k+1)} - \widehat{\mathbf{w}}\|_2 \leq \varepsilon^{(\text{tol})}$  is (see (4.6))

$$k^{(\varepsilon^{(\text{tol})})} \geq \log(\|\mathbf{w}^{(0)} - \widehat{\mathbf{w}}\|_2 / \varepsilon^{(\text{tol})}) / \log(1/\kappa^{(\eta)}(\mathbf{Q})). \quad (4.8)$$

According to (4.6), smaller values of the contraction factor  $\kappa^{(\eta)}(\mathbf{Q})$  guarantee a faster convergence of (4.2) towards the solution of (4.1). Figure 4.3 illustrates the dependence of  $\kappa^{(\eta)}(\mathbf{Q})$  on the learning rate  $\eta$ . Thus, choosing a small  $\eta$  (close to 0) will typically result in a larger  $\kappa^{(\eta)}(\mathbf{Q})$  and, in turn, require more iterations to ensure optimization error level  $\varepsilon^{(\text{tol})}$  via (4.6).

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

$$\eta^{(*)} := \frac{1}{\lambda_1 + \lambda_d}. \quad (4.9)$$

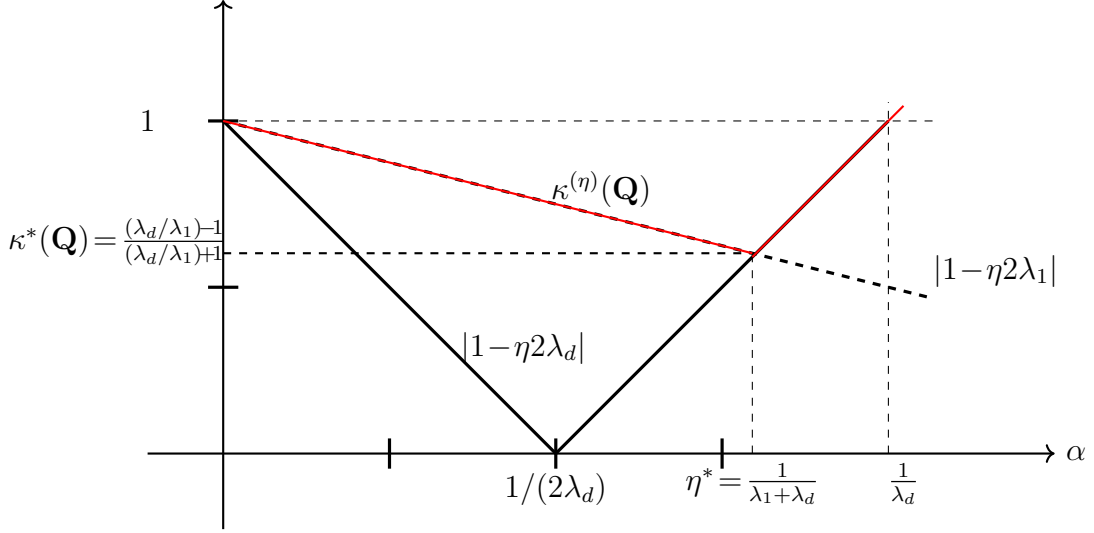


Figure 4.3: The contraction factor  $\kappa^{(\eta)}(\mathbf{Q})$  (4.7), used in the upper bound (4.6), as a function of the learning rate  $\eta$ . Note that  $\kappa^{(\eta)}(\mathbf{Q})$  also depends on the eigenvalues of the matrix  $\mathbf{Q}$  in (4.1).

[Note that evaluating (4.9) requires to know the extremal eigenvalues  $\lambda_1, \lambda_d$  of  $\mathbf{Q}$ .] Inserting the optimal learning rate (4.9) into (4.6),

$$\|\mathbf{w}^{(k+1)} - \widehat{\mathbf{w}}\|_2 \leq \underbrace{\frac{(\lambda_d/\lambda_1) - 1}{(\lambda_d/\lambda_1) + 1}}_{:=\kappa^*(\mathbf{Q})} \|\mathbf{w}^{(k)} - \widehat{\mathbf{w}}\|_2. \quad (4.10)$$

Carefully note that the formula (4.10) is valid only if the matrix  $\mathbf{Q}$  in (4.1) is invertible, i.e., if  $\lambda_1 > 0$ . If the matrix  $\mathbf{Q}$  is singular ( $\lambda_1 = 0$ ), the convergence of (4.2) towards a solution of (4.1) is much slower than the decrease of the bound (4.10). However, we can still ensure converge of gradient steps by using a fixed learning rate  $\eta_k = \eta$  that satisfies [45, Thm.

2.1.14]

$$0 < \eta < 1/\lambda_d(\mathbf{Q}). \quad (4.11)$$

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 (2.17)) but not on their labels  $y^{(r)}$ . Thus, the convergence of gradient steps is only affected by the features, whereas the labels are irrelevant. The same is true for ridge regression and GTVMin (using local linear models).

Note that both, the optimal learning rate (4.9) and the optimal contraction factor

$$\kappa^*(\mathbf{Q}) := \frac{(\lambda_d/\lambda_1) - 1}{(\lambda_d/\lambda_1) + 1} \quad (4.12)$$

depend on the eigenvalues of the matrix  $\mathbf{Q}$  in (4.1). According to (4.10), 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 (4.1).

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 (2.7) is determined by the features of data points in the training set. These features might be obtained from sensing devices and therefore beyond our control. However, other 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 (2.7) more towards a scaled identity matrix.

## 4.5 Perturbed Gradient Step

Consider the gradient step (4.2) used to find a minimum of (4.1). We again assume that the matrix  $\mathbf{Q}$  in (4.1) is invertible ( $\lambda_1(\mathbf{Q}) > 0$ ) and, in turn, (4.1)

has a unique solution  $\widehat{\mathbf{w}}$ .

In some applications, it might be difficult to evaluate the gradient  $\nabla f(\mathbf{w}) = \mathbf{Q}\mathbf{w} + \mathbf{q}$  of (4.1) exactly. For example, this evaluation might require to gather data points from distributed storage locations. These storage locations might become unavailable during the computation of  $\nabla f(\mathbf{w})$  due to software or hardware failures (e.g., limited connectivity).

We can model imperfections during the computation of (4.2) as the perturbed gradient step

$$\begin{aligned}\mathbf{w}^{(k+1)} &:= \mathbf{w}^{(k)} - \eta \nabla f(\mathbf{w}^{(k)}) + \boldsymbol{\varepsilon}^{(k)} \\ &\stackrel{(4.1)}{=} \mathbf{w}^{(k)} - \eta(2\mathbf{Q}\mathbf{w}^{(k)} + \mathbf{q}) + \boldsymbol{\varepsilon}^{(k)}.\end{aligned}\tag{4.13}$$

We can use the contraction factor (4.7) to upper bound the deviation between  $\mathbf{w}^{(k)}$  and the optimum  $\widehat{\mathbf{w}}$  as (see (4.6))

$$\|\mathbf{w}^{(k)} - \widehat{\mathbf{w}}\|_2 \leq (\kappa^{(\eta)}(\mathbf{Q}))^k \|\mathbf{w}^{(0)} - \widehat{\mathbf{w}}\|_2 + \sum_{k'=1}^k (\kappa^{(\eta)}(\mathbf{Q}))^{k'} \|\boldsymbol{\varepsilon}^{(k-k')}\|_2.\tag{4.14}$$

This bound applies for any number of iterations  $k = 1, 2, \dots$  of the perturbed gradient step (4.13).

## 4.6 Handling Constraints - Projected Gradient Descent

Many important ML and FL methods amount to the minimization of an objective function of the form (4.1). The optimization variable  $\mathbf{w}$  in (4.1) represents some model parameters.

Sometimes we might require the parameters  $\mathbf{w}$  to belong to a subset  $\mathcal{C} \subset \mathbb{R}^d$ . One example is regularization via model pruning (see Lecture 2).

Another example are FL methods that learn identical local model parameters  $\mathbf{w}^{(i)}$  at all nodes  $i \in \mathcal{V}$  of an empirical graph. This can be implemented by requiring the stacked local model parameters  $\mathbf{w} = (\mathbf{w}^{(1)}, \dots, \mathbf{w}^{(n)})^T$  to belong to the subset

$$\mathcal{C} = \left\{ (\mathbf{w}^{(1)}, \dots, \mathbf{w}^{(n)})^T : \mathbf{w}^{(1)} = \dots = \mathbf{w}^{(n)} \right\}.$$

Let us now show how to adapt the gradient step (4.2) to solve the constrained problem

$$f^* = \min_{\mathbf{w} \in \mathcal{C}} \mathbf{w}^T \mathbf{Q} \mathbf{w} + \mathbf{q}^T \mathbf{w}. \quad (4.15)$$

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

$$P_{\mathcal{C}}(\mathbf{w}) = \operatorname{argmin}_{\mathbf{w}' \in \mathcal{C}} \|\mathbf{w} - \mathbf{w}'\|_2 \text{ for any } \mathbf{w} \in \mathbb{R}^d. \quad (4.16)$$

A suitable modification of the gradient step (4.2) to solve the constrained variant (4.15) is [43]

$$\begin{aligned} \mathbf{w}^{(k+1)} &:= P_{\mathcal{C}}(\mathbf{w}^{(k)} - \eta \nabla f(\mathbf{w}^{(k)})) \\ &\stackrel{(4.1)}{=} P_{\mathcal{C}}(\mathbf{w}^{(k)} - \eta(2\mathbf{Q}\mathbf{w}^{(k)} + \mathbf{q})). \end{aligned} \quad (4.17)$$

The projected GD step (4.17) amounts to

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

Note that we re-obtain the basic gradient step (4.2) from the projected gradient step (4.17) for the specific constraint set  $\mathcal{C} = \mathbb{R}^d$ .



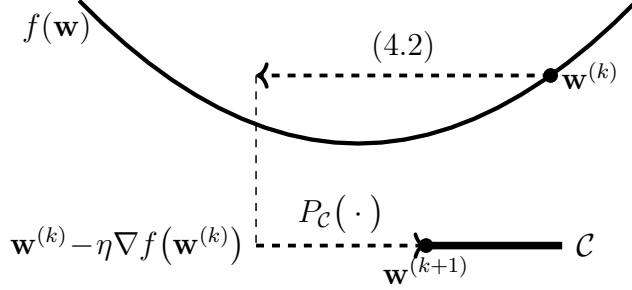


Figure 4.4: Projected GD augments a basic gradient step with a projection back onto the constraint set  $\mathcal{C}$ .

The approaches for choosing the learning rate  $\eta$  and stopping criterion for basic gradient step (4.2) explained in Sections 4.3 and 4.4 work also for the projected gradient step (4.17). In particular, the convergence speed of the projected gradient step is also characterized by (4.6) [43, Ch. 6]. This follows from the fact that the concatenation of a contraction (such as the gradient step (4.2) for sufficiently small  $\eta$ ) and a projection (such as  $P_{\mathcal{C}}(\cdot)$ ) 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 (4.6) is only telling about the number of projected gradient steps (4.17) required to achieve a guaranteed level of sub-optimality  $|f(\mathbf{w}^{(k)}) - f^*|$ . The iteration (4.17) of projected GD might require significantly more computation than the basic gradient step, as it requires to compute the projection (4.16).

## 4.7 Generalizing the Gradient Step

The gradient-based methods discussed so far can be used to learn a hypothesis from a parametrized model. Let us now sketch one possible generalization of the gradient step (4.2) for a model  $\mathcal{H}$  without a parametrization.

We start with rewriting the gradient step (4.2) as the optimization

$$\mathbf{w}^{(k+1)} = \underset{\mathbf{w} \in \mathbb{R}^d}{\operatorname{argmin}} \left( \frac{1}{2\eta} \left\| \mathbf{w} - \mathbf{w}^{(k)} \right\|_2^2 + \underbrace{f(\mathbf{w}^{(k)}) + (\mathbf{w} - \mathbf{w}^{(k)})^T \nabla f(\mathbf{w}^{(k)})}_{\approx f(\mathbf{w})} \right). \quad (4.18)$$

Note that the objective function in (4.18) is the first-order approximation

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

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

Let us modify (4.18) by using  $f(\mathbf{w})$  itself (instead of an approximation),

$$\mathbf{w}^{(k+1)} = \underset{\mathbf{w} \in \mathbb{R}^d}{\operatorname{argmin}} \left( \frac{1}{2\eta} \left\| \mathbf{w} - \mathbf{w}^{(k)} \right\|_2^2 + f(\mathbf{w}) \right). \quad (4.19)$$

Like the gradient step, also (4.19) maps a given vector  $\mathbf{w}^{(k)}$  to an updated vector  $\mathbf{w}^{(k+1)}$ . The mapping or operator defined by (4.19) is known as the proximity operator of the function  $f(\mathbf{w})$  [36]. Similar to the gradient step being the main building block of gradient-based methods, the proximity operator (4.19) is the main building block of proximal algorithms [36].

To obtain a version of (4.19) 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 (4.19) consists of two components. The second component  $f(\cdot)$ , which is the function we want to minimize, is obtained from a training error incurred by a hypothesis, which

might be parametrized  $h^{(\mathbf{w})}$ . Thus, we can evaluate the function  $f(h)$  by computing the training error for a given hypothesis.

The first component of (4.19) uses  $\|\mathbf{w} - \mathbf{w}^{(k)}\|_2^2$  to measure the difference between the hypothesis maps  $h^{(\mathbf{w})}$  and  $h^{(\mathbf{w}^{(k)})}$ . Another measure for the difference two hypothesis maps can be obtained by using some dataset  $\mathcal{D}' = \{\mathbf{x}^{(1)}, \dots, \mathbf{x}^{(m')}\}$ .

The dis-similarity between two hypothesis maps  $h$  and  $h^{(k)}$  is reflected by the difference of their predictions  $h(\mathbf{x}) - h^{(k)}(\mathbf{x})$ . In particular, we use the average squared difference between their predictions,

$$(1/m') \sum_{r=1}^{m'} \left( h(\mathbf{x}^{(r)}) - h^{(k)}(\mathbf{x}^{(r)}) \right)^2. \quad (4.20)$$

It is interesting to note that (4.20) coincides with  $\|\mathbf{w} - \mathbf{w}^{(k)}\|_2^2$  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  $\mathbf{x}^{(1)}, \mathbf{x}^{(2)}, \dots$  of i.i.d. RVs with common probability distribution  $\mathbf{x} \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$ . Indeed, by the law of large numbers

$$\begin{aligned} & \lim_{m' \rightarrow \infty} (1/m') \sum_{r=1}^{m'} \left( h^{(\mathbf{w})}(\mathbf{x}^{(r)}) - h^{(\mathbf{w}^{(k)})}(\mathbf{x}^{(r)}) \right)^2 \\ &= \lim_{m' \rightarrow \infty} (1/m') \sum_{r=1}^{m'} \left( (\mathbf{w} - \mathbf{w}^{(k)})^T \mathbf{x}^{(r)} \right)^2 \\ &= \lim_{m' \rightarrow \infty} (1/m') \sum_{r=1}^{m'} (\mathbf{w} - \mathbf{w}^{(k)})^T \mathbf{x}^{(r)} (\mathbf{x}^{(r)})^T (\mathbf{w} - \mathbf{w}^{(k)}) \\ &= (\mathbf{w} - \mathbf{w}^{(k)})^T \underbrace{\left[ \lim_{m' \rightarrow \infty} (1/m') \sum_{r=1}^{m'} \mathbf{x}^{(r)} (\mathbf{x}^{(r)})^T \right]}_{=\mathbf{I}} (\mathbf{w} - \mathbf{w}^{(k)}) \\ &= \|\mathbf{w} - \mathbf{w}^{(k)}\|_2^2. \end{aligned} \quad (4.21)$$

To arrive at our generalization of the gradient step, we replace  $\|\mathbf{w} - \mathbf{w}^{(k)}\|_2^2$  in (4.19) with (4.20),

$$h^{(k+1)} = \operatorname{argmin}_{h \in \mathcal{H}} \left[ (1/(2\eta m')) \sum_{r=1}^{m'} \left( h(\mathbf{x}^{(r)}) - h^{(k)}(\mathbf{x}^{(r)}) \right)^2 + f(h) \right]. \quad (4.22)$$

We can modify gradient-based methods (such as Algorithm 1), by replacing the gradient step with the update (4.22), to obtain training algorithms for a non-parametric model hypothesis space.

## 4.8 Overview of Coding Assignment

**Python Notebook.** `GradientMethods_CodingAssignment.ipynb`

**Data File.** `Assignment_MLBasicsData.csv`

**Description.** This assignment builds on the assignment for Lecture 2 which used ready-made Python implementations of linear regression and ridge regression. Now you have to implement ridge regression “from scratch” by applying Algorithm 1 to solve the ridge regression problem (2.24).

The features are (normalized) values of the latitude and longitude of the FMI station as well as the (normalized) year, month, day, hour and minute during which the measurement has been taken. Your tasks include

- Generate `numpy` arrays  $\mathbf{X}, \mathbf{y}$ , whose  $r$ -th row hold the features  $\mathbf{x}^{(r)}$  and label  $y^{(r)}$ , respectively, of the  $r$ -th data point in the csv file.
- Split the dataset into a training set and validation set. The size of the training set should be 100.
- Train a linear model, using the `RidgeRegression` class of the `scikit-learn` package, on the training set and determine the resulting training error and validation error.
- Implement the GD Algorithm 1 for the objective function (2.24). Use the initialization  $\mathbf{w}^{(0)} = \mathbf{0}$ .
- To develop some intuition for the behaviour of Algorithm 1, you could try out different choices for the learning rate  $\eta$  and maximum number  $k_{\max}$  of gradient steps. For each choice, you could monitor and plot the

objective function value  $f(\mathbf{w}^{(k)})$  for the model parameters as a function of the iteration counter  $k$  in Algorithm 1.

## 5 Lecture - “FL Algorithms”

February 27, 2024

Lecture 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 empirical graph. The solutions of GTVMin are local model parameters that strike a balance between the loss incurred on local datasets and the total variation. This lecture applies the gradient-based methods from Lecture 4 to solve GTVMin. The resulting FL algorithms can be implemented by message passing over the edges of the empirical graph. We will also use the analysis of gradient-based methods to study the computational aspects and statistical aspects of the resulting algorithms.

### 5.1 Learning Goals

After this lecture, you

- can apply gradient-based methods to GTVMin for local linear models,
- can implement a gradient step via message passing over empirical graphs,
- know some motivation for stochastic gradient descent (SGD),
- know how federated averaging (FedAvg) is obtained from projected GD.

### 5.2 Gradient Step for GTVMin

Consider a collection of  $n$  local datasets represented by the nodes  $\mathcal{V} = \{1, \dots, n\}$  of an empirical graph  $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ . Each undirected edge  $\{i, i'\} \in \mathcal{E}$  in empirical graph  $\mathcal{G}$  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  $i = 1, \dots, n$ . To this end, we solve the GTVMin instance

$$\{\hat{\mathbf{w}}^{(i)}\}_{i=1}^n \in \underset{\{\mathbf{w}^{(i)}\}}{\operatorname{argmin}} \underbrace{\sum_{i \in \mathcal{V}} (1/m_i) \|\mathbf{y}^{(i)} - \mathbf{X}^{(i)} \mathbf{w}^{(i)}\|_2^2 + \alpha \sum_{i, i' \in \mathcal{V}} A_{i, i'} \|\mathbf{w}^{(i)} - \mathbf{w}^{(i')}\|_2^2}_{=: f(\mathbf{w})}. \quad (5.1)$$

As discussed in Lecture 3, the objective function in (5.1) - viewed as a function of the stacked local model parameters  $\mathbf{w} := \operatorname{stack}\{\mathbf{w}^{(i)}\}_{i=1}^n$  - is a quadratic function of the form (4.1),

$$\mathbf{w}^T \underbrace{\left( \begin{pmatrix} \mathbf{Q}^{(1)} & 0 & \dots & 0 \\ 0 & \mathbf{Q}^{(2)} & \dots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \dots & \mathbf{Q}^{(n)} \end{pmatrix} + \alpha \mathbf{L} \otimes \mathbf{I} \right)}_{:= \mathbf{Q}} \mathbf{w} + \underbrace{\left( (\mathbf{q}^{(1)})^T, \dots, (\mathbf{q}^{(n)})^T \right)}_{:= \mathbf{q}^T} \mathbf{w} \quad (5.2)$$

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

Therefore, the discussion and analysis of gradient-based methods from Lecture 4 also apply to GTVMin (5.1). In particular, we use the gradient step

$$\begin{aligned} \mathbf{w}^{(k+1)} &:= \mathbf{w}^{(k)} - \eta \nabla f(\mathbf{w}^{(k)}) \\ &\stackrel{(4.1)}{=} \mathbf{w}^{(k)} - \eta (2\mathbf{Q}\mathbf{w}^{(k)} + \mathbf{q}) \end{aligned} \quad (5.3)$$

to iteratively compute an approximate solution  $\hat{\mathbf{w}}$  to (5.1). This solution consists of learnt local model parameters  $\hat{\mathbf{w}}^{(i)}$ , i.e.,  $\hat{\mathbf{w}} = \operatorname{stack}\{\hat{\mathbf{w}}^{(i)}\}$ .

According to the convergence analysis in Lecture 4, the convergence rate of the iterations (5.3) is determined by the eigenvalues of the matrix  $\mathbf{Q}$  in (5.2).



Clearly, the eigenvalues  $\lambda_j(\mathbf{Q})$  of  $\mathbf{Q}$  are related to the eigenvalues  $\lambda_j(\mathbf{Q}^{(i)})$  and to the eigenvalues  $\lambda_j(\mathbf{L})$  of the Laplacian matrix of the empirical graph  $\mathcal{G}$ . In particular, we will use the following two summary parameters

$$\lambda_{\max} := \max_{i=1,\dots,n} \lambda_d(\mathbf{Q}^{(i)}), \text{ and } \bar{\lambda}_{\min} := \lambda_1\left((1/n) \sum_{i=1}^n \mathbf{Q}^{(i)}\right). \quad (5.4)$$

We first present an upper bound  $U$  (see (4.5)) on the eigenvalues of the matrix  $\mathbf{Q}$  in (5.2).

**Proposition 5.1.** *The eigenvalues of  $\mathbf{Q}$  in (5.2) are upper-bounded as*

$$\begin{aligned} \lambda_j(\mathbf{Q}) &\leq \lambda_{\max} + \alpha \lambda_n(\mathbf{L}) \\ &\leq \underbrace{\lambda_{\max} + \alpha 2d_{\max}}_{:=U}, \text{ for } j = 1, \dots, dn. \end{aligned} \quad (5.5)$$

*Proof.* See Section 5.7.1. □

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

**Proposition 5.2.** *Consider the matrix  $\mathbf{Q}$  in (5.2). If  $\lambda_2(\mathbf{L}) > 0$  (i.e., the empirical graph in (5.1) 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*

$$\lambda_1(\mathbf{Q}) \geq \frac{1}{1 + \rho^2} \min\{\lambda_2(\mathbf{L})\alpha\rho^2, \bar{\lambda}_{\min}/2\}. \quad (5.6)$$

Here, we used the shorthand  $\rho := \bar{\lambda}_{\min}/(4\lambda_{\max})$  (see (5.4)).

*Proof.* See Section 5.7.2. □

### 5.3 Message Passing Implementation

We now discuss in more detail the implementation of gradient-based methods to solve the GTVMin instance (5.1). The core computational step is of course the gradient step

$$\mathbf{w}^{(k+1)} := \mathbf{w}^{(k)} - \eta \nabla f(\mathbf{w}^{(k)}) \text{ with } f(\mathbf{w}) \text{ defined in (5.1).} \quad (5.7)$$

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

$$\mathbf{w}^{(k)} =: \text{stack}\{\mathbf{w}^{(i,k)}\}_{i=1}^n. \quad (5.8)$$

Using (5.1), we can express the gradient  $\nabla f(\mathbf{w}^{(k)})$  and the corresponding gradient step as (see Figure 5.1)

$$\begin{aligned} \mathbf{w}^{(i,k+1)} := \mathbf{w}^{(i,k)} + \eta \left[ \underbrace{(2/m_i)(\mathbf{X}^{(i)})^T(\mathbf{y}^{(i)} - \mathbf{X}^{(i)}\mathbf{w}^{(i,k)})}_{\text{(I)}} \right. \\ \left. + \underbrace{2\alpha \sum_{i' \in \mathcal{V} \setminus \{i\}} A_{i,i'}(\mathbf{w}^{(i',k)} - \mathbf{w}^{(i,k)})}_{\text{(II)}} \right]. \quad (5.9) \end{aligned}$$

The update (5.9) consists of two components, denoted (I) and (II). The purpose of component (I) is to minimize the local loss, i.e., to learn local model parameters with small deviation between labels  $y^{(i,r)}$  and the predictions  $(\mathbf{w}^{(i,k+1)})^T \mathbf{x}^{(i,r)}$ . Note that we can rewrite the component (I) in (5.9) as

$$(2/m_i) \sum_{r=1}^{m_i} \mathbf{x}^{(i,r)} (y^{(i,r)} - (\mathbf{x}^{(i,r)})^T \mathbf{w}^{(i,k)}). \quad (5.10)$$

The purpose of component (II) in (5.9) 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  $\alpha$ : Choosing a large value for  $\alpha$  puts more emphasis on enforcing similar local model parameters across the edges. Using a smaller  $\alpha$  puts more emphasis on having accurate predictions on the local dataset.

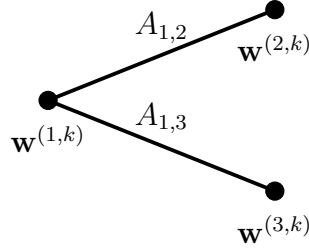


Figure 5.1: At the beginning of iteration  $k$ , node  $i = 1$  collects the current local model parameters  $\mathbf{w}^{(2,k)}$  and  $\mathbf{w}^{(3,k)}$  from its neighbours. Then, it computes the gradient step (5.9) to obtain the new local model parameters  $\mathbf{w}^{(1,k+1)}$ . These updated parameters are then used in the next iteration for the local updates at the neighbours  $i = 2, 3$ .

We obtain Algorithm 2 by repeating the gradient step, simultaneously for each node  $i \in \mathcal{V}$ , until a stopping criterion is met. Algorithm 2 allows for potentially different learning rates  $\eta_{k,i}$  at different nodes  $i$  and iterations  $k$  (see Section 4.3). We stress that Algorithm 2 requires a synchronous (simultaneous) execution of the updates (5.9) at all nodes  $i \in \mathcal{V}$ . When implementing Algorithm 2 in mobile networks there might be deviations from this synchronous operation. This deviation can be modelled as a perturbation of the gradient step (5.7) and therefore analyzed using the concepts of Section 4.5 on perturbed GD.

---

**Algorithm 2** FedGD for Local Linear Models

---

**Input:** empirical graph  $\mathcal{G}$ ; GTV parameter  $\alpha$ ; learning rate  $\eta_{k,i}$

local dataset  $\mathcal{D}^{(i)} = \{(\mathbf{x}^{(i,1)}, y^{(i,1)}), \dots, (\mathbf{x}^{(i,m_i)}, y^{(i,m_i)})\}$  for each  $i$ ; some stopping criterion.

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

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

- 1: **while** stopping criterion is not satisfied **do**
  - 2:     **for** all nodes  $i \in \mathcal{V}$  (simultaneously) **do**
  - 3:         share local model parameters  $\mathbf{w}^{(i,k)}$  with each neighbour  $i' \in \mathcal{N}^{(i)}$
  - 4:         update local model parameters via (5.9)
  - 5:     **end for**
  - 6:     increment iteration counter:  $k := k + 1$
  - 7: **end while**
  - 8:  $\hat{\mathbf{w}}^{(i)} := \mathbf{w}^{(i,k)}$  for all nodes  $i \in \mathcal{V}$
-

## 5.4 FedSGD

For some applications, it might be infeasible to compute the sum (5.10) exactly for each gradient step. For example, local datasets might consist of a large number of data points which cannot be accessed quickly enough (stored in the cloud). It might then be useful to approximate the sum by

$$\underbrace{(2/m_i) \sum_{r \in \mathcal{B}} \mathbf{x}^{(i,r)} (y^{(i,r)} - (\mathbf{x}^{(i,r)})^T \mathbf{w}^{(i)})}_{\approx (5.10)}. \quad (5.11)$$

The approximation (5.11) uses a subset (so called “batch”)

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

of  $B$  randomly chosen data points from  $\mathcal{D}^{(i)}$ . While (5.10) requires summing over  $m$  data points, the approximation requires to sum over  $B$  (typically  $B \ll m$ ) data points.

Inserting the approximation (5.11) into the gradient step (5.9) yields the approximate gradient step

$$\begin{aligned} \mathbf{w}^{(i,k+1)} := \mathbf{w}^{(i,k)} + \eta \left[ \underbrace{(2/m_i) \sum_{r \in \mathcal{B}} \mathbf{x}^{(i,r)} (y^{(i,r)} - (\mathbf{x}^{(i,r)})^T \mathbf{w}^{(i)})}_{\approx (5.10)} \right. \\ \left. + 2\alpha \underbrace{\sum_{i' \in \mathcal{V} \setminus \{i\}} A_{i,i'} (\mathbf{w}^{(i',k)} - \mathbf{w}^{(i,k)})}_{(\text{II})} \right]. \end{aligned} \quad (5.12)$$

We obtain Algorithm 3 from Algorithm 2 by replacing the gradient step (5.9) with the approximation (5.12).

---

**Algorithm 3** FedSGD for Local Linear Models

---

**Input:** empirical graph  $\mathcal{G}$ ; GTV parameter  $\alpha$ ; learning rate  $\eta_{k,i}$ ;

local datasets  $\mathcal{D}^{(i)} = \{(\mathbf{x}^{(i,1)}, y^{(i,1)}), \dots, (\mathbf{x}^{(i,m_i)}, y^{(i,m_i)})\}$  for each node  $i$ ;

batch size  $B$ ; some stopping criterion.

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

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

- 1: **while** stopping criterion is not satisfied **do**
  - 2:     **for** all nodes  $i \in \mathcal{V}$  (simultaneously) **do**
  - 3:         share local model parameters  $\mathbf{w}^{(i,k)}$  with all neighbours  $i' \in \mathcal{N}^{(i)}$
  - 4:         draw fresh batch  $\mathcal{B}^{(i)} := \{r_1, \dots, r_B\}$
  - 5:         update local model parameters via (5.12)
  - 6:     **end for**
  - 7:     increment iteration counter  $k := k + 1$
  - 8: **end while**
  - 9:  $\widehat{\mathbf{w}}^{(i)} := \mathbf{w}^{(i,k)}$  for all nodes  $i \in \mathcal{V}$
-

## 5.5 FedAvg

Consider a FL use-case that requires to learn model parameters  $\hat{\mathbf{w}} \in \mathbb{R}^d$  for a single (global) linear model from local datasets  $\mathcal{D}^{(i)}$ ,  $i = 1, \dots, n$ . How can we learn  $\hat{\mathbf{w}}$  without exchanging local datasets but only some model parameters (updates)?

One approach could be to use GTVMin (5.1) and choose  $\alpha$  such that its solutions  $\hat{\mathbf{w}}^{(i)}$  are identical for 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. According to our analysis of GTVMin in Lecture 3 (in particular, the upper bound in Prop. 3.1), we can enforce this agreement by a sufficiently large parameter  $\alpha$ .

Note that the bound in Prop. 3.1 only applies if the empirical graph (used in GTVMin) is connected. One example for a connected empirical graph is the star as depicted in Figure 5.2. 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  $i = 2, \dots, n$ . The star graph is distinct in the sense of using the minimum number of edges required to connect  $n$  nodes [46].



Figure 5.2: Star graph  $\mathcal{G}^{(\text{star})}$  with centre node representing a server and peripheral nodes representing clients that generate local datasets.

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

$$\widehat{\mathbf{w}} \in \arg \min_{\mathbf{w} \in \mathcal{C}} \sum_{i \in \mathcal{V}} (1/m_i) \|\mathbf{y}^{(i)} - \mathbf{X}^{(i)} \mathbf{w}^{(i)}\|_2^2$$

with  $\mathcal{C} = \{\mathbf{w} = \text{stack}\{\mathbf{w}^{(i)}\}_{i=1}^n : \mathbf{w}^{(i)} = \mathbf{w}^{(i')} \text{ for any } i, i' \in \mathcal{V}\}.$  (5.13)

Note that the constraint set  $\mathcal{C}$  is nothing but the subspace defined in (3.13). The projection of a given collection of local model parameters  $\mathbf{w} = \text{stack}\{\mathbf{w}^{(i)}\}$  on  $\mathcal{C}$  is given by

$$P_{\mathcal{C}}(\mathbf{w}) = (\mathbf{v}^T, \dots, \mathbf{v}^T)^T \text{ with } \mathbf{v} := (1/n) \sum_{i \in \mathcal{V}} \mathbf{w}^{(i)}. \quad (5.14)$$

We can solve (5.13) using projected GD from Lecture 4 (see Section 4.6). The resulting projected gradient step for solving (5.13) is

$$\widehat{\mathbf{w}}_k^{(i)} := \underbrace{\mathbf{w}^{(i,k)} + \eta_{i,k} (2/m_i) (\mathbf{X}^{(i)})^T (\mathbf{y}^{(i)} - \mathbf{X}^{(i)} \mathbf{w}^{(i,k)})}_{\text{(local gradient step)}} \quad (5.15)$$

$$\mathbf{w}^{(i,k+1)} := (1/n) \sum_{i' \in \mathcal{V}} \widehat{\mathbf{w}}_k^{(i')} \quad (\text{projection}). \quad (5.16)$$

We can implement this iteration of projected GD conveniently in a server-client system with each node  $i$  being a client. Indeed, the first step (5.15) implements, separately for each node  $i$ , a gradient step towards a minimum of the local loss  $\|\mathbf{y}^{(i)} - \mathbf{X}^{(i)} \mathbf{w}\|_2^2$ . Each node  $i$  sends the result  $\widehat{\mathbf{w}}_k^{(i)}$  of its local gradient step to a server. After receiving these vectors from all nodes  $i \in \mathcal{V}$ , the server computes the projection step (5.16). This projection results in the new local model parameters  $\mathbf{w}^{(i,k+1)}$  that are sent back to each client  $i$ .

The averaging step (5.16) might take much longer to execute than the local update step (5.15). Indeed, (5.16) 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 (5.15) it must wait until the server (i) has collected the updates  $\widehat{\mathbf{w}}_k^{(i)}$  from all clients and (ii) sent back their average  $\mathbf{w}^{(i,k+1)}$  to  $i \in \mathcal{V}$ .

Instead of using a computational cheap gradient step (5.15),<sup>5</sup> and then being forced to wait for receiving  $\mathbf{w}^{(i,k+1)}$  back from the server, a client might “make better use of the time”. For example, the client  $i$  could execute several local gradient steps (5.15) in order to make more progress towards the optimum. Another option is to use a local minimization of  $L_i(\mathbf{v}) := (1/m_i) \|\mathbf{y}^{(i)} - \mathbf{X}^{(i)}\mathbf{v}\|_2^2$  around  $\mathbf{w}^{(i,k)}$ ,

$$\widehat{\mathbf{w}}_k^{(i)} := \operatorname{argmin}_{\mathbf{v} \in \mathbb{R}^d} \left[ (1/m_i) \underbrace{\|\mathbf{y}^{(i)} - \mathbf{X}^{(i)}\mathbf{v}\|_2^2}_{=L_i(\mathbf{v})} + (1/\eta) \|\mathbf{v} - \mathbf{w}^{(i,k)}\|_2^2 \right]. \quad (5.17)$$

Note that (5.17) is nothing but the proximity operator of  $L_i(\mathbf{v})$  (see (4.19)). We obtain Algorithm 4 from (5.15)-(5.16) by replacing the gradient step (5.15) with the local minimization (5.17).

The local minimization step (5.17) is closely related to ridge regression. Indeed, (5.17) is obtained from ridge regression (2.22) by replacing the regularizer  $\mathcal{R}\{\mathbf{v}\} := \|\mathbf{v}\|_2^2$  with the regularizer  $\mathcal{R}\{\mathbf{v}\} := \|\mathbf{v} - \widehat{\mathbf{w}}_k^{(i)}\|_2^2$ .

As the notation in (5.17) indicates, the parameter  $\eta$  plays a role similar to the learning rate of a gradient step (4.2). It controls the size of the neighbourhood of  $\mathbf{w}^{(i,k)}$  over which (5.17) optimizes the local loss function  $L_i(\cdot)$ . Choosing a small  $\eta$  forces the update (5.17) to not move too far from

---

<sup>5</sup>For a large local dataset, the local gradient step (5.15) might actually be computationally expensive and should be replaced by an approximation, e.g., based on the stochastic gradient approximation (5.11).

the given model parameters  $\mathbf{w}^{(i,k)}$ .

We can interpret the update (5.17) as a form of ERM (2.3) for linear regression. Indeed,

$$\begin{aligned} & \min_{\mathbf{v} \in \mathbb{R}^d} \left[ (1/m_i) \|\mathbf{y}^{(i)} - \mathbf{X}^{(i)} \mathbf{v}\|_2^2 + (1/\eta) \|\mathbf{w}^{(i,k)} - \mathbf{v}\|_2^2 \right] = \\ & \min_{\mathbf{v} \in \mathbb{R}^d} \left[ (1/m_i) \sum_{r=1}^{m_i} (y^{(i,r)} - (\mathbf{x}^{(i,r)})^T \mathbf{v})^2 + (1/\eta) \|\mathbf{w}^{(i,k)} - \mathbf{v}\|_2^2 \right] = \\ & \min_{\mathbf{v} \in \mathbb{R}^d} (1/m_i) \left[ \sum_{r=1}^{m_i} \left( y^{(i,r)} - (\mathbf{x}^{(i,r)})^T \mathbf{v} \right)^2 + \sum_{r=1}^d (m_i/\eta) \left( w_r^{(i,k)} - (\mathbf{e}^{(r)})^T \mathbf{v} \right)^2 \right]. \quad (5.18) \end{aligned}$$

Here,  $\mathbf{e}^{(r)}$  denotes the vector obtained from the  $r$ th column of the identity matrix  $\mathbf{I}$ . The only difference between (5.18) and (2.3) is the presence of the sample weights  $(m_i/\eta)$ .<sup>6</sup>

---

<sup>6</sup>The `.fit()` method of the Python class `sklearn.linear_model.LinearRegression` allows to specify sample weights via the parameter `sample_weight` [26].

---

**Algorithm 4** FedAvg to train a linear model

---

**Input:** client list  $\mathcal{V}$

**Server. Initialize.**  $k := 0$

- 1: **while** stopping criterion is not satisfied **do**
- 2:   receive local model parameters  $\mathbf{w}^{(i)}$  from all clients  $i \in \mathcal{V}$
- 3:   update global model parameters

$$\bar{\mathbf{w}}[k] := (1/|\mathcal{V}|) \sum_{i \in \mathcal{V}} \mathbf{w}^{(i)}.$$

- 4:   send updated global model parameters  $\bar{\mathbf{w}}[k]$  to all clients  $i \in \mathcal{V}$
- 5:    $k := k + 1$
- 6: **end while**

**Client.** (at some node  $i \in \mathcal{V}$ )

- 1: **while** stopping criterion is not satisfied **do**
- 2:   receive global model parameters  $\bar{\mathbf{w}}[k]$  from server
- 3:   update local model parameters by RERM (see (5.17))

$$\mathbf{w}^{(i)} := \operatorname{argmin}_{\mathbf{v} \in \mathbb{R}^d} \left[ (1/m_i) \sum_{r=1}^{m_i} (\mathbf{v}^T \mathbf{x}^{(i,r)} - y^{(i,r)})^2 + (1/\eta) \|\mathbf{v} - \bar{\mathbf{w}}[k]\|_2^2 \right].$$

- 4:   send  $\mathbf{w}^{(i)}$  back to server
  - 5: **end while**
-

## 5.6 Overview of Coding Assignment

**Python Notebook.** FLAlgorithms\_CodingAssignment.ipynb

**Data File.** Assignment\_MLBasicsData.csv

This coding assignment builds on the coding assignment “ML Basics” (see Section 2.7) and the coding assignment “FL Design Principle” (see Section 3.4). In particular, we consider an empirical graph  $\mathcal{G}^{(\text{FMI})}$  with each node  $i \in \mathcal{V}$  being a FMI weather station. The node  $i \in \mathcal{V}$  holds a local dataset  $\mathcal{D}^{(i)}$  that consists of  $m_i$  data points. Each data point is a temperature measurement, taken at station  $i$ , and characterized by  $d = 7$  features  $\mathbf{x} = (x_1, \dots, x_7)^T$  and a label  $y$  which is the temperature measurement itself. The features are (normalized) values of the latitude and longitude of the FMI station as well as the (normalized) year, month, day, hour, minute when the temperature has been measured.

The edges of  $\mathcal{G}^{(\text{FMI})}$  are obtained using the `Python` function `add_edges()`. Each FMI station  $i$  is connected to its nearest neighbours  $i'$ , using the Euclidean distance between the corresponding vectors  $(\text{lat}^{(i)}, \text{lon}^{(i)})^T \in \mathbb{R}^2$ . The number of neighbours is controlled by the input parameter `numneighbors`. All edges  $\{i, i'\} \in \mathcal{E}$  have the same edge weight  $A_{i,i'} = 1$ .

Your tasks involve

- Construct the empirical graph  $\mathcal{G}^{(\text{FMI})}$  as a `networkx.Graph()` object.
- For each node  $i \in \mathcal{V}$  (FMI station), add node attributes that store the feature matrix (as `numpy` arrays)  $\mathbf{X}^{(i)}$  and label vector  $\mathbf{y}^{(i)}$ . The  $r$ -th row of these holds the features  $\mathbf{x}^{(i,r)}$  and label  $y^{(i,r)}$ , respectively, of the

$r$ -th data point recorded for FMI station  $i$  in the csv file.

- For each node  $i \in \mathcal{V}$  (FMI station), add a node attribute that stores local model parameters  $\mathbf{w}^{(i)}$  (as a `numpy` array).
- Split each local dataset into a training set and a validation set.
- Learn local model parameters for a linear model by applying FedGD and FedSGD to the local training sets. Diagnose the resulting model parameters by computing the average (over all nodes) training error and validation error.
- Try to develop some intuition for the effect of choosing different values for the hyper-parameters such as GTVMin parameter  $\alpha$ , learning rate  $\eta$  or batch size in Algorithms 2 and 3.

## 5.7 Proofs

### 5.7.1 Proof of Proposition 5.1

The first inequality in (5.5) follows from well-known results on the eigenvalues of a sum of symmetric matrices (see, e.g., [34, Thm 8.1.5]). In particular,

$$\lambda_{\max}(\mathbf{Q}) \leq \max \left\{ \underbrace{\max_{i=1,\dots,n} \lambda_d(\mathbf{Q}^{(i)})}_{\stackrel{(5.4)}{=} \lambda_{\max}}, \lambda_{\max}(\alpha \mathbf{L} \otimes \mathbf{I}) \right\}. \quad (5.19)$$

The second inequality in (5.5) uses the following upper bound on the maximum eigenvalue  $\lambda_n(\mathbf{L})$  of the Laplacian matrix:

$$\begin{aligned} \lambda_n(\mathbf{L}) &\stackrel{(a)}{=} \max_{\|\mathbf{w}\|_2^2=1} \mathbf{w}^T \mathbf{L} \mathbf{w} \\ &\stackrel{(3.8)}{=} \max_{\|\mathbf{w}\|_2^2=1} \sum_{\{i,i'\} \in \mathcal{E}} A_{i,i'} (w_i - w_{i'})^2 \\ &\stackrel{(b)}{\leq} \max_{\|\mathbf{w}\|_2^2=1} \sum_{\{i,i'\} \in \mathcal{E}} A_{i,i'} 2(w_i^2 + w_{i'}^2) \\ &\stackrel{(c)}{=} \max_{\|\mathbf{w}\|_2^2=1} \sum_{i \in \mathcal{V}} 2w_i^2 \sum_{i' \in \mathcal{N}(i)} A_{i,i'} \\ &\stackrel{(3.5)}{\leq} \max_{\|\mathbf{w}\|_2^2=1} \sum_{i \in \mathcal{V}} 2w_i^2 d_{\max} \\ &= 2d_{\max}. \end{aligned} \quad (5.20)$$

Here, step (a) uses Courant–Fischer–Weyl [34, Thm 8.1.2] and step (b)) uses the inequality  $(u+v)^2 \leq 2(u^2+v^2)$ , which is valid for any  $u, v \in \mathbb{R}$ . The bound (5.20) is essentially tight.<sup>7</sup>

---

<sup>7</sup>Consider an empirical graph being a chain (or path).



Figure 5.3: Illustration of step (c) in (5.20).

### 5.7.2 Proof of Proposition 5.2

Similar to the upper bound (5.20) we also start with the Courant–Fischer–Weyl min-max characterization for the eigenvalues of  $\mathbf{Q}$  in (5.2). In particular,

$$\lambda_1 = \min_{\|\mathbf{w}\|_2^2=1} \mathbf{w}^T \mathbf{Q} \mathbf{w}. \quad (5.21)$$

We next analyze the RHS of (5.21) by partitioning the constraint set  $\{\mathbf{w} : \|\mathbf{w}\|_2^2 = 1\}$  of (5.21) 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

$$\mathbf{w} = \bar{\mathbf{w}} + \tilde{\mathbf{w}}. \quad (5.22)$$

The component  $\bar{\mathbf{w}} = \text{stack}\{\mathbf{c}\}_{i=1}^n$  consists of identical local model parameters  $\mathbf{c} = (1/n) \sum_{i=1}^n \mathbf{w}^{(i)}$ , for each node  $i \in \mathcal{V}$ , and is the orthogonal projection of  $\mathbf{w}$  on the subspace (3.13). The second component  $\tilde{\mathbf{w}}$  is the orthogonal projection on the complement of (3.13). Note that

$$\|\mathbf{w}\|_2^2 = \|\bar{\mathbf{w}}\|_2^2 + \|\tilde{\mathbf{w}}\|_2^2. \quad (5.23)$$

**Regime I.** This regime is obtained for  $\|\tilde{\mathbf{w}}\|_2 \geq \rho \|\bar{\mathbf{w}}\|_2$ . Since  $\|\mathbf{w}\|_2^2 = 1$ , and due to (5.23), we have

$$\|\tilde{\mathbf{w}}\|_2^2 \geq \rho^2 / (1 + \rho^2). \quad (5.24)$$

This implies, in turn, via (3.12) that

$$\begin{aligned}\mathbf{w}^T \mathbf{Q} \mathbf{w} &\stackrel{(5.2)}{\geq} \alpha \mathbf{w}^T (\mathbf{L} \otimes \mathbf{I}) \mathbf{w} \\ &\stackrel{(5.24)}{\geq} \lambda_2(\mathbf{L}) \alpha \rho^2 / (1 + \rho^2).\end{aligned}\tag{5.25}$$

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

$$n \|\mathbf{c}\|_2^2 = \|\bar{\mathbf{w}}\|_2^2 > 1/(1 + \rho^2).\tag{5.26}$$

We next develop the RHS of (5.21) according to

$$\begin{aligned}\mathbf{w}^T \mathbf{Q} \mathbf{w} &\stackrel{(5.2)}{\geq} \sum_{i=1}^n (\mathbf{w}^{(i)})^T \mathbf{Q}^{(i)} \mathbf{w}^{(i)} \\ &\stackrel{(5.22)}{\geq} \sum_{i=1}^n (\mathbf{c} + \tilde{\mathbf{w}}^{(i)})^T \mathbf{Q}^{(i)} (\mathbf{c} + \tilde{\mathbf{w}}^{(i)}) \\ &\stackrel{(5.26)}{\geq} \underbrace{\|\bar{\mathbf{w}}\|_2^2 \lambda_1 \left( (1/n) \sum_{i=1}^n \mathbf{Q}^{(i)} \right)}_{\bar{\lambda}_{\min}} + \sum_{i=1}^n \left[ 2(\tilde{\mathbf{w}}^{(i)})^T \mathbf{Q}^{(i)} \mathbf{c} + \underbrace{(\tilde{\mathbf{w}}^{(i)})^T \mathbf{Q}^{(i)} \tilde{\mathbf{w}}^{(i)}}_{\geq 0} \right] \\ &\geq \|\bar{\mathbf{w}}\|_2^2 \bar{\lambda}_{\min} + \sum_{i=1}^n 2(\tilde{\mathbf{w}}^{(i)})^T \mathbf{Q}^{(i)} \mathbf{c}.\end{aligned}\tag{5.27}$$

To develop (5.27) further, we note that

$$\begin{aligned}\left| \sum_{i=1}^n 2(\tilde{\mathbf{w}}^{(i)})^T \mathbf{Q}^{(i)} \mathbf{c} \right| &\stackrel{(a)}{\leq} 2\lambda_{\max} \|\tilde{\mathbf{w}}\|_2 \|\bar{\mathbf{w}}\|_2 \\ &\stackrel{\|\tilde{\mathbf{w}}\|_2 < \rho \|\bar{\mathbf{w}}\|_2}{\leq} 2\lambda_{\max} \rho \|\bar{\mathbf{w}}\|_2^2.\end{aligned}\tag{5.28}$$

Here, step (a) follows from  $\max_{\|\mathbf{y}\|_2=1, \|\mathbf{x}\|_2=1} \mathbf{y}^T \mathbf{Q} \mathbf{x} = \lambda_{\max}$ . Inserting (5.28) into (5.27) for  $\rho = \bar{\lambda}_{\min}/(4\lambda_{\max})$ ,

$$\mathbf{w}^T \mathbf{Q} \mathbf{w} \geq \|\bar{\mathbf{w}}\|_2^2 \bar{\lambda}_{\min} / 2 \stackrel{(5.26)}{\geq} (1/(1 + \rho^2)) \bar{\lambda}_{\min} / 2\tag{5.29}$$

For each  $\mathbf{w}$  with  $\|\mathbf{w}\|_2^2 = 1$ , either (5.25) or (5.29) must hold.



## 6 Lecture - “FL Main Flavours”

February 27, 2024

Lecture 3 discussed GTVMin as a main design principle for FL algorithms. Lecture 5 showed how to obtain practical FL algorithms by applying optimization methods, such as those from Lecture 4, to solve GTVMin. This lecture discusses some important special cases (“main flavours”) of GTVMin that are obtained for specific design choices for the components of GTVMin. These include the construction of local datasets, the choice of local models and, last but not least, the weighted edges in the empirical graph.

### 6.1 Learning Goals

After this lecture, you should know about the following main flavours of FL:

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

### 6.2 Single Model FL

Some FL use cases require to train a single (global) model  $\mathcal{H}$  from a decentralized collection of local datasets  $\mathcal{D}^{(i)}$ ,  $i = 1, \dots, n$  [14, 47]. In what follows we assume that the model  $\mathcal{H}$  is parametrized by a vector  $\mathbf{w} \in \mathbb{R}^d$ . Figure 6.2 depicts a server-client architecture for an iterative FL algorithm that

generates a sequence of (global) model parameters  $\mathbf{w}^{(k)}$ ,  $k = 1, \dots$ . After computing the new model parameters  $\mathbf{w}^{(k+1)}$ , the server broadcasts it to the clients  $i$ . During the next iteration, each client  $i$  uses the current global model parameters  $\mathbf{w}^{(k)}$  to compute a local update  $\mathbf{w}^{(i,k)}$  based on its local dataset  $\mathcal{D}^{(i)}$ . The precise implementation of this local update step depends on the choice for the global model  $\mathcal{H}$  (trained by the server). One example for such a local update has been discussed in Lecture 5 (see (5.17)).

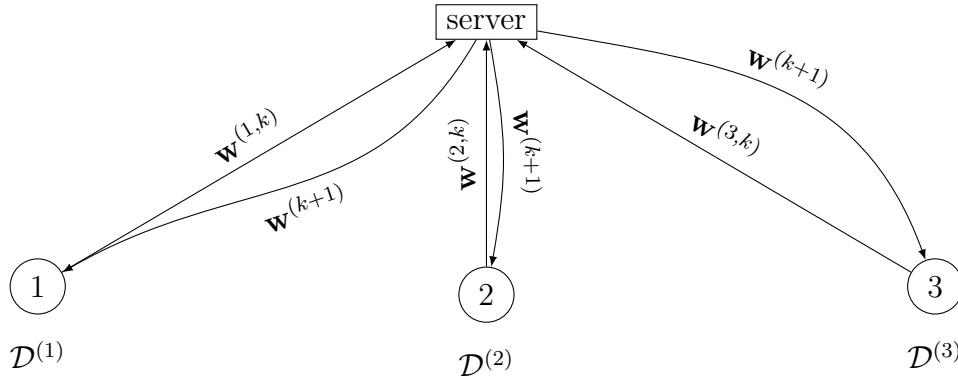


Figure 6.1: A server-based FL system that learns model parameters  $\mathbf{w}^{(k+1)}$  by aggregating the updates  $\mathbf{w}^{(i,k)}$  received from every client  $i$ . The client  $i$  computes the update  $\mathbf{w}^{(i,k)}$  by combining the previous model parameters  $\mathbf{w}^{(k)}$  (received from the server) and its local dataset  $\mathcal{D}^{(i)}$ .

Lecture 5 already hinted at an alternative to the server-client architecture in Figure 6.2. Indeed, we might learn local model parameters  $\mathbf{w}^{(i)}$  for each client  $i$  using a GTVMin-based method, but force them to be (approximately) identical. We can achieve this by using a connected empirical graph and a sufficiently large GTVMin parameter  $\alpha$ . To minimize the computational

complexity of the resulting FL algorithms, we might prefer empirical graphs with small number of edges such as the star graph in Figure 5.2 [46]. However, to increase the robustness against node/link failures we might prefer using an empirical graph that has more edges.

### 6.3 Clustered FL

Single-model FL systems require the local datasets to be well approximated as i.i.d. realizations from a common underlying probability distribution. However, requiring homogeneous local datasets, generated from the same probability distribution might be overly restrictive. Indeed, the local datasets might be heterogeneous and need to be modelled using different probability distribution [17, 32].

Clustered FL relaxes the requirement of a common probability distribution underlying all local datasets. Instead, we might be able to approximate some of the local datasets as i.i.d. realizations from a common probability distribution. These local datasets form a cluster  $\mathcal{C} \subseteq \mathcal{V}$  with 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  $\hat{\mathbf{w}}^{(\mathcal{C})}$ . Each node  $i \in \mathcal{C}$  then uses these learnt model parameters  $\hat{\mathbf{w}}^{(\mathcal{C})}$ . A main challenge in CFL is that the cluster assignments of the local datasets are unknown in general.

To determine a cluster  $\mathcal{C}$ , we could apply basic clustering methods, such as  $k$ -means or Gaussian mixture model (GMM) to vector representations for local datasets [4, Ch. 5]. We can obtain a vector representation for local dataset  $\mathcal{D}^{(i)}$  via the learnt model parameters  $\hat{\mathbf{w}}$  of some parametric ML model

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

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

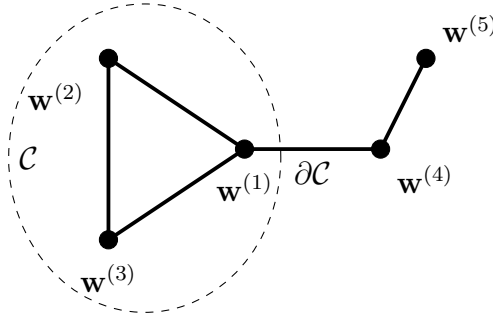


Figure 6.2: The solutions of GTVMin are local model parameters that are approximately identical for all nodes in a tight-knit cluster  $\mathcal{C}$ .

Lecture 3 discussed how the eigenvalues of the Laplacian matrix can be used to measure the connectivity of  $\mathcal{G}$ . In a similar spirit, we measure the connectivity of a cluster  $\mathcal{C}$  via the eigenvalues of the Laplacian matrix  $\mathbf{L}^{(\mathcal{C})}$  of the induced sub-graph  $\mathcal{G}^{(\mathcal{C})}$ .<sup>8</sup>

The larger  $\lambda_2(\mathbf{L}^{(\mathcal{C})})$ , the better the connectivity among the nodes in  $\mathcal{C}$ . While  $\lambda_2(\mathbf{L}^{(\mathcal{C})})$  describes the intrinsic connectivity of a cluster  $\mathcal{C}$ , we also need to characterize its connectivity with the other nodes in the empirical graph.

---

<sup>8</sup>The graph  $\mathcal{G}^{(\mathcal{C})}$  consists of the nodes in  $\mathcal{C}$  and the edges  $\{i, i'\} \in \mathcal{E}$  for  $i, i' \in \mathcal{C}$ .

To this end, we will use cluster boundary

$$|\partial\mathcal{C}| := \sum_{\{i,i'\} \in \partial\mathcal{C}} A_{i,i'} \text{ with } \partial\mathcal{C} := \{\{i,i'\} \in \mathcal{E} : i \in \mathcal{C}, i' \notin \mathcal{C}\}. \quad (6.1)$$

Note that for a single-node cluster  $\mathcal{C} = \{i\}$ , the cluster boundary coincides with the node degree,  $|\partial\mathcal{C}| = d^{(i)}$  (see (3.4)).

Intuitively, GTVMin tends to deliver (approximately) identical model parameters  $\mathbf{w}^{(i)}$  for nodes  $i \in \mathcal{C}$  if  $\lambda_2(\mathbf{L}^{(\mathcal{C})})$  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 (5.1) for local linear models.

**Proposition 6.1.** *Consider an empirical graph  $\mathcal{G}$  which contains a cluster  $\mathcal{C}$  of local datasets with labels  $\mathbf{y}^{(i)}$  and feature matrix  $\mathbf{X}^{(i)}$  related via*

$$\mathbf{y}^{(i)} = \mathbf{X}^{(i)} \bar{\mathbf{w}}^{(\mathcal{C})} + \boldsymbol{\varepsilon}^{(i)}, \text{ for all } i \in \mathcal{C}. \quad (6.2)$$

*We learn local model parameters  $\hat{\mathbf{w}}^{(i)}$  via solving GTVMin (5.1). The error component*

$$\tilde{\mathbf{w}}^{(i)} := \hat{\mathbf{w}}^{(i)} - (1/|\mathcal{C}|) \sum_{i \in \mathcal{C}} \hat{\mathbf{w}}^{(i)} \quad (6.3)$$

*is upper bounded as*

$$\sum_{i \in \mathcal{C}} \|\tilde{\mathbf{w}}^{(i)}\|_2^2 \leq \frac{1}{\alpha \lambda_2(\mathbf{L}^{(\mathcal{C})})} \left[ \sum_{i \in \mathcal{C}} (1/m_i) \|\boldsymbol{\varepsilon}^{(i)}\|_2^2 + \alpha |\partial\mathcal{C}| 2 \left( \|\bar{\mathbf{w}}^{(\mathcal{C})}\|_2^2 + R^2 \right) \right]. \quad (6.4)$$

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

*Proof.* See Sec. 6.8.1. □

Note that Prop. 6.1 covers single model FL from Section 6.2 as the extreme where all nodes belong to the same cluster  $\mathcal{C} = \mathcal{V}$ . Trivially, the cluster boundary is then equal to 0 and the bound (6.4) specializes to (3.27).

## 6.4 Horizontal FL

Horizontal FL uses local datasets  $\mathcal{D}^{(i)}$ , for  $i \in \mathcal{V}$ , that contain data points characterized by the same features [48]. We can think of each local dataset  $\mathcal{D}^{(i)}$  as being a subset (or batch) of an underlying global dataset

$$\mathcal{D}^{(\text{global})} := \{(\mathbf{x}^{(1)}, y^{(1)}), \dots, (\mathbf{x}^{(m)}, y^{(m)})\}.$$

In particular, local dataset  $\mathcal{D}^{(i)}$  is constituted by the data points of  $\mathcal{D}^{(\text{global})}$  with indices in  $\{r_1, \dots, r_{m_i}\}$ ,

$$\mathcal{D}^{(i)} := \{(\mathbf{x}^{(r_1)}, y^{(r_1)}), \dots, (\mathbf{x}^{(r_{m_i})}, y^{(r_{m_i})})\}.$$



Figure 6.3: Horizontal FL uses the same features to characterize data points in different local datasets. Different local datasets are constituted by different subsets of an underlying global dataset.

We can interpret horizontal FL as a generalization of semi-supervised learning (SSL) [49]: 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 empirical graph. To implement

SSL, we can solve GTVMin using trivial loss function  $L_i(\mathbf{w}^{(i)}) = 0$  for each “unlabelled” node  $i \in \mathcal{U}$ .

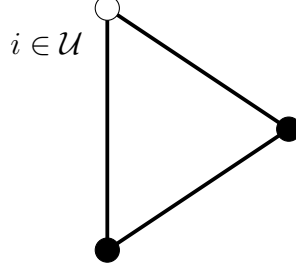


Figure 6.4: A special case of horizontal FL is a form of SSL: Some local datasets  $i \in \mathcal{U}$  do not contain labels and therefore we must use a trivial loss function  $L_i(\cdot) = 0$ . However, we can still use their features to construct an empirical graph for GTVMin based methods. These methods can learn model parameters also for  $i \in \mathcal{U}$  by combining the information in local datasets  $i \notin \mathcal{U}$  and their connections with nodes in  $\mathcal{U}$ .

## 6.5 Vertical FL

Vertical FL uses local datasets that are constituted by the same (identical!) data points. However, each local dataset uses a different choice of features to characterize these data points [50]. Formally, vertical FL applications revolve around an underlying global dataset

$$\mathcal{D}^{(\text{global})} := \{(\mathbf{x}^{(1)}, y^{(1)}), \dots, (\mathbf{x}^{(m)}, y^{(m)})\}.$$

Each data point in the global dataset is characterized by  $d'$  features  $\mathbf{x}^{(r)} = (x_1^{(r)}, \dots, x_{d'}^{(r)})^T$ . The global dataset can only be accessed indirectly via local datasets that use different subsets of the feature vectors  $\mathbf{x}^{(r)}$ .

Formally, the local dataset  $\mathcal{D}^{(i)}$  contains the pairs  $(\mathbf{x}^{(i,r)}, y^{(i,r)})$ , for  $r = 1, \dots, m$ . The labels  $y^{(i,r)}$  are identical to the labels in the global dataset,  $y^{(i,r)} = y^{(r)}$ . The feature vectors  $\mathbf{x}^{(i,r)}$  are obtained by a subset  $\mathcal{F}^{(i)} := \{j_1, \dots, j_d\}$  of the original  $d'$  features in  $\mathbf{x}^{(r)}$ ,

$$\mathbf{x}^{(i,r)} = (x_{j_1}^{(r)}, \dots, x_{j_d}^{(r)})^T.$$

## 6.6 Personalized Federated Learning

Consider GTVMin (3.14) for learning local model parameters  $\widehat{\mathbf{w}}^{(i)}$  for each local dataset  $\mathcal{D}^{(i)}$ . If the value of  $\alpha$  in (3.14) 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 (3.14).

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



$$\begin{array}{c}
\mathcal{D}^{(1)} \quad \mathcal{D}^{(i)} \\
\left[ \begin{array}{ccccc}
x_1^{(1)} & x_2^{(1)} & \cdots & x_d^{(1)} & y^{(1)} \\
x_1^{(2)} & x_2^{(2)} & \cdots & x_d^{(2)} & y^{(2)} \\
\vdots & \vdots & \ddots & \vdots & \vdots \\
x_1^{(m)} & x_2^{(m)} & \cdots & x_d^{(m)} & y^{(m)}
\end{array} \right] \\
\mathcal{D}^{(\text{global})}
\end{array}$$

Figure 6.5: Vertical FL uses local datasets that consist of the same (identical!) data points. However, local datasets use different features to characterize data points.

identical while the parameters of the deeper layers might be different for each local dataset.

The partial parameter sharing for local models can be implemented in many different ways [51, Sec. 4.3.]. One way is to use a choice for the GTV penalty function that is different from  $\phi = \|\mathbf{w}^{(i)} - \mathbf{w}^{(i')}\|_2^2$ , which is the main choice in our course. In particular, we could construct the penalty function as a combination of two terms,

$$\phi(\mathbf{w}^{(i)} - \mathbf{w}^{(i')}) := \alpha^{(1)} \phi^{(1)}(\mathbf{w}^{(i)} - \mathbf{w}^{(i')}) + \alpha^{(2)} \phi^{(2)}(\mathbf{w}^{(i)} - \mathbf{w}^{(i')}). \quad (6.5)$$

Each component  $\phi^{(1)}$ ,  $\phi^{(2)}$  measures different components of the variation  $\mathbf{w}^{(i)} - \mathbf{w}^{(i')}$  of local model parameters at connected nodes  $\{i, i'\} \in \mathcal{E}$ .

Moreover, we might use different regularization strengths  $\alpha^{(1)}$  and  $\alpha^{(2)}$  for different penalty components in (6.5) to enforce different subsets of the

model parameters to be clustered with different granularity (cluster size). For local models being deep ANNs, we might want to enforce the low-level layers (closer to the input) to have same model parameters (weights and bias terms), while deeper (closer to the output) layers can have different model parameters. Figure 6.6 illustrates this setting for local models constituted by ANNs with a single hidden layer. Yet another technique for partial sharing of model parameters is to train a hyper-model which, in turn, is used to initialize the training of local models [52].



Figure 6.6: Personalized FL with local models being ANNs with one hidden layer. The ANN  $h^{(i)}$  is parametrized by the vector  $\mathbf{w}^{(i)} = \left( (\mathbf{u}^{(i)})^T, (\mathbf{v}^{(i)})^T \right)^T$ , with parameters  $\mathbf{u}^{(i)}$  of 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  $\phi = \|\mathbf{u}^{(i)} - \mathbf{u}^{(i')}\|_2^2$ .

## 6.7 Overview of Coding Assignment

**Python Notebook.** FLFlavors\_CodingAssignment.ipynb

**Data File.** Assignment\_MLBasicsData.csv

This coding assignment builds on the coding assignment “ML Basics” (see Section 2.7) and the coding assignment “FL Design Principle” (see Section 3.4). In particular, we consider an empirical graph  $\mathcal{G}^{(\text{FMI})}$  with each node  $i \in \mathcal{V}$  being a FMI weather station. The node  $i \in \mathcal{V}$  holds a local dataset

$$\mathcal{D}^{(i)} = \left\{ (\mathbf{x}^{(i,1)}, y^{(i,1)}), \dots, (\mathbf{x}^{(i,m_i)}, y^{(i,m_i)}) \right\}$$

that consists of  $m_i$  data points. Each data point is a temperature measurement, taken at station  $i$ , and characterized by  $d = 7$  features  $\mathbf{x} = (x_1, \dots, x_7)^T$  and a label  $y$  which is the temperature measurement itself. The features are (normalized) values of the latitude and longitude of the FMI station as well as the (normalized) year, month, day, hour, minute during which this measurements has been taken.

The edges of  $\mathcal{G}^{(\text{FMI})}$  are obtained using the `Python` function `add_edges()`. Each FMI station  $i$  is connected to its nearest neighbours  $i'$ , using the Euclidean distance between the corresponding vectors  $(\text{lat}^{(i)}, \text{lon}^{(i)})^T \in \mathbb{R}^2$ . The number of neighbours is controlled by the input parameter `numneighbors`. All edges  $\{i, i'\} \in \mathcal{E}$  have the same edge weight  $A_{i,i'} = 1$ .

Your tasks involve

- Construct the empirical graph  $\mathcal{G}^{(\text{FMI})}$  as a `networkx.Graph()` object.
- For each node  $i \in \mathcal{V}$  (FMI station), add node attributes that store the feature matrix (as `numpy` arrays)  $\mathbf{X}^{(i)}$  and label vector  $\mathbf{y}^{(i)}$ . The  $r$ -th

row of these holds the features  $\mathbf{x}^{(i,r)}$  and label  $y^{(i,r)}$ , respectively, of the  $r$ -th data point recorded for FMI station  $i$  in the csv file. Add another node attribute that stores the index of the cluster to which this node belongs to.

- Cluster the nodes of  $\mathcal{G}^{(\text{FMI})}$  using the Python class `sklearn.cluster.KMeans` which implements the  $k$ -means clustering method (see [4, Ch. 8]). This clustering method requires a vector representation  $\mathbf{z}^{(i)}$  for each local dataset  $\mathcal{D}^{(i)}$ . You should try out different choices for the vector representation  $\mathbf{z}^{(i)}$  of a local dataset  $\mathcal{D}^{(i)}$ :
  - $\mathbf{z}^{(i)} \in \mathbb{R}^2$  with entries being the latitude and longitude of the FMI station  $i \in \mathcal{V}$ .
  - $\mathbf{z}^{(i)}$  is obtained from stacking the parameters of a GMM fitted to the feature vectors  $\mathbf{x}^{(i,1)}, \dots, \mathbf{x}^{(i,m_i)}$ .
  - $\mathbf{z}^{(i)} = (u_i^{(1)}, u_i^{(2)}, \dots, u_i^{(k)})^T$  given by the  $i$ -th entries of the eigenvectors  $\mathbf{u}^{(1)}, \mathbf{u}^{(2)}, \dots, \mathbf{u}^{(k)}$  of the Laplacian matrix  $\mathbf{L}^{(\text{FMI})}$ . Note that  $\mathbf{u}^{(j)}$  is an eigenvector corresponding to some  $j$ -th smallest eigenvalue  $\lambda_j$  of  $\mathbf{L}^{(\text{FMI})}$  (see (3.9)).
- For each clustering obtained in the previous task, compute the cluster-wise average temperature  $\hat{y}^{(\mathcal{C})} = \frac{1}{\sum_{i' \in \mathcal{C}} m_{i'}} \sum_{i' \in \mathcal{C}} \sum_{r=1}^{m_{i'}} y^{(i',r)}$  for each cluster  $\mathcal{C}$ . Let  $\mathcal{C}^{(i)}$  denote the cluster to which node  $i$  belongs, then  $\hat{y}^{(i,r)} := \hat{y}^{(\mathcal{C}^{(i)})}$  is a prediction for the actual temperature  $y^{(i,r)}$ . Compute the average squared error loss  $\frac{1}{\sum_{i=1}^n m_i} \sum_{i \in \mathcal{V}} \sum_{r=1}^{m_i} (y^{(i,r)} - \hat{y}^{(i,r)})^2$ .

## 6.8 Proofs

### 6.8.1 Proof of Proposition 6.1

We will verify (6.4) via a similar argument as used in the proof (see Section 3.5.1) of Prop. 3.1.

First we decompose the objective function  $f(\mathbf{w})$  in (5.1) as follows:

$$\begin{aligned}
 f(\mathbf{w}) = & \underbrace{\sum_{i \in \mathcal{C}} (1/m_i) \left\| \mathbf{y}^{(i)} - \mathbf{X}^{(i)} \mathbf{w}^{(i)} \right\|_2^2 + \alpha \left[ \sum_{i, i' \in \mathcal{C}} A_{i, i'} \left\| \mathbf{w}^{(i)} - \mathbf{w}^{(i')} \right\|_2^2 + \sum_{\{i, i'\} \in \partial \mathcal{C}} A_{i, i'} \left\| \mathbf{w}^{(i)} - \mathbf{w}^{(i')} \right\|_2^2 \right]}_{=: f'(\mathbf{w})} \\
 & + f''(\mathbf{w}). \tag{6.6}
 \end{aligned}$$

Note that only of the first component  $f'$  depends on the local model parameters  $\mathbf{w}^{(i)}$  of nodes  $i \in \mathcal{C}$ :

Let us introduce the shorthand  $f'(\mathbf{w}^{(i)})$  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 verify the bound (6.4) by showing that if it does not hold, the local model parameters  $\overline{\mathbf{w}}^{(i)} := \overline{\mathbf{w}}^{(\mathcal{C})}$ , for  $i \in \mathcal{C}$ , results in a smaller value  $f'(\overline{\mathbf{w}}^{(i)}) < f'(\widehat{\mathbf{w}}^{(i)})$  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 (5.1).

First, note that

$$\begin{aligned}
f'(\bar{\mathbf{w}}^{(i)}) &= \sum_{i \in \mathcal{C}} (1/m_i) \|\mathbf{y}^{(i)} - \mathbf{X}^{(i)} \bar{\mathbf{w}}^{(\mathcal{C})}\|_2^2 \\
&\quad + \alpha \left[ \sum_{i, i' \in \mathcal{V}} A_{i, i'} \|\bar{\mathbf{w}}^{(\mathcal{C})} - \bar{\mathbf{w}}^{(\mathcal{C})}\|_2^2 + \sum_{\substack{\{i, i'\} \in \mathcal{E} \\ i \in \mathcal{C}, i' \notin \mathcal{C}}} A_{i, i'} \|\bar{\mathbf{w}}^{(\mathcal{C})} - \hat{\mathbf{w}}^{(i')}\|_2^2 \right] \\
&\stackrel{(6.2)}{=} \sum_{i \in \mathcal{C}} (1/m_i) \|\boldsymbol{\varepsilon}^{(i)}\|_2^2 + \alpha \sum_{\substack{\{i, i'\} \in \mathcal{E} \\ i \in \mathcal{C}, i' \notin \mathcal{C}}} A_{i, i'} \|\bar{\mathbf{w}}^{(\mathcal{C})} - \hat{\mathbf{w}}^{(i')}\|_2^2 \\
&\stackrel{(a)}{\leq} \sum_{i \in \mathcal{C}} (1/m_i) \|\boldsymbol{\varepsilon}^{(i)}\|_2^2 + \alpha \sum_{\substack{\{i, i'\} \in \mathcal{E} \\ i \in \mathcal{C}, i' \notin \mathcal{C}}} A_{i, i'} 2 \left( \|\bar{\mathbf{w}}^{(\mathcal{C})}\|_2^2 + \|\hat{\mathbf{w}}^{(i')}\|_2^2 \right) \\
&\leq \sum_{i \in \mathcal{C}} (1/m_i) \|\boldsymbol{\varepsilon}^{(i)}\|_2^2 + \alpha |\partial \mathcal{C}| 2 \left( \|\bar{\mathbf{w}}^{(\mathcal{C})}\|_2^2 + R^2 \right). \tag{6.7}
\end{aligned}$$

Step (a) uses the inequality  $\|\mathbf{u} + \mathbf{v}\|_2^2 \leq 2(\|\mathbf{u}\|_2^2 + \|\mathbf{v}\|_2^2)$  which is valid for any two vectors  $\mathbf{u}, \mathbf{v} \in \mathbb{R}^d$ .

On the other hand,

$$\begin{aligned}
f'(\hat{\mathbf{w}}^{(i)}) &\geq \alpha \sum_{i, i' \in \mathcal{C}} A_{i, i'} \underbrace{\|\hat{\mathbf{w}}^{(i)} - \hat{\mathbf{w}}^{(i')}\|_2^2}_{\stackrel{(6.3)}{=} \|\tilde{\mathbf{w}}^{(i)} - \tilde{\mathbf{w}}^{(i')}\|_2^2} \\
&\stackrel{(3.12)}{\geq} \alpha \lambda_2(\mathbf{L}^{(\mathcal{C})}) \sum_{i \in \mathcal{C}} \|\tilde{\mathbf{w}}^{(i)}\|_2^2. \tag{6.8}
\end{aligned}$$

If the bound (6.4) would not hold, then by (6.8) and (6.7) we would obtain  $f'(\hat{\mathbf{w}}^{(i)}) > f'(\bar{\mathbf{w}}^{(i)})$ , which contradicts the fact that  $\hat{\mathbf{w}}^{(i)}$  solves (5.1).

## 7 Lecture - “Graph Learning”

Lecture 3 discussed GTVMin as a main design principle for FL algorithms. In particular, Lecture 5 discusses FL algorithms that arise from the application of optimization methods, such as the gradient-based methods from Lecture 4, to solve GTVMin. The computational (how much compute required?) and statistical (what is the accuracy of learnt model parameters?) properties of these algorithms crucially depend on the choice for the empirical graph. In some applications, domain expertise can guide the choice for the empirical graph. However, it might also be useful to learn the empirical graph in a more data-driven fashion. This lecture discusses some methods for learning an empirical graph solely from a given collection of local datasets  $\mathcal{D}^{(i)}$  and corresponding local loss function  $L_i(\mathbf{w}^{(i)})$ . We tacitly assume that each node  $i$  uses a model  $\mathcal{H}$  parametrized by local model parameters  $\mathbf{w}^{(i)}$ .

### 7.1 Learning Goals

After this lecture, you should

- know how the computational and statistical properties of GTVMin-based methods depend on the structure of the empirical graph,
- know some measures for (dis-)similarity between local datasets,
- be able to learn a graph from given pairwise similarities and structural constraints (e.g., prescribed node degree).

## 7.2 Empirical Graph is a Design Choice

Consider the GTVMin instance (3.16) for learning the model parameters of a local linear model for each local dataset  $\mathcal{D}^{(i)}$ . To solve (3.16), we use Algorithm 2 as a message passing implementation of the basic gradient step (5.3). Note that GTVMin (3.16) is defined for a given empirical graph  $\mathcal{G}$ . Therefore, the choice for  $\mathcal{G}$  is critical for the statistical and computational properties of Algorithm 1.

**Statistical Properties.** The statistical properties of Algorithm 2 can be assessed via a probabilistic model for the local datasets. One important example for such a probabilistic model is the clustering assumption (6.2) of CFL (see Section 3.3.1). For CFL, we would like to learn similar model parameters for nodes in the same cluster.

According to Prop. 6.1, the GTVMin solutions will be approximately constant over  $\mathcal{C}$ , if  $\lambda_2(\mathbf{L}^{(\mathcal{C})})$  is large and the cluster boundary  $|\partial\mathcal{C}|$  is small. Here,  $\lambda_2(\mathbf{L}^{(\mathcal{C})})$  denotes the smallest non-zero eigenvalue of the Laplacian matrix associated with the induced sub-graph  $\mathcal{G}^{(\mathcal{C})}$ .

Roughly speaking,  $\lambda_2(\mathbf{L}^{(\mathcal{C})})$  will be larger if there are more edges connecting the nodes in  $\mathcal{C}$ . This informal statement can be made precise using a celebrated result from spectral graph theory, known as Cheeger’s inequality [53, Ch. 21]. Alternatively, we can analyse  $\lambda_2(\mathbf{L}^{(\mathcal{C})})$  by interpreting (or approximate) the induced sub-graph  $\mathcal{G}^{(\mathcal{C})}$  as the realization of an Erdős-Rényi (ER) random graph.

The ER model for  $\mathcal{G}^{(\mathcal{C})}$  postulates that two nodes  $i, i' \in \mathcal{C}$  are connected by an edge with probability  $p_e$ . The presence of an edge between a given pair of nodes does not depend on the presence of an edge between any other pair



of nodes (“edges are i.i.d.”).

Since we model the presence of edges as realizations of RVs also the node degrees  $d^{(i)}$  as well as the eigenvalue  $\lambda_2(\mathbf{L}^{(c)})$  become realizations of RVs. The expected node degree is given as  $\mathbb{E}\{d^{(i)}\} = p_e(|\mathcal{C}| - 1)$ . With high probability, a realization of an ER graph has maximum node degree

$$d_{\max} \approx \mathbb{E}\{d^{(i)}\} = p_e(|\mathcal{C}| - 1) \quad (7.1)$$

Clearly, increasing the ER parameter  $p_e$  results in a higher connectivity of  $\mathcal{G}^{(c)}$ .

We can approximate  $\lambda_2(\mathbf{L}^{(c)})$  with the second smallest eigenvalue  $\lambda_2(\bar{\mathbf{L}})$  of the expected Laplacian matrix  $\bar{\mathbf{L}} := \mathbb{E}\{\mathbf{L}^{(c)}\} = |\mathcal{C}|p_e\mathbf{I} - p_e\mathbf{1}\mathbf{1}^T$  [54]. The two smallest eigenvalues of  $\bar{\mathbf{L}}$  are  $\lambda_1(\bar{\mathbf{L}}) = 0$  and  $\lambda_2(\bar{\mathbf{L}}) = |\mathcal{C}|p_e$ . Thus, we have the approximation

$$\lambda_2(\mathbf{L}^{(c)}) \approx \lambda_2(\bar{\mathbf{L}}) = |\mathcal{C}|p_e \stackrel{(7.1)}{\approx} d_{\max}. \quad (7.2)$$

**Computational Properties.** The computational complexity of Algorithm 2 depends on the amount of computation required by a single iteration of its steps (3) - (4). Clearly, this per-iteration complexity increases with increasing node degrees  $d^{(i)}$ . In other words, the smaller  $d^{(i)}$ , the less operations and communication is needed at node  $i \in \mathcal{V}$  during steps (3) - (4) of Algorithm 2.

The computational complexity of Algorithm 2 also depends on the number of iterations required to achieve an approximate solution to GTVMin (3.16). According to (4.6), the convergence speed of the gradient steps (5.9) used in Algorithm 2 depends on the condition number  $\lambda_{nd}(\mathbf{Q})/\lambda_1(\mathbf{Q})$  of the matrix  $\mathbf{Q}$  in (5.2). Algorithm 2 tends to require fewer iterations when the ratio between

the largest and smallest eigenvalues of  $\mathbf{Q}$  is small (closer to 1). The condition number of  $\mathbf{Q}$  tends to be smaller for a smaller ratio between the maximum node degree  $d_{\max}$  and eigenvalue  $\lambda_2(\mathbf{L}^{(\mathcal{G})})$  (see (5.5) and (5.6)).

To summarize, the per-iteration complexity of Algorithm 2 increases with the node degrees  $d^{(i)}$  of the empirical graph  $\mathcal{G}$ . A given computational and communication budget results in an upper bound on the node degrees and, in turn, the number of edges in  $\mathcal{G}$ . The edges should then be placed such that the eigenvalue  $\lambda_2(\mathbf{L}^{(\mathcal{G})})$ , or  $\lambda_2(\mathbf{L}^{(\mathcal{C})})$  for each cluster  $\mathcal{C}$  in CFL, is as large as possible. Some explicit graph constructions that optimize these eigenvalues can be found in the spectral graph theory literature (see [33, 53] and references therein).

### 7.3 Measuring (Dis-)Similarity Between Datasets

The whole idea of GTVMin is to enforce similar model parameters at two nodes  $i, i'$  that are connected by an edge  $\{i, i'\}$  with (relatively) large edge weight  $A_{i, i'}$ . In general, the edges (and their weights) of the empirical graph are a design choice. However, the edges should somehow reflect the actual similarities between local datasets. We next discuss different approaches to measuring the similarity or, equivalently, the discrepancy (the lack of similarity) between two local datasets.

The first approach is based on a probabilistic model, i.e., we interpret the local dataset  $\mathcal{D}^{(i)}$  as realizations of RVs with some parametrized probability distribution  $p^{(i)}(\mathcal{D}^{(i)}; \mathbf{w}^{(i)})$ . We can then measure the discrepancy between  $\mathcal{D}^{(i)}$  and  $\mathcal{D}^{(i')}$  via the Euclidean distance  $\|\mathbf{w}^{(i)} - \mathbf{w}^{(i')}\|_2$  between the parameters  $\mathbf{w}^{(i)}, \mathbf{w}^{(i')}$  of the probability distributions.

In general, we do not know the parameters of the probability distribution  $p^{(i)}(\mathcal{D}^{(i)}; \mathbf{w}^{(i)})$  underlying a local dataset.<sup>9</sup> We might still be able to estimate these parameters, e.g., using variants of maximum likelihood [4, Ch. 3]. Given the estimates  $\widehat{\mathbf{w}}^{(i)}, \widehat{\mathbf{w}}^{(i')}$  for the model parameters, we can then compute the discrepancy measure

$$d^{(i,i')} := \left\| \widehat{\mathbf{w}}^{(i)} - \widehat{\mathbf{w}}^{(i')} \right\|_2.$$

As an illustrative example, consider local datasets consisting of a single number  $y^{(i)}$  which is modelled as a noisy observation  $y^{(i)} = w^{(i)} + n^{(i)}$  with  $n^{(i)} \sim \mathcal{N}(0, 1)$ . The maximum likelihood estimator for  $w^{(i)}$  is then obtained as  $\hat{w}^{(i)} = y^{(i)}$  [29, 55] and, in turn, the resulting discrepancy measure  $d^{(i,i')} := |y^{(i)} - y^{(i')}|$  [56].

The above approach can be interpreted as a special case of a more general approach. Indeed, the learnt parameters  $\widehat{\mathbf{w}}^{(i)}$  (e.g., using maximum likelihood estimation) of a probabilistic model for the local dataset  $\mathcal{D}^{(i)}$  is a form of representation learning. We use the parameter estimate  $\widehat{\mathbf{w}}^{(i)}$  as a vector representation of  $\mathcal{D}^{(i)}$ .

There are also other means to learn a vector representation for a dataset. For example, we could use the gradient  $\nabla f(\mathbf{w}')$  of the average loss  $f(\mathbf{w}) := (1/|\mathcal{D}^{(i)}|) \sum_{(\mathbf{x}, y) \in \mathcal{D}^{(i)}} L((\mathbf{x}, y), h(\mathbf{w}))$ . The properties of this representation will depend on the choice of parametrized model, i.e., how  $h(\mathbf{w})$  depends precisely on the parameters  $\mathbf{w}$ . Another design choice is the location  $\mathbf{w}'$  at which the gradient is evaluated.

---

<sup>9</sup>One exception is when we generate the local dataset by drawing i.i.d. realizations from  $p^{(i)}(\mathcal{D}^{(i)}; \mathbf{w}^{(i)})$ .

## 7.4 Graph Learning Methods

Assume we have constructed a useful measure  $d^{(i,i')} \in \mathbb{R}_+$  for the discrepancy between any two local datasets  $\mathcal{D}^{(i)}, \mathcal{D}^{(i')}$ . We could then construct an empirical graph by connecting each node  $i$  with its nearest neighbours. The nearest neighbours of  $i$  are those other nodes  $i' \in \mathcal{V} \setminus \{i\}$  with smallest discrepancy  $d^{(i,i')}$ . We next discuss an alternative to the nearest-neighbour graph construction. This alternative approach formulates graph learning as a constrained linear optimization problem.

Let us measure the usefulness of a given choice for the edge weights  $A_{i,i'} \in \mathbb{R}_+$  via

$$\sum_{i,i' \in \mathcal{V}} A_{i,i'} d^{(i,i')}. \quad (7.3)$$

The objective function (7.3) penalizes having a large edge weight  $A_{i,i'}$  between two nodes  $i, i'$  with a large discrepancy  $d^{(i,i')}$ . Note that the objective function (7.3) is minimized by the trivial choice  $A_{i,i'} = 0$ , i.e., the empty graph without any edges.

As discussed in Section 7.2, the empirical graph should have a sufficient amount of edges in order to ensure that the GTVMin solutions are useful model parameters. Indeed, the desired pooling effect of GTVMin requires the eigenvalue  $\lambda_2(\mathbf{L}^{(\mathcal{G})})$  to be sufficiently large. Ensuring a large  $\lambda_2(\mathbf{L}^{(\mathcal{G})})$  requires, in turn, that the empirical graph  $\mathcal{G}$  contains a sufficiently large number of edges and corresponding node degrees (see (7.2)).

We can enforce the presence of edges (with positive weight) by adding constraints to (7.3). For example, we might require

$$A_{i,i} = 0, \sum_{i' \neq i} A_{i,i'} = d_{\max} \text{ for all } i \in \mathcal{V}, A_{i,i'} \in [0, 1] \text{ for all } i, i' \in \mathcal{V}. \quad (7.4)$$

The constraints (7.4) require that each node  $i$  is connected with other nodes using total edge weight (weighted node degree)  $\sum_{i' \neq i} A_{i,i'} = d_{\max}$ .

Combining the constraints (7.4) with the objective function (7.3) results in the following graph learning principle,

$$\begin{aligned} \hat{A}_{i,i'} &\in \underset{A_{i,i'}=A_{i',i}}{\operatorname{argmin}} A_{i,i'} d^{(i,i')} & (7.5) \\ A_{i,i'} &\in [0, 1] \text{ for all } i, i' \in \mathcal{V}, \\ A_{i,i} &= 0 \text{ for all } i \in \mathcal{V}, \\ \sum_{i' \neq i} A_{i,i'} &= d_{\max} \text{ for all } i \in \mathcal{V}. \end{aligned}$$

Note that (7.5) is an instance of the constrained quadratic minimization problem (4.15). We can therefore use (variants of) projected GD from Section 4.6 to iteratively compute (approximate) solutions to (7.5).

The first constraint in (7.5) requires each edge weight to belong to the interval  $[0, 1]$ . The second constraint prohibits any self-loops in the resulting empirical graph. Indeed, adding self-loops to an empirical graph has no effect on the resulting GTVMin-based method (see (3.14)). The last constraint of the learning principle (7.5) enforces a regular empirical graph: Each node  $i$  having the same weighted node degree  $d^{(i)} = \sum_{i' \neq i} A_{i,i'} = d_{\max}$ .

For some FL applications it might be detrimental to insist in identical node degrees of the empirical graph. Instead, we might prefer other structural properties such as a small total number of edges or presence of few hub nodes (with exceptionally large node degree [12, 56]). We can enforce an upper bound

on the total number of edges by modifying the last constraint in (7.5),

$$\begin{aligned}
\hat{A}_{i,i'} &\in \underset{A_{i,i'}=A_{i',i}}{\operatorname{argmin}} A_{i,i'} d^{(i,i')} & (7.6) \\
A_{i,i'} &\in [0, 1] \text{ for all } i, i' \in \mathcal{V}, \\
A_{i,i} &= 0 \text{ for all } i \in \mathcal{V}, \\
\sum_{i', i \in \mathcal{V}} A_{i,i'} &\leq s.
\end{aligned}$$

The problem has a closed-form solution as explained in [56].

## 7.5 Overview of Coding Assignment

**Python Notebook.** `GraphLearning_CodingAssignment.ipynb`

**Data File.** `Assignment_MLBasicsData.csv`

This coding assignment builds on the previous coding assignments “FL Algorithms” (see Section 5.6) and “FL Main Flavors” (see Section 6.7). In particular, we again construct an empirical graph  $\mathcal{G}$  with nodes  $i \in \mathcal{V}$  representing FMI stations. The node  $i \in \mathcal{V}$  holds a local dataset  $\mathcal{D}^{(i)}$  that consists of  $m_i$  data points. Each of these data points is a temperature measurement, taken at station  $i$ , and characterized by  $d = 7$  features  $\mathbf{x} = (x_1, \dots, x_7)^T$  and a label  $y$  which is the temperature measurement itself. The features are (normalized) values of the latitude and longitude of the FMI station as well as the (normalized) year, month, day, hour, minute when the temperature has been measured. We then apply some of the graph learning methods from this lecture to construct the edge set of  $\mathcal{G}$ .

Similar to the coding assignment “FL Main Flavors”, we again connect

each node  $i$  to its nearest neighbours. However, in contrast to previous assignments, we use a different criterion for which are the nearest neighbours of  $i$ . In particular, we define the nearest neighbours  $i' \in \mathcal{V} \setminus \{i\}$  as those with smallest discrepancy  $d^{(i,i')}$ . You have to try out the following choices for this discrepancy measure:

- **I:**  $d^{(i,i')} := |\hat{y}^{(i)} - \hat{y}^{(i')}|$  with  $\hat{y}^{(i)} = (1/m_i) \sum_{r=1}^{m_i} y^{(i,r)}$  being the average temperature at FMI station  $i$ .
- **II:**  $d^{(i,i')} := \|\mathbf{z}^{(i)} - \mathbf{z}^{(i')}\|_2$  with the vector  $\mathbf{z}^{(i)}$  consisting of the parameters of a GMM that is fit to the feature vectors  $\mathbf{x}^{(i,1)}, \dots, \mathbf{x}^{(i,m_i)}$  in the local dataset  $\mathcal{D}^{(i)}$ .
- **III:**  $d^{(i,i')} := \|\mathbf{z}^{(i)} - \mathbf{z}^{(i')}\|_2$  with the vector  $\mathbf{z}^{(i)} := \nabla L_i(\hat{\mathbf{w}})$  being the gradient of the average squared error loss  $L_i(\hat{\mathbf{w}}) = \frac{1}{m_i} \sum_{r=1}^{m_i} \left( y^{(i,r)} - (\mathbf{x}^{(i,r)})^T \hat{\mathbf{w}} \right)^2$ . The gradient  $\nabla L_i(\hat{\mathbf{w}})$  is evaluated for the model parameters  $\hat{\mathbf{w}}$  of a linear model that is obtained by minimizing the average squared error loss, over all local datasets,

$$\hat{\mathbf{w}} = \operatorname{argmin}_{\mathbf{w} \in \mathbb{R}^7} \frac{1}{\sum_{i=1}^n m_i} \sum_{i=1}^n \sum_{r=1}^{m_i} \left( y^{(i,r)} - (\mathbf{x}^{(i,r)})^T \mathbf{w} \right)^2.$$

Note that for each of the above discrepancy measures, we obtain (via connecting nearest neighbours) a potentially different empirical graph  $\mathcal{G}^{(\text{I})}, \mathcal{G}^{(\text{II})}, \mathcal{G}^{(\text{III})}$ . For each of these empirical graphs, learn local model parameters for a linear model by applying FedGD Algorithm 2. To this end, you have to split each local dataset into a training set and a validation set (Algorithm 2 is applied to the training sets). Diagnose the resulting model parameters by computing the average (over all nodes) training error and validation error.

## 8 Lecture - “Trustworthy FL”

This lecture discusses some key requirements for trustworthy AI that have been put forward by the European Union. We will also see how these requirements might guide the design choices for GTVMin. Our focus will be on the four design criteria: robustness, privacy protection and explainability. This lecture discusses the robustness and explainability of basic linear regression that we encountered in Lecture 2. We will see that regularization techniques allow to navigate robustness-explainability-accuracy trade-offs.

### 8.1 Learning Goals

After this lecture, you should

- know some key requirements for trustworthy AI
- be familiar with quantitative measures of robustness and explainability
- have some intuition about how robustness, privacy and transparency guides design choices for local models, loss functions and empirical graph in GTVMin

We can use the upper bound (2.18) to study the effect of perturbing features and labels of data points.



## 9 Lecture - “Privacy-Protection in FL”

FL is inherently based on sharing information. Without any information sharing between the owners (or generators) of local datasets, FL is not possible.

### 9.1 Learning Goals

Afer this lecture, you should

- be aware of threats to privacy and the need to protect it,
- know some quantitative measures for privacy leakage,
- understand the effect of GTVMin design choices on user privacy,
- be able to implement FL algorithms with privacy guarantees.

### 9.2 Assignment

## 10 Lecture - “Data and Model Poisoning in FL”

This lecture discusses the robustness of FL systems against data poisoning which are a specific type of cyber attacks.

### 10.1 Learning Goals

After this lecture, you should

1. be aware of threats posed by data and model poisoning
2. understand how GTVMin design choices result in more or less safe FL algorithms

### 10.2 Data Poisoning

### 10.3 Model Poisoning

There is a trade-off between privacy protection and robustness against model poisoning attacks. From [57]: *Without secure aggregation, privacy is lost, but the aggregator may attempt to filter out “anomalous” contributions. Since the weights of a model created using*

### 10.4 Assignment

## Glossary

***k*-means** The *k*-means algorithm is a hard clustering method which assigns each data points to precisely one out of *k* different clusters. The method iteratively updates this assignment in order to minimize the average distance between data points in their nearest cluster mean (centre). 3, 12

**activation function** Each artificial neuron within an ANN consists of an activation function that maps the inputs of the neuron to a single output value. In general, an activation function is a non-linear map of the weighted sum of neuron inputs (this weighted sum is the activation of the neuron). 14

**artificial intelligence** Artificial intelligence aims to develop systems that behave rational in the sense of maximizing a long-term reward. 9

**artificial neural network** An artificial neural network is a graphical (signal-flow) representation of a map from features of a data point at its input to a predicted label at its output. 1, 6–8, 10–12, 14, 19, 26

**baseline** A reference value or benchmark for the average loss incurred by a hypothesis when applied to the data points generated in a specific ML application. Such a reference value might be obtained from human performance (e.g., error rate of dermatologists diagnosing cancer from visual inspection of skin areas) or other ML methods (“competitors”) 10, 11

**batch** A set of randomly selected data points out of a (typically very large) dataset. 6, 8, 15

**Bayes estimator** A hypothesis  $h$  whose Bayes risk is minimal [29]. 2, 10

**Bayes risk** We use the term Bayes risk as a synonym for the risk or expected loss of a hypothesis. Some authors reserve the term Bayes risk for the risk of a hypothesis that achieves minimum risk, such a hypothesis being referred to as a Bayes estimator [29]. 2

**bias** Consider some unknown quantity  $\bar{w}$ , e.g., the true weight in a linear model  $y = \bar{w}x + e$  relating feature and label of a data point. We might use an ML method (e.g., based on ERM) to compute an estimate  $\hat{w}$  for the  $\bar{w}$  based on a set of data points that are realizations of RVs. The (squared) bias incurred by the estimate  $\hat{w}$  is typically defined as  $B^2 := (\mathbb{E}\{\hat{w}\} - \bar{w})^2$ . We extend this definition to vector-valued quantities using the squared Euclidean norm  $B^2 := \|\mathbb{E}\{\hat{\mathbf{w}}\} - \bar{\mathbf{w}}\|_2^2$ . 13

**classification** Classification is the task of determining a discrete-valued label  $y$  of a data point based solely on its features  $\mathbf{x}$ . The label  $y$  belongs to a finite set, such as  $y \in \{-1, 1\}$ , or  $y \in \{1, \dots, 19\}$  and represents a category to which the corresponding data point belongs to. 12

**cluster** A cluster is a subset of data points that are more similar to each other than to the data points outside the cluster. The notion and measure of similarity between data points is a design choice. If data points are characterized by numeric feature vectors in some Euclidean space we

can define the similarity between two data points via the Euclidean distance between their feature vectors. 3–5, 10, 12

**clustered federated learning (CFL)** Clustered FL assumes that local datasets clusters. The local datasets belonging to the same cluster have similar statistical properties. 1–4

**clustering** Clustering methods decompose a given set of data points into few subsets, which are referred to as clusters. Each cluster consists of data points that are more similar to each other than to data points outside the cluster. Different clustering methods use different measures for the similarity between data points and different representation of clusters. The clustering method  $k$ -means uses the average feature vector (“cluster means”) of a cluster as its representative. A popular soft-clustering method based on GMM represents a cluster by a multivariate normal distribution. 1, 3, 12

**clustering assumption** The clustering assumption postulates that data points in a dataset form a (small) number of groups or clusters. Data points in the same cluster are more similar with each other than with those outside the cluster [49]. We obtain different clustering methods by using different notions of similarity between data points. 2

**computational aspects** By computational aspects of a ML method, we mainly refer to the computational resources required for its implementation. For example, if a ML method uses iterative optimization techniques to solve ERM, then its computational aspects include (i) how many arithmetic operations are needed to implement a single

iteration (gradient step) and (ii) how many iterations are needed to obtain useful model parameters. One important example for an iterative optimization technique is GD. 1, 4, 7, 10, 15, 16

**condition number** The condition number  $\kappa(\mathbf{Q}) \geq 1$  of a psd matrix  $\mathbf{Q}$  is the ratio  $\lambda_{\max}/\lambda_{\min}$  between the largest  $\lambda_{\max}$  and the smallest  $\lambda_{\min}$  eigenvalue of  $\mathbf{Q}$ . One example for such a matrix  $\mathbf{Q}$  is obtained from the feature matrix  $\mathbf{X}$  of a dataset,  $\mathbf{Q} = \mathbf{X}\mathbf{X}^T$ . From a computational perspective, we prefer  $\mathbf{Q}$  to have a condition number close to 1. 3

**connected graph** An undirected graph  $\mathcal{G} = (\mathcal{V}, \mathcal{E})$  is connected if we cannot find a (non-empty) subset  $\mathcal{V}' \subset \mathcal{V}$  with no edges leaving  $\mathcal{V}'$ . 5

**convex** A set  $\mathcal{C} \subseteq \mathbb{R}^d$  is convex if it contains the line segment between any two points of that set. We define a function as convex if its epigraph is a convex set [28]. 2–4, 7, 9, 15, 22, 28

**covariance matrix** The covariance matrix of a RV  $\mathbf{x} \in \mathbb{R}^d$  is defined as  $\mathbb{E}\left\{(\mathbf{x} - \mathbb{E}\{\mathbf{x}\})(\mathbf{x} - \mathbb{E}\{\mathbf{x}\})^T\right\}$ . 10, 11, 14, 18

**data** A (indexed) set of data points. 1, 2, 4, 7, 8, 21

**data augmentation** Data augmentation methods add synthetic data points to an existing set of data points. These synthetic data points might be obtained by perturbations (adding noise) or transformations (rotations of images) of the original data points. 12, 14, 15

**data point** A data point is any object that conveys information [58]. Data points might be students, radio signals, trees, forests, images, RVs, real

numbers or proteins. We characterize data points using two types of properties. One type of property is referred to as a feature. Features are properties of a data point that can be measured or computed in an automated fashion. Another type of property is referred to as labels. The label of a data point represents some higher-level fact (or quantity of interest). In contrast to features, determining the label of a data point typically requires human experts (domain experts). Roughly speaking, ML aims at predicting the label of a data point based solely on its features. 1–17, 20, 23–25, 27, 28

**data poisoning** FL methods allow to leverage the information contained in local datasets generated by other parties to improve the training of a tailored model. Depending on how much we trust the other parties, FL can be compromised by data poisoning. Data poisoning refers to the intentional manipulation (or fabrication) of local datasets to steer the training of a specific local model [59, 60]. 3

**dataset** With a slight abuse of notation we use the terms “dataset” or “set of data points” to refer to an indexed list of data points  $\mathbf{z}^{(1)}, \mathbf{z}^{(2)}, \dots$ . Thus, there is a first data point  $\mathbf{z}^{(1)}$ , a second data point  $\mathbf{z}^{(2)}$  and so on. Strictly speaking a dataset is a list and not a set [61]. By using indexed lists of data points we avoid some of the challenges arising in concept of an abstract set. 2, 4–6, 8, 9, 11, 13–16, 20

**decision region** Consider a hypothesis map  $h$  that delivers values from a finite set  $\mathcal{Y}$ . We refer to the set of features  $\mathbf{x} \in \mathcal{X}$  that result in the same output  $h(\mathbf{x}) = a$  as a decision region of the hypothesis  $h$ . 14

**decision tree** A decision tree is a flow-chart like representation of a hypothesis map  $h$ . More formally, a decision tree is a directed graph which reads in the feature vector  $\mathbf{x}$  of a data point at its root node. The root node then forwards the data point to one of its children nodes based on some elementary test on the features  $\mathbf{x}$ . If the receiving children node is not a leaf node, i.e., it has itself children nodes, it represents another test. Based on the test result, the data point is further pushed to one of its neighbours. This testing and forwarding of the data point is repeated until the data point ends up in a leaf node (having no children nodes). The leaf nodes represent sets (decision regions) constituted by feature vectors  $\mathbf{x}$  that are mapped to the same function value  $h(\mathbf{x})$ . 5, 11, 12, 14

**deep net** We refer to an ANN with a (relatively) large number of hidden layers as a deep ANN or “deep net”. Deep nets are used to represent the hypothesis spaces of deep learning methods [62]. 12

**differentiable** A function  $f : \mathbb{R}^d \rightarrow \mathbb{R}$  is differentiable if it has a gradient  $\nabla f(\mathbf{x})$  everywhere (for every  $\mathbf{x} \in \mathbb{R}^d$ ) [5]. 3, 10, 11, 14, 25

**differential privacy** Consider an algorithm used to compute some output based on a dataset. Differential privacy is a precise measure of this algorithm’s privacy leakage. Roughly speaking, the algorithm is differentially private if the probability distribution of its output does not allow it to infer if the dataset contains a specific data point. 4

**discrepancy** Consider a FL application with networked data represented by an empirical graph. FL methods use a discrepancy measure to compare



hypothesis maps from local models at nodes  $i, i'$  connected by an edge in the empirical graph. 3–6, 9, 10

**edge weight** Each edge  $\{i, i'\}$  of an empirical graph is assigned a non-negative edge weight  $A_{i, i'} \geq 0$ . A zero weight  $A_{i, i'} = 0$  indicates the absence of an edge between nodes  $i, i' \in \mathcal{V}$ . 1, 3, 4, 6–8, 11, 12, 14, 15

**effective dimension** The effective dimension  $d_{\text{eff}}(\mathcal{H})$  of an infinite hypothesis space  $\mathcal{H}$  is a measure of its size. Loosely speaking, the effective dimension is equal to the number of “independent” tunable parameters of the model. These parameters might be the coefficients used in a linear map or the weights and bias terms of an ANN. 11, 12

**eigenvalue** We refer to a number  $\lambda \in \mathbb{R}$  as eigenvalue of a square matrix  $\mathbf{A} \in \mathbb{R}^{d \times d}$  if there is a non-zero vector  $\mathbf{x} \in \mathbb{R}^d \setminus \{\mathbf{0}\}$  such that  $\mathbf{A}\mathbf{x} = \lambda\mathbf{x}$ . 2–9, 12–14, 16, 17

**eigenvalue decomposition** The eigenvalue decomposition for a square matrix  $\mathbf{A} \in \mathbb{R}^{d \times d}$  is a factorization of the form

$$\mathbf{A} = \mathbf{V}\mathbf{\Lambda}\mathbf{V}^{-1}.$$

The columns of the matrix  $\mathbf{V} = (\mathbf{v}^{(1)}, \dots, \mathbf{v}^{(d)})$  are the eigenvectors of the matrix  $\mathbf{V}$ . The diagonal matrix  $\mathbf{\Lambda} = \text{diag}\{\lambda_1, \dots, \lambda_d\}$  contains the eigenvalues  $\lambda_j$  corresponding to the eigenvectors  $\mathbf{v}^{(j)}$ . Note that the above decomposition exists only if the matrix  $\mathbf{A}$  is diagonalizable. 4, 5, 8

**eigenvector** An eigenvector of a matrix  $\mathbf{A}$  is a non-zero vector  $\mathbf{x} \in \mathbb{R}^d \setminus \{\mathbf{0}\}$  such that  $\mathbf{A}\mathbf{x} = \lambda\mathbf{x}$  with some eigenvalue  $\lambda$ . 4, 5, 7, 12

**empirical graph** Empirical graphs represent collections of local datasets and corresponding local models [49]. An empirical graph is an undirected weighted empirical graph whose nodes carry local datasets and models. FL methods learn a local hypothesis  $h^{(i)}$ , for each node  $i \in \mathcal{V}$ , such that it incurs small loss on the local datasets. 1–16, 19, 27

**empirical risk** The empirical risk of a given hypothesis on a given set of data points is the average loss of the hypothesis computed over all data points in that set. 2, 8, 13, 19, 26

**empirical risk minimization** Empirical risk minimization is the optimization problem of finding the hypothesis with minimum average loss (or empirical risk) on a given set of data points (the training set). Many ML methods are special cases of empirical risk. 1–4, 6–8, 10–14, 19, 23, 27, 28

**estimation error** Consider data points with feature vectors  $\mathbf{x}$  and label  $y$ . In some applications we can model the relation between features and label of a data point as  $y = \bar{h}(\mathbf{x}) + \varepsilon$ . Here we used some true hypothesis  $\bar{h}$  and a noise term  $\varepsilon$  which might represent modelling or labelling errors. The estimation error incurred by a ML method that learns a hypothesis  $\hat{h}$ , e.g., using ERM, is defined as  $\hat{h} - \bar{h}$ . For a parametrized hypothesis space, consisting of hypothesis maps that are determined by a parameter vector  $\mathbf{w}$ , we define the estimation error in terms of parameter vectors as  $\Delta\mathbf{w} = \hat{\mathbf{w}} - \bar{\mathbf{w}}$ . first 7, 9

**Euclidean space** The Euclidean space  $\mathbb{R}^d$  of dimension  $d$  refers to the space of all vectors  $\mathbf{x} = (x_1, \dots, x_d)$ , with real-valued entries  $x_1, \dots, x_d \in \mathbb{R}$ ,

whose geometry is defined by the inner product  $\mathbf{x}^T \mathbf{x}' = \sum_{j=1}^d x_j x'_j$  between any two vectors  $\mathbf{x}, \mathbf{x}' \in \mathbb{R}^d$  [5]. 2, 4, 10, 23

**feature** A feature of a data point is one of its properties that can be measured or computed in an automated fashion. For example, if a data point is a bitmap image, then we could use the red-green-blue intensities of its pixels as features. Some widely used synonyms for the term feature are “covariate”, “explanatory variable”, “independent variable”, “input (variable)”, “predictor (variable)” or “regressor” [63–65]. However, this book makes consequent use of the term features for low-level properties of data points that can be measured easily. 1–3, 5–17, 20, 23–25, 28

**feature map** A map that transforms the original features of a data point into new features. The so-obtained new features might be preferable over the original features for several reasons. For example, the shape of datasets might become simpler in the new feature space, allowing to use linear models in the new features. Another reason could be that the number of new features is much smaller which is preferable in terms of avoiding overfitting. The special case of feature maps that deliver two numeric features are particularly useful for data visualization. Indeed, we can then depict data points in a scatterplot by using these two features as the coordinates of a data point. 14

**feature matrix** Consider a dataset  $\mathcal{D}$  of  $m$  data points that are characterized by feature vectors  $\mathbf{x}^{(1)}, \dots, \mathbf{x}^{(m)}$ . It is convenient to collect these feature vectors into a feature matrix  $\mathbf{X} := (\mathbf{x}^{(1)}, \dots, \mathbf{x}^{(m)})$ . 2, 4, 5, 8, 11, 12, 14

**feature space** The feature space of a given ML application or method is constituted by all potential values that the feature vector of a data point can take on. Within this book the most frequently used choice for the feature space is the Euclidean space  $\mathbb{R}^d$  with dimension  $d$  being the number of individual features of a data point. 11, 13, 14

**federated averaging (FedAvg)** Federated averaging is an iterative FL algorithm that alternates between local model trainings and averaging the resulting local model parameters. Different variants of this algorithm are obtained by different techniques for the model training. The authors of [13] consider federated averaging methods where the local model training is implemented by running several GD steps 1

**federated learning (FL)** Federated learning is an umbrella term for ML methods that train models in a collaborative fashion using decentralized data and computation. 1–12, 14, 20, 28

**Finnish Meteorological Institute** The Finnish Meteorological Institute is a government agency responsible for gathering and reporting weather data in Finland. 2, 8, 9, 11, 12, 14–16

**Gaussian mixture model** Gaussian mixture models (GMM) are a family of probabilistic models for data points characterized by a numeric feature vector  $\mathbf{x}$ . A GMM interprets  $\mathbf{x}$  as being drawn from one out of  $k$  different multivariate normal distributions  $p^{(c)} = \mathcal{N}(\boldsymbol{\mu}^{(c)}, \mathbf{C}^{(c)})$ , indexed by  $c = 1, \dots, k$ . The probability that  $\mathbf{x}$  is drawn from the  $c$ -th multivariate normal distribution is denoted  $p_c$ . Thus, a GMM is

parametrized by the probability  $p_c$ , the mean vector  $\boldsymbol{\mu}^{(c)}$  and covariance matrix  $\boldsymbol{\Sigma}^{(c)}$  for each  $c = 1, \dots, k$ . 3, 9, 12

**General Data Protection Regulation** The General Data Protection Regulation (GDPR) is a law that has been passed by the European Union (EU) and put into effect on May 25, 2018 <https://gdpr.eu/tag/gdpr/>. The GDPR imposes obligations onto organizations anywhere, so long as they target, collect or in any other way process data related to people (i.e., personal data) in the EU. 6

**generalized total variation** Generalized total variation measures the changes of vector-valued node attributes of a graph. 6–9, 12

**gradient** For a real-valued function  $f : \mathbb{R}^d \rightarrow \mathbb{R} : \mathbf{w} \mapsto f(\mathbf{w})$ , a vector  $\mathbf{g}$  such that  $\lim_{\mathbf{w} \rightarrow \mathbf{w}'} \frac{f(\mathbf{w}) - (f(\mathbf{w}') + \mathbf{g}^T(\mathbf{w} - \mathbf{w}'))}{\|\mathbf{w} - \mathbf{w}'\|} = 0$  is referred to as the gradient of  $f$  at  $\mathbf{w}'$ . If such a vector exists it is denoted  $\nabla f(\mathbf{w}')$  or  $\nabla f(\mathbf{w})|_{\mathbf{w}'}$  [5]. 1–6, 9–12, 14, 25, 26, 28

**gradient descent (GD)** Gradient descent is an iterative method for finding the minimum of a differentiable function  $f(\mathbf{w})$ . 1, 4–7, 10–12, 15, 16, 21, 22, 26

**gradient step** Given a differentiable real-valued function  $f(\mathbf{w})$  and a vector  $\mathbf{w}'$ , the gradient step updates  $\mathbf{w}'$  by adding the scaled negative gradient  $\nabla f(\mathbf{w}')$ ,  $\mathbf{w}' \mapsto \mathbf{w}' - \eta \nabla f(\mathbf{w}')$ . 1–13, 15, 16, 22

**gradient-based method** Gradient-based methods are iterative algorithms for finding the minimum (or maximum) of a differentiable objective function of the model parameters. These algorithms construct a sequence

of approximations to an optimal choice for model parameters that results in a minimum objective function value. As their name indicates, gradient-based methods use the gradients of the objective function evaluated during previous iterations to construct new (hopefully) improved model parameters. 1–5, 7, 8, 10–13, 15, 26

**graph** A graph  $\mathcal{G} = (\mathcal{V}, \mathcal{E})$  is a pair that consists of a node set  $\mathcal{V}$  and an edge set  $\mathcal{E}$ . In general, a graph is specified by a map that assigns to each edge  $e \in \mathcal{E}$  a pair of nodes [66]. One important family of graphs (simple undirected graphs) is obtained by identifying each edge  $e \in \mathcal{E}$  with two different nodes  $\{i, i'\}$ . Weighted graphs also specify numeric weights  $A_e$  for each edge  $e \in \mathcal{E}$ . 6, 8, 19

**GTV minimization** GTV minimization is an instance of RERM using the GTV of local model parameters as a regularizer. 1–15, 17

**horizontal FL** Horizontal FL refers to applications with local datasets that are constituted by different data points but using the same features to characterize them [?]. One example for horizontal FL is numerical weather prediction using a network of weather (observation) stations. Each weather station measures the same quantities such as daily temperature, air pressure and precipitation. However, different weather stations measure the characteristics or features of different spatio-temporal regions (each such region being a separate data point). 6, 7

**hypothesis** A map (or function)  $h : \mathcal{X} \rightarrow \mathcal{Y}$  from the feature space  $\mathcal{X}$  to the label space  $\mathcal{Y}$ . Given a data point with features  $\mathbf{x}$  we use a hypothesis

map  $h$  to estimate (or approximate) the label  $y$  using the predicted label  $\hat{y} = h(\mathbf{x})$ . ML is about learning (or finding) a hypothesis map  $h$  such that  $y \approx h(\mathbf{x})$  for any data point. 1–20, 23–28

**hypothesis space** Every practical ML method uses a specific hypothesis space (or model)  $\mathcal{H}$ . The hypothesis space of a ML method is a subset of all possible maps from the feature space to label space. The design choice of the hypothesis space should take into account available computational resources and statistical aspects. If the computational infrastructure allows for efficient matrix operations, and there is a (approximately) linear relation between features and label, a useful choice for the hypothesis space might be the linear model. 1, 6, 7, 11–16, 18, 20, 24, 27, 28

**i.i.d.** It can be useful to interpret data points  $\mathbf{z}^{(1)}, \dots, \mathbf{z}^{(m)}$  as realizations of independent and identically distributed RVs with a common probability distribution. If these RVs are continuous, their joint probability density function (pdf) is  $p(\mathbf{z}^{(1)}, \dots, \mathbf{z}^{(m)}) = \prod_{r=1}^m p(\mathbf{z}^{(r)})$  with  $p(\mathbf{z})$  being the common marginal pdf of the underlying RVs. 3, 5, 6, 12–14, 17, 24

**i.i.d. assumption** The i.i.d. assumption interprets data points of a dataset as the realizations of i.i.d. RVs. 6, 13, 24

**interpretability** A ML method is interpretable for a specific user if she can well anticipate the predictions delivered by the method. The notion of interpretability can be made precise using quantitative measures of the uncertainty about the predictions [67]. 16

**label** A higher level fact or quantity of interest associated with a data point.

If a data point is an image, its label might be the fact that it shows a cat (or not). Some widely used synonyms for the term label are "response variable", "output variable" or "target" [63–65]. 1–17, 19, 20, 23, 25, 27

**label space** Consider a ML application that involves data points characterized by features and labels. The label space is constituted by all potential values that the label of a data point can take on. Regression methods, aiming at predicting numeric labels, often use the label space  $\mathcal{Y} = \mathbb{R}$ . Binary classification methods use a label space that consists of two different elements, e.g.,  $\mathcal{Y} = \{-1, 1\}$ ,  $\mathcal{Y} = \{0, 1\}$  or  $\mathcal{Y} = \{\text{"cat image"}, \text{"no cat image"}\}$  13

**Laplacian matrix** The geometry or structure of a graph  $\mathcal{G}$  can be analyzed using the properties of special matrices that are associated with  $\mathcal{G}$ . One such matrix is the graph Laplacian matrix  $\mathbf{L}$  which is defined for an undirected and weighted graph (e.g., the empirical graph of networked data) [68, 69]. 1–5, 12, 15, 16

**law of large numbers** The law of large numbers refers to the convergence of the average of an increasing (large) number of i.i.d. RVs to the mean (or expectation) of their common probability distribution. Different instances of the law of large numbers are obtained using different notions of convergence. 12, 14

**learning rate** Consider an iterative method for finding or learning a good choice for a hypothesis. Such an iterative method repeats similar computational (update) steps that adjust or modify the current choice



for the hypothesis to obtain an improved hypothesis. A prime example for such an iterative learning method is GD and its variants. We refer by learning rate to any parameter of an iterative learning method that controls the extent by which the current hypothesis might be modified or improved in each iteration. A prime example for such a parameter is the step size used in GD. Some authors use the term learning rate mostly as a synonym for the step size of (a variant of) GD 1–9, 11, 12, 15, 16, 26

**learning task** A learning task consists of a specific choice for a collection of data points (e.g., all images stored in a particular database), their features and labels. 3, 13

**least absolute shrinkage and selection operator (Lasso)** The least absolute shrinkage and selection operator (Lasso) is an instance of structural risk minimization (SRM) for learning the weights  $\mathbf{w}$  of a linear map  $h(\mathbf{x}) = \mathbf{w}^T \mathbf{x}$ . The Lasso minimizes the sum consisting of an average squared error loss (as in linear regression) and the scaled  $\ell_1$  norm of the weight vector  $\mathbf{w}$ . 24

**linear model** We use the term linear model in a very specific sense. In particular, a linear model is a hypothesis space which consists of all linear maps,

$$\mathcal{H}^{(d)} := \{h(\mathbf{x}) = \mathbf{w}^T \mathbf{x} : \mathbf{w} \in \mathbb{R}^d\}. \quad (10.1)$$

Note that (10.1) defines an entire family of hypothesis spaces, which is parametrized by the number  $d$  of features that are linearly combined to form the prediction  $h(\mathbf{x})$ . The design choice of  $d$  is guided by

computational aspects (smaller  $d$  means less computation), statistical aspects (increasing  $d$  might reduce prediction error) and interpretability (a linear model using few carefully chosen features might be considered interpretable). 1–3, 5, 6, 8–10, 12–17

**linear regression** Linear regression aims at learning a linear hypothesis map to predict a numeric label based on numeric features of a data point. The quality of a linear hypothesis map is measured using the average squared error loss incurred on a set of labeled data points (which we refer to as training set). 1–4, 7, 9, 12, 14–16

**local dataset** The concept of a local dataset is in-between the concept of a data point and a dataset. A local dataset consists of several individual data points which are characterized by features and labels. In contrast to a single dataset used in basic ML methods, a local dataset is also related to other local datasets via different notions of similarities. These similarities might arise from probabilistic models or communication infrastructure and are encoded in the edges of an empirical graph. 1–9, 11–16, 20, 28

**local model** Consider a collections of local datasets that are assigned to the nodes of an empirical graph. A local model  $\mathcal{H}^{(i)}$  is a hypothesis space that is assigned to a node  $i \in \mathcal{V}$ . Different nodes might be assigned different hypothesis spaces, i.e., in general  $\mathcal{H}^{(i)} \neq \mathcal{H}^{(i')}$  for different nodes  $i, i' \in \mathcal{V}$ . 1–11, 13, 15–17, 27

**loss** With a slight abuse of language, we use the term loss either for the loss function itself or for its value for a specific pair of a data point and a

hypothesis. 1, 2, 4–10, 12–17, 19, 23–28

**loss function** A loss function is a map

$$L : \mathcal{X} \times \mathcal{Y} \times \mathcal{H} \rightarrow \mathbb{R}_+ : ((\mathbf{x}, y), h) \mapsto L((\mathbf{x}, y), h)$$

which assigns a pair consisting of a data point, with features  $\mathbf{x}$  and label  $y$ , and a hypothesis  $h \in \mathcal{H}$  the non-negative real number  $L((\mathbf{x}, y), h)$ . The loss value  $L((\mathbf{x}, y), h)$  quantifies the discrepancy between the true label  $y$  and the predicted label  $h(\mathbf{x})$ . Smaller (closer to zero) values  $L((\mathbf{x}, y), h)$  mean a smaller discrepancy between predicted label and true label of a data point. Figure 10.1 depicts a loss function for a given data point, with features  $\mathbf{x}$  and label  $y$ , as a function of the hypothesis  $h \in \mathcal{H}$ . 1–3, 5, 7, 10, 11, 16, 17, 24, 25

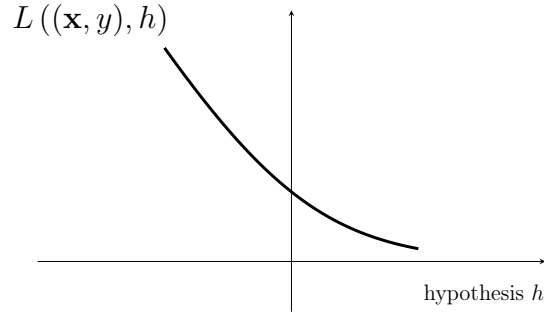


Figure 10.1: Some loss function  $L((\mathbf{x}, y), h)$  for a fixed data point, with feature vector  $\mathbf{x}$  and label  $y$ , and varying hypothesis  $h$ . ML methods try to find (learn) a hypothesis that incurs minimum loss.

**maximum likelihood** Consider data points  $\mathcal{D} = \{\mathbf{z}^{(1)}, \dots, \mathbf{z}^{(m)}\}$  that are interpreted as realizations of i.i.d. RVs with a common probability

distribution  $p(\mathbf{z}; \mathbf{w})$  which depends on a parameter vector  $\mathbf{w} \in \mathcal{W} \subseteq \mathbb{R}^n$ . Maximum likelihood methods aim at finding a parameter vector  $\mathbf{w}$  such that the probability (density)  $p(\mathcal{D}; \mathbf{w}) = \prod_{r=1}^m p(\mathbf{z}^{(r)}; \mathbf{w})$  of observing the data is maximized. Thus, the maximum likelihood estimator is obtained as a solution to the optimization problem  $\max_{\mathbf{w} \in \mathcal{W}} p(\mathcal{D}; \mathbf{w})$ . 5

**model** We use the term model as a synonym for hypothesis space 1, 3–5, 7–13, 15, 17, 19

**model parameters** Model parameters are numbers that select a hypothesis map out of a hypothesis space. 1–15, 17, 22

**multivariate normal distribution** The multivariate normal distribution  $\mathcal{N}(\mathbf{m}, \mathbf{C})$  is an important family of probability distributions for a continuous RV  $\mathbf{x} \in \mathbb{R}^d$  [3, 70, 71]. This family is parametrized by the mean  $\mathbf{m}$  and covariance matrix  $\mathbf{C}$  of  $\mathbf{x}$ . If the covariance matrix is invertible, the probability distribution of  $\mathbf{x}$  is

$$p(\mathbf{x}) \propto \exp \left( - (1/2) (\mathbf{x} - \mathbf{m})^T \mathbf{C}^{-1} (\mathbf{x} - \mathbf{m}) \right).$$

3, 10

**mutual information** The mutual information  $I(\mathbf{x}; y)$  between two RVs  $\mathbf{x}$ ,  $y$  defined on the same probability space is given by

$$I(\mathbf{x}; y) := \mathbb{E} \left\{ \log \frac{p(\mathbf{x}, y)}{p(\mathbf{x})p(y)} \right\}.$$

It is a measure for how well we can estimate  $y$  based solely from  $\mathbf{x}$ . A large value of  $I(\mathbf{x}; y)$  means that  $y$  can be well predicted solely from  $\mathbf{x}$ . The prediction could be obtained by a hypothesis learnt by a ML method. 20

**neighbours** The neighbours of a node  $i \in \mathcal{V}$  within an empirical graph are those nodes  $i' \in \mathcal{V}$  that are connected by an edge with  $i$ . 4

**node degree** The degree of a node  $i$  in an undirected empirical graph is the number of its neighbours. 3–7

**objective function** An objective function is a map that assigns each possible value of an optimization variable, such as the parameters  $\mathbf{w}$  of a hypothesis  $h^{(\mathbf{w})}$ , to an objective value  $f(\mathbf{w})$ . The objective value  $f(\mathbf{w})$  could be the risk or the empirical risk of a hypothesis  $h^{(\mathbf{w})}$ . 1–4, 6, 7, 9, 10, 13, 14, 16, 17, 28

**overfitting** Consider a ML method that uses ERM to learn a hypothesis with minimum empirical risk on a given training set. Such a method is “overfitting” the training set if it learns hypothesis with small empirical risk on the training set but unacceptably large loss outside the training set. 9, 11

**parameters** The parameters of a ML model are tunable (learnable or adjustable) quantities that allow to choose between different hypothesis maps. For example, the linear model  $\mathcal{H} := \{h : h(x) = w_1x + w_2\}$  consists of all hypothesis maps  $h(x) = w_1x + w_2$  with a particular choice for the parameters  $w_1, w_2$ . Another example of parameters are the weights assigned to the connections of an ANN. 1–17, 27

**polynomial regression** Polynomial regression aims at learning a polynomial hypothesis map to predict a numeric label based on numeric

features of a data point. For data points characterized by a single numeric features, polynomial regression uses the hypothesis space  $\mathcal{H}_d^{(\text{poly})} := \{h(x) = \sum_{j=0}^{d-1} x^j w_j\}$ . The quality of a polynomial hypothesis map is measured using the average squared error loss incurred on a set of labeled data points (which we refer to as training set). 12

**positive semi-definite** A symmetric matrix  $\mathbf{Q} = \mathbf{Q}^T \in \mathbb{R}^{d \times d}$  is referred to as positive semi-definite if  $\mathbf{x}^T \mathbf{Q} \mathbf{x} \geq 0$  for every vector  $\mathbf{x} \in \mathbb{R}^d$ . 2, 4–6, 9, 14

**prediction** A prediction is an estimate or approximation for some quantity of interest. ML revolves around learning or finding a hypothesis map  $h$  that reads in the features  $\mathbf{x}$  of a data point and delivers a prediction  $\hat{y} := h(\mathbf{x})$  for its label  $y$ . 4, 6, 13, 15, 16, 18, 24

**privacy leakage** Consider a (ML or FL) system that processes a local dataset  $\mathcal{D}^{(i)}$  and shares data, such as the predictions obtained for new data points, with other parties. Privacy leakage arises if the shared data carries information about a private (sensitive) feature of a data point (which might be a human) of  $\mathcal{D}^{(i)}$ . The amount of privacy leakage can be measured via mutual information using a probabilistic model for the local dataset. 20

**privacy protection** Privacy protection aims at avoiding (or minimizing) the privacy leakage occurring within data processing systems (such as ML or FL methods). 1

**probabilistic model** A probabilistic model interprets data points as realiza-

tions of RVs with a joint probability distribution. This joint probability distribution typically involves parameters which have to be manually chosen (=design choice) or learnt via statistical inference methods [29]. 2, 4–8, 10–12, 15, 16, 20, 25

**probability density function (pdf)** The probability density function (pdf)  $p(x)$  of a real-valued RV  $x \in \mathbb{R}$  is a particular representation of its probability distribution. If the pdf exists, it can be used to compute the probability that  $x$  takes on a value from a (measurable) set  $\mathcal{B} \subseteq \mathbb{R}$  via  $p(x \in \mathcal{B}) = \int_{\mathcal{B}} p(x') dx'$  [3, Ch. 3]. The pdf of a vector-valued RV  $\mathbf{x} \in \mathbb{R}^d$  (if it exists) allows to compute the probability that  $\mathbf{x}$  falls into a (measurable) region  $\mathcal{R}$  via  $p(\mathbf{x} \in \mathcal{R}) = \int_{\mathcal{R}} p(\mathbf{x}') dx'_1 \dots dx'_d$  [3, Ch. 3]. 13

**probability distribution** The data generated in some ML applications can be reasonably well modelled as realizations of a RV. The overall statistical properties (or intrinsic structure) of such data are then governed by the probability distribution of this RV. We use the term probability distribution in a highly informal manner and mean the collection of probabilities assigned to different values or value ranges of a RV. The probability distribution of a binary RV  $y \in \{0, 1\}$  is fully specified by the probabilities  $p(y = 0)$  and  $p(y = 1) (= 1 - p(y = 0))$ . The probability distribution of a real-valued RV  $x \in \mathbb{R}$  might be specified by a probability density function  $p(x)$  such that  $p(x \in [a, b]) \approx p(a)|b - a|$ . In the most general case, a probability distribution is defined by a probability measure [70, 72]. 3–6, 10, 12–14, 17, 18, 21, 25

**projected GD** Projected GD extends basic GD for unconstrained opti-

mization to handle constraints on the optimization variable (model parameters). A single iteration of projected GD consists of first taking a gradient step and then projecting the result back into a constrain set. 1, 7, 10, 12

**proximable** A Convex function for which the proximity operator can be computed efficiently are sometimes referred to as “proximable” or “simple” [35]. 10

**proximal operator** Given a convex function and a vector  $\mathbf{x}$ , we define its proximity operator as

$$\mathbf{prox}_{L_i(\cdot), 2\alpha}(\mathbf{w}'') := \underset{\mathbf{w} \in \mathbb{R}^d}{\operatorname{argmin}} f(\mathbf{w}) + (\rho/2) \|\mathbf{w} - \mathbf{w}'\|_2^2 \text{ with } \rho > 0.$$

Convex functions for which the proximity operator can be computed efficiently are sometimes referred to as “proximable” or “simple” [35]. 10, 11, 13, 22

**quadratic function** A quadratic function  $f(\mathbf{w})$ , reading in a vector  $\mathbf{w} \in \mathbb{R}^d$  as its argument, is such that

$$f(\mathbf{w}) = \mathbf{w}^T \mathbf{Q} \mathbf{w} + \mathbf{q}^T \mathbf{w} + a,$$

with some matrix  $\mathbf{Q} \in \mathbb{R}^{d \times d}$ , vector  $\mathbf{q} \in \mathbb{R}^d$  and scalar  $a \in \mathbb{R}$ . 2, 4, 7, 9, 15

**random variable (RV)** A random variable is a mapping from a probability space  $\mathcal{P}$  to a value space [72]. The probability space, whose elements are elementary events, is equipped with a probability measure that



assigns a probability to subsets of  $\mathcal{P}$ . A binary random variable maps elementary events to a set containing two different values, e.g.,  $\{-1, 1\}$  or  $\{\text{cat}, \text{no cat}\}$ . A real-valued random variable maps elementary events to real numbers  $\mathbb{R}$ . A vector-valued random variable maps elementary events to the Euclidean space  $\mathbb{R}^d$ . Probability theory uses the concept of measurable spaces to rigorously define and study the properties of (large) collections of random variables [70, 72]. 2–4, 6, 10, 12–14, 17, 18, 21, 23, 24, 28

**realization** Consider a RV  $x$  which maps each element (outcome, or elementary event)  $\omega \in \mathcal{P}$  of a probability space  $\mathcal{P}$  to an element  $a$  of a measurable space  $\mathcal{N}$  [5, 72, 73]. A realization of  $x$  is any element  $a' \in \mathcal{N}$  such that there is an element  $\omega' \in \mathcal{P}$  with  $x(\omega') = a'$ . 3–6, 12–14, 20, 24

**regression** Regresion problems revolve around the problem of predicting a numeric label solely from the features of a data point. 12

**regularization** Regularization techniques modify the ERM principle such that the learnt hypothesis performs well (generalizes) beyond the training set. One specific implementation of regularization is to add a penalty or regularization term to the objective function of ERM (which is the average loss on the training set). This regularization term can be interpreted as an estimate for the increase in the expected loss (risk) compared to the average loss on the training set. 1, 5, 7, 9–11, 13, 14, 23, 24, 26

**regularized empirical risk minimization** Synonym for SRM. 7, 8, 12, 13

**regularizer** A regularizer assigns each hypothesis  $h$  from a hypothesis space  $\mathcal{H}$  a quantitative measure  $\mathcal{R}\{h\}$  for how much its prediction error on a training set might differ from its prediction errors on data points outside the training set. Ridge regression uses the regularizer  $\mathcal{R}\{h\} := \|\mathbf{w}\|_2^2$  for linear hypothesis maps  $h^{(\mathbf{w})}(\mathbf{x}) := \mathbf{w}^T \mathbf{x}$  [4, Ch. 3]. The least absolute shrinkage and selection operator (Lasso) uses the regularizer  $\mathcal{R}\{h\} := \|\mathbf{w}\|_1$  for linear hypothesis maps  $h^{(\mathbf{w})}(\mathbf{x}) := \mathbf{w}^T \mathbf{x}$  [4, Ch. 3]. 7, 8, 11, 12, 14

**ridge regression** Ridge regression learns the parameter (or weight) vector  $\mathbf{w}$  of a linear hypothesis map  $h^{(\mathbf{w})}(\mathbf{x}) = \mathbf{w}^T \mathbf{x}$ . The quality of a particular choice for the parameter vector  $\mathbf{w}$  is measured by the sum of two components. The first component is the average squared error loss incurred by  $h^{(\mathbf{w})}$  on a set of labeled data points (the training set). The second component is the scaled squared Euclidean norm  $\alpha \|\mathbf{w}\|_2^2$  with a regularization parameter  $\alpha > 0$ . It can be shown that the effect of adding to  $\alpha \|\mathbf{w}\|_2^2$  to the average squared error loss is equivalent to replacing the original data points by an ensemble of realizations of a RV centered around these data points. 2, 9, 11, 14–17, 24

**risk** Consider a hypothesis  $h$  that is used to predict the label  $y$  of a data point based on its features  $\mathbf{x}$ . We measure the quality of a particular prediction using a loss function  $L((\mathbf{x}, y), h)$ . If we interpret data points as the realizations of i.i.d. RVs, also the  $L((\mathbf{x}, y), h)$  becomes the realization of a RV. Using such an i.i.d. assumption allows to define the risk of a hypothesis as the expected loss  $\mathbb{E}\{L((\mathbf{x}, y), h)\}$ . Note that

the risk of  $h$  depends on both, the specific choice for the loss function and the probability distribution of the data points. 2, 6, 10, 19, 23

**scatterplot** A visualization technique that depicts data points by markers in a two-dimensional plane. 9

**semi-supervised learning** Semi-supervised learning methods use (large amounts of) unlabeled data points to support the learning of a hypothesis from (a small number of) labeled data points [49]. 6, 7

**smooth** We refer to a real-valued function as smooth if it is differentiable and its gradient is continuous [45, 74]. In particular, a differentiable function  $f(\mathbf{w})$  is referred to as  $\beta$ -smooth if the gradient  $\nabla f(\mathbf{w})$  is Lipschitz continuous with Lipschitz constant  $\beta$ , i.e.,

$$\|\nabla f(\mathbf{w}) - \nabla f(\mathbf{w}')\| \leq \beta \|\mathbf{w} - \mathbf{w}'\|.$$

1, 3, 5, 28

**squared error loss** The squared error loss measures the prediction error of a hypothesis  $h$  when predicting a numeric label  $y \in \mathbb{R}$  from the features  $\mathbf{x}$  of a data point. It is defined as

$$L((\mathbf{x}, y), h) := \left(y - \underbrace{h(\mathbf{x})}_{=\hat{y}}\right)^2. \quad (10.2)$$

2, 3, 7–9, 12, 15, 16, 20, 24

**statistical aspects** By statistical aspects of a ML method, we refer to (properties of) the probability distribution of its output given a probabilistic model for the data fed into the method. 1, 4, 7, 15, 16

**step size** Many ML methods use iterative optimization methods (such as gradient-based methods) to construct a sequence of increasingly accurate hypothesis maps  $h^{(1)}, h^{(2)}, \dots$ . The  $k$ th iteration of such an algorithm starts from the current hypothesis  $h^{(k)}$  and tries to modify it to obtain an improved hypothesis  $h^{(k+1)}$ . Iterative algorithms often use a step size (hyper-) parameter. The step size controls the amount by which a single iteration can change or modify the current hypothesis. Since the overall goal of such iteration ML methods is to learn a (approximately) optimal hypothesis we refer to a step size parameter also as a learning rate. 1

**stochastic gradient descent** Stochastic GD is obtained from GD by replacing the gradient of the objective function with a noisy (or stochastic) estimate. 1

**stopping criterion** Many ML methods use iterative algorithms that construct a sequence of model parameters (such as the weights of a linear map or the weights of an ANN) that (hopefully) converge to an optimal choice for the model parameters. In practice, given finite computational resources, we need to stop iterating after a finite number of times. A stopping criterion is any well-defined condition required for stopping iterating. 1–6, 8, 11–13

**structural risk minimization** Structural risk minimization is the problem of finding the hypothesis that optimally balances the average loss (or empirical risk) on a training set with a regularization term. The regularization term penalizes a hypothesis that is not robust against (small)

perturbations of the data points in the training set. 15, 23

**total variation** Consider some empirical graph whose nodes carry (personalized) local model parameters. The total variation measures the discrepancies between across edges  $\{i, i'\}$ . 1, 3, 6, 7, 15, 16

**training error** The average loss of a hypothesis when predicting the labels of data points in a training set. We sometimes refer by training error also the minimum average loss incurred on the training set by any hypothesis out of a hypothesis space. 1, 9–17, 27, 28

**training set** A set of data points that is used in ERM to learn a hypothesis  $\hat{h}$ . The average loss of  $\hat{h}$  on the training set is referred to as the training error. The comparison between training error and validation error of  $\hat{h}$  allows to diagnose ML methods and informs how to improve them (e.g., using a different hypothesis space or collecting more data points). 3–9, 11–16, 19, 20, 23, 24, 26–28

**validation** Consider a hypothesis  $\hat{h}$  that has been learn via ERM on some training set  $\mathcal{D}$ . Validation refers to the practice of trying out a hypothesis  $\hat{h}$  on a validation set that consists of data points that are not contained in the training set  $\mathcal{D}$ . 1, 6, 9

**validation error** Consider a hypothesis  $\hat{h}$  which is obtained by ERM on a training set. The average loss of  $\hat{h}$  on a validation set, which is different from the training set, is referred to as the validation error. 1, 9–13, 15–17, 27, 28

**validation set** A set of data points that have not been used as training set in ERM to learn a hypothesis  $\hat{h}$ . The average loss of  $\hat{h}$  on the validation set is referred to as the validation error and used to diagnose the ML method (see [4, Sec. 6.6.]). The comparison between training error and validation error can inform directions for improvements of the ML method (such as using a different hypothesis space). 9, 12, 13, 15, 16, 27

**variance** The variance of a real-valued RV  $x$  is defined as the expectation  $\mathbb{E}\{(x - \mathbb{E}\{x\})^2\}$  of the squared difference  $x$  and its expectation  $\mathbb{E}\{x\}$ . We extend this definition to vector-valued RVs  $\mathbf{x}$  as  $\mathbb{E}\{\|\mathbf{x} - \mathbb{E}\{\mathbf{x}\}\|_2^2\}$ . 13

**vertical FL** Vertical FL refers to applications with local datasets that are constituted by the same data points but characterizing them with different features [50]. For example, different healthcare providers might all contain information about the same population of patients. However, different healthcare providers collect different measurements (blood values, electrocardiography, lung x-ray) for the same patients. 8, 9

**zero-gradient condition** Consider the unconstrained optimization problem  $\min_{\mathbf{w} \in \mathbb{R}^d} f(\mathbf{w})$  with a smooth and convex objective function  $f(\mathbf{w})$ . A necessary and sufficient condition for a vector  $\hat{\mathbf{w}} \in \mathbb{R}^d$  to solve this problem is that the gradient  $\nabla f(\hat{\mathbf{w}})$  is the zero-vector,

$$\nabla f(\hat{\mathbf{w}}) = \mathbf{0} \Leftrightarrow f(\hat{\mathbf{w}}) = \min_{\mathbf{w} \in \mathbb{R}^d} f(\mathbf{w}).$$

10

## References

- [1] W. Rudin, *Real and Complex Analysis*, 3rd ed. New York: McGraw-Hill, 1987.
- [2] 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.
- [3] D. Bertsekas and J. Tsitsiklis, *Introduction to Probability*, 2nd ed. Athena Scientific, 2008.
- [4] A. Jung, *Machine Learning: The Basics*, 1st ed. Springer Singapore, Feb. 2022.
- [5] W. Rudin, *Principles of Mathematical Analysis*, 3rd ed. New York: McGraw-Hill, 1976.
- [6] 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.
- [7] 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>
- [8] 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>

- [9] 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>
- [10] S. Cui, A. Hero, Z.-Q. Luo, and J. Moura, Eds., *Big Data over Networks*. Cambridge Univ. Press, 2016.
- [11] A. Barabási, N. Gulbahce, and J. Loscalzo, “Network medicine: a network-based approach to human disease,” *Nature Reviews Genetics*, vol. 12, no. 56, 2011.
- [12] M. E. J. Newman, *Networks: An Introduction*. Oxford Univ. Press, 2010.
- [13] 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>
- [14] 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.



- [15] 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.
- [16] 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.
- [17] 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>
- [18] J. You, J. Wu, X. Jin, and M. Chowdhury, “Ship compute or ship data? why not both?” in *18th USENIX Symposium on Networked Systems Design and Implementation (NSDI 21)*. USENIX Association, April 2021, pp. 633–651. [Online]. Available: <https://www.usenix.org/conference/nsdi21/presentation/you>
- [19] D. Tse and P. Viswanath, *Fundamentals of Wireless Communication*. Cambridge University Press, 2005.
- [20] T. Yang, G. Andrew, H. Eichner, H. Sun, W. Li, N. Kong, D. Ramage, and F. Beaufays, “Applied federated learning: Improving google keyboard query suggestions,” 2018. [Online]. Available: <https://arxiv.org/abs/1812.02903>
- [21] A. Ghosh, J. Chung, D. Yin, and K. Ramchandran, “An efficient framework for clustered federated learning,” in *34th Conference on Neural*

- Information Processing Systems (NeurIPS 2020)*, Vancouver, Canada, 2020.
- [22] F. Sattler, K. Müller, and W. Samek, “Clustered federated learning: Model-agnostic distributed multitask optimization under privacy constraints,” *IEEE Transactions on Neural Networks and Learning Systems*, 2020.
  - [23] G. Strang, *Computational Science and Engineering*. Wellesley-Cambridge Press, MA, 2007.
  - [24] ———, *Introduction to Linear Algebra*, 5th ed. Wellesley-Cambridge Press, MA, 2016.
  - [25] H. H. Bauschke and P. L. Combettes, *Convex Analysis and Monotone Operator Theory in Hilbert Spaces*. New York: Springer, 2011.
  - [26] F. Pedregosa, “Scikit-learn: Machine learning in python,” *Journal of Machine Learning Research*, vol. 12, no. 85, pp. 2825–2830, 2011. [Online]. Available: <http://jmlr.org/papers/v12/pedregosa11a.html>
  - [27] J. Hirvonen and J. Suomela. (2023) Distributed algorithms 2020.
  - [28] S. Boyd and L. Vandenberghe, *Convex Optimization*. Cambridge, UK: Cambridge Univ. Press, 2004.
  - [29] E. L. Lehmann and G. Casella, *Theory of Point Estimation*, 2nd ed. New York: Springer, 1998.

- [30] 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.
- [31] H. Lütkepohl, *New Introduction to Multiple Time Series Analysis*. New York: Springer, 2005.
- [32] 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.
- [33] D. Spielman, “Spectral graph theory,” in *Combinatorial Scientific Computing*, U. Naumann and O. Schenk, Eds. Chapman and Hall/CRC, 2012.
- [34] G. H. Golub and C. F. Van Loan, *Matrix Computations*, 4th ed. Baltimore, MD: Johns Hopkins University Press, 2013.
- [35] 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.
- [36] N. Parikh and S. Boyd, “Proximal algorithms,” *Foundations and Trends in Optimization*, vol. 1, no. 3, pp. 123–231, 2013.
- [37] 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.

- [38] 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/10.1561/04000000054>
- [39] 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](https://proceedings.neurips.cc/paper_files/paper/2018/file/5a4be1fa34e62bb8a6ec6b91d2462f5a-Paper.pdf)
- [40] 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>
- [41] 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>
- [42] 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.
- [43] D. P. Bertsekas, *Convex Optimization Algorithms*. Athena Scientific, 2015.
- [44] Y. L. T. Schaul, X. Zhang, “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.
- [45] 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>
- [46] R. Diestel, *Graph Theory*. Springer Berlin Heidelberg, 2005.
- [47] Q. Yang, Y. Liu, Y. Cheng, Y. Kang, T. Chen, and H. Yu, *Federated Learning*, 1st ed. Springer, 2022.
- [48] ———, *Horizontal Federated Learning*. Cham: Springer International Publishing, 2020, pp. 49–67. [Online]. Available: [https://doi.org/10.1007/978-3-031-01585-4\\_4](https://doi.org/10.1007/978-3-031-01585-4_4)
- [49] O. Chapelle, B. Schölkopf, and A. Zien, Eds., *Semi-Supervised Learning*. Cambridge, Massachusetts: The MIT Press, 2006.

- [50] 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](https://doi.org/10.1007/978-3-031-01585-4_5)
- [51] H. Ludwig and N. Baracaldo, Eds., *Federated Learning: A Comprehensive Overview of Methods and Applications*. Springer, 2022.
- [52] 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>
- [53] D. Spielman, “Spectral and algebraic graph theory,” 2019.
- [54] J. Tropp, “An introduction to matrix concentration inequalities,” *Found. Trends Mach. Learn.*, May 2015.
- [55] S. M. Kay, *Fundamentals of Statistical Signal Processing: Estimation Theory*. Englewood Cliffs, NJ: Prentice Hall, 1993.
- [56] 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.
- [57] 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>
- [58] T. M. Cover and J. A. Thomas, *Elements of Information Theory*, 2nd ed. New Jersey: Wiley, 2006.
  - [59] X. Liu, H. Li, G. Xu, Z. Chen, X. Huang, and R. Lu, “Privacy-enhanced federated learning against poisoning adversaries,” *IEEE Transactions on Information Forensics and Security*, vol. 16, pp. 4574–4588, 2021.
  - [60] J. Zhang, B. Chen, X. Cheng, H. T. T. Binh, and S. Yu, “PoisonGAN: Generative poisoning attacks against federated learning in edge computing systems,” *IEEE Internet of Things Journal*, vol. 8, no. 5, pp. 3310–3322, 2021.
  - [61] P. Halmos, *Naive set theory*. Springer-Verlag, 1974.
  - [62] I. Goodfellow, Y. Bengio, and A. Courville, *Deep Learning*. MIT Press, 2016.
  - [63] D. Gujarati and D. Porter, *Basic Econometrics*. McGraw Hill, 2009.
  - [64] Y. Dodge, *The Oxford Dictionary of Statistical Terms*. Oxford University Press, 2003.
  - [65] B. Everitt, *Cambridge Dictionary of Statistics*. Cambridge University Press, 2002.
  - [66] R. T. Rockafellar, *Network Flows and Monotropic Optimization*. Athena Scientific, Jul. 1998.

- [67] A. Jung and P. Nardelli, “An information-theoretic approach to personalized explainable machine learning,” *IEEE Sig. Proc. Lett.*, vol. 27, pp. 825–829, 2020.
- [68] U. von Luxburg, “A tutorial on spectral clustering,” *Statistics and Computing*, vol. 17, no. 4, pp. 395–416, Dec. 2007.
- [69] A. Y. Ng, M. I. Jordan, and Y. Weiss, “On spectral clustering: Analysis and an algorithm,” in *Adv. Neur. Inf. Proc. Syst.*, 2001.
- [70] R. Gray, *Probability, Random Processes, and Ergodic Properties*, 2nd ed. New York: Springer, 2009.
- [71] A. Lapidoth, *A Foundation in Digital Communication*. New York: Cambridge University Press, 2009.
- [72] P. Billingsley, *Probability and Measure*, 3rd ed. New York: Wiley, 1995.
- [73] P. R. Halmos, *Measure Theory*. New York: Springer, 1974.
- [74] S. Bubeck, “Convex optimization. algorithms and complexity.” in *Foundations and Trends in Machine Learning*. Now Publishers, 2015, vol. 8.