TECHNICAL UNIVERSITY OF CRETE
SCHOOL OF ELECTRICAL AND COMPUTER ENGINEERING

Distributed Optimization Algorithms: Performance Analysis and Application in Machine Learning Problems

Technical University of Crete

by

Ioannis Chariompolis

A THESIS SUBMITTED IN PARTIAL FULFILLMENT OF

THE REQUIREMENTS FOR THE DIPLOMA DEGREE OF

ELECTRICAL AND COMPUTER ENGINEERING

February 2025

THESIS COMMITTEE

Professor Athanasios Liavas, *Thesis Supervisor*
Professor Georgios Karystinos
Professor Thrasyvoulos Spyropoulos

Abstract

Distributed optimization allows a set of agents to collectively solve a global minimization problem, defined as the numerical average of each agent's local objective function. In this thesis we examine distributed optimization algorithms in the setting where the agents communicate via a network, where each agent can only communicate with its immediate neighbors, and can not share its objective function (and by extension its training data) directly. More specifically, we investigate how these algorithms can be applied to the classical Machine Learning framework of training a parametrized model to predict labels y from input data x. Typically, distributed optimization algorithms over networks have two main steps: the pooling and then averaging of the parameters of each agent's neighbors via the consensus protocol, and a gradient descent step towards the minimum of the local function of each agent. We also consider algorithms that build on this scheme by incorporating correction terms, acceleration via momentum, dual ascent, and multiple communication rounds per gradient computation. We experimentally evaluate the performance of a number of distributed algorithms for functions of different classes, namely smooth, and strongly convex functions, as well as objectives with constraints.

Περίληψη

Οι αλγόριθμοι κατανεμημένης βελτιστοποίησης επιτρέπουν σε ένα σετ από πράκτορες να λύσουν συλλογικά ένα καθολικό πρόβλημα ελαχιστοποίησης, ορισμένο ως τον αριθμητικό μέσο της τοπικής συνάρτησης κάθε πράκτορα. Σε αυτή τη διπλωματική εργασία εξετάζουμε κατανεμημένους αλγόριθμους βελτιστοποίησης στο πλαίσιο όπου οι πράκτορες επικοινωνούν μέσω ενός δικτύου, όπου κάθε πράκτορας μπορεί να επικοινωνήσει μόνο με τους άμεσους γείτονες του, και δεν μπορεί να μοιραστεί την αντικειμενική συνάρτηση (και κατ΄ επέκταση τα δεδομένα εκπαίδευσης) άμεσα. Πιο συγκεκριμένα, ερευνούμε πως αυτοί οι αλγόριθμοι μπορούν να εφαρμοστούν στο κλασσικό πλαίσιο Μηχανικής Μάθησης της εκπαίδευσης ενός παραμετρικού μοντέλου ώστε να προβλέπει ετικέτες y από δεδομένα εισόδου x. Συνήθως, οι κατανεμημένοι αλγόριθμοι βελτιστοποίησης σε δίκτυα αποτελούνται από δύο βήματα: τη συγκέντρωση και υπολογισμό της μέσης τιμής των παραμέτρων των γειτόνων κάθε πράκτορα, και ένα βήμα καθόδου βαθμίδας προς το ελάχιστο της τοπικής συνάρτησης κάθε πράκτορα. Επιπλέον, θεωρούμε αλγορίθμους που χτίζουν πάνω σε αυτό το σχέδιο με την ενσωμάτωση όρων διόρθωσης, επιτάχυνσης μέσω ορμής, διπλής ανάβασης, και πολλαπλών γύρων επικοινωνίας για κάθε υπολογισμό βαθμίδας. Αξιολογούμε πειραματικά την απόδοση ορισμένων κατανεμημένων αλγορίθμων για συναρτήσεις διαφορετικών κλάσεων, συγκεκριμένα ομαλές, και ισχυρά κυρτές συναρτήσεις, καθώς και προβλήματα με περιορισμούς.

Contents

Chapter 1

Introduction

1.1 Problem

In this thesis, we focus on distributed optimization algorithms and, more specifically, on their application for the training of machine learning (ML) models in a decentralized fashion.

In a typical (supervised) ML environment, one has access to a training dataset consisting of pairs $\{(\mathbf{x}_i, y_i)\}_{i=1}^n$, and tries to fit a model function that is parametrized by parameter $\boldsymbol{\theta}$. This is typically done by minimizing the empirical average of the loss function, i.e. the difference between the predicted output from the model, given a specific input, and the corresponding desired output: $\min_{\boldsymbol{\theta}} \frac{1}{n} \sum_{i=1}^n \ell(\boldsymbol{\theta}; \mathbf{x}_i, y_i)$.

Consider a system of N agents which communicate via a network described by a graph. Each agent has access to its own data and, therefore, local cost function. The goal is to collaboratively minimize the average of the N local cost functions, without directly sharing data between agents. Each agent's model is parametrized, as before, by a parameter $\boldsymbol{\theta}_i$. The goal then is for all agents to agree on a common parameter $\boldsymbol{\theta}$. The global problem is formulated as follows:

$$\begin{aligned} \underset{\boldsymbol{\theta}_i \in \mathbb{R}^p,\, i \in [N]}{\text{minimize}} \quad & \frac{1}{N} \sum_{k=1}^{N} f_i(\boldsymbol{\theta}_i) \\ \text{subject to} \quad & \boldsymbol{\theta}_i = \boldsymbol{\theta}_j,\ \forall \{i, j\} \in [N] \times [N], \end{aligned} \tag{1.1}$$

where $f_i(\cdot)$ is the average of the losses for the data points of agent i.

A consensus mechanism can be used to align each agent's estimate of the parameter, such that they asymptotically agree on a common variable. Roughly, at each iteration, each agent broadcasts its current estimate of the parameter, receives its neighbors' estimates and finally updates its own estimate by taking a weighted average of the estimates of its neighbors and its own. Given enough iterations and under some assumptions on the underlying graph, the agents will agree.

The combination of standard gradient descent methods with the consensus mechanism gives rise to a plethora of algorithms that allow agents who are part of a communication graph to collaboratively train an ML model.

Various such methods exist that tackle problems with different loss function properties (Lipschitz continuity and/or strong convexity) as well as graph structures (directed or undirected). Additionally, recent methods provide significant speedups by incorporating auxiliary information, making use of momentum, and applying cumulative corrections to the updates.

1.2 Importance

The volume of data each person produces has been growing rapidly, images and videos grow in resolution (and, by extension, size), new text is constantly being written and Internet of Things (IoT) devices are constantly downloading and uploading new data to cloud services.

This new regime of Big Data has given rise to multiple ways of training ML models in a distributed fashion. Distributed training methods are of interest, since they are either faster than transferring all local datasets to a single server, or individual users/organizations do not desire to share their highly sensitive data with potentially untrusted parties, or both.

Furthermore, in recent years, there has been a shift to individuals being more aware of their data (GDPR, cookies, personalized advertisements), as well as preferring decentralized ways to perform certain tasks (file sharing via torrents, monetary transactions via blockchain).

Problems of the form (1.1) have appeared in multiple fields, including distributed learning and estimation, multi-vehicle formation and coordination, sensor networks, power system control. Distributed optimization algorithms are applicable when it is impractical to move all data to a central server, or constant communication between a central server and all agents is not feasible, or both.

1.3 Notation

We briefly lay out some of the notation used in the following text.

A bold lower case letter represents a column vector, say $\mathbf{v} \in \mathbb{R}^n$

$$\mathbf{v} = (v_1, \ldots, v_n) = \begin{bmatrix} v_1 \\ \vdots \\ v_n \end{bmatrix} = \begin{bmatrix} v_1, \ldots, v_n \end{bmatrix}^\top .$$

A vector's elements, and all other reals, are in regular lower case, and $[\cdot]^\top$ represents the transpose operator.

We denote the basis vectors of $\mathbb{R}^n$ as $\mathbf{e}_1, \ldots, \mathbf{e}_n$, with $\mathbf{e}_i$ being the column vector of n elements, the i-th of which is one while the rest are zeros.

Further, $\mathbf{0}$ and $\mathbf{1}$ are vectors of all zeros and all ones, respectively.

Bold upper case letters are reserved for matrices, $\mathbf{A} \in \mathbb{R}^{m \times n}$.

Chapter 2

Mathematical Preliminaries

2.1 Linear Algebra

Definition 2.1 (**dot product**). *Let vectors* $\mathbf{x} \in \mathbb{R}^n$ *and* $\mathbf{y} \in \mathbb{R}^n$. *Their* dot product *is defined as*

$$\langle \mathbf{x}, \mathbf{y} \rangle \coloneqq \mathbf{x}^\top \mathbf{y} = \mathbf{y}^\top \mathbf{x} = \sum_{i=1}^{n} x_i y_i. \tag{2.1}$$

Definition 2.2 (**norm**). *A* norm $\|\cdot\|$ *on a vector space* $\mathbb{V}$ *is a function* $\|\cdot\| \colon \mathbb{V} \to \mathbb{R}$ *which satisfies the following:*

1. **(nonnegativity)** $\|\mathbf{x}\| \geq 0$, $\forall \mathbf{x} \in \mathbb{V}$ *and* $\|\mathbf{x}\| = 0$ *if, and only if,* $\mathbf{x} = \mathbf{0}$.
2. **(positive homogeneity)** $\|\alpha \mathbf{x}\| = |\alpha| \cdot \|\mathbf{x}\|$ *for any* $\mathbf{x} \in \mathbb{V}$ *and* $\alpha \in \mathbb{R}$.
3. **(triangle inequality)** $\|\mathbf{x} + \mathbf{y}\| \leq \|\mathbf{x}\| + \|\mathbf{y}\|$ *for any* $\mathbf{x}, \mathbf{y} \in \mathbb{V}$.

When $\mathbb{R}^n$ is endowed with the *dot product*, the Euclidean norm is the l_2-norm

$$\|\mathbf{x}\|_2 = \langle \mathbf{x}, \mathbf{x} \rangle = \sqrt{\sum_{i=1}^{n} x_i^2}. \tag{2.2}$$

The class of p-norms is defined for any $p \geq 1$, as

$$\|\mathbf{x}\|_p \coloneqq \sqrt[p]{\sum_{i=1}^{n} |x_i|^p}. \tag{2.3}$$

Definition 2.3 (**matrix inner product**). *Let matrices* $\mathbf{X} \in \mathbb{R}^{m \times n}$ *and* $\mathbf{Y} \in \mathbb{R}^{m \times n}$. *Their* matrix inner product *is defined as*

$$\langle \mathbf{X}, \mathbf{Y} \rangle \coloneqq \operatorname{tr}(\mathbf{X}^\top \mathbf{Y}) = \sum_{i,j} X_{ij} Y_{ij}. \tag{2.4}$$

2.2 Real Analysis

Definition 2.4 (**differentiability**). *A function* $f : \mathbb{R}^n \to \mathbb{R}$ *is said to be* differentiable *at* $\mathbf{x} \in \mathbb{R}^n$ *if there exists a vector* $\mathbf{g} \in \mathbb{R}^n$ *such that for all* $\mathbf{y} \in \mathbb{R}^n$, *the following holds:*

$$f(\mathbf{x} + \mathbf{y}) = f(\mathbf{x}) + \langle \mathbf{g}, \mathbf{y} \rangle + o(\mathbf{y}). \tag{2.5}$$

The unique vector $\mathbf{g}$ *which satisfies* (2.5) *is called the* gradient *of* f *at* $\mathbf{x}$*, and is denoted by* $\nabla f(\mathbf{x})$*, i.e.,*

$$f(\mathbf{x}+\mathbf{y}) = f(\mathbf{x}) + \langle \nabla f(\mathbf{x}), \mathbf{y} \rangle + o(\mathbf{y}). \tag{2.6}$$

More intuitively, a function f is differentiable at $\mathbf{x}$ if we can find a linear function $g(\mathbf{y}) = f(\mathbf{x}) + \langle \nabla f(\mathbf{x}), \mathbf{y} \rangle$ such that $f(\mathbf{x}+\mathbf{y}) - g(\mathbf{y}) = o(\mathbf{y})$.

Definition 2.5 (**gradient**)**.** *The* gradient *of a differentiable function* $f\colon \mathbb{R}^n \to \mathbb{R}$ *at a point* $\mathbf{x} \in \mathbb{R}^n$ *is equal to*

$$\nabla f(\mathbf{x}) \coloneqq \begin{bmatrix} \frac{\partial f(\mathbf{x})}{\partial x_1} \\ \vdots \\ \frac{\partial f(\mathbf{x})}{\partial x_n} \end{bmatrix}. \tag{2.7}$$

Definition 2.6 (**Hessian**)**.** *Let* $f\colon \mathbb{R}^n \to \mathbb{R}$*. If* ∇f *is differentiable at* $\mathbf{x} \in \mathbb{R}^n$*, then* f *is doubly differentiable at* $\mathbf{x}$*. The* second derivative *of* f *is the derivative of* ∇f *and is defined as*

$$\nabla^2 f(\mathbf{x}) \coloneqq \begin{bmatrix} \frac{\partial^2 f(\mathbf{x})}{\partial x_1^2} & \frac{\partial^2 f(\mathbf{x})}{\partial x_2 x_1} & \cdots & \frac{\partial^2 f(\mathbf{x})}{\partial x_n x_1} \\ \frac{\partial^2 f(\mathbf{x})}{\partial x_1 x_2} & \frac{\partial^2 f(\mathbf{x})}{\partial x_2^2} & \cdots & \frac{\partial^2 f(\mathbf{x})}{\partial x_n x_2} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{\partial^2 f(\mathbf{x})}{\partial x_1 x_n} & \frac{\partial^2 f(\mathbf{x})}{\partial x_2 x_n} & \cdots & \frac{\partial^2 f(\mathbf{x})}{\partial x_n^2} \end{bmatrix}. \tag{2.8}$$

The $n \times n$ *matrix* $\nabla^2 f(\mathbf{x})$ *is called the* Hessian *of* f *at* $\mathbf{x}$*.*

2.3 Convex Functions

Definition 2.7 (**convex set**)**.** *A set* $\mathbb{C} \subseteq \mathbb{R}^n$ *is called* convex *if, for all pairs* $\mathbf{x}, \mathbf{y} \in \mathbb{C}$ *and reals* $0 \leq \theta \leq 1$*, it holds that*

$$\theta\mathbf{x} + (1-\theta)\mathbf{y} \in \mathbb{C}. \tag{2.9}$$

Definition 2.8 (**projection mapping**)**.** *The projection of a point* $\mathbf{x} \in \mathbb{R}^n$ *onto a non-empty, closed, and convex set* $\mathcal{C}$ *is defined as*

$$P_{\mathcal{C}}(\mathbf{x}) \coloneqq \operatorname*{argmin}_{\mathbf{y} \in \mathcal{C}} \|\mathbf{y} - \mathbf{x}\|_2. \tag{2.10}$$

Thus, the projection mapping returns the point that belongs to $\mathcal{C}$ and is closest to the original point $\mathbf{x}$.

Definition 2.9 (**indicator function**)**.** *The* indicator function *of a set* $\mathcal{C}$ *is defined as*

$$\delta_{\mathcal{C}}(\mathbf{x}) = \begin{cases} 0, & \mathbf{x} \in \mathcal{C} \\ \infty, & \mathbf{x} \notin \mathcal{C}. \end{cases} \tag{2.11}$$

Definition 2.10 (**proximal mapping**)**.** *For a function* $f\colon \mathbb{R}^p \to (-\infty, \infty]$*, the* proximal mapping *of* f *is defined as*

$$\operatorname{prox}_f(\mathbf{x}) \coloneqq \operatorname*{argmin}_{\mathbf{y}} \left\{ f(\mathbf{y}) + \frac{1}{2}\|\mathbf{y} - \mathbf{x}\|_2^2 \right\}. \tag{2.12}$$

Definition 2.11 (**Fenchel conjugate**). *Let $f\colon \mathbb{R}^n \to \mathbb{R}$. The function $f^*\colon \mathbb{R}^n \to \mathbb{R}$, defined by*

$$f^*(\mathbf{y}) \coloneqq \max_{\mathbf{x}\in\mathbb{R}^n} \{\langle \mathbf{y}, \mathbf{x}\rangle - f(\mathbf{x})\}, \tag{2.13}$$

is called the Fenchel conjugate *of f.*

Definition 2.12 (**convex function**). *A function $f\colon \mathbb{R}^n \to \mathbb{R}$ is called* convex *if the set $\mathbf{dom}\, f$ is convex, and if, for all pairs $\mathbf{x}, \mathbf{y} \in \mathbf{dom}\, f$ and reals $0 \leq \theta \leq 1$, it holds that*

$$f(\theta\mathbf{x} + (1-\theta)\mathbf{y}) \leq \theta f(\mathbf{x}) + (1-\theta) f(\mathbf{y}). \tag{2.14}$$

Theorem 2.13 (**convexity condition of differentiable functions** [BV04]). *A differentiable function $f\colon \mathbb{R}^n \to \mathbb{R}$ (its gradient exists at each point in $\mathbf{dom}\, f$) is convex if and only if $\mathbf{dom}\, f$ is convex and $\forall \mathbf{x}, \mathbf{y} \in \mathbb{R}^n$*

$$f(\mathbf{y}) \geq f(\mathbf{x}) + \nabla f(\mathbf{x})^\top (\mathbf{y} - \mathbf{x}). \tag{2.15}$$

Inequality (2.15) states that the first-order Taylor approximation of a convex function is a global underestimator of the function at any point of its domain.

Definition 2.14 (**strong convexity**). *A function $f\colon \mathbb{R}^n \to \mathbb{R}$ is called* μ-strongly convex*, for $\mu > 0$, if the set $\mathbf{dom}\, f$ is convex, and for any $\mathbf{x}, \mathbf{y} \in \mathbf{dom}\, f$ and $\lambda \in [0, 1]$:*

$$f(\lambda\mathbf{x} + (1-\lambda)\mathbf{y}) \leq \lambda f(\mathbf{x}) + (1-\lambda) f(\mathbf{y}) - \frac{\mu}{2}\lambda(1-\lambda)\|\mathbf{x} - \mathbf{y}\|^2. \tag{2.16}$$

Lemma 2.15. [Bec17] *Let $f\colon \mathbb{R}^n \to \mathbb{R}$ be a μ-strongly convex function, and $g\colon \mathbb{R}^n \to \mathbb{R}$ be convex. Then $f + g$ is μ-strongly convex.*

Theorem 2.16 (**first-order inequality of strongly convex functions** [Bec17]). *Let $f\colon \mathbb{R}^n \to \mathbb{R}$ be μ-strongly convex. Then, for any $\mathbf{x} \in \mathbf{dom}\, \partial f$, $\mathbf{y} \in \mathbf{dom}\, f$, the following holds true:*

$$f(\mathbf{y}) \geq f(\mathbf{x}) + \langle \nabla f(\mathbf{x}), \mathbf{y} - \mathbf{x}\rangle + \frac{\mu}{2}\|\mathbf{y} - \mathbf{x}\|^2. \tag{2.17}$$

Definition 2.17 (L-**smoothness**). *A function $f\colon \mathbb{R}^n \to \mathbb{R}$ is called* L-smooth*, for $L > 0$, if it is differentiable and, for all $\mathbf{x}, \mathbf{y} \in \mathbf{dom}\, f$, satisfies:*

$$\|\nabla f(\mathbf{x}) - \nabla f(\mathbf{y})\| \leq L\|\mathbf{x} - \mathbf{y}\|. \tag{2.18}$$

Theorem 2.18 (**first-order inequality of smooth functions** [Bec17]). *Let $f\colon \mathbb{R}^n \to \mathbb{R}$ be L-smooth. Then, for any $\mathbf{x} \in \mathbf{dom}\, \partial f$, $\mathbf{y} \in \mathbf{dom}\, f$, the following holds true:*

$$f(\mathbf{y}) \leq f(\mathbf{x}) + \langle \nabla f(\mathbf{x}), \mathbf{y} - \mathbf{x}\rangle + \frac{L}{2}\|\mathbf{x} - \mathbf{y}\|^2. \tag{2.19}$$

One can notice an immediate parallel between Theorem 2.16 and 2.18. More specifically, a μ-strongly convex and L-smooth function is lower bounded by the RHS of (2.17) and upper bounded by the RHS of (2.19).

Chapter 3

Optimization

Mathematical (constrained) optimization tackles the problem of finding the minimum of a real valued function, possibly under inequality and/or equality constraints, that must hold for the solution to be feasible.

3.1 Optimization Problems

Definition 3.1 (**optimization problem**). *An* optimization problem *is defined as*

$$\begin{array}{ll} \text{minimize} & f_0(\mathbf{x}) \\ \text{subject to} & f_i(\mathbf{x}) \leq 0, \quad i = 1, \ldots, m, \\ & h_i(\mathbf{x}) = 0, \quad i = 1, \ldots, q, \end{array} \tag{3.1}$$

where $\mathbf{x} \in \mathbb{R}^p$ *is the optimization variable,* $f_0 \colon \mathbb{R}^p \to \mathbb{R}$ *the objective function, functions* $f_i \colon \mathbb{R}^p \to \mathbb{R}$ *for* $i = 1, \ldots, m$ *and* $h_i \colon \mathbb{R}^p \to \mathbb{R}$ *for* $i = 1, \ldots, q$ *are the inequality and equality constraints, respectively.*

The set of points where the objective function and all constraint functions are defined, is given by

$$\mathbb{D} = \bigcap_{i=0}^{m} \mathbf{dom} f_i \cap \bigcap_{i=1}^{q} \mathbf{dom} h_i, \tag{3.2}$$

and is called the domain of the optimization problem (3.1).

Definition 3.2 (**feasible point**). *A point* $\mathbf{x} \in \mathbb{D}$ *is called* feasible *if it satisfies all constraints.*

The set of feasible points of an optimization problem,

$$\mathbb{X} = \{\mathbf{x} \in \mathbb{D} \mid f_i(\mathbf{x}) \leq 0,\ i = 0, \ldots, m,\ h_i(\mathbf{x}) = 0,\ i = 1, \ldots, q\} \tag{3.3}$$

is called the *feasible set* of the problem.

Definition 3.3 (**optimal value**). *The* optimal value $p^\star$ *of problem (3.1) is*

$$p^\star = \inf\{f_0(\mathbf{x}) \mid \mathbf{x} \in \mathbb{X}\}. \tag{3.4}$$

Definition 3.4 (**optimal point**). *A point* $\mathbf{x}^\star \in \mathbb{X}$ *is called an* optimal point *if* $f_0(\mathbf{x}^\star) = p^\star$. *The set of all* optimal points *is the* optimal set, *defined as*

$$\mathbf{X}^\star = \{\mathbf{x} \mid f_i(\mathbf{x}) \leq 0,\ i = 0, \ldots, m,\ h_i(\mathbf{x}) = 0,\ i = 1, \ldots, q,\ f_0(\mathbf{x}) = p^\star\}. \tag{3.5}$$

3.2 Convex Problems

Definition 3.5 (**convex optimization problem**). *An optimization problem of the form*

$$\begin{array}{ll} \text{minimize} & f_0(\mathbf{x}) \\ \text{subject to} & f_i(\mathbf{x}) \leq 0, \quad i = 1, \ldots, m \\ & \mathbf{a}_i^\top \mathbf{x} = b_i, \quad i = 1, \ldots, q \end{array} \tag{3.6}$$

is said to be convex *if the functions f_i, for $i = 0, \ldots, m$, are convex.*

The feasible set of a convex optimization problem with domain

$$\mathbb{D} = \bigcap_{i=0}^{m} \mathbf{dom} f_i, \tag{3.7}$$

is

$$\mathbb{X} = \left\{ \mathbf{x} \in \mathbb{D} \mid f_i(\mathbf{x}) \leq 0,\ i = 0, \ldots, m,\ \mathbf{a}_i^\top \mathbf{x} = b_i,\ i = 1, \ldots, q \right\}. \tag{3.8}$$

Theorem 3.6 (**convexity of the feasible set** [BV04]). *The feasible set of a convex optimization problem is convex.*

Theorem 3.7 (**global optimality and convexity of minima** [BV04]). *For the convex optimization problem (3.6), the set of points $\mathbf{x} \in \mathbb{X}$ which minimize f_0 is convex. Additionally, every local minimum of f_0 is also a global minimum.*

Theorem 3.8 (**optimality criterion for differentiable objective** [BV04]). *For a convex optimization problem (3.6) whose objective function f_0 is differentiable, such that, for all $\mathbf{x}, \mathbf{y} \in \mathbf{dom} f_0$,*

$$f_0(\mathbf{y}) \geq f_0(\mathbf{x}) + \nabla f_0(\mathbf{y})^\top (\mathbf{y} - \mathbf{x}), \tag{3.9}$$

let the feasible set be

$$\mathbb{X} = \left\{ \mathbf{x} \in \mathbb{D} \mid f_i(\mathbf{x}) \leq 0,\ i = 0, \ldots, m,\ \mathbf{a}_i^\top \mathbf{x} = b_i,\ i = 1, \ldots, q \right\}. \tag{3.10}$$

Then, a point $\mathbf{x} \in \mathbb{X}$ is optimal if, and only if,

$$\nabla f_0(\mathbf{x})^\top (\mathbf{y} - \mathbf{x}) \geq 0,\ \forall \mathbf{y} \in \mathbb{X}. \tag{3.11}$$

3.3 Unconstrained Optimization

Definition 3.9 (**unconstrained convex optimization problem**). *An unconstrained convex optimization problem is of the form*

$$\text{minimize} \quad f(\mathbf{x}), \tag{3.12}$$

where $f \colon \mathbb{R}^n \to \mathbb{R}$ is a (differentiable) convex function.

Theorem 3.10 (**optimality criterion for unconstrained convex problems** [BV04]). *The point $\mathbf{x}^\star \in \mathbb{R}^n$ is an optimal point for the unconstrained convex optimization problem (3.12) if, and only if,*

$$\nabla f(\mathbf{x}^\star) = \mathbf{0}. \tag{3.13}$$

Usually, equation (3.13) is non-linear and does not have a closed form solution. Thus, we must use an iterative process for solving equation (3.13) and, by extension, problem (3.12). Such a procedure is one which harnesses specific properties of the objective function f to generate a sequence of points $\mathbf{x}^t \in \mathbb{R}^n$, such that $\mathbf{x}^t \to \mathbf{x}^\star$, as $t \to \infty$.

3.3.1 Gradient Descent

A general procedure for solving unconstrained convex optimization problems (3.12) is one where the sequence $\mathbf{x}^t$ is produced by the update rule

$$\mathbf{x}^{t+1} = \mathbf{x}^t + \alpha_t \mathbf{d}^t, \tag{3.14}$$

where, $\mathbf{d}^t$ is a vector in $\mathbb{R}^n$ called the *search direction*, and $\alpha_t \in \mathbb{R}$ is called the stepsize.

Definition 3.11 (**descent methods**). *Iterative methods that solve problem (3.12) by producing sequences $\mathbf{x}^t \in \mathbb{R}^n$ for which*

$$f(\mathbf{x}^{t+1}) \leq f(\mathbf{x}^t) \tag{3.15}$$

holds for all t, are called descent methods. *Equality holds only when $\mathbf{x}^t \in \mathbf{X}^\star$.*

Since f is convex and differentiable, we have from Theorem 2.13 that

$$\begin{aligned} & f(\mathbf{x}^{t+1}) \geq f(\mathbf{x}^t) + \nabla f(\mathbf{x}^t)^\top (\mathbf{x}^{t+1} - \mathbf{x}^t) \\ \iff & f(\mathbf{x}^{t+1}) - f(\mathbf{x}^t) \geq \nabla f(\mathbf{x}^t)^\top (\alpha_t \mathbf{d}^t). \end{aligned} \tag{3.16}$$

We can see that if $\nabla f(\mathbf{x}^t)^\top (\alpha_t \mathbf{d}^t) \geq 0$, then $f(\mathbf{x}^{t+1}) \geq f(\mathbf{x}^t)$. Therefore, we must have $\alpha_t \mathbf{d}^t \leq 0$ and, since $\alpha_t > 0$, we need to choose a search direction $\mathbf{d}^t$ which satisfies

$$\nabla f(\mathbf{x}^t)^\top \mathbf{d}^t < 0. \tag{3.17}$$

The most prevalent choice for the search direction is to use the *negative gradient*, that is,

$$\mathbf{d}^t = -\nabla f(\mathbf{x}^t) \tag{3.18}$$

It is easy to see that, with this choice, (3.17) is satisfied, since

$$\nabla f(\mathbf{x}^t)^\top \mathbf{d}^t = -\nabla f(\mathbf{x}^t)^\top \nabla f(\mathbf{x}^t) = -\|\nabla f(\mathbf{x}^t)\|_2^2 < 0, \tag{3.19}$$

if $\nabla f(\mathbf{x}^t) \neq \mathbf{0}$.

We can iteratively apply (3.14) until $\nabla f(\mathbf{x}^t)$ is sufficiently close to $\mathbf{0}$. The above, result in the method described in Algorithm 1.

Algorithm 1 Gradient Descent

initialization: choose $\mathbf{x}^0 \in \mathbf{dom} f$
for $t = 0 \rightarrow$ max_iters **do**
  $\mathbf{x}^{t+1} \leftarrow \mathbf{x}^t - \alpha_t \nabla f(\mathbf{x}^t)$
  if $\|\nabla f(\mathbf{x}^t)\|_2 \leq \epsilon$ **then**
    break
  end if
end for

An important parameter of the gradient descent algorithm is the stepsize α_t at each iteration t. Choosing a stepsize that is too large or too small, can make the algorithm either diverge or converge so slowly that it becomes impractical.

One method for choosing the stepsize is *line-search*, where we practically try to find a stepsize that results in sufficiently high decrease of f when (3.14) is applied.

An obvious method for the computation of the step size is *exact line-search*, that is,

$$\alpha = \underset{s \geq 0}{\operatorname{argmin}} f(\mathbf{x} + s\mathbf{d}). \tag{3.20}$$

In cases where (3.20) is difficult to solve, we turn to *backtracking line-search*, an inexact line-search method, where our aim is to find a stepsize that provides a sufficient decrease to f, as before, when compared to its first-order Taylor approximation.

Algorithm 2 Backtracking Line-Search [BV04]

given: descent direction $\mathbf{d}$ of f at $\mathbf{x} \in \mathbf{dom} f$, $\gamma \in (0, 0.5)$, $\beta \in (0, 1)$
initialization: $\alpha = 1$
while $f(\mathbf{x} + \alpha\mathbf{d}) \geq f(\mathbf{x}) + \gamma\alpha\nabla f(\mathbf{x})^\top \mathbf{d}$ **do**
  $\alpha \leftarrow \beta\alpha$
end while

Both line-search methods mentioned above require additional function and/or gradient computations, increasing the runtime of the algorithm. If, however, the objective function f possesses some desirable properties, a constant stepsize can be used.

Theorem 3.12 (**step size conditions of gradient descent** [Pol87])**.** *Let $f(\mathbf{x})$ be differentiable on $\mathbb{R}^n$, let the gradient of the $f(\mathbf{x})$ satisfy a Lipschitz condition:*

$$\|\nabla f(\mathbf{x}) - \nabla f(\mathbf{y})\| \leq L\|\mathbf{x} - \mathbf{y}\|, \tag{3.21}$$

let $f(\mathbf{x})$ be bounded below:

$$f(\mathbf{x}) \geq f^* > -\infty, \tag{3.22}$$

and let α_t satisfy the condition

$$0 < \alpha_t < 2/L. \tag{3.23}$$

Then, in Algorithm 1*, the* gradient tends to zero

$$\lim_{t \to \infty} \nabla f(\mathbf{x}^t) = \mathbf{0}, \tag{3.24}$$

and the function $f(\mathbf{x})$ monotonically decreases*: $f(\mathbf{x}^{t+1}) \leq f(\mathbf{x}^t)$.*

Now, if f is additionally strongly convex, then the following theorem gives us the convergence rate of Algorithm 1.

Theorem 3.13 (**convergence of gradient descent** [Pol87])**.** *Let f be a differentiable and strongly convex function with constant μ on $\mathbb{R}^n$. Additionally, let its gradient satisfy a Lipschitz condition with constant L. Then, for $0 < \alpha_t < 2/L$,* Algorithm 1 *converges to a unique global minimum point $\mathbf{x}^\star$ with the rate of geometric progression:*

$$\|\mathbf{x}^t - \mathbf{x}^*\| \leq cq^t, \quad 0 \leq q < 1. \tag{3.25}$$

3.3.2 Accelerated Methods

For convex problems that are in addition smooth and/or strongly convex, accelerated versions of Gradient Descent employ various forms of momentum to speed up convergence. Algorithm 3 is

Algorithm 3 Nesterov Accelerated Gradient Descent

assertion: f is L-smooth and μ-strongly convex
initialization: choose $\mathbf{x}^0, \mathbf{y}^0 \in \mathbf{dom} f$
for $t = 0 \rightarrow$ max_iters **do**
  $\mathbf{x}^{t+1} \leftarrow \mathbf{y}^t - \frac{1}{L}\nabla f(\mathbf{y}^t)$
  $\mathbf{y}^{t+1} \leftarrow \mathbf{x}^{t+1} - \frac{\sqrt{\mu}}{\sqrt{L}+\sqrt{\mu}}(\mathbf{x}^{t+1} - \mathbf{x}^t)$
  if $\|\nabla f(\mathbf{x}_k)\|_2 \leq \epsilon$ **then**
    break
  end if
end for

suitable for smooth and strongly convex problems, while Algorithm 4 is suitable only for smooth ones.

Algorithm 4 Nesterov Accelerated Gradient Descent NSC

assertion: f is L-smooth
initialization: choose $\mathbf{x}^0, \mathbf{y}^0 \in \mathbf{dom} f$
for $t = 0 \rightarrow$ max_iters **do**
  $\mathbf{x}^{t+1} \leftarrow \mathbf{y}^t - \frac{1}{L}\nabla f(\mathbf{y}^t)$
  $\beta \leftarrow \alpha_t \cdot \frac{1-\alpha_t}{\alpha_t^2+\alpha_{t+1}}$
  $\mathbf{y}^{t+1} \leftarrow \mathbf{x}^{t+1} + \beta(\mathbf{x}^{t+1} - \mathbf{x}^t)$
  if $\|\nabla f(\mathbf{x}_k)\|_2 \leq \epsilon$ **then**
    break
  end if
end for

The sequence produced by α_t in Algorithm 4 is the solution to

$$\alpha_{t+1}^2 = (1 - \alpha_{t+1})\alpha_t^2 \tag{3.26}$$

and can, at each time step, be computed by

$$\alpha_{t+1} = \frac{-\alpha_t^2 + \sqrt{\alpha_t^4 + 4\alpha_t^2}}{2}, \tag{3.27}$$

with $\alpha_0 = 0.5$.

3.4 Constrained Optimization

Thus far, the method (Algorithm 1) we have presented can be used to solve unconstrained convex optimization problems of the form (3.12).

A simple extension of this method, which allows us to solve constrained convex optimization problems (3.1), is the *projected gradient descent* method. It makes use of the *projection* operator (2.10), which maps a point onto the set of constraints $\mathcal{C}$. The method first performs a standard gradient step, using the previous value of the optimization variable and, then, projects the result onto the constraint set. Note that the magnitude of the gradient cannot be used as a stopping criterion here, since it will not, in general, approach $\mathbf{0}$ inside the feasible set.

Algorithm 5 Projected Gradient Descent

initialization: choose $\mathbf{x}^0 \in \mathcal{C}$
for $t = 0 \rightarrow$ max_iters **do**
  $\mathbf{x}^{t+1} \leftarrow P_{\mathcal{C}}\left(\mathbf{x}^t - \alpha_t \nabla f(\mathbf{x}^t)\right)$
  if $\|\mathbf{x}^{t+1} - \mathbf{x}^t\|_2 \leq \epsilon$ **then**
    break
  end if
end for

3.5 Composite Problems

Composite problems are optimization problems whose objective can be split into two functions, f and g.

$$\underset{\mathbf{x}\in\mathbb{R}^p}{\text{minimize}} \quad F(\mathbf{x}) \coloneqq f(\mathbf{x}) + g(\mathbf{x}). \tag{3.28}$$

Generally, f will be a smooth function, while g will be non-smooth, both being defined on $\mathbb{R}^p$. Some common examples are:

- Unconstrained problems, with $g = 0$.
- Constrained problems, with $g = \delta_{\mathcal{C}}(\mathbf{x})$.
- l_1 regularized problems, with $g = \lambda\|\mathbf{x}\|_1$.

Problems of the form (3.28) can be solved via a generalized form of the Projected Gradient Descent (Algorithm 5). The update step

$$\mathbf{x}^{t+1} = P_{\mathcal{C}}\left(\mathbf{x}^t - \alpha_t \nabla f(\mathbf{x}^t)\right), \tag{3.29}$$

can be expressed as

$$\mathbf{x}^{t+1} = \underset{\mathbf{x}\in\mathcal{C}}{\operatorname{argmin}} \left\{ f(\mathbf{x}^t) + \langle \nabla f(\mathbf{x}^t), \mathbf{x} - \mathbf{x}^t \rangle + \frac{1}{2\alpha_t}\|\mathbf{x} - \mathbf{x}^t\|_2^2 \right\}. \tag{3.30}$$

An interpretation of this update is the point in $\mathcal{C}$ that achieves the minimum value of a linearization of f at $\mathbf{x}^t$ plus a proximal term.

Applying equation (3.30) to problem (3.28) we get

$$\mathbf{x}^{t+1} = \underset{\mathbf{x}\in\mathbb{R}^p}{\operatorname{argmin}} \left\{ f(\mathbf{x}^t) + \langle \nabla f(\mathbf{x}^t), \mathbf{x} - \mathbf{x}^t \rangle + g(\mathbf{x}) + \frac{1}{2\alpha_t}\|\mathbf{x} - \mathbf{x}^t\|_2^2 \right\}, \tag{3.31}$$

which is equal to [Bec17]:

$$\mathbf{x}^{t+1} = \underset{\mathbf{x}\in\mathbb{R}^p}{\operatorname{argmin}} \left\{ \alpha_t g(\mathbf{x}) + \frac{1}{2\alpha_t}\|\mathbf{x} - \left(\mathbf{x}^t - \alpha_t \nabla f(\mathbf{x}^t)\right)\|_2^2 \right\}. \tag{3.32}$$

By the definition of the *proximal mapping* (2.12), equation (3.32) can be rewritten as

$$\mathbf{x}^{t+1} = \operatorname{prox}_{\alpha_t g}\left(\mathbf{x} - \alpha_t \nabla f(\mathbf{x}^t)\right). \tag{3.33}$$

The above leads to an algorithm which at each iteration performs a gradient step of f, followed by a proximal mapping to g.

Algorithm 6 Proximal Gradient Method [Bec17]

initialization: choose $\mathbf{x}^0 \in \mathbf{dom} f$
for $t = 0 \to \max\_\text{iters}$ **do**
  $\mathbf{x}^{t+1} \leftarrow \text{prox}_{\frac{1}{L}g}\left(\mathbf{x}^t - \frac{1}{L}\nabla f(\mathbf{x}^t)\right)$
  if $\|\mathbf{x}^{t+1} - \mathbf{x}^t\|_2 \leq \epsilon$ **then**
    break
  end if
end for

3.6 Machine Learning

Definition 3.14 (**machine learning model**). *The standard supervised* machine learning model *is that of minimizing the numerical average of the losses over all data points:*

$$\underset{\boldsymbol{\theta}\in\mathbb{R}^p}{\text{minimize}} \quad \frac{1}{|\mathcal{D}|}\sum_{k=1}^{|\mathcal{D}|}\ell(\boldsymbol{\theta};\mathbf{x}_k,y_k), \tag{3.34}$$

where the set $\mathcal{D}$ contains pairs $\{(\mathbf{x}_i,y_i)\}_{i=1}^n$ of data features $\mathbf{x}$ and their corresponding labels y, and $\ell\colon \mathbb{R}^p \to \mathbb{R}$ is the loss function.

Given that the loss function ℓ is convex, the objective function in (3.34) is a normalized sum of convex functions, which is a convex function. Therefore, the standard ML model with convex loss function can be seen as an unconstrained optimization problem (3.12).

Therefore, (3.34) can be readily solved via gradient descent (or its accelerated versions), where the gradient at each iteration of Algorithm 1 is the average of the gradients of ℓ for every data pair. Thus, the parameter $\boldsymbol{\theta}$ will be updated by

$$\boldsymbol{\theta}^{t+1} = \boldsymbol{\theta}^t - \frac{\alpha_t}{|\mathcal{D}|}\sum_{k=1}^{|\mathcal{D}|}\nabla\ell(\boldsymbol{\theta}^t;\mathbf{x}_k,y_k). \tag{3.35}$$

Chapter 4

Consensus

The consensus protocol [Deg74] allows a set of agents, communicating via a network, to agree upon the value of a parameter. Each agent has its own estimate for the value of the parameter. Using an iterative process, whereby each agent incorporates the estimates of its neighbors to update its own, a consensus can be reached.

4.1 Definition

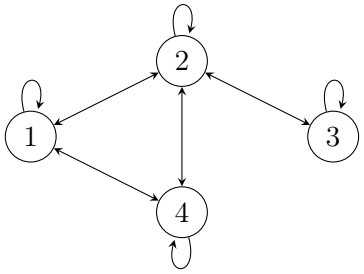
For a network of N agents, let $\mathbf{x}_i \in \mathbb{R}^p$, be the estimate of parameter $\mathbf{x}$, for agent i. The objective is for the agents to agree upon a common value for the parameter $\mathbf{x}$, via communication with their immediate neighbors.

To facilitate this, each agent i assigns a weight, $w_{ij} \geq 0 \ \forall\, j \in [N]$, to the «opinion» of the other agents. Naturally, these weights are nonnegative, zero for non-neighboring agents (since information cannot be exchanged between them), and sum to 1, akin to a probability distribution.

4.2 Graph structure

The network over which the agents communicate can be represented by an undirected graph $G = (\mathcal{V}, \mathcal{E})$. The set of vertices contains a vertex for every one of the agents $\mathcal{V} = [N] \coloneqq \{1, \ldots, N\}$, and the set of edges $\mathcal{E}$ contains all the communication links, with $\{i, j\} \in \mathcal{E}$ denoting that agents/vertices i and j are connected.

We denote the $N \times N$ matrix of weights by $\mathbf{W}$, where w_{ij} is the weight agent i assigns to agent j's estimate. This weight matrix is compatible with the graph G, since the ij-th entry is positive only when an edge connects agents i and j, much like the adjacency matrix (Figure 4.1) associated with the graph. Moreover, it is a *row stochastic* matrix [Law06], since its entries are nonnegative, and its rows sum to 1.



$$\mathbf{A} = \begin{bmatrix} 1 & 1 & 0 & 1 \\ 1 & 1 & 1 & 1 \\ 0 & 1 & 1 & 0 \\ 1 & 1 & 0 & 1 \end{bmatrix}$$

Figure 4.1: Connected graph with 4 vertices and its adjacency matrix.

4.3 Algorithm

At each step of the iterative algorithm, each agent updates its current estimate by simply pooling the estimates of its neighbors. Then, it computes a weighted average of these values, which results in a convex combination [Roc97] of $\mathbf{x}_i$ for $i = 1, \ldots, N$. More formally, agent i revises its estimate as

$$\mathbf{x}_i^{t+1} = \sum_{j \in \mathcal{N}_i \cup \{i\}} w_{ij} \mathbf{x}_j^t. \tag{4.1}$$

Since the weight w_{ij} will be zero if $\{i, j\} \notin \mathcal{E}$, (4.1) can be written as

$$\mathbf{x}_i^{t+1} = \sum_{j=1}^{N} w_{ij} \mathbf{x}_j^t. \tag{4.2}$$

Moreover, if we consider the matrix

$$\mathbf{X} = \begin{bmatrix} \mathbf{x}_1^\top \\ \mathbf{x}_2^\top \\ \vdots \\ \mathbf{x}_N^\top \end{bmatrix} \in \mathbb{R}^{N \times p}, \tag{4.3}$$

whose rows are comprised of each agent's estimate of $\mathbf{x}$, we can write the global update as

$$\mathbf{X}^{t+1} = \mathbf{W}\mathbf{X}^t. \tag{4.4}$$

Even though the agents do not have direct knowledge of the global estimates, that information is gradually diffused throughout the network. This process continues until the estimates of all agents become equal, i.e.,

$$\mathbf{x}_i = \mathbf{x}_j, \, \forall \, \{i, j\} \in [N] \times [N]. \tag{4.5}$$

It is, however, more practical to halt the iterative process when the estimates stop changing significantly.

The above result in the following algorithm.

Algorithm 7 Consensus

assertion: Network of N agents, forming a communication graph $G = \{[N], \mathcal{E}\}$ that is connected, weight matrix $\mathbf{W}$ satisfying (4.8)
initialization: arbitrarily choose $\mathbf{x}_i^0$ for $i = 1, \ldots, N$
for $t = 0 \rightarrow$ max_iters **do**
  for each agent i **do**
    Transmit $\mathbf{x}_i^t$ and receive $\mathbf{x}_j^t$, to and from agents $j \in \mathcal{N}_i$
    $\mathbf{x}_i^{t+1} \leftarrow \sum_{j \in \mathcal{N}_i \cup \{i\}} w_{ij} \mathbf{x}_j^t$
  end for
  if $\|\mathbf{x}_i^t - \mathbf{x}_i^{t-1}\| \leq \epsilon, \; \forall \, i = 1, \ldots, N$ **then**
    break
  end if
end for

4.4 Convergence

We say a consensus is reached if and only if there exists a value $\mathbf{x}^\star$ such that

$$\lim_{t\to\infty} \mathbf{x}_i^t = \mathbf{x}^\star, \quad \forall\, i = 1, \ldots, N. \tag{4.6}$$

Additionally, this implies that $\mathbf{x}^\star = \mathbf{W}\mathbf{x}^\star$, i.e., another round of communication and information exchange between the agents will not alter the estimate. The common estimate reached is given by a convex combination of the initial estimates of each agent

$$\mathbf{x}^\star = \sum_{i=1}^{N} \pi_i \mathbf{x}_i^0, \tag{4.7}$$

where $\boldsymbol{\pi} = (\pi_1, \ldots, \pi_N)^\top$ is the eigenvector of matrix $\mathbf{W}$ corresponding to an eigenvalue of 1, such that $\boldsymbol{\pi}\mathbf{W} = \boldsymbol{\pi}$ is satisfied.

The necessary conditions for convergence [Deg74] are that graph G be connected and that the $N \times N$ weight matrix $\mathbf{W}$ be compatible with the graph and satisfy the following:

$$\begin{aligned} &w_{ij} \geq 0 \quad \text{and} \quad w_{ii} > 0, \\ &\sum_{j=1}^{N} w_{ij} = 1, \\ &w_{ij} = 0 \iff j \notin \mathcal{N}_i \cup \{i\}, \end{aligned} \tag{4.8}$$

with $\mathcal{N}_i$ being the set of neighboring vertices to agent i.

If the weight matrix $\mathbf{W}$ is additionally *doubly stochastic*, that is, both its rows and columns sum to 1, then $\mathbf{x}^\star$ is the average of the agents' starting estimates.

Chapter 5

Distributed Optimization

Distributed optimization allows a set of agents to collectively solve the global minimization problem, defined as the sum of each agent's local objective function. Even though the agents do not all communicate with each other directly, they are able to solve the global problem via the consensus mechanism, while at the same time aligning their local minimization variables.

5.1 Unconstrained Distributed Optimization

Definition 5.1 (**distributed optimization problem).** *The minimization of the average of N objective functions while simultaneously aligning the N minimization variables is a* distributed optimization problem*, defined as*

$$\begin{aligned} \underset{\mathbf{x}_i \in \mathbb{R}^p,\, i \in [N]}{\text{minimize}} \quad & f(\mathbf{X}) \coloneqq \frac{1}{N} \sum_{i=1}^{N} f_i(\mathbf{x}_i) \\ \text{subject to} \quad & \mathbf{x}_1 = \mathbf{x}_2 = \cdots = \mathbf{x}_N. \end{aligned} \tag{5.1}$$

The N agents, which are connected by a communication network represented by a graph $G = \{[N], \mathcal{E}\}$, have their own objective functions $f_i \colon \mathbb{R}^n \to \mathbb{R}$ and minimization variable $\mathbf{x}_i \in \mathbb{R}^n$, for $i = 1, \ldots, N$. The global function f is a function of the matrix $\mathbf{X} \in \mathbb{R}^{N \times p}$ whose rows are comprised of each agent's estimate.

If we consider a dataset $\mathcal{D}$ split up amongst the agents, each having access to a privately owned subset $\mathcal{D}_i \subset \mathcal{D}$, the agents can collectively solve (5.1) to train an ML model of the form (3.34) with information from the whole dataset $\mathcal{D}$ without ever sharing data points directly. The objective function of each agent is the average of the losses of its data points, that is,

$$f_i(\mathbf{x}_i) = \frac{1}{|\mathcal{D}_i|} \sum_{(\mathbf{z}, y) \in \mathcal{D}_i} \ell(\mathbf{x}_i; \mathbf{z}, y). \tag{5.2}$$

One way to solve distributed optimization problems amounts to combining two iterative methods, namely the consensus protocol (Algorithm 7), in order to align the N decision variables $\mathbf{x}_i$, and the gradient descent method (Algorithm 1), for the minimization of each agent's objective function.

For most algorithms, at each time step, each agent pools the values of the optimization variables from its neighbors, and computes a weighted average, just as in the consensus protocol. Using this average it computes the gradient of its own objective function and updates its optimization variable. More advanced algorithms have been introduced that allow for faster and more precise convergence, by using auxiliary variables, gradient tracking, cumulative correction terms, and other techniques.

5.1.1 Assumptions

We state the assumptions for the algorithms that we will study, starting with one that is assumed to hold for all the algorithms to follow.

Assumption 1 (**graph and mixing matrix**). *The underlying communication graph* $G = \{[N], \mathcal{E}\}$ *linking the* N *agents is* undirected, connected*, and* does not change over time. *The mixing matrix* $\mathbf{W}$ *is* symmetric *and* doubly stochastic.

Assumption 2 (**smooth objectives**). *The objective function of agent* i *is* L_i-smooth*, for some* $L_i > 0$.

Assumption 3 (**strongly convex objectives**). *The objective function of agent* i *is* μ_i-strongly convex*, for some* $\mu_i > 0$.

Many algorithms make the assumption that all objective functions f_i share the same Lipschitz and/or strong convexity parameter. These parameters often appear as terms in the stepsize, the convergence rate results, or both. However, it is natural to assume that the different objective functions will have different constants L_i and μ_i, even if only slightly. In these cases, it is sufficient to use the greatest Lipschitz constant as an upper bound for the smoothness of the global function, i.e., $L_h \triangleq \max_{i\in[N]} L_i \geq L_f$. Similarly, we can use the smallest strong convexity parameter, $\mu_h \triangleq \min_{i\in[N]} \mu_i \leq \mu_f$, as a lower bound.

5.1.2 Distributed Gradient Descent

One of the first and, arguably, simpler algorithms is *distributed gradient descent* (DGD), described in Algorithm 8. The algorithm was introduced in [NO09] and, as mentioned before, a weighted average of the neighbors' optimization variables is used to take a single gradient descent step.

Algorithm 8 Distributed Gradient Descent [NO09]

assertion: Assumption 1 holds
initialization: arbitrarily choose $\mathbf{x}_i^0 \in \mathbb{R}^p$ for $i = 1, \ldots, N$
for $t = 0 \to$ max_iters **do**
  for each agent i **do**
    $\mathbf{v}_i^{t+1} \leftarrow \sum_{j=1}^N w_{ij}\mathbf{x}_j^t$ ▷ consensus
    $\mathbf{x}_i^{t+1} \leftarrow \mathbf{v}_i^{t+1} - \alpha_t \nabla f_i(\mathbf{v}_i^{t+1})$ ▷ gradient descent
  end for
end for

Here, the consensus step comes first, followed by the gradient descent step. This scheme is aptly called combine-then-adapt (CTA). Exchanging the order of these operations gives rise to the adapt-then-combine (ATC) scheme. These two schemes, however, function exactly the same in the limit and which of the two should be used is entirely preferential [Ned20].

Algorithm 8 was first studied [NO09] for general convex and non-smooth objectives, making use of subgradients for the gradient descent step. It was later shown that, for a class of stepsizes satisfying the standard conditions

$$\sum_{t=1}^{\infty} \alpha_t = \infty, \quad \sum_{t=1}^{\infty} \alpha_t^2 < \infty, \tag{5.3}$$

the algorithm converges at a rate of $O\left(\frac{\log t}{\sqrt{t}}\right)$ [Che12].

If we assume that the individual objective functions of the agents are smooth, i.e. Assumption 2 holds, a constant stepsize can be used. For

$$\alpha_t = \min\left\{\frac{1+\lambda_N(\mathbf{W})}{L_h}, \frac{1}{L_f}\right\} = O\left(\frac{1}{L_h}\right), \tag{5.4}$$

the algorithm converges at a rate of $O\left(\frac{1}{\alpha t}\right) = O\left(\frac{1}{t}\right)$, until reaching an $O(\frac{\alpha}{1-\beta})$-neighborhood of the solution, where $\beta = \lambda_2(\mathbf{W})$ is the second-largest eigenvalue modulus of the mixing matrix [YLY16].

If, in addition, each f_i is strongly convex, i.e. Assumption 3 holds, then with stepsize

$$\alpha_t = \min\left\{\frac{1+\lambda_N(\mathbf{W})}{L_h}, \frac{1}{L_f+\mu_f}\right\} = O\left(\frac{1}{L_h}\right), \tag{5.5}$$

we have geometric convergence $O\left(c^t\right)$ until, once again, reaching an $O(\frac{\alpha}{1-\beta})$-neighborhood of the solution [YLY16]. The constant c is defined as

$$c = \sqrt{1-\frac{\alpha c_0}{2}} \in (0,1), \tag{5.6}$$

where

$$c_0 = \frac{\mu_f L_f}{\mu_f + L_f}. \tag{5.7}$$

It is, thus, evident that there is a tradeoff between convergence speed and optimality, since both the convergence rate and the neighborhood size are proportional to the constant stepsize.

A simple way to see why neither exact convergence nor consensus is achieved is to consider what the gradient update will be as t goes to infinity [Shi+15b]. Using matrix notation, much akin to (4.4), if we assume that the sequence $\mathbf{X}^t$ is convergent to $\mathbf{X}^\infty$, then

$$\mathbf{X}^\infty = \mathbf{W}\mathbf{X}^\infty - \alpha\nabla f(\mathbf{X}^\infty). \tag{5.8}$$

If we assume consensus at infinity, we have $\mathbf{X}^\infty = \mathbf{W}\mathbf{X}^\infty$. Then, in order for (5.8) to hold, we need $\nabla f(\mathbf{X}^\infty)$ to be $\mathbf{0}$, i.e., the same optimization variable $\mathbf{x}^\infty$ to simultaneously minimize each agent's objective function f_i. This does not hold, in general, preventing $\mathbf{X}^t$ from being consensual at the limit, since we cannot drive the gradient to 0 for all agents at once. However, alternative methods, although more complex, alleviate this shortcoming.

5.1.3 Exact First-Order Algorithm

The first algorithm to provide exact convergence with a constant stepsize, in linear time, is EXTRA [Shi+15b], described in Algorithm 9. It does so while using only first-order information and requiring a single round of communication and local gradient computation, per iteration.

Note that, for simplicity, matrix notation is used for the update rule. The gradient of the global function f at $\mathbf{X}$ is defined, with some abuse of notation, as

$$\nabla f(\mathbf{X}) = \begin{bmatrix} \nabla f_1(\mathbf{x}_1)^\top \\ \vdots \\ \nabla f_N(\mathbf{x}_N)^\top \end{bmatrix}, \tag{5.9}$$

Algorithm 9 EXTRA [Shi+15b]

assertion: Assumptions 1 and 2 hold, $\tilde{\mathbf{W}} = \frac{\mathbf{W}+\mathbf{I}}{2}$
initialization: arbitrarily choose $\mathbf{x}_i^0 \in \mathbb{R}^p$ for $i = 1, \dots, N$
$\mathbf{X}^1 \leftarrow \mathbf{W}\mathbf{X}^0 - \alpha \nabla f(\mathbf{X}^0)$
for $t = 0 \rightarrow$ max_iters **do**
  $\mathbf{X}^{t+2} \leftarrow (\mathbf{I} + \mathbf{W})\,\mathbf{X}^{t+1} - \tilde{\mathbf{W}}\mathbf{X}^t - \alpha\left(\nabla f(\mathbf{X}^{t+1}) - \nabla f(\mathbf{X}^t)\right)$
end for

i.e., the matrix whose rows are the gradients of each agent's function f_i at $\mathbf{x}_i$.

The algorithm, however, remains fully decentralized. This can be easily seen if one considers the rows of matrix $\mathbf{X}^t$, where at each update step only local information is required for agent i to perform the following update:

$$\mathbf{x}_i^{t+2} = \mathbf{x}_i^t + \sum_{j=0}^{N} w_{ij}\mathbf{x}_j^{t+1} - \sum_{j}^{N} \tilde{w}_{ij}\mathbf{x}_j^t - \alpha\left(\nabla f_i(\mathbf{x}_i^{t+1}) - \nabla f_i(\mathbf{x}^t)\right). \tag{5.10}$$

The global update step can be derived by considering two DGD updates one time step apart

$$\mathbf{X}^{t+2} = \mathbf{W}\mathbf{X}^{t+1} - \alpha\nabla f(\mathbf{X}^{t+1}), \tag{5.11}$$
$$\mathbf{X}^{t+1} = \tilde{\mathbf{W}}\mathbf{X}^t - \alpha\nabla f(\mathbf{X}^t), \tag{5.12}$$

where

$$\tilde{\mathbf{W}} = \frac{\mathbf{W}+\mathbf{I}}{2}, \tag{5.13}$$

and taking their difference:

$$\mathbf{X}^{t+2} - \mathbf{X}^{t+1} = \mathbf{W}\mathbf{X}^{t+1} - \tilde{\mathbf{W}}\mathbf{X}^t - \alpha\left(\nabla f(\mathbf{X}^{t+1}) - \nabla f(\mathbf{X}^t)\right). \tag{5.14}$$

The algorithm needs to satisfy two conditions in order to solve problem (5.1). If there exists a limit point of the generated sequence of update (5.14), that is,

$$\mathbf{X}^\star = \lim_{t\to\infty} \mathbf{X}^t, \tag{5.15}$$

then $\mathbf{X}^\star$ should

1. be in *consensus*, i.e., $\mathbf{X}^\star = \mathbf{W}\mathbf{X}^\star$
2. be an *optimal point* such that $\mathbf{1}^\top \nabla f(\mathbf{X}^\star) = 0$.

Letting $t \to \infty$ we have:

$$\begin{aligned} &\mathbf{X}^\star - \mathbf{X}^\star = \mathbf{W}\mathbf{X}^\star - \tilde{\mathbf{W}}\mathbf{X}^\star - \alpha\left(\nabla f(\mathbf{X}^\star) - \nabla f(\mathbf{X}^\star)\right) \\ \implies &(\mathbf{W} - \tilde{\mathbf{W}})\mathbf{X}^\star = 0 \implies \left(\frac{\mathbf{W}-\mathbf{I}}{2}\right)\mathbf{X}^\star = 0 \\ \implies &\mathbf{W}\mathbf{X}^\star = \mathbf{X}^\star. \end{aligned}$$

Thus, the first condition is satisfied.

For the optimality condition, we sum up k updates:

$$\begin{aligned}
&\sum_{t=0}^{k} \mathbf{X}^{t+2} - \mathbf{X}^{t+1} = \sum_{t=0}^{k} \mathbf{W}\mathbf{X}^{t+1} - \sum_{t=0}^{k} \tilde{\mathbf{W}}\mathbf{X}^{t} - \alpha \sum_{t=0}^{k} \left(\nabla f(\mathbf{X}^{t+1}) - \nabla f(\mathbf{X}^{t})\right) \\
&\implies \mathbf{X}^{k+2} - \mathbf{X}^{1} = \sum_{t=0}^{k} \mathbf{W}\mathbf{X}^{t+1} - \sum_{t=0}^{k} \tilde{\mathbf{W}}\mathbf{X}^{t} - \alpha \left(\nabla f(\mathbf{X}^{k+1}) - \nabla f(\mathbf{X}^{0})\right) \\
&\implies \mathbf{X}^{k+2} = \sum_{t=1}^{k} (\mathbf{W} - \tilde{\mathbf{W}})\mathbf{X}^{t} + \mathbf{W}\mathbf{X}^{k+1} - \tilde{\mathbf{W}}\mathbf{X}^{0} + \mathbf{W}\mathbf{X}^{0} - \alpha \nabla f(\mathbf{X}^{k+1}) \\
&\implies \mathbf{X}^{k+2} = \sum_{t=0}^{k} (\mathbf{W} - \tilde{\mathbf{W}})\mathbf{X}^{t} + \mathbf{W}\mathbf{X}^{k+1} - \alpha \nabla f(\mathbf{X}^{k+1}),
\end{aligned}$$

where we applied telescopic summation and substituted $\mathbf{X}^1$ with $\mathbf{W}\mathbf{X}^0 - \alpha\nabla f(\mathbf{X}^0)$. Taking $k \to \infty$, we get:

$$\begin{aligned}
\mathbf{X}^{\star} &= \sum_{t=0}^{\infty} (\mathbf{W} - \tilde{\mathbf{W}})\mathbf{X}^{t+1} + \mathbf{W}\mathbf{X}^{\star} - \alpha \nabla f(\mathbf{X}^{\star}) \\
&\implies \sum_{t=0}^{\infty} (\mathbf{W} - \tilde{\mathbf{W}})\mathbf{X}^{t} = \alpha \nabla f(\mathbf{X}^{\star}).
\end{aligned}$$

Left multiplying by $\mathbf{1}^\top$, we have:

$$\sum_{t=0}^{\infty} \mathbf{1}^\top (\mathbf{W} - \tilde{\mathbf{W}})\mathbf{X}^{t} = \alpha \mathbf{1}^\top \nabla f(\mathbf{X}^{\star}).$$

Thus, $\mathbf{1}^\top \nabla f(\mathbf{X}^\star) = 0$ is satisfied if $\mathbf{1}^\top(\mathbf{W} - \tilde{\mathbf{W}}) = 0$, which holds since $\mathbf{W}$, and by extension $\tilde{\mathbf{W}}$, are doubly stochastic.

As was shown, equation (5.14) can be rewritten as

$$\mathbf{X}^{t+2} = \underbrace{\mathbf{W}\mathbf{X}^{t+1} - \alpha \nabla f(\mathbf{X}^{t+1})}_{\text{DGD update}} + \underbrace{\sum_{t=0}^{t} (\mathbf{W} - \tilde{\mathbf{W}})\mathbf{X}^{t}}_{\text{correction term}}. \tag{5.16}$$

Thus, the update of EXTRA is simply the update of DGD with a cumulative correction term. This correction term is what alleviates the problem that pure DGD had, preventing it from asymptotically reaching consensus.

The optimal step size for smooth objective functions is

$$\alpha = \frac{1}{L_h}, \tag{5.17}$$

which, importantly, does not depend on the properties of the network or the mixing matrix. Albeit using a constant step size, exact convergence is achieved at a rate of $O(1/t)$ [Shi+15b].

For strongly convex objective functions, i.e. Assumption 3 holds, when using a stepsize that satisfies

$$\alpha < \frac{2\mu_f \lambda_{\min}(\tilde{\mathbf{W}})}{L_f^2}, \tag{5.18}$$

the algorithm converges with rate $O((1+\delta)^{-t})$ for some $\delta > 0$ [Shi+15b].

5.1.4 Accelerated Dual Algorithm

The distributed optimization problem (5.1) can be rewritten as

$$\begin{aligned} \underset{\mathbf{X}\in\mathbb{R}^{N\times p}}{\text{minimize}} \quad & f(\mathbf{X}) \coloneqq \frac{1}{N}\sum_{i=1}^{N} f_i(\mathbf{x}_i) \\ \text{subject to} \quad & \sqrt{\mathbf{W}}\mathbf{X} = \mathbf{0}, \end{aligned} \tag{5.19}$$

where only the consensus constraint differs. More specifically, we take $\mathbf{W}$ to be a mixing matrix which satisfies $\ker(\mathbf{W}) = \operatorname{span}(\mathbf{1})$, with its square root is defined as $\sqrt{\mathbf{W}} = \mathbf{V}^\top \mathbf{\Sigma}^{1/2}\mathbf{V}$, where $\mathbf{W} = \mathbf{V}^\top\mathbf{\Sigma}\mathbf{V}$ is the singular value decomposition of $\mathbf{W}$. Consensus is achieved when this matrix and the estimates $\mathbf{X}$ satisfy $\sqrt{\mathbf{W}}\mathbf{X} = \mathbf{0}$. Then, the columns of $\mathbf{X}$ will be constant, where column j contains the j-th element of each agent's estimate.

This convex problem is equivalent to its *dual* formulation:

$$\underset{\mathbf{\Lambda}\in\mathbb{R}^{N\times p}}{\text{maximize}} \quad -f^*(\sqrt{\mathbf{W}}\mathbf{\Lambda}), \tag{5.20}$$

where f^* is the Fenchel conjugate of f.

To see why these problems are equivalent we begin by splitting the objective and constraints to separate functions of different variables we have

$$\begin{aligned} \underset{\mathbf{X},\mathbf{Z}}{\text{minimize}} \quad & f(\mathbf{X}) + g(\mathbf{Z}) \\ \text{subject to} \quad & \sqrt{\mathbf{W}}\mathbf{X} - \mathbf{Z} = \mathbf{0}, \end{aligned} \tag{5.21}$$

where g is the indicator function of $\mathbf{0}$, $\delta_{\mathbf{0}}(\cdot)$.

Consider the Largrangian of the above problem, with dual variable $\mathbf{\Lambda}$,

$$\begin{aligned} L(\mathbf{X},\mathbf{Z};\mathbf{\Lambda}) &= f(\mathbf{X}) + g(\mathbf{Z}) - \langle \mathbf{\Lambda}, \sqrt{\mathbf{W}}\mathbf{X} - \mathbf{Z}\rangle \\ &= f(\mathbf{X}) + g(\mathbf{Z}) - \langle \sqrt{\mathbf{W}}^\top\mathbf{\Lambda}, \mathbf{X}\rangle + \langle \mathbf{\Lambda}, \mathbf{Z}\rangle. \end{aligned} \tag{5.22}$$

To obtain the dual function, we minimize the Lagrangian w.r.t. $\mathbf{X}$ and $\mathbf{Z}$:

$$\begin{aligned} \min_{\mathbf{X},\mathbf{Z}} L(\mathbf{X},\mathbf{Z};\mathbf{\Lambda}) &= \min_{\mathbf{X}}\{f(\mathbf{X}) - \langle \sqrt{\mathbf{W}}^\top\mathbf{\Lambda}, \mathbf{X}\rangle\} + \min_{\mathbf{Z}}\{g(\mathbf{Z}) - \langle -\mathbf{\Lambda}, \mathbf{Z}\rangle\} \\ &= f^*(\sqrt{\mathbf{W}}^\top\mathbf{\Lambda}) + g^*(-\mathbf{\Lambda}). \end{aligned} \tag{5.23}$$

Thus, the dual problem is the maximization of (5.23) w.r.t. $\mathbf{\Lambda}$:

$$\underset{\mathbf{\Lambda}}{\text{maximize}} \quad -f^*(\sqrt{\mathbf{W}}^\top\mathbf{\Lambda}) - g^*(-\mathbf{\Lambda}). \tag{5.24}$$

Since g is the indicator function of $\mathbf{0}$, and the conjugate of the indicator function is its support function [Bec17], we have for the support function of $\mathbf{0}$ that:

$$\sigma_{\mathbf{0}}(\mathbf{x}) = \max_{\mathbf{y}=\mathbf{0}} \langle \mathbf{x}, \mathbf{y}\rangle = 0, \tag{5.25}$$

thus $g^* = 0$, and (5.24) is equal to (5.20).

Dual problems can be solved via dual ascent. As such, the update step for problem (5.20) is

$$\mathbf{\Lambda}^{t+1} = \mathbf{\Lambda}^t - \alpha\sqrt{\mathbf{W}} \cdot \nabla f^*(\sqrt{\mathbf{W}}\mathbf{\Lambda}^t). \tag{5.26}$$

Introducing the change of variable $\mathbf{Y}^t = \sqrt{\mathbf{W}}\mathbf{\Lambda}^t$ and multiplying both sides of (5.26) by $\sqrt{\mathbf{W}}$, the update can be written as

$$\mathbf{Y}^{t+1} = \mathbf{Y}^t - \alpha \mathbf{W} \cdot \nabla f^*(\mathbf{Y}^t). \tag{5.27}$$

One interpretation of equation (5.27) is that the agents gossip the gradients of their local conjugate functions $\nabla f_i^*(\mathbf{y}_i^t)$. Combining this update with Nesterov acceleration, gives rise to the single-step Dual Accelerated (SSDA) method proposed in [Sca+17].

Algorithm 10 Single-Step Dual Accelerated (SSDA) [Sca+17]

assertion: Assumptions 2 and 3 hold, $\mathbf{W}$ symmetric, positive semi-definite, and $\ker(\mathbf{W}) = \text{span}(\mathbf{1})$
initialization: set $\mathbf{x}_i^0 = \mathbf{0}, \mathbf{y}_i^0 = \mathbf{0}$ for $i = 1, \ldots, N$
for $t = 0 \to \text{max\_iters}$ **do**
  $\mathbf{x}_i^t \leftarrow \nabla f_i^\star(\mathbf{y}_i^t)$, for $i = 1, \ldots, N$
  $\mathbf{\Theta}^{t+1} \leftarrow \mathbf{Y}^t - \alpha \mathbf{W}\mathbf{X}^t$
  $\mathbf{Y}^{t+1} \leftarrow (1+\beta)\mathbf{\Theta}^{t+1} - \beta\mathbf{\Theta}^t$
end for

Algorithm 10 uses stepsize

$$\alpha = \frac{\max_i \mu_i}{\lambda_1(\mathbf{W})} \tag{5.28}$$

and momentum

$$\beta = \frac{\sqrt{\kappa_l} - \sqrt{\gamma}}{\sqrt{\kappa_l} + \sqrt{\gamma}}, \tag{5.29}$$

where the constants are described in Theorem 5.2 below.

The mixing matrix $\mathbf{W}$ is required to be positive semi-definite, symmetric and have $\ker(\mathbf{W}) = \text{span}(\mathbf{1})$. An easy choice is the Laplacian matrix, $\mathbf{L} = \mathbf{D} - \mathbf{A}$, where $\mathbf{D}$ is the degree matrix and $\mathbf{A}$ the adjacency matrix of graph G, that is,

$$l_{ij} = \begin{cases} \deg(v_i), & \text{if } i = j, \\ -1, & \text{if } i \neq j \text{ and } (i,j) \in \mathcal{E}, \\ 0, & \text{otherwise.} \end{cases} \tag{5.30}$$

It is easy to see that the rows (and columns) of $\mathbf{L}$ sum to 0, and that the matrix is symmetric. It is known that the Laplacian matrix can be rewritten as $\mathbf{L} = \mathbf{B}^\top\mathbf{B}$, where $\mathbf{B}$ is the oriented incidence matrix whose elements are:

$$b_{ij} = \begin{cases} -1, & \text{if edge } e_j \text{ leaves vertex } v_i \\ 1, & \text{if edge } e_j \text{ enters vertex } v_i \\ 0, & \text{otherwise.} \end{cases} \tag{5.31}$$

Thus, we have that $\mathbf{L}$ is positive semi-definite, since $\mathbf{x}^\top\mathbf{L}\mathbf{x} = \mathbf{x}^\top\mathbf{B}^\top\mathbf{B}\mathbf{x} = (\mathbf{B}\mathbf{x})^\top(\mathbf{B}\mathbf{x}) = \|\mathbf{B}\mathbf{x}\|_2^2 \geq 0$.

Moreover, for any vector $\mathbf{v} = c \cdot \mathbf{1}$, we have that the i-th element of $\mathbf{L}\mathbf{v}$ is equal to

$$\sum_{j=1}^N l_{ij} v_j = c \cdot \sum_{j=1}^N l_{ij} = 0, \tag{5.32}$$

satisfying the null-space condition.

The gradient of the conjugate function is computed as [Bec17]

$$\nabla f^*(\mathbf{y}) = \underset{\mathbf{x}}{\operatorname{argmax}} \{\langle \mathbf{y}, \mathbf{x} \rangle - f(\mathbf{x})\}. \tag{5.33}$$

Therefore, each of the update steps for each agent in the SSDA algorithm is itself a maximization problem. Given the demanding update step, this algorithm is suitable in settings where the communication is slower, such that there is enough time between sharing the estimates and calculating them.

In some simple but important cases, like, for example, Linear Regression via Least Squares, the solution of the maximization problem in (5.33) can be computed in closed form, significantly reducing the computational cost at each iteration of the algorithm.

In spite of the high computational load of the SSDA algorithm, it enjoys an almost optimal convergence rate compared to the lower bound for decentralized algorithms, provided in [Sca+17].

Theorem 5.2 (**convergence lower bound for decentralized methods).** *For the minimization of the sum of a set of strongly convex and smooth functions f_i, $i = 1, \ldots, N$, and for any decentralized black-box procedure using mixing matrix $\mathbf{W}$, the time needed to reach precision $\epsilon > 0$ is lower bounded by*

$$\Omega\left(\sqrt{\kappa_l}\left(1 + \frac{\tau}{\sqrt{\gamma}}\right)\ln\left(\frac{1}{\epsilon}\right)\right). \tag{5.34}$$

The mixing matrix has eigengap γ, the relative communication time is τ, and the local condition number is defined as $\kappa_l = \frac{\max_i L_i}{\min_i \mu_i}$, where L_i and μ_i are the smoothness and strong convexity constants of f_i.

This theorem states that any decentralized optimization algorithm will require at least $\sqrt{\kappa_l}\ln(\frac{1}{\epsilon})$ gradient computations, and $\frac{\sqrt{\kappa_l}}{\sqrt{\gamma}}\ln\left(\frac{1}{\epsilon}\right)$ communications each taking time τ. Indeed, the number of gradient computations required are the same as any optimal centralized algorithm. The communication rounds scale inversely with the eigengap of the mixing matrix, which on simple networks is known to be closely related to the diameter of the network ($\Delta \approxeq \frac{1}{\sqrt{\gamma}}$) [Sca+17].

The time needed for Algorithm 10 to achieve precision ϵ is

$$O\left((1+\tau)\sqrt{\frac{\kappa_l}{\gamma}}\ln\left(\frac{1}{\epsilon}\right)\right), \tag{5.35}$$

where the constants are as in Theorem 5.2.

An algorithm that provides performance closer to the optimal bound of Theorem 5.2 is the Multi-Step Dual Accelerated method (Algorithm 11). It is virtually the same as SSDA, however, multiple consensus communication rounds are performed for every local computation of the gradient. Thus, it is only applicable when the gradient computation takes much more time than one communication round ($\tau \ll 1$).

Chebyshev acceleration is applied to iteratively produce an alternative mixing matrix $P_K(\mathbf{W})$ in K steps, that has a larger eigengap and is thus faster to reach consensus. Here, $P_K(\mathbf{W})$ is itself a mixing matrix, a polynomial of degree at most K, produced via the iterative procedure:

$$P_K(x) = 1 - \frac{T_K(c_2(1-x))}{T_K(c_2)}. \tag{5.36}$$

The Chebyshev polynomials with $T_0(x) = 1$ and $T_1(x) = x$, are defined as

$$T_{k+1}(x) = 2xT_k(x) - T_{k-1}(x). \tag{5.37}$$

Algorithm 11 Multi-Step Dual Accelerated (MSDA) [Sca+17]

assertion: Assumptions 2 and 3 hold, $\mathbf{W}$ symmetric, positive semi-definite, and $\ker(\mathbf{W}) = \text{span}(\mathbf{1})$
initialization: set $\mathbf{x}_i^0 = \mathbf{0}, \mathbf{y}_i^0 = \mathbf{0}$ for $i = 1, \ldots, N$
for $t = 0 \to \text{max\_iters}$ **do**
  $\mathbf{x}_i^t \leftarrow \nabla f_i^\star(\mathbf{y}_i^t)$, for $i = 1, \ldots, N$
  $\mathbf{\Theta}^{t+1} \leftarrow \mathbf{Y}^t - \alpha \cdot \textsc{AcceleratedGossip}(\mathbf{X}, \mathbf{W}, K)$
  $\mathbf{Y}^{t+1} \leftarrow (1+\beta)\mathbf{\Theta}^{t+1} - \beta\mathbf{\Theta}^t$
end for
procedure $\textsc{AcceleratedGossip}(\mathbf{X}, \mathbf{W}, K)$
  $a_0 \leftarrow 1$, $a_1 \leftarrow c_2$
  $\mathbf{X}^0 \leftarrow \mathbf{X}$, $\mathbf{X}^1 \leftarrow c_2(\mathbf{I} - c_3\mathbf{W})\mathbf{X}^0$
  for $k = 1 \to K$ **do**
    $a_{k+1} \leftarrow 2c_2 a_k - a_{k-1}$
    $\mathbf{X}^{k+1} \leftarrow 2c_2(\mathbf{I} - c_3\mathbf{W})\mathbf{X}^k - \mathbf{X}^{k-1}$
  end for
  return $\mathbf{X}^0 - \mathbf{X}^k / a_K$
end procedure

The values of the common parameters of MSDA differ from SSDA, the latter requiring some additional parameters for the accelerated gossip sub-routine.

$$\begin{aligned} \alpha &= \frac{(\max_i \mu_i)(1 + 2c_1^{2K})}{(1 + 2c_1^K)^2}, & \beta &= \frac{(1 + c_1^K)\sqrt{\kappa} - 1 + c_1^K}{(1 + c_1^K)\sqrt{\kappa} + 1 - c_1^K}, & c_1 &= \frac{1 - \sqrt{\gamma}}{1 + \sqrt{\gamma}}, \\ c_2 &= \frac{1 - \gamma}{1 + \gamma}, & c_3 &= \frac{2}{(1 + \gamma)\lambda_1(\mathbf{W})}, & K &= \left\lfloor \frac{1}{\sqrt{\gamma}} \right\rfloor. \end{aligned} \tag{5.38}$$

The time needed for Algorithm 11 to achieve precision ϵ is

$$O\left(\sqrt{\kappa_l}\left(1 + \frac{\tau}{\sqrt{\gamma}}\right)\ln\left(\frac{1}{\epsilon}\right)\right). \tag{5.39}$$

5.2 Distributed Constrained Optimization

For problems of the form (5.1) that additionally have some constraints on the optimization variables, we turn to constrained optimization. There are two kinds of distributed constrained optimization problems, those with *local*, and those with *global constraints*.

For the case of *local* constraints, each agent i requires that $\mathbf{x}_i \in \mathcal{C}_i$, where $\mathcal{C}_i$ is a closed and convex set of constraints. After convergence, all local variables should, therefore, satisfy $\mathbf{x}_i \in \bigcap_{i=1}^N \mathcal{C}_i$. With global constraints, however, each agent shares a common set of constraints $\mathcal{C}$, such that $\mathcal{C}_i \coloneqq \mathcal{C},\ \forall i = 1, \ldots, N$.

Definition 5.3 (distributed constrained optimization problem)**.** *The minimization of the sum of N objective functions while simultaneously aligning the N minimization variables to be equal and belong to the intersection of the constraints, is a* distributed constrained optimization problem*,*

defined as

$$\begin{array}{ll} \underset{\mathbf{x}_i \in \mathbb{R}^p,\, i \in [N]}{\text{minimize}} & f(\mathbf{X}) \coloneqq \dfrac{1}{N} \displaystyle\sum_{i=1}^{N} f_i(\mathbf{x}_i) \\ \text{subject to} & \mathbf{x}_1 = \mathbf{x}_2 = \cdots = \mathbf{x}_N, \\ & \mathbf{x}_i \in \bigcap_{i=1}^{N} \mathcal{C}_i, \ \forall i = 1, \ldots, N. \end{array} \tag{5.40}$$

The N agents, connected by a communication network represented by a graph $G = \{[N], \mathcal{E}\}$, each have their own objective function $f_i \colon \mathbb{R}^n \to \mathbb{R}$, constraint set $\mathcal{C}_i$, and minimization variable $\mathbf{x}_i \in \mathbb{R}^n$.

5.2.1 Distributed Projected Gradient Descent

Adapting the *distributed gradient descent* method (Algorithm 8) to facilitate constrained optimization looks very similar to the centralized Projected Gradient Descent methods (Algorithm 5). After the gradient descent step, each agent i projects its optimization variable $\mathbf{x}_i$ onto its constraint set $\mathcal{C}_i$, via the orthogonal projection mapping (2.10).

Algorithm 12 Distributed Projected Gradient Descent

assertion: Assumption 1 holds, $\bigcap_{i=1}^{N} \mathcal{C}_i$ is non-empty, closed, and convex
initialization: arbitrarily choose $\mathbf{x}_i^0 \in \mathbb{R}^p$ for $i = 1, \ldots, N$
for $t = 0 \to$ max_iters **do**
  for each agent i **do**
    $\mathbf{v}_i^{t+1} \leftarrow \sum_{j \leftarrow 1}^{N} w_{ij} \mathbf{x}_j^t$ ▷ consensus
    $\mathbf{x}_i^{t+1} \leftarrow P_{C_i}\left(\mathbf{v}_i^{t+1} - \alpha_t \nabla f_i(\mathbf{v}_i^{t+1})\right)$ ▷ gradient descent & projection
  end for
end for

5.2.2 PG-EXTRA

An exact algorithm for constrained optimization is PG-EXTRA, which is an adaptation of Algorithm 9 that can handle composite problems (3.28). The algorithm first performs the same update as regular EXTRA, involving the gradient and mixing step, followed by a proximal term to the non-smooth function g.

Algorithm 13 PG-EXTRA [Shi+15a]

assertion: Assumptions 1 and 2 hold, $\tilde{\mathbf{W}} = \frac{\mathbf{W}+\mathbf{I}}{2}$
initialization: arbitrarily choose $\mathbf{x}_i^0 \in \mathbb{R}^p$ for $i = 1, \ldots, N$
for each agent i **do**
  $\mathbf{z}_i^1 \leftarrow \sum_{j=1}^{N} w_{ij} \mathbf{x}_j^0 - \alpha \nabla f_i(\mathbf{x}_i^0)$
  $\mathbf{x}_i^1 \leftarrow \operatorname{prox}_{g_i}(\mathbf{z}_i^1)$
end for
for $t = 0 \to$ max_iters **do**
  for each agent i **do**
    $\mathbf{z}_i^{t+2} \leftarrow \sum_{j=1}^{N} w_{ij} \mathbf{x}_j^{t+1} + \mathbf{z}_i^t - \sum_{j=1}^{N} \tilde{w}_{ij} \mathbf{x}_j^t + \alpha \left(\nabla f_i(\mathbf{x}_i^{t+1}) - \nabla f_i(\mathbf{x}_i^t)\right)$
    $\mathbf{x}_i^{t+2} \leftarrow \operatorname{prox}_{g_i}(\mathbf{z}_i^{t+2})$
  end for
end for

The proximal mapping of an *indicator function* of a set, is equal to the orthogonal projection onto that set:

$$\text{prox}_{\delta_{\mathcal{C}}}(\mathbf{x}) = \underset{\mathbf{y}}{\text{argmin}} \left\{ \delta_{\mathcal{C}}(\mathbf{y}) + \frac{1}{2}\|\mathbf{y}-\mathbf{x}\|_2^2 \right\} = \underset{\mathbf{y}\in\mathcal{C}}{\text{argmin}} \|\mathbf{y}-\mathbf{x}\|_2^2 = P_{\mathcal{C}}(\mathbf{x}). \tag{5.41}$$

This allows the use of PG-EXTRA for distributed constrained optimization (5.40), by simply letting g_i be $\delta_{\mathcal{C}_i}$, for each agent i.

Chapter 6

Distributed Optimization Problems

This chapter defines the optimization problems we will solve in a distributed fashion in the experiments that follow.

convexity	smoothness	problem
C	×	LS fat + l_1
C	✓	LS fat, LR
SC	×	LS tall + l_1, SVM
SC	✓	LS tall (+ l_2), LS fat + l_2, LR + l_2

Table 6.1: Distributed optimization problems of different function classes.

6.1 Linear Regression via Least Squares

A set of N agents, each one having access to its own regressors $\mathbf{A}_i$ and target variables $\mathbf{b}_i$, aim to estimate the value of the parameter vector $\mathbf{x}$ which best fits the following system of linear equations

$$\mathbf{b}_i = \mathbf{A}_i\mathbf{x} + \boldsymbol{\epsilon}, \tag{6.1}$$

where $\boldsymbol{\epsilon}$ is the measurements noise.

This *Linear Regression* problem can be solved via ordinary *Least Squares*. If the noise is normally distributed, then the Maximum Likelihood estimate of $\mathbf{x}$ is computed via the ordinary Least Squares.

We can express the above distributed problem in the form of (5.1) as

$$\begin{aligned} \text{minimize} \quad & \frac{1}{N}\sum_{i=1}^{N} f_i(\mathbf{x}_i) = \frac{1}{N}\sum_{i=1}^{N} \frac{1}{2}\|\mathbf{A}_i\mathbf{x}_i - \mathbf{b}_i\|_2^2 \\ \text{subject to} \quad & \mathbf{x}_1 = \mathbf{x}_2 = \cdots = \mathbf{x}_N, \end{aligned} \tag{6.2}$$

where $\mathbf{x}_i \in \mathbb{R}^p$, $\mathbf{A}_i \in \mathbb{R}^{n_i \times p}$, $\mathbf{b}_i \in \mathbb{R}^{n_i}$.

Each subproblem, itself a Linear Regression problem and defined by the agent's private objective function f_i, has some important properties:

- (**quadratic form**) The objective function can be re-written in the standard quadratic form,

$$q_i(\mathbf{x}) = \frac{1}{2}\mathbf{x}^\top\mathbf{P}_i\mathbf{x} + \mathbf{v}_i^\top\mathbf{x} + c_i, \tag{6.3}$$

as:

$$\begin{aligned}\frac{1}{2}\|\mathbf{A}_i\mathbf{x}-\mathbf{b}_i\|_2^2 &= \frac{1}{2}\left(\mathbf{A}_i\mathbf{x}-\mathbf{b}_i\right)^\top\left(\mathbf{A}_i\mathbf{x}-\mathbf{b}_i\right)\\ &= \frac{1}{2}\mathbf{x}^\top\underbrace{\mathbf{A}_i^\top\mathbf{A}_i}_{\mathbf{P}_i\in\mathbb{S}^p}\mathbf{x}\ \underbrace{-\mathbf{b}_i^\top\mathbf{A}_i}_{\mathbf{v}_i^\top\in\mathbb{R}^p}\mathbf{x}+\underbrace{\frac{1}{2}\|\mathbf{b}_i\|_2^2}_{c_i\in\mathbb{R}}.\end{aligned} \tag{6.4}$$

- (**gradient**) Differentiating a quadratic function w.r.t. $\mathbf{x}$ amounts to:

$$\nabla_{\mathbf{x}}\left(\frac{1}{2}\mathbf{x}^\top\mathbf{P}_i\mathbf{x}+\mathbf{v}_i^\top\mathbf{x}+c_i\right)=\mathbf{P}_i\mathbf{x}+\mathbf{v}_i. \tag{6.5}$$

  Thus, the gradient of f_i at a point $\mathbf{x}$ can be computed as

$$\nabla f_i(\mathbf{x})=\mathbf{A}_i^\top\mathbf{A}_i\mathbf{x}-\mathbf{A}_i^\top\mathbf{b}_i=\mathbf{A}_i^\top(\mathbf{A}_i\mathbf{x}-\mathbf{b}_i). \tag{6.6}$$

- (**convexity**) By Example 5.19 [Bec17], we have that quadratic functions are strongly convex, with $\mu=\lambda_{\min}(\mathbf{P})$. Therefore, if $\mathbf{A}_i$ is a skinny/tall matrix, i.e. $m\geq p$, with linearly independent columns, then f_i is strongly convex with parameter

$$\mu_i=\lambda_{\min}\left(\mathbf{A}_i^\top\mathbf{A}_i\right). \tag{6.7}$$

  Otherwise, the system of equations is underdetermined and the above quantity is 0, thus, the objective function is not strongly convex.

- (**smoothness**) By Example 5.2 [Bec17], we have that quadratic functions are smooth with a Lipschitz constant of $L=\|\mathbf{P}\|_{2,2}\equiv\sigma_{\max}(\mathbf{P})$. Hence, f_i is smooth with

$$L_i=\sigma_{\max}(\mathbf{A}_i^\top\mathbf{A}_i). \tag{6.8}$$

For each agent, we first generate the matrix of regressors, where each of the m rows is distributed according to $\mathcal{N}(\mathbf{0},\mathbf{I}_p)$. We additionally generate the true optimal value of $\mathbf{x}_\star\sim\mathcal{N}(\mathbf{0},\mathbf{I}_p)$, which is to be estimated, and is necessarily common to all agents. Using these, we generate the *target variables* as $\mathbf{b}_i=\mathbf{A}_i\mathbf{x}_\star+\boldsymbol{\epsilon}$, with $\epsilon\sim\mathcal{N}(\mathbf{0},\sigma^2\mathbf{I}_m)$.

6.2 Logistic Regression

A set of N agents, each one having collected n_i pairs of data pairs $(\mathbf{x}_j,y_j)$, where $\mathbf{x}_j\in\mathbb{R}^p$ and $y\in\{0,1\}$. We assume that there exists an optimal hyperplane

$$\mathcal{H}_{\mathbf{w}_\star,b_\star}=\left\{\mathbf{x}_\star\in\mathbb{R}^p\mid\mathbf{w}_\star^\top\mathbf{x}=b_\star\right\}, \tag{6.9}$$

such that

$$y_j=\begin{cases}1, & \text{if } \mathbf{w}_\star^\top\mathbf{x}_j-b_\star>0,\\ 0, & \text{otherwise.}\end{cases} \tag{6.10}$$

If the data are linearly separable, the optimal hyperplane can be used to correctly classify each data sample to its class (0 or 1). In the case of non-separable data, the optimal hyperplane is one which makes the least amount of misclassifications.

Our goal is to estimate the parameters of the optimal hyperplane, such that all samples from all agents are correctly classified. To aid that goal, we re-parametrize the hyperplane by defining

$\boldsymbol{\theta} \coloneqq (b, \mathbf{w})$, which additionally requires us to augment each data point $\mathbf{x}_j$ by adding a -1 in its first element. Using the logistic function, we define:

$$h_{\boldsymbol{\theta}}(\mathbf{x}) = \frac{1}{1 + e^{-\boldsymbol{\theta}^\top \mathbf{x}}}. \tag{6.11}$$

When $\boldsymbol{\theta}^\top \mathbf{x} \gg 0$, then $h_{\boldsymbol{\theta}} \approx 1$, when $\boldsymbol{\theta}^\top \mathbf{x} \ll 0$, then $h_{\boldsymbol{\theta}} \approx 0$.

Thus, we can define the loss function of each agent i as:

$$f_i(\boldsymbol{\theta}) = -\frac{1}{n_i} \sum_{j=1}^{n_i} y_j \log h_{\boldsymbol{\theta}}(\mathbf{x}_j) + (1 - y_j) \log(1 - h_{\boldsymbol{\theta}}(\mathbf{x}_j)), \tag{6.12}$$

which can be re-written as

$$f_i(\boldsymbol{\theta}) = -\frac{1}{n_i} \mathbf{y} \mathbf{X} \boldsymbol{\theta} + \frac{1}{n_i} \sum_{j=1}^{n_i} \log \left(\boldsymbol{\theta}^\top \mathbf{x}_j \right), \tag{6.13}$$

where $\mathbf{X} \coloneqq [\mathbf{x}_1, \mathbf{x}_2, \cdots, \mathbf{x}_{n_i}] \in \mathbb{R}^{n_i \times p}$, $\mathbf{y} \coloneqq [y_1, y_2, \cdots, y_{n_i}] \in \mathbb{R}^{n_i}$, for all agents i.

The distributed Logistic Regression problem is then

$$\begin{aligned} &\text{minimize} \quad \frac{1}{N} \sum_{i=1}^{N} f_i(\boldsymbol{\theta}_i) \\ &\text{subject to} \quad \boldsymbol{\theta}_1 = \boldsymbol{\theta}_2 = \cdots = \boldsymbol{\theta}_N, \end{aligned} \tag{6.14}$$

Each function f_i satisfies some important properties:

- (**gradient**) The gradient of f_i at a point $\boldsymbol{\theta}$ is given by

$$\nabla f_i(\boldsymbol{\theta}) = \frac{1}{n_i} \mathbf{X}(h_{\boldsymbol{\theta}}(\mathbf{X}^\top) - \mathbf{y}). \tag{6.15}$$

- (**Hessian**) The Hessian of f_i is

$$\nabla^2 f_i(\boldsymbol{\theta}) = \frac{1}{n_i} \sum_{j=1}^{n_i} h_{\boldsymbol{\theta}}(\mathbf{x}_j)(1 - h_{\boldsymbol{\theta}}(\mathbf{x}_j)) \mathbf{x}_j \mathbf{x}_j^\top. \tag{6.16}$$

  Note that $0 \leq h_{\boldsymbol{\theta}}(\mathbf{x}_j)(1 - h_{\boldsymbol{\theta}}(\mathbf{x}_j)) \leq \frac{1}{4}$, therefore

$$\mathbf{0} \leq \nabla^2 f_i(\boldsymbol{\theta}) \leq \frac{1}{4} \mathbf{X} \mathbf{X}^\top. \tag{6.17}$$

- (**convexity**) Since $\nabla^2 f_i(\boldsymbol{\theta}) \geq \mathbf{0}$, then f_i is convex. If we further add, the common l_2 regularizer, as

$$\frac{\lambda}{2} \|\boldsymbol{\theta}\|_2^2, \tag{6.18}$$

  then the function becomes λ-strongly convex.

- (**smoothness**) Since $\nabla^2 f_i(\boldsymbol{\theta}) \leq \frac{1}{4} \mathbf{X} \mathbf{X}^\top$, then $\frac{1}{4} \mathbf{X} \mathbf{X}^\top$ is an upper bound on the Lipschitz constant of f_i. Thus, f_i is L-smooth with $L = \frac{1}{4} \mathbf{X} \mathbf{X}^\top$.

We generate the, common and hidden, optimal hyperplane parameters as $\mathbf{w}_\star \sim \mathcal{N}(\mathbf{0}, I)$ and $b_\star \sim \mathcal{N}(0, 1)$. Then, from a randomly selected point that lies on the generated hyperplane, we translate each $\mathbf{x}_j \sim \mathcal{N}(\mathbf{0}, \boldsymbol{\sigma} I)$ by either $\pm \mathbf{w}_\star$ depending on its label y_j. Since $\mathbf{w}_\star$ is a vector orthogonal to the hyperplane, points from one class will be on one side, and points from the other class on the other side of the hyperplane. By changing σ, we can control how much overlap, if any, there is between points of the two classes, determining if they are lineary separable, or not. The above scheme is repeated for all agents with the same generated hyperplane, for consistency.

6.3 LASSO

The Least Absolute Shrinkage and Selection Operator is essentially a Least Squares Linear Regression problem with an l_1 regularizer.

$$\begin{aligned} \text{minimize} \quad & \frac{1}{N}\sum_{i=1}^{N} f_i(\mathbf{x}_i) = \frac{1}{N}\sum_{i=1}^{N} \frac{1}{2}\|\mathbf{A}_i\mathbf{x}_i - \mathbf{b}_i\|_2^2 + \lambda\|\mathbf{x}_i\|_1 \\ \text{subject to} \quad & \mathbf{x}_1 = \mathbf{x}_2 = \cdots = \mathbf{x}_N. \end{aligned} \tag{6.19}$$

Regularization in the form of $\lambda\|\mathbf{x}\|_1$ is known to promote sparsity (few non-zero elements) on the optimization parameter $\mathbf{x}$. It is used in place of the l_0 pseudo-norm, which measures the number of non-zero elements of $\mathbf{x}$.

Assuming that the optimal value is also sparse, LASSO can be used to estimate $\mathbf{x}$ with much fewer rows of data from the matrix $\mathbf{A}$.

When the number of rows m (i.e. number of samples) is less than the dimension of $\mathbf{x}$, the system of equations is underdetermined and therefore the loss function is *not strongly convex*, but convex nonetheless. In addition, the l_1 norm is non-smooth, making the objective function as a whole, *non-smooth* as well.

We generate the data for each agent exactly as in the Least Squares case, however the common optimal value $\mathbf{x}_\star$ is generated with few non-zero elements.

Chapter 7

Numerical Experiments

We investigate the performance of the algorithms presented in Chapter 6. To that end, we will, in general, illustrate the convergence to the optimal point via two quantities over the duration of the algorithm (in steps): (a) the *iterate residual* and (b) the *objective residual*. As we will often compare these distributed algorithms to their centralized counterparts, different notions of residuals are required.

We define the *relative parameter residual* at time t as the distance between the current iterate and the optimal point, normalized by the magnitude of the optimal point. Thus, for the centralized case, we have

$$\mathrm{rpr}^t \coloneqq \frac{\|\mathbf{x}^t - \mathbf{x}^\star\|_2}{\|\mathbf{x}^\star\|_2}. \tag{7.1}$$

As for the distributed case, we have

$$\mathrm{rpr}^t \coloneqq \frac{\|\mathbf{X}^t - \mathbf{X}^\star\|_F}{\|\mathbf{X}^\star\|_F} \tag{7.2}$$

which is, in addition, a metric of the disagreement between the agents' estimates.

The *objective function residual* at time t is defined as the difference of the value of f at the current estimate with the optimal value of f. For the centralized case we have

$$\mathrm{ofr}^t \coloneqq f(\mathbf{x}^t) - f^\star. \tag{7.3}$$

For the distributed case, it is the empirical average of the values of each agent's f_i at its respective $\mathbf{x}_i$, and is defined as

$$\mathrm{ofr}^t \coloneqq f(\mathbf{X}^t) - f^\star, \tag{7.4}$$

where the global function f is defined in (5.1).

Note, that when the optimal point $\mathbf{x}^\star$ is not known beforehand, we solve the problem via the `cvxpy` library [DB16] and use its estimate instead.

7.1 Random Network Generation

For all experiments that will follow, we generate an undirected connected graph and a mixing matrix that is doubly stochastic and symmetric (Assumption 1).

We generate random Erdős–Rényi graphs, with the probability of an edge existing between any two vertices set to $p = 0.2$. We repeat until a connected graph is generated.

There are many ways of choosing the mixing matrix $\mathbf{W}$ in a decentralized fashion [Shi+15b], each with its own advantages and disadvantages. We chose the Metropolis-Hastings weights [XBK07]

for their ease of implementation:

$$w_{ij} = \begin{cases} \frac{1}{2\max\{d_i, d_j\}}, & i \neq j,\ i, j \in \mathcal{E}, \\ 2 - \sum_{k \in \mathcal{N}_i} w_{ik}, & i = j, \\ 0, & i \neq j\ i, j \notin \mathcal{E}, \end{cases} \tag{7.5}$$

where d_i is the degree of agent/vertex i, and $\mathcal{N}_i$ is the set of its neighbors. Importantly, this choice of mixing matrix satisfies Assumption 1, as it is symmetric and each row (and each column) sums to 1.

7.2 Non-Smooth Problems

We solve the distributed *non-smooth* convex problem of LASSO (6.19), i.e., Linear Regression via Least Squares with l_1 regularization. The optimal parameter vector $\mathbf{x}^\star$ is sparse, and the system of equations is underdetermined, as is often assumed in the compressed sensing setting.

Note that problems with l_1 regularization belong to a subclass of non-smooth problems, and are easier to solve than other non-smooth problems, mainly due to our ability to compute the proximal mapping of the l_1 norm.

We compare the following algorithms for the solution of the non-smooth problem:

- Centralized Subgradient Descent (Algorithm 1) with
$$\alpha_t = \frac{1}{\sqrt{t+1} \cdot \|\nabla_f(\mathbf{x}^t)\|_2}. \tag{7.6}$$
- ISTA (Algorithm 6).
- Distributed Subgradient Descent (Algorithm 8) with
$$\alpha_{t,i} = \frac{1}{\sqrt{t+1} \cdot \|\nabla f_i(\mathbf{x}_i^t)\|_2}. \tag{7.7}$$
- PG-EXTRA (Algorithm 13) with
$$\alpha_t = \frac{2\lambda_{\min}(\tilde{\mathbf{W}})}{\max_i L_i}, \tag{7.8}$$
where $\lambda_{\min}$ denotes the smallest eigenvalue, and L_i is the smoothness parameter of f_i.

For both ISTA and PG-EXTRA, we let $g(\mathbf{x}) = \lambda\|\mathbf{x}^t\|_1$. ISTA (iterative shrinkage-thresholding algorithm) [Bec17], is the name given to Proximal Gradient Descent for l_1 regularized problems.

The proximal mapping (2.12) of the l_1-norm has a closed-form solution [Bec17]
$$prox_{\lambda\|\cdot\|_1}(\mathbf{x}) = [\|\mathbf{x}\| - \lambda]_+ \cdot \operatorname{sgn}(\mathbf{x}). \tag{7.9}$$

Note that, for both centralized and decentralized GD, the stepsizes satisfy Equations (5.3), and the gradients are actually subgradients, due to the non-differentiability of the objectives.

The convergence of these algorithms is illustrated in Figure 7.1 for the LASSO problem with $N = 10$ agents (network structure as in Section 7.1), parameter dimension of $p = 100$, and $n_i = 5$ data examples per agent ($n = 50$ total). Data generation is described in Section 6.3, more specifically, we set $\sigma_\epsilon = 10^{-3}$, and $\lambda = 10^{-2}$ for all agents. Additionally, we set s, the number of non-zero elements of $\mathbf{x}^\star$, equal to 5% of its total size, and the total number of data points m is given by $m \gtrapprox 2s\log(p) \ll p$.

Both the centralized and decentralized versions of Gradient Descent struggle with this problem, with convergence appearing to slow down to a halt. The proximal methods however, readily solve the problem, due to the proximal mapping to the l_1 norm. Although PG-EXTRA is slower than its centralized counterpart, ISTA, it behaves very similarly, that is, being slow to converge at the start, then making rapid progress to the exact solution.

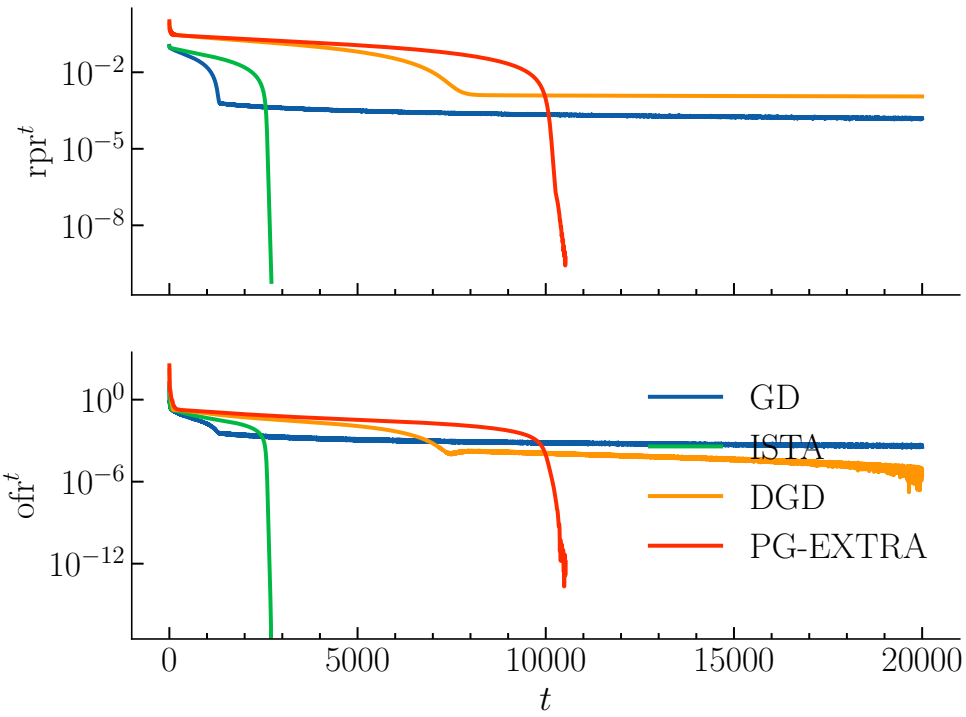


Figure 7.1: GD vs ISTA vs DGD vs PG-EXTRA for non-smooth objective (LASSO). $N = 10$, $p = 100$, $n_i = 5$ ($n = 50$), $\sigma_\epsilon = 10^{-3}$, $\lambda = 10^{-2}$.

7.3 Smooth Problems

We consider two types of distributed *smooth* convex problems:

- Linear Regression via Least Squares (6.2), without regularization, where the system of equations is underdetermined (the global system is overdetermined however) and noisy.
- Logistic Regression (6.14) without regularization, for non-separable data.

Smooth convex problems facilitate the use of accelerated algorithms, as well as allowing the stepsizes to be constant (inversely proportional to the smoothness parameter L). To that end, we compare the following algorithms:

- Centralized Gradient Descent (Algorithm 1) with
$$\alpha_t = \frac{2}{L}. \tag{7.10}$$
- Centralized Nesterov Accelerated Gradient Descent NSC (Algorithm 4)
- Distributed Gradient Descent (Algorithm 8) with
$$\alpha_t = \frac{1}{\max_i L_i}. \tag{7.11}$$
- EXTRA (Algorithm 9) with
$$\alpha_t = \frac{1}{\max_i L_i}. \tag{7.12}$$

The stepsizes of these distributed algorithms require the agents to compute $\max_i L_i$, where L_i the smoothness parameter of agent i. Thus, a maximum consensus algorithm needs to be run, whereby each agent updates their estimate of $\max_i L_i$ by taking the maximum of its neighbors' estimates.

Accelerated optimization methods tend to exhibit greater performance gains, when compared to their non-accelerated counterparts, for harder problems, i.e., problems with large condition number ($\kappa = L/\mu$). Since, however, these problems are not strongly convex ($\mu = 0$), we use the number of available data points as a proxy to the condition number, where more data would correspond to a smaller condition number.

In order to test the speedup for harder problems, we solve each of the two problems twice, where the second we significantly decrease the amount of data points.

For Linear Regression we generate data as shown in Section 6.1 ($\sigma_\epsilon = 10^{-3}$), we have $N = 10$ agents (network structure as in Section 7.1), parameter dimension of $p = 100$. We use $n_i = 20$ and $n_i = 11$ data examples per agent ($n = 200$ and $n = 110$ total) for the experiments in Figure 7.2 and Figure 7.3 respectively.

For Logistic Regression we generate data as shown in Section 6.2, we have $N = 10$ agents (network structure as in Section 7.1), parameter dimension of $p = 10$ ($p = 11$ via augmentation). We use $n_i = 10$ and $n_i = 5$ data examples per agent ($n = 100$ and $n = 50$ total) for the experiments in Figure 7.4 and Figure 7.5 respectively.

First, for the centralized algorithms, it is evident that NGD provides a speedup compared to GD for the harder problems, as expected. DGDs performance plateaus early on, especially for the harder problem of Logistic Regression. It reaches only a neighborhood of the solution, as predicted by the theory. Lastly, EXTRA achieves exact convergence, albeit being slightly slower than both of the centralized algorithms.

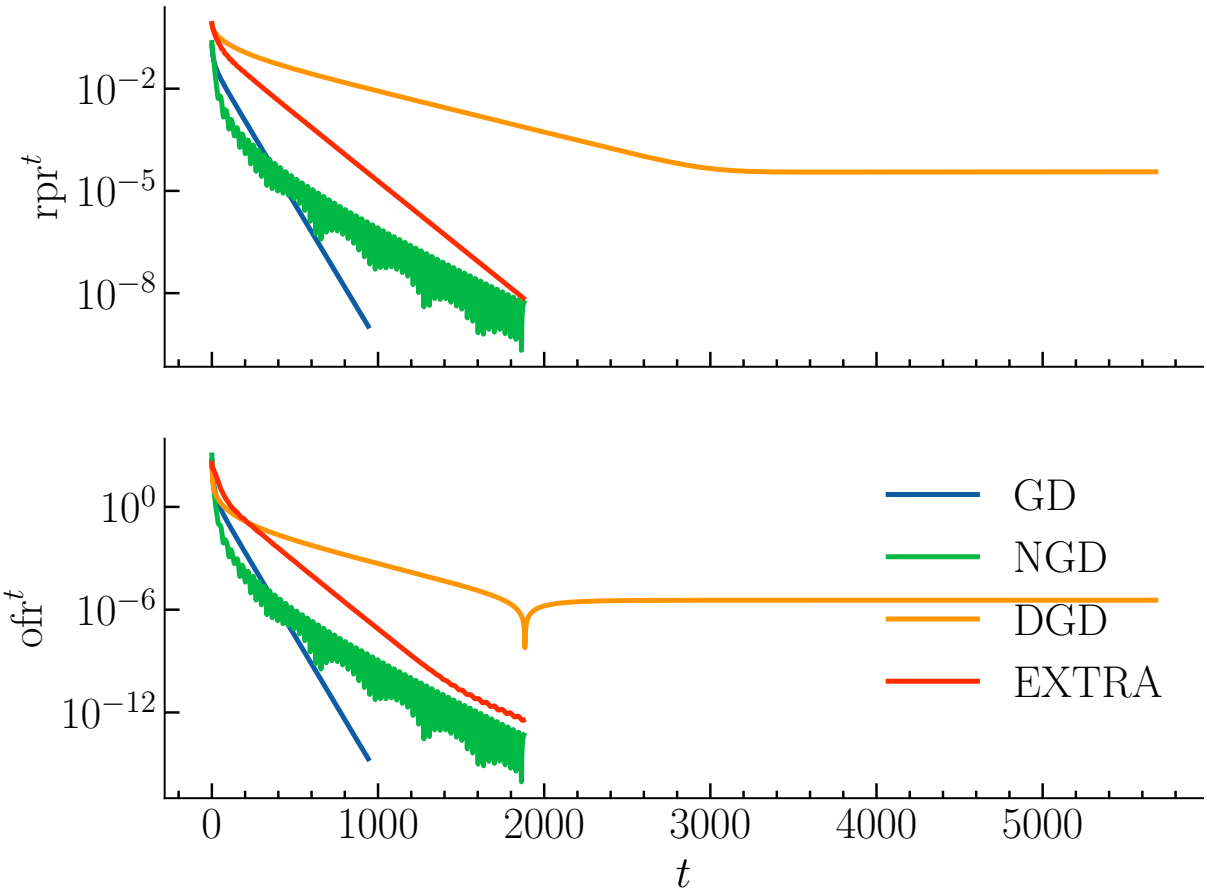


Figure 7.2: GD vs NGD vs DGD vs EXTRA for smooth objective (Linear Regression) with many data examples. $N = 10$, $p = 100$, $n_i = 20$ ($n = 200$), $\sigma_\epsilon = 10^{-3}$, $\lambda = 0$.

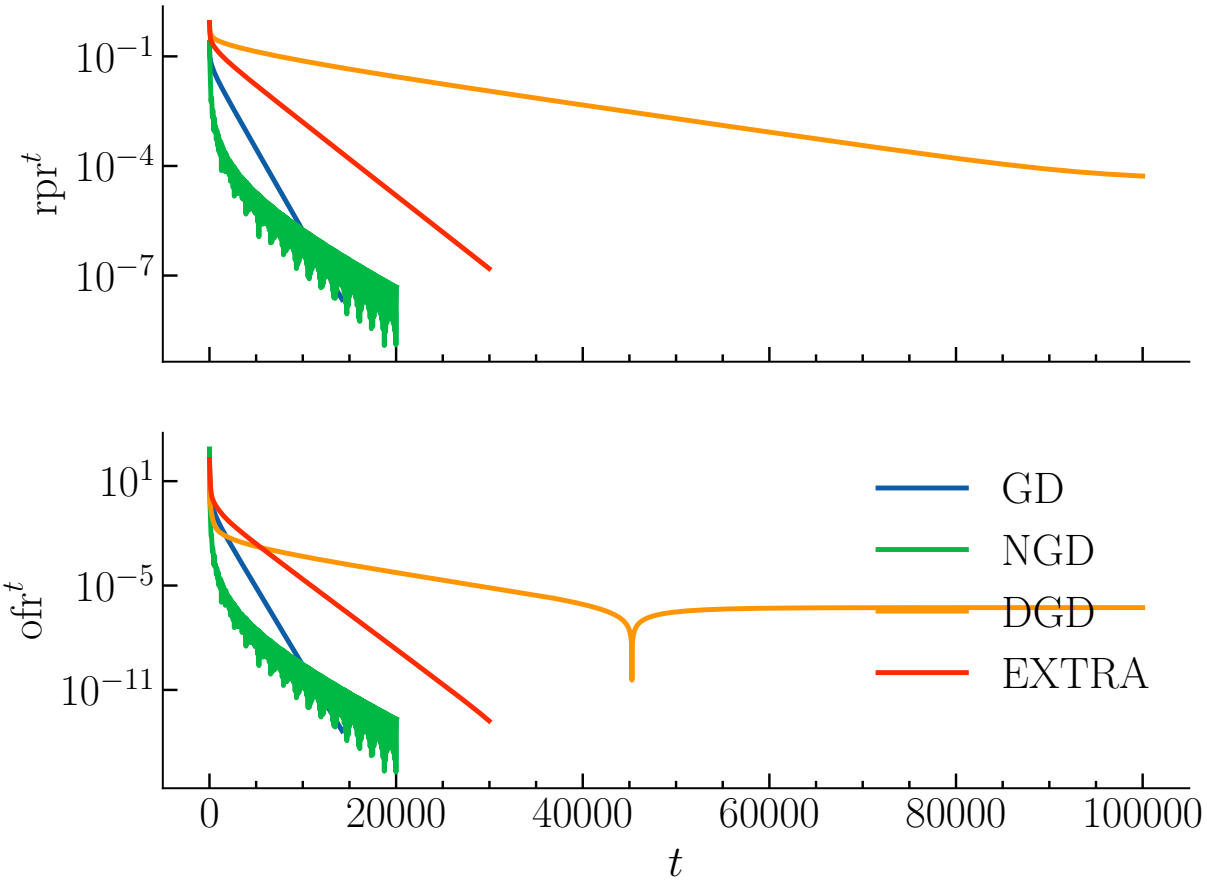


Figure 7.3: GD vs NGD vs DGD vs EXTRA for smooth objective (Linear Regression) with few data examples. $N = 10$, $p = 100$, $n_i = 10$ ($n = 110$), $\sigma_\epsilon = 10^{-3}$, $\lambda = 0$.

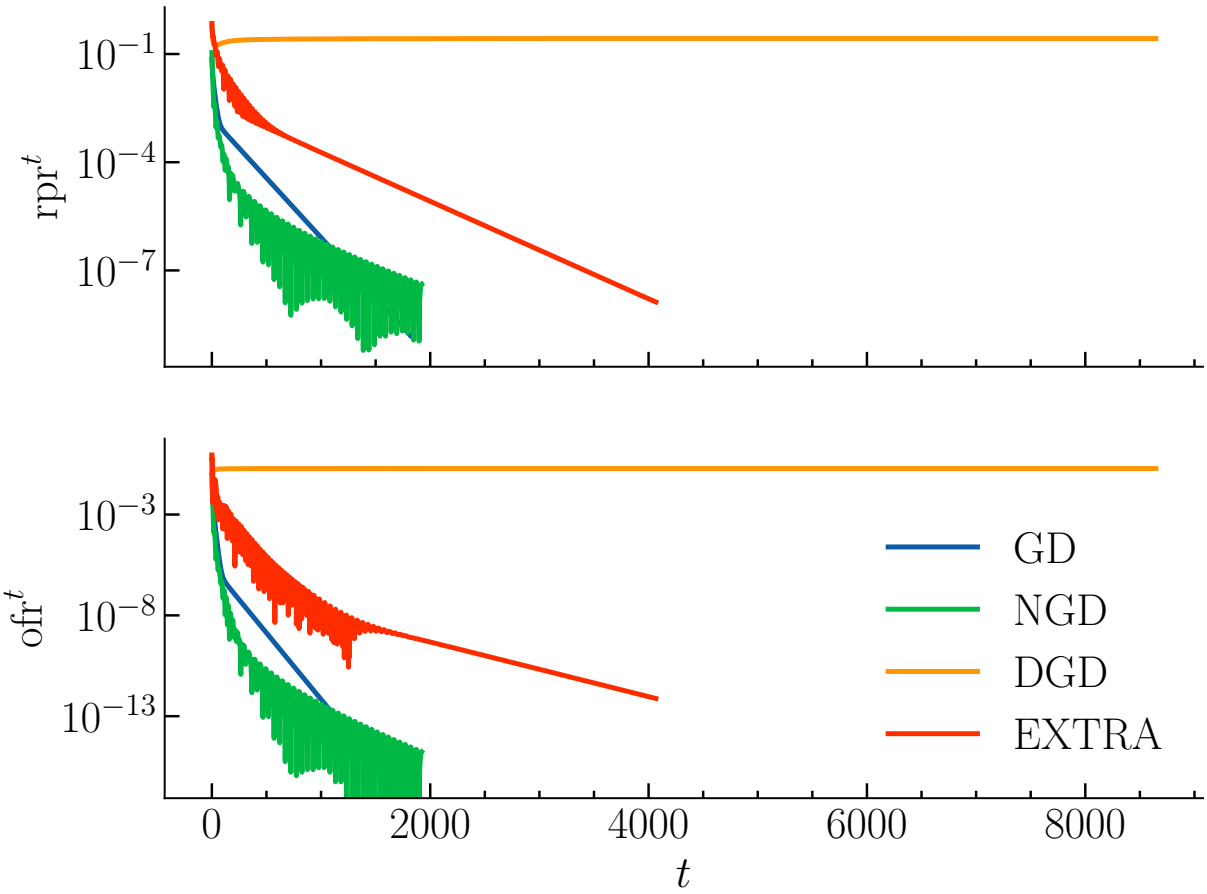


Figure 7.4: GD vs NGD vs DGD vs EXTRA for smooth objective (Logistic Regression) with many data examples. $N = 10$, $p = 10$, $n_i = 10$ ($n = 100$), $\lambda = 0$.

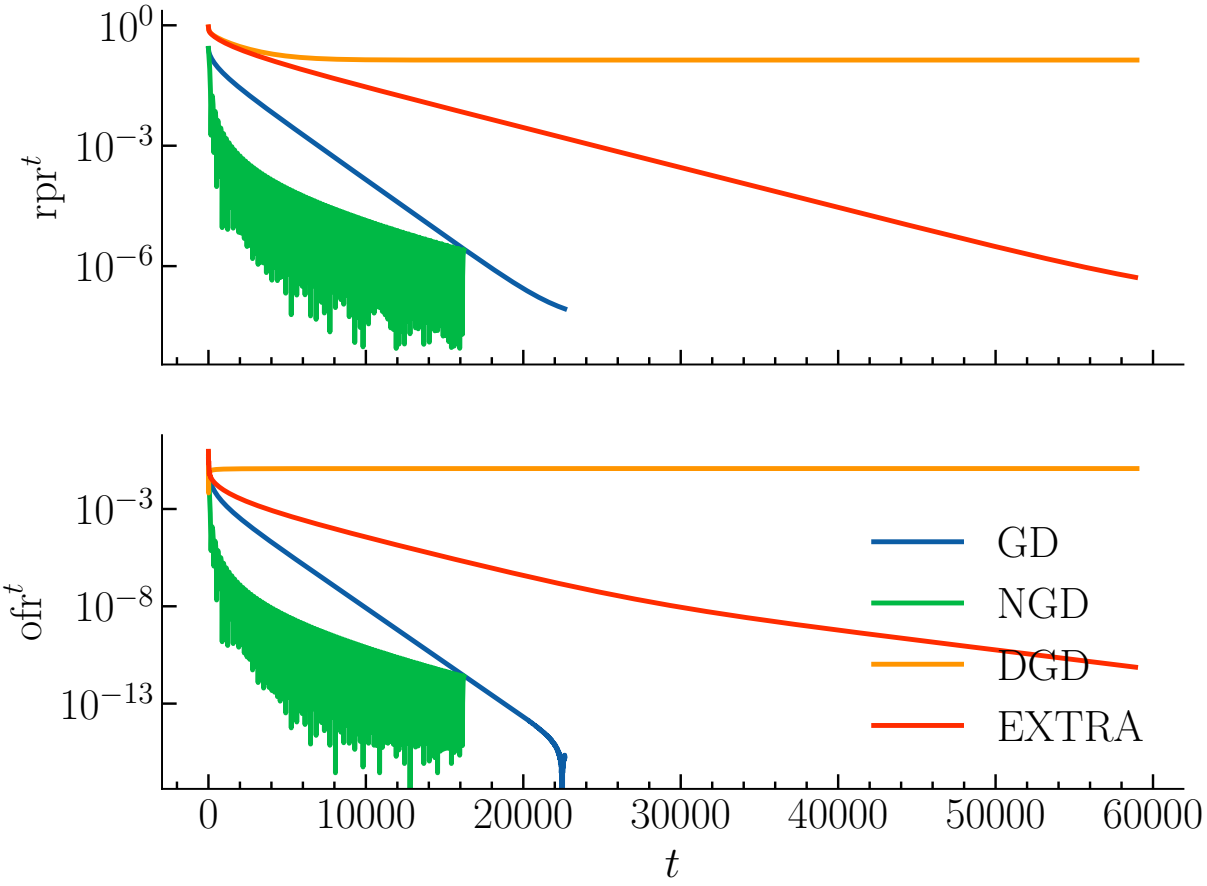


Figure 7.5: GD vs NGD vs DGD vs EXTRA for smooth objective (Logistic Regression) with few data examples. $N = 10$, $p = 10$, $n_i = 5$ ($n = 50$), $\lambda = 0$.

7.4 Strongly Convex and Smooth Problems

We consider two types of distributed *strongly convex* and *smooth* problems:

- Linear Regression via Least Squares (6.2), with l_2 regularization, where the system of equations is underdetermined (the global system is overdetermined however) and noisy.
- Logistic Regression (6.14) with l_2 regularization, for separable data. Some form of regularization is required, for non-separable data, since the objective can be unbounded from below without it.

To that end, we compare the following algorithms:

- Centralized Gradient Descent (Algorithm 1) with
$$\alpha_t = \frac{2}{L}. \tag{7.13}$$
- Centralized Nesterov Accelerated Gradient Descent (Algorithm 3)
- Distributed Gradient Descent (Algorithm 8) with
$$\alpha_t = \frac{2}{\max_i L_i + \min_i \mu_i}. \tag{7.14}$$
- EXTRA (Algorithm 9) with
$$\alpha_t = \frac{1}{\max_i L_i}. \tag{7.15}$$
- SSDA (Algorithm 10).

SSDA uses the gradient of the conjugate function, which requires us to solve the maximization problem (5.33). For Logistic Regression we solve this problem via the `cvxpy` library [DB16]. However, for Linear Regression we can find a closed for solution to the problem. Problem (5.33) will achieve its maximum value when its gradient is 0, i.e.,

$$\begin{aligned} \mathbf{y} - \nabla f(\mathbf{x}) = 0 &\implies \mathbf{y} - \Big(\mathbf{A}^\top(\mathbf{A}\mathbf{x} - \mathbf{b}) + \lambda \mathbf{x}\Big) = 0 \\ &\implies \mathbf{y} + \mathbf{A}^\top \mathbf{b} - (\mathbf{A}^\top \mathbf{A} + \lambda \mathbf{I})\mathbf{x} = 0 \\ &\implies \mathbf{x} = (\mathbf{A}^\top \mathbf{A} + \lambda \mathbf{I})^{-1}(\mathbf{y} + \mathbf{A}^\top \mathbf{b}). \end{aligned} \tag{7.16}$$

Therefore, we have
$$\nabla f^*(\mathbf{y}) = (\mathbf{A}^\top \mathbf{A} + \lambda \mathbf{I})^{-1}(\mathbf{A}^\top \mathbf{b} + \mathbf{y}). \tag{7.17}$$

In contrast to the previous experiments of smooth problems, here we use the standard form of NGD, a larger step of DGD, and the dual algorithm SSDA. These distributed algorithms require a maximum/minimum consensus algorithm (described in Section 7.3) to be run for the computation of $\max_i L_i$ and $\min_i \mu_i$, respectively.

We solve each of the two problems twice, the second time with a smaller regularization parameter λ. Since the strong convexity parameter is equal to the regularization parameter, smaller values lead to a larger condition number ($\kappa = L/\mu$), and thus a harder problem to solve.

For Linear Regression we generate data as shown in Section 6.1 ($\sigma_\epsilon = 10^{-3}$), we have $N = 10$ agents (network structure as in Section 7.1), parameter dimension of $p = 100$, $n_i = 11$ data examples per agent ($n = 110$ total). We use $\lambda = 10^{-1}$ and $\lambda = 10^{-3}$ for the experiments in Figure 7.6 and Figure 7.7 respectively.

For Logistic Regression we generate data as shown in Section 6.2, we have $N = 10$ agents (network structure as in Section 7.1), parameter dimension of $p = 10$ ($p = 11$ via augmentation), $n_i = 5$ data examples per agent ($n = 50$ total). We use $\lambda = 10^{-1}$ and $\lambda = 10^{-3}$ for the experiments in Figure 7.8 and Figure 7.9 respectively.

The results here are similar to the experiments for the smooth problems in Section 7.3. NGD is generally much faster that the non-accelerated GD, especially for smaller λ values in the harder Logistic Regression problem. DGD converges to a neighborhood of the solution, and then plateaus. Interestingly, for the harder problems EXTRA is closer to centralized GD, both, however, being much slower than centralized NGD. Lastly, SSDA is able to either match or exceed the performance of centralized GD (and by extension EXTRA), despite being fully decentralized.

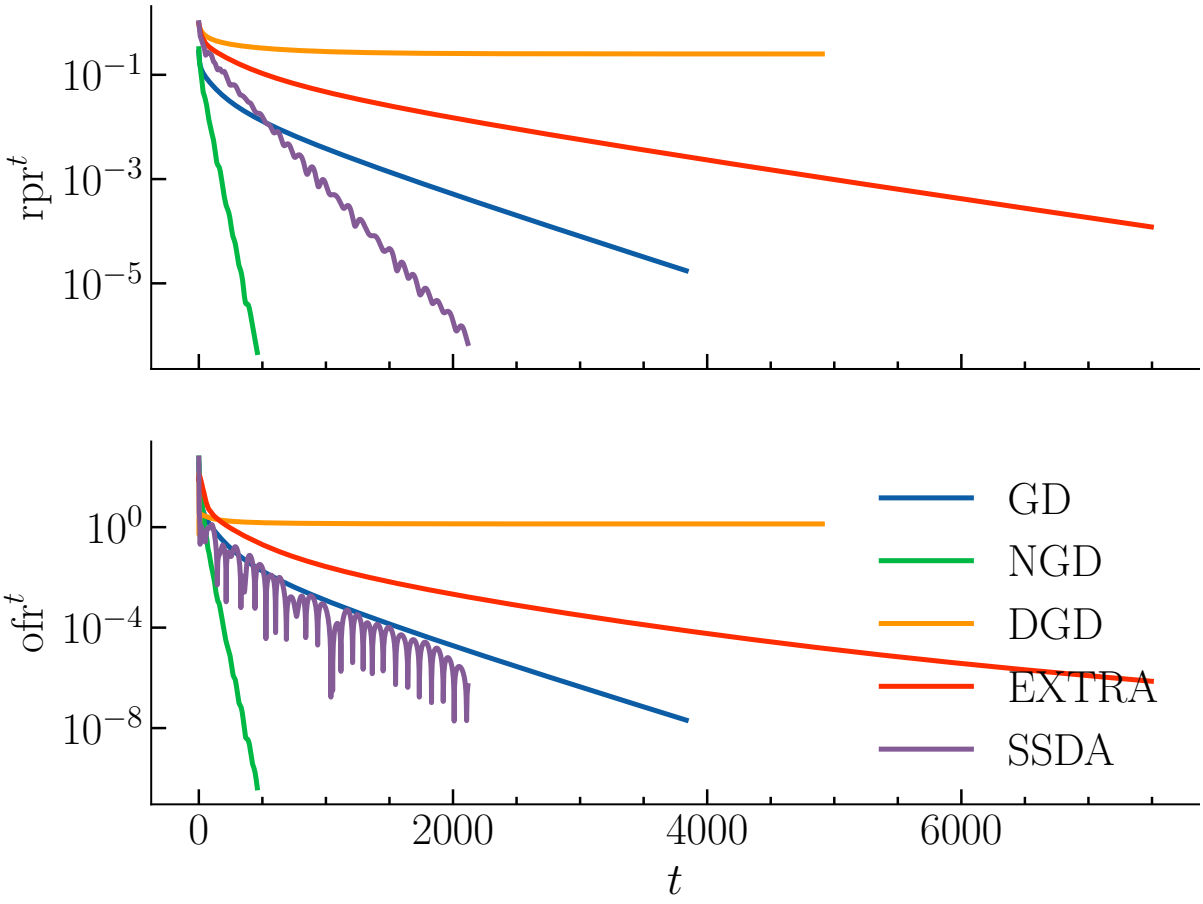


Figure 7.6: GD vs NGD vs DGD vs EXTRA vs SSDA for strongly convex & smooth objective (Linear Regression). $N = 10$, $p = 100$, $n_i = 11$ ($n = 110$), $\sigma_\epsilon = 10^{-3}$, $\lambda = 10^{-1}$.

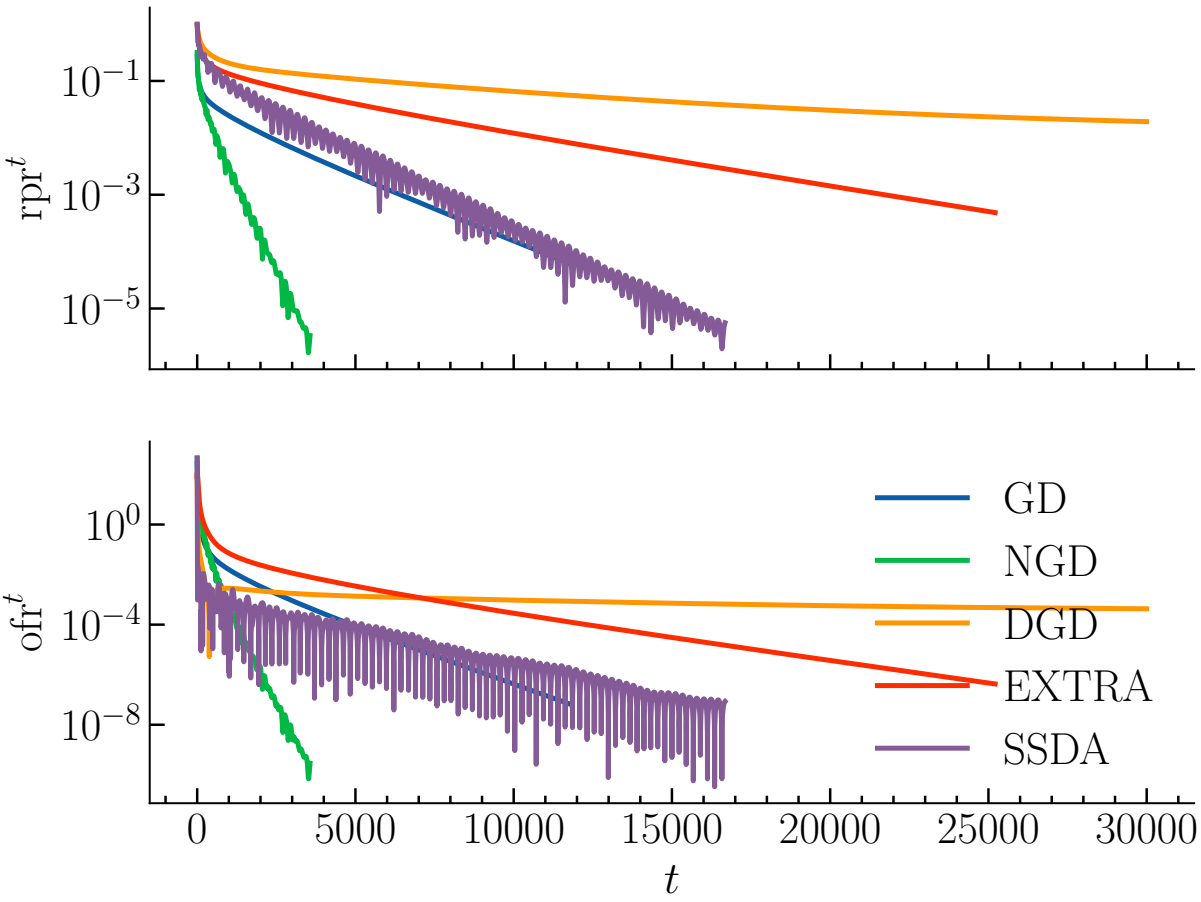


Figure 7.7: GD vs NGD vs DGD vs EXTRA vs SSDA for strongly convex & smooth objective (Linear Regression). $N = 10$, $p = 100$, $n_i = 11$ ($n = 110$), $\sigma_\epsilon = 10^{-3}$, $\lambda = 10^{-3}$.

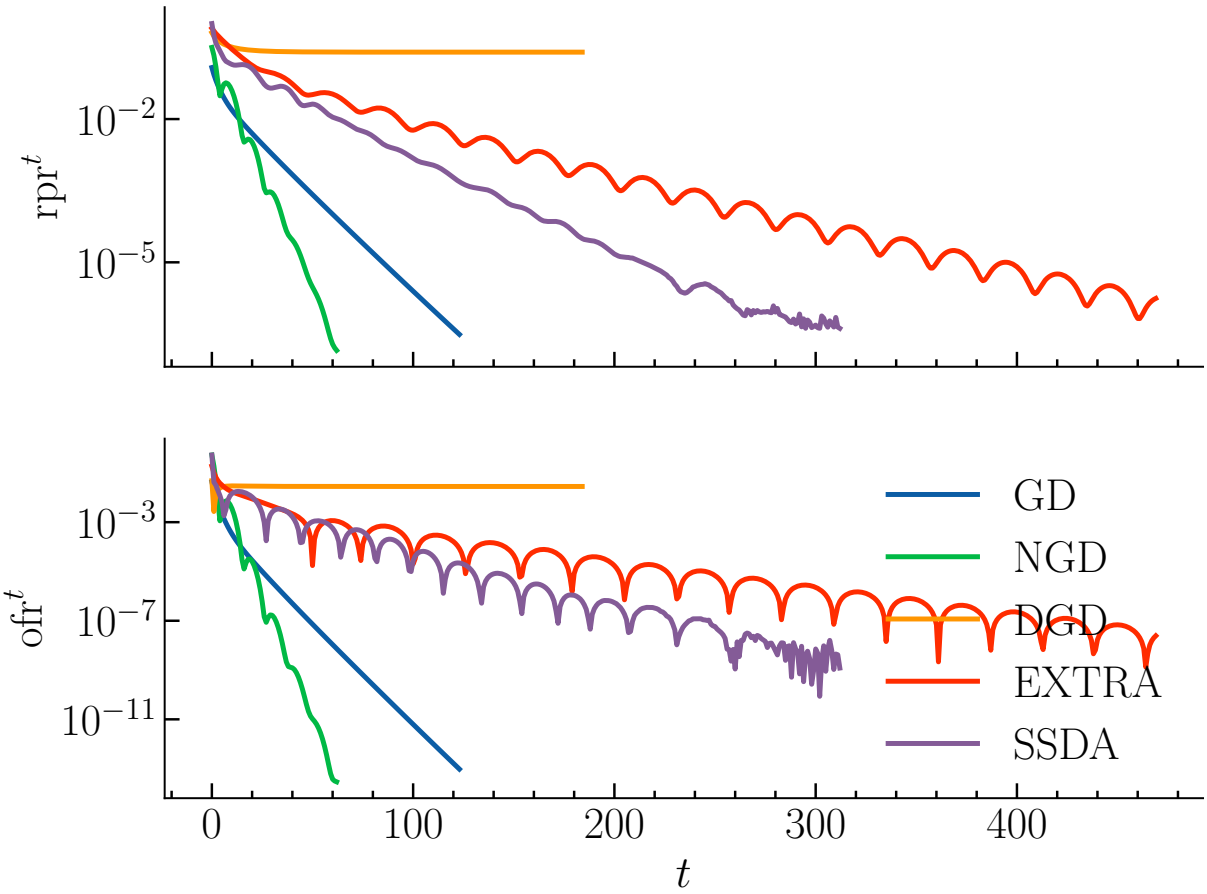


Figure 7.8: GD vs NGD vs DGD vs EXTRA vs SSDA for strongly convex & smooth objective (Logistic Regression). $N = 10$, $p = 10$, $n_i = 5$ ($n = 50$), $\lambda = 10^{-1}$.

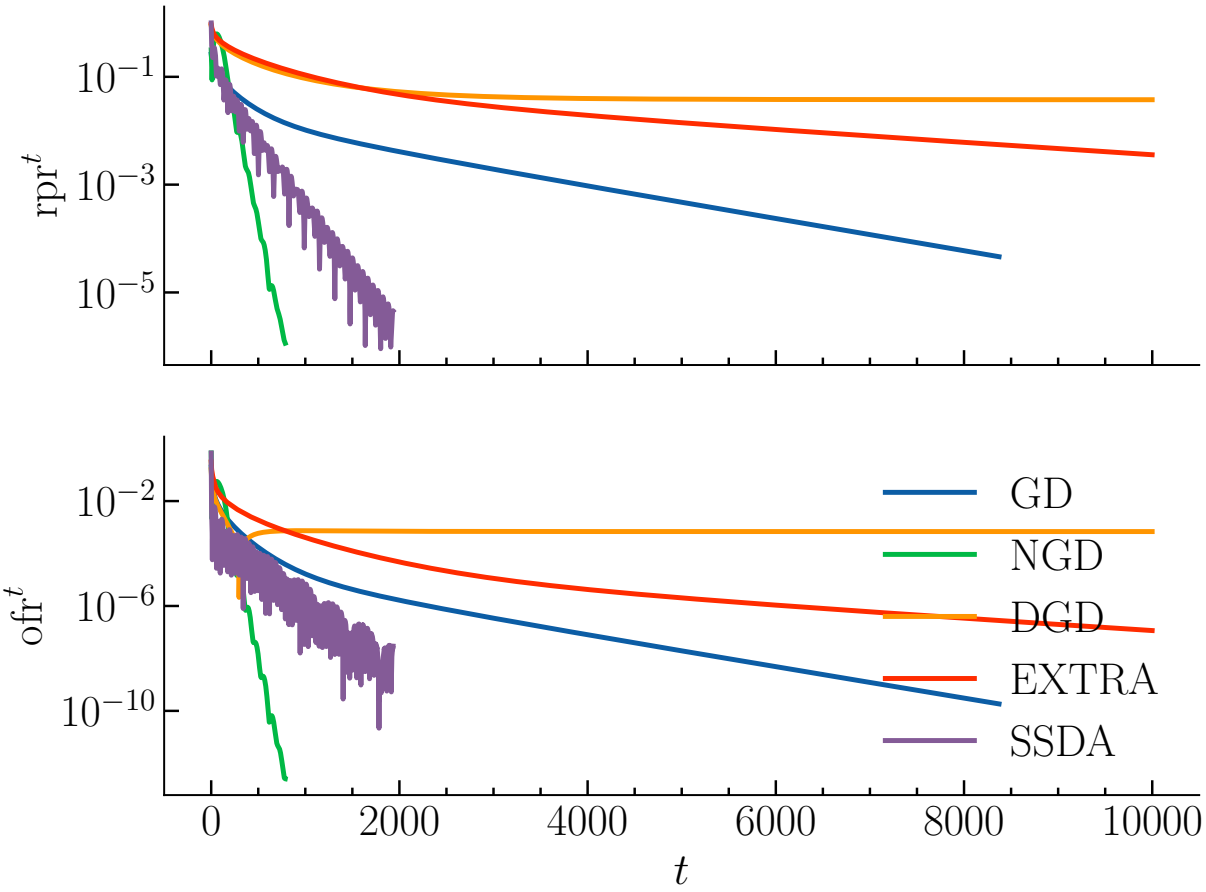


Figure 7.9: GD vs NGD vs DGD vs EXTRA vs SSDA for strongly convex & smooth objective (Logistic Regression). $N = 10$, $p = 10$, $n_i = 5$ ($n = 50$), $\lambda = 10^{-3}$.

7.5 Communication vs Computation tradeoff for Dual Algorithms

We examine the performance of SSDA (Algorithm 10) and MSDA (Algorithm 11), comparing their performance w.r.t. the number of gradient computations and the number of communication rounds between the agents. The same problems as in Section 7.4 are solved. More specifically, we employ the two dual algorithms to solve the hard version (smaller regularization parameter λ, larger condition number) of both Linear and Logistic Regression.

Figures 7.10 and 7.11 show the performance of the dual algorithms for the distributed Linear Regression problem, and Figures 7.12 and 7.13 for the Logistic Regression problem. Both problems have their parameters as in Section 7.4, for the case of $\lambda = 10^{-3}$. For reference, the amount of additional communication rounds K was 4 for these experiments.

In terms of the number of gradient computations, MSDA is much faster to converge than SSDA. Interestingly, when comparing the number of communication rounds, both algorithms seem to be very similar. Thus, one can readily use MSDA instead of SSDA, provided that the communication costs do not exceed the gradient computation costs.

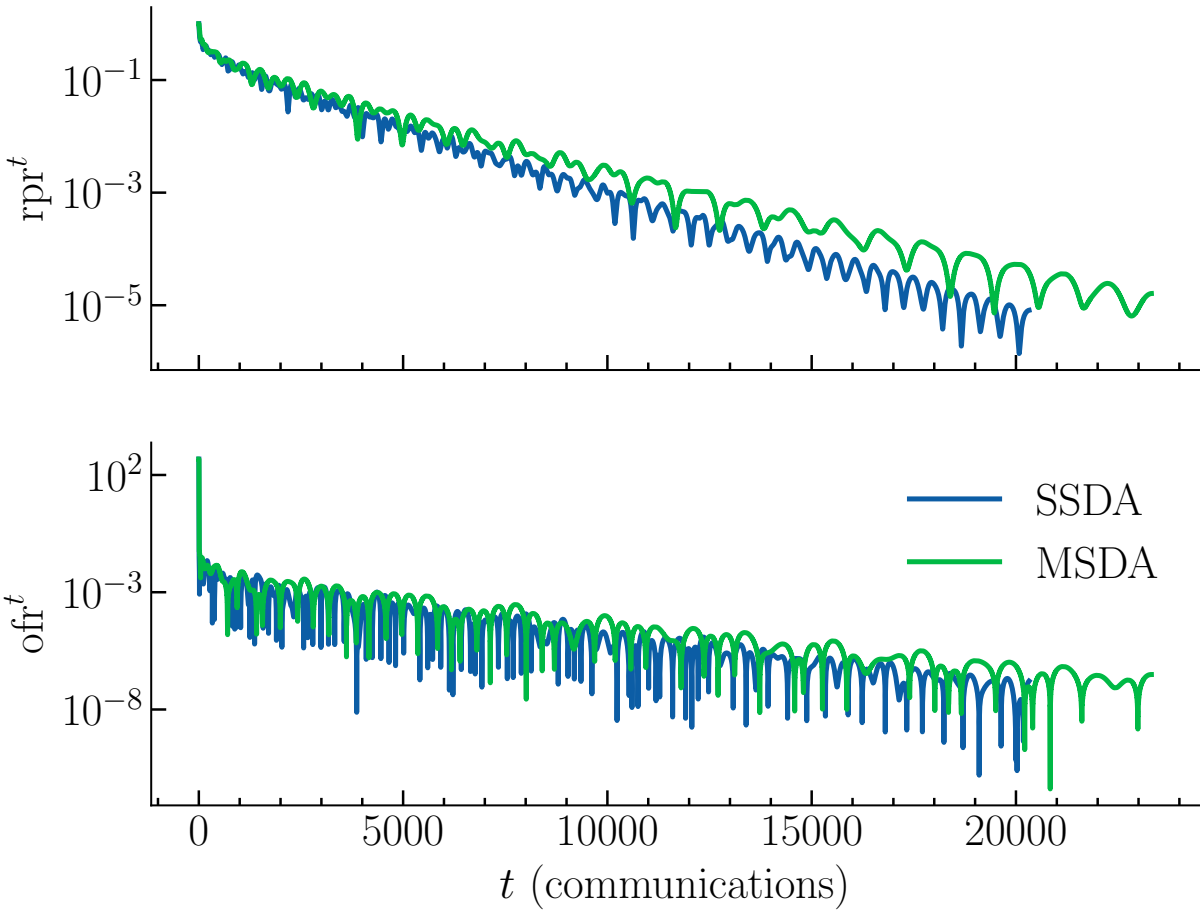


Figure 7.10: SSDA vs MSDA for Linear Regression, over the number of communications. $N = 10$, $p = 100$, $n_i = 11$ $(n = 110)$, $\sigma_\epsilon = 10^{-3}$, $\lambda = 10^{-3}$.

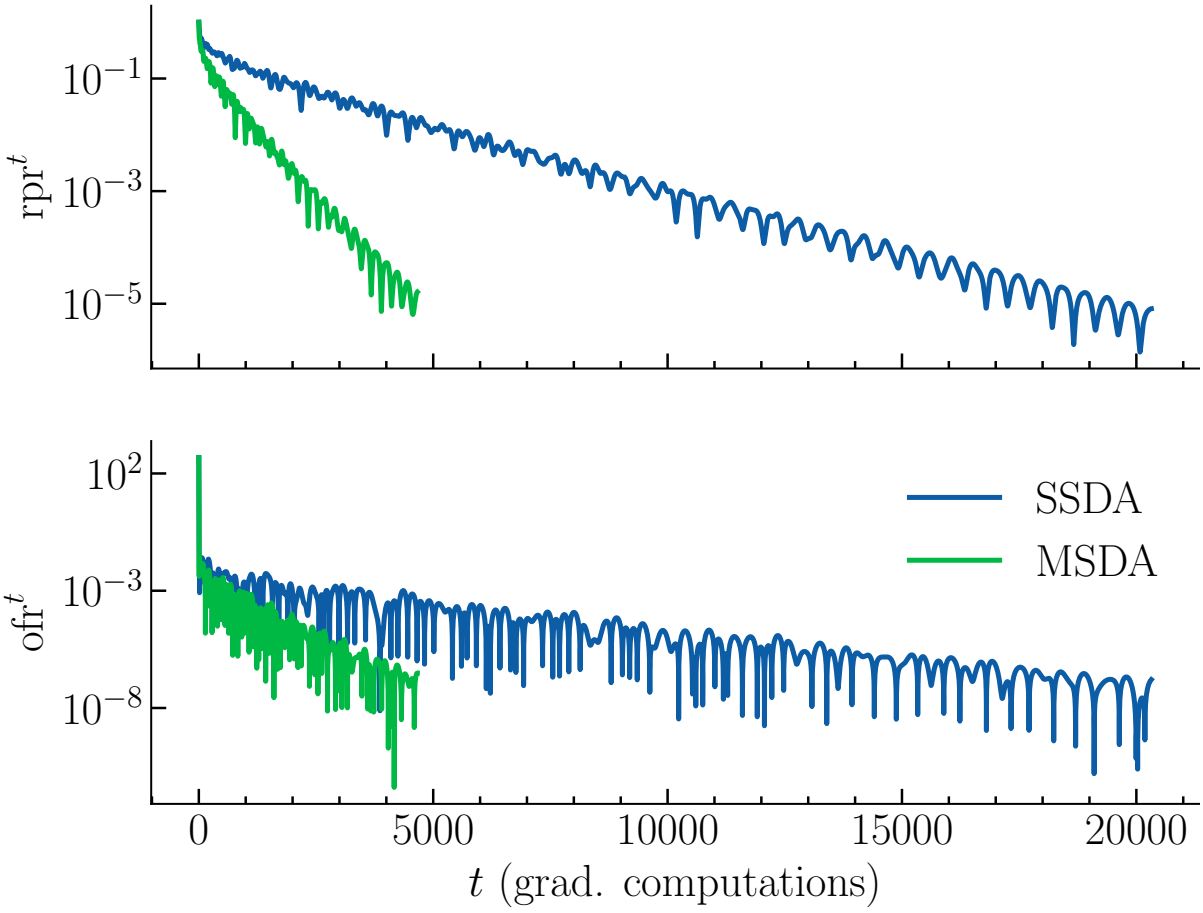


Figure 7.11: SSDA vs MSDA for Linear Regression, over the number of gradient computations. $N = 10$, $p = 100$, $n_i = 11$ ($n = 110$), $\sigma_\epsilon = 10^{-3}$, $\lambda = 10^{-3}$.

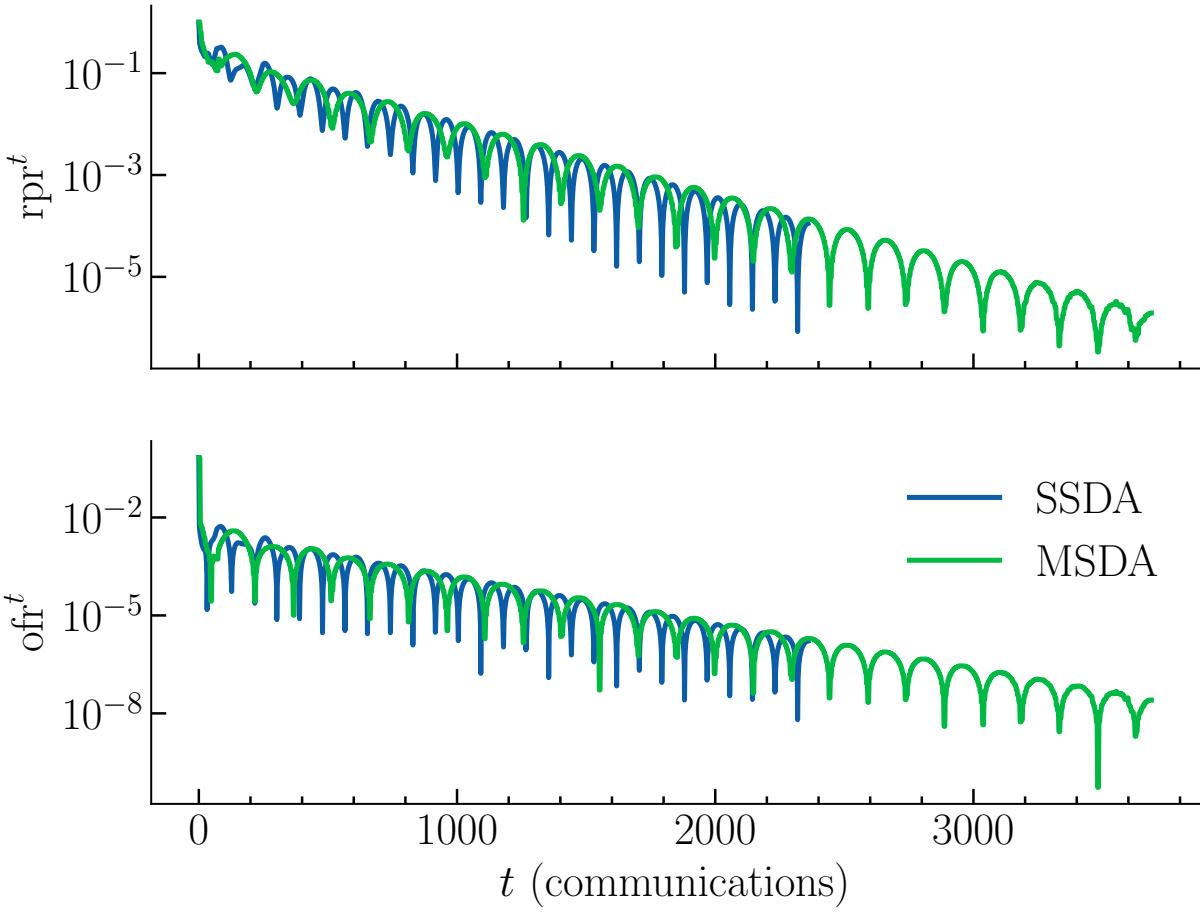


Figure 7.12: SSDA vs MSDA for Logistic Regression, over the number of communications. $N = 10$, $p = 10$, $n_i = 5$ ($n = 50$), $\lambda = 10^{-3}$.

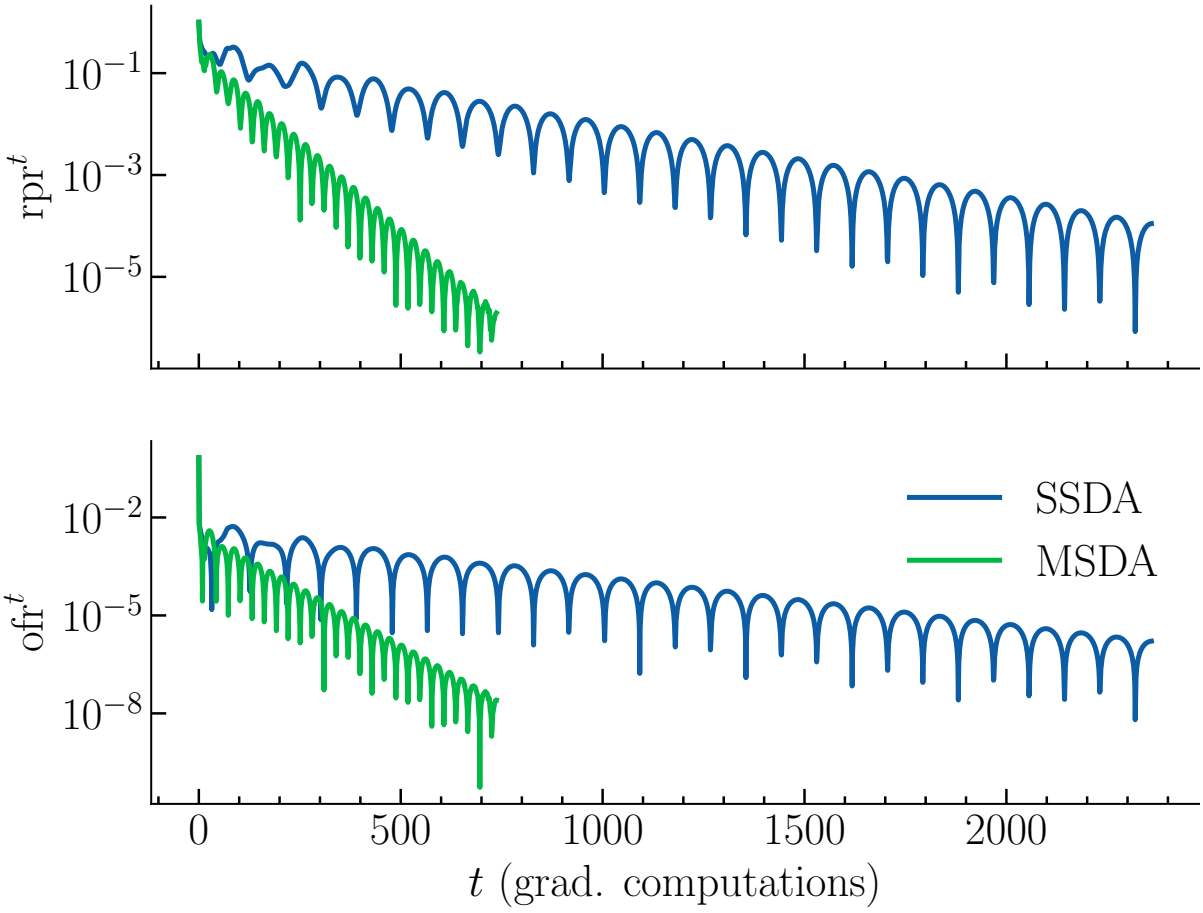


Figure 7.13: SSDA vs MSDA for Linear Regression, over the number of gradient computations. $N = 10$, $p = 10$, $n_i = 5$ ($n = 50$), $\lambda = 10^{-3}$.

7.6 Constrained Problems

We solve the distributed Least Squares problem (6.2) with additional *affine constraints* (5.40) on the optimization variable, both *local* and *global*.

A set of affine constraints is one such that $\{\mathbf{x} \mid \mathbf{A}\mathbf{x} = \mathbf{b}\}$, with $\mathbf{A} \in \mathbb{R}^{m \times p}$ and $\mathbf{b} \in \mathbb{R}^m$, where m is the number of constraints. It is important to note that for an optimization variable in $\mathbb{R}^p$ the number of total affine constraints, m, can be at most equal to p. That is because the intersection of the set of constraints represented by the rows of $\mathbf{A}$, $\{\mathbf{x} \mid \mathbf{a}_i^\top \mathbf{x} = \mathbf{b}_i\}$, would be empty, given that all rows/constraints are distinct.

We generate the rows of the constraint matrix $\mathbf{A}$ as $\mathbf{a}_i \sim \mathcal{N}(\mathbf{0}, I_p)$. Then, using the optimal value $\mathbf{x}_\star$ (common to all agents, and hidden), we set $\mathbf{b} = \mathbf{A}\mathbf{x}^{\text{init}}$. Since the rows of $\mathbf{A}$ are independent, the intersection of all constraints is non-empty with probability 1. In the case of local constraints, we repeat this scheme for each agent separately.

We compare the centralized Projected Gradient Descent method (Algorithm 5), with the Distributed Projected Gradient Descent (Algorithm 12) and PG-EXTRA (Algorithm 13). All three algorithms make use of the projection operator onto the affine set after taking the gradient descent step. For affine sets the orthogonal projection can be calculated by [Bec17]:

$$P_{\{\mathbf{x} \mid \mathbf{A}\mathbf{x} - \mathbf{b}\}}(\mathbf{x}) = \mathbf{x} - \mathbf{A}^\top \left(\mathbf{A}\mathbf{A}^\top\right)^{-1} (\mathbf{A}\mathbf{x} - \mathbf{b}) \,. \tag{7.18}$$

We first run experiments for an increasing amount of global constraints. Then, we repeat the experiments for local constraints, while keeping the rest unchanged.

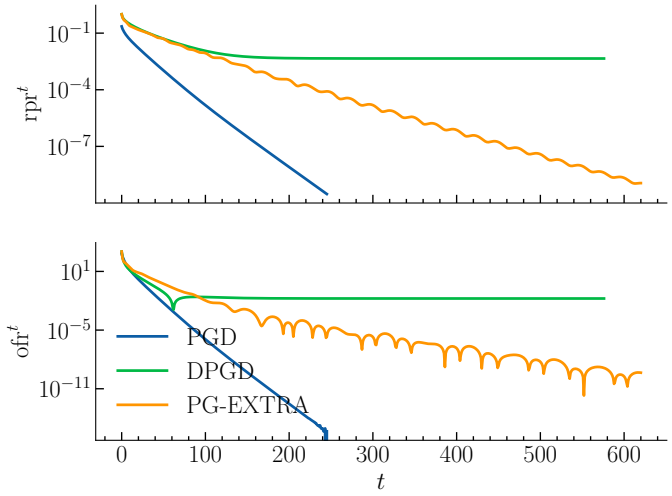
Figure 7.14 illustrates experiments for Linear Regression with *global* affine constraints. We generate data as shown in Section 6.1 ($\sigma_\epsilon = 10^{-3}$), we have $N = 10$ agents (network structure as in Section 7.1), parameter dimension of $p = 50$, $n_i = 25$ data examples per agent ($n = 250$ total), and $m_i = 10/30/50$ constraints per agent ($m = 10/30/50$ total).

Figure 7.15 depicts experiments with the same parameters as before, except here the constraints are local to each agent with $m_i = 1/3/5$ constraints per agent ($m = 10/30/50$ total).

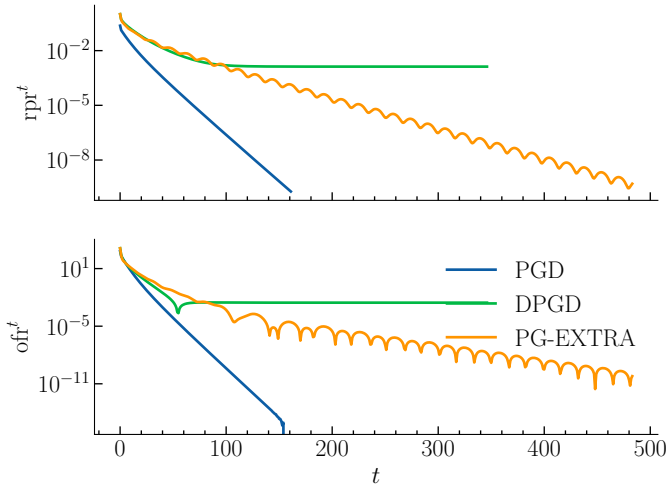
When the constraints are global DPGD, as expected does not achieve exact convergence, while PG-EXTRA does so with ease. As the number of constraints grow the problem seems to only be easier. Specifically, for the case when the number of constraints is equal to the dimension of the parameter, all algorithms require only a single iteration to find the exact solution, as only one exists.

For local constraints, the inverse seems to apply. DPGD, again does not converge, and PG-EXTRA slows down considerably as the number of total constraints grow. In fact, when the number of total constraints are equal to the dimension of $\mathbf{x}$ ($m = n$), PG-EXTRA reaches only a neighborhood of the solution, although, with higher accuracy than DPGD.

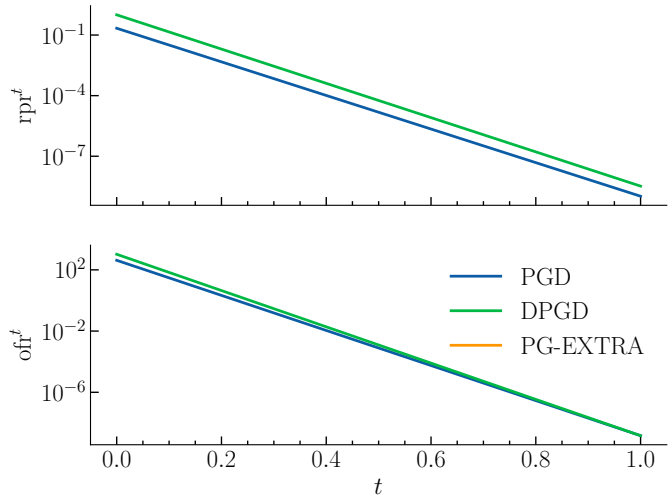
It is much harder to achieve consensus, and thus convergence to the solution, when the agents each have many private constraints.



(a) $m = 10$ global constraints

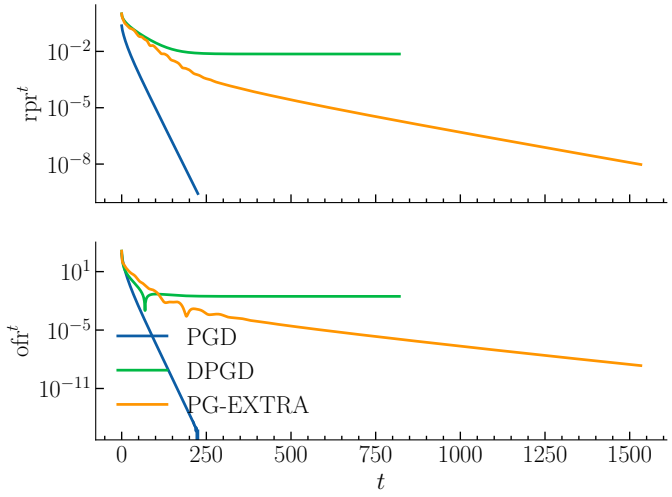


(b) $m = 30$ global constraints

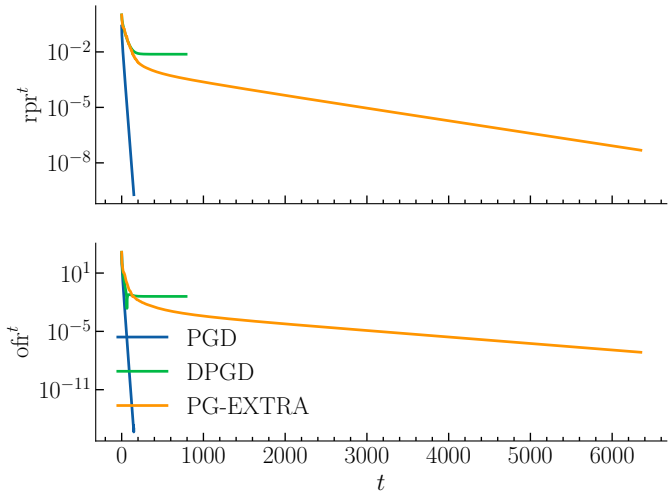


(c) $m = 50$ global constraints

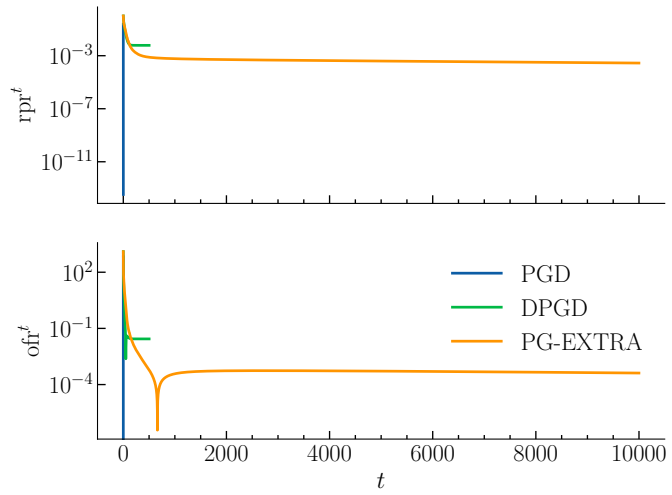
Figure 7.14: PGD vs DPGD vs PG-EXTRA for Linear Regression with global affine constraints. $N = 10$, $p = 50$, $n_i = 25$ ($n = 250$), $\sigma_\epsilon = 10^{-3}$.



(a) $m_i = 1$ local constraints ($m = 10$)



(b) $m_i = 3$ local constraints ($m = 30$)



(c) $m_i = 5$ local constraints ($m = 50$)

Figure 7.15: PGD vs DPGD vs PG-EXTRA for Linear Regression with local affine constraints. $N = 10$, $p = 50$, $n_i = 25$ ($n = 250$), $\sigma_\epsilon = 10^{-3}$.

Conclusions and Future Work

In this thesis, we initially focused on demonstrating how the combination of gradient descent methods with the consensus algorithm leads to decentralized optimization algorithms, capable of solving distributed optimization problems. The simple Decentralized Gradient Descent method is not capable of finding the exact solution to the global problem, instead converging to a neighborhood of the solution. Thus, we examined an exact first order method, which can alleviate this shortcoming. We provided an intuitive explanation for how the method works and why it is indeed capable of finding the exact solution to an optimization problem in a decentralized manner.

Next, we took a look at dual algorithms for decentralized optimization. Although computation of the dual gradient is expensive, especially since it is required at every step of the algorithm, the single-step dual algorithm provides a near optimal rate of convergence. By performing multiple consensus communication rounds in between each dual gradient computation, and employing Chebysev Acceleration on the mixing matrix used by the consensus mechanism, the Multi-Step Dual algorithm can achieve the optimal rates for any first order decentralized optimization algorithm. Obviously, the communication cost is high, thus requiring the communication time to be considerably shorter than the time needed for the computation of the dual gradient.

Finally, we shifted to distributed constrained optimization and composite optimization problems. These problems can be solved via similar methods in both the centralized and decentralized case. Thus, we examined two decentralized algorithms: Distributed Projected Gradient Descent, which is the decentralized version of centralized Projected Gradient Descent, and PG-EXTRA, which is an adaptation of the exact first-order method, EXTRA, for composite (and by extension constrained) distributed problems.

We experimentally compared a handful of centralized algorithms to the distributed algorithms mentioned previously. We performed separate experiments for functions of different classes, namely non-smooth, smooth, and strongly convex functions, as for each class different optimization algorithms are suitable. The claims of inexact convergence for the simple Distributed Gradient Descent method were supported by multiple experiments, for all functions classes. The algorithm's convergence plateaued early on, reaching only a neighborhood of the solution, with the size of that neighborhood being larger for harder problems, as expected.

The exact first order algorithm EXTRA on the other hand readily solves smooth convex problems. The smoothness assumption allows the algorithm to use a large and constant stepsize inversely proportional to the smoothness parameter of the global objective funciton, allowing for fast and exact convergence to the optimum.

For smooth and strongly convex problems we tested the performance of the dual algorithm SSDA, where its eficaccy was shown, especially in harder problems (i.e., problems with larger condition number). For such problems, it surpassed not only the decentralized algorithm EXTRA but also centralized Gradient Descent.

Later, comparing the single-step dual algorithm to its multi-step version, we showed the possible performance gains to be had if the computation of the gradients are expensive. Since with many communication rounds in between each dual computation the multi-step algorithm surpasses its single-step counterpart, requiring much fewer gradient computations.

Finally, we performed experiments for the solution of distributed constrained optimization problems, for both global and local constraints (i.e., private constraints per agents). The Distributed Projected Gradient Descent method, once again, did not achieve exact convergence, except for the trivial case where the number of constraints equaled the parameter's dimension (there exists only one solution that is easy to find via projection alone). The exact proximal method, PG-EXTRA, although being able to solve problems with global constraints with ease, also started to plateau for problems with many local constraints. This case of distributed problem is especially hard since each agent performs a separate projection to its own constraints, making consensus harder.

While this thesis focused on decentralized optimization algorithms for time-invariant networks, a logical next step is to investigate algorithms that can solve distributed optimization problems where the underlying network is time-varying. That is, not all agents are guaranteed to communicate at each time step, or the communications and computations happen asynchronously. These assumptions are far more realistic, requiring different algorithms that can be applied with greater ease, in practice.

Another promising direction for future work are accelerated decentralized methods that lessen the amount of communications required, which can be a potential bottleneck as ML models grow larger. Algorithms that do not perform multiple communication rounds per gradient computation would potentially allow for much larger ML models to be trained over a network.

Similarly, methods that incorporate quantization during communication pose an interesting challenge of ensuring exact convergence while communications are lossy, albeit with some regularity. Following the theme of large ML models, it is most common to use stochastic gradients to train such models in the centralized case (or in master-slave environments, which are distributed bot not decentralized). Decentralized Stochastic Gradient Descent methods are, as such, of great interest. Combining stochastic methods, with quantized communications, acceleration via momentum, for time-varying networks, if possible could provide an algorithm that can be used to train modern ML models in a realistic setting without the burden of expensive communications.

Lastly, all distributed optimization algorithms can, potentially, suffer from information leakage. That is, some information about the private data that an agent uses to solve their local optimization problem, can be gathered by another malicious agent in the system. Gradient inversion attacks, their implications on the security of such systems, and more, are being investigated by the field of Difference Privacy. As such, decentralized algorithms that can ensure that their original goal, of training ML models without sharing data directly, is achieved securely, are of great significance.

Bibliography

[Deg74] Morris H Degroot. «Reaching a Consensus». In: *J. Am. Stat. Assoc.* 69.345 (Mar. 1974), pp. 118–121.

[Pol87] Boris Teodorovich Poliak. *Introduction to optimization*. en. 1987.

[Roc97] Ralph Tyrell Rockafellar. *Convex Analysis*. Princeton Landmarks in Mathematics and Physics. Princeton, NJ: Princeton University Press, Jan. 1997.

[BV04] Stephen Boyd and Lieven Vandenberghe. *Convex Optimization*. Cambridge, England: Cambridge University Press, Mar. 2004.

[Law06] Gregory F Lawler. *Introduction to stochastic processes*. 2nd ed. Chapman & Hall/CRC Probability Series. Philadelphia, PA: Chapman & Hall/CRC, May 2006.

[XBK07] Lin Xiao, Stephen Boyd, and Seung-Jean Kim. «Distributed average consensus with least-mean-square deviation». In: *Journal of Parallel and Distributed Computing* 67.1 (Jan. 2007), pp. 33–46. ISSN: 0743-7315. DOI: 10.1016/j.jpdc.2006.08.010. URL: http://dx.doi.org/10.1016/j.jpdc.2006.08.010.

[NO09] Angelia Nedic and Asuman Ozdaglar. «Distributed Subgradient Methods for Multi-Agent Optimization». In: *IEEE Transactions on Automatic Control* 54.1 (Jan. 2009), pp. 48–61. ISSN: 0018-9286. DOI: 10.1109/tac.2008.2009515. URL: http://dx.doi.org/10.1109/TAC.2008.2009515.

[Che12] Annie I-An Chen. «Fast distributed first-order methods». PhD thesis. Massachusetts Institute of Technology, 2012.

[Shi+15a] Wei Shi et al. «A Proximal Gradient Algorithm for Decentralized Composite Optimization». In: *IEEE Transactions on Signal Processing* 63.22 (Nov. 2015), pp. 6013–6023. ISSN: 1941-0476. DOI: 10.1109/tsp.2015.2461520. URL: http://dx.doi.org/10.1109/TSP.2015.2461520.

[Shi+15b] Wei Shi et al. «EXTRA: An Exact First-Order Algorithm for Decentralized Consensus Optimization». In: *SIAM Journal on Optimization* 25.2 (Jan. 2015), pp. 944–966. ISSN: 1095-7189. DOI: 10.1137/14096668x. URL: http://dx.doi.org/10.1137/14096668X.

[DB16] Steven Diamond and Stephen Boyd. «CVXPY: A Python-Embedded Modeling Language for Convex Optimization». In: *Journal of Machine Learning Research* (2016). To appear. URL: https://stanford.edu/~boyd/papers/pdf/cvxpy_paper.pdf.

[YLY16] Kun Yuan, Qing Ling, and Wotao Yin. «On the Convergence of Decentralized Gradient Descent». In: *SIAM Journal on Optimization* 26.3 (Jan. 2016), pp. 1835–1854. ISSN: 1095-7189. DOI: 10.1137/130943170. URL: http://dx.doi.org/10.1137/130943170.

[Bec17] Amir Beck. *First-Order Methods in Optimization*. Society for Industrial and Applied Mathematics, Oct. 2017. ISBN: 9781611974997. DOI: 10.1137/1.9781611974997. URL: http://dx.doi.org/10.1137/1.9781611974997.

[Sca+17] Kevin Scaman et al. «Optimal Algorithms for Smooth and Strongly Convex Distributed Optimization in Networks». In: *Proceedings of the 34th International Conference on Machine Learning*. Ed. by Doina Precup and Yee Whye Teh. Vol. 70. Proceedings of Machine Learning Research. PMLR, Aug. 2017, pp. 3027–3036. URL: https://proceedings.mlr.press/v70/scaman17a.html.

[Ned20] Angelia Nedic. «Distributed Gradient Methods for Convex Machine Learning Problems in Networks: Distributed Optimization». In: *IEEE Signal Processing Magazine* 37.3 (May 2020), pp. 92–101. ISSN: 1558-0792. DOI: 10.1109/msp.2020.2975210. URL: http://dx.doi.org/10.1109/MSP.2020.2975210.