

5OD14. Cooperative Optimization and Learning

Andrea Simonetto

Academic year 2025/2026

General info

- 5 classes (2h class, 1h30 project) [The last class, but only the class, it is possible that is done remotely!]
- All the info are on moodle
- 2h written Exam: 24/03, with an A4 paper (recto-verso) of your hand-written notes.
- Grade: 40% written exam, 40% python project, 20% literature study
- Deadline project and literature study (March 31, 13H00 Paris Time)
- python project and literature study in groups of three that we make today
- Contacts

Andrea SIMONETTO

andrea.simonetto@ensta.fr,

Romain POUJOL

romain.poujol@ensta.fr

Content

- ➊ Introduction [1]
- ➋ Basics [1]
- ➌ Distributed Optimization, primal methods [1]
- ➍ Consensus [2]
- ➎ Distributed Optimization, dual methods [2,3]
- ➏ Intermezzo: Stochastic gradient [3]
- ➐ Federated Learning [4]
- ➑ Differential Privacy [5]

Material: lecture notes, suggested references to books/articles as we go along.

!! study the lecture notes !!

Remarks

- This is probably the last course you'll do in school;
- This is **an advanced course**, for 3rd year students (no solving exercises on the board, no explaining you everything): **!! study the lecture notes !! do your own exercises therein !!**
- This course requires a good deal of self-work
- It'll make you see some leading-edge techniques that AI companies use today

!! study the lecture notes !!

Part I

Introduction

Some recap of useful notions

- We look at continuous optimization problems (like in OPT201, OPT202), i.e.,

$$\min_{\mathbf{x} \in X \subseteq \mathbf{R}^n} f(\mathbf{x}),$$

where the set X is closed and convex, $f : \mathbf{R}^n \rightarrow \mathbf{R}$ is a convex function and the minimum is attained for a $\mathbf{x} \in X$.

Some recap of useful notions

- We look at continuous optimization problems (like in OPT201, OPT202), i.e.,

$$\min_{\mathbf{x} \in X \subseteq \mathbf{R}^n} f(\mathbf{x}),$$

where the set X is closed and convex, $f : \mathbf{R}^n \rightarrow \mathbf{R}$ is a convex function and the minimum is attained for a $\mathbf{x} \in X$.

- What you need to remember from past courses will be reviewed as we go along, but mainly:
 - ▶ Optimality conditions (KKT), which are here necessary and sufficient (under a Slater's constraint qualification assumption)
 - ▶ How to derive dual problems (Lagrangian function, dual function, problems)
 - ▶ Subgradients
 - ▶ Algorithms: How to derive first-order algorithms (e.g., gradient descent) from the optimality conditions, and how to prove their convergence and convergence rate
 - ▶ Some basic linear algebra: eigenvalues, singular value decomposition, solution of linear systems, etc..
 - ▶ [Check out the lecture notes of OPT202](#)

The optimization problem of interest

- Said so, the problem we will look at in this course has the form,

$$(P) \quad \min_{\mathbf{x} \in X \subseteq \mathbf{R}^n} \sum_{i=1}^N f_i(\mathbf{x}), \quad (1)$$

where the set X is closed and convex, $f_i : \mathbf{R}^n \rightarrow \mathbf{R}$ is a convex function for all i 's and the minimum is attained for a $\mathbf{x} \in X$.

Here N is the number of agents/players/nodes/sub-systems/etc.. that **cooperate** to solve the optimization problem.

Examples and applications

- **Cooperative least-squares.** Imagine N users collect noisy measurements $\mathbf{y}_i \in \mathbf{R}^n$ about a quantity $\mathbf{x} \in X$. A way to estimate the true value for \mathbf{x} is to set up a least-squares problem as

$$\min_{\mathbf{x} \in X} \sum_{i=1}^N \|\mathbf{x} - \mathbf{y}_i\|^2 = \sum_{i=1}^N f_i(\mathbf{x})$$

Examples and applications

- **Cooperative least-squares.** Imagine N users collect noisy measurements $\mathbf{y}_i \in \mathbf{R}^n$ about a quantity $\mathbf{x} \in X$. A way to estimate the true value for \mathbf{x} is to set up a least-squares problem as

$$\min_{\mathbf{x} \in X} \sum_{i=1}^N \|\mathbf{x} - \mathbf{y}_i\|^2 = \sum_{i=1}^N f_i(\mathbf{x})$$

- **Cooperative linear model training.** Imagine N users collect input-output pairs $\mathbf{w}_i \in \mathbf{R}^{n-1}$, $\mathbf{y}_i \in \mathbf{R}$ (e.g., features and labels), and they want to train a global model with weights $\mathbf{x} \in X \subseteq \mathbf{R}^n$, $\mathbf{x} = [\theta \in \mathbf{R}^{n-1}, c \in \mathbf{R}]$. Let the local input-output mapping be affine,

$$\mathbf{y}_i = \theta^\top \mathbf{w}_i + c, \quad \forall i,$$

then the training problem can be written as

$$\min_{\mathbf{x} \in X} \sum_{i=1}^N \|\mathbf{y}_i - (\theta^\top \mathbf{w}_i + c)\|^2 = \sum_{i=1}^N f_i(\mathbf{x})$$

Examples and applications

- **Cooperative network problems.** Take a sensor network of N sensors. You want to compute the localization of the whole network based on pair-wise distance measurements (e.g., sensors can be cars, or people with their phones). Then each measurement is

$$m_{ij} = \|\mathbf{x}_i - \mathbf{x}_j\| + \text{noise}, \quad \mathbf{x}_i \in \mathbf{R}^2 \text{ is the position of node } i.$$

You have E pair-wise measurements. Then you can write the problem as

$$\min_{\mathbf{x} \in X \subset \mathbf{R}^{2E}} \sum_{i,j}^E (m_{ij} - \|\mathbf{x}_i - \mathbf{x}_j\|)^2 = \sum_{k=1}^E f_k(\mathbf{x})$$

Examples and applications

- **Cooperative network problems.** Take a sensor network of N sensors. You want to compute the localization of the whole network based on pair-wise distance measurements (e.g., sensors can be cars, or people with their phones). Then each measurement is

$$m_{ij} = \|\mathbf{x}_i - \mathbf{x}_j\| + \text{noise}, \quad \mathbf{x}_i \in \mathbf{R}^2 \text{ is the position of node } i.$$

You have E pair-wise measurements. Then you can write the problem as

$$\min_{\mathbf{x} \in \mathbf{X} \subset \mathbf{R}^{2E}} \sum_{i,j}^E (m_{ij} - \|\mathbf{x}_i - \mathbf{x}_j\|)^2 = \sum_{k=1}^E f_k(\mathbf{x})$$

- **Cooperative consensus.** You have N robots at different locations \mathbf{s}_i and moving at different speeds v_i , and you want to find the best position in space for the fastest rendez-vous:

$$\min_{\mathbf{x} \in \mathbf{X} \subset \mathbf{R}^3} \sum_i^N \frac{\|\mathbf{s}_i - \mathbf{x}\|}{v_i} = \sum_{i=1}^N f_i(\mathbf{x})$$

Main challenges and plan for the course

- Large-scale problems: N is big or n is big and the problem cannot be solved (nor stored) locally. You need to solve it on separate machines, or iteratively.
 - ▶ If n is big, usually we talk about parallel methods;
 - ▶ If N is big, usually we talk about distributed methods.

Main challenges and plan for the course

- Large-scale problems: N is big or n is big and the problem cannot be solved (nor stored) locally. You need to solve it on separate machines, or iteratively.
 - ▶ If n is big, usually we talk about parallel methods;
 - ▶ If N is big, usually we talk about distributed methods.
- Distributed localization: the data that builds f_i is stored in separate machines (think: mobile phones). Then, you have communication issues and latencies..
 - ▶ Are we communicating to a server? (Cloud-based)
 - ▶ Are we communicating to each other (Peer-to-peer)
 - ▶ Common issues: asynchronicity, bi-directionality, packet-losses, communication overhead,..
 - ▶ Here: graph theory and Markov chains will be mixed with optimization

The SneakerNet paradox. What is the fastest way to send large data sets?

Main challenges and plan for the course

- Large-scale problems: N is big or n is big and the problem cannot be solved (nor stored) locally. You need to solve it on separate machines, or iteratively.
 - ▶ If n is big, usually we talk about parallel methods;
 - ▶ If N is big, usually we talk about distributed methods.
- Distributed localization: the data that builds f_i is stored in separate machines (think: mobile phones). Then, you have communication issues and latencies..
 - ▶ Are we communicating to a server? (Cloud-based)
 - ▶ Are we communicating to each other (Peer-to-peer)
 - ▶ Common issues: asynchronicity, bi-directionality, packet-losses, communication overhead,..
 - ▶ Here: graph theory and Markov chains will be mixed with optimization

The SneakerNet paradox. What is the fastest way to send large data sets?
FedEx

Main challenges and plan for the course

- Large-scale problems: N is big or n is big and the problem cannot be solved (nor stored) locally. You need to solve it on separate machines, or iteratively.
 - ▶ If n is big, usually we talk about parallel methods;
 - ▶ If N is big, usually we talk about distributed methods.
- Distributed localization: the data that builds f_i is stored in separate machines (think: mobile phones). Then, you have communication issues and latencies..
 - ▶ Are we communicating to a server? (Cloud-based)
 - ▶ Are we communicating to each other (Peer-to-peer)
 - ▶ Common issues: asynchronicity, bi-directionality, packet-losses, communication overhead,..
 - ▶ Here: graph theory and Markov chains will be mixed with optimization
- Data and functions are private: you don't want to disclose f_i or the local decision to the other players. How do you do that? (Example: training language models based on private text messages on your phone)

Main challenges and plan for the course

- In this course, we will look at algorithms to tackle the challenges above while solving problem (P)

Main challenges and plan for the course

- In this course, we will look at algorithms to tackle the challenges above while solving problem (P)
- The algorithms will be of the first-order kind (more later)

Main challenges and plan for the course

- In this course, we will look at algorithms to tackle the challenges above while solving problem (P)
- The algorithms will be of the first-order kind (more later)
- We divide the course in three main parts (distributed, federated, private)

Part II

Basics

Some definitions

Definition 1 (Convex functions)

A function $f : X \subseteq \mathbf{R}^n \rightarrow \mathbf{R}$ is convex iff X is convex and

$$(C1) \quad \forall \mathbf{x}, \mathbf{y} \in X, \lambda \in [0, 1] : \quad f(\lambda \mathbf{x} + (1 - \lambda) \mathbf{y}) \leq \lambda f(\mathbf{x}) + (1 - \lambda) f(\mathbf{y}).$$

Multiple definitions exists, for example:

$$(C1) + f \in \mathcal{C}^1(X) \quad \Longleftrightarrow \quad \forall \mathbf{x}, \mathbf{y} \in X, f(\mathbf{x}) \geq f(\mathbf{y}) + \langle \nabla f(\mathbf{y}), \mathbf{x} - \mathbf{y} \rangle$$

$$(C1) + f \in \mathcal{C}^2(X) \quad \Longleftrightarrow \quad \forall \mathbf{x}, \mathbf{y} \in X, \nabla^2 f(\mathbf{x}) \succeq 0$$

Some definitions

Definition 1 (Strongly convex functions)

A convex function $f : X \subseteq \mathbf{R}^n \rightarrow \mathbf{R}$ is m -strongly convex iff

$$(\text{SC}) \quad f(\mathbf{x}) - \frac{m}{2} \|\mathbf{x}\|^2 \text{ is convex.}$$

f doesn't need to be differentiable!

Multiple definitions exists, for example:

$$\begin{aligned} (\text{SC}) + f \in \mathcal{C}^1(X) &\iff \\ &\forall \mathbf{x}, \mathbf{y} \in X, \langle \nabla f(\mathbf{x}) - \nabla f(\mathbf{y}), \mathbf{x} - \mathbf{y} \rangle \geq m \|\mathbf{x} - \mathbf{y}\|^2 \\ (\text{SC}) + f \in \mathcal{C}^2(X) &\iff \forall \mathbf{x}, \mathbf{y} \in X, \nabla^2 f(\mathbf{x}) \succeq mI_n \end{aligned}$$

Some definitions

Definition 1 (Smooth functions)

A convex function $f : X \subseteq \mathbf{R}^n \rightarrow \mathbf{R}$ is L -smooth iff

$$(LC) \quad \frac{L}{2} \|\mathbf{x}\|^2 - f(\mathbf{x}) \text{ is convex.}$$

Important $(LC) \implies f \in \mathcal{C}^1(X)$!

Multiple definitions exists, for example:

$$(LC) \iff \forall \mathbf{x}, \mathbf{y} \in X, \|\nabla f(\mathbf{x}) - \nabla f(\mathbf{y})\| \leq L \|\mathbf{x} - \mathbf{y}\|$$

$$(LC) \iff \forall \mathbf{x}, \mathbf{y} \in X,$$

$$\frac{1}{L} \|\nabla f(\mathbf{x}) - \nabla f(\mathbf{y})\|^2 \leq \langle \nabla f(\mathbf{x}) - \nabla f(\mathbf{y}), \mathbf{x} - \mathbf{y} \rangle$$

$$(LC) + f \in \mathcal{C}^2(X) \iff \forall \mathbf{x}, \mathbf{y} \in X, 0 \preceq \nabla^2 f(\mathbf{x}) \preceq L I_n$$

Gradient algorithm

Consider solving $\min_{\mathbf{x} \in \mathbb{R}^n} f(\mathbf{x})$.

Rem: the gradient is an ascent direction, and a first-order method

Gradient algorithm

Consider solving $\min_{\mathbf{x} \in \mathbb{R}^n} f(\mathbf{x})$.

Rem: the gradient is an ascent direction, and a first-order method

The simplest scheme, let $\alpha_k > 0$:

- Start with $\mathbf{x}_0 \in \mathbb{R}^n$
- Iterate $\mathbf{x}_{k+1} = \mathbf{x}_k - \alpha_k \nabla f(\mathbf{x}_k)$, $k = 0, 1, \dots$

There are different methods to choose α_k (either a priori or online):

- Constant: $\alpha_k = \alpha$
- Vanishing: $\alpha_k = \frac{\alpha}{\sqrt{k+1}}$
- ...

Why the difference? Convergence

Gradient algorithm: convergence recap

- Recap from OPT201-202: gradient converges in a well-defined sense provided the step size α_k is chosen appropriately.
- Depending on the functional class, the gradient algorithm behaves differently with the number of iterations t .
- The table below recaps the main results that one can expect:


| Convex function type: | General | Smooth | Smooth+Strongly convex |
|-----------------------|-------------------------------|---|------------------------|
| | First-order algorithms | | |
| Convergence metric | $\ \nabla f\ , f - f^*$ | $f - f^*$ | $\ x - x^*\ $ |
| Convergence rate | $O(1/\sqrt{t})$ | $O(1/t) \rightsquigarrow O(1/t^2)$ (Nesterov's acceleration) | $O(\rho^t)$ |

Homework: Revise your theory, e.g., what does $O(\cdot)$ mean ?

Let's finally start!

- We want to solve problem (P) but N is too big. How do we do?


$$(f_1) \quad (f_2) \quad \dots \quad (f_i) \quad \dots \quad (f_N)$$

You  (P) $\min_{x \in \mathbf{R}^n} \sum_{i=1}^N f_i(x)$

Let's finally start!

- We want to solve problem (P) but N is too big. How do we do?

$$(f_1) \quad (f_2) \quad \dots \quad (f_i) \quad \dots \quad (f_N)$$

You  (P) $\min_{x \in \mathbf{R}^n} \sum_{i=1}^N f_i(x)$


- First possibility: you ask one device to apply a local gradient, then you get back the result and you ask a second device..

Incremental/ Gauss-Seidel Gradient

Let's finally start!

- We want to solve problem (P) but N is too big. How do we do?

$$(f_1) \quad (f_2) \quad \dots \quad (f_i) \quad \dots \quad (f_N)$$

You  (P) $\min_{x \in \mathbf{R}^n} \sum_{i=1}^N f_i(x)$

- First possibility: you ask one device to apply a local gradient, then you get back the result and you ask a second device..


Incremental/ Gauss-Seidel Gradient

- ▶ Start with $x_0 \in \mathbf{R}^n$

Let's finally start!

- We want to solve problem (P) but N is too big. How do we do?

$$(f_1) (f_2) \dots (f_i) \dots (f_N)$$

You  (P) $\min_{x \in \mathbf{R}^n} \sum_{i=1}^N f_i(x)$

- First possibility: you ask one device to apply a local gradient, then you get back the result and you ask a second device..


Incremental/ Gauss-Seidel Gradient

- ▶ Start with $\mathbf{x}_0 \in \mathbf{R}^n$
- ▶ Iterate: ask device i : $\mathbf{x}_{k+1} = \mathbf{x}_k - \alpha_k \nabla f_i(\mathbf{x}_k)$, $k = 0, 1, \dots$

Let's finally start!

- We want to solve problem (P) but N is too big. How do we do?

$$(f_1) (f_2) \dots (f_i) \dots (f_N)$$

You  (P) $\min_{x \in \mathbf{R}^n} \sum_{i=1}^N f_i(x)$

- First possibility: you ask one device to apply a local gradient, then you get back the result and you ask a second device..

Incremental/ Gauss-Seidel Gradient


- ▶ Start with $\mathbf{x}_0 \in \mathbf{R}^n$
- ▶ Iterate: ask device i : $\mathbf{x}_{k+1} = \mathbf{x}_k - \alpha_k \nabla f_i(\mathbf{x}_k)$, $k = 0, 1, \dots$
- Second possibility: you ask every device to apply a local gradient, then you get back the result and ??

Parallel/ Jacobi Gradient

Let's finally start!

- We want to solve problem (P) but N is too big. How do we do?

$$(f_1) (f_2) \dots (f_i) \dots (f_N)$$

You  (P) $\min_{x \in \mathbf{R}^n} \sum_{i=1}^N f_i(x)$

- First possibility: you ask one device to apply a local gradient, then you get back the result and you ask a second device..

Incremental/ Gauss-Seidel Gradient

- ▶ Start with $\mathbf{x}_0 \in \mathbf{R}^n$
- ▶ Iterate: ask device i : $\mathbf{x}_{k+1} = \mathbf{x}_k - \alpha_k \nabla f_i(\mathbf{x}_k)$, $k = 0, 1, \dots$
- Second possibility: you ask every device to apply a local gradient, then you get back the result and ??


Parallel/ Jacobi Gradient

- ▶ Start with $\mathbf{x}_0 \in \mathbf{R}^n$

Let's finally start!

- We want to solve problem (P) but N is too big. How do we do?

$$(f_1) (f_2) \dots (f_i) \dots (f_N)$$

You  (P) $\min_{x \in \mathbf{R}^n} \sum_{i=1}^N f_i(x)$

- First possibility: you ask one device to apply a local gradient, then you get back the result and you ask a second device..

Incremental/ Gauss-Seidel Gradient

- ▶ Start with $\mathbf{x}_0 \in \mathbf{R}^n$
- ▶ Iterate: ask device i : $\mathbf{x}_{k+1} = \mathbf{x}_k - \alpha_k \nabla f_i(\mathbf{x}_k)$, $k = 0, 1, \dots$
- Second possibility: you ask every device to apply a local gradient, then you get back the result and ??


Parallel/ Jacobi Gradient

- ▶ Start with $\mathbf{x}_0 \in \mathbf{R}^n$
- ▶ Iterate: ask every devices: $\mathbf{x}_{k+1}^i = \mathbf{x}_k - \alpha_k \nabla f_i(\mathbf{x}_k)$, $\forall i$

Let's finally start!

- We want to solve problem (P) but N is too big. How do we do?

$$(f_1) (f_2) \dots (f_i) \dots (f_N)$$

You  (P) $\min_{x \in \mathbf{R}^n} \sum_{i=1}^N f_i(x)$

- First possibility: you ask one device to apply a local gradient, then you get back the result and you ask a second device..

Incremental/ Gauss-Seidel Gradient

- ▶ Start with $\mathbf{x}_0 \in \mathbf{R}^n$
- ▶ Iterate: ask device i : $\mathbf{x}_{k+1} = \mathbf{x}_k - \alpha_k \nabla f_i(\mathbf{x}_k)$, $k = 0, 1, \dots$
- Second possibility: you ask every device to apply a local gradient, then you get back the result and ??

Parallel/ Jacobi Gradient

- ▶ Start with $\mathbf{x}_0 \in \mathbf{R}^n$
- ▶ Iterate: ask every devices: $\mathbf{x}_{k+1}^i = \mathbf{x}_k - \alpha_k \nabla f_i(\mathbf{x}_k)$, $\forall i$
- ▶ $\mathbf{x}_{k+1} = ?$

Properties and convergence of incremental gradient

- The data that generates f_i stays private, and if you have enough time you don't care about latencies (etc..): the scheme is quite robust

Properties and convergence of incremental gradient

- The data that generates f_i stays private, and if you have enough time you don't care about latencies (etc..): the scheme is quite robust
- Also known in the ML community as: online back-propagation, finite sum stochastic gradient descent (careful: step size is called the learning rate)

Theorem 2

Consider problem (P) for a m -strongly convex and L -smooth function f . Consider the incremental gradient method with step size α_k . Assume that $\|\nabla f_i(\mathbf{x}) - \sum_i \nabla f_i(\mathbf{x})\| \leq G$ for all $\mathbf{x} \in \mathbf{R}^n$, choose $\alpha_k < 2/L$, and define the quantity, $\rho_k = \max\{|1 - \alpha_k m|, |1 - \alpha_k L|\} < 1$.

Then convergence of the incremental gradient goes as

$$\|\mathbf{x}_{k+1} - \mathbf{x}^*\| \leq \rho_k \|\mathbf{x}_k - \mathbf{x}^*\| + \alpha_k G.$$

Corollary 3

If $\alpha_k = 1/k^s$, $0 < s < 1$ then, $\|\mathbf{x}_k - \mathbf{x}^*\| \leq O(1/k^s)$.

Proof

- Define $f(\mathbf{x}) = \sum_i f_i(\mathbf{x})$. Write

$$\|\mathbf{x}_{k+1} - \mathbf{x}^*\| \leq \|\mathbf{x}_k - \alpha_k \nabla f(\mathbf{x}_k) - \mathbf{x}^* + \alpha_k \nabla f(\mathbf{x}^*)\| + \alpha_k \|\nabla f_i(\mathbf{x}_k) - \nabla f(\mathbf{x}_k)\|$$

Proof

- Define $f(\mathbf{x}) = \sum_i f_i(\mathbf{x})$. Write

$$\|\mathbf{x}_{k+1} - \mathbf{x}^*\| \leq \|\mathbf{x}_k - \alpha_k \nabla f(\mathbf{x}_k) - \mathbf{x}^* + \alpha_k \nabla f(\mathbf{x}^*)\| + \alpha_k \|\nabla f_i(\mathbf{x}_k) - \nabla f(\mathbf{x}_k)\|$$

- Use strongly convex and smoothness property to say (check OPT202)

$$\|\mathbf{x}_k - \alpha_k \nabla f(\mathbf{x}_k) - \mathbf{x}^* + \alpha_k \nabla f(\mathbf{x}^*)\| \leq \rho_k \|\mathbf{x}_k - \mathbf{x}^*\|$$

Proof

- Define $f(\mathbf{x}) = \sum_i f_i(\mathbf{x})$. Write

$$\|\mathbf{x}_{k+1} - \mathbf{x}^*\| \leq \|\mathbf{x}_k - \alpha_k \nabla f(\mathbf{x}_k) - \mathbf{x}^* + \alpha_k \nabla f(\mathbf{x}^*)\| + \alpha_k \|\nabla f_i(\mathbf{x}_k) - \nabla f(\mathbf{x}_k)\|$$

- Use strongly convex and smoothness property to say (check OPT202)

$$\|\mathbf{x}_k - \alpha_k \nabla f(\mathbf{x}_k) - \mathbf{x}^* + \alpha_k \nabla f(\mathbf{x}^*)\| \leq \rho_k \|\mathbf{x}_k - \mathbf{x}^*\|$$

- The thesis follows. The corollary is a bit more involved but not impossible, see for example: [arXiv:1510.08562](https://arxiv.org/abs/1510.08562)



Proof

- Define $f(\mathbf{x}) = \sum_i f_i(\mathbf{x})$. Write

$$\|\mathbf{x}_{k+1} - \mathbf{x}^*\| \leq \|\mathbf{x}_k - \alpha_k \nabla f(\mathbf{x}_k) - \mathbf{x}^* + \alpha_k \nabla f(\mathbf{x}^*)\| + \alpha_k \|\nabla f_i(\mathbf{x}_k) - \nabla f(\mathbf{x}_k)\|$$

- Use strongly convex and smoothness property to say (check OPT202)

$$\|\mathbf{x}_k - \alpha_k \nabla f(\mathbf{x}_k) - \mathbf{x}^* + \alpha_k \nabla f(\mathbf{x}^*)\| \leq \rho_k \|\mathbf{x}_k - \mathbf{x}^*\|$$

- The thesis follows. The corollary is a bit more involved but not impossible, see for example: [arXiv:1510.08562](https://arxiv.org/abs/1510.08562)

□

- A lot of research is devoted in lifting the assumptions (strong convexity, smoothness), and relaxing the assumption on G , but the basic ideas are still valid: you have a trade-off between speed and error.

Convergence of parallel gradient?

- It seems that it could be a faster algorithm, but how can we combine different updates \mathbf{x}_{k+1}^i ? Can we just average them ?

Convergence of parallel gradient?

- It seems that it could be a faster algorithm, but how can we combine different updates \mathbf{x}_{k+1}^i ? Can we just average them ?
- The scheme is less robust to asynchronicity, package drops, etc..

Convergence of parallel gradient?

- It seems that it could be a faster algorithm, but how can we combine different updates \mathbf{x}_{k+1}^i ? Can we just average them ?
- The scheme is less robust to asynchronicity, package drops, etc..
- The scheme is the starting point of distributed optimization (next!) and federated learning (later!)

Part III

Distributed optimization

Let's go Graph theory!

- We will need some basic graph theory to devise methods to combine different updates in the parallel scheme and analyze communication issues

Let's go Graph theory!

- We will need some basic graph theory to devise methods to combine different updates in the parallel scheme and analyze communication issues
- Consider a graph \mathcal{G} as a collection of vertices \mathcal{V} , edges \mathcal{E} , and edge weights \mathcal{W} .

Let's go Graph theory!

- We will need some basic graph theory to devise methods to combine different updates in the parallel scheme and analyze communication issues
- Consider a graph \mathcal{G} as a collection of vertices \mathcal{V} , edges \mathcal{E} , and edge weights \mathcal{W} .
- We define undirected a graph for which an edge can carry information in both directions. Otherwise we call it directed.

Let's go Graph theory!

- We will need some basic graph theory to devise methods to combine different updates in the parallel scheme and analyze communication issues
- Consider a graph \mathcal{G} as a collection of vertices \mathcal{V} , edges \mathcal{E} , and edge weights \mathcal{W} .
- We define undirected a graph for which an edge can carry information in both directions. Otherwise we call it directed.
- We define the Adjacency matrix as $\mathcal{A} \in \{0, 1\}^{|\mathcal{V}| \times |\mathcal{V}|}$, with 1 entries if node i and j share an edge.

Let's go Graph theory!

- We will need some basic graph theory to devise methods to combine different updates in the parallel scheme and analyze communication issues
- Consider a graph \mathcal{G} as a collection of vertices \mathcal{V} , edges \mathcal{E} , and edge weights \mathcal{W} .
- We define undirected a graph for which an edge can carry information in both directions. Otherwise we call it directed.
- We define the Adjacency matrix as $\mathcal{A} \in \{0, 1\}^{|\mathcal{V}| \times |\mathcal{V}|}$, with 1 entries if node i and j share an edge.
- The Laplacian matrix of a undirected graph \mathcal{G} is an $|\mathcal{V}| \times |\mathcal{V}|$ symmetric matrix defined by

$$\mathcal{L}_{\mathcal{G}} = \mathcal{D} - \mathcal{A},$$

where $\mathcal{D} = \text{diag}(d_1, \dots, d_n)$ is the degree matrix, which is the diagonal matrix formed from the vertex degrees.

Let's go Graph theory!

- We will need some basic graph theory to devise methods to combine different updates in the parallel scheme and analyze communication issues
- Consider a graph \mathcal{G} as a collection of vertices \mathcal{V} , edges \mathcal{E} , and edge weights \mathcal{W} .
- We define undirected a graph for which an edge can carry information in both directions. Otherwise we call it directed.
- We define the Adjacency matrix as $\mathcal{A} \in \{0, 1\}^{|\mathcal{V}| \times |\mathcal{V}|}$, with 1 entries if node i and j share an edge.
- The Laplacian matrix of a undirected graph \mathcal{G} is an $|\mathcal{V}| \times |\mathcal{V}|$ symmetric matrix defined by

$$\mathcal{L}_{\mathcal{G}} = \mathcal{D} - \mathcal{A},$$

where $\mathcal{D} = \text{diag}(d_1, \dots, d_n)$ is the degree matrix, which is the diagonal matrix formed from the vertex degrees.

- The vertex degree d_i is the number of neighbors the node i has. Therefore, $\mathcal{L}_{\mathcal{G}} \mathbf{1} = \mathbf{0}$.

Let's go Graph theory!

- We will need some basic graph theory to devise methods to combine different updates in the parallel scheme and analyze communication issues
- Consider a graph \mathcal{G} as a collection of vertices \mathcal{V} , edges \mathcal{E} , and edge weights \mathcal{W} .
- We define undirected a graph for which an edge can carry information in both directions. Otherwise we call it directed.
- We define the Adjacency matrix as $\mathcal{A} \in \{0, 1\}^{|\mathcal{V}| \times |\mathcal{V}|}$, with 1 entries if node i and j share an edge.
- The Laplacian matrix of a undirected graph \mathcal{G} is an $|\mathcal{V}| \times |\mathcal{V}|$ symmetric matrix defined by

$$\mathcal{L}_{\mathcal{G}} = \mathcal{D} - \mathcal{A},$$

where $\mathcal{D} = \text{diag}(d_1, \dots, d_n)$ is the degree matrix, which is the diagonal matrix formed from the vertex degrees.

- The vertex degree d_i is the number of neighbors the node i has. Therefore, $\mathcal{L}_{\mathcal{G}} \mathbf{1} = \mathbf{0}$.
- We define with N_i the set of nodes that node i communicates to (its neighborhood).

Let's go Graph theory!

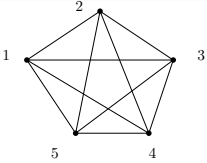
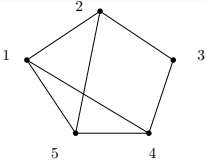
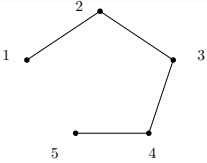
- We will need some basic graph theory to devise methods to combine different updates in the parallel scheme and analyze communication issues
- Consider a graph \mathcal{G} as a collection of vertices \mathcal{V} , edges \mathcal{E} , and edge weights \mathcal{W} .
- We define undirected a graph for which an edge can carry information in both directions. Otherwise we call it directed.
- We define the Adjacency matrix as $\mathcal{A} \in \{0, 1\}^{|\mathcal{V}| \times |\mathcal{V}|}$, with 1 entries if node i and j share an edge.
- The Laplacian matrix of a undirected graph \mathcal{G} is an $|\mathcal{V}| \times |\mathcal{V}|$ symmetric matrix defined by

$$\mathcal{L}_{\mathcal{G}} = \mathcal{D} - \mathcal{A},$$

where $\mathcal{D} = \text{diag}(d_1, \dots, d_n)$ is the degree matrix, which is the diagonal matrix formed from the vertex degrees.

- The vertex degree d_i is the number of neighbors the node i has. Therefore, $\mathcal{L}_{\mathcal{G}} \mathbf{1} = \mathbf{0}$.
- We define with N_i the set of nodes that node i communicates to (its neighborhood).
- A doubly stochastic matrix W is a matrix for which, each entry is non-negative, and $W\mathbf{1} = \mathbf{1}$, and $\mathbf{1}^T W = \mathbf{1}^T$.

Examples

| Graph example | Name | \mathcal{L}_G |
|---|-----------------|---|
|  | full graph | $\mathcal{L}_G = \begin{bmatrix} 4 & -1 & -1 & -1 & -1 \\ -1 & 4 & -1 & -1 & -1 \\ -1 & -1 & 4 & -1 & -1 \\ -1 & -1 & -1 & 4 & -1 \\ -1 & -1 & -1 & -1 & 4 \end{bmatrix}$ |
|  | a generic graph | $\mathcal{L}_G = \begin{bmatrix} 3 & -1 & 0 & -1 & -1 \\ -1 & 3 & -1 & 0 & -1 \\ 0 & -1 & 2 & -1 & 0 \\ -1 & 0 & -1 & 3 & -1 \\ -1 & -1 & 0 & -1 & 3 \end{bmatrix}$ |
|  | line graph | $\mathcal{L}_G = \begin{bmatrix} 1 & -1 & 0 & 0 & 0 \\ -1 & 2 & -1 & 0 & 0 \\ 0 & -1 & 2 & -1 & 0 \\ 0 & 0 & -1 & 2 & -1 \\ 0 & 0 & 0 & -1 & 1 \end{bmatrix}$ |

Decentralized Gradient Descent (DGD)

- We are ready to get back to our parallel update method. Consider the following update:

Decentralized Gradient Descent (DGD)

- We are ready to get back to our parallel update method. Consider the following update:

Decentralized Gradient Descent

- ▶ Start with $\mathbf{x}_0^i \in \mathbf{R}^n$ at each device i
- ▶ Iterate: ask every devices: $\mathbf{x}_{k+1}^i = \sum_{j=1}^N w_{ij} \mathbf{x}_k^j - \alpha_k \nabla f_i(\mathbf{x}_k^i), \quad \forall i$

Here $w_{ij} = 0$ if device i and j are not connected.

Decentralized Gradient Descent (DGD)

- We are ready to get back to our parallel update method. Consider the following update:

Decentralized Gradient Descent

- ▶ Start with $\mathbf{x}_0^i \in \mathbf{R}^n$ at each device i
- ▶ Iterate: ask every devices: $\mathbf{x}_{k+1}^i = \sum_{j=1}^N w_{ij} \mathbf{x}_k^j - \alpha_k \nabla f_i(\mathbf{x}_k^i), \quad \forall i$

Here $w_{ij} = 0$ if device i and j are not connected.

- Effectively we are giving a copy of \mathbf{x} to each device and looking at the iterates $\mathbf{y} = [\mathbf{x}^1; \mathbf{x}^2; \dots; \mathbf{x}^N]$

$$\mathbf{y}_{k+1} = \mathbf{W} \mathbf{y}_k - \alpha_k \nabla_{\mathbf{y}} F(\mathbf{y}_k), \quad F(\mathbf{y}) := \sum_{i=1}^N f_i(\mathbf{x}^i),$$

where matrix $\mathbf{W} = [w_{ij}]$ is the collection of the weights, it is assumed doubly stochastic and it is the mixing matrix. And $\mathbf{W} = \mathbf{W} \otimes \mathbf{I}_n$

Decentralized Gradient Descent (DGD)

Recall.

$$W = \begin{bmatrix} w_{11} & w_{12} & \cdots & w_{1N} \\ w_{12} & w_{22} & \cdots & \vdots \\ \vdots & \vdots & w_{ij} & \vdots \\ \cdots & \cdots & \cdots & \cdots \end{bmatrix}$$

And Kroenecker product:

$$\mathbf{W} = W \otimes I_n = \begin{bmatrix} w_{11}I_n & \cdots & w_{1N}I_n \\ \vdots & \ddots & \vdots \\ w_{1N}I_n & \cdots & w_{NN}I_n \end{bmatrix} \in \mathbf{R}^{Nn \times Nn}.$$

Decentralized Gradient Descent (DGD)

Recall.

$$W = \begin{bmatrix} w_{11} & w_{12} & \cdots & w_{1N} \\ w_{12} & w_{22} & \cdots & \vdots \\ \vdots & \vdots & w_{ij} & \vdots \\ \cdots & \cdots & \cdots & \cdots \end{bmatrix}$$

And Kroenecker product:

$$\mathbf{W} = W \otimes I_n = \begin{bmatrix} w_{11}I_n & \cdots & w_{1N}I_n \\ \vdots & \ddots & \vdots \\ w_{1N}I_n & \cdots & w_{NN}I_n \end{bmatrix} \in \mathbf{R}^{Nn \times Nn}.$$

- DGD is peer-to-peer:

- ▶ Each device gets and sends $\mathbf{x}_k^j / \mathbf{x}_k^i$ from/to the neighbors
- ▶ Each device computes: $\mathbf{x}_{k+1}^i = \sum_{j=1}^N w_{ij} \mathbf{x}_k^j - \alpha_k \nabla f_i(\mathbf{x}_k^i), \quad \forall i$

Decentralized Gradient Descent (DGD)

Recall.

$$W = \begin{bmatrix} w_{11} & w_{12} & \cdots & w_{1N} \\ w_{12} & w_{22} & \cdots & \vdots \\ \vdots & \vdots & w_{ij} & \vdots \\ \cdots & \cdots & \cdots & \cdots \end{bmatrix}$$

And Kroenecker product:

$$\mathbf{W} = W \otimes I_n = \begin{bmatrix} w_{11}I_n & \cdots & w_{1N}I_n \\ \vdots & \ddots & \vdots \\ w_{1N}I_n & \cdots & w_{NN}I_n \end{bmatrix} \in \mathbf{R}^{Nn \times Nn}.$$

- DGD is peer-to-peer:
 - ▶ Each device gets and sends $\mathbf{x}_k^j / \mathbf{x}_k^i$ from/to the neighbors
 - ▶ Each device computes: $\mathbf{x}_{k+1}^i = \sum_{j=1}^N w_{ij} \mathbf{x}_k^j - \alpha_k \nabla f_i(\mathbf{x}_k^i), \quad \forall i$
- Recall: $\sum_{j=1}^N w_{ij} \mathbf{x}_k^j = w_{ii} \mathbf{x}_k^i + \sum_{j \in N_i} w_{ij} \mathbf{x}_k^j$

DGD revisited

- DGD is the first distributed algorithm that we see in this course and it does not need a server. Each device can compute its version of the optimal decision.

DGD revisited

- DGD is the first distributed algorithm that we see in this course and it does not need a server. Each device can compute its version of the optimal decision.
- Communication needs to happen synchronously and bi-directionally

DGD revisited

- DGD is the first distributed algorithm that we see in this course and it does not need a server. Each device can compute its version of the optimal decision.
- Communication needs to happen synchronously and bi-directionally
- Other versions exist with the mixing matrix being applied before/after the local gradient, but the idea stays the same

DGD revisited

- DGD is the first distributed algorithm that we see in this course and it does not need a server. Each device can compute its version of the optimal decision.
- Communication needs to happen synchronously and bi-directionally
- Other versions exist with the mixing matrix being applied before/after the local gradient, but the idea stays the same
- What can we say about convergence? For a constant step size α , it turns out that it is not so difficult, since

$$\mathbf{y}_{k+1} = \mathbf{W}\mathbf{y}_k - \alpha \nabla_{\mathbf{y}} F(\mathbf{y}_k) = \mathbf{y}_k - \alpha \left[\nabla_{\mathbf{y}} F(\mathbf{y}_k) - \frac{1}{\alpha} (\mathbf{W} - \mathbf{I}) \mathbf{y}_k \right]$$

so it's the same as running a standard gradient descent on the problem,

$$\min_{\mathbf{y} \in \mathbb{R}^{Nn}} \alpha F(\mathbf{y}) + \frac{1}{2} \mathbf{y}^\top (\mathbf{I} - \mathbf{W}) \mathbf{y}.$$

DGD revisited

- DGD is the first distributed algorithm that we see in this course and it does not need a server. Each device can compute its version of the optimal decision.
- Communication needs to happen synchronously and bi-directionally
- Other versions exist with the mixing matrix being applied before/after the local gradient, but the idea stays the same
- What can we say about convergence? For a constant step size α , it turns out that it is not so difficult, since

$$\mathbf{y}_{k+1} = \mathbf{W}\mathbf{y}_k - \alpha \nabla_{\mathbf{y}} F(\mathbf{y}_k) = \mathbf{y}_k - \alpha \left[\nabla_{\mathbf{y}} F(\mathbf{y}_k) - \frac{1}{\alpha} (\mathbf{W} - \mathbf{I}) \mathbf{y}_k \right]$$

so it's the same as running a standard gradient descent on the problem,

$$\min_{\mathbf{y} \in \mathbb{R}^{Nn}} \alpha F(\mathbf{y}) + \frac{1}{2} \mathbf{y}^\top (\mathbf{I} - \mathbf{W}) \mathbf{y}.$$

- For a general diminishing α_k sequence, it is a bit more complex but doable

DGD convergence

- We assume that the mixing matrix $W = [w_{ij}]$ is symmetric and doubly stochastic.

DGD convergence

- We assume that the mixing matrix $W = [w_{ij}]$ is symmetric and doubly stochastic.
- The eigenvalues of W are real and lie in the unit disk; they can be sorted in a nonincreasing order

$$1 = \lambda_1(W) \geq \lambda_2(W) \geq \dots \geq \lambda_N(W) \geq -1.$$

Let the second largest magnitude of the eigenvalues of W be denoted as

$$\gamma = \max\{|\lambda_2(W)|, |\lambda_N(W)|\}$$

DGD convergence

- We assume that the mixing matrix $W = [w_{ij}]$ is symmetric and doubly stochastic.
- The eigenvalues of W are real and lie in the unit disk; they can be sorted in a nonincreasing order

$$1 = \lambda_1(W) \geq \lambda_2(W) \geq \dots \geq \lambda_N(W) \geq -1.$$

Let the second largest magnitude of the eigenvalues of W be denoted as

$$\gamma = \max\{|\lambda_2(W)|, |\lambda_N(W)|\}$$

- Example: $W = I - \frac{1}{\max(d_i)+1} \mathcal{L}_{\mathcal{G}}$ is doubly-stochastic.

DGD convergence

- We assume that the mixing matrix $W = [w_{ij}]$ is symmetric and doubly stochastic.
- The eigenvalues of W are real and lie in the unit disk; they can be sorted in a nonincreasing order

$$1 = \lambda_1(W) \geq \lambda_2(W) \geq \dots \geq \lambda_N(W) \geq -1.$$

Let the second largest magnitude of the eigenvalues of W be denoted as

$$\gamma = \max\{|\lambda_2(W)|, |\lambda_N(W)|\}$$

- Example: $W = I - \frac{1}{\max(d_i)+1} \mathcal{L}_{\mathcal{G}}$ is doubly-stochastic.
- **Some basic questions** in the decentralized/distributed setting arise: (1) When does \mathbf{x}_k^i converge? (2) Does it converge to \mathbf{x}^* (3) If not does consensus (i.e., $\mathbf{x}_k^i = \mathbf{x}_k^j$) hold asymptotically? (4) How do the properties of f_i and the network affect convergence?

DGD convergence: statement

Theorem 4 (DGD convergence)

Consider Problem (P) and its solution via a decentralized gradient descent algorithm with constant step size α , and doubly stochastic communication matrix W with $\gamma < 1$.

Let convex functions f_i be L_i -smooth and let $L = \max_i \{L_i\}$. Let the mean value be

$$\bar{x}_k = \frac{1}{N} \sum_{i=1}^N x_k^i.$$

If α is chosen small enough and in particular $\leq O(1/L)$, then

① Consensus:

$$\|x_k^i - \bar{x}_k\| \rightarrow O\left(\frac{\alpha}{1-\gamma}\right);$$

② Convergence

$$f(\bar{x}_k) - f^* \rightarrow O\left(\frac{\alpha}{1-\gamma}\right).$$

DGD convergence: statement, cont'ed

- In the statement, we observe two things: convergence of the mean value and consensus around the mean, this is a general characteristic of distributed optimization

DGD convergence: statement, cont'ed

- In the statement, we observe two things: convergence of the mean value and consensus around the mean, this is a general characteristic of distributed optimization
- We also see a dependency on the graph entering via its spectrum γ , and in particular (here) the second largest eigenvalue (in modulus).

DGD convergence: statement, cont'd

- In the statement, we observe two things: convergence of the mean value and consensus around the mean, this is a general characteristic of distributed optimization
- We also see a dependency on the graph entering via its spectrum γ , and in particular (here) the second largest eigenvalue (in modulus).
- The proofs can be extended to cases in which f_i are strongly convex (easier to handle), but it is much harder to work with non-doubly stochastic matrices (i.e., communication issues). See arXiv:1310.7063

DGD convergence: statement, cont'd

- In the statement, we observe two things: convergence of the mean value and consensus around the mean, this is a general characteristic of distributed optimization
- We also see a dependency on the graph entering via its spectrum γ , and in particular (here) the second largest eigenvalue (in modulus).
- The proofs can be extended to cases in which f_i are strongly convex (easier to handle), but it is much harder to work with non-doubly stochastic matrices (i.e., communication issues). See [arXiv:1310.7063](https://arxiv.org/abs/1310.7063)
- We can see how for a constant step size we obtain a constant error bound (not so surprising, since we are changing the cost function!)

DGD convergence: statement, cont'd

- In the statement, we observe two things: convergence of the mean value and consensus around the mean, this is a general characteristic of distributed optimization
- We also see a dependency on the graph entering via its spectrum γ , and in particular (here) the second largest eigenvalue (in modulus).
- The proofs can be extended to cases in which f_i are strongly convex (easier to handle), but it is much harder to work with non-doubly stochastic matrices (i.e., communication issues). See [arXiv:1310.7063](https://arxiv.org/abs/1310.7063)
- We can see how for a constant step size we obtain a constant error bound (not so surprising, since we are changing the cost function!)
- When α is diminishing, you can obtain a zero error bound, but the analysis is more complicated, and you require typically extra assumptions.

Network effects

- The parameter γ depends on the graph. For a full graph with weights $1/N$, $\gamma = 0$, because you can choose,

$$W = \begin{bmatrix} \frac{1}{N} & \cdots & \frac{1}{N} \\ \vdots & \ddots & \vdots \\ \frac{1}{N} & \cdots & \frac{1}{N} \end{bmatrix},$$

which has one eigenvalue at 1 and all the rests at 0. **No network effect!**

Network effects

- The parameter γ depends on the graph. For a full graph with weights $1/N$, $\gamma = 0$, because you can choose,

$$W = \begin{bmatrix} \frac{1}{N} & \cdots & \frac{1}{N} \\ \vdots & \ddots & \vdots \\ \frac{1}{N} & \cdots & \frac{1}{N} \end{bmatrix},$$

which has one eigenvalue at 1 and all the rests at 0. **No network effect!**

- Other γ can depend on the number of nodes, and in the worst case (aka path graph)

$$\frac{1}{1 - \gamma} \sim N^2,$$

so the error can get pretty bad. **Total network effect!**

Network effects

- The parameter γ depends on the graph. For a full graph with weights $1/N$, $\gamma = 0$, because you can choose,

$$W = \begin{bmatrix} \frac{1}{N} & \cdots & \frac{1}{N} \\ \vdots & \ddots & \vdots \\ \frac{1}{N} & \cdots & \frac{1}{N} \end{bmatrix},$$

which has one eigenvalue at 1 and all the rests at 0. **No network effect!**

- Other γ can depend on the number of nodes, and in the worst case (aka path graph)

$$\frac{1}{1-\gamma} \sim N^2,$$

so the error can get pretty bad. **Total network effect!**

- γ will grow depending on how the graph is connected (in general, the more connected the lower is the γ).

Communication issues

- **Directed communication**: this is hard, since we lose double-stochasticity. Also you can have cycles in the graph and counter-intuitive behaviors. Usually solved with algorithms of the type “push-sum”

Communication issues

- **Directed communication**: this is hard, since we lose double-stochasticity. Also you can have cycles in the graph and counter-intuitive behaviors. Usually solved with algorithms of the type “push-sum”
- **Package drops, asynchronous communication**. This is often less hard to deal with, if the instantaneous W stay symmetric and doubly stochastic. **Typically**, the idea is to define a finite set of communication matrices $\{W^r\}_r$ and at each time k we randomly pick from them. If each W^r has the same structural properties of W and the eigenvalues of a restricted sequence of W^r have the same properties of the eigenvalues of W , then we can extend convergence results. (More later.)

Communication issues

- **Directed communication**: this is hard, since we lose double-stochasticity. Also you can have cycles in the graph and counter-intuitive behaviors. Usually solved with algorithms of the type “push-sum”
- **Package drops, asynchronous communication**. This is often less hard to deal with, if the instantaneous W stay symmetric and doubly stochastic. **Typically**, the idea is to define a finite set of communication matrices $\{W^r\}_r$ and at each time k we randomly pick from them. If each W^r has the same structural properties of W and the eigenvalues of a restricted sequence of W^r have the same properties of the eigenvalues of W , then we can extend convergence results. (More later.)
- **Gossip protocols**: asynchronous and directed. A node wakes up and transmits to some of its neighbors, it also analyses what it has received, then it sleeps again. This is **very realistic** but **quite hard to do**. Here latencies.. (More later.)

Communication issues

- **Directed communication**: this is hard, since we lose double-stochasticity. Also you can have cycles in the graph and counter-intuitive behaviors. Usually solved with algorithms of the type “push-sum”
- **Package drops, asynchronous communication**. This is often less hard to deal with, if the instantaneous W stay symmetric and doubly stochastic. **Typically**, the idea is to define a finite set of communication matrices $\{W^r\}_r$ and at each time k we randomly pick from them. If each W^r has the same structural properties of W and the eigenvalues of a restricted sequence of W^r have the same properties of the eigenvalues of W , then we can extend convergence results. (More later.)
- **Gossip protocols**: asynchronous and directed. A node wakes up and transmits to some of its neighbors, it also analyses what it has received, then it sleeps again. This is **very realistic** but **quite hard to do**. Here latencies.. (More later.)
- We will study these in the **next lesson**.

DGD is the only algorithm?

- Of course not: many variants, but the asymptotic non-zero error for non-vanishing step size will stay the same,

DGD is the only algorithm?

- Of course not: many variants, but the asymptotic non-zero error for non-vanishing step size will stay the same,
- For instance, we have a distributed algorithm (which will encounter later) which does,

$$\mathbf{y}_{k+1} = \mathbf{W}(\mathbf{y}_k - \alpha \nabla_{\mathbf{y}} F(\mathbf{y}_k)),$$

Let's call it **distributed gradient descent** for us

DGD is the only algorithm?

- Of course not: many variants, but the asymptotic non-zero error for non-vanishing step size will stay the same,
- For instance, we have a distributed algorithm (which will encounter later) which does,

$$\mathbf{y}_{k+1} = \mathbf{W}(\mathbf{y}_k - \alpha \nabla_{\mathbf{y}} F(\mathbf{y}_k)),$$

Let's call it **distributed gradient descent** for us

- Convergence is more involved, but distributed-GD has the property that iff \mathbf{W} is fully connected and $W_{ij} = 1/N$ for all i, j , then $\mathbf{y}_{k+1} \rightarrow \mathbf{y}^*$

DGD is the only algorithm?

- Of course not: many variants, but the asymptotic non-zero error for non-vanishing step size will stay the same,
- For instance, we have a distributed algorithm (which will encounter later) which does,

$$\mathbf{y}_{k+1} = \mathbf{W}(\mathbf{y}_k - \alpha \nabla_{\mathbf{y}} F(\mathbf{y}_k)),$$

Let's call it **distributed gradient descent** for us

- Convergence is more involved, but distributed-GD has the property that iff \mathbf{W} is fully connected and $W_{ij} = 1/N$ for all i, j , then $\mathbf{y}_{k+1} \rightarrow \mathbf{y}^*$
- Proof: Since \mathbf{W} averages all the elements, all the \mathbf{x}_{k+1}^i are the same, and $\alpha \nabla_{\mathbf{y}} F(\mathbf{y}_{k+1}) = (\alpha/N) \sum_i \nabla_{\mathbf{x}} f_i(\mathbf{x}_{k+1})$, that is the standard gradient descent. □

DGD is the only algorithm?

- Of course not: many variants, but the asymptotic non-zero error for non-vanishing step size will stay the same,
- For instance, we have a distributed algorithm (which will encounter later) which does,

$$\mathbf{y}_{k+1} = \mathbf{W}(\mathbf{y}_k - \alpha \nabla_{\mathbf{y}} F(\mathbf{y}_k)),$$

Let's call it **distributed gradient descent** for us

- Convergence is more involved, but distributed-GD has the property that iff \mathbf{W} is fully connected and $W_{ij} = 1/N$ for all i, j , then $\mathbf{y}_{k+1} \rightarrow \mathbf{y}^*$
- Proof: Since \mathbf{W} averages all the elements, all the \mathbf{x}_{k+1}^i are the same, and $\alpha \nabla_{\mathbf{y}} F(\mathbf{y}_{k+1}) = (\alpha/N) \sum_i \nabla_{\mathbf{x}} f_i(\mathbf{x}_{k+1})$, that is the standard gradient descent. □
- This is the only real property that is different for distributed-GD with DGD..

Gradient tracking

- We have seen that DGD has a fundamental trade-off between accuracy and performance. Can we do better?

Gradient tracking

- We have seen that DGD has a fundamental trade-off between accuracy and performance. Can we do better?
- Consider the following two-communication rounds algorithm

Gradient tracking

- We have seen that DGD has a fundamental trade-off between accuracy and performance. Can we do better?
- Consider the following two-communication rounds algorithm
 - **Decentralized Gradient Tracking**

Gradient tracking

- We have seen that DGD has a fundamental trade-off between accuracy and performance. Can we do better?
- Consider the following two-communication rounds algorithm

- **Decentralized Gradient Tracking**

- ▶ Initialize $\mathbf{x}_k^i, \mathbf{g}_k^i$

Gradient tracking

- We have seen that DGD has a fundamental trade-off between accuracy and performance. Can we do better?
- Consider the following two-communication rounds algorithm

- **Decentralized Gradient Tracking**

- ▶ Initialize $\mathbf{x}_k^i, \mathbf{g}_k^i$
- ▶ Each device gets and sends $\mathbf{x}_k^j / \mathbf{x}_k^i$ from/to the neighbors

Gradient tracking

- We have seen that DGD has a fundamental trade-off between accuracy and performance. Can we do better?
- Consider the following two-communication rounds algorithm

- **Decentralized Gradient Tracking**

- ▶ Initialize $\mathbf{x}_k^i, \mathbf{g}_k^i$
 - ▶ Each device gets and sends $\mathbf{x}_k^j / \mathbf{x}_k^i$ from/to the neighbors
 - ▶ Each device computes:

$$\mathbf{x}_{k+1}^i = \sum_{j=1}^N w_{ij} \mathbf{x}_k^j - \alpha \mathbf{g}_k^i, \quad \forall i$$

Gradient tracking

- We have seen that DGD has a fundamental trade-off between accuracy and performance. Can we do better?
- Consider the following two-communication rounds algorithm

- **Decentralized Gradient Tracking**

- ▶ Initialize $\mathbf{x}_k^i, \mathbf{g}_k^i$
 - ▶ Each device gets and sends $\mathbf{x}_k^j / \mathbf{x}_k^i$ from/to the neighbors
 - ▶ Each device computes:

$$\mathbf{x}_{k+1}^i = \sum_{j=1}^N w_{ij} \mathbf{x}_k^j - \alpha \mathbf{g}_k^i, \quad \forall i$$

- ▶ Each device gets and sends $\mathbf{g}_k^j / \mathbf{g}_k^i$ from/to the neighbors

Gradient tracking

- We have seen that DGD has a fundamental trade-off between accuracy and performance. Can we do better?
- Consider the following two-communication rounds algorithm

- **Decentralized Gradient Tracking**

- ▶ Initialize $\mathbf{x}_k^i, \mathbf{g}_k^i$
- ▶ Each device gets and sends $\mathbf{x}_k^j / \mathbf{x}_k^i$ from/to the neighbors
- ▶ Each device computes:

$$\mathbf{x}_{k+1}^i = \sum_{j=1}^N w_{ij} \mathbf{x}_k^j - \alpha \mathbf{g}_k^i, \quad \forall i$$

- ▶ Each device gets and sends $\mathbf{g}_k^j / \mathbf{g}_k^i$ from/to the neighbors
- ▶ Each device computes:

$$\mathbf{g}_{k+1}^i = \sum_{j=1}^N w_{ij} \mathbf{g}_k^j + (\nabla f_i(\mathbf{x}_{k+1}^i) - \nabla f_i(\mathbf{x}_k^i)), \quad \forall i$$

Gradient tracking

- We have seen that DGD has a fundamental trade-off between accuracy and performance. Can we do better?
- Consider the following two-communication rounds algorithm

- **Decentralized Gradient Tracking**

- ▶ Initialize $\mathbf{x}_k^i, \mathbf{g}_k^i$
- ▶ Each device gets and sends $\mathbf{x}_k^j / \mathbf{x}_k^i$ from/to the neighbors
- ▶ Each device computes:

$$\mathbf{x}_{k+1}^i = \sum_{j=1}^N w_{ij} \mathbf{x}_k^j - \alpha \mathbf{g}_k^i, \quad \forall i$$

- ▶ Each device gets and sends $\mathbf{g}_k^j / \mathbf{g}_k^i$ from/to the neighbors
- ▶ Each device computes:

$$\mathbf{g}_{k+1}^i = \sum_{j=1}^N w_{ij} \mathbf{g}_k^j + (\nabla f_i(\mathbf{x}_{k+1}^i) - \nabla f_i(\mathbf{x}_k^i)), \quad \forall i$$

- We have now a term that tracks the gradient values and “integrates” the errors

Gradient tracking

- Gradient tracking can be seen as a function of $\mathbf{y} = [\mathbf{x}^1, \dots, \mathbf{x}^N]$ and $\mathbf{q} = [\mathbf{g}^1, \dots, \mathbf{g}^N]$, simply as,

$$\begin{cases} \mathbf{y}_{k+1} = \mathbf{W}\mathbf{y}_k - \alpha\mathbf{q}_k, \\ \mathbf{q}_{k+1} = \mathbf{W}\mathbf{q}_k + \nabla_{\mathbf{y}}F(\mathbf{y}_{k+1}) - \nabla_{\mathbf{y}}F(\mathbf{y}_k) \end{cases}$$

Gradient tracking convergence

- Start by defining the mean quantity, $\bar{\mathbf{x}}_k = \frac{1}{N} \sum_{i=1}^N \mathbf{x}_k^i$

Gradient tracking convergence

- Start by defining the mean quantity, $\bar{\mathbf{x}}_k = \frac{1}{N} \sum_{i=1}^N \mathbf{x}_k^i$

Theorem 5 (Gradient Tracking convergence)

Consider Problem (P) and its solution via a gradient tracking algorithm with constant step size α , and doubly stochastic communication matrix W with $w_{ij} \geq 0$. Let convex functions f_i be L -smooth and strongly convex. If α is chosen small enough and in particular $\leq O(1/L)$, then

- 1 Consensus:

$$\|\mathbf{x}_k^i - \bar{\mathbf{x}}_k\| \rightarrow 0;$$

- 2 Convergence

$$\|\bar{\mathbf{x}}_k - \mathbf{x}^*\| \rightarrow 0.$$

And convergence is linear.

Proof: arXiv:1906.10760



Gradient tracking explanation

- The proof construct a recursion of the form $\mathbf{v}_{k+1} \leq J(\alpha)\mathbf{v}_k$, and the condition on α guarantees that $\|J(\alpha)\| < 1$.

Gradient tracking explanation

- The proof construct a recursion of the form $\mathbf{v}_{k+1} \leq J(\alpha)\mathbf{v}_k$, and the condition on α guarantees that $\|J(\alpha)\| < 1$.
- Extensions to non-strongly convex or to other communication patters are possible, but very involved

Gradient tracking explanation

- The proof construct a recursion of the form $\mathbf{v}_{k+1} \leq J(\alpha)\mathbf{v}_k$, and the condition on α guarantees that $\|J(\alpha)\| < 1$.
- Extensions to non-strongly convex or to other communication patters are possible, but very involved
- On the good side, we have a first-order primal method that is fully distributed and allows us to reach zero consensus and residual error.

Gradient tracking explanation

- The proof constructs a recursion of the form $\mathbf{v}_{k+1} \leq J(\alpha)\mathbf{v}_k$, and the condition on α guarantees that $\|J(\alpha)\| < 1$.
- Extensions to non-strongly convex or to other communication patterns are possible, but very involved
- On the good side, we have a first-order primal method that is fully distributed and allows us to reach zero consensus and residual error.
- On the minus side, we communicate quite a lot in terms of gradients, so we lose in privacy

Gradient tracking explanation

- The proof construct a recursion of the form $\mathbf{v}_{k+1} \leq J(\alpha)\mathbf{v}_k$, and the condition on α guarantees that $\|J(\alpha)\| < 1$.
- Extensions to non-strongly convex or to other communication patters are possible, but very involved
- On the good side, we have a first-order primal method that is fully distributed and allows us to reach zero consensus and residual error.
- On the minus side, we communicate quite a lot in terms of gradients, so we lose in privacy
- The only algorithm? Not really! Another example: **EXTRA** [arXiv:1404.6264]

What have we learned?

- The general problem of interest is minimizing $\sum_i f_i(\mathbf{x})$

What have we learned?

- The general problem of interest is minimizing $\sum_i f_i(\mathbf{x})$
- When N is big or f_i cannot be shared (efficiently or at all), decentralized schemes play an important role

What have we learned?

- The general problem of interest is minimizing $\sum_i f_i(\mathbf{x})$
- When N is big or f_i cannot be shared (efficiently or at all), decentralized schemes play an important role
- Communication and computation and their trade-off are then key: and of course, first-order algorithms

What have we learned?

- The general problem of interest is minimizing $\sum_i f_i(\mathbf{x})$
- When N is big or f_i cannot be shared (efficiently or at all), decentralized schemes play an important role
- Communication and computation and their trade-off are then key: and of course, first-order algorithms
- Two type of schemes: Incremental or parallel

What have we learned?

- The general problem of interest is minimizing $\sum_i f_i(\mathbf{x})$
- When N is big or f_i cannot be shared (efficiently or at all), decentralized schemes play an important role
- Communication and computation and their trade-off are then key: and of course, first-order algorithms
- Two type of schemes: Incremental or parallel
- For the former, we have seen a convergence result. The latter yields “distributed optimization” and, in the primal space, we have seen two example of algorithms: DGD and Gradient Tracking in a simplified “controlled” setting

What have we learned?

- The general problem of interest is minimizing $\sum_i f_i(\mathbf{x})$
- When N is big or f_i cannot be shared (efficiently or at all), decentralized schemes play an important role
- Communication and computation and their trade-off are then key: and of course, first-order algorithms
- Two type of schemes: Incremental or parallel
- For the former, we have seen a convergence result. The latter yields “distributed optimization” and, in the primal space, we have seen two example of algorithms: DGD and Gradient Tracking in a simplified “controlled” setting
- Convergence proofs in distributed settings are harder since the communication plays an important role!

What have we learned?

- The general problem of interest is minimizing $\sum_i f_i(\mathbf{x})$
- When N is big or f_i cannot be shared (efficiently or at all), decentralized schemes play an important role
- Communication and computation and their trade-off are then key: and of course, first-order algorithms
- Two type of schemes: Incremental or parallel
- For the former, we have seen a convergence result. The latter yields “distributed optimization” and, in the primal space, we have seen two example of algorithms: DGD and Gradient Tracking in a simplified “controlled” setting
- Convergence proofs in distributed settings are harder since the communication plays an important role!
- Why would I care about communication or distributed when we can all upload on the cloud?

What have we learned?

- The general problem of interest is minimizing $\sum_i f_i(\mathbf{x})$
- When N is big or f_i cannot be shared (efficiently or at all), decentralized schemes play an important role
- Communication and computation and their trade-off are then key: and of course, first-order algorithms
- Two type of schemes: Incremental or parallel
- For the former, we have seen a convergence result. The latter yields “distributed optimization” and, in the primal space, we have seen two example of algorithms: DGD and Gradient Tracking in a simplified “controlled” setting
- Convergence proofs in distributed settings are harder since the communication plays an important role!
- Why would I care about communication or distributed when we can all upload on the cloud? (Think BIG: large-data files, server farms, or many sensors, etc..)

Sample references

- 1 The standard book: *Dimitri P. Bertsekas and John N. Tsitsiklis*, **Parallel and Distributed Computation: Numerical Methods**, 1997,
<https://web.mit.edu/dimitrib/www/pdc.html>
- 2 Two recent articles:
Kun Yuan, Qing Ling, Wotao Yin, **On the Convergence of Decentralized Gradient Descent**, arXiv:1310.7063 and SIAM Journal on Optimization, 2016
Giuseppe Notarstefano, Ivano Notarnicola, Andrea Camisa, **Distributed Optimization for Smart Cyber-Physical Networks**, arXiv:1906.10760 and Foundations and Trends in Systems and Control, 2019
- 3 Many variants out there.

Projects

The literature project (20%)

- Divide yourselves in groups of 3.
- In the lecture notes you find papers marked as **Research papers**. Choose one paper per group (each group needs to have a different paper). Read it and fill the literature.tex file on the moodle.

The numerical python project (40%)

- Stay in the same group of 3.
- And..

A numerical example: kernel ridge regression

- We are now looking at a numerical example, which you will code in a python project.

A numerical example: kernel ridge regression

- We are now looking at a numerical example, which you will code in a python project.
- Kernel ridge regression is a machine learning task that amounts to fit a functional model to noisy data, in a non-parametric form. You can think of it as linear regression plus plus.

A numerical example: kernel ridge regression

- We are now looking at a numerical example, which you will code in a python project.
- Kernel ridge regression is a machine learning task that amounts to fit a functional model to noisy data, in a non-parametric form. You can think of it as linear regression plus plus.
- Let y_i for $i = 1, \dots, n$ be scalar evaluations of a certain unknown function $f(x) : \mathbf{R} \rightarrow \mathbf{R}$ at points x_i for $i = 1, \dots, n$. In our case,

$$y_i = f(x_i) + \epsilon, \quad \epsilon \sim \mathcal{N}(0, \sigma^2).$$

A numerical example: kernel ridge regression

- We use a kernel representation of the function as

$$f(x) = \sum_{i=1}^n \alpha_i k(x, x_i),$$

where $k(x, x_i)$ is the kernel. Here we will use an Euclidean kernel as

$$k(x, x_i) = \exp(-\|x - x_i\|^2).$$

We also define the kernel matrix as $K = [k(x_i, x_j)]_{i,j=1,\dots,n}$.

A numerical example: kernel ridge regression

- We use a kernel representation of the function as

$$f(x) = \sum_{i=1}^n \alpha_i k(x, x_i),$$

where $k(x, x_i)$ is the kernel. Here we will use an Euclidean kernel as

$$k(x, x_i) = \exp(-\|x - x_i\|^2).$$

We also define the kernel matrix as $K = [k(x_i, x_j)]_{i,j=1,\dots,n}$.

- Function f is linear in the parameters α_i . To lower the computational requirement, we also use a Nyström approximation. We select uniformly at random amongst the n points, $m \sim \sqrt{n}$ points. We let \mathcal{M} be the set of indexes of these points w.r.t. the original set of points. For example, if $m = 3$ then \mathcal{M} could be the set $\{1, 4, 7\}$, corresponding to the first, fourth, and seventh point of the original n points.

A numerical example: kernel ridge regression

- We use a kernel representation of the function as

$$f(x) = \sum_{i=1}^n \alpha_i k(x, x_i),$$

where $k(x, x_i)$ is the kernel. Here we will use an Euclidean kernel as

$$k(x, x_i) = \exp(-\|x - x_i\|^2).$$

We also define the kernel matrix as $K = [k(x_i, x_j)]_{i,j=1,\dots,n}$.

- Function f is linear in the parameters α_i . To lower the computational requirement, we also use a Nyström approximation. We select uniformly at random amongst the n points, $m \sim \sqrt{n}$ points. We let \mathcal{M} be the set of indexes of these points w.r.t. the original set of points. For example, if $m = 3$ then \mathcal{M} could be the set $\{1, 4, 7\}$, corresponding to the first, fourth, and seventh point of the original n points.
- The approximation then reads

$$f(x) = \sum_{i=1}^n \alpha_i k(x, x_i) \approx \sum_{j \in \mathcal{M}} \alpha_j k(x, x_j).$$

A numerical example: kernel ridge regression

- Determining the vector $\alpha \in \mathbf{R}^m$ is a model training problem which can be written as,

$$\alpha^* = \arg \min_{\alpha \in \mathbf{R}^m} \frac{\sigma^2}{2} \alpha^\top K_{mm} \alpha + \frac{1}{2} \|y - K_{nm} \alpha\|_2^2,$$

where $K_{nm} = [k(x_i, x_j)]_{i=1, \dots, n; j \in \mathcal{M}}$, and y is the stacked version of all y_i .

A numerical example: kernel ridge regression

- Determining the vector $\alpha \in \mathbf{R}^m$ is a model training problem which can be written as,

$$\alpha^* = \arg \min_{\alpha \in \mathbf{R}^m} \frac{\sigma^2}{2} \alpha^\top K_{mm} \alpha + \frac{1}{2} \|y - K_{nm} \alpha\|_2^2,$$

where $K_{nm} = [k(x_i, x_j)]_{i=1, \dots, n; j \in \mathcal{M}}$, and y is the stacked version of all y_i .

- Note then that,

$$\|y - K_{nm} \alpha\|_2^2 = \sum_{i=1}^n \|y_i - [k(x_i, x_j)]_{j \in \mathcal{M}} \alpha\|_2^2.$$

A numerical example: kernel ridge regression

- Determining the vector $\alpha \in \mathbf{R}^m$ is a model training problem which can be written as,

$$\alpha^* = \arg \min_{\alpha \in \mathbf{R}^m} \frac{\sigma^2}{2} \alpha^\top K_{mm} \alpha + \frac{1}{2} \|y - K_{nm} \alpha\|_2^2,$$

where $K_{nm} = [k(x_i, x_j)]_{i=1, \dots, n; j \in \mathcal{M}}$, and y is the stacked version of all y_i .

- Note then that,

$$\|y - K_{nm} \alpha\|_2^2 = \sum_{i=1}^n \|y_i - [k(x_i, x_j)]_{j \in \mathcal{M}} \alpha\|_2^2.$$

- To make things simpler, we also add a small regularisation, such that,

$$\alpha^* = \arg \min_{\alpha \in \mathbf{R}^m} \frac{\sigma^2}{2} \alpha^\top K_{mm} \alpha + \frac{1}{2} \|y - K_{nm} \alpha\|_2^2 + \frac{\nu}{2} \|\alpha\|_2^2,$$

for $\nu = 1.0$, and the problem becomes strongly convex and smooth.

A numerical example: kernel ridge regression

- If all the data is available in one single location, we can solve for α^* via the optimality conditions, and

$$[\sigma^2 K_{mm} + K_{nm}^\top K_{nm} + \nu I] \alpha^* = K_{nm}^\top y,$$

which is a linear algebra problem.

A numerical example: kernel ridge regression

- If all the data is available in one single location, we can solve for α^* via the optimality conditions, and

$$[\sigma^2 K_{mm} + K_{nm}^\top K_{nm} + \nu I] \alpha^* = K_{nm}^\top y,$$

which is a linear algebra problem.

- Remember never to invert matrices! Use the linear system solve in numpy.

A numerical example: kernel ridge regression

- We consider a database containing the x_i points (or features), and y_i noisy data (or labels). Here we will use $\sigma = 0.5$. There are one million points.

A numerical example: kernel ridge regression

- We consider a database containing the x_i points (or features), and y_i noisy data (or labels). Here we will use $\sigma = 0.5$. There are one million points.
- We choose the first $n = 100$, $m = 10$ points and share them across $a = 5$ agents or computers. The problem reads,

$$\begin{aligned}\alpha^* &= \arg \min_{\alpha \in \mathbb{R}^m} \sum_{a=1}^5 \left[\frac{1}{5} \frac{\sigma^2}{2} \alpha^\top K_{mm} \alpha + \frac{1}{2} \sum_{i \in A} \|y_i - K_{(i)m} \alpha\|_2^2 + \frac{\nu}{10} \|\alpha\|_2^2 \right], \\ &= \arg \min_{\alpha \in \mathbb{R}^m} \sum_{a=1}^5 f_i(\alpha)\end{aligned}$$

where we indicated with $i \in A$ the data points that belong to agent A , and with $K_{(i)m} = [k(x_i, x_j)]_{j \in \mathcal{M}}$.

A numerical example: kernel ridge regression

- We consider a database containing the x_i points (or features), and y_i noisy data (or labels). Here we will use $\sigma = 0.5$. There are one million points.
- We choose the first $n = 100$, $m = 10$ points and share them across $a = 5$ agents or computers. The problem reads,

$$\begin{aligned}\alpha^* &= \arg \min_{\alpha \in \mathbb{R}^m} \sum_{a=1}^5 \left[\frac{1}{5} \frac{\sigma^2}{2} \alpha^\top K_{mm} \alpha + \frac{1}{2} \sum_{i \in A} \|y_i - K_{(i)m} \alpha\|_2^2 + \frac{\nu}{10} \|\alpha\|_2^2 \right], \\ &= \arg \min_{\alpha \in \mathbb{R}^m} \sum_{a=1}^5 f_i(\alpha)\end{aligned}$$

where we indicated with $i \in A$ the data points that belong to agent A , and with $K_{(i)m} = [k(x_i, x_j)]_{j \in \mathcal{M}}$.

- Each agent has 20 points, each agent has different points.

A numerical example: kernel ridge regression

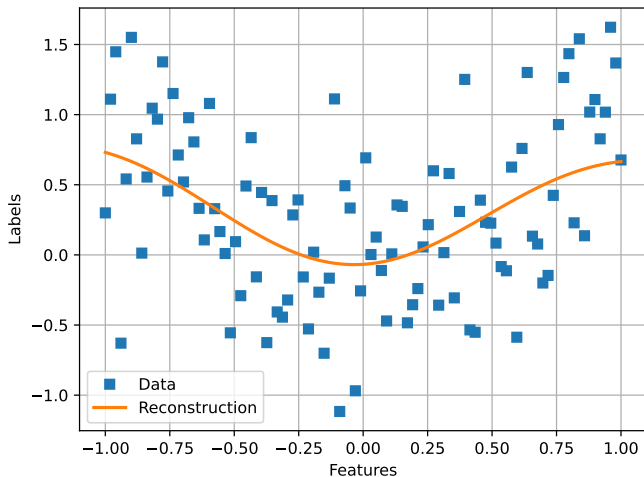
- We consider a database containing the x_i points (or features), and y_i noisy data (or labels). Here we will use $\sigma = 0.5$. There are one million points.
- We choose the first $n = 100, m = 10$ points and share them across $a = 5$ agents or computers. The problem reads,

$$\begin{aligned}\alpha^* &= \arg \min_{\alpha \in \mathbb{R}^m} \sum_{a=1}^5 \left[\frac{1}{5} \frac{\sigma^2}{2} \alpha^\top K_{mm} \alpha + \frac{1}{2} \sum_{i \in A} \|y_i - K_{(i)m} \alpha\|_2^2 + \frac{\nu}{10} \|\alpha\|_2^2 \right], \\ &= \arg \min_{\alpha \in \mathbb{R}^m} \sum_{a=1}^5 f_i(\alpha)\end{aligned}$$

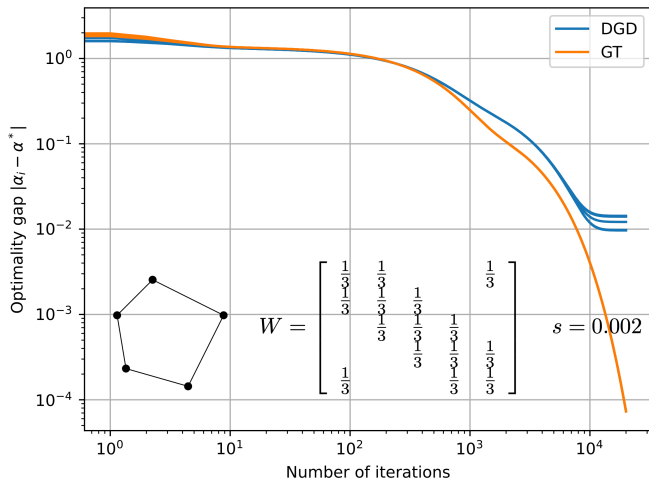
where we indicated with $i \in A$ the data points that belong to agent A , and with $K_{(i)m} = [k(x_i, x_j)]_{j \in \mathcal{M}}$.

- Each agent has 20 points, each agent has different points.
- we set a communication graph (connected and undirected) between the agents

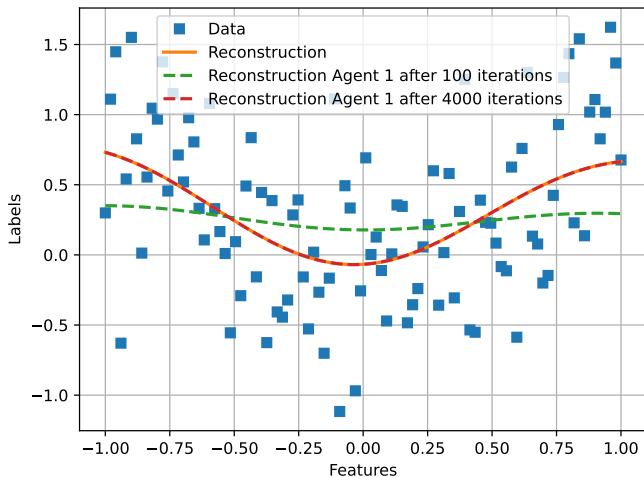
A numerical example: Centralized



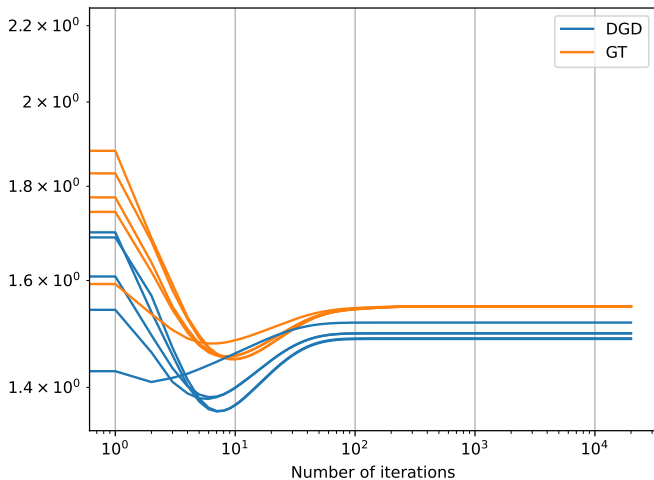
A numerical example: Convergence 1



A numerical example: Convergence 2



A numerical example: No convergence, $w_{51} = 0$



Class 2

Consensus and networks

- Today we start with the problem,

$$\underset{\mathbf{x} \in \mathbf{R}^n}{\text{minimize}} \frac{1}{2} \sum_{i=1}^N \|\mathbf{x} - \mathbf{v}_i\|_2^2, \quad (2)$$

where $\mathbf{v}_i \in \mathbf{R}^n$ are data proper to device i .

Consensus and networks

- Today we start with the problem,

$$\underset{\mathbf{x} \in \mathbf{R}^n}{\text{minimize}} \frac{1}{2} \sum_{i=1}^N \|\mathbf{x} - \mathbf{v}_i\|_2^2, \quad (2)$$

where $\mathbf{v}_i \in \mathbf{R}^n$ are data proper to device i .

- Since the optimizer of such simple problem is the average of the data points, i.e., $\mathbf{x}^* = \frac{1}{N} \sum_{i=1}^N \mathbf{v}_i$, then the question is how to ensure that the devices can reach a **consensus** on what the average is.

Consensus and networks

- Today we start with the problem,

$$\underset{\mathbf{x} \in \mathbb{R}^n}{\text{minimize}} \frac{1}{2} \sum_{i=1}^N \|\mathbf{x} - \mathbf{v}_i\|_2^2, \quad (2)$$

where $\mathbf{v}_i \in \mathbb{R}^n$ are data proper to device i .

- Since the optimizer of such simple problem is the average of the data points, i.e., $\mathbf{x}^* = \frac{1}{N} \sum_{i=1}^N \mathbf{v}_i$, then the question is how to ensure that the devices can reach a **consensus** on what the average is.
- If we were to employ DGD on this problem, we already know that we would not be able to reach consensus with a constant stepsize. It is easy to see that, since the DGD iteration is,

$$\mathbf{x}_{k+1}^i = \sum_{j=1}^N w_{ij} \mathbf{x}_k^j - \alpha (\mathbf{x}_k^i - \mathbf{v}_i),$$

and if all the \mathbf{x}_k^i were the same (and equal to \mathbf{x}^*), the iterate $k+1$ would bring us away from it by the term $(1-\alpha)\mathbf{x}^* + \alpha\mathbf{v}_i$.

Average consensus

- Consider the consensus iteration as follows.

Average consensus

- Consider the consensus iteration as follows.
 - ▶ Start by initializing $\mathbf{x}_0^i = \mathbf{v}_i$

Average consensus

- Consider the consensus iteration as follows.
 - ▶ Start by initializing $x_0^i = v_i$
 - ▶ Iterate for all $k = 0, 1, \dots$

Average consensus

- Consider the consensus iteration as follows.
 - ▶ Start by initializing $x_0^i = v_i$
 - ▶ Iterate for all $k = 0, 1, \dots$
 - ★ Communication step: send and receive x_k^i to and from your neighbors

Average consensus

- Consider the consensus iteration as follows.
 - ▶ Start by initializing $\mathbf{x}_0^i = \mathbf{v}_i$
 - ▶ Iterate for all $k = 0, 1, \dots$
 - ★ Communication step: send and receive \mathbf{x}_k^i to and from your neighbors
 - ★ Computation step: each device computes,

$$\mathbf{x}_{k+1}^i = \sum_{j=1}^N w_{ij} \mathbf{x}_k^j.$$

Average consensus

- Consider the consensus iteration as follows.
 - ▶ Start by initializing $\mathbf{x}_0^i = \mathbf{v}_i$
 - ▶ Iterate for all $k = 0, 1, \dots$
 - ★ Communication step: send and receive \mathbf{x}_k^i to and from your neighbors
 - ★ Computation step: each device computes,

$$\mathbf{x}_{k+1}^i = \sum_{j=1}^N w_{ij} \mathbf{x}_k^j.$$

- The algorithm can be put into the matrix-vector form, as usual, by introducing the stacked vector \mathbf{y} and the matrix \mathbf{W} as,

$$\mathbf{y}_{k+1} = \mathbf{W} \mathbf{y}_k, \quad \mathbf{y}_0 = [\mathbf{v}_1^\top, \dots, \mathbf{v}_N^\top]^\top. \quad (3)$$

Average consensus

- Consider the consensus iteration as follows.
 - ▶ Start by initializing $\mathbf{x}_0^i = \mathbf{v}_i$
 - ▶ Iterate for all $k = 0, 1, \dots$
 - ★ Communication step: send and receive \mathbf{x}_k^i to and from your neighbors
 - ★ Computation step: each device computes,

$$\mathbf{x}_{k+1}^i = \sum_{j=1}^N w_{ij} \mathbf{x}_k^j.$$

- The algorithm can be put into the matrix-vector form, as usual, by introducing the stacked vector \mathbf{y} and the matrix \mathbf{W} as,

$$\mathbf{y}_{k+1} = \mathbf{W} \mathbf{y}_k, \quad \mathbf{y}_0 = [\mathbf{v}_1^\top, \dots, \mathbf{v}_N^\top]^\top. \quad (3)$$

- We then study the more general recursion,

$$\mathbf{y}_{k+1} = \mathbf{W}_k \mathbf{y}_k, \quad \mathbf{y}_0 = [\mathbf{v}_1^\top, \dots, \mathbf{v}_N^\top]^\top, \quad (4)$$

where we allow the communication weights to change with the iteration counter to simulate time-varying communication graphs.

General recursion

- So, let's look at the recursion,

$$\mathbf{y}_{k+1} = \mathbf{W}_k \mathbf{y}_k$$

for arbitrary \mathbf{W}_k . The aim is to understand a bit better the results we found in distributed optimization for doubly-stochastic matrices, as well as why considering asynchronous and directed communication is problematic

General recursion

- So, let's look at the recursion,

$$\mathbf{y}_{k+1} = \mathbf{W}_k \mathbf{y}_k$$

for arbitrary \mathbf{W}_k . The aim is to understand a bit better the results we found in distributed optimization for doubly-stochastic matrices, as well as why considering asynchronous and directed communication is problematic

- This also makes us understand why purely gossip communication protocols (directed, asynchronous) are hard to work with

General recursion

- So, let's look at the recursion,

$$\mathbf{y}_{k+1} = \mathbf{W}_k \mathbf{y}_k$$

for arbitrary \mathbf{W}_k . The aim is to understand a bit better the results we found in distributed optimization for doubly-stochastic matrices, as well as why considering asynchronous and directed communication is problematic

- This also makes us understand why purely gossip communication protocols (directed, asynchronous) are hard to work with
- The theory we present here is related to stochastic matrix theory, and Markov chain

General recursion

- So, let's look at the recursion,

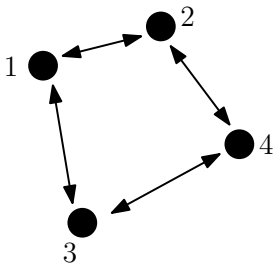
$$\mathbf{y}_{k+1} = \mathbf{W}_k \mathbf{y}_k$$

for arbitrary \mathbf{W}_k . The aim is to understand a bit better the results we found in distributed optimization for doubly-stochastic matrices, as well as why considering asynchronous and directed communication is problematic

- This also makes us understand why purely gossip communication protocols (directed, asynchronous) are hard to work with
- The theory we present here is related to stochastic matrix theory, and Markov chain
- Simplified setting: N agents $\mathbf{x}^i \in \mathbf{R}$, $\mathbf{y} \in \mathbf{R}^N$ as collection of all the scalar x^i . This can be easily extended for $\mathbf{x}^i \in \mathbf{R}^n$ with the Kroenecker product \otimes . The results stay the same.

Consensus, averages

- Start with a static doubly-stochastic matrix W as the example below:



Then,

Consensus, averages

- Start with a static doubly-stochastic matrix W as the example below:

Then,

- The average is preserved

$$\frac{\mathbf{1}}{N}^\top \mathbf{y}_{k+1} = \frac{\mathbf{1}}{N}^\top W \mathbf{y}_k = \frac{\mathbf{1}}{N}^\top \mathbf{y}_k$$

Consensus, averages

- Start with a static doubly-stochastic matrix W as the example below:

Then,

- The average is preserved

$$\frac{\mathbf{1}}{N}^\top \mathbf{y}_{k+1} = \frac{\mathbf{1}}{N}^\top W \mathbf{y}_k = \frac{\mathbf{1}}{N}^\top \mathbf{y}_k$$

- Consensus is preserved. If all $x_k^i = \bar{x}$ are the same then,

$$\mathbf{y}_{k+1} = W \mathbf{1} \bar{y} = \mathbf{1} \bar{x}$$

I.e., if we reach consensus, we stay at consensus.

Consensus, averages

- Start with a static doubly-stochastic matrix W as the example below:

Then,

- 1 The average is preserved

$$\frac{\mathbf{1}}{N}^\top \mathbf{y}_{k+1} = \frac{\mathbf{1}}{N}^\top W \mathbf{y}_k = \frac{\mathbf{1}}{N}^\top \mathbf{y}_k$$

- 2 Consensus is preserved. If all $x_k^i = \bar{x}$ are the same then,

$$\mathbf{y}_{k+1} = W \mathbf{1} \bar{x} = \mathbf{1} \bar{x}$$

I.e., if we reach consensus, we stay at consensus.

- Then,

Consensus, averages

- Start with a static doubly-stochastic matrix W as the example below:
Then,

- 1 The average is preserved

$$\frac{\mathbf{1}}{N}^\top \mathbf{y}_{k+1} = \frac{\mathbf{1}}{N}^\top W \mathbf{y}_k = \frac{\mathbf{1}}{N}^\top \mathbf{y}_k$$

- 2 Consensus is preserved. If all $x_k^i = \bar{x}$ are the same then,

$$\mathbf{y}_{k+1} = W \mathbf{1} \bar{x} = \mathbf{1} \bar{x}$$

I.e., if we reach consensus, we stay at consensus.

- Then,

Theorem 6

Consider the discrete-time dynamical system $\mathbf{y}_{k+1} = W \mathbf{y}_k$. If W is doubly stochastic, the graph is connected, and $\|W - \mathbf{1}\mathbf{1}^\top / N\| < 1$ then linearly

$$\lim_{k \rightarrow \infty} \mathbf{y}_k = \frac{\mathbf{1}\mathbf{1}^\top}{N} \mathbf{y}_0.$$

Consensus, averages

- Proof.

$$e_{k+1} = \mathbf{y}_{k+1} - \frac{\mathbf{1}\mathbf{1}^\top}{N}\mathbf{y}_0 = W\mathbf{y}_k - \frac{\mathbf{1}\mathbf{1}^\top}{N}\mathbf{y}_0 = (W^{k+1} - \frac{\mathbf{1}\mathbf{1}^\top}{N})\mathbf{e}_0$$

Consensus, averages

- Proof.

$$e_{k+1} = \mathbf{y}_{k+1} - \frac{\mathbf{1}\mathbf{1}^\top}{N} \mathbf{y}_0 = W \mathbf{y}_k - \frac{\mathbf{1}\mathbf{1}^\top}{N} \mathbf{y}_0 = (W^{k+1} - \frac{\mathbf{1}\mathbf{1}^\top}{N}) \mathbf{e}_0$$

- Then,

$$\begin{aligned} W^2 - \frac{\mathbf{1}\mathbf{1}^\top}{N} &= (W - \frac{\mathbf{1}\mathbf{1}^\top}{N})^2 \\ &= W^2 - 2\frac{\mathbf{1}\mathbf{1}^\top}{N} + \frac{\mathbf{1}\mathbf{1}^\top}{N} \end{aligned}$$

and similarly,

$$W^{k+1} - \frac{\mathbf{1}\mathbf{1}^\top}{N} = (W - \frac{\mathbf{1}\mathbf{1}^\top}{N})^{k+1}$$

Consensus, averages

- Proof.

$$e_{k+1} = \mathbf{y}_{k+1} - \frac{\mathbf{1}\mathbf{1}^\top}{N}\mathbf{y}_0 = W\mathbf{y}_k - \frac{\mathbf{1}\mathbf{1}^\top}{N}\mathbf{y}_0 = (W^{k+1} - \frac{\mathbf{1}\mathbf{1}^\top}{N})e_0$$

- Then,

$$\begin{aligned} W^2 - \frac{\mathbf{1}\mathbf{1}^\top}{N} &= (W - \frac{\mathbf{1}\mathbf{1}^\top}{N})^2 \\ &= W^2 - 2\frac{\mathbf{1}\mathbf{1}^\top}{N} + \frac{\mathbf{1}\mathbf{1}^\top}{N} \end{aligned}$$

and similarly,

$$W^{k+1} - \frac{\mathbf{1}\mathbf{1}^\top}{N} = (W - \frac{\mathbf{1}\mathbf{1}^\top}{N})^{k+1}$$

- Then if $\|W - \frac{\mathbf{1}\mathbf{1}^\top}{N}\| < 1$, the proof is concluded. □

Consensus, averages

- Proof.

$$e_{k+1} = \mathbf{y}_{k+1} - \frac{\mathbf{1}\mathbf{1}^\top}{N}\mathbf{y}_0 = W\mathbf{y}_k - \frac{\mathbf{1}\mathbf{1}^\top}{N}\mathbf{y}_0 = (W^{k+1} - \frac{\mathbf{1}\mathbf{1}^\top}{N})\mathbf{e}_0$$

- Then,

$$\begin{aligned} W^2 - \frac{\mathbf{1}\mathbf{1}^\top}{N} &= (W - \frac{\mathbf{1}\mathbf{1}^\top}{N})^2 \\ &= W^2 - 2\frac{\mathbf{1}\mathbf{1}^\top}{N} + \frac{\mathbf{1}\mathbf{1}^\top}{N} \end{aligned}$$

and similarly,

$$W^{k+1} - \frac{\mathbf{1}\mathbf{1}^\top}{N} = (W - \frac{\mathbf{1}\mathbf{1}^\top}{N})^{k+1}$$

- Then if $\|W - \frac{\mathbf{1}\mathbf{1}^\top}{N}\| < 1$, the proof is concluded. □

- Note that,

$$\text{eig}(W - \frac{\mathbf{1}\mathbf{1}^\top}{N}) = \begin{cases} \text{eig}(W) & \text{for all eigenvect. } \neq \mathbf{1} \\ 0 & \text{for all eigenvect. } = \mathbf{1} \end{cases}$$

so the dominant eigenvalue is the second largest in modulus (γ).

Consensus, averages

- Theorem 6 tells you that if W is doubly stochastic, then, eventually (by communicating and communicating), you approximate the average matrix $\frac{\mathbf{1}\mathbf{1}^\top}{N}$, exactly.

Consensus, averages

- Theorem 6 tells you that if W is doubly stochastic, then, eventually (by communicating and communicating), you approximate the average matrix $\frac{11^T}{N}$, exactly.
- This is good news. (However .. As we saw in DGD, the interleaved scheme yield a non-zero error, so the harder setting is harder indeed !)

Consensus, averages

- Theorem 6 tells you that if W is doubly stochastic, then, eventually (by communicating and communicating), you approximate the average matrix $\frac{11^T}{N}$, exactly.
- This is good news. (However .. As we saw in DGD, the interleaved scheme yield a non-zero error, so the harder setting is harder indeed !)
- Also, it shouldn't come as a surprise that typically the error in distributed optimization, where we do a mixing plus a term is

$$\sim O\left(\frac{1}{1-\gamma}\right),$$

where $1 - \gamma$ is called the spectral gap. ($\gamma \geq 1$ if the graph is not connected)

Consensus, averages

- Theorem 6 tells you that if W is doubly stochastic, then, eventually (by communicating and communicating), you approximate the average matrix $\frac{11^T}{N}$, exactly.
- This is good news. (However .. As we saw in DGD, the interleaved scheme yield a non-zero error, so the harder setting is harder indeed !)
- Also, it shouldn't come as a surprise that typically the error in distributed optimization, where we do a mixing plus a term is

$$\sim O\left(\frac{1}{1-\gamma}\right),$$

where $1 - \gamma$ is called the spectral gap. ($\gamma \geq 1$ if the graph is not connected)

- You can try to design optimal W 's, but as said in Class I, often depends on the underlying graph, and on the number of nodes. It is normal, W tells you how the information mixes..

Consensus, averages

- Theorem 6 tells you that if W is doubly stochastic, then, eventually (by communicating and communicating), you approximate the average matrix $\frac{11^T}{N}$, exactly.
- This is good news. (However .. As we saw in DGD, the interleaved scheme yield a non-zero error, so the harder setting is harder indeed !)
- Also, it shouldn't come as a surprise that typically the error in distributed optimization, where we do a mixing plus a term is

$$\sim O\left(\frac{1}{1-\gamma}\right),$$

where $1 - \gamma$ is called the spectral gap. ($\gamma \geq 1$ if the graph is not connected)

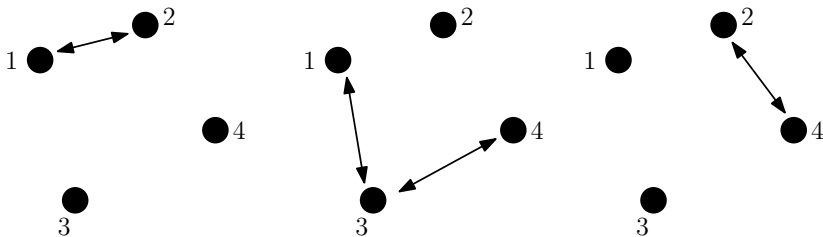
- You can try to design optimal W 's, but as said in Class I, often depends on the underlying graph, and on the number of nodes. It is normal, W tells you how the information mixes..
- What happens for W_k ? If you have “waking up edges” and you maintain double-stochasticity and you have connectivity infinitely often... what ??

Edge asynchronous graphs

- What happens for W_k ? If you have “waking up edges” and you maintain double-stochasticity and you have connectivity infinitely often... **what ??**

Edge asynchronous graphs

- What happens for W_k ? If you have “waking up edges” and you maintain double-stochasticity and you have connectivity infinitely often... **what ??**
- Consider a graph whose edges appear and disappear with a certain probability. Each edge carries bidirectional communication, but the instantaneous graph may be not connected.



Edge asynchronous graphs

- What happens for W_k ? If you have “waking up edges” and you maintain double-stochasticity and you have connectivity infinitely often... **what ??**
- Consider a graph whose edges appear and disappear with a certain probability. Each edge carries bidirectional communication, but the instantaneous graph may be not connected.
- We can design a W_k which is still doubly stochastic, albeit non-connected, e.g., consider the Metropolis weights:

$$W_k^{ij} = \begin{cases} \frac{1}{1 + \max\{d_k^i, d_k^j\}} & \text{for } (i, j) \in E_k \\ 1 - \sum_j W_k^{ij} & \text{for } i = j \\ 0 & \text{otherwise} \end{cases}$$

Here d^i is the degree (rem: the number of neighbors) of node i

Edge asynchronous graphs

- What happens for W_k ? If you have “waking up edges” and you maintain double-stochasticity and you have connectivity infinitely often... **what ??**
- Consider a graph whose edges appear and disappear with a certain probability. Each edge carries bidirectional communication, but the instantaneous graph may be not connected.
- We can design a W_k which is still doubly stochastic, albeit non-connected, e.g., consider the Metropolis weights:

$$W_k^{ij} = \begin{cases} \frac{1}{1 + \max\{d_k^i, d_k^j\}} & \text{for } (i, j) \in E_k \\ 1 - \sum_j W_k^{ij} & \text{for } i = j \\ 0 & \text{otherwise} \end{cases}$$

Here d^i is the degree (rem: the number of neighbors) of node i

- It may be difficult to build such W_k online, but if you have it, then it is symmetric and doubly stochastic

Edge asynchronous graphs

- Then,

Edge asynchronous graphs

- Then,

Theorem 7

Consider matrix W_k with Metropolis weights. If the collection of communication graphs that occur infinitely often are jointly connected, then for any \mathbf{y}_0 the recursion $\mathbf{y}_{k+1} = W_k \mathbf{y}_k$, converges as,

$$\lim_{k \rightarrow \infty} \mathbf{y}_k = \frac{\mathbf{1}\mathbf{1}^\top}{N} \mathbf{y}_0.$$

Edge asynchronous graphs

- Then,

Theorem 7

Consider matrix W_k with Metropolis weights. If the collection of communication graphs that occur infinitely often are jointly connected, then for any \mathbf{y}_0 the recursion $\mathbf{y}_{k+1} = W_k \mathbf{y}_k$, converges as,

$$\lim_{k \rightarrow \infty} \mathbf{y}_k = \frac{\mathbf{1}\mathbf{1}^\top}{N} \mathbf{y}_0.$$

- Other similar results for edge asynchronous situations are available under different assumptions.

Edge asynchronous graphs

- Then,
- Other similar results for edge asynchronous situations are available under different assumptions.
- Good news here. And in the harder setting?
- A way to look at it is that you are looking at this cost function instead:

$$\min_{\mathbf{y} \in \mathbb{R}^{Nn}} \mathbf{E}_{\mathbf{W}} [\alpha F(\mathbf{y}) + \frac{1}{2} \mathbf{y}^\top (\mathbf{I} - \mathbf{W}) \mathbf{y}].$$

Edge asynchronous graphs

- Then,
- Other similar results for edge asynchronous situations are available under different assumptions.
- Good news here. And in the harder setting?
- A way to look at it is that you are looking at this cost function instead:

$$\min_{\mathbf{y} \in \mathbb{R}^{Nn}} \mathbf{E}_{\mathbf{W}} [\alpha F(\mathbf{y}) + \frac{1}{2} \mathbf{y}^\top (\mathbf{I} - \mathbf{W}) \mathbf{y}].$$

- And you are performing stochastic updates to it. So you are converging with some errors, as we know.

Edge asynchronous graphs

- Then,
- Other similar results for edge asynchronous situations are available under different assumptions.
- Good news here. And in the harder setting?
- A way to look at it is that you are looking at this cost function instead:

$$\min_{\mathbf{y} \in \mathbb{R}^{Nn}} \mathbf{E}_{\mathbf{W}} [\alpha F(\mathbf{y}) + \frac{1}{2} \mathbf{y}^\top (\mathbf{I} - \mathbf{W}) \mathbf{y}].$$

- And you are performing stochastic updates to it. So you are converging with some errors, as we know.
- Gradient tracking can be made converge to 0 in this case too, but I spare you the details.

Node asynchronous graphs?

- Edge asynchronicity may not be extremely realistic. We would like to have a node waking up and sending messages to its neighbors.

Node asynchronous graphs?

- Edge asynchronicity may not be extremely realistic. We would like to have a node waking up and sending messages to its neighbors.
- In this case W_k is not doubly stochastic, since the graph may be directed too! So we only have either row or column stochastic, meaning $W_k \mathbf{1} = \mathbf{1}$ or $\mathbf{1}^\top W_k = \mathbf{1}^\top$. So either consensus or the average are preserved, but not both.

Node asynchronous graphs?

- Edge asynchronicity may not be extremely realistic. We would like to have a node waking up and sending messages to its neighbors.
- In this case W_k is not doubly stochastic, since the graph may be directed too! So we only have either row or column stochastic, meaning $W_k \mathbf{1} = \mathbf{1}$ or $\mathbf{1}^\top W_k = \mathbf{1}^\top$. So either consensus or the average are preserved, but not both.
- This causes problems in convergence of the simple protocol to the average and additional sequences need to be computed and kept in memory to force convergence where we want it to be.

Node asynchronous graphs?

- Edge asynchronicity may not be extremely realistic. We would like to have a node waking up and sending messages to its neighbors.
- In this case W_k is not doubly stochastic, since the graph may be directed too! So we only have either row or column stochastic, meaning $W_k \mathbf{1} = \mathbf{1}$ or $\mathbf{1}^\top W_k = \mathbf{1}^\top$. So either consensus or the average are preserved, but not both.
- This causes problems in convergence of the simple protocol to the average and additional sequences need to be computed and kept in memory to force convergence where we want it to be.
- One notable example for column stochastic matrices: **Push-Sum protocol**.

Node asynchronous graphs?

- Edge asynchronicity may not be extremely realistic. We would like to have a node waking up and sending messages to its neighbors.
- In this case W_k is not doubly stochastic, since the graph may be directed too! So we only have either row or column stochastic, meaning $W_k \mathbf{1} = \mathbf{1}$ or $\mathbf{1}^\top W_k = \mathbf{1}^\top$. So either consensus or the average are preserved, but not both.
- This causes problems in convergence of the simple protocol to the average and additional sequences need to be computed and kept in memory to force convergence where we want it to be.
- One notable example for column stochastic matrices: **Push-Sum protocol**.
- Setting: Each node j wakes up and send two quantities to all its active neighbors: $x_k^j / d_{o,k}^j$ and supporting variable $\varphi_k^j / d_{o,k}^j$. Here d_o^j is the outer degree: the number of neighbors node j communicates to.

Node asynchronous graphs?

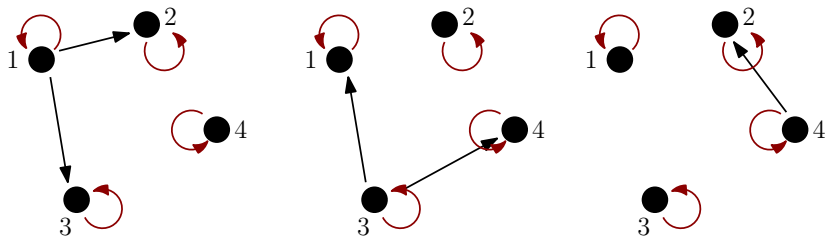
- Edge asynchronicity may not be extremely realistic. We would like to have a node waking up and sending messages to its neighbors.
- In this case W_k is not doubly stochastic, since the graph may be directed too! So we only have either row or column stochastic, meaning $W_k \mathbf{1} = \mathbf{1}$ or $\mathbf{1}^\top W_k = \mathbf{1}^\top$. So either consensus or the average are preserved, but not both.
- This causes problems in convergence of the simple protocol to the average and additional sequences need to be computed and kept in memory to force convergence where we want it to be.
- One notable example for column stochastic matrices: **Push-Sum protocol**.
- Setting: Each node j wakes up and send two quantities to all its active neighbors: $x_k^j / d_{o,k}^j$ and supporting variable $\varphi_k^j / d_{o,k}^j$. Here d_o^j is the outer degree: the number of neighbors node j communicates to.
- Each node i that receives the update, computes,

$$x_{k+1}^i = \sum_{j \in N_i^{\text{in}}} \frac{x_k^j}{d_{o,k}^j + 1}, \quad \varphi_{k+1}^i = \sum_{j \in N_i^{\text{in}}} \frac{\varphi_k^j}{d_{o,k}^j + 1}, \quad z_{k+1}^i = \frac{x_{k+1}^i}{\varphi_{k+1}^i},$$

here N_i^{in} is the set of neighbors that are communicating to i (incoming).

Node asynchronous graphs? An example

Consider a set of directed communication graphs 1, 2, ...



then, the update,

$$\mathbf{x}_{k+1}^i = \sum_{j \in N_i^{\text{in}}} \frac{\mathbf{x}_k^j}{d_{o,k}^j + 1}, \quad \varphi_{k+1}^i = \sum_{j \in N_i^{\text{in}}} \frac{\varphi_k^j}{d_{o,k}^j + 1}, \quad z_{k+1}^i = \frac{\mathbf{x}_{k+1}^i}{\varphi_{k+1}^i},$$

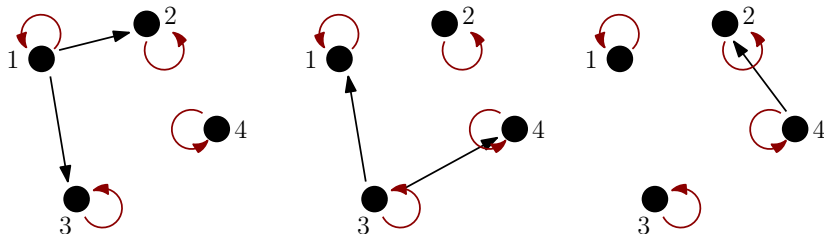
can be interpreted as,

$$\mathbf{y}_{k+1} = W_k \mathbf{y}_k, \quad \varphi_{k+1} = W_k \varphi_k, \quad z_{k+1}^i = \frac{\mathbf{x}_{k+1}^i}{\varphi_{k+1}^i}.$$

with W_k column stochastic. Can we write W_1 ?

Node asynchronous graphs? An example

Consider a set of directed communication graphs 1, 2, ...



then, the update,

$$\mathbf{x}_{k+1}^i = \sum_{j \in N_i^{\text{in}}} \frac{\mathbf{x}_k^j}{d_{o,k}^j + 1}, \quad \varphi_{k+1}^i = \sum_{j \in N_i^{\text{in}}} \frac{\varphi_k^j}{d_{o,k}^j + 1}, \quad z_{k+1}^i = \frac{\mathbf{x}_{k+1}^i}{\varphi_{k+1}^i},$$

•

$$W_1 = \begin{bmatrix} \frac{1}{3} & 0 & 0 & 0 \\ \frac{1}{3} & 1 & 0 & 0 \\ \frac{1}{3} & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Node asynchronous graphs?

- By initializing $\varphi_0 = \mathbf{1}$, we force z to converge to the mean!

Node asynchronous graphs?

- By initializing $\varphi_0 = \mathbf{1}$, we force z to converge to the mean!

Theorem 7 (Push-sum)

Consider a node-asynchronous protocol. Then the use of push-sum allows us to converge as,

$$\lim_{k \rightarrow \infty} z_k = \frac{\mathbf{1}\mathbf{1}^\top}{N} \mathbf{y}_0.$$

Node asynchronous graphs?

- By initializing $\varphi_0 = \mathbf{1}$, we force z to converge to the mean!

Theorem 7 (Push-sum)

Consider a node-asynchronous protocol. Then the use of push-sum allows us to converge as,

$$\lim_{k \rightarrow \infty} z_k = \frac{\mathbf{1}\mathbf{1}^\top}{N} \mathbf{y}_0.$$

- This is a gossip protocol, and the proof is quite beautiful, but brings us far from our intent.

Node asynchronous graphs?

- By initializing $\varphi_0 = \mathbf{1}$, we force z to converge to the mean!

Theorem 7 (Push-sum)

Consider a node-asynchronous protocol. Then the use of push-sum allows us to converge as,

$$\lim_{k \rightarrow \infty} z_k = \frac{\mathbf{1}\mathbf{1}^\top}{N} \mathbf{y}_0.$$

- This is a gossip protocol, and the proof is quite beautiful, but brings us far from our intent.
- In the harder setting, a lot of math is involved, but we can save most of the results we obtained, with a slow down in convergence rate due to the network.

Some references (useful to both FL and DO)

- L. Xiao, S. Boyd and S. Lall, **Distributed Average Consensus with Time-Varying Metropolis Weights**, 2006
- L. Xiao, S. Boyd, **Optimal Scaling of a Gradient Method for Distributed Resource Allocation**, 2006
- K. I. Tsianos, S. Lawlor and M. G. Rabbat, **Push-Sum Distributed Dual Averaging for Convex Optimization**, 2012

Distributed Optimization, dual methods

Let's revisit our problem

- We remind that the problem at hand is

$$(P) \quad \min_{\mathbf{x} \in \mathbb{R}^n} \sum_{i=1}^N f_i(\mathbf{x}).$$

and we proceed as in the first class by endowing each device with a copy of \mathbf{x} , so that the problem becomes,

$$(P') \quad \min_{\mathbf{x}^i \in \mathbb{R}^n, i=1, \dots, N} \sum_{i=1}^N f_i(\mathbf{x}^i) \quad \text{subject to } \mathbf{x}^i = \mathbf{x}^j, \forall i \sim j$$

Let's revisit our problem

- We remind that the problem at hand is

$$(P) \quad \min_{\mathbf{x} \in \mathbb{R}^n} \sum_{i=1}^N f_i(\mathbf{x}).$$

and we proceed as in the first class by endowing each device with a copy of \mathbf{x} , so that the problem becomes,

$$(P') \quad \min_{\mathbf{x}^i \in \mathbb{R}^n, i=1, \dots, N} \sum_{i=1}^N f_i(\mathbf{x}^i) \quad \text{subject to } \mathbf{x}^i = \mathbf{x}^j, \forall i \sim j$$

- We can then write (P') compactly as,

$$(P'') \quad \min_{\mathbf{y} \in \mathbb{R}^{Nn}} F(\mathbf{y}) \quad \text{subject to } A\mathbf{y} = 0$$

Let's recall that..

- The variable

$$\mathbf{x}_k^i \in \mathbf{R}^n$$

is the decision of agent i at iteration k , and

Let's recall that..

- The variable

$$\mathbf{x}_k^i \in \mathbf{R}^n$$

is the decision of agent i at iteration k , and

- The stacked version,

$$\mathbf{y} = \begin{bmatrix} \mathbf{x}^1 \\ \vdots \\ \mathbf{x}^N \end{bmatrix} \in \mathbf{R}^{Nn}$$

Let's recall that..

- The variable

$$\mathbf{x}_k^i \in \mathbf{R}^n$$

is the decision of agent i at iteration k , and

- The stacked version,

$$\mathbf{y} = \begin{bmatrix} \mathbf{x}^1 \\ \vdots \\ \mathbf{x}^N \end{bmatrix} \in \mathbf{R}^{Nn}$$

- Also, let f_i be L_i smooth and m_i strongly convex. Then,

$$\sum_{i=1}^N f_i(\mathbf{x}),$$

is L -smooth with $L = \sum_i L_i$ and $m = \sum_i m_i$. **However,**

Let's recall that..

- The variable

$$\mathbf{x}_k^i \in \mathbf{R}^n$$

is the decision of agent i at iteration k , and

- The stacked version,

$$\mathbf{y} = \begin{bmatrix} \mathbf{x}^1 \\ \vdots \\ \mathbf{x}^N \end{bmatrix} \in \mathbf{R}^{Nn}$$

- Also, let f_i be L_i smooth and m_i strongly convex. Then,

$$\sum_{i=1}^N f_i(\mathbf{x}),$$

is L -smooth with $L = \sum_i L_i$ and $m = \sum_i m_i$. **However,**

- The function,

$$F(\mathbf{y}) = \sum_{i=1}^N f_i(\mathbf{x}^i),$$

is L' -smooth with $L' = \max_i \{L_i\}$ and $m' = \min_i \{m_i\}$.

What's A ?

- We can then write (P') compactly as,

$$(P'') \quad \min_{\mathbf{y} \in \mathbb{R}^{N_n}} F(\mathbf{y}) \quad \text{subject to } A\mathbf{y} = 0$$

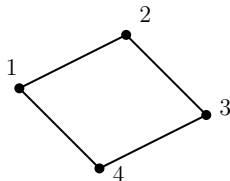
What's A ?

- We can then write (P') compactly as,

$$(P'') \quad \min_{\mathbf{y} \in \mathbb{R}^{N_n}} F(\mathbf{y}) \quad \text{subject to } A\mathbf{y} = 0$$

- Depending on the connectivity, and the number of constraints, A can be defined in different ways, for example,

$$A = \begin{bmatrix} I_n & -I_n & 0 & 0 \\ 0 & I_n & -I_n & 0 \\ 0 & 0 & I_n & -I_n \\ I_n & 0 & 0 & -I_n \end{bmatrix}$$



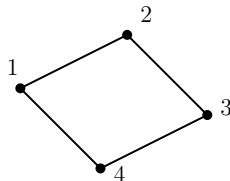
What's A ?

- We can then write (P') compactly as,

$$(P'') \quad \min_{\mathbf{y} \in \mathbb{R}^{N_n}} F(\mathbf{y}) \quad \text{subject to } A\mathbf{y} = 0$$

- Depending on the connectivity, and the number of constraints, A can be defined in different ways, for example,

$$A = \begin{bmatrix} I_n & -I_n & 0 & 0 \\ 0 & I_n & -I_n & 0 \\ 0 & 0 & I_n & -I_n \\ I_n & 0 & 0 & -I_n \end{bmatrix}$$



- Or,

$$A = \begin{bmatrix} -I_n & I_n & 0 & 0 \\ 0 & -I_n & I_n & 0 \\ 0 & 0 & -I_n & I_n \end{bmatrix}$$

and so forth..

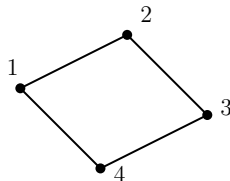
What's A ?

- We can then write (P') compactly as,

$$(P'') \quad \min_{\mathbf{y} \in \mathbb{R}^{N_n}} F(\mathbf{y}) \quad \text{subject to } A\mathbf{y} = 0$$

- Depending on the connectivity, and the number of constraints, A can be defined in different ways, for example,

$$A = \begin{bmatrix} I_n & -I_n & 0 & 0 \\ 0 & I_n & -I_n & 0 \\ 0 & 0 & I_n & -I_n \\ I_n & 0 & 0 & -I_n \end{bmatrix}$$



- Or,

$$A = \begin{bmatrix} -I_n & I_n & 0 & 0 \\ 0 & -I_n & I_n & 0 \\ 0 & 0 & -I_n & I_n \end{bmatrix}$$

and so forth..

- Can we now look at its dual problem?

Let's revisit duality

- Consider the problem

$$\min_{\mathbf{x} \in \mathbf{R}^n} f(\mathbf{x}) \quad \text{subject to } A\mathbf{x} = b,$$

for a convex function $f : \mathbf{R}^n \rightarrow \mathbf{R}$, and matrix $A \in \mathbf{R}^{p \times n}$, vector $b \in \mathbf{R}^p$ (with b in the image of A).

Let's revisit duality

- Consider the problem

$$\min_{\mathbf{x} \in \mathbf{R}^n} f(\mathbf{x}) \quad \text{subject to } A\mathbf{x} = b,$$

for a convex function $f : \mathbf{R}^n \rightarrow \mathbf{R}$, and matrix $A \in \mathbf{R}^{p \times n}$, vector $b \in \mathbf{R}^p$ (with b in the image of A).

- The Lagrangian function is defined as,

$$\mathcal{L}(\mathbf{x}, \lambda) = f(\mathbf{x}) + \lambda^\top (A\mathbf{x} - b),$$

where $\lambda \in \mathbf{R}^p$ are the Lagrangian multipliers (aka the dual variables)

Let's revisit duality

- Consider the problem

$$\min_{\mathbf{x} \in \mathbf{R}^n} f(\mathbf{x}) \quad \text{subject to } A\mathbf{x} = b,$$

for a convex function $f : \mathbf{R}^n \rightarrow \mathbf{R}$, and matrix $A \in \mathbf{R}^{p \times n}$, vector $b \in \mathbf{R}^p$ (with b in the image of A).

- The Lagrangian function is defined as,

$$\mathcal{L}(\mathbf{x}, \lambda) = f(\mathbf{x}) + \lambda^\top (A\mathbf{x} - b),$$

where $\lambda \in \mathbf{R}^p$ are the Lagrangian multipliers (aka the dual variables)

- The Lagrangian dual function is then,

$$q(\lambda) = \inf_{\mathbf{x} \in \mathbf{R}^n} \mathcal{L}(\mathbf{x}, \lambda) = -(f^*(-A^\top \lambda) + \lambda^\top b),$$

where f^* is the conjugate function of f , i.e., $f^*(\mathbf{y}) = \sup_{\mathbf{x}} \{\mathbf{y}^\top \mathbf{x} - f(\mathbf{x})\}$.

Homework. Prove the last equality.

Dual problems

- The dual problem is then,

$$\max_{\lambda \in \mathbb{R}^p} q(\lambda)$$

Dual problems

- The dual problem is then,

$$\max_{\lambda \in \mathbb{R}^p} q(\lambda)$$

- The dual problem is always convex, even if the primal is not and it provides a lower bound on the minimum of the primal problem, i.e.,

$$q^* \leq f^*.$$

Dual problems

- The dual problem is then,

$$\max_{\lambda \in \mathbb{R}^p} q(\lambda)$$

- The dual problem is always convex, even if the primal is not and it provides a lower bound on the minimum of the primal problem, i.e.,

$$q^* \leq f^*.$$

- Equality holds if some constraint qualification holds. For convex problems, if Slater's condition is verified (there exists a strictly feasible solution), then equality holds and we say that we have strong duality.

Dual problems

- The dual problem is then,

$$\max_{\lambda \in \mathbb{R}^p} q(\lambda)$$

- The dual problem is always convex, even if the primal is not and it provides a lower bound on the minimum of the primal problem, i.e.,

$$q^* \leq f^*.$$

- Equality holds if some constraint qualification holds. For convex problems, if Slater's condition is verified (there exists a strictly feasible solution), then equality holds and we say that we have strong duality.
- For this course, strong duality always holds because I don't look at dualizing inequality constraints.

Let's revisit our problem

- We remind that the problem at hand is

$$(P'') \quad \min_{\mathbf{y} \in \mathbb{R}^{N_n}} F(\mathbf{y}) \quad \text{subject to } A\mathbf{y} = 0$$

whose Lagrangian is

$$\mathcal{L}(\mathbf{y}, \lambda) = F(\mathbf{y}) + \lambda^\top A\mathbf{y}$$

and dual problem,

$$\max_{\lambda} \left[q(\lambda) := \inf_{\mathbf{y}} \{ F(\mathbf{y}) + \lambda^\top A\mathbf{y} \} = -F^*(-A^\top \lambda) \right].$$

Let's revisit our problem

- We remind that the problem at hand is

$$(P'') \quad \min_{\mathbf{y} \in \mathbb{R}^{N_n}} F(\mathbf{y}) \quad \text{subject to } A\mathbf{y} = 0$$

whose Lagrangian is

$$\mathcal{L}(\mathbf{y}, \lambda) = F(\mathbf{y}) + \lambda^\top A\mathbf{y}$$

and dual problem,

$$\max_{\lambda} \left[q(\lambda) := \inf_{\mathbf{y}} \{ F(\mathbf{y}) + \lambda^\top A\mathbf{y} \} = -F^*(-A^\top \lambda) \right].$$

- Can we set up a dual ascent algorithm?

Dual ascent = gradient descent for dual

- Start with a λ_0 and iterate:

Dual ascent = gradient descent for dual

- Start with a λ_0 and iterate:
- $\lambda_{k+1} = \lambda_k + \alpha_k \mathbf{v}_k$, with $\mathbf{v}_k \in \partial q(\lambda_k)$, for all $k = 0, 1, \dots$
Rem: the dual function may not be differentiable even if F is so, so we need the sub-differential

Dual ascent = gradient descent for dual

- Start with a λ_0 and iterate:
- $\lambda_{k+1} = \lambda_k + \alpha_k \mathbf{v}_k$, with $\mathbf{v}_k \in \partial q(\lambda_k)$, for all $k = 0, 1, \dots$
Rem: the dual function may not be differentiable even if F is so, so we need the sub-differential
- What's $\partial q(\lambda_k)$?

Dual ascent = gradient descent for dual

- Start with a λ_0 and iterate:
- $\lambda_{k+1} = \lambda_k + \alpha_k \mathbf{v}_k$, with $\mathbf{v}_k \in \partial q(\lambda_k)$, for all $k = 0, 1, \dots$
Rem: the dual function may not be differentiable even if F is so, so we need the sub-differential
- What's $\partial q(\lambda_k)$?
- Start with

$$\partial_{\lambda} q(\lambda_k) = -A \partial_{\lambda} [-F^*(-A^{\top} \lambda_k)]$$

Then, we know that for convex functions $(\partial_v F)^{-1}(v) = \partial_u F^*(u)$, and in addition let,

$$\begin{aligned} \mathbf{y}^*(\lambda) &:= \arg \min_{\mathbf{y}} \mathcal{L}(\mathbf{y}, \lambda) \iff \partial F(\mathbf{y}^*(\lambda)) + A^{\top} \lambda \ni \mathbf{0} \iff \\ &\mathbf{y}^*(\lambda) = (\partial F)^{-1}(-A^{\top} \lambda) = \partial F^*(-A^{\top} \lambda) \end{aligned}$$

Therefore,

$$\partial q(\lambda_k) = A \mathbf{y}^*(\lambda_k) \quad \text{residual map!}$$

Dual decomposition

- Therefore the dual ascent yields, Start with a λ_0 and iterate:

Dual decomposition

- Therefore the dual ascent yields, Start with a λ_0 and iterate:
- Solve locally

$$\mathbf{y}^*(\lambda) := \arg \min_{\mathbf{y}} \mathcal{L}(\mathbf{y}, \lambda_k) = \arg \min_{\mathbf{x}^1, \dots, \mathbf{x}^N} \left\{ \sum_{i=1}^n f_i(\mathbf{x}^i) + \lambda_k^\top A[\mathbf{x}^1, \dots, \mathbf{x}^N]^\top \right\}$$

Upon dividing $A = [A_1 | \dots | A_N]$,

$$\iff \mathbf{x}^{i,*}(\lambda_k) := \arg \min_{\mathbf{x}_i} \{ f_i(\mathbf{x}^i) + \lambda_k^\top A_i \mathbf{x}^i \} \quad \forall i,$$

Dual decomposition

- Therefore the dual ascent yields, Start with a λ_0 and iterate:
- Solve locally

$$\mathbf{y}^*(\lambda) := \arg \min_{\mathbf{y}} \mathcal{L}(\mathbf{y}, \lambda_k) = \arg \min_{\mathbf{x}^1, \dots, \mathbf{x}^N} \left\{ \sum_{i=1}^n f_i(\mathbf{x}^i) + \lambda_k^\top A[\mathbf{x}^1, \dots, \mathbf{x}^N]^\top \right\}$$

Upon dividing $A = [A_1 | \dots | A_N]$,

$$\iff \mathbf{x}^{i,*}(\lambda_k) := \arg \min_{\mathbf{x}_i} \{ f_i(\mathbf{x}^i) + \lambda_k^\top A_i \mathbf{x}^i \} \quad \forall i,$$

- Send and update $\mathbf{x}^{i,*}(\lambda_k)$ to all, and gather,

$$\partial q(\lambda_k) = A \mathbf{y}^*(\lambda_k) = \sum_i A_i \mathbf{x}^{i,*}(\lambda_k)$$

Dual decomposition

- Therefore the dual ascent yields, Start with a λ_0 and iterate:
- Solve locally

$$\mathbf{y}^*(\lambda) := \arg \min_{\mathbf{y}} \mathcal{L}(\mathbf{y}, \lambda_k) = \arg \min_{\mathbf{x}^1, \dots, \mathbf{x}^N} \left\{ \sum_{i=1}^n f_i(\mathbf{x}^i) + \lambda_k^\top A[\mathbf{x}^1, \dots, \mathbf{x}^N]^\top \right\}$$

Upon dividing $A = [A_1 | \dots | A_N]$,

$$\iff \mathbf{x}^{i,*}(\lambda_k) := \arg \min_{\mathbf{x}_i} \{ f_i(\mathbf{x}^i) + \lambda_k^\top A_i \mathbf{x}^i \} \quad \forall i,$$

- Send and update $\mathbf{x}^{i,*}(\lambda_k)$ to all, and gather,

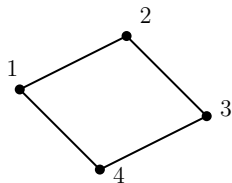
$$\partial q(\lambda_k) = A \mathbf{y}^*(\lambda_k) = \sum_i A_i \mathbf{x}^{i,*}(\lambda_k)$$

- Update then, $\lambda_{k+1} = \lambda_k + \alpha_k \sum_i A_i \mathbf{x}^{i,*}(\lambda_k)$, for all $k = 0, 1, \dots$

Dual decomposition: fix the ideas on A

- Depending on the connectivity, and the number of constraints, A can be defined in different ways, for example,

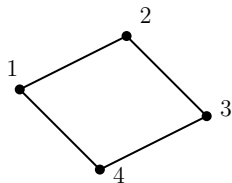
$$A = \left[\begin{array}{c|c|c|c} I_n & -I_n & 0 & 0 \\ 0 & I_n & -I_n & 0 \\ 0 & 0 & I_n & -I_n \\ I_n & 0 & 0 & -I_n \end{array} \right] =: [A_1|A_2|A_3|A_4]$$



Dual decomposition: fix the ideas on A

- Depending on the connectivity, and the number of constraints, A can be defined in different ways, for example,

$$A = \left[\begin{array}{c|c|c|c} I_n & -I_n & 0 & 0 \\ 0 & I_n & -I_n & 0 \\ 0 & 0 & I_n & -I_n \\ I_n & 0 & 0 & -I_n \end{array} \right] =: [A_1|A_2|A_3|A_4]$$



- And

$$A^{\text{y}}(\lambda_k) = [A_1|A_2|A_3|A_4] \begin{bmatrix} \textcolor{red}{x}^{1,*}(\lambda_k) \\ \textcolor{red}{x}^{2,*}(\lambda_k) \\ \textcolor{red}{x}^{3,*}(\lambda_k) \\ \textcolor{red}{x}^{4,*}(\lambda_k) \end{bmatrix} = \sum_{i=1}^4 A_i \textcolor{red}{x}^{i,*}(\lambda_k).$$

Dual decomposition

- The method is called dual decomposition, since you decompose the primal problem via the dual variables;

Dual decomposition

- The method is called dual decomposition, since you decompose the primal problem via the dual variables;
- The dual variables are updated globally, the primal locally, and this is convenient if f_i depends on data that you can't share/ don't want to!

Dual decomposition

- The method is called dual decomposition, since you decompose the primal problem via the dual variables;
- The dual variables are updated globally, the primal locally, and this is convenient if f_i depends on data that you can't share/ don't want to!
- Remember that $\lambda \in \mathbf{R}^{nC}$, where C are the number of constraints.

Dual decomposition

- The method is called dual decomposition, since you decompose the primal problem via the dual variables;
- The dual variables are updated globally, the primal locally, and this is convenient if f_i depends on data that you can't share/ don't want to!
- Remember that $\lambda \in \mathbf{R}^{nC}$, where C are the number of constraints.
- Now, we can try to let each node updates the λ 's that belong to the constraints that it sees. For example, node i could update the λ for $\mathbf{x}_i - \mathbf{x}_j = 0$, for all j it can communicate to.

Dual decomposition

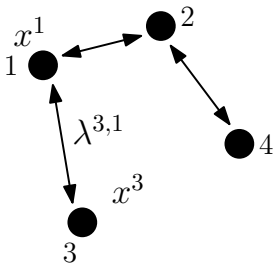
- The method is called dual decomposition, since you decompose the primal problem via the dual variables;
- The dual variables are updated globally, the primal locally, and this is convenient if f_i depends on data that you can't share/ don't want to!
- Remember that $\lambda \in \mathbf{R}^{nC}$, where C are the number of constraints.
- Now, we can try to let each node updates the λ 's that belong to the constraints that it sees. For example, node i could update the λ for $\mathbf{x}_i - \mathbf{x}_j = 0$, for all j it can communicate to.
- So, λ can also live locally on the edges!

Dual decomposition

- The method is called dual decomposition, since you decompose the primal problem via the dual variables;
- The dual variables are updated globally, the primal locally, and this is convenient if f_i depends on data that you can't share/ don't want to!
- Remember that $\lambda \in \mathbf{R}^{nC}$, where C are the number of constraints.
- Now, we can try to let each node updates the λ 's that belong to the constraints that it sees. For example, node i could update the λ for $x_i - x_j = 0$, for all j it can communicate to.
- So, λ can also live locally on the edges!
- Let's write this properly

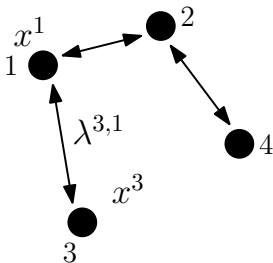
Dual decomposition: a local update rule

- Consider a graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$, undirected. We indicate $i \sim j$ if there is an edge between i and j .



Dual decomposition: a local update rule

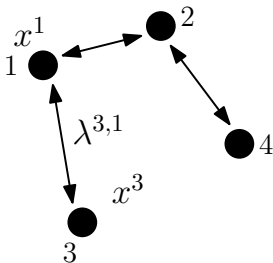
- Consider a graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$, undirected. We indicate $i \sim j$ if there is an edge between i and j .



- Let A be the collection of $x^i - x^j$ for the edge set. Then the associated dual variable is λ^{ij} and that can live on the edge i, j .

Dual decomposition: a local update rule

- Consider a graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$, undirected. We indicate $i \sim j$ if there is an edge between i and j .



- Let A be the collection of $\mathbf{x}^i - \mathbf{x}^j$ for the edge set. Then the associated dual variable is λ^{ij} and that can live on the edge i, j .
- The number of edges is E , and we index the set of edges as,

$$\mathcal{E} = \{(i, j) \mid j < i, i \sim j\}.$$

Since the graph is undirected, there is no reason to count an edge twice, hence $j < i$.

Dual decomposition: a local update rule

- Consider a graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$, undirected. We indicate $i \sim j$ if there is an edge between i and j .
- Let A be the collection of $\mathbf{x}^i - \mathbf{x}^j$ for the edge set. Then the associated dual variable is λ^{ij} and that can live on the edge i, j .
- The number of edges is E , and we index the set of edges as,

$$\mathcal{E} = \{(i, j) \mid j < i, i \sim j\}.$$

Since the graph is undirected, there is no reason to count an edge twice, hence $j < i$.

- As such $\mathbf{x}^i \in \mathbf{R}^n$, $A \in \mathbf{R}^{nE \times nN}$, if E edges and N nodes.

Dual decomposition: a local update rule

- Consider a graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$, undirected. We indicate $i \sim j$ if there is an edge between i and j .
- Let A be the collection of $\mathbf{x}^i - \mathbf{x}^j$ for the edge set. Then the associated dual variable is λ^{ij} and that can live on the edge i, j .
- The number of edges is E , and we index the set of edges as,

$$\mathcal{E} = \{(i, j) \mid j < i, i \sim j\}.$$

Since the graph is undirected, there is no reason to count an edge twice, hence $j < i$.

- As such $\mathbf{x}^i \in \mathbf{R}^n$, $A \in \mathbf{R}^{nE \times nN}$, if E edges and N nodes.
- As such, λ^{ij} is associated to the edge of nodes i and j , and they can update it as,

$$\lambda_{k+1}^{ij} = \lambda_k^{ij} + \alpha_k [\mathbf{x}^{i,*}(\lambda_k) - \mathbf{x}^{j,*}(\lambda_k)], \quad \forall k = 0, 1, \dots$$

Dual decomposition: a local update rule

- Initialize $\lambda_0^{ij} \in \mathbf{R}^n$ for $(i,j) \in \mathcal{E}, j < i$, interpret λ^{ij} for $j > i$ as λ^{ji} .

Dual decomposition: a local update rule

- Initialize $\lambda_0^{ij} \in \mathbf{R}^n$ for $(i, j) \in \mathcal{E}, j < i$, interpret λ^{ij} for $j > i$ as λ^{ji} .
- For all nodes i do,

$$\mathbf{x}^{i,*}(\lambda_k) := \arg \min_{\mathbf{x}_i} \left\{ f_i(\mathbf{x}^i) + \sum_{i \sim j \equiv j \in N_i} (-1)^a \lambda_k^{ij, \top} \mathbf{x}^i \right\}$$

where

$$a = \begin{cases} 0 & \text{if } i > j \\ 1 & \text{otherwise} \end{cases}$$

Dual decomposition: a local update rule

- Initialize $\lambda_0^{ij} \in \mathbf{R}^n$ for $(i, j) \in \mathcal{E}, j < i$, interpret λ^{ij} for $j > i$ as λ^{ji} .
- For all nodes i do,

$$\mathbf{x}^{i,*}(\lambda_k) := \arg \min_{\mathbf{x}_i} \left\{ f_i(\mathbf{x}^i) + \sum_{i \sim j \equiv j \in N_i} (-1)^a \lambda_k^{ij, \top} \mathbf{x}^i \right\}$$

where

$$a = \begin{cases} 0 & \text{if } i > j \\ 1 & \text{otherwise} \end{cases}$$

- Send $\mathbf{x}^{i,*}(\lambda_k)$ to neighbors

Dual decomposition: a local update rule

- Initialize $\lambda_0^{ij} \in \mathbf{R}^n$ for $(i, j) \in \mathcal{E}, j < i$, interpret λ^{ij} for $j > i$ as λ^{ji} .
- For all nodes i do,

$$\mathbf{x}^{i,*}(\lambda_k) := \arg \min_{\mathbf{x}_i} \left\{ f_i(\mathbf{x}^i) + \sum_{i \sim j \equiv j \in N_i} (-1)^a \lambda_k^{ij, \top} \mathbf{x}^i \right\}$$

where

$$a = \begin{cases} 0 & \text{if } i > j \\ 1 & \text{otherwise} \end{cases}$$

- Send $\mathbf{x}^{i,*}(\lambda_k)$ to neighbors
- Update for node i, j :

$$\lambda_{k+1}^{ij} = \lambda_k^{ij} + \alpha_k [\mathbf{x}^{i,*}(\lambda_k) - \mathbf{x}^{j,*}(\lambda_k)], \quad j < i$$

Dual decomposition: a local update rule

- Initialize $\lambda_0^{ij} \in \mathbf{R}^n$ for $(i, j) \in \mathcal{E}, j < i$, interpret λ^{ij} for $j > i$ as λ^{ji} .
- For all nodes i do,

$$\mathbf{x}^{i,*}(\lambda_k) := \arg \min_{\mathbf{x}_i} \left\{ f_i(\mathbf{x}^i) + \sum_{i \sim j \equiv j \in N_i} (-1)^a \lambda_k^{ij, \top} \mathbf{x}^i \right\}$$

where

$$a = \begin{cases} 0 & \text{if } i > j \\ 1 & \text{otherwise} \end{cases}$$

- Send $\mathbf{x}^{i,*}(\lambda_k)$ to neighbors
- Update for node i, j :

$$\lambda_{k+1}^{ij} = \lambda_k^{ij} + \alpha_k [\mathbf{x}^{i,*}(\lambda_k) - \mathbf{x}^{j,*}(\lambda_k)], \quad j < i$$

- This has two local updates and one communication round (synchronous, undirected)

Dual decomposition: convergence I

- We look now at the convergence of dual decomposition. We need to recall some of the properties of the dual function. In particular, let $\sigma(A)$ be the singular values of A .

If F is m -strongly convex then $-q$ is $\sigma_{\max}^2(A)/m$ smooth;

If F is L -smooth then $-q$ is $\sigma_{\min}^2(A)/L$ strongly convex;

Homework. Remind yourself of why it is so (check OPT202).

Dual decomposition: convergence I

- We look now at the convergence of dual decomposition. We need to recall some of the properties of the dual function. In particular, let $\sigma(A)$ be the singular values of A .

If F is m -strongly convex then $-q$ is $\sigma_{\max}^2(A)/m$ smooth;

If F is L -smooth then $-q$ is $\sigma_{\min}^2(A)/L$ strongly convex;

Homework. Remind yourself of why it is so (check OPT202).

- Then we have the following

Dual decomposition: convergence I

- We look now at the convergence of dual decomposition. We need to recall some of the properties of the dual function. In particular, let $\sigma(A)$ be the singular values of A .

If F is m -strongly convex then $-q$ is $\sigma_{\max}^2(A)/m$ smooth;

If F is L -smooth then $-q$ is $\sigma_{\min}^2(A)/L$ strongly convex;

Homework. Remind yourself of why it is so (check OPT202).

- Then we have the following

Theorem 8 (Dual ascent convergence)

Consider problem (P) and the dual decomposition approach for a certain connection matrix A . If f_i 's are m -strongly convex and L -smooth, then we can select $\alpha < 2m/\sigma_{\max}^2(A)$ and obtaining a linear rate of convergence as,

$$\|\lambda_k - \lambda^*\| \leq \left[\max\left\{ \left| 1 - \alpha \frac{\sigma_{\max}^2(A)}{m} \right|, \left| 1 - \alpha \frac{\sigma_{\min}^2(A)}{L} \right| \right\} \right]^k \|\lambda_0 - \lambda^*\|$$

and

$$\|\mathbf{x}_k^i - \mathbf{x}^*\| \leq \frac{\sigma_{\max}(A)}{m} \|\lambda_k - \lambda^*\|.$$

Dual decomposition: convergence II

- Proof follows from OPT201/202. **Homework.** Work it out.

Dual decomposition: convergence II

- Proof follows from OPT201/202. **Homework.** Work it out.
- Be careful with the f_i 's: they need to be strongly convex.

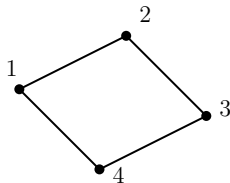
Dual decomposition: convergence II

- Convergence is proven when A is full row rank (so that $\sigma_{\min} > 0$). But why would it be?

Dual decomposition: convergence II

- Convergence is proven when A is full row rank (so that $\sigma_{\min} > 0$). But why would it be?
- Depending on the connectivity, and the number of constraints, A can be defined in different ways, for example,

$$A = \begin{bmatrix} I_n & -I_n & 0 & 0 \\ 0 & I_n & -I_n & 0 \\ 0 & 0 & I_n & -I_n \\ I_n & 0 & 0 & -I_n \end{bmatrix}$$

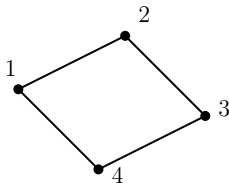


which is not full row rank;

Dual decomposition: convergence II

- Convergence is proven when A is full row rank (so that $\sigma_{\min} > 0$). But why would it be?
- Depending on the connectivity, and the number of constraints, A can be defined in different ways, for example,

$$A = \begin{bmatrix} I_n & -I_n & 0 & 0 \\ 0 & I_n & -I_n & 0 \\ 0 & 0 & I_n & -I_n \\ I_n & 0 & 0 & -I_n \end{bmatrix}$$



which is not full row rank;

- Or,

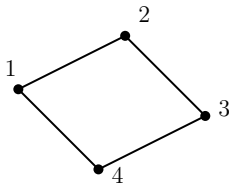
$$A = \begin{bmatrix} -I_n & I_n & 0 & 0 \\ 0 & -I_n & I_n & 0 \\ 0 & 0 & -I_n & I_n \end{bmatrix}$$

which is full row rank !

Dual decomposition: convergence II

- Convergence is proven when A is full row rank (so that $\sigma_{\min} > 0$). But why would it be?
- Depending on the connectivity, and the number of constraints, A can be defined in different ways, for example,

$$A = \begin{bmatrix} I_n & -I_n & 0 & 0 \\ 0 & I_n & -I_n & 0 \\ 0 & 0 & I_n & -I_n \\ I_n & 0 & 0 & -I_n \end{bmatrix}$$



which is not full row rank;

- Or,

$$A = \begin{bmatrix} -I_n & I_n & 0 & 0 \\ 0 & -I_n & I_n & 0 \\ 0 & 0 & -I_n & I_n \end{bmatrix}$$

which is full row rank !

- The more constraints, the better the mixing is, however we lose the rank ...

Dual decomposition: convergence II

- In the case A is not full row rank? λ^* is not unique and $-q$ is not strongly convex. But, we can look at **restricted strong convexity instead** !

Dual decomposition: convergence II

- In the case A is not full row rank? λ^* is not unique and $-q$ is not strongly convex. But, we can look at **restricted strong convexity instead** !

Let's see how to do it

Dual decomposition: convergence II

- In the case A is not full row rank? λ^* is not unique and $-q$ is not strongly convex. But, we can look at **restricted strong convexity instead** !

Let's see how to do it

- First $\lambda^* = \lambda_0^* + \lambda_1^*$, with $\lambda_0^* \in \text{im}(A)$ and $\lambda_1^* \in \text{null}(A^\top)$. One can show that λ_0^* is unique and we concentrate on that one, since λ_1^* is redundant $A^\top \lambda_1^* = 0$.

Dual decomposition: convergence II

- In the case A is not full row rank? λ^* is not unique and $-q$ is not strongly convex. But, we can look at **restricted strong convexity instead** !

Let's see how to do it

- First $\lambda^* = \lambda_0^* + \lambda_1^*$, with $\lambda_0^* \in \text{im}(A)$ and $\lambda_1^* \in \text{null}(A^\top)$. One can show that λ_0^* is unique and we concentrate on that one, since λ_1^* is redundant $A^\top \lambda_1^* = 0$.
- Start with $\lambda_0 \in \text{im}(A)$; then, it is direct that:

$$\lambda_{k+1} = \lambda_k + \alpha A \mathbf{x} \in \text{im}(A)$$

Dual decomposition: convergence II

- In the case A is not full row rank? λ^* is not unique and $-q$ is not strongly convex. But, we can look at **restricted strong convexity instead** !

Let's see how to do it

- First $\lambda^* = \lambda_0^* + \lambda_1^*$, with $\lambda_0^* \in \text{im}(A)$ and $\lambda_1^* \in \text{null}(A^\top)$. One can show that λ_0^* is unique and we concentrate on that one, since λ_1^* is redundant $A^\top \lambda_1^* = 0$.
- Start with $\lambda_0 \in \text{im}(A)$; then, it is direct that:

$$\lambda_{k+1} = \lambda_k + \alpha A \mathbf{x} \in \text{im}(A)$$

- Then, can we say that (restricted strong convexity)

$$(\partial q(\lambda) - \partial q(\lambda'))^\top (\lambda' - \lambda) \geq \sigma_{\min}^2 / L \|\lambda' - \lambda\|^2, \quad \forall \lambda, \lambda' \in \text{im}(A)?$$

with σ_{\min} the minimum non-zero singular value of A ?

Dual decomposition: convergence II

- Yes, substitute $\partial q(\lambda) = A\partial F^*(-A^\top \lambda)$, then,

$$\begin{aligned}(\partial F^*(-A^\top \lambda) - \partial F^*(-A^\top \lambda'))^\top A^\top (\lambda' - \lambda) &= \\(\partial F^*(-A^\top \lambda) - \partial F^*(-A^\top \lambda'))^\top (-A^\top \lambda + A^\top \lambda') &\geq \\ \text{steps from OPT202..} &\geq 1/L \|A^\top (\lambda' - \lambda)\|^2 \geq \sigma_{\min}^2 / L \|\lambda' - \lambda\|^2.\end{aligned}$$

Dual decomposition: convergence II

- Yes, substitute $\partial q(\lambda) = A\partial F^*(-A^\top \lambda)$, then,

$$\begin{aligned} (\partial F^*(-A^\top \lambda) - \partial F^*(-A^\top \lambda'))^\top A^\top (\lambda' - \lambda) &= \\ (\partial F^*(-A^\top \lambda) - \partial F^*(-A^\top \lambda'))^\top (-A^\top \lambda + A^\top \lambda') &\geq \\ \text{steps from OPT202..} \geq 1/L \|A^\top (\lambda' - \lambda)\|^2 &\geq \sigma_{\min}^2 / L \|\lambda' - \lambda\|^2. \end{aligned}$$

- The latest step is due to the fact that $A^\top (\lambda - \lambda') = 0$ iff $\lambda = \lambda'$.

Dual decomposition: convergence II

- Yes, substitute $\partial q(\lambda) = A\partial F^*(-A^\top \lambda)$, then,

$$\begin{aligned}(\partial F^*(-A^\top \lambda) - \partial F^*(-A^\top \lambda'))^\top A^\top (\lambda' - \lambda) &= \\(\partial F^*(-A^\top \lambda) - \partial F^*(-A^\top \lambda'))^\top (-A^\top \lambda + A^\top \lambda') &\geq \\ \text{steps from OPT202..} &\geq 1/L \|A^\top (\lambda' - \lambda)\|^2 \geq \sigma_{\min}^2 / L \|\lambda' - \lambda\|^2.\end{aligned}$$

- The latest step is due to the fact that $A^\top (\lambda - \lambda') = 0$ iff $\lambda = \lambda'$.
- So the Theorem becomes:

Dual decomposition: convergence II

- Yes, substitute $\partial q(\lambda) = A\partial F^*(-A^\top \lambda)$, then,

$$\begin{aligned}(\partial F^*(-A^\top \lambda) - \partial F^*(-A^\top \lambda'))^\top A^\top (\lambda' - \lambda) &= \\(\partial F^*(-A^\top \lambda) - \partial F^*(-A^\top \lambda'))^\top (-A^\top \lambda + A^\top \lambda') &\geq \\ \text{steps from OPT202..} \geq 1/L \|A^\top (\lambda' - \lambda)\|^2 &\geq \sigma_{\min}^2 / L \|\lambda' - \lambda\|^2.\end{aligned}$$

- The latest step is due to the fact that $A^\top (\lambda - \lambda') = 0$ iff $\lambda = \lambda'$.
- So the Theorem becomes:

Theorem 9 (Dual decomposition convergence)

Select $\alpha < 2m/\sigma_{\max}^2(A)$ and $\lambda_0 \in \text{im}(A)$, and you obtain a linear rate of convergence as,

$$\|\lambda_k - \lambda^*\| \leq \left[\max\left\{ \left| 1 - \alpha \frac{\sigma_{\max}^2(A)}{m} \right|, \left| 1 - \alpha \frac{\sigma_{\min}^2(A)}{L} \right| \right\} \right]^k \|\lambda_0 - \lambda^*\|$$

Dual decomposition: convergence II

- Yes, substitute $\partial q(\lambda) = A\partial F^*(-A^\top \lambda)$, then,

$$\begin{aligned}(\partial F^*(-A^\top \lambda) - \partial F^*(-A^\top \lambda'))^\top A^\top (\lambda' - \lambda) &= \\(\partial F^*(-A^\top \lambda) - \partial F^*(-A^\top \lambda'))^\top (-A^\top \lambda + A^\top \lambda') &\geq \\ \text{steps from OPT202..} \geq 1/L \|A^\top (\lambda' - \lambda)\|^2 &\geq \sigma_{\min}^2 / L \|\lambda' - \lambda\|^2.\end{aligned}$$

- The latest step is due to the fact that $A^\top (\lambda - \lambda') = 0$ iff $\lambda = \lambda'$.
- So the Theorem becomes:

Theorem 9 (Dual decomposition convergence)

Select $\alpha < 2m/\sigma_{\max}^2(A)$ and $\lambda_0 \in \text{im}(A)$, and you obtain a linear rate of convergence as,

$$\|\lambda_k - \lambda^*\| \leq \left[\max\left\{ \left| 1 - \alpha \frac{\sigma_{\max}^2(A)}{m} \right|, \left| 1 - \alpha \frac{\sigma_{\min}^2(A)}{L} \right| \right\} \right]^k \|\lambda_0 - \lambda^*\|$$

- Note: $\lambda_0 = \mathbf{0} \in \text{im}(A)$

Dual decomposition: convergence III

- The step size can be chosen constant and you still converge linearly ! (As in the gradient tracking case); but the step size has to be chosen carefully

Dual decomposition: convergence III

- The step size can be chosen constant and you still converge linearly ! (As in the gradient tracking case); but the step size has to be chosen carefully
- Convergence depends on the graph, and in particular on $\sigma_{\min}, \sigma_{\max}$
It is not only a matter of function properties, but also graph properties!

Dual decomposition: convergence III

- The step size can be chosen constant and you still converge linearly ! (As in the gradient tracking case); but the step size has to be chosen carefully
- Convergence depends on the graph, and in particular on $\sigma_{\min}, \sigma_{\max}$
It is not only a matter of function properties, but also graph properties!
- Furthermore, we can show that,

$$\|\mathbf{x}^{i,*}(\lambda_k) - \mathbf{x}^*\| \leq \frac{\sigma_{\max}}{m} \|\lambda_k - \lambda^*\| \quad \text{Primal Recovery}$$

Dual decomposition: convergence III

- The step size can be chosen constant and you still converge linearly ! (As in the gradient tracking case); but the step size has to be chosen carefully
- Convergence depends on the graph, and in particular on $\sigma_{\min}, \sigma_{\max}$
It is not only a matter of function properties, but also graph properties!
- Furthermore, we can show that,

$$\|\mathbf{x}^{i,*}(\lambda_k) - \mathbf{x}^*\| \leq \frac{\sigma_{\max}}{m} \|\lambda_k - \lambda^*\| \quad \text{Primal Recovery}$$

- Convergence is trickier when F is not strongly convex, since q will not be differentiable, so it is also hard to recover the primal solution here (see OPT202)

Dual decomposition: convergence III

- The step size can be chosen constant and you still converge linearly ! (As in the gradient tracking case); but the step size has to be chosen carefully
- Convergence depends on the graph, and in particular on $\sigma_{\min}, \sigma_{\max}$
It is not only a matter of function properties, but also graph properties!
- Furthermore, we can show that,

$$\|\mathbf{x}^{i,*}(\lambda_k) - \mathbf{x}^*\| \leq \frac{\sigma_{\max}}{m} \|\lambda_k - \lambda^*\| \quad \text{Primal Recovery}$$

- Convergence is trickier when F is not strongly convex, since q will not be differentiable, so it is also hard to recover the primal solution here (see OPT202)
- Convergence may be complicated when communication is directed, or asynchronous, or you have latencies, as in the first class

Dual decomposition: numerical result (from 1st class)

Here it is useful to introduce some new data structure,

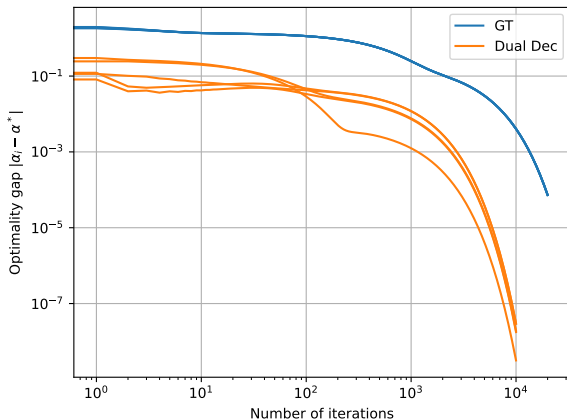
```
Edge_map = dict(), Edge_map[1] = (node_1, node_2)
```

```
Inverse_edge_map = dict(), Inverse_edge_map[(node_1, node_2)] = 1
```

```
Neighbors_map = dict(), Neighbors_map[1] = [2, 5]
```

Dual decomposition: numerical result

Same kernel setting, but now you can go to stepsize $s = 0.1$, since.. (rem: $m = 1/5$)



Rem: number of iterations = number of communication rounds.

Class 3

The Alternating Direction Method of Multipliers (ADMM)

- To overcome the drawbacks of dual decomposition, we introduce another iterative method: the ADMM. We are going to develop it for a general case and then focus it on our distributed problem

The Alternating Direction Method of Multipliers (ADMM)

- To overcome the drawbacks of dual decomposition, we introduce another iterative method: the ADMM. We are going to develop it for a general case and then focus it on our distributed problem
- Consider the problem:

$$\begin{array}{ll} \min_{\mathbf{x} \in \mathbb{R}^n, \mathbf{y} \in \mathbb{R}^p} & f(\mathbf{x}) + g(\mathbf{y}) \\ \text{subject to} & A\mathbf{x} + B\mathbf{y} = \mathbf{c}. \end{array}$$

For well-defined matrices and vectors, as well as convex functions f, g .

The Alternating Direction Method of Multipliers (ADMM)

- To overcome the drawbacks of dual decomposition, we introduce another iterative method: the ADMM. We are going to develop it for a general case and then focus it on our distributed problem
- Consider the problem:

$$\begin{array}{ll} \min_{\mathbf{x} \in \mathbb{R}^n, \mathbf{y} \in \mathbb{R}^p} & f(\mathbf{x}) + g(\mathbf{y}) \\ \text{subject to} & A\mathbf{x} + B\mathbf{y} = \mathbf{c}. \end{array}$$

For well-defined matrices and vectors, as well as convex functions f, g .

- Define the augmented Lagrangian,

$$\mathcal{L}_\beta(\mathbf{x}, \mathbf{y}, \lambda) := f(\mathbf{x}) + g(\mathbf{y}) + \lambda^\top (A\mathbf{x} + B\mathbf{y} - \mathbf{c}) + \frac{\beta}{2} \|A\mathbf{x} + B\mathbf{y} - \mathbf{c}\|^2,$$

for **any** scalar $\beta > 0$.

ADMM: iterations

- Then the ADMM does: start with an initial value for $\mathbf{x}_0, \mathbf{y}_0, \lambda_0$,

ADMM: iterations

- Then the ADMM does: start with an initial value for $\mathbf{x}_0, \mathbf{y}_0, \lambda_0$,
- Iterate:

$$\mathbf{x}_{k+1} = \arg \min_{\mathbf{x} \in \mathbb{R}^n} \mathcal{L}_\beta(\mathbf{x}, \mathbf{y}_k, \lambda_k)$$

$$\mathbf{y}_{k+1} = \arg \min_{\mathbf{y} \in \mathbb{R}^p} \mathcal{L}_\beta(\mathbf{x}_{k+1}, \mathbf{y}, \lambda_k)$$

$$\lambda_{k+1} = \lambda_k + \beta(A\mathbf{x}_{k+1} + B\mathbf{y}_{k+1} - c)$$

ADMM: iterations

- Then the ADMM does: start with an initial value for $\mathbf{x}_0, \mathbf{y}_0, \lambda_0$,
- Iterate:

$$\mathbf{x}_{k+1} = \arg \min_{\mathbf{x} \in \mathbb{R}^n} \mathcal{L}_\beta(\mathbf{x}, \mathbf{y}_k, \lambda_k)$$

$$\mathbf{y}_{k+1} = \arg \min_{\mathbf{y} \in \mathbb{R}^p} \mathcal{L}_\beta(\mathbf{x}_{k+1}, \mathbf{y}, \lambda_k)$$

$$\lambda_{k+1} = \lambda_k + \beta(A\mathbf{x}_{k+1} + B\mathbf{y}_{k+1} - c)$$

- ADMM is similar to an incremental approach, but on two variables. It is also similar to dual decomposition, but on two variables

ADMM: iterations

- Then the ADMM does: start with an initial value for $\mathbf{x}_0, \mathbf{y}_0, \lambda_0$,
- Iterate:

$$\mathbf{x}_{k+1} = \arg \min_{\mathbf{x} \in \mathbb{R}^n} \mathcal{L}_\beta(\mathbf{x}, \mathbf{y}_k, \lambda_k)$$

$$\mathbf{y}_{k+1} = \arg \min_{\mathbf{y} \in \mathbb{R}^p} \mathcal{L}_\beta(\mathbf{x}_{k+1}, \mathbf{y}, \lambda_k)$$

$$\lambda_{k+1} = \lambda_k + \beta(A\mathbf{x}_{k+1} + B\mathbf{y}_{k+1} - c)$$

- ADMM is similar to an incremental approach, but on two variables. It is also similar to dual decomposition, but on two variables
- It looks more complicated than dual ascent, is it better in some sense?

ADMM: iterations

- Then the ADMM does: start with an initial value for $\mathbf{x}_0, \mathbf{y}_0, \lambda_0$,
- Iterate:

$$\mathbf{x}_{k+1} = \arg \min_{\mathbf{x} \in \mathbb{R}^n} \mathcal{L}_\beta(\mathbf{x}, \mathbf{y}_k, \lambda_k)$$

$$\mathbf{y}_{k+1} = \arg \min_{\mathbf{y} \in \mathbb{R}^p} \mathcal{L}_\beta(\mathbf{x}_{k+1}, \mathbf{y}, \lambda_k)$$

$$\lambda_{k+1} = \lambda_k + \beta(A\mathbf{x}_{k+1} + B\mathbf{y}_{k+1} - c)$$

- ADMM is similar to an incremental approach, but on two variables. It is also similar to dual decomposition, but on two variables
- It looks more complicated than dual ascent, is it better in some sense?
- Assume strong duality holds and primal and dual solutions exist, then **convergence is ensured for any $\beta > 0$ in a weak sense.**

ADMM: convergence in some special cases

- Assume f to be m -strongly convex and L -smooth and that a primal-dual solution exists (g can be a generic convex function). This is a special case and ADMM generally converges with minimal assumptions, but in a much weaker sense, so we keep here the stronger assumptions.

ADMM: convergence in some special cases

- Assume f to be m -strongly convex and L -smooth and that a primal-dual solution exists (g can be a generic convex function). This is a special case and ADMM generally converges with minimal assumptions, but in a much weaker sense, so we keep here the stronger assumptions.
- Assume that A is full row rank.

ADMM: convergence in some special cases

- Assume f to be m -strongly convex and L -smooth and that a primal-dual solution exists (g can be a generic convex function). This is a special case and ADMM generally converges with minimal assumptions, but in a much weaker sense, so we keep here the stronger assumptions.
- Assume that A is full row rank.
- Then ADMM converges linearly for any step size $\beta > 0$.

ADMM: convergence in some special cases

- Assume f to be m -strongly convex and L -smooth and that a primal-dual solution exists (g can be a generic convex function). This is a special case and ADMM generally converges with minimal assumptions, but in a much weaker sense, so we keep here the stronger assumptions.
- Assume that A is full row rank.
- Then ADMM converges linearly for any step size $\beta > 0$.

Theorem 10 (ADMM convergence)

With the assumptions in place,

$$\|\lambda_k - \lambda^*\| \leq \varrho^k \|\lambda_0 - \lambda^*\| \quad \varrho = \max \left\{ \left| \frac{1 - \beta \frac{\sigma_{\max}^2(A)}{m}}{1 + \beta \frac{\sigma_{\max}^2(A)}{m}} \right|, \left| \frac{1 - \beta \frac{\sigma_{\min}^2(A)}{L}}{1 + \beta \frac{\sigma_{\min}^2(A)}{L}} \right| \right\}$$

and,

$$\|\mathbf{x}_k - \mathbf{x}^*\| \leq \frac{\sigma_{\max}(A)}{m} \|\lambda_k - \lambda^*\|.$$

Proof?

- ADMM is extremely popular, for its loose assumptions and the fact that it works in many contexts, even when we have no idea why (non-convex, combinatorial, etc..)

Proof?

- ADMM is extremely popular, for its loose assumptions and the fact that it works in many contexts, even when we have no idea why (non-convex, combinatorial, etc..)
- Hence, we have many ways to prove its convergence. Here, I want to give a glimpse of an operator-splitting technique (check OPT202).

Proof?

- ADMM is extremely popular, for its loose assumptions and the fact that it works in many contexts, even when we have no idea why (non-convex, combinatorial, etc..)
- Hence, we have many ways to prove its convergence. Here, I want to give a glimpse of an operator-splitting technique (check OPT202).
- The proof goes as follows: ADMM is an application of a special algorithm (the Douglas-Rachford splitting) applied to the dual of our initial problem. The Douglas-Rachford splitting converges in a certain way given functional properties. We derive the dual of those functional properties and (as in the dual decomposition case) the convergence of the dual algorithm (ADMM).

Proof?

- ADMM is extremely popular, for its loose assumptions and the fact that it works in many contexts, even when we have no idea why (non-convex, combinatorial, etc..)
- Hence, we have many ways to prove its convergence. Here, I want to give a glimpse of an operator-splitting technique (check OPT202).
- The proof goes as follows: ADMM is an application of a special algorithm (the Douglas-Rachford splitting) applied to the dual of our initial problem. The Douglas-Rachford splitting converges in a certain way given functional properties. We derive the dual of those functional properties and (as in the dual decomposition case) the convergence of the dual algorithm (ADMM).
- This is why the result looks very similar to the result of the dual decomposition. ADMM is a dual algorithm.

Proof: step I, Douglas-Rachford splitting

- Consider the problem,

$$\min_{x \in \mathbb{R}^n} f(x) + g(x),$$

with f and g convex closed and proper (CCP). Consider now the following method to find a solution x^* .

Proof: step I, Douglas-Rachford splitting

- Consider the problem,

$$\min_{x \in \mathbb{R}^n} f(x) + g(x),$$

with f and g convex closed and proper (CCP). Consider now the following method to find a solution x^* .

- Start at a certain z_0 and iterate for all $k \in \mathbb{N}$:

$$x_k = \text{prox}_{\beta f}(z_k) \tag{5a}$$

$$z_{k+1} = z_k + \text{prox}_{\beta g}(2x_k - z_k) - x_k, \tag{5b}$$

where $\text{prox}_{\beta \phi}$ is the usual prox operator:

$$\text{prox}_{\beta \phi}(u) = \arg \min_v \left\{ \phi(v) + \frac{1}{2\beta} \|v - u\|^2 \right\}$$

The method is called the Douglas-Rachford splitting and it converges in some defined sense.

Proof: step I, Douglas-Rachford splitting

- Consider the problem,

$$\min_{x \in \mathbb{R}^n} f(x) + g(x),$$

with f and g convex closed and proper (CCP). Consider now the following method to find a solution x^* .

- Start at a certain z_0 and iterate for all $k \in \mathbb{N}$:

$$x_k = \text{prox}_{\beta f}(z_k) \tag{5a}$$

$$z_{k+1} = z_k + \text{prox}_{\beta g}(2x_k - z_k) - x_k, \tag{5b}$$

where $\text{prox}_{\beta \phi}$ is the usual prox operator:

$$\text{prox}_{\beta \phi}(u) = \arg \min_v \{ \phi(v) + \frac{1}{2\beta} \|v - u\|^2 \}$$

The method is called the Douglas-Rachford splitting and it converges in some defined sense.

- In particular if f is m -strongly convex and L -smooth, then, for all $\beta > 0$

$$\|z_{k+1} - z^*\| \leq \varrho^k \|z_k - z^*\|, \quad \varrho = \max \left\{ \left| \frac{1 - \beta L}{1 + \beta L} \right|, \left| \frac{1 - \beta m}{1 + \beta m} \right| \right\}$$

Proof: step II, The dual problem and its properties

- Start from our problem:

$$\begin{array}{ll} \min_{\mathbf{x} \in \mathbb{R}^n, \mathbf{y} \in \mathbb{R}^p} & f(\mathbf{x}) + g(\mathbf{y}) \\ \text{subject to} & A\mathbf{x} + B\mathbf{y} = \mathbf{c}, \end{array}$$

form the Lagrangian and take the dual

Proof: step II, The dual problem and its properties

- Start from our problem:

$$\begin{array}{ll}\min_{\mathbf{x} \in \mathbb{R}^n, \mathbf{y} \in \mathbb{R}^p} & f(\mathbf{x}) + g(\mathbf{y}) \\ \text{subject to} & A\mathbf{x} + B\mathbf{y} = \mathbf{c},\end{array}$$

form the Lagrangian and take the dual

- Dual problem is,

$$\max_{\lambda} \inf_{\mathbf{x}, \mathbf{y}} \{f(\mathbf{x}) + g(\mathbf{y}) + \lambda^{\top}(A\mathbf{x} + B\mathbf{y} - \mathbf{c})\}$$

and so,

$$\min_{\lambda} f^{*}(-A^{\top}\lambda) + g^{*}(-B^{\top}\lambda) + \mathbf{c}^{\top}\lambda$$

Proof: step II, The dual problem and its properties

- Start from our problem:

$$\begin{array}{ll}\min_{\mathbf{x} \in \mathbb{R}^n, \mathbf{y} \in \mathbb{R}^p} & f(\mathbf{x}) + g(\mathbf{y}) \\ \text{subject to} & A\mathbf{x} + B\mathbf{y} = \mathbf{c},\end{array}$$

form the Lagrangian and take the dual

- Dual problem is,

$$\max_{\lambda} \inf_{\mathbf{x}, \mathbf{y}} \{f(\mathbf{x}) + g(\mathbf{y}) + \lambda^{\top}(A\mathbf{x} + B\mathbf{y} - \mathbf{c})\}$$

and so,

$$\min_{\lambda} f^*(-A^{\top}\lambda) + g^*(-B^{\top}\lambda) + \mathbf{c}^{\top}\lambda$$

- Consider “ f ” = $f^*(-A^{\top}\lambda) + \mathbf{c}^{\top}\lambda$ and “ g ” = $g^*(-B^{\top}\lambda)$

Proof: step II, The dual problem and its properties

- Start from our problem:

$$\begin{aligned} \min_{\mathbf{x} \in \mathbb{R}^n, \mathbf{y} \in \mathbb{R}^p} \quad & f(\mathbf{x}) + g(\mathbf{y}) \\ \text{subject to} \quad & A\mathbf{x} + B\mathbf{y} = \mathbf{c}, \end{aligned}$$

form the Lagrangian and take the dual

- Dual problem is,

$$\max_{\lambda} \inf_{\mathbf{x}, \mathbf{y}} \{f(\mathbf{x}) + g(\mathbf{y}) + \lambda^\top (A\mathbf{x} + B\mathbf{y} - \mathbf{c})\}$$

and so,

$$\min_{\lambda} f^*(-A^\top \lambda) + g^*(-B^\top \lambda) + \mathbf{c}^\top \lambda$$

- Consider “ f ” = $f^*(-A^\top \lambda) + \mathbf{c}^\top \lambda$ and “ g ” = $g^*(-B^\top \lambda)$
- As before we know that “ f ” is σ_{\min}^2/L -strongly convex and σ_{\max}^2/m -smooth

Proof: step III, applying DR to the dual

- Apply DRS " f " = $f^*(-A^\top \lambda) + c^\top \lambda$ and " g " = $g^*(-B^\top \lambda)$ and use its convergence properties. Here we find the rate ρ of the Theorem.

Proof: step III, applying DR to the dual

- Apply DRS " f " = $f^*(-A^\top \lambda) + c^\top \lambda$ and " g " = $g^*(-B^\top \lambda)$ and use its convergence properties. Here we find the rate ρ of the Theorem.
- **Do some computational gymnastic** and arrive at the ADMM iterations \square

ADMM: distributed optimization

- Let's go back to:

$$\begin{array}{ll} \min_{\mathbf{x} \in \mathbb{R}^n, \mathbf{y} \in \mathbb{R}^p} & f(\mathbf{x}) + g(\mathbf{y}) \\ \text{subject to} & A\mathbf{x} + B\mathbf{y} = \mathbf{c}, \end{array}$$

and our problem,

$$\begin{array}{ll} \min_{\mathbf{x}^i \in \mathbb{R}^n, \mathbf{y}^{ij} \in \mathbb{R}^{|\mathcal{E}|}} & \sum_{i=1}^N f_i(\mathbf{x}^i) \\ \text{subject to} & \mathbf{x}^i = \mathbf{y}^{ij}, \quad \forall i \sim j. \end{array}$$

ADMM: distributed optimization

- Let's go back to:

$$\begin{array}{ll}\min_{\mathbf{x} \in \mathbb{R}^n, \mathbf{y} \in \mathbb{R}^p} & f(\mathbf{x}) + g(\mathbf{y}) \\ \text{subject to} & A\mathbf{x} + B\mathbf{y} = \mathbf{c},\end{array}$$

and our problem,

$$\begin{array}{ll}\min_{\mathbf{x}^i \in \mathbb{R}^n, \mathbf{y}^{ij} \in \mathbb{R}^{|\mathcal{E}|}} & \sum_{i=1}^N f_i(\mathbf{x}^i) \\ \text{subject to} & \mathbf{x}^i = \mathbf{y}^{ij}, \quad \forall i \sim j.\end{array}$$

- Think a bit at what is written. This is of course not the only possibility. Different splittings yield different properties. This will give a distributed algorithm similar to the one we have seen for dual decomposition.

ADMM: distributed optimization

- Let's go back to:

$$\begin{aligned} \min_{\mathbf{x} \in \mathbb{R}^n, \mathbf{y} \in \mathbb{R}^p} \quad & f(\mathbf{x}) + g(\mathbf{y}) \\ \text{subject to} \quad & A\mathbf{x} + B\mathbf{y} = c, \end{aligned}$$

and our problem,

$$\begin{aligned} \min_{\mathbf{x}^i \in \mathbb{R}^n, \mathbf{y}^{ij} \in \mathbb{R}^{|\mathcal{E}|}} \quad & \sum_{i=1}^N f_i(\mathbf{x}^i) \\ \text{subject to} \quad & \mathbf{x}^i = \mathbf{y}^{ij}, \quad \forall i \sim j. \end{aligned}$$

- Think a bit at what is written. This is of course not the only possibility. Different splittings yield different properties. This will give a distributed algorithm similar to the one we have seen for dual decomposition.
- Later, we will see a different cloud-based splitting instead.

ADMM: distributed optimization

- Let's go back to:

$$\begin{aligned} \min_{\mathbf{x} \in \mathbb{R}^n, \mathbf{y} \in \mathbb{R}^p} \quad & f(\mathbf{x}) + g(\mathbf{y}) \\ \text{subject to} \quad & A\mathbf{x} + B\mathbf{y} = \mathbf{c}, \end{aligned}$$

and our problem,

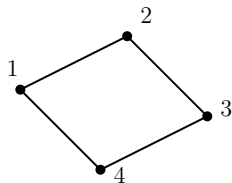
$$\begin{aligned} \min_{\mathbf{x}^i \in \mathbb{R}^n, \mathbf{y}^{ij} \in \mathbb{R}^{|\mathcal{E}|}} \quad & \sum_{i=1}^N f_i(\mathbf{x}^i) \\ \text{subject to} \quad & \mathbf{x}^i = \mathbf{y}^{ij}, \quad \forall i \sim j. \end{aligned}$$

- Think a bit at what is written. This is of course not the only possibility. Different splittings yield different properties. This will give a distributed algorithm similar to the one we have seen for dual decomposition.
- Later, we will see a different cloud-based splitting instead.
- Compactify $\mathbf{x}^i = \mathbf{y}^{ij}$ as, $A\mathbf{x} + B\mathbf{y} = \mathbf{0}$. Careful here that A is not full row rank, so linear convergence requires more work, but possible.

ADMM: let's look at A, B



$$A = \begin{bmatrix} I_n & 0 & 0 & 0 \\ 0 & I_n & 0 & 0 \\ 0 & I_n & 0 & 0 \\ 0 & 0 & I_n & 0 \\ 0 & 0 & I_n & 0 \\ 0 & 0 & 0 & I_n \\ 0 & 0 & 0 & I_n \\ I_n & 0 & 0 & 0 \end{bmatrix}$$



$$B = - \begin{bmatrix} I_n & 0 & 0 & 0 \\ I_n & 0 & 0 & 0 \\ 0 & I_n & 0 & 0 \\ 0 & I_n & 0 & 0 \\ 0 & 0 & I_n & 0 \\ 0 & 0 & I_n & 0 \\ 0 & 0 & 0 & I_n \\ 0 & 0 & 0 & I_n \end{bmatrix}$$

$\mathbf{x}^i \in \mathbf{R}^n, \mathbf{y} \in \mathbf{R}^{nE}$, BUT : $\lambda \in \mathbf{R}^{2nE}$

ADMM: distributed algorithm

- Then you apply ADMM and simplify,

ADMM: distributed algorithm

- Then you apply ADMM and simplify,
- For all nodes update:

$$\mathbf{x}_{k+1}^i = \arg \min_{\mathbf{x}^i} \left\{ f_i(\mathbf{x}^i) + \sum_{j \sim i} \left((\lambda_k^{ij})^\top (\mathbf{x}^i - \mathbf{y}_k^{ij}) + \frac{\beta}{2} \|\mathbf{x}^i - \mathbf{y}_k^{ij}\|^2 \right) \right\}$$

which is equivalent to,

$$\mathbf{x}_{k+1}^i = \arg \min_{\mathbf{x}^i} \left\{ f_i(\mathbf{x}^i) + \sum_{j \sim i} \frac{\beta}{2} \|\mathbf{x}^i - \mathbf{y}_k^{ij} + \frac{\lambda_k^{ij}}{\beta}\|^2 \right\}$$

Rem: $\mathbf{y}^{ij} = \mathbf{y}^{ji}$, but $\lambda^{ij} \neq \lambda^{ji}$!

ADMM: distributed algorithm

- Then you apply ADMM and simplify,
- For all nodes update:

$$\mathbf{x}_{k+1}^i = \arg \min_{\mathbf{x}^i} \left\{ f_i(\mathbf{x}^i) + \sum_{j \sim i} \left((\lambda_k^{ij})^\top (\mathbf{x}^i - \mathbf{y}_k^{ij}) + \frac{\beta}{2} \|\mathbf{x}^i - \mathbf{y}_k^{ij}\|^2 \right) \right\}$$

which is equivalent to,

$$\mathbf{x}_{k+1}^i = \arg \min_{\mathbf{x}^i} \left\{ f_i(\mathbf{x}^i) + \sum_{j \sim i} \frac{\beta}{2} \|\mathbf{x}^i - \mathbf{y}_k^{ij} + \frac{\lambda_k^{ij}}{\beta}\|^2 \right\}$$

Rem: $\mathbf{y}^{ij} = \mathbf{y}^{ji}$, but $\lambda^{ij} \neq \lambda^{ji}$!

- Communicate \mathbf{x}_{k+1}^i to your neighbors

ADMM: distributed algorithm

- Then, each node updates,

$$\mathbf{y}_{k+1}^{ij} = \arg \min_{\mathbf{y}^{ij}} \left\{ (\lambda_k)^\top (A\mathbf{x}_{k+1} + B\mathbf{y}) + \frac{\beta}{2} \|A\mathbf{x}_{k+1} + B\mathbf{y}\|^2 \right\}$$

which is equivalent to,

$$\mathbf{y}_{k+1}^{ij} = \arg \min_{\mathbf{y}^{ij}} \left\{ \frac{\beta}{2} \|\mathbf{x}_{k+1}^i - \mathbf{y}^{ij} + \frac{\lambda_k^{ij}}{\beta}\|^2 + \frac{\beta}{2} \|\mathbf{x}_{k+1}^j - \mathbf{y}^{ij} + \frac{\lambda_k^{ij}}{\beta}\|^2 \right\},$$

ADMM: distributed algorithm

- Then, each node updates,

$$\mathbf{y}_{k+1}^{ij} = \arg \min_{\mathbf{y}^{ij}} \left\{ (\lambda_k)^\top (A\mathbf{x}_{k+1} + B\mathbf{y}) + \frac{\beta}{2} \|A\mathbf{x}_{k+1} + B\mathbf{y}\|^2 \right\}$$

which is equivalent to,

$$\mathbf{y}_{k+1}^{ij} = \arg \min_{\mathbf{y}^{ij}} \left\{ \frac{\beta}{2} \|\mathbf{x}_{k+1}^i - \mathbf{y}^{ij} + \frac{\lambda_k^{ij}}{\beta}\|^2 + \frac{\beta}{2} \|\mathbf{x}_{k+1}^j - \mathbf{y}^{ij} + \frac{\lambda_k^{jj}}{\beta}\|^2 \right\},$$

- Solving for \mathbf{y}_{k+1}^{ij} in this unconstrained problem gives,

$$\mathbf{y}_{k+1}^{ij} = \frac{\mathbf{x}_{k+1}^i + \mathbf{x}_{k+1}^j}{2} + \frac{\lambda_k^{ij} + \lambda_k^{jj}}{2\beta}$$

ADMM: distributed algorithm cont'ed

- Then, each node updates,

$$\lambda_{k+1}^{ij} = \lambda_k^{ij} + \beta(\mathbf{x}_{k+1}^i - \mathbf{y}_{k+1}^{ij})$$

ADMM: distributed algorithm cont'ed

- Then, each node updates,

$$\lambda_{k+1}^{ij} = \lambda_k^{ij} + \beta(\mathbf{x}_{k+1}^i - \mathbf{y}_{k+1}^{ij})$$

- Further, note that,

$$\lambda_{k+1}^{ij} + \lambda_{k+1}^{ji} = \lambda_k^{ij} + \lambda_k^{ji} + \beta(\mathbf{x}_{k+1}^i + \mathbf{x}_{k+1}^j - 2\mathbf{y}_{k+1}^{ij}) = \mathbf{0}.$$

$$\text{So, } \mathbf{y}_{k+1}^{ij} = \frac{\mathbf{x}_{k+1}^i + \mathbf{x}_{k+1}^j}{2} + \frac{\lambda_k^{ij} + \lambda_k^{ji}}{2\beta} = \frac{\mathbf{x}_{k+1}^i + \mathbf{x}_{k+1}^j}{2}$$

ADMM: distributed algorithm condensed

- 1 Start with $\lambda_0^{ij} = 0$, and \mathbf{y}_0^{ij} then,
- 2 Each node updates,

$$\mathbf{x}_{k+1}^i = \arg \min_{\mathbf{x}^i} \left\{ f_i(\mathbf{x}^i) + \sum_{j \sim i} \frac{\beta}{2} \|\mathbf{x}^i - \mathbf{y}_k^{ij} + \frac{\lambda_k^{ij}}{\beta}\|^2 \right\}$$

- 3 Each node communicates \mathbf{x}_{k+1}^i to its neighbors
- 4 Each node updates,

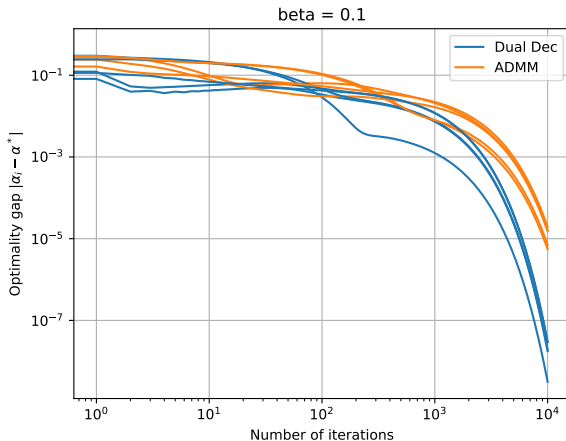
$$\mathbf{y}_{k+1}^{ij} = \frac{\mathbf{x}_{k+1}^i + \mathbf{x}_{k+1}^j}{2}$$

and,

$$\lambda_{k+1}^{ij} = \lambda_k^{ij} + \beta(\mathbf{x}_{k+1}^i - \mathbf{y}_{k+1}^{ij})$$

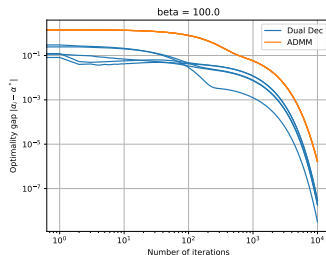
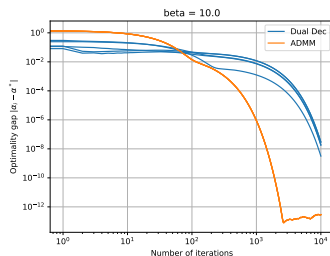
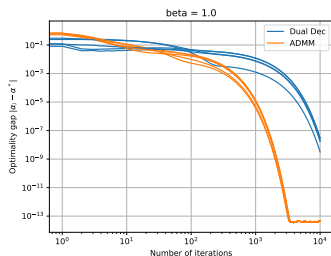
ADMM: numerical result

Same kernel setting, but now you can go to whichever β



ADMM: numerical result

Same kernel setting, but now you can go to whichever β



ADMM: Example in model fitting

- Consider the task of training a model via the convex problem,

$$\min_{\mathbf{x} \in \mathbf{R}^n} \ell(A\mathbf{x} - b) + r(\mathbf{x}),$$

where $\ell : \mathbf{R}^p \rightarrow \mathbf{R}$ is a convex loss function, $A \in \mathbf{R}^{p \times n}$ is the feature matrix, $b \in \mathbf{R}^p$ is the output vector, \mathbf{x} are the parameters of the model, and $r : \mathbf{R}^n \rightarrow \mathbf{R}$ is a convex regularization function.

ADMM: Example in model fitting

- Consider the task of training a model via the convex problem,

$$\min_{\mathbf{x} \in \mathbf{R}^n} \ell(A\mathbf{x} - b) + r(\mathbf{x}),$$

where $\ell : \mathbf{R}^p \rightarrow \mathbf{R}$ is a convex loss function, $A \in \mathbf{R}^{p \times n}$ is the feature matrix, $b \in \mathbf{R}^p$ is the output vector, \mathbf{x} are the parameters of the model, and $r : \mathbf{R}^n \rightarrow \mathbf{R}$ is a convex regularization function.

- Assume that ℓ is additive across training examples, as

$$\ell(\mathbf{x}) = \sum_{i=1}^m \ell_i(a_i^\top \mathbf{x} - b_i).$$

ADMM: Example in model fitting

- Consider the task of training a model via the convex problem,

$$\min_{\mathbf{x} \in \mathbf{R}^n} \ell(A\mathbf{x} - b) + r(\mathbf{x}),$$

where $\ell : \mathbf{R}^p \rightarrow \mathbf{R}$ is a convex loss function, $A \in \mathbf{R}^{p \times n}$ is the feature matrix, $b \in \mathbf{R}^p$ is the output vector, \mathbf{x} are the parameters of the model, and $r : \mathbf{R}^n \rightarrow \mathbf{R}$ is a convex regularization function.

- Assume that ℓ is additive across training examples, as

$$\ell(\mathbf{x}) = \sum_{i=1}^m \ell_i(a_i^\top \mathbf{x} - b_i).$$

- r is often separable too. For instance if r is the Tikhonov regularization (aka ridge penalty): $r(\mathbf{x}) = \nu \|\mathbf{x}\|_2^2$, or if r is the lasso penalty: $r(\mathbf{x}) = \nu \|\mathbf{x}\|_1$

ADMM: Example in model fitting

- Consider the task of training a model via the convex problem,

$$\min_{\mathbf{x} \in \mathbf{R}^n} \ell(A\mathbf{x} - b) + r(\mathbf{x}),$$

where $\ell : \mathbf{R}^p \rightarrow \mathbf{R}$ is a convex loss function, $A \in \mathbf{R}^{p \times n}$ is the feature matrix, $b \in \mathbf{R}^p$ is the output vector, \mathbf{x} are the parameters of the model, and $r : \mathbf{R}^n \rightarrow \mathbf{R}$ is a convex regularization function.

- Assume that ℓ is additive across training examples, as

$$\ell(\mathbf{x}) = \sum_{i=1}^m \ell_i(a_i^\top \mathbf{x} - b_i).$$

- r is often separable too. For instance if r is the Tikhonov regularization (aka ridge penalty): $r(\mathbf{x}) = \nu \|\mathbf{x}\|_2^2$, or if r is the lasso penalty: $r(\mathbf{x}) = \nu \|\mathbf{x}\|_1$
- Typically you have a modest number of features but a very large number of training examples (i.e., $m \gg n$).

ADMM: Example in model fitting II

- The goal is to solve the problem in a distributed way, with each processor handling a subset of the training data. This is useful either when there are so many training examples that it is inconvenient or impossible to process them on a single machine or when the data is naturally collected or stored in a distributed fashion. This includes, for example, online social network data, webserver access logs, wireless sensor networks, and many cloud computing applications more generally.

ADMM: Example in model fitting II

- The goal is to solve the problem in a distributed way, with each processor handling a subset of the training data. This is useful either when there are so many training examples that it is inconvenient or impossible to process them on a single machine or when the data is naturally collected or stored in a distributed fashion. This includes, for example, online social network data, webserver access logs, wireless sensor networks, and many cloud computing applications more generally.
- Suppose you associate a number of training examples to a number N of processors (in the extreme $N = m$). Then, you may solve

$$\begin{aligned} \min_{\mathbf{x}^j \in \mathbb{R}^n, \mathbf{y} \in \mathbb{R}^n} \quad & \sum_{j=1}^N \ell_j(A_j \mathbf{x}^j - b_j) + r(\mathbf{y}), \\ \text{subject to} \quad & \mathbf{x}^j = \mathbf{y}, \quad i = 1, \dots, N. \end{aligned}$$

ADMM: Example in model fitting II

- The goal is to solve the problem in a distributed way, with each processor handling a subset of the training data. This is useful either when there are so many training examples that it is inconvenient or impossible to process them on a single machine or when the data is naturally collected or stored in a distributed fashion. This includes, for example, online social network data, webserver access logs, wireless sensor networks, and many cloud computing applications more generally.
- Suppose you associate a number of training examples to a number N of processors (in the extreme $N = m$). Then, you may solve

$$\begin{aligned} \min_{\mathbf{x}^j \in \mathbb{R}^n, \mathbf{y} \in \mathbb{R}^n} \quad & \sum_{j=1}^N \ell_j(A_j \mathbf{x}^j - b_j) + r(\mathbf{y}), \\ \text{subject to} \quad & \mathbf{x}^j = \mathbf{y}, \quad i = 1, \dots, N. \end{aligned}$$

- This can be solved via ADMM

ADMM: Example in model fitting III

- Each processor solves

$$\begin{aligned}\mathbf{x}_{k+1}^j &= \arg \min_{\mathbf{x}^j} \{ \ell_j(A_j \mathbf{x}^j - b^j) + \lambda_k^j(\mathbf{x}^j - \mathbf{y}_k) + \frac{\beta}{2} \|\mathbf{x}^j - \mathbf{y}_k\|^2 \} \\ &= \arg \min_{\mathbf{x}^j} \{ \ell_j(A_j \mathbf{x}^j - b^j) + \frac{\beta}{2} \|\mathbf{x}^j - \mathbf{y}_k\|^2 + \frac{\lambda_k^j}{\beta} \|^2 \}\end{aligned}$$

ADMM: Example in model fitting III

- Each processor solves

$$\begin{aligned}\mathbf{x}_{k+1}^j &= \arg \min_{\mathbf{x}^j} \{ \ell_j(A_j \mathbf{x}^j - b^j) + \lambda_k^j(\mathbf{x}^j - \mathbf{y}_k) + \frac{\beta}{2} \|\mathbf{x}^j - \mathbf{y}_k\|^2 \} \\ &= \arg \min_{\mathbf{x}^j} \{ \ell_j(A_j \mathbf{x}^j - b^j) + \frac{\beta}{2} \|\mathbf{x}^j - \mathbf{y}_k\|^2 + \frac{\lambda_k^j}{\beta} \|\mathbf{x}^j - \mathbf{y}_k\|^2 \}\end{aligned}$$

- Communication to the cloud of \mathbf{x}_{k+1}^j

ADMM: Example in model fitting III

- Each processor solves

$$\begin{aligned}\mathbf{x}_{k+1}^j &= \arg \min_{\mathbf{x}^j} \{ \ell_j(A_j \mathbf{x}^j - b^j) + \lambda_k^j (\mathbf{x}^j - \mathbf{y}_k) + \frac{\beta}{2} \|\mathbf{x}^j - \mathbf{y}_k\|^2 \} \\ &= \arg \min_{\mathbf{x}^j} \{ \ell_j(A_j \mathbf{x}^j - b^j) + \frac{\beta}{2} \|\mathbf{x}^j - \mathbf{y}_k + \frac{\lambda_k^j}{\beta}\|^2 \}\end{aligned}$$

- Communication to the cloud of \mathbf{x}_{k+1}^j
- On the cloud we solve,

$$\begin{aligned}\mathbf{y}_{k+1} &= \arg \min_{\mathbf{y}} \{ r(\mathbf{y}) + \sum_{j=1}^N \lambda_k^j (\mathbf{x}_{k+1}^j - \mathbf{y}) + \frac{\beta}{2} \|\mathbf{x}_{k+1}^j - \mathbf{y}\|^2 \} \\ &= \arg \min_{\mathbf{y}} \{ r(\mathbf{y}) + \sum_{j=1}^N \frac{\beta}{2} \|\mathbf{x}_{k+1}^j - \mathbf{y} - \frac{\lambda_k^j}{\beta}\|^2 \}\end{aligned}$$

ADMM: Example in model fitting III

- Each processor solves

$$\begin{aligned}\mathbf{x}_{k+1}^j &= \arg \min_{\mathbf{x}^j} \{ \ell_j(A_j \mathbf{x}^j - b^j) + \lambda_k^j(\mathbf{x}^j - \mathbf{y}_k) + \frac{\beta}{2} \|\mathbf{x}^j - \mathbf{y}_k\|^2 \} \\ &= \arg \min_{\mathbf{x}^j} \{ \ell_j(A_j \mathbf{x}^j - b^j) + \frac{\beta}{2} \|\mathbf{x}^j - \mathbf{y}_k + \frac{\lambda_k^j}{\beta}\|^2 \}\end{aligned}$$

- Communication to the cloud of \mathbf{x}_{k+1}^j
- On the cloud we solve,

$$\begin{aligned}\mathbf{y}_{k+1} &= \arg \min_{\mathbf{y}} \{ r(\mathbf{y}) + \sum_{j=1}^N \lambda_k^j(\mathbf{x}_{k+1}^j - \mathbf{y}) + \frac{\beta}{2} \|\mathbf{x}_{k+1}^j - \mathbf{y}\|^2 \} \\ &= \arg \min_{\mathbf{y}} \{ r(\mathbf{y}) + \sum_{j=1}^N \frac{\beta}{2} \|\mathbf{x}_{k+1}^j - \mathbf{y} - \frac{\lambda_k^j}{\beta}\|^2 \}\end{aligned}$$

- Update $\lambda_{k+1}^j = \lambda_k^j + \beta(\mathbf{x}_{k+1}^j - \mathbf{y}_{k+1})$ and communicate back to processors $\mathbf{y}_{k+1}, \lambda_{k+1}^j$

ADMM: Example in model fitting IV

- The data never leaves the processors

ADMM: Example in model fitting IV

- The data never leaves the processors
- Different updates of \mathbf{y} depending on r and on \mathbf{x} depending on ℓ .

ADMM: Example in model fitting IV

- The data never leaves the processors
- Different updates of \mathbf{y} depending on r and on \mathbf{x} depending on ℓ .
- Consider for example: $\ell(\cdot) = \|\cdot\|_2^2$, $r(\mathbf{y}) = \nu\|\mathbf{y}\|_1$, then we obtain the algorithm,

ADMM: Example in model fitting IV

- The data never leaves the processors
- Different updates of \mathbf{y} depending on r and on \mathbf{x} depending on ℓ .
- Consider for example: $\ell(\cdot) = \|\cdot\|_2^2$, $r(\mathbf{y}) = \nu\|\mathbf{y}\|_1$, then we obtain the algorithm,
- Each processor solves

$$\begin{aligned}\mathbf{x}_{k+1}^j &= \arg \min_{\mathbf{x}^j} \left\{ \|A_j \mathbf{x}^j - b_j\|^2 + \frac{\beta}{2} \|\mathbf{x}^j - \mathbf{y}_k + \frac{\lambda_k^j}{\beta}\|^2 \right\} \\ &= (A_j^\top A_j + \beta I)^{-1} (A_j^\top b_j + \beta \mathbf{y}_k - \lambda_k^j).\end{aligned}$$

ADMM: Example in model fitting IV

- The data never leaves the processors
- Different updates of \mathbf{y} depending on r and on \mathbf{x} depending on ℓ .
- Consider for example: $\ell(\cdot) = \|\cdot\|_2^2$, $r(\mathbf{y}) = \nu\|\mathbf{y}\|_1$, then we obtain the algorithm,
- Each processor solves

$$\begin{aligned}\mathbf{x}_{k+1}^j &= \arg \min_{\mathbf{x}^j} \left\{ \|A_j \mathbf{x}^j - b_j\|^2 + \frac{\beta}{2} \|\mathbf{x}^j - \mathbf{y}_k + \frac{\lambda_k^j}{\beta}\|^2 \right\} \\ &= (A_j^\top A_j + \beta I)^{-1} (A_j^\top b_j + \beta \mathbf{y}_k - \lambda_k^j).\end{aligned}$$

- Communication to the cloud of \mathbf{x}_{k+1}^j

ADMM: Example in model fitting IV

- The data never leaves the processors
- Different updates of \mathbf{y} depending on r and on \mathbf{x} depending on ℓ .
- Consider for example: $\ell(\cdot) = \|\cdot\|_2^2$, $r(\mathbf{y}) = \nu\|\mathbf{y}\|_1$, then we obtain the algorithm,
- Each processor solves

$$\begin{aligned}\mathbf{x}_{k+1}^j &= \arg \min_{\mathbf{x}^j} \{ \|A_j \mathbf{x}^j - b_j\|^2 + \frac{\beta}{2} \|\mathbf{x}^j - \mathbf{y}_k + \frac{\lambda_k^j}{\beta}\|^2 \} \\ &= (A_j^\top A_j + \beta I)^{-1} (A_j^\top b_j + \beta \mathbf{y}_k - \lambda_k^j).\end{aligned}$$

- Communication to the cloud of \mathbf{x}_{k+1}^j
- On the cloud we solve,

$$\mathbf{y}_{k+1} = \arg \min_{\mathbf{y}} \{ \nu \|\mathbf{y}\|_1 + \sum_{j=1}^N \frac{\beta}{2} \|\mathbf{x}_{k+1}^j - \mathbf{y} + \frac{\lambda_k^j}{\beta}\|^2 \} = S_{\nu/\beta}(\bar{\mathbf{x}}_{k+1} + \frac{\bar{\lambda}_k}{\beta})$$

ADMM: Example in model fitting IV

- The data never leaves the processors
- Different updates of \mathbf{y} depending on r and on \mathbf{x} depending on ℓ .
- Consider for example: $\ell(\cdot) = \|\cdot\|_2^2$, $r(\mathbf{y}) = \nu\|\mathbf{y}\|_1$, then we obtain the algorithm,
- Each processor solves

$$\begin{aligned}\mathbf{x}_{k+1}^j &= \arg \min_{\mathbf{x}^j} \left\{ \|A_j \mathbf{x}^j - b_j\|^2 + \frac{\beta}{2} \|\mathbf{x}^j - \mathbf{y}_k + \frac{\lambda_k^j}{\beta}\|^2 \right\} \\ &= (A_j^\top A_j + \beta I)^{-1} (A_j^\top b_j + \beta \mathbf{y}_k - \lambda_k^j).\end{aligned}$$

- Communication to the cloud of \mathbf{x}_{k+1}^j
- On the cloud we solve,

$$\mathbf{y}_{k+1} = \arg \min_{\mathbf{y}} \left\{ \nu \|\mathbf{y}\|_1 + \sum_{j=1}^N \frac{\beta}{2} \|\mathbf{x}_{k+1}^j - \mathbf{y} + \frac{\lambda_k^j}{\beta}\|^2 \right\} = S_{\nu/\beta}(\bar{\mathbf{x}}_{k+1} + \frac{\bar{\lambda}_k}{\beta})$$

- Update $\lambda_{k+1}^j = \lambda_k^j + \beta(\mathbf{x}_{k+1}^j - \mathbf{y}_{k+1})$ and communicate back to processors

Communication issues

- In the description above, we have used bi-direction synchronous communication, this may not be the case in reality

Communication issues

- In the description above, we have used bi-direction synchronous communication, this may not be the case in reality
- Asynchronicity, latencies, package losses, all add to the difficulty in proving convergence of the algorithm

Communication issues

- In the description above, we have used bi-direction synchronous communication, this may not be the case in reality
- Asynchronicity, latencies, package losses, all add to the difficulty in proving convergence of the algorithm
- Research in this domain is very rich and active!

What have we learned?

- We have looked at distributed optimization in the dual domain, with two “classical” algorithms: dual decomposition and the ADMM

What have we learned?

- We have looked at distributed optimization in the dual domain, with two “classical” algorithms: dual decomposition and the ADMM
- We have studied their convergence, which seem better (in some sense) than primal methods

What have we learned?

- We have looked at distributed optimization in the dual domain, with two “classical” algorithms: dual decomposition and the ADMM
- We have studied their convergence, which seem better (in some sense) than primal methods
- ADMM in particular offers multiple ways to distributed computations across multiple devices and it has received a lot of attention in recent years

What have we learned?

- We have looked at distributed optimization in the dual domain, with two “classical” algorithms: dual decomposition and the ADMM
- We have studied their convergence, which seem better (in some sense) than primal methods
- ADMM in particular offers multiple ways to distributed computations across multiple devices and it has received a lot of attention in recent years
- ADMM works by splitting the problem into a part that can be solved locally, and a part that we can afford to solve sharing information

What have we learned?

- We have looked at distributed optimization in the dual domain, with two “classical” algorithms: dual decomposition and the ADMM
- We have studied their convergence, which seem better (in some sense) than primal methods
- ADMM in particular offers multiple ways to distributed computations across multiple devices and it has received a lot of attention in recent years
- ADMM works by splitting the problem into a part that can be solved locally, and a part that we can afford to solve sharing information
- ADMM can be applied to many settings (multi-core, cloud-computing, distributed computing, etc..)

Sample references

- ① *Stephen Boyd, Neal Parikh, Eric Chu, Borja Peleato, and Jonathan Eckstein, **Distributed Optimization and Statistical Learning via the Alternating Direction Method of Multipliers**, Foundations and Trends in Machine Learning, 2010*
- ② *Ernest Ryu, Stephen Boyd, **A Primer on Monotone Operator Methods**, Appl. Comput. Math., 2016*
- ③ Many variants out there.

Part IV

Intermezzo

Similar worlds

- Let's look at the problem

$$(EP) \quad \min_{\mathbf{x} \in \mathbb{R}^n} \mathbf{E}_{\theta \sim \Theta} [f(\mathbf{x}; \theta)].$$

with a convex f in \mathbf{x} and noise $\theta \sim \Theta$. This is a stochastic optimization problem often encountered in machine learning.

Similar worlds

- Let's look at the problem

$$(EP) \quad \min_{\mathbf{x} \in \mathbb{R}^n} \mathbf{E}_{\theta \sim \Theta} [f(\mathbf{x}; \theta)].$$

with a convex f in \mathbf{x} and noise $\theta \sim \Theta$. This is a stochastic optimization problem often encountered in machine learning.

- Here, as usual, \mathbf{E} is the expectation over a certain distribution Θ ,

$$\mathbf{E}_{\theta \sim \Theta} [f(\mathbf{x}; \theta)] = \int_{\theta \sim \Theta} f(\mathbf{x}; \theta) p(\theta) d\theta,$$

which is convex, if f is convex in \mathbf{x} , since convex combination of convex functions.

Similar worlds

- Let's look at the problem

$$(EP) \quad \min_{\mathbf{x} \in \mathbb{R}^n} \mathbf{E}_{\theta \sim \Theta} [f(\mathbf{x}; \theta)].$$

with a convex f in \mathbf{x} and noise $\theta \sim \Theta$. This is a stochastic optimization problem often encountered in machine learning.

- Here, as usual, \mathbf{E} is the expectation over a certain distribution Θ ,

$$\mathbf{E}_{\theta \sim \Theta} [f(\mathbf{x}; \theta)] = \int_{\theta \sim \Theta} f(\mathbf{x}; \theta) p(\theta) d\theta,$$

which is convex, if f is convex in \mathbf{x} , since convex combination of convex functions.

- Example: training a machine learning model with loss f , then the problem can be seen as risk minimization.

Similar worlds

- Let's look at the problem

$$(EP) \quad \min_{\mathbf{x} \in \mathbb{R}^n} \mathbf{E}_{\theta \sim \Theta} [f(\mathbf{x}; \theta)].$$

with a convex f in \mathbf{x} and noise $\theta \sim \Theta$. This is a stochastic optimization problem often encountered in machine learning.

- Here, as usual, \mathbf{E} is the expectation over a certain distribution Θ ,

$$\mathbf{E}_{\theta \sim \Theta} [f(\mathbf{x}; \theta)] = \int_{\theta \sim \Theta} f(\mathbf{x}; \theta) p(\theta) d\theta,$$

which is convex, if f is convex in \mathbf{x} , since convex combination of convex functions.

- Example: training a machine learning model with loss f , then the problem can be seen as risk minimization.
- Is (EP) very different from (P) ?

Similar worlds

- Let's look at the problem

$$(EP) \quad \min_{\mathbf{x} \in \mathbb{R}^n} \mathbf{E}_{\theta \sim \Theta} [f(\mathbf{x}; \theta)].$$

with a convex f in \mathbf{x} and noise $\theta \sim \Theta$. This is a stochastic optimization problem often encountered in machine learning.

- Here, as usual, \mathbf{E} is the expectation over a certain distribution Θ , which is convex, if f is convex in \mathbf{x} , since convex combination of convex functions.
- Example: training a machine learning model with loss f , then the problem can be seen as risk minimization.
- Is (EP) very different from (P) ?
- In fact, we can think of (EP) as,

$$(EP) \quad \min_{\mathbf{x} \in \mathbb{R}^n} \mathbf{E}_{\theta \sim \Theta} [f(\mathbf{x}; \theta)] = \lim_{N \rightarrow \infty} \frac{1}{N} \sum_{i=1}^N f_i(\mathbf{x}),$$

where each $f_i = f(\mathbf{x}; \theta_i)$.

Similar worlds

- Let's look at the problem

$$(EP) \quad \min_{\mathbf{x} \in \mathbb{R}^n} \mathbf{E}_{\theta \sim \Theta} [f(\mathbf{x}; \theta)].$$

with a convex f in \mathbf{x} and noise $\theta \sim \Theta$. This is a stochastic optimization problem often encountered in machine learning.

- Here, as usual, \mathbf{E} is the expectation over a certain distribution Θ , which is convex, if f is convex in \mathbf{x} , since convex combination of convex functions.
- Example: training a machine learning model with loss f , then the problem can be seen as risk minimization.
- Is (EP) very different from (P) ?
- In fact, we can think of (EP) as,

$$(EP) \quad \min_{\mathbf{x} \in \mathbb{R}^n} \mathbf{E}_{\theta \sim \Theta} [f(\mathbf{x}; \theta)] = \lim_{N \rightarrow \infty} \frac{1}{N} \sum_{i=1}^N f_i(\mathbf{x}),$$

where each $f_i = f(\mathbf{x}; \theta_i)$.

- Solving (EP) can't be very different from solving (P) ..

Stochastic gradient descent

- An important difference with distributed optimization is that in the latter f_i can be anything, while here $f_i(\mathbf{x}) = f(\mathbf{x}; \theta_i)$. In this context, by linearity of $\mathbf{E}[\cdot]$,

$$\nabla_{\mathbf{x}} \mathbf{E}[f(\mathbf{x}; \theta)] = \mathbf{E}[\nabla_{\mathbf{x}} f(\mathbf{x}; \theta)] = \frac{1}{N} \sum_{i=1}^N \nabla f_i(\mathbf{x})$$

so that each $\nabla f_i(\mathbf{x})$ is an **unbiased estimator** of $\nabla_{\mathbf{x}} \mathbf{E}[f(\mathbf{x}; \theta)]$.

Stochastic gradient descent

- An important difference with distributed optimization is that in the latter f_i can be anything, while here $f_i(\mathbf{x}) = f(\mathbf{x}; \theta_i)$. In this context, by linearity of $\mathbf{E}[\cdot]$,

$$\nabla_{\mathbf{x}} \mathbf{E}[f(\mathbf{x}; \theta)] = \mathbf{E}[\nabla_{\mathbf{x}} f(\mathbf{x}; \theta)] = \frac{1}{N} \sum_{i=1}^N \nabla f_i(\mathbf{x})$$

so that each $\nabla f_i(\mathbf{x})$ is an **unbiased estimator** of $\nabla_{\mathbf{x}} \mathbf{E}[f(\mathbf{x}; \theta)]$.

- So we could use $\nabla f_i(\mathbf{x}) \approx \nabla_{\mathbf{x}} \mathbf{E}[f(\mathbf{x}; \theta)]$

Stochastic gradient descent

- An important difference with distributed optimization is that in the latter f_i can be anything, while here $f_i(\mathbf{x}) = f(\mathbf{x}; \theta_i)$. In this context, by linearity of $\mathbf{E}[\cdot]$,

$$\nabla_{\mathbf{x}} \mathbf{E}[f(\mathbf{x}; \theta)] = \mathbf{E}[\nabla_{\mathbf{x}} f(\mathbf{x}; \theta)] = \frac{1}{N} \sum_{i=1}^N \nabla f_i(\mathbf{x})$$

so that each $\nabla f_i(\mathbf{x})$ is an **unbiased estimator** of $\nabla_{\mathbf{x}} \mathbf{E}[f(\mathbf{x}; \theta)]$.

- So we could use $\nabla f_i(\mathbf{x}) \approx \nabla_{\mathbf{x}} \mathbf{E}[f(\mathbf{x}; \theta)]$
- **Stochastic Gradient Descent** \approx Incremental gradient

Stochastic gradient descent

- An important difference with distributed optimization is that in the latter f_i can be anything, while here $f_i(\mathbf{x}) = f(\mathbf{x}; \theta_i)$. In this context, by linearity of $\mathbf{E}[\cdot]$,

$$\nabla_{\mathbf{x}} \mathbf{E}[f(\mathbf{x}; \theta)] = \mathbf{E}[\nabla_{\mathbf{x}} f(\mathbf{x}; \theta)] = \frac{1}{N} \sum_{i=1}^N \nabla f_i(\mathbf{x})$$

so that each $\nabla f_i(\mathbf{x})$ is an **unbiased estimator** of $\nabla_{\mathbf{x}} \mathbf{E}[f(\mathbf{x}; \theta)]$.

- So we could use $\nabla f_i(\mathbf{x}) \approx \nabla_{\mathbf{x}} \mathbf{E}[f(\mathbf{x}; \theta)]$
- **Stochastic Gradient Descent** \approx Incremental gradient
 - ▶ Iterate, pick at random i ,

$$\mathbf{x}_{k+1} = \mathbf{x}_k - \alpha_k \nabla f_i(\mathbf{x}_k)$$

Stochastic gradient descent

- An important difference with distributed optimization is that in the latter f_i can be anything, while here $f_i(\mathbf{x}) = f(\mathbf{x}; \theta_i)$. In this context, by linearity of $\mathbf{E}[\cdot]$,

$$\nabla_{\mathbf{x}} \mathbf{E}[f(\mathbf{x}; \theta)] = \mathbf{E}[\nabla_{\mathbf{x}} f(\mathbf{x}; \theta)] = \frac{1}{N} \sum_{i=1}^N \nabla f_i(\mathbf{x})$$

so that each $\nabla f_i(\mathbf{x})$ is an **unbiased estimator** of $\nabla_{\mathbf{x}} \mathbf{E}[f(\mathbf{x}; \theta)]$.

- So we could use $\nabla f_i(\mathbf{x}) \approx \nabla_{\mathbf{x}} \mathbf{E}[f(\mathbf{x}; \theta)]$
- **Stochastic Gradient Descent** \approx Incremental gradient
 - ▶ Iterate, pick at random i ,

$$\mathbf{x}_{k+1} = \mathbf{x}_k - \alpha_k \nabla f_i(\mathbf{x}_k)$$

- ▶ For ML people α_k is the learning rate (for us is the step size)

Stochastic gradient descent

- An important difference with distributed optimization is that in the latter f_i can be anything, while here $f_i(\mathbf{x}) = f(\mathbf{x}; \theta_i)$. In this context, by linearity of $\mathbf{E}[\cdot]$,

$$\nabla_{\mathbf{x}} \mathbf{E}[f(\mathbf{x}; \theta)] = \mathbf{E}[\nabla_{\mathbf{x}} f(\mathbf{x}; \theta)] = \frac{1}{N} \sum_{i=1}^N \nabla f_i(\mathbf{x})$$

so that each $\nabla f_i(\mathbf{x})$ is an **unbiased estimator** of $\nabla_{\mathbf{x}} \mathbf{E}[f(\mathbf{x}; \theta)]$.

- So we could use $\nabla f_i(\mathbf{x}) \approx \nabla_{\mathbf{x}} \mathbf{E}[f(\mathbf{x}; \theta)]$
- **Stochastic Gradient Descent** \approx Incremental gradient
 - ▶ Iterate, pick at random i ,

$$\mathbf{x}_{k+1} = \mathbf{x}_k - \alpha_k \nabla f_i(\mathbf{x}_k)$$

- ▶ For ML people α_k is the learning rate (for us is the step size)
- ▶ Convergence?

Stochastic gradient descent: convergence

- Recall Class 1!

Stochastic gradient descent: convergence

- Recall Class 1!
- Assume a bounded variance

$$\mathbf{E}[\|\nabla f_i(\mathbf{x}) - \frac{1}{N} \sum_j \nabla f_j(\mathbf{x}^*)\|^2] = \mathbf{E}[\|\nabla f_i(\mathbf{x})\|^2] \leq G^2$$

Stochastic gradient descent: convergence

- Recall Class 1!
- Assume a bounded variance
$$\mathbf{E}[\|\nabla f_i(\mathbf{x}) - \frac{1}{N} \sum_j \nabla f_j(\mathbf{x}^*)\|^2] = \mathbf{E}[\|\nabla f_i(\mathbf{x})\|^2] \leq G^2$$
- Assume m -strong convexity and differentiability (you don't need to assume Lipschitz, since you assume bounded variance).

Stochastic gradient descent: convergence

- Recall Class 1!
- Assume a bounded variance
$$\mathbf{E}[\|\nabla f_i(\mathbf{x}) - \frac{1}{N} \sum_j \nabla f_j(\mathbf{x}^*)\|^2] = \mathbf{E}[\|\nabla f_i(\mathbf{x})\|^2] \leq G^2$$
- Assume m -strong convexity and differentiability (you don't need to assume Lipschitz, since you assume bounded variance).
- Then

Stochastic gradient descent: convergence

- Recall Class 1!
- Assume a bounded variance
$$\mathbf{E}[\|\nabla f_i(\mathbf{x}) - \frac{1}{N} \sum_j \nabla f_j(\mathbf{x}^*)\|^2] = \mathbf{E}[\|\nabla f_i(\mathbf{x})\|^2] \leq G^2$$
- Assume m -strong convexity and differentiability (you don't need to assume Lipschitz, since you assume bounded variance).
- Then

Theorem 11 (SGD convergence, strong convexity)

SGD convergence for constant step size $\alpha \leq 1/(2m)$ as,

$$\mathbf{E}[\|\mathbf{x}_k - \mathbf{x}^*\|^2] \leq (1 - 2m\alpha)^k \mathbf{E}[\|\mathbf{x}_0 - \mathbf{x}^*\|^2] + \frac{\alpha}{2m} G^2$$

Stochastic gradient descent: convergence proof

- The proof of the theorem is standard:

$$\begin{aligned}\|\mathbf{x}_{k+1} - \mathbf{x}^*\|^2 &= \|\mathbf{x}_k - \alpha \nabla f_i(\mathbf{x}_k) - \mathbf{x}^*\|^2 \\ &= \|\mathbf{x}_k - \mathbf{x}^*\|^2 - 2\alpha(\nabla f_i(\mathbf{x}_k))^\top (\mathbf{x}_k - \mathbf{x}^*) + \alpha^2 \|\nabla f_i(\mathbf{x}_k)\|^2\end{aligned}$$

Taking expectation with respect to i , since $\mathbf{E}_i[\nabla f_i(\mathbf{x}_k)] = \nabla f(\mathbf{x}_k)$, and recalling strong convexity, $(\nabla f(\mathbf{x}_k) - \nabla f(\mathbf{x}^*))^\top (\mathbf{x}_k - \mathbf{x}^*) \geq m\|\mathbf{x}_k - \mathbf{x}^*\|^2$,

Stochastic gradient descent: convergence proof

- The proof of the theorem is standard:

$$\begin{aligned}\|\mathbf{x}_{k+1} - \mathbf{x}^*\|^2 &= \|\mathbf{x}_k - \alpha \nabla f_i(\mathbf{x}_k) - \mathbf{x}^*\|^2 \\ &= \|\mathbf{x}_k - \mathbf{x}^*\|^2 - 2\alpha (\nabla f_i(\mathbf{x}_k))^\top (\mathbf{x}_k - \mathbf{x}^*) + \alpha^2 \|\nabla f_i(\mathbf{x}_k)\|^2\end{aligned}$$

Taking expectation with respect to i , since $\mathbf{E}_i[\nabla f_i(\mathbf{x}_k)] = \nabla f(\mathbf{x}_k)$, and recalling strong convexity, $(\nabla f(\mathbf{x}_k) - \nabla f(\mathbf{x}^*))^\top (\mathbf{x}_k - \mathbf{x}^*) \geq m \|\mathbf{x}_k - \mathbf{x}^*\|^2$,

- we have

$$\mathbf{E}_i[\|\mathbf{x}_{k+1} - \mathbf{x}^*\|^2] \leq (1 - 2m\alpha) \|\mathbf{x}_k - \mathbf{x}^*\|^2 + \alpha^2 G^2.$$

Stochastic gradient descent: convergence proof

- The proof of the theorem is standard:

$$\begin{aligned}\|\mathbf{x}_{k+1} - \mathbf{x}^*\|^2 &= \|\mathbf{x}_k - \alpha \nabla f_i(\mathbf{x}_k) - \mathbf{x}^*\|^2 \\ &= \|\mathbf{x}_k - \mathbf{x}^*\|^2 - 2\alpha (\nabla f_i(\mathbf{x}_k))^\top (\mathbf{x}_k - \mathbf{x}^*) + \alpha^2 \|\nabla f_i(\mathbf{x}_k)\|^2\end{aligned}$$

Taking expectation with respect to i , since $\mathbf{E}_i[\nabla f_i(\mathbf{x}_k)] = \nabla f(\mathbf{x}_k)$, and recalling strong convexity, $(\nabla f(\mathbf{x}_k) - \nabla f(\mathbf{x}^*))^\top (\mathbf{x}_k - \mathbf{x}^*) \geq m \|\mathbf{x}_k - \mathbf{x}^*\|^2$,

- we have

$$\mathbf{E}_i[\|\mathbf{x}_{k+1} - \mathbf{x}^*\|^2] \leq (1 - 2m\alpha) \|\mathbf{x}_k - \mathbf{x}^*\|^2 + \alpha^2 G^2.$$

- Taking total expectation,

$$\mathbf{E}[\|\mathbf{x}_{k+1} - \mathbf{x}^*\|^2] \leq (1 - 2m\alpha) \mathbf{E}[\|\mathbf{x}_k - \mathbf{x}^*\|^2] + \alpha^2 G^2.$$

Stochastic gradient descent: convergence proof

- The proof of the theorem is standard:

$$\begin{aligned}\|\mathbf{x}_{k+1} - \mathbf{x}^*\|^2 &= \|\mathbf{x}_k - \alpha \nabla f_i(\mathbf{x}_k) - \mathbf{x}^*\|^2 \\ &= \|\mathbf{x}_k - \mathbf{x}^*\|^2 - 2\alpha (\nabla f_i(\mathbf{x}_k))^\top (\mathbf{x}_k - \mathbf{x}^*) + \alpha^2 \|\nabla f_i(\mathbf{x}_k)\|^2\end{aligned}$$

Taking expectation with respect to i , since $\mathbf{E}_i[\nabla f_i(\mathbf{x}_k)] = \nabla f(\mathbf{x}_k)$, and recalling strong convexity, $(\nabla f(\mathbf{x}_k) - \nabla f(\mathbf{x}^*))^\top (\mathbf{x}_k - \mathbf{x}^*) \geq m\|\mathbf{x}_k - \mathbf{x}^*\|^2$,

- we have

$$\mathbf{E}_i[\|\mathbf{x}_{k+1} - \mathbf{x}^*\|^2] \leq (1 - 2m\alpha)\|\mathbf{x}_k - \mathbf{x}^*\|^2 + \alpha^2 G^2.$$

- Taking total expectation,

$$\mathbf{E}[\|\mathbf{x}_{k+1} - \mathbf{x}^*\|^2] \leq (1 - 2m\alpha)\mathbf{E}[\|\mathbf{x}_k - \mathbf{x}^*\|^2] + \alpha^2 G^2.$$

- By geometric series, the theorem is then proven. □

Stochastic gradient descent: considerations

- Again a trade-off between speed and error!

Stochastic gradient descent: considerations

- Again a trade-off between speed and error!
- You can take $\alpha_k = 1/(2mk)$, and doing the calculations,

$$\mathbf{E}[\|\mathbf{x}_{k+1} - \mathbf{x}^*\|^2] \leq (1 - 1/k) \mathbf{E}[\|\mathbf{x}_k - \mathbf{x}^*\|^2] + (1/2mk)^2 G^2 = \dots = O(\log(k)/k)$$

Stochastic gradient descent: considerations

- Again a trade-off between speed and error!
- You can take $\alpha_k = 1/(2mk)$, and doing the calculations,

$$\mathbf{E}[\|\mathbf{x}_{k+1} - \mathbf{x}^*\|^2] \leq (1 - 1/k) \mathbf{E}[\|\mathbf{x}_k - \mathbf{x}^*\|^2] + (1/2mk)^2 G^2 = \dots = O(\log(k)/k)$$

- Which again tells us that SGD is either fast but with error or slow with no error.

Stochastic gradient descent: considerations

- Again a trade-off between speed and error!
- You can take $\alpha_k = 1/(2mk)$, and doing the calculations,

$$\mathbf{E}[\|\mathbf{x}_{k+1} - \mathbf{x}^*\|^2] \leq (1 - 1/k) \mathbf{E}[\|\mathbf{x}_k - \mathbf{x}^*\|^2] + (1/2mk)^2 G^2 = \dots = O(\log(k)/k)$$

- Which again tells us that SGD is either fast but with error or slow with no error.
- Can we do better?

Stochastic gradient descent: considerations

- Again a trade-off between speed and error!
- You can take $\alpha_k = 1/(2mk)$, and doing the calculations,

$$\mathbf{E}[\|\mathbf{x}_{k+1} - \mathbf{x}^*\|^2] \leq (1 - 1/k) \mathbf{E}[\|\mathbf{x}_k - \mathbf{x}^*\|^2] + (1/2mk)^2 G^2 = \dots = O(\log(k)/k)$$

- Which again tells us that SGD is either fast but with error or slow with no error.
- Can we do better?
- Yes, remember gradient tracking?

Variance reduction

- Instead of using directly $\nabla f_i(\mathbf{x}) \approx \nabla f(\mathbf{x})$, use $\nabla f_i(\mathbf{x})$ to update an estimate $\mathbf{g} \approx \nabla f(\mathbf{x})$

Variance reduction

- Instead of using directly $\nabla f_i(\mathbf{x}) \approx \nabla f(\mathbf{x})$, use $\nabla f_i(\mathbf{x})$ to update an estimate $\mathbf{g} \approx \nabla f(\mathbf{x})$
- So we do $\mathbf{x}_{k+1} = \mathbf{x}_k - \alpha \mathbf{v}_k$

Variance reduction

- Instead of using directly $\nabla f_i(\mathbf{x}) \approx \nabla f(\mathbf{x})$, use $\nabla f_i(\mathbf{x})$ to update an estimate $\mathbf{g} \approx \nabla f(\mathbf{x})$
- So we do $\mathbf{x}_{k+1} = \mathbf{x}_k - \alpha \mathbf{v}_k$
- With $\mathbf{v}_k = \mathbf{g}_{k-1} + \frac{1}{n}(\nabla f_i(\mathbf{x}_k) - \nabla f_i(\mathbf{x}_{k-1}))$ for a randomly picked i , and $\mathbf{g}_k = \mathbf{v}_k$

Variance reduction

- Instead of using directly $\nabla f_i(\mathbf{x}) \approx \nabla f(\mathbf{x})$, use $\nabla f_i(\mathbf{x})$ to update an estimate $\mathbf{g} \approx \nabla f(\mathbf{x})$
- So we do $\mathbf{x}_{k+1} = \mathbf{x}_k - \alpha \mathbf{v}_k$
- With $\mathbf{v}_k = \mathbf{g}_{k-1} + \frac{1}{n}(\nabla f_i(\mathbf{x}_k) - \nabla f_i(\mathbf{x}_{k-1}))$ for a randomly picked i , and $\mathbf{g}_k = \mathbf{v}_k$
- This is an incremental version of the gradient tracking! (Stochastic Average Gradient, SAG)

Variance reduction

- Instead of using directly $\nabla f_i(\mathbf{x}) \approx \nabla f(\mathbf{x})$, use $\nabla f_i(\mathbf{x})$ to update an estimate $\mathbf{g} \approx \nabla f(\mathbf{x})$
- So we do $\mathbf{x}_{k+1} = \mathbf{x}_k - \alpha \mathbf{v}_k$
- With $\mathbf{v}_k = \mathbf{g}_{k-1} + \frac{1}{n}(\nabla f_i(\mathbf{x}_k) - \nabla f_i(\mathbf{x}_{k-1}))$ for a randomly picked i , and $\mathbf{g}_k = \mathbf{v}_k$
- This is an incremental version of the gradient tracking! (Stochastic Average Gradient, SAG)
- A even better version: SAGA, which has the same \mathbf{v}_k but a different \mathbf{g}_k :

$$\mathbf{g}_k = \mathbf{g}_{k-1} + (\nabla f_i(\mathbf{x}_k) - \nabla f_i(\mathbf{x}_{k-1}))$$

Rather technical (higher variance but unbiased)

Variance reduction

- Instead of using directly $\nabla f_i(\mathbf{x}) \approx \nabla f(\mathbf{x})$, use $\nabla f_i(\mathbf{x})$ to update an estimate $\mathbf{g} \approx \nabla f(\mathbf{x})$
- So we do $\mathbf{x}_{k+1} = \mathbf{x}_k - \alpha \mathbf{v}_k$
- With $\mathbf{v}_k = \mathbf{g}_{k-1} + \frac{1}{n}(\nabla f_i(\mathbf{x}_k) - \nabla f_i(\mathbf{x}_{k-1}))$ for a randomly picked i , and $\mathbf{g}_k = \mathbf{v}_k$
- This is an incremental version of the gradient tracking! (Stochastic Average Gradient, SAG)
- A even better version: SAGA, which has the same \mathbf{v}_k but a different \mathbf{g}_k :

$$\mathbf{g}_k = \mathbf{g}_{k-1} + (\nabla f_i(\mathbf{x}_k) - \nabla f_i(\mathbf{x}_{k-1}))$$

Rather technical (higher variance but unbiased)

- Is its convergence better?

Variance reduction

- Instead of using directly $\nabla f_i(\mathbf{x}) \approx \nabla f(\mathbf{x})$, use $\nabla f_i(\mathbf{x})$ to update an estimate $\mathbf{g} \approx \nabla f(\mathbf{x})$
- So we do $\mathbf{x}_{k+1} = \mathbf{x}_k - \alpha \mathbf{v}_k$
- With $\mathbf{v}_k = \mathbf{g}_{k-1} + \frac{1}{n}(\nabla f_i(\mathbf{x}_k) - \nabla f_i(\mathbf{x}_{k-1}))$ for a randomly picked i , and $\mathbf{g}_k = \mathbf{v}_k$
- This is an incremental version of the gradient tracking! (Stochastic Average Gradient, SAG)
- A even better version: SAGA, which has the same \mathbf{v}_k but a different \mathbf{g}_k :

$$\mathbf{g}_k = \mathbf{g}_{k-1} + (\nabla f_i(\mathbf{x}_k) - \nabla f_i(\mathbf{x}_{k-1}))$$

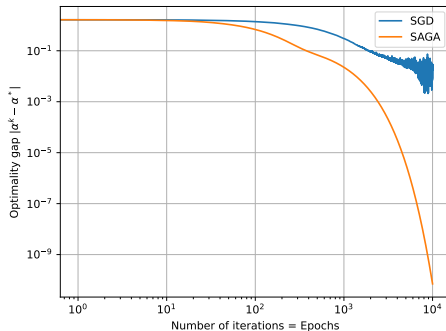
Rather technical (higher variance but unbiased)

- Is its convergence better?
- Yes, for SC functions

$$\mathbf{E}[\|\mathbf{x}_k - \mathbf{x}^*\|^2] \leq O(\rho^k).$$

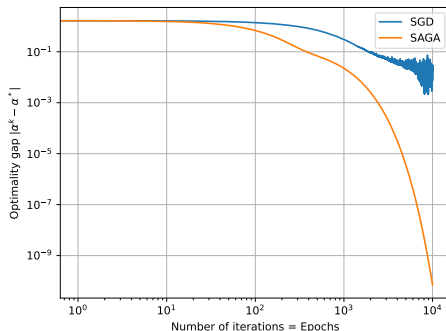
Back to numerical example

- It's incremental: each agent goes in random turn.



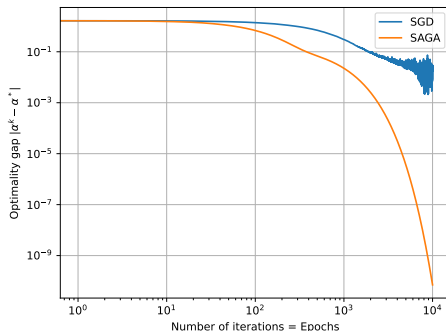
Back to numerical example

- It's incremental: each agent goes in random turn.
- Each agent has its own number of points: it's mini-batch.



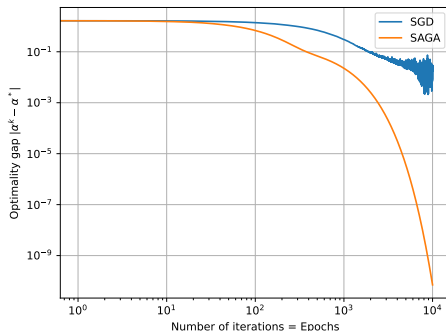
Back to numerical example

- It's incremental: each agent goes in random turn.
- Each agent has its own number of points: it's mini-batch.
- Here, step-size is the learning rate $s = 0.002$, the number of iterations are the epochs (in this case of a small number of agents, it is even better than GT (check the corresponding figure)!).



Back to numerical example

- It's incremental: each agent goes in random turn.
- Each agent has its own number of points: it's mini-batch.
- Here, step-size is the learning rate $s = 0.002$, the number of iterations are the epochs (in this case of a small number of agents, it is even better than GT (check the corresponding figure)!).
- In the SC setting, we are done, for non-convex many other methods (you will encounter Adam: adaptive momentum estimator)



Why bothering with SGD, SAG, SAGA, ... ?

- Think large-scale, think N as time

Why bothering with SGD, SAG, SAGA, ... ?

- Think large-scale, think N as time
- Each sample is obtained from different devices or a group of samples

Why bothering with SGD, SAG, SAGA, ... ?

- Think large-scale, think N as time
- Each sample is obtained from different devices or a group of samples
- If a device carries a group of samples, we often call the approach “mini-batch”

Why bothering with SGD, SAG, SAGA, ... ?

- Think large-scale, think N as time
- Each sample is obtained from different devices or a group of samples
- If a device carries a group of samples, we often call the approach “mini-batch”
- **An alternative angle as well.** In SGD, SAG, SAGA and ML in general the “distributed” part is more in time: you can’t wait to get all the training data and you update online. That’s why needs to be incremental.

Why bothering with SGD, SAG, SAGA, ... ?

- Think large-scale, think N as time
- Each sample is obtained from different devices or a group of samples
- If a device carries a group of samples, we often call the approach “mini-batch”
- **An alternative angle as well.** In SGD, SAG, SAGA and ML in general the “distributed” part is more in time: you can’t wait to get all the training data and you update online. That’s why needs to be incremental.
- But, wait, we saw that parallel is perhaps better than incremental (**for spatially distributed situations**).. is there a ML version of distributed optimization?

Class 4

Part V

Federated Learning

The setting revisited

- Federated learning is the machine learning variant of distributed optimization. In other words: it is distributed optimization brought to the extreme (huge datasets, almost no communication, very simple global computations)

The setting revisited

- Federated learning is the machine learning variant of distributed optimization. In other words: it is distributed optimization brought to the extreme (huge datasets, almost no communication, very simple global computations)
- Federated learning was proposed in 2016/2017 by Google researchers working with academia, and it is now a very vibrant research endeavor.

The setting revisited

- Federated learning is the machine learning variant of distributed optimization. In other words: it is distributed optimization brought to the extreme (huge datasets, almost no communication, very simple global computations)
- Federated learning was proposed in 2016/2017 by Google researchers working with academia, and it is now a very vibrant research endeavor.
- FL considers a number of nodes (workers) that train local models and communicate their model weights to a cloud service (or central coordinator). This is the baseline, then peer-to-peer variants start to exist.

The setting revisited

- Federated learning is the machine learning variant of distributed optimization. In other words: it is distributed optimization brought to the extreme (huge datasets, almost no communication, very simple global computations)
- Federated learning was proposed in 2016/2017 by Google researchers working with academia, and it is now a very vibrant research endeavor.
- FL considers a number of nodes (workers) that train local models and communicate their model weights to a cloud service (or central coordinator). This is the baseline, then peer-to-peer variants start to exist.
- The problem could be cast as risk minimization

$$(EP - FL) \quad \min_{\mathbf{x} \in X \subseteq \mathbb{R}^n} \mathbf{E}_{\theta \sim \Theta} [f(\mathbf{x}; \theta)]$$

where f are (here convex) functions and θ are noisy evaluations. As before, θ represents the data.

The setting revisited

- Federated learning is the machine learning variant of distributed optimization. In other words: it is distributed optimization brought to the extreme (huge datasets, almost no communication, very simple global computations)
- Federated learning was proposed in 2016/2017 by Google researchers working with academia, and it is now a very vibrant research endeavor.
- FL considers a number of nodes (workers) that train local models and communicate their model weights to a cloud service (or central coordinator). This is the baseline, then peer-to-peer variants start to exist.
- The problem could be cast as risk minimization

$$(EP - FL) \quad \min_{\mathbf{x} \in X \subseteq \mathbb{R}^n} \mathbf{E}_{\theta \sim \Theta} [f(\mathbf{x}; \theta)]$$

where f are (here convex) functions and θ are noisy evaluations. As before, θ represents the data.

- \mathbf{x} is often associated with model weights, so often it is called the “weights” or loosely “the model”.

The setting revisited

- The problem could be cast as risk minimization

$$(EP - FL) \quad \min_{\mathbf{x} \in X \subseteq \mathbb{R}^n} \mathbf{E}_{\theta \sim \Theta} [f(\mathbf{x}; \theta)]$$

where f are (here convex) functions and θ are noisy evaluations. As before, θ represents the data.

The setting revisited

- The problem could be cast as risk minimization

$$(EP - FL) \quad \min_{\mathbf{x} \in X \subseteq \mathbb{R}^n} \mathbf{E}_{\theta \sim \Theta} [f(\mathbf{x}; \theta)]$$

where f are (here convex) functions and θ are noisy evaluations. As before, θ represents the data.

- Think of it as,

$$(EP - FL) \quad \min_{\mathbf{x} \in X \subseteq \mathbb{R}^n} \mathbf{E}_{\theta \sim \Theta} [\|a^\top \mathbf{x} - (b + \theta)\|^2 + r(\mathbf{x})],$$

where a is the feature vector, b are the true labels and θ is some noise and r is a regularization. Of course, this is just an embodiment.

The setting revisited

- The problem could be cast as risk minimization

$$(EP - FL) \quad \min_{\mathbf{x} \in X \subseteq \mathbb{R}^n} \mathbf{E}_{\theta \sim \Theta} [f(\mathbf{x}; \theta)]$$

where f are (here convex) functions and θ are noisy evaluations. As before, θ represents the data.

- Think of it as,

$$(EP - FL) \quad \min_{\mathbf{x} \in X \subseteq \mathbb{R}^n} \mathbf{E}_{\theta \sim \Theta} [\|a^\top \mathbf{x} - (b + \theta)\|^2 + r(\mathbf{x})],$$

where a is the feature vector, b are the true labels and θ is some noise and r is a regularization. Of course, this is just an embodiment.

- From risk minimization you move to the empirical risk minimization for a finite number of samples N ,

$$\min_{\mathbf{x} \in X \subseteq \mathbb{R}^n} \frac{1}{N} \sum_{i=1}^N f_i(\mathbf{x}).$$

The setting revisited

- The problem could be cast as risk minimization

$$(EP - FL) \quad \min_{\mathbf{x} \in X \subseteq \mathbb{R}^n} \mathbf{E}_{\theta \sim \Theta} [f(\mathbf{x}; \theta)]$$

where f are (here convex) functions and θ are noisy evaluations. As before, θ represents the data.

-
- From risk minimization you move to the empirical risk minimization for a finite number of samples N ,

$$\min_{\mathbf{x} \in X \subseteq \mathbb{R}^n} \frac{1}{N} \sum_{i=1}^N f_i(\mathbf{x}).$$

- So far, we have worked with distributed optimization: we use the empirical risk minimization form as a starting point. In FL we use the risk minimization, since the training data may be generated differently across different devices! (Think images on your phone)

Federated Learning setting

The problem could be cast as risk minimization over P processors (e.g., phones)

$$\begin{aligned} \min_{\mathbf{x} \in X \subseteq \mathbb{R}^n} \quad & \mathbf{E}_{\theta \sim \Theta}[f(\mathbf{x}; \theta)] \\ & \equiv \frac{1}{P} \sum_{p=1}^P \mathbf{E}_{\theta \sim \Theta}[f(\mathbf{x}; \theta)] \end{aligned} \tag{6}$$

(8)

- The first step 6 is exact

Federated Learning setting

The problem could be cast as risk minimization over P processors (e.g., phones)

$$\min_{\mathbf{x} \in X \subseteq \mathbb{R}^n} \mathbf{E}_{\theta \sim \Theta} [f(\mathbf{x}; \theta)]$$
$$\equiv \frac{1}{P} \sum_{p=1}^P \mathbf{E}_{\theta \sim \Theta} [f(\mathbf{x}; \theta)] \quad (6)$$

$$\approx \frac{1}{P} \sum_{p=1}^P \mathbf{E}_{\theta \sim \Theta_p} [f(\mathbf{x}; \theta)] \quad (7)$$

(8)

- The first step 6 is exact
- The second step 7 is the first FL step (relaxing the IID assumption: independent and identically distributed!)

Federated Learning setting

The problem could be cast as risk minimization over P processors (e.g., phones)

$$\begin{aligned} \min_{\mathbf{x} \in X \subseteq \mathbb{R}^n} \quad & \mathbf{E}_{\theta \sim \Theta} [f(\mathbf{x}; \theta)] \\ \equiv & \frac{1}{P} \sum_{p=1}^P \mathbf{E}_{\theta \sim \Theta} [f(\mathbf{x}; \theta)] \end{aligned} \tag{6}$$

$$\approx \frac{1}{P} \sum_{p=1}^P \mathbf{E}_{\theta \sim \Theta_p} [f(\mathbf{x}; \theta)] \tag{7}$$

$$\approx \frac{1}{P} \sum_{p=1}^P \frac{1}{N_p} \sum_{j=1}^{N_p} f_j(\mathbf{x}) = \frac{1}{P} \sum_{p=1}^P f_p(\mathbf{x}) \tag{8}$$

- The first step 6 is exact
- The second step 7 is the first FL step (relaxing the IID assumption: independent and identically distributed!)
- The third step 8 is the second FL step (relaxing the training size!)

Federated Learning setting

The problem could be cast as risk minimization over P processors (e.g., phones)

$$\begin{aligned} \min_{\mathbf{x} \in X \subseteq \mathbb{R}^n} \quad & \mathbf{E}_{\theta \sim \Theta} [f(\mathbf{x}; \theta)] \\ \equiv & \frac{1}{P} \sum_{p=1}^P \mathbf{E}_{\theta \sim \Theta} [f(\mathbf{x}; \theta)] \end{aligned} \quad (6)$$

$$\approx \frac{1}{P} \sum_{p=1}^P \mathbf{E}_{\theta \sim \Theta_p} [f(\mathbf{x}; \theta)] \quad (7)$$

$$\approx \frac{1}{P} \sum_{p=1}^P \frac{1}{N_p} \sum_{j=1}^{N_p} f_j(\mathbf{x}) = \frac{1}{P} \sum_{p=1}^P f_p(\mathbf{x}) \quad (8)$$

The last problem can be solved with a distributed algorithm, but the important here is also how it has been generated and whether is reasonable or not w.r.t. the starting problem we wanted to solve!

Think taking two pictures with an old phone, and a thousands with the newest phone you can find. Are the models comparable?

FL idea

- FL puts more weight on the single devices (your phone can do a lot already), and every phone is different. So we are in a **federation**

FL idea

- FL puts more weight on the single devices (your phone can do a lot already), and every phone is different. So we are in a **federation**
- If then some device wants to get its model upgraded with extra info, it can share it (aka its weights) with the cloud and the cloud combine it with the other already shared models.

FL idea

- FL puts more weight on the single devices (your phone can do a lot already), and every phone is different. So we are in a **federation**
- If then some device wants to get its model upgraded with extra info, it can share it (aka its weights) with the cloud and the cloud combine it with the other already shared models.
- That's a weird way to do distributed optimization ?! Or..?

FL idea

- FL puts more weight on the single devices (your phone can do a lot already), and every phone is different. So we are in a **federation**
- If then some device wants to get its model upgraded with extra info, it can share it (aka its weights) with the cloud and the cloud combine it with the other already shared models.
- That's a weird way to do distributed optimization ?! Or..?
- Think Jacobi/Parallel algorithm of Class 1.

FL idea

- FL puts more weight on the single devices (your phone can do a lot already), and every phone is different. So we are in a **federation**
- If then some device wants to get its model upgraded with extra info, it can share it (aka its weights) with the cloud and the cloud combine it with the other already shared models.
- That's a weird way to do distributed optimization ?! Or..?
- Think Jacobi/Parallel algorithm of Class 1.
- The aim of FL is a bit different than DO: you just want to do a bit better than being all alone by sharing, you don't want to necessarily reach the same solution as a centralized scheme. In any case, that would be hard since you are not-iid, and possibly very data unbalanced

FedAvg: the baseline algorithm of FL

- First a bit of change of “jargon”: processors/devices \rightarrow clients; cloud/central processor \rightarrow server; batch size B : number of data points each clients use to update the local model; local epochs E : times local data is used to update the local models; step size \rightarrow learning rate

FedAvg: the baseline algorithm of FL

- First a bit of change of “jargon”: processors/devices \rightarrow clients; cloud/central processor \rightarrow server; batch size B : number of data points each clients use to update the local model; local epochs E : times local data is used to update the local models; step size \rightarrow learning rate
- **FedAvg**

FedAvg: the baseline algorithm of FL

- First a bit of change of “jargon”: processors/devices \rightarrow clients; cloud/central processor \rightarrow server; batch size B : number of data points each clients use to update the local model; local epochs E : times local data is used to update the local models; step size \rightarrow learning rate
- **FedAvg**
 - ▶ **Server update.** Initialize \mathbf{x}_0 . For each time $t = 1, \dots$ do:

FedAvg: the baseline algorithm of FL

- First a bit of change of “jargon”: processors/devices \rightarrow clients; cloud/central processor \rightarrow server; batch size B : number of data points each clients use to update the local model; local epochs E : times local data is used to update the local models; step size \rightarrow learning rate
- **FedAvg**
 - ▶ **Server update.** Initialize \mathbf{x}_0 . For each time $t = 1, \dots$ do:
 - ★ Select $C \leq P$ clients, and for each client

FedAvg: the baseline algorithm of FL

- First a bit of change of “jargon”: processors/devices \rightarrow clients; cloud/central processor \rightarrow server; batch size B : number of data points each clients use to update the local model; local epochs E : times local data is used to update the local models; step size \rightarrow learning rate
- **FedAvg**
 - ▶ **Server update.** Initialize \mathbf{x}_0 . For each time $t = 1, \dots$ do:
 - ★ Select $C \leq P$ clients, and for each client
 - ★ Send \mathbf{x}_t to client

FedAvg: the baseline algorithm of FL

- First a bit of change of “jargon”: processors/devices \rightarrow clients; cloud/central processor \rightarrow server; batch size B : number of data points each clients use to update the local model; local epochs E : times local data is used to update the local models; step size \rightarrow learning rate
- **FedAvg**
 - ▶ **Server update.** Initialize \mathbf{x}_0 . For each time $t = 1, \dots$ do:
 - ★ Select $C \leq P$ clients, and for each client
 - ★ Send \mathbf{x}_t to client
 - ★ Receive \mathbf{x}_{t+1}^i from client update

FedAvg: the baseline algorithm of FL

- First a bit of change of “jargon”: processors/devices \rightarrow clients; cloud/central processor \rightarrow server; batch size B : number of data points each clients use to update the local model; local epochs E : times local data is used to update the local models; step size \rightarrow learning rate
- **FedAvg**

► **Server update.** Initialize \mathbf{x}_0 . For each time $t = 1, \dots$ do:

- ★ Select $C \leq P$ clients, and for each client
- ★ Send \mathbf{x}_t to client
- ★ Receive \mathbf{x}_{t+1}^i from client update
- ★ Mixing

$$\mathbf{x}_{t+1} = \sum_{c=1}^C \frac{N_c}{\sum_c N_c} \mathbf{x}_{t+1}^c$$

FedAvg: the baseline algorithm of FL

- First a bit of change of “jargon”: processors/devices \rightarrow clients; cloud/central processor \rightarrow server; batch size B : number of data points each clients use to update the local model; local epochs E : times local data is used to update the local models; step size \rightarrow learning rate

- **FedAvg**

- ▶ **Server update.** Initialize \mathbf{x}_0 . For each time $t = 1, \dots$ do:

- ★ Select $C \leq P$ clients, and for each client
 - ★ Send \mathbf{x}_t to client
 - ★ Receive \mathbf{x}_{t+1}^i from client update
 - ★ Mixing

$$\mathbf{x}_{t+1} = \sum_{c=1}^C \frac{N_c}{\sum_c N_c} \mathbf{x}_{t+1}^c$$

- ▶ **Client update.** Start from $\mathbf{x}_0^i = \mathbf{x}_t$. Select batches \mathcal{B} of data, each containing B data points. For epoch $k = 1, \dots, E$

FedAvg: the baseline algorithm of FL

- First a bit of change of “jargon”: processors/devices \rightarrow clients; cloud/central processor \rightarrow server; batch size B : number of data points each clients use to update the local model; local epochs E : times local data is used to update the local models; step size \rightarrow learning rate

• FedAvg

- ▶ **Server update.** Initialize \mathbf{x}_0 . For each time $t = 1, \dots$ do:

- ★ Select $C \leq P$ clients, and for each client
- ★ Send \mathbf{x}_t to client
- ★ Receive \mathbf{x}_{t+1}^i from client update
- ★ Mixing

$$\mathbf{x}_{t+1} = \sum_{c=1}^C \frac{N_c}{\sum_c N_c} \mathbf{x}_{t+1}^c$$

- ▶ **Client update.** Start from $\mathbf{x}_0^i = \mathbf{x}_t$. Select batches \mathcal{B} of data, each containing B data points. For epoch $k = 1, \dots, E$

- ★ Local SGD,

$$\mathbf{x}_{k+1}^i = \mathbf{x}_k^i - \alpha_k \nabla f_{\mathcal{B}_j}(\mathbf{x}_k^i), \quad f_{\mathcal{B}_j} := \frac{1}{B} \sum_{j \in \mathcal{B}_j} f_j$$

FedAvg: example

- Batches ? Imagine you have n_i data points $\{1, 2, 3, \dots, n_i\}$. You can group them with repetition in groups (batches) of data. Say $B = 3$:

FedAvg: example

- Batches ? Imagine you have n_i data points $\{1, 2, 3, \dots, n_i\}$. You can group them with repetition in groups (batches) of data. Say $B = 3$:
- Example:

$$\mathcal{B}_1 = \{3, 4, n_i\}, \quad \mathcal{B}_2 = \{1, 2, 3\}, \quad \dots \quad \mathcal{B}_p = \{4, 6, 8\}.$$

FedAvg: example

- Batches ? Imagine you have n_i data points $\{1, 2, 3, \dots, n_i\}$. You can group them with repetition in groups (batches) of data. Say $B = 3$:
- Example:

$$\mathcal{B}_1 = \{3, 4, n_i\}, \quad \mathcal{B}_2 = \{1, 2, 3\}, \quad \dots \quad \mathcal{B}_p = \{4, 6, 8\}.$$

- You can decide to have uniform batching, or one one batch, etc. to decrease the computations or increase the convergence speed.

FedAvg: example

- Batches ? Imagine you have n_i data points $\{1, 2, 3, \dots, n_i\}$. You can group them with repetition in groups (batches) of data. Say $B = 3$:
- Example:

$$\mathcal{B}_1 = \{3, 4, n_i\}, \quad \mathcal{B}_2 = \{1, 2, 3\}, \quad \dots \quad \mathcal{B}_p = \{4, 6, 8\}.$$

- You can decide to have uniform batching, or one one batch, etc. to decrease the computations or increase the convergence speed.
- For the kernel problem, each client decides a batch of points they use at each update $\mathcal{B}_j, |\mathcal{B}_j| = B \leq n_i$, so,

$$\mathbf{x}_{k+1}^i = \mathbf{x}_k^i - \alpha_k \nabla f_{\mathcal{B}_j}(\mathbf{x}_k^i),$$

has

$$f_{\mathcal{B}_j}(\mathbf{x}) = \frac{1}{5} \frac{1}{2} \alpha^\top K_{mm} \alpha + \frac{1}{2\sigma^2} \frac{|\mathcal{Y}_a|}{B} \sum_{j \in \mathcal{B}_j(a=i)} \|y_j - K_{(j)m} \alpha\|_2^2 + \frac{\nu}{10} \|\alpha\|_2^2$$

FedAvg: example

- Batches ? Imagine you have n_i data points $\{1, 2, 3, \dots, n_i\}$. You can group them with repetition in groups (batches) of data. Say $B = 3$:
- Example:

$$\mathcal{B}_1 = \{3, 4, n_i\}, \quad \mathcal{B}_2 = \{1, 2, 3\}, \quad \dots \quad \mathcal{B}_p = \{4, 6, 8\}.$$

- You can decide to have uniform batching, or one one batch, etc. to decrease the computations or increase the convergence speed.
- For the kernel problem, each client decides a batch of points they use at each update $\mathcal{B}_j, |\mathcal{B}_j| = B \leq n_i$, so,

$$\mathbf{x}_{k+1}^i = \mathbf{x}_k^i - \alpha_k \nabla f_{\mathcal{B}_j}(\mathbf{x}_k^i),$$

has

$$f_{\mathcal{B}_j}(\mathbf{x}) = \frac{1}{5} \frac{1}{2} \alpha^\top K_{mm} \alpha + \frac{1}{2\sigma^2} \frac{|\mathcal{Y}_a|}{B} \sum_{j \in \mathcal{B}_j(a=i)} \|y_j - K_{(j)m} \alpha\|_2^2 + \frac{\nu}{10} \|\alpha\|_2^2$$

- Here we stress that $j \in \mathcal{B}_j(a=i)$ represents a sub-group of the points given to agent/client i .

FedAvg: the baseline algorithm of FL

- FedAvg is a Jacobi algorithm with a very simple update rule.

FedAvg: the baseline algorithm of FL

- FedAvg is a Jacobi algorithm with a very simple update rule.
- It has many tuning knobs and in practice clients can come in and drop out randomly, and there is no need to have complicated optimization problems in the server.

FedAvg: the baseline algorithm of FL

- FedAvg is a Jacobi algorithm with a very simple update rule.
- It has many tuning knobs and in practice clients can come in and drop out randomly, and there is no need to have complicated optimization problems in the server.
- Of course, we don't expect great convergence rates from it! But, that's hardly the point in FL.

FedAvg: the baseline algorithm of FL

- FedAvg is a Jacobi algorithm with a very simple update rule.
- It has many tuning knobs and in practice clients can come in and drop out randomly, and there is no need to have complicated optimization problems in the server.
- Of course, we don't expect great convergence rates from it! But, that's hardly the point in FL.
- FedAvg can be rewritten as,

Most of the times \mathcal{T} , do:

$$\mathbf{x}_{k+1}^i = \mathbf{x}_k^i - \alpha_k \nabla f_{\mathcal{B}_j}(\mathbf{x}_k^i), \quad k \in \mathcal{T}$$

In the other times, merge also as distributed-GD:

$$\mathbf{x}_{k+1}^i = \sum_{c=1}^C w_c [\mathbf{x}_k^c - \alpha_k \nabla f_{\mathcal{B}_j}(\mathbf{x}_k^c)], \quad k \notin \mathcal{T}$$

FedAvg: the baseline algorithm of FL

- FedAvg is a Jacobi algorithm with a very simple update rule.
- It has many tuning knobs and in practice clients can come in and drop out randomly, and there is no need to have complicated optimization problems in the server.
- Of course, we don't expect great convergence rates from it! But, that's hardly the point in FL.
- FedAvg can be rewritten as,

Most of the times \mathcal{T} , do:

$$\mathbf{x}_{k+1}^i = \mathbf{x}_k^i - \alpha_k \nabla f_{\mathcal{B}_j}(\mathbf{x}_k^i), \quad k \in \mathcal{T}$$

In the other times, merge also as distributed-GD:

$$\mathbf{x}_{k+1}^i = \sum_{c=1}^C w_c [\mathbf{x}_k^c - \alpha_k \nabla f_{\mathcal{B}_j}(\mathbf{x}_k^c)], \quad k \notin \mathcal{T}$$

- It's basically sometimes SGD and sometimes distributed-GD, so we can't expect great performance.

FedAvg: convergence

- There are many control knobs and therefore many variants out there. Functional class, increasing/decreasing communications, one-shot averaging, time-dependent batch sizes, time-dependent epochs, etc..

FedAvg: convergence

- There are many control knobs and therefore many variants out there. Functional class, increasing/decreasing communications, one-shot averaging, time-dependent batch sizes, time-dependent epochs, etc..
- **In general**, assuming:

FedAvg: convergence

- There are many control knobs and therefore many variants out there. Functional class, increasing/decreasing communications, one-shot averaging, time-dependent batch sizes, time-dependent epochs, etc..
- **In general**, assuming:
- IID data, so $F(\mathbf{x}) := \frac{1}{P} \sum_{p=1}^P \mathbf{E}_p[f(\mathbf{x}; \theta)] = \mathbf{E}[f(\mathbf{x}; \theta)]$

FedAvg: convergence

- There are many control knobs and therefore many variants out there. Functional class, increasing/decreasing communications, one-shot averaging, time-dependent batch sizes, time-dependent epochs, etc..
- **In general**, assuming:
 - IID data, so $F(\mathbf{x}) := \frac{1}{P} \sum_{p=1}^P \mathbf{E}_p[f(\mathbf{x}; \theta)] = \mathbf{E}[f(\mathbf{x}; \theta)]$
 - Convexity and L -Lipschitz continuous gradient for f

FedAvg: convergence

- There are many control knobs and therefore many variants out there. Functional class, increasing/decreasing communications, one-shot averaging, time-dependent batch sizes, time-dependent epochs, etc..
- **In general**, assuming:
 - IID data, so $F(\mathbf{x}) := \frac{1}{P} \sum_{p=1}^P \mathbf{E}_p[f(\mathbf{x}; \theta)] = \mathbf{E}[f(\mathbf{x}; \theta)]$
 - Convexity and L -Lipschitz continuous gradient for f
 - Bounded variance $\mathbf{E}[\|\nabla f(\mathbf{x}; \theta) - \nabla F(\mathbf{x})\|^2] \leq \sigma^2$,

FedAvg: convergence

- There are many control knobs and therefore many variants out there. Functional class, increasing/decreasing communications, one-shot averaging, time-dependent batch sizes, time-dependent epochs, etc..
- **In general**, assuming:
 - IID data, so $F(\mathbf{x}) := \frac{1}{P} \sum_{p=1}^P \mathbf{E}_p[f(\mathbf{x}; \theta)] = \mathbf{E}[f(\mathbf{x}; \theta)]$
 - Convexity and L -Lipschitz continuous gradient for f
 - Bounded variance $\mathbf{E}[\|\nabla f(\mathbf{x}; \theta) - \nabla F(\mathbf{x})\|^2] \leq \sigma^2$,
 - A diminishing α_k , plus some technical assumptions, we can get

$$\mathbf{E}[F(\hat{\mathbf{x}}_k) - F(\mathbf{x}^*)] \leq \underbrace{O(L/T)}_{\text{bias}} + \underbrace{O(\sigma/\sqrt{T})}_{\text{variance}} = O(1/\sqrt{T}),$$

for the ergodic mean $\hat{\mathbf{x}}_k$.

FedAvg: convergence

- There are many control knobs and therefore many variants out there. Functional class, increasing/decreasing communications, one-shot averaging, time-dependent batch sizes, time-dependent epochs, etc..
- **In general**, assuming:
 - IID data, so $F(\mathbf{x}) := \frac{1}{P} \sum_{p=1}^P \mathbf{E}_p[f(\mathbf{x}; \theta)] = \mathbf{E}[f(\mathbf{x}; \theta)]$
 - Convexity and L -Lipschitz continuous gradient for f
 - Bounded variance $\mathbf{E}[\|\nabla f(\mathbf{x}; \theta) - \nabla F(\mathbf{x})\|^2] \leq \sigma^2$,
 - A diminishing α_k , plus some technical assumptions, we can get

$$\mathbf{E}[F(\hat{\mathbf{x}}_k) - F(\mathbf{x}^*)] \leq \underbrace{O(L/T)}_{\text{bias}} + \underbrace{O(\sigma/\sqrt{T})}_{\text{variance}} = O(1/\sqrt{T}),$$

for the ergodic mean $\hat{\mathbf{x}}_k$.

- When strong convexity is also imposed, then one can get $O(1/T)$ with a decreasing learning rate $\alpha_k = O(1/k)$.

FedAvg: convergence

- There are many control knobs and therefore many variants out there. Functional class, increasing/decreasing communications, one-shot averaging, time-dependent batch sizes, time-dependent epochs, etc..
- **In general**, assuming:
- IID data, so $F(\mathbf{x}) := \frac{1}{P} \sum_{p=1}^P \mathbf{E}_p[f(\mathbf{x}; \theta)] = \mathbf{E}[f(\mathbf{x}; \theta)]$
- Convexity and L -Lipschitz continuous gradient for f
- Bounded variance $\mathbf{E}[\|\nabla f(\mathbf{x}; \theta) - \nabla F(\mathbf{x})\|^2] \leq \sigma^2$,
- A diminishing α_k , plus some technical assumptions, we can get

$$\mathbf{E}[F(\hat{\mathbf{x}}_k) - F(\mathbf{x}^*)] \leq \underbrace{O(L/T)}_{\text{bias}} + \underbrace{O(\sigma/\sqrt{T})}_{\text{variance}} = O(1/\sqrt{T}),$$

for the ergodic mean $\hat{\mathbf{x}}_k$.

- When strong convexity is also imposed, then one can get $O(1/T)$ with a decreasing learning rate $\alpha_k = O(1/k)$.
- There is a lot of research in getting better constants, as well as variance reduction via memory, and so forth. But at the end, I don't think convergence is something we care about very deeply here.

FedAvg: convergence

- The above can be extended to non-IDD data, and for strong convex functions we can still get $O(1/T)$ convergence rate with diminishing learning rate $\alpha_k = O(1/k)$.

FedAvg: convergence

- The above can be extended to non-IDD data, and for strong convex functions we can still get $O(1/T)$ convergence rate with diminishing learning rate $\alpha_k = O(1/k)$.
- We have also a communication rounds T_ϵ to get a certain error ϵ trade off, as

$$T_\epsilon \sim \frac{1}{\epsilon} \left[O\left(\frac{1}{E}\right) + O(E) \right]$$

which tells us about a trade-off between local computations and how many mixing steps. To reduce communication, we need to find the best number of local epochs E

FedAvg: convergence

- The above can be extended to non-IID data, and for strong convex functions we can still get $O(1/T)$ convergence rate with diminishing learning rate $\alpha_k = O(1/k)$.
- We have also a communication rounds T_ϵ to get a certain error ϵ trade off, as

$$T_\epsilon \sim \frac{1}{\epsilon} \left[O\left(\frac{1}{E}\right) + O(E) \right]$$

which tells us about a trade-off between local computations and how many mixing steps. To reduce communication, we need to find the best number of local epochs E

- Again, it all depends to what we are converging to and if we care about convergence. As said, we can have,

$$\mathbf{E}[F(\mathbf{x}_k) - F(\mathbf{x}^*)] \rightarrow 0,$$

where in the non-IID setting $F(\mathbf{x}) = \frac{1}{P} \sum_{p=1}^P \mathbf{E}_p[f(\mathbf{x}; \theta)] \approx \mathbf{E}_{\theta \sim \Theta}[f(\mathbf{x}; \theta)]$, so does it really matter?

FedAvg: convergence

- The above can be extended to non-IID data, and for strong convex functions we can still get $O(1/T)$ convergence rate with diminishing learning rate $\alpha_k = O(1/k)$.
- We have also a communication rounds T_ϵ to get a certain error ϵ trade off, as

$$T_\epsilon \sim \frac{1}{\epsilon} \left[O\left(\frac{1}{E}\right) + O(E) \right]$$

which tells us about a trade-off between local computations and how many mixing steps. To reduce communication, we need to find the best number of local epochs E

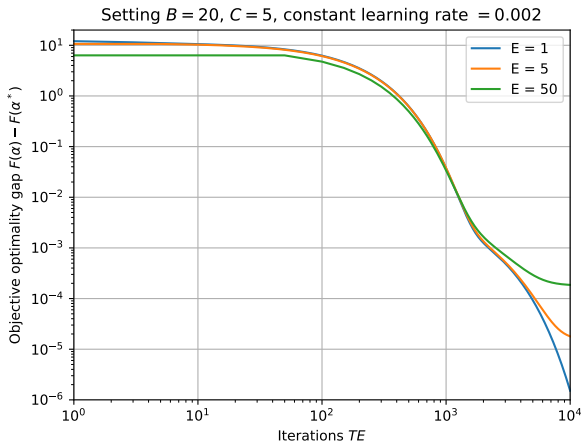
- Again, it all depends to what we are converging to and if we care about convergence. As said, we can have,

$$\mathbf{E}[F(\mathbf{x}_k) - F(\mathbf{x}^*)] \rightarrow 0,$$

where in the non-IID setting $F(\mathbf{x}) = \frac{1}{P} \sum_{p=1}^P \mathbf{E}_p[f(\mathbf{x}; \theta)] \approx \mathbf{E}_{\theta \sim \Theta}[f(\mathbf{x}; \theta)]$, so does it really matter?

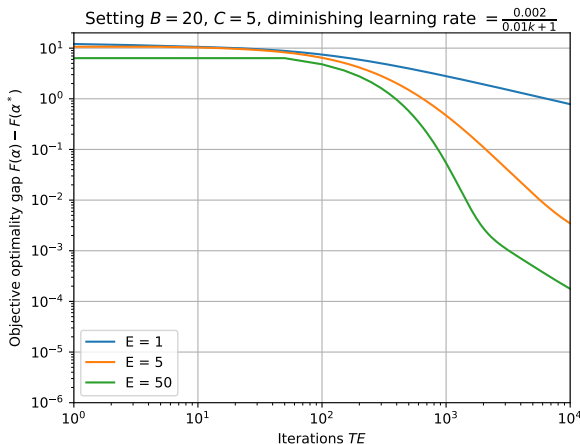
- But, anyway, let's see what it means on our kernel setting !

Distributed-GD setting in FedAvg



$E = 1$ is Distributed-GD with full averaging W , so error $\rightarrow 0$, the others plateau
Here B is the batch-size $B = 20$ is full batch.

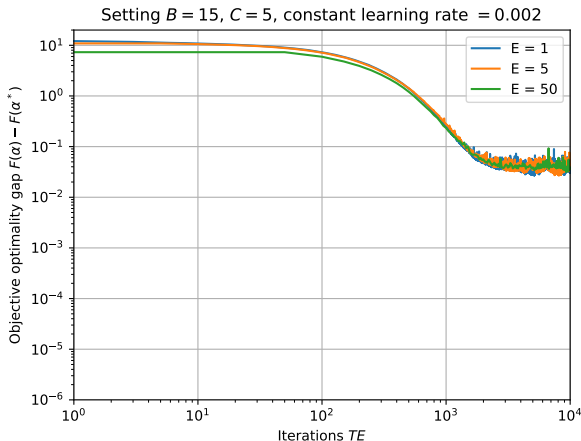
Distributed-GD setting in FedAvg / Diminishing learning rate



Error goes to zero as $O(1/T)$ but it could be slow

Here B is the batch-size $B = 20$ is full batch.

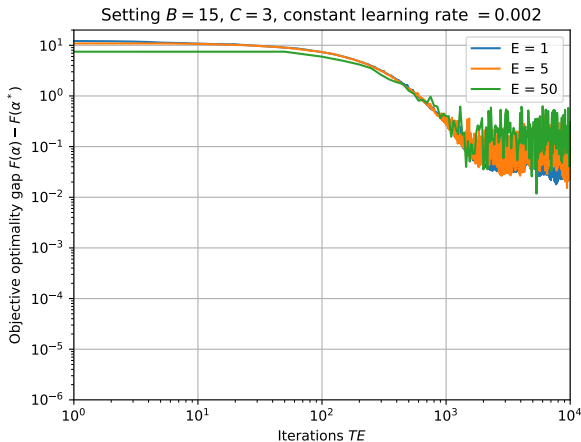
SGD setting in FedAvg



For $E = 1$ we get back SGD !

Here B is the batch-size $B = 15 < 20$ is mini-batch = SGD.

Full FedAvg setting



Less clients $C = 3 < 5$ per round

Here B is the batch-size $B = 15 < 20$ is mini-batch = (local) SGD.

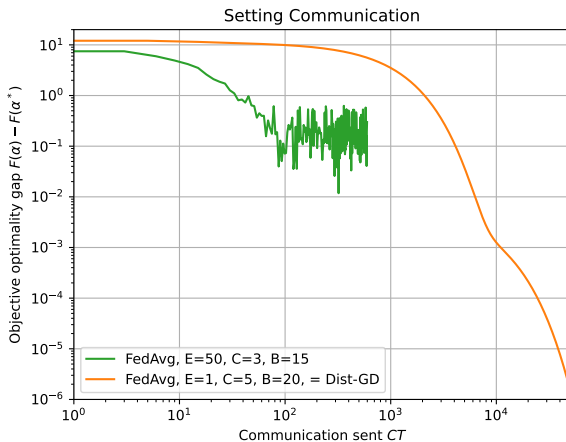
Communication rounds

Is FedAvg no good ? No, we don't care about convergence really, but communication rounds and get better than if we were alone !

Communication rounds

Is FedAvg no good ? No, we don't care about convergence really, but communication rounds and get better than if we were alone !

So, if we stop at, say $CT \sim 100$, we are way better ! ($CT \sim \text{time}$)



What have we learned?

- Stochastic problems in machine learning are quite similar in spirit to distributed optimization (finite sum structure)

What have we learned?

- Stochastic problems in machine learning are quite similar in spirit to distributed optimization (finite sum structure)
- ML uses different names for things but the tools are similar, however often the assumptions we are allowed to make are not the same.

What have we learned?

- Stochastic problems in machine learning are quite similar in spirit to distributed optimization (finite sum structure)
- ML uses different names for things but the tools are similar, however often the assumptions we are allowed to make are not the same.
- SGD is the workhorse of ML, if you work in ML you will use to train all your complicated models (e.g., neural networks are trained using variants of SGD)

What have we learned?

- Stochastic problems in machine learning are quite similar in spirit to distributed optimization (finite sum structure)
- ML uses different names for things but the tools are similar, however often the assumptions we are allowed to make are not the same.
- SGD is the workhorse of ML, if you work in ML you will use to train all your complicated models (e.g., neural networks are trained using variants of SGD)
- SAG and SAGA can boost performance of SGD as gradient tracking for DGD

What have we learned?

- Stochastic problems in machine learning are quite similar in spirit to distributed optimization (finite sum structure)
- ML uses different names for things but the tools are similar, however often the assumptions we are allowed to make are not the same.
- SGD is the workhorse of ML, if you work in ML you will use to train all your complicated models (e.g., neural networks are trained using variants of SGD)
- SAG and SAGA can boost performance of SGD as gradient tracking for DGD
- Federated learning is a way to improve local models without sharing (private) data, but only updating the weights with minimal requirements.

What have we learned?

- Stochastic problems in machine learning are quite similar in spirit to distributed optimization (finite sum structure)
- ML uses different names for things but the tools are similar, however often the assumptions we are allowed to make are not the same.
- SGD is the workhorse of ML, if you work in ML you will use to train all your complicated models (e.g., neural networks are trained using variants of SGD)
- SAG and SAGA can boost performance of SGD as gradient tracking for DGD
- Federated learning is a way to improve local models without sharing (private) data, but only updating the weights with minimal requirements.
- There are many parameters in FL, but the mixing model is very simple (averaging) → FedAvg as the baseline FL algorithm

What have we learned?

- Stochastic problems in machine learning are quite similar in spirit to distributed optimization (finite sum structure)
- ML uses different names for things but the tools are similar, however often the assumptions we are allowed to make are not the same.
- SGD is the workhorse of ML, if you work in ML you will use to train all your complicated models (e.g., neural networks are trained using variants of SGD)
- SAG and SAGA can boost performance of SGD as gradient tracking for DGD
- Federated learning is a way to improve local models without sharing (private) data, but only updating the weights with minimal requirements.
- There are many parameters in FL, but the mixing model is very simple (averaging) → FedAvg as the baseline FL algorithm
- FedAvg is local SGD steps followed by averaging (trade-off between number of local steps and number of averaging)

Sample references

SGD, SAG, SAGA:

- ① *Aaron Defazio, Francis Bach, Simon Lacoste-Julien, **SAGA: A Fast Incremental Gradient Method With Support for Non-Strongly Convex Composite Objectives**, NeurIPS, 2014*

Federated learning:

- ① *Many, many authors, **Advances and Open Problems in Federated Learning**, arXiv:1912.04977, 2019*
- ② *Li et al., **On the Convergence of FedAvg on Non-IID Data**, arXiv:1907.02189, 2019*
- ③ Many references in the above

Federated Learning with non-IID data

- We move now from IID data (independent, identically distributed) to the more realistic non-IID data

Federated Learning with non-IID data

- We move now from IID data (independent, identically distributed) to the more realistic non-IID data
- We have seen how FedAvg can “converge” even in this setting; but we shouldn’t be really happy about the results

Federated Learning with non-IID data

- We move now from IID data (independent, identically distributed) to the more realistic non-IID data
- We have seen how FedAvg can “converge” even in this setting; but we shouldn’t be really happy about the results
- Non-IID data comes from the fact that different devices are indeed different and their distribution is different

Federated Learning with non-IID data

- We move now from IID data (independent, identically distributed) to the more realistic non-IID data
- We have seen how FedAvg can “converge” even in this setting; but we shouldn’t be really happy about the results
- Non-IID data comes from the fact that different devices are indeed different and their distribution is different
- A good way to think about it is “cross-device” FL and “cross-silo” FL

Federated Learning with non-IID data

- We move now from IID data (independent, identically distributed) to the more realistic non-IID data
- We have seen how FedAvg can “converge” even in this setting; but we shouldn’t be really happy about the results
- Non-IID data comes from the fact that different devices are indeed different and their distribution is different
- A good way to think about it is “cross-device” FL and “cross-silo” FL
- **Cross-device FL** is FL when you have a huge number of small devices (like phones) in an IoT setting. If all the devices are identical, then you could assume IID data. If the devices are very different then you cannot assume IID data. The IID assumption here can save communication rounds and complexity.

Federated Learning with non-IID data

- We move now from IID data (independent, identically distributed) to the more realistic non-IID data
- We have seen how FedAvg can “converge” even in this setting; but we shouldn’t be really happy about the results
- Non-IID data comes from the fact that different devices are indeed different and their distribution is different
- A good way to think about it is “cross-device” FL and “cross-silo” FL
- **Cross-device FL** is FL when you have a huge number of small devices (like phones) in an IoT setting. If all the devices are identical, then you could assume IID data. If the devices are very different then you cannot assume IID data. The IID assumption here can save communication rounds and complexity.
- **Cross-silo FL** is FL when you have very few agents who carry a huge quantity of data. Think of it as FL between companies or hospitals. Each of the agents have their local, very sophisticated models, and they share them to get even better models. In this case, non-IID is unavoidable. Here you can assume that you have all the bandwidth that you want, so you can do something more sophisticated.

Federated Learning with non-IID data

- In any case, recall the problem that we have,

$$\begin{aligned} \min_{\mathbf{x} \in \mathbb{R}^n} \quad & \mathbf{E}_{\theta \sim \Theta} [f(\mathbf{x}; \theta)] \\ \equiv & \frac{1}{P} \sum_{p=1}^P \mathbf{E}_{\theta \sim \Theta} [f(\mathbf{x}; \theta)] \approx \frac{1}{P} \sum_{p=1}^P \mathbf{E}_{\theta \sim \Theta_p} [f(\mathbf{x}; \theta)] \\ \approx & \frac{1}{P} \sum_{p=1}^P \frac{1}{N_p} \sum_{j=1}^{N_p} f_j(\mathbf{x}) = \frac{1}{P} \sum_{p=1}^P f_p(\mathbf{x}) \end{aligned}$$

Federated Learning with non-IID data

- In any case, recall the problem that we have,

$$\begin{aligned} \min_{\mathbf{x} \in \mathbb{R}^n} \quad & \mathbf{E}_{\theta \sim \Theta} [f(\mathbf{x}; \theta)] \\ \equiv & \frac{1}{P} \sum_{p=1}^P \mathbf{E}_{\theta \sim \Theta} [f(\mathbf{x}; \theta)] \approx \frac{1}{P} \sum_{p=1}^P \mathbf{E}_{\theta \sim \Theta_p} [f(\mathbf{x}; \theta)] \\ \approx & \frac{1}{P} \sum_{p=1}^P \frac{1}{N_p} \sum_{j=1}^{N_p} f_j(\mathbf{x}) = \frac{1}{P} \sum_{p=1}^P f_p(\mathbf{x}) \end{aligned}$$

- The fundamental issue is that,

$$\begin{aligned} \mathbf{x}^* &= \arg \min_{\mathbf{x} \in \mathbb{R}^n} \mathbf{E}_{\theta \sim \Theta} [f(\mathbf{x}; \theta)] = \frac{1}{P} \sum_{p=1}^P \arg \min_{\mathbf{x} \in \mathbb{R}^n} \mathbf{E}_{\theta \sim \Theta} [f(\mathbf{x}; \theta)] \\ &\neq \frac{1}{P} \sum_{p=1}^P \arg \min_{\mathbf{x} \in \mathbb{R}^n} \mathbf{E}_{\theta \sim \Theta_p} [f(\mathbf{x}; \theta)] = \frac{1}{P} \sum_{p=1}^P \mathbf{x}_p^* \end{aligned}$$

Federated Learning with non-IID data

- So, no matter the mixing matrix we select at the server level, the model \mathbf{x} is going to drift. We call this phenomenon client-drift.

Federated Learning with non-IID data

- So, no matter the mixing matrix we select at the server level, the model \mathbf{x} is going to drift. We call this phenomenon client-drift.
- How can we fix it? Do we need to fix it?

Federated Learning with non-IID data

- So, no matter the mixing matrix we select at the server level, the model \mathbf{x} is going to drift. We call this phenomenon client-drift.
- How can we fix it? Do we need to fix it?
- Of course we would like to get better models, so we want to fix it. But one has to be careful when to do it (sometimes the data are so different than local models are the only thing you need).

Federated Learning with non-IID data

- So, no matter the mixing matrix we select at the server level, the model \mathbf{x} is going to drift. We call this phenomenon client-drift.
- How can we fix it? Do we need to fix it?
- Of course we would like to get better models, so we want to fix it. But one has to be careful when to do it (sometimes the data are so different than local models are the only thing you need).
- We are going to see one way to fix the issue of non-IID data with variance reduction, aka gradient tracking. The method is called SCAFFOLD. Many other methods exist, but we will focus on this one due to its popularity.

Federated Learning with non-IID data

- So, no matter the mixing matrix we select at the server level, the model \mathbf{x} is going to drift. We call this phenomenon client-drift.
- How can we fix it? Do we need to fix it?
- Of course we would like to get better models, so we want to fix it. But one has to be careful when to do it (sometimes the data are so different than local models are the only thing you need).
- We are going to see one way to fix the issue of non-IID data with variance reduction, aka gradient tracking. The method is called SCAFFOLD. Many other methods exist, but we will focus on this one due to its popularity.
- SCAFFOLD is a generalization of SAGA, as we will prove.

Federated Learning with non-IID data

- So, no matter the mixing matrix we select at the server level, the model x is going to drift. We call this phenomenon client-drift.
- How can we fix it? Do we need to fix it?
- Of course we would like to get better models, so we want to fix it. But one has to be careful when to do it (sometimes the data are so different than local models are the only thing you need).
- We are going to see one way to fix the issue of non-IID data with variance reduction, aka gradient tracking. The method is called SCAFFOLD. Many other methods exist, but we will focus on this one due to its popularity.
- SCAFFOLD is a generalization of SAGA, as we will prove.
- Incidentally, SCAFFOLD will have better convergence guarantees

SCAFFOLD: setting and assumptions

- Rewrite the closest problem we can solve, for C client as,

$$\min_{\mathbf{x} \in \mathbb{R}^n} f(\mathbf{x}) = \frac{1}{C} \sum_{c=1}^C (f_c(\mathbf{x}) := \mathbf{E}_{\theta \in \Theta_c} [f(\mathbf{x}; \theta)])$$

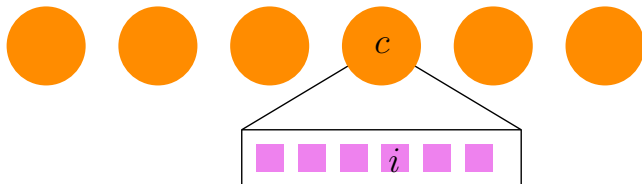
SCAFFOLD: setting and assumptions

- Rewrite the closest problem we can solve, for C client as,

$$\min_{\mathbf{x} \in \mathbb{R}^n} f(\mathbf{x}) = \frac{1}{C} \sum_{c=1}^C (f_c(\mathbf{x}) := \mathbf{E}_{\theta \in \Theta_c} [f(\mathbf{x}; \theta)])$$

- Call $\mathbf{g}_{i,c}(\mathbf{x}) := \nabla f(\mathbf{x}; \theta_{i,c})$ as the gradient sampled from the training data i (or batch i) belonging to client c . Assume that $\mathbf{g}_{i,c}(\mathbf{x})$ is an unbiased estimator for $\nabla f_c(\mathbf{x})$ with variance,

$$\mathbf{E}_i [\|\mathbf{g}_{i,c}(\mathbf{x}) - \nabla f_c(\mathbf{x})\|^2] \leq \pi^2.$$



Federated Learning with non-IID data

- Cross check: e.g., for SAGA, we have access to f_c directly, so $\pi = 0$. Rem for SGD/SAGA our cost is the empirical risk,

$$\min_{\mathbf{x}} \frac{1}{C} \sum_{c=1}^C f_c(\mathbf{x})$$

where each “client” has some realizations of the data.

Federated Learning with non-IID data

- Cross check: e.g., for SAGA, we have access to f_c directly, so $\pi = 0$. Rem for SGD/SAGA our cost is the empirical risk,

$$\min_{\mathbf{x}} \frac{1}{C} \sum_{c=1}^C f_c(\mathbf{x})$$

where each “client” has some realizations of the data.

- Here, we solve instead,

$$\min_{\mathbf{x}} \frac{1}{C} \sum_{c=1}^C \mathbf{E}_{\theta \in \Theta_c} [f(\mathbf{x}; \theta)] \approx \min_{\mathbf{x}} \frac{1}{C} \sum_{c=1}^C \frac{1}{N_c} \sum_{i=1}^{N_c} f(\mathbf{x}; \theta_i)$$

Federated Learning with non-IID data

- Cross check: e.g., for SAGA, we have access to f_c directly, so $\pi = 0$. Rem for SGD/SAGA our cost is the empirical risk,

$$\min_{\mathbf{x}} \frac{1}{C} \sum_{c=1}^C f_c(\mathbf{x})$$

where each “client” has some realizations of the data.

- Here, we solve instead,

$$\min_{\mathbf{x}} \frac{1}{C} \sum_{c=1}^C \mathbf{E}_{\theta \in \Theta_c} [f(\mathbf{x}; \theta)] \approx \min_{\mathbf{x}} \frac{1}{C} \sum_{c=1}^C \frac{1}{N_c} \sum_{i=1}^{N_c} f(\mathbf{x}; \theta_i)$$

- Back to SCAFFOLD: Assume also, we have $G \geq 0, B \geq 1$ such that:

$$\frac{1}{C} \sum_{c=1}^C \|\nabla f_c(\mathbf{x})\|^2 \leq G^2 + B^2 \|\nabla f(\mathbf{x})\|^2, \quad \forall \mathbf{x}$$

SCAFFOLD: setting and assumptions

- π takes care of the inter-client variance

How accurate is this approximation?

$$f_c(\mathbf{x}) = \mathbf{E}_{\theta \in \Theta_c}[f(\mathbf{x}; \theta)] \approx \frac{1}{N_c} \sum_{i=1}^{N_c} f(\mathbf{x}; \theta_i)$$

SCAFFOLD: setting and assumptions

- π takes care of the inter-client variance
- G, B take care of the non-IIDness and it is a slightly different assumption than the one in SGD.

How different is the gradient mean with the true gradient ?

$$\frac{1}{C} \sum_{c=1}^C \|\nabla f_c(\mathbf{x})\|^2 \leq G^2 + B^2 \|\nabla f(\mathbf{x})\|^2, \quad \forall \mathbf{x}$$

SCAFFOLD: setting and assumptions

- π takes care of the inter-client variance
- G, B take care of the non-IIDness and it is a slightly different assumption than the one in SGD.
- Many sets of different assumptions exist, but the bottom line is: bound the inter-client variance, bound the infra-client variance.

SCAFFOLD: setting and assumptions

- π takes care of the inter-client variance
- G, B take care of the non-IIDness and it is a slightly different assumption than the one in SGD.
- Many sets of different assumptions exist, but the bottom line is: bound the inter-client variance, bound the infra-client variance.
- Also here we talk about convex function, but ML and FL cares about non-convex function (more). Most of the results can be extended to smooth non-convex functions of various type. An important type is the weakly convex type.

SCAFFOLD: Algorithm

- **Server update.** Initialize $\mathbf{x}_0, \mathbf{y}_0, \mathbf{c}_c$. For each time $t = 1, \dots$ do:
 - ▶ Select $S \leq C$ clients, and for each client
 - ▶ Send $\mathbf{x}_t, \mathbf{y}_t$ to client
 - ▶ Receive $\Delta \mathbf{x}_{t+1}^s, \Delta \mathbf{y}_{t+1}^s$ from the s -th **client update**

SCAFFOLD: Algorithm

- **Server update.** Initialize $\mathbf{x}_0, \mathbf{y}_0, \mathbf{c}_c$. For each time $t = 1, \dots$ do:
 - ▶ Select $S \leq C$ clients, and for each client
 - ▶ Send $\mathbf{x}_t, \mathbf{y}_t$ to client
 - ▶ Receive $\Delta \mathbf{x}_{t+1}^s, \Delta \mathbf{y}_{t+1}^s$ from the s -th **client update**
- **Client update.** Start from $\mathbf{x}_0^i = \mathbf{x}_t, \mathbf{y}_0^i = \mathbf{y}_t$. Select batches of data B .
 - ▶ For epoch $k = 1, \dots, E$:
Local SAGA, Compute mini-batch gradient $\mathbf{g}_{j,c}(\mathbf{x}_k^c)$ for batch j , client c , update:

$$\mathbf{x}_{k+1}^c = \mathbf{x}_k^c - \alpha (\mathbf{g}_{j,c}(\mathbf{x}_k^c) - \mathbf{c}_c + \mathbf{y}_t)$$

- ▶ Compute new quantities,

$$\mathbf{c}_c^+ = \begin{cases} \mathbf{g}_{j,c}(\mathbf{x}_t) & \text{option I} \\ \mathbf{c}_c - \mathbf{y}_t + \frac{1}{E\alpha}(\mathbf{x}_t - \mathbf{x}_{E+1}^c) & \text{option II} \end{cases}$$

and the delta's:

$$(\Delta \mathbf{x}_{t+1}^c, \Delta \mathbf{y}_{t+1}^c) = (\mathbf{x}_{E+1}^c - \mathbf{x}_t, \mathbf{c}_c^+ - \mathbf{c}_c) \quad \text{then: } \mathbf{c}_c \leftarrow \mathbf{c}_c^+$$

SCAFFOLD: Algorithm

- **Server update.** Initialize $\mathbf{x}_0, \mathbf{y}_0, \mathbf{c}_c$. For each time $t = 1, \dots$ do:

- ▶ Select $S \leq C$ clients, and for each client
- ▶ Send $\mathbf{x}_t, \mathbf{y}_t$ to client
- ▶ Receive $\Delta \mathbf{x}_{t+1}^s, \Delta \mathbf{y}_{t+1}^s$ from the s -th **client update**
- ▶ Mixing:

$$(\mathbf{x}_{t+1}, \mathbf{y}_{t+1}) = (\mathbf{x}_t + \gamma \frac{1}{S} \sum_{s=1}^S \Delta \mathbf{x}_{t+1}^s, \mathbf{y}_t + \frac{1}{C} \sum_{s=1}^S \Delta \mathbf{y}_{t+1}^s)$$

- **Client update.** Start from $\mathbf{x}_0^i = \mathbf{x}_t, \mathbf{y}_0^i = \mathbf{y}_t$. Select batches of data B .

- ▶ For epoch $k = 1, \dots, E$:

Local SAGA, Compute mini-batch gradient $\mathbf{g}_{j,c}(\mathbf{x}_k^c)$ for batch j , client c , update:

$$\mathbf{x}_{k+1}^c = \mathbf{x}_k^c - \alpha (\mathbf{g}_{j,c}(\mathbf{x}_k^c) - \mathbf{c}_c + \mathbf{y}_t)$$

- ▶ Compute new quantities,

$$\mathbf{c}_c^+ = \begin{cases} \mathbf{g}_{j,c}(\mathbf{x}_t) & \text{option I} \\ \mathbf{c}_c - \mathbf{y}_t + \frac{1}{E\alpha}(\mathbf{x}_t - \mathbf{x}_{E+1}^c) & \text{option II} \end{cases}$$

and the delta's:

$$(\Delta \mathbf{x}_{t+1}^c, \Delta \mathbf{y}_{t+1}^c) = (\mathbf{x}_{E+1}^c - \mathbf{x}_t, \mathbf{c}_c^+ - \mathbf{c}_c) \quad \text{then: } \mathbf{c}_c \leftarrow \mathbf{c}_c^+$$

SCAFFOLD: special cases

- If you get $\gamma = 1$, $\alpha = \alpha_k$ and do not consider $-\mathbf{c}_c + \mathbf{y}_t$, then we get back FedAvg

SCAFFOLD: special cases

- If you get $\gamma = 1$, $\alpha = \alpha_k$ and do not consider $-\mathbf{c}_c + \mathbf{y}_t$, then we get back FedAvg
- In SCAFFOLD we use two step sizes α, γ , which can be constant!

SCAFFOLD: convergence

Theorem 12 (SCAFFOLD)

For any L -smooth and m -strongly convex f_c function, with the assumptions above, the output of SCAFFOLD has expected error smaller than ϵ for small enough step size selections $\alpha \leq \frac{1}{L}$ and iterations bounded as

$$T = \tilde{O}\left(\frac{\pi^2}{mES\epsilon} + \frac{L}{m} + \frac{C}{S}\right)$$

as well as,

$$\mathbf{E}[F(\bar{\mathbf{x}}_T) - F(\mathbf{x}^*)] = \tilde{O}\left(\frac{\pi^2}{mTCE} + \exp(-T)\right)$$

for a suitable average sequence $\bar{\mathbf{x}}_T$.

- The notation $\tilde{O}(\cdot)$ means $O(\cdot)$ up to poly-log factors.

SCAFFOLD: convergence

Theorem 12 (SCAFFOLD)

For any L -smooth and m -strongly convex f_c function, with the assumptions above, the output of SCAFFOLD has expected error smaller than ϵ for small enough step size selections $\alpha \leq \frac{1}{L}$ and iterations bounded as

$$T = \tilde{O}\left(\frac{\pi^2}{mES\epsilon} + \frac{L}{m} + \frac{C}{S}\right)$$

as well as,

$$\mathbf{E}[F(\bar{\mathbf{x}}_T) - F(\mathbf{x}^*)] = \tilde{O}\left(\frac{\pi^2}{mTCE} + \exp(-T)\right)$$

for a suitable average sequence $\bar{\mathbf{x}}_T$.

- The notation $\tilde{O}(\cdot)$ means $O(\cdot)$ up to poly-log factors.
- This is one variant. Other exists for different assumptions.

SCAFFOLD: convergence

Theorem 12 (SCAFFOLD)

For any L -smooth and m -strongly convex f_c function, with the assumptions above, the output of SCAFFOLD has expected error smaller than ϵ for small enough step size selections $\alpha \leq \frac{1}{L}$ and iterations bounded as

$$T = \tilde{O}\left(\frac{\pi^2}{mES\epsilon} + \frac{L}{m} + \frac{C}{S}\right)$$

as well as,

$$\mathbf{E}[F(\bar{\mathbf{x}}_T) - F(\mathbf{x}^*)] = \tilde{O}\left(\frac{\pi^2}{mTCE} + \exp(-T)\right)$$

for a suitable average sequence $\bar{\mathbf{x}}_T$.

- The notation $\tilde{O}(\cdot)$ means $O(\cdot)$ up to poly-log factors.
- This is one variant. Other exists for different assumptions.
- SAGA: $\pi = 0$, so $T = \tilde{O}(1)$ which matches SAGA $O(\log(1/\epsilon)) = \tilde{O}(1)$

SCAFFOLD: convergence

Theorem 12 (SCAFFOLD)

For any L -smooth and m -strongly convex f_c function, with the assumptions above, the output of SCAFFOLD has expected error smaller than ϵ for small enough step size selections $\alpha \leq \frac{1}{L}$ and iterations bounded as

$$T = \tilde{O}\left(\frac{\pi^2}{mES\epsilon} + \frac{L}{m} + \frac{C}{S}\right)$$

as well as,

$$\mathbf{E}[F(\bar{\mathbf{x}}_T) - F(\mathbf{x}^*)] = \tilde{O}\left(\frac{\pi^2}{mTCE} + \exp(-T)\right)$$

for a suitable average sequence $\bar{\mathbf{x}}_T$.

- The notation $\tilde{O}(\cdot)$ means $O(\cdot)$ up to poly-log factors.
- This is one variant. Other exists for different assumptions.
- SAGA: $\pi = 0$, so $T = \tilde{O}(1)$ which matches SAGA $O(\log(1/\epsilon)) = \tilde{O}(1)$
- SCAFFOLD is not the only method for non-IID data that matches SAGA and the research stays very rich

Let's summarize!

- It's very hard to feel a bit lost in the FL literature, since there are many different algorithms with weird names, etc..

Let's summarize!

- It's very hard to feel a bit lost in the FL literature, since there are many different algorithms with weird names, etc..
- But at the end, don't forget that the ideas are simple:

Let's summarize!

- It's very hard to feel a bit lost in the FL literature, since there are many different algorithms with weird names, etc..
- But at the end, don't forget that the ideas are simple:
 - ▶ You get your training data or functions f_i , and you take directions along the gradients of them (either sequentially, or in parallel)

Let's summarize!

- It's very hard to feel a bit lost in the FL literature, since there are many different algorithms with weird names, etc..
- But at the end, don't forget that the ideas are simple:
 - ▶ You get your training data or functions f_i , and you take directions along the gradients of them (either sequentially, or in parallel)
 - ▶ Then you mix the result with a weight matrix.

Let's summarize!

- It's very hard to feel a bit lost in the FL literature, since there are many different algorithms with weird names, etc..
- But at the end, don't forget that the ideas are simple:
 - ▶ You get your training data or functions f_i , and you take directions along the gradients of them (either sequentially, or in parallel)
 - ▶ Then you mix the result with a weight matrix.
 - ▶ If you want to do fancier things, you add an “integrator” (aka gradient tracking)

Let's summarize!

- It's very hard to feel a bit lost in the FL literature, since there are many different algorithms with weird names, etc..
- But at the end, don't forget that the ideas are simple:
 - ▶ You get your training data or functions f_i , and you take directions along the gradients of them (either sequentially, or in parallel)
 - ▶ Then you mix the result with a weight matrix.
 - ▶ If you want to do fancier things, you add an “integrator” (aka gradient tracking)
 - ▶ In distributed optimization we can even look at the dual problems; in federated that may be too memory costly (remember: you give copies of each variables to all the participating nodes).

Let's summarize!

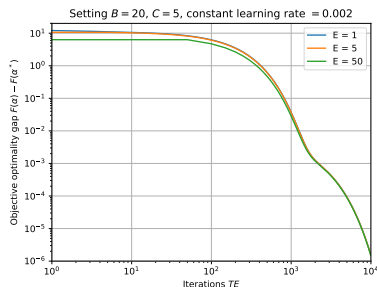
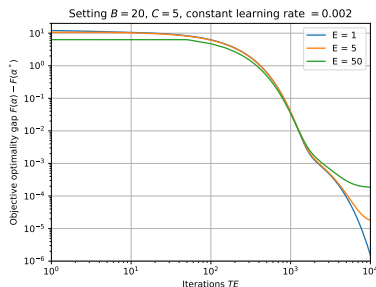
- It's very hard to feel a bit lost in the FL literature, since there are many different algorithms with weird names, etc..
- But at the end, don't forget that the ideas are simple:
 - ▶ You get your training data or functions f_i , and you take directions along the gradients of them (either sequentially, or in parallel)
 - ▶ Then you mix the result with a weight matrix.
 - ▶ If you want to do fancier things, you add an “integrator” (aka gradient tracking)
 - ▶ In distributed optimization we can even look at the dual problems; in federated that may be too memory costly (remember: you give copies of each variables to all the participating nodes).
 - ▶ But in FL, since you are allowed to do more on the single client, you may be tempted to use second-order methods (quasi-Newton), see the **DANE** algorithm [arXiv: 1312.7853], and others

Let's summarize!

- It's very hard to feel a bit lost in the FL literature, since there are many different algorithms with weird names, etc..
- But at the end, don't forget that the ideas are simple:
 - ▶ You get your training data or functions f_i , and you take directions along the gradients of them (either sequentially, or in parallel)
 - ▶ Then you mix the result with a weight matrix.
 - ▶ If you want to do fancier things, you add an “integrator” (aka gradient tracking)
 - ▶ In distributed optimization we can even look at the dual problems; in federated that may be too memory costly (remember: you give copies of each variables to all the participating nodes).
 - ▶ But in FL, since you are allowed to do more on the single client, you may be tempted to use second-order methods (quasi-Newton), see the **DANE** algorithm [arXiv: 1312.7853], and others
 - ▶ Then, it's mix and match!

Numerical results I

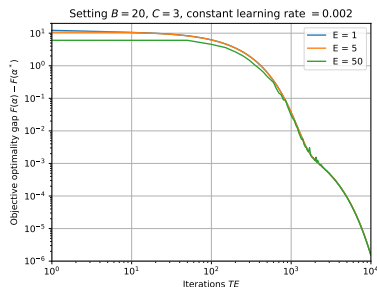
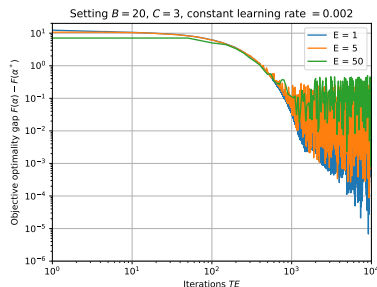
FedAvg on the left \rightarrow SCAFFOLD on the right



- SCAFFOLD $E = 1$, $B = 20$ it's SAGA.
- SCAFFOLD benefits from local computation (a larger E): reduces communication without compromising convergence. Here: Option I.

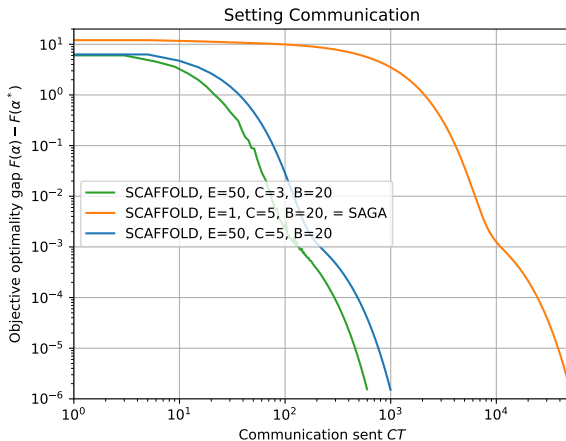
Numerical results II

FedAvg on the left \rightarrow SCAFFOLD on the right



- SCAFFOLD full batch setting. Here: Option II.

Numerical results III



- Communication: SCAFFOLD benefits from $E > 1$! But you have trade-offs.

Part VI

Advanced FL

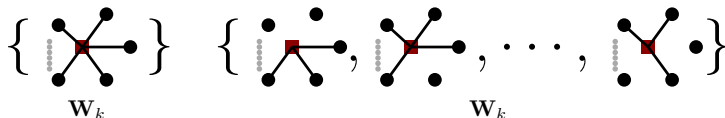
Peer-to-peer (aka distributed) FL ?

Let us see if we can “distribute” FL

Recall: FedAvg can be rewritten as,

$$\begin{aligned} \mathbf{x}_{k+1}^i &= \mathbf{x}_k^i - \alpha_k \nabla f_{B_j}(\mathbf{x}_k^i), & k \in \mathcal{T} \\ \mathbf{x}_{k+1}^i &= \sum_{c=1}^C w_c [\mathbf{x}_k^c - \alpha_k \nabla f_{B_j}(\mathbf{x}_k^c)], & k \notin \mathcal{T} \end{aligned}$$

- FedAvg uses therefore a star-topology if you select all the nodes, and a sequence of star topologies, if you select a few clients at the time. Here \square is the server.



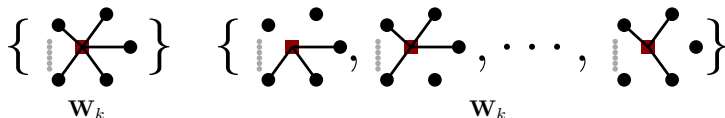
Peer-to-peer (aka distributed) FL ?

Let us see if we can “distribute” FL

Recall: FedAvg can be rewritten as,

$$\begin{aligned} \mathbf{x}_{k+1}^i &= \mathbf{x}_k^i - \alpha_k \nabla f_{B_j}(\mathbf{x}_k^i), & k \in \mathcal{T} \\ \mathbf{x}_{k+1}^i &= \sum_{c=1}^C w_c [\mathbf{x}_k^c - \alpha_k \nabla f_{B_j}(\mathbf{x}_k^c)], & k \notin \mathcal{T} \end{aligned}$$

- FedAvg uses therefore a star-topology if you select all the nodes, and a sequence of star topologies, if you select a few clients at the time. Here \square is the server.



- Here, we want to remove the need for a central server and instead of $\sum_{c=1}^C w_c$, we only want to sum with our neighbors!

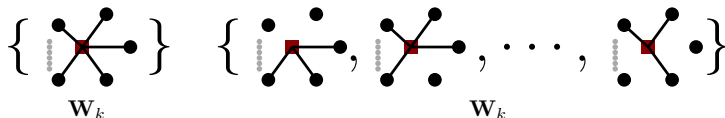
Peer-to-peer (aka distributed) FL ?

Let us see if we can “distribute” FL

Recall: FedAvg can be rewritten as,

$$\begin{aligned} \mathbf{x}_{k+1}^i &= \mathbf{x}_k^i - \alpha_k \nabla f_{B_j}(\mathbf{x}_k^i), & k \in \mathcal{T} \\ \mathbf{x}_{k+1}^i &= \sum_{c=1}^C w_c [\mathbf{x}_k^c - \alpha_k \nabla f_{B_j}(\mathbf{x}_k^c)], & k \notin \mathcal{T} \end{aligned}$$

- FedAvg uses therefore a star-topology if you select all the nodes, and a sequence of star topologies, if you select a few clients at the time. Here \square is the server.



- Here, we want to remove the need for a central server and instead of $\sum_{c=1}^C w_c$, we only want to sum with our neighbors!
- This mixes FedAvg with distributed optimization and consensus.

Peer-to-peer (aka distributed) FL ?

- Call as usual, $\mathbf{y}_k := [\mathbf{x}_k^1, \dots, \mathbf{x}_k^C]$, $\mathbf{x} \in \mathbf{R}^n$, then, we can write the mixing step as,

$$\mathbf{y}_{k+1} = \mathbf{W}_k \mathbf{y}_k.$$

Peer-to-peer (aka distributed) FL ?

- Call as usual, $\mathbf{y}_k := [\mathbf{x}_k^1, \dots, \mathbf{x}_k^C]$, $\mathbf{x} \in \mathbf{R}^n$, then, we can write the mixing step as,

$$\mathbf{y}_{k+1} = \mathbf{W}_k \mathbf{y}_k.$$

- In the star topology case with full participation then

$$\mathbf{W}_k = \left(\frac{\mathbf{1}_C \mathbf{1}_C^\top}{C} \right) \otimes I_n = \frac{1}{C} \begin{bmatrix} I_n & \cdots & I_n \\ \vdots & \ddots & \vdots \\ I_n & \cdots & I_n \end{bmatrix},$$

which is the mixing matrix.

Peer-to-peer (aka distributed) FL ?

- Call as usual, $\mathbf{y}_k := [\mathbf{x}_k^1, \dots, \mathbf{x}_k^C]$, $\mathbf{x} \in \mathbf{R}^n$, then, we can write the mixing step as,

$$\mathbf{y}_{k+1} = \mathbf{W}_k \mathbf{y}_k.$$

- In the star topology case with full participation then

$$\mathbf{W}_k = \left(\frac{\mathbf{1}_C \mathbf{1}_C^\top}{C} \right) \otimes I_n = \frac{1}{C} \begin{bmatrix} I_n & \cdots & I_n \\ \vdots & \ddots & \vdots \\ I_n & \cdots & I_n \end{bmatrix},$$

which is the mixing matrix.

- In all other cases \mathbf{W}_k approximates $\left(\frac{\mathbf{1}_C \mathbf{1}_C^\top}{C} \right) \otimes I_n$. In the distributed case, we consider only the neighbors.

Now what?

- You can go back to the second class and substitute the star topology with a distributed topology and do consensus on it.

Now what?

- You can go back to the second class and substitute the star topology with a distributed topology and do consensus on it.
- The most general setting is when you do one communication round (mixing round) and then you continue with your other computations. This is the setting of FedAvg and DGD.

Now what?

- You can go back to the second class and substitute the star topology with a distributed topology and do consensus on it.
- The most general setting is when you do one communication round (mixing round) and then you continue with your other computations. This is the setting of FedAvg and DGD.
- In the peer-to-peer FL task, we will have convergence guarantees that depend on the spectral gap (with an additional term), and on the choice of the communication protocol.

Now what?

- You can go back to the second class and substitute the star topology with a distributed topology and do consensus on it.
- The most general setting is when you do one communication round (mixing round) and then you continue with your other computations. This is the setting of FedAvg and DGD.
- In the peer-to-peer FL task, we will have convergence guarantees that depend on the spectral gap (with an additional term), and on the choice of the communication protocol.
- So the FL task is typically slowed down and you know who is the culprit.

FL in the cross-silo setting

- Let's now take a step back and look at the realistic cross-silo setting. In this case you have a few entities with large quantities of data, which train their local models

FL in the cross-silo setting

- Let's now take a step back and look at the realistic cross-silo setting. In this case you have a few entities with large quantities of data, which train their local models
- Visualise it as training a classification model to detect cancer in a hospital setting. Each hospital is an agent/entity and hospitals may share their models to get better global models.

FL in the cross-silo setting

- Let's now take a step back and look at the realistic cross-silo setting. In this case you have a few entities with large quantities of data, which train their local models
- Visualise it as training a classification model to detect cancer in a hospital setting. Each hospital is an agent/entity and hospitals may share their models to get better global models.
- Are they really better these aggregated models? And for whom are they better?

FL in the cross-silo setting

- Let's now take a step back and look at the realistic cross-silo setting. In this case you have a few entities with large quantities of data, which train their local models
- Visualise it as training a classification model to detect cancer in a hospital setting. Each hospital is an agent/entity and hospitals may share their models to get better global models.
- Are they really better these aggregated models? And for whom are they better?
- This last section brings a bit beyond “classical” optimization into heuristic evidence. Recall what we wanted to solve and what we ended up solving:

$$\min_{\mathbf{x}} \mathbf{E}_{\theta \in \Theta} [f(\mathbf{x}; \theta)] \approx \min_{\mathbf{x}} \frac{1}{P} \sum_{p=1}^P \mathbf{E}_{\theta \in \Theta_p} [f(\mathbf{x}; \theta)] \approx \min_{\mathbf{x}} \frac{1}{P} \sum_{p=1}^P \frac{1}{N_p} \sum_{i=1}^{N_p} f_i(\mathbf{x})$$

FL in the cross-silo setting

- Let's now take a step back and look at the realistic cross-silo setting. In this case you have a few entities with large quantities of data, which train their local models
- Visualise it as training a classification model to detect cancer in a hospital setting. Each hospital is an agent/entity and hospitals may share their models to get better global models.
- Are they really better these aggregated models? And for whom are they better?
- This last section brings a bit beyond “classical” optimization into heuristic evidence. Recall what we wanted to solve and what we ended up solving:

$$\min_{\mathbf{x}} \mathbf{E}_{\theta \in \Theta} [f(\mathbf{x}; \theta)] \approx \min_{\mathbf{x}} \frac{1}{P} \sum_{p=1}^P \mathbf{E}_{\theta \in \Theta_p} [f(\mathbf{x}; \theta)] \approx \min_{\mathbf{x}} \frac{1}{P} \sum_{p=1}^P \frac{1}{N_p} \sum_{i=1}^{N_p} f_i(\mathbf{x})$$

- Especially in cross-silo FL, the first problem does not make sense: Θ_p are so different, that trying to get back to Θ has little meaning.

FL in the cross-silo setting

- Let's now take a step back and look at the realistic cross-silo setting. In this case you have a few entities with large quantities of data, which train their local models
- Visualise it as training a classification model to detect cancer in a hospital setting. Each hospital is an agent/entity and hospitals may share their models to get better global models.
- Are they really better these aggregated models? And for whom are they better?
- This last section brings a bit beyond “classical” optimization into heuristic evidence. Recall what we wanted to solve and what we ended up solving:

$$\min_{\mathbf{x}} \mathbf{E}_{\theta \in \Theta} [f(\mathbf{x}; \theta)] \approx \min_{\mathbf{x}} \frac{1}{P} \sum_{p=1}^P \mathbf{E}_{\theta \in \Theta_p} [f(\mathbf{x}; \theta)] \approx \min_{\mathbf{x}} \frac{1}{P} \sum_{p=1}^P \frac{1}{N_p} \sum_{i=1}^{N_p} f_i(\mathbf{x})$$

- Especially in cross-silo FL, the first problem does not make sense: Θ_p are so different, that trying to get back to Θ has little meaning.
- However, staying with local models may be too restrictive.

Personalized FL

- Idea: share the model, do the mixing, but then “project” back onto the local dataset → personalization

Personalized FL

- Idea: share the model, do the mixing, but then “project” back onto the local dataset → personalization
- Visualise it as training a language model, where each local models are trained on local dialects

Personalized FL

- Idea: share the model, do the mixing, but then “project” back onto the local dataset → personalization
- Visualise it as training a language model, where each local models are trained on local dialects
- How would you do that?

Personalized FL

- Idea: share the model, do the mixing, but then “project” back onto the local dataset → personalization
- Visualise it as training a language model, where each local models are trained on local dialects
- How would you do that?
- “Tamper” with the local functions f_i , adding a local bonus. So instead of $f_i(\mathbf{x})$, you have $f_i(\xi_i(\mathbf{x}))$. And in particular you will have a local model ξ_i and a global model \mathbf{x} .

Personalized FL

- Idea: share the model, do the mixing, but then “project” back onto the local dataset → personalization
- Visualise it as training a language model, where each local models are trained on local dialects
- How would you do that?
- “Tamper” with the local functions f_i , adding a local bonus. So instead of $f_i(\mathbf{x})$, you have $f_i(\xi_i(\mathbf{x}))$. And in particular you will have a local model ξ_i and a global model \mathbf{x} .
- You are decoupling the local and the global models so that they may converge to different things!

Personalized FL

- Idea: share the model, do the mixing, but then “project” back onto the local dataset → personalization
- Visualise it as training a language model, where each local models are trained on local dialects
- How would you do that?
- “Tamper” with the local functions f_i , adding a local bonus. So instead of $f_i(\mathbf{x})$, you have $f_i(\xi_i(\mathbf{x}))$. And in particular you will have a local model ξ_i and a global model \mathbf{x} .
- You are decoupling the local and the global models so that they may converge to different things!
- Examples

$$\text{(Regularization)} \quad f_i(\xi_i(\mathbf{x})) = \min_{\xi_i} \{ f_i(\xi_i) + \frac{\lambda}{2} \|\xi_i - \mathbf{x}\|^2 \}$$

$$\text{(Initialization)} \quad f_i(\xi_i(\mathbf{x})) = f_i(\underbrace{\mathbf{x} - \alpha \nabla f_i(\mathbf{x})}_{=\xi_i})$$

Personalized FL

- The rest stays the same, and you can use your FedAvg applied to $f_i(\xi_i(\mathbf{x}))$ to determine both ξ_i and \mathbf{x} .

Personalized FL

- The rest stays the same, and you can use your FedAvg applied to $f_i(\xi_i(\mathbf{x}))$ to determine both ξ_i and \mathbf{x} .
- Does this work? Theoretically it converges in some sense, and in practice: sort of, when it makes sense to use it.

Personalized FL

- The rest stays the same, and you can use your FedAvg applied to $f_i(\xi_i(\mathbf{x}))$ to determine both ξ_i and \mathbf{x} .
- Does this work? Theoretically it converges in some sense, and in practice: sort of, when it makes sense to use it.
- All of this FL seems a bit wacky... but,

Personalized FL

- The rest stays the same, and you can use your FedAvg applied to $f_i(\xi_i(\mathbf{x}))$ to determine both ξ_i and \mathbf{x} .
- Does this work? Theoretically it converges in some sense, and in practice: sort of, when it makes sense to use it.
- All of this FL seems a bit wacky... but,
- It is one of the leading edge research domain in cooperative ML. It was initiated by Google, and it widely used in practice (Owkin, IBM, ...)

What have we learned?

- Federated learning is an approach to cooperative learning (with optimization within). In contrast to pure distributed optimization, FL has challenges related to how the local functions are generated and what they mean.

What have we learned?

- Federated learning is an approach to cooperative learning (with optimization within). In contrast to pure distributed optimization, FL has challenges related to how the local functions are generated and what they mean.
- A challenge is related to non-IID data, which is still an open problem, but some algorithms can deal with it, e.g., SCAFFOLD.

What have we learned?

- Federated learning is an approach to cooperative learning (with optimization within). In contrast to pure distributed optimization, FL has challenges related to how the local functions are generated and what they mean.
- A challenge is related to non-IID data, which is still an open problem, but some algorithms can deal with it, e.g., SCAFFOLD.
- SCAFFOLD uses a variant of gradient tracking / SAGA to lessen the non-IID issue.

What have we learned?

- Federated learning is an approach to cooperative learning (with optimization within). In contrast to pure distributed optimization, FL has challenges related to how the local functions are generated and what they mean.
- A challenge is related to non-IID data, which is still an open problem, but some algorithms can deal with it, e.g., SCAFFOLD.
- SCAFFOLD uses a variant of gradient tracking / SAGA to lessen the non-IID issue.
- All FL algorithms require a mixing step, which is often computing the mean. This can be done in a cloud setting, or in a peer-to-peer setting. This is similar to distributed optimization and it has the same limitation (communication, ...)

What have we learned?

- Federated learning is an approach to cooperative learning (with optimization within). In contrast to pure distributed optimization, FL has challenges related to how the local functions are generated and what they mean.
- A challenge is related to non-IID data, which is still an open problem, but some algorithms can deal with it, e.g., SCAFFOLD.
- SCAFFOLD uses a variant of gradient tracking / SAGA to lessen the non-IID issue.
- All FL algorithms require a mixing step, which is often computing the mean. This can be done in a cloud setting, or in a peer-to-peer setting. This is similar to distributed optimization and it has the same limitation (communication, ...)
- Finally, if the local models are very different, sometimes we can decouple them from the global model via personalization.

Sample references

Federated learning:

- ① *Many, many authors*, **Advances and Open Problems in Federated Learning**, arXiv:1912.04977, 2019
- ② Many references in the above to SCAFFOLD and personalization

Class 5

Privacy

- Today we look at privacy issues in cooperative optimization. It goes without saying that maintaining private data private is key in many of the applications we have seen. And in practice.

Privacy

- Today we look at privacy issues in cooperative optimization. It goes without saying that maintaining private data private is key in many of the applications we have seen. And in practice.
- What's private? Sometimes the local cost functions f_i , that is the data associated to each agent. Sometimes the local features or models \mathbf{x}^i . Sometimes something else.

Privacy

- Today we look at privacy issues in cooperative optimization. It goes without saying that maintaining private data private is key in many of the applications we have seen. And in practice.
- What's private? Sometimes the local cost functions f_i , that is the data associated to each agent. Sometimes the local features or models \mathbf{x}^i . Sometimes something else.
- How do we ensure privacy, and what is privacy anyway?

Privacy

- Today we look at privacy issues in cooperative optimization. It goes without saying that maintaining private data private is key in many of the applications we have seen. And in practice.
- What's private? Sometimes the local cost functions f_i , that is the data associated to each agent. Sometimes the local features or models \mathbf{x}^i . Sometimes something else.
- How do we ensure privacy, and what is privacy anyway?
- We will look at three different notions of privacy.

1. Anonymization

- You may have heard about it: you may render the data private by stripping some sensitive data, like names.

1. Anonymization

- You may have heard about it: you may render the data private by stripping some sensitive data, like names.
- Often this type of procedure does not work: you can cross reference your information with publicly available information to get the stripped information back!

1. Anonymization

- You may have heard about it: you may render the data private by stripping some sensitive data, like names.
- Often this type of procedure does not work: you can cross reference your information with publicly available information to get the stripped information back!
- This is well-known, and scary!

1. Anonymization

- You may have heard about it: you may render the data private by stripping some sensitive data, like names.
- Often this type of procedure does not work: you can cross reference your information with publicly available information to get the stripped information back!
- This is well-known, and scary!
- What does it mean for optimization? **Distributed optimization is not per se privacy-preserving.**

Distributed optimization

- You may read that distributed optimization help node preserve their privacy, since they don't have to disclose f_i , but only quantities like \mathbf{x}^i , or Lagrangian multipliers. This is not strictly true.

Distributed optimization

- You may read that distributed optimization help node preserve their privacy, since they don't have to disclose f_i , but only quantities like \mathbf{x}^i , or Lagrangian multipliers. This is not strictly true.
- Imagine you have two agents, and you aim at solving,

$$\min_{\mathbf{x} \in \mathbb{R}^n} \sum_{i=1}^2 \left[f_i(\mathbf{x}) := \frac{1}{2} \|A_i \mathbf{x} - \mathbf{y}_i\|^2 \right].$$

Distributed optimization

- You may read that distributed optimization help node preserve their privacy, since they don't have to disclose f_i , but only quantities like \mathbf{x}^i , or Lagrangian multipliers. This is not strictly true.
- Imagine you have two agents, and you aim at solving,

$$\min_{\mathbf{x} \in \mathbb{R}^n} \sum_{i=1}^2 \left[f_i(\mathbf{x}) := \frac{1}{2} \|A_i \mathbf{x} - \mathbf{y}_i\|^2 \right].$$

- If you reach optimality, for instance via gradient tracking, then,

$$\nabla f_1(\mathbf{x}^*) + \nabla f_2(\mathbf{x}^*) = 0,$$

if you are agent 1, you also know then $\nabla f_2(\mathbf{x}^*)$.

Distributed optimization

- You may read that distributed optimization help node preserve their privacy, since they don't have to disclose f_i , but only quantities like \mathbf{x}^i , or Lagrangian multipliers. This is not strictly true.
- Imagine you have two agents, and you aim at solving,

$$\min_{\mathbf{x} \in \mathbb{R}^n} \sum_{i=1}^2 \left[f_i(\mathbf{x}) := \frac{1}{2} \|A_i \mathbf{x} - \mathbf{y}_i\|^2 \right].$$

- If you reach optimality, for instance via gradient tracking, then,

$$\nabla f_1(\mathbf{x}^*) + \nabla f_2(\mathbf{x}^*) = 0,$$

if you are agent 1, you also know then $\nabla f_2(\mathbf{x}^*)$.

- And if you have side information on the fact that agent 2 is training a similar model than you, i.e., $A_1 \approx A_2$,

$$\nabla f_2(\mathbf{x}^*) \approx A_1^\top (A_1 \mathbf{x}^* - \mathbf{y}_2),$$

so you can infer the data \mathbf{y}_2 .

Distributed optimization

- This also works in dual decomposition, since at each step we solve,

$$\mathbf{x}^i(\lambda_k) = \arg \min_{\mathbf{x}} \{f_i(\mathbf{x}) \pm \lambda_k^\top \mathbf{x}\} \iff \nabla f_i(\mathbf{x}^i) \pm \lambda_k = 0$$

Distributed optimization

- This also works in dual decomposition, since at each step we solve,

$$\mathbf{x}^i(\lambda_k) = \arg \min_{\mathbf{x}} \{f_i(\mathbf{x}) \pm \lambda_k^\top \mathbf{x}\} \iff \nabla f_i(\mathbf{x}^i) \pm \lambda_k = 0$$

- Since at each step you share \mathbf{x}^i with your neighbor and you know λ_k , then you can infer $\nabla f_i(\mathbf{x}^i)$! And with side information \mathbf{y}_i : for example for agent 1,

$$\nabla f_2(\mathbf{x}^2) - \lambda_k = 0 \implies A_2^\top (A_2 \mathbf{x}^2 - \mathbf{y}_2) = +\lambda_k \approx A_1^\top (A_1 \mathbf{x}^2 - \mathbf{y}_2)$$

Distributed optimization

- This also works in dual decomposition, since at each step we solve,

$$\mathbf{x}^i(\lambda_k) = \arg \min_{\mathbf{x}} \{f_i(\mathbf{x}) \pm \lambda_k^\top \mathbf{x}\} \iff \nabla f_i(\mathbf{x}^i) \pm \lambda_k = 0$$

- Since at each step you share \mathbf{x}^i with your neighbor and you know λ_k , then you can infer $\nabla f_i(\mathbf{x}^i)$! And with side information \mathbf{y}_i : for example for agent 1,

$$\nabla f_2(\mathbf{x}^2) - \lambda_k = 0 \implies A_2^\top (A_2 \mathbf{x}^2 - \mathbf{y}_2) = +\lambda_k \approx A_1^\top (A_1 \mathbf{x}^2 - \mathbf{y}_2)$$

- So, simple distributed optimization (\approx anonymization) doesn't work.

2. Encryption

- You may have heard about it: you may render the data private by encrypting it, so that only yourself can read it, **but others can use it to do their computations.**

2. Encryption

- You may have heard about it: you may render the data private by encrypting it, so that only yourself can read it, **but others can use it to do their computations.**
- This may work. If we apply an encryption function $\varphi(x)$, and we endow this encryption with rules, such that we can do simple sums and multiplications in the encrypted domain, as if we were doing them in the non-encrypted domain, we are in business!

2. Encryption

- You may have heard about it: you may render the data private by encrypting it, so that only yourself can read it, **but others can use it to do their computations.**
- This may work. If we apply an encryption function $\varphi(x)$, and we endow this encryption with rules, such that we can do simple sums and multiplications in the encrypted domain, as if we were doing them in the non-encrypted domain, we are in business!
- Define such a function $x \mapsto \varphi(x)$, such that,

$$\varphi(x_1) \circ \varphi(x_2) = x_1 \circ x_2$$

for a given set of operations \circ (e.g., multiplications, additions)

2. Encryption

- You may have heard about it: you may render the data private by encrypting it, so that only yourself can read it, **but others can use it to do their computations.**
- This may work. If we apply an encryption function $\varphi(x)$, and we endow this encryption with rules, such that we can do simple sums and multiplications in the encrypted domain, as if we were doing them in the non-encrypted domain, we are in business!
- Define such a function $x \mapsto \varphi(x)$, such that,

$$\varphi(x_1) \circ \varphi(x_2) = x_1 \circ x_2$$

for a given set of operations \circ (e.g., multiplications, additions)

- Then we could do: Encode \rightarrow Operations \rightarrow Decode

2. Encryption

- You may have heard about it: you may render the data private by encrypting it, so that only yourself can read it, **but others can use it to do their computations.**
- This may work. If we apply an encryption function $\varphi(x)$, and we endow this encryption with rules, such that we can do simple sums and multiplications in the encrypted domain, as if we were doing them in the non-encrypted domain, we are in business!
- Define such a function $x \mapsto \varphi(x)$, such that,

$$\varphi(x_1) \circ \varphi(x_2) = x_1 \circ x_2$$

for a given set of operations \circ (e.g., multiplications, additions)

- Then we could do: Encode \rightarrow Operations \rightarrow Decode
- This works in practice: **Homomorphic encryption** (with several standards for different operations)

2. Encryption

- You may have heard about it: you may render the data private by encrypting it, so that only yourself can read it, **but others can use it to do their computations.**
- This may work. If we apply an encryption function $\varphi(x)$, and we endow this encryption with rules, such that we can do simple sums and multiplications in the encrypted domain, as if we were doing them in the non-encrypted domain, we are in business!
- Define such a function $x \mapsto \varphi(x)$, such that,

$$\varphi(x_1) \circ \varphi(x_2) = x_1 \circ x_2$$

for a given set of operations \circ (e.g., multiplications, additions)

- Then we could do: Encode \rightarrow Operations \rightarrow Decode
- This works in practice: **Homomorphic encryption** (with several standards for different operations)
- But it can be slow! The encryption makes vectors and matrices grow in size: think at the RSA encryption for your credit card data.

3. Forget about it

- The method that is used the most nowadays is changing the notion of privacy, aka **differential privacy** (2006).

3. Forget about it

- The method that is used the most nowadays is changing the notion of privacy, aka **differential privacy** (2006).
- The idea is you can disclose your private data, provided you add enough noise that you make your data look like everybody else's data.

3. Forget about it

- The method that is used the most nowadays is changing the notion of privacy, aka **differential privacy** (2006).
- The idea is you can disclose your private data, provided you add enough noise that you make your data look like everybody else's data.
- So you are private w.r.t. others, nobody can distinguish it is you.

3. Forget about it

- The method that is used the most nowadays is changing the notion of privacy, aka **differential privacy** (2006).
- The idea is you can disclose your private data, provided you add enough noise that you make your data look like everybody else's data.
- So you are private w.r.t. others, nobody can distinguish it is you.
- How this can even work? Think about means and averages..

3. Forget about it

- The method that is used the most nowadays is changing the notion of privacy, aka **differential privacy** (2006).
- The idea is you can disclose your private data, provided you add enough noise that you make your data look like everybody else's data.
- So you are private w.r.t. others, nobody can distinguish it is you.
- How this can even work? Think about means and averages..
- If you start with N agents and you want to know the mean of their age, you ask them, and they reply with age plus a zero-mean random noise. If the noise is big enough, you cannot distinguish who's who, but if N is big enough, the mean will be computed accurately!

$$\text{result} = \frac{1}{N} \sum_{i=1}^N [\text{age}_i + r_i] \approx \mathbf{E}[\text{age} + r] = \overline{\text{age}}$$

3. Differential privacy

- If you start with N agents and you want to know the mean of their age, you ask them, and they reply with age plus a zero-mean random noise. If the noise is big enough, you cannot distinguish who's who, but if N is big enough, the mean will be computed accurately!

$$\text{result} = \frac{1}{N} \sum_{i=1}^N [\text{age}_i + r_i] \approx \mathbf{E}[\text{age} + r] = \overline{\text{age}}$$

3. Differential privacy

- If you start with N agents and you want to know the mean of their age, you ask them, and they reply with age plus a zero-mean random noise. If the noise is big enough, you cannot distinguish who's who, but if N is big enough, the mean will be computed accurately!

$$\text{result} = \frac{1}{N} \sum_{i=1}^N [\text{age}_i + r_i] \approx \mathbf{E}[\text{age} + r] = \overline{\text{age}}$$

- Wow, this is cool! Since distributed optimization and federated learning are mostly about computing averages, we may have found a good direction.

3. Differential privacy

- If you start with N agents and you want to know the mean of their age, you ask them, and they reply with age plus a zero-mean random noise. If the noise is big enough, you cannot distinguish who's who, but if N is big enough, the mean will be computed accurately!

$$\text{result} = \frac{1}{N} \sum_{i=1}^N [\text{age}_i + r_i] \approx \mathbf{E}[\text{age} + r] = \overline{\text{age}}$$

- Wow, this is cool! Since distributed optimization and federated learning are mostly about computing averages, we may have found a good direction.
- However, we will need to be careful. As for consensus, we compute the means (or mixing) many times, remember:

$$\mathbf{y}_{k+1} = \mathbf{W} \mathbf{y}_k, \quad k = 1, \dots, K$$

and the iterative process is the hard part.

3. Differential privacy

- Think about asking the age over and over, and each time k the agents reply with a different noise r_i , then,

$$\text{result}_k = \frac{1}{N} \sum_{i=1}^N [\text{age}_i + r_{i,k}] \approx \mathbf{E}_i[\text{age} + r_k] = \overline{\text{age}},$$

but also,

$$\frac{1}{K} \sum_{k=1}^K [\text{age}_i + r_{i,k}] \approx \text{age}_i + \mathbf{E}_k[r_k] = \text{age}_i.$$

Since you may compute an average across the iterations, and the noise may disappear there too, and you can reveal the age of a single agent.

3. Differential privacy

- Think about asking the age over and over, and each time k the agents reply with a different noise r_i , then,

$$\text{result}_k = \frac{1}{N} \sum_{i=1}^N [\text{age}_i + r_{i,k}] \approx \mathbf{E}_i[\text{age} + r_k] = \overline{\text{age}},$$

but also,

$$\frac{1}{K} \sum_{k=1}^K [\text{age}_i + r_{i,k}] \approx \text{age}_i + \mathbf{E}_k[r_k] = \text{age}_i.$$

Since you may compute an average across the iterations, and the noise may disappear there too, and you can reveal the age of a single agent.

- So, we need to study the noise process within the algorithmic framework a bit carefully. Typically, there is going to be a trade-off between noise level, N , and K .

Differential privacy: definition

- The actual definition is rather complicated, but let's build the intuition. We have two datasets \mathcal{D}_1 and \mathcal{D}_2 , that are equivalent in all but one data.

Differential privacy: definition

- The actual definition is rather complicated, but let's build the intuition. We have two datasets \mathcal{D}_1 and \mathcal{D}_2 , that are equivalent in all but one data.
- The idea here is ask any queries to the databases and see if we can distinguish them. If we can, then the one data difference can be identified and (possibly) the data itself can be revealed.

Differential privacy: definition

- The actual definition is rather complicated, but let's build the intuition. We have two datasets \mathcal{D}_1 and \mathcal{D}_2 , that are equivalent in all but one data.
- The idea here is ask any queries to the databases and see if we can distinguish them. If we can, then the one data difference can be identified and (possibly) the data itself can be revealed.
- The formal definition of ϵ -DP is that,

$$P(\mathcal{D}_1|Q) \leq e^\epsilon P(\mathcal{D}_2|Q)$$

for any query Q .

Differential privacy: definition

- The actual definition is rather complicated, but let's build the intuition. We have two datasets \mathcal{D}_1 and \mathcal{D}_2 , that are equivalent in all but one data.
- The idea here is ask any queries to the databases and see if we can distinguish them. If we can, then the one data difference can be identified and (possibly) the data itself can be revealed.
- The formal definition of ϵ -DP is that,

$$P(\mathcal{D}_1|Q) \leq e^\epsilon P(\mathcal{D}_2|Q)$$

for any query Q .

- And therefore, by symmetry,

$$P(\mathcal{D}_1|Q) \geq e^{-\epsilon} P(\mathcal{D}_2|Q)$$

Differential privacy: definition

- The actual definition is rather complicated, but let's build the intuition. We have two datasets \mathcal{D}_1 and \mathcal{D}_2 , that are equivalent in all but one data.
- The idea here is ask any queries to the databases and see if we can distinguish them. If we can, then the one data difference can be identified and (possibly) the data itself can be revealed.
- The formal definition of ϵ -DP is that,

$$P(\mathcal{D}_1|Q) \leq e^\epsilon P(\mathcal{D}_2|Q)$$

for any query Q .

- And therefore, by symmetry,

$$P(\mathcal{D}_1|Q) \geq e^{-\epsilon} P(\mathcal{D}_2|Q)$$

- So \mathcal{D}_1 and \mathcal{D}_2 are almost undistinguishable up to multiplicative scalars.

Differential privacy: Laplacian noise

- Why the definition is so complicated? It is due to the **Laplacian mechanism**.

Differential privacy: Laplacian noise

- Why the definition is so complicated? It is due to the **Laplacian mechanism**.
- Recall. The Laplacian noise distribution of mean 0 with scale b is

$$\text{Lap}(x|b) = \frac{1}{2b} e^{-\frac{|x|}{b}}.$$

Differential privacy: Laplacian noise

- Why the definition is so complicated? It is due to the **Laplacian mechanism**.
- Recall. The Laplacian noise distribution of mean 0 with scale b is

$$\text{Lap}(x|b) = \frac{1}{2b} e^{-\frac{|x|}{b}}.$$

- Consider now the age example and ask the question: **what is the uncertainty in the response that we must introduce in order to hide the participation of a single individual?**

Differential privacy: Laplacian noise

- Why the definition is so complicated? It is due to the **Laplacian mechanism**.
- Recall. The Laplacian noise distribution of mean 0 with scale b is

$$\text{Lap}(x|b) = \frac{1}{2b} e^{-\frac{|x|}{b}}.$$

- Consider now the age example and ask the question: **what is the uncertainty in the response that we must introduce in order to hide the participation of a single individual?**
- Imagine that is the first individual that is different across the two databased. Introduce the sensitivity

$$|\text{age}_1^1 - \text{age}_1^2| \leq \Delta f$$

for age of i individual and database i .

Differential privacy: Laplacian noise

- Why the definition is so complicated? It is due to the **Laplacian mechanism**.
- Recall. The Laplacian noise distribution of mean 0 with scale b is

$$\text{Lap}(x|b) = \frac{1}{2b} e^{-\frac{|x|}{b}}.$$

- Consider now the age example and ask the question: **what is the uncertainty in the response that we must introduce in order to hide the participation of a single individual?**
- Imagine that is the first individual that is different across the two databased. Introduce the sensitivity

$$|\text{age}_1^1 - \text{age}_1^2| \leq \Delta f$$

for age of $_1$ individual and database i .

- we will formalize the uncertainty as: the sensitivity of a function gives an upper bound on how much we must perturb its output to preserve privacy. One noise distribution naturally lends itself to differential privacy.

Differential privacy: Laplacian noise

- Why the definition is so complicated? It is due to the **Laplacian mechanism**.
- Recall. The Laplacian noise distribution of mean 0 with scale b is

$$\text{Lap}(x|b) = \frac{1}{2b} e^{-\frac{|x|}{b}}.$$

- Consider now the age example and ask the question: **what is the uncertainty in the response that we must introduce in order to hide the participation of a single individual?**
- Imagine that is the first individual that is different across the two databased. Introduce the sensitivity

$$|\text{age}_1^1 - \text{age}_1^2| \leq \Delta f$$

for age of $_1$ individual and database i .

- we will formalize the uncertainty as: the sensitivity of a function gives an upper bound on how much we must perturb its output to preserve privacy. One noise distribution naturally lends itself to differential privacy.
- Now compute ϵ -DP when adding Laplacian noise

Differential privacy: Laplacian noise

- We add noise to the respective ages age_1^1 and age_1^2 and the resulting noisy versions are x_1 and x_2 .

Differential privacy: Laplacian noise

- We add noise to the respective ages age_1^1 and age_1^2 and the resulting noisy versions are x_1 and x_2 .
- Now we look at the noise distribution, when asking if the age is z to both databases,

Differential privacy: Laplacian noise

- We add noise to the respective ages age_1^1 and age_1^2 and the resulting noisy versions are x_1 and x_2 .
- Now we look at the noise distribution, when asking if the age is z to both databases,
- By definition,

$$\frac{P(\mathcal{D}_1|z)}{P(\mathcal{D}_2|z)} = \frac{\exp(-|x_1 - z|/b)}{\exp(-|x_2 - z|/b)} = \exp([|x_2 - z| - |x_1 - z|]/b) \leq \exp\left(\frac{\Delta f}{b}\right)$$

Differential privacy: Laplacian noise

- We add noise to the respective ages age_1^1 and age_1^2 and the resulting noisy versions are x_1 and x_2 .
- Now we look at the noise distribution, when asking if the age is z to both databases,
- By definition,

$$\frac{P(\mathcal{D}_1|z)}{P(\mathcal{D}_2|z)} = \frac{\exp(-|x_1 - z|/b)}{\exp(-|x_2 - z|/b)} = \exp([|x_2 - z| - |x_1 - z|]/b) \leq \exp\left(\frac{\Delta f}{b}\right)$$

- If you then set the noise variance $b = \Delta f / \epsilon$, then the Laplacian mechanism is ϵ -DP!

Differential privacy: Laplacian noise

- We add noise to the respective ages age_1^1 and age_1^2 and the resulting noisy versions are x_1 and x_2 .
- Now we look at the noise distribution, when asking if the age is z to both databases,
- By definition,

$$\frac{P(\mathcal{D}_1|z)}{P(\mathcal{D}_2|z)} = \frac{\exp(-|x_1 - z|/b)}{\exp(-|x_2 - z|/b)} = \exp([|x_2 - z| - |x_1 - z|]/b) \leq \exp\left(\frac{\Delta f}{b}\right)$$

- If you then set the noise variance $b = \Delta f / \epsilon$, then the Laplacian mechanism is ϵ -DP!
- I.e., you cannot distinguish one individual contribution as long as it doesn't change the database of more than Δf .

Differential privacy: Laplacian noise

- We add noise to the respective ages age_1^1 and age_1^2 and the resulting noisy versions are x_1 and x_2 .
- Now we look at the noise distribution, when asking if the age is z to both databases,
- By definition,

$$\frac{P(\mathcal{D}_1|z)}{P(\mathcal{D}_2|z)} = \frac{\exp(-|x_1 - z|/b)}{\exp(-|x_2 - z|/b)} = \exp((|x_2 - z| - |x_1 - z|)/b) \leq \exp\left(\frac{\Delta f}{b}\right)$$

- If you then set the noise variance $b = \Delta f / \epsilon$, then the Laplacian mechanism is ϵ -DP!
- I.e., you cannot distinguish one individual contribution as long as it doesn't change the database of more than Δf .
- This is the basic idea which is applied in real scenarios and databases, and in many flavors.

DGD with Laplacian noise

- Can we do privacy in distributed optimization, and if so, what are we protecting?

DGD with Laplacian noise

- Can we do privacy in distributed optimization, and if so, what are we protecting?
- **Adding noise, we can always do, proving convergence it is another story**, and: what are we protecting? The single const functions (i.e., the local data).

DGD with Laplacian noise

- Can we do privacy in distributed optimization, and if so, what are we protecting?
- **Adding noise, we can always do, proving convergence it is another story**, and: what are we protecting? The single cost functions (i.e., the local data).
- In particular, we have to make sure that any algorithm operating on two sets of local cost functions equal but by one of them, will be undistinguishable in the DP sense.

DGD with Laplacian noise

- Can we do privacy in distributed optimization, and if so, what are we protecting?
- **Adding noise, we can always do, proving convergence it is another story**, and: what are we protecting? The single cost functions (i.e., the local data).
- In particular, we have to make sure that any algorithm operating on two sets of local cost functions equal but by one of them, will be undistinguishable in the DP sense.
- This is can very involved in theory, but in practice it is easy to put together:

DGD with Laplacian noise

- Can we do privacy in distributed optimization, and if so, what are we protecting?
- **Adding noise, we can always do, proving convergence it is another story**, and: what are we protecting? The single cost functions (i.e., the local data).
- In particular, we have to make sure that any algorithm operating on two sets of local cost functions equal but by one of them, will be undistinguishable in the DP sense.
- This is can very involved in theory, but in practice it is easy to put together:
- DGD with Laplacian Noise (DGD-DP):

DGD with Laplacian noise

- Can we do privacy in distributed optimization, and if so, what are we protecting?
- **Adding noise, we can always do, proving convergence it is another story**, and: what are we protecting? The single const functions (i.e., the local data).
- In particular, we have to make sure that any algorithm operating on two sets of local cost functions equal but by one of them, will be undistinguishable in the DP sense.
- This is can very involved in theory, but in practice it is easy to put together:
- DGD with Laplacian Noise (DGD-DP):
 - ▶ Start initializing at random \mathbf{x}_k^i , then,

DGD with Laplacian noise

- Can we do privacy in distributed optimization, and if so, what are we protecting?
- **Adding noise, we can always do, proving convergence it is another story**, and: what are we protecting? The single const functions (i.e., the local data).
- In particular, we have to make sure that any algorithm operating on two sets of local cost functions equal but by one of them, will be undistinguishable in the DP sense.
- This is can very involved in theory, but in practice it is easy to put together:
- DGD with Laplacian Noise (DGD-DP):
 - ▶ Start initializing at random \mathbf{x}_k^i , then,
 - ▶ Add a Laplacian noise $\mathbf{x}_k^i + \zeta_k^i$, and send the obscured state to your neighbors

DGD with Laplacian noise

- Can we do privacy in distributed optimization, and if so, what are we protecting?
- **Adding noise, we can always do, proving convergence it is another story**, and: what are we protecting? The single const functions (i.e., the local data).
- In particular, we have to make sure that any algorithm operating on two sets of local cost functions equal but by one of them, will be undistinguishable in the DP sense.
- This is can very involved in theory, but in practice it is easy to put together:
- **DGD with Laplacian Noise (DGD-DP):**
 - ▶ Start initializing at random \mathbf{x}_k^i , then,
 - ▶ Add a Laplacian noise $\mathbf{x}_k^i + \zeta_k^i$, and send the obscured state to your neighbors
 - ▶ Update the state

$$\mathbf{x}_{k+1}^i = \mathbf{x}_k^i + \sum_{j \in N_i, j \neq i} \gamma_k \widehat{w}_{ij} (\mathbf{x}_k^j + \zeta_k^j - \mathbf{x}_k^i) - \alpha_k \nabla f_i(\mathbf{x}_k^i)$$

DGD with Laplacian noise

- Can we do privacy in distributed optimization, and if so, what are we protecting?
- **Adding noise, we can always do, proving convergence it is another story**, and: what are we protecting? The single const functions (i.e., the local data).
- In particular, we have to make sure that any algorithm operating on two sets of local cost functions equal but by one of them, will be undistinguishable in the DP sense.
- This is can very involved in theory, but in practice it is easy to put together:
- DGD with Laplacian Noise (DGD-DP):
 - ▶ Start initializing at random \mathbf{x}_k^i , then,
 - ▶ Add a Laplacian noise $\mathbf{x}_k^i + \zeta_k^i$, and send the obscured state to your neighbors
 - ▶ Update the state

$$\mathbf{x}_{k+1}^i = \mathbf{x}_k^i + \sum_{j \in N_i, j \neq i} \gamma_k \widehat{w}_{ij} (\mathbf{x}_k^j + \zeta_k^j - \mathbf{x}_k^i) - \alpha_k \nabla f_i(\mathbf{x}_k^i)$$

- $\{\gamma_k\} \rightarrow 0$, and $\{\alpha_k\}$ is the stepsize. This is DGD with some weighting to limit the noise and variable stepsize.

Guarantees

- Consider $\hat{w}_{ij} = \hat{w}_{ji} \geq 0$ for $i \neq j$, and $\hat{w}_{ii} = -\sum_{j \neq i} \hat{w}_{ij}$. Define the matrix $\widehat{W} = [\hat{w}_{ij}]$, which is symmetric and $\widehat{W}\mathbf{1} = \mathbf{0}$. Assume that $W = \widehat{W} + I$ is doubly stochastic, then,

Guarantees

- Consider $\widehat{w}_{ij} = \widehat{w}_{ji} \geq 0$ for $i \neq j$, and $\widehat{w}_{ii} = -\sum_{j \neq i} \widehat{w}_{ij}$. Define the matrix $\widehat{W} = [\widehat{w}_{ij}]$, which is symmetric and $\widehat{W}\mathbf{1} = \mathbf{0}$. Assume that $W = \widehat{W} + I$ is doubly stochastic, then,
- The update:

$$\mathbf{x}_{k+1}^i = \mathbf{x}_k^i + \sum_{j \in N_i, j \neq i} \gamma_k \widehat{w}_{ij} (\mathbf{x}_k^j + \zeta_k^j - \mathbf{x}_k^i) - \alpha_k \nabla f_i(\mathbf{x}_k^i)$$

is the standard DGD for $W = \widehat{W} + I$ doubly stochastic, no noise, and $\gamma_k = 1$, since we can write,

$$\mathbf{y}_{k+1} = \mathbf{y}_k + \widehat{W}\mathbf{y}_k - \alpha_k \nabla F(\mathbf{y}_k) = W\mathbf{y}_k - \alpha_k \nabla F(\mathbf{y}_k).$$

Guarantees

-
- The update:

$$\mathbf{x}_{k+1}^i = \mathbf{x}_k^i + \sum_{j \in N_i, j \neq i} \gamma_k \widehat{w}_{ij} (\mathbf{x}_k^j + \zeta_k^j - \mathbf{x}_k^i) - \alpha_k \nabla f_i(\mathbf{x}_k^i)$$

is the standard DGD for $W = \widehat{W} + I$ doubly stochastic, no noise, and $\gamma_k = 1$, since we can write,

$$\mathbf{y}_{k+1} = \mathbf{y}_k + \widehat{W} \mathbf{y}_k - \alpha_k \nabla F(\mathbf{y}_k) = W \mathbf{y}_k - \alpha_k \nabla F(\mathbf{y}_k).$$

- Till recently, no one could prove optimality and privacy. The idea was to give an iteration budget not to pass a privacy budget. Something like,

$$\epsilon \leq \sum_{k=1}^T \frac{\alpha_k}{\nu_k},$$

where ν_k is the variance of the Laplacian distribution.

Guarantees

-
- The update:

$$\mathbf{x}_{k+1}^i = \mathbf{x}_k^i + \sum_{j \in N_i, j \neq i} \gamma_k \widehat{w}_{ij} (\mathbf{x}_k^j + \zeta_k^j - \mathbf{x}_k^i) - \alpha_k \nabla f_i(\mathbf{x}_k^i)$$

is the standard DGD for $W = \widehat{W} + I$ doubly stochastic, no noise, and $\gamma_k = 1$, since we can write,

$$\mathbf{y}_{k+1} = \mathbf{y}_k + \widehat{W} \mathbf{y}_k - \alpha_k \nabla F(\mathbf{y}_k) = W \mathbf{y}_k - \alpha_k \nabla F(\mathbf{y}_k).$$

- Till recently, no one could prove optimality and privacy. The idea was to give an iteration budget not to pass a privacy budget. Something like,

$$\epsilon \leq \sum_{k=1}^T \frac{\alpha_k}{\nu_k},$$

where ν_k is the variance of the Laplacian distribution.

- However, in 2022, we discovered that by increasing ν_k and diminishing γ_k faster, then optimality and privacy could be enforced together!

Convergence and privacy

Theorem 13 (Convergence)

Consider the convex optimization problem

$$\min_{\mathbf{x} \in \mathbb{R}^n} \sum_{i=1}^n f_i(\mathbf{x}),$$

and the DGD-DP algorithm to find one of its solutions \mathbf{x}^* in a ϵ -DP fashion.

Assume Lipschitz continuous gradients for f_i 's, $W = [w_{ij}]$ symmetric and $I + W$ doubly stochastic with $\|I + W - \frac{11^\top}{N}\| < 1$.

Assume that the noise is zero mean and with variance σ_k^i for which,

$$\sum_{k=0}^{\infty} \gamma_k^2 \max_i (\sigma_k^i)^2 < \infty.$$

Then, if $\sum_{k=0}^{\infty} \gamma_k = \infty$, $\sum_{k=0}^{\infty} \alpha_k = \infty$, and $\sum_{k=0}^{\infty} (\alpha_k^2)/\gamma_k < \infty$, we have

$$\|\mathbf{x}_k^i - \mathbf{x}^*\| \rightarrow 0 \quad \text{almost surely } \forall i.$$

Convergence and privacy

Theorem 14 (Privacy)

Furthermore, assume bounded sensitivity $\|\nabla f_i(x)\|_1 \leq C$.

Let

$$\bar{\epsilon} = \sum_{k=1}^T \frac{2C\alpha_k}{\nu_k}, \quad \nu_k = (\sigma_k^i)^2.$$

Then DGD-DP is $\epsilon \leq \bar{\epsilon}$ -DP up to iteration T .

Moreover if $\sum_{k=1}^T \frac{\alpha_k}{\nu_k} < \infty$, then DGD-DP is $\epsilon \leq \bar{\epsilon}$ -DP for all T 's.

Convergence and privacy

Theorem 14 (Privacy)

Furthermore, assume bounded sensitivity $\|\nabla f_i(x)\|_1 \leq C$.

Let

$$\bar{\epsilon} = \sum_{k=1}^T \frac{2C\alpha_k}{\nu_k}, \quad \nu_k = (\sigma_k^i)^2.$$

Then DGD-DP is $\epsilon \leq \bar{\epsilon}$ -DP up to iteration T .

Moreover if $\sum_{k=1}^T \frac{\alpha_k}{\nu_k} < \infty$, then DGD-DP is $\epsilon \leq \bar{\epsilon}$ -DP for all T 's.

Example: $\alpha_k = O(1/k)$, $\gamma_k = O(1/k^{0.9})$, $\nu_k \leq O(k^{0.3}/\epsilon)$.

Convergence and privacy

Theorem 14 (Privacy)

Furthermore, assume bounded sensitivity $\|\nabla f_i(x)\|_1 \leq C$.

Let

$$\bar{\epsilon} = \sum_{k=1}^T \frac{2C\alpha_k}{\nu_k}, \quad \nu_k = (\sigma_k^i)^2.$$

Then DGD-DP is $\epsilon \leq \bar{\epsilon}$ -DP up to iteration T .

Moreover if $\sum_{k=1}^T \frac{\alpha_k}{\nu_k} < \infty$, then DGD-DP is $\epsilon \leq \bar{\epsilon}$ -DP for all T 's.

Example: $\alpha_k = O(1/k)$, $\gamma_k = O(1/k^{0.9})$, $\nu_k \leq O(k^{0.3}/\epsilon)$.

Proof. See [arXiv: 2202.01113]



Convergence and privacy

Theorem 14 (Privacy)

Furthermore, assume bounded sensitivity $\|\nabla f_i(x)\|_1 \leq C$.

Let

$$\bar{\epsilon} = \sum_{k=1}^T \frac{2C\alpha_k}{\nu_k}, \quad \nu_k = (\sigma_k^i)^2.$$

Then DGD-DP is $\epsilon \leq \bar{\epsilon}$ -DP up to iteration T .

Moreover if $\sum_{k=1}^T \frac{\alpha_k}{\nu_k} < \infty$, then DGD-DP is $\epsilon \leq \bar{\epsilon}$ -DP for all T 's.

Example: $\alpha_k = O(1/k)$, $\gamma_k = O(1/k^{0.9})$, $\nu_k \leq O(k^{0.3}/\epsilon)$.

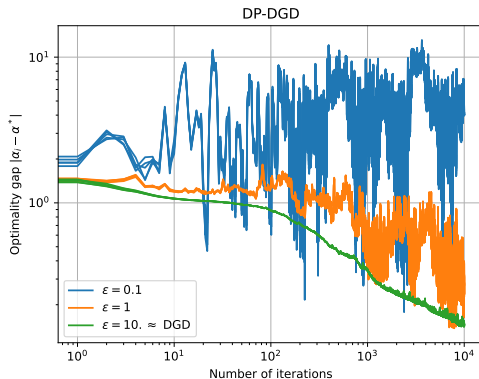
Proof. See [arXiv: 2202.01113]

□

Different papers will have different assumptions. For instance, the bounded sensitivity may be a little strong, you may relax that but then allow for errors or

..

Convergence and privacy: numerical results (kernel setting)



Parameters (and trade-offs):

$$\alpha_k = \frac{0.002}{1 + 0.001k}, \gamma_k = \frac{1}{1 + 0.001k^{0.9}}, \nu_k = \frac{0.01}{\epsilon} \frac{1}{1 + 0.001k^{0.1}}.$$

Gradient Tracking with Laplacian noise

- In [arXiv: 2202.01113], we can also find a gradient tracking version with Laplacian noise and similar results as before

Gradient Tracking with Laplacian noise

- In [arXiv: 2202.01113], we can also find a gradient tracking version with Laplacian noise and similar results as before
- I spare you the details.

Gradient Tracking with Laplacian noise

- In [arXiv: 2202.01113], we can also find a gradient tracking version with Laplacian noise and similar results as before
- I spare you the details.
- But, take-home: **Privacy is hard in distributed optimization.**
Optimization put hard requirements on what you can do, and adding noise makes you go very slowly to the optimizer. In this sense, DO is good for collaboration, not really for privacy.

Differentially Private Federated Learning

- There are many ways to do FL, there are many ways to do DP-FL!

Differentially Private Federated Learning

- There are many ways to do FL, there are many ways to do DP-FL!
- We see here one way to do it.

Differentially Private Federated Learning

- There are many ways to do FL, there are many ways to do DP-FL!
- We see here one way to do it.
- First thing first, classical ϵ -DP is a bit too restrictive in general, and we relax it to (ϵ, δ) -DP

$$P(\mathcal{D}_1|Q) \leq e^\epsilon P(\mathcal{D}_2|Q) + \delta$$

for any query Q .

Differentially Private Federated Learning

- There are many ways to do FL, there are many ways to do DP-FL!
- We see here one way to do it.
- First thing first, classical ϵ -DP is a bit too restrictive in general, and we relax it to (ϵ, δ) -DP

$$P(\mathcal{D}_1|Q) \leq e^\epsilon P(\mathcal{D}_2|Q) + \delta$$

for any query Q .

- Not such a big deal, but in this way we can also use other noise distributions and we are less in trouble when probability vanishes.

Differentially Private Federated Learning

- There are many ways to do FL, there are many ways to do DP-FL!
- We see here one way to do it.
- First thing first, classical ϵ -DP is a bit too restrictive in general, and we relax it to (ϵ, δ) -DP

$$P(\mathcal{D}_1|Q) \leq e^\epsilon P(\mathcal{D}_2|Q) + \delta$$

for any query Q .

- Not such a big deal, but in this way we can also use other noise distributions and we are less in trouble when probability vanishes.
- Then we don't extremely care if a data point is private or not, but here we care about privacy of the whole dataset of a client.

Differentially Private Federated Learning

- There are many ways to do FL, there are many ways to do DP-FL!
- We see here one way to do it.
- First thing first, classical ϵ -DP is a bit too restrictive in general, and we relax it to (ϵ, δ) -DP

$$P(\mathcal{D}_1|Q) \leq e^\epsilon P(\mathcal{D}_2|Q) + \delta$$

for any query Q .

- Not such a big deal, but in this way we can also use other noise distributions and we are less in trouble when probability vanishes.
- Then we don't extremely care if a data point is private or not, but here we care about privacy of the whole dataset of a client.
- The latter is encapsulated into the model parameters that have to be obfuscated

Gaussian Mechanism and an example

- We call Gaussian mechanism the act of adding a Gaussian distributed noise to our algorithms.

Gaussian Mechanism and an example

- We call Gaussian mechanism the act of adding a Gaussian distributed noise to our algorithms.
- Consider the example of estimating a real valued function $f : D \rightarrow \mathbf{R}$ with a differentially private Gaussian mechanism. Specifically, a GM adds Gaussian noise calibrated to the functions data set sensitivity S_f .

Gaussian Mechanism and an example

- We call Gaussian mechanism the act of adding a Gaussian distributed noise to our algorithms.
- Consider the example of estimating a real valued function $f : D \rightarrow \mathbf{R}$ with a differentially private Gaussian mechanism. Specifically, a GM adds Gaussian noise calibrated to the functions data set sensitivity S_f .
- This sensitivity is defined as the maximum of the absolute distance $\|f(d) - f(d')\|_2$, where d' and d are two adjacent inputs. A GM is then defined as $M(d) = f(d) + \mathcal{N}(0, \sigma^2 S_f^2)$.

Gaussian Mechanism and an example

- We call Gaussian mechanism the act of adding a Gaussian distributed noise to our algorithms.
- Consider the example of estimating a real valued function $f : D \rightarrow \mathbf{R}$ with a differentially private Gaussian mechanism. Specifically, a GM adds Gaussian noise calibrated to the functions data set sensitivity S_f .
- This sensitivity is defined as the maximum of the absolute distance $\|f(d) - f(d')\|_2$, where d' and d are two adjacent inputs. A GM is then defined as $M(d) = f(d) + \mathcal{N}(0, \sigma^2 S_f^2)$.
- Note the parallel with the Laplacian mechanism

Gaussian Mechanism and an example

- We call Gaussian mechanism the act of adding a Gaussian distributed noise to our algorithms.
- Consider the example of estimating a real valued function $f : D \rightarrow \mathbf{R}$ with a differentially private Gaussian mechanism. Specifically, a GM adds Gaussian noise calibrated to the functions data set sensitivity S_f .
- This sensitivity is defined as the maximum of the absolute distance $\|f(d) - f(d')\|_2$, where d' and d are two adjacent inputs. A GM is then defined as $M(d) = f(d) + \mathcal{N}(0, \sigma^2 S_f^2)$.
- Note the parallel with the Laplacian mechanism
- We can then bound the probability that (ϵ, δ) -DP is broken according to:

$$\delta \leq \frac{4}{5} \exp(-(\sigma\epsilon)^2/2).$$

Gaussian Mechanism and an example

- We call Gaussian mechanism the act of adding a Gaussian distributed noise to our algorithms.
- Consider the example of estimating a real valued function $f : D \rightarrow \mathbf{R}$ with a differentially private Gaussian mechanism. Specifically, a GM adds Gaussian noise calibrated to the functions data set sensitivity S_f .
- This sensitivity is defined as the maximum of the absolute distance $\|f(d) - f(d')\|_2$, where d' and d are two adjacent inputs. A GM is then defined as $M(d) = f(d) + \mathcal{N}(0, \sigma^2 S_f^2)$.
- Note the parallel with the Laplacian mechanism
- We can then bound the probability that (ϵ, δ) -DP is broken according to:

$$\delta \leq \frac{4}{5} \exp(-(\sigma\epsilon)^2/2).$$

- This means that, with a single query to the mechanism, δ needs to be $> \frac{4}{5} \exp(-(\sigma\epsilon)^2/2)$ to ensure (ϵ, δ) -DP.

Gaussian Mechanism and an example

- We call Gaussian mechanism the act of adding a Gaussian distributed noise to our algorithms.
- Consider the example of estimating a real valued function $f : D \rightarrow \mathbf{R}$ with a differentially private Gaussian mechanism. Specifically, a GM adds Gaussian noise calibrated to the functions data set sensitivity S_f .
- This sensitivity is defined as the maximum of the absolute distance $\|f(d) - f(d')\|_2$, where d' and d are two adjacent inputs. A GM is then defined as $M(d) = f(d) + \mathcal{N}(0, \sigma^2 S_f^2)$.
- Note the parallel with the Laplacian mechanism
- We can then bound the probability that (ϵ, δ) -DP is broken according to:

$$\delta \leq \frac{4}{5} \exp(-(\sigma\epsilon)^2/2).$$

- This means that, with a single query to the mechanism, δ needs to be $> \frac{4}{5} \exp(-(\sigma\epsilon)^2/2)$ to ensure (ϵ, δ) -DP.
- Careful δ is accumulative and grows for consecutive queries. Therefore, to protect privacy, an accountant keeps track of δ . Once a certain threshold for δ is reached, the GM shall not answer any new queries.

Gaussian Mechanism and an example

- We call Gaussian mechanism the act of adding a Gaussian distributed noise to our algorithms.
- Consider the example of estimating a real valued function $f : D \rightarrow \mathbf{R}$ with a differentially private Gaussian mechanism. Specifically, a GM adds Gaussian noise calibrated to the functions data set sensitivity S_f .
- This sensitivity is defined as the maximum of the absolute distance $\|f(d) - f(d')\|_2$, where d' and d are two adjacent inputs. A GM is then defined as $M(d) = f(d) + \mathcal{N}(0, \sigma^2 S_f^2)$.
- Note the parallel with the Laplacian mechanism
- We can then bound the probability that (ϵ, δ) -DP is broken according to:

$$\delta \leq \frac{4}{5} \exp(-(\sigma\epsilon)^2/2).$$

- This means that, with a single query to the mechanism, δ needs to be $> \frac{4}{5} \exp(-(\sigma\epsilon)^2/2)$ to ensure (ϵ, δ) -DP.
- Careful δ is accumulative and grows for consecutive queries. Therefore, to protect privacy, an accountant keeps track of δ . Once a certain threshold for δ is reached, the GM shall not answer any new queries.
- This is the same as in DO, or in the age example; iterative queries can reveal the dataset, if the noise stay at the same level.

DP FedAvg

- **DP-FedAvg**

DP FedAvg

- **DP-FedAvg**

- ▶ **Server update.** Initialize \mathbf{x}_0 . For each time $t = 1, \dots$ do:

DP FedAvg

- **DP-FedAvg**

- ▶ **Server update.** Initialize \mathbf{x}_0 . For each time $t = 1, \dots$ do:
 - ★ Select $C \leq P$ clients, and for each client

DP FedAvg

- **DP-FedAvg**

- ▶ **Server update.** Initialize \mathbf{x}_0 . For each time $t = 1, \dots$ do:
 - ★ Select $C \leq P$ clients, and for each client
 - ★ Send \mathbf{x}_t to client

DP FedAvg

- **DP-FedAvg**

- ▶ **Server update.** Initialize \mathbf{x}_0 . For each time $t = 1, \dots$ do:
 - ★ Select $C \leq P$ clients, and for each client
 - ★ Send \mathbf{x}_t to client
 - ★ Receive \mathbf{x}_{t+1}^i from client update

DP FedAvg

- **DP-FedAvg**

- ▶ **Server update.** Initialize \mathbf{x}_0 . For each time $t = 1, \dots$ do:
 - ★ Select $C \leq P$ clients, and for each client
 - ★ Send \mathbf{x}_t to client
 - ★ Receive \mathbf{x}_{t+1}^i from client update
 - ★ Mixing

$$\mathbf{x}_{t+1} = \sum_{c=1}^C \frac{N_c}{\sum_c N_c} \mathbf{x}_{t+1}^c$$

DP FedAvg

• DP-FedAvg

- ▶ **Server update.** Initialize \mathbf{x}_0 . For each time $t = 1, \dots$ do:

- ★ Select $C \leq P$ clients, and for each client
- ★ Send \mathbf{x}_t to client
- ★ Receive \mathbf{x}_{t+1}^i from client update
- ★ Mixing

$$\mathbf{x}_{t+1} = \sum_{c=1}^C \frac{N_c}{\sum_c N_c} \mathbf{x}_{t+1}^c$$

- ▶ **Client update.** Start from $\mathbf{x}_0^i = \mathbf{x}_t$. Define the sensitivity \mathcal{C} and the noise variance σ_g^2 as well as the scaling $\beta = 2\mathcal{C}/B$. Select batches \mathcal{B} of data, each containing B data points. For epoch $k = 1, \dots, E$

DP FedAvg

• DP-FedAvg

- ▶ **Server update.** Initialize \mathbf{x}_0 . For each time $t = 1, \dots$ do:

- ★ Select $C \leq P$ clients, and for each client
- ★ Send \mathbf{x}_t to client
- ★ Receive \mathbf{x}_{t+1}^i from client update
- ★ Mixing

$$\mathbf{x}_{t+1} = \sum_{c=1}^C \frac{N_c}{\sum_c N_c} \mathbf{x}_{t+1}^c$$

- ▶ **Client update.** Start from $\mathbf{x}_0^i = \mathbf{x}_t$. Define the sensitivity \mathcal{C} and the noise variance σ_g^2 as well as the scaling $\beta = 2\mathcal{C}/B$. Select batches \mathcal{B} of data, each containing B data points. For epoch $k = 1, \dots, E$
 - ★ Select a data batch \mathcal{B}_j and compute the local gradient $g_{ij} = \nabla f_{\mathcal{B}_j}(\mathbf{x}_k^i)$

DP FedAvg

• DP-FedAvg

- ▶ **Server update.** Initialize \mathbf{x}_0 . For each time $t = 1, \dots$ do:

- ★ Select $C \leq P$ clients, and for each client
- ★ Send \mathbf{x}_t to client
- ★ Receive \mathbf{x}_{t+1}^i from client update
- ★ Mixing

$$\mathbf{x}_{t+1} = \sum_{c=1}^C \frac{N_c}{\sum_c N_c} \mathbf{x}_{t+1}^c$$

- ▶ **Client update.** Start from $\mathbf{x}_0^i = \mathbf{x}_t$. Define the sensitivity \mathcal{C} and the noise variance σ_g^2 as well as the scaling $\beta = 2\mathcal{C}/B$. Select batches \mathcal{B} of data, each containing B data points. For epoch $k = 1, \dots, E$
 - ★ Select a data batch \mathcal{B}_j and compute the local gradient $g_{ij} = \nabla f_{\mathcal{B}_j}(\mathbf{x}_k^i)$
 - ★ Clip the local gradient: $\tilde{g}_{ij} = g_{ij} / \max\{1, \|g_{ij}\|_2 / \mathcal{C}\}$

DP FedAvg

• DP-FedAvg

- ▶ **Server update.** Initialize \mathbf{x}_0 . For each time $t = 1, \dots$ do:

- ★ Select $C \leq P$ clients, and for each client
- ★ Send \mathbf{x}_t to client
- ★ Receive \mathbf{x}_{t+1}^i from client update
- ★ Mixing

$$\mathbf{x}_{t+1} = \sum_{c=1}^C \frac{N_c}{\sum_c N_c} \mathbf{x}_{t+1}^c$$

- ▶ **Client update.** Start from $\mathbf{x}_0^i = \mathbf{x}_t$. Define the sensitivity \mathcal{C} and the noise variance σ_g^2 as well as the scaling $\beta = 2\mathcal{C}/B$. Select batches \mathcal{B} of data, each containing B data points. For epoch $k = 1, \dots, E$

- ★ Select a data batch \mathcal{B}_j and compute the local gradient $g_{ij} = \nabla f_{\mathcal{B}_j}(\mathbf{x}_k^i)$
- ★ Clip the local gradient: $\tilde{g}_{ij} = g_{ij} / \max\{1, \|g_{ij}\|_2 / \mathcal{C}\}$
- ★ Compute

$$\mathbf{x}_{k+1}^i = \mathbf{x}_k^i - \alpha_k(\tilde{g}_{ij} + \beta\eta), \quad \eta \sim \mathcal{N}(0, \sigma_g^2)$$

DP FedAvg

• DP-FedAvg

- ▶ **Server update.** Initialize \mathbf{x}_0 . For each time $t = 1, \dots$ do:

- ★ Select $C \leq P$ clients, and for each client
- ★ Send \mathbf{x}_t to client
- ★ Receive \mathbf{x}_{t+1}^i from client update
- ★ Mixing

$$\mathbf{x}_{t+1} = \sum_{c=1}^C \frac{N_c}{\sum_c N_c} \mathbf{x}_{t+1}^c$$

- ▶ **Client update.** Start from $\mathbf{x}_0^i = \mathbf{x}_t$. Define the sensitivity \mathcal{C} and the noise variance σ_g^2 as well as the scaling $\beta = 2\mathcal{C}/B$. Select batches \mathcal{B} of data, each containing B data points. For epoch $k = 1, \dots, E$

- ★ Select a data batch \mathcal{B}_j and compute the local gradient $g_{ij} = \nabla f_{\mathcal{B}_j}(\mathbf{x}_k^i)$
- ★ Clip the local gradient: $\tilde{g}_{ij} = g_{ij} / \max\{1, \|g_{ij}\|_2 / \mathcal{C}\}$
- ★ Compute

$$\mathbf{x}_{k+1}^i = \mathbf{x}_k^i - \alpha_k(\tilde{g}_{ij} + \beta\eta), \quad \eta \sim \mathcal{N}(0, \sigma_g^2)$$

- Many variants and trade-off exist (for instance, level of privacy vs. accuracy, etc..)

DP FedAvg

• DP-FedAvg

- ▶ **Server update.** Initialize \mathbf{x}_0 . For each time $t = 1, \dots$ do:

- ★ Select $C \leq P$ clients, and for each client
- ★ Send \mathbf{x}_t to client
- ★ Receive \mathbf{x}_{t+1}^i from client update
- ★ Mixing

$$\mathbf{x}_{t+1} = \sum_{c=1}^C \frac{N_c}{\sum_c N_c} \mathbf{x}_{t+1}^c$$

- ▶ **Client update.** Start from $\mathbf{x}_0^i = \mathbf{x}_t$. Define the sensitivity \mathcal{C} and the noise variance σ_g^2 as well as the scaling $\beta = 2\mathcal{C}/B$. Select batches \mathcal{B} of data, each containing B data points. For epoch $k = 1, \dots, E$

- ★ Select a data batch \mathcal{B}_j and compute the local gradient $g_{ij} = \nabla f_{\mathcal{B}_j}(\mathbf{x}_k^i)$
- ★ Clip the local gradient: $\tilde{g}_{ij} = g_{ij} / \max\{1, \|g_{ij}\|_2 / \mathcal{C}\}$
- ★ Compute

$$\mathbf{x}_{k+1}^i = \mathbf{x}_k^i - \alpha_k(\tilde{g}_{ij} + \beta\eta), \quad \eta \sim \mathcal{N}(0, \sigma_g^2)$$

- Many variants and trade-off exist (for instance, level of privacy vs. accuracy, etc..)
- This is a fast growing area of research

DP FedAvg

• DP-FedAvg

- ▶ **Server update.** Initialize \mathbf{x}_0 . For each time $t = 1, \dots$ do:

- ★ Select $C \leq P$ clients, and for each client
- ★ Send \mathbf{x}_t to client
- ★ Receive \mathbf{x}_{t+1}^i from client update
- ★ Mixing

$$\mathbf{x}_{t+1} = \sum_{c=1}^C \frac{N_c}{\sum_c N_c} \mathbf{x}_{t+1}^c$$

- ▶ **Client update.** Start from $\mathbf{x}_0^i = \mathbf{x}_t$. Define the sensitivity \mathcal{C} and the noise variance σ_g^2 as well as the scaling $\beta = 2\mathcal{C}/B$. Select batches \mathcal{B} of data, each containing B data points. For epoch $k = 1, \dots, E$

- ★ Select a data batch \mathcal{B}_j and compute the local gradient $g_{ij} = \nabla f_{\mathcal{B}_j}(\mathbf{x}_k^i)$
- ★ Clip the local gradient: $\tilde{g}_{ij} = g_{ij} / \max\{1, \|g_{ij}\|_2 / \mathcal{C}\}$
- ★ Compute

$$\mathbf{x}_{k+1}^i = \mathbf{x}_k^i - \alpha_k (\tilde{g}_{ij} + \beta \eta), \quad \eta \sim \mathcal{N}(0, \sigma_g^2)$$

- Many variants and trade-off exist (for instance, level of privacy vs. accuracy, etc..)
- This is a fast growing area of research
- Since here convergence may not be a goal, privacy is “easier” to impose than in DO.

Convergence of DP-FedAvg

Theorem 15

Consider the same setting as in FedAvg, with the addition of a clipping constant for which,

$$\|\nabla f_c(\mathbf{x})\| \leq \mathcal{C}, \quad \forall \mathbf{x},$$

and additional technical assumptions. Then DP-FedAvg can be made converge as,

$$\mathbf{E}[F(\bar{\mathbf{x}}_T)] - F^* \leq \underbrace{O(\sqrt{\log(T/\delta)}/\epsilon)}_{\text{privacy bound}} + \underbrace{O(1/\sqrt{T}) + O(1/T)}_{\text{optimization bound}},$$

where $\bar{\mathbf{x}}_T$ is an appropriate mean of the iterates.

- The theorem tells you that DP-FedAvg can obtain an error bound **which increases as $\log(T)$** , due to the privacy restriction. You can then appreciate the trade-off between having a good accuracy and a good privacy, once more.
- Observe the dependency on δ and ϵ , the more privacy you want, the larger is the error (or you need to stop early enough).

What have we learned?

- Ensuring privacy is very important in distributed processing

What have we learned?

- Ensuring privacy is very important in distributed processing
- There are at least three possible strategies of ensuring privacy, but in algorithmic design differential privacy is the one that is preferred

What have we learned?

- Ensuring privacy is very important in distributed processing
- There are at least three possible strategies of ensuring privacy, but in algorithmic design differential privacy is the one that is preferred
- DP enforces that all the participating entities are indistinguishable from one another; DP works by adding noise to the exchanged messages

What have we learned?

- Ensuring privacy is very important in distributed processing
- There are at least three possible strategies of ensuring privacy, but in algorithmic design differential privacy is the one that is preferred
- DP enforces that all the participating entities are indistinguishable from one another; DP works by adding noise to the exchanged messages
- Different noise mechanisms exist, and different algorithms for distributed optimization and federated learning can trade-off accuracy with privacy

What have we learned?

- Ensuring privacy is very important in distributed processing
- There are at least three possible strategies of ensuring privacy, but in algorithmic design differential privacy is the one that is preferred
- DP enforces that all the participating entities are indistinguishable from one another; DP works by adding noise to the exchanged messages
- Different noise mechanisms exist, and different algorithms for distributed optimization and federated learning can trade-off accuracy with privacy
- DGD and GT can be made convergent and private (albeit with some strong requirements), but research is still open

What have we learned?

- Ensuring privacy is very important in distributed processing
- There are at least three possible strategies of ensuring privacy, but in algorithmic design differential privacy is the one that is preferred
- DP enforces that all the participating entities are indistinguishable from one another; DP works by adding noise to the exchanged messages
- Different noise mechanisms exist, and different algorithms for distributed optimization and federated learning can trade-off accuracy with privacy
- DGD and GT can be made convergent and private (albeit with some strong requirements), but research is still open
- SGD/FedAvg can also be made private at the expense of stopping the iterations early

Sample references

Differential privacy:

- ① *C. Dwork, A. Roth, **The Algorithmic Foundations of Differential Privacy***, Foundations and Trends in Theoretical Computer Science, 2014

DP-Distributed optimization:

- ① *Y. Wang, A. Nedic, **Tailoring Gradient Methods for Differentially-Private Distributed Optimization***, arXiv:2202.01113, 2022
- ② References therein

DP-Federated learning:

- ① *Many, many authors, **Advances and Open Problems in Federated Learning***, arXiv:1912.04977, 2019