

Distributed asynchronous non-convex optimization: Fundamental limits of convergence rates

Matthew Hale

University of Florida

AFOSR DSCT Review

September 13, 2023

AFOSR YIP Grant #FA9550-23-1-0120

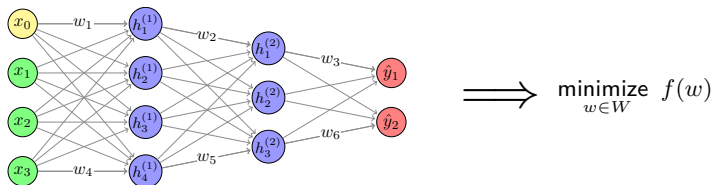
Project title: A Morse-Theoretic Approach
to Non-Convex Optimization



Georgia Institute
of Technology

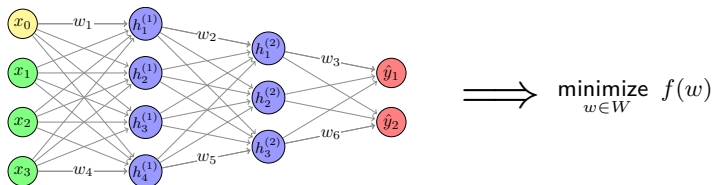
This project is motivated by learning and autonomy

- Example #1: Training neural networks leads to non-convex optimization problems

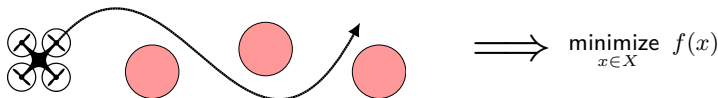


This project is motivated by learning and autonomy

- ▶ Example #1: Training neural networks leads to non-convex optimization problems

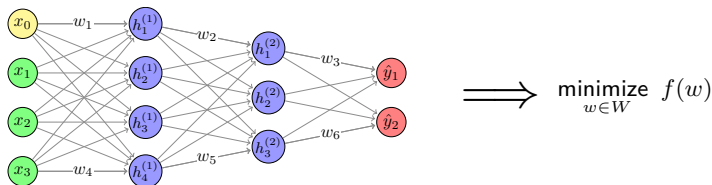


- ▶ Example #2: Trajectory planning also leads to non-convex optimization problems

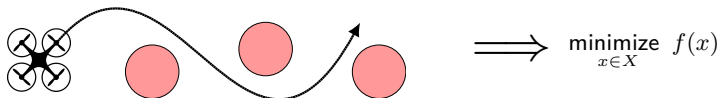


This project is motivated by learning and autonomy

- ▶ Example #1: Training neural networks leads to non-convex optimization problems



- ▶ Example #2: Trajectory planning also leads to non-convex optimization problems



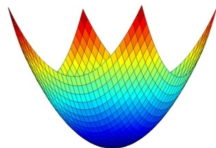
- ▶ Many other problems in learning and autonomy are non-convex

Guiding question for this project

How can we develop (i) novel algorithms and (ii) novel convergence guarantees when minimizing a non-convex f ?

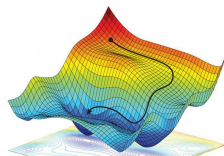
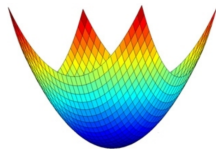
There is a growing literature on non-convex optimization

- ▶ Convex optimization has become a standard tool
 - ▶ Large literature on it
 - ▶ Numerous software packages
 - ▶ Relies on nice convex geometries



There is a growing literature on non-convex optimization

- ▶ Convex optimization has become a standard tool
 - ▶ Large literature on it
 - ▶ Numerous software packages
 - ▶ Relies on nice convex geometries
- ▶ Non-convex optimization is often much less structured



There is a growing literature on non-convex optimization

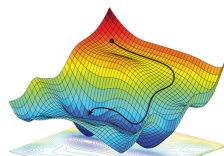
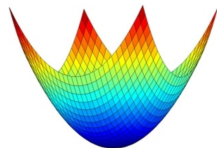
- ▶ Convex optimization has become a standard tool
 - ▶ Large literature on it
 - ▶ Numerous software packages
 - ▶ Relies on nice convex geometries
- ▶ Non-convex optimization is often much less structured
- ▶ One segment of the literature has identified geometric and topological properties that help analysis, e.g.,

Error bound condition: $\|\nabla f(x)\| \geq \alpha \|\Pi_{X^*}[x] - x\|$

Polyak-Łojasiewicz inequality: $\frac{1}{2} \|\nabla f(x)\|^2 \geq \beta (f(x) - f^*)$

Restricted secant inequality: $\langle \nabla f(x), x - \Pi_{X^*}[x] \rangle \geq \delta \|\Pi_{X^*}[x] - x\|^2$

Quadratic growth condition: $f(x) - f^* \geq \frac{\gamma}{2} \|\Pi_{X^*}[x] - x\|^2$



There is a growing literature on non-convex optimization

- ▶ Convex optimization has become a standard tool
 - ▶ Large literature on it
 - ▶ Numerous software packages
 - ▶ Relies on nice convex geometries
- ▶ Non-convex optimization is often much less structured
- ▶ One segment of the literature has identified geometric and topological properties that help analysis, e.g.,

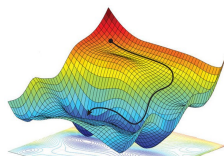
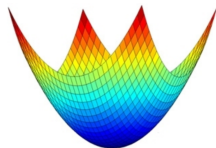
Error bound condition: $\|\nabla f(x)\| \geq \alpha \|\Pi_{X^*}[x] - x\|$

Polyak-Łojasiewicz inequality: $\frac{1}{2} \|\nabla f(x)\|^2 \geq \beta (f(x) - f^*)$

Restricted secant inequality: $\langle \nabla f(x), x - \Pi_{X^*}[x] \rangle \geq \delta \|\Pi_{X^*}[x] - x\|^2$

Quadratic growth condition: $f(x) - f^* \geq \frac{\gamma}{2} \|\Pi_{X^*}[x] - x\|^2$

- ▶ These are about functions, but we want to analyze function-algorithm pairs



This project will connect non-convex optimization to Morse theory

- ▶ A function-algorithm pair gives us a dynamical system
- ▶ E.g., given f , the gradient descent mapping $I - \gamma \nabla f$ gives the system

$$x_{k+1} = (I - \gamma \nabla f) x_k = x_k - \gamma \nabla f(x_k)$$

This project will connect non-convex optimization to Morse theory

- ▶ A function-algorithm pair gives us a dynamical system
- ▶ E.g., given f , the gradient descent mapping $I - \gamma \nabla f$ gives the system

$$x_{k+1} = \underbrace{(I - \gamma \nabla f)}_{\text{We choose this!}} x_k = x_k - \gamma \nabla f(x_k)$$

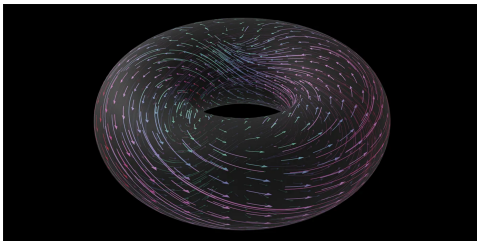
- ▶ Here, f is given but the algorithmic mapping is our choice

This project will connect non-convex optimization to Morse theory

- ▶ A function-algorithm pair gives us a dynamical system
- ▶ E.g., given f , the gradient descent mapping $I - \gamma \nabla f$ gives the system

$$x_{k+1} = \underbrace{(I - \gamma \nabla f)}_{\text{We choose this!}} x_k = x_k - \gamma \nabla f(x_k)$$

- ▶ Here, f is given but the algorithmic mapping is our choice
- ▶ We believe that non-convex optimization can benefit from Morse theory



Conventional Morse theory

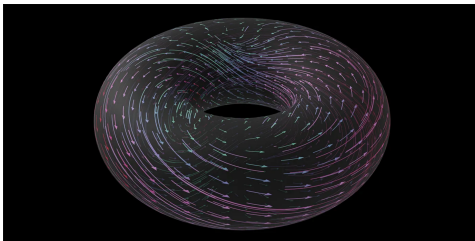
Choose a vector field on a manifold to study the manifold itself, which may have many maxima, minima, and saddles

This project will connect non-convex optimization to Morse theory

- ▶ A function-algorithm pair gives us a dynamical system
- ▶ E.g., given f , the gradient descent mapping $I - \gamma \nabla f$ gives the system

$$x_{k+1} = \underbrace{(I - \gamma \nabla f)}_{\text{We choose this!}} x_k = x_k - \gamma \nabla f(x_k)$$

- ▶ Here, f is given but the algorithmic mapping is our choice
- ▶ We believe that non-convex optimization can benefit from Morse theory



Conventional Morse theory

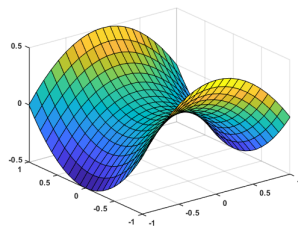
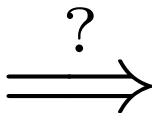
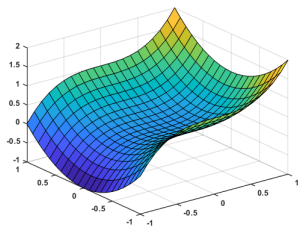
Choose a vector field on a manifold to study the manifold itself, which may have many maxima, minima, and saddles

In this project

Our manifolds are the graphs of f 's. We study & design vector fields on them (and these are optimization algorithms).

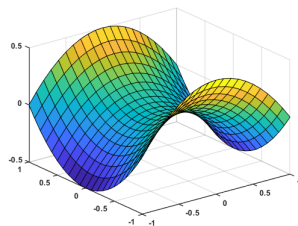
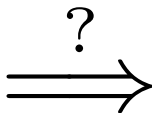
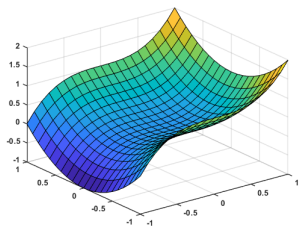
We have a 3-part plan to execute this project

- ▶ Thrust #1: Regularization for non-convex functions
 - ▶ What properties do we want functions to have?
 - ▶ How do we force those properties to hold?



We have a 3-part plan to execute this project

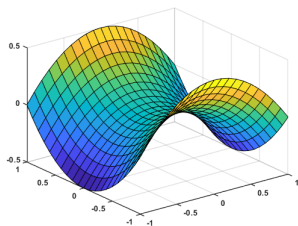
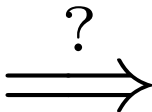
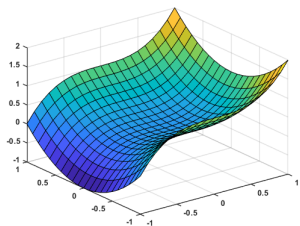
- ▶ Thrust #1: Regularization for non-convex functions
 - ▶ What properties do we want functions to have?
 - ▶ How do we force those properties to hold?



- ▶ Thrust #2: A framework for convergence analysis of decentralized algorithms
 - ▶ What conditions can eliminate case-by-case analyses?
 - ▶ How can we leverage properties gained by regularizing?

We have a 3-part plan to execute this project

- ▶ Thrust #1: Regularization for non-convex functions
 - ▶ What properties do we want functions to have?
 - ▶ How do we force those properties to hold?

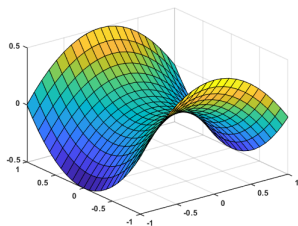
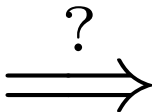
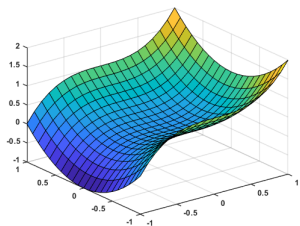


- ▶ Thrust #2: A framework for convergence analysis of decentralized algorithms
 - ▶ What conditions can eliminate case-by-case analyses?
 - ▶ How can we leverage properties gained by regularizing?
- ▶ Thrust #3: Design of new update laws
 - ▶ How can we use the above framework and regularizations to design faster algorithms?
 - ▶ Should we re-purpose algorithms for convex problems?

We have a 3-part plan to execute this project

► Thrust #1: Regularization for non-convex functions

- What properties do we want functions to have?
- How do we force those properties to hold?



► Thrust #2: A framework for convergence analysis of decentralized algorithms

- What conditions can eliminate case-by-case analyses?
- How can we leverage properties gained by regularizing?

Today: Results so far on these two

► Thrust #3: Design of new update laws

- How can we use the above framework and regularizations to design faster algorithms?
- Should we re-purpose algorithms for convex problems?

Thrust #1: Regularization for
non-convex functions

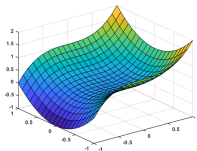
The strict saddle property is now common

Strict saddle property¹

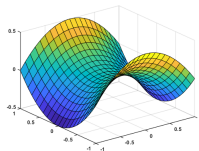
Any saddle point of f , denoted \hat{x}_{saddle} , satisfies

$$\lambda_{min}\left(\nabla^2 f(\hat{x}_{saddle})\right) < 0$$

Non-Strict



Strict



¹R. Ge, F. Huang, C. Jin, and Y. Yuan, "Escaping From Saddle Points — Online Stochastic Gradient for Tensor Decomposition," *Proceedings of The 28th Conference on Learning Theory*, no. 40, pp. 797-842.

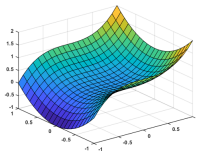
The strict saddle property is now common

Strict saddle property¹

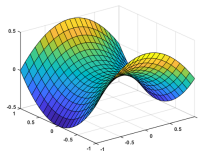
Any saddle point of f , denoted \hat{x}_{saddle} , satisfies

$$\lambda_{min}\left(\nabla^2 f(\hat{x}_{saddle})\right) < 0$$

Non-Strict



Strict



- The goal is to avoid all saddles and reach a minimum

¹R. Ge, F. Huang, C. Jin, and Y. Yuan, “Escaping From Saddle Points — Online Stochastic Gradient for Tensor Decomposition,” *Proceedings of The 28th Conference on Learning Theory*, no. 40, pp. 797-842.

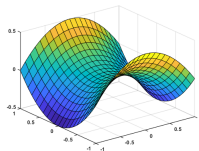
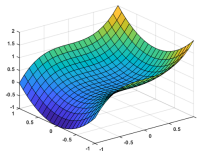
The strict saddle property is now common

Strict saddle property¹

Any saddle point of f , denoted \hat{x}_{saddle} , satisfies

$$\lambda_{min}\left(\nabla^2 f(\hat{x}_{saddle})\right) < 0$$

Non-Strict



Strict

- ▶ The goal is to avoid all saddles and reach a minimum
- ▶ Without SSP: algorithms can converge to saddles (giving high cost)
- ▶ With SSP: Saddles are almost always escaped (often quickly)

¹R. Ge, F. Huang, C. Jin, and Y. Yuan, "Escaping From Saddle Points — Online Stochastic Gradient for Tensor Decomposition," *Proceedings of The 28th Conference on Learning Theory*, no. 40, pp. 797-842.

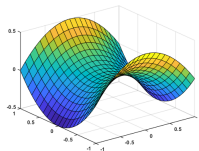
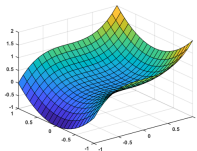
The strict saddle property is now common

Strict saddle property¹

Any saddle point of f , denoted \hat{x}_{saddle} , satisfies

$$\lambda_{min}\left(\nabla^2 f(\hat{x}_{saddle})\right) < 0$$

Non-Strict



Strict

- ▶ The goal is to avoid all saddles and reach a minimum
- ▶ Without SSP: algorithms can converge to saddles (giving high cost)
- ▶ With SSP: Saddles are almost always escaped (often quickly)

Fundamental question

Can we enforce the SSP when it doesn't hold? How?
(For simplicity, use gradient descent in this talk)

¹R. Ge, F. Huang, C. Jin, and Y. Yuan, "Escaping From Saddle Points — Online Stochastic Gradient for Tensor Decomposition," *Proceedings of The 28th Conference on Learning Theory*, no. 40, pp. 797-842.

We will apply linear regularizers locally

Perturbations give Morse functions (Milnor, *Lectures on the h-cobordism theorem*, 1965)

Let $f : \mathbb{R}^n \rightarrow \mathbb{R}$ be twice continuously differentiable. Then for almost every $\ell \in \mathbb{R}^n$ the function $x \mapsto f(x) + \ell^T x$ is Morse.

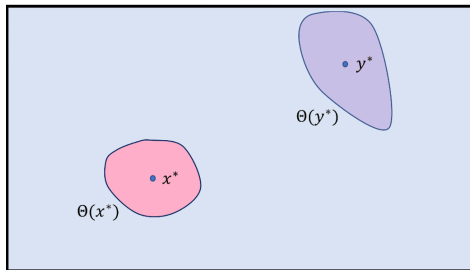
²M. Ubl, M. Hale, and K. Yazdani, “Linear Regularizers Enforce the Strict Saddle Property”, *Proceedings of the 37th AAAI Conference on Artificial Intelligence (AAAI)*, vol. 37, no. 8, pp. 10017-10024, Jun. 2023.

We will apply linear regularizers locally

Perturbations give Morse functions (Milnor, *Lectures on the h-cobordism theorem*, 1965)

Let $f : \mathbb{R}^n \rightarrow \mathbb{R}$ be twice continuously differentiable. Then for almost every $\ell \in \mathbb{R}^n$ the function $x \mapsto f(x) + \ell^T x$ is Morse.

- ▶ We will regularize locally around stationary points
- ▶ At a high level, we do this:
- ▶ Fix a parameter $\theta > 0$
- ▶ If $\|\nabla f(x_k)\| > \theta$:
$$x_{k+1} = x_k - \gamma \nabla f(x_k)$$
- ▶ If $\|\nabla f(x_k)\| \leq \theta$:
$$x_{k+1} = x_k - \gamma (\nabla f(x_k) + \ell)$$



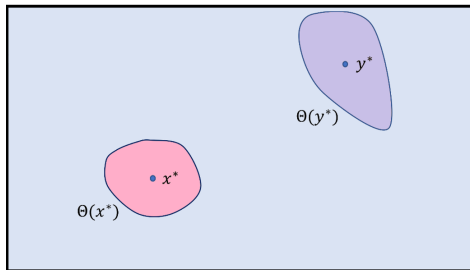
²M. Ubl, M. Hale, and K. Yazdani, “Linear Regularizers Enforce the Strict Saddle Property”, *Proceedings of the 37th AAAI Conference on Artificial Intelligence (AAAI)*, vol. 37, no. 8, pp. 10017-10024, Jun. 2023.

We will apply linear regularizers locally

Perturbations give Morse functions (Milnor, *Lectures on the h-cobordism theorem*, 1965)

Let $f : \mathbb{R}^n \rightarrow \mathbb{R}$ be twice continuously differentiable. Then for almost every $\ell \in \mathbb{R}^n$ the function $x \mapsto f(x) + \ell^T x$ is Morse.

- ▶ We will regularize locally around stationary points
- ▶ At a high level, we do this:
- ▶ Fix a parameter $\theta > 0$
- ▶ If $\|\nabla f(x_k)\| > \theta$:
$$x_{k+1} = x_k - \gamma \nabla f(x_k)$$
- ▶ If $\|\nabla f(x_k)\| \leq \theta$:
$$x_{k+1} = x_k - \gamma (\nabla f(x_k) + \ell)$$



Theorem #2: This escapes non-strict saddles!

Under mild technical conditions² this algorithm enforces SSP and escapes neighborhoods of saddles in a *finite* number of iterations.

²M. Ubl, M. Hale, and K. Yazdani, “Linear Regularizers Enforce the Strict Saddle Property”, *Proceedings of the 37th AAAI Conference on Artificial Intelligence (AAAI)*, vol. 37, no. 8, pp. 10017-10024, Jun. 2023.

Analytical error bounds reveal tradeoffs

- ▶ We consider any f definable in an o-minimal structure
- ▶ Recall: an o-minimal structure on a set M is $S = \{S_n\}_{n=0,1,2,\dots}$ such that for all n :
 - 1 S_n is a Boolean algebra of subsets of M^n
 - 2 if $A \in S_n$, then both $M \times A$ and $A \times M$ are in S_{n+1}
 - 3 the set $\{(x_1, \dots, x_n) \in M^n : x_1 = x_n\}$ is in S_n
 - 4 if $A \in S_{n+1}$ and $\pi : M^{n+1} \rightarrow M^n$ projects onto the first n coordinates, then $\pi(A) \in S_n$

Analytical error bounds reveal tradeoffs

- ▶ We consider any f definable in an o-minimal structure
- ▶ Recall: an o-minimal structure on a set M is $S = \{S_n\}_{n=0,1,2,\dots}$ such that for all n :
 - 1 S_n is a Boolean algebra of subsets of M^n
 - 2 if $A \in S_n$, then both $M \times A$ and $A \times M$ are in S_{n+1}
 - 3 the set $\{(x_1, \dots, x_n) \in M^n : x_1 = x_n\}$ is in S_n
 - 4 if $A \in S_{n+1}$ and $\pi : M^{n+1} \rightarrow M^n$ projects onto the first n coordinates, then $\pi(A) \in S_n$
- ▶ Functions definable in this way include
 - ▶ Semi-algebraic functions
 - ▶ Analytic functions
 - ▶ Sub-analytic functions
 - ▶ Semi-analytic functions
 - ▶ Many more

Analytical error bounds reveal tradeoffs

- ▶ We consider any f definable in an o-minimal structure
- ▶ Recall: an o-minimal structure on a set M is $S = \{S_n\}_{n=0,1,2,\dots}$ such that for all n :
 - 1 S_n is a Boolean algebra of subsets of M^n
 - 2 if $A \in S_n$, then both $M \times A$ and $A \times M$ are in S_{n+1}
 - 3 the set $\{(x_1, \dots, x_n) \in M^n : x_1 = x_n\}$ is in S_n
 - 4 if $A \in S_{n+1}$ and $\pi : M^{n+1} \rightarrow M^n$ projects onto the first n coordinates, then $\pi(A) \in S_n$
- ▶ Functions definable in this way include
 - ▶ Semi-algebraic functions
 - ▶ Analytic functions
 - ▶ Sub-analytic functions
 - ▶ Semi-analytic functions
 - ▶ Many more

Theorem #3: Regularization error bound

Let x^* be a minimizer of f . Let x_ℓ^* be its perturbed location after regularizing. Then

$$|f(x^*) - f(x_\ell^*)| \leq C\theta^{1/\alpha}$$

for some $\alpha \in (0, 2)$.

Analytical error bounds reveal tradeoffs

- ▶ We consider any f definable in an o-minimal structure
- ▶ Recall: an o-minimal structure on a set M is $S = \{S_n\}_{n=0,1,2,\dots}$ such that for all n :
 - 1 S_n is a Boolean algebra of subsets of M^n
 - 2 if $A \in S_n$, then both $M \times A$ and $A \times M$ are in S_{n+1}
 - 3 the set $\{(x_1, \dots, x_n) \in M^n : x_1 = x_n\}$ is in S_n
 - 4 if $A \in S_{n+1}$ and $\pi : M^{n+1} \rightarrow M^n$ projects onto the first n coordinates, then $\pi(A) \in S_n$
- ▶ Functions definable in this way include
 - ▶ Semi-algebraic functions
 - ▶ Analytic functions
 - ▶ Sub-analytic functions
 - ▶ Semi-analytic functions
 - ▶ Many more

Theorem #3: Regularization error bound

Let x^* be a minimizer of f . Let x_ℓ^* be its perturbed location after regularizing. Then

$$|f(x^*) - f(x_\ell^*)| \leq C\theta^{1/\alpha}$$

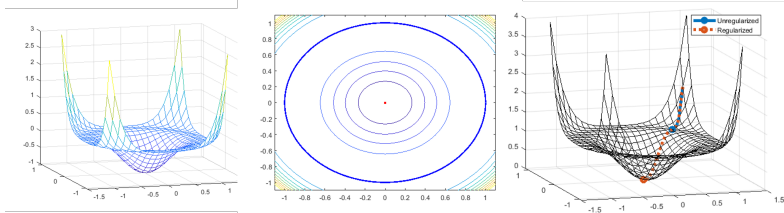
for some $\alpha \in (0, 2)$.

- ▶ There is a speed-accuracy tradeoff here:

Larger $\theta \Rightarrow$ Regularize sooner \Rightarrow escape saddles sooner, but larger error

Next steps for Thrust #1

- What is a faster regularizer?
 - Can randomness help?
 - Should we use 2^{nd} -order information? When? How?



Counterpart to Tikhonov regularization

For convex f , one often adds a quadratic term to enforce strong convexity.
For non-convex f , linear regularizers are the counterpart for enforcing SSP.

Thrust #2: A framework for convergence analysis of decentralized algorithms

Thrust #2 examines asynchronous parallelized algorithms

- Consider using N agents to solve

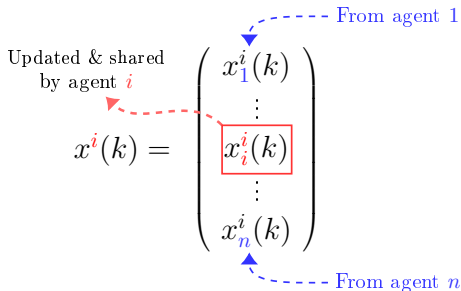
$$\underset{x \in \mathbb{R}^n}{\text{minimize}} \ f(x)$$

Thrust #2 examines asynchronous parallelized algorithms

- Consider using N agents to solve

$$\underset{x \in \mathbb{R}^n}{\text{minimize}} \ f(x)$$

- Agent i stores a local copy of x , denoted $x^i \in \mathbb{R}^n$, which is



Thrust #2 examines asynchronous parallelized algorithms

- Consider using N agents to solve

$$\underset{x \in \mathbb{R}^n}{\text{minimize}} f(x)$$

- Agent i stores a local copy of x , denoted $x^i \in \mathbb{R}^n$, which is

Updated & shared
by agent i

$$x^i(k) = \begin{pmatrix} x_1^i(k) \\ \vdots \\ x_i^i(k) \\ \vdots \\ x_n^i(k) \end{pmatrix}$$

From agent 1

From agent n

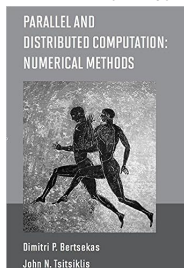
- Its computations are

$$x_i^i(k+1) = x_i^i(k) - \gamma \frac{\partial f}{\partial x_i}(x^i(k))$$

There are 2 classes of asynchrony

1 Asynchrony class #1: Total asynchrony

- ▶ All delays are finite, but can be arbitrarily long
- ▶ Only guaranteed to converge for a subset of strongly convex problems
- ▶ Known to be faster than stopping to synchronize³:



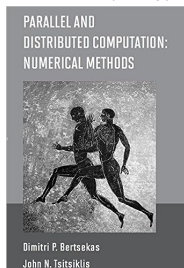
processors send the same number of messages and the messages have the same delays. We may conclude that, in the case of monotone iterations, it is preferable to perform as many updates as possible even if they are based on outdated information and, therefore, asynchronous algorithms are advantageous.

³D. Bertsekas and J. Tsitsiklis, *Parallel and distributed computation: Numerical methods*, Athena Scientific, 1989.

There are 2 classes of asynchrony

1 Asynchrony class #1: Total asynchrony

- ▶ All delays are finite, but can be arbitrarily long
- ▶ Only guaranteed to converge for a subset of strongly convex problems
- ▶ Known to be faster than stopping to synchronize³:



...the messages have the same delays. We may conclude that, in the case of monotone iterations, it is preferable to perform as many updates as possible even if they are based on outdated information and, therefore, asynchronous algorithms are advantageous.

2 Asynchrony class #2: Partial asynchrony (PA)

- ▶ All delays are bounded by some $B > 0$
- ▶ Does *not* require convexity

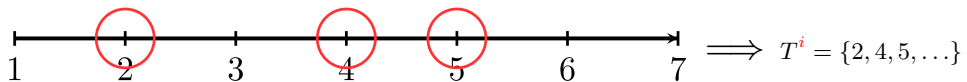
Fundamental question

Is PA always faster than stopping and synchronizing?

³D. Bertsekas and J. Tsitsiklis, *Parallel and distributed computation: Numerical methods*, Athena Scientific, 1989.

We need to specify the data of an execution

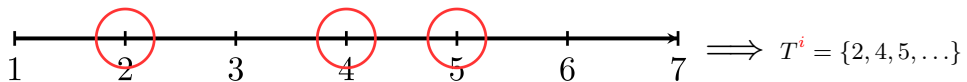
- We define $T^i \subseteq \mathbb{N}$ to be the (infinite) set of times at which agent i computes



- Then $T = \{T^1, T^2, \dots, T^N\}$ is all agents' computation times

We need to specify the data of an execution

- We define $T^i \subseteq \mathbb{N}$ to be the (infinite) set of times at which agent i computes



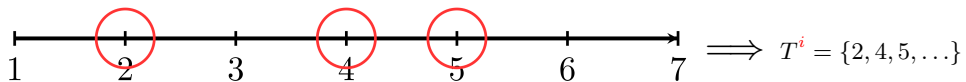
- Then $T = \{T^1, T^2, \dots, T^N\}$ is all agents' computation times
- Inside the vector

$$x^i(k) = \begin{pmatrix} x_1^i(k) \\ \vdots \\ x_i^i(k) \\ \vdots \\ x_n^i(k) \end{pmatrix}$$

the value of $x_j^i(k)$ is old. When was it up-to-date for agent j ?

We need to specify the data of an execution

- ▶ We define $T^i \subseteq \mathbb{N}$ to be the (infinite) set of times at which agent i computes



- ▶ Then $T = \{T^1, T^2, \dots, T^N\}$ is all agents' computation times
- ▶ Inside the vector

$$x^i(k) = \begin{pmatrix} x_1^i(k) \\ \vdots \\ x_i^i(k) \\ \vdots \\ x_n^i(k) \end{pmatrix}$$

the value of $x_j^i(k)$ is old. When was it up-to-date for agent j ?

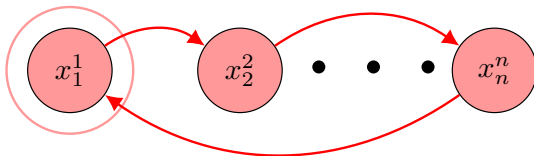
- ▶ Answer: $\tau_j^i(k)$ is when agent j had $x_j^i(k)$ onboard, i.e.,

$$x_j^i(k) = x_j^j(\tau_j^i(k))$$

- ▶ We define $\tau = \{\tau_j^i\}_{i,j \in \{1, \dots, N\}}$

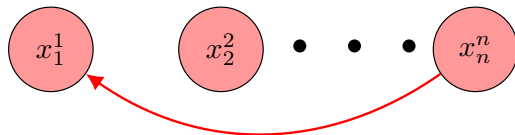
When are asynchronous executions faster?

- We define $(x(0), T, \tau)$ as a PA execution



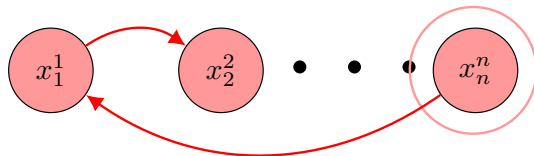
When are asynchronous executions faster?

- We define $(x(0), T, \tau)$ as a PA execution



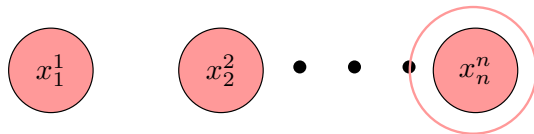
When are asynchronous executions faster?

- We define $(x(0), T, \tau)$ as a PA execution



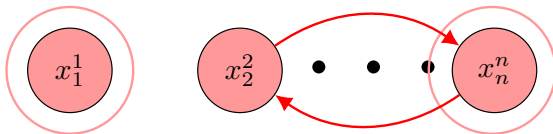
When are asynchronous executions faster?

- We define $(x(0), T, \tau)$ as a PA execution



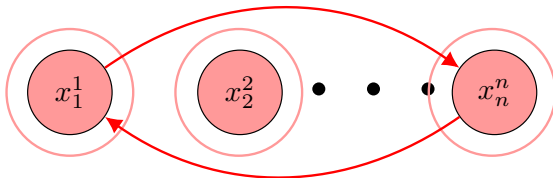
When are asynchronous executions faster?

- We define $(x(0), T, \tau)$ as a PA execution



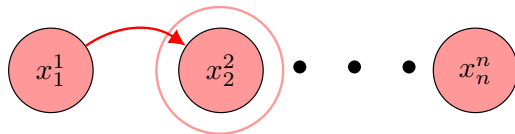
When are asynchronous executions faster?

- We define $(x(0), T, \tau)$ as a PA execution



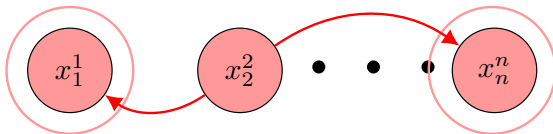
When are asynchronous executions faster?

- We define $(x(0), T, \tau)$ as a PA execution



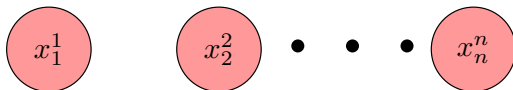
When are asynchronous executions faster?

- We define $(x(0), T, \tau)$ as a PA execution



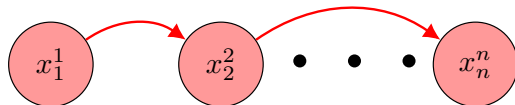
When are asynchronous executions faster?

- We define $(x(0), T, \tau)$ as a PA execution



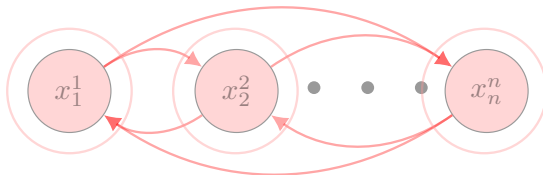
When are asynchronous executions faster?

- We define $(x(0), T, \tau)$ as a PA execution



When are asynchronous executions faster?

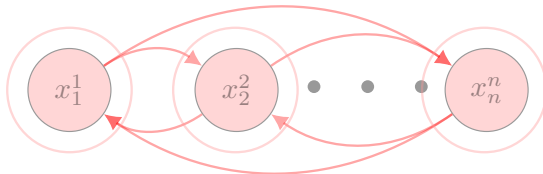
- ▶ We define $(x(0), T, \tau)$ as a PA execution



- ▶ We compare to “synchronization by idling” (SI)
 - ▶ All agents wait to hear from all others before computing
 - ▶ They idle between communications

When are asynchronous executions faster?

- ▶ We define $(x(0), T, \tau)$ as a PA execution



- ▶ We compare to “synchronization by idling” (SI)
 - ▶ All agents wait to hear from all others before computing
 - ▶ They idle between communications

Fundamental comparison

When is PA guaranteed to be faster than SI?

We cannot guarantee that PA is faster than SI

- Recall that agent i 's update law is

$$x_i^i(k+1) = x_i^i(k) - \gamma \frac{\partial f}{\partial x_i}(x^i(k))$$

Theorem #4: SI can be faster

Given B , there exist functions f and executions $(x(0), T, \tau)$ in which there does not exist a constant stepsize γ such that PA is guaranteed to be faster than SI.

We cannot guarantee that PA is faster than SI

- Recall that agent i 's update law is

$$x_i^i(k+1) = x_i^i(k) - \gamma \frac{\partial f}{\partial x_i}(x^i(k))$$

Theorem #4: SI can be faster

Given B , there exist functions f and executions $(x(0), T, \tau)$ in which there does not exist a constant stepsize γ such that PA is guaranteed to be faster than SI.

Theorem #5: SI can be faster even when we favor PA

Suppose B and $(x(0), T, \tau)$ are given. Then there still exist functions f for which there does not exist a constant stepsize γ such that PA is guaranteed to be faster than SI.

We cannot guarantee that PA is faster than SI

- Recall that agent i 's update law is

$$x_i^i(k+1) = x_i^i(k) - \gamma \frac{\partial f}{\partial x_i}(x^i(k))$$

Theorem #4: SI can be faster

Given B , there exist functions f and executions $(x(0), T, \tau)$ in which there does not exist a constant stepsize γ such that PA is guaranteed to be faster than SI.

Theorem #5: SI can be faster even when we favor PA

Suppose B and $(x(0), T, \tau)$ are given. Then there still exist functions f for which there does not exist a constant stepsize γ such that PA is guaranteed to be faster than SI.

- Unpacking these results:

- 1 These results are about guarantees. PA can still be faster than SI if we're lucky
- 2 These results do not apply to delay-adaptive stepsizes, e.g., using $\gamma_i(k)$ for each i

We cannot guarantee that PA is faster than SI

- ▶ Recall that agent i 's update law is

$$x_i^i(k+1) = x_i^i(k) - \gamma \frac{\partial f}{\partial x_i}(x^i(k))$$

Theorem #4: SI can be faster

Given B , there exist functions f and executions $(x(0), T, \tau)$ in which there does not exist a constant stepsize γ such that PA is guaranteed to be faster than SI.

Theorem #5: SI can be faster even when we favor PA

Suppose B and $(x(0), T, \tau)$ are given. Then there still exist functions f for which there does not exist a constant stepsize γ such that PA is guaranteed to be faster than SI.

- ▶ Unpacking these results:

- 1 These results are about guarantees. PA can still be faster than SI if we're lucky
- 2 These results do not apply to delay-adaptive stepsizes, e.g., using $\gamma_i(k)$ for each i

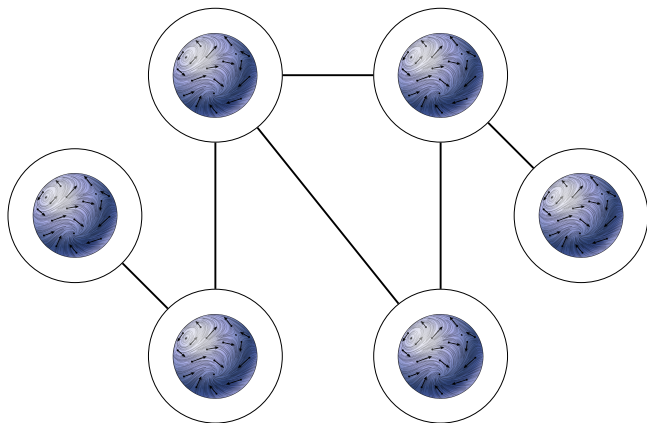
PA algorithms are still useful

SI requires fine-grained control over the timing of agents' behaviors. We don't always have that, so we still need PA algorithms.



Next steps in Thrust #2

- What are the weakest conditions on f under which distributed asynchronous non-convex optimization will converge?



Thank you

Matthew Hale
University of Florida
`matthewhale@ufl.edu`
`http://corelab.mae.ufl.edu`

Grant no. FA9550-23-1-0120