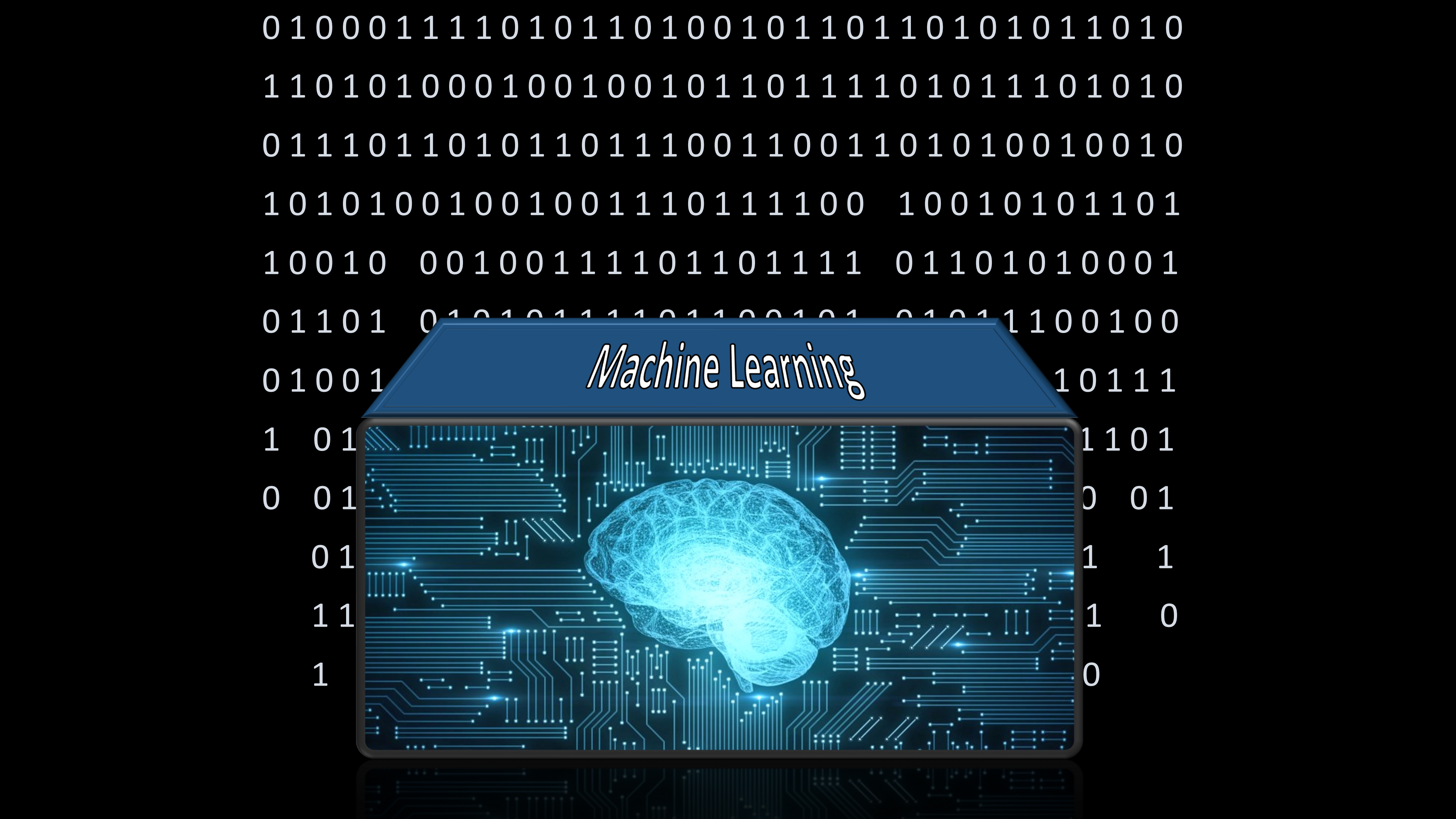
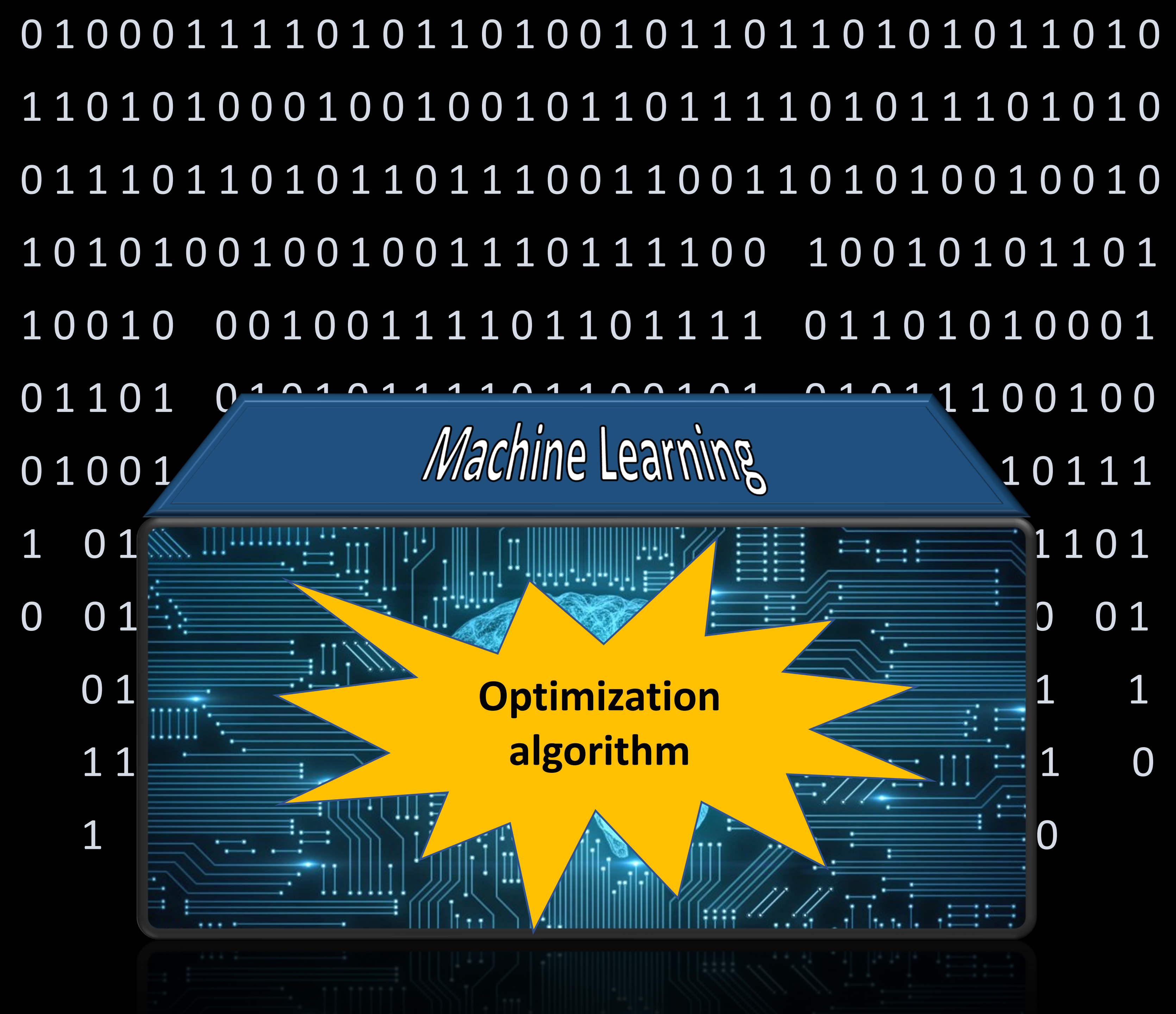
Memory as a lens to understand efficient learning and optimization

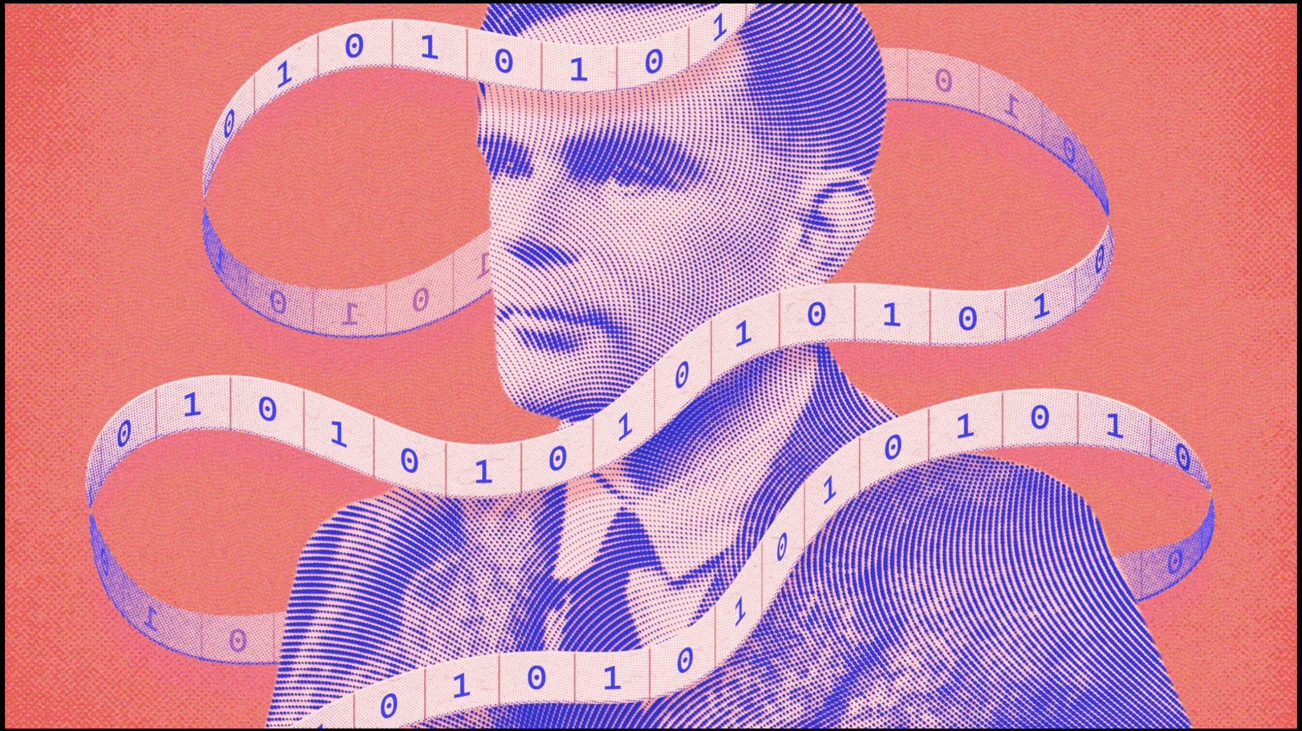
Vatsal Sharan (USC)


Machine Learning


Machine Learning
Optimization
algorithm

How do information and computation interact for optimization?

Memory as the Computational Resource

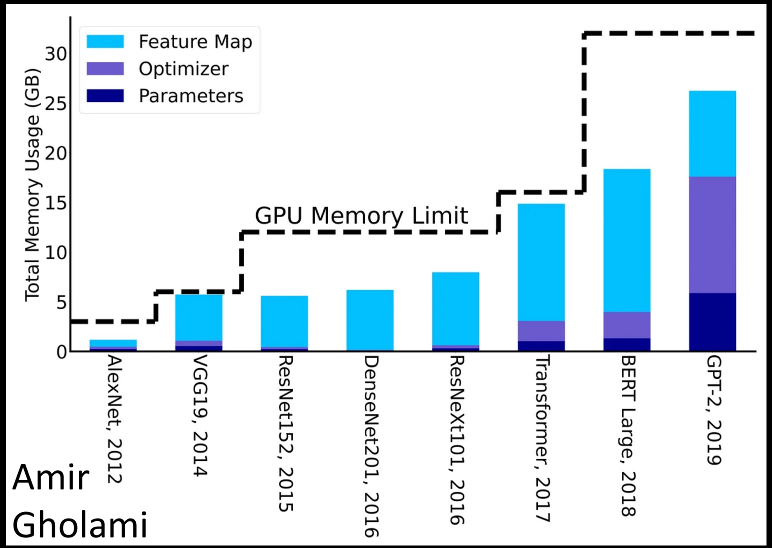


Traditionally in TCS, Memory has been a fundamental computational resource

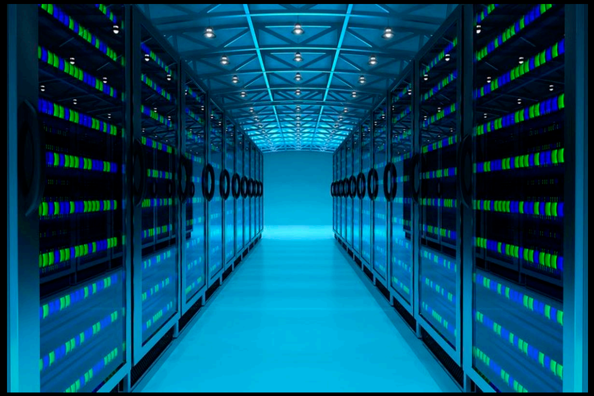
Pic: Quanta Magazine

Memory is a Constraint in Many Modern Practical Settings

Small memory



Large models



Huge datasets

``Memory is the dominant performance and energy bottleneck in modern computing systems; data movement is much more expensive than computation, both in latency and energy.” [Falcao and Ferreira, CACM, 2023]

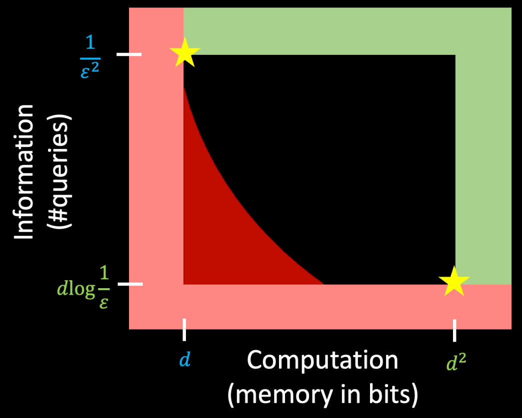
Memory is a fundamental computation resource, is crucial in practice.

What is the role of memory in learning and **optimization**?
Are there tradeoffs between available **memory** and required **information**?

#gradient queries

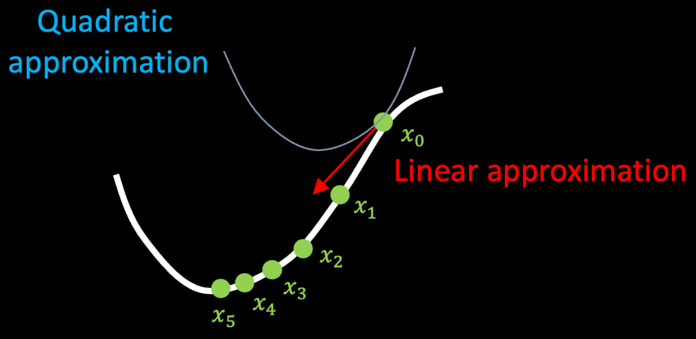
#data points

[This talk] Memory Dichotomy Hypothesis: It is not possible to significantly improve on the convergence rate of known memory efficient techniques without using significantly more memory.



Lower bounds: Convex optimization with first-order oracle

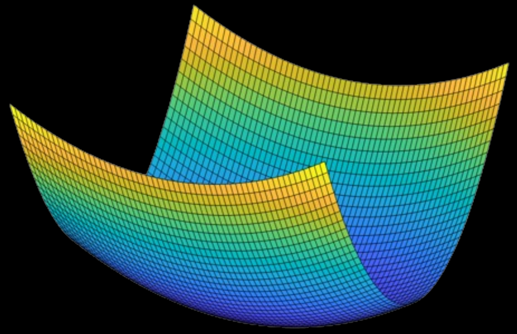
(with Annie Marsden, Aaron Sidford & Greg Valiant)



1st order vs. 2nd order methods

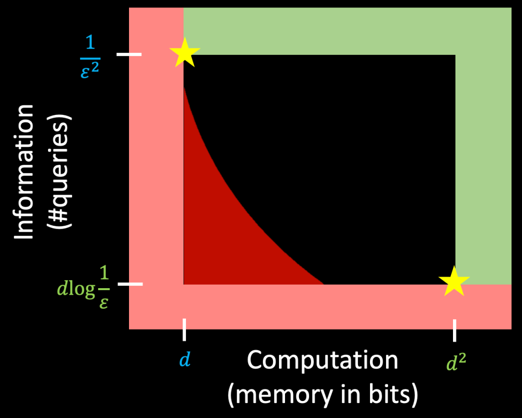
Lower bounds: Convex optimization with stochastic gradient oracle

(with Aaron Sidford & Greg Valiant)



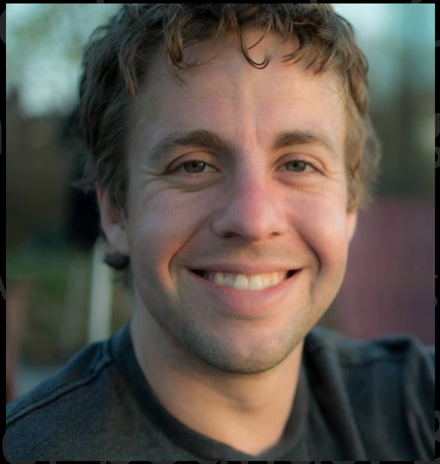
Upper bounds: Better convergence with small memory

(with Jon Kelner, Annie Marsden, Aaron Sidford, Greg Valiant, Honglin Yuan)

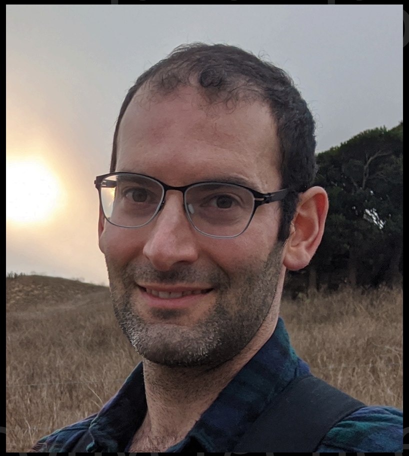


Lower bounds: Convex optimization with first-order oracle

Annie Marsden



Aaron Sidford



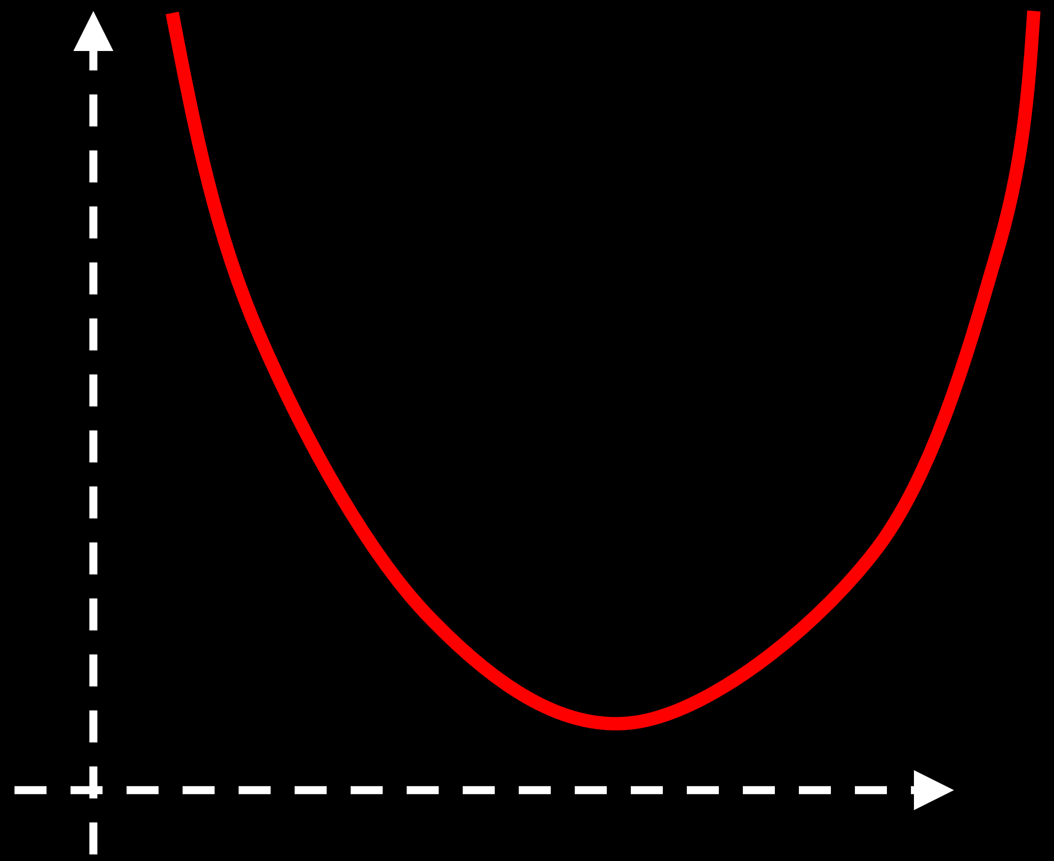
Greg Valiant

Efficient Convex Optimization Requires Superlinear Memory,
Annie Marsden, Vatsal Sharan, Aaron Sidford, Gregory Valiant, 2022

A canonical optimization problem

Consider minimizing convex, 1- Lipschitz functions:

$$\min_{x \in R^d : \|x\| \leq 1} F(x)$$



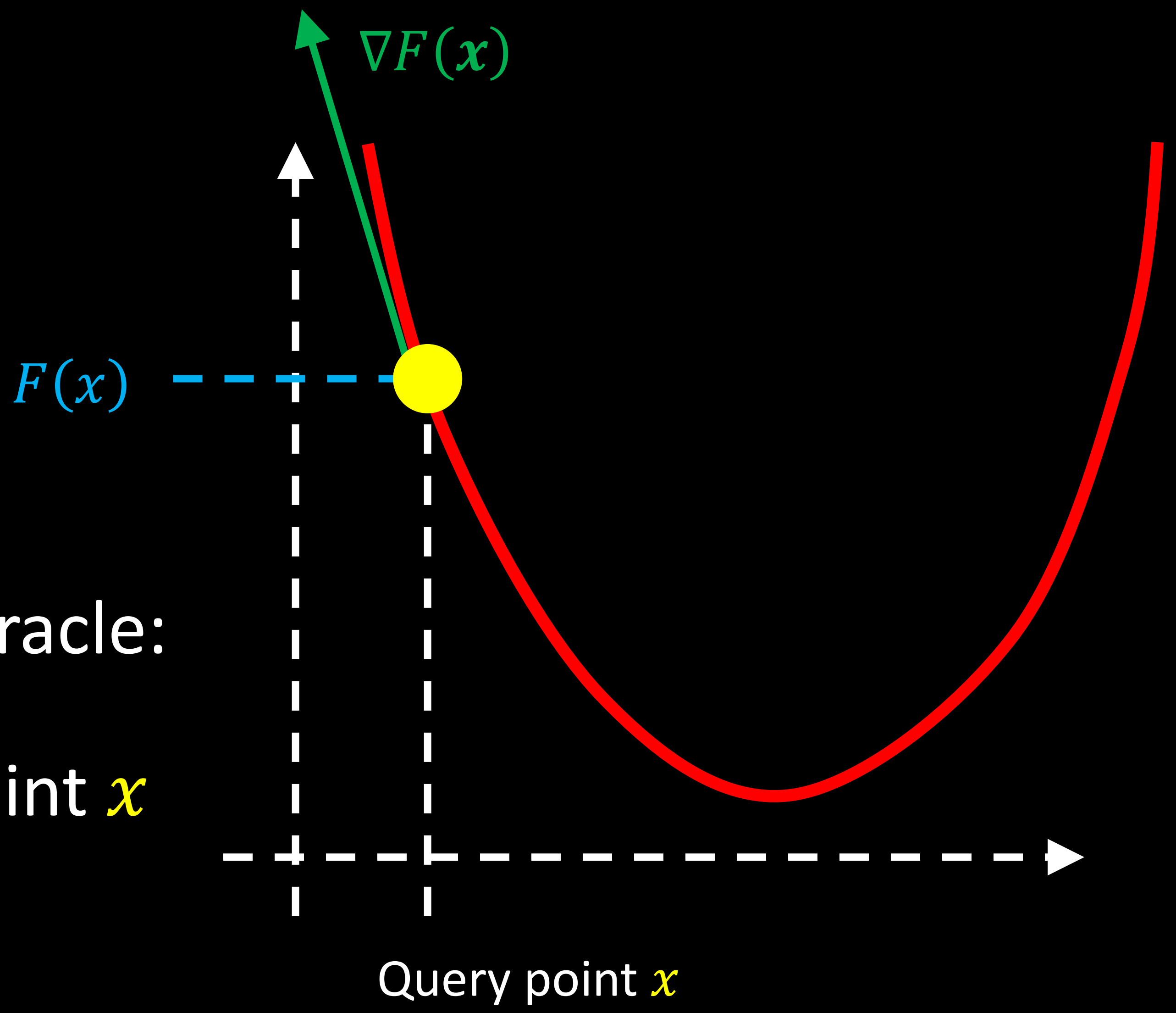
A canonical optimization problem

Consider minimizing convex,
1- Lipschitz functions:

$$\min_{x \in R^d : \|x\| \le 1} F(x)$$

Given access to a first-order oracle:

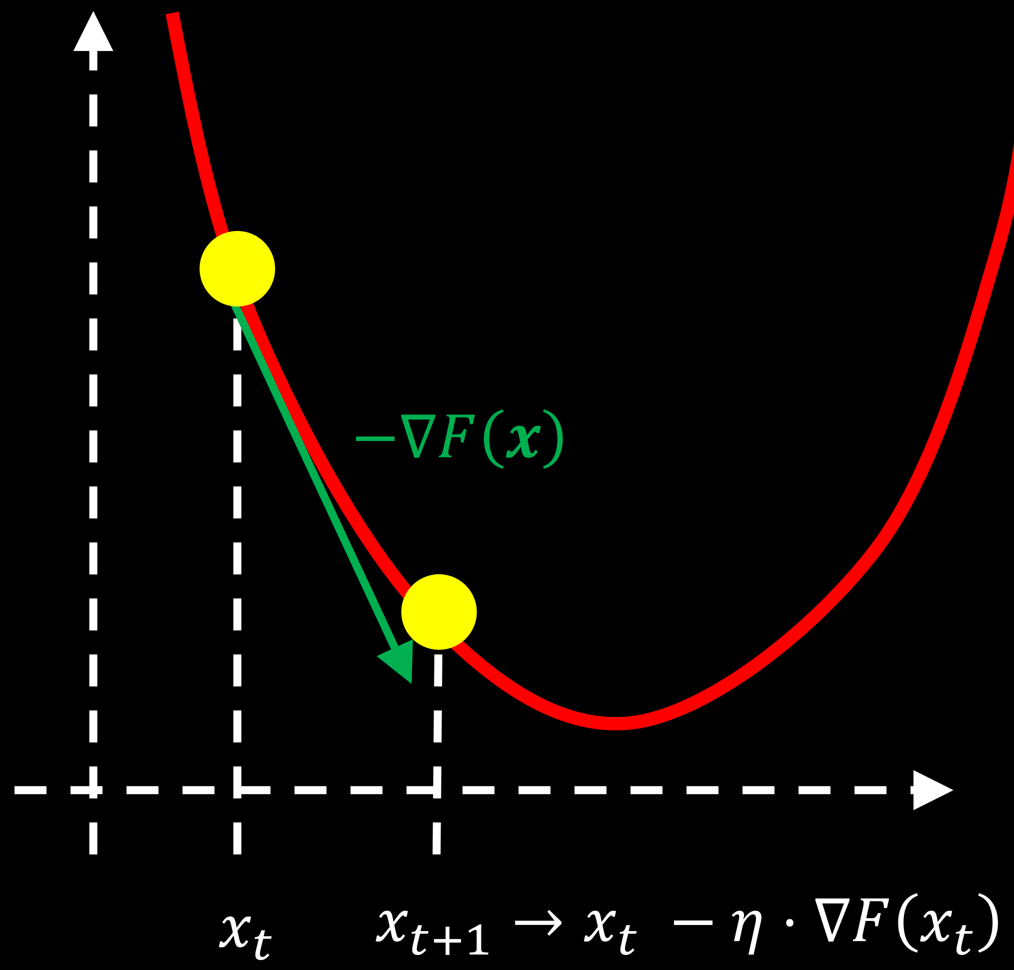
- Algorithm queries some point x
- Oracle responds with $(F(x), \nabla F(x))$



Algorithms we know

Gradient Descent

Initialize x_0 . At time t,
Query point x_t
Receive gradient $\nabla F(x_t)$ at x_t
Update $x_{t+1} \to x_t - \eta \cdot \nabla F(x_t)$

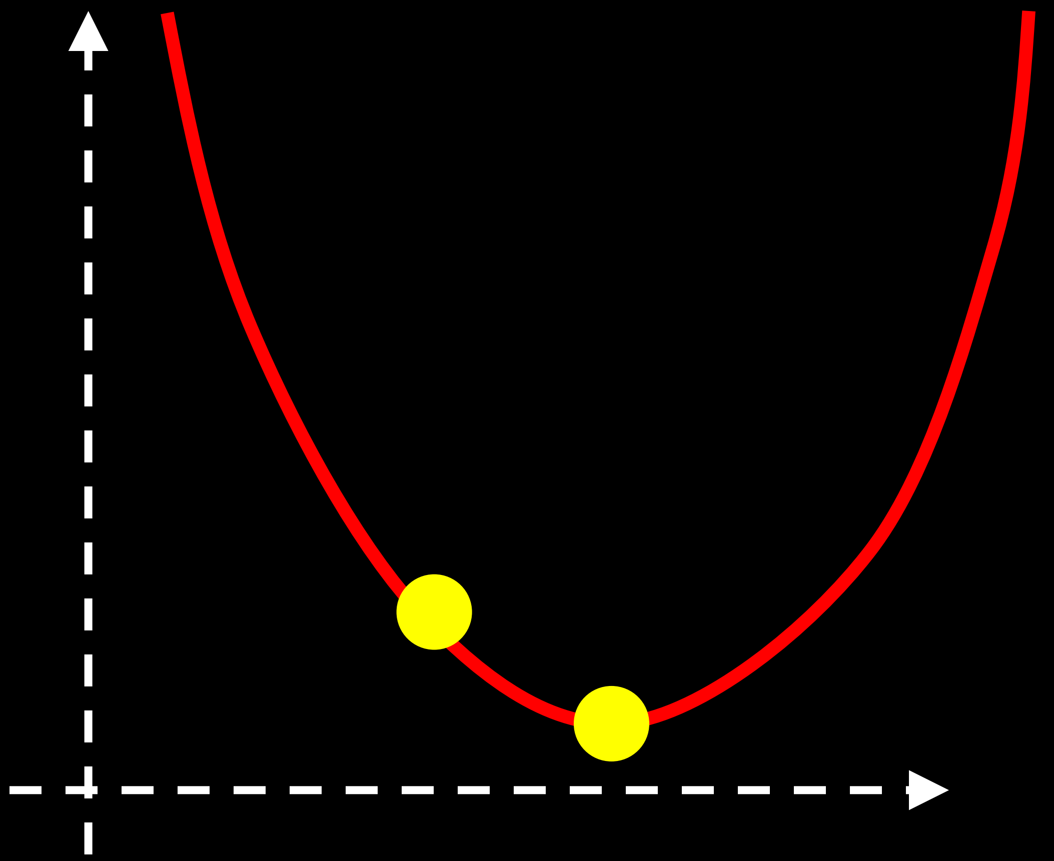


Algorithms we know

Gradient Descent

Initialize x_0 . At time t,
Query point x_t
Receive gradient $\nabla F(x_t)$ at x_t
Update $x_{t+1} \to x_t - \eta \cdot \nabla F(x_t)$

- $O(d)$ computation time per query
- $O(d)$ memory per query
- Query complexity large with respect to desired error ϵ: need ϵ^{-2} queries to find ϵ optimal answer



Algorithms we know

Gradient Descent

Initialize x_0 . At time t,
Query point x_t
Receive gradient $\nabla F(x_t)$ at x_t
Update $x_{t+1} \rightarrow x_t - \eta \cdot \nabla F(x_t)$

Suite of other techniques

- Based on the ellipsoid algorithm
- Does something like high-dimensional binary search

- $O(d)$ computation time per query
- $O(d)$ memory per query
- Query complexity large with respect to desired error ϵ: need ϵ^{-2} queries to find ϵ optimal answer

- $> d^2$ computation time per query
- $> d^2$ memory per query
- Query complexity small with respect to desired error ϵ: need $d \log\left(\frac{1}{\epsilon}\right)$ queries to find ϵ optimal answer

Algorithms we know

Gradient Descent

Suite of other techniques

Are complex algorithms necessary?

Algorithms we know

Gradient Descent

Initialize x_0 . At time t,
Query point x_t
Receive gradient $\nabla F(x_t)$ at x_t
Update $x_{t+1} \rightarrow x_t - \eta \cdot \nabla F(x_t)$

- $O(d)$ computation time per query
- $O(d)$ memory per query
- Query complexity large with respect to desired error ϵ: need ϵ^{-2} queries to find ϵ optimal answer

Suite of other techniques

- Based on the ellipsoid algorithm
- Does something like high-dimensional binary search

- $> d^2$ computation time per query
- $> d^2$ memory per query
- Query complexity small with respect to desired error ϵ: need $d \log\left(\frac{1}{\epsilon}\right)$ queries to find ϵ optimal answer

Algorithms we know

Gradient Descent

Initialize x_0 . At time t,
Query point x_t
Receive gradient $\nabla F(x_t)$ at x_t
Update $x_{t+1} \to x_t - \eta \cdot \nabla F(x_t)$

- $O(d)$ computation time per query
- $O(d)$ memory per query
- Query complexity large with respect to desired error ϵ: need ϵ^{-2} queries to find ϵ optimal answer

Suite of other techniques

- Based on the ellipsoid algorithm
- Does something like high-dimensional binary search

- $> d^2$ computation time per query
- $> d^2$ memory per query
- Query complexity small with respect to desired error ϵ: need $d \log\left(\frac{1}{\epsilon}\right)$ queries to find ϵ optimal answer

Algorithms we know

Gradient Descent

Initialize x_0 . At time t,
Query point x_t
Receive gradient $\nabla F(x_t)$ at x_t
Update $x_{t+1} \rightarrow x_t - \eta \cdot \nabla F(x_t)$

- $O(d)$ computation time per query
- $O(d)$ memory per query
- Query complexity large with respect to desired error ϵ: need ϵ^{-2} queries to find ϵ optimal answer

Suite of other techniques

- Based on the ellipsoid algorithm
- Does something like high-dimensional binary search

- $> d^2$ computation time per query
- $> d^2$ memory per query
- Query complexity small with respect to desired error ϵ: need $d \log\left(\frac{1}{\epsilon}\right)$ queries to find ϵ optimal answer

Algorithms we know

Gradient Descent

Initialize x_0 . At time t,
Query point x_t
Receive gradient $\nabla F(x_t)$ at x_t
Update $x_{t+1} \rightarrow x_t - \eta \cdot \nabla F(x_t)$

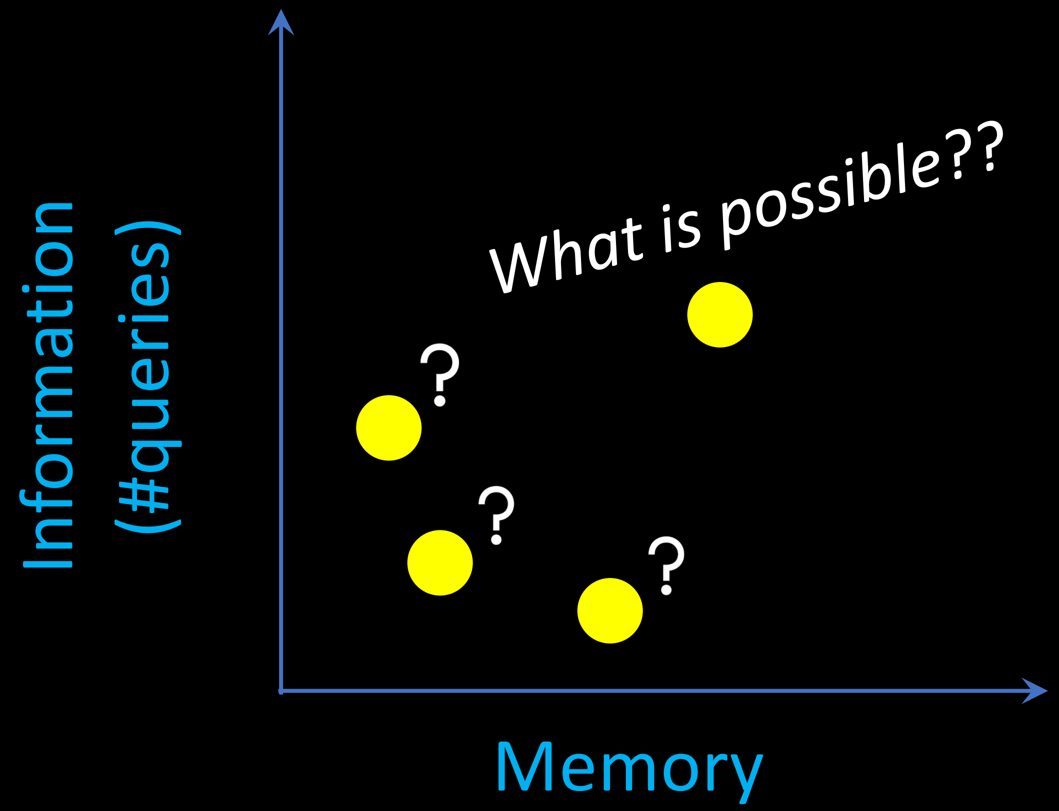
- $O(d)$ computation time per query
- $O(d)$ memory per query
- Query complexity large with respect to desired error ϵ: need ϵ^{-2} queries to find ϵ optimal answer

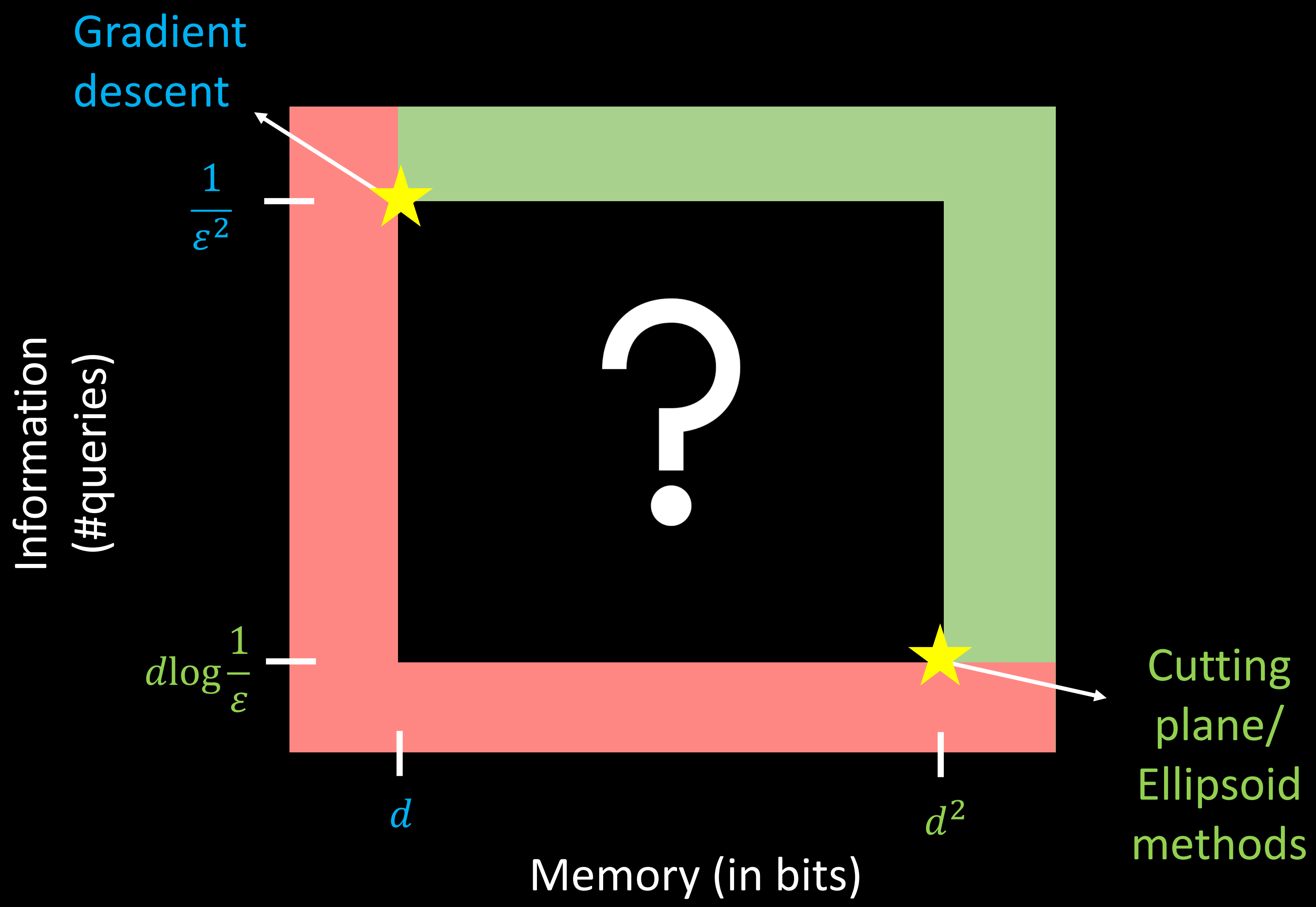
Suite of other techniques

- Based on the ellipsoid algorithm
- Does something like high-dimensional binary search

- $> d^2$ computation time per query
- $> d^2$ memory per query
- Query complexity small with respect to desired error ϵ: need $d \log\left(\frac{1}{\epsilon}\right)$ queries to find ϵ optimal answer

Are there inherent **tradeoffs** between available **memory** and **information** requirement?




Gradient descent
$\frac{1}{\varepsilon^2}$
$d \log \frac{1}{\varepsilon}$
Information (#queries)
?
d
d^2
Memory (in bits)
Cutting plane/ Ellipsoid methods

What is known?

Info-theoretic bounds for optimization algorithms

Nemirovski-Yudin'83,
Shamir’13,
Nesterov’14,
Bubeck’15,
Duchi-Jordan-Wainwright-Wibisono’15,
Woodworth-Srebro’16,
Carmon-Duchi-Hinder-Sidford’17ab,
Arjevani-Shamir’17,
Agarwal-Hazan’18,
Diakonikolas-Guzman’19

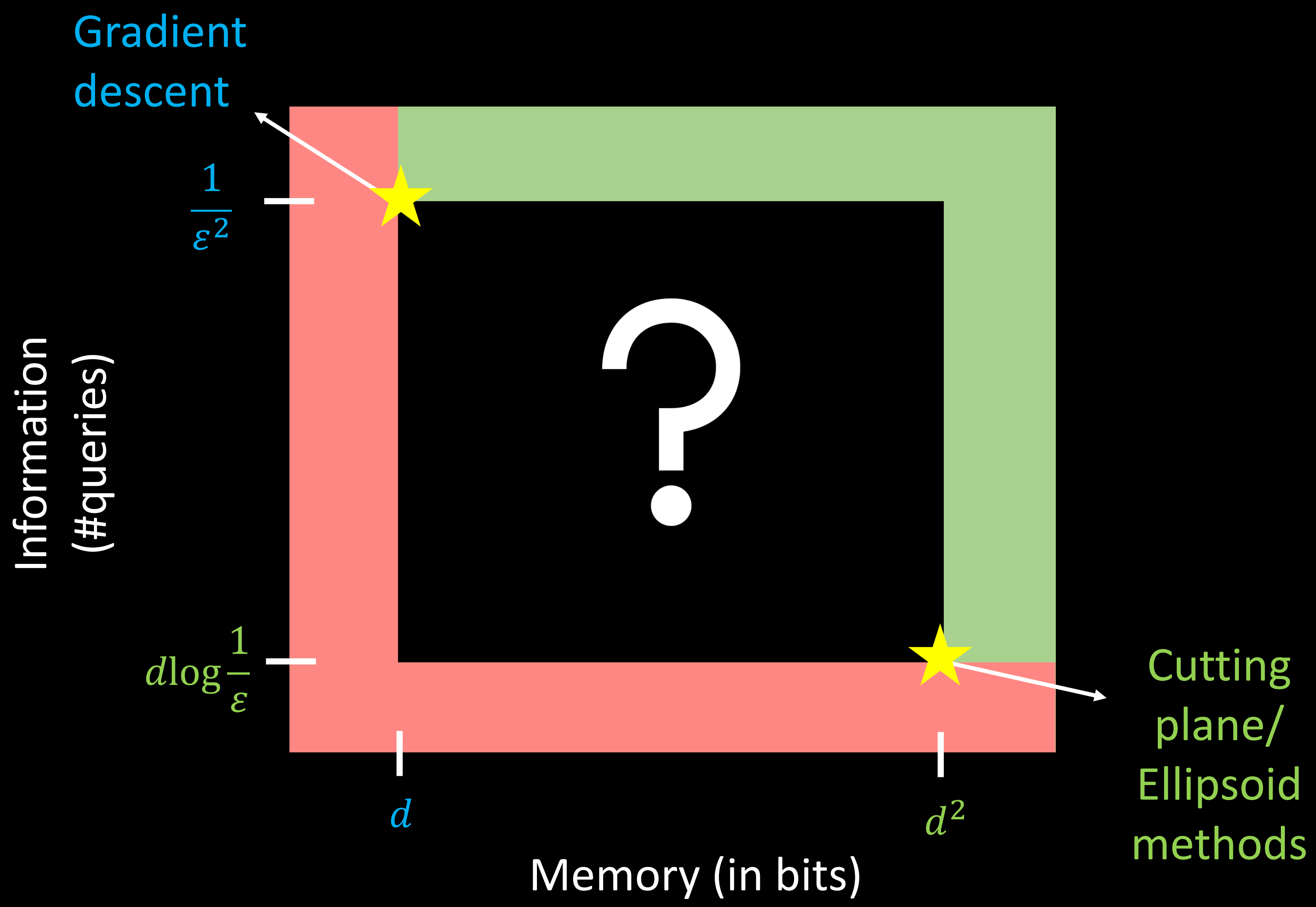
Memory bounds for streaming data

Alon-Matias-Szegedy’99,
Indyk-Woodruff’03
Bar-Yossef-Jayaram-Kumar-Sivakumar’04,
Nelson-Le Huy’13,
Steinhardt-Duchi’15,
Braverman-Garg-Ma-Nguyen-Woodruff’16,
Kapralov-Nelson-Pachocki-Wang-Woodruff-Yahyazadeh’17,
Nelson-Yu’19,
Dagan-Kur-Shamir’19

Memory bounds over finite fields

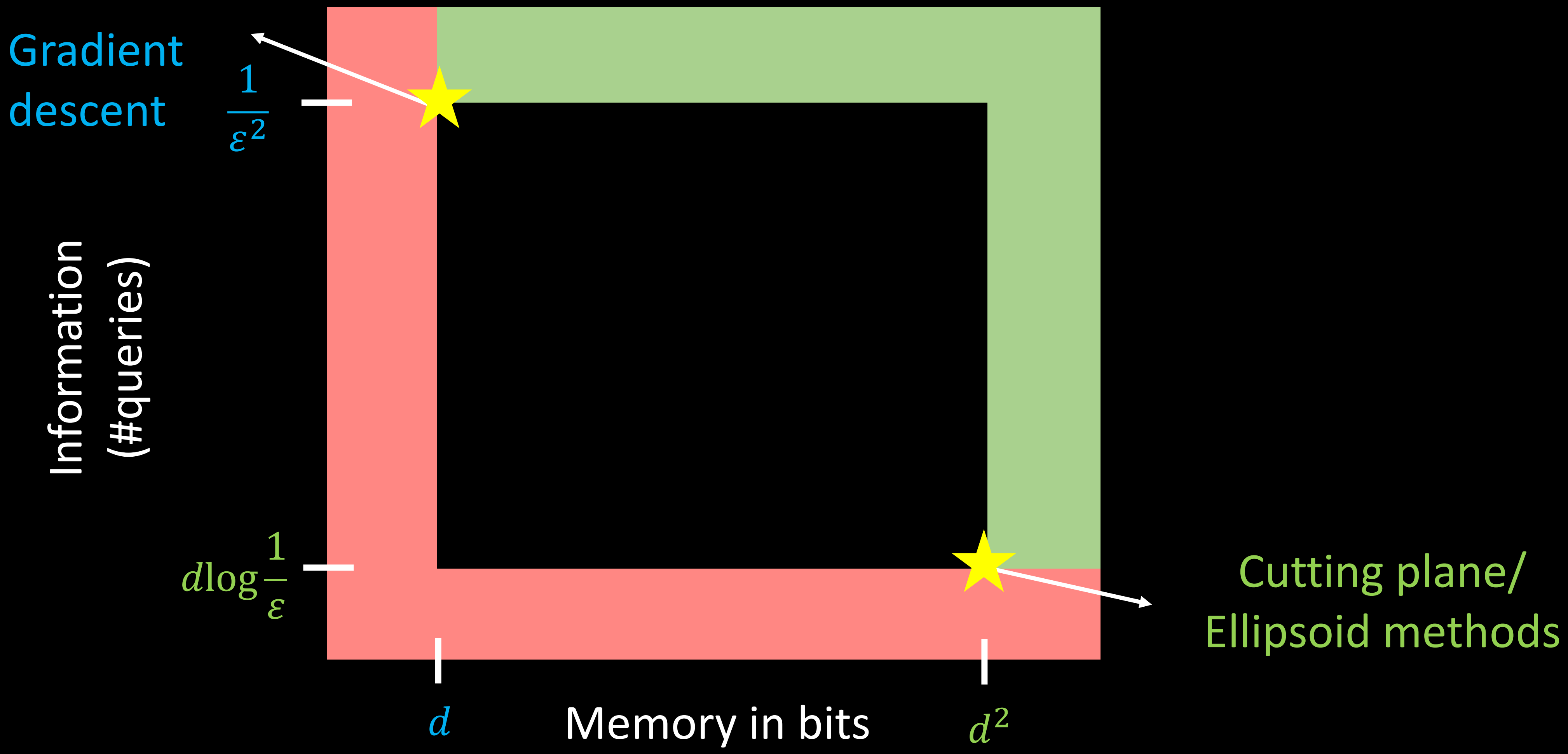
Shamir’14,
Steinhardt-Valiant-Wager’16,
Raz’17,
Moshkovitz-Moshkovitz’17
Kol-Raz-Tal’17,
Moshkovitz-Moshkovitz’18,
Garg-Raz-Tal’18,
Beame-Oveis Gharan-Yang’18,
Garg-Raz-Tal’19,
Raz-Zhan’20,
Gonen-Lovett-Moshkovitz’20,
Garg-Kothari-Raz’20

Memory bounds for continuous optimization


Gradient
descent
$\frac{1}{\varepsilon^2}$
$d \log \frac{1}{\varepsilon}$
Information
(#queries)
?
d
d^2
Memory (in bits)
Cutting
plane/
Ellipsoid
methods

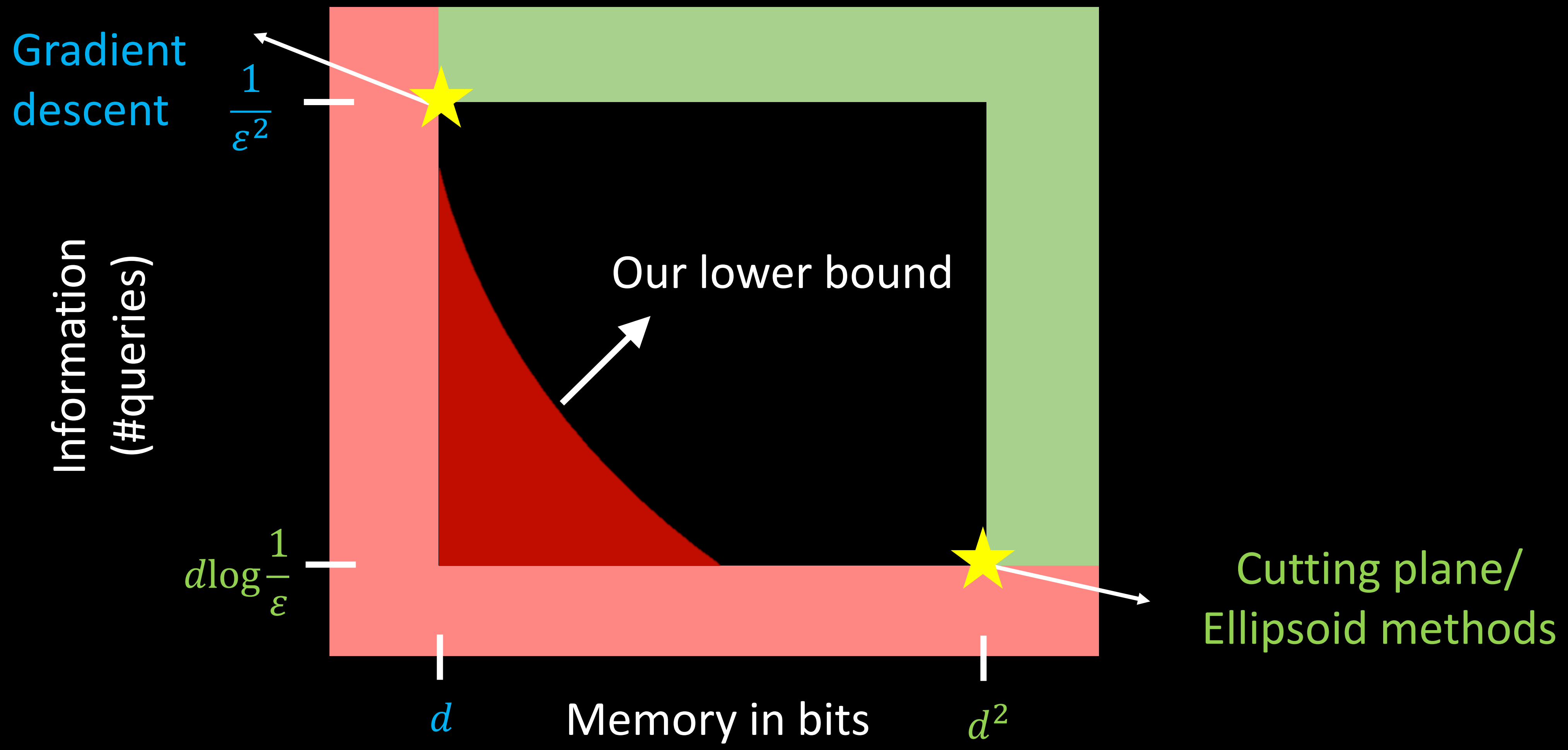
Theorem [Marsden, Sharan, Sidford, Valiant]:

For $\epsilon \geq \frac{1}{\mathrm{poly}(d)}$ and $\delta \in [0,0.25]$, any (randomized) algorithm with memory $\boldsymbol{d^{1.25-\delta}}$ requires at least $\boldsymbol{d^{1+1.33\delta}}$ first-order queries to find ϵ-optimal point.



Theorem [Marsden, Sharan, Sidford, Valiant]:

For $\epsilon \geq \frac{1}{\text{poly}(d)}$ and $\delta \in [0,0.25]$, any (randomized) algorithm with memory $\boldsymbol{d^{1.25-\delta}}$ requires at least $\boldsymbol{d^{1+1.33\delta}}$ first-order queries to find ϵ-optimal point.



High-level proof

Step one

Construct a distribution over functions that seems hard to optimize with limited memory

Step two

Relate optimizing these functions to winning a communication game

Step three

For the communication game, prove a memory/query tradeoff

Hard distribution over functions

Step one

Construct a distribution over functions that seems hard to optimize with limited memory

$$F_{h,A,\eta,\rho}(x) = \max\{\eta\|Ax\|_\infty - \rho, h(x)\}$$

$A \sim \mathrm{Unif}(\{\pm 1\}^{\frac{d}{2}\times d})$ $\quad h(x) = \max_{i\in[N]} v_i^T x - i\gamma$ (variant of Nemirovski function)

To receive first order information about h, must make query which is reasonably orthogonal to A

Nemirovski property: To continue receiving new or informative subgradients, queries must be robustly linearly independent

From optimization to winning a game

Step two

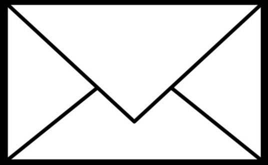
Relate optimizing these functions to winning a communication game

$$F_{h,A,\eta,\rho}(x) = \max\{\eta\|Ax\|_\infty - \rho, h(x)\}$$

$h(x) = \max_{i \in [N]} v_i^T x - i\gamma$ (variant of the Nemirovski function)

Relating optimizing $F_{h,A}(x)$ to winning an **Orthogonal Vector Game**

The Orthogonal Vector Game



M-bit message

Player

A

Random matrix

$A \sim \mathrm{Unif}(\{\pm 1\}^{\frac{d}{2} \times d})$

Game oracle

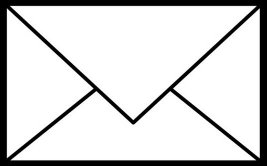
To win, find $y_1, y_2, \ldots, y_k$ which are roughly orthogonal* to A

d: dimension k: #vectors to be returned m: #oracle queries M: size of message (in bits)

The Orthogonal Vector Game

Query x_1

$g_1 \in \partial \|Ax_1\|_\infty$



M-bit message

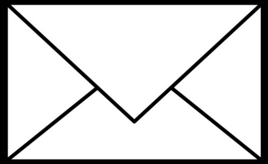
Player

Game oracle

To win, find $y_1, y_2, \dots, y_k$ which are roughly orthogonal* to A

d: dimension k: #vectors to be returned m: #oracle queries M: size of message (in bits)

The Orthogonal Vector Game



(x_1, g_1)

M-bit message

Player

Game oracle

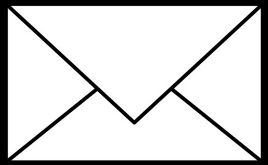
To win, find $y_1, y_2, \ldots, y_k$ which are roughly orthogonal* to A

d: dimension k: #vectors to be returned m: #oracle queries M: size of message (in bits)

The Orthogonal Vector Game

Query x_2

$g_2 \in \partial \|Ax_2\|_\infty$



M-bit message

Player

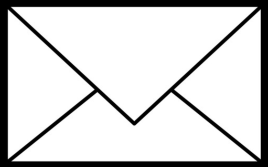
(x_1, g_1)

Game oracle

To win, find $y_1, y_2, \ldots, y_k$ which are roughly orthogonal* to A

d: dimension k: #vectors to be returned m: #oracle queries M: size of message (in bits)

The Orthogonal Vector Game



M-bit message

Player

(x_1, g_1)
(x_2, g_2)

Game oracle

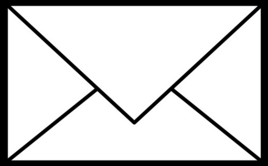
To win, find $y_1, y_2, \dots, y_k$ which are roughly orthogonal* to A

d: dimension k: #vectors to be returned m: #oracle queries M: size of message (in bits)

The Orthogonal Vector Game

Query x_m

$g_m \in \partial \|Ax_m\|_\infty$



M-bit message

Player

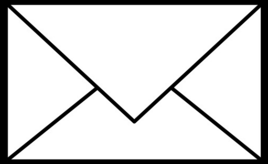
(x_1, g_1)
(x_2, g_2)
.
.
.

Game oracle

To win, find $y_1, y_2, \dots, y_k$ which are roughly orthogonal* to A

d: dimension k: #vectors to be returned m: #oracle queries M: size of message (in bits)

The Orthogonal Vector Game



M-bit message

Player

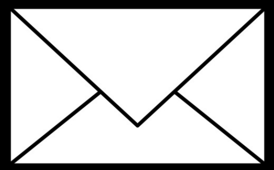
(x_1, g_1)
(x_2, g_2)
.
.
.
(x_m, g_m)

Game oracle

To win, find $y_1, y_2, \ldots, y_k$ which are roughly orthogonal* to A

d: dimension k: #vectors to be returned m: #oracle queries M: size of message (in bits)

The Orthogonal Vector Game



M-bit message

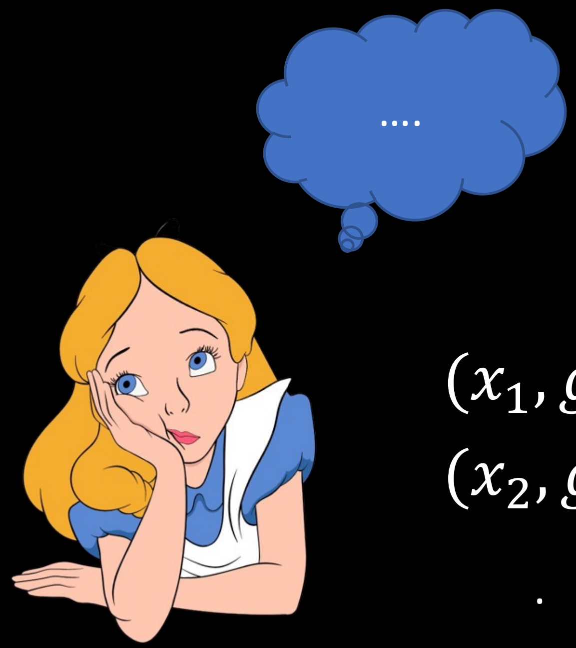
Player

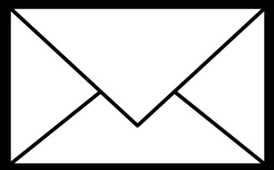
(x_1, g_1)
(x_2, g_2)
$\vdots$
(x_m, g_m)

To win, find $y_1, y_2, \ldots, y_k$ which are roughly orthogonal* to A

d: dimension k: #vectors to be returned m: #oracle queries M: size of message (in bits)

The Orthogonal Vector Game





M-bit message

Player

(x_1, g_1)
(x_2, g_2)
.
.
.
(x_m, g_m)

To win, find $y_1, y_2, \ldots, y_k$ which are roughly orthogonal* to A

d: dimension k: #vectors to be returned m: #oracle queries M: size of message (in bits)

The Orthogonal Vector Game

$y_1, y_2, \dots, y_k$

Player

To win, $y_1, y_2, \dots, y_k$ must be:

- Roughly orthogonal to A: $\|Ay_i\|_\infty \le 1/d^4$
- Robustly linearly independent $\left\|\mathrm{Proj}_{\mathrm{span}(y_1,\dots,y_{i-1})}(y_i)\right\| \le 1 - 1/d^2$

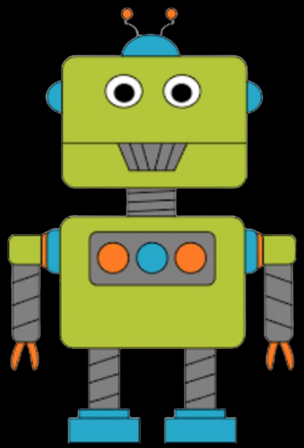
To win, find $y_1, y_2, \dots, y_k$ which are roughly orthogonal* to A

d: dimension k: #vectors to be returned m: #oracle queries M: size of message (in bits)

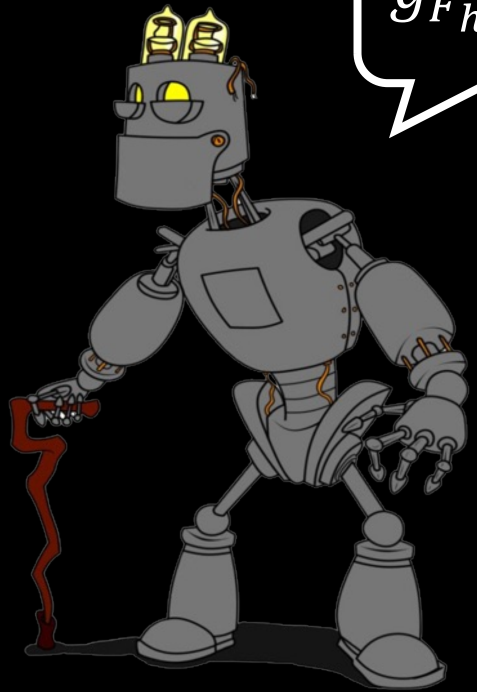
From Optimization to winning the Game

Query x

$g_{F_{h,A}}(x)$



Optimization algorithm

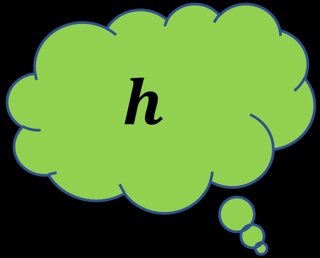


Optimization oracle

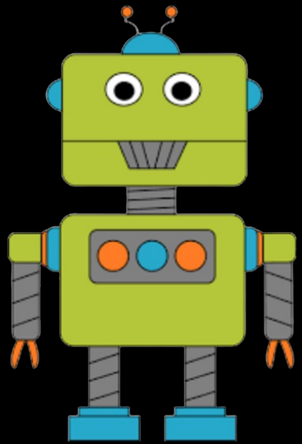
M-bit memory state

$$F_{h,A}(x) = \max\{\eta\|Ax\|_\infty - \rho, h(x)\}$$

From Optimization to winning the Game



Generates Nemirovski function h
Wants to optimize $F_{h,A}$

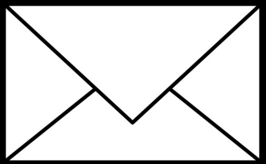


Optimization algorithm

A

Random matrix

Game oracle



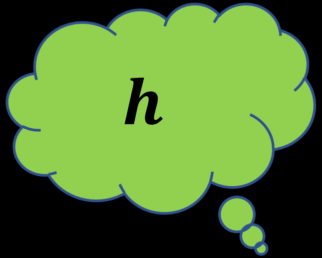
M-bit memory state

$$F_{h,A}(x) = \max\{\eta\|Ax\|_\infty - \rho, h(x)\}$$

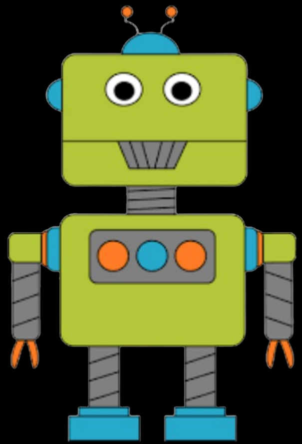
From Optimization to winning the Game

Query x_1

$g_1 \in \partial \|Ax_1\|_\infty$

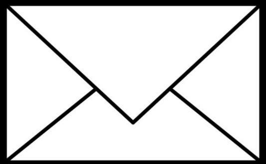


Generates Nemirovski function h
Wants to optimize $F_{h,\boldsymbol{A}}$



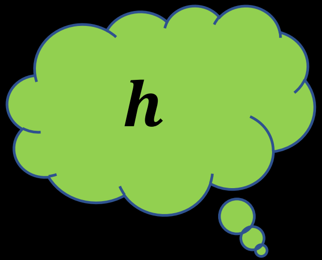
Optimization algorithm

Game oracle



M-bit memory state

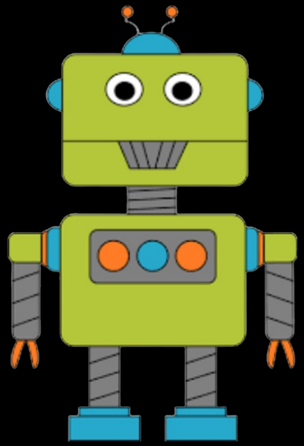
From Optimization to winning the Game



Query x_1

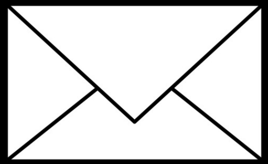
$g_{F_{h,A}}(x_1)$

Generates Nemirovski function h
Wants to optimize $F_{h,A}$



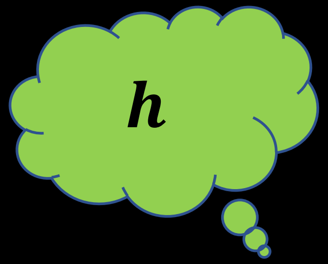
Optimization algorithm

Optimization oracle



M-bit memory state

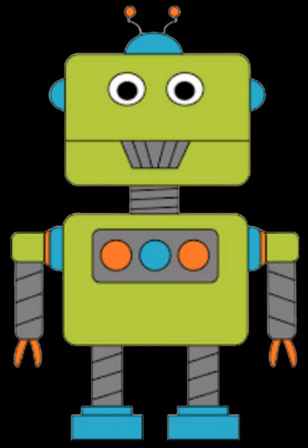
From Optimization to winning the Game



Query x_2

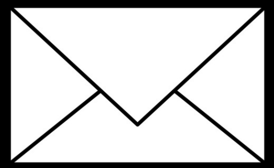
$g_2 \in \partial\|Ax_2\|_\infty$

Generates Nemirovski function h
Wants to optimize $F_{h,\boldsymbol{A}}$



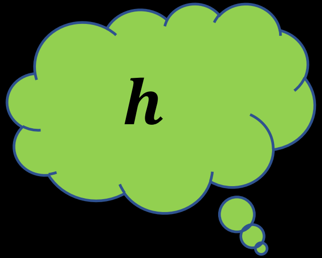
Optimization algorithm

Game oracle



M-bit memory state

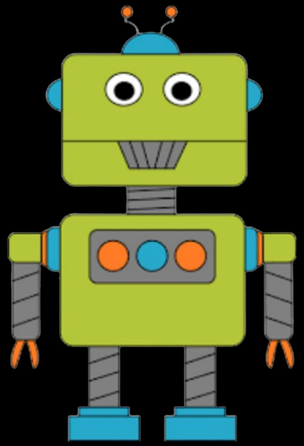
From Optimization to winning the Game



Query x_2

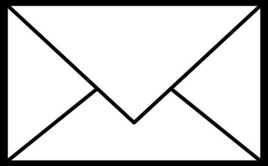
$g_{F_{h,A}}(x_2)$

Generates Nemirovski function h
Wants to optimize $F_{h,A}$



Optimization algorithm

Optimization oracle



M-bit memory state

Memory/Query tradeoffs for the Game

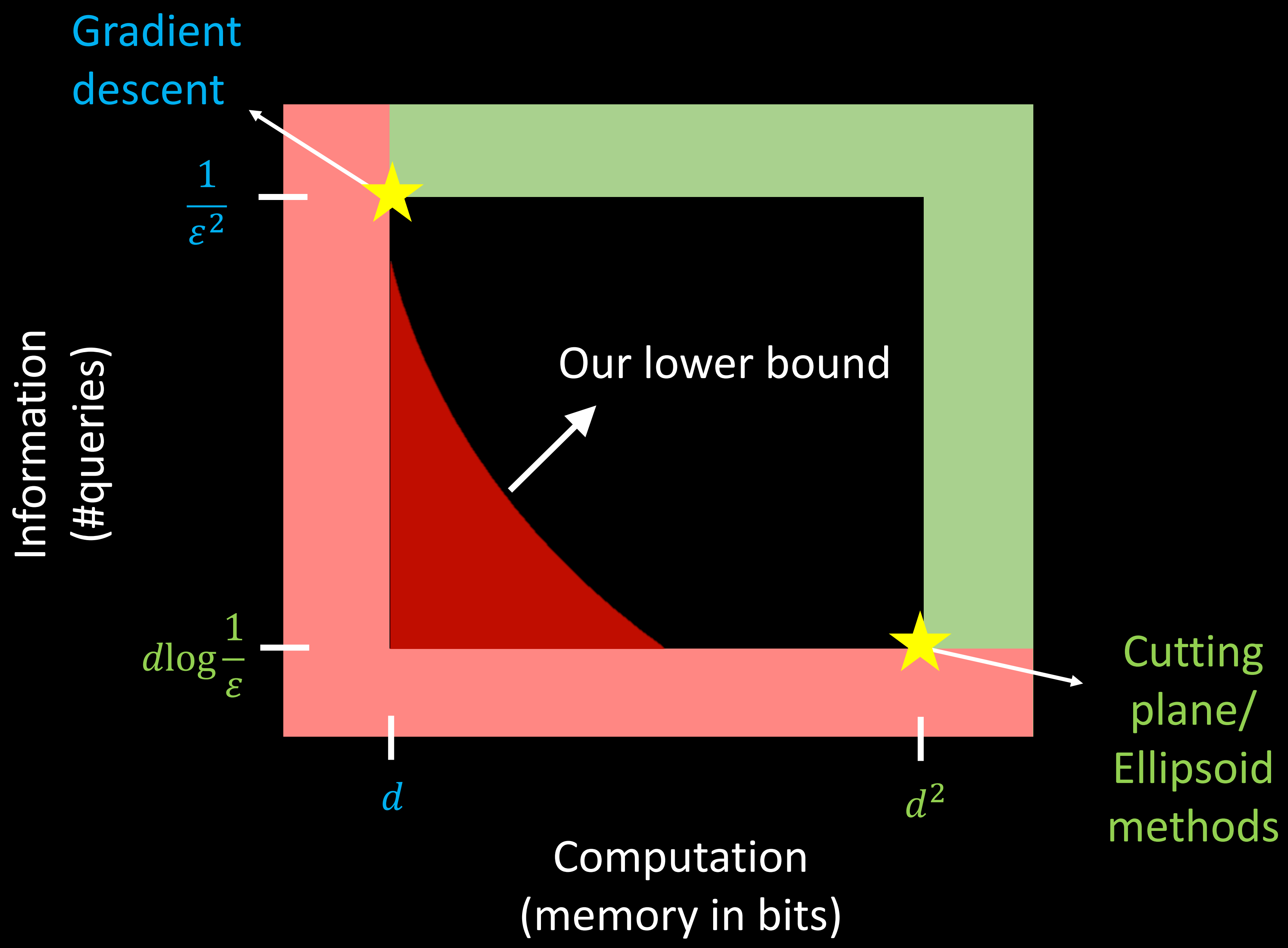
Step three

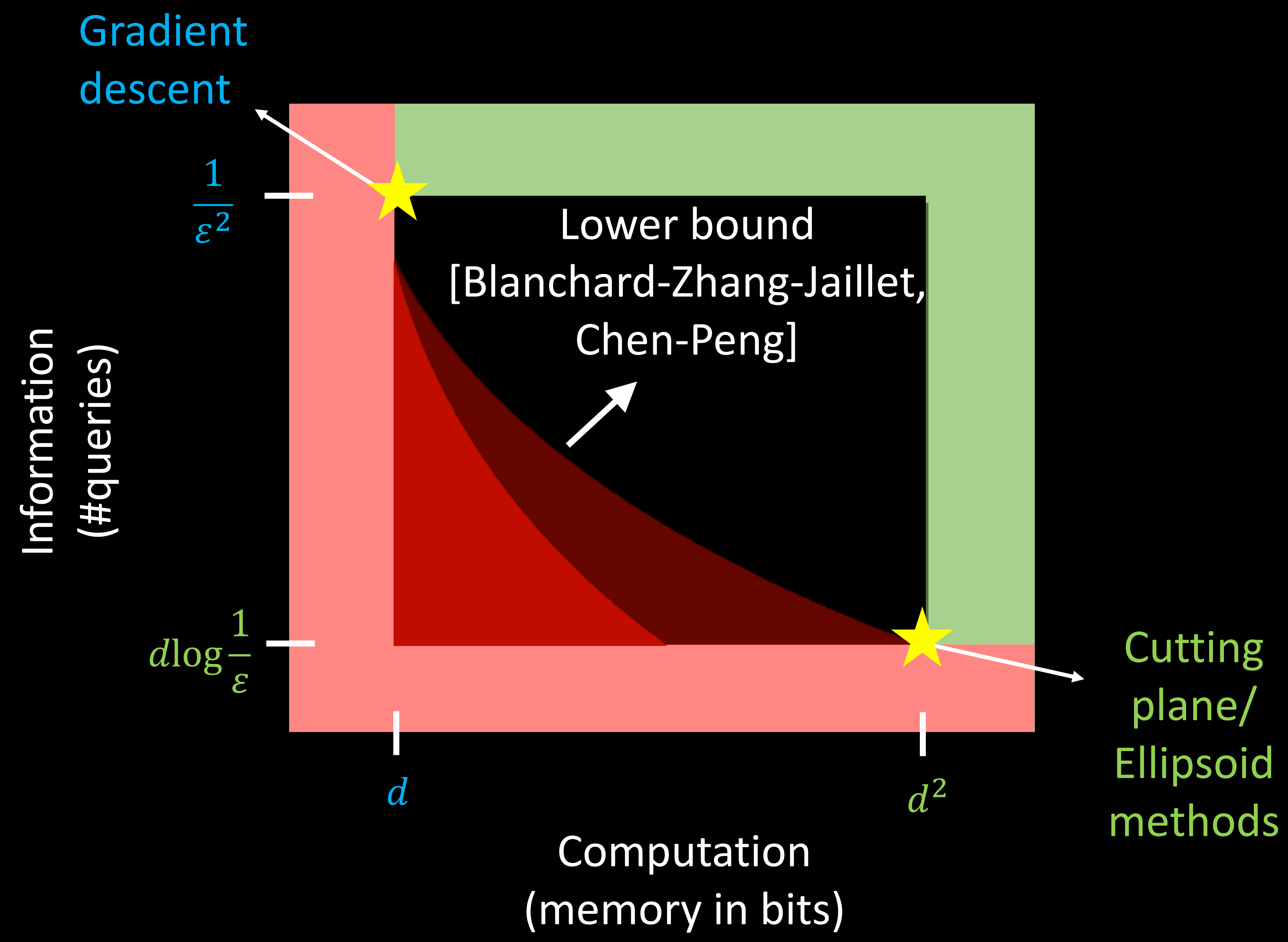
For the communication game, prove a memory/query tradeoff

If available memory $< kd$, then Player must make $\approx d$ queries

To win, find $y_1, y_2, \ldots, y_k$ which are roughly orthogonal* to A

d: dimension k: #vectors to be returned m: #oracle queries M: size of message (in bits)


Gradient descent
$\frac{1}{\varepsilon^2}$
Our lower bound
Information (#queries)
$d\log\frac{1}{\varepsilon}$
Cutting plane/ Ellipsoid methods
d
d^2
Computation (memory in bits)

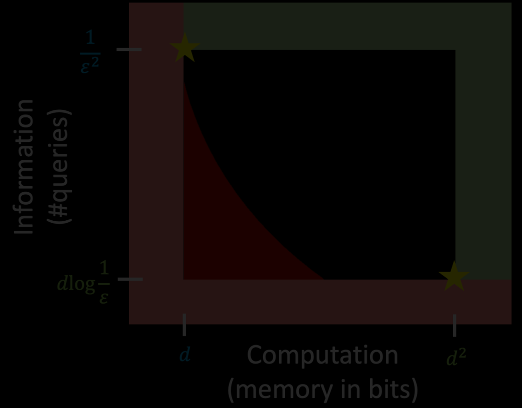

Gradient descent
$\frac{1}{\varepsilon^2}$
Lower bound [Blanchard-Zhang-Jaillet, Chen-Peng]
Information (#queries)
$d \log \frac{1}{\varepsilon}$
Cutting plane/ Ellipsoid methods
d
d^2
Computation (memory in bits)

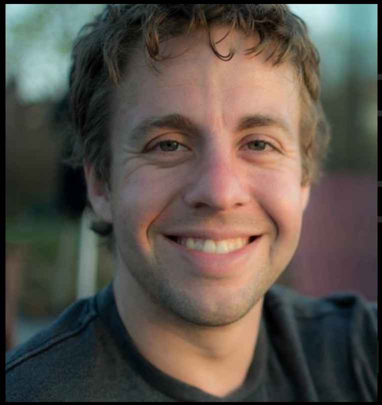
Open Questions

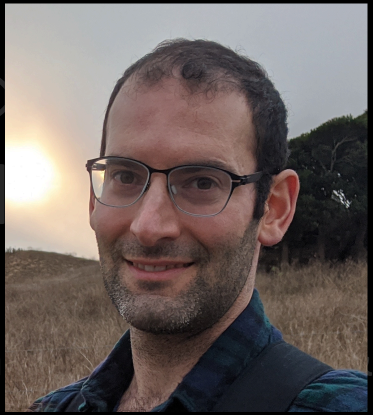
- Randomized algorithms, for poly-small ϵ?
- What happens for **smooth functions**?
- Can you improve on the $\text{poly}(1/\epsilon)$ rate of gradient descent for **super-poly small ϵ**?

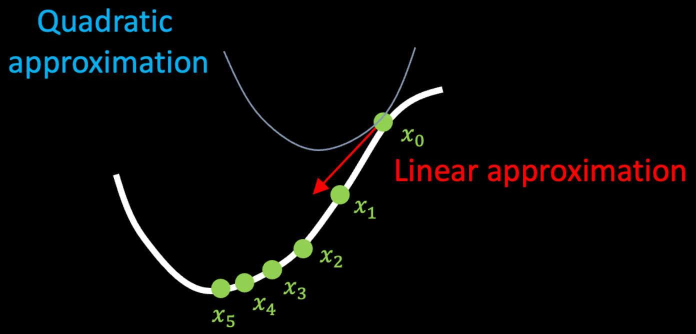
Conjecture: Cannot improve gradient descent's convergence rate without using quadratic memory.

[This talk] Memory Dichotomy Hypothesis: It is not possible to significantly improve on the convergence rate of known memory efficient techniques without using significantly more memory.

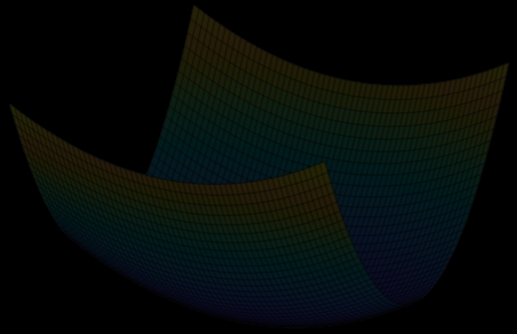








1^{st} order vs. 2^{nd} order methods



Lower bounds: Convex optimization with stochastic gradient oracle

(with Aaron Sidford & Greg Valiant)

Upper bounds: Better convergence with small memory

(with Jon Kelner, Annie Marsden, Aaron Sidford, Greg Valiant, Honglin Yuan)

Memory-Sample Tradeoffs for Linear Regression with Small Error
Vatsal Sharan, Aaron Sidford, Gregory Valiant, 2019

Stochastic optimization

In many modern ML settings, we work with stochastic gradients $g(x)$:

$$\mathrm{E}[g(x)] = \nabla F(x)$$

min. $F(x)$

$x \in R^d$: $||x|| \leq 1$

If $F(x)$ is expected loss with respect to data points sampled from some distribution, we can find stochastic gradient using a randomly sampled labelled datapoint.

What is the tradeoff between available memory and number of samples needed to optimize?

Linear model: Data vs. Memory?

$$\begin{bmatrix} & & \\ & \boldsymbol{A} & \\ & & \end{bmatrix} \begin{bmatrix} \\ \boldsymbol{x} \\ \\ \end{bmatrix} = \begin{bmatrix} \\ \boldsymbol{b} \\ \\ \end{bmatrix}$$

Find x

$$\boldsymbol{x}, \boldsymbol{a_i} \in R^d$$
$$b_i \in R$$

$$\underbrace{\begin{bmatrix} 3 & & 5 \end{bmatrix}}_{\boldsymbol{a}_1} \begin{bmatrix} \\ \boldsymbol{x} \\ \\ \end{bmatrix} = \underbrace{\begin{bmatrix} 4 \end{bmatrix}}_{b_1}$$

$$< \boldsymbol{a}_1, \boldsymbol{x} > = b_1$$

Find x

$\boldsymbol{x}, \boldsymbol{a_i} \in R^d$

$b_i \in R$

$$\begin{bmatrix} -2 & 2 \end{bmatrix} \begin{bmatrix} \\ \boldsymbol{x} \\ \\ \end{bmatrix} = \begin{bmatrix} 8 \end{bmatrix}$$

$\boldsymbol{a_2}$ $\quad$ b_2

$$< \boldsymbol{a_2}, \boldsymbol{x} > = b_2$$

Find x

$\boldsymbol{x}, \boldsymbol{a_i} \in R^d$

$b_i \in R$

$$\underbrace{\begin{bmatrix} -7 & & -2 \end{bmatrix}}_{\boldsymbol{a}_3} \begin{bmatrix} \\ \boldsymbol{x} \\ \\ \end{bmatrix} = \underbrace{\begin{bmatrix} 10 \end{bmatrix}}_{b_3}$$

$$<\boldsymbol{a}_3, \boldsymbol{x}> = b_3$$

Find x

$$\boldsymbol{x}, \boldsymbol{a_i} \in R^d$$
$$b_i \in R$$

$$\underbrace{\begin{bmatrix} 4 & & -1 \end{bmatrix}}_{\boldsymbol{a}_4} \begin{bmatrix} \boldsymbol{x} \\ \\ \\ \end{bmatrix} = \underbrace{\begin{bmatrix} -10 \end{bmatrix}}_{b_4}$$

$$< \boldsymbol{a}_4, \boldsymbol{x} > = b_4$$

Find x

$$\boldsymbol{x}, \boldsymbol{a_i} \in R^d$$
$$b_i \in R$$

$$\underbrace{\begin{bmatrix} -8 & -6 \end{bmatrix}}_{\boldsymbol{a_5}} \begin{bmatrix} \boldsymbol{x} \\ \\ \end{bmatrix} = \underbrace{\begin{bmatrix} 4 \end{bmatrix}}_{b_5}$$

$$<\boldsymbol{a_5}, \boldsymbol{x}> = b_5$$

Find x

$\boldsymbol{x}, \boldsymbol{a_i} \in R^d$

$b_i \in R$

$$\begin{bmatrix} -8 & -6 \end{bmatrix} \begin{bmatrix} -2 \\ 2 \end{bmatrix} = \begin{bmatrix} 4 \end{bmatrix}$$

$\boldsymbol{a}_5$ b_5

1. Memory = #bits
2. Samples drawn from Gaussian

$$<\boldsymbol{a}_5, \boldsymbol{x}> = b_5$$

What can you do?

Gaussian Elimination

$$\begin{bmatrix} 4 & -1 \\ -8 & -6 \end{bmatrix} \begin{bmatrix} \boldsymbol{x} \end{bmatrix} = \begin{bmatrix} -10 \\ 4 \end{bmatrix}$$

What can you do?

Gaussian Elimination

$$\begin{bmatrix} 1 & -1/4 \\ -8 & -6 \end{bmatrix} \begin{bmatrix} x \end{bmatrix} = \begin{bmatrix} -5/2 \\ 4 \end{bmatrix}$$

What can you do?

Gaussian Elimination

$$\begin{bmatrix} 1 & -1/4 \\ 0 & -8 \end{bmatrix} \begin{bmatrix} \boldsymbol{x} \end{bmatrix} = \begin{bmatrix} -5/2 \\ -16 \end{bmatrix}$$

What can you do?

Gaussian Elimination

$$\begin{bmatrix} 1 & -1/4 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} \boldsymbol{x} \end{bmatrix} = \begin{bmatrix} -5/2 \\ 2 \end{bmatrix}$$

What can you do?

Gaussian Elimination

$$\begin{bmatrix} 1 & -1/4 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} -2 \\ 2 \end{bmatrix} = \begin{bmatrix} -5/2 \\ 2 \end{bmatrix}$$

d examples

$\approx d^2$ memory

What can you do?

Gaussian Elimination

$$\begin{bmatrix} 1 & -1/4 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} -2 \\ 2 \end{bmatrix} = \begin{bmatrix} -5/2 \\ 2 \end{bmatrix}$$

d examples

≈ d^2 memory

Gradient Descent

Initialize x_0 . At time i,

Get (a_i, b_i) . Update $x_i \rightarrow x_{i+1}$.

What can you do?

Gaussian Elimination

$$\begin{bmatrix} 1 & -1/4 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} -2 \\ 2 \end{bmatrix} = \begin{bmatrix} -5/2 \\ 2 \end{bmatrix}$$

d examples

≈ d^2 memory

Gradient Descent

Initialize x_0 . At time i,

Get (a_i, b_i) . Update $x_i \rightarrow x_{i+1}$.

$$x_0 = (-0.25, 0.98\,)$$

What can you do?

Gaussian Elimination

$$\begin{bmatrix} 1 & -1/4 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} -2 \\ 2 \end{bmatrix} = \begin{bmatrix} -5/2 \\ 2 \end{bmatrix}$$

d examples

$\approx$ d^2 memory

Gradient Descent

Initialize x_0 . At time i,

Get (a_i, b_i) . Update $x_i \rightarrow x_{i+1}$.

$$x_0 = (-0.25, 0.98\,)$$

What can you do?

Gaussian Elimination

$$\begin{bmatrix} 1 & -1/4 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} -2 \\ 2 \end{bmatrix} = \begin{bmatrix} -5/2 \\ 2 \end{bmatrix}$$

d examples

$\approx$ d^2 memory

Gradient Descent

Initialize x_0 . At time i,

Get (a_i, b_i) . Update $x_i \rightarrow x_{i+1}$.

$$x_1 = (-0.45, 0.74)$$

What can you do?

Gaussian Elimination

$$\begin{bmatrix} 1 & -1/4 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} -2 \\ 2 \end{bmatrix} = \begin{bmatrix} -5/2 \\ 2 \end{bmatrix}$$

d examples

$\approx$ d^2 memory

Gradient Descent

Initialize x_0 . At time i,

Get (a_i, b_i) . Update $x_i \rightarrow x_{i+1}$.

$$x_2 = (-0.74, 2.24)$$

What can you do?

Gaussian Elimination

$$\begin{bmatrix} 1 & -1/4 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} -2 \\ 2 \end{bmatrix} = \begin{bmatrix} -5/2 \\ 2 \end{bmatrix}$$

d examples

$\approx$ d[2] memory

Gradient Descent

Initialize x_0 . At time i,

Get (a_i, b_i) . Update $x_i \rightarrow x_{i+1}$.

$$x_3 = (-1.64, 2.70)$$

What can you do?

Gaussian Elimination

$$\begin{bmatrix} 1 & -1/4 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} -2 \\ 2 \end{bmatrix} = \begin{bmatrix} -5/2 \\ 2 \end{bmatrix}$$

d examples

≈ d^2 memory

Gradient Descent

Initialize x_0 . At time i,

Get (a_i, b_i) . Update $x_i \rightarrow x_{i+1}$.

$$x_4 = (-1.85, 2.74)$$

What can you do?

Gaussian Elimination

$$\begin{bmatrix} 1 & -1/4 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} -2 \\ 2 \end{bmatrix} = \begin{bmatrix} -5/2 \\ 2 \end{bmatrix}$$

d examples

$\approx$ d^2 memory

Gradient Descent

Initialize x_0 . At time i,

Get (a_i, b_i) . Update $x_i \rightarrow x_{i+1}$.

$$x_5 = (-2.27, 2.53)$$

What can you do?

Gaussian Elimination

$$\begin{bmatrix} 1 & -1/4 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} -2 \\ 2 \end{bmatrix} = \begin{bmatrix} -5/2 \\ 2 \end{bmatrix}$$

d examples

$\approx$ d^2 memory

Gradient Descent

Initialize x_0 . At time i,

Get (a_i, b_i) . Update $x_i \rightarrow x_{i+1}$.

$$x_6 = (-1.99, 2.52)$$

What can you do?

Gaussian Elimination

$$\begin{bmatrix} 1 & -1/4 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} -2 \\ 2 \end{bmatrix} = \begin{bmatrix} -5/2 \\ 2 \end{bmatrix}$$

d examples

≈ d² memory

Gradient Descent

Initialize x_0 . At time i,

Get (a_i, b_i) . Update $x_i \rightarrow x_{i+1}$.

$$x_7 = (-1.83, 2.47)$$

What can you do?

Gaussian Elimination

$$\begin{bmatrix} 1 & -1/4 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} -2 \\ 2 \end{bmatrix} = \begin{bmatrix} -5/2 \\ 2 \end{bmatrix}$$

d examples

≈ d^2 memory

Gradient Descent

Initialize x_0 . At time i,

Get (a_i, b_i) . Update $x_i \rightarrow x_{i+1}$.

$$x_8 = (-1.92, 2.48)$$

What can you do?

Gaussian Elimination

$$\begin{bmatrix} 1 & -1/4 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} -2 \\ 2 \end{bmatrix} = \begin{bmatrix} -5/2 \\ 2 \end{bmatrix}$$

d examples

≈ d² memory

Gradient Descent

Initialize x_0 . At time i,

Get (a_i, b_i) . Update $x_i \rightarrow x_{i+1}$.

$$x_9 = (-2.20, 2.17)$$

What can you do?

Gaussian Elimination

$$\begin{bmatrix} 1 & -1/4 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} -2 \\ 2 \end{bmatrix} = \begin{bmatrix} -5/2 \\ 2 \end{bmatrix}$$

d examples

$\approx$ d^2 memory

Gradient Descent

Initialize x_0 . At time i,

Get (a_i, b_i) . Update $x_i \rightarrow x_{i+1}$.

$$x_{10} = (-1.97, 2.08)$$

What can you do?

Gaussian Elimination

$$\begin{bmatrix} 1 & -1/4 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} -2 \\ 2 \end{bmatrix} = \begin{bmatrix} -5/2 \\ 2 \end{bmatrix}$$

d examples

≈ d^2 memory

Gradient Descent

Initialize x_0 . At time i,

Get (a_i, b_i) . Update $x_i \rightarrow x_{i+1}$.

$$x_{11} = (-2.02, 2.01)$$

What can you do?

Gaussian Elimination

$$\begin{bmatrix} 1 & -1/4 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} -2 \\ 2 \end{bmatrix} = \begin{bmatrix} -5/2 \\ 2 \end{bmatrix}$$

d examples

$\approx$ d^2 memory

Gradient Descent

Initialize x_0 . At time i,

Get (a_i, b_i) . Update $x_i \rightarrow x_{i+1}$.

$$x_{12} = (-2.01, 2.00\ )$$

What can you do?

Gaussian Elimination

$$\begin{bmatrix} 1 & -1/4 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} -2 \\ 2 \end{bmatrix} = \begin{bmatrix} -5/2 \\ 2 \end{bmatrix}$$

d examples

$\approx d^2$ memory

Gradient Descent

Initialize x_0 . At time i,

Get (a_i, b_i) . Update $x_i \rightarrow x_{i+1}$.

$$x_{12} = (-2.01, 2.00)$$

$O\left(d \log \frac{1}{\varepsilon}\right)$ examples

$\approx$ d memory

What can you do?

Gaussian Elimination

$$\begin{bmatrix} 1 & -1/4 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} -2 \\ 2 \end{bmatrix} = \begin{bmatrix} -5/2 \\ 2 \end{bmatrix}$$

d examples

$\approx d^2$ memory

Gradient Descent

Initialize x_0 . At time i,

Get (a_i, b_i) . Update $x_i \rightarrow x_{i+1}$.

$$x_{12} = (-2.01, 2.00)$$

$>$ d examples

$\approx$ d memory

Gaussian Elimination	Gradient Descent
d examples	> d examples
$\approx d^2$ memory	$\approx$ d memory

Gaussian Elimination

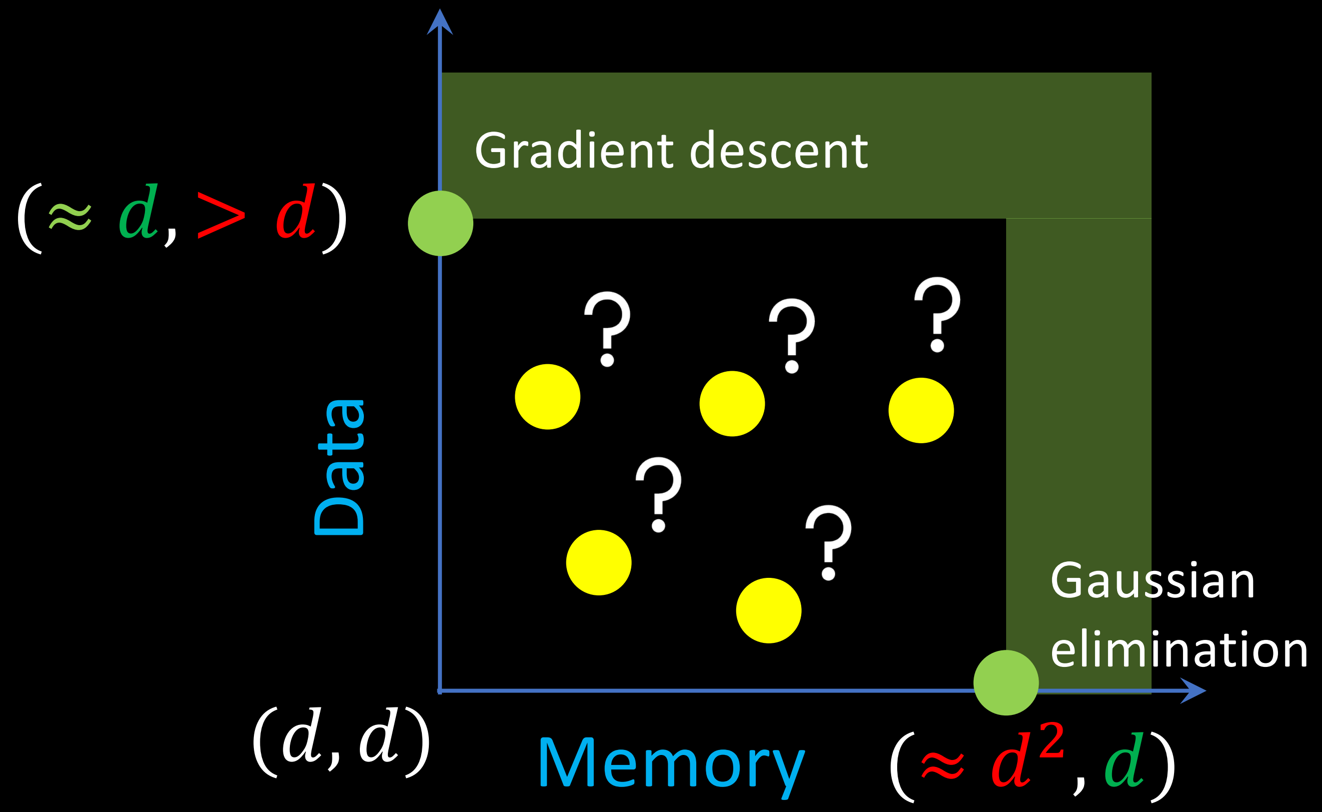
d examples

≈ d^2 memory

Gradient Descent

> d examples

≈ d memory



Gaussian Elimination

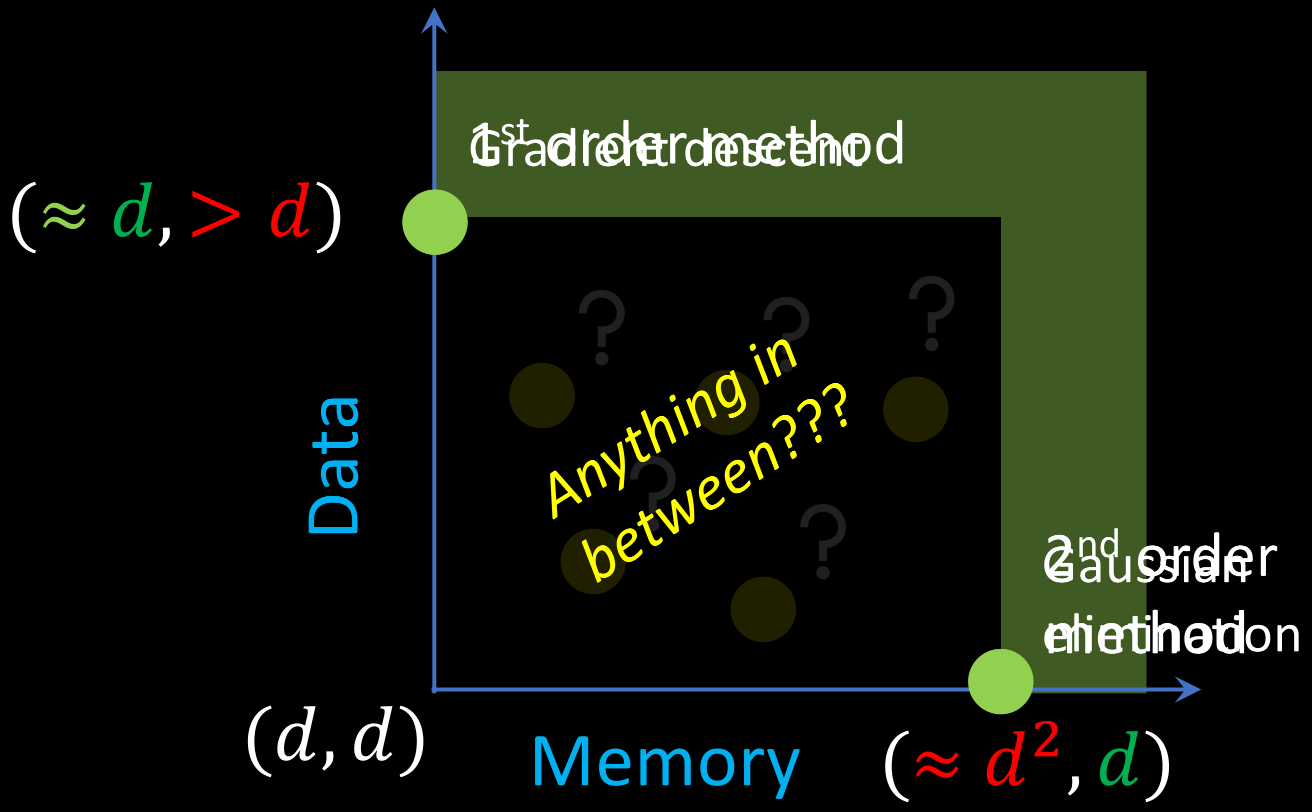
d examples

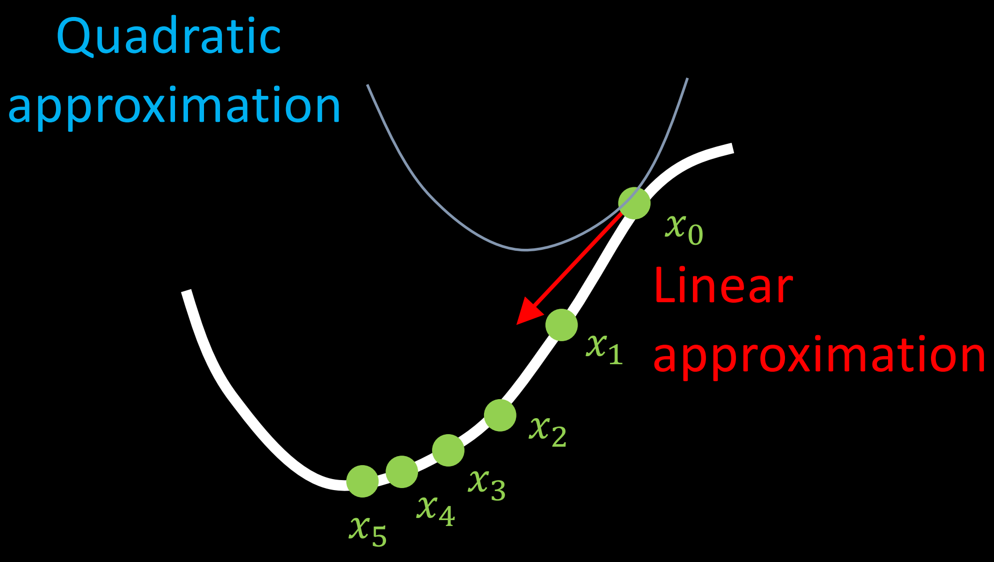
$\approx d^2$ memory

Gradient Descent

$>$ d examples

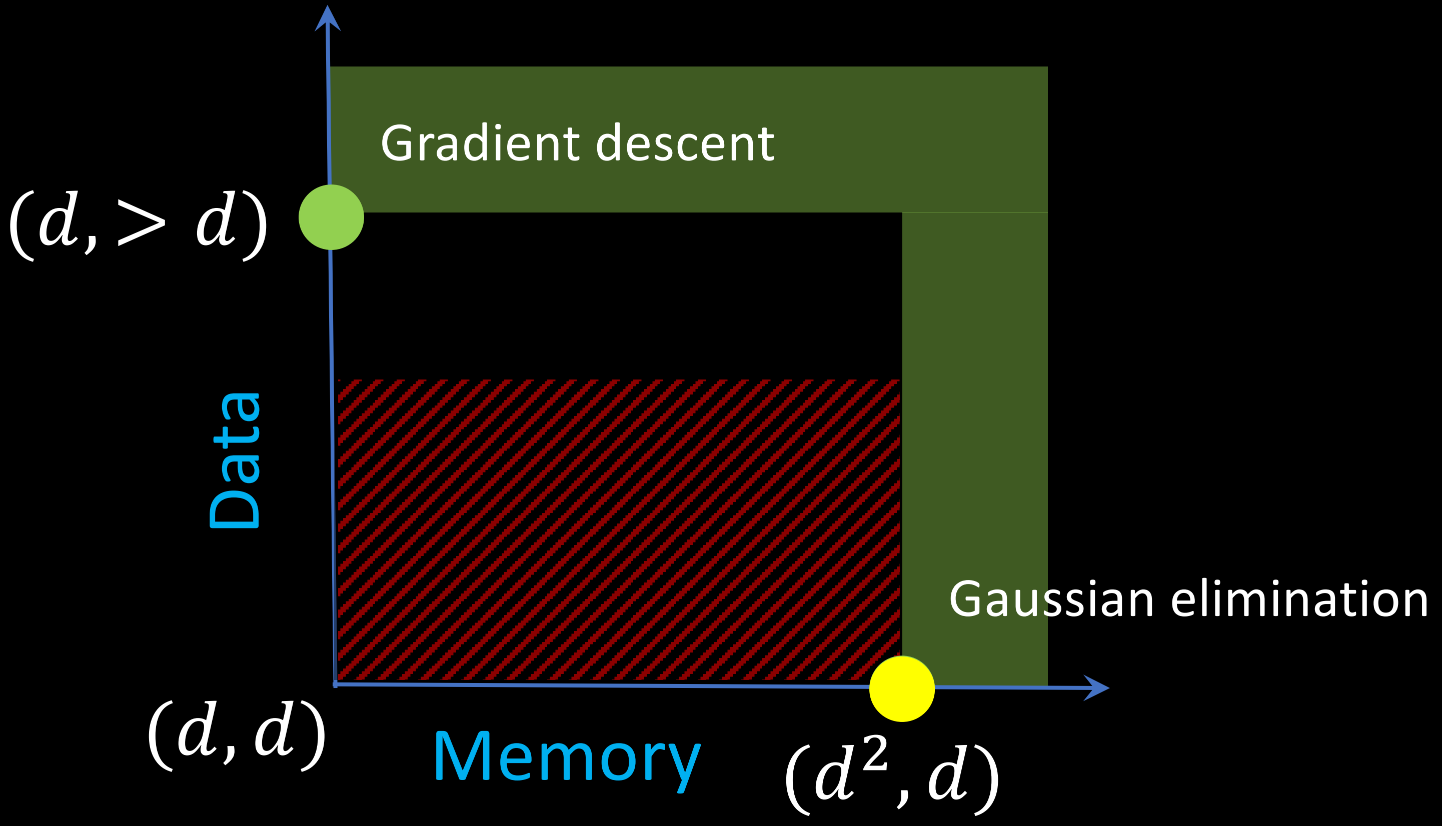
$\approx$ d memory





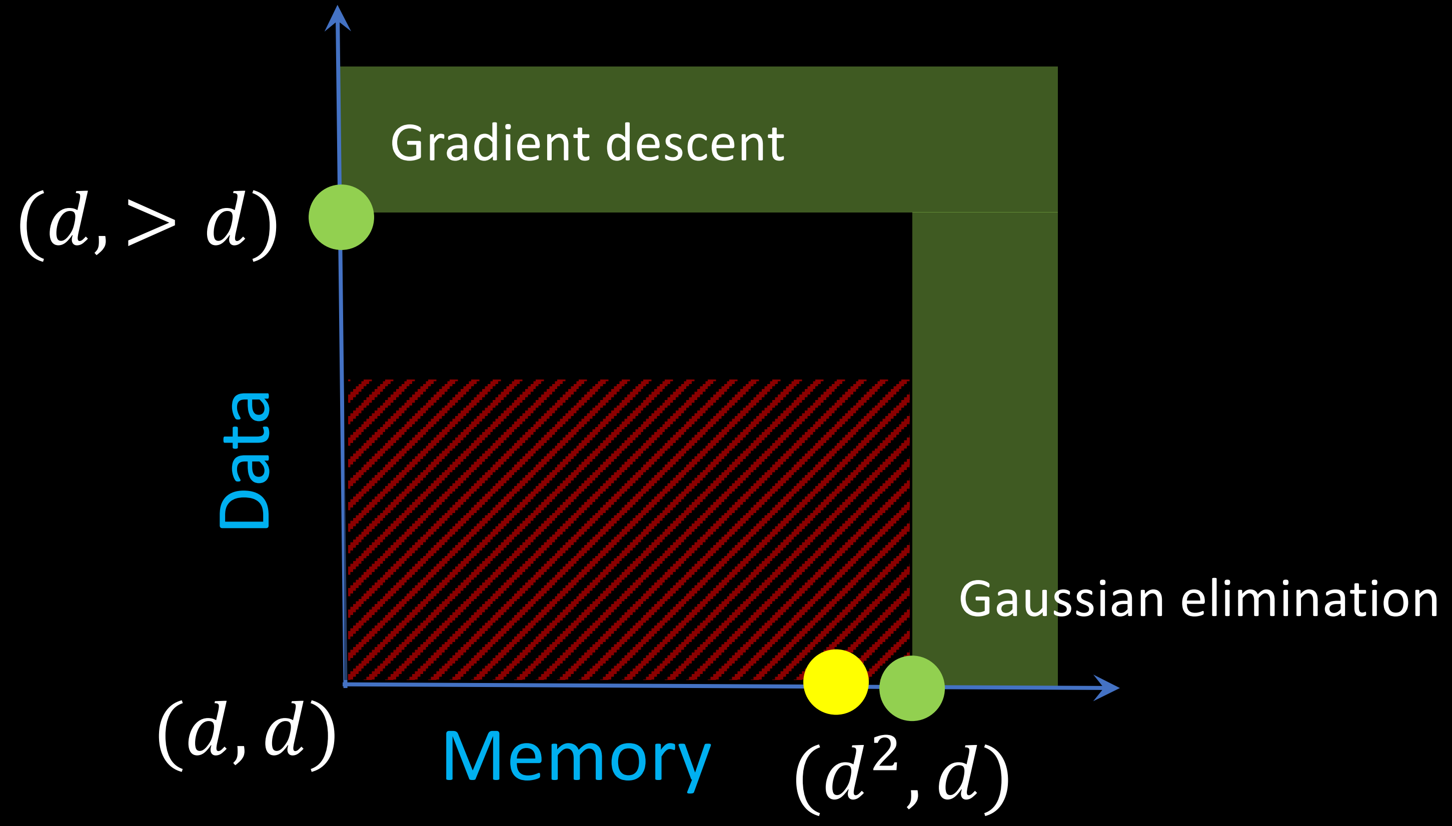
1st order vs. 2nd order methods

Informal Theorem[Sharan, Sidford, Valiant]:
Any sub-quadratic memory algorithm requires more data.

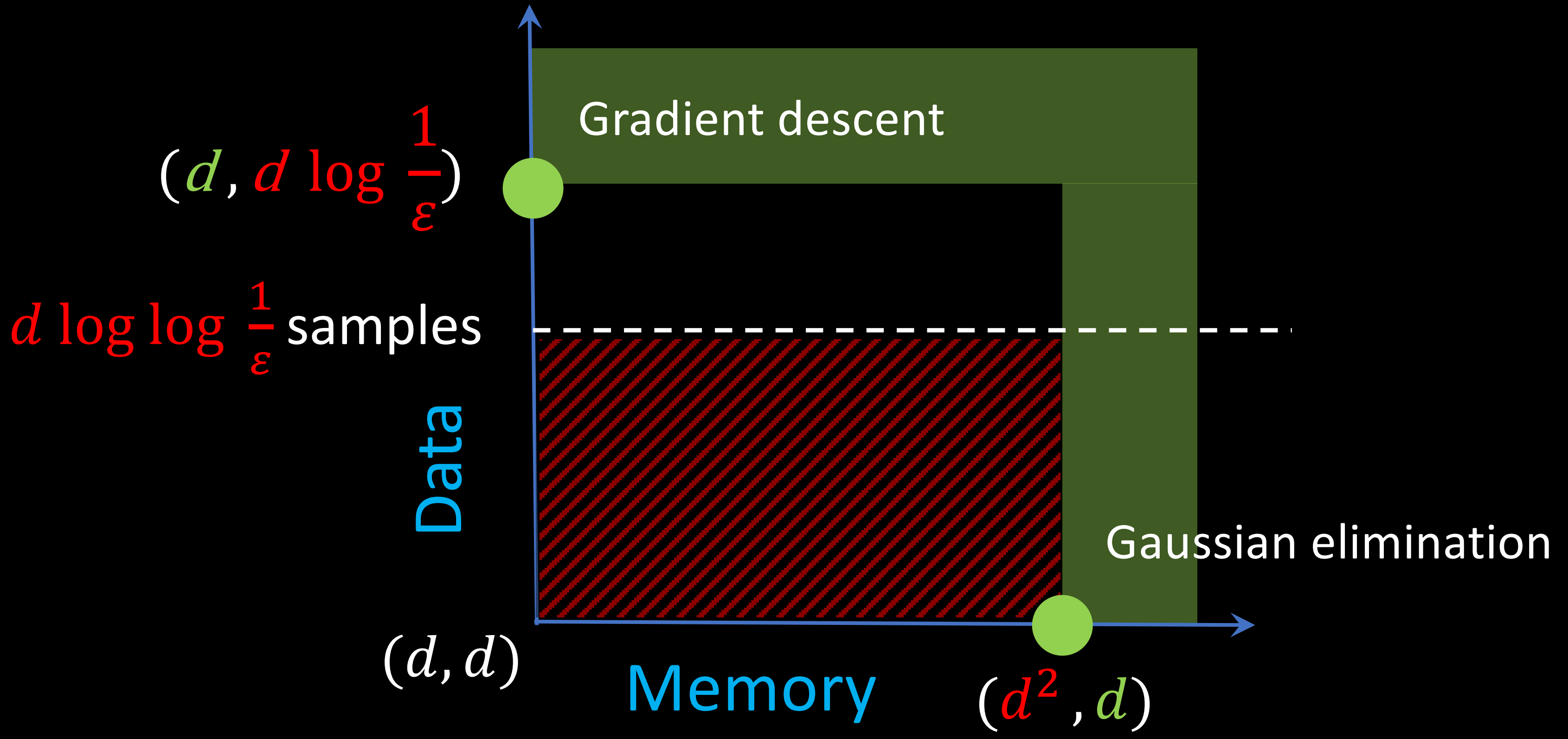


Informal Theorem[Sharan, Sidford, Valiant]:

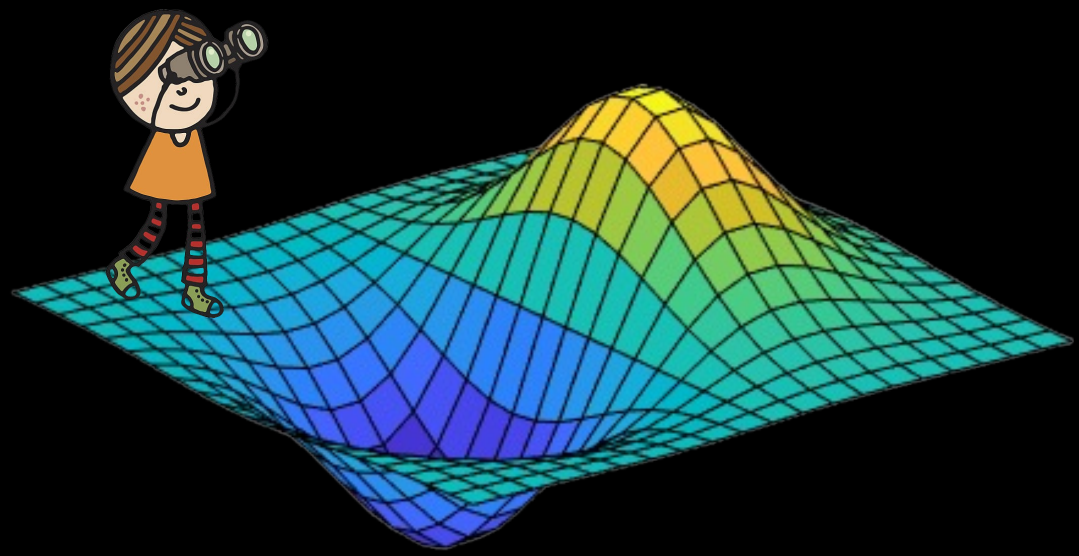
Any sub-quadratic memory algorithm requires more data.



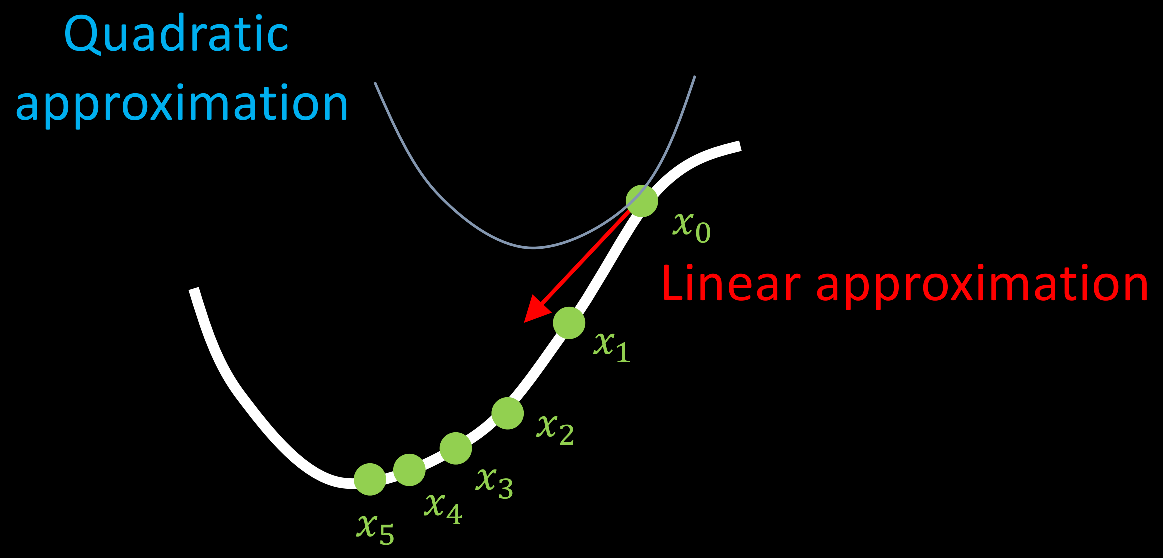
Informal Theorem[Sharan, Sidford, Valiant]:
Any sub-quadratic memory algorithm requires more data.



DISCUSSION



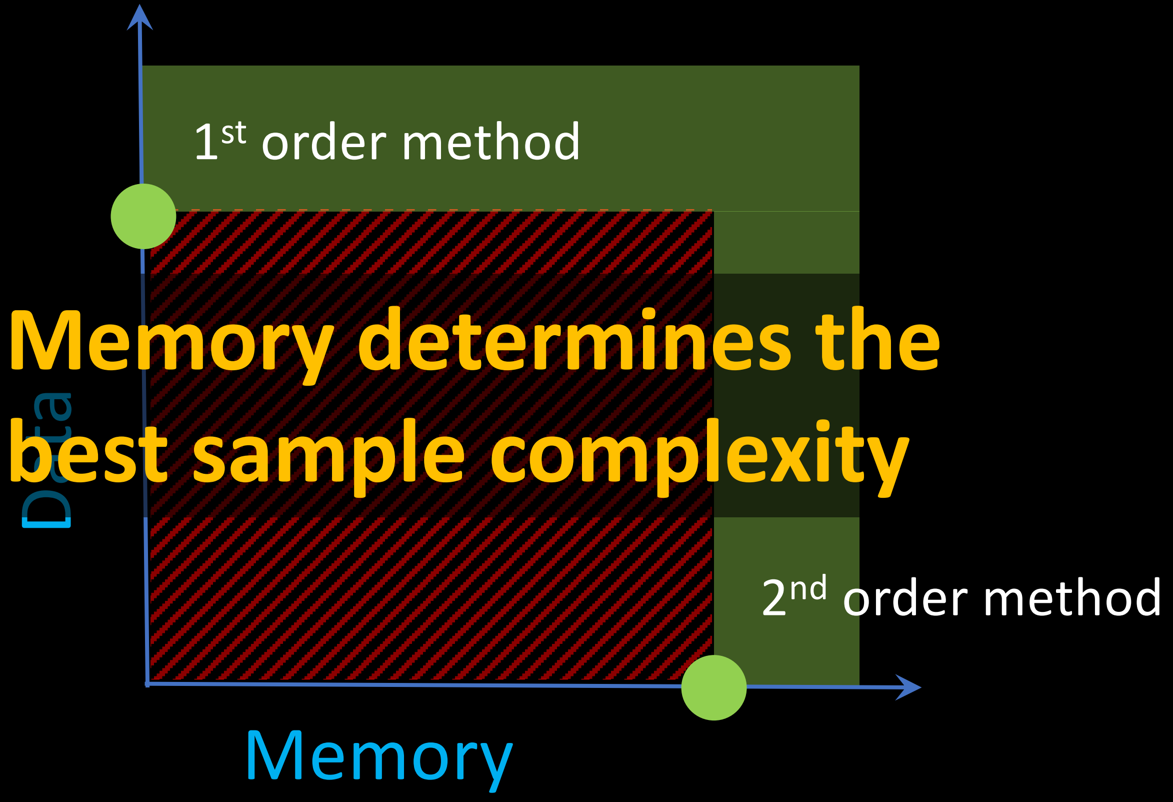
1.5th order method?



1st order vs. 2nd order methods

Our Conjecture:
Any algorithm that improves on convergence rate of best known "first-order" methods, requires quadratic memory.

1.5th order method?



Our Conjecture:
Any algorithm that improves on convergence rate of best known “first-order” methods, requires quadratic memory.

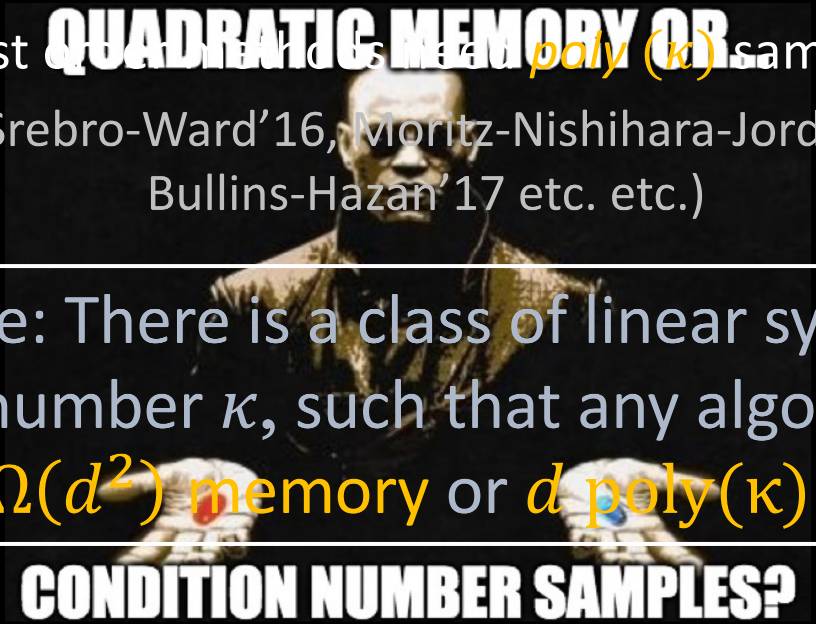
1.5th order method?

Our Conjecture:
Any algorithm that improves on convergence rate of best known “first-order” methods, requires quadratic memory.

Ill-conditioned distribution:

First order methods need *poly* (κ) samples

(e.g. Needell-Srebro-Ward’16, Moritz-Nishihara-Jordan’16, Agarwal-Bullins-Hazan’17 etc. etc.)

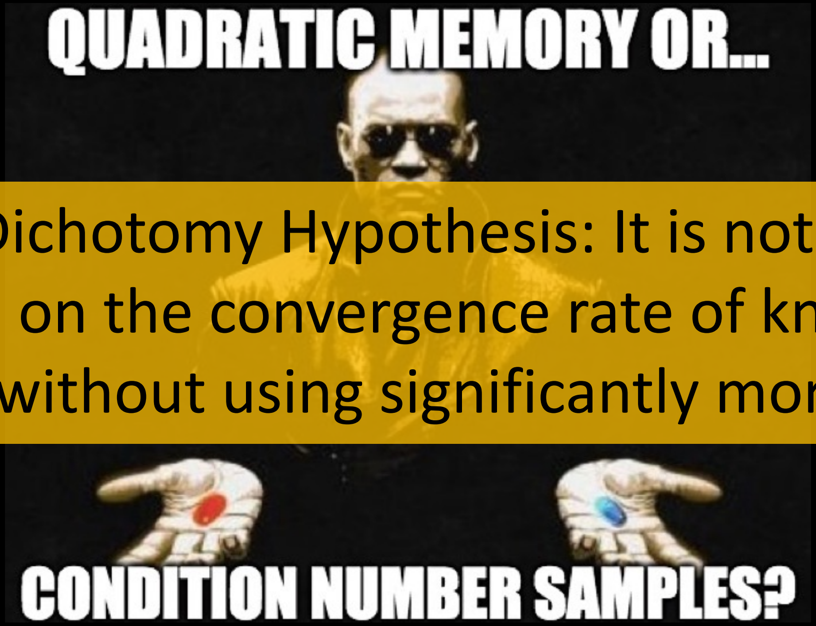


Conjecture: There is a class of linear systems with condition number κ, such that any algorithm either requires $\Omega(d^2)$ memory or $d\ \mathrm{poly}(\kappa)$ examples.

1.5th order method?

Our Conjecture:
Any algorithm that improves on convergence rate of best known "first-order" methods, requires quadratic memory.

Ill-conditioned distribution:



[This talk] Memory Dichotomy Hypothesis: It is not possible to significantly improve on the convergence rate of known memory efficient techniques without using significantly more memory.

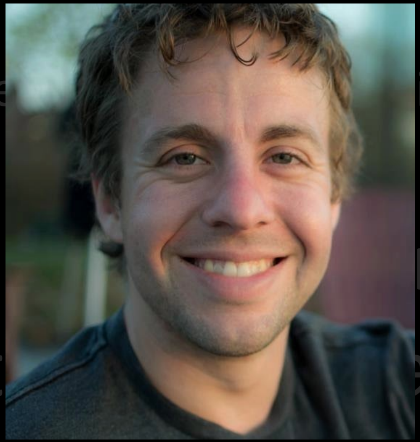
Broader question:
Understand the landscape of continuous **optimization** with **memory constraints**.

Big-Step-Little-Step: Efficient Gradient Methods for Objectives with Multiple Scales

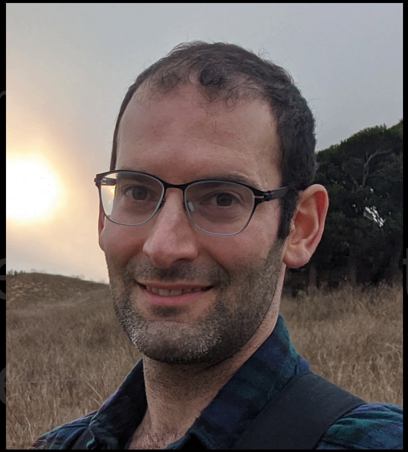
Jonathan Kelner, Annie Marsden, Vatsal Sharan, Aaron Sidford, Gregory Valiant, Honglin Yuan, 2022

Jon Kelner

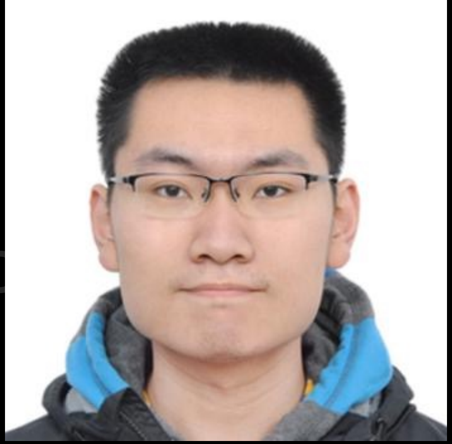
Annie Marsden



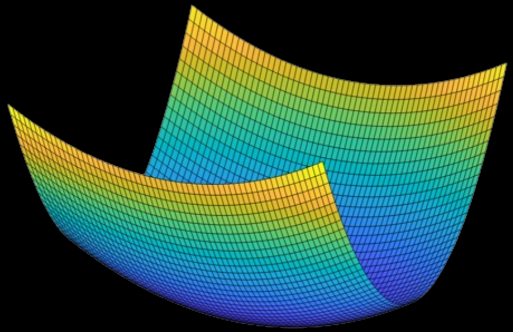
Aaron Sidford



Greg Valiant

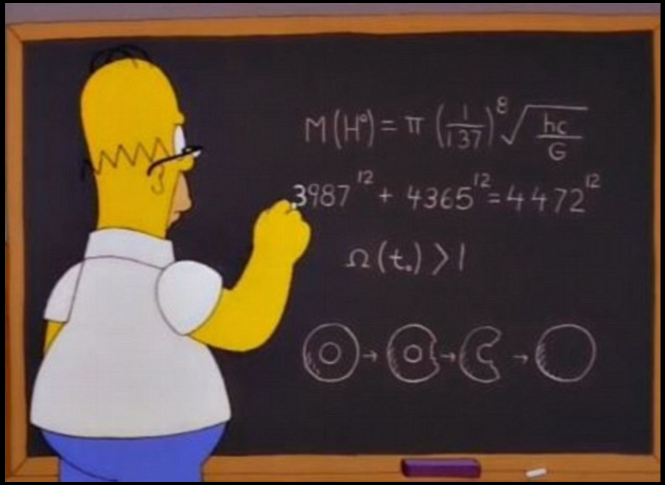


Honglin Yuan



Upper bounds: Better convergence with small memory

Using memory considerations to develop **more efficient optimization algorithms**



Memory-efficient Algorithms for Optimization

Our Conjecture: There is a class of linear systems with condition number κ, such that any algorithm either requires $\Omega(d^2)$ memory or $d\ \mathrm{poly}(\kappa)$ examples.

With more structure, can get best of both worlds!

Result (Informal):
For some structured linear systems, can get $d\ \mathbf{polylog}(\kappa)$ examples with $O(d)$ memory!

This is true more broadly beyond linear systems, and holds for any ”multiscale” optimization problem.

Memory-efficient Algorithms for Optimization

Result (Informal):
For some structured linear systems, can get d **polylog**(κ) examples with $O(d)$ memory!

Linear system has small number of unique eigenvalues:

$$\begin{bmatrix} 1 & & & & & \\ & \ddots & & & & \\ & & 1 & & & \\ & & & 1/\kappa & & \\ & & & & \ddots & \\ & & & & & 1/\kappa \end{bmatrix}$$

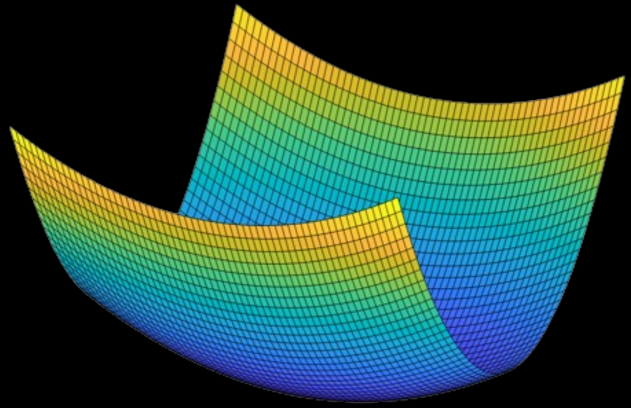
Memory-efficient Algorithms for Optimization

Linear system has two unique eigenvalues

Too large for larger eigendirections!

$$\begin{bmatrix} 1 & & & & & \\ & \ddots & & & & \\ & & 1 & & & \\ & & & 1/\kappa & & \\ & & & & \ddots & \\ & & & & & 1/\kappa \end{bmatrix}$$

Need about $\approx \kappa$ steps because of small eigendirections



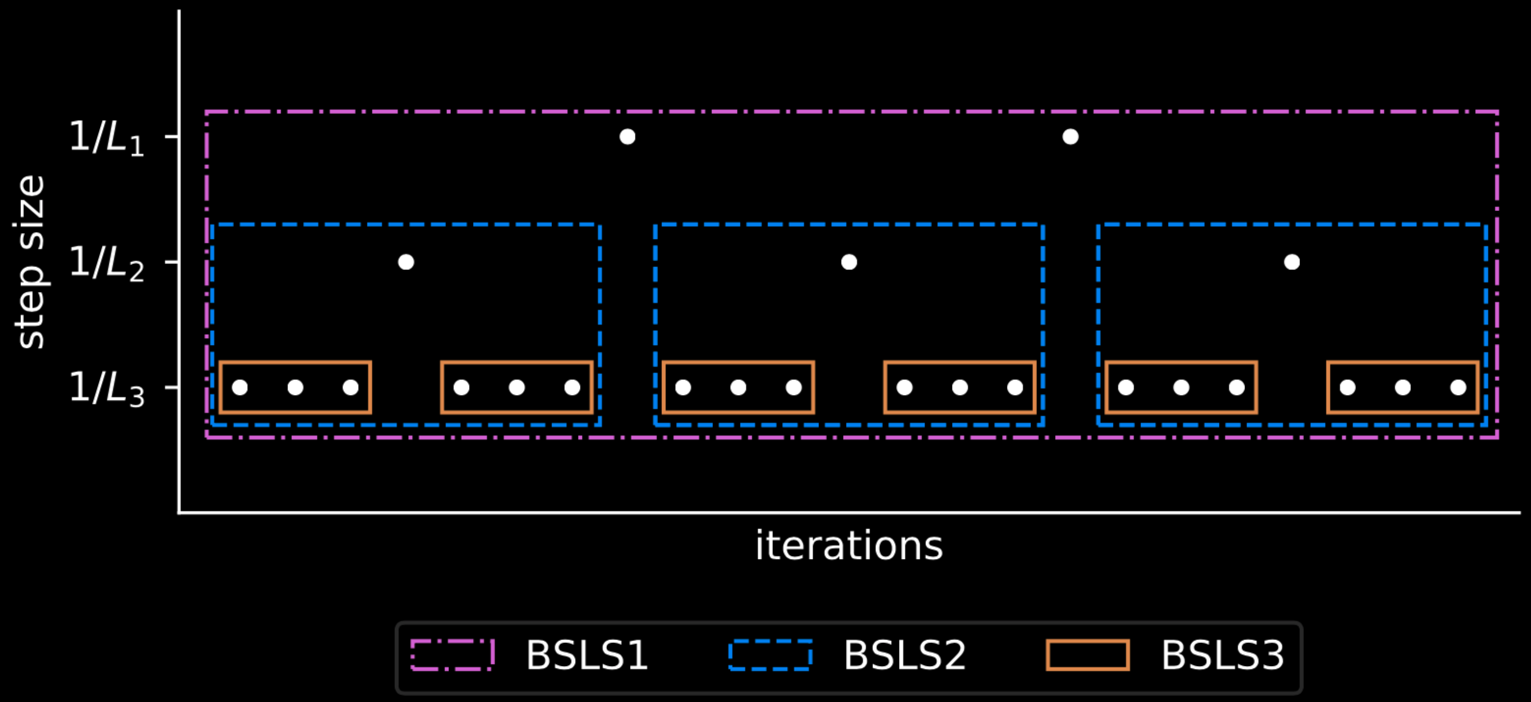
Safest choice: Take step size ≈ 1

Aggressive choice: Take step size $\approx \kappa$

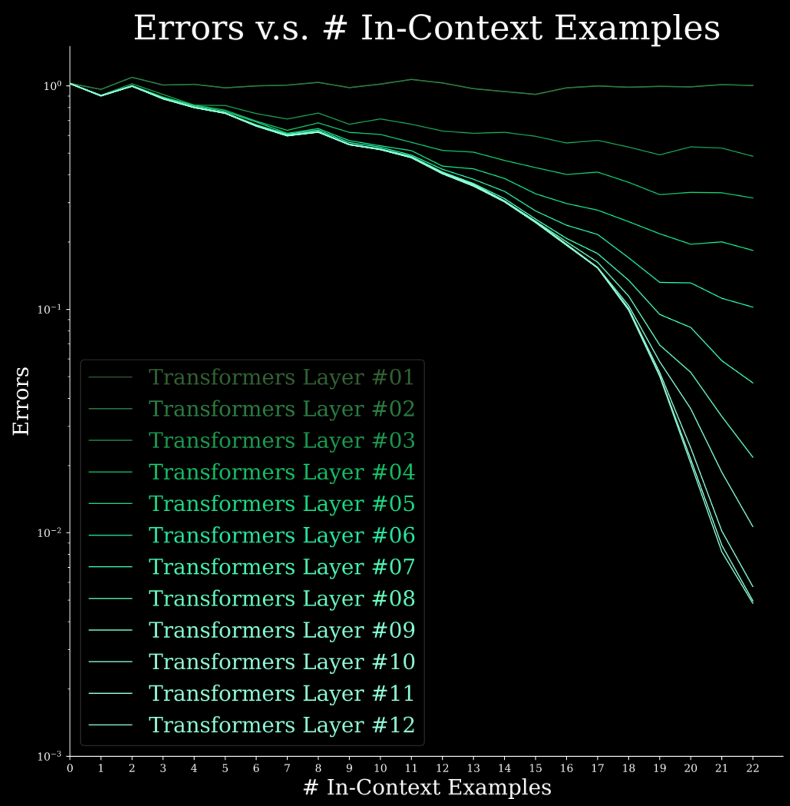
Solution: Follow large step with small steps to fix error along larger eigendirections

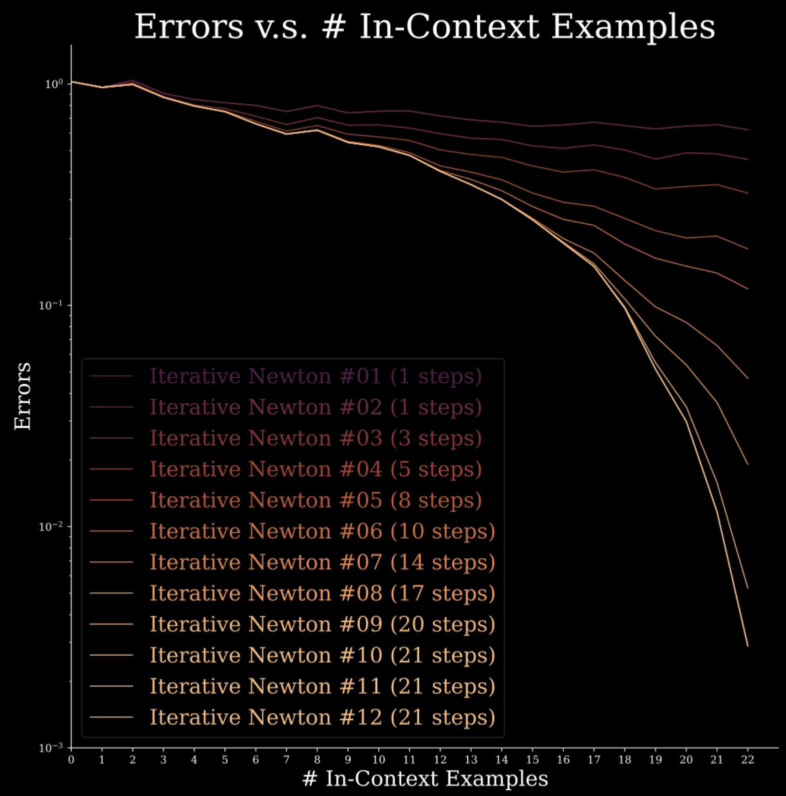
Memory-efficient Algorithms for Optimization

Theorem (Kelner, Marsden, Sharan, Sidford, Yuan, Valiant):
For some structured linear systems, recursive sequence of large and small steps solves the problem with d polylog(κ) **examples/gradient queries** and $O(d)$ **memory**.



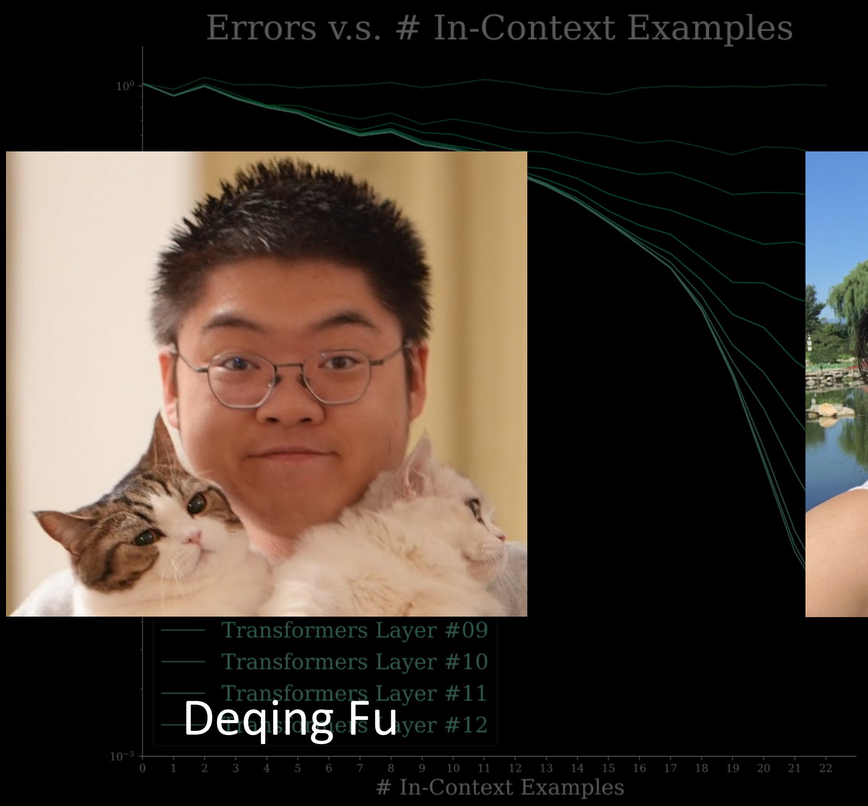
Using theory to understand what deep learning models learn?



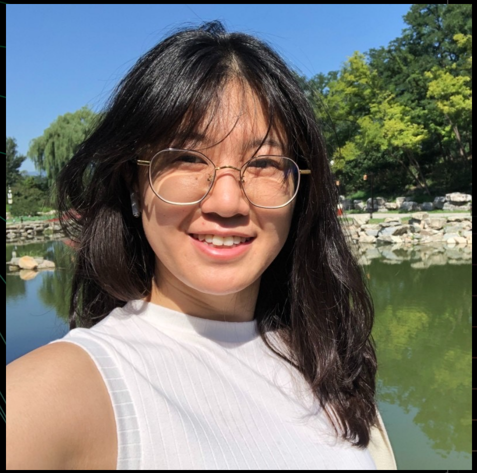


Transformers Learn Higher-Order Optimization Methods for In-Context Learning: A Study with Linear Models
Deqing Fu, Tian-Qi Chen, Robin Jia, Vatsal Sharan, 2023

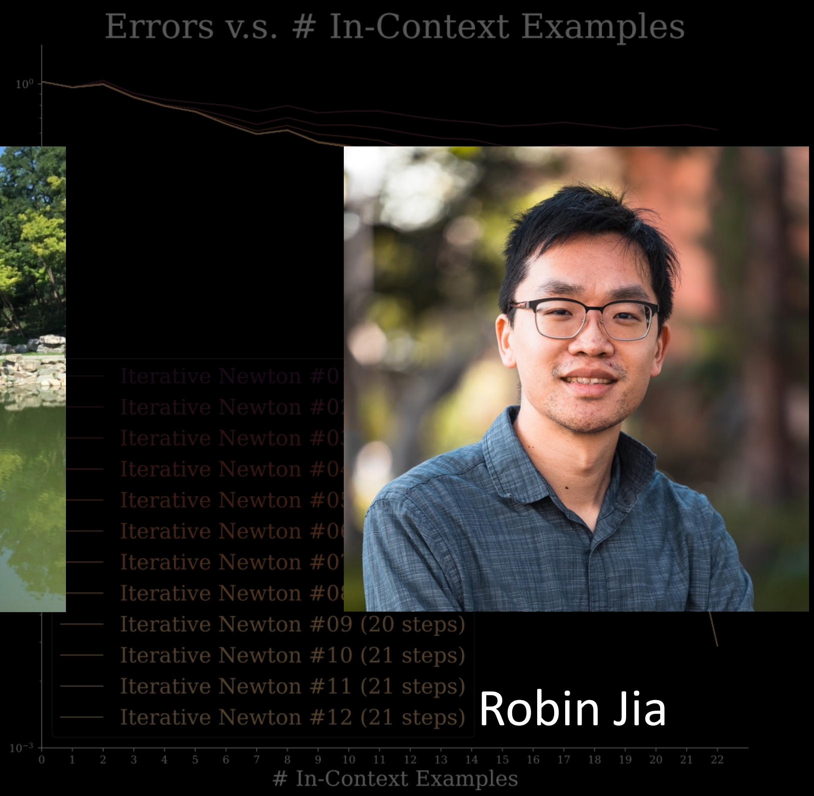
Using theory to understand what deep learning models learn?



Deqing Fu



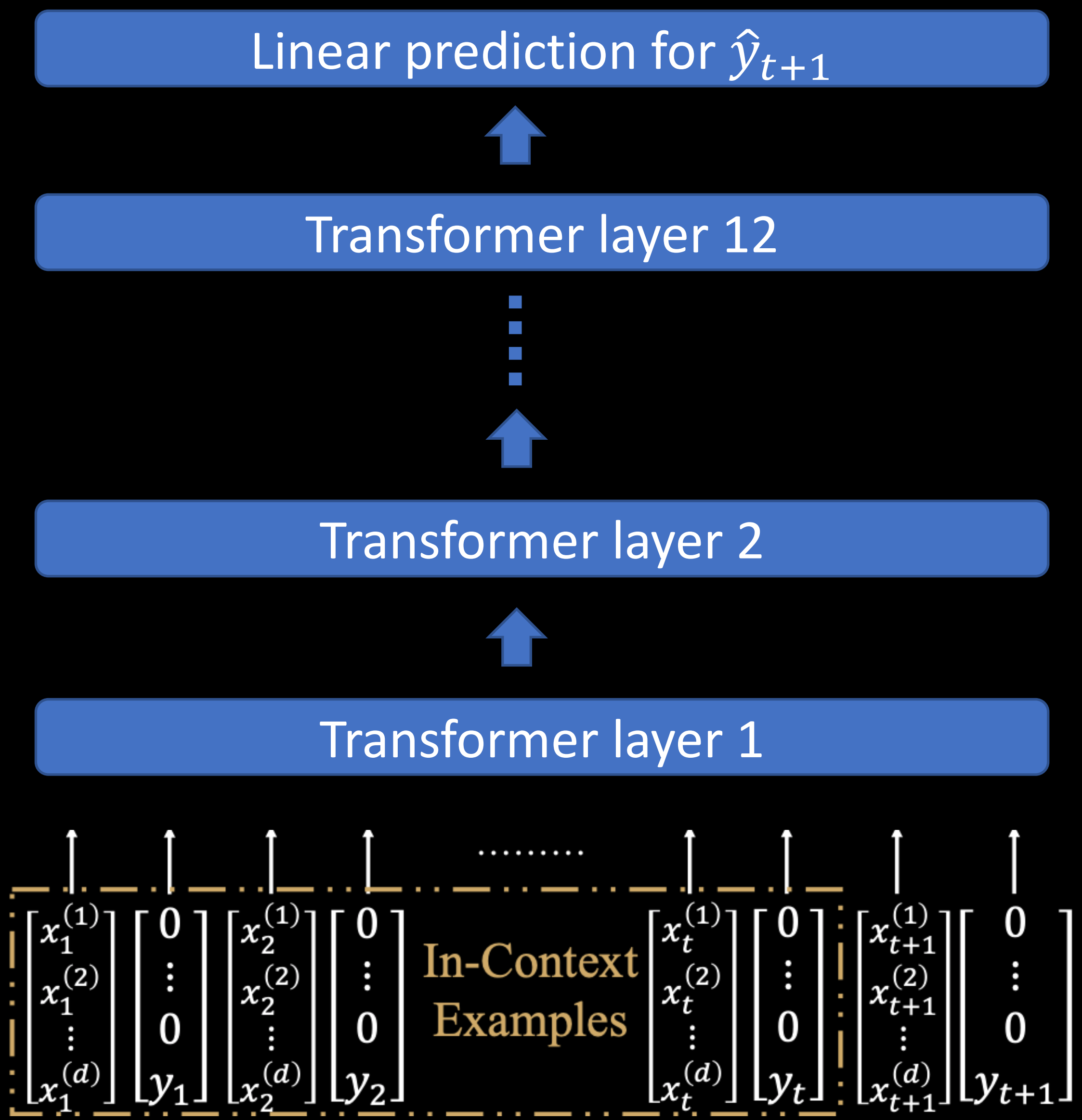
Tian-Qi Chen



Robin Jia

Transformers Learn Higher-Order Optimization Methods for In-Context Learning: A Study with Linear Models
Deqing Fu, Tian-Qi Chen, Robin Jia, Vatsal Sharan, 2023

Transformers for linear regression

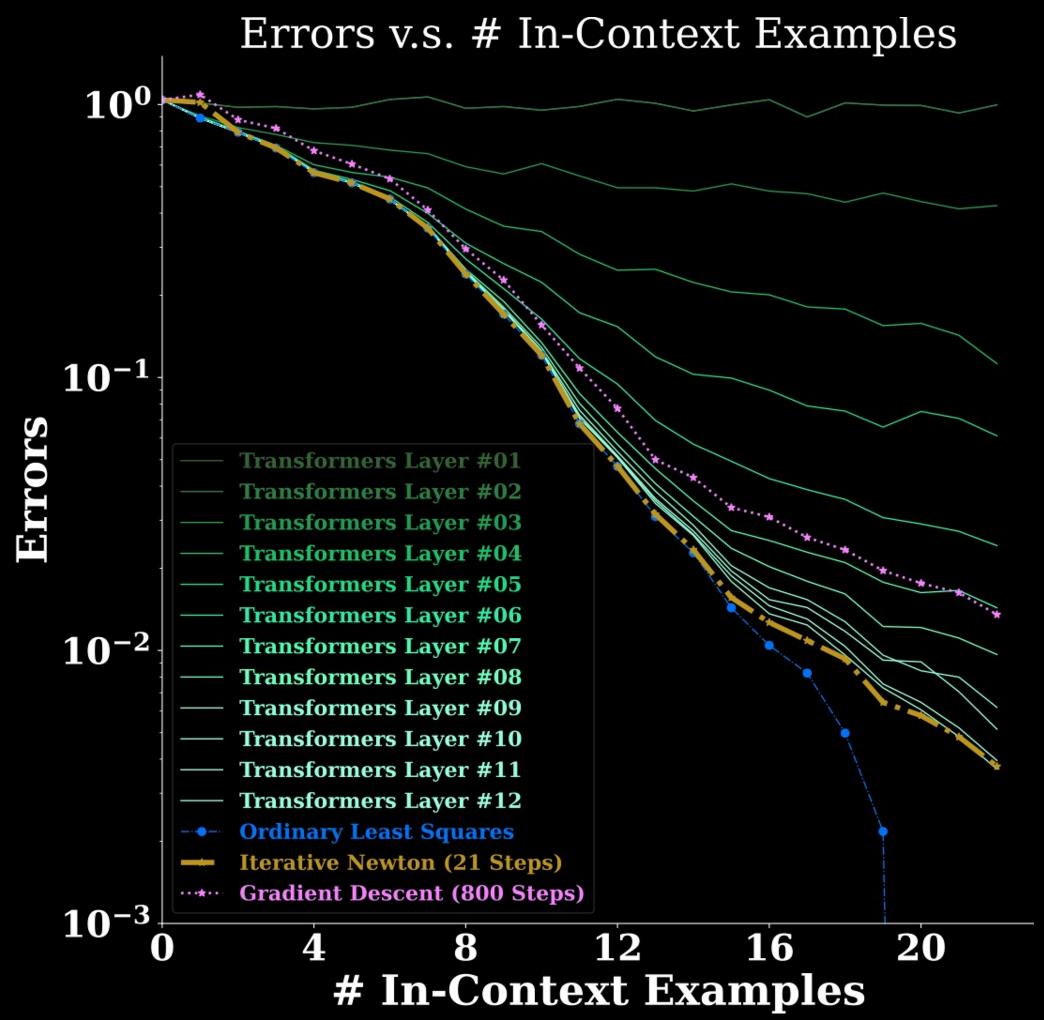


“Applied theory”?

Claim:

1. We can use understanding of statistical and computational gaps to understand mechanisms of models
2. Available memory may explain differences in behavior between different architectures

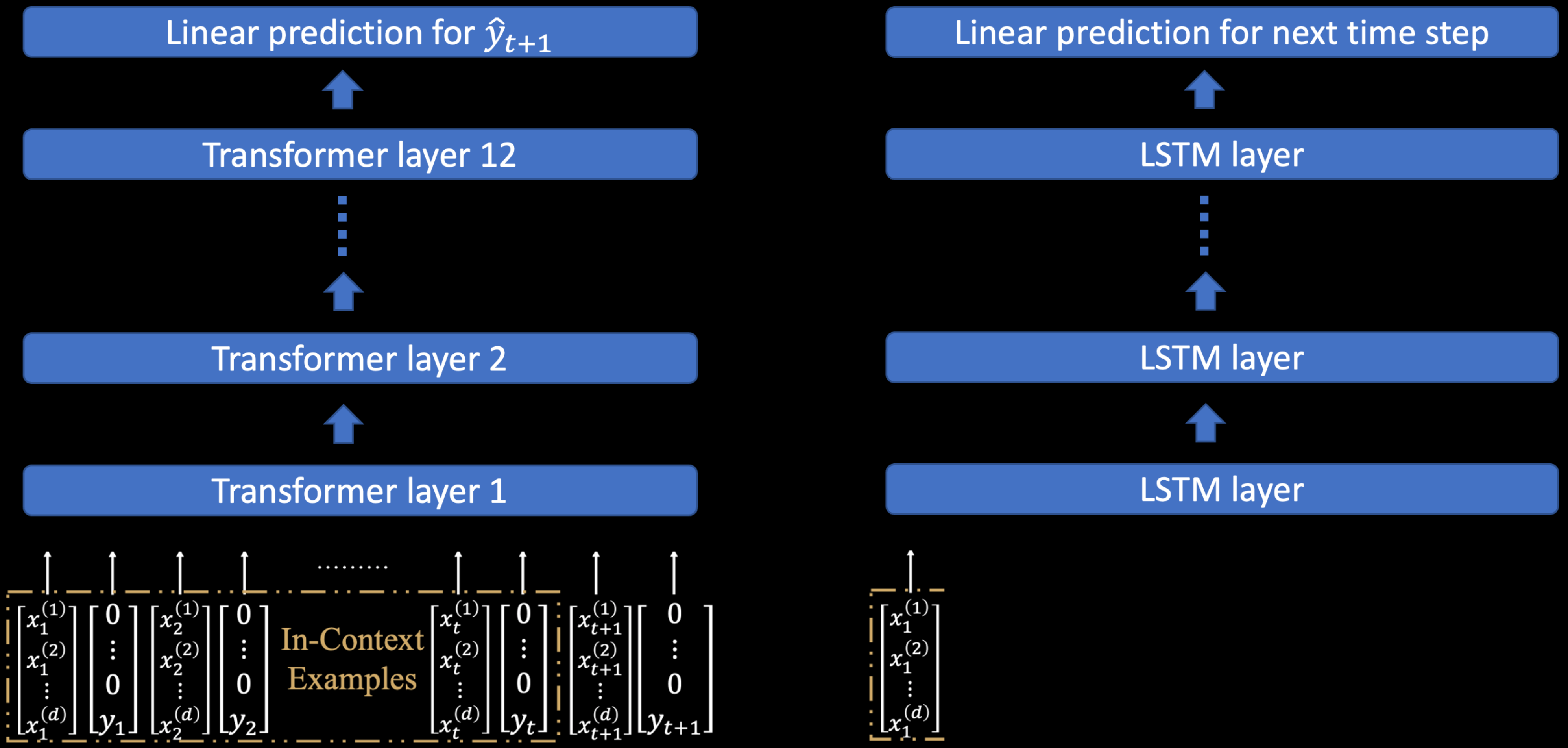
1. We can use understanding of statistical and computational gaps to understand mechanisms of models



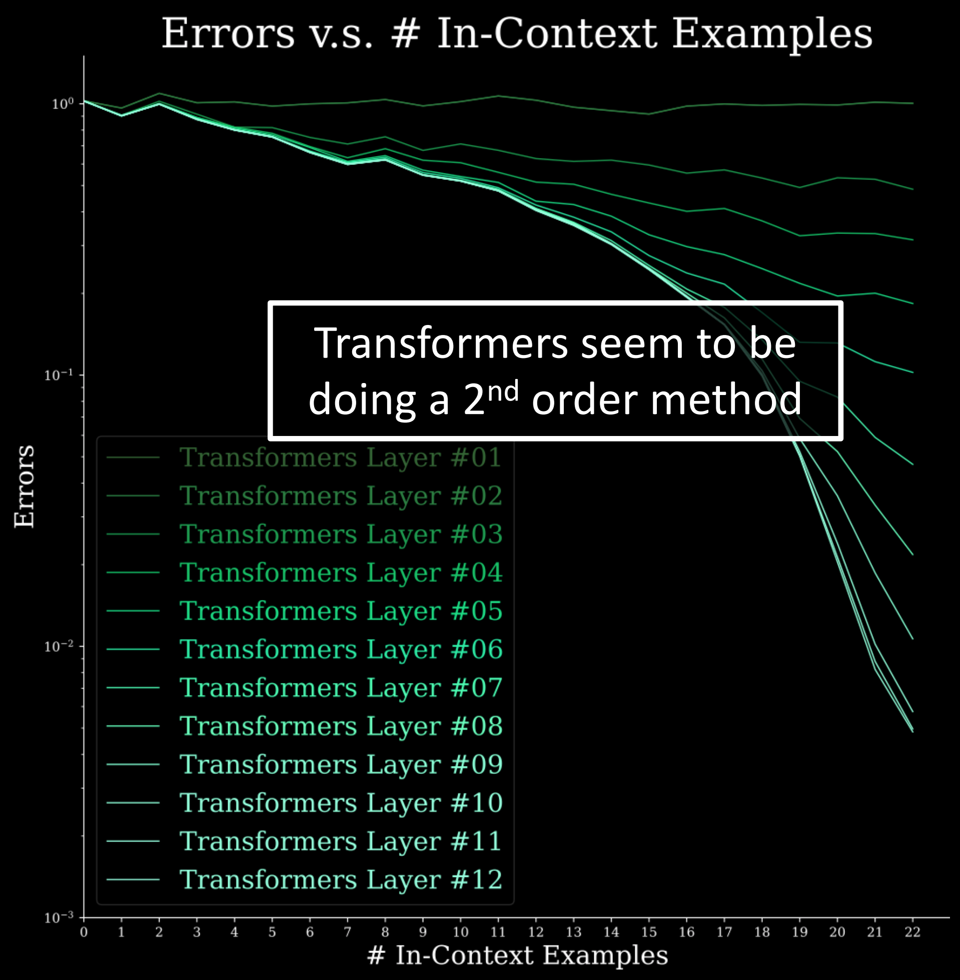
Based on rate of convergence, argue that Transformers cannot be doing any 1st order method

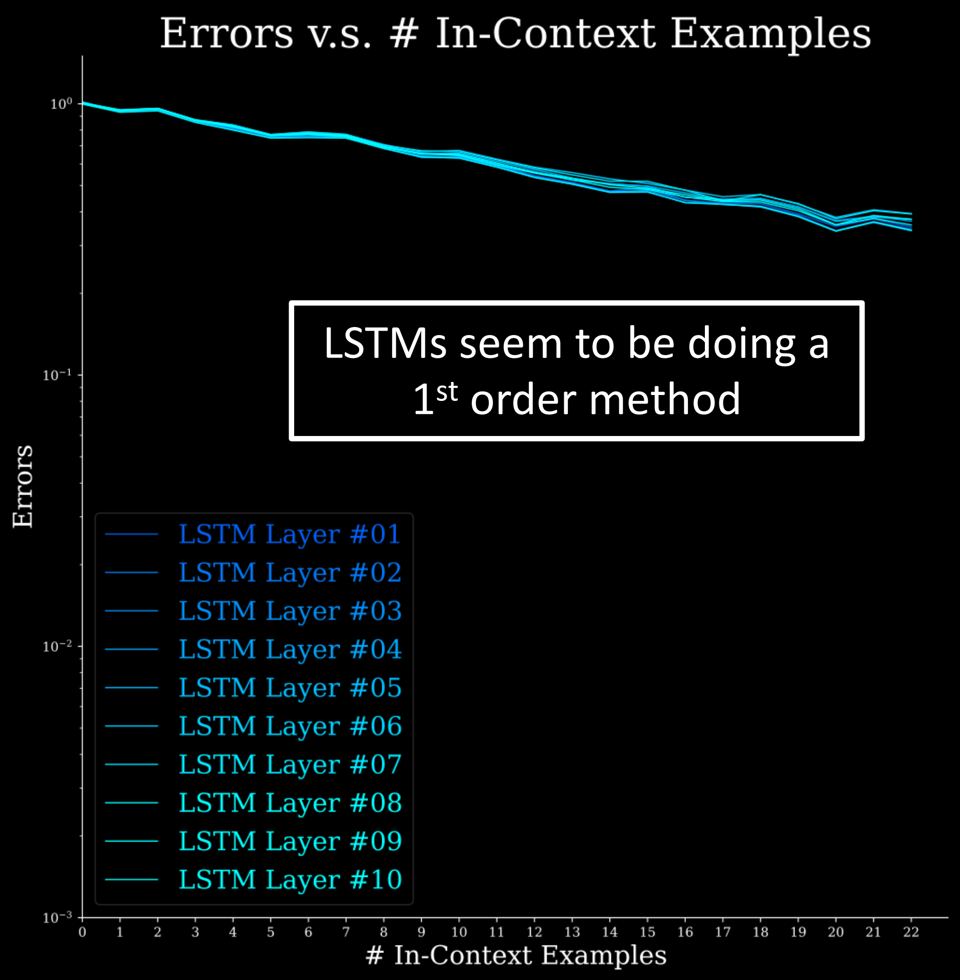
Test on ill-conditioned settings where gap between 1st and 2nd order methods is largest

2. Available memory may explain differences in behavior between different architectures



2. Available memory may explain differences in behavior between different architectures





Memory is a fundamental computation resource.
Memory considerations are crucial in practice.

What is the role of memory in learning and optimization?
Are there tradeoffs between available memory and information requirement?

Memory Dichotomy Hypothesis: It is not possible to significantly improve on the convergence rate of known memory efficient techniques without using significantly more memory.

- Memory determines the best available convergence rate
- Memory provides a separation between simple and complex techniques
- New problem structures where we can circumvent lower bounds, new variants of GD