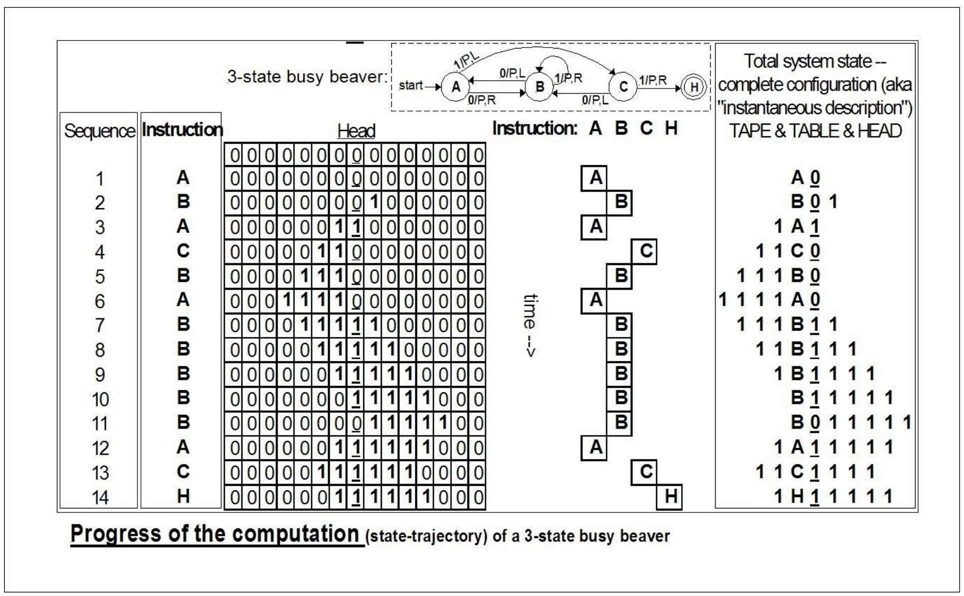
Query Complexity

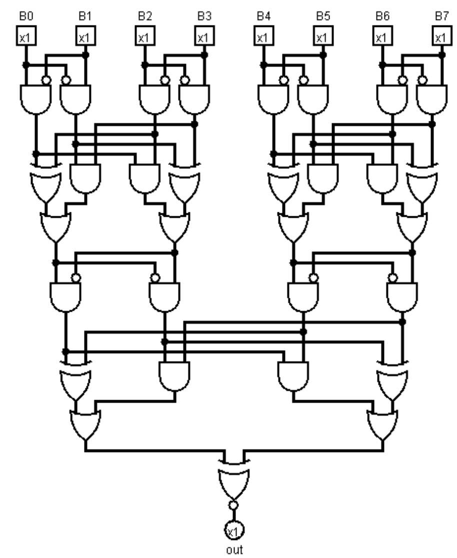
Simple, Structured and Significant

Suhail Sherif, TSS

Turing Machines and Circuits

Nice, relevant models of computation, but…

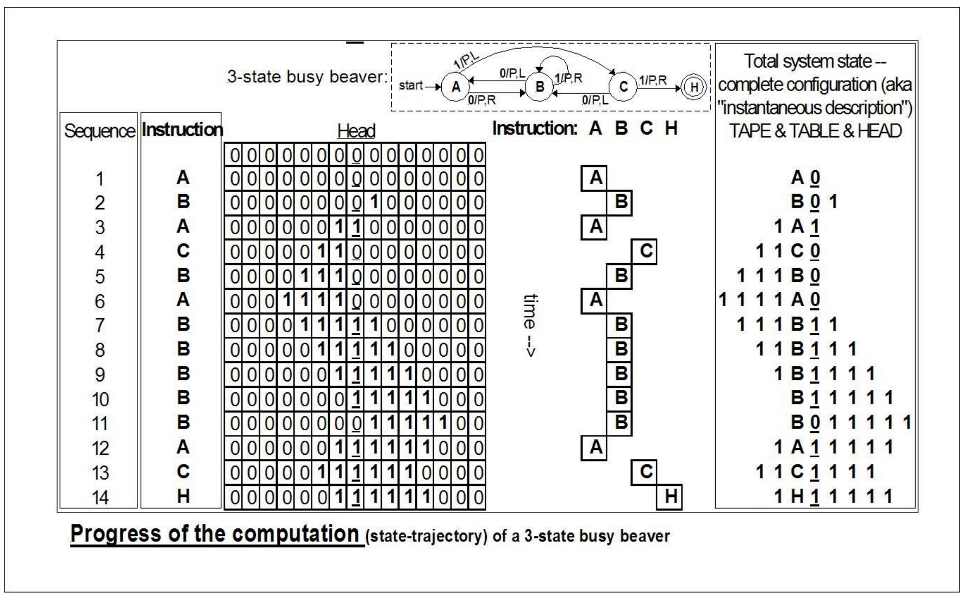


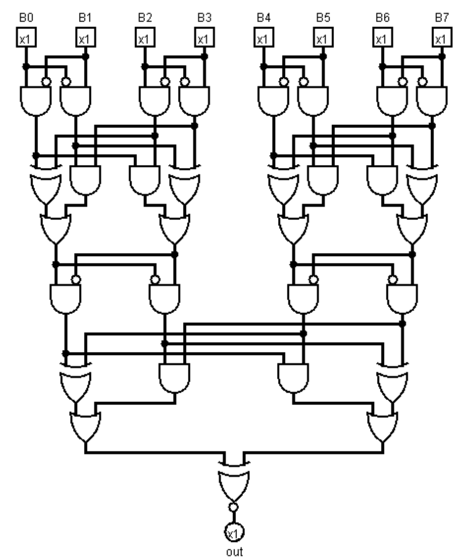


Turing Machines and Circuits

Nice, relevant models of computation, but…

- Too hard to reason about what they can do.





The Query Model

The Query Model

Input: Some input x, say as a bitstring of n bits.
Output: $f(x)$.

The Query Model

Input: Some input x, say as a bitstring of n bits.
Output: $f(x)$.

We don't know x. We can make queries to x.
"What is the 5th bit of x?"

The Query Model

Input: Some input x, say as a bitstring of n bits.
Output: $f(x)$.

We don't know x. We can make queries to x.
"What is the 5th bit of x?"

How many queries are needed in order to find out $f(x)$?

Query Complexity

Easier than TMs, Circuits

Query Complexity

Easier than TMs, Circuits

- With $f(x) := \quad x_i \quad \bmod 2$,

  $i \quad [n]$

  deterministic, nondeterministic, randomized query complexities are n.

Query Complexity

Easier than TMs, Circuits

- With $f(x) := \ _{i \quad [n]} x_i \mod 2$,
  deterministic, nondeterministic, randomized query complexities are n.

- With $f(x) := \ _{i \quad [n]} x_i$,
  deterministic query complexity is n, but nondeterministic is 1.

Query Complexity

Easier than TMs, Circuits

- With $f(x) :=$ x_i $\bmod 2$,

  i $[n]$

  deterministic, nondeterministic, randomized query complexities are n.

- With $f(x) :=$ x_i,

  i $[n]$

  deterministic query complexity is n, but nondeterministic is 1.

- With $f(x) :=$ smallest prime factor of x,
  deterministic and non-deterministic query complexity is n.

Query Complexity

But why?

Query Complexity

But why?

- Query complexities of functions don’t seem to reflect how hard the functions are to compute.

Query Complexity

But why?

- Query complexities of functions don't seem to reflect how hard the functions are to compute.
- Can never get a query complexity larger than n.

Query Complexity

But why?

- Query complexities of functions don't seem to reflect how hard the functions are to compute.
- Can never get a query complexity larger than n.
- Looking at input bits isn't hard, why are you even counting that?

Query Complexity

But why?

- Query complexities of functions don't seem to reflect how hard the functions are to compute.
- Can never get a query complexity larger than n.
- Looking at input bits isn't hard, why are you even counting that?
- To sum it up, researching query complexity is useless.

Query Complexity

But why?

- Query complexities of functions don’t seem to reflect how hard the functions are to compute.
- Can never get a query complexity larger than n.
- Looking at input bits isn’t hard, why are you even counting that?
- To sum it up, researching query complexity is useless.

Thank you for your attention.

I am now open to questions.

Query Complexity
Relevance to TM Complexity

Solve an impossible task with the help of a cryptic oracle.

Query Complexity

Relevance to TM Complexity

Query Complexity

Relevance to TM Complexity

Elements of language L
000
0000
00000
0000000

Query Complexity

Relevance to TM Complexity

Elements of language L	Elements of language O
000	110
0000	1001
00000	10000
0000000	1011111

Query Complexity
Relevance to TM Complexity

$0^i \quad L \qquad\qquad x \quad O$

$x \quad \{0,1\}^i$

Elements of language L	Elements of language O
000	110
0000	1001
00000	10000
0000000	1011111

Query Complexity

Relevance to TM Complexity

$0^i \quad L \qquad x \quad O$

$x \quad \{0,1\}^i$

$L \quad NP^O$

Elements of language L	Elements of language O
000	110
0000	1001
00000	10000
0000000	1011111

Query Complexity
Relevance to TM Complexity

$0^i \quad L \qquad\qquad x \quad O$

$x \quad \{0,1\}^i$

$M_1 \leftarrow 0^{n_1}$

M_2

$\vdots$

$L \quad NP^O$

Elements of language L	Elements of language O
000	110
0000	1001
00000	10000
0000000	1011111

$L \quad P^O$

Oracle Separations

Oracle Separations

- P cannot “simulate” NP.

Oracle Separations

- P cannot "simulate" NP.
- Any proof that $\mathsf{P} = \mathsf{NP}$ has to be subtle enough to not hold when there are oracles.

Oracle Separations

- P cannot “simulate” NP.
- Any proof that $\mathsf{P} = \mathsf{NP}$ has to be subtle enough to not hold when there are oracles.
  - For instance, it cannot just be a diagonalization proof.

Oracle Separations

- P cannot "simulate" NP.
- Any proof that P = NP has to be subtle enough to not hold when there are oracles.
  - For instance, it cannot just be a diagonalization proof.
  - Also holds for proving P NP.

Oracle Separations

- P cannot “simulate” NP.
- Any proof that P = NP has to be subtle enough to not hold when there are oracles.
  - For instance, it cannot just be a diagonalization proof.
  - Also holds for proving P NP.
- Similar results for many pairs of complexity classes.

Oracle Separations

- P cannot "simulate" NP.
- Any proof that P = NP has to be subtle enough to not hold when there are oracles.
  - For instance, it cannot just be a diagonalization proof.
  - Also holds for proving P NP.
- Similar results for many pairs of complexity classes.
  - EXPTIME can simulate polytime quantum, but PH cannot.

Nonoracular

Still spectacular

- In the oracle separations, we created languages forcing the algorithm to stick to using the oracle.
- More generally, we can abstract out certain approaches to solving problems by forcing our algorithm to only use data relevant to the approach.

Sort [100,23,13,141,2,20,15]

Sort [100,23,13,141,2,20,15]

Desired approach: Don’t look at the numbers except to compare them.

Sort [100,23,13,141,2,20,15]

Desired approach: Don't look at the numbers except to compare them.

Input: A sequence of numbers $x_1, \ldots, x_n$.

Output: The sorted sequence.

Sort [100,23,13,141,2,20,15]

Desired approach: Don’t look at the numbers except to compare them.

Input: A sequence of numbers $x_1, \ldots, x_n$.
Output: The sorted sequence.

Not what we wanted.
n queries

Sort [100,23,13,141,2,20,15]

Desired approach: Don’t look at the numbers except to compare them.

Input: A sequence of bits $\{ b_{i,j} \}_{(i,j) \quad \binom{[n]}{2}}$ with the promise that there exists $x_1, \ldots, x_n$ such that $b_{i,j} = 0 \quad x \quad i \quad x_j$.

Output: The sequence $i_1, \ldots, i_n$ such that $x_{i_1} \quad x_{i_2} \quad \ldots \quad x_{i_n}$.

Factor 14017

Factor 14017

$10^1 \bmod 14017 = 10$

$10^2 \bmod 14017 = 100$

$10^3 \bmod 14017 = 1000$

$10^4 \bmod 14017 = 10000$

$10^5 \bmod 14017 = 1881$

$10^6 \bmod 14017 = 4793$

$10^7 \bmod 14017 = 5879$

$10^8 \bmod 14017 = 2722$

$10^{6890} \bmod 14017 = 1$

Factor 14017

$10^1 \mod 14017 = 10$
$10^2 \mod 14017 = 100$
$10^3 \mod 14017 = 1000$
$10^4 \mod 14017 = 10000$
$10^5 \mod 14017 = 1881$
$10^6 \mod 14017 = 4793$
$10^7 \mod 14017 = 5879$
$10^8 \mod 14017 = 2722$

$10^{6890} \mod 14017 = 1$

If we can find the period (6890 above), we can factor.

Factor 14017

10^1 mod 14017 = 10
10^2 mod 14017 = 100
10^3 mod 14017 = 1000
10^4 mod 14017 = 10000
10^5 mod 14017 = 1881
10^6 mod 14017 = 4793
10^7 mod 14017 = 5879
10^8 mod 14017 = 2722

10^{6890} mod 14017 = 1

If we can find the period (6890 above), we can factor.

Ideal approach: Find the period only by computing elements of the sequence with no further involvement of 14017.

Factor 14017

10^{1} mod 14017 = 10
10^{2} mod 14017 = 100
10^{3} mod 14017 = 1000
10^{4} mod 14017 = 10000
10^{5} mod 14017 = 1881
10^{6} mod 14017 = 4793
10^{7} mod 14017 = 5879
10^{8} mod 14017 = 2722

10^{6890} mod 14017 = 1

If we can find the period (6890 above), we can factor.

Ideal approach: Find the period only by computing elements of the sequence with no further involvement of 14017.

Factor 14017

$10^1 \mod 14017 = 10$
$10^2 \mod 14017 = 100$
$10^3 \mod 14017 = 1000$
$10^4 \mod 14017 = 10000$
$10^5 \mod 14017 = 1881$
$10^6 \mod 14017 = 4793$
$10^7 \mod 14017 = 5879$
$10^8 \mod 14017 = 2722$

$10^{6890} \mod 14017 = 1$

If we can find the period (6890 above), we can factor.

Ideal approach: Find the period only by computing elements of the sequence with no further involvement of 14017.

> Input: A sequence of numbers in the range $[M]$ with the promise that the ith element is $a^i \mod N$. (a and N are unknown.)
>
> Output: The period of the input sequence.

Factor 14017

10^1 mod 14017 = 10
10^2 mod 14017 = 100
10^3 mod 14017 = 1000
10^4 mod 14017 =10000
10^5 mod 14017 = 1881
10^6 mod 14017 = 4793
10^7 mod 14017 = 5879
10^8 mod 14017 = 2722

10^{6890} mod 14017 = 1

If we can find the period (6890 above), we can factor.

Ideal approach: Find the period only by computing elements of the sequence with no further involvement of 14017.

Input: A sequence of numbers in the range $[M]$ with the promise that the ith element is $a^i \bmod N$. (a and N are unknown.)

Output: The period of the input sequence.

Can still use a,N.

Factor 14017

$$10^1 \bmod 14017 = 10$$
$$10^2 \bmod 14017 = 100$$
$$10^3 \bmod 14017 = 1000$$
$$10^4 \bmod 14017 = 10000$$
$$10^5 \bmod 14017 = 1881$$
$$10^6 \bmod 14017 = 4793$$
$$10^7 \bmod 14017 = 5879$$
$$10^8 \bmod 14017 = 2722$$

$$10^{6890} \bmod 14017 = 1$$

If we can find the period (6890 above), we can factor.

Ideal approach: Find the period only by computing elements of the sequence with no further involvement of 14017.

Factor 14017

10^1 mod 14017 = 10
10^2 mod 14017 = 100
10^3 mod 14017 = 1000
10^4 mod 14017 =10000
10^5 mod 14017 = 1881
10^6 mod 14017 = 4793
10^7 mod 14017 = 5879
10^8 mod 14017 = 2722

10^{6890} mod 14017 = 1

If we can find the period (6890 above), we can factor.

Ideal approach: Find the period only by computing elements of the sequence with no further involvement of 14017.

Input: A periodic sequence of numbers in the range $[N]$ such that the elements in a period are all distinct.

Output: The period of the input sequence.

Factor 14017

$$
\begin{aligned}
10^1 &\bmod 14017 = 10\\
10^2 &\bmod 14017 = 100\\
10^3 &\bmod 14017 = 1000\\
10^4 &\bmod 14017 = 10000\\
10^5 &\bmod 14017 = 1881\\
10^6 &\bmod 14017 = 4793\\
10^7 &\bmod 14017 = 5879\\
10^8 &\bmod 14017 = 2722
\end{aligned}
$$

$$10^{6890} \bmod 14017 = 1$$

If we can find the period (6890 above), we can factor.

Ideal approach: Find the period only by computing elements of the sequence with no further involvement of 14017.

Input: A periodic sequence of numbers in the range $[N]$ such that the elements in a period are all distinct.

Requires $(\sqrt{N})$ queries.

Output: The period of the input sequence.

Factor 14017

10^1 mod 14017 = 10
10^2 mod 14017 = 100
10^3 mod 14017 = 1000
10^4 mod 14017 = 10000
10^5 mod 14017 = 1881
10^6 mod 14017 = 4793
10^7 mod 14017 = 5879
10^8 mod 14017 = 2722

10^{6890} mod 14017 = 1

If we can find the period (6890 above), we can factor.

Ideal approach: Find the period only by computing elements of the sequence with no further involvement of 14017.

Input: A periodic sequence of numbers in the range $[N]$ such that the elements in a period are all distinct.

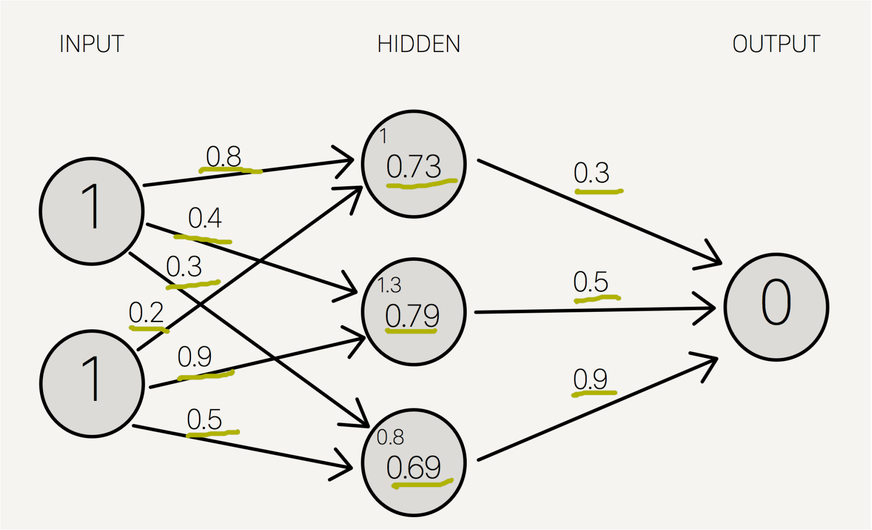
Requires $(\sqrt{N})$ queries.

But only $O(\log \log N)$ queries on a quantum computer.

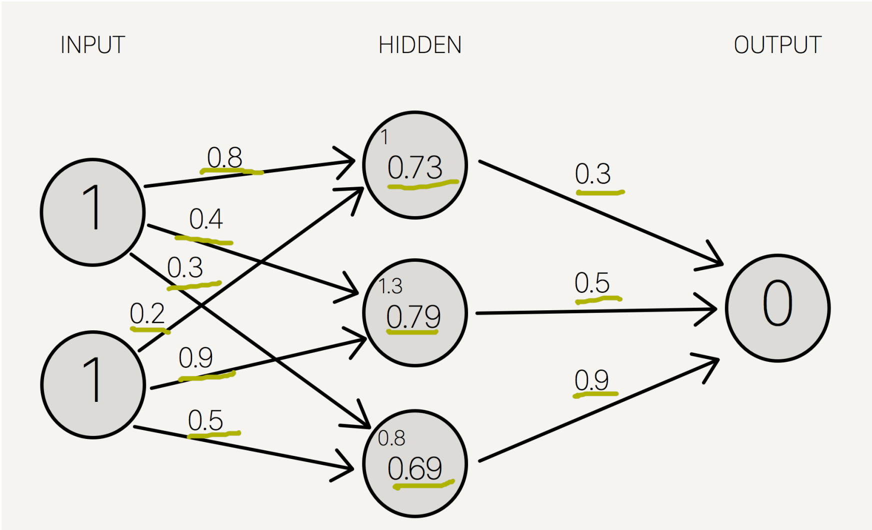
Output: The period of the input sequence.

Another example

Another example

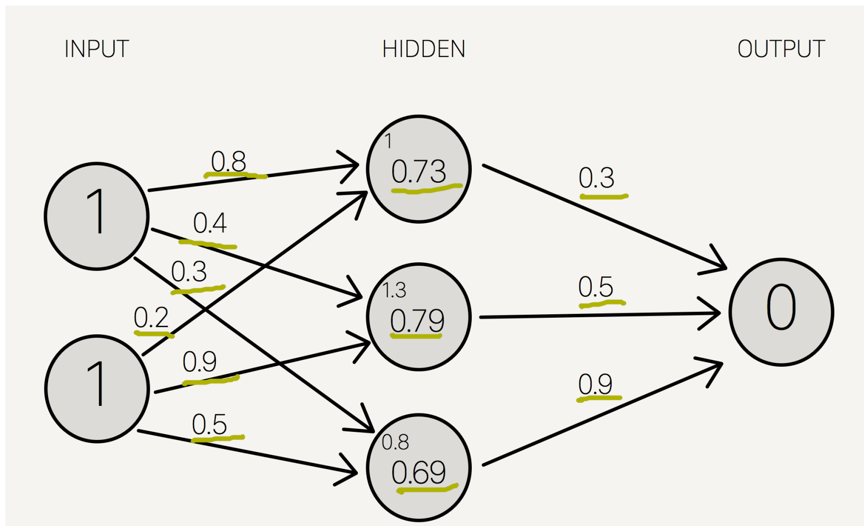


Another example



Find parameters that minimize the loss.

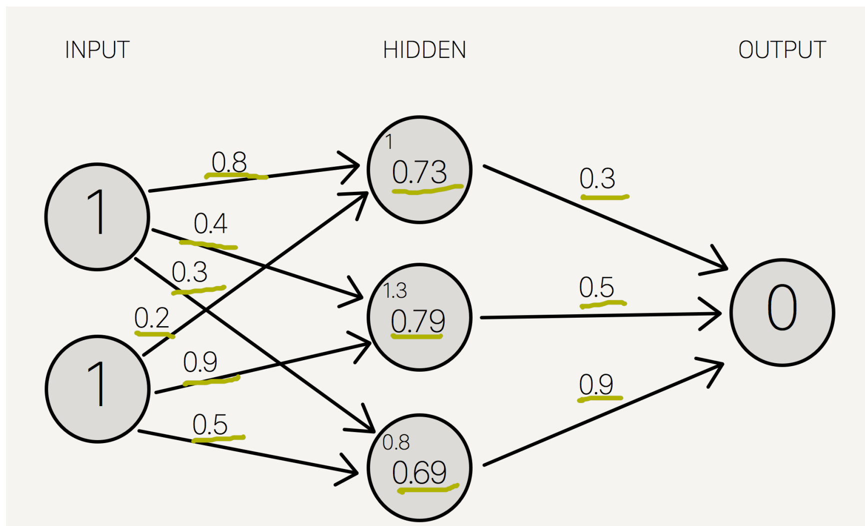
Another example



Find parameters that minimize the loss.

loss :

Another example



Find parameters that minimize the loss.

loss :

Easy to compute loss.

Making it a query problem

Making it a query problem

- Think of the input as $\{\text{loss}(x)\}_{x \quad {}^{n}}$

Making it a query problem

- Think of the input as $\{\mathrm{loss}(x)\}_{x \quad {}^{n}}$
- Easy to compute loss Querying the input is doable.

Making it a query problem

- Think of the input as $\{\mathrm{loss}(x)\}_{x \quad n}$
- Easy to compute loss Querying the input is doable.
- But the popular algorithm for this is Gradient Descent.

Making it a query problem

- Think of the input as $\{\text{loss}(x)\}_x$ n
- Easy to compute loss Querying the input is doable.
- But the popular algorithm for this is Gradient Descent.
  - Finding f given query access to f:

Making it a query problem

- Think of the input as $\{\text{loss}(x)\}_{x \quad n}$
- Easy to compute loss Querying the input is doable.
- But the popular algorithm for this is Gradient Descent.
  - Finding f given query access to f:
    - Requires $(\sqrt{n})$ queries.

Making it a query problem

- Think of the input as $\{\mathrm{loss}(x)\}_{x} \quad {}^{n}$
- Easy to compute loss Querying the input is doable.
- But the popular algorithm for this is Gradient Descent.
  - Finding f given query access to f:
    - Requires $(\sqrt{n})$ queries.
    - Or 1 query with a quantum computer.

Making it a query problem

- Think of the input as $\{\text{loss}(x)\}_{x \quad n}$
- Easy to compute loss Querying the input is doable.
- But the popular algorithm for this is Gradient Descent.
    - Finding f given query access to f:
        - Requires $(\sqrt{n})$ queries.
        - Or 1 query with a quantum computer.
    - Finding f given access to the network for f is easy.

Making it a query problem

- Think of the input as $\{\mathrm{loss}(x)\}_{x \quad n}$
- Easy to compute loss Querying the input is doable.
- But the popular algorithm for this is Gradient Descent.
  - Finding f given query access to f:
    - Requires $(\sqrt{n})$ queries.
    - Or 1 query with a quantum computer.
  - Finding f given access to the network for f is easy.
- Think of the input as $\{(\mathrm{loss}(x), \quad \mathrm{loss}(x))\}_{x \quad n}$

First-Order Convex Optimization

Joint work with Ankit Garg, Robin Kothari and Praneeth Netrapalli.

First-Order Optimization

First-Order Optimization

Input: $\{(f(x), \quad f(x))\}_{x \quad ^{n}, \; x \quad 1}$ for a continuous f.

Output: $\arg\min_{x \quad B} f(x)$

First-Order Optimization

Input: $\{(\ f(x), \quad f(x))\}_{x \quad {}^{n}, \ x \quad 1}$ for a continuous f.

Output: $x \quad B$ such that $f(x\ ) \quad \min_{x \ B} f(x) +$

First-Order Optimization

Input: $\{(\ f(x), \quad f(x))\}_{x \quad B}$ for a continuous f with $\quad f(x) \quad 1.$

Output: $x \quad B$ such that $f(x\ ) \quad \min_{x \quad B} f(x) +$

First-Order Optimization

Input: $\{(f(x), \quad f(x))\}_{x \quad B}$ for a continuous f with $\quad f(x) \quad 1.$

Output: $x \quad B$ such that $f(x\ ) \quad \min_{x \quad B} f(x) +$

Doable with $\left(1 + \frac{1}{\ }\right)^{n}$ queries.

First-Order Convex Optimization

First-Order Convex Optimization

Input: $\{(f(x), \quad f(x))\}_{x \quad B}$ for a convex f with $\quad f(x) \quad 1.$

Output: $x \quad B$ such that $f(x\ ) \quad \min_{x \quad B} f(x) +$

First-Order Convex Optimization

Input: $\{(f(x), \quad f(x))\}_{x \quad B}$ for a convex f with $\quad f(x) \quad 1.$

Output: $x \quad B$ such that $f(x\,) \quad \min_{x \quad B} f(x) +$

Doable with $\min \{ \quad n \log (\frac{1}{\ }) , \frac{1}{\ ^{2}} \}$ queries.

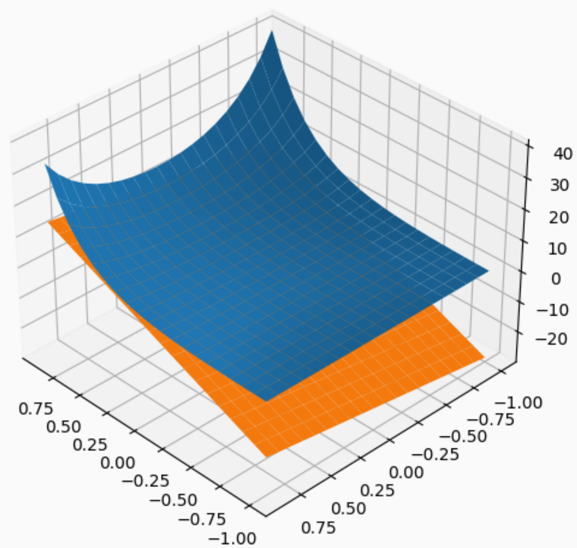
Theorem (Garg Kothari Netrapalli S `20)

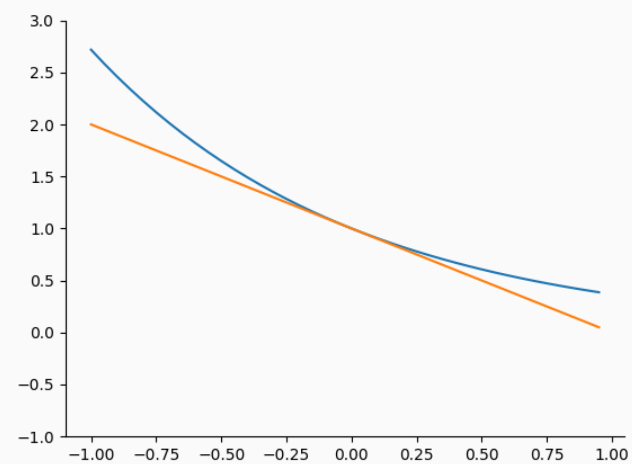
The dimension-independent complexity of first-order convex optimization is $(1/\ ^2)$ even for quantum algorithms.

The Task

Given: a convex region B, first-order oracle access to a convex function $f : \mathbb{R}^n \to \mathbb{R}$.

Find $x' \in B$ s.t. $f(x') \le \min_{x \in B} f(x) + \epsilon$.

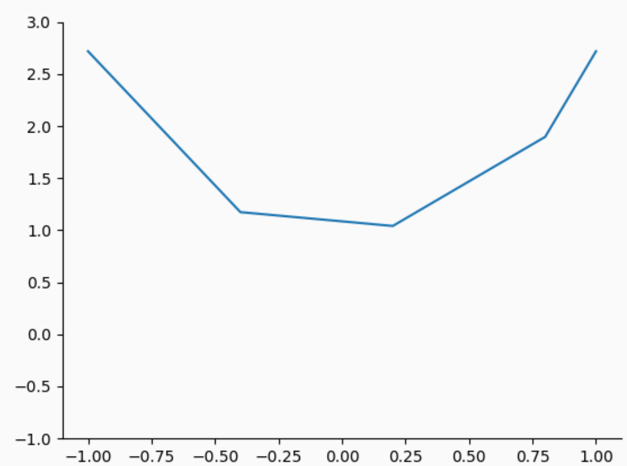




The Task

Given: a convex region B, rst-order oracle access to a convex function $f : \mathbb{R}^n \to \mathbb{R}$.

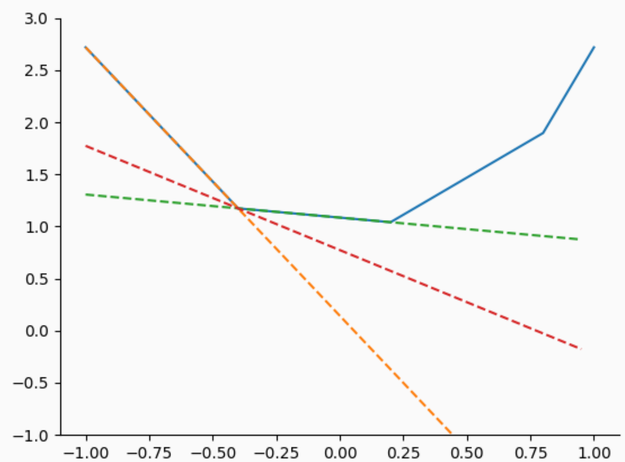
Find $x' \in B$ s.t. $f(x') \leq \min_{x \in B} f(x) + \epsilon$.



The Task

Given: a convex region B, first-order oracle access to a convex function $f : \mathbb{R}^n \to \mathbb{R}$.

Find $x' \in B$ s.t. $f(x') \leq \min_{x \in B} f(x) + \epsilon$.



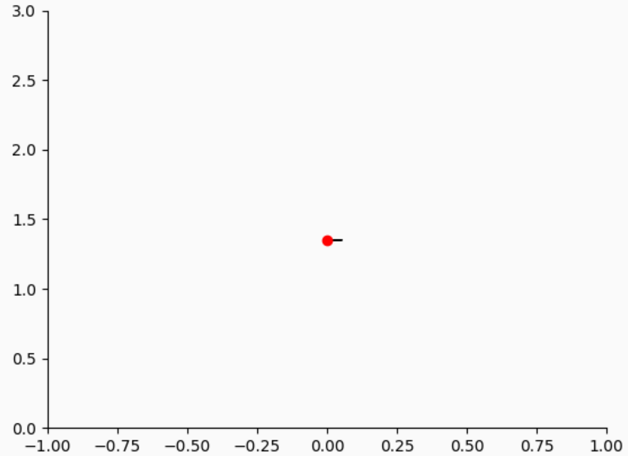
$g \in \partial f(x) \Leftrightarrow f(x+v) \geq f(x) + \langle v, g \rangle$ for all v

Known Algorithms I

1-dimensional fn:

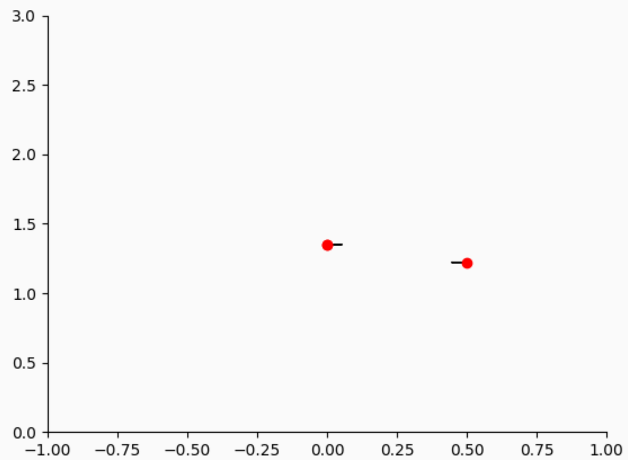
Known Algorithms I

1-dimensional fn:



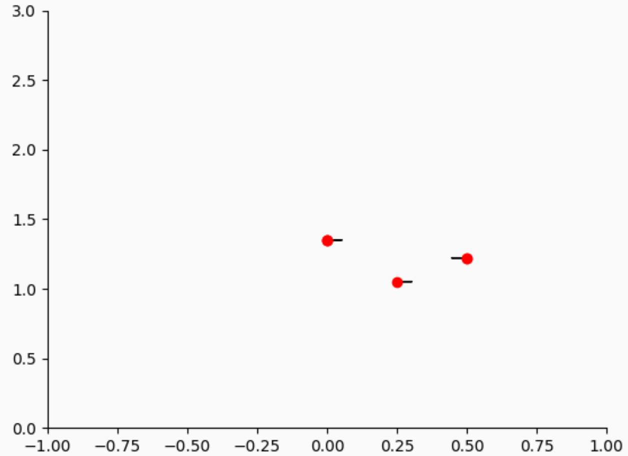
Known Algorithms I

1-dimensional fn:



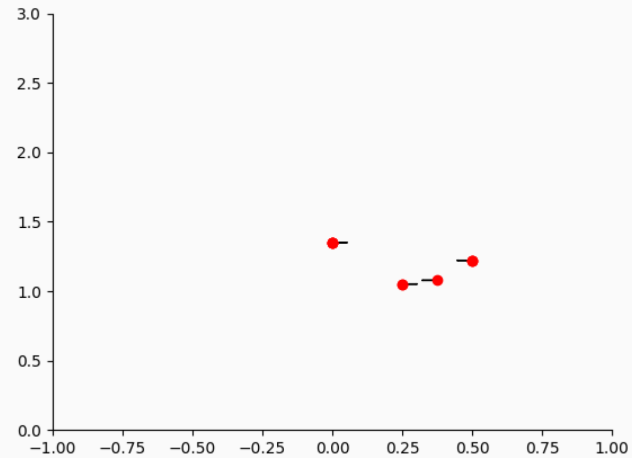
Known Algorithms I

1-dimensional fn:



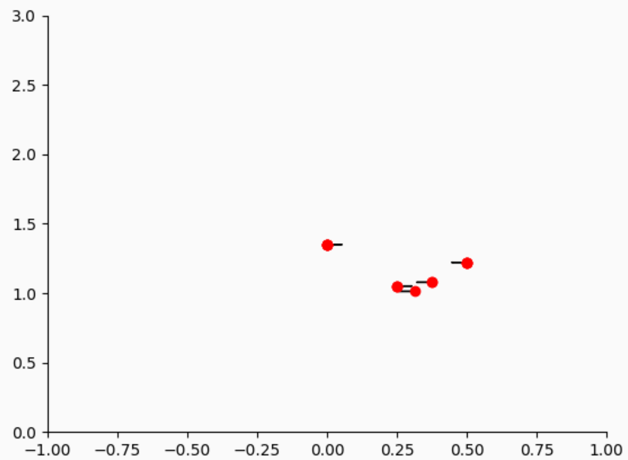
Known Algorithms I

1-dimensional fn:



Known Algorithms I

1-dimensional fn:



Known Algorithms I

1-dimensional fn:

log(1/) steps.

Known Algorithms I

n-dimensional fn:

Known Algorithms I

n-dimensional fn:

(Center of Gravity Method)

Known Algorithms I

n-dimensional fn:

(Center of Gravity Method)

$$\log \frac{\mathrm{Vol}(B(1))}{\mathrm{Vol}(B(\ ))}$$

$$= n \log(1/\ ) \text{ steps.}$$

Known Algorithms II

Center of Gravity Method

n log(1/) steps.

Projected Subgradient Descent

Known Algorithms II

Center of Gravity Method

$n \log(1/\ )$ steps.

Projected Subgradient Descent

$\bullet\, x$

$x \bullet$ —— $\bullet\, x^0 = x \quad g \quad {}_x$

Known Algorithms II

Center of Gravity Method

$n \log(1/ \;)$ steps.

Projected Subgradient Descent

x

x $x^0 = x \quad g \quad x$

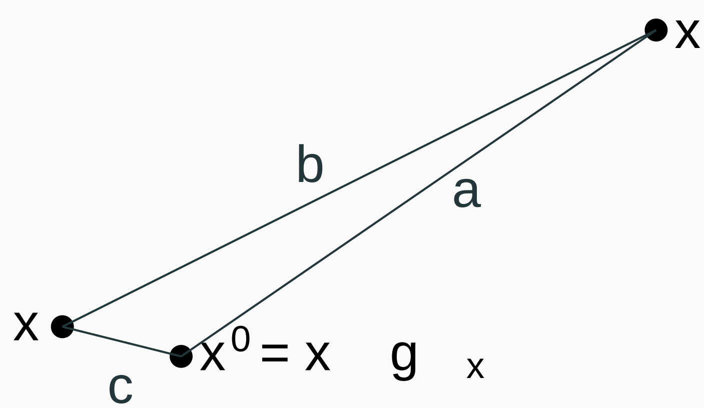
$h\, g\, {}_x, x \quad x\, i \quad f(x) \quad f(x \qquad)$

Known Algorithms II

Center of Gravity Method

n log(1/) steps.

Projected Subgradient Descent



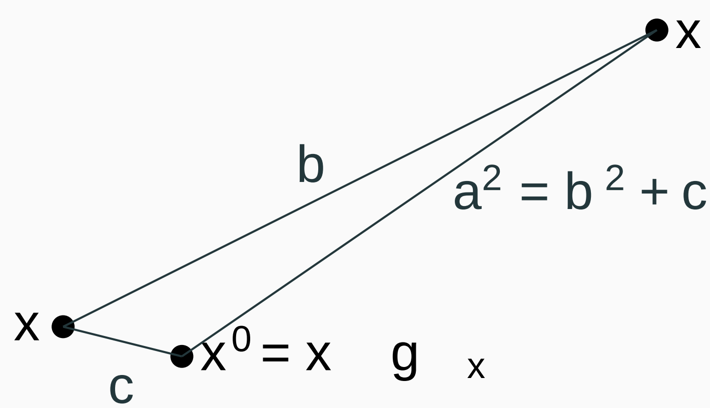
$h\, g\, {}_x , x \quad x i \quad f(x) \quad f(x \qquad)$

Known Algorithms II

Center of Gravity Method

n log(1/) steps.

Projected Subgradient Descent



$a^2 = b^2 + c^2 \quad 2\langle b, c\rangle$

$\langle g_x, x \quad x\rangle \quad f(x) \quad f(x \qquad)$

$\|x \quad x^0\|^2 \quad \|x \qquad x\|^2 + \quad ^2 \quad 2 (f(x) \quad f(x \qquad))$

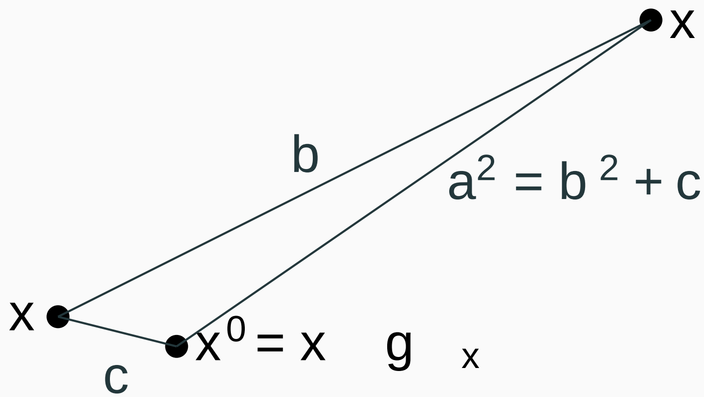
Known Algorithms II

Center of Gravity Method

$n \log(1/\ )$ steps.

Projected Subgradient Descent

$1/\ ^2$ steps.



$a^2 = b^2 + c^2 \quad 2\langle b, c\rangle$

$\langle g_x, x \quad x\rangle \quad f(x) \quad f(x \qquad)$

$\|x \quad x^0\|^2 \quad \|x \qquad x\|^2 + \quad ^2 \quad 2\ (f(x) \quad f(x \qquad))$

With $\ = \ $, $\|x \qquad x^0\|^2$ drops by $\ ^2$ when $f(x) > f(x \ ) + \ $.

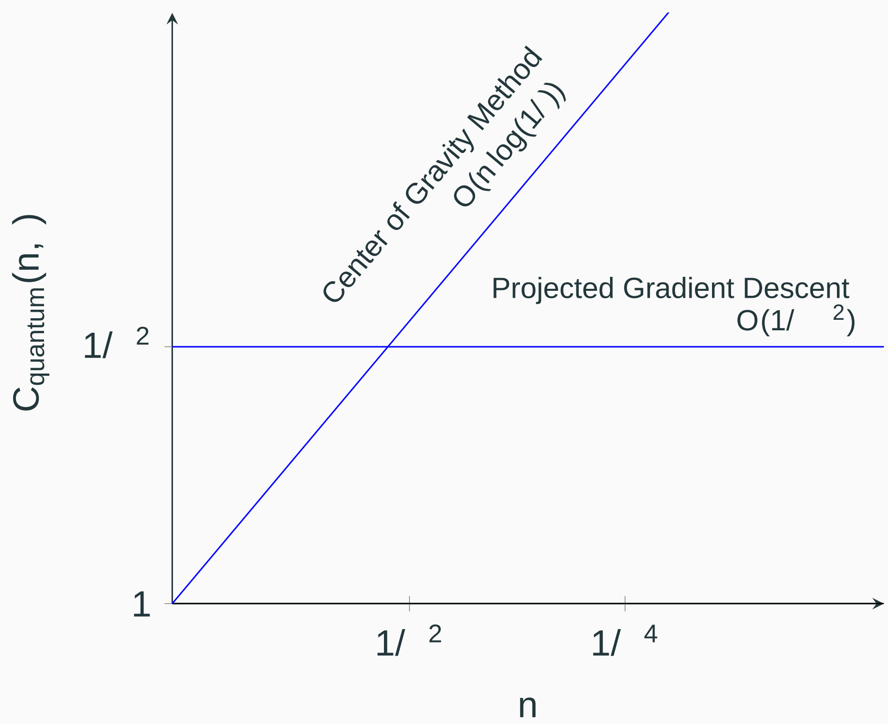
Known Algorithms II

Center of Gravity Method

n log(1/) steps.

Projected Subgradient Descent

1/ ^{2} steps.



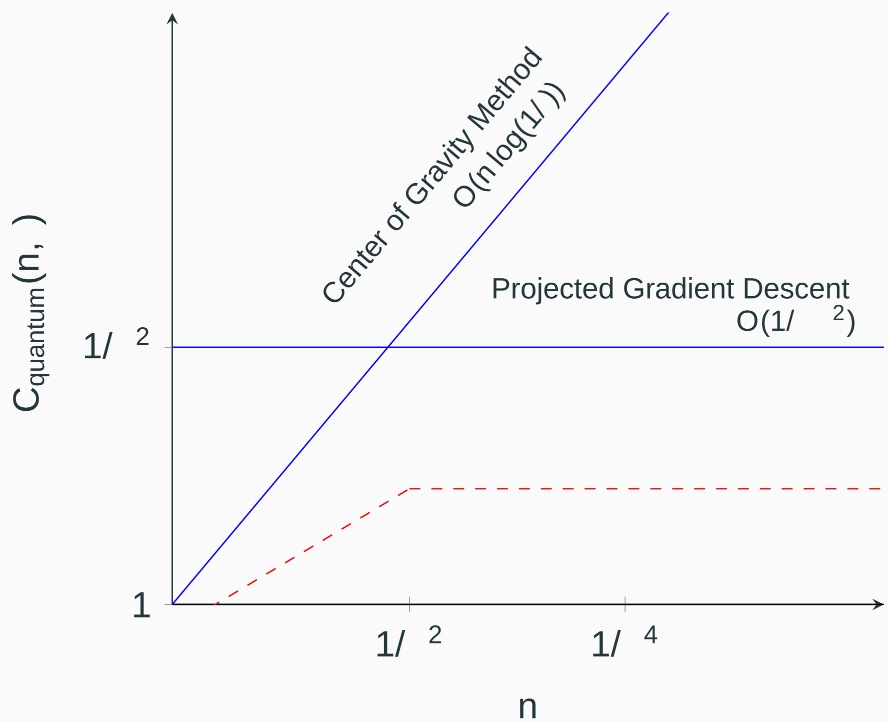
Known Algorithms II

Center of Gravity Method

n log(1/) steps.

Projected Subgradient Descent

1/ ^{2} steps.



The dimension-independent complexity is at least 1/ [CCLW ‘19].

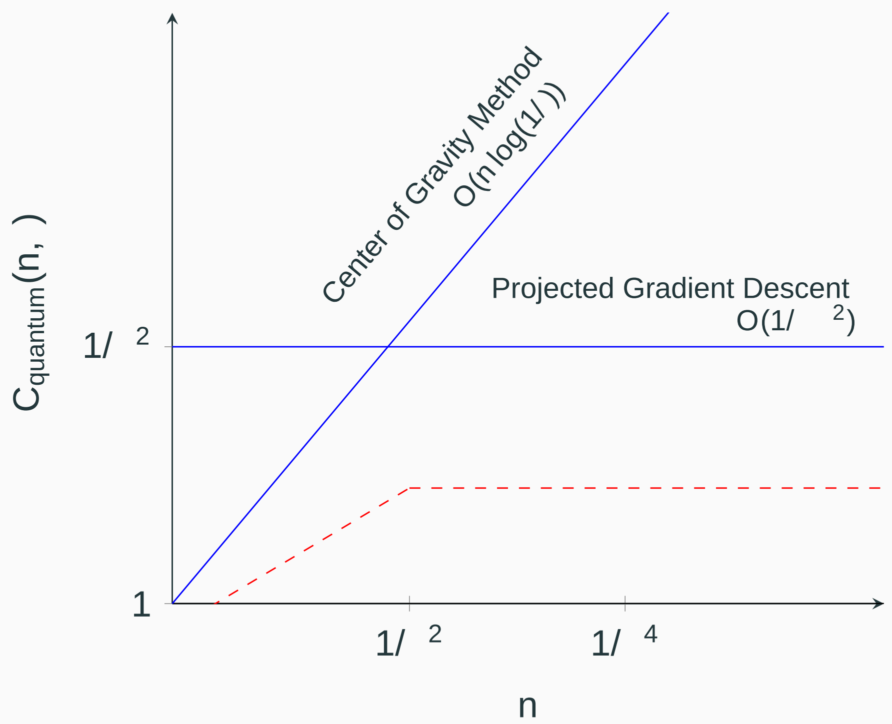
Known Algorithms II

Center of Gravity Method

n log(1/) steps.

Projected Subgradient Descent

$1/\ ^2$ steps.



The dimension-independent complexity is at least $1/\ ^2$ [GKNS '20].

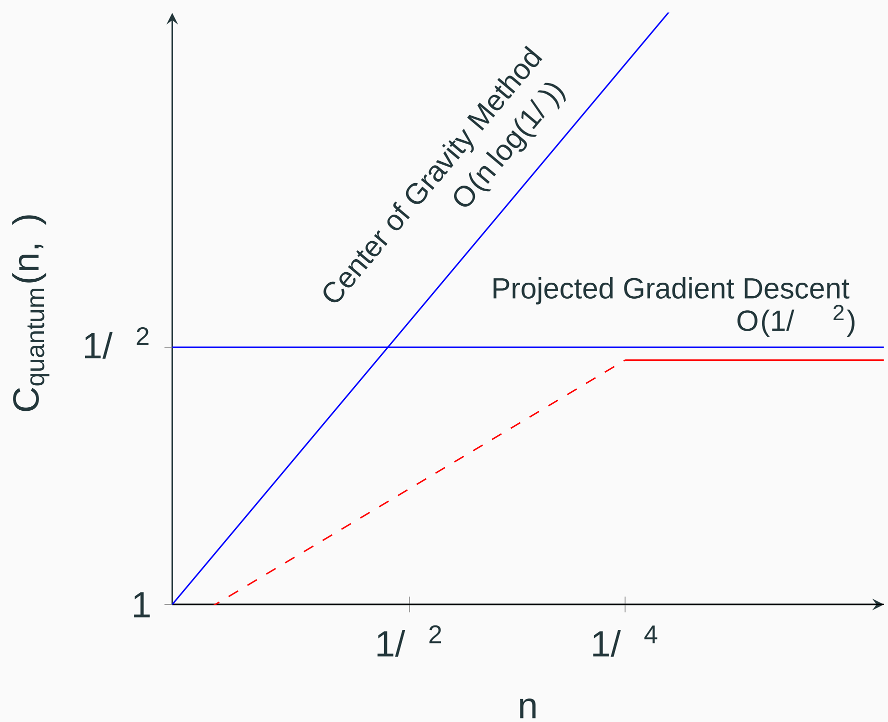
Known Algorithms II

Center of Gravity Method

n log(1/) steps.

Projected Subgradient Descent

$1/\ ^2$ steps.



$1/\ ^2$ is the correct complexity for $n > 1/\ ^4$ [GKNS '20].

Lower Bounds

The Base Function

$$f : R^n \ ! \ R$$

$$f(x) = \max\{x_1, x_2, \ldots, x_n\}.$$

The Base Function

$$f : \mathbb{R}^n \to \mathbb{R}$$

$$f(x) = \max\{x_1, x_2, \ldots, x_n\}.$$

$$\text{Minimum} = -\frac{1}{\sqrt{n}}, \text{ at } x = \left(-\frac{1}{\sqrt{n}}, \ldots, -\frac{1}{\sqrt{n}}\right).$$

The Base Function

$$f : \mathbb{R}^n \to \mathbb{R}$$

$$f(x) = \max\{x_1, x_2, \dots, x_n\}.$$

$$\text{Minimum} = \frac{1}{\sqrt{n}}, \text{ at } x = \frac{1}{\sqrt{n}}, \dots, \frac{1}{\sqrt{n}}.$$

If x_i is a maximum, then e_i is a subgradient.

Function Class

$$z\ 2\ \{+1,\ 1\}^n$$

$$f_z(x) = \max\{z_1 x_1, z_2 x_2, \ldots, z_n x_n\}.$$

Function Class

$$z \in \{+1, -1\}^n$$

$$f_z(x) = \max\{z_1 x_1, z_2 x_2, \ldots, z_n x_n\}.$$

$$\text{Minimum} = -\frac{1}{\sqrt{n}}, \text{ at } x = \left(-\frac{z_1}{\sqrt{n}}, \ldots, -\frac{z_n}{\sqrt{n}}\right).$$

$$\text{Set} \quad = \frac{0.9}{\sqrt{n}}.$$

Function Class

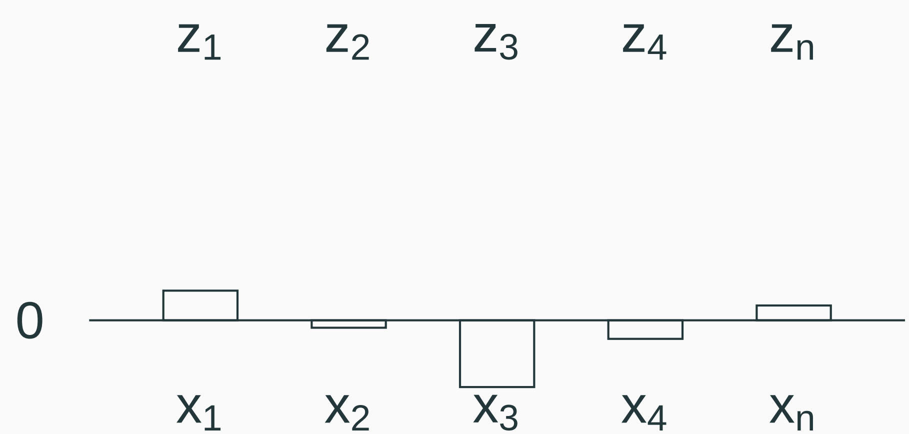
$$z \in \{+1, -1\}^n$$

$$f_z(x) = \max\{z_1 x_1, z_2 x_2, \ldots, z_n x_n\}.$$

$$\text{Minimum} = -\frac{1}{\sqrt{n}}, \text{ at } x = \left(\frac{z_1}{\sqrt{n}}, \ldots, \frac{z_n}{\sqrt{n}}\right).$$

$$\text{Set } \epsilon = \frac{0.9}{\sqrt{n}}.$$

The behaviour of f



Function Class

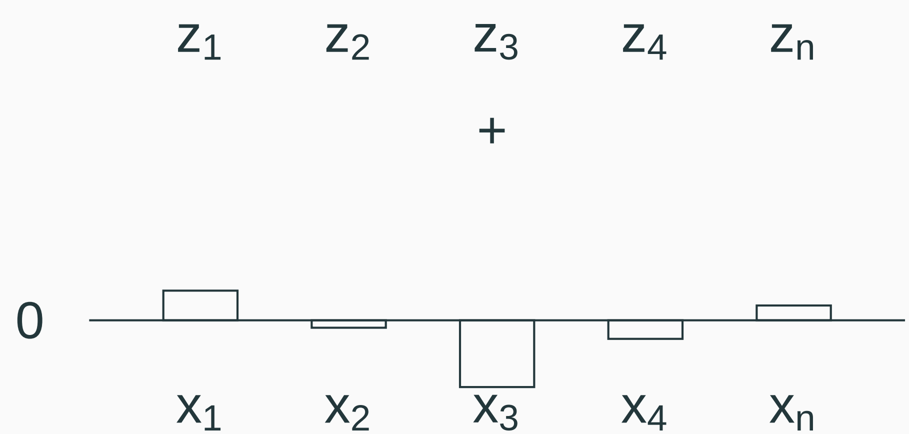
$$z \in \{+1, -1\}^n$$

$$f_z(x) = \max\{z_1 x_1, z_2 x_2, \ldots, z_n x_n\}.$$

$$\text{Minimum} = -\frac{1}{\sqrt{n}}, \text{ at } x = \left(-\frac{z_1}{\sqrt{n}}, \ldots, -\frac{z_n}{\sqrt{n}}\right).$$

$$\text{Set} \quad = \frac{0.9}{\sqrt{n}}.$$

The behaviour of f



Function Class

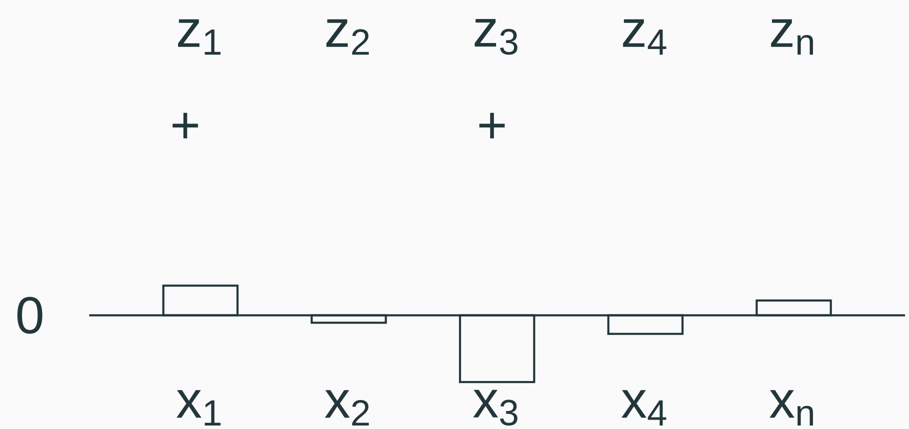
$$z \in \{+1, -1\}^n$$

$$f_z(x) = \max\{z_1 x_1, z_2 x_2, \ldots, z_n x_n\}.$$

$$\text{Minimum} = -\frac{1}{\sqrt{n}}, \text{ at } x = \left(-\frac{z_1}{\sqrt{n}}, \ldots, -\frac{z_n}{\sqrt{n}}\right).$$

$$\text{Set } \varepsilon = \frac{0.9}{\sqrt{n}}.$$

The behaviour of f



Function Class

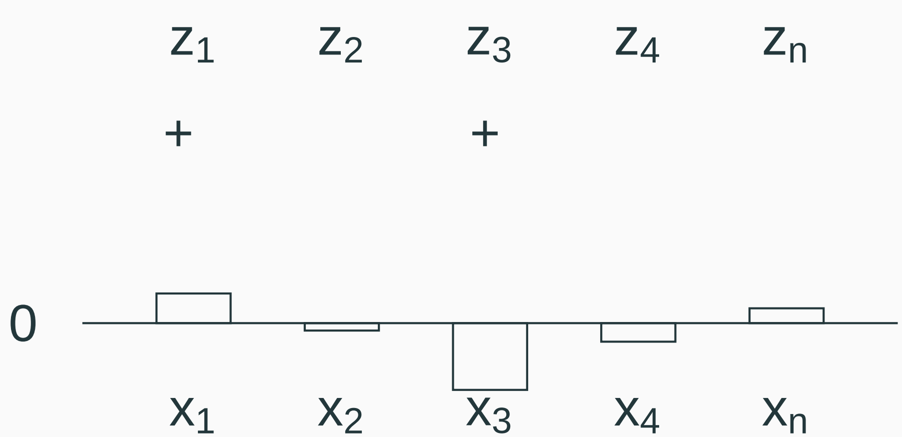
$$z \; 2 \; \{+1, \; 1\}^n$$

$$f_z(x) = \max\{z_1 x_1, z_2 x_2, \dots, z_n x_n\}.$$

$$\text{Minimum} = \frac{1}{\sqrt{n}}, \text{ at } x = \frac{z_1}{\sqrt{n}}, \dots, \frac{z_n}{\sqrt{n}} .$$

$$\text{Set} \quad = \frac{0.9}{\sqrt{n}}.$$

The behaviour of f



2 bits of z revealed per query.

Function Class

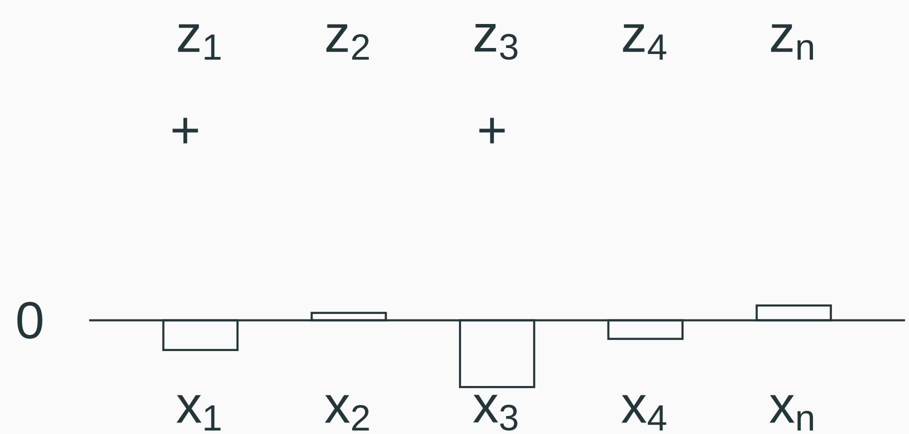
$$z \in \{+1, -1\}^n$$

$$f_z(x) = \max\{z_1 x_1, z_2 x_2, \ldots, z_n x_n\}.$$

$$\text{Minimum} = -\frac{1}{\sqrt{n}}, \text{ at } x = \left(-\frac{z_1}{\sqrt{n}}, \ldots, -\frac{z_n}{\sqrt{n}}\right).$$

$$\text{Set } \epsilon = \frac{0.9}{\sqrt{n}}.$$

The behaviour of f



Function Class

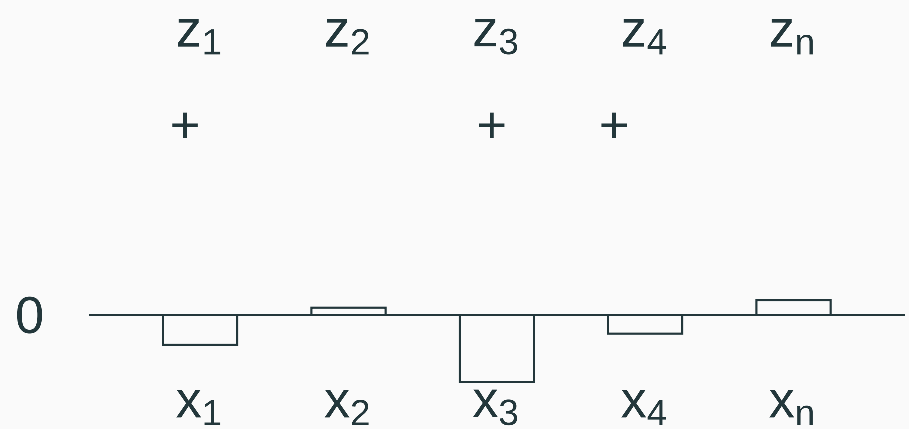
$$z \in \{+1, -1\}^n$$

$$f_z(x) = \max\{z_1 x_1, z_2 x_2, \ldots, z_n x_n\}.$$

$$\text{Minimum} = -\frac{1}{\sqrt{n}}, \text{ at } x = \left(-\frac{z_1}{\sqrt{n}}, \ldots, -\frac{z_n}{\sqrt{n}}\right).$$

$$\text{Set } \epsilon = \frac{0.9}{\sqrt{n}}.$$

The behaviour of f



Function Class

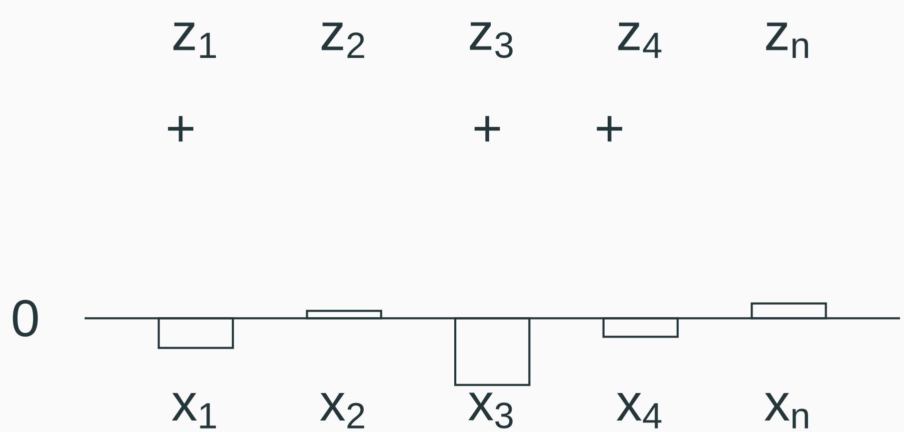
$$z \in \{+1, -1\}^n$$

$$f_z(x) = \max\{z_1 x_1, z_2 x_2, \ldots, z_n x_n\}.$$

$$\text{Minimum} = -\frac{1}{\sqrt{n}}, \text{ at } x = \left(-\frac{z_1}{\sqrt{n}}, \ldots, -\frac{z_n}{\sqrt{n}}\right).$$

$$\text{Set } \epsilon = \frac{0.9}{\sqrt{n}}.$$

The behaviour of f



Finding ϵ-optimal point $\Rightarrow$ learning z.

Function Class

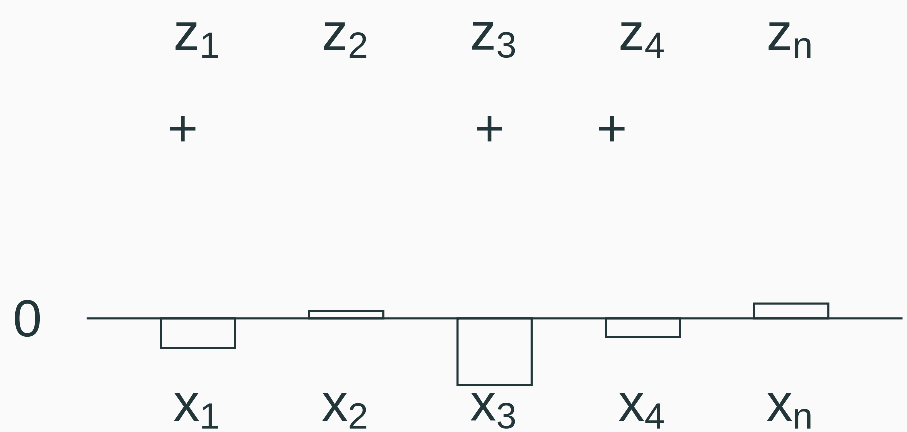
$$z\ 2\ \{+1,\ 1\}^{n}$$

$$f_z(x) = \max\{z_1x_1, z_2x_2, \ldots, z_nx_n\}.$$

Minimum = $\frac{1}{p\overline{n}}$, at x = $\frac{z_1}{p\overline{n}}, \ldots, \frac{z_n}{p\overline{n}}$.

Set = $\frac{0.9}{p\overline{n}}$.

The behaviour of f



Requires (n) = (1/ ^{2}) queries.

Quantum Speedup

Belovs' algorithm:

Given query access to ORs of z,

can find z in $\sqrt{n}$ queries.

Lower Bounds

A Sequential Lower Bound

Forcing Sequentiality: II

Nemirovsky Yudin 83

$$f_{v_1,v_2,\ldots,v_k}(x) = \max\{\langle v_1, x\rangle, \langle v_2, x\rangle \quad, \langle v_3, x\rangle\ 2\ , \cdots, \langle v_k, x\rangle\ (k\ 1)\ \}$$

where $\{v_1, \ldots, v_k\}$ form an orthonormal set in R^n.

Forcing Sequentiality: II

$f_{v_1,v_2,\ldots,v_k}(x) = \max\{\langle v_1, x\rangle, \langle v_2, x\rangle - \epsilon, \langle v_3, x\rangle - 2\epsilon, \cdots, \langle v_k, x\rangle - (k-1)\epsilon\}$

where $\{v_1, \ldots, v_k\}$ form an orthonormal set in $\mathbb{R}^n$.

$\max\{\langle v_1, x\rangle, \langle v_2, x\rangle, \ldots, \langle v_k, x\rangle\}$ takes a minimum value of $-\frac{1}{\sqrt{k}}$ in the unit ball.

Forcing Sequentiality: II

$$f_{v_1,v_2,\dots,v_k}(x) = \max\{\langle v_1, x\rangle, \langle v_2, x\rangle \ \ , \langle v_3, x\rangle\ 2\ , \cdots, \langle v_k, x\rangle\ (k\ \ 1)\ \}$$

where $\{v_1, \dots, v_k\}$ form an orthonormal set in $\mathbb{R}^n$.

$\max\{\langle v_1, x\rangle, \langle v_2, x\rangle, \dots, \langle v_k, x\rangle\}$ takes a minimum value of $\frac{1}{\sqrt{k}}$ in the unit ball.

$$1/k^{3/2}.$$

Forcing Sequentiality: II

$$f_{v_1,v_2,\ldots,v_k}(x) = \max\{\langle v_1, x\rangle, \langle v_2, x\rangle \ , \langle v_3, x\rangle\ 2\ , \cdots, \langle v_k, x\rangle\ (k\ \ 1)\ \}$$

where $\{v_1, \ldots, v_k\}$ form an orthonormal set in $\mathbb{R}^n$.

$\max\{\langle v_1, x\rangle, \langle v_2, x\rangle, \ldots, \langle v_k, x\rangle\}$ takes a minimum value of $\frac{1}{\sqrt{k}}$ in the unit ball.

$$1/k^{3/2}, \qquad \sqrt{\frac{\log n}{n}}.$$

Forcing Sequentiality: II

$$f_{v_1,v_2,\dots,v_k}(x) = \max\{\langle v_1, x\rangle, \langle v_2, x\rangle \;, \langle v_3, x\rangle \; 2 \;, \cdots, \langle v_k, x\rangle \; (k \; 1) \;\}$$

where $\{v_1, \dots, v_k\}$ form an orthonormal set in R^n.

$\max\{\langle v_1, x\rangle, \langle v_2, x\rangle, \dots, \langle v_k, x\rangle\}$ takes a minimum value of $\frac{1}{\sqrt{k}}$ in the unit ball.

$$1/k^{3/2}, \qquad \sqrt{\frac{\log n}{n}}.$$

Want to say that each vector has to be learnt in order.

The Hybrid Argument

Let A be an algorithm making k 1 queries.

A Run of A

Make query to

$$\max\{hv_1, xi, hv_2, xi \;\; , \cdots , hv \quad _k, xi\ (k \;\; 1)\ \}.$$

Make query to

$$\max\{hv_1, xi, hv_2, xi \;\; , \cdots , hv \quad _k, xi\ (k \;\; 1)\ \}.$$

$\vdots$

Make query to

$$\max\{hv_1, xi, hv_2, xi \;\; , \cdots , hv \quad _k, xi\ (k \;\; 1)\ \}.$$

The Hybrid Argument

Let A be an algorithm making k   1 queries.

A Run of A

Make query to

$\max\{hv_1, xi, hv_2, xi\ , \cdots , hv\ \ _k, xi\ (k\ \ 1)\ \}.$

Make query to

$\max\{hv_1, xi, hv_2, xi\ , \cdots , hv\ \ _k, xi\ (k\ \ 1)\ \}.$

$\vdots$

Make query to

$\max\{hv_1, xi, hv_2, xi\ , \cdots , hv\ \ _k, xi\ (k\ \ 1)\ \}.$

Once-Corrupted Run of A

Make query to

$\max\{hv_1, xi\}.$

Make query to

$\max\{hv_1, xi, hv_2, xi\ , \cdots , hv\ \ _k, xi\ (k\ \ 1)\ \}.$

$\vdots$

Make query to

$\max\{hv_1, xi, hv_2, xi\ , \cdots , hv\ \ _k, xi\ (k\ \ 1)\ \}.$

The Hybrid Argument

Let A be an algorithm making k 1 queries.

Once-Corrupted Run of A

Make query to

max{hv$_1$, xi}.

Make query to

max{hv$_1$, xi, hv$_2$, xi , · · · , hv $_k$, xi (k 1) }.

⋮

Make query to

max{hv$_1$, xi, hv$_2$, xi , · · · , hv $_k$, xi (k 1) }.

Twice-Corrupted Run of A

Make query to

max{hv$_1$, xi}.

Make query to

max{hv$_1$, xi, hv$_2$, xi }.

⋮

Make query to

max{hv$_1$, xi, hv$_2$, xi , · · · , hv $_k$, xi (k 1) }.

The Hybrid Argument

Let A be an algorithm making $k - 1$ queries.

$k - 1$-times Corrupted Run of A

Make query to

$\max\{\langle v_1, x\rangle\}.$

Make query to

$\max\{\langle v_1, x\rangle, \langle v_2, x\rangle \quad \}.$

$\vdots$

Make query to

$\max\{\langle v_1, x\rangle, \langle v_2, x\rangle \quad , \cdots , \langle v \quad _{k-1} \quad , x\rangle \quad (k-2) \}.$

Quantum Algorithms

- Can make queries in superposition.
- The state of the algorithm is represented by a vector.
- All quantum operations are unitary.

Quantum Algorithms

- Can make queries in superposition.
- The state of the algorithm is represented by a vector.
- All quantum operations are unitary.
  Given two states $_1$ and $_2$ such that k $_1$ $_2$k = c, then after applying the same quantum operations on both, the resulting states also have distance c.

NB

- Actual function used is slightly modi ed to account for queries outside the unit ball.

NB

- Actual function used is slightly modi ed to account for queries outside the unit ball.
- n can be as small as 1/ ^{6} for the above argument.

- Actual function used is slightly modi ed to account for queries outside the unit ball.
- n can be as small as 1/ ^{6} for the above argument.
- Can bring n down to 1/ ^{4} using a clever trick from
  “Complexity of Highly Parallel Non-Smooth Convex Optimization”
  - Sébastien Bubeck, Qijia Jiang, Yin Tat Lee, Yuanzhi Li, Aaron Sidford

Smooth Convex Optimization

Higher-Order Convex Optimization

- Promise: f is convex, p-times differentiable with $\nabla^p f$ being L_p-Lipschitz.
- Query access to $O_f : x \mapsto (f(x), \nabla f(x), \nabla^2 f(x), \ldots, \nabla^p f(x))$.

The Setting	The Upper Bound	The Lower Bound
$p = 1$	$O(\sqrt{L_1/\epsilon})$	Det: $\Omega(\sqrt{L_1/\epsilon})$ Rand: $\Omega(\sqrt{L_1/\epsilon})$ Quant: -
$p = 2$	$O(\sqrt[7/2]{L_2/\epsilon})$	Det: $\Omega(\sqrt[7/2]{L_2/\epsilon})$ Rand: $\Omega(\sqrt[11/2]{L_2/\epsilon})$ Quant: -
p	$O(\sqrt[(3p+1)/2]{L_p/\epsilon})$	Det: $\Omega(\sqrt[(3p+1)/2]{L_p/\epsilon})$ Rand: $\Omega(\sqrt[(5p+1)/2]{L_p/\epsilon})$ Quant: -

[Bubeck Jiang Lee Li Sidford ‘19]

[Gasnikov Dvurechensky Gorbunov Vorontsova Selikhanovych Uribe ‘19]

[Jiang Wang Zhang ‘19]

Our Contribution

We show that neither randomized nor quantum algorithms can do any better than deterministic algorithms.

$$\forall p \in \mathbb{N}, Q \qquad \tilde{\ }(D).$$

More about Query Complexity

Structure (for total Boolean functions $f : \{0,1\}^n \,!\, \{0,1\}$)

Degree of a function f:

$$\text{PARITY}(x_1, x_2, x_3) = 4x_1x_2x_3 \quad 2x_1x_2 \quad 2x_1x_3 \quad 2x_2x_3 + x_1 + x_2 + x_3.$$

Easy to prove: Query complexity of f degree of f.

Nisan Szegedy '92

Degree of f query complexity of f . (degree of f)^4.

Query complexity of f is large , Degree of f is large.

Also randomized query complexity, quantum query complexity, approximate degree, certi cate complexity, sensitivity.