CS-GY 6763: Lecture 7
Fine Grained Complexity

NYU Tandon School of Engineering, Feyza Duman Keles

Limitations for the Algorithms

Previously, our focus in class has been on algorithms and their theoretical guarantees of success.

The first application was CAPTCHAs,

Instead of counting all the cases, we developed a faster algorithm (counting duplicates) that requires less checking and gives true results with high probability.

Another problem was k-means clustering,

By using JL, we decreased the cost by $(1 + \epsilon)$ times.

Limitations for the Algorithms

Main question: Can we always improve algorithms if everyone works to their full potential, or will progress eventually stall?

There are some problems for which the best algorithms available today are still 40 years old. Until now, no one has been able to beat them.

Self-Attention

Self-Attention

For given 3 inputs: Query $Q \in \mathbb{R}^{n \times d_k}$, Key $K \in \mathbb{R}^{n \times d_k}$, and Value $V \in \mathbb{R}^{n \times d_v}$, the attention is calculated as

$$Attention(Q, K, V) = softmax\left(\frac{QK^T}{\sqrt{d_k}}\right) V$$

We will focus on the self-attention function as an example of fine-grained complexity.

Self-attention is the main layer in Transformers (Architectures of Chat-GPT, Dall-E, and many other deep models are based on Transformers.).

Self-Attention

For given 3 inputs: Query $Q \in \mathbb{R}^{n \times d_k}$, Key $K \in \mathbb{R}^{n \times d_k}$, and Value $V \in \mathbb{R}^{n \times d_v}$, the attention is calculated as

$$Attention(Q, K, V) = softmax\left(\frac{QK^T}{\sqrt{d_k}}\right) V$$

What is computational complexity?

Q matrix size $n \times d_k$

K^T matrix size $d_k \times n$

Brute force calculation is $O(n^2)$ because of the input for softmax.

Can we do better than quadratic?

| Model / Paper	Complexity
Memory Compressed[†] (Liu et al., 2018)	$\mathcal{O}(n_c^2)$
Image Transformer[†] (Parmar et al., 2018)	$\mathcal{O}(n.m)$
Set Transformer[†] (Lee et al., 2019)	$\mathcal{O}(nk)$
Transformer-XL[†] (Dai et al., 2019)	$\mathcal{O}(n^2)$
Sparse Transformer (Child et al., 2019)	$\mathcal{O}(n\sqrt{n})$
Reformer[†] (Kitaev et al., 2020)	$\mathcal{O}(n \log n)$
Routing Transformer (Roy et al., 2020)	$\mathcal{O}(n \log n)$
Axial Transformer (Ho et al., 2019)	$\mathcal{O}(n\sqrt{n})$
Compressive Transformer[†] (Rae et al., 2020)	$\mathcal{O}(n^2)$
Sinkhorn Transformer[†] (Tay et al., 2020b)	$\mathcal{O}(b^2)$
Longformer (Beltagy et al., 2020)	$\mathcal{O}(n(k + m))$
ETC (Ainslie et al., 2020)	$\mathcal{O}(n_g^2 + nn_g)$
Synthesizer (Tay et al., 2020a)	$\mathcal{O}(n^2)$
Performer (Choromanski et al., 2020)	$\mathcal{O}(n)$
Linformer (Wang et al., 2020b)	$\mathcal{O}(n)$
Linear Transformers[†] (Katharopoulos et al., 2020)	$\mathcal{O}(n)$
Big Bird (Zaheer et al., 2020)	$\mathcal{O}(n)$

Theoretical guarantees?

Limitations for the Algorithms

How can we say no one can have a faster algorithm for a problem?

It is not easy to prove directly.

Can we prove there is no other life in the universe other than Earth?
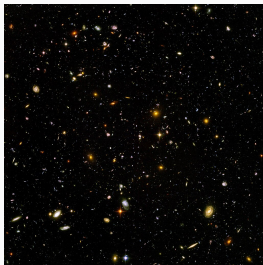

We have to look at every inch of the universe (until we find life), if there is nothing then we can conclude that there is no life.

Limitations for the Algorithms

However, if we know that some problems are hard, then we can relate our current problem with the known problems.

Consider this scenario; Mars has been observed extensively, and while it has not been conclusively proven, it is widely believed that there is no life on the planet. By comparing the environmental conditions of Jupiter and Mars, one could hypothesize that if life were to exist on Jupiter, then it could also exist on Mars because the conditions on Mars are more favorable.
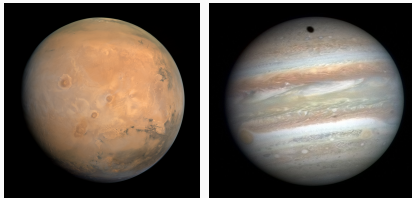
Limitations for the Algorithms

Similarly, there are some fundamental questions (k-SAT, k-clique, etc.) and no one will be able to improve their algorithms for decades. And people conjectured their hardness. And extend their assumptions to other problems.

P ≠ NP

The main conjecture in complexity theory. (Levin-Cook 1971)

In informal terms, it asks whether every problem whose solution can be quickly verified can also be quickly solved.

P = problems can be solved in poly-time. $O(n)$, $O(n^k)$, etc.

NP = problems can be verified in poly-time.

Boolean Satisfiability Problem (SAT)

Consider a Boolean function $f(x_1, x_2, \ldots, x_n) : \{0,1\}^n \to \{0,1\}$

If the function has some instances to make $f(x_1, x_2, \ldots, x_n) = 1$, then it is called satisfiable.

$f(x_1, x_2) = x_1 \wedge x_2'$ is satisfiable ($x_1 = 1$ and $x_2 = 0$)

$f(x_1, x_2) = x_1 \wedge x_1'$ is un-satisfiable.

It is the first proven NP-complete problem. (Cook/Levin Theorem)

Conjunctive Normal Form (CNF): conjunction of clauses

$$(x_1 \vee x_2) \wedge (x_1' \vee x_3) \wedge (x_2 \vee x_3' \vee x_4) \quad \leftarrow \text{3-SAT}$$

k-SAT: In each clause, there are are most k literals.
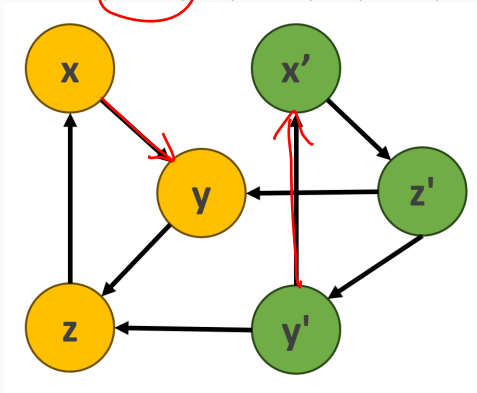
2-SAT is in P.

3-SAT is NP-complete. (Also, when $P \neq NP$, 3-SAT complexity is not in P)

2-SAT in P

Construct directed graph $G(V, E)$ on $2n$ edges, by using the formula:

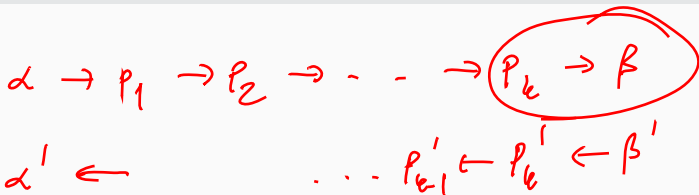$(a \lor b)$ is same with $(a' \Rightarrow b)$ and also $(b' \Rightarrow a)$

Example: $f = (x' \lor y) \land (y' \lor y) \land (x \lor z') \land (z \lor y)$ ⇐ 2-SAT

$x \Rightarrow y$

$y' \Rightarrow x'$

Claim 1. If G contains a path from α to β, then it also contains a path from β' to α'.

$$\alpha \rightarrow p_1 \rightarrow p_2 \rightarrow \cdots \rightarrow p_k \rightarrow \beta$$

$$\alpha' \leftarrow \qquad \cdots \; p_{k-1}' \leftarrow p_k' \leftarrow \beta'$$
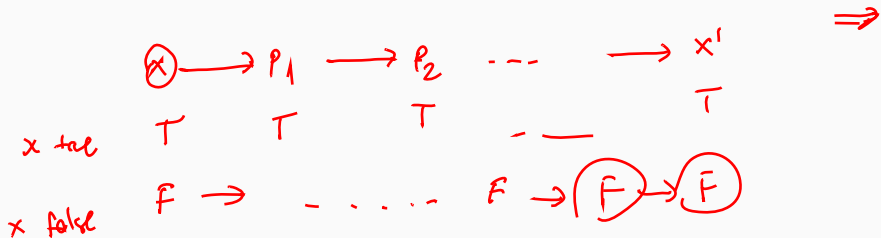
$O(n^4)$

Claim 2. A 2-CNF function f is unsatisfiable iff there exists a variable x, such that:

1. There is a path from x to x' in the graph

2. There is a path from x' to x in the graph

Path from x to x' and f is satisfiable.

$$\circledast \longrightarrow P_1 \longrightarrow P_2 \; \text{---} \; \longrightarrow x'$$

$\Longrightarrow$

x true $\top$ $\top$ $\top$ $\top$

x false $F \Rightarrow$ — . . . — $F \Rightarrow \boxed{F} \rightarrow \boxed{F}$

3-SAT is NP-complete.

There is a reduction from SAT (Cook/Levin Theorem) to 3-SAT.

Consider a clause with m variables: $x_1 \vee x_2 \vee \cdots \vee x_m$.

We can express it as the construction of $m - 2$ clauses:

$(x_1 \vee x_2 \vee y_2)\wedge$

$\quad (y_2 \vee x_3 \vee y_3)\wedge$

$\quad (y_3 \vee x_4 \vee y_4)\wedge$

$\quad \quad \cdots$

$\quad (y_{n-3} \vee x_{n-2} \vee y_{n-2})\wedge$

$\quad (y_{n-2} \vee x_{n-1} \vee y_n)\wedge$

where $y_2, y_3, \ldots y_{n-2}$ are fresh variables not occurring elsewhere.

$k\text{-SAT}$
$\downarrow$
2-SAT

Is $P \neq NP$ adequate?

Even polynomial time might not be efficient,

- $O(n^2)$-time algorithm can take 1000 CPU years for $n = 10^9$
- So we need almost linear time.

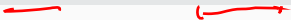$P \neq NP$ cannot distinguish $O(n)$ and $O(n^{100})$, both in polynomial time.

The best-known algorithms for k-SAT run in exponential time in n (the number of variables).

The hardness assumption: better algorithms do not exist.

Exponential Time Hypothesis(ETH)

For any k, there is a constant s_k, such that the the time complexity of n variable k-SAT is at least $O(2^{s_k n})$.

Besides, the best algorithms known for k-SAT have exponents that approach n as k increases.

Strong Exponential Time Hypothesis(SETH)

For every ϵ, there is a k, such that n variable k-SAT requires more than $O(2^{(1-\epsilon)n})$ time.

This is a well-accepted hypothesis among the researchers in this area.

All Pairs Shortest Path(APSP)

There is no $O(n^{3-\epsilon})$-time algorithm to find the shortest path between all pairs in an n-vertices graph.

3SUM

There is no $O(n^{2-\epsilon})$-time algorithm to determine if there is a triplet in n numbers that their sum is 0.

k-clique

There is no $O(n^{(1-\epsilon)k})$-time algorithm to determine if there is a k-clique in an n-vertices graph.

$$\binom{n}{k} = O($$
$$4^k)$$

Orthogonal Vectors(OV)

There is no $O(n^{2-\epsilon})$-time algorithm to determine if there is an orthogonal pair between 2 sets that both include n vectors.

Orthogonal Vectors Problem (Formal Definition)

Given two sets: $A = \{a_1, a_2, \ldots, a_n\} \subseteq \{0, 1\}^d$ and
$B = \{b_1, b_2, \ldots, b_n\} \subseteq \{0, 1\}^d$ of n binary vectors.

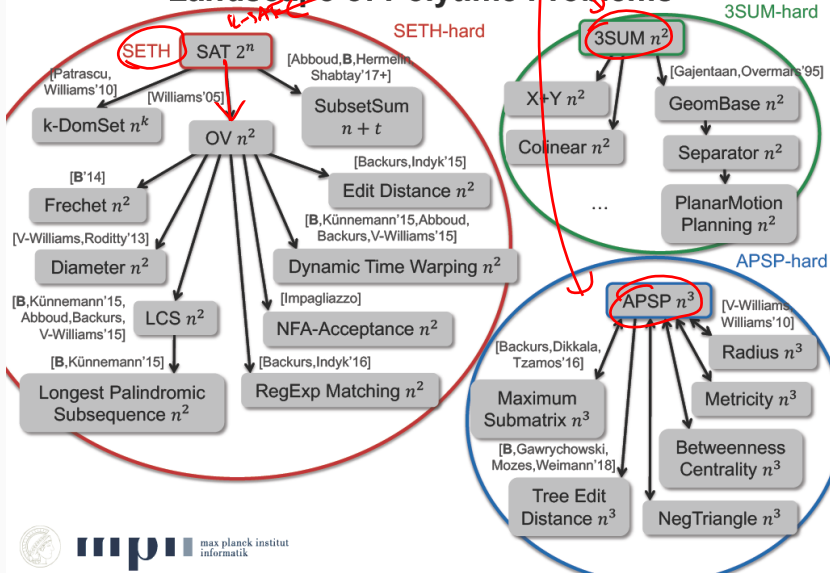Decide if there exists a pair $a \in A$ and $b \in B$ such that $a^T b = 0$

$O(n^2 d)$ baseline algorithm that checks the inner products of all possible pairs of vectors.

Theorem (Williams, 2005)

Assume SETH. Then for all ϵ, there is no $O(n^{2-\epsilon})$-time algorithm solving OV problem, for $d = \omega(\log n)$.
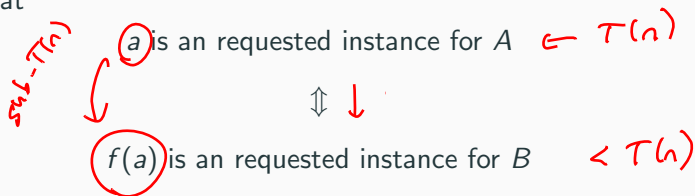
Landscape of Polytime Problems

A reduction from problem A to another problem B, is a transformation f of any instance a of A into an instance $f(a)$ of B, such that

a is an requested instance for A $\leftarrow T(n)$

$\Updownarrow$

$f(a)$ is an requested instance for B $\quad < T(n)$

i.e. Suppose we have an information that problem A cannot be solved in $T(n)$-time. Also, suppose construction of $f(a)$ from a requires sub-$T(n)$ time. What can we say about problem B?

Threshold Vectors Product Problem

Threshold Vectors Product Problem (TVPP)

Given two sets: $A = \{a_1, a_2, \ldots, a_n\} \subseteq \{0,1\}^d$ and
$B = \{b_1, b_2, \ldots, b_n\} \subseteq \{0,1\}^d$ of n binary vectors.

Decide if there exists a pair $a \in A$ and $b \in B$ such that $a^T b \geq t$

$$a^T b = 0$$

Lemma 1. TVPP Hardness (Keles et al. 2023)

TVPP: Given two sets $A = \{a_1, a_2, \ldots, a_n\} \subseteq \{0,1\}^d$ and $B = \{b_1, b_2, \ldots, b_n\} \subseteq \{0,1\}^d$ of n binary vectors. Decide if there exists a pair $a \in A$ and $b \in B$ such that $a^T b \geq t$

Assume SETH. Then for all ϵ, there is no $O(n^{2-\epsilon})$-time algorithm solving TVPP problem, for $d = \omega(\log n)$.

Proof.

(Handwritten annotations:)

$$\bar{a_i} = \boxed{a_i} \quad \boxed{1-a_i}$$
$$00 k \qquad 11 \; 00$$

n^2 alg

$SETH \rightarrow OV \cdots \rightarrow TVPP$

$A = \{a_1 \ldots a_n\} \quad B = \{b_1 \ldots b_n\} \subseteq \{0,1\}^d$ for OV problem

$$\bar{a_i} = \boxed{a_i} \quad \boxed{1-a_i} \longrightarrow \subseteq \{0,1\}^{2d} \quad a_i = 01 \quad \bar{a_i} = 0110$$

$$\bar{b_j} = \boxed{1-b_j} \quad \boxed{11 \cdots}$$

$|\bar{a_i}| = d \quad \langle a_i, b_j \rangle = 0$

$t = d \leq \bar{a_i}^T \cdot \bar{b_j} = \langle a_i, 1-b_j \rangle + \langle 1-a_i, 1 \rangle$

$|a_i| - \langle a_i, b_j \rangle \quad |1-a_i| = d - \langle a_i, b_j \rangle \geq d$

25

Definition. Exponential Dot Product Self-Attention

For any $i, j = 1, 2, \ldots n$, let $S_{ij} = e^{Q_i^T Q_j}$ and Y=SV be the self-attention mechanism.
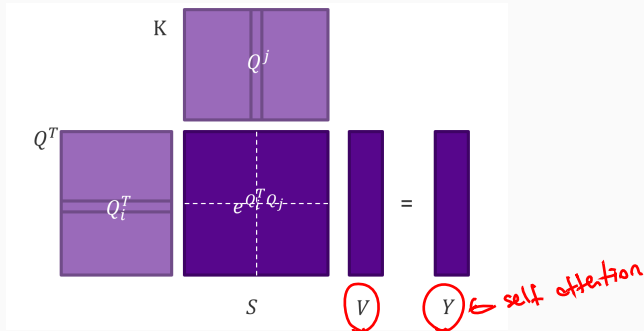
Theorem. Exponential Dot Product Self-Attention Hardness (Keles et al. 2023)
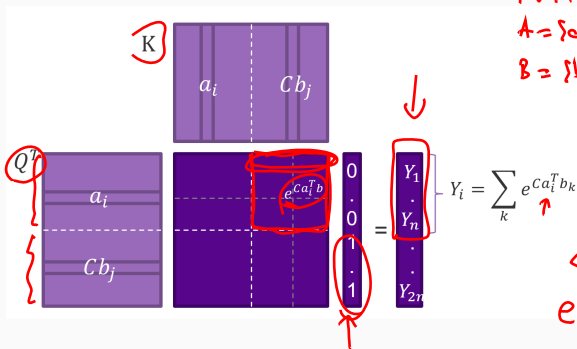
For any $i, j = 1, 2, \ldots n$, let $S_{ij} = e^{Q_i^T Q_j}$ and Y=SV be the self-attention mechanism. Provided $d = \omega(\log n)$. For any ϵ, computing Y requires $\Omega(n^{2(1-\epsilon)})$-time.

$$Y_i = \sum_k e^{Ca_i^T b_k}$$

Handwritten annotations:

Y

$Y(1 \pm \epsilon)$

TVPP

$A = \{a_1 \ldots a_n\} \subseteq (0,1)^q$

$B = \{b_1 \ldots b_n\}$

$a_i^T \cdot b_j \geq t$

$\langle a_i, C b_j \rangle$

e

Proof.

$$Y_i = \sum_{\boxed{k}} e^{c a_i^T b_i}$$

$$\exists a, b \quad a^T b \geq t \quad \Rightarrow \quad Y_i \geq e^{ct}$$

$$\text{no } a, b \quad a^T b \geq t \quad a^T b \leq t-1 \quad \forall a, b$$

$$Y_i \leq n \cdot e^{c(t-1)}$$

$$c = \log 2n \qquad n \cdot e^{c(t-1)} < e^{ct}$$

Fine-grained complexity is not common in Machine Learning/Deep Learning Area. Some other works include;

Other Types of Self-Attention

(Window Sliding, Gaussian Kernel)

→ Inverse Generative Neural Networks e

Viterbi Algorithm $O(T \sigma^2)$

Gaussian Kernel Density Estimation

Approximated Nearest Neighbor Search

$$e^{q_i^T b_j}$$
$$e^{-c\|q_i - b_j\|_2}$$