The Fine-Grained Complexity of Gradient Computation for Training Large Language Models

Josh Alman
Department of Computer Science
Columbia University
josh@cs.columbia.edu

Zhao Song
Simons Institute for the Theory of Computing
University of California, Berkeley
magic.linuxkde@gmail.com

Abstract

Large language models (LLMs) have made fundamental contributions over the last a few years. To train an LLM, one needs to alternatingly run 'forward' computations and 'backward' computations. The forward computation can be viewed as attention function evaluation, and the backward computation can be viewed as a gradient computation. In previous work by [Alman and Song, NeurIPS 2023], it was proved that the forward step can be performed in almost-linear time in certain parameter regimes, but that there is no truly sub-quadratic time algorithm in the remaining parameter regimes unless the popular hypothesis SETH is false. In this work, we show nearly identical results for the harder-seeming problem of computing the gradient of loss function of one layer attention network, and thus for the entire process of LLM training. This completely characterizes the fine-grained complexity of every step of LLM training.

1 Introduction

Large language models (LLMs) have emerged as popular technologies, driving breakthroughs across many applications in natural language processing, computer vision, translation, and many other areas [VSP$^+$17, DCLT18, LOG$^+$19, YDY$^+$19, BMR$^+$20, JZLD21, ZRG$^+$22, CND$^+$22, TLI$^+$23, TMS$^+$23, Man23, TDFH$^+$22, YCRI22, WTB$^+$22, WSD$^+$23, WCZ$^+$23, ZJL$^+$23, ZWH$^+$24, LLS$^+$24a, XSL24, CLL$^+$24, WMS$^+$24]. The training of these models is a computationally intensive process, characterized by alternating between two primary operations: forward computation and backward computation. Forward computation, or function evaluation, involves the propagation of input data through the network to generate predictions. Conversely, backward computation, or gradient computation, is the process of calculating the gradient of the loss function with respect to the model's parameters, facilitating the optimization of these parameters during training.

The efficiency of these computations directly impacts the feasibility and scalability of training LLMs, particularly as models grow in size and complexity. Recent work by [AS23, AS24c, AS24a] has carefully studied the *forward* computation step. They demonstrated a sharp computational boundary, showing that how quickly the forward steps can be performed depends critically on how large the entries are of the matrices which define the model parameters. They showed a near-linear time algorithm when these entries are small, and also proved that when the entries are large, there is no algorithm much faster than the trivial algorithm, assuming the Strong Exponential Time Hypothesis (SETH) [IP01] holds.

The Strong Exponential Time Hypothesis (SETH) was introduced by Impagliazzo and Paturi [IP01] over 20 years ago. It is a strengthening of the $\mathsf{P} \neq \mathsf{NP}$ conjecture, and asserts that our current best SAT algorithms are roughly optimal (for detailed statement, see Hypothesis 3.1 below). SETH is a popular conjecture from fine-grained complexity theory which has been used to prove lower bounds for a wide variety of algorithmic problems. See, for instance, the survey [Wil18].

38th Conference on Neural Information Processing Systems (NeurIPS 2024).

https://doi.org/10.52202/079017-1963

In other words, in some parameter regimes, the algorithm of [AS23] performs the forward steps about as quickly as one could hope for, whereas in other regimes, assuming SETH, it is impossible to design a nontrivially fast algorithm. However, this leaves open many important questions about LLM training. In the case when forward computation can be done quickly, can the same be said for backward computation? If not, then the entire training process would still be slow. Relatedly, in parameter regimes where forward computation is known to be hard, is backward computation also hard? If not, perhaps heuristic tricks could be used, or other details of the model could be modified, to speed up the overall training. As we will see shortly, the backward step is defined in a much more complicated way than the forward step, and it is not evident that algorithms or lower bounds for one extend to the other.

Our study aims to resolve these questions and determine the fine-grained complexity of the backward computation phase. Our main result (which we state more formally shortly) shows that the same computational threshold from forward computation also arises for the backward problem, and that the problems are easy (or hard) in the exact same parameter regimes. Thus, the forward algorithm of [AS23] can be combined with our novel backward algorithm to perform each training step for LLMs in almost linear time when the parameter matrix entries are small enough, whereas when the entries are not small enough, neither step can be performed quickly.

In addition to characterizing the fine-grained complexity of LLM training, our result for gradient computation is novel for a few reasons.

- Previous work on computational lower bounds only focuses on forward computations, see [AS23, KWH23, AS24c, AS24a]. To our knowledge, ours is the first work to prove hardness of a backward computation step for training an LLM or similar model.
- There has been previous work on algorithms for backward/gradient computation problems [BPSW21, SYZ21, DHS$^+$22, ALS$^+$23, GQSW24, SZZ24]. That said, most of these works focus on backwards computation in other settings. The only previous work we're aware of that studies the optimization of attention layers (for LLMs) is [GSWY23], which uses Newton methods that rely on Hessian computations. However, Hessian computation is substantially more expensive than gradient computation; our algorithm and results apply directly to the gradient computation and get around the Hessian "barrier", allowing for faster algorithms in some parameter regimes, and more powerful lower bounds in others.

Bounded entries. Our result proves that the size of the entries of the matrices defining the LLM play a substantial role in determining how quickly LLM training can be performed. Prior work on LLM implementations has observed a similar phenomenon, that algorithmic techniques like quantization [ZBIW19, HCL$^+$24] and low-degree polynomial approximation [KVPF20], which *require* bounded or low-precision entries, can substantially speed up LLM operations. See, for instance, the discussion of these phenomena in [ZBIW19, Section 2] and [KVPF20, Section 3.2.1]. Our work can be viewed as giving a theoretical explanation for this phenomenon.

Polynomial approximation. Our new algorithmic approach, which uses a polynomial to approximate the softmax function, is also not unlike algorithms which have found success in practice [BGVM20, KWH23, ZBKR24]. For example, see detailed discussions in in [ZBKR24, Section 4.1]. Our new algorithm improves on these approaches by using *theoretically optimal* polynomials for softmax, and combining them with a number of linear algebraic techniques, to give provable guarantees about their correctness and near linear running time.

Follow-up work of this paper. Recently, a number of works have considered different extensions of this paper. [LSSZ24] extends our analysis into tensor attention gradient computation and [LSS$^+$24] extends our results to multi-layer Transformers. On the other hand, [LLS$^+$24b] borrows our techniques and provides a fine-grained attention I/O complexity for attention backward. [LLS$^+$24c] uses our techniques to provide a fast attention gradient approximation based on Fourier transform. [LLS$^+$24d] computes a sparse attention matrix based on our analysis as well.

1.1 Problem Definition

Before formally stating our results, we begin by precisely defining the problems we study. We begin with the following problem of computing a general Attention forward layer.

https://doi.org/10.52202/079017-1963

Definition 1.1 (ℓ-th layer forward computation)**.** *Given weights $Q, K, V \in \mathbb{R}^{d \times d}$, and letting $E_\ell \in \mathbb{R}^{n \times d}$ denote the ℓ-th layer input, then $E_{\ell+1} \in \mathbb{R}^{n \times d}$ is defined recursively as*

$$E_{\ell+1} \leftarrow D^{-1} \exp(E_\ell Q K^\top E_\ell^\top / d) E_\ell V$$

where

- $D := \mathrm{diag}(\exp(E_\ell Q K^\top E_\ell^\top / d) \mathbf{1}_n)$.
- $\exp$ *denotes the exponential function which is applied entry-wise, i.e.,* $\exp(A)_{i,j} = \exp(A_{i,j})$ *for all matrices A.*
- $\mathrm{diag}()$ *operation takes a vector as input and generates a diagonal matrix with the entries of that vector.*
- $\mathbf{1}_n$ *denotes the length-n all ones vector.*

In mathematical terms, optimization in the context of attention computation is described as (by renaming the $QK^\top \in \mathbb{R}^{d \times d}$ to be $X \in \mathbb{R}^{d \times d}$ and $V \in \mathbb{R}^{d \times d}$ to be $Y \in \mathbb{R}^{d \times d}$):

Definition 1.2 (Attention optimization)**.** *Given four $n \times d$ size matrices A_1, A_2, A_3 and $E \in \mathbb{R}^{n \times d}$. Suppose that a $d \times d$ size square matrix $Y \in \mathbb{R}$ is also given. The attention optimization problem is formulated as:*

$$\min_{X \in \mathbb{R}^{d \times d}} L(X) := 0.5 \| D(X)^{-1} \exp(A_1 X A_2^\top / d) A_3 Y - E \|_F^2.$$

Here $D(X) \in \mathbb{R}^{n \times n}$ is

$$D(X) := \mathrm{diag}(\exp(A_1 X A_2^\top / d) \mathbf{1}_n).$$

and $\| \cdot \|_F^2$ denotes the squared Frobenius norm, i.e., $\|A\|_F^2 := \sum_{i,j} A_{i,j}^2$.

Remark 1.3. *In principle, the loss function above, and resulting gradients below, should depend on both X and Y. However, since the final matrix computed in the norm in L depends only linearly on Y, it is straightforward to incorporate it into either an algorithm or lower bound. Thus, in this work, we focus on the case where X is variable and Y is a fixed input to simplify some arguments.*

We thus define the Approximate Attention Loss function Gradient Computation problem as follows:

Definition 1.4 (Approximate Attention Loss Gradient Computation ($\mathsf{AAttLGC}(n, d, \epsilon)$))**.** *Given four $n \times d$ size matrices $A_1 \in \mathbb{R}^{n \times d}, A_2 \in \mathbb{R}^{n \times d}, A_3 \in \mathbb{R}^{n \times d}$,, $E \in \mathbb{R}^{n \times d}$ and a square matrix $Y \in \mathbb{R}^{d \times d}$, which we think of as fixed matrices. Assume that $\|A_1 X\|_\infty \leq B$, $\|A_2\|_\infty \leq B$ for a positive parameter B. Further assume that all the entries of these matrices can be represented as $O(\log n)$-bit rational numbers. Let $L(X)$ be defined as Definition 1.2. Let $\frac{\mathrm{d}L(X)}{\mathrm{d}X}$ denote the gradient of loss function $L(x)$.*

The goal is to output a vector $\widetilde{g}$ such that

$$\| \widetilde{g} - \frac{\mathrm{d}L(X)}{\mathrm{d}X} \|_\infty \leq \epsilon.$$

Here for matrix A, $\|A\|_\infty := \max_{i,j} |A_{i,j}|$.

1.2 Main Results

Our main results show that there is a threshold in the computational complexity of $\mathsf{AAttLGC}(n, d = O(\log n))$ depending on the bound B. When $B = o(\sqrt{\log n})$ we give a new near-linear-time algorithm, and when $B = \omega(\sqrt{\log n})$, we show that such an algorithm is impossible assuming SETH. This matches the results of [AS23], where a nearly identical threshold at B around $\sqrt{\log n}$ was also observed. Our results therefore imply that the entire LLM training process has this computational threshold.

Theorem 1.5 (Main result, Lower bound, informal version of Theorem E.5)**.** *Assuming* SETH*, there is no algorithm running in time $O(n^{2-q})$ for any $q > 0$ for the* $\mathsf{AAttLGC}(n, d = O(\log n), B = \omega(\sqrt{\log n}))$ *(see Definition 1.4).*

https://doi.org/10.52202/079017-1963

Theorem 1.6 (Main result, Upper bound, informal version of Theorem D.6)**.** *Assuming entries are bounded, there is a* $n^{1+o(1)}$ *time algorithm to solve* $\mathsf{AAttLGC}(n, d = O(\log n), B = o(\sqrt{\log n}))$ *(see Definition 1.4) up to* $1/\operatorname{poly}(n)$ *accuracy.*

Our new algorithm (Theorem 1.6) builds on a low-rank approximation for the attention matrix from prior work [AA22, AS23]. Incorporating these approximation into the gradient computation is not straightforward; in the forward problem, one simply multiplies the attention matrix by an input value matrix, but in the backward problem, it is combined with other matrices in an intricate (non-linear) way. We ultimately use tools from tensor algebra to get a handle on the entry-wise products and high-rank sparse matrices which arise in the gradient computation but do not typically preserve the needed low-rank structure.

Our new lower bound (Theorem 1.5) comes from a careful reduction from a special case the forward problem (where hardness is known from prior work) to the backward problem. Reducing from computing a function to computing its gradient in general is quite challenging or impossible without control over how quickly the gradient may be growing or changing, and in general, the gradient of the forward (attention) computation can behave quite erratically (which is likely necessary for the expressive power of attention units). Nonetheless, in the special case of the inputs for which attention computation is known to be hard from prior work, we are able to reasonably control the growth of these gradients and successfully perform our reduction.

Roadmap. We discuss other related works in Section 2. In Section 3, we provide the basic notation, definitions, backgrounds, and facts which we will use. In Section 4, we provide the proof sketch of our algorithm and defer the details to the Appendix. In Section 5, we briefly conclude our paper. In Section 6, we discuss the limitations of our paper. In Section 7, we provide the broader impact statement.

2 Related Work

Fine-grained Complexity. Numerous algorithmic techniques have been used in theory and in practice for attention computations. The first algorithm with provable guarantees, by Zandieh, Han, Daliri, and Karbasi [ZHDK23], used locality sensitive hashing (LSH) techniques [CKNS20], while later work by Alman and Song [AS23] used polynomial approxmation methods [ACSS20, AA22]. We particularly focus here on the latter technique, which is the only algorithm we're aware of which achieves near-linear running time.

Keles, Wijewardena, and Hedge [KWH23] established the first lower bound on attention computation under the assumption of SETH. Their findings demonstrated that when $d = \omega(\log n)$, it is not possible to execute forward computations in subquadratic time. The later lower bound of [AS23] further incorporated the magnitudes of the input entries into the lower bound to tightly match the aforementioned algorithms. Both use the high-level technique of [BIS17] from kernel density estimation, and build on methods derived from fine-grained complexity associated with approximate nearest neighbor search [Rub18] and the polynomial method [AA22].

Fast Attention Computation. Optimizing the computation of attention mechanisms in pre-trained LLMs, given their extensive parameter sets, has been a focal point of recent research. Various studies have explored the application of locality sensitive hashing (LSH) techniques to approximate attention mechanisms. [KKL20] introduced two methods to enhance computational efficiency, including the use of LSH to replace dot product attention and a reversible residual layer to substitute the standard residual layer. [CLP$^+$21] refined this approximation, noting that LSH's efficiency does not require constant parameter updates. [ZHDK23] proposed an innovative estimator based on Kernel Density Estimation (KDE) to speed up the softmax function and matrix multiplication computations. Some recent works [HJK$^+$23, KMZ23] have specifically used sketching techniques to avoid large entries in the attention matrix. [PMXA23] developed techniques utilizing a transformer within a transformer (TinT) model to simulate the transformer's forward and backward passes, significantly increasing parameter efficiency. [MGN$^+$23] tackled the challenge of fine-tuning LLMs with high memory demands by improving the classical ZO-SCD optimizer, creating a memory-efficient gradient estimator that requires only a forward pass. [BSZ24] provided insights into dynamic attention problems, they provide algorithm and hardness for the dynamic setting of attention problem. [GSY$^+$23a] introduces a quantum algorithm for attention computation, opening new avenues for efficiency improvements. [GSYZ24] provides a result for computing the attention matrix differentially privately. [DMS23] introduces a randomized and deterministic attention sparsification algorithms for over-parameterized feature dimension. [DLMS23] provides a zero-th order method to accelerate the computation of attention. [FA23, SLBK23, LLSS24] use weights sparsity to accelerate the attention computation, but cannot reduce the time complexity. [SMN$^+$24] compress the input token length to accelerate attention inference. [CLS$^+$24] uses Half-Space Reporting (HSR) techniques to accelerate attention computation. [SYZ24] studies proxy for softmax attention such as matrix exponential and provides fast algorithms for these proxies.

Transformer Training. Transformer architectures (the backbone of LLMs) have been trained with alternating steps of forward and backward computations since their introduction [VSP$^+$17, DCLT18, LOG$^+$19, YDY$^+$19, BMR$^+$20, ZRG$^+$22]. In Appendix B below, we perform computations to verify that our stated problems are the same as the forward and backward steps from the literature. Note that there are many weights update methods, such as LoRA [HSW$^+$21, ZL23, HSK$^+$24], prefix turning [LL21, LSSY24], and many so on. In this paper, we consider the standard training algorithm with gradient back-propagation. On the other hand [HYW$^+$23, HLSL24, WHHL24, HCL$^+$24, HWL24a, HCW$^+$24] introduce the modern Hopfield models as a proxy for possible fast attention computation in training (and inference), which have been used in various applications [XHH$^+$24, WHL$^+$24]. Similar analyses of computational feasibility have also been conducted for transformer-based diffusion models, such as Diffusion Transformers (DiTs) [HWL$^+$24b, Ano24].

3 Preliminary

In Section 3.1, we define some basic notation we will use. In Section A.3, we state important facts related to fast matrix multiplication. In Section 3.2, provide the formal definition of the Strong Exponential Time Hypothesis. In Section 3.3, we define several intermediate functions related to softmax and exponential which will arise in our algorithms. In Section 3.4, we define the loss function. In Section 3.5, we provide standard tensor tricks which we will use. In Section 3.6, we show how to reformulate the loss function for our purposes.

3.1 Notation

For any positive integer n, we define $[n] := \{1, 2, \ldots, n\}$. For two same length vector x and y, we use $\langle x, y \rangle$ to denote the inner product between x and y, i.e., $\langle x, y \rangle = \sum_{i=1}^n x_i y_i$. We use $x \circ y$ to denote vector that i-th entry is $x_i y_i$. Let $\mathbf{1}_n$ denote the length-n all ones vector. It is not hard to see that $\langle x \circ y, \mathbf{1}_n \rangle = \langle x, y \rangle$. For a vector x, we use $x^\top$ to denote the transpose of x. For a matrix M, we use $M^\top$ to denote the transpose of matrix M. For a vector x, we use $\exp(z)$ to denote the vector that i-th coordinate is $\exp(z_i)$. For a matrix M, we use $\exp(M)$ to denote the matrix that (i, j)-th coordinate is $\exp(M_{i,j})$. For a function f, we use $\widetilde{O}(f)$ to denote $f \cdot \mathrm{poly}(\log f)$. Let n_0, n_1, m_0, m_1 be positive integers. Let $X \in \mathbb{R}^{n_0 \times m_0}$ and $Y \in \mathbb{R}^{n_1 \times m_1}$. We define the Kronecker product between matrices X and Y, denoted $X \otimes Y \in \mathbb{R}^{n_0 n_1 \times m_0 m_1}$, as $(X \otimes Y)_{(j_0-1)n_1+j_1,(i_0-1)m_2+i_1}$ is equal to $X_{j_0,i_0} Y_{j_1,i_1}$, where $j_0 \in [n_0], i_0 \in [m_0], j_1 \in [n_1], i_1 \in [m_1]$.

3.2 Backgrounds on Complexity

Over 20 years ago, Impagliazzo and Paturi [IP01] introduced the Strong Exponential Time Hypothesis (SETH), an enhancement of the P $\neq$ NP conjecture. It posits that the existing algorithms for solving SAT problems are essentially as efficient as possible:

Hypothesis 3.1 (Strong Exponential Time Hypothesis (SETH))**.** *For any $\epsilon > 0$, there exists a positive integer $k \geq 3$ for which solving k-SAT problems with n variables in $O(2^{(1-\epsilon)n})$ time is impossible, including with the use of randomized algorithms.*

SETH, a widely recognized conjecture, has been instrumental in establishing fine-grained lower bounds across a broad spectrum of algorithmic challenges, as highlighted in the survey [Wil18].

3.3 Definitions related with Softmax

Now, we start by some definitions about $X \in \mathbb{R}^{d \times d}$ which will be helpful. Let x denote the vectorization of X.

Definition 3.2. *Let $A_1, A_2 \in \mathbb{R}^{n \times d}$ be two matrices. Suppose that $\mathsf{A} = A_1 \otimes A_2 \in \mathbb{R}^{n^2 \times d^2}$. We define $\mathsf{A}_{j_0} \in \mathbb{R}^{n \times d^2}$ be a $n \times d^2$ size sub-block from A. Note that there n such sub-blocks.*

For every $j_0 \in [n]$, let us define function $u(x)_{j_0} : \mathbb{R}^{d^2} \to \mathbb{R}^n$ to be:

$$u(x)_{j_0} := \underbrace{\exp(\mathsf{A}_{j_0} x)}_{n \times 1}.$$

Definition 3.3. *Suppose that there are two $n \times d$ size matrices $A_1, A_2 \in \mathbb{R}^{n \times d}$. We define $\mathsf{A}_{j_0} \in \mathbb{R}^{n \times d^2}$ be a $n \times d^2$ size sub-block from A. (Recall that $\mathsf{A} = A_1 \otimes A_2 \in \mathbb{R}^{n^2 \times d^2}$.)*

For every index $j_0 \in [n]$, we consider a function, $\alpha(x)_{j_0} : \mathbb{R}^{d^2} \to \mathbb{R}$ as:

$$\alpha(x)_{j_0} := \langle \underbrace{\exp(\mathsf{A}_{j_0} x)}_{n \times 1}, \underbrace{\mathbf{1}_n}_{n \times 1} \rangle.$$

Definition 3.4. *Suppose that $\alpha(x)_{j_0} \in \mathbb{R}$ is defined as in Definition 3.3.*

Recall $u(x)_{j_0} \in \mathbb{R}^n$ is defined as in Definition 3.2.

For a fixed $j_0 \in [n]$, let us consider function $f(x)_{j_0} : \mathbb{R}^{d^2} \to \mathbb{R}^n$

$$f(x)_{j_0} := \underbrace{\alpha(x)_{j_0}^{-1}}_{\text{scalar}} \underbrace{u(x)_{j_0}}_{n \times 1}.$$

Let $f(x) \in \mathbb{R}^{n \times n}$ denote the matrix where j_0-th row is $(f(x)_{j_0})^\top$.

Definition 3.5. *For every $i_0 \in [d]$, we define $h()_{i_0} : \mathbb{R}^{d^2} \to \mathbb{R}^n$ as:*

$$h(y)_{i_0} := \underbrace{A_3}_{n \times d} \underbrace{Y_{*,i_0}}_{d \times 1}.$$

Here let $Y \in \mathbb{R}^{d \times d}$ denote the matrix representation of $y \in \mathbb{R}^{d^2}$. Let $h(y) \in \mathbb{R}^{n \times d}$ matrix where i_0 column is $h(y)_{i_0}$.

3.4 Loss Functions

In this section, we introduce some helpful definitions related to both $x \in \mathbb{R}^{d^2}$.

Definition 3.6. *For every $j_0 \in [n]$, we use $f(x)_{j_0} \in \mathbb{R}^n$ to denote the normalized vector defined by Definition 3.4. For every $i_0 \in [d]$, we let $h(y)_{i_0}$ to be defined in Definition 3.5.*

Consider every $j_0 \in [n]$, every $i_0 \in [d]$. Let us consider $c(x)_{j_0,i_0} : \mathbb{R}^{d^2} \times \mathbb{R}^{d^2} \to \mathbb{R}$ as follows:

$$c(x)_{j_0,i_0} := \langle f(x)_{j_0}, h(y)_{i_0} \rangle - E_{j_0,i_0}.$$

Here E_{j_0,i_0} is the (j_0, i_0)-th coordinate/location of $E \in \mathbb{R}^{n \times d}$ for $j_0 \in [n], i_0 \in [d]$.

This is equivalent to

$$\underbrace{c(x)}_{n \times d} = \underbrace{f(x)}_{n \times n} \underbrace{h(y)}_{n \times d} - \underbrace{E}_{n \times d}.$$

Definition 3.7. *For every $j_0 \in [n]$, for every $i_0 \in [d]$. Let us define $L(x)_{j_0,i_0}$ to be $:= 0.5 c(x)_{j_0,i_0}^2$.*

3.5 Tensor Trick

We state the well-known tensor-trick. It has been widely used in literature of linear algebra related to tensor computations [SWZ19, DSSW18, DJS$^+$19, SWYZ21, AS24c, GSX23, Zha22, RSZ22, GSY23b, DSY23, DGS23].

https://doi.org/10.52202/079017-1963

Fact 3.8 (Tensor trick)**.** *For two matrices A_1 and $A_2 \in \mathbb{R}^{n \times d}$, define $\mathsf{A} = A_1 \otimes A_2$. Let $X \in \mathbb{R}^{d \times d}$. Let $x \in \mathbb{R}^{d^2}$ denote the vector representation of X. Then we have $\mathrm{vec}(A_1 X A_2^\top) = \mathsf{A}\, x$.*

Using the above tensor-trick, it is easy to observe that

Fact 3.9. *For two matrices A_1 and $A_2 \in \mathbb{R}^{n \times d}$, denote $\mathsf{A} = A_1 \otimes A_2$. Let $X \in \mathbb{R}^{d \times d}$. Let $\mathsf{A}_{j_0} \in \mathbb{R}^{n \times d^2}$ a submatrix of A (by properly selecting n rows of A). Let $x \in \mathbb{R}^{d^2}$ denote the vector representation of X. Then, we have*

- $\mathrm{vec}(\exp(A_1 X A_2^\top)) = \exp(\mathsf{A}\, x)$
- $(\exp(A_1 X A_2^\top)_{j_0,*})^\top = \exp(\mathsf{A}_{j_0}\, x)$,

Here $\exp(A_1 X A_2^\top)_{j_0,*}$ is the j_0-th row of $n \times n$ matrix $\exp(A_1 X A_2^\top)$.

Proof. We can use the definition in fact and Fact 3.8, to prove it. □

3.6 Reshape the Loss function via Tensor Trick

Lemma 3.10. *Given the below requirements*

- *Here are three matrices $A_1 \in \mathbb{R}^{n \times d}$, $A_2 \in \mathbb{R}^{n \times d}$, and $A_3 \in \mathbb{R}^{n \times d}$.*
- *Let $\mathsf{A} = A_1 \otimes A_2 \in \mathbb{R}^{n^2 \times d^2}$ to be the Kronecker product of the two matrices A_1 and A_2.*
  - *For every $j_0 \in [n]$, define $\mathsf{A}_{j_0} \in \mathbb{R}^{n \times d^2}$ to be a $n \times d^2$ sized block in the matrix $\mathsf{A} \in \mathbb{R}^{n^2 \times d^2}$.*
- *$E \in \mathbb{R}^{n \times d}$ be a matrix. Define E_{j_0,i_0} as the (j_0, i_0)-th coordinate/location of $E \in \mathbb{R}^{n \times d}$ for every pair of $j_0 \in [n]$ and $i_0 \in [d]$.*
- *Here are two square matrices $X \in \mathbb{R}^{d \times d}$, let $Y \in \mathbb{R}^{d \times d}$.*
- *Let $L(X)$ be defined as Definition 1.2.*
- *For every pair of $j_0 \in [n]$, $i_0 \in [d]$, recall that definition of $L(x)_{j_0,i_0}$ can be found in in Definition 3.7.*

Then, we have

$$L(X) = \sum_{j_0 \in [n]} \sum_{i_0 \in [d]} L(x)_{j_0,i_0}.$$

Proof. We can show that

$$\begin{aligned}
L(X) &= 0.5 \cdot \| \underbrace{D(X)^{-1}}_{n \times n} \underbrace{\exp(A_1 X A_2^\top)}_{n \times n} \underbrace{A_3}_{n \times d} \underbrace{Y}_{d \times d} - \underbrace{E}_{n \times d} \|_F^2 \\
&= \sum_{j_0=1}^{n} \sum_{i_0=1}^{d} 0.5 \cdot (\langle \langle \exp(\mathsf{A}_{j_0}\, x), \mathbf{1}_n \rangle^{-1} \cdot \exp(\mathsf{A}_{j_0}\, x), A_3 Y_{*,i_0} \rangle - E_{j_0,i_0})^2 \\
&= \sum_{j_0=1}^{n} \sum_{i_0=1}^{d} 0.5 (\langle f(x)_{j_0}, h(y)_{i_0} \rangle - E_{j_0,i_0})^2 \\
&= \sum_{j_0=1}^{n} \sum_{i_0=1}^{d} L(x)_{j_0,i_0}
\end{aligned}$$

where the first step follows from definition, the second step follows from writing down the summation, the third step follows from definition of $f(x)_{j_0}$ (recall the Definition 3.4) and $h(y)_{i_0}$ (recall the Definition 3.5), and the last step follows from $L(x)_{j_0,i_0}$ (see Definition 3.7). □

https://doi.org/10.52202/079017-1963

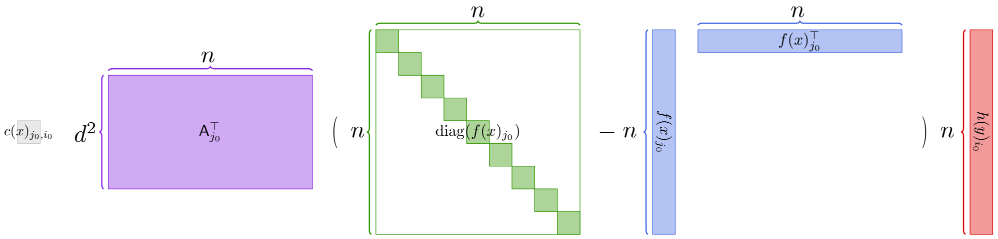

Figure 1: An example of $c(x,y)_{j_0,i_0} \cdot \mathsf{A}_{j_0,i}^\top(\mathrm{diag}(f(x)_{j_0}) - f(x)_{j_0} f(x)_{j_0}^\top) h(y)_{i_0}$.

4 Proof Sketch for General Upper Bound

The most straightforward way to compute the gradient would take $O(n^2 d^2)$ time in order to explicitly write down the matrix A. We first show how to obtain an intermediate algorithm, which runs in slightly improved time $O(n^2 d + nd^2)$ to compute the gradient. Our final algorithm will build on this idea.

Lemma 4.1 (Warmup, attention gradient computation, informal version of Lemma C.8)**.** *If the following conditions hold*

- *Define four $n \times d$ size matrices E, A_1, A_2, A_3 and two $d \times d$ square matrices X, Y to be input fixed matrices.*
- *Let $X \in \mathbb{R}^{d \times d}$ and $Y \in \mathbb{R}^{d \times d}$ denote matrix variables (we will compute gradient with respect to X).*
- *Let $g = \frac{\mathrm{d}L(X)}{\mathrm{d}X}$.*

Then the gradient g can be calculated in $O(n^2 d + nd^2)$ time.

The key idea behind Lemma 4.1 is to use algebraic manipulations to quickly compute the quantities defined in the previous section. We first compute $f(x)$ in $O(nd^2)$ time, then show $c(x)$ and $q(x)$ can be computed in $O(n^2 d)$ time. Using these, we compute $p(x)$ in (n^2) time, then putting in all together, we compute g in $O(n^2 d + nd^2)$ time. (We refer the details to Section C.)

For notational simplicity, we also write $x \in \mathbb{R}^{d^2 \times 1}$ to denote the vectorized version of X, and similarly $y \in \mathbb{R}^{d^2 \times 1}$ for Y.

Next, we will show how to improve the running time of computing the gradient from quadratic time ($\geq n^2$) to almost linear time $n^{1+o(1)}$. We build on the approach of Lemma 4.1 for computing the intermediate quantities f, c, q, and p, but speed up the time it takes to *implicitly*, rather than explicitly, represent these quantities. We want to emphasize that, although our algorithm relies on careful manipulation of the input matrices, our main algorithmic result does not make use of fast matrix multiplication (which may otherwise be quite impractical).

We now sketch the main algorithmic ideas. First, by linearity of derivative, we can show that

$$\frac{\mathrm{d}L(x)}{\mathrm{d}x} = \sum_{j_0=1}^{n} \sum_{i_0=1}^{d} \frac{\mathrm{d}L(x)_{j_0,i_0}}{\mathrm{d}x}$$

Based on calculations we perform in Section B, Section C, and several linear algebra facts, we can show that

$$\begin{aligned} & \frac{\mathrm{d}L(x)_{j_0,i_0}}{\mathrm{d}x} \\ = & \underbrace{c(x)_{j_0,i_0}}_{\text{scalar}} \cdot \underbrace{\mathsf{A}_{j_0}^\top}_{d^2 \times n} \underbrace{(\mathrm{diag}(f(x)_{j_0}) - f(x)_{j_0} f(x)_{j_0}^\top)}_{n \times n} \underbrace{h(y)_{i_0}}_{n \times 1} \end{aligned}$$

For any fixed $j_0 \in [n]$, consider this quantity. Since this expression involves an $n \times n$ matrix, the most straightforward way to calculate it would take $\Theta(n^2)$ time, and so summing over all $j_0 \in [n]$

https://doi.org/10.52202/079017-1963

would lead to a cubic-time algorithm. It is not too difficult to improve this: the $n \times n$ matrix (see Figure 1 for an illustration)

$$(\underbrace{\mathrm{diag}(f(x)_{j_0})}_{\text{a diagonal matrix}} - \underbrace{f(x)_{j_0} f(x)_{j_0}^\top}_{\text{a rank 1 matrix}})$$

is easily decomposed into a low-rank part ($f(x)_{j_0} f(x)_{j_0}^\top$ which has size $n \times n$) and a sparse part ($\mathrm{diag}(f(x)_{j_0})$ which also has size $n \times n$), which reduces the calculation of each part to only $\widetilde{O}(n)$ time, and the total running time to $\widetilde{O}(n^2)$ time.

However, we are aiming for a almost-linear time algorithm, and it is not possible to achieve this by treating the different j_0 separately, since a given j_0 must take $\Omega(n)$ time to process. Instead, we use tensor techniques related to low-rank approximations to simultanouesly compute all j_0 together and sum them in almost-linear time.

To do that, we create several extra artificial or intermediate matrices $q(x) \in \mathbb{R}^{n \times n}$(see Section C), $p(x) \in \mathbb{R}^{n \times n}$ (see Section C). We will show the gradient can be finally constructed using a simple chaining technique (see Section D for more details), from f, c, q, p_1 (handling $\mathrm{diag}(f(x)_{j_0})$ similarly), p_2 (handling $f(x)_{j_0} f(x)_{j_0}^\top$ similarly), p ($p = p_1 - p_2$) to $\frac{\mathrm{d}L}{\mathrm{d}x}$. Intuitively, the chaining shows that a low rank representation for f yields one for c, and these in turn yield one for q, and so on.

In particular, using $q(x)$, we obtain that $\frac{\mathrm{d}L(x)}{\mathrm{d}x}$ can be written as

$$\sum_{j_0=1}^{n} \mathsf{A}_{j_0}^\top (\underbrace{\text{a diagonal matrix}}_{\mathrm{diag}(f(x)_{j_0})} - \underbrace{\text{a rank 1 matrix}}_{f(x)_{j_0} f(x)_{j_0}^\top}) \underbrace{\text{a column vector}}_{q(x)_{j_0}}$$

which in fact notably removes the summation step of $i_0 = 1$ to d. Using the notation of $p(x)$, we finally yield that we need to compute $A_1^\top p(x) A_2$. Thus as long as $p(x)$ has a low-rank representation, then we can solve the in $n^{1+o(1)}$ time (see Section D for more details). In particular, we will find that $p(x)$ is the entry-wise product of two matrices with low-rank representations from prior work, which we can combine using a column-wise Kronecker product to approximate $p(x)$ itself.

5 Conclusion

Our results give a complete fine-grained analysis of the running time needed to train LLMs. We show that there is a threshold depending on the parameter B, the magnitude of the parameter matrix entries. In settings where B is small, a near-linear-time algorithm for LLM training is possible by using our novel algorithm for backward computation. In settings where B is large, not only does our algorithm not apply, but we show it is impossible to design a nontrivially-fast algorithm (barring a breakthrough in satisfiability algorithms that would refute the popular SETH).

These insights can guide LLM designers to more efficient algorithms. When B can be made small, it would lead to substantial savings in the computational resources needed for training and expression. When B must be large (perhaps to achieve a high expressiveness?), our lower bounds show that one may as well use straigthforward algorithms and focus on other aspects of algorithm speedup such as parallelization. The magnitude of B needed has been studied more recently (e.g., [AS24c]), and the need for fast training algorithms may further motivate this direction of research.

6 Limitations

Our main algorithm shows that the polynomial method can be used to quickly train LLMs with provable guarantees. While polynomial methods are frequently used for LLM operations in practice, they are typically simpler than the algorithms we present here. Implementing our algorithm in a practical way would require substantial future engineering work which is beyond the scope of this paper. Our lower bound is predicated on the Strong Exponential Time Hypothesis (SETH), a popular conjecture from fine-grained complexity theory. As with most results in complexity theory, results proved using a conjecture like this naturally come with associated limitations: the hard instances of SAT may not translate to the most important instances of LLM training, or the conjecture may

https://doi.org/10.52202/079017-1963

not even be true! That said, we wish to emphasize that SETH is the most popular conjecture in fine-grained complexity, used to prove the optimality of many algorithms in nearly every domain of computation, and decades of research in satisfiability algorithms have supported its veracity.

7 Broader Impact Statement

We give a new algorithm with provable guarantees for LLM training, which can help to guide future algorithm design in practice. This will help to develop the many positive broader impacts of LLMs. Since this is a purely theoretical work, which addresses theoretical computational concerns for implementing known algorithms, we believe it does not introduce any negative societal impact.

Acknowledgement

We would like to thank Yichuan Deng, Yingyu Liang, Han Liu, Jerry YC Hu, Zhenmei Shi, and Lichen Zhang for valuable discussions and anonymous reviewers for helpful comments. About how to upload this paper to arXiv (arxiv link https://arxiv.org/pdf/2402.04497 [AS24b]), the readers can watch video (this link https://youtu.be/QMzHigjMbTk) in this youtube channel: https://www.youtube.com/@zhaosong2031 .

References

[AA22] Amol Aggarwal and Josh Alman. Optimal-degree polynomial approximations for exponentials and gaussian kernel density estimation. In *37th Computational Complexity Conference (CCC 2022)*. Schloss Dagstuhl-Leibniz-Zentrum für Informatik, 2022.

[ACSS20] Josh Alman, Timothy Chu, Aaron Schild, and Zhao Song. Algorithms and hardness for linear algebra on geometric graphs. In *2020 IEEE 61st Annual Symposium on Foundations of Computer Science (FOCS)*, pages 541–552. IEEE, 2020.

[ALS$^+$23] Josh Alman, Jiehao Liang, Zhao Song, Ruizhe Zhang, and Danyang Zhuo. Bypass exponential time preprocessing: Fast neural network training via weight-data correlation preprocessing. In *NeurIPS*. arXiv preprint arXiv:2211.14227, 2023.

[Ano24] Anonymous. On statistical rates of conditional diffusion transformer: Approximation and estimation. In *Submitted to The Thirteenth International Conference on Learning Representations*, 2024. under review.

[AS23] Josh Alman and Zhao Song. Fast attention requires bounded entries. In *NeurIPS*, 2023.

[AS24a] Josh Alman and Zhao Song. Fast rope attention: Combining the polynomial method and fast fourier transform. In *manuscript*, 2024.

[AS24b] Josh Alman and Zhao Song. The fine-grained complexity of gradient computation for training large language models. In *NeurIPS*, 2024.

[AS24c] Josh Alman and Zhao Song. How to capture higher-order correlations? generalizing matrix softmax attention to kronecker computation. In *ICLR*, 2024.

[BCS97] Peter Bürgisser, Michael Clausen, and Mohammad A Shokrollahi. *Algebraic complexity theory*, volume 315. Springer Science & Business Media, 1997.

[BGVM20] Kunal Banerjee, Rishi Raj Gupta, Karthik Vyas, and Biswajit Mishra. Exploring alternatives to softmax function. *arXiv preprint arXiv:2011.11538*, 2020.

[BIS17] Arturs Backurs, Piotr Indyk, and Ludwig Schmidt. On the fine-grained complexity of empirical risk minimization: Kernel methods and neural networks. *Advances in Neural Information Processing Systems (NeurIPS)*, 30, 2017.

[Blä13] Markus Bläser. Fast matrix multiplication. *Theory of Computing*, pages 1–60, 2013.

https://doi.org/10.52202/079017-1963

[BMR$^+$20] Tom Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared D Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, et al. Language models are few-shot learners. *Advances in neural information processing systems*, 33:1877–1901, 2020.

[BPSW21] Jan van den Brand, Binghui Peng, Zhao Song, and Omri Weinstein. Training (overparametrized) neural networks in near-linear time. *12th Innovations in Theoretical Computer Science Conference (ITCS)*, 2021.

[BSZ24] Jan van den Brand, Zhao Song, and Tianyi Zhou. Algorithm and hardness for dynamic attention maintenance in large language models. In *ICML*. arXiv preprint arXiv:2304.02207, 2024.

[CKNS20] Moses Charikar, Michael Kapralov, Navid Nouri, and Paris Siminelakis. Kernel density estimation through density constrained near neighbor search. In *2020 IEEE 61st Annual Symposium on Foundations of Computer Science (FOCS)*, pages 172–183. IEEE, 2020.

[CLL$^+$24] Bo Chen, Xiaoyu Li, Yingyu Liang, Zhenmei Shi, and Zhao Song. Bypassing the exponential dependency: Looped transformers efficiently learn in-context by multi-step gradient descent. *arXiv preprint arXiv:2410.11268*, 2024.

[CLP$^+$21] Beidi Chen, Zichang Liu, Binghui Peng, Zhaozhuo Xu, Jonathan Lingjie Li, Tri Dao, Zhao Song, Anshumali Shrivastava, and Re.Mongoose Christopher. A learnable lsh framework for efficient neural network training. *International Conference on Learning Representation*, 2021.

[CLS$^+$24] Bo Chen, Yingyu Liang, Zhizhou Sha, Zhenmei Shi, and Zhao Song. Hsr-enhanced sparse attention acceleration. *arXiv preprint arXiv:2410.10165*, 2024.

[CND$^+$22] Aakanksha Chowdhery, Sharan Narang, Jacob Devlin, Maarten Bosma, Gaurav Mishra, Adam Roberts, Paul Barham, Hyung Won Chung, Charles Sutton, Sebastian Gehrmann, et al. Palm: Scaling language modeling with pathways. *arXiv preprint arXiv:2204.02311*, 2022.

[DCLT18] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. Bert: Pretraining of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805*, 2018.

[DGS23] Yichuan Deng, Yeqi Gao, and Zhao Song. Solving tensor low cycle rank approximation. *arXiv preprint arXiv:2304.06594*, 2023.

[DHS$^+$22] Yichuan Deng, Hang Hu, Zhao Song, Omri Weinstein, and Danyang Zhuo. Training overparametrized neural networks in sublinear time. *arXiv preprint arXiv:2208.04508*, 2022.

[DJS$^+$19] Huaian Diao, Rajesh Jayaram, Zhao Song, Wen Sun, and David Woodruff. Optimal sketching for kronecker product regression and low rank approximation. *Advances in neural information processing systems*, 32, 2019.

[DLMS23] Yichuan Deng, Zhihang Li, Sridhar Mahadevan, and Zhao Song. Zero-th order algorithm for softmax attention optimization. *arXiv preprint arXiv:2307.08352*, 2023.

[DMS23] Yichuan Deng, Sridhar Mahadevan, and Zhao Song. Randomized and deterministic attention sparsification algorithms for over-parameterized feature dimension. *arXiv preprint arXiv:2304.04397*, 2023.

[DSSW18] Huaian Diao, Zhao Song, Wen Sun, and David Woodruff. Sketching for kronecker product regression and p-splines. In *International Conference on Artificial Intelligence and Statistics*, pages 1299–1308. PMLR, 2018.

[DSY23] Yichuan Deng, Zhao Song, and Junze Yin. Faster robust tensor power method for arbitrary order. *arXiv preprint arXiv:2306.00406*, 2023.

https://doi.org/10.52202/079017-1963

[FA23] Elias Frantar and Dan Alistarh. Sparsegpt: Massive language models can be accurately pruned in one-shot. In *International Conference on Machine Learning*, pages 10323–10337. PMLR, 2023.

[GQSW24] Yeqi Gao, Lianke Qin, Zhao Song, and Yitan Wang. A sublinear adversarial training algorithm. In *ICLR*. arXiv preprint arXiv:2208.05395, 2024.

[GSWY23] Yeqi Gao, Zhao Song, Weixin Wang, and Junze Yin. A fast optimization view: Reformulating single layer attention in llm based on tensor and svm trick, and solving it in matrix multiplication time. *arXiv preprint arXiv:2309.07418*, 2023.

[GSX23] Yeqi Gao, Zhao Song, and Shenghao Xie. In-context learning for attention scheme: from single softmax regression to multiple softmax regression via a tensor trick. *arXiv preprint arXiv:2307.02419*, 2023.

[GSY$^+$23a] Yeqi Gao, Zhao Song, Xin Yang, Ruizhe Zhang, and Yufa Zhou. Fast quantum algorithm for attention computation. *arXiv preprint arXiv:2307.08045*, 2023.

[GSY23b] Yeqi Gao, Zhao Song, and Junze Yin. Gradientcoin: A peer-to-peer decentralized large language models. *arXiv preprint arXiv:2308.10502*, 2023.

[GSYZ24] Yeqi Gao, Zhao Song, Xin Yang, and Yufa Zhou. Differentially private attention computation. In *Neurips Safe Generative AI Workshop 2024*, 2024.

[HCL$^+$24] Jerry Yao-Chieh Hu, Pei-Hsuan Chang, Haozheng Luo, Hong-Yu Chen, Weijian Li, Wei-Po Wang, and Han Liu. Outlier-efficient hopfield layers for large transformer-based models. In *Forty-first International Conference on Machine Learning (ICML)*, 2024.

[HCW$^+$24] Jerry Yao-Chieh Hu, Bo-Yu Chen, Dennis Wu, Feng Ruan, and Han Liu. Nonparametric modern hopfield models. *arXiv preprint arXiv:2404.03900*, 2024.

[HJK$^+$23] Insu Han, Rajesh Jarayam, Amin Karbasi, Vahab Mirrokni, David P. Woodruff, and Amir Zandieh. Hyperattention: Long-context attention in near-linear time. *arXiv preprint arXiv:2310.05869*, 2023.

[HLSL24] Jerry Yao-Chieh Hu, Thomas Lin, Zhao Song, and Han Liu. On computational limits of modern hopfield models: A fine-grained complexity analysis. In *Forty-first International Conference on Machine Learning (ICML)*, 2024.

[HSK$^+$24] Jerry Yao-Chieh Hu, Maojiang Su, En-Jui Kuo, Zhao Song, and Han Liu. Computational limits of low-rank adaptation (lora) for transformer-based models. *arXiv preprint arXiv:2406.03136*, 2024.

[HSW$^+$21] Edward J Hu, Yelong Shen, Phillip Wallis, Zeyuan Allen-Zhu, Yuanzhi Li, Shean Wang, Lu Wang, and Weizhu Chen. Lora: Low-rank adaptation of large language models. *arXiv preprint arXiv:2106.09685*, 2021.

[HWL24a] Jerry Yao-Chieh Hu, Dennis Wu, and Han Liu. Provably optimal memory capacity for modern hopfield models: Transformer-compatible dense associative memories as spherical codes. In *Thirty-eighth Conference on Neural Information Processing Systems (NeurIPS)*, 2024.

[HWL$^+$24b] Jerry Yao-Chieh Hu, Weimin Wu, Zhuoru Li, Sophia Pi, , Zhao Song, and Han Liu. On statistical rates and provably efficient criteria of latent diffusion transformers (dits). In *Thirty-eighth Conference on Neural Information Processing Systems (NeurIPS)*, 2024.

[HYW$^+$23] Jerry Yao-Chieh Hu, Donglin Yang, Dennis Wu, Chenwei Xu, Bo-Yu Chen, and Han Liu. On sparse modern hopfield model. In *Thirty-seventh Conference on Neural Information Processing Systems (NeurIPS)*, 2023.

[IP01] Russell Impagliazzo and Ramamohan Paturi. On the complexity of k-sat. *Journal of Computer and System Sciences*, 62(2):367–375, 2001.

https://doi.org/10.52202/079017-1963

[JZLD21] Yanrong Ji, Zhihan Zhou, Han Liu, and Ramana V Davuluri. Dnabert: pre-trained bidirectional encoder representations from transformers model for dna-language in genome. *Bioinformatics*, 37(15):2112–2120, 2021.

[KKL20] Nikita Kitaev, Łukasz Kaiser, and Anselm Levskaya. Reformer: The efficient transformer. *arXiv preprint arXiv:2001.04451*, 2020.

[KMZ23] Praneeth Kacham, Vahab Mirrokni, and Peilin Zhong. Polysketchformer: Fast transformers via sketches for polynomial kernels. *arXiv preprint arXiv:2310.01655*, 2023.

[KVPF20] Angelos Katharopoulos, Apoorv Vyas, Nikolaos Pappas, and François Fleuret. Transformers are rnns: Fast autoregressive transformers with linear attention. In *International conference on machine learning*, pages 5156–5165. PMLR, 2020.

[KWH23] Feyza Duman Keles, Pruthuvi Mahesakya Wijewardena, and Chinmay Hegde. On the computational complexity of self-attention. In *International Conference on Algorithmic Learning Theory*, pages 597–619. PMLR, 2023.

[LL21] Xiang Lisa Li and Percy Liang. Prefix-tuning: Optimizing continuous prompts for generation. *arXiv preprint arXiv:2101.00190*, 2021.

[LLS$^+$24a] Chenyang Li, Yingyu Liang, Zhenmei Shi, Zhao Song, and Tianyi Zhou. Fourier circuits in neural networks and transformers: A case study of modular arithmetic with multiple inputs. *arXiv preprint arXiv:2402.09469*, 2024.

[LLS$^+$24b] Xiaoyu Li, Yingyu Liang, Zhenmei Shi, Zhao Song, and Yufa Zhou. Fine-grained attention i/o complexity: Comprehensive analysis for backward passes. *arXiv preprint arXiv:2410.09397*, 2024.

[LLS$^+$24c] Yingyu Liang, Heshan Liu, Zhenmei Shi, Zhao Song, Zhuoyan Xu, and Junze Yin. Conv-basis: A new paradigm for efficient attention inference and gradient computation in transformers. *arXiv preprint arXiv:2405.05219*, 2024.

[LLS$^+$24d] Yingyu Liang, Jiangxuan Long, Zhenmei Shi, Zhao Song, and Yufa Zhou. Beyond linear approximations: A novel pruning approach for attention matrix. *arXiv preprint arXiv:2410.11261*, 2024.

[LLSS24] Xiaoyu Li, Yingyu Liang, Zhenmei Shi, and Zhao Song. A tighter complexity analysis of sparsegpt. *arXiv preprint arXiv:2408.12151*, 2024.

[LOG$^+$19] Yinhan Liu, Myle Ott, Naman Goyal, Jingfei Du, Mandar Joshi, Danqi Chen, Omer Levy, Mike Lewis, Luke Zettlemoyer, and Veselin Stoyanov. Roberta: A robustly optimized bert pretraining approach. *arXiv preprint arXiv:1907.11692*, 2019.

[LSS$^+$24] Yingyu Liang, Zhizhou Sha, Zhenmei Shi, Zhao Song, and Yufa Zhou. Multi-layer transformers gradient can be approximated in almost linear time. *arXiv preprint arXiv:2408.13233*, 2024.

[LSSY24] Yingyu Liang, Zhenmei Shi, Zhao Song, and Chiwun Yang. Toward infinite-long prefix in transformer. *arXiv preprint arXiv:2406.14036*, 2024.

[LSSZ24] Yingyu Liang, Zhenmei Shi, Zhao Song, and Yufa Zhou. Tensor attention training: Provably efficient learning of higher-order transformers. *arXiv preprint arXiv:2405.16411*, 2024.

[Man23] James Manyika. An overview of bard: an early experiment with generative ai. Technical report, Tech. rep., Technical report, Google AI, 2023.

[MGN$^+$23] Sadhika Malladi, Tianyu Gao, Eshaan Nichani, Alex Damian, Jason D Lee, Danqi Chen, and Sanjeev Arora. Fine-tuning language models with just forward passes. *arXiv preprint arXiv:2305.17333*, 2023.

[PMXA23] Abhishek Panigrahi, Sadhika Malladi, Mengzhou Xia, and Sanjeev Arora. Trainable transformer in transformer. *arXiv preprint arXiv:2307.01189*, 2023.

https://doi.org/10.52202/079017-1963

[RSZ22] Aravind Reddy, Zhao Song, and Lichen Zhang. Dynamic tensor product regression. In *NeurIPS*, 2022.

[Rub18] Aviad Rubinstein. Hardness of approximate nearest neighbor search. In *Proceedings of the 50th annual ACM SIGACT symposium on theory of computing (STOC)*, pages 1260–1268, 2018.

[SLBK23] Mingjie Sun, Zhuang Liu, Anna Bair, and J Zico Kolter. A simple and effective pruning approach for large language models. *arXiv preprint arXiv:2306.11695*, 2023.

[SMN$^+$24] Zhenmei Shi, Yifei Ming, Xuan-Phi Nguyen, Yingyu Liang, and Shafiq Joty. Discovering the gems in early layers: Accelerating long-context llms with 1000x input token reduction. *arXiv preprint arXiv:2409.17422*, 2024.

[SWYZ21] Zhao Song, David Woodruff, Zheng Yu, and Lichen Zhang. Fast sketching of polynomial kernels of polynomial degree. In *International Conference on Machine Learning*, pages 9812–9823. PMLR, 2021.

[SWZ19] Zhao Song, David P Woodruff, and Peilin Zhong. Relative error tensor low rank approximation. In *SODA*. arXiv preprint arXiv:1704.08246, 2019.

[SYZ21] Zhao Song, Shuo Yang, and Ruizhe Zhang. Does preprocessing help training over-parameterized neural networks? *35th Conference on Neural Information Processing Systems*, 2021.

[SYZ24] Zhao Song, Junze Yin, and Lichen Zhang. Solving attention kernel regression problem via pre-conditioner. In *International Conference on Artificial Intelligence and Statistics (AISTATS)*, pages 208–216. PMLR, 2024.

[SZZ24] Zhao Song, Lichen Zhang, and Ruizhe Zhang. Training multi-layer over-parametrized neural network in subquadratic time. In *ITCS*. arXiv preprint arXiv:2112.07628, 2024.

[TDFH$^+$22] Romal Thoppilan, Daniel De Freitas, Jamie Hall, Noam Shazeer, Apoorv Kulshreshtha, Heng-Tze Cheng, Alicia Jin, Taylor Bos, Leslie Baker, Yu Du, et al. Lamda: Language models for dialog applications. *arXiv preprint arXiv:2201.08239*, 2022.

[TLI$^+$23] Hugo Touvron, Thibaut Lavril, Gautier Izacard, Xavier Martinet, Marie-Anne Lachaux, Timothée Lacroix, Baptiste Rozière, Naman Goyal, Eric Hambro, Faisal Azhar, Aurelien Rodriguez, Armand Joulin, Edouard Grave, and Guillaume Lample. Llama: Open and efficient foundation language models. *arXiv preprint arXiv:2302.13971*, 2023.

[TMS$^+$23] Hugo Touvron, Louis Martin, Kevin Stone, Peter Albert, Amjad Almahairi, Yasmine Babaei, Nikolay Bashlykov, Soumya Batra, Prajjwal Bhargava, Shruti Bhosale, et al. Llama 2: Open foundation and fine-tuned chat models. *arXiv preprint arXiv:2307.09288*, 2023.

[VSP$^+$17] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. *Advances in neural information processing systems*, 30, 2017.

[WCZ$^+$23] Yilin Wang, Zeyuan Chen, Liangjun Zhong, Zheng Ding, Zhizhou Sha, and Zhuowen Tu. Dolfin: Diffusion layout transformers without autoencoder. *arXiv preprint arXiv:2310.16305*, 2023.

[WHHL24] Dennis Wu, Jerry Yao-Chieh Hu, Teng-Yun Hsiao, and Han Liu. Uniform memory retrieval with larger capacity for modern hopfield models. In *Forty-first International Conference on Machine Learning (ICML)*, 2024.

[WHL$^+$24] Dennis Wu, Jerry Yao-Chieh Hu, Weijian Li, Bo-Yu Chen, and Han Liu. STanhop: Sparse tandem hopfield model for memory-enhanced time series prediction. In *The Twelfth International Conference on Learning Representations (ICLR)*, 2024.

[Wil18] Virginia Vassilevska Williams. On some fine-grained questions in algorithms and complexity. In *Proceedings of the international congress of mathematicians: Rio de janeiro 2018*, pages 3447–3487. World Scientific, 2018.

[WMS$^+$24] Jiayu Wang, Yifei Ming, Zhenmei Shi, Vibhav Vineet, Xin Wang, Yixuan Li, and Neel Joshi. Is a picture worth a thousand words? delving into spatial reasoning for vision language models. *arXiv preprint arXiv:2406.14852*, 2024.

[WSD$^+$23] Zirui Wang, Zhizhou Sha, Zheng Ding, Yilin Wang, and Zhuowen Tu. Tokencompose: Grounding diffusion with token-level supervision. *arXiv preprint arXiv:2312.03626*, 2023.

[WTB$^+$22] Jason Wei, Yi Tay, Rishi Bommasani, Colin Raffel, Barret Zoph, Sebastian Borgeaud, Dani Yogatama, Maarten Bosma, Denny Zhou, Donald Metzler, et al. Emergent abilities of large language models. *arXiv preprint arXiv:2206.07682*, 2022.

[XHH$^+$24] Chenwei Xu, Yu-Chao Huang, Jerry Yao-Chieh Hu, Weijian Li, Ammar Gilani, Hsi-Sheng Goan, and Han Liu. Bishop: Bi-directional cellular learning for tabular data with generalized sparse modern hopfield model. In *Forty-first International Conference on Machine Learning (ICML)*, 2024.

[XSL24] Zhuoyan Xu, Zhenmei Shi, and Yingyu Liang. Do large language models have compositional ability? an investigation into limitations and scalability. In *ICLR 2024 Workshop on Mathematical and Empirical Understanding of Foundation Models*, 2024.

[YCRI22] Ann Yuan, Andy Coenen, Emily Reif, and Daphne Ippolito. Wordcraft: story writing with large language models. In *27th International Conference on Intelligent User Interfaces*, pages 841–852, 2022.

[YDY$^+$19] Zhilin Yang, Zihang Dai, Yiming Yang, Jaime Carbonell, Russ R Salakhutdinov, and Quoc V Le. Xlnet: Generalized autoregressive pretraining for language understanding. *Advances in neural information processing systems*, 32, 2019.

[ZBIW19] Ofir Zafrir, Guy Boudoukh, Peter Izsak, and Moshe Wasserblat. Q8bert: Quantized 8bit bert. In *2019 Fifth Workshop on Energy Efficient Machine Learning and Cognitive Computing-NeurIPS Edition (EMC2-NIPS)*, pages 36–39. IEEE, 2019.

[ZBKR24] Michael Zhang, Kush Bhatia, Hermann Kumbong, and Christopher Ré. The hedgehog & the porcupine: Expressive linear attentions with softmax mimicry. In *ICLR*, 2024.

[Zha22] Lichen Zhang. Speeding up optimizations via data structures: Faster search, sample and maintenance. Master's thesis, Carnegie Mellon University, 2022.

[ZHDK23] Amir Zandieh, Insu Han, Majid Daliri, and Amin Karbasi. Kdeformer: Accelerating transformers via kernel density estimation. In *ICML*. arXiv preprint arXiv:2302.02451, 2023.

[ZJL$^+$23] Zhihan Zhou, Yanrong Ji, Weijian Li, Pratik Dutta, Ramana Davuluri, and Han Liu. Dnabert-2: Efficient foundation model and benchmark for multi-species genome. *arXiv preprint arXiv:2306.15006*, 2023.

[ZL23] Yuchen Zeng and Kangwook Lee. The expressive power of low-rank adaptation. *arXiv preprint arXiv:2310.17513*, 2023.

[ZRG$^+$22] Susan Zhang, Stephen Roller, Naman Goyal, Mikel Artetxe, Moya Chen, Shuohui Chen, Christopher Dewan, Mona Diab, Xian Li, Xi Victoria Lin, et al. Opt: Open pre-trained transformer language models. *arXiv preprint arXiv:2205.01068*, 2022.

[ZWH$^+$24] Zhihan Zhou, Winmin Wu, Harrison Ho, Jiayi Wang, Lizhen Shi, Ramana V Davuluri, Zhong Wang, and Han Liu. Dnabert-s: Learning species-aware dna embedding with genome foundation models. *arXiv preprint arXiv:2402.08777*, 2024.

https://doi.org/10.52202/079017-1963

Appendix

Roadmap.

In Section A, we provide basic notation and facts. In Section B, we provide details about gradient computations. In Section C, we explain the computation time for the gradient of attention loss. In Section D, we show how to further improve the gradient computation from quadratic time to almost linear time. In Section E, we provide our main lower bound result.

A Preliminaries

In Section A.1, we define some basic notation. In Section A.2, we state several facts which we will use.

A.1 Notation

For any positive integer n, we define $[n] := \{1, 2, \dots, n\}$.

For two same length vector x and y, we use $\langle x, y \rangle$ to denote the inner product between x and y, i.e., $\langle x, y \rangle = \sum_{i=1}^n x_i y_i$. We use $x \circ y$ to denote vector that i-th entry is $x_i y_i$. Let $\mathbf{1}_n$ denote the length-n all ones vector. It is not hard to see that $\langle x \circ y, \mathbf{1}_n \rangle = \langle x, y \rangle$.

For a vector u, we use $u^\top$ to denote the transpose of u. For a matrix M, we use $M^\top$ to denote the transpose of matrix M.

For a vector u, we use $\exp(u)$ to denote the vector that i-th coordinate is $\exp(u_i)$. For a matrix A, we use $\exp(A)$ to denote the matrix that (i, j)-th coordinate is $\exp(A_{i,j})$.

We define the Kronecker product between matrices X and Y, denoted $X \otimes Y \in \mathbb{R}^{n_0 n_1 \times m_0 m_1}$, as $(X \otimes Y)_{(j_0-1)n_1+j_1,(i_0-1)m_2+i_1}$ is equal to $X_{j_0,i_0} Y_{j_1,i_1}$, where $j_0 \in [n_0], i_0 \in [m_0], j_1 \in [n_1], i_1 \in [m_1]$.

For each positive integers m_1, m_2, m_3, we use $\mathcal{T}_{\mathrm{mat}}(m_1, m_2, m_3)$ to denote the time of multiplying $m_1 \times m_2$ matrix with another $m_2 \times m_3$ matrix.

A.2 Basic Facts

Fact A.1. *Let $x, y, z \in \mathbb{R}^n$. Then we have*

- $\langle x \circ y, z \rangle = x^\top \mathrm{diag}(y) z$.
- $\langle x, y \rangle = \langle x \circ y, \mathbf{1}_n \rangle$.

Fact A.2 (Folklore)**.** *Let $U_1, V_1 \in \mathbb{R}^{n \times k_1}$. Let $U_2, V_2 \in \mathbb{R}^{n \times k_2}$. Then we have*

$$(U_1 V_1^\top) \circ (U_2 V_2^\top) = (U_1 \oslash U_2)(V_1 \oslash V_2)^\top$$

Here, given $U_1 \in \mathbb{R}^{n \times k_1}$ and $U_2 \in \mathbb{R}^{n \times k_2}$, the $U_1 \oslash U_2 \in \mathbb{R}^{n \times k_1 k_2}$ is the row-wise Kronecker product, i.e., $(U_1 \oslash U_2)_{i,l_1+(l_2-1)k_1} := (U_1)_{i,l_1} U_{i,l_2}$ for all $i \in [n]$, $l_1 \in [k_1]$ and $l_2 \in [k_2]$

A.3 Matrix Multiplication

We define matrix multiplication notation and state some well-know facts here.

Definition A.3. *Let n_1, n_2, n_3, denote any three positive integers. We use $\mathcal{T}_{\mathrm{mat}}(n_1, n_2, n_3)$ to denote the time of multiplying an $n_1 \times n_2$ matrix with another $n_2 \times n_3$.*

The straightgforward algorithm following the definition of matrix multiplication gives that $\mathcal{T}_{\mathrm{mat}}(n_1, n_2, n_3) \leq O(n_1 n_2 n_3)$. In fact, we will only use this straightforward bound in all our algorithms in this paper, and we avoid needing any other (potentially impractical) matrix multiplication algorithms. Nonetheless, we will emphasize the appearances of $\mathcal{T}_{\mathrm{mat}}$ in our algorithms below, since then any fast algorithms or systems for performing matrix multiplication could be used to speed up these steps of our approach.

It is well-known that

Fact A.4 ([BCS97, Blä13]). *Let n_1, n_2, n_3, denote any three positive integers.* $\mathcal{T}_{\mathrm{mat}}(n_1, n_2, n_3) = O(\mathcal{T}_{\mathrm{mat}}(n_1, n_3, n_2)) = O(\mathcal{T}_{\mathrm{mat}}(n_2, n_1, n_3)) = O(\mathcal{T}_{\mathrm{mat}}(n_2, n_3, n_1)) = O(\mathcal{T}_{\mathrm{mat}}(n_3, n_1, n_2)) = O(\mathcal{T}_{\mathrm{mat}}(n_3, n_2, n_1))$.

B More Details about Gradient Computation

In this section, we provide details and calculations to assist with gradient and derivative computations. We remark that, in this section, for convenience of computing a closed form for the gradient, we ignore the $1/d$ factor in function f. Since it is only a rescaling factor, it won't affect how we compute these matrices in general.

Lemma B.1 (The gradient computation for several different functions with respect to x_i). *For every $i \in [d^2]$, define $\mathsf{A}_{j_0,i} \in \mathbb{R}^n$ to be the i-th column for $\mathsf{A}_{j_0} \in \mathbb{R}^{n \times d}$. $u(x)_{j_0} \in \mathbb{R}^n$. The scalar function $\alpha(x)_{j_0} \in \mathbb{R}$, column function $f(x)_{j_0} \in \mathbb{R}^n$, scalar function $c(x)_{j_0,i_0} \in \mathbb{R}$ and scalar function $L(x)_{j_0,i_0} \in \mathbb{R}$ are defined as in Definitions 3.2, 3.3, 3.4, 3.6 and 3.7 respectively.*

Then, for each $i \in [d^2]$, we have

- **Part 1.**
$$\frac{\mathrm{d}x}{\mathrm{d}x_i} = e_i$$
- **Part 2.** *For each $j_0 \in [n]$,*
$$\frac{\mathrm{d}\,\mathsf{A}_{j_0}\,x}{\mathrm{d}x_i} = (\mathsf{A}_{j_0})_i$$
- **Part 3.** *For each $j_0 \in [n]$*
$$\frac{\mathrm{d}u(x)_{j_0}}{\mathrm{d}x_i} = \mathsf{A}_{j_0,i} \circ u(x)_{j_0}$$
- **Part 4.** *For each $j_0 \in [n]$,*
$$\frac{\mathrm{d}\alpha(x)_{j_0}}{\mathrm{d}x_i} = \langle \mathsf{A}_{j_0,i}, u(x)_{j_0} \rangle$$
- **Part 5.** *For each $j_0 \in [n]$,*
$$\frac{\mathrm{d}f(x)_{j_0}}{\mathrm{d}x_i} = \mathsf{A}_{j_0,i} \circ f(x)_{j_0} - \langle \mathsf{A}_{j_0,i}, f(x)_{j_0} \rangle \cdot f(x)_{j_0}$$
- **Part 6.** *For each $j_0 \in [n]$, for each $i_0 \in [d]$,*
$$\frac{\mathrm{d}\langle f(x)_{j_0}, h(y)_{i_0} \rangle}{\mathrm{d}x_i} = \langle h(y)_{i_0}, \mathsf{A}_{j_0,i} \circ f(x)_{j_0} \rangle - \langle h(y)_{i_0}, f(x)_{j_0} \rangle \cdot \langle \mathsf{A}_{j_0,i}, f(x)_{j_0} \rangle$$
- **Part 7.** *For each $j_0 \in [n]$, for every $i_0 \in [d]$*
$$\frac{\mathrm{d}c(x)_{j_0,i_0}}{\mathrm{d}x_i} = \langle \mathsf{A}_{j_0,i} \circ f(x)_{j_0}, h(y)_{i_0} \rangle - \langle f(x)_{j_0}, h(y)_{i_0} \rangle \cdot \langle \mathsf{A}_{j_0,i}, f(x)_{j_0} \rangle$$
- **Part 8.** *For each $j_0 \in [n]$, for each $i_0 \in [d]$*
$$\frac{\mathrm{d}L(x)_{j_0,i_0}}{\mathrm{d}x_i} = (\langle h(y)_{i_0}, \mathsf{A}_{j_0,i} \circ f(x)_{j_0} \rangle - \langle f(x)_{j_0}, \mathsf{A}_{j_0,i} \rangle \cdot \langle h(y)_{i_0}, f(x)_{j_0} \rangle) \cdot c(x)_{j_0,i_0}$$

Proof. **Proof of Part 1.** We have

$$\frac{\mathrm{d}x}{\mathrm{d}x_i} = e_i$$

https://doi.org/10.52202/079017-1963

Proof of Part 2. We have

$$\begin{aligned} \frac{\mathrm{d}\,\mathsf{A}_{j_0}\,x}{\mathrm{d}x_i} = & ~ \underbrace{\mathsf{A}_{j_0}}_{n \times d^2} \underbrace{\frac{\mathrm{d}x}{\mathrm{d}x_i}}_{d^2 \times 1} \\ = & ~ \underbrace{\mathsf{A}_{j_0}}_{n \times d^2} \cdot \underbrace{e_i}_{d^2 \times 1} \\ = & ~ \mathsf{A}_{j_0,i} \end{aligned}$$

Proof of Part 3.

We can show

$$\begin{aligned} \frac{\mathrm{d}u(x)_{j_0}}{\mathrm{d}x_i} = & \frac{\mathrm{d}\exp(\mathsf{A}_{j_0}\,x)}{\mathrm{d}x_i} \\ = & ~ \exp(\mathsf{A}_{j_0}\,x) \circ \frac{\mathrm{d}\,\mathsf{A}_{j_0}\,x}{\mathrm{d}x_i} \\ = & ~ \exp(\mathsf{A}_{j_0}\,x) \circ \mathsf{A}_{j_0,i} \\ = & \, u(x)_{j_0} \circ \mathsf{A}_{j_0,i} \end{aligned}$$

where the 3rd step follows from Part 2, the last step follows from definition of $u(x)_{j_0}$.

Proof of Part 4.

For simplicity of writing proofs, we use $(\cdot)$ to denote (x).

We can show

$$\begin{aligned} \frac{\mathrm{d}\alpha(\cdot)_{j_0}}{\mathrm{d}x_i} = & \frac{\mathrm{d}\langle u(\cdot)_{j_0}, \mathbf{1}_n\rangle}{\mathrm{d}x_i} \\ = & \langle u(\cdot)_{j_0} \circ \mathsf{A}_{j_0,i}, \mathbf{1}_n \rangle \\ = & \langle u(\cdot)_{j_0}, \mathsf{A}_{j_0,i} \rangle \end{aligned}$$

where the 1st step follows from definition of $\alpha(\cdot)$, the 2nd step follows from Part 3, the 3rd step follows from Fact A.1.

Proof of Part 5. For simplicity of writing proofs, we use $(\cdot)$ to denote (x).

We can show that

$$\begin{aligned} \frac{\mathrm{d}f(\cdot)_{j_0}}{\mathrm{d}x_i} = & \frac{\mathrm{d}\alpha(\cdot)_{j_0}^{-1} u(\cdot)_{j_0}}{\mathrm{d}x_i} \\ = & \, \alpha(\cdot)_{j_0}^{-1} \frac{\mathrm{d}u(\cdot)_{j_0}}{\mathrm{d}x_i} + (\frac{\mathrm{d}\alpha(\cdot)_{j_0}^{-1}}{\mathrm{d}x_i}) u(\cdot)_{j_0} \end{aligned}$$

For the first term, we have

$$\begin{aligned} \alpha(\cdot)_{j_0}^{-1} \frac{\mathrm{d}u(\cdot)_{j_0}}{\mathrm{d}x_i} = & \, \alpha(\cdot)_{j_0}^{-1} u(\cdot)_{j_0} \circ \mathsf{A}_{j_0,i} \\ = & \, f(\cdot)_{j_0} \circ \mathsf{A}_{j_0,i} \end{aligned}$$

where the 1st step follows from Part 3, the 2nd step follows from definition of $f(\cdot)$.

For the second term, we have

$$\begin{aligned} (\frac{\mathrm{d}\alpha(\cdot)_{j_0}^{-1}}{\mathrm{d}x_i}) u(\cdot)_{j_0} = & ~ -\alpha(\cdot)_{j_0}^{-2} \frac{\mathrm{d}\alpha(\cdot)_{j_0}}{\mathrm{d}x_i} u(\cdot)_{j_0} \\ = & ~ -\alpha(\cdot)_{j_0}^{-2} \cdot \langle u(\cdot)_{j_0}, \mathsf{A}_{j_0,i} \rangle \cdot u(\cdot)_{j_0} \\ = & ~ -f(\cdot)_{j_0} \cdot \langle f(\cdot)_{j_0}, \mathsf{A}_{j_0,i} \rangle \end{aligned}$$

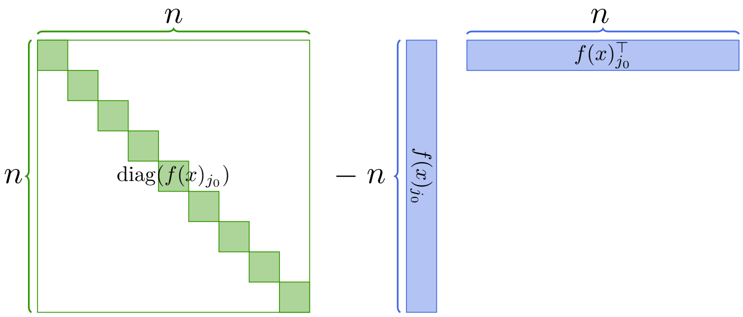

Figure 2: An example of $\mathrm{diag}(f(x)_{j_0}) - f(x)_{j_0} f(x)_{j_0}^\top$.

where the 1st step follows from basic calculus, the 2nd step follows from Part 4, the 3rd step follows from definition of $f(\cdot)_{j_0}$.

Using all of the results above, it holds that

$$\frac{\mathrm{d} f(\cdot)_{j_0}}{\mathrm{d} x_i} = f(\cdot)_{j_0} \circ \mathsf{A}_{j_0,i} - f(\cdot)_{j_0} \cdot \langle f(\cdot)_{j_0}, \mathsf{A}_{j_0,i} \rangle$$

Proof of Part 6. It follows Part 5 directly.

Proof of Part 7. For simplicity of writing proofs, we use $(\cdot)$ to denote (x).

Following the definition of c in Definition 3.6, it holds that

$$c(\cdot)_{j_0,i_0} := \langle f(\cdot)_{j_0}, h(y) \rangle - E_{j_0,i_0} \tag{1}$$

Thus it holds that

$$\begin{aligned} \frac{\mathrm{d} c(\cdot)_{j_0,i_0}}{\mathrm{d} x_i} &= \frac{\mathrm{d}(\langle f(\cdot)_{j_0}, h(y)_{i_0} \rangle - E_{j_0,i_0})}{\mathrm{d} x_i} \\ &= \frac{\mathrm{d} \langle f(\cdot)_{j_0}, h(y)_{i_0} \rangle}{\mathrm{d} x_i} \\ &= \langle f(\cdot)_{j_0} \circ \mathsf{A}_{j_0,i}, h(y)_{i_0} \rangle - \langle f(\cdot)_{j_0}, h(y)_{i_0} \rangle \cdot \langle f(\cdot)_{j_0}, \mathsf{A}_{j_0,i} \rangle, \end{aligned}$$

where the 1st step is because of Eq. (1), the 2nd step is from $\frac{\mathrm{d} E_{j_0,i_0}}{\mathrm{d} x_i} = 0$, and the 3rd step is followed by **Part 4**.

Proof of Part 8. For simplicity of writing proofs, we use $(\cdot)$ to denote (x). Following the definition of $L(\cdot)$ in Definition 3.7, it holds that

$$L(\cdot)_{j_0,i_0} = 0.5 c(\cdot)_{j_0,i_0}^2 \tag{2}$$

Thus, we have

$$\begin{aligned} \frac{\mathrm{d} L(\cdot)_{j_0,i_0}}{\mathrm{d} x_i} &= \frac{\mathrm{d}(0.5 c(\cdot)_{j_0,i_0}^2)}{\mathrm{d} x_i} \\ &= c(\cdot)_{j_0,i_0} \frac{\mathrm{d} c(\cdot)}{\mathrm{d} x_i} \\ &= c(\cdot)_{j_0,i_0} \cdot (\langle f(\cdot)_{j_0} \circ \mathsf{A}_{j_0,i}, h(y)_{i_0} \rangle - \langle f(\cdot)_{j_0}, h(y)_{i_0} \rangle \cdot \langle f(\cdot)_{j_0}, \mathsf{A}_{j_0,i} \rangle), \end{aligned}$$

where the 1st step is followed by the Eq. (2), the 2nd step is due to the chain rule, the last step followed by **Part 5**.

□

C Time for Straightforward Computation

In Section C.1, we show the calculation of f (Similarly as Section B, we still ignore the $1/d$ factor here) and h. In Section C.2, we show the way we calculate c in straightforward way. In Section C.3 and Section C.4, we define two artificial functions p and q, and show how to compute them. In Section C.5, we provide the way to re-write the gradient in an elegant way. In Section C.6, we finally put these all together and find the running time of our algorithm.

C.1 Compute f and h

Lemma C.1 (Computing f and h). *Suppose the following objects are given*

- *Let $f(x)$ be defined as Definition 3.4*
- *Let $h(y)$ be defined as Definition 3.5*

Then, we have

- *$f(x)$ can be calculated in time of $\mathcal{T}_{\mathrm{mat}}(n, d, n) + \mathcal{T}_{\mathrm{mat}}(n, d, d)$*
- *$h(y)$ can be calculated in time of $\mathcal{T}_{\mathrm{mat}}(n, d, d)$*

Proof. Note that

$$f(x) = D^{-1} \exp(A_1 X A_2^\top)$$

and

$$D = \mathrm{diag}(\exp(A_1 X A_2^\top) \mathbf{1}_n)$$

We firstly compute $\exp(A_1 X A_2^\top)$, this takes time of $\mathcal{T}_{\mathrm{mat}}(n, d, d)$ and $\mathcal{T}_{\mathrm{mat}}(n, d, n)$.

Then we can compute D, which takes $O(n^2)$ time.

Then we can compute $D^{-1} \exp(A_1 X A_2^\top)$, this takes $O(n^2)$ time.

Thus, the overall time is

$$\begin{aligned} & \mathcal{T}_{\mathrm{mat}}(n, d, d) + \mathcal{T}_{\mathrm{mat}}(n, d, n) + O(n^2) \\ = & O(\mathcal{T}_{\mathrm{mat}}(n, d, d) + \mathcal{T}_{\mathrm{mat}}(n, d, n)) \end{aligned}$$

Note that $h(y) = A_3 Y$ which takes time of $\mathcal{T}_{\mathrm{mat}}(n, d, d)$.

Thus, the proof is completed. □

C.2 Compute c

Lemma C.2 (Computing c). *Suppose the following objects are given*

- *$E \in \mathbb{R}^{n \times d}$*
- *$f(x) \in \mathbb{R}^{n \times n}$ is given*
- *$h(y) \in \mathbb{R}^{n \times d}$ is given,*

Then one can compute $c(x) \in \mathbb{R}^{n \times d}$ in $\mathcal{T}_{\mathrm{mat}}(n, n, d)$ time.

Proof. Based on Definition of $c(x) \in \mathbb{R}^{n \times d}$ which is

$$c(x) = f(x)h(y) - E$$

Computing $f(x)h(y)$ takes time of $\mathcal{T}_{\mathrm{mat}}(n, n, d)$, and calculating $f(x)h(y) - E$ takes time of $O(nd)$.

Thus, finally, overall time is

$$\mathcal{T}_{\mathrm{mat}}(n, n, d) + O(nd).$$

□

https://doi.org/10.52202/079017-1963

C.3 Computation for q

We will define q, and then explain how to calculate q.

Definition C.3. *Define $c(x) \in \mathbb{R}^{n \times d}$ as in Definition 3.6. Define $h(y) \in \mathbb{R}^{n \times d}$ as in Definition 3.5. We define $q(x) \in \mathbb{R}^{n \times n}$ as*

$$q(x) := \underbrace{c(x)}_{n \times d} \underbrace{h(y)^\top}_{d \times n}$$

Then we use $q(x)_{j_0}^\top$ to denote the j_0-th row of $q(x) \in \mathbb{R}^{n \times n}$.

Lemma C.4. *If it holds that*

- *Suppose $c(x) \in \mathbb{R}^{n \times d}$ is given*
- *Suppose $h(y) \in \mathbb{R}^{n \times d}$ is given*

Then, we can compute $q(x)$ in the time of $O(\mathcal{T}_{\mathrm{mat}}(n, n, d))$.

Proof. Recall that $q(x) = c(x) h(y)^\top$. Thus it takes time of $\mathcal{T}_{\mathrm{mat}}(n, d, n) = O(\mathcal{T}_{\mathrm{mat}}(n, n, d))$. □

C.4 Computation for $p(x)$

Let us firstly define p, and then we can show how to construct it.

Definition C.5. *For every index $j_0 \in [n]$, we define $p(x)_{j_0} \in \mathbb{R}^n$ as*

$$p(x)_{j_0} := (\mathrm{diag}(f(x)_{j_0}) - f(x)_{j_0} f(x)_{j_0}^\top) q(x)_{j_0}.$$

We define $p(x) \in \mathbb{R}^{n \times n}$ in the sense that $p(x)_{j_0}^\top$ is the j_0-th row of $p(x)$.

Lemma C.6. *If the below requirements are holding that*

- *Suppose $f(x) \in \mathbb{R}^{n \times n}$ is given*
- *Suppose $q(x) \in \mathbb{R}^{n \times n}$ is given*

Then, we can compute $p(x)$ in $O(n^2)$ time.

Proof. Since $\mathrm{diag}(f(x)_{j_0})$ is a diagonal matrix and $f(x)_{j_0} f(x)_{j_0}^\top$ is a rank-one matrix, we know that $p(x)_{j_0} \in \mathbb{R}^n$ can be computed in $O(n)$, for each $j_0 \in [n]$. Thus we can construct matrix $p(x) \in \mathbb{R}^{n \times n}$ in $n \times O(n) = O(n^2)$ time in total. □

C.5 Analyze the closed form of gradient

Lemma C.7. *Define the functions $f(x) \in \mathbb{R}^{n \times n}$, $c(x) \in \mathbb{R}^{n \times d}$, $h(y) \in \mathbb{R}^{n \times d}$, $q(x) \in \mathbb{R}^{n \times n}$ and $p(x) \in \mathbb{R}^{n \times n}$ as in Definitions 3.4, 3.6, 3.5, C.3 and C.5 respectively. $A_1, A_2 \in \mathbb{R}^{n \times d}$ are two given matrices. We define $\mathsf{A} = A_1 \otimes A_2$. Let $L(x)$ be defined as Definition 1.2. Let $L(x)_{j_0, i_0}$ be defined as Definition 3.7. Then, we can show that $\frac{\mathrm{d}L(x)}{\mathrm{d}x} = \mathrm{vec}(A_1^\top p(x) A_2)$.*

Proof. From the Lemma statement, we have

$$\frac{\mathrm{d}L(x, y)_{j_0, i_0}}{\mathrm{d}x_i} = c(x, y)_{j_0, i_0} \cdot (\langle f(x)_{j_0} \circ \mathsf{A}_{j_0, i}, h(y)_{i_0} \rangle - \langle f(x)_{j_0}, h(y)_{i_0} \rangle \cdot \langle f(x)_{j_0}, \mathsf{A}_{j_0, i} \rangle) \quad (3)$$

Note that by Fact A.1, it holds that

$$\langle f(x)_{j_0} \circ \mathsf{A}_{j_0, i}, h(y)_{i_0} \rangle = \mathsf{A}_{j_0, i}^\top \mathrm{diag}(f(x)_{j_0}) h(y)_{i_0}$$

and

$$\langle f(x)_{j_0}, v \rangle \cdot \langle f(x)_{j_0}, \mathsf{A}_{j_0, i} \rangle = \mathsf{A}_{j_0, i}^\top f(x)_{j_0} f(x)_{j_0}^\top h(y)_{i_0}$$

Therefore, Eq. (3) becomes

$$\begin{aligned}\frac{\mathrm{d}L(x)_{j_0,i_0}}{\mathrm{d}x_i} &= c(x,y)_{j_0,i_0} \cdot (\mathsf{A}_{j_0,i}^\top \operatorname{diag}(f(x)_{j_0}) h(y)_{i_0} - \mathsf{A}_{j_0,i}^\top f(x)_{j_0} f(x)_{j_0}^\top h(y)_{i_0}) \\ &= c(x,y)_{j_0,i_0} \cdot \mathsf{A}_{j_0,i}^\top (\operatorname{diag}(f(x)_{j_0}) - f(x)_{j_0} f(x)_{j_0}^\top) h(y)_{i_0}, \end{aligned} \tag{4}$$

where the 2nd step follows from simple algebra.

Recall the way we define $q(x)_{j_0}$ (see Definition C.3).

$$q(x)_{j_0} := \sum_{i_0=1}^{d} c(x)_{j_0,i_0} h(y)_{i_0}. \tag{5}$$

Recall that $p(x)_{j_0} \in \mathbb{R}^n$ is define as Definition C.5,

$$p(x)_{j_0} := (\operatorname{diag}(f(x)_{j_0}) - f(x)_{j_0} f(x)_{j_0}^\top) q(x)_{j_0}. \tag{6}$$

It holds that

$$\begin{aligned} &\frac{\mathrm{d}L(x)}{\mathrm{d}x} \\ = &\sum_{j_0=1}^{n} \sum_{i_0=1}^{d} \frac{\mathrm{d}L(x)_{j_0,i_0}}{\mathrm{d}x} \\ = &\sum_{j_0=1}^{n} \sum_{i_0=1}^{d} \underbrace{c(x)_{j_0,i_0}}_{\text{scalar}} \cdot \underbrace{\mathsf{A}_{j_0}^\top}_{d^2 \times n} \underbrace{(\operatorname{diag}(f(x)_{j_0}) - f(x)_{j_0} f(x)_{j_0}^\top)}_{n \times n} \underbrace{h(y)_{i_0}}_{n \times 1} \\ = &\sum_{j_0=1}^{n} \mathsf{A}_{j_0}^\top (\operatorname{diag}(f(x)_{j_0}) - f(x)_{j_0} f(x)_{j_0}^\top) q(x)_{j_0} \\ = &\sum_{j_0=1}^{n} \mathsf{A}_{j_0}^\top p(x)_{j_0} \\ = &\operatorname{vec}(A_1^\top p(x) A_2) \end{aligned}$$

where the 1st step is because of Definition 1.2, the 2nd step is based on Eq. (4), the 3rd step is followed by Eq. (5), the 4th step is due to Eq. (6), and the last step uses tensor-trick.

□

C.6 Putting it together

Lemma C.8 (Attention gradient computation, formal version of Lemma 4.1)**.** *If it holds that*

- *Define $A_1, A_2, A_3, E \in \mathbb{R}^{n \times d}$. Define $X, Y \in \mathbb{R}^{d \times d}$ to be several input fixed matrices.*
- *Let $X, Y \in \mathbb{R}^{d \times d}$ denote matrix variables (we will compute gradient with respect to X)*
  - *For easy of writing, we also use vector variables $x \in \mathbb{R}^{d^2 \times 1}$ and $y \in \mathbb{R}^{d^2 \times 1}$, i.e., $\operatorname{vec}(X) = x$.*
- *Let $g = \frac{\mathrm{d}L(X)}{\mathrm{d}x} \in \mathbb{R}^{d^2}$ (where $L(X)$ is defined as Definition 1.2)*

Then we can show that gradient $g \in \mathbb{R}^{d^2}$ can be computed in $\mathcal{T}_{\mathrm{mat}}(n,d,n) + \mathcal{T}_{\mathrm{mat}}(n,d,d)$ time.

Proof. Step 1. we compute $f(x)$, $h(y)$. This takes $O(\mathcal{T}_{\mathrm{mat}}(n,n,d) + \mathcal{T}_{\mathrm{mat}}(n,d,d))$ time due to Lemma C.1.

Step 2. we compute $c(x)$. This takes time of $O(\mathcal{T}_{\mathrm{mat}}(n,n,d) + \mathcal{T}_{\mathrm{mat}}(n,d,d))$ due to Lemma C.2.

Step 3. we compute $q(x)$. This take time of $O(\mathcal{T}_{\mathrm{mat}}(n,n,d))$ due to Lemma C.4.

Step 4. we compute $p(x)$. This take time of $O(n^2)$ due to Lemma C.6.

Step 5. using Lemma C.7, we know that gradient is equivalent to $\mathrm{vec}(A_1^\top p(x) A_2)$. Suppose $A_1^\top \in \mathbb{R}^{d \times n}, p(x) \in \mathbb{R}^{n \times n}, A_2 \in \mathbb{R}^{n \times d}$ are given, then it can be calculated in time of $O(\mathcal{T}_{\mathrm{mat}}(n,n,d) + \mathcal{T}_{\mathrm{mat}}(n,d,d))$.

Thus, overall running for computing gradient is

$$O(\mathcal{T}_{\mathrm{mat}}(n,d,d) + \mathcal{T}_{\mathrm{mat}}(n,d,n))$$

time. □

D Fast Running Time via Polynomial Method

Recall that in the previous section, for convenience of computing the derivative, we ignored the d factor in f. That factor d doesn't impact the running time of our algorithms since it is just a rescaling factor. To apply the tools from previous work [AS23], we will now reconsider the $1/d$ factor in f. In Section D.1, we will show how to efficiently and explicitly construct a low rank representation for f. In Section D.2, we show how to create a low rank construction for $c(x)$. In Section D.3, Section D.4 and Section D.5, we further give low rank presentations for $q(x), p_1(x), p_2(x)$. In Section D.6, we prove our final algorithmic result by putting everything together.

D.1 Low rank representation to f

Using [AS23]'s polynomial method result, we are able to obtain the following low-rank representation result,

Lemma D.1 (Section 3 of [AS23])**.** *For any $B = o(\sqrt{\log n})$, there exists a $k_1 = n^{o(1)}$ such that: Let $A_1, A_2 \in \mathbb{R}^{n \times d}$ be two matrices and $X \in \mathbb{R}^{d \times d}$ be a square matrix. It holds that $\|A_1^\top X\|_\infty \leq B, \|A_2\|_\infty \leq B$, then there are two matrices $U_1, V_1 \in \mathbb{R}^{n \times k_1}$ such that $\|U_1 V_1^\top - f(x)\|_\infty \leq \epsilon / \mathrm{poly}(n)$. Here $f(x) = D^{-1} \exp(A_1 X A_2^\top / d)$ and we define $D = \mathrm{diag}(\exp(A_1 X A_2^\top / d) \mathbf{1}_n)$. Moreover, these matrices U_1, V_1 can be explicitly constructed in $n^{1+o(1)}$ time.*

D.2 Low rank representation to c

Lemma D.2. *Let $d = O(\log n)$. Assume that each number in the $n \times d$ matrices E and $h(y)$ can be written using $O(\log n)$ bits. Let $n \times d$ matrix $c(x)$ be defined as Definition 3.6. Then, there are two matrices $U_1, V_1 \in \mathbb{R}^{n \times k_1}$ we have $\|U_1 V_1^\top h(y) - E - c(x)\|_\infty \leq \epsilon / \mathrm{poly}(n)$.*

Proof. We can show that

$$\begin{aligned} \|U_1 V_1^\top h(y) - E - c(x)\|_\infty &= \|U_1 V_1^\top h(y) - E - f(x) h(y) + E\|_\infty \\ &= \|(U_1 V_1^\top - f(x)) h(y)\|_\infty \\ &\leq \epsilon / \mathrm{poly}(n) \end{aligned}$$

where the first step follows from $c(x) = f(x) h(y) - E$.

□

D.3 Low rank representation to q

Lemma D.3. *Let $k_2 = n^{o(1)}$. Define $c(x) \in \mathbb{R}^{n \times d}$ to be as in Definition 3.6. Define $h(y) \in \mathbb{R}^{n \times d}$ to be as in Definition 3.5. Assume that $q(x) := h(y) c(x)^\top \in \mathbb{R}^{n \times n}$. There are two matrices $U_2, V_2 \in \mathbb{R}^{n \times k_2}$ such that $\|U_2 V_2^\top - q(x)\|_\infty \leq \epsilon / \mathrm{poly}(n)$. The matrices U_2, V_2 can be explicitly constructed in $n^{1+o(1)}$ time.*

Proof. We define $\widetilde{q}(x)$ to be the approximation of $q(x)$.

From Lemma D.2, we know that $U_1 V_1^\top h(y) - E$ is a good approximation to $c(x)$.

Then we should pick in this way $\widetilde{q}(x) = h(y)(U_1 V_1^\top h(y) - E)^\top$.

https://doi.org/10.52202/079017-1963

Now, let us turn $\widetilde{q}(x)$ into some low-rank representation

$$\widetilde{q}(x) = \underbrace{h(y)}_{n \times d} \underbrace{h(y)^\top}_{d \times n} \underbrace{V_1}_{n \times k_1} \underbrace{U_1^\top}_{k_1 \times n} - \underbrace{h(y)}_{n \times d} \underbrace{E^\top}_{d \times n}$$

It is obvious that we should can first compute $h(y)^\top V_1$ which only takes $n^{1+o(1)}$ time. Then since all the low rank matrices are known, then we can explicitly construct $U_2, V_2 \in \mathbb{R}^{n \times k_2}$ where $k_2 = \max\{d, k\} + d = n^{o(1)}$.

For controlling the error, we can show

$$\begin{aligned} \|\widetilde{q}(x) - q(x)\|_\infty &= \|h(y)(U_1 V_1^\top h(y)) - E)^\top - h(y) c(x)^\top\|_\infty \\ &\leq d \cdot \|h(y)\|_\infty \cdot \|U_1 V_1^\top h(y)) - E - c(x)\|_\infty \\ &\leq \epsilon / \operatorname{poly}(n) \end{aligned}$$

Thus, we complete the proof. □

D.4 Low rank representation to $p_1(x)$

Lemma D.4. *Let $k_1 = n^{o(1)}$. Let $k_2 = n^{o(1)}$. Assume that $p_1(x) := f(x) \circ q(x)$. Assume $U_1, V_1 \in \mathbb{R}^{n \times k_1}$ approximates the $f(x)$ such that $\|U_1 V_1^\top - f(x)\|_\infty \leq \epsilon / \operatorname{poly}(n)$. Assume $U_2, V_2 \in \mathbb{R}^{n \times k_2}$ approximates the $q(x) \in \mathbb{R}^{n \times n}$ such that $\|U_2 V_2^\top - q(x)\|_\infty \leq \epsilon / \operatorname{poly}(n)$. Then there are matrices $U_3, V_3 \in \mathbb{R}^{n \times k_3}$ such that $\|U_3 V_3^\top - p_1(x)\|_\infty \leq \epsilon / \operatorname{poly}(n)$. The matrices U_3, V_3 can be explicitly constructed in $n^{1+o(1)}$ time.*

Proof. We choose $U_3 = U_1 \oslash U_2$ and $V_3 = V_1 \oslash V_2$. This can be computed in $n^{1+o(1)}$ time.

For easy of writing proofs, we call $\widetilde{f}(x) = U_1 V_1^\top$ and $\widetilde{q}(x) = U_2 V_2^\top$.

Using Fact A.2, we know that

$$\begin{aligned} \|U_3 V_3^\top - p_1(x)\|_\infty &\leq \|U_3 V_3^\top - f(x) \circ q(x)\|_\infty \\ &= \|(U_1 \oslash U_2)(V_1 \oslash V_2)^\top - f(x) \circ q(x)\|_\infty \\ &= \|(U_1 V_1^\top) \circ (U_2 V_2^\top) - f(x) \circ q(x)\|_\infty \\ &= \|\widetilde{f}(x) \circ \widetilde{q}(x) - f(x) \circ q(x)\|_\infty \\ &= \|\widetilde{f}(x) \circ \widetilde{q}(x) - \widetilde{f}(x) \circ q(x) + \widetilde{f}(x) \circ q(x) - f(x) \circ q(x)\|_\infty \\ &\leq \|\widetilde{f}(x) \circ \widetilde{q}(x) - \widetilde{f}(x) \circ q(x)\|_\infty + \|\widetilde{f}(x) \circ q(x) - f(x) \circ q(x)\|_\infty \\ &\leq \epsilon / \operatorname{poly}(n) \end{aligned}$$

where the 1st step follows from the way we define $p_1(x)$, the 2nd step follows from the way we define U_3 and V_3, the 3rd step follows from Fact A.2, the 4th step follows from the way we define $\widetilde{f}(x)$ and $\widetilde{q}(x)$, the 5th step follows from simple algebra, the 6th step follows by triangle inequality, and the last step follows by that entries are bounded and $\|\widetilde{f}(x) - f(x)\|_\infty \leq \epsilon / \operatorname{poly}(n)$ (Lemma assumption) and $\|\widetilde{q}(x) - q(x)\|_\infty \leq \epsilon / \operatorname{poly}(n)$ (Lemma assumption)

□

D.5 Low rank representation $p_2(x)$

Lemma D.5. *Let $k_1 = n^{o(1)}$. Let $k_2 = n^{o(1)}$. Let $k_4 = n^{o(1)}$. Assume that $p_2(x)$ is an $n \times n$ where j_0-th column $p_2(x)_{j_0} = f(x)_{j_0} f(x)_{j_0}^\top q(x)_{j_0}$ for each $j_0 \in [n]$. Assume $U_1, V_1 \in \mathbb{R}^{n \times k_1}$ approximates the $f(x)$ such that $\|U_1 V_1^\top - f(x)\|_\infty \leq \epsilon / \operatorname{poly}(n)$. Assume $U_2, V_2 \in \mathbb{R}^{n \times k_2}$ approximates the $q(x) \in \mathbb{R}^{n \times n}$ such that $\|U_2 V_2^\top - q(x)\|_\infty \leq \epsilon / \operatorname{poly}(n)$. Then there are matrices $U_4, V_4 \in \mathbb{R}^{n \times k_4}$ such that $\|U_4 V_4^\top - p_2(x)\|_\infty \leq \epsilon / \operatorname{poly}(n)$. The matrices U_4, V_4 can be explicitly constructed in $n^{1+o(1)}$ time.*

https://doi.org/10.52202/079017-1963

Proof. We define a local vector function $r(x) \in \mathbb{R}^n$ where $r(x)_{j_0}$ is $f(x)_{j_0} q(x)_{j_0}$. Let $\widetilde{r}(x)$ denote the approximation of $r(x)$.

Note that $(U_1 V_1)_{j_0,*}^\top$ is a good approximation to $f(x)_{j_0}$.

Note that $(U_2 V_2)_{j_0,*}^\top$ is a good approximation to $q(x)_{j_0}$.

Let $\widetilde{r}(x)_{j_0} := \langle \widetilde{f}(x)_{j_0}, \widetilde{q}(x)_{j_0} \rangle = (U_1 V_1)_{j_0,*} \cdot (U_2 V_2)_{j_0,*}^\top$.

For the computation side, we firstly compute $V_1 V_2^\top$. This takes $n^{1+o(1)}$ time.

Next, we we have

$$\begin{aligned} \widetilde{r}(x)_{j_0} &= (U_1 V_1)_{j_0,*} \cdot (U_2 V_2)_{j_0,*}^\top \\ &= \underbrace{(U_1)_{j_0,*}}_{1 \times k_1} \underbrace{V_1 V_2^\top}_{k_1 \times k_2} \underbrace{(U_2)_{j_0,*}^\top}_{k_2 \times 1} \end{aligned}$$

Once the $V_1 V_2^\top$ are pre-computed, the above step only takes $O(k_1 k_2)$ time. Since there n coordinates, so the overall time is still $O(n k_1 k_2) = n^{1+o(1)}$.

Let $\widetilde{f}(x) = U_1 V_1^\top$ denote the approximation of $f(x)$. Then we just use $\widetilde{f}(x)$ and $\widetilde{r}(x)$ to approximate $p_2(x)$ in the following sense, let $\widetilde{p}_2(x) = \widetilde{f}(x) \mathrm{diag}(\widetilde{r}(x))$. Since $\widetilde{f}(x)$ has low rank representation, and $\mathrm{diag}(\widetilde{r}(x))$ is a diagonal matrix, then it is obvious how to construct U_4 and V_4. Basically $U_4 = U_1$ and $V_4 = \mathrm{diag}(\widetilde{r}(x)) V_1$.

Now, we need to control the error, we have

$$\begin{aligned} \|U_4 V_4^\top - p_2(x)\|_\infty &= \|\widetilde{p}_2(x) - p_2(x)\|_\infty \\ &= \max_{j_0 \in [n]} \|\widetilde{f}(x)_{j_0} \widetilde{r}(x)_{j_0} - f(x)_{j_0} r(x)_{j_0}\|_\infty \\ &= \max_{j_0 \in [n]} \|\widetilde{f}(x)_{j_0} \widetilde{r}(x)_{j_0} - \widetilde{f}(x)_{j_0} r(x)_{j_0} + \widetilde{f}(x)_{j_0} r(x)_{j_0} - f(x)_{j_0} r(x)_{j_0}\|_\infty \\ &\leq \max_{j_0 \in [n]} \|\widetilde{f}(x)_{j_0} \widetilde{r}(x)_{j_0} - \widetilde{f}(x)_{j_0} r(x)_{j_0}\|_\infty + \|\widetilde{f}(x)_{j_0} r(x)_{j_0} - f(x)_{j_0} r(x)_{j_0}\|_\infty \end{aligned}$$

where the 2nd step follows follows from definition of $p_2(x)$ and $\widetilde{p}_2(x)$.

For the first term, we have

$$\begin{aligned} \max_{j_0 \in [n]} \|\widetilde{f}(x)_{j_0} \widetilde{r}(x)_{j_0} - \widetilde{f}(x)_{j_0} r(x)_{j_0}\|_\infty &\leq \max_{j_0 \in [n]} \|\widetilde{f}(x)_{j_0}\|_\infty \cdot |\widetilde{r}(x)_{j_0} - r(x)_{j_0}| \\ &\leq \epsilon / \mathrm{poly}(n) \end{aligned}$$

For the second term, we have

$$\begin{aligned} \max_{j_0 \in [n]} \|\widetilde{f}(x)_{j_0} r(x)_{j_0} - f(x)_{j_0} r(x)_{j_0}\|_\infty &\leq \max_{j_0 \in [n]} \|\widetilde{f}(x)_{j_0} - f(x)_{j_0}\|_\infty \cdot |r(x)_{j_0}| \\ &\leq \epsilon / \mathrm{poly}(n) \end{aligned}$$

Using the three equations we obtained above, the proof is completed. □

D.6 Fast Computation in Almost Linear Time

Theorem D.6 (Main result, formal version of Theorem 1.6)**.** *Assuming the entries of A_1, A_2, X, A_3, Y, E are represented using $O(\log n)$ bits, there is a $n^{1+o(1)}$ time algorithm to solve $\mathsf{AAttLGC}(n, d = O(\log n), B = o(\sqrt{\log n}))$ (see Definition 1.4) up to $1/\mathrm{poly}(n)$ accuracy. In particular, our algorithm outputs a gradient vector $\widetilde{g} \in \mathbb{R}^{d^2}$ such that $\|\frac{\mathrm{d}L}{\mathrm{d}x} - \widetilde{g}\|_\infty \leq 1/\mathrm{poly}(n)$.*

Proof. Recall definition of $n \times n$ matrices $p(x)$ (Definition C.5), $p_1(x)$ (see Lemma D.5) and $p_2(x)$ (Lemma D.4), it is straightforward that

$$p(x) = p_1(x) - p_2(x).$$

Using Lemma D.1, Lemma D.2, Lemma D.3, we know that assumptions in Lemma D.4 and Lemma D.5 are holding, so that we can use Lemma D.4 and Lemma D.5 to obtain that

- $p_1(x)$ has approximate low rank representation U_3, V_3, let $\widetilde{p}_1(x)$ denote $U_3 V_3^\top$
- $p_2(x)$ has approximate low rank representation U_4, V_4, let $\widetilde{p}_2(x)$ denote $U_4 V_4^\top$

All of the Lemmas D.1, D.2, D.3, D.4 and D.5 are taking $n^{1+o(1)}$ time.

According to the proof for the Lemma C.7, we have that

$$\frac{L(X)}{\mathrm{d}x} = \mathrm{vec}(A_1^\top p(x) A_2)$$

Thus, we firstly compute $A_1^\top U_3 V_3^\top A_2$,

- We compute $A_1^\top U_3 \in \mathbb{R}^{d \times k_3}$, this takes $n^{1+o(1)}$ time
- We compute $V_3^\top A_2 \in \mathbb{R}^{k_3 \times d}$, this takes $n^{1+o(1)}$ time
- Compute $(A_1^\top U_3) \cdot (V_3^\top A_2)$, this takes $d^2 n^{o(1)}$ time

Second, we can compute $A_1^\top U_4 V_4^\top A_2$,

- We compute $A_1^\top U_4 \in \mathbb{R}^{d \times k_4}$, this takes $n^{1+o(1)}$ time
- We compute $V_4^\top A_2 \in \mathbb{R}^{k_4 \times d}$, this takes $n^{1+o(1)}$ time
- Compute $(A_1^\top U_4) \cdot (V_4^\top A_2)$, this takes $d^2 n^{o(1)}$ time

So, overall running time is still $n^{1+o(1)}$.

We have

$$\begin{aligned}
\| \frac{\mathrm{d}L(X)}{\mathrm{d}x} - \widetilde{g} \|_\infty &= \| \mathrm{vec}(A_1^\top p(x) A_2) - \mathrm{vec}(A_1^\top \widetilde{p}(x) A_2) \|_\infty \\
&= \| A_1^\top p(x) A_2 - A_1^\top \widetilde{p}(x) A_2 \|_\infty \\
&= \| A_1^\top (p_1(x) - p_2(x)) A_2 - A_1^\top (\widetilde{p}_1(x) - \widetilde{p}_2(x)) A_2 \|_\infty \\
&\leq \| A_1^\top (p_1(x) - \widetilde{p}_1(x)) A_2 \|_\infty + \| A_1^\top (p_2(x) - \widetilde{p}_2(x)) A_2 \|_\infty \\
&\leq \| A_1 \|_\infty \| A_2 \|_\infty \cdot n^2 \cdot (\| p_1(x) - \widetilde{p}_1(x) \|_\infty + \| p_2(x) - \widetilde{p}_2(x) \|_\infty) \\
&\leq \epsilon / \mathrm{poly}(n)
\end{aligned}$$

where the 4th step follows from triangle inequality, the last step follows from entries in A_1, A_2 are bounded, and $\| p_1(x) - \widetilde{p}_1(x) \|_\infty \leq \epsilon / \mathrm{poly}(n)$, $\| p_2(x) - \widetilde{p}_2(x) \|_\infty \leq \epsilon / \mathrm{poly}(n)$.

Picking $\epsilon = 1/\mathrm{poly}(n)$, we have the proof completed. □

E General Lower Bound

We will critically make use of the known hardness result for attention computation itself, which we state now.

Definition E.1 (Attention Computation). *Given as input matrices $Q, K, V \in \mathbb{R}^{n \times d}$ and a parameter $\varepsilon > 0$, compute a matrix $T \in \mathbb{R}^{n \times d}$ satisfying*

$$\| T - D^{-1} A V \|_\infty \leq \varepsilon,$$

where $A = \exp(QK^\top)$ and $D = \mathrm{diag}(A \mathbf{1}_n)$.

Lemma E.2 (Lemma 4.7 in [AS23]). *Assuming* SETH*, there is no algorithm running in time $O(n^{2-\delta})$ for any constant $\delta > 0$ that solves Attention Computation (Definition E.1), even when the inputs satisfy the following constraints, for any parameter $\kappa \geq 0$:*

- *$d = O(\log n)$,*

https://doi.org/10.52202/079017-1963

- $V \in \{0,1\}^{n \times d}$,
- *There is a value* $B \leq O(\log^2 n \cdot (1+\kappa))$ *such that every entry of* $QK^\top$ *is in the interval* $[0, B]$ *and at least half the entries in each row of* $QK^\top$ *are equal to* B,
- *moreover* $\|Q\|_\infty, \|K\|_\infty \leq O(\sqrt{\log n(1+\kappa)})$, *and*
- $\varepsilon < n^{\kappa - O(1)}$.

Next, we show that the attention optimization problem behaves particularly well when given matrices constrained as in Lemma E.2:

Lemma E.3. *Let* A *be a fixed* $n \times n$ *matrix whose entries are real numbers in the interval* $[0, B]$, *and such that in each row of* A, *at least half the entries are equal to* B. *Let* V *be any* $n \times d$ *matrix whose entries are all in* $\{0, 1\}$. *For* $\lambda \in \mathbb{R}$, *define the* $n \times n$ *matrix* $M_\lambda := \exp(\lambda A)$, *where* exp *is applied entry-wise. Define the function* $f : \mathbb{R} \to \mathbb{R}$ *by*

$$f(\lambda) := \|\mathrm{diag}(M_\lambda \mathbf{1}_n)^{-1} M_\lambda V\|_F^2,$$

Then, for all $\lambda \in \mathbb{R}$ *we have*

- $|f'(\lambda)| \leq O(Bn)$,
- $|f''(\lambda)| \leq O(B^2 n)$.

Proof. Let C denote the $n \times n$ matrix $C = \mathrm{diag}(M_\lambda \mathbf{1}_n)^{-1} M_\lambda$. For $i, j \in [n]$, we calculate that $M_\lambda[i,j] = e^{\lambda A[i,j]}$ and so

$$C[i,j] = \frac{e^{\lambda A[i,j]}}{\sum_{k=1}^n e^{\lambda A[i,k]}}.$$

For $\ell \in [d]$, let $S_\ell \subseteq [n]$ be the set of 1s in column ℓ of V, i.e., $S_\ell = \{j \in [n] \mid V[j,\ell] = 1\}$. Hence, for $i \in [n]$ and $\ell \in [d]$, the entry (i, ℓ) of the matrix $\mathrm{diag}(M_\lambda \mathbf{1}_n)^{-1} M_\lambda V$ is given by

$$\begin{aligned}\mathrm{diag}(M_\lambda \mathbf{1}_n)^{-1} M_\lambda V[i,\ell] &= CV[i,\ell] \\ &= \sum_{j=1}^n C[i,j] V[j,\ell] \\ &= \sum_{j \in S_\ell} C[i,j] \\ &= \frac{\sum_{j \in S_\ell} e^{\lambda A[i,j]}}{\sum_{k=1}^n e^{\lambda A[i,k]}}.\end{aligned}$$

where the first step follows from definition, the second step follows from simple algebra.

We thus get an explicit expression for $f(\lambda)$:

$$\begin{aligned}f(\lambda) &= \sum_{i=1}^n \frac{\sum_{\ell=1}^d \left(\sum_{j \in S_\ell} e^{\lambda A[i,j]}\right)^2}{\left(\sum_{k=1}^n e^{\lambda A[i,k]}\right)^2} \\ &= \sum_{i=1}^n \frac{\sum_{\ell=1}^d \sum_{j_1 \in S_\ell}^n \sum_{j_2 \in S_\ell}^n e^{\lambda(A[i,j_1] + A[i,j_2])}}{\sum_{k_1=1}^n \sum_{k_2=1}^n e^{\lambda(A[i,k_1] + A[i,k_2])}}.\end{aligned}$$

We define

$$a(\lambda, i) := \sum_{\ell=1}^d \sum_{j_1 \in S_\ell}^n \sum_{j_2 \in S_\ell}^n e^{\lambda(A[i,j_1] + A[i,j_2])}$$

https://doi.org/10.52202/079017-1963

and then we define

$$b(\lambda, i) := \sum_{k_1=1}^{n} \sum_{k_2=1}^{n} e^{\lambda(A[i,k_1]+A[i,k_2])}$$

Combining the above three equations, we can obtain

$$f(\lambda) = \sum_{i=1}^{n} a(\lambda, i)/b(\lambda, i).$$

Since, for each row of A, at least half the entries equal B, and all the entries are in the interval $[1, B]$, we can bound

$$\left(\frac{n}{2}\right)^2 \cdot e^{2B\lambda} \leq b(\lambda, i) \leq (n)^2 \cdot e^{2B\lambda}. \tag{7}$$

Furthermore, since the derivative of $e^{\lambda(A[i,k_1]+A[i,k_2])}$ with respect to λ is $(A[i, k_1] + A[i, k_2]) \cdot e^{\lambda(A[i,k_1]+A[i,k_2])}$, we can bound

$$2 \cdot b(\lambda, i) \leq \frac{\mathrm{d}b(\lambda, i)}{\mathrm{d}\lambda} \leq 2B \cdot b(\lambda, i). \tag{8}$$

We may similarly bound

$$0 \leq a(\lambda, i) \leq n^2 \cdot e^{2B\lambda}, \tag{9}$$

and

$$2 \cdot a(\lambda, i) \leq \frac{\mathrm{d}a(\lambda, i)}{\mathrm{d}\lambda} \leq 2B \cdot a(\lambda, i). \tag{10}$$

We can thus bound the derivative of f (where here, all the $'$ notation means derivative with respect to λ):

$$\begin{aligned} f'(\lambda) &= \sum_{i=1}^{n} \frac{a'(\lambda, i) \cdot b(\lambda, i) - a(\lambda, i) \cdot b'(\lambda, i)}{(b(\lambda, i))^2} \\ &\leq \sum_{i=1}^{n} \frac{a'(\lambda, i) \cdot b(\lambda, i)}{(b(\lambda, i))^2} \\ &= \sum_{i=1}^{n} \frac{a'(\lambda, i)}{b(\lambda, i)} \\ &\leq \sum_{i=1}^{n} \frac{2B \cdot n^2 e^{2B\lambda}}{(n/2)^2 \cdot e^{2B\lambda}} \\ &= \sum_{i=1}^{n} 8B \\ &= 8B \cdot n. \end{aligned}$$

where the 1st step follows from definition, the 2nd step follows from simple algebra, the 3rd step follows from cancelling $b(\lambda, i)$, the 4th step is using Eq. (7) (for $b(\lambda, i)$) and Eq. (10) (for $a'(\lambda, i)$), the 5th step follows from simple algebra, and the last step follows from simple algebra.

Similarly, we can provide a lower bound $f'(\lambda)$,

$$\begin{aligned} f'(\lambda) &= \sum_{i=1}^{n} \frac{a'(\lambda, i) \cdot b(\lambda, i) - a(\lambda, i) \cdot b'(\lambda, i)}{(b(\lambda, i))^2} \\ &\geq -\sum_{i=1}^{n} \frac{a(\lambda, i) \cdot b'(\lambda, i)}{(b(\lambda, i))^2} \end{aligned}$$

$$
\begin{aligned}
&\geq -\sum_{i=1}^{n} \frac{(n^2 \cdot e^{2B\lambda}) \cdot (2B \cdot b(\lambda, i))}{((n/2)^2 \cdot e^{2B\lambda}) \cdot (b(\lambda, i))} \\
&= -\sum_{i=1}^{n} 8B \\
&= -8B \cdot n.
\end{aligned}
$$

where the 1st step follows from definition, the 2nd step follows form simple algebra, the 3rd step follows Eq. (8) (for $b'(\lambda, i)$) and Eq. (9) (for $a(\lambda, i)$), the 4th step follows from simple algebra, and the last step follows from simple algbera.

Finally, letting $f(\lambda, i) := a(\lambda, i)/b(\lambda, i)$, we have again by the quotient rule that $f''(\lambda)$ is equal to

$$\sum_{i=1}^{n} \frac{a''(\lambda, i) - b''(\lambda, i) \cdot f(\lambda, i) - 2 \cdot b'(\lambda, i) \cdot f'(\lambda, i)}{b(\lambda, i)}$$

which we similarly bound in magnitude by $O(B^2 n)$. □

We recall a simple approximation from calculus:

Lemma E.4. *Let $f : [0, 1] \to \mathbb{R}$ be a twice-differentiable function such that $|f''(\lambda)| \leq b$ for all $\lambda \in [0, 1]$. For any positive integer m, define the sum*

$$t_m := \sum_{i=0}^{m-1} \frac{f'(i/m)}{m}.$$

Then,

$$|t_m - (f(1) - f(0))| \leq b/m.$$

Proof. If two $\lambda_0, \lambda_1 \in [0, 1]$ have $|\lambda_0 - \lambda_1| \leq 1/m$, then from our bound on $f''(\lambda)$, we know that $|f'(\lambda_1) - f'(\lambda_0)| \leq b/m$. We can thus bound the difference

$$f(1) - f(0) = \int_0^1 f'(\lambda) d\lambda$$

by

$$f(1) - f(0) \leq \sum_{i=0}^{m-1} \frac{f'(i/m) + (b/m)}{m} = t_m + b/m$$

and

$$f(1) - f(0) \geq \sum_{i=0}^{m-1} \frac{f'(i/m) - (b/m)}{m} = t_m - b/m.$$

Thus, we complete the proof. □

Finally, we are ready for our main result:

Theorem E.5 (Formal version of Theorem 1.5)**.** *Let $\kappa : \mathcal{N} \to \mathcal{N}$ by any function with $\kappa(n) = \omega(1)$ and $\kappa(n) = o(\log n)$. Assuming SETH, there is no algorithm running in time $O(n^{2-\delta})$ for any constant $\delta > 0$ for Approximate Attention Loss Gradient Computation (Definition 1.4), even in the case where $d = O(\log n)$ and the input matrices satisfy $\|A_1\|_\infty, \|A_2\|_\infty, \|A_3\|_\infty \leq O(\sqrt{\log n} \cdot \kappa(n))$, $B = 0$, $Y = I$, $X = \lambda I$ for some scalar $\lambda \in [0, 1]$, and $\varepsilon = O(1/(\log n)^4)$.*

Proof. Suppose there were such an algorithm. We call it $O((\log n)^4)$ times to refute Lemma E.2 (with parameter $\kappa = \kappa(n)$). Let Q, K, V be the input matrices to Lemma E.2, and set $A_1 = Q$, $A_2 = K$, $A_3 = V$, $Y = I$, and $X = \lambda I$ for a parameter $\lambda \in [0, 1]$. Suppose the function $f : [0, 1] \to \mathbb{R}$ is in Lemma E.3 where A is the matrix $A_1 A_2^\top$, so that M_λ is the matrix $\exp(A_1 X A_2^\top)$. It follows from Lemma E.3 that

$$|f''(\lambda)| \leq O(n \log^2 n \cdot (\kappa(n))^2).$$

https://doi.org/10.52202/079017-1963

We can compute $f(0)$ in $\widetilde{O}(n)$ time since then M_f is the all-1s matrix, and our goal is to output $f(1)$.

Thus, by Lemma E.4, it suffices to compute $f'(\lambda)$ on $O(\log^2(n)(\kappa(n))^2) = O(\log^4 n)$ points up to $O(1/(\log n)^4)$ error, and return their average. But, since we have picked $X = \lambda I$, we can calculate $f'(\lambda)$ from the gradient $\frac{\mathrm{d}L(X)}{\mathrm{d}X}$ (from Definition 1.4), which is approximated by our assumed algorithm. □

https://doi.org/10.52202/079017-1963

NeurIPS Paper Checklist

1. **Claims**

   Question: Do the main claims made in the abstract and introduction accurately reflect the paper's contributions and scope?

   Answer: [Yes]

   Justification: All the formal statements made in the introduction, and summarized in the abstract, are proved in the appendix. We prove our algorithmic result in Section D, and prove our lower bound in Section E.

   Guidelines:

   - The answer NA means that the abstract and introduction do not include the claims made in the paper.
   - The abstract and/or introduction should clearly state the claims made, including the contributions made in the paper and important assumptions and limitations. A No or NA answer to this question will not be perceived well by the reviewers.
   - The claims made should match theoretical and experimental results, and reflect how much the results can be expected to generalize to other settings.
   - It is fine to include aspirational goals as motivation as long as it is clear that these goals are not attained by the paper.

2. **Limitations**

   Question: Does the paper discuss the limitations of the work performed by the authors?

   Answer: [Yes]

   Justification: We discuss the limitations in Section 6.

   Guidelines:

   - The answer NA means that the paper has no limitation while the answer No means that the paper has limitations, but those are not discussed in the paper.
   - The authors are encouraged to create a separate "Limitations" section in their paper.
   - The paper should point out any strong assumptions and how robust the results are to violations of these assumptions (e.g., independence assumptions, noiseless settings, model well-specification, asymptotic approximations only holding locally). The authors should reflect on how these assumptions might be violated in practice and what the implications would be.
   - The authors should reflect on the scope of the claims made, e.g., if the approach was only tested on a few datasets or with a few runs. In general, empirical results often depend on implicit assumptions, which should be articulated.
   - The authors should reflect on the factors that influence the performance of the approach. For example, a facial recognition algorithm may perform poorly when image resolution is low or images are taken in low lighting. Or a speech-to-text system might not be used reliably to provide closed captions for online lectures because it fails to handle technical jargon.
   - The authors should discuss the computational efficiency of the proposed algorithms and how they scale with dataset size.
   - If applicable, the authors should discuss possible limitations of their approach to address problems of privacy and fairness.
   - While the authors might fear that complete honesty about limitations might be used by reviewers as grounds for rejection, a worse outcome might be that reviewers discover limitations that aren't acknowledged in the paper. The authors should use their best judgment and recognize that individual actions in favor of transparency play an important role in developing norms that preserve the integrity of the community. Reviewers will be specifically instructed to not penalize honesty concerning limitations.

3. **Theory Assumptions and Proofs**

   Question: For each theoretical result, does the paper provide the full set of assumptions and a complete (and correct) proof?

https://doi.org/10.52202/079017-1963

Answer: [Yes]

Justification: All our results are completely formally stated and proved. The main assumption we use in our lower bound, SETH, is defined in Section 3 and we state that we use it in our lower bound Theorem 1.5. Theorem 1.6 is our main algorithmic result, which we prove in Section D. Our warmup Lemma 4.1 is proved in Section C. Theorem 1.5 is our main lower bound, which we prove in Section E. All other Lemmas and Theorems are proved immediately after their statements.

Guidelines:

- The answer NA means that the paper does not include theoretical results.
- All the theorems, formulas, and proofs in the paper should be numbered and cross-referenced.
- All assumptions should be clearly stated or referenced in the statement of any theorems.
- The proofs can either appear in the main paper or the supplemental material, but if they appear in the supplemental material, the authors are encouraged to provide a short proof sketch to provide intuition.
- Inversely, any informal proof provided in the core of the paper should be complemented by formal proofs provided in appendix or supplemental material.
- Theorems and Lemmas that the proof relies upon should be properly referenced.

4. **Experimental Result Reproducibility**

Question: Does the paper fully disclose all the information needed to reproduce the main experimental results of the paper to the extent that it affects the main claims and/or conclusions of the paper (regardless of whether the code and data are provided or not)?

Answer: [NA]

Justification: This paper is a purely theoretical paper, and it doesn't include any experiments.

Guidelines:

- The answer NA means that the paper does not include experiments.
- If the paper includes experiments, a No answer to this question will not be perceived well by the reviewers: Making the paper reproducible is important, regardless of whether the code and data are provided or not.
- If the contribution is a dataset and/or model, the authors should describe the steps taken to make their results reproducible or verifiable.
- Depending on the contribution, reproducibility can be accomplished in various ways. For example, if the contribution is a novel architecture, describing the architecture fully might suffice, or if the contribution is a specific model and empirical evaluation, it may be necessary to either make it possible for others to replicate the model with the same dataset, or provide access to the model. In general. releasing code and data is often one good way to accomplish this, but reproducibility can also be provided via detailed instructions for how to replicate the results, access to a hosted model (e.g., in the case of a large language model), releasing of a model checkpoint, or other means that are appropriate to the research performed.
- While NeurIPS does not require releasing code, the conference does require all submissions to provide some reasonable avenue for reproducibility, which may depend on the nature of the contribution. For example
  (a) If the contribution is primarily a new algorithm, the paper should make it clear how to reproduce that algorithm.
  (b) If the contribution is primarily a new model architecture, the paper should describe the architecture clearly and fully.
  (c) If the contribution is a new model (e.g., a large language model), then there should either be a way to access this model for reproducing the results or a way to reproduce the model (e.g., with an open-source dataset or instructions for how to construct the dataset).
  (d) We recognize that reproducibility may be tricky in some cases, in which case authors are welcome to describe the particular way they provide for reproducibility. In the case of closed-source models, it may be that access to the model is limited in

https://doi.org/10.52202/079017-1963

some way (e.g., to registered users), but it should be possible for other researchers to have some path to reproducing or verifying the results.

5. **Open access to data and code**

   Question: Does the paper provide open access to the data and code, with sufficient instructions to faithfully reproduce the main experimental results, as described in supplemental material?

   Answer: [NA]

   Justification: This paper is a purely theoretical paper, and it doesn't include any experiments.

   Guidelines:

   - The answer NA means that paper does not include experiments requiring code.
   - Please see the NeurIPS code and data submission guidelines (https://nips.cc/public/guides/CodeSubmissionPolicy) for more details.
   - While we encourage the release of code and data, we understand that this might not be possible, so "No" is an acceptable answer. Papers cannot be rejected simply for not including code, unless this is central to the contribution (e.g., for a new open-source benchmark).
   - The instructions should contain the exact command and environment needed to run to reproduce the results. See the NeurIPS code and data submission guidelines (https://nips.cc/public/guides/CodeSubmissionPolicy) for more details.
   - The authors should provide instructions on data access and preparation, including how to access the raw data, preprocessed data, intermediate data, and generated data, etc.
   - The authors should provide scripts to reproduce all experimental results for the new proposed method and baselines. If only a subset of experiments are reproducible, they should state which ones are omitted from the script and why.
   - At submission time, to preserve anonymity, the authors should release anonymized versions (if applicable).
   - Providing as much information as possible in supplemental material (appended to the paper) is recommended, but including URLs to data and code is permitted.

6. **Experimental Setting/Details**

   Question: Does the paper specify all the training and test details (e.g., data splits, hyperparameters, how they were chosen, type of optimizer, etc.) necessary to understand the results?

   Answer: [NA]

   Justification: This paper is a purely theoretical paper, and it doesn't include any experiments.

   Guidelines:

   - The answer NA means that the paper does not include experiments.
   - The experimental setting should be presented in the core of the paper to a level of detail that is necessary to appreciate the results and make sense of them.
   - The full details can be provided either with the code, in appendix, or as supplemental material.

7. **Experiment Statistical Significance**

   Question: Does the paper report error bars suitably and correctly defined or other appropriate information about the statistical significance of the experiments?

   Answer: [NA]

   Justification: This paper is a purely theoretical paper, and it doesn't include any experiments.

   Guidelines:

   - The answer NA means that the paper does not include experiments.
   - The authors should answer "Yes" if the results are accompanied by error bars, confidence intervals, or statistical significance tests, at least for the experiments that support the main claims of the paper.

https://doi.org/10.52202/079017-1963

- The factors of variability that the error bars are capturing should be clearly stated (for example, train/test split, initialization, random drawing of some parameter, or overall run with given experimental conditions).
- The method for calculating the error bars should be explained (closed form formula, call to a library function, bootstrap, etc.)
- The assumptions made should be given (e.g., Normally distributed errors).
- It should be clear whether the error bar is the standard deviation or the standard error of the mean.
- It is OK to report 1-sigma error bars, but one should state it. The authors should preferably report a 2-sigma error bar than state that they have a 96% CI, if the hypothesis of Normality of errors is not verified.
- For asymmetric distributions, the authors should be careful not to show in tables or figures symmetric error bars that would yield results that are out of range (e.g. negative error rates).
- If error bars are reported in tables or plots, The authors should explain in the text how they were calculated and reference the corresponding figures or tables in the text.

8. **Experiments Compute Resources**

   Question: For each experiment, does the paper provide sufficient information on the computer resources (type of compute workers, memory, time of execution) needed to reproduce the experiments?

   Answer: [NA]

   Justification: This paper is a purely theoretical paper, and it doesn't include any experiments.

   Guidelines:

   - The answer NA means that the paper does not include experiments.
   - The paper should indicate the type of compute workers CPU or GPU, internal cluster, or cloud provider, including relevant memory and storage.
   - The paper should provide the amount of compute required for each of the individual experimental runs as well as estimate the total compute.
   - The paper should disclose whether the full research project required more compute than the experiments reported in the paper (e.g., preliminary or failed experiments that didn't make it into the paper).

9. **Code Of Ethics**

   Question: Does the research conducted in the paper conform, in every respect, with the NeurIPS Code of Ethics https://neurips.cc/public/EthicsGuidelines?

   Answer: [Yes]

   Justification: We have the read the NeurIPS code of Ethics.

   Guidelines:

   - The answer NA means that the authors have not reviewed the NeurIPS Code of Ethics.
   - If the authors answer No, they should explain the special circumstances that require a deviation from the Code of Ethics.
   - The authors should make sure to preserve anonymity (e.g., if there is a special consideration due to laws or regulations in their jurisdiction).

10. **Broader Impacts**

    Question: Does the paper discuss both potential positive societal impacts and negative societal impacts of the work performed?

    Answer: [Yes]

    Justification: We have discussed that in Section 7.

    Guidelines:

    - The answer NA means that there is no societal impact of the work performed.
    - If the authors answer NA or No, they should explain why their work has no societal impact or why the paper does not address societal impact.

https://doi.org/10.52202/079017-1963

- Examples of negative societal impacts include potential malicious or unintended uses (e.g., disinformation, generating fake profiles, surveillance), fairness considerations (e.g., deployment of technologies that could make decisions that unfairly impact specific groups), privacy considerations, and security considerations.
- The conference expects that many papers will be foundational research and not tied to particular applications, let alone deployments. However, if there is a direct path to any negative applications, the authors should point it out. For example, it is legitimate to point out that an improvement in the quality of generative models could be used to generate deepfakes for disinformation. On the other hand, it is not needed to point out that a generic algorithm for optimizing neural networks could enable people to train models that generate Deepfakes faster.
- The authors should consider possible harms that could arise when the technology is being used as intended and functioning correctly, harms that could arise when the technology is being used as intended but gives incorrect results, and harms following from (intentional or unintentional) misuse of the technology.
- If there are negative societal impacts, the authors could also discuss possible mitigation strategies (e.g., gated release of models, providing defenses in addition to attacks, mechanisms for monitoring misuse, mechanisms to monitor how a system learns from feedback over time, improving the efficiency and accessibility of ML).

11. **Safeguards**

    Question: Does the paper describe safeguards that have been put in place for responsible release of data or models that have a high risk for misuse (e.g., pretrained language models, image generators, or scraped datasets)?

    Answer: [NA]

    Justification: This paper is a purely theoretical paper, and it doesn't include any experiments.

    Guidelines:

    - The answer NA means that the paper poses no such risks.
    - Released models that have a high risk for misuse or dual-use should be released with necessary safeguards to allow for controlled use of the model, for example by requiring that users adhere to usage guidelines or restrictions to access the model or implementing safety filters.
    - Datasets that have been scraped from the Internet could pose safety risks. The authors should describe how they avoided releasing unsafe images.
    - We recognize that providing effective safeguards is challenging, and many papers do not require this, but we encourage authors to take this into account and make a best faith effort.

12. **Licenses for existing assets**

    Question: Are the creators or original owners of assets (e.g., code, data, models), used in the paper, properly credited and are the license and terms of use explicitly mentioned and properly respected?

    Answer: [NA]

    Justification: This paper is a purely theoretical paper, and it doesn't include any experiments.

    Guidelines:

    - The answer NA means that the paper does not use existing assets.
    - The authors should cite the original paper that produced the code package or dataset.
    - The authors should state which version of the asset is used and, if possible, include a URL.
    - The name of the license (e.g., CC-BY 4.0) should be included for each asset.
    - For scraped data from a particular source (e.g., website), the copyright and terms of service of that source should be provided.
    - If assets are released, the license, copyright information, and terms of use in the package should be provided. For popular datasets, paperswithcode.com/datasets has curated licenses for some datasets. Their licensing guide can help determine the license of a dataset.

https://doi.org/10.52202/079017-1963

- For existing datasets that are re-packaged, both the original license and the license of the derived asset (if it has changed) should be provided.
- If this information is not available online, the authors are encouraged to reach out to the asset's creators.

13. **New Assets**

    Question: Are new assets introduced in the paper well documented and is the documentation provided alongside the assets?

    Answer: [NA]

    Justification: This paper is a purely theoretical paper, and it doesn't include any experiments.

    Guidelines:

    - The answer NA means that the paper does not release new assets.
    - Researchers should communicate the details of the dataset/code/model as part of their submissions via structured templates. This includes details about training, license, limitations, etc.
    - The paper should discuss whether and how consent was obtained from people whose asset is used.
    - At submission time, remember to anonymize your assets (if applicable). You can either create an anonymized URL or include an anonymized zip file.

14. **Crowdsourcing and Research with Human Subjects**

    Question: For crowdsourcing experiments and research with human subjects, does the paper include the full text of instructions given to participants and screenshots, if applicable, as well as details about compensation (if any)?

    Answer: [NA]

    Justification: This paper is a purely theoretical paper, and it doesn't include any experiments.

    Guidelines:

    - The answer NA means that the paper does not involve crowdsourcing nor research with human subjects.
    - Including this information in the supplemental material is fine, but if the main contribution of the paper involves human subjects, then as much detail as possible should be included in the main paper.
    - According to the NeurIPS Code of Ethics, workers involved in data collection, curation, or other labor should be paid at least the minimum wage in the country of the data collector.

15. **Institutional Review Board (IRB) Approvals or Equivalent for Research with Human Subjects**

    Question: Does the paper describe potential risks incurred by study participants, whether such risks were disclosed to the subjects, and whether Institutional Review Board (IRB) approvals (or an equivalent approval/review based on the requirements of your country or institution) were obtained?

    Answer: [NA]

    Justification: This paper is a purely theoretical paper, and it doesn't include any experiments.

    Guidelines:

    - The answer NA means that the paper does not involve crowdsourcing nor research with human subjects.
    - Depending on the country in which research is conducted, IRB approval (or equivalent) may be required for any human subjects research. If you obtained IRB approval, you should clearly state this in the paper.
    - We recognize that the procedures for this may vary significantly between institutions and locations, and we expect authors to adhere to the NeurIPS Code of Ethics and the guidelines for their institution.
    - For initial submissions, do not include any information that would break anonymity (if applicable), such as the institution conducting the review.

https://doi.org/10.52202/079017-1963