How Well Can Transformers Emulate In-Context Newton's Method?

Angeliki Giannou
University of Wisconsin-Madison

Liu Yang
University of Wisconsin-Madison

Tianhao Wang
Toyota Technological Institute at Chicago

Dimitris Papailiopoulos
Microsoft Research, AI Frontiers
University of Wisconsin-Madison

Jason D. Lee
Princeton University

Abstract

Transformer-based models have demonstrated remarkable in-context learning capabilities, prompting extensive research into its underlying mechanisms. Recent studies have suggested that Transformers can implement first-order optimization algorithms for in-context learning and even second order ones for the case of linear regression. In this work, we study whether Transformers can perform higher order optimization methods, beyond the case of linear regression. We establish that linear attention Transformers with ReLU layers can approximate second order optimization algorithms for the task of logistic regression and achieve ϵ error with only logarithmic to the error more layers. Our results suggest the ability of the Transformer architecture to implement complex algorithms, beyond gradient descent.

1 INTRODUCTION

Transformer networks have had a significant impact in machine learning, particularly in tasks related to natural language processing and computer vision [Vaswani et al., 2017, Khan et al., 2022, Yuan et al., 2021, Dosovitskiy et al., 2020]. A key building block of Transformers is the self-attention mechanism, which enables the model to weigh the significance of different parts of the input data with respect to each other. This allows the model to capture long-range dependencies and learn complex patterns of the data, yielding state-of-the-art performance across several tasks, including but not limited to language translation, text summarization, and conversational agents [Vaswani et al., 2017, Kenton and Toutanova, 2019].

It has been long observed that Transformers are able to perform various downstream tasks at inference without any parameter updates [Brown et al., 2020, Lieber et al., 2021, Black et al., 2022]. This ability, known as *in-context learning*, has attracted the interest of the community, resulting in a line of works aiming to interpret and understand it. Towards this direction, Garg et al. [2022] were the first to consider a setting, in which the "language" component of the problem is removed from the picture, allowing the authors to study the ability of Transformers to learn how to learn in regression settings. However, even in this simple setting, the mechanics of the architecture that enable such capability are still not well understood.

Following research has presented constructive methods to explain what type of algorithms these models can implement internally, by designing model weights that lead to a model that implements specific meta-algorithms. Akyürek et al. [2022] constructed Transformers that implement one step of gradient descent for the task of linear regression with $O(1)$ layers. Other works have focused on the linear attention (removing the softmax) and have shown empirically [von Oswald et al., 2022] and theoretically [Ahn et al., 2023, Mahankali et al., 2023] that the optimum for one layer, is in essence one step of preconditioned gradient descent. Ahn et al. [2023] also showed that the global minimizer in the two-layer case corresponds to

Proceedings of the 28th International Conference on Artificial Intelligence and Statistics (AISTATS) 2025, Mai Khao, Thailand. PMLR: Volume 258.

gradient descent with adaptive step size, but the optimality is restricted to only a class of sparse weights configurations. Beyond these, it still remains open how to characterize the global minimizer of the in-context loss for Transformers with multiple layers.

Another approach is to approximate the closed-form solution of linear regression, instead of minimizing the loss. For that purpose, one needs to be able to perform matrix inversion. There are multiple approaches to matrix inversion and in terms of iterative algorithms, one of the most popular ones is the iteration presented by Schulz [1933] (also known as Newton's iteration), which is a second-order method. Specifically, the method has a warm-up state with logarithmic, to the condition number of the matrix, steps and afterwards quadratic rate of convergence to arbitrary accuracy. The work of Giannou et al. [2023] showed that Transformers can implement second-order methods, like Newton's iteration [1] for matrix inversion. Fu et al. [2023] implemented Newton's iteration with Transformers for in-context linear regression; the authors also performed an empirical study to argue that Newton's iteration is closer to the trained models output rather than gradient descent.

Newton's iteration for matrix inversion is of the form $\mathbf{X}_{t+1} = \mathbf{X}_t(2\mathbb{I} - \mathbf{A}\mathbf{X}_t)$, where $\mathbf{A}$ is the matrix we want to invert, and $\mathbf{X}_t$ is the approximation of the inverse. The implementation of matrix inversion, serves as a stepping stone towards answering the following question:

"*How well can Transformers implement higher-order optimization methods?*"

In pursuit of a concrete answer, we focus on the well-known Newton's method, a second-order optimization algorithm. For minimizing a function f, the method can be described as $\mathbf{x}_{t+1} = \mathbf{x}_t - \eta(\mathbf{x}_t)(\nabla^2 f(\mathbf{x}_t))^{-1}\nabla f(\mathbf{x}_t)$ for some choice of step-size $\eta(\cdot)$. To implement such updates, one essentially needs to first compute the step size $\eta(\mathbf{x}_t)$, then invert the Hessian $\nabla^2 f(\mathbf{x}_t)$, and finally multiply them together with the gradient $\nabla f(\mathbf{x}_t)$. In general, the step size $\eta(\mathbf{x}_t)$ may also depend on quantities computed from $\nabla f(\mathbf{x}_t)$ and $\nabla^2 f(\mathbf{x}_t)$. It is relatively straightforward for Transformers to perform operations like matrix transposition and multiplication, while the challenge is to devise an organic combination of all the above components to deliver an efficient implementation of Newton's method with convergence guarantees. In particular, it further requires rigorous convergence analysis to verify the effectiveness of the construction in concrete examples.

[1] We use Newton's iteration for the matrix inversion algorithm and Newton's method for the optimization algorithm.

Main contributions. In this work we tackle the challenge from the perspective of in-context learning for the task of logistic regression. We consider Transformers with linear attention and position-wise feed-forward layers with the ReLU activation. We provide concrete constructions of Transformer to solve the task, and derive explicit upper bounds on the depth and width of the model with respect to the targeted error threshold. At a high level, our main findings are summarized in the following informal theorem.

Theorem 1.1 (Informal)**.** *Transformers can efficiently perform matrix inversion via Newton's iteration, based on which they can further 1) compute the least-square solution for linear regression, and 2) perform Newton's method to efficiently optimize the regularized logistic loss for logistic regression. In particular, in the latter case, only* $\log\log(1/\epsilon)$ *many layers and* $1/\epsilon^8$ *width are required for the Transformer to implement Newton's method on the regularized logistic loss to achieve* ϵ *error.*

We note here that 1) has been already been shown in Fu et al. [2023], but we present it and prove it in the Appendix for completeness. We also corroborate our results with experimental evidence. Interestingly, trained Transformers seem to outperform Newton's method for the initial layers/steps. To understand what the models are actually learning we also train models with different number of layers for the simpler task of linear regression. We compare the Transformers with variations of Newton's iteration with even higher orders of convergence.

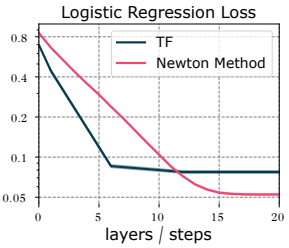

Figure 1: The logistic regression loss of a trained Transformer with 1/6/12/20 layers and corresponding steps of Newton's method.

2 RELATED WORK

It has been observed that Transformer-based models have the ability of performing in-context learning, as well as the ability of algorithmic reasoning [Brown et al., 2020, Nye et al., 2021, Wei et al., 2022a,b,

Dasgupta et al., 2022, Zhou et al., 2022]. Recently, Garg et al. [2022] initiated the mathematical formulation of the in-context learning problem, studying empirically the controlled setting of linear regression. Transformers were able to perform in-context linear regression, given only $(\mathbf{x}_i, y_i)$ pairs, generated as $y_i = \mathbf{w}_*^\top \mathbf{x}_i$, which were not seen during training. Later on, other works studied this setting both empirically and theoretically [Akyürek et al., 2022, von Oswald et al., 2022, Bai et al., 2023, Li et al., 2023, Guo et al., 2023, Chen et al., 2024].

Towards explaining this capability, Akyürek et al. [2022], von Oswald et al. [2023a] showed by construction that Transformers can emulate gradient descent for the task of linear regression. von Oswald et al. [2023a], also observed that empirically, one layer of linear attention Transformer had very similar performance with one step of gradient descent. Indeed, Ahn et al. [2023], Mahankali et al. [2023] proved that the global minimizer of the in-context learning loss for linear regression corresponds to one step of (preconditioned) gradient descent.

Related to our work is also the work of Bai et al. [2023], which is the only work - to the best of our knowledge - that provides a construction of gradient based algorithms for various in-context learning tasks, including logistic regression; they also demonstrate the ability of Transformer based models to perform algorithm selection. In the case of learning non-linear functions, Cheng et al. [2023] showed that Transformers can perform functional gradient descent.

Focusing on performing linear regression through matrix inversion, the recent work of von Oswald et al. [2023b] is of interest. The authors approximate the inverse of a matrix using Neumman series. For the task of linear regression this approach requires less memory compared to Newton's iteration, but it has a linear rate of convergence [Wu et al., 2014].

Considering higher order optimization methods, Giannou et al. [2023] first implemented matrix inversion using Newton's iteration, their construction though is sub-optimal, since it uses thirteen layers to perform just one step of the method and it is given in a general template. In a recent work by Fu et al. [2023], the authors use Newton's iteration for matrix inversion to approximate the closed form solution of linear regression; they compare it with a 12-layer trained transformer, by linear-probing each layer's output and comparing it with steps of the iterative algorithm. They furthermore conclude that Transformers are closer to Newton's iteration rather than gradient descent.

In terms of the optimization dynamics, Zhang et al. [2023] proved for one-layer linear attention the convergence of gradient flow to the global minimizer of the population loss given suitable initialization. Huang et al. [2023] showed the convergence of gradient descent for training softmax-attention Transformer under certain orthogonality assumption on the data features.

3 PRELIMINARIES

Notation. We use lowercase bold letters for vectors e.g., $\mathbf{x}, \mathbf{y}$, and by convention we consider them to be column vectors; for matrices we use uppercase bold letters e.g., $\mathbf{A}, \mathbf{B}$. We use $\lambda(\mathbf{A}), \sigma(\mathbf{A})$ to denote the eigenvalues and singular values of a matrix $\mathbf{A}$ respectively; we use $\kappa(\mathbf{A}) = \sigma_{\max}(\mathbf{A})/\sigma_{\min}(\mathbf{A})$ to denote the condition number of $\mathbf{A}$. For a Positive Symmetric Definite (PSD) matrix $\mathbf{A}$, we denote $\|\mathbf{x}\|_{\mathbf{A}} = \sqrt{\mathbf{x}^\top \mathbf{A}\mathbf{x}}$. We use $_d$ and $\mathbb{I}_d$ to denote a $d \times d$ matrix of zeros and the $d \times d$ identity matrix, respectively. If not specified otherwise, we use $*$ to denote inconsequential values.

3.1 The Transformer architecture

There are multiple variations of the Transformer architecture, depending on which type of attention (e.g., softmax, ReLU, and linear) is used. In this work, we focus on Transformers with linear self-attention described as follows: For each layer, let $\mathbf{H} \in \mathbb{R}^{d \times n}$ be the input, where each column is a d-dimensional embedding vector for each token. Let H be the number of attention heads, and for each head $i \in [H]$, we denote by $\mathbf{W}_K^{(i)}, \mathbf{W}_Q^{(i)}, \mathbf{W}_V^{(i)} \in \mathbb{R}^{d \times d}$ the key, query, and value weight matrices, respectively. Further let $\mathbf{W}_1 \in \mathbb{R}^{D \times d}$ and $\mathbf{W}_2 \in \mathbb{R}^{d \times D}$ be the weights of the feed-forward network, then the output of this layer is given by computing consecutively,

$$\mathsf{Att}(\mathbf{H}) = \mathbf{H} + \sum_{i=1}^{H} \mathbf{W}_V^{(i)} \mathbf{H} (\mathbf{W}_K^{(i)} \mathbf{H})^\top (\mathbf{W}_Q^{(i)} \mathbf{H}), \quad (3.1a)$$

$$\mathsf{TF}(\mathbf{H}) = \mathsf{Att}(\mathbf{H}) + \mathbf{W}_2 \sigma(\mathbf{W}_1 \mathsf{Att}(\mathbf{H})). \quad (3.1b)$$

Here $\sigma(\cdot)$ denotes the ReLU activation. Consistent with previous literature, the first equation (3.1a) represents the attention layer, combining which with the feed-forward layer in (3.1b) yields a single Transformer layer. We note here that the only difference with standard Transformer architecture is the elimination of the softmax operation and attention mask in the attention layer.

A Transformer model can contain multiple Transformer layers defined as above, and the output of the whole model would be the composition of multiple layers. From now on, we refer to Transformers with linear attention layers as *linear Transformers*.

3.2 In-context learning using Transformers

In this work we mostly consider the task of logistic regression. Our target would be to use the Transformer architecture to emulate in-context Newton's method for solving this task. To do so we also implement and use Newton's method as part of our construction.

For that reason we also present the task of linear regression, which can be performed once Newton's iteration has been implemented. We will compare the performance of Transformers with even higher order methods in Section 5.

3.2.1 Linear Regression

For the task of linear regression, let the pairs $\{(\mathbf{a}_i, y_i)\}_{i=1}^n$ be given as input to the Transformer, where $\mathbf{a}_i \in \mathbb{R}^d$ and $y_i \in \mathbb{R}$ for all $i = 1, \ldots, n$. We assume that for each such sequence of pairs, there is a weight vector $\mathbf{w}_*$, such that $y_i = \mathbf{w}_*^\top \mathbf{a}_i + \epsilon_i$ for all $i = 1, \ldots, n$, where ϵ_i is some noise. Given these samples, we want the Transformer to approximate the weight vector $\mathbf{w}_*$ or make a new prediction on a test point $\mathbf{a}_{test}$.

Define $\mathbf{y} = (y_1, \ldots, y_n)^\top \in \mathbb{R}^n$ and $\mathbf{A} = [\mathbf{a}_1, \ldots, \mathbf{a}_n]^\top \in \mathbb{R}^{n \times d}$. The standard least-square solution is given by

$$\hat{\mathbf{w}} = (\mathbf{A}^\top \mathbf{A})^{-1} \mathbf{A}^\top \mathbf{y}. \tag{3.2}$$

As a minimizer of the square loss $\ell(\mathbf{w}) = \sum_{i=1}^n (y_i - \mathbf{w}^\top \mathbf{a}_i)^2$, $\hat{\mathbf{w}}$ can also be obtained by minimizing $\ell(\mathbf{w})$ using, e.g., gradient descent. The number of steps required for convergence up to ϵ error is of order $\mathcal{O}(\kappa(\mathbf{A}^\top \mathbf{A}) \log(1/\epsilon))$.

Another approach is to compute directly the closed form solution (3.2), which involves the matrix inversion. One choice is Newton's iteration, an iterative method that approximates the inverse of a matrix with logarithmic dependence on the condition number and quadratic convergence to the accuracy improving upon gradient descent.

Newton's iteration for matrix inversion. The iteration can be described as follows: Suppose we want to invert a matrix $\mathbf{A}$, then with initialization $\mathbf{X}_0 = \alpha \mathbf{A}^\top$, we compute

$$\mathbf{X}_{t+1} = \mathbf{X}_t (2\mathbb{I} - \mathbf{A}\mathbf{X}_t) \tag{3.3}$$

For $\alpha \in (0, \frac{2}{\lambda_{\max}(\mathbf{A}^\top \mathbf{A})})$, it can be shown that the estimate is ϵ-accurate after $O(\log_2 \kappa(\mathbf{A}) + \log_2 \log_2(1/\epsilon))$ steps [Ogden, 1969, Pan and Schreiber, 1991].

One interesting generalization of the well-known Newton's iteration for matrix inversion is the following family of algorithms [Li and Li, 2010]: Initialized at $\mathbf{X}_0 = \alpha \mathbf{A}^\top$,

$$\mathbf{X}_{t+1} = \mathbf{X}_t \sum_{m=0}^{n-1} (-1)^m \binom{n}{m+1} (\mathbf{A}\mathbf{X}_t)^m. \tag{3.4}$$

We can see that for $n = 2$ we get the standard Newton's iteration described in Equation (3.3). For any fixed $n \geq 2$, the corresponding algorithm has an n-th order convergence to the inverse matrix, i.e., $(\mathbb{I} - \mathbf{X}_{k+1}\mathbf{A}) = (\mathbb{I} - \mathbf{X}_k \mathbf{A})^n$, suggesting that the error decays exponentially fast in an order of n. This results in the improvement of the convergence rate by changing the logarithm basis from $\log_2$ to $\log_n$. More importantly, the initial overhead of constant steps Newton's iteration is decreased from $\log_2 \kappa(\mathbf{A})$ to $\log_n \kappa(\mathbf{A})$. This would become clear in Section 5, where we compare these methods against the Transformer architecture.

3.2.2 Logistic Regression

For in-context learning of logistic regression, we consider pairs of examples $\{(\mathbf{a}_i, y_i)\}_{i=1}^n$ where each $\mathbf{a}_i \in \mathbb{R}^d$ is the covariate vector and $y_i \in \{-1, 1\}$ is the corresponding label. We assume that $y_i = \text{sign}(\mathbf{a}_i^\top \mathbf{w}_*)$ for some vector $\mathbf{w}_* \in \mathbb{R}^d$. Our target is to find a vector $\hat{\mathbf{w}} = \text{argmin}_{\mathbf{w} \in \mathbb{R}^d} f(\mathbf{w})$ where $f : \mathbb{R}^d \to \mathbb{R}$ is the regularized logistic loss defined as

$$f(\mathbf{w}) := \frac{1}{n} \sum_{i=1}^n \log(1 + \exp(-y_i \mathbf{w}^\top \mathbf{a}_i)) + \frac{\mu}{2} \|\mathbf{w}\|_2^2 \tag{3.5}$$

As in the setting of linear regression, we can use the vector $\hat{\mathbf{w}}$ to make a new prediction on some point $\mathbf{a}_{test}$ by calculating $1/(1 + \exp(-\hat{\mathbf{w}}^\top \mathbf{a}_{test}))$.

The L_2 penalty term is needed to ensure that the loss function is self-concordant in the following sense.

Definition 3.1. [Self-concordant function; Definition 5.1.1, Nesterov et al. 2018] Let $f : \mathbb{R}^d \to \mathbb{R}$ be a closed convex function that is 3 times continuously differentiable on its domain $\text{dom}(f) := \{\mathbf{x} \in \mathbb{R}^d \mid f(\mathbf{x}) < \infty\}$. For any fixed $\mathbf{x}, \mathbf{u} \in \mathbb{R}^d$ and $t \in \mathbb{R}$, define $\phi(t; \mathbf{x}, \mathbf{u}) := f(\mathbf{x} + t\mathbf{u})$ as a function of t. Then we say f is *self-concordant* if there exists a constant M_f such that, for all $\mathbf{x} \in \text{dom}(f)$ and $\mathbf{u} \in \mathbb{R}^d$ with $\mathbf{x} + t\mathbf{u} \in \text{dom}(f)$ for all sufficiently small t,

$$|\phi'''(0; \mathbf{x}, \mathbf{u})| \leq 2M_f (\mathbf{u}^\top \nabla^2 f(\mathbf{x}) \mathbf{u})^{3/2}$$

We say f is *standard self-concordant* when $M_f = 1$.

In particular, the regularized logistic loss is a self-concordant function under a mild assumption on the data.

Assumption 3.2. For the data $\{(\mathbf{a}_i, y_i)\}_{i=1}^n$ in (3.5), it holds that $\|\mathbf{a}_i\|_2 \leq 1$ for all $i = 1, \ldots, n$.

Proposition 3.3 (Lemma 2, Zhang and Xiao 2015)**.** *For $\mu > 0$, the regularized logistic loss $f(\cdot)$ defined in (3.5) is self-concordant with $M_f = 1/\sqrt{\mu}$ under assumption 3.2. Furthermore, the function $f/4\mu$ is standard self-concordant.*

Self-concordance ensures rapid convergence of second-order optimization algorithms such as Newton's method. As in the case of matrix inversion, the rate of convergence is quadratic after a constant number of steps that depend on how close the algorithm was initialized to the minimum.

Newton's method. Given the initialization $\mathbf{x}_0 \in \mathbb{R}^d$, Newton's method updates as follows:

$$\mathbf{x}_{t+1} = \mathbf{x}_t - \eta(\mathbf{x}_t)[\nabla^2 f(\mathbf{x}_t)]^{-1}\nabla f(\mathbf{x}_t). \tag{3.6}$$

Different choices of the step-size $\eta(\mathbf{x}_t)$ lead to different variants of the algorithm: For $\eta = 1$ we have the "classic" Newton's method; for $\eta(\mathbf{x}_t) = \frac{1}{1+\lambda(\mathbf{x}_t)}$ with $\lambda(\mathbf{x}_t) = \sqrt{\nabla f(\mathbf{x}_t)^\top [\nabla^2 f(\mathbf{x}_t)]^{-1}\nabla f(\mathbf{x}_t)}$, we have the so-called damped Newton's method (see, e.g., Section 4 in Nesterov et al. [2018]).

For self-concordant functions, the damped Newton's method has guarantees for global convergence, which contains two phases: In the initial phase, the quantity $\lambda(\mathbf{x})$ is decreased until it drops below the threshold of $1/4$. While $\lambda(\mathbf{x}) \geq 1/4$, there is a constant decrease per step of the self-concordant function g of at least 0.02. Then, the second phase begins once $\lambda(\mathbf{x})$ drops below the $1/4$ threshold, afterwards it will decay with a quadratic rate, implying a quadratic convergence of the objective value. All together, this implies that $c + \log\log(1/\epsilon)$ steps are required to reach ϵ accuracy. For the analysis of the exact Newton's method for self-concordant functions one may refer to Nesterov et al. [2018].

4 MAIN RESULTS

In this section we will present constructive arguments showing that Transformers can approximately implement Newton's method for logistic regression. To the best of our knowledge, the only result prior to our work for logistic regression is that of Bai et al. [2023] in which they implement gradient descent on the logistic loss using transformers. Newton's method can achieve a quadratic rate of convergence instead of linear with respect to the achieved accuracy, providing a tighter upper bound on the expressive capacity of these models. To implement this step we also provide in the Appendix a construction for Newton's iteration for matrix inversion, which directly implies a construction for in-context linear regression. We omit these results since similar results have been obtained in Fu et al. [2023].

As presented in Section 3.2.2, we seek to minimize the regularized logistic loss defined in (3.5), which is self-concordant by Proposition 3.3. In a nutshell, our main result in this case shows that linear Transformer can efficiently emulate Newton's method to minimize the loss. This is summarized in the following theorem. We remind that f is the regularized logistic loss of Equation (3.5).

Theorem 4.1. *For any dimension d, consider the regularized logistic loss defined in (3.5) with regularization parameter $\mu > 0$, and define $\kappa_f = \max_{\mathbf{x}\in\mathbb{R}^d} \kappa(\nabla^2 f(\mathbf{x}))$. Then for any $T > 0$ and $\epsilon > 0$, there exists a linear Transformer that can approximate T iterations of Newton's method on the regularized logistic loss up to error ϵ per iteration. In particular, the width of such a linear Transformer can be bounded by $O(d(1+\mu)^6/\epsilon^4\mu^8)$, and its depth can be bounded by $T(11+2k)$, where $k \leq 2\log\kappa_f + \log\log\frac{(1+\mu)^3}{\epsilon^2\mu^2}$. Furthermore, there is a constant $c > 0$ depending on μ such that if $\epsilon < c$, then the output of the Transformer provides a $\tilde{\mathbf{w}}$ satisfying that $\|\tilde{\mathbf{w}} - \hat{\mathbf{w}}\|_2 \leq O(\sqrt{\epsilon(1+\mu)/(4\mu)})$, where $\hat{\mathbf{w}}$ is the global minimizer of the loss.*

The proof of Theorem 4.1 contains two main components: The approximate implementation of Newton's method by Transformer and the convergence analysis of the resulting inexact Newton's method. Below we will address these two parts separately in Section 4.1 and Section 4.2.

4.1 Transformers can implement Newton's method for logisitic regression

To analyze the convergence properties of Newton's method on f, we actually implement the updates on $g = f/4\mu$, which based on Proposition 3.3 is standard self-concordant. Specifically, the targeted update is

$$\begin{aligned}\mathbf{x}_{t+1} &= \mathbf{x}_t - \frac{1}{1+\lambda_g(\mathbf{x}_t)}(\nabla^2 g(\mathbf{x}_t))^{-1}\nabla g(\mathbf{x}_t)\\ &= \mathbf{x}_t - \frac{2\sqrt{\mu}}{2\sqrt{\mu}+\lambda_f(\mathbf{x}_t)}(\nabla^2 f(\mathbf{x}_t))^{-1}\nabla f(\mathbf{x}_t)\end{aligned}$$

where the second equality follows directly from the definition $g = f/4\mu$. Our first result is that a linear Transformer can approximately implement the update above.

Theorem 4.2. *Under the setting of Theorem 4.1, there exists a Transformer consisting of linear attention with ReLU layers that can approximately perform damped Newton's method on the regularized logistic loss as follows*

$$\mathbf{x}_{t+1} = \mathbf{x}_t - \frac{2\sqrt{\mu}}{2\sqrt{\mu} + \lambda(\mathbf{x}_t)}(\nabla^2 f(\mathbf{x}_t))^{-1}\nabla f(\mathbf{x}_t) + \varepsilon$$

where ε is an error term. For any $\epsilon > 0$, to achieve that $\|\varepsilon\|_2 \le \epsilon$, the width of such a Transformer can be bounded by $O(d\frac{(1+\mu)^8}{\epsilon^4\mu^{10}})$, and its depth can be bounded by $11 + 2k$ where $k \le 2\log\kappa_f + \log\log\frac{(1+\mu)^3}{\epsilon^2\mu^2}$.

Remark 4.3. Here we omit the details of input and output format for ease of presentation. See Section 3 in the Appendix for details. Roughly speaking, the input to the Transformer is of dimension $(6d+4)\times n$ and contains the matrix $\mathbf{A}$, the labels $\mathbf{y}$, and the initialization $\mathbf{x}_0$.

Below we provide the main idea of the proof by describing main steps of our construction.

Proof sketch of Theorem 4.2. To construct a Transformer that implements Newton's method to optimize the regularized logistic loss, we implement first each one of the required components, including gradient $\nabla f(\mathbf{x})$, Hessian $\nabla^2 f(\mathbf{x})$, and the damping parameter $\lambda(\mathbf{x})$. The gradient and Hessian of f are

$$\nabla f(\mathbf{x}) = -\frac{1}{n}\sum_{i=1}^{n} y_i p_i \mathbf{a}_i + \mu\mathbf{x},$$
$$\nabla^2 f(\mathbf{x}) = \frac{1}{n}\mathbf{A}^\top\mathbf{D}\mathbf{A} + \mu\mathbb{I}_d \succeq \mu\mathbb{I}_d,$$

where each $p_i := \exp(-y_i\mathbf{x}^\top\mathbf{a}_i)/(1+\exp(-y_i\mathbf{x}^\top\mathbf{a}_i))$ and $\mathbf{D} := \mathrm{diag}(p_1(1-p_1),\dots,p_n(1-p_n))$.

Step 1: Approximate gradient and Hessian. We use the linear attention layer to perform matrix multiplications, and we use the ReLU network to approximate the following quantities:

- The values p_i.
- The diagonal elements of the matrix $\mathbf{D}$.
- The "hidden" dot product of the diagonal elements of $\mathbf{D}$ and the matrix $\mathbf{A}$, since the i-th row of $\mathbf{DA}$ is $d_i\mathbf{a}_i^\top$.

These approximations give rise to the approximated gradient and Hessian of f, which we denote by $\hat{\nabla} f(\mathbf{x}), \hat{\nabla}^2 f(\mathbf{x})$.

Step 2: Invert the Hessian. For the next step, we use the construction presented in Appendix (Theorem A.2) , which implements the inversion of the matrix $\hat{\nabla}^2 f(\mathbf{x})$. We leverage classical results in matrix perturbation theory to show that $(\hat{\nabla}^2 f(\mathbf{x}))^{-1} = (\nabla^2 f(\mathbf{x}))^{-1} + \mathbf{E}_{1,t}$, where $\mathbf{E}_{1,t}$ is an error matrix whose norm can be bounded based on the number of iterations performed for the inversion.

Step 3: Approximate the step size. Next, we need to approximate the step size $\frac{2\sqrt{\mu}}{2\sqrt{\mu}+\lambda(\mathbf{x})}$. This is done using the ReLU layers. Recall that $\lambda(\mathbf{x})^2 = \nabla f(\mathbf{x})^\top(\nabla^2 f(\mathbf{x}))^{-1}\nabla f(\mathbf{x})$, which can be approximated using $\hat{\nabla} f(\mathbf{x}), \nabla^2 f(\mathbf{x})$ from the previous steps. To get the step size, we need to further approximate the function $g(z) = 1/(1+\sqrt{z})$ and evaluate it at $\lambda(\mathbf{x})^2$. Thus, any error in the approximation of $\lambda(\mathbf{x})^2$ translates to a square root error in the calculation of $g(\lambda(\mathbf{x})^2)$. To see this, consider the derivative of the function g, $g'(z) = -2\sqrt{z}/(1+\sqrt{z})$, and observe that for z of constant order, if z changes by ϵ, the corresponding change in $g(z)$ would be of order $\sqrt{\epsilon}$. This leads to the quadratic requirement of width with respect to the desired error threshold.

Step 4: Aggregate all approximations. Finally, we aggregate all the error terms induced by each approximation step and get the desired approximation to one step of Newton's method. ■

4.2 Convergence of inexact Newton's method

Theorem 4.2 shows that linear Transformers can approximately implement damped Newton's method for logistic regression, while providing width requirements in order to control the approximation error. In complement to this, we further provide convergence analysis for the resulting algorithm.

Theorem 4.4. *Under Assumption 3.2, for the regularized logistic loss f defined in (3.5) with regularization parameter $\mu > 0$, consider a sequence of iterates $\{\mathbf{x}_t\}_{t\ge 0}$ satisfying*

$$\mathbf{x}_{t+1} = \mathbf{x}_t - \frac{2\sqrt{\mu}}{2\sqrt{\mu} + \lambda(\mathbf{x}_t)}\nabla^2 f(\mathbf{x}_t)^{-1}\nabla f(\mathbf{x}_t) + \varepsilon_t$$

where $\|\varepsilon_t\|_2 \le \epsilon$ for all $t \ge 0$. Then there exists constants c, C_1, C_2 depending only on μ such that for any $\epsilon \le c$, it holds that $g(\mathbf{x}_t) - g(\mathbf{x}^*) \le \epsilon$ for all $t \ge C_1 + C_2\log\log\frac{1}{\epsilon}$.

The proof of Theorem 4.4 follows the proof presented in Nesterov et al. [2018] but accounts for the error

term. Similar analysis has been performed in the past in Sun et al. [2020]. Our proof consists of the following steps:

- **Feasibility**: We first show by induction that there exists a constant depending on μ such that $\|\mathbf{x}_t\|_2 \leq C$ for all $t \geq 0$. This is a consequence of the bounded norm assumption in Assumption 3.2 and the strongly convex L_2 regularizer.

- **Constant decrease**: We then show constant decreasing of the loss value per step when $\lambda(\mathbf{x}_t) \geq 1/6$. This is achieved by upper bounding the difference $g(\mathbf{x}_{t+1}) - g(\mathbf{x}_t)$ with a function of $\lambda(\mathbf{x}_t)$, which we prove that achieves a maximum of -0.01 for $\lambda(\mathbf{x}) \geq 1/6$.

- **Quadratic convergence**: In the last step, we show that for $\lambda(\mathbf{x}_t) < 1/6$ it holds $g(\mathbf{x}_t) - g(\mathbf{x}^*) \leq \frac{3}{5}\lambda(\mathbf{x}_t)$; Furthermore, when $\lambda(\mathbf{x}_t) < 1/6$, the inequality $\lambda(\mathbf{x}_{t+1}) \leq c\lambda(\mathbf{x}_t)^2 + \epsilon'$ holds, which implies quadratic convergence to achieving $O(\epsilon)$ error.

See section 2 in the Appendix for the complete proof of Theorem 4.4. Finally, combining the construction in Theorem 4.2 and the convergence analysis in Theorem 4.4 yields the performance guarantee of the constructed Transformer.

5 EXPERIMENTS

In this section, we corroborate our theoretical findings with empirical evidence. Specifically, we aim to empirically demonstrate the effectiveness of encoder-based linear self-attention when trained to solve logistic regression as well as linear regression. Our code is available here[2].

Logistic regression. We analyze the Transformer's ability to solve the task of logistic regression. To simplify the setting, we focus on training the Transformer to predict the logistic regression parameter $\mathbf{w}$ learned by Newton's method upon convergence. The reason is that predicting the true underlying weight $\mathbf{w}^*$ directly is a hard task and is not necessarily the solution of the minimization problem described in Equation (3.5); we instead opt to train the Transformer to output a solution comparable to that computed by Newton's Method.

[2] https://github.com/Leiay/icl_newton

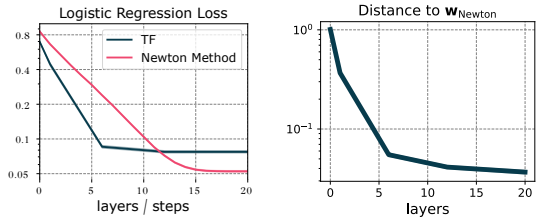

Figure 2: *Performance of Transformer on logistic regression tasks. (Left)* The logistic regression loss for the Transformer (TF) and the Newton Method, with a regularization $\mu = 0.01$. The Transformer is shown to approximate the method more effectively within a few layers. *(Right)* The l_2 norm of the difference between the Transformer's prediction and Newton's method solution, when the model is trained to predict the solution derived from Newton's Method.

We train on data dimension 5, and the input contains 26 in-context samples. The Transformer model utilizes a GPT-2 backbone [Radford et al., 2019] with the causal mask removed, and has embedding dimension 256, and 4 attention heads. We set the regularization parameter $\mu = 0.01$ in the regularized logistic loss in (3.5). In Figure 2, we compare the loss value of the trained Transformer across different number of layers and the loss value of different iterates of Newton's method in the left plot; we further plot the corresponding error between Transformer's prediction and Newton's method converged solution. According to our construction in Theorem 4.2, a single step of the Newton Method is equivalent to $11 + 2k$ layers of a standard Transformer, where k depends on the condition number of the Hessian. However, when trained, Transformers can approximate Newton's Method even more effectively as shown in Figure 2. To better understand the model's capabilities we also explore the performance in the linear regression task below.

Linear regression. For the task of linear regression, we train models consisting of linear self-attention (LSA) to learn in-context. We use an embedding dimension of 64 and 4 attention heads. The data dimension is 10, and the input contains 50 in-context samples.

We train models with LSA, having from 1 to 6 layers, employing the training technique of von Oswald et al. [2023a] to stabilize the training process. Consistent with the findings reported in von Oswald et al. [2023a], we observe that after four layers, there is no significant improvement in performance by adding more layers. We attribute this to limitations in the training process, and the optimal training method for linear Transformers beyond $5 - 6$ layers remains

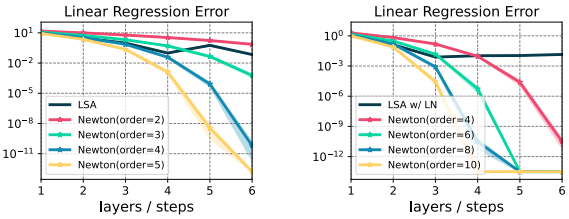

Figure 3: *Linear Regression Error of LSA (left), LSA with LayerNorm (right) and higher-order variations of Newton's iteration.*

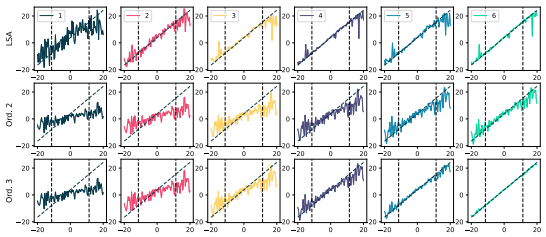

Figure 4: *Output of LSA and higher-order variations of Newton's iteration by keeping the test sample fixed and changing one of its coordinates, in-distribution. We plot* $1-6$ *layers against* $1-6$ *steps of second and third order Newton's iteration; we observe that the model lies between second and third order. The in-distribution range is indicated by the dashed line.*

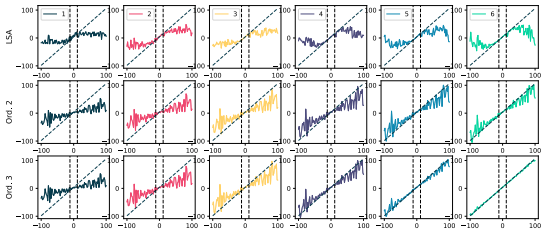

Figure 5: *Same setting as above, but the models/algorithms are tested with out-of-distribution values as well. We observe that the model does not actually learn the underlying linear function.*

unidentified. Since LSA models are challenging to train beyond 6 layers, we incorporate LayerNorm to stabilize the training process, and also assessing how LayerNorm influences their performance. These results are deferred in the Appendix.

In Figure 3, we plot the loss on new test samples, by keeping the same sampling method with the training process. We observe that models having 4 or more layers have almost the same performance in terms of the loss.

To get a clearer picture of what the models are learning, we further provide plots illustrating the actual outputs of the trained Transformers, as shown in Figure 4. Here we test the trained models in the following set-up: we first sample $N = 5000$ in-context examples, i.e., $\{\{(\mathbf{x}_i, y_i)\}_{i=1}^{n}\}_{j=1}^{N}$ that all share the same underlying weight vector $y_i = \mathbf{w}_*^\top \mathbf{x}_i$, $\forall i, j$. We then pick a specific $\mathbf{x}_{test}$ sample, which we keep the same for all the in-context examples. For this test sample, the value of its first coordinate varies across $[-a, a]$ for different values of a.

We test two different cases: 1) *in-distribution*: the value a has been encountered in the training set with probability at least 99% and 2) *out-of-distribution*: the value is an outlier. Based on our experiment setup, the in-distribution range is $[-14.9, 14.9]$, as indicated by the dashed line in Figure 4 and 5.

Looking at Figure 4, one may observe that the performance of the model lies between the second- and third-order Newton's iteration. Intuitively, Transformers do have slighter higher order capacity than Newton's iteration. To see that, assume that the model is given in the input a symmetric matrix $\mathbf{A}$, then in the first layer from Equation (3.1a) the model can create up to the third power of the matrix $\mathbf{A}$, i.e., all powers for $1-3$. Now in the second layer the higher-order term can be up to power of 9; in contrast the second-order Newton's iteration can reach up to order 7 (the first iteration $\mathbf{X}_1 \sim \mathbf{A}^3$, and in the second one $\mathbf{X}_2 \sim \mathbf{X}_1\mathbf{A}\mathbf{X}_1 \sim \mathbf{A}^7$.

The gap between the possible powers of the matrix, between second-order Newton's and Transformers will increase as the number of layers increases. On the other hand, third-order Newton's has up to power $\mathbf{A}^5$ in the fist step $\mathbf{X}_1$. While in second one the maximum power is $\mathbf{X}_1(\mathbf{A}\mathbf{X}_1)^2 \sim \mathbf{A}^{17}$. Thus, the LSA model will not be able to outperform it.

6 DISCUSSION

In this paper we provide an explicit construction of a Transformer model for in-context learning of logistic regression, that can perform Newton's method to optimize the regularized logistic loss. Moreover, we provide a detailed convergence analysis of the inexact Newton's method emulated by the constructed Transformer with explicit upper bounds on the number of heads, layers and width for a specified achieved accuracy. We further examine and compare the empirical performance of the trained Transformers with that of higher-than two-order methods.

Acknowledgements

JDL acknowledges support of Open Philanthropy, NSF IIS 2107304, NSF CCF 2212262, NSF CAREER Award 2144994, and NSF CCF 2019844. The work of Dimitris Papailiopoulos is supported in part by ONR Grant No. N00014-21-1-2806 and No. N00014-23-1-2848. We would like to thank the reviewers for their constructive feedback and suggestions, which helped improving this paper.

References

K. Ahn, X. Cheng, H. Daneshmand, and S. Sra. Transformers learn to implement preconditioned gradient descent for in-context learning, 2023.

E. Akyürek, D. Schuurmans, J. Andreas, T. Ma, and D. Zhou. What learning algorithm is in-context learning? investigations with linear models. *arXiv preprint arXiv:2211.15661*, 2022.

Y. Bai, F. Chen, H. Wang, C. Xiong, and S. Mei. Transformers as statisticians: Provable in-context learning with in-context algorithm selection, 2023.

S. Black, S. Biderman, E. Hallahan, Q. Anthony, L. Gao, L. Golding, H. He, C. Leahy, K. McDonell, J. Phang, et al. Gpt-neox-20b: An open-source autoregressive language model. *arXiv preprint arXiv:2204.06745*, 2022.

S. P. Boyd and L. Vandenberghe. *Convex optimization*. Cambridge university press, 2004.

T. Brown, B. Mann, N. Ryder, M. Subbiah, J. D. Kaplan, P. Dhariwal, A. Neelakantan, P. Shyam, G. Sastry, A. Askell, et al. Language models are few-shot learners. *Advances in neural information processing systems*, 33:1877–1901, 2020.

S. Chen, H. Sheen, T. Wang, and Z. Yang. Training dynamics of multi-head softmax attention for in-context learning: Emergence, convergence, and optimality. *arXiv preprint arXiv:2402.19442*, 2024.

X. Cheng, Y. Chen, and S. Sra. Transformers implement functional gradient descent to learn non-linear functions in context, 2023.

I. Dasgupta, A. K. Lampinen, S. C. Chan, A. Creswell, D. Kumaran, J. L. McClelland, and F. Hill. Language models show human-like content effects on reasoning. *arXiv preprint arXiv:2207.07051*, 2022.

A. Dosovitskiy, L. Beyer, A. Kolesnikov, D. Weissenborn, X. Zhai, T. Unterthiner, M. Dehghani, M. Minderer, G. Heigold, S. Gelly, et al. An image is worth 16x16 words: Transformers for image recognition at scale. In *International Conference on Learning Representations*, 2020.

D. Fu, T.-Q. Chen, R. Jia, and V. Sharan. Transformers learn higher-order optimization methods for in-context learning: A study with linear models, 2023.

S. Garg, D. Tsipras, P. S. Liang, and G. Valiant. What can transformers learn in-context? a case study of simple function classes. *Advances in Neural Information Processing Systems*, 35:30583–30598, 2022.

A. Giannou, S. Rajput, J.-y. Sohn, K. Lee, J. D. Lee, and D. Papailiopoulos. Looped transformers as programmable computers. *arXiv preprint arXiv:2301.13196*, 2023.

T. Guo, W. Hu, S. Mei, H. Wang, C. Xiong, S. Savarese, and Y. Bai. How do transformers learn in-context beyond simple functions? a case study on learning with representations. *arXiv preprint arXiv:2310.10616*, 2023.

Y. Huang, Y. Cheng, and Y. Liang. In-context convergence of transformers. *arXiv preprint arXiv:2310.05249*, 2023.

J. D. M.-W. C. Kenton and L. K. Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding. In *Proceedings of NAACL-HLT*, pages 4171–4186, 2019.

S. Khan, M. Naseer, M. Hayat, S. W. Zamir, F. S. Khan, and M. Shah. Transformers in vision: A survey. *ACM computing surveys (CSUR)*, 54(10s): 1–41, 2022.

W. Li and Z. Li. A family of iterative methods for computing the approximate inverse of a square matrix and inner inverse of a non-square matrix. *Applied Mathematics and Computation*, 215(9): 3433–3442, 2010. ISSN 0096-3003.

Y. Li, M. E. Ildiz, D. Papailiopoulos, and S. Oymak. Transformers as algorithms: Generalization and stability in in-context learning. *International Conference on Machine Learning*, 2023.

O. Lieber, O. Sharir, B. Lenz, and Y. Shoham. Jurassic-1: Technical details and evaluation. *White Paper. AI21 Labs*, 1:9, 2021.

A. Mahankali, T. B. Hashimoto, and T. Ma. One step of gradient descent is provably the optimal in-context learner with one layer of linear self-attention. *arXiv preprint arXiv:2307.03576*, 2023.

Y. Nesterov et al. *Lectures on convex optimization*, volume 137. Springer, 2018.

M. Nye, A. J. Andreassen, G. Gur-Ari, H. Michalewski, J. Austin, D. Bieber, D. Dohan, A. Lewkowycz, M. Bosma, D. Luan, et al. Show your work: Scratchpads for intermediate computation with language models. *arXiv preprint arXiv:2112.00114*, 2021.

H. C. Ogden. Iterative methods of matrix inversion. 1969.

V. Pan and R. Schreiber. An improved newton iteration for the generalized inverse of a matrix, with applications. *SIAM Journal on Scientific and Statistical Computing*, 12(5):1109–1130, 1991.

A. Radford, J. Wu, R. Child, D. Luan, D. Amodei, I. Sutskever, et al. Language models are unsupervised multitask learners. *OpenAI blog*, 1(8):9, 2019.

G. Schulz. Iterative berechung der reziproken matrix. *ZAMM - Journal of Applied Mathematics and Mechanics / Zeitschrift für Angewandte Mathematik und Mechanik*, 13(1):57–59, 1933.

G. W. Stewart and J. guang Sun. Matrix perturbation theory. 1990.

T. Sun, I. Necoara, and Q. Tran-Dinh. Composite convex optimization with global and local inexact oracles. *Computational Optimization and Applications*, 76:69–124, 2020.

A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Ł. Kaiser, and I. Polosukhin. Attention is all you need. *Advances in neural information processing systems*, 30, 2017.

J. von Oswald, E. Niklasson, E. Randazzo, J. Sacramento, A. Mordvintsev, A. Zhmoginov, and M. Vladymyrov. Transformers learn in-context by gradient descent. *arXiv preprint arXiv:2212.07677*, 2022.

J. von Oswald, E. Niklasson, E. Randazzo, J. Sacramento, A. Mordvintsev, A. Zhmoginov, and M. Vladymyrov. Transformers learn in-context by gradient descent, 2023a.

J. von Oswald, E. Niklasson, M. Schlegel, S. Kobayashi, N. Zucchet, N. Scherrer, N. Miller, M. Sandler, B. A. y Arcas, M. Vladymyrov, R. Pascanu, and J. Sacramento. Uncovering mesa-optimization algorithms in transformers, 2023b.

J. Wei, Y. Tay, R. Bommasani, C. Raffel, B. Zoph, S. Borgeaud, D. Yogatama, M. Bosma, D. Zhou, D. Metzler, et al. Emergent abilities of large language models. *arXiv preprint arXiv:2206.07682*, 2022a.

J. Wei, X. Wang, D. Schuurmans, M. Bosma, E. Chi, Q. Le, and D. Zhou. Chain of thought prompting elicits reasoning in large language models. *arXiv preprint arXiv:2201.11903*, 2022b.

M. Wu, B. Yin, G. Wang, C. Dick, J. R. Cavallaro, and C. Studer. Large-scale mimo detection for 3gpp lte: Algorithms and fpga implementations. *IEEE Journal of Selected Topics in Signal Processing*, 8 (5):916–929, 2014.

L. Yuan, Y. Chen, T. Wang, W. Yu, Y. Shi, Z.-H. Jiang, F. E. Tay, J. Feng, and S. Yan. Tokens-to-token vit: Training vision transformers from scratch on imagenet. In *Proceedings of*

the IEEE/CVF International Conference on Computer Vision, pages 558–567, 2021.

R. Zhang, S. Frei, and P. L. Bartlett. Trained transformers learn linear models in-context. *arXiv preprint arXiv:2306.09927*, 2023.

Y. Zhang and L. Xiao. Communication-efficient distributed optimization of self-concordant empirical loss, 2015.

H. Zhou, A. Nova, H. Larochelle, A. Courville, B. Neyshabur, and H. Sedghi. Teaching algorithmic reasoning via in-context learning. *arXiv preprint arXiv:2211.09066*, 2022.

Checklist

1. For all models and algorithms presented, check if you include:
   (a) A clear description of the mathematical setting, assumptions, algorithm, and/or model. [Yes]
   (b) An analysis of the properties and complexity (time, space, sample size) of any algorithm. [Yes]
   (c) (Optional) Anonymized source code, with specification of all dependencies, including external libraries. [Yes]

2. For any theoretical claim, check if you include:
   (a) Statements of the full set of assumptions of all theoretical results. [Yes]
   (b) Complete proofs of all theoretical results. [Yes]
   (c) Clear explanations of any assumptions. [Yes]

3. For all figures and tables that present empirical results, check if you include:
   (a) The code, data, and instructions needed to reproduce the main experimental results (either in the supplemental material or as a URL). [Yes]
   (b) All the training details (e.g., data splits, hyperparameters, how they were chosen). [Yes]
   (c) A clear definition of the specific measure or statistics and error bars (e.g., with respect to the random seed after running experiments multiple times). [Yes]
   (d) A description of the computing infrastructure used. (e.g., type of GPUs, internal cluster, or cloud provider). [Yes]

4. If you are using existing assets (e.g., code, data, models) or curating/releasing new assets, check if you include:
   (a) Citations of the creator if your work uses existing assets. [Yes]
   (b) The license information of the assets, if applicable. [Yes]
   (c) New assets either in the supplemental material or as a URL, if applicable. [Not Applicable]
   (d) Information about consent from data providers/curators. [Not Applicable]
   (e) Discussion of sensible content if applicable, e.g., personally identifiable information or offensive content. [Not Applicable]

5. If you used crowdsourcing or conducted research with human subjects, check if you include:
   (a) The full text of instructions given to participants and screenshots. [Not Applicable]
   (b) Descriptions of potential participant risks, with links to Institutional Review Board (IRB) approvals if applicable. [Not Applicable]
   (c) The estimated hourly wage paid to participants and the total amount spent on participant compensation. [Not Applicable]

Organization of the Appendix

A Constructions of Transformers for linear regression

In this section we provide the exact constructions for implementing Newton's Iteration for matrix inversion. We then use this construction to approximate the closed form solution of linear regression in-context.

A.1 Newton-Raphson Method for Matrix Inverse

The Newton-Raphson method [Schulz, 1933] for inverting a matrix $\mathbf{A} \in \mathbb{R}^{d\times d}$ is defined by the following update rule:

$$\mathbf{X}_{t+1} = 2\mathbf{X}_t - \mathbf{X}_t\mathbf{A}\mathbf{X}_t. \tag{A.1}$$

An initialization for this method that guarantees convergence is $\mathbf{X}_0 = \epsilon\mathbf{A}^\top$, where $\epsilon \in (0, \frac{2}{\lambda_1(\mathbf{A}\mathbf{A}^\top)})$. We prove below that linear-attention transformers can emulate the above update.

Lemma A.1. *For any dimension d, there exists a linear Transformer consisting of 2 linear attention layers, each of which has 2 attention heads and width $4d$, such that it can perform one step of Newton's iteration for any target matrix $\mathbf{A} \in \mathbb{R}^{d\times d}$. Specifically, the Transformer realizes the following mapping from input to output for any $\mathbf{X}_0 \in \mathbb{R}^{d\times d}$:*

$$\begin{pmatrix} \mathbf{X}_0 \\ \mathbf{A}^\top \\ d \\ \mathbb{I}_d \end{pmatrix} \mapsto \begin{pmatrix} \mathbf{X}_1 \\ \mathbf{A}^\top \\ d \\ \mathbb{I}_d \end{pmatrix}$$

where $\mathbf{X}_1 = \mathbf{X}_0(2\mathbb{I}_d - \mathbf{A}\mathbf{X}_0)$, corresponding to one step of Newton's iteration in Equation (3.3). Furthermore, if restricted to only symmetric $\mathbf{A}$, then 1 layer suffices.

Proof. Assume that we are given the following input

$$\mathbf{H}_0 = \begin{pmatrix} \mathbf{X}_0 \\ \mathbf{A} \\ \\ \mathbb{I}_d \end{pmatrix} \in \mathbb{R}^{4d\times d}.$$

For the first layer, we choose

$$\mathbf{W}_V = \begin{pmatrix} d & d & d & d \\ d & d & d & d \\ d & \mathbb{I}_d & d & d \\ d & d & d & d \end{pmatrix}, \mathbf{W}_K = \begin{pmatrix} d & d & d & \mathbb{I}_d \\ d & d & d & d \\ d & d & d & d \\ d & d & d & d \end{pmatrix}, \mathbf{W}_Q = \begin{pmatrix} \mathbb{I}_d & d & d & d \\ d & d & d & d \\ d & d & d & d \\ d & d & d & d \end{pmatrix}$$

so that

$$\mathbf{W}_V\mathbf{H}_0 = \begin{pmatrix} d \\ d \\ \mathbf{A} \\ d \end{pmatrix}, \quad \mathbf{W}_K\mathbf{H}_0 = \begin{pmatrix} \mathbb{I}_d \\ d \\ d \\ d \end{pmatrix}, \quad \mathbf{W}_Q\mathbf{H}_0 = \begin{pmatrix} \mathbf{X}_0 \\ d \\ d \\ d \end{pmatrix}.$$

Then the output of the first layer is

$$\mathbf{H}_{1/2} = \mathbf{H}_0 + \mathbf{W}_V\mathbf{H}_0(\mathbf{W}_K\mathbf{H}_0)^\top\mathbf{W}_Q\mathbf{H}_0 = \begin{pmatrix} \mathbf{X}_0 \\ \mathbf{A} \\ \mathbf{A}\mathbf{X}_0 \\ \mathbb{I}_d \end{pmatrix}.$$

Next, we use two attention heads for the second layers. For the first head, we choose

$$\mathbf{W}_V^{(1)} = \begin{pmatrix} \mathbb{I}_d & d & d & d \\ d & d & d & d \\ d & d & d & d \\ d & d & d & d \end{pmatrix}, \mathbf{W}_K^{(1)} = \begin{pmatrix} d & d & d & \mathbb{I}_d \\ d & d & d & d \\ d & d & d & d \\ d & d & d & d \end{pmatrix}, \mathbf{W}_Q^{(1)} = \begin{pmatrix} d & d & -\mathbb{I}_d & d \\ d & d & d & d \\ d & d & d & d \\ d & d & d & d \end{pmatrix}$$

so that

$$\mathbf{W}_V^{(1)}\mathbf{H}_{1/2} = \begin{pmatrix} \mathbf{X}_0 \\ d \\ d \\ d \end{pmatrix}, \quad \mathbf{W}_K^{(1)}\mathbf{H}_{1/2} = \begin{pmatrix} \mathbb{I}_d \\ d \\ d \\ d \end{pmatrix}, \quad \mathbf{W}_Q^{(1)}\mathbf{H}_{1/2} = \begin{pmatrix} -\mathbf{A}\mathbf{X}_0 \\ d \\ d \\ d \end{pmatrix}.$$

For the second head, we choose

$$\mathbf{W}_V^{(2)} = \begin{pmatrix} \mathbb{I}_d & d & d & d \\ d & d & d & d \\ d & d & -\mathbb{I}_d & d \\ d & d & d & d \end{pmatrix}, \mathbf{W}_K^{(2)} = \begin{pmatrix} d & d & d & \mathbb{I}_d \\ d & d & d & d \\ d & d & d & d \\ d & d & d & d \end{pmatrix}, \mathbf{W}_Q^{(2)} = \begin{pmatrix} d & d & d & \mathbb{I}_d \\ d & d & d & d \\ d & d & d & d \\ d & d & d & d \end{pmatrix}$$

so that

$$\mathbf{W}_V^{(2)}\mathbf{H}_{1/2} = \begin{pmatrix} \mathbf{X}_0 \\ d \\ -\mathbf{A}\mathbf{X}_0 \\ d \end{pmatrix}, \quad \mathbf{W}_K^{(2)}\mathbf{H}_{1/2} = \begin{pmatrix} \mathbb{I}_d \\ d \\ d \\ d \end{pmatrix}, \quad \mathbf{W}_Q^{(2)}\mathbf{H}_{1/2} = \begin{pmatrix} \mathbb{I}_d \\ d \\ d \\ d \end{pmatrix}.$$

Combining the outputs of the two heads, we get the output of the second layer as

$$\begin{aligned} \mathbf{H}_1 &= \mathbf{H}_{1/2} + \mathbf{W}_V^{(1)}\mathbf{H}_{1/2}(\mathbf{W}_K^{(1)}\mathbf{H}_{1/2})^\top \mathbf{W}_Q^{(1)}\mathbf{H}_{1/2} + \mathbf{W}_V^{(2)}\mathbf{H}_{1/2}(\mathbf{W}_K^{(2)}\mathbf{H}_{1/2})^\top \mathbf{W}_Q^{(2)}\mathbf{H}_{1/2} \\ &= \mathbf{H}_{1/2} + \begin{pmatrix} -\mathbf{X}_0\mathbf{A}\mathbf{X}_0 & & \\ & & \\ & & \end{pmatrix} + \begin{pmatrix} \mathbf{X}_0 & & \\ & & \\ -\mathbf{A}\mathbf{X}_0 & & \end{pmatrix} \\ &= \begin{pmatrix} \mathbf{X}_1 & & \\ \mathbf{A} & & \\ & & \\ \mathbb{I}_d & & \end{pmatrix}. \end{aligned}$$

This completes the proof. For the case where the matrix $\mathbf{A}$ is symmetric, we give the construction as part of the proof for linear regression. See the proof of Theorem A.2 in Appendix A.2. ■

A.2 Linear regression through matrix inversion.

Given the least-square solution, the prediction for a fresh test point $\mathbf{a}_{\text{test}}$ is

$$y_{\text{test}} = \mathbf{a}_{\text{test}}^\top(\mathbf{A}^\top\mathbf{A})^{-1}\mathbf{A}^\top\mathbf{y} = \mathbf{y}^\top\mathbf{A}(\mathbf{A}^\top\mathbf{A})^{-1}\mathbf{a}_{\text{test}}.$$

Below we give a construction of Transformer that can emulate the Newton-Raphson method to approximate the inverse of $\mathbf{A}^\top\mathbf{A}$ and then multiply the result with the necessary quantities to obtain an approximation of y_{test}. Notice that here the matrix we want to invert, $\mathbf{A}^\top\mathbf{A}$, is symmetric.

Theorem A.2 (Linear regression)**.** *For any dimension d, n and index $T > 0$, there exists a linear Transformer consisting of $3 + T$ layers, where each layer has 2 attention heads and width equal to $4d + 3$, such that it realizes the following mapping from input to output:*

$$\begin{pmatrix} [\mathbb{I}_d\ ] \\ [\mathbb{I}_d\ ] \\ [\mathbb{I}_d\ ] \\ \mathbf{A}^\top \\ \mathbf{a}_{\text{test}}^\top, \\ \mathbf{y}^\top \\ 0, 0, \ldots, 0 \end{pmatrix} \mapsto \begin{pmatrix} [\mathbb{I}_d\ ] \\ [\mathbb{I}_d\ ] \\ [\mathbb{I}_d\ ] \\ \mathbf{A}^\top \\ \mathbf{a}_{\text{test}}^\top, \\ \mathbf{y}^\top \\ \hat{y}, 0, \ldots, 0 \end{pmatrix}$$

where $\mathbf{A} = (\mathbf{a}_1, \ldots, \mathbf{a}_n)^\top$, $\mathbf{y} = (y_1, \ldots, y_n)^\top$, and $\hat{y} = \mathbf{a}_{\text{test}}^\top\mathbf{X}_T\mathbf{A}^\top\mathbf{y}$ is the prediction with $\mathbf{X}_T$ being the output of T steps of Newton's iteration for inversion on the matrix $\mathbf{A}^\top\mathbf{A}$, where the initialization is $\mathbf{X}_0 = \epsilon\mathbf{A}^\top\mathbf{A}$ for some $\epsilon \in (0, \frac{2}{\lambda_{\max}^2(\mathbf{A}^\top\mathbf{A})})$.

Proof. We denote by $\mathbf{H}_0$ the input to the Transformer. To prove the desired result we present in steps the construction.

Step 1: Initialize (1 layer). We consider one Transformer layer with 2 heads. For the first head, we choose

$$\mathbf{W}_V^{(1)} = \begin{pmatrix} & \epsilon\mathbb{I}_d & & & & & \\ & \mathbb{I}_d & & & & & \\ & & & & & & \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ & & & & & & \end{pmatrix}, \mathbf{W}_K^{(1)} = \begin{pmatrix} & \mathbb{I}_d & & & & & \\ & & & & & & \\ & & & & & & \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ & & & & & & \end{pmatrix}, \mathbf{W}_Q^{(1)} = \begin{pmatrix} \mathbb{I}_d & & & & & & \\ & & & & & & \\ & & & & & & \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ & & & & & & \end{pmatrix}$$

so that

$$\mathbf{W}_V^{(1)}\mathbf{H}_0 = \begin{pmatrix} \epsilon\mathbf{A}^\top \\ \mathbf{A}^\top \\ \\ \vdots \\ \end{pmatrix}, \quad \mathbf{W}_K^{(1)}\mathbf{H}_0 = \begin{pmatrix} \mathbf{A}^\top \\ \\ \\ \vdots \\ \end{pmatrix}, \quad \mathbf{W}_Q^{(1)}\mathbf{H}_0 = \begin{pmatrix} [\mathbb{I}_d\ ] \\ \\ \\ \vdots \\ \end{pmatrix}$$

For the second head, we choose

$$\mathbf{W}_V^{(2)} = \begin{pmatrix} -\mathbb{I}_d & & & & & & \\ -\mathbb{I}_d & & & & & & \\ & & & & & & \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ & & & & & & \end{pmatrix}, \mathbf{W}_K^{(2)} = \begin{pmatrix} \mathbb{I}_d & & & & & & \\ & & & & & & \\ & & & & & & \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ & & & & & & \end{pmatrix}, \mathbf{W}_Q^{(2)} = \begin{pmatrix} \mathbb{I}_d & & & & & & \\ & & & & & & \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ & & & & & & \end{pmatrix}$$

so that

$$\mathbf{W}_V^{(2)}\mathbf{H}_0 = \begin{pmatrix} [-\mathbb{I}_d\ ] \\ [-\mathbb{I}_d\ ] \\ \\ \vdots \\ \end{pmatrix}, \quad \mathbf{W}_K^{(2)}\mathbf{H}_0 = \begin{pmatrix} [\mathbb{I}_d\ ] \\ \\ \\ \vdots \\ \end{pmatrix}, \quad \mathbf{W}_Q^{(2)}\mathbf{H}_0 = \begin{pmatrix} [\mathbb{I}_d\ ] \\ \\ \\ \vdots \\ \end{pmatrix}.$$

Then combining the above two heads, we have

$$\begin{aligned} \mathbf{H}_1 &= \mathbf{H}_0 + \mathbf{W}_V^{(1)}\mathbf{H}_0(\mathbf{W}_K^{(1)}\mathbf{H}_0)^\top\mathbf{W}_Q^{(1)}\mathbf{H}_0 + \mathbf{W}_V^{(2)}\mathbf{H}_0(\mathbf{W}_K^{(2)}\mathbf{H}_0)^\top\mathbf{W}_Q^{(2)}\mathbf{H}_0 \\ &= \mathbf{H}_0 - \begin{pmatrix} [-\mathbb{I}_d\ ] \\ [-\mathbb{I}_d\ ] \\ \\ \\ \\ \\ \end{pmatrix} + \begin{pmatrix} [\epsilon\mathbf{A}^\top\mathbf{A}\ ] \\ [\mathbf{A}^\top\mathbf{A}\ ] \\ \\ \\ \\ \\ \end{pmatrix} \\ &= \begin{pmatrix} [\epsilon\mathbf{A}^\top\mathbf{A}\ ] \\ [\mathbf{A}^\top\mathbf{A}\ ] \\ [\mathbb{I}_d\ ] \\ \mathbf{A}^\top \\ [\mathbf{a}_{\text{test}}^\top\ ] \\ \mathbf{y}^\top \\ 0, 0, \ldots, 0 \end{pmatrix}. \end{aligned}$$

Step 2: Implement T steps of Newton-Raphson (T layers). We now define $\mathbf{X}_0 = \epsilon \mathbf{A}^\top \mathbf{A}$ and $\mathbf{R} = \mathbf{A}^\top \mathbf{A}$. The input matrix to the next layer is in the following form (for $t = 0$)

$$\mathbf{H}_t = \begin{pmatrix} [\mathbf{X}_t\ ] \\ [\mathbf{R}\ ] \\ [\mathbb{I}_d\ ] \\ \mathbf{A}^\top \\ [\mathbf{a}_{\text{test}}^\top\ ] \\ \mathbf{y}^\top \\ 0, \ldots, 0 \end{pmatrix}.$$

We will show that one Transformer laye with two heads can yield the following output

$$\mathbf{H}_{t+1} = \begin{pmatrix} [\mathbf{X}_{t+1}\ ] \\ [\mathbf{R}\ ] \\ [\mathbb{I}_d\ ] \\ \mathbf{A}^\top \\ [\mathbf{a}_{\text{test}}^\top\ ] \\ \mathbf{y}^\top \\ \end{pmatrix}.$$

Here we choose the weight matrices for the first head to be

$$\mathbf{W}_V^{(1)} = \begin{pmatrix} -\mathbb{I}_d & & & & & & \\ & & & & & & \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ & & & & & & \end{pmatrix}, \mathbf{W}_K^{(1)} = \begin{pmatrix} \mathbb{I}_d & & & & & & \\ & & & & & & \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ & & & & & & \end{pmatrix}, \mathbf{W}_Q^{(1)} = \begin{pmatrix} \mathbb{I}_d & & & & & & \\ & & & & & & \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ & & & & & & \end{pmatrix},$$

so that

$$\mathbf{W}_V^{(1)}\mathbf{H}_t = \begin{pmatrix} [-\mathbf{X}_t\ ] \\ \\ \vdots \\ \\ \end{pmatrix}, \quad \mathbf{W}_K^{(1)}\mathbf{H}_t = \begin{pmatrix} [\mathbf{R}\ ] \\ \\ \vdots \\ \\ \end{pmatrix}, \quad \mathbf{W}_Q^{(1)}\mathbf{H}_t = \begin{pmatrix} [\mathbf{X}_t\ ] \\ \\ \vdots \\ \\ \end{pmatrix}.$$

Similarly, for the second head, we choose

$$\mathbf{W}_V^{(2)} = \begin{pmatrix} \mathbb{I}_d & & & & & & \\ & & & & & & \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ & & & & & & \end{pmatrix}, \mathbf{W}_K^{(2)} = \begin{pmatrix} & \mathbb{I}_d & & & & & \\ & & & & & & \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ & & & & & & \end{pmatrix}, \mathbf{W}_Q^{(2)} = \begin{pmatrix} & \mathbb{I}_d & & & & & \\ & & & & & & \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ & & & & & & \end{pmatrix},$$

so that

$$\mathbf{W}_V^{(2)}\mathbf{H}_t = \begin{pmatrix} [\mathbf{X}_t\ ] \\ \\ \vdots \\ \\ \end{pmatrix}, \quad \mathbf{W}_K^{(2)}\mathbf{H}_t = \begin{pmatrix} [\mathbb{I}_d\ ] \\ \\ \vdots \\ \\ \end{pmatrix}, \quad \mathbf{W}_Q^{(2)}\mathbf{H}_t = \begin{pmatrix} [\mathbb{I}_d\ ] \\ \\ \vdots \\ \\ \end{pmatrix}.$$

Combining these two heads, we obtain

$$\mathbf{H}_{t+1} = \mathbf{H}_t + \mathbf{W}_V^{(1)}\mathbf{H}_t(\mathbf{W}_K^{(1)}\mathbf{H}_t)^\top \mathbf{W}_Q^{(1)}\mathbf{H}_t + \mathbf{W}_V^{(2)}\mathbf{H}_t(\mathbf{W}_K^{(2)}\mathbf{H}_t)^\top \mathbf{W}_Q^{(2)}\mathbf{H}_t$$

$$
\begin{aligned}
&= \mathbf{H}_t + \begin{pmatrix} -[\mathbf{X}_t\mathbf{R}\mathbf{X}_t\,] \\ \\ \vdots \end{pmatrix} + \begin{pmatrix} [\mathbf{X}_t\,] \\ \\ \vdots \end{pmatrix} \\
&= \begin{pmatrix} [2\mathbf{X}_t - \mathbf{X}_t\mathbf{R}\mathbf{X}_t\,] \\ [\mathbf{R}\,] \\ [\mathbb{I}_d\,] \\ \mathbf{A}^\top \\ [\mathbf{a}_{\text{test}}^\top\,] \\ \mathbf{y}^\top \\ 0, \dots, 0 \end{pmatrix} \\
&= \begin{pmatrix} [\mathbf{X}_{t+1}\,] \\ [\mathbf{R}\,] \\ [\mathbb{I}_d\,] \\ \mathbf{A}^\top \\ [\mathbf{a}_{\text{test}}^\top\,] \\ \mathbf{y}^\top \\ 0, \dots, 0 \end{pmatrix}.
\end{aligned}
$$

Repeating the above for T many layers yields

$$
\mathbf{H}_{T+1} = \begin{pmatrix} [\mathbf{X}_T\,] \\ [\mathbf{R}\,] \\ [\mathbb{I}_d\,] \\ \mathbf{A}^\top \\ [\mathbf{a}_{\text{test}}^\top\,] \\ \mathbf{y}^\top \\ 0, \dots, 0 \end{pmatrix}.
$$

Step 3: Output (2 layers). We now create first the matrix $\mathbf{y}^\top\mathbf{A}\mathbf{X}_T$ with one Transformer layer and then the final prediction output with another layer. For the first layer, we choose

$$
\mathbf{W}_V = \begin{pmatrix} & & & & & & & \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ & & & & & & & \mathbf{1} \end{pmatrix}, \mathbf{W}_K = \begin{pmatrix} & & & \mathbb{I}_d & & & & \\ & & & & & & & \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ & & & & & & & \end{pmatrix}, \mathbf{W}_Q = \begin{pmatrix} & & & \mathbb{I}_d & & & & \\ & & & & & & & \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ & & & & & & & \end{pmatrix}
$$

so that

$$
\mathbf{W}_V\mathbf{H}_{T+1} = \begin{pmatrix} \\ \vdots \\ \\ \mathbf{y}^\top \end{pmatrix}, \quad \mathbf{W}_K\mathbf{H}_{T+1} = \begin{pmatrix} \mathbf{A}^\top \\ \\ \vdots \\ \end{pmatrix}, \quad \mathbf{W}_Q\mathbf{H}_{T+1} = \begin{pmatrix} [\mathbf{X}_T\,] \\ \\ \vdots \\ \end{pmatrix}.
$$

The output of this layer is

$$
\mathbf{H}_{T+2} = \mathbf{H}_{T+1} + \mathbf{W}_V\mathbf{H}_{T+1}(\mathbf{W}_K\mathbf{H}_{T+1})^\top\mathbf{W}_Q\mathbf{H}_{T+1} = \mathbf{H}_{T+1} + \begin{pmatrix} \\ \\ \\ \\ [\mathbf{y}^\top\mathbf{A}\mathbf{X}_T\,] \end{pmatrix} = \begin{pmatrix} [\mathbf{X}_T\,] \\ [\mathbf{R}\,] \\ [\mathbb{I}_d\,] \\ \mathbf{A}^\top \\ [\mathbf{a}_{\text{test}}^\top\,] \\ \mathbf{y}^\top \\ [\mathbf{y}^\top\mathbf{A}\mathbf{X}_T\,] \end{pmatrix}.
$$

Now for the final step, we construct a Transformer layer with two heads. For the first head, we choose

$$\mathbf{W}_V^{(1)} = \begin{pmatrix} & & & & & & \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ & & & & & & \mathbf{1} \end{pmatrix}, \mathbf{W}_K^{(1)} = \begin{pmatrix} & & & & & & 1 \\ & & & & & & \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ & & & & & & \end{pmatrix}, \mathbf{W}_Q^{(1)} = \begin{pmatrix} & & & & & & 1 \\ & & & & & & \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ & & & & & & \end{pmatrix}$$

so that

$$\mathbf{W}_V^{(1)}\mathbf{H}_{T+2} = \begin{pmatrix} & & \\ & \vdots & \\ & & \\ [\mathbf{y}^\top \mathbf{A}\mathbf{X}_T\ ] & & \end{pmatrix}, \quad \mathbf{W}_K^{(1)}\mathbf{H}_{T+2} = \begin{pmatrix} [\mathbf{a}_{\text{test}}^\top\ ] & & \\ & & \\ & \vdots & \\ & & \end{pmatrix}, \quad \mathbf{W}_Q^{(1)}\mathbf{H}_{T+2} = \begin{pmatrix} [1\ ] & & \\ & & \\ & \vdots & \\ & & \end{pmatrix}.$$

For the second head, we choose

$$\mathbf{W}_V^{(2)} = \begin{pmatrix} & & & & & & \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ & & & & & & -\mathbf{1} \end{pmatrix}, \mathbf{W}_K^{(2)} = \begin{pmatrix} & & & & & & \mathbb{I}_d \\ & & & & & & \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ & & & & & & \end{pmatrix}, \mathbf{W}_Q^{(2)} = \begin{pmatrix} & & & & & & \mathbb{I}_d \\ & & & & & & \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ & & & & & & \end{pmatrix}$$

so that

$$\mathbf{W}_V^{(2)}\mathbf{H}_{T+2} = \begin{pmatrix} & & \\ & \vdots & \\ & & \\ [\mathbf{y}^\top \mathbf{A}\mathbf{X}_T\ ] & & \end{pmatrix}, \quad \mathbf{W}_K^{(2)}\mathbf{H}_{T+2} = \begin{pmatrix} [\mathbb{I}_d\ ] & & \\ & & \\ & \vdots & \\ & & \end{pmatrix}, \quad \mathbf{W}_Q^{(2)}\mathbf{H}_{T+2} = \begin{pmatrix} [\mathbb{I}_d\ ] & & \\ & & \\ & \vdots & \\ & & \end{pmatrix}.$$

Putting these together, we obtain

$$\mathbf{H}_{T+3} = \mathbf{H}_{T+2} + \begin{pmatrix} & & \\ & & \\ & & \\ [\mathbf{y}^\top \mathbf{A}\mathbf{X}_T\mathbf{a}_{\text{test}}\ ] & & \end{pmatrix} + \begin{pmatrix} & & \\ & & \\ & & \\ -[\mathbf{y}^\top \mathbf{A}\mathbf{X}_T\ ] & & \end{pmatrix} = \begin{pmatrix} [\mathbf{X}_T\ ] \\ [\mathbf{R}\ ] \\ [\mathbb{I}_d\ ] \\ \mathbf{A}^\top \\ [\mathbf{a}_{\text{test}}^\top\ ] \\ \mathbf{y}^\top \\ [\mathbf{y}^\top \mathbf{A}\mathbf{X}_T\mathbf{a}_{test}\ ] \end{pmatrix}$$

Notice that the last element of the last row is the desired quantity $\hat{y} = \mathbf{y}^\top \mathbf{A}\mathbf{X}_t\mathbf{a}_{\text{test}}$. This completes the proof. ■

B Convergence of inexact damped Newton's method for regularized logistic regression

We first review some basics for self-concordant functions in Appendix B.1, and then provide the proof of Theorem 4.4 in Appendix B.2.

B.1 Preliminaries on self-concordant functions

We review some results on self-concordant functions that will be useful for the next sections. Most of these theorems can be found in Boyd and Vandenberghe [2004], Nesterov et al. [2018].

First recall the definition of self-concordant functions.

Definition 3.1. [Self-concordant function; Definition 5.1.1, Nesterov et al. 2018] Let $f : \mathbb{R}^d \to \mathbb{R}$ be a closed convex function that is 3 times continuously differentiable on its domain $\mathrm{dom}(f) := \{\mathbf{x} \in \mathbb{R}^d \mid f(\mathbf{x}) < \infty\}$. For any fixed $\mathbf{x}, \mathbf{u} \in \mathbb{R}^d$ and $t \in \mathbb{R}$, define $\phi(t; \mathbf{x}, \mathbf{u}) := f(\mathbf{x} + t\mathbf{u})$ as a function of t. Then we say f is *self-concordant* if there exists a constant M_f such that, for all $\mathbf{x} \in \mathrm{dom}(f)$ and $\mathbf{u} \in \mathbb{R}^d$ with $\mathbf{x} + t\mathbf{u} \in \mathrm{dom}(f)$ for all sufficiently small t,

$$|\phi'''(0; \mathbf{x}, \mathbf{u})| \leq 2M_f(\mathbf{u}^\top \nabla^2 f(\mathbf{x})\mathbf{u})^{3/2}$$

We say f is *standard self-concordant* when $M_f = 1$.

For the regularized logistic regression problem that we consider, the objective function is strongly convex, and the theorems provided below use the strong convexity. Nonetheless, these theorems have more general forms, as detailed in Chapter 5 of Nesterov et al. [2018].

In the sequel, let f be a self-concordant function.

Definition B.1 (Dikin ellipsoid)**.** For a function $f : \mathbb{R}^d \to \mathbb{R}$, consider the following sets for any $\mathbf{x} \in \mathbb{R}^d$ and $r > 0$:

$$\mathcal{W}(\mathbf{x}; r) = \{\mathbf{y} \in \mathbb{R}^d : \|\mathbf{y} - \mathbf{x}\|_{\nabla^2 f(\mathbf{x})} < r\}$$

where $cl(\cdot)$ defines the closure of a set. This set is called the *Dikin ellipsoid* of the function f at $\mathbf{x}$.

Theorem B.2 (Theorem 5.1.5 in Nesterov et al. [2018])**.** *Let $f : \mathbb{R}^d \to \mathbb{R}$ be a self-concordant function. Then for any $\mathbf{x} \in \mathrm{dom}(f)$, it holds that $\mathcal{W}(\mathbf{x}; 1/M_f) \subseteq \mathrm{dom}(f)$.*

Theorem B.3 (Theorem 5.1.8 & 5.1.9 in Nesterov et al. [2018])**.** *Let $f : \mathbb{R}^d \to \mathbb{R}$ be a self-concordant function. Then for any $\mathbf{x}, \mathbf{y} \in \mathrm{dom}(f)$, it holds that*

$$f(\mathbf{x}) + \langle \nabla f(\mathbf{x}), \mathbf{y} - \mathbf{x} \rangle + \frac{\omega(M_f \|\mathbf{y} - \mathbf{x}\|_{\nabla^2 f(\mathbf{x})})}{M_f^2} \leq f(\mathbf{y}) \leq f(\mathbf{x}) + \langle \nabla f(\mathbf{x}), \mathbf{y} - \mathbf{x} \rangle + \frac{\omega_*(M_f \|\mathbf{y} - \mathbf{x}\|_{\nabla^2 f(\mathbf{x})})}{M_f^2}$$

where $\omega(t) = t - \ln(1 + t)$, $\omega_*(t) = -t - \ln(1 - t)$.

Lemma B.4 (Lemma 5.1.5 in Nesterov et al. [2018])**.** *For any $t \geq 0$, the functions $\omega(t), \omega_*(t)$ in Theorem B.3 satisfy:*

$$\frac{t^2}{2(1+t)} \leq \frac{t^2}{2(1+2t/3)} \leq \omega(t) \leq \frac{t^2}{2+t}, \quad \frac{t^2}{2-t} \leq \omega_*(t) \leq \frac{t^2}{2(1-t)}.$$

Theorem B.5 (Theorem 5.1.7, Nesterov et al. 2018)**.** *Let $f : \mathbb{R}^d \to \mathbb{R}$ be a self-concordant function. Then for any $\mathbf{x} \in \mathrm{dom}(f)$ and any $\mathbf{y} \in \mathcal{W}(\mathbf{x}; 1/M_f)$, it holds that*

$$(1 - M_f r)^2 \cdot \nabla^2 f(\mathbf{x}) \preceq \nabla^2 f(\mathbf{y}) \preceq \frac{1}{(1 - M_f r)^2} \cdot \nabla^2 f(\mathbf{x}).$$

where $r = \|\mathbf{y} - \mathbf{x}\|_{\nabla^2 f(\mathbf{x})}$.

A useful corollary of the above theorem is

Corollary B.6 (Corollary 5.1.5, Nesterov et al. 2018)**.** *Let $f : \mathbb{R}^d \to \mathbb{R}$ be a self-concordant function. Then for any $\mathbf{x} \in \mathrm{dom}(f)$ and any $\mathbf{y}$ such that $r = \|\mathbf{y} - \mathbf{x}\|_{\nabla^2 f(\mathbf{x})} < 1/M_f$, it holds that*

$$\left(1 - M_f r + \frac{1}{3} M_f^2 r^2\right) \cdot \nabla^2 f(\mathbf{x}) \preceq \int_0^1 \nabla^2 f(\mathbf{x} + \tau(\mathbf{y} - \mathbf{x}))\mathrm{d}\tau \preceq \frac{1}{1 - M_f r} \cdot \nabla^2 f(\mathbf{x}).$$

A key quantity in the analysis of Newton method for self-concordant functions is the so-called Newton decrement:

$$\lambda_f(\mathbf{x}) = (\nabla f(\mathbf{x})^\top \nabla^2 f(\mathbf{x})^{-1} \nabla f(\mathbf{x}))^{1/2} \tag{B.1}$$

The following theorem characterizes the sub-optimality gap in terms of the Newton decrement for a strongly convex self-concordant function.

Theorem B.7 (Theorem 5.1.13 in Nesterov et al. [2018]). *Let $f : \mathbb{R}^d \to \mathbb{R}$ be a strongly convex self-concordant function with $\mathbf{x}^* = \operatorname{argmin}_{\mathbf{x}} f(\mathbf{x})$. Suppose $\lambda_f(\mathbf{x}) < 1/M_f$ for some $\mathbf{x} \in \operatorname{dom}(f)$, then the following holds:*

$$f(\mathbf{x}) - f(\mathbf{x}^*) \leq \frac{1}{M_f^2} \cdot \omega_*(M_f \lambda_f(\mathbf{x}))$$

where $\omega_*(\cdot)$ is the function defined in Theorem B.3.

B.2 Convergence analysis of inexact damped Newton's method

In this section we consider the inexact damped Newton's method for optimizing the regularized logistic loss:

$$\hat{\mathbf{x}}_{t+1} = \hat{\mathbf{x}}_t - \eta(\hat{\mathbf{x}}_t)(\nabla^2 f(\hat{\mathbf{x}}_t))^{-1} \nabla f(\hat{\mathbf{x}}_t) + \varepsilon_t \tag{B.2}$$

where ε_t is an error term. For simplicity, we also define

$$_t := -\eta(\hat{\mathbf{x}}_t)(\nabla^2 f(\hat{\mathbf{x}}_t))^{-1} \nabla f(\hat{\mathbf{x}}_t) + \varepsilon_t. \tag{B.3}$$

In other words, we have $\hat{\mathbf{x}}_{t+1} = \hat{\mathbf{x}}_t + _t$.

Recall that the regularized logistic loss defined in Equation (3.5) is self-concordant with $M_f = 1/\sqrt{\mu}$, as a consequence of Proposition 3.3.

Lemma B.8. *Under Assumption 3.2, let f be the regularized logistic loss defined in Equation (3.5) with regularization parameter $\mu > 0$. Then the following bounds hold*

$$-1 + \mu\|\mathbf{x}\|_2 \leq \|\nabla f(\mathbf{x})\|_2 \leq 1 + \mu\|\mathbf{x}\|_2,$$
$$\mu \leq \left\|\nabla^2 f(\mathbf{x})\right\|_{\mathrm{op}} \leq 1 + \mu.$$

Consequently, for the Newton decrement $\lambda(\mathbf{x}) = \sqrt{\nabla f(\mathbf{x})^\top (\nabla^2 f(\mathbf{x}))^{-1} \nabla f(\mathbf{x})}$, it holds that

$$\frac{-1 + \mu\|\mathbf{x}\|_2}{\sqrt{1+\mu}} \leq \lambda(\mathbf{x}) \leq \frac{1 + \mu\|\mathbf{x}\|_2}{2\sqrt{\mu}}.$$

Proof of Lemma B.8. For $|f(\mathbf{x})|$, note that for each $i \in [n]$, we have $-y_i \mathbf{x}^\top \mathbf{a}_i \leq \|\mathbf{x}\|_2$ because $y_i \in \{-1, 1\}$ and $\|\mathbf{a}_i\|_2 \leq 1$. This implies the first bound on $|f(\mathbf{x})|$.

Next, recall the gradient and Hessian of f:

$$\nabla f(\mathbf{x}) = -\frac{1}{n}\sum_{i=1}^n y_i p_i \mathbf{a}_i + \mu\mathbf{x}, \quad \nabla^2 f(\mathbf{x}) = \frac{1}{n}\mathbf{A}^\top \mathbf{D}\mathbf{A} + \mu\mathbb{I}_d,$$

where each $p_i := \exp(-y_i \mathbf{x}^\top \mathbf{a}_i)/(1 + \exp(-y_i \mathbf{x}^\top \mathbf{a}_i))$ and $\mathbf{D} := \operatorname{diag}(p_1(1-p_1), \ldots, p_n(1-p_n))$. By triangle inequality, we have

$$\|\nabla f(\mathbf{x})\|_2 \leq \frac{1}{n}\sum_{i=1}^n |y_i p_i| \|\mathbf{a}_i\|_2 + \mu\|\mathbf{x}\|_2 \leq 1 + \mu\|\mathbf{x}\|_2$$

where the last inequality follows from Assumption 3.2. Similarly, for the Hessian, we have

$$\|\nabla^2 f(\mathbf{x})\|_{\mathrm{op}} \leq \frac{1}{n}\|\mathbf{A}^\top \mathbf{D}\mathbf{A}\|_{\mathrm{op}} + \mu \leq \frac{1}{n}\|\mathbf{A}\|_{\mathrm{op}}^2 \|\mathbf{D}\|_{\mathrm{op}} + \mu \leq 1 + \mu.$$

Finally, the lower bound on $\|\nabla^2 f(\mathbf{x})\|_{\mathrm{op}}$ follows from the fact that $\frac{1}{n}\mathbf{A}^\top \mathbf{D}\mathbf{A}$ is positive definite. This completes the proof. ■

Lemma B.9. *Under Assumption 3.2, let f be the regularized logistic loss defined in Equation (3.5) with regularization parameter $\mu > 0$. Then there exists some constant $C > 0$ depending on μ such that for any $\varepsilon \in \mathbb{R}^d$ with $\|\varepsilon\|_2 \leq \mu^2$, if $\|\mathbf{x}\|_2 \geq C$, then*

$$\left\|\mathbf{x} - \frac{2\sqrt{\mu}}{2\sqrt{\mu} + \lambda(\mathbf{x})} \nabla^2 f(\mathbf{x})^{-1} \nabla f(\mathbf{x}) + \varepsilon\right\|_2 \leq \|\mathbf{x}\|_2.$$

Proof of Lemma B.9. For simplicity, denote

$$\mathbf{x}' = \mathbf{x} - \frac{2\sqrt{\mu}}{2\sqrt{\mu}+\lambda(\mathbf{x})}\nabla^2 f(\mathbf{x})^{-1}\nabla f(\mathbf{x}) + \varepsilon, \quad \eta(\mathbf{x}) = \frac{2\sqrt{\mu}}{2\sqrt{\mu}+\lambda(\mathbf{x})}.$$

It follows that

$$\begin{aligned}\|\mathbf{x}'\|_2^2 &= \|\mathbf{x}\|_2^2 - 2\eta(\mathbf{x})\mathbf{x}^\top\nabla^2 f(\mathbf{x})^{-1}\nabla f(\mathbf{x}) + \eta(\mathbf{x})^2\|\nabla^2 f(\mathbf{x})^{-1}\nabla f(\mathbf{x})\|_2^2 \\ &\quad + 2\varepsilon^\top(\mathbf{x} - \eta(\mathbf{x})\nabla^2 f(\mathbf{x})^{-1}\nabla f(\mathbf{x})) + \|\varepsilon\|_2^2 \\ &\le \|\mathbf{x}\|_2^2 - 2\eta(\mathbf{x})\mathbf{x}^\top\nabla^2 f(\mathbf{x})^{-1}\nabla f(\mathbf{x}) + \eta(\mathbf{x})^2\|\nabla^2 f(\mathbf{x})^{-1}\nabla f(\mathbf{x})\|_2^2 \\ &\quad + 2\mu^2\|\mathbf{x}\|_2 + 2\mu\eta(\mathbf{x})\|\nabla^2 f(\mathbf{x})^{-1}\nabla f(\mathbf{x})\| + \mu^4 \end{aligned} \tag{B.4}$$

where we used the fact that $\|\varepsilon\|_2 \le \mu^2$ and triangle inequality. Plugging in the expression for $\nabla f(\mathbf{x})$, we have

$$\begin{aligned}\mathbf{x}^\top\nabla^2 f(\mathbf{x})^{-1}\nabla f(\mathbf{x}) &= -\frac{1}{n}\sum_{i=1}^n y_i p_i \mathbf{x}^\top\nabla^2 f(\mathbf{x})^{-1}\mathbf{a}_i^\top + \mu\mathbf{x}^\top\nabla^2 f(\mathbf{x})^{-1}\mathbf{x} \\ &\ge -\|\nabla^2 f(\mathbf{x})^{-1}\mathbf{x}\|_2 + \mu\mathbf{x}^\top\nabla^2 f(\mathbf{x})^{-1}\mathbf{x} \\ &\ge -\frac{1}{\mu}\|\mathbf{x}\|_2 + \mu^2\|\mathbf{x}\|_2^2 \end{aligned} \tag{B.5}$$

where the first inequality follows from Assumption 3.2 and triangle inequality, and the second inequality is due to Lemma B.8. Similarly, we also have

$$\|\nabla^2 f(\mathbf{x})^{-1}\nabla f(\mathbf{x})\|_2 \le \frac{1}{\mu}\|\nabla f(\mathbf{x})\|_2 \le \frac{1}{\mu} + \|\mathbf{x}\|_2.$$

Combining this with Equations (B.4) and (B.5), we obtain (after rearrangement of the terms)

$$\begin{aligned}\|\mathbf{x}'\|_2^2 &\le \|\mathbf{x}\|_2^2 + \frac{2}{\mu}\eta(\mathbf{x})\|\mathbf{x}\|_2 - 2\mu^2\eta(\mathbf{x})\|\mathbf{x}\|_2^2 + \eta(\mathbf{x})^2\left(\frac{1}{\mu} + \|\mathbf{x}\|_2\right)^2 \\ &\quad + 2\mu^2\|\mathbf{x}\|_2 + 2\mu\eta(\mathbf{x})\left(\frac{1}{\mu} + \|\mathbf{x}\|_2\right) + \mu^4 \\ &= \|\mathbf{x}\|_2^2 - 2\mu^2\eta(\mathbf{x})\|\mathbf{x}\|_2^2 + 2\mu^2\|\mathbf{x}\|_2 + \mu^4 \\ &\quad + \frac{2}{\mu}\eta(\mathbf{x})\|\mathbf{x}\|_2 + \eta(\mathbf{x})^2\left(\frac{1}{\mu} + \|\mathbf{x}\|_2\right)^2 + 2\mu\eta(\mathbf{x})\left(\frac{1}{\mu} + \|\mathbf{x}\|_2\right)\end{aligned}$$

Further applying the bounds on $\lambda(\mathbf{x})$ from Lemma B.8, with direct computation, we have

$$\begin{aligned}\|\mathbf{x}'\|_2^2 \le \|\mathbf{x}\|_2^2 &\underbrace{- \frac{8\mu^3}{4\mu + 1 + \mu\|\mathbf{x}\|_2}\|\mathbf{x}\|_2^2 + 2\mu^2\|\mathbf{x}\|_2}_{\mathcal{I}_1} + \mu^4 \\ &+ \underbrace{\frac{4\sqrt{\mu(1+\mu)}\|\mathbf{x}\|_2}{2\sqrt{\mu^3(1+\mu)} - \mu + \mu^2\|\mathbf{x}\|_2} + \frac{4\mu(1+\mu)(1/\mu + \|\mathbf{x}\|_2)^2}{(2\sqrt{\mu(1+\mu)} - 1 + \mu\|\mathbf{x}\|_2)^2} + \frac{8\mu(1/\mu + \|\mathbf{x}\|_2)}{4\mu + 1 + \mu\|\mathbf{x}\|_2}}_{\mathcal{I}_2}.\end{aligned}$$

Observe that for sufficiently large $\|\mathbf{x}\|_2$, we have $\mathcal{I}_1 \approx -6\mu^2\|\mathbf{x}\|_2$ and $\mathcal{I}_2 = O(1)$. This implies that $\|\mathbf{x}'\|_2 \le \|\mathbf{x}\|_2$ for sufficiently large $\|\mathbf{x}\|_2$, and thus completes the proof. ■

Lemma B.10. *Under Assumption 3.2, let f be the regularized logistic loss defined in Equation (3.5) with regularization parameter $\mu > 0$. Consider a sequence of iterates $\{\mathbf{x}_t\}_{t\ge 0}$ satisfying*

$$\mathbf{x}_{t+1} = \mathbf{x}_t - \frac{2\sqrt{\mu}}{2\sqrt{\mu}+\lambda(\mathbf{x}_t)}\nabla^2 f(\mathbf{x}_t)^{-1}\nabla f(\mathbf{x}_t) + \varepsilon_t$$

where $\|\varepsilon_t\|_2 \le \mu^2$ for all $t \ge 0$. Then there exists a constant C depending on μ such that $\|\mathbf{x}_t\|_2 \le C$ for all $t \ge 0$.

Proof of Lemma B.10. First, there exists a constant C_1 (given by Lemma B.9) such that if $\|\mathbf{x}_t\|_2 \geq C_1$, then $\|\mathbf{x}_{t+1}\|_2 \leq \|\mathbf{x}_t\|_2$. Then we define

$$C_2 = \max_{\substack{\mathbf{x}:\|\mathbf{x}\|_2 \leq C_1 \\ \varepsilon:\|\varepsilon\|_2 \leq \mu^2}} \left\| \mathbf{x} - \frac{2\sqrt{\mu}}{2\sqrt{\mu} + \lambda(\mathbf{x})} \nabla^2 f(\mathbf{x})^{-1} \nabla f(\mathbf{x}) + \varepsilon \right\|_2.$$

Note that by Lemma B.8, C_2 is a constant depending only on C_1 and the regularization parameter μ. Finally, we choose $C = \max\{\|\mathbf{x}_0\|_2, C_1, C_2\}$, and the result follows. ■

Our target is to prove convergence of the inexact damped Newton's method up to some error threshold, which depends on the bound of the error terms $\{\varepsilon_t\}$. At a high level, the proof strategy is as follows:

- **Step 1**: Show by induction that if $\mathbf{x}_t$ is feasible, then for sufficiently small error ε_t, so is the next iteration. This leads to some constraint on the error term ε_t.
- **Step 2**: Show constant decrease of the loss function when $\lambda(\hat{\mathbf{x}}_t) \geq 1/6$.
- **Step 3**: Show that when $\lambda(\mathbf{x}_t) < 1/6$, we have $\lambda(\mathbf{x}_{t+1}) \leq c\lambda_t^2 + \epsilon'$ for some constant $c > 0$, so we enter the regime of quadratic convergence, which is maintained up to error ε.

Theorem 4.4. *Under Assumption 3.2, for the regularized logistic loss f defined in (3.5) with regularization parameter $\mu > 0$, consider a sequence of iterates $\{\mathbf{x}_t\}_{t\geq 0}$ satisfying*

$$\mathbf{x}_{t+1} = \mathbf{x}_t - \frac{2\sqrt{\mu}}{2\sqrt{\mu} + \lambda(\mathbf{x}_t)} \nabla^2 f(\mathbf{x}_t)^{-1} \nabla f(\mathbf{x}_t) + \varepsilon_t$$

where $\|\varepsilon_t\|_2 \leq \epsilon$ for all $t \geq 0$. Then there exists constants c, C_1, C_2 depending only on μ such that for any $\epsilon \leq c$, it holds that $g(\mathbf{x}_t) - g(\mathbf{x}^*) \leq \epsilon$ for all $t \geq C_1 + C_2 \log\log \frac{1}{\epsilon}$.

Remark B.11. In more detail the error obtained is $\alpha + \varepsilon$, where $\alpha \geq \sqrt{\epsilon(1+\mu)/(4\mu)}$, given that the algorithm performs $T = c + \log\log \dfrac{1}{1/2 + 3\alpha} + \log\log \dfrac{1}{\varepsilon}$, where c denotes the initial steps of constant decrease of the function g. By picking $\varepsilon = \sqrt{\epsilon}$ we get the desired result.

Proof. First by Lemma B.10, we know that there exists a constant C such that $\|\mathbf{x}_t\|_2 \leq C$ for all $t \geq 0$. We first derive an upper bound for $\delta_t := \|_t\|_{\nabla^2 g(\mathbf{x}_t)}$. By definition, we have

$$\begin{aligned}
\delta_t^2 &= (-\eta(\mathbf{x}_t)\nabla^2 g(\mathbf{x}_t)^{-1}\nabla g(\mathbf{x}_t) + \varepsilon_t)^\top \nabla^2 g(\mathbf{x}_t)(-\eta(\mathbf{x}_t)\nabla^2 g(\mathbf{x}_t)^{-1}\nabla g(\mathbf{x}_t) + \varepsilon_t) \\
&= \eta(\mathbf{x}_t)^2 \nabla g(\mathbf{x}_t)^\top \nabla^2 g(\mathbf{x}_t)^{-1} \nabla g(\mathbf{x}_t) - 2\eta(\mathbf{x}_t)\varepsilon_t^\top \nabla g(\mathbf{x}_t) + \varepsilon_t^\top \nabla^2 g(\mathbf{x}_t)^{-1}\varepsilon_t \\
&= \frac{\lambda(\mathbf{x}_t)^2}{(1+\lambda(\mathbf{x}_t))^2} - \frac{2\varepsilon_t^\top \nabla g(\mathbf{x}_t)}{1+\lambda(\mathbf{x}_t)} + \varepsilon_t^\top \nabla^2 g(\mathbf{x}_t)^{-1}\varepsilon_t.
\end{aligned}$$

It follows from Lemma B.8 that for all $t \geq 0$, $|\varepsilon_t^\top \nabla g(\mathbf{x}_t)| \leq \epsilon(1+\mu\|\mathbf{x}_t\|_2)/(4\mu) \leq \epsilon(1+C\mu)/(4\mu)$, and also $|\varepsilon_t^\top \nabla^2 g(\mathbf{x}_t)^{-1}\varepsilon_t| \leq \epsilon^2/4$. Therefore, we have

$$\begin{aligned}
\delta_t &\leq \sqrt{\frac{\lambda(\mathbf{x}_t)^2}{(1+\lambda(\mathbf{x}_t))^2} + \frac{\epsilon(1+C\mu)}{2\mu} + \frac{\epsilon^2}{4}} \\
&\leq \frac{\lambda(\mathbf{x}_t)}{1+\lambda(\mathbf{x}_t)} + \sqrt{\frac{\epsilon(1+C\mu)}{2\mu} + \frac{\epsilon^2}{4}}.
\end{aligned} \tag{B.6}$$

Note that by Lemma B.8, we have $\lambda(\mathbf{x}_t) \leq (1+\mu\|\mathbf{x}_t\|_2)/\sqrt{\mu} \leq (1+C\mu)/(2\sqrt{\mu})$. Then there exists some constant c_1 depending only on μ such that $\delta_t < 1$ when $\epsilon \leq c_1$.

Now, we proceed to show that there is a constant decrease of the loss value up to the point that $\lambda(\mathbf{x}) \leq 1/6$, after which we enter the regime of quadratic convergence.

Phase I: Constant decrease of the loss function. Suppose $\lambda(\mathbf{x}_t) \geq 1/6$ (note that if $\lambda(\mathbf{x}_0) < 1/6$, then we can directly proceed to the next phase). Since g is standard self-concordant, it follows from Theorem B.3 that

$$\begin{aligned} g(\mathbf{x}_{t+1}) - g(\mathbf{x}_t) &\leq \nabla g(\mathbf{x}_t)^\top (\mathbf{x}_{t+1} - \mathbf{x}_t) + \omega_*(\|\mathbf{x}_{t+1} - \mathbf{x}_t\|_{\nabla^2 g(\mathbf{x}_t)}) \\ &= -\eta(\mathbf{x}_t) \nabla g(\mathbf{x}_t)^\top \nabla^2 g(\mathbf{x}_t)^{-1} \nabla g(\mathbf{x}_t) + \varepsilon_t^\top \nabla g(\mathbf{x}_t) + \omega_*(\|\mathbf{x}_{t+1} - \mathbf{x}_t\|_{\nabla^2 g(\mathbf{x}_t)}) \\ &= -\frac{\lambda(\mathbf{x}_t)^2}{1+\lambda(\mathbf{x}_t)} + \varepsilon_t^\top \nabla g(\mathbf{x}_t) + \omega_*(\|\mathbf{x}_{t+1} - \mathbf{x}_t\|_{\nabla^2 g(\mathbf{x}_t)}) \end{aligned}$$

where we applied the definition of $\lambda(\mathbf{x}_t)$ and $\eta(\mathbf{x}_t)$ in the last equality. Combining the above two equations, we have

$$g(\mathbf{x}_{t+1}) - g(\mathbf{x}_t) \leq -\frac{\lambda(\mathbf{x}_t)^2}{1+\lambda(\mathbf{x}_t)} + \varepsilon_t^\top \nabla g(\mathbf{x}_t) + \omega_*\bigg(\frac{\lambda(\mathbf{x}_t)^2}{(1+\lambda(\mathbf{x}_t))^2} - \frac{2\varepsilon_t^\top \nabla g(\mathbf{x}_t)}{1+\lambda(\mathbf{x}_t)} + \varepsilon_t^\top \nabla^2 g(\mathbf{x}_t)^{-1} \varepsilon_t \bigg).$$

Recall that $w_*(x) = -x - \log(1-x)$, and we view the right-hand side as a function of $\lambda(\mathbf{x}_t)$ while regarding $c \equiv \varepsilon_t^\top \nabla g(\mathbf{x}_t)$ and $c' \equiv \varepsilon_t^\top \nabla^2 g(\mathbf{x}_t)^{-1} \varepsilon_t$ as constants, yielding

$$h(x) := -\frac{x^2}{1+x} - \bigg(\frac{x^2}{(1+x)^2} - \frac{2c}{1+x} + c' \bigg) - \log\bigg(1 - \frac{x^2}{(1+x)^2} + \frac{2c}{1+x} - c' \bigg) + c.$$

Since $|c| \leq 0.06$ by our assumption on ε_t, it can be verified that $h(x)$ is decreasing in x for $x \geq 1/6$, and moreover $h(1/6) \leq 0.01$ by our assumption on ε_t (see Appendix E.1). Therefore, we have $g(\mathbf{x}_{t+1}) - g(\mathbf{x}_t) \leq -0.01$ for all t such that $\lambda(\mathbf{x}_t) \geq 1/6$.

Phase II: Quadratic convergence. Now suppose $\sqrt{\epsilon} < \lambda(\mathbf{x}_t) < 1/6$ (again, if $\lambda(\mathbf{x}_0) < \sqrt{\epsilon}$, then we are done). Note that there exists some constant C_1 such that $t \leq C_1$ due to the constant decrease of the loss function in the previous phase. Then by Theorem B.7 and Lemma B.4, we have

$$g(\mathbf{x}_t) - g(\mathbf{x}^*) \leq \frac{3\lambda(\mathbf{x}_t)^2}{5} \tag{B.7}$$

where $\mathbf{x}^*$ is the global minimizer of g. Thus it suffices to characterize the decrease of $\lambda(\mathbf{x}_t)$.

Applying Theorem B.5, we get

$$\lambda(\mathbf{x}_{t+1}) = \|\nabla^2 g(\mathbf{x}_{t+1})^{-1/2} \nabla g(\mathbf{x}_{t+1})\|_2 \leq \frac{1}{1 - \|_t\|_{\nabla^2 g(\mathbf{x}_t)}} \|\nabla^2 g(\mathbf{x}_t)^{-1/2} \nabla g(\mathbf{x}_{t+1})\|_2. \tag{B.8}$$

Note that $\nabla g(\mathbf{x}_{t+1}) = \nabla g(\mathbf{x}_t) + \int_0^1 \nabla^2 g(\mathbf{x}_t + s_t)_t \mathrm{d}s$. Also, we have $\nabla g(\mathbf{x}_t) = -(1+\lambda(\mathbf{x}_t)) \nabla^2 g(\mathbf{x}_t)_t + (1+\lambda(\mathbf{x}_t)) \nabla^2 g(\mathbf{x}_t) \varepsilon_t$. Combining these two equations, we obtain

$$\nabla g(\mathbf{x}_{t+1}) = \underbrace{\int_0^1 \big(\nabla^2 g(\mathbf{x}_t + s_t) - (1+\lambda(\mathbf{x}_t) \nabla^2 g(\mathbf{x}_t)) \big) \mathrm{d}s}_{=:\mathbf{G}_t} \cdot_t + (1+\lambda(\mathbf{x}_t)) \nabla^2 g(\mathbf{x}_t) \varepsilon_t$$

where we introduced the notation $\mathbf{G}_t$ for the integral term. Therefore, by triangle inequality,

$$\begin{aligned} \|\nabla^2 g(\mathbf{x}_t)^{-1/2} \nabla g(\mathbf{x}_{t+1})\|_2 &\leq \big\| \nabla^2 g(\mathbf{x}_t)^{-1/2} \mathbf{G}_{tt} \big\|_2 + (1+\lambda(\mathbf{x}_t)) \big\| \nabla^2 g(\mathbf{x}_t)^{1/2} \varepsilon_t \big\|_2 \\ &= \big\| \eta(\mathbf{x}_t) \nabla^2 g(\mathbf{x}_t)^{-1/2} \mathbf{G}_t \nabla^2 g(\mathbf{x}_t)^{-1} \nabla g(\mathbf{x}_t) \big\|_2 \\ &\quad + \big\| \nabla^2 g(\mathbf{x}_t)^{-1/2} \mathbf{G}_t \varepsilon_t \big\|_2 + (1+\lambda(\mathbf{x}_t)) \big\| \nabla^2 g(\mathbf{x}_t)^{1/2} \varepsilon_t \big\|_2 \end{aligned} \tag{B.9}$$

By Corollary B.6, it holds that

$$\bigg(-\delta_t + \frac{\delta_t^2}{3} - \lambda(\mathbf{x}_t) \bigg) \nabla^2 g(\mathbf{x}_t) \preceq \mathbf{G}_t \preceq \bigg(\frac{1}{1-\delta_t} - 1 - \lambda(\mathbf{x}_t) \bigg) \nabla^2 g(\mathbf{x}_t).$$

By direct computation, it can be verified that $0 \le \delta_t/(1-\delta_t) - \lambda(\mathbf{x}_t) \le \delta_t + \lambda(\mathbf{x}_t)$. This implies that $\|\nabla^2 g(\mathbf{x}_t)^{-1/2}\mathbf{G}_t\|_{\mathrm{op}} \le (\delta_t + \lambda(\mathbf{x}_t))\|\nabla^2 g(\mathbf{x}_t)^{-1/2}\|_{\mathrm{op}}$ and $\|\nabla^2 g(\mathbf{x}_t)^{-1/2}\mathbf{G}_t\nabla^2 g(\mathbf{x}_t)^{-1/2}\|_{\mathrm{op}} \le \delta_t + \lambda(\mathbf{x}_t)$. Applying these bounds to Equation (B.9), we obtain

$$\begin{aligned}\|\nabla^2 g(\mathbf{x}_t)^{-1/2}\nabla g(\mathbf{x}_{t+1})\|_2 &\le \eta(\mathbf{x}_t)(\delta_t + \lambda(\mathbf{x}_t))\|\nabla^2 g(\mathbf{x}_t)^{-1/2}\nabla g(\mathbf{x}_t)\|_2 \\ &\quad + \epsilon(\delta_t + \lambda(\mathbf{x}_t))\|\nabla^2 g(\mathbf{x}_t)^{-1/2}\|_{\mathrm{op}} + \epsilon(1+\lambda(\mathbf{x}_t))\|\nabla^2 g(\mathbf{x}_t)^{1/2}\|_{\mathrm{op}} \\ &\le \frac{\lambda(\mathbf{x}_t)(\delta_t+\lambda(\mathbf{x}_t))}{1+\lambda(\mathbf{x}_t)} + \epsilon(\delta_t+\lambda(\mathbf{x}_t))\left(\frac{1}{2} + \sqrt{\frac{1+\mu}{4\mu}}\right)\end{aligned}$$

where the last inequality follows from Lemma B.8. Plugging this into Equation (B.8), we obtain

$$\begin{aligned}\lambda(\mathbf{x}_{t+1}) &\le \frac{\lambda(\mathbf{x}_t)(\delta_t+\lambda(\mathbf{x}_t))}{(1+\lambda(\mathbf{x}_t))(1-\delta_t)} + \frac{\epsilon(\delta_t+\lambda(\mathbf{x}_t))}{1-\delta_t}\left(\frac{1}{2} + \sqrt{\frac{1+\mu}{4\mu}}\right) \\ &= \frac{\lambda(\mathbf{x}_t)^2}{(1+\lambda(\mathbf{x}_t))(1-\delta_t)} + \frac{\delta_t\lambda(\mathbf{x}_t)}{(1+\lambda(\mathbf{x}_t))(1-\delta_t)} + \frac{\epsilon(\delta_t+\lambda(\mathbf{x}_t))}{1-\delta_t}\left(\frac{1}{2} + \sqrt{\frac{1+\mu}{4\mu}}\right).\end{aligned}$$

Note that there exists some constant c_2 depending only on μ such that if $\epsilon < c_2$, then when $\lambda(\mathbf{x}_t) < 1/6$, we have $\delta_t \le 1/5$ by Equation (B.6). Then, for some constant $C' > 0$, we further have

$$\lambda(\mathbf{x}_{t+1}) \le 3\lambda(\mathbf{x}_t)^2 + C'\epsilon.$$

Let $\alpha = \sqrt{C'\epsilon/3} \le 1/6$. We can rewrite the above inequality as

$$\lambda(\mathbf{x}_{t+1}) - \alpha \le 3\lambda(\mathbf{x}_t)^2 + C'\epsilon + \alpha = 3(\lambda(\mathbf{x}_t) - \alpha)^2.$$

Telescoping this inequality, we obtain that for some constant $C_2 > 0$, $\lambda(\mathbf{x}_t) \le C''\sqrt{\epsilon}$ when $t \ge C_1 + C_2 \log\log\frac{1}{\epsilon}$. Combining this with (B.7), we see that for any such t, we have $g(\mathbf{x}_t) - g(\mathbf{x}^*) \le \frac{3C''}{5}\epsilon$. Finally, choosing $c = \min\{c_1, c_2\}$ completes the proof. ■

C Transformers for logistic regression

In this section, we present the explicit construction of a linear Transformer that can emulate Newton's method on the regularized logistic loss, and we further provide its error analysis.

Recall the regularized logistic loss defined as

$$f(\mathbf{x}) = \frac{1}{n}\sum_{i=1}^{n}\log(1+\exp(-y_i\mathbf{x}^\top\mathbf{a}_i)) + \frac{\mu}{2}\|\mathbf{x}\|_2^2,$$

and its gradient and Hessian

$$\nabla f(\mathbf{x}) = -\frac{1}{n}\sum_{i=1}^{n} y_i p_i \mathbf{a}_i + \mu\mathbf{x}, \qquad \nabla^2 f(\mathbf{x}) = \frac{1}{n}\mathbf{A}^\top\mathbf{D}\mathbf{A} + \mu\mathbb{I}_d \succeq \mu\mathbb{I}_d$$

where each $p_i = \exp(-y_i\mathbf{x}^\top\mathbf{a}_i)/(1+\exp(-y_i\mathbf{x}^\top\mathbf{a}_i))$ and $\mathbf{D} = \mathrm{diag}(p_1(1-p_1), \ldots, p_n(1-p_n))$.

Letting $g(\mathbf{x}) \equiv f(\mathbf{x})/(4\mu)$, Newton's method on g updates as follows:

$$\begin{aligned}\mathbf{x}_{t+1} &= \mathbf{x}_t - \frac{1}{1+\lambda_g(\mathbf{x}_t)}(\nabla^2 g(\mathbf{x}_t))^{-1}\nabla g(\mathbf{x}_t) \\ &= \mathbf{x}_t - \frac{2\sqrt{\mu}}{2\sqrt{\mu}+\lambda_f(\mathbf{x}_t)}(\nabla^2 f(\mathbf{x}_t))^{-1}\nabla f(\mathbf{x}_t) \\ &= \mathbf{x}_t - \frac{2\sqrt{\mu}}{2\sqrt{\mu}+\lambda_f(\mathbf{x}_t)}\left(\frac{1}{n}\mathbf{A}^\top\mathbf{D}\mathbf{A} + \mu\mathbb{I}_t\right)^{-1}\left(\frac{1}{n}\sum_{i=1}^{n} y_i p_i \mathbf{a}_i + \mu\mathbf{x}_t\right)\end{aligned}$$

where $\lambda_g(\mathbf{x}) = \sqrt{\nabla g(\mathbf{x})^\top(\nabla^2 g(\mathbf{x}))^{-1}\nabla g(\mathbf{x})}$ and $\lambda_f(\mathbf{x}) = \sqrt{\nabla f(\mathbf{x})^\top(\nabla^2 f(\mathbf{x}))^{-1}\nabla f(\mathbf{x})}$. To emulate the above update, we need to approximate the following components:

1. The values of the diagonal entries of the matrix $\mathbf{D}$, resulting an error vector $\mathbf{u}_1 \in \mathbb{R}^n$.
2. The multiplication of $\mathbf{DA}$, which incurs an error $\mathbf{U}_2 \in \mathbb{R}^{n\times d}$.
3. The inversion of the Hessian matrix, which incurs an error $\mathbf{E}_2 \in \mathbb{R}^{d\times d}$.
4. The values of $\mathbf{p} = (p_1, \ldots, p_n)^\top \in \mathbb{R}^n$, which incurs an error $\mathbf{u}_3 \in \mathbb{R}^n$.
5. The step-size $2\sqrt{\mu}/(2\sqrt{\mu} + \lambda_f(\mathbf{x}_t))$, which incurs an error ϵ_4. Notice that to do this, we need to first approximate the value of $\lambda_f(\mathbf{x}_t)$.

For simplicity, we drop the subscript f from λ_f, and we just write λ from now on. The resulting update for one step admits the following form:

$$\mathbf{x}_1 = \mathbf{x}_0 - \left(\frac{2\sqrt{\mu}}{2\sqrt{\mu} + \lambda(\mathbf{x}_0)} + \epsilon_4\right)\left[\left(\frac{1}{n}\mathbf{A}^\top\hat{\mathbf{D}}\mathbf{A} + \mu\mathbb{I} + \mathbf{E}_1\right)^{-1} + \mathbf{E}_2\right]\left(\frac{1}{n}\sum_{i=1}^n y_i(\hat{p}_i - u_{3,i})\mathbf{a}_i + \mu\mathbf{x}_0\right) \tag{C.1}$$

where $\mathbf{E}_1 = \mathbf{A}^\top \mathrm{diag}(\mathbf{u}_1)\mathbf{A} + \mathbf{A}^\top\mathbf{U}_2$.

Input format. We consider the following input format:

$$\mathbf{H}_0 = \begin{pmatrix} [\mathbb{I}_d\ ] \\ [\mathbb{I}_d\ ] \\ [\mathbb{I}_d\ ] \\ \\ \mathbf{A}^\top \\ \mathbf{y}^\top \\ \mathbf{x}_0\mathbb{1}_n^\top \\ \frac{1}{n}\mathbf{e}_1^\top \\ \\ \mathbb{1}_n^\top \end{pmatrix} \in \mathbb{R}^{(7d+5)\times n}. \tag{C.2}$$

Here the first identity matrix will be used to store the updates for calculating the inverse, the second one for the initialization, while the third one for required computation. The initialization $\mathbf{x}_0$ is copied n times The second to last line is used to store the parameter η, the step-size. Below we will provide explicit construction of Transformers that can realize the following map from input to output:

$$\mathbf{H}_0 \mapsto \mathbf{H}_K = \begin{pmatrix} [\mathbb{I}_d\ ] \\ [\mathbb{I}_d\ ] \\ [\mathbb{I}_d\ ] \\ \\ \mathbf{A}^\top \\ \mathbf{y}^\top \\ \mathbf{x}_1\mathbb{1}_n^\top \\ \frac{1}{n}\mathbf{e}_1^\top \\ \\ \mathbb{1}_n^\top \end{pmatrix} \in \mathbb{R}^{(7d+5)\times n} \tag{C.3}$$

where K denotes the number of Transformer layers.

Theorem 4.2. *Under the setting of Theorem 4.1, there exists a Transformer consisting of linear attention with ReLU layers that can approximately perform damped Newton's method on the regularized logistic loss as follows*

$$\mathbf{x}_{t+1} = \mathbf{x}_t - \frac{2\sqrt{\mu}}{2\sqrt{\mu} + \lambda(\mathbf{x}_t)}(\nabla^2 f(\mathbf{x}_t))^{-1}\nabla f(\mathbf{x}_t) + \varepsilon$$

where ε is an error term. For any $\epsilon > 0$, to achieve that $\|\varepsilon\|_2 \le \epsilon$, the width of such a Transformer can be bounded by $O(d\frac{(1+\mu)^8}{\epsilon^4\mu^{10}})$, and its depth can be bounded by $11 + 2k$ where $k \le 2\log\kappa_f + \log\log\frac{(1+\mu)^3}{\epsilon^2\mu^2}$.

For clarity, we split the proof into two parts: the first part is to construct the approximate updates of the Newton's algorithm, and the second part is devoted to the error analysis.

Proof of Theorem 4.2: Construction of the approximate updates. For ease of presentation, in the following proof we omit the explicit expressions of the weights and describe instead the operations they induce. The constructions are straightforward to be determined, similar to the proofs in Appendix A. Notice that each of the value, key and query weight matrices can always make any row selection by zero-padding and ignoring the rest of the rows of the input matrix.

Step 1 - Create d. We start by creating the diagonal entries of $\mathbf{D}$. We choose $\mathbf{W}_V, \mathbf{W}_K, \mathbf{W}_Q$ such that

$$\mathbf{W}_V\mathbf{H}_0 = \begin{pmatrix} \mathbf{e}_1^\top \end{pmatrix}, \quad \mathbf{W}_K\mathbf{H}_0 = \mathbf{x}_0\mathbb{1}_n^\top, \quad \mathbf{W}_Q\mathbf{H}_0 = \mathbf{A}^\top, \tag{C.4}$$

Note that here the value weight matrix $\mathbf{W}_V$ just picks the first row of the identity matrix, zeroes out any other row, and performs a row permutation to place the result in the second to last line. Then we have

$$\mathbf{H}_1^{\texttt{Attn}} = \mathbf{H}_0 + \begin{pmatrix} \mathbf{x}_0^\top\mathbf{A}^\top \end{pmatrix} = \begin{pmatrix} [\mathbb{I}_d\ ] \\ [\mathbb{I}_d\ ] \\ [\mathbb{I}_d\ ] \\ \\ \mathbf{A}^\top \\ \mathbf{y}^\top \\ \mathbf{x}_0\mathbb{1}_n^\top \\ \mathbf{e}_1^\top/n \\ \mathbf{x}_0^\top\mathbf{A}^\top \\ \mathbb{1}_n^\top \end{pmatrix} \tag{C.5}$$

we then use the ReLU network to approximate the values $d_i = D_{ii} = p(\mathbf{x}_0^\top\mathbf{a}_i, y_i)(1 - p(\mathbf{x}_0^\top\mathbf{a}_i, y_i))$ where $p(x, y) = \exp(-yx)/(1+\exp(-yx))$. We zero out all the rows except for the rows of $\mathbf{y}^\top, \mathbf{x}_0^\top\mathbf{A}^\top$ and construct the weights of the ReLU network to approximate the function $p(x, y)(1 - p(x, y))$. Notice that this function takes values between $[0, 1]$ and it domain contains $x \in \mathbb{R}$ and $y \in \{-1, 1\}$ in this case. Also note that it is symmetric with respect to the value y. Therefore, it suffices to approximate the function $f(x) = e^x/(1+e^x)^2$, which is increasing for $x \leq 0$ and decreasing for $x \geq 0$. We approximate it by splitting $[0, 1]$ into $1/\epsilon$ intervals and deal with each of these intervals seperately for $x \geq 0$ and $x \leq 0$ using a total number of $4/\epsilon$ ReLU neurons. This gives rise to

$$\mathbf{H}_1 = \begin{pmatrix} [\mathbb{I}_d\ ] \\ [\mathbb{I}_d\ ] \\ [\mathbb{I}_d\ ] \\ \\ \mathbf{A}^\top \\ \mathbf{y}^\top \\ \mathbf{x}_0\mathbb{1}_n^\top \\ \mathbf{e}_1^\top/n \\ \hat{\mathbf{d}}^\top \\ \mathbb{1}_n^\top \end{pmatrix}$$

where $\hat{\mathbf{d}} = \mathbf{d} + \mathbf{u}_1$. Notice that

$$\|\mathbf{u}_1\| \leq \frac{4}{N} \tag{C.6}$$

where N is the width of the ReLU layer.

Step 2 - Approximate $\frac{1}{n}\mathbf{D}\odot\mathbf{A}^\top$. In the attention of the second layer, we compute $\hat{\mathbf{d}}/n$ by setting

$$\mathbf{W}_V^{(1)}\mathbf{H}_1=\begin{pmatrix}\mathbf{e}_1^\top\end{pmatrix},\quad \mathbf{W}_K^{(1)}\mathbf{H}_1=\frac{1}{n}\mathbf{e}_1^\top,\quad \mathbf{W}_Q^{(1)}\mathbf{H}_1=\hat{\mathbf{d}}, \tag{C.7}$$

and we use one more head to subtract the residual by letting

$$\mathbf{W}_V^{(2)}\mathbf{H}_1=\begin{pmatrix}-\mathbf{e}_1^\top\end{pmatrix},\quad \mathbf{W}_K^{(2)}\mathbf{H}_1=\mathbf{e}_1^\top,\quad \mathbf{W}_Q^{(2)}\mathbf{H}_1=\hat{\mathbf{d}}. \tag{C.8}$$

Thus,

$$\mathbf{H}_2^{\texttt{Attn}}=\mathbf{H}_1+\begin{pmatrix}\hat{\mathbf{d}}/n\end{pmatrix}-\begin{pmatrix}\hat{\mathbf{d}}\end{pmatrix}=\begin{pmatrix}[\mathbb{I}_d\,]\\ [\mathbb{I}_d\,]\\ [\mathbb{I}_d\,]\\ \\ \mathbf{A}^\top\\ \mathbf{y}^\top\\ \mathbf{x}_0\mathbb{1}_n^\top\\ \mathbf{e}_1^\top/n\\ \hat{\mathbf{d}}/n\\ \mathbb{1}_n^\top\end{pmatrix}.$$

The next ReLU layer approximates the multiplication of the diagonal matrix $\frac{1}{n}\hat{\mathbf{D}}$ with the matrix $\mathbf{A}$, which is the same as creating the vectors $\hat{d}_1\mathbf{a}_1/n,\hat{d}_2\mathbf{a}_2/n,\dots\hat{d}_n\mathbf{a}_n/n$, or equivalently, the matrix $\frac{1}{n}\hat{\mathbf{D}}\odot\mathbf{A}^\top$. This can be implemented with one ReLU layer since the elements will be processed serially and the information needed is on the same column. This approximation incurs an error matrix $\mathbf{U}_2$. This yields the output of the second Transformer layer:

$$\mathbf{H}_2=\begin{pmatrix}\mathbb{1}_d\hat{\mathbf{d}}^\top\odot\mathbf{A}^\top/n+\mathbf{U}_2\\ [\mathbb{I}_d\,]\\ [\mathbb{I}_d\,]\\ \\ \mathbf{A}^\top\\ \mathbf{y}^\top\\ \mathbf{x}_0\mathbb{1}_n^\top\\ \mathbf{e}_1^\top/n\\ \top\\ n\\ \mathbb{1}_n^\top\end{pmatrix}. \tag{C.9}$$

Notice that we can easily subtract the identity matrix since we have another copy of it. Using Proposition A.1 in Bai et al. [2023], we have that $s=2$ and each $\hat{d}_i\in[-0.1,1.1]$ thus it requires a total number of $O(1/(n^2\epsilon^2))$ ReLUs to achieve ϵ accuracy for each entry. Moreover,

$$\mathbb{1}_d\frac{\hat{\mathbf{d}}^\top\odot\mathbf{A}^\top}{n}+\mathbf{U}_2=\mathbb{1}_d\frac{\mathbf{d}^\top\odot\mathbf{A}^\top}{n}+\mathbb{1}_d\frac{\mathbf{u}_1^\top\odot\mathbf{A}^\top}{n}+\mathbf{U}_2,$$

and thus

$$\|\mathbf{U}_2\|_2\le\|\mathbf{U}_2\|_F\le\widetilde{\mathcal{O}}(\sqrt{dn}\epsilon)=\widetilde{\mathcal{O}}\left(\frac{\sqrt{d}}{\sqrt{nN}}\right) \tag{C.10}$$

where N is the width of the ReLU layer and we omit logarithmic factors in $\widetilde{\mathcal{O}}(\cdot)$.

Step 3 - Create the matrix $\frac{1}{n}\mathbf{A}^\top\mathbf{D}\mathbf{A}+\mu\mathbb{I}$. For this step, we first use the attention layer to implement

$$\mathbf{W}_V^{(1)}\mathbf{H}_2=\begin{pmatrix}\alpha\mathbf{A}^\top\\ \mathbf{A}^\top\end{pmatrix},\quad \mathbf{W}_K^{(1)}\mathbf{H}_2=\frac{1}{n}\mathbb{1}_d^\top\hat{\mathbf{d}}\odot\mathbf{A}^\top+\mathbf{U}_2,\quad \mathbf{W}_Q^{(1)}\mathbf{H}_2=[\mathbb{I}_d\,].$$

Notice that $(\mathbf{W}_K^{(1)}\mathbf{H}_2)^\top=\frac{1}{n}\hat{\mathbf{D}}\mathbf{A}+\mathbf{U}_2^\top$. We also use two extra heads such that

$$\mathbf{W}_V^{(2)}\mathbf{H}_2=\begin{pmatrix}\alpha\mu[\mathbb{I}_d\,]\\ (\mu-1)[\mathbb{I}_d\,]\end{pmatrix},\quad \mathbf{W}_K^{(2)}\mathbf{H}_2=[\mathbb{I}_d\,],\quad \mathbf{W}_Q^{(2)}\mathbf{H}_2=[\mathbb{I}_d\,],$$
$$\mathbf{W}_V^{(3)}\mathbf{H}_2=\begin{pmatrix}[\mathbb{I}_d\,]\\ \end{pmatrix},\quad \mathbf{W}_K^{(3)}\mathbf{H}_2=[\mathbb{I}_d\,],\quad \mathbf{W}_Q^{(3)}\mathbf{H}_2=-\mathbb{1}_d\hat{\mathbf{d}}^\top\odot\mathbf{A}^\top/n-\mathbf{U}_2.$$

Combining these three heads, we get

$$\begin{aligned}\mathbf{H}_3&=\mathbf{H}_2+\begin{pmatrix}[\alpha(\frac{1}{n}\mathbf{A}^\top\hat{\mathbf{D}}\mathbf{A}+\mathbf{A}^\top\mathbf{U}_2^\top)\,]\\ [\frac{1}{n}\mathbf{A}^\top\hat{\mathbf{D}}\mathbf{A}+\mathbf{A}^\top\mathbf{U}_2^\top\,]\\ \end{pmatrix}+\begin{pmatrix}[\alpha\mu\mathbb{I}_d\,]\\ [(\mu-1)\mathbb{I}_d\,]\end{pmatrix}+\begin{pmatrix}-\frac{\mathbb{1}_d}{n}\hat{\mathbf{d}}^\top\odot\mathbf{A}^\top-\mathbf{U}_2\\ \end{pmatrix}\\ &=\begin{pmatrix}[\alpha(\frac{1}{n}\mathbf{A}^\top\hat{\mathbf{D}}\mathbf{A}+\mathbf{A}^\top\mathbf{U}_2^\top+\mu\mathbb{I}_d);]\\ [\frac{1}{n}\mathbf{A}^\top\hat{\mathbf{D}}\mathbf{A}+\mathbf{A}^\top\mathbf{U}_2^\top+\mu\mathbb{I}_d;]\\ [\mathbb{I}_d\,]\\ \\ \mathbf{A}^\top\\ \mathbf{y}^\top\\ \mathbf{x}_0\mathbb{1}_n^\top\\ \mathbf{e}_1^\top/n\\ \overset{\top}{n}\\ \mathbb{1}_n^\top\end{pmatrix}.\end{aligned}$$

For simplicity, we denote $\mathbf{B}:=\frac{1}{n}\mathbf{A}^\top\hat{\mathbf{D}}\mathbf{A}+\mathbf{A}^\top\mathbf{U}_2^\top+\mu\mathbb{I}_d$. We then use one more attention layer with two head as follows: For the first head,

$$\mathbf{W}_V^{(1)}\mathbf{H}_3=\begin{pmatrix}[\mathbb{I}_d\,]\\ \end{pmatrix},\quad \mathbf{W}_K^{(1)}\mathbf{H}_3=\alpha\mathbf{B},\quad \mathbf{W}_Q^{(1)}\mathbf{H}_3=[\mathbb{I}_d\,].\tag{C.11}$$

The second head is used to subtract the residual, so that the output of this layer is

$$\mathbf{H}_4=\begin{pmatrix}[\alpha\mathbf{B}^\top;]\\ [\mathbf{B};]\\ [\mathbb{I}_d\,]\\ \\ \mathbf{A}^\top\\ \mathbf{y}^\top\\ \mathbf{x}_0\mathbb{1}_n^\top\\ \mathbf{e}_1^\top/n\\ \overset{\top}{n}\\ \mathbb{1}_n^\top\end{pmatrix}.$$

We keep track of the errors in the following way:

$$\mathbf{B}=\frac{1}{n}\mathbf{A}^\top\mathbf{D}\mathbf{A}+\mu\mathbb{I}+\frac{1}{n}\mathbf{A}^\top\mathrm{diag}(\mathbf{u}_1)\mathbf{A}+\mathbf{A}^\top\mathbf{U}_2=\frac{1}{n}\mathbf{A}^\top\mathbf{D}\mathbf{A}+\mu\mathbb{I}+\mathbf{E}_1\tag{C.12}$$

where $\mathbf{E}_1$ can be controlled as

$$\begin{aligned}\|\mathbf{E}_1\|_2 &\leq \frac{1}{n}\|\mathrm{diag}(\mathbf{u}_1)\|_2 \|\mathbf{A}\|_2^2 + \|\mathbf{A}\|_2 \|\mathbf{U}_2\|_2 \\ &\leq \frac{1}{n}\|\mathbf{u}_1\|_2 \|\mathbf{A}\|_F^2 + \|\mathbf{A}\|_F \|\mathbf{U}_2\|_2 \\ &\leq \|\mathbf{u}_1\|_2 + \sqrt{n}\|\mathbf{U}_2\|_2\end{aligned} \tag{C.13}$$

since $\|\mathbf{a}_i\|^2 \leq 1$ for all $i = 1, \ldots, n$ by Assumption 3.2.

Step 4 - Invert the matrix. Next, we implement Newton's iteration as in the previous section for k steps and we get

$$\mathbf{H}_{2k+4} = \begin{pmatrix} [\mathbf{B}^{-1} + \mathbf{E}_2\,] \\ [\mathbb{I}_d\,] \\ [\mathbb{I}_d\,] \\ \\ \mathbf{A}^\top \\ \mathbf{y}^\top \\ \mathbf{x}_0\mathbb{1}_n^\top \\ \mathbf{e}_1^\top/n \\ {}^\top\!/_n \\ \mathbb{1}_n^\top \end{pmatrix}$$

Step 5 - Create the $\{p_i\}_{i=1}^n$. We repeat Step 1 to get

$$\mathbf{H}_{2k+5}^{\texttt{Attn}} = \begin{pmatrix} [\mathbf{B}^{-1} + \mathbf{E}_2\,] \\ [\mathbb{I}_d\,] \\ [\mathbb{I}_d\,] \\ \\ \mathbf{A}^\top \\ \mathbf{y}^\top \\ \mathbf{x}_0\mathbb{1}_n^\top \\ \mathbf{e}_1^\top/n \\ \mathbf{x}_0^\top\mathbf{A}^\top \\ \mathbb{1}_n^\top \end{pmatrix}. \tag{C.14}$$

We then use the ReLU layer to approximate $\{p_i\}_{i=1}^n$ and we have

$$\mathbf{H}_{2k+5} = \begin{pmatrix} [\mathbf{B}^{-1} + \mathbf{E}_2\,] \\ [\mathbb{I}_d\,] \\ [\mathbb{I}_d\,] \\ \\ \mathbf{A}^\top \\ \mathbf{y}^\top \\ \mathbf{x}_0\mathbb{1}_n^\top \\ \mathbf{e}_1^\top/n \\ \mathbf{p}^\top + \mathbf{u}_3^\top \\ \mathbb{1}_n^\top \end{pmatrix}. \tag{C.15}$$

The function approximation here is the same as the one in Step 1, and it requires $2/\epsilon$ ReLUs to achieve ϵ accuracy. Then

$$\|\mathbf{u}_3\| \leq \frac{2}{N} \tag{C.16}$$

where N is the width of the ReLU layer.

Step 6 - Calculate the values $\{y_i p_i/n\}_{i=1}^n$. In the attention layer, we multiply each p_i with $1/n$ as we did in Step 2, and we have

$$\mathbf{H}_{2k+6}^{\texttt{Attn}} = \begin{pmatrix} [\mathbf{B}^{-1} + \mathbf{E}_2\ ] \\ [\mathbb{I}_d\ ] \\ [\mathbb{I}_d\ ] \\ \\ \mathbf{A}^\top \\ \mathbf{y}^\top \\ \mathbf{x}_0 \mathbb{1}_n^\top \\ \mathbf{e}_1^\top/n \\ (\mathbf{p}^\top + \mathbf{u}_3^\top)/n \\ \mathbb{1}_n^\top \end{pmatrix}.$$

We then use the ReLU layer to approximate the inner product between $\frac{1}{n}\mathbf{p}$ and $\mathbf{y}$. To do so, notice that each $p_i \in (0,1)$, and $y = \{-1, 1\}$ and consider the following sets of ReLUs:

$$o_1 = \sigma\Big(\frac{1}{2}x + 2y\Big) - \sigma\Big(-\frac{1}{2}x + 2y\Big), \quad o_2 = \sigma\Big(-\frac{1}{2}x - 2y\Big) - \sigma\Big(+\frac{1}{2}x - 2y\Big)$$

Suppose $x \in (0,1)$, then if $y = 1$, the outputs are $o_1 = x$, $o_2 = 0$, and for $y = -1$, the outputs are $o_1 = 0$ and $o_2 = -x$, thus $o_1 + o_2 = xy$. Consequently,

$$\mathbf{H}_{2k+6} = \begin{pmatrix} [\mathbf{B}^{-1} + \mathbf{E}_2\ ] \\ [\mathbb{I}_d\ ] \\ [\mathbb{I}_d\ ] \\ \\ \mathbf{A}^\top \\ \mathbf{y}^\top \\ \mathbf{x}_0 \mathbb{1}_n^\top \\ \mathbf{e}_1^\top/n \\ (\mathbf{p}^\top + \mathbf{u}_3^\top) \odot \mathbf{y}^\top/n \\ \mathbb{1}_n^\top \end{pmatrix}$$

Step 7 - Calculate the gradient. Next, we want to calculate the quantity $-\frac{1}{n}\sum_{i=1}^n y_i p_i \mathbf{a}_i + \mu \mathbf{x}_0$. We first set the weight matrices of the attention layer such that

$$\mathbf{W}_V^{(1)}\mathbf{H}_{2k+6} = \begin{pmatrix} -\mathbf{A}^\top \end{pmatrix}, \quad \mathbf{W}_K^{(1)}\mathbf{H}_{2k+6} = \frac{1}{n}(\mathbf{p}^\top + \mathbf{u}_4^\top) \odot \mathbf{y}^\top, \quad \mathbf{W}_Q^{(1)}\mathbf{H}_{2k+6} = \mathbf{e}_1^\top$$

$$\mathbf{W}_V^{(2)}\mathbf{H}_{2k+6} = \begin{pmatrix} -[\mathbb{I}_d\ ] \end{pmatrix}, \quad \mathbf{W}_K^{(2)}\mathbf{H}_{2k+6} = [\mathbb{I}_d\ ], \quad \mathbf{W}_Q^{(2)}\mathbf{H}_{2k+6} = [\mathbb{I}_d\ ],$$

and we need a third head to add $\mu\mathbf{x}_0$ by setting

$$\mathbf{W}_V^{(3)}\mathbf{H}_{2k+6} = \begin{pmatrix} \mathbf{x}_0 \mathbb{1}_n^\top \end{pmatrix}, \quad \mathbf{W}_K^{(3)}\mathbf{H}_{2k+6} = \mathbf{e}_1, \quad \mathbf{W}_Q^{(3)}\mathbf{H}_{2k+6} = \mu\mathbf{e}_1, \tag{C.17}$$

Thus, we place the result in the second block of the matrix and we have

$$\mathbf{H}_{2k+7} = \begin{pmatrix} [\mathbf{B}^{-1} + \mathbf{E}_2\ ] \\ [\mathbf{b}\ ] \\ [\mathbb{I}_d\ ] \\ \\ \mathbf{A}^\top \\ \mathbf{y}^\top \\ \mathbf{x}_0 \mathbb{1}_n^\top \\ \mathbf{e}_1^\top / n \\ (\mathbf{p}^\top + \mathbf{u}_3^\top) \odot \mathbf{y}^\top / n \\ \mathbb{1}_n^\top \end{pmatrix} \tag{C.18}$$

where $\mathbf{E}_2$ is the error incurred by running Newton's method for the inversion of the matrix and $\mathbf{b} = -\frac{1}{n}\sum_{i=1}^n (y_i p_i + y_i u_{3i})\mathbf{a}_i + \mu \mathbf{x}_0 = -\frac{1}{n}\sum_{i=1}^n y_i p_i \mathbf{a}_i + \mu \mathbf{x}_0 + \boldsymbol{\epsilon}_3$, $\boldsymbol{\epsilon}_3 = -\frac{1}{n}\sum_{i=1}^n y_i u_{3i} \mathbf{a}_i$. Thus,

$$\|\boldsymbol{\epsilon}_3\|_2 \leq \frac{2}{N} \tag{C.19}$$

where is the width of the ReLU layer used to approximate the error $\mathbf{u}_3$ (Equation (C.16)).

Step 8 - Calculate the stepsize. We proceed to approximate $\hat{\lambda}(\mathbf{x})^2 = \nabla^\top f(\mathbf{x})(\nabla^2 f(\mathbf{x}))^{-1}\nabla f(\mathbf{x}) = \mathbf{b}^\top(\mathbf{B}^{-1} + \mathbf{E}_2)\mathbf{b}$. We first create the quantity $(\mathbf{B}^{-1} + \mathbf{E}_2)\mathbf{b}$ which we also need for the update, store it, and then calculate in the next layer the parameter $\lambda(\mathbf{x})^2$. For the first layer we set

$$\mathbf{W}_V^{(1)}\mathbf{H}_{2k+7} = \begin{pmatrix} [\mathbf{B}^{-1} + \mathbf{E}_2\ ] \\ \\ \end{pmatrix}, \quad \mathbf{W}_K^{(1)}\mathbf{H}_{2k+7} = [\mathbb{I}_d\ ], \quad \mathbf{W}_Q^{(1)}\mathbf{H}_{2k+7} = [\mathbf{b}\ ],$$
$$\mathbf{W}_V^{(2)}\mathbf{H}_{2k+7} = -\begin{pmatrix} [\mathbf{B}^{-1} + \mathbf{U}_3\ ] \\ \\ \end{pmatrix}, \quad \mathbf{W}_K^{(2)}\mathbf{H}_{2k+7} = [\mathbb{I}_d\ ], \quad \mathbf{W}_Q^{(2)}\mathbf{H}_{2k+7} = [\mathbb{I}_d\ ].$$

Then we get

$$\mathbf{H}_{2k+8} = \begin{pmatrix} [(\mathbf{B}^{-1} + \mathbf{E}_2)\mathbf{b}\ ] \\ [\mathbf{b}\ ] \\ [\mathbb{I}_d\ ] \\ \\ \mathbf{A}^\top \\ \mathbf{y}^\top \\ \mathbf{x}_0 \mathbb{1}_n^\top \\ \mathbf{e}_1^\top / n \\ (\mathbf{p}^\top + \mathbf{u}_3^\top) \odot \mathbf{y}^\top / n \\ \mathbb{1}_n^\top \end{pmatrix}.$$

We use another Transformer layer to calculate the quantity $\lambda(\hat{\mathbf{x}}_t)^2$. We set the attention layer as follows

$$\mathbf{W}_V^{(1)}\mathbf{H}_{2k+8} = \begin{pmatrix} \mathbf{e}_1^\top \\ \\ \end{pmatrix}, \quad \mathbf{W}_K^{(1)}\mathbf{H}_{2k+8} = [\mathbf{b}\ ], \quad \mathbf{W}_Q^{(1)}\mathbf{H}_{2k+8} = [(\mathbf{B}^{-1} + \mathbf{E}_2)\mathbf{b}\ ]$$

where we place the $\mathbf{e}_1^\top$ in the second to last position. We also use as before an extra head to remove the residual. Then we get

$$\mathbf{H}_{2k+9}^{\texttt{Attn}} = \begin{pmatrix} [(\mathbf{B}^{-1}+\mathbf{E}_2)\mathbf{b}\ ] \\ [\mathbf{b}\ ] \\ [\mathbb{I}_d\ ] \\ \\ \mathbf{A}^\top \\ \mathbf{y}^\top \\ \mathbf{x}_0\mathbb{1}_n^\top \\ \mathbf{e}_1^\top/n \\ [\hat{\lambda}(\hat{\mathbf{x}}_t)^2\ ] \\ \mathbb{1}_n^\top \end{pmatrix}.$$

In the ReLU layer, we approximate the function $\frac{2\sqrt{\mu}}{2\sqrt{\mu}+\sqrt{x}}$. Notice that this function take values in $(0,1]$. The first derivative of the function is $-\frac{2\sqrt{\mu}}{2(2\sqrt{\mu}+\sqrt{x})^2\sqrt{x}} < 0$, so it is a monotonically decreasing function. Thus, using the same argument with previous steps we can approximate it up to error ϵ with $2/\epsilon$ ReLUs. This yields an error ϵ_4

$$|\epsilon_4| \leq \frac{2}{N} \tag{C.20}$$

where N is the width of the ReLU layer and we can write the stepsize as $\hat{\eta}(\mathbf{x}_0) = 2\sqrt{\mu}/(2\sqrt{\mu}+\lambda(\mathbf{x}_0)) + \epsilon_4$. Thus,

$$\mathbf{H}_{2k+9} = \begin{pmatrix} [(\mathbf{B}^{-1}+\mathbf{E}_2)\mathbf{b}\ ] \\ [\mathbf{b}\ ] \\ [\mathbb{I}_d\ ] \\ \\ \mathbf{A}^\top \\ \mathbf{y}^\top \\ \hat{\mathbf{x}}_t\mathbb{1}_n^\top \\ \mathbf{e}_1^\top/n \\ [\hat{\eta}(\mathbf{x}_0)\ *] \\ \mathbb{1}_n^\top \end{pmatrix}$$

where $*$ denotes inconsequential values.

Step 9 - Update For the last step, we use two more attention layers. We set the weights for the first layer such that

$$\mathbf{W}_V^{(1)}\mathbf{H}_{2k+9} = \begin{pmatrix} [\hat{\eta}(\mathbf{x}_0)\ *] \end{pmatrix}, \quad \mathbf{W}_K^{(1)}\mathbf{H}_{2k+9} = [(\mathbf{B}^{-1}+\mathbf{E}_2)\mathbf{b}\ ], \quad \mathbf{W}_Q^{(1)}\mathbf{H}_{2k+9} = [\mathbb{I}_d\ ].$$

We also use an extra head to remove the residual by setting

$$\mathbf{W}_V^{(2)}\mathbf{H}_{2k+9} = -\begin{pmatrix} [\hat{\eta}(\mathbf{x}_0)\ *] \end{pmatrix}, \quad \mathbf{W}_K^{(2)}\mathbf{H}_{2k+9} = [\mathbb{I}_d\ ], \quad \mathbf{W}_Q^{(2)}\mathbf{H}_{2k+9} = [\mathbb{I}_d\ ], \tag{C.21}$$

where the values $[\hat{\eta}(\mathbf{x}_0)\ *]$ are placed in the second to last row. Combining these two heads, we get

$$\mathbf{H}_{2k+10} = \mathbf{H}_{2k+9} + \begin{pmatrix} [\hat{\eta}(\mathbf{x}_0)\mathbf{b}^\top(\mathbf{B}^{-1}+\mathbf{E}_2)^\top\ ] \end{pmatrix} - \begin{pmatrix} [\hat{\eta}(\mathbf{x}_0)\ *] \end{pmatrix}$$

$$= \begin{pmatrix} [(\mathbf{B}^{-1}+\mathbf{E}_2)\mathbf{b}\ ] \\ [\mathbf{b}\ ] \\ [\mathbb{I}_d\ ] \\ \\ \mathbf{A}^\top \\ \mathbf{y}^\top \\ \mathbf{x}_0\mathbb{1}_n^\top \\ \mathbf{e}_1^\top/n \\ [\hat{\eta}(\mathbf{x}_0)\mathbf{b}^\top(\mathbf{B}^{-1}+\mathbf{E}_2)^\top\ *] \\ \mathbb{1}_n^\top \end{pmatrix}$$

where $*$ are again inconsequential values. Notice that when subtracting the residual with the second head, we correct only the first d columns.

Finally, we perform the update with one more attention layer:

$$\mathbf{W}_V^{(1)}\mathbf{H}_{2k+10} = \begin{pmatrix} -[\mathbb{I}_d\ ] \end{pmatrix}, \quad \mathbf{W}_K^{(1)}\mathbf{H}_{2k+10} = [\hat{\eta}(\mathbf{x}_0)\mathbf{b}^\top(\mathbf{B}^{-1}+\mathbf{E}_2)^\top\ *], \quad \mathbf{W}_Q^{(1)}\mathbf{H}_{2k+10} = \mathbb{1}_n^\top.$$

Another head is used to restore the matrix in its initial form:

$$\mathbf{W}_V^{(2)}\mathbf{H}_{2k+10} = \begin{pmatrix} [-\mathbb{I}_d\ \ \mathbb{I}_d\ ] \\ [;-\mathbb{I}_d\ \mathbb{I}_d\ ] \end{pmatrix}\mathbf{H}_{2k+10}, \quad \mathbf{W}_K^{(2)}\mathbf{H}_{2k+10} = [\mathbb{I}_d\ ], \quad \mathbf{W}_Q^{(2)}\mathbf{H}_{2k+10} = [\mathbb{I}_d\ ].$$

As a result, we get

$$\mathbf{H}_{2k+11}^{\texttt{Attn}} = \begin{pmatrix} [\mathbb{I}_d\ ] \\ [\mathbb{I}_d\ ] \\ [\mathbb{I}_d\ ] \\ \\ \mathbf{A}^\top \\ \mathbf{y}^\top \\ (\hat{\mathbf{x}}_t - \hat{\eta}(\mathbf{x}_0)(\mathbf{B}^{-1}+\mathbf{E}_2)\mathbf{b})\mathbb{1}_n^\top \\ \mathbf{e}_1^\top/n \\ [\hat{\eta}(\mathbf{x}_0)\mathbf{b}^\top(\mathbf{B}^{-1}+\mathbf{E}_2)^\top\ *] \\ \mathbb{1}_n^\top \end{pmatrix}. \tag{C.22}$$

We further use the ReLUs to zero out the second to last row. Note that the inconsequential values wer create when approximating the function $1/(1+\sqrt{x})$, so they are close to one. Thus, we can use the following simple ReLU to zero out this line:

$$\sigma(-x/2+5y) - \sigma(x/2+5y)$$

where we use as x the elements of the second to last row and y, the last row and all the other connections are zero. The final output is

$$\mathbf{H}_{2k+11} = \begin{pmatrix} [\mathbb{I}_d\ ] \\ [\mathbb{I}_d\ ] \\ [\mathbb{I}_d\ ] \\ \\ \mathbf{A}^\top \\ \mathbf{y}^\top \\ (\mathbf{x}_0 - \hat{\eta}(\mathbf{x}_0)(\mathbf{B}^{-1}+\mathbf{E}_2)\mathbf{b})\mathbb{1}_n^\top \\ \mathbf{e}_1^\top/n \\ \vdots \\ \mathbb{1}_n^\top \end{pmatrix} = \begin{pmatrix} [\mathbb{I}_d\ ] \\ [\mathbb{I}_d\ ] \\ [\mathbb{I}_d\ ] \\ \\ \mathbf{A}^\top \\ \mathbf{y}^\top \\ \mathbf{x}_1\mathbb{1}_n^\top \\ \mathbf{e}_1^\top/n \\ \vdots \\ \mathbb{1}_n^\top \end{pmatrix}.$$

Thus, for the next step the input is in the correct form and we can repeat the steps above for the next iteration. ■

Next, we present the error analysis of the emulated iterates. The target is to show that the updates described in Equation (C.1) can be recasted to the following updates:

$$\hat{\mathbf{x}}_{t+1} = \hat{\mathbf{x}}_t - \eta(\hat{\mathbf{x}}_t)(\nabla^2 f(\hat{\mathbf{x}}_t))^{-1}\nabla f(\hat{\mathbf{x}}_t) + \varepsilon_t$$

where the error term ε_t is controlled to satisfy the considitions of Theorem 4.4.

Proof of Theorem 4.2: Error analysis. For $\mathbf{x}_0$, let C be the constant given by Lemma B.10. From the previous step of weight constructions, we have the following update

$$\mathbf{x}_1 = \mathbf{x}_0 - \left(\frac{2\sqrt{\mu}}{2\sqrt{\mu} + \hat{\lambda}(\mathbf{x}_0)} + \epsilon_4\right)[(\nabla^2 f(\mathbf{x}_0) + \mathbf{E}_1)^{-1} + \mathbf{E}_2](\nabla f(\mathbf{x}_0) + \boldsymbol{\epsilon}_3) \tag{C.23}$$

where

$$\begin{aligned}
\mathbf{E}_1 &= \frac{1}{n}\mathbf{A}^\top \mathrm{diag}(\mathbf{u}_1)\mathbf{A} + \mathbf{A}^\top \mathbf{U}_2 \\
\mathbf{E}_2 &= \text{Error of Newton's method for inversion} \\
\boldsymbol{\epsilon}_3 &= -\frac{1}{n}\sum_{i=1}^n y_i u_{4i}\mathbf{a}_i \\
\epsilon_4 &= \text{Approximation error for the quantity } \lambda.
\end{aligned}$$

where the error terms admit the following bounds

$$\|\mathbf{u}_1\| \le \frac{4}{N}, \quad \|\mathbf{U}_2\|_F \le \widetilde{\mathcal{O}}\left(\frac{\sqrt{d}}{\sqrt{nN}}\right), \quad \|\mathbf{E}_2\| \le \epsilon_2, \quad \|\boldsymbol{\epsilon}_3\| \le \frac{2}{N}, \quad |\epsilon_4| \le \frac{2}{N} \tag{C.24}$$

where N is the width. These bounds were derived in Equations (C.6), (C.10), (C.16) and (C.20) for $\mathbf{u}_1, \mathbf{U}_2$ $\boldsymbol{\epsilon}_3$ and ϵ_4 respectively. The error term ϵ_2 is controlled by the number of layers, for $k = c + 2\log\log(1/\epsilon)$ layers we achieve error less than ϵ.

Applying Corollary E.2, we have $(\nabla^2 f(\mathbf{x}_0) + \mathbf{E}_1)^{-1} = \nabla^2 f(\mathbf{x}_0)^{-1} + \mathbf{E}_1'$ where $\mathbf{E}_1'$ satisfies $\|\mathbf{E}_1'\|_2 \le \|\mathbf{E}_1\|_2 / (\mu(\mu - \|\mathbf{E}_1\|_2))$. Further writing $\mathbf{E}_2' := \mathbf{E}_1' + \mathbf{E}_2$, then by (C.24), it holds that

$$\|\mathbf{E}_2'\|_2 \le \|\mathbf{E}_2\|_2 + \frac{\|\mathbf{E}_1\|_2}{\mu(\mu - \|\mathbf{E}_1\|_2)}. \tag{C.25}$$

Now we can rewrite (C.23) as

$$\mathbf{x}_1 = \mathbf{x}_0 - \left(\frac{2\sqrt{\mu}}{2\sqrt{\mu} + \hat{\lambda}(\mathbf{x}_0)} + \epsilon_4\right)[(\nabla^2 f(\mathbf{x}_0))^{-1} + \mathbf{E}_2'](\nabla f(\mathbf{x}_0) + \boldsymbol{\epsilon}_3)$$

Next we analyze the error involved in the quantity $\hat{\lambda}^2(\hat{\mathbf{x}}_t)$. Recall that

$$\begin{aligned}
\hat{\lambda}(\mathbf{x}_0)^2 = \mathbf{b}^\top \mathbf{B}\mathbf{b} &= (\nabla f(\mathbf{x}_0) + \boldsymbol{\epsilon}_3)^\top [(\nabla^2 f(\mathbf{x}_0))^{-1} + \mathbf{E}_2'](\nabla f(\mathbf{x}_0) + \boldsymbol{\epsilon}_3) \\
&= \underbrace{\nabla f(\mathbf{x}_0)^\top (\nabla^2 f(\mathbf{x}_0))^{-1}\nabla f(\mathbf{x}_0)}_{\lambda(\mathbf{x}_0)^2} \\
&\quad + \underbrace{2\boldsymbol{\epsilon}_3^\top (\nabla^2 f(\mathbf{x}_0))^{-1}(\nabla f(\mathbf{x}_0) + \boldsymbol{\epsilon}_3) + (\nabla f(\mathbf{x}_0) + \boldsymbol{\epsilon}_3)^\top \mathbf{E}_2'(\nabla f(\mathbf{x}_0) + \boldsymbol{\epsilon}_3)}_{=:\epsilon_4'}
\end{aligned}$$

By triangle inequality and the bounds from Lemma B.8, we can bound ϵ_4' as follows:

$$|\epsilon_4'| < \frac{2(1+\mu C)\left\|\boldsymbol{\epsilon}_3\right\| + \left\|\boldsymbol{\epsilon}_3\right\|^2}{\mu} + \left\|\boldsymbol{\epsilon}_3\right\|^2 \left\|\mathbf{E}_2'\right\|_2 + 2(1+\mu C)\left\|\boldsymbol{\epsilon}_3\right\| \left\|\mathbf{E}_2'\right\|_2 + (1+\mu C)^2 \left\|\mathbf{E}_2'\right\|_2 \tag{C.26}$$

Now, as long as it holds that

$$\left\|\mathbf{E}_2'\right\|_2 = \mathcal{O}\left(\frac{\mu\epsilon_4^2}{(1+C\mu)^2}\right) \text{ and } \left\|\boldsymbol{\epsilon}_3\right\| = \mathcal{O}\left(\frac{\epsilon_4^2\mu^2}{(1+C\mu)}\right) \tag{C.27}$$

we would have

$$\frac{2(1+C\mu)\left\|\boldsymbol{\epsilon}_3\right\| + \left\|\boldsymbol{\epsilon}_3\right\|^2}{\mu} + \left\|\boldsymbol{\epsilon}_3\right\|^2 \left\|\mathbf{E}_2'\right\|_2 + 2(1+C\mu)\left\|\boldsymbol{\epsilon}_3\right\| \left\|\mathbf{E}_2'\right\|_2 + (1+C\mu)^2 \left\|\mathbf{E}_2'\right\|_2 < \frac{4\mu\epsilon_4^2}{1-\epsilon_4}. \tag{C.28}$$

Then by Appendix E.2 we have that if the condition above holds

$$\left|\frac{2\sqrt{\mu}}{2\sqrt{\mu}+\hat{\lambda}(\mathbf{x}_0)} - \frac{2\sqrt{\mu}}{2\sqrt{\mu}+\lambda(\mathbf{x}_0)}\right| \leq \epsilon_4$$

This implies that

$$\frac{2\sqrt{\mu}}{2\sqrt{\mu}+\hat{\lambda}(\mathbf{x}_0)} = \frac{2\sqrt{\mu}}{2\sqrt{\mu}+\lambda(\mathbf{x}_0)} + \tilde{\epsilon}_4$$

for some $\tilde{\epsilon}_4$, such that $|\tilde{\epsilon}_4| \leq \epsilon_4$, since we also account for the approximation error from the ReLUs. Then it follows that

$$\begin{aligned}
\mathbf{x}_1 &= \mathbf{x}_0 - \left(\frac{2\sqrt{\mu}}{2\sqrt{\mu}+\lambda(\mathbf{x}_0)} + \tilde{\epsilon}_4\right)[(\nabla^2 f(\mathbf{x}_0))^{-1} + \mathbf{E}_2'](\nabla f(\mathbf{x}_0) + \boldsymbol{\epsilon}_3) \\
&= \mathbf{x}_0 - \frac{2\sqrt{\mu}}{2\sqrt{\mu}+\lambda(\mathbf{x}_0)}(\nabla^2 f(\mathbf{x}_0))^{-1}\nabla f(\mathbf{x}_0) - \tilde{\epsilon}_4[(\nabla^2 f(\mathbf{x}_0))^{-1} + \mathbf{E}_2'](\nabla f(\mathbf{x}_0) + \boldsymbol{\epsilon}_3) \\
&\quad - \frac{2\sqrt{\mu}}{2\sqrt{\mu}+\lambda(\mathbf{x}_0)}\mathbf{E}_2'(\nabla f(\mathbf{x}_0) + \boldsymbol{\epsilon}_3) \\
&= \mathbf{x}_0 - \frac{2\sqrt{\mu}}{2\sqrt{\mu}+\lambda(\mathbf{x}_0)}(\nabla^2 f(\mathbf{x}_0))^{-1}\nabla f(\mathbf{x}_0) + \varepsilon
\end{aligned}$$

where the error term ε satisfies

$$\left\|\varepsilon\right\|_2 \leq |\tilde{\epsilon}_4|\left(\frac{1}{\mu} + \left\|\mathbf{E}_2'\right\|_2\right)(1 + C\mu + \left\|\boldsymbol{\epsilon}_3\right\|_2) + \left\|\mathbf{E}_2'\right\|_2 (1 + C\mu + \left\|\boldsymbol{\epsilon}_3\right\|). \tag{C.29}$$

Using Equation (C.27) we have that

$$\begin{aligned}
\left\|\varepsilon\right\|_2 &\leq |\tilde{\epsilon}_4|\left(\frac{1}{\mu} + \left\|\mathbf{E}_2'\right\|_2\right)(1 + C\mu + \left\|\boldsymbol{\epsilon}_3\right\|_2) + \left\|\mathbf{E}_2'\right\|_2 (1 + C\mu + \left\|\boldsymbol{\epsilon}_3\right\|) \\
&\leq \frac{2\epsilon_4}{\mu}(1 + C\mu + \left\|\boldsymbol{\epsilon}_3\right\|_2) + (1+\epsilon_4)\left\|\mathbf{E}_2'\right\|_2 (1 + C\mu + \left\|\boldsymbol{\epsilon}_3\right\|_2)
\end{aligned}$$

Notice now if

$$\left\|\mathbf{E}_2'\right\|_2 = \mathcal{O}\left(\frac{\epsilon^2\mu^3}{(1+C\mu)^4}\right) \text{ and } \left\|\boldsymbol{\epsilon}_3\right\|_2 = \mathcal{O}\left(\frac{\epsilon^2\mu^4}{(1+C\mu)^3}\right), \tag{C.30}$$

then

$$\epsilon_4 = \mathcal{O}\left(\frac{\mu\epsilon}{1+C\mu+C\mu^3}\right) = \mathcal{O}\left(\frac{\mu\epsilon}{1+C\mu}\right), \tag{C.31}$$

and consequently, $\|\varepsilon\|_2 \leq \epsilon$.

We will now study how we can bound the error term $\mathbf{E}_2'$. Notice that from Equations (C.13) and (C.25), the following conditions hold:

$$\|\mathbf{E}_1\|_2 \leq \|\mathbf{u}_1\|_2 + \sqrt{n}\,\|\mathbf{U}_2\|_2\,, \quad \|\mathbf{E}_2'\|_2 \leq \|\mathbf{E}_2\|_2 + \frac{\|\mathbf{E}_1\|_2}{\mu(\mu - \|\mathbf{E}_1\|_2)}. \tag{C.32}$$

Now, given that

$$\|\mathbf{E}_1\|_2 = \mathcal{O}\left(\frac{\epsilon^2\mu^5}{(1+C\mu)^4}\right), \quad \|\mathbf{E}_2\|_2 = \mathcal{O}\left(\frac{\epsilon^2\mu^3}{(1+C\mu)^4}\right), \tag{C.33}$$

it is straightforward to verify that indeed $\mathbf{E}_2'$ satisfies Equation (C.27). For the bound on $\mathbf{E}_1$, it suffices to have

$$\|\mathbf{u}_1\|_2 = \mathcal{O}\left(\frac{\epsilon^2\mu^5}{(1+C\mu)^4}\right), \quad \|\mathbf{U}_2\|_2 = \mathcal{O}\left(\frac{\epsilon^2\mu^5}{\sqrt{n}(1+C\mu)^4}\right) \tag{C.34}$$

It remains to determine the necessary width and depth for these error bounds to be achieved. We combine the bounds from Equations (C.24), (C.31), (C.33) and (C.34) to get that

$$\begin{aligned}
\|\mathbf{u}_1\| \sim \frac{1}{N} &\Rightarrow N \sim \frac{(1+C\mu)^4}{\epsilon^2\mu^5} \\
\|\mathbf{U}_2\|_F \sim \frac{\sqrt{d}}{\sqrt{nN}} &\Rightarrow N \sim \frac{d(1+C\mu)^8}{\epsilon^4\mu^{10}} \\
\|\boldsymbol{\epsilon}_3\| \sim \frac{1}{N} &\Rightarrow N \sim \frac{(1+C\mu)^3}{\epsilon^2\mu^4} \\
|\epsilon_4| \sim \frac{1}{N} &\Rightarrow N \sim \frac{1+C\mu}{\epsilon\mu}
\end{aligned}$$

Finally, the error term $\|\mathbf{E}_2\|_2$ is controlled by the depth of the network and scales as $k \sim 2\log\kappa(\frac{1}{n}\mathbf{A}^\top\hat{\mathbf{D}}\mathbf{A} + \mu\mathbb{I} + \mathbf{E}_1) + \log\log\frac{(1+C\mu)^3}{\epsilon\mu^2}$, where the condition number of $\frac{1}{n}\mathbf{A}^\top\hat{\mathbf{D}}\mathbf{A} + \mu\mathbb{I} + \mathbf{E}_1$ should be close to the condition number $\max_{\mathbf{x}}\kappa(\nabla^2 f(\mathbf{x}))$ (For a formal proof of this statement see Appendix E.4). This completes the proof.

■

C.1 Main Result

We are now able to state our main result with explicit bounds.

Theorem 4.1. *For any dimension d, consider the regularized logistic loss defined in (3.5) with regularization parameter $\mu > 0$, and define $\kappa_f = \max_{\mathbf{x}\in\mathbb{R}^d}\kappa(\nabla^2 f(\mathbf{x}))$. Then for any $T > 0$ and $\epsilon > 0$, there exists a linear Transformer that can approximate T iterations of Newton's method on the regularized logistic loss up to error ϵ per iteration. In particular, the width of such a linear Transformer can be bounded by $O(d(1+\mu)^6/\epsilon^4\mu^8)$, and its depth can be bounded by $T(11+2k)$, where $k \leq 2\log\kappa_f + \log\log\frac{(1+\mu)^3}{\epsilon^2\mu^2}$. Furthermore, there is a constant $c > 0$ depending on μ such that if $\epsilon < c$, then the output of the Transformer provides a $\tilde{\mathbf{w}}$ satisfying that $\|\tilde{\mathbf{w}} - \hat{\mathbf{w}}\|_2 \leq O(\sqrt{\epsilon(1+\mu)/(4\mu)})$, where $\hat{\mathbf{w}}$ is the global minimizer of the loss.*

Remark C.1. Both T and k are actually have just one extra additive constant term. In the first case, depends on the number of steps needed for $\lambda(\mathbf{x})$ to become less than $1/6$, while for k the constant term is $2\lceil\log\kappa(\nabla^2 f(\mathbf{x}))\rceil$.

Proof. The proof follows by combining Theorem 4.4 and Theorem 4.2. ■

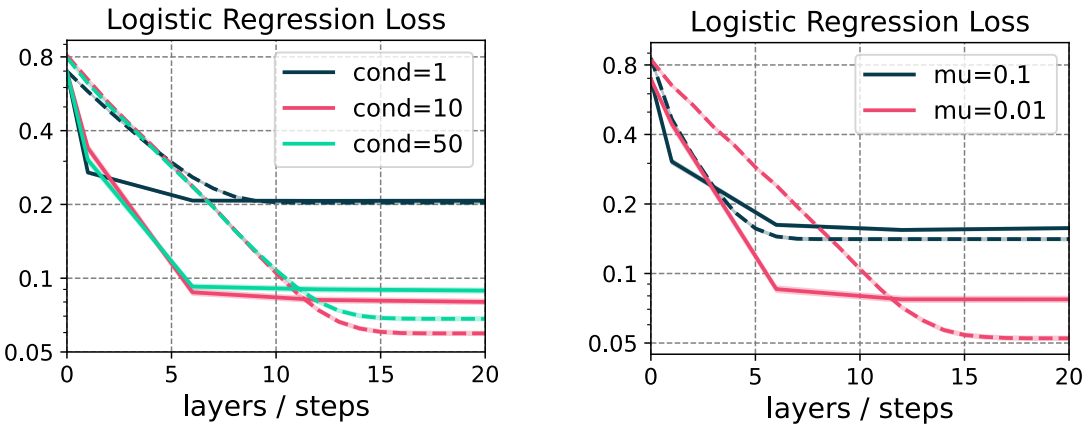

Figure 6: *Loss of trained models (noted with solid line) and the achieved loss by Newton's method (dotted line). We observe that for initial steps the model outperforms Newton's method.*

D Experiments

D.1 Experimental details

During training, we utilize the Adam optimizer, with a carefully planned learning rate schedule. The task is learned through curriculum learning Garg et al. [2022], where we begin training with a small problem dimension and gradually increase the difficulty of the tasks. When implementing the Transformer backbone, we adhere to the structure of NanoGPT2[3], modifying only the causal attention to full attention.

For both the task of linear and logistic regression, we sample the data points as follows: We sample a random matrix $\mathbf{A}$ and create its SVD decomposition, i.e., $\mathbf{A} = \mathbf{USV}^\top$. We then sample the maximum eigenvalue λ_{max} uniformly random in the range of $[1, 100]$. We then set the minimum eigenvalue as $\lambda_{min} = \lambda_{max}/\kappa$, where κ is the condition number of the problem at hand and it is fixed. Finally, we sample the rest of the eigenvalues uniformly random between $[\lambda_{min}, \lambda_{max}]$ and recreate the matrix $\mathbf{S}$, with the new eigenvalues. We then create our covariance matrix as $\Sigma = \mathbf{USU}^\top$. We use Σ, to sample the data samples $\mathbf{x}_i$ from a multivariate Gaussian with mean 0 and covariance matrix Σ. To get $\mathbf{w}^*$ for Newton's method, we considered that the method has converged when two consecutive steps of Newton's method differ less than $1e^{-6}$.

D.2 Additional experiments

In Figure 6 we consider different values of the parameter μ and different condition numbers for the covariance matrix Σ. In Figures 7 and 8, we present comprehensive results for the LSA and LSA with layernorm models, trained on linear regression tasks across various condition numbers and noise levels. The trained LSA and LSA with layernorm consistently find higher-order methods when attempting to solve ill-conditioned linear regression problems.

Effects of the Optimization Target in Logistic Regression Task. In our logistic regression experiments, the default objective is training the Transformer to predict logistic regression parameters derived from Newton's method (referred to as "target=newton"). To understand the impact of different optimization objectives, we also trained the Transformer explicitly to predict the underlying logistic regression weight (referred to as "target=w"), and compared the mean squared distances to their respective targets. As illustrated in Figure 9 (left), the Transformer achieves a lower error when trained to approximate Newton's solution directly, rather than predicting the ground-truth weight. Since our primary purpose is to understand how well Transformer

[3]https://github.com/karpathy/nanoGPT/blob/master/model.py

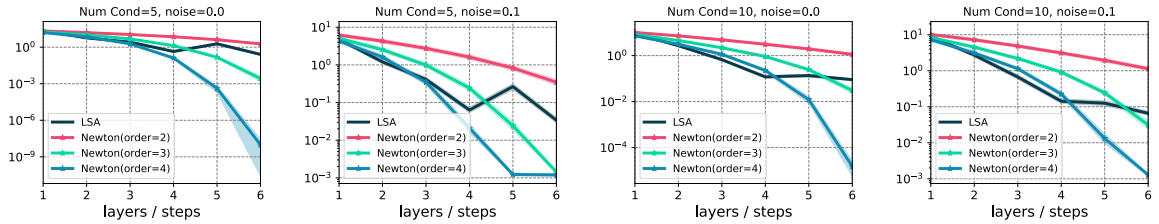

Figure 7: *Loss of LSA and different order Newton iteration for linear regression with different input conditions.*

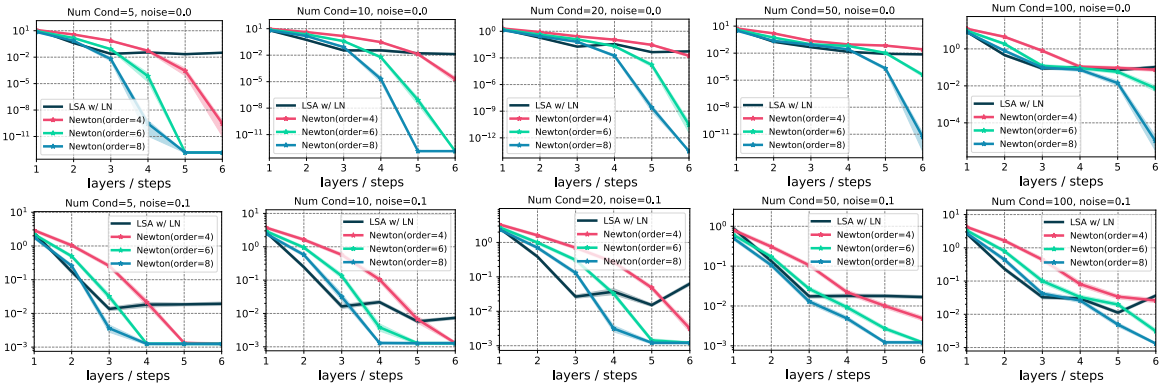

Figure 8: *Loss of LSA w/ layernorm and different order Newton iteration for linear regression with different input conditions.*

emulate the Newton's method, we choose to train the Transformer using Newton's solution as our primary objective.

Investigating the Mapping Between Transformer Depth and Newton's Iteration. While our theoretical construction explicitly demonstrates that each step of Newton's method can be approximated by a fixed number of Transformer layers (as detailed in Theorem 4.2), in this paragraph, we try to understand the actual mapping obtained by the trained Transformer, by measuring the distance between the trained Transformer's predicted solution and the converged Newton solution, and comparing it with Newton's iterations. As shown in Figure 9 (middle), the trained Transformer initially approximates the first few iterations of Newton's method using relatively few layers, suggesting a potential nonlinear mapping between Transformer layers and Newton iteration steps. However, the Transformer's approximation saturates after approximately 10 layers, whereas Newton's method continues to converge further. This observation implies that although the Transformer successfully learns to approximate the overall Newton solution, it does not necessarily encode each Newton iteration explicitly in a step-by-step manner.

Impact of Transformer Depth on Logistic Regression Performance. By default, we train the Transformer model up to 20 layers for the logistic regression task. To further investigate the impact of model depth, we conducted an additional experiment where we increased the number of Transformer layers to 40, maintaining the experimental setup from Figure 2, using a Hessian condition number of 1. The results, presented in Figure 9 (right), demonstrate that deeper Transformer models can indeed be effectively trained and successfully minimize logistic regression loss. Notably, Newton's method converges in roughly 10 iterations, while the Transformer's performance saturates around 6 layers, indicating no significant benefit from extending the depth beyond approximately 30 layers.

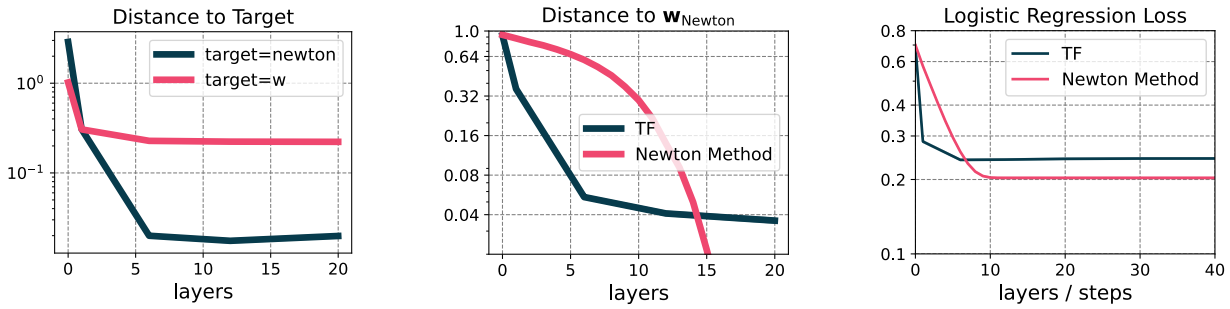

Figure 9: *(Left)* Distance of the Transformer as the number of layers is increased to the minimum found by Newton's Method (black) and the true underlying $\mathbf{w}^*$ (pink). *(Middle)* Distance of Transformer and Newton's method to the final prediction of Newton's method. *(Right)* Logistic regression loss achieved by Transformer and Newton's Method.

E Auxiliary results

We collect here some auxiliary results that are used in the proofs of the main results.

E.1 Auxiliary result for constant decrease of the logistic loss

Let

$$h(x) = -\frac{x^2}{1+x} + c - log(1-\tilde{\delta}) - \tilde{\delta}$$

where $\tilde{\delta} = \sqrt{x^2/(1+x)^2 - 2c/(1+x) + c'}$ and c, c' are constants with respect to x. Mainly, we see the RHS of the previous inequality as a function of λ and consider $\mathbf{E}_t^\top \nabla g(\hat{\mathbf{x}}_t), \mathbf{E}_t^\top \nabla^2 g(\hat{\mathbf{x}}_t)\mathbf{E}_t$ as constants. Then we have that

$$\begin{aligned}
h'(x) &= -\frac{x^2+2x}{(1+x)^2} + \frac{1}{1-\tilde{\delta}}\frac{\mathrm{d}\tilde{\delta}}{\mathrm{d}x} - \frac{\mathrm{d}\tilde{\delta}}{\mathrm{d}x} \\
&= -\frac{x^2+2x}{(1+x)^2} + \frac{\tilde{\delta}}{1-\tilde{\delta}}\frac{\mathrm{d}\tilde{\delta}}{\mathrm{d}x} \\
&= -\frac{x^2+2x}{(1+x)^2} + \frac{\tilde{\delta}}{1-\tilde{\delta}}\frac{x+c(1+x)}{\tilde{\delta}(1+x)^3} \\
&= -\frac{(x^2+2x)(1+x)(1-\tilde{\delta})}{(1+x)^3(1-\tilde{\delta})} + \frac{x+c(1+x)}{(1-\tilde{\delta})(1+x)^3} \\
&= \frac{x+c(1+x)-(x^2+2x)(1+x)(1-\tilde{\delta})}{(1-\tilde{\delta})(1+x)^3}
\end{aligned}$$

We use mathematica to plot this function (see Figure 10) and the max value it can attain, when $x \in [1/6, 1]$ and $|c| \leq 0.06$ and we have that the maximum value of g' is approximately -0.02. This implies that h is decreasing, since we have that $|c| \leq \|\mathbf{E}_t\| \, \|\nabla f(\hat{\mathbf{x}}_t)\| \leq \frac{\epsilon(1+\mu)}{4\mu} \leq 0.06$. Thus, we have

$$\begin{aligned}
g(\hat{\mathbf{x}}_{t+1}) - g(\hat{\mathbf{x}}_t) &\leq h(1/6) \\
&= -\frac{1}{42} + y - \log(1 - \sqrt{1/49 - 12y/7 + z} - \sqrt{1/49 - 12y/7 + z}
\end{aligned}$$

where $y = \mathbf{E}_t^\top \nabla g(\hat{\mathbf{x}}_t)$ and $z = \mathbf{E}_t^\top \nabla^2 g(\hat{\mathbf{x}}_t)\mathbf{E}_t$. Notice now that

$$|z| \leq \frac{\epsilon^2(1+\mu)}{4\mu} \leq 0.01^2 \text{ and } |y| \leq 0.01$$

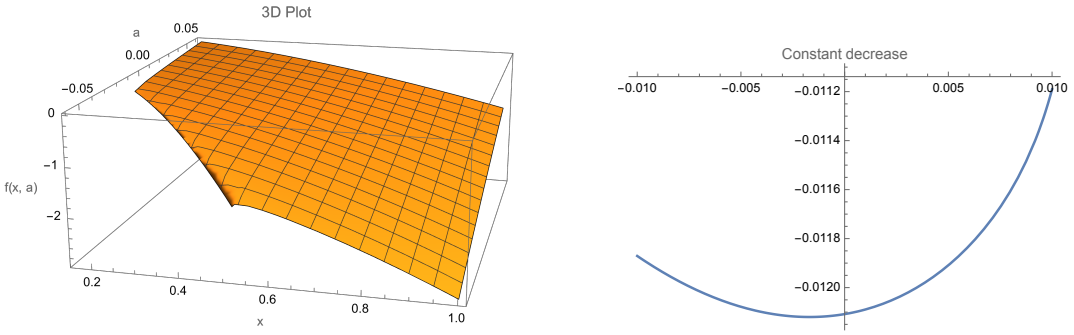

Figure 10: Left: The derivative of h as a function of both x and c for $x \in [1/6, 1]$ and $|c| \leq 0.06$. Right: We see that the function is decreasing at least -0.01 at each step.

Since $\epsilon \leq \min\{0.01, 0.04\mu/(1+\mu)\}$. Given these bounds we have that

$$g(\hat{\mathbf{x}}_{t+1}) - g(\hat{\mathbf{x}}_t) \leq -\frac{1}{42} + y - \log(1 - \sqrt{1/49 - 12y/7 + 0.01^2} - \sqrt{1/49 - 12y/7}$$

We again use mathematica and plot this function for $|y| \leq 0.012$, which can be viewed in Figure 10 and we see that we get a constant decrease of at least 0.01.

E.2 Auxiliary result for controlling the error

To control the change that this quantity can evoke, we note that we have approximated the function $g(x) = 2\sqrt{\mu}/(2\sqrt{\mu} + \sqrt{x})$ by discretizing $(0, 1]$, so whenever $x \leq \alpha^2$, $g(x) \geq 2\sqrt{\mu}/(2\sqrt{\mu} + \alpha)$. Thus, if $x \leq 4\mu\epsilon_4^2/(1-\epsilon_4)^2$ we have that $g(x) \geq 1 - \epsilon_4$, similarly if $x \geq 4\mu(1-\epsilon_4)^2/\epsilon_4^2$ we have that $g(x) \leq \epsilon_4$. Now notice that given $x, \tilde{x}$ such that $\max\{\tilde{x}, x\} \geq 4\mu\epsilon_4^2/(1-\epsilon_4)^2$ (otherwise we have already covered the case) we have

$$\begin{aligned}
|g(x) - g(\tilde{x})| &= \left| \frac{2\sqrt{\mu}}{2\sqrt{\mu} + \sqrt{x}} - \frac{2\sqrt{\mu}}{2\sqrt{\mu} + \sqrt{x'}} \right| \\
&\leq 2\sqrt{\mu} \left| \frac{\sqrt{x} - \sqrt{x'}}{(2\sqrt{\mu} + \sqrt{x})(2\sqrt{\mu} + \sqrt{x'})} \right| \\
&\leq 2\sqrt{\mu} \left| \frac{x - x'}{4\mu(\sqrt{x} + \sqrt{x'})} \right| \\
&\leq 2\sqrt{\mu} \left| \frac{x - x'}{4\mu\sqrt{\max\{\tilde{x}, x\}}} \right| \\
&\leq \frac{|x - x'|\,(1 - \epsilon_4)}{4\mu\epsilon_4}.
\end{aligned}$$

Thus, if it holds that

$$|x - x'| \leq 4\mu \frac{\epsilon_4^2}{(1-\epsilon_4)}, \tag{E.1}$$

then the function is always less than ϵ_4.

E.3 Perturbation bounds

Theorem E.1 (Corollary 2.7, p. 119 in Stewart and guang Sun [1990]). *Let $\kappa(\mathbf{A}) = \|\mathbf{A}\|_2 \left\|\mathbf{A}^{-1}\right\|_2$ be the condition number of $\mathbf{A}$. If $\tilde{\mathbf{A}} = \mathbf{A} + \mathbf{E}$ is non-singular, then*

$$\left\|\tilde{\mathbf{A}}^{-1} - \mathbf{A}^{-1}\right\|_2 \leq \kappa(\mathbf{A}) \frac{\|\mathbf{E}\|_2 \left\|\tilde{\mathbf{A}}^{-1}\right\|_2}{\|\mathbf{A}\|_2}.$$

If in addition $\kappa(\mathbf{A}) \frac{\|\mathbf{E}\|_2}{\|\mathbf{A}\|_2} < 1$ then

$$\left\|\tilde{\mathbf{A}}^{-1}\right\|_2 \leq \frac{\left\|\mathbf{A}^{-1}\right\|_2}{1 - \kappa(\mathbf{A}) \frac{\|\mathbf{E}\|_2}{\|\mathbf{A}\|_2}},$$

and thus

$$\left\|\tilde{\mathbf{A}}^{-1} - \mathbf{A}^{-1}\right\|_2 \leq \frac{\kappa(\mathbf{A}) \frac{\|\mathbf{E}\|_2}{\|\mathbf{A}\|_2}}{1 - \kappa(\mathbf{A}) \frac{\|\mathbf{E}\|_2}{\|\mathbf{A}\|_2}} \left\|\mathbf{A}^{-1}\right\|_2.$$

Corollary E.2. *Let f be the regularized logistic loss defined in (3.5) with regularization parameter $\mu > 0$. For the matrix $\mathbf{B} = (\nabla^2 f(\mathbf{x}) + \mathbf{E}_1)$, it holds that*

$$\left\|\mathbf{B}^{-1} - (\nabla^2 f(\mathbf{x}))^{-1}\right\|_2 \leq \frac{\|\mathbf{E}_1\|_2}{\mu(\mu - \|\mathbf{E}_1\|_2)}$$

or equivalently, $\mathbf{B}^{-1} = (\nabla^2 f(\mathbf{x}))^{-1} + \mathbf{E}_1'$ with $\|\mathbf{E}_1'\|_2 \leq \frac{\|\mathbf{E}_1\|_2}{\mu(\mu - \|\mathbf{E}_1\|_2)}$.

Proof. From Theorem E.1 we have that

$$\left\|\mathbf{B}^{-1} - (\nabla^2 f(\mathbf{x}))^{-1}\right\|_2 \leq \frac{\left\|(\nabla^2 f(\mathbf{x}))^{-1}\right\|_2^2 \|\mathbf{E}_1\|_2}{1 - \|(\nabla^2 f(\mathbf{x}))^{-1}\|_2 \|\mathbf{E}_1\|_2} \leq \frac{\|\mathbf{E}_1\|_2}{\mu(\mu - \|\mathbf{E}_1\|_2)}$$

because $\left\|(\nabla^2 f(\mathbf{x}))^{-1}\right\|_2 \leq 1/\mu$ and we have assumed that $\|\mathbf{E}_1\|_2 \leq \mu$. ■

E.4 Condition number of perturbed matrix

To show that the condition number of the matrix $\mathbf{B} = \nabla f(\mathbf{x}) + \mathbf{E}$ is close to the condition number of $\nabla f(\mathbf{x})$ we will use Weyl's inequality.

Lemma E.3 (Weyl's Corollary 4.9 in Stewart and guang Sun [1990]). *Let λ_i be the eigenvalues of a matrix $\mathbf{A}$ with $\lambda_1 \geq \ldots \geq \lambda_n$, $\tilde{\lambda}_i$ be the eigenvalues of a perturbed matrix $\tilde{\mathbf{A}} = \mathbf{A} + \mathbf{E}$ and finally let $\epsilon_1 \geq \ldots \epsilon_m$ be the eigenvalues of $\mathbf{E}$. For $i = 1, \ldots, n$ it holds that*

$$\tilde{\lambda}_i \in [\lambda_i + \epsilon_n, \lambda_i + \epsilon_1] \tag{E.2}$$

Thus, for the matrix $\mathbf{B}$ we have that the condition number of the eigenvalues of $\mathbf{B}$ can be bounded as follows

$$\lambda_{\min}(\nabla f(\mathbf{x})) + \lambda_{\min}(\mathbf{E}) \leq \lambda_{\min}(\mathbf{B}) \leq \lambda_{\min}(\nabla f(\mathbf{x})) + \lambda_{\max}(\mathbf{E}) \tag{E.3}$$

$$\lambda_{\max}(\nabla f(\mathbf{x})) + \lambda_{\min}(\mathbf{E}) \leq \lambda_{\max}(\mathbf{B}) \leq \lambda_{\max}(\nabla f(\mathbf{x})) + \lambda_{\max}(\mathbf{E}) \tag{E.4}$$

For $\|\mathbf{E}\|_2 < \mu$ we have that

$$\frac{\lambda_{\max}(\nabla f(\mathbf{x})) + \lambda_{\min}(\mathbf{E})}{\lambda_{\min}(\nabla f(\mathbf{x})) + \lambda_{\max}(\mathbf{E})} \leq \kappa(\mathbf{B}) \leq \frac{\lambda_{\max}(\nabla f(\mathbf{x})) + \lambda_{\max}(\mathbf{E})}{\lambda_{\min}(\nabla f(\mathbf{x})) + \lambda_{\min}(\mathbf{E})} \tag{E.5}$$

$$\frac{\kappa(\nabla f(\mathbf{x})) + \lambda_{\min}(\mathbf{E})/\lambda_{\min}(\nabla f(\mathbf{x}))}{1 + \lambda_{\max}(\mathbf{E})/\lambda_{\min}(\nabla f(\mathbf{x}))} \leq \kappa(\mathbf{B}) \leq \frac{\kappa(\nabla f(\mathbf{x})) + \lambda_{\max}(\mathbf{E})/\lambda_{\min}(\nabla f(\mathbf{x}))}{1 + \lambda_{\min}(\mathbf{E})/\lambda_{\min}(\nabla f(\mathbf{x}))} \tag{E.6}$$

And since $\|\mathbf{E}\|_2$ is small the two condition numbers are close.