EFFICIENT ASYNCHRONIZE STOCHASTIC GRADIENT ALGORITHM WITH STRUCTURED DATA

Zhizhou Sha* **Zhao Song†** **Mingquan Ye‡**

ABSTRACT

Deep neural networks have demonstrated remarkable success across various domains, including computer vision, natural language processing, and bioinformatics. However, the increasing depth and complexity of these networks have led to significant computational and storage challenges. While prior research has addressed these issues through techniques such as network pruning and the use of high-dimensional data structures like locality-sensitive hashing (LSH) and space-partitioning trees, the computational cost per iteration during training remains linear in the data dimension d. In this work, we explore the potential of leveraging special structures in the input data to reduce this cost. Specifically, we consider input data points that can be represented as tensor products of lower-dimensional vectors, a common scenario in applications. We present a novel stochastic gradient descent algorithm that, under mild assumptions on the input data structure, achieves a per-iteration training cost that is sublinear in the data dimension d. To the best of our knowledge, this is the first work to achieve such a result, marking a significant advancement in the efficiency of training deep neural networks. Our theoretical findings demonstrate that the proposed algorithm can train a two-layer fully connected neural network with a per-iteration cost independent of d.

1 INTRODUCTION

Deep neural networks have achieved great success in many fields, e.g., computer vision LeCun et al. (1998); Krizhevsky et al. (2012); Szegedy et al. (2015); Simonyan & Zisserman (2015); He et al. (2016), natural language processing Collobert et al. (2011); Devlin et al. (2018); Peters et al. (2018); Radford et al. (2018); Yang et al. (2019), and bioinformatics Min et al. (2017), just to name a few. Despite the excellent performances in a variety of applications, deep neural networks have brought intensive computation and occupied large storage with the growth of layers. For example, the *ResNet* proposed by He et al. (2016) has 152 layers and the parameters of VGG-16 Simonyan & Zisserman (2015) take 552MB memory Han et al. (2015a). To get around these problems, a lot of relevant approaches have been proposed. In terms of the storage issue, Han et al. (2015a;b) compressed the deep networks significantly without loss of accuracy by pruning redundant connections between layers, and deployed the compressed networks on embedded systems. For the intensive computation issue, researchers have focused on reducing training time for deep networks. Total training time involves the number of iterations and the time cost per iteration. We focus on the latter aspect of our work.

To execute faster computation in each iteration, one natural choice is to utilize some high-dimensional data structures that can query points in some geometric regions efficiently. The first kind of method is based on locality-sensitive hashing (LSH) Indyk & Motwani (1998) which returns a point from a set that is closest to a given query point under some metric (e.g., ℓ_p norm). Chen et al. (2020a) built an end-to-end LSH framework MONGOOSE to train neural networks efficiently via a modified learnable version of data-dependent LSH. Chen et al. (2020b) proposed SLIDE that significantly reduces the computations in both training and inference stages by taking advantage of nearest neighbor search based on LSH. Sima & Xue (2021) proposed a unified framework LSH-SMILE that

* shazz20@mails.tsinghua.edu.cn. Tsinghua University.
† magic.linuxkde@gmail.com. The Simons Institute for the Theory of Computing at UC Berkeley.
‡ mye9@uic.edu. UIC.

scales up both forward simulation and backward learning by the locality of partial differential equations update. The second kind of method utilizes the data structures on space partitioning, including k-d tree Bentley (1975); Chan (2019), Quadtree Finkel & Bentley (1974), and partition tree Agarwal et al. (1992); Matousek (1992a;b); Afshani & Chan (2009); Chan (2012), etc. Song et al. (2021) employed the Half-Space Reporting (HSR) data structures Agarwal et al. (1992), which can return all the points having large inner products and support data updates, and improved the time complexity of each iteration in training neural networks to sublinear in network width.

The above-mentioned works have tried to accelerate the training time of deep networks from the perspective of data structures. In this paper, we try doing that from the perspective of input data. A natural question to ask is that.

Is there some mild assumption on the input data so that each iteration only takes sublinear time in the data dimension in training neural networks?

In this work, we answer this question positively. To the best of our knowledge, all the previous work needs to pay linear in data dimension d at each iteration Li & Liang (2018); Du et al. (2019b); Allen-Zhu et al. (2019a;b); Du et al. (2019a); Song & Yang (2019); Song et al. (2021). This is the first work that achieves the cost per iteration independent of dimensionality d.

We are motivated by the phenomenon that the training data often have a variety of features extracted from different methods or domains. To enhance robustness and discrimination, researchers combine various features into one holistic feature using tensor products before training. Specifically, given two vectors $u \in \mathbb{R}^{d_1}$ and $v \in \mathbb{R}^{d_2}$ representing different features, the tensor product $u \otimes v$ gives a $d_1 \times d_2$ matrix, which can be vectorized to a $(d_1 \times d_2)$-dimensional vector. In the bioinformatics field Ben-Hur & Noble (2005), the fusion of different features of proteins can help us analyze their characteristics effectively. Smalter et al. (2009) employed tensor product feature space to model interactions between feature sets in different domains and proposed two methods to circumvent the feature selection problem in the tensor product feature space. For click-through rate prediction Juan et al. (2016); Naumov et al. (2019), the accuracy can be improved by the fusion of two features. In computer vision, Zhou et al. (2012) combined three features HOG Dalal & Triggs (2005), LBP Ojala et al. (2002), and Haar-like Babenko et al. (2010) by tensor products and applied the new feature to visual tracking. Apart from the above-mentioned applications, the Kronecker structure has also been widely applied to deep neural networks Gao et al. (2020); Jagtap et al. (2022); Feng & Yang (2022); Patro et al. (2023).

1.1 Our Results

We try improving the cost of per iteration when the input data has some special structures. Assume that the input data points satisfy that for any $i \in [n]$, $x_i = \mathrm{vec}(\overline{x}_i \overline{x}_i^\top) \in \mathbb{R}^d$ with $\overline{x}_i \in \mathbb{R}^{\sqrt{d}}$. For this setting, we have the following theorem, which is our main contribution.

Theorem 1.1 (Informal version of Theorem E.1)**.** *Given n training samples $\{(x_i, y_i)\}_{i=1}^n$ such that for each $i \in [n]$, $x_i = \mathrm{vec}(\overline{x}_i \overline{x}_i^\top) \in \mathbb{R}^d$, there exists a stochastic gradient descent algorithm that can train a two-layer fully connected neural network such that each iteration takes time $o(m) \cdot n$, where m is the width of the neural network, that is independent of the data dimension d.*

The conclusion also holds for the general case: for each $i \in [n]$, $x_i = b_i \otimes a_i \in \mathbb{R}^d$ with $a_i \in \mathbb{R}^p$, $b_i \in \mathbb{R}^q$ and $p, q = O(\sqrt{d})$.

2 Related Work

Kernel Matrix. Let $X := [x_1, \cdots, x_n] \in \mathbb{R}^{d \times n}$, then the Gram matrix $G \in \mathbb{R}^{n \times n}$ of the n columns of X satisfies that $G_{i,j} = x_i^\top x_j$, i.e., $G = X^\top X$. The Gram matrix $K \in \mathbb{R}^{n \times n}$ with respect to the n columns of X such that $K_{i,j} = k(x_i, x_j)$ is called a kernel matrix, where k is referred to as a kernel function.

Daniely (2017) showed that in polynomial time, the stochastic gradient descent algorithm can learn a function which is competitive with the best function in the conjugate kernel space of the network, and established connection between neural networks and kernel methods. Jacot et al. (2018) proved that

for a multi-layer fully connected neural network, if the weight matrix of each layer has infinite width, then the convergence of gradient descent method can be described by the Neural Tangent Kernel (NTK). Du et al. (2019b) researched a two-layer neural network with ReLU activation function, which is not smooth, and proved that the Gram matrix, which is the kernel matrix in Jacot et al. (2018), keeps stable in infinite training time.

Convergence of Neural Network. There have been two lines of work proving the convergence of neural networks: the first is based on the assumption that the input data are from Gaussian distribution; the other follows the NTK regime Jacot et al. (2018); Li & Liang (2018); Du et al. (2019b); Allen-Zhu et al. (2019a;b); Gu et al. (2024); Shi et al. (2021; 2024). In Jacot et al. (2018), the NTK is first proposed and is central to characterize the generalization features of neural networks. Moreover, it is proven that the convergence of training is related to the positive-definiteness of the limiting NTK. Li & Liang (2018) observed that in the training of a two-layer fully connected neural network, a fraction of neurons are not activated over iterations, i.e., $w_r(t)^\top x_i \leq \tau$, where $r \in [m]$ and $\tau \in \mathbb{R}$ is the threshold of the shifted ReLU activation function. Based on this observation, Li & Liang (2018) obtained the convergence rate by using stochastic gradient descent to optimize the cross-entropy loss function. However, the network width m depends on $\mathrm{poly}(1/\epsilon)$, where ϵ is the desired accuracy, and approaches infinity when ϵ approaches 0. In Du et al. (2019b), the lower bound of m is improved to $\mathrm{poly}(n, 1/\rho, \log(m/\rho))$, where ρ is the probability parameter, by setting the amount of over-parameterization to be independent of ϵ.

Roadmap This paper is organized as follows: In Section 3, we define notations used in the paper. In Section 4, we illustrated the gradient descent formula for shifted ReLU activated neural network. In Section 5, we introduced the functionality of proposed data structure. For Section 6, we demonstrated the informal version of training algorithm and presented the convergence theorem. In Section 7, we introduced the formal training algorithm and cost analysis for each iteration. Then we provide a discussion on our technical novelty in Section 8.

3 NOTATION

For a positive integer n, let $[n]$ represent the set $\{1, 2, \cdots, n\}$. Let $\mathrm{vec}(\cdot)$ denote the *vectorization* operator. Specifically, for a matrix $A = [a_1, \cdots, a_d] \in \mathbb{R}^{d \times d}$, $\mathrm{vec}(A) = [a_1^\top, \cdots, a_d^\top]^\top \in \mathbb{R}^{d^2}$. For the $\mathrm{vec}(\cdot)$ operator, let $\mathrm{vec}^{-1}(\cdot)$ be its inverse such that $\mathrm{vec}^{-1}(\mathrm{vec}(A)) = A$. For a matrix $A \in \mathbb{R}^{d \times n}$ and a subset $S \subset [n]$, $A_{i,j}$ is the entry of A at the i-th row and the j-th column, and $A_{:,S}$ represents the matrix whose columns correspond to the columns of A indexed by the set S. Similarly, for a vector $x \in \mathbb{R}^n$, x_S is a vector whose entries correspond to the entries in x indexed by the set S. Let $\|\cdot\|_2$ and $\|\cdot\|_F$ represent the ℓ_2 norm and Frobenius norm respectively. The symbol $\mathbb{1}(\cdot)$ represents the indicator function. For a positive integer d, I_d denotes the $d \times d$ identity matrix. For two vectors $a, b \in \mathbb{R}^n$, let $a \odot b \in \mathbb{R}^n$ represent the entry-wise product of a and b. For any two matrices A and B, $A \otimes B$ represents the Kronecker product of A and B.

4 PROBLEM FORMULATION

Our problem formulation is similar to that of Du et al. (2019b); Song & Yang (2019); Song et al. (2021). Define the shifted ReLU function to be $\phi_\tau(x) := \max\{x - \tau, 0\}$, where $x, \tau \in \mathbb{R}$ and $\tau \geq 0$ is the threshold. We consider a two-layer shifted ReLU activated neural network with m neurons in the hidden layer $f(W, a, x) := \frac{1}{\sqrt{m}} \sum_{r=1}^m a_r \cdot \phi_\tau(w_r^\top x)$, where $x \in \mathbb{R}^d$ is the input, $W = \{w_1, \cdots, w_m\} \subset \mathbb{R}^d$ are weight vectors in the first layer, and $a_1, \cdots, a_m \in \mathbb{R}$ are weights in the second layer. For simplicity, we only optimize the m weight vectors $w_1, \cdots, w_m$ without training a. Then for each $r \in [m]$, we have $\frac{\partial f(W,a,x)}{\partial w_r} = \frac{1}{\sqrt{m}} a_r \cdot \mathbb{1}(w_r^\top x > \tau) \cdot x$. Given n training samples $\{(x_i, y_i)\}_{i=1}^n$ with $x_i \in \mathbb{R}^d$ and $y_i \in \mathbb{R}$ for each $i \in [n]$, the objective function $L(W)$ is defined by $L(W) := \frac{1}{2} \sum_{i=1}^n (f(W, a, x_i) - y_i)^2$.

Additionally, for a specific batch $S \subseteq [n]$, the objective function denoted as $L(W, S)$ is defined to be

$$L(W, S) := \frac{n}{|S|} \cdot \frac{1}{2} \sum_{i \in S} (f(W, a, x_i) - y_i)^2.$$

Gradient Descent (GD). We defined the standard GD optimizer as follows: for each $r \in [m]$, let $w_r(t)$ represent the weight vector w_r at the t-th iteration. Then we have the update for $t+1$, $w_r(t+1) = w_r(t) - \eta \cdot \frac{\partial L(W(t))}{\partial w_r(t)}, r \in [m]$, where η is the step size and $\frac{\partial L(W(t))}{\partial w_r(t)}$ has the following formulation

$$\frac{\partial L(W(t))}{\partial w_r(t)} = \frac{1}{\sqrt{m}} \sum_{i=1}^{n} (f(W(t), a, x_i) - y_i) \cdot a_r \cdot \mathbb{1}(w_r(t)^\top x_i > \tau) \cdot x_i.$$

Stochastic Gradient Descent (SGD). We use the SGD optimizer defined as follows:

$$w_r(t+1) = w_r(t) - \eta \cdot \frac{\partial L(W(t), S_t)}{\partial w_r(t)}, r \in [m], \tag{1}$$

where the batch set S_t is a uniform sub-sample of $[n]$. For simplicity, we define

$$G_{t,r} := \frac{\partial L(W(t), S_t)}{\partial w_r(t)} = \frac{n}{|S_t|} \cdot \frac{1}{\sqrt{m}} \sum_{i \in S_t} (f(W(t), a, x_i) - y_i) \cdot a_r \cdot \mathbb{1}(w_r(t)^\top x_i > \tau) \cdot x_i. \tag{2}$$

At iteration t, let $u(t) := [u_1(t), \cdots, u_n(t)]^\top \in \mathbb{R}^n$ be the prediction vector, where each $u_i(t)$ satisfies that $u_i(t) = f(W(t), a, x_i)$, $i \in [n]$.

5 ASYNCHRONIZE TREE

In this section, we will briefly describe the outline of the proposed algorithm.

Asynchronize Tree Data Structure. In each iteration of the training algorithm, there are two main parts: forward computation and backward computation. The goal of forward computation is to compute the prediction vector. By the property of the shift ReLU function, for each sampled data point x_i, we need to find which nodes in hidden layer are activated, which is a **query** operation. In backward computation, we need to compute the gradient vectors and then update the weight vectors, which is an **update** operation.

In view of this situation, we propose the ASYNCHRONIZETREE data structure which is a binary tree. It mainly supports **query** and **update** operations. Since the inner product of each weight vector w_r, $r \in [m]$ and input data point x_i, $i \in [n]$ is frequently compared with the threshold τ, we maintain n trees $T_1, \cdots, T_n$ for the n data points respectively. For the i-th tree T_i, the leaf nodes of T_i are the inner products of the m weight vectors with x_i and the value of each inner node is the maximum of the values of its two children. Hence, when executing the **query** operation in T_i, we start with the root of T_i and recurse on its two children. Let the number of satisfactory items be Q, then the time for **query** operation would be $O(Q \cdot \log m)$ because the depth of each tree is $O(\log m)$. When some weight vector w_r, $r \in [m]$ changes, we need to recompute the inner products between w_r and the n data points and then update the n corresponding trees, so the time for **update** operation is $O(n \cdot (d + \log m))$.

Note that the above statements are for the general case, that is, there are no requirements for the input data. Since we use the stochastic gradient descent method, each iteration randomly selects a batch S_t from the set $[n]$. The **query** operation is executed for the trees whose indexes are in the set S_t but not all the n trees, that is why this data structure is called ASYNCHRONIZETREE.

Kronecker Structured Data. Recall that in the **update** operation of data structure ASYNCHRONIZETREE, we need to compute the inner products between w_r and the n data points, i.e., the quantity $X^\top w_r$, where $X = [x_1, \cdots, x_n]$. When the data points have Kronecker property such that

$x_i = \mathrm{vec}(\overline{x}_i \overline{x}_i^\top) \in \mathbb{R}^d$ for each $i \in [n]$, it would be efficient to compute the matrix-vector multiplication $X^\top w_r$. In particular, we have the following equation $(X^\top w_r)_i = (\overline{X}^\top \cdot \mathrm{vec}^{-1}(w_r) \cdot \overline{X})_{i,i}$, where $\overline{X} = [\overline{x}_1, \cdots, \overline{x}_n]$ and $\mathrm{vec}^{-1}(\cdot)$ is the inverse vectorization operator. Then the computing of $X^\top w_r$ is transferred to the fast matrix multiplication.

At the t-th iteration, we need to compute the gradient vector denoted as $\delta_{t,r}$ to update the weight vector w_r with $r \in [m]$, that is, $w_r(t+1) = w_r(t) + \delta_{t,r}$. When w_r changes, we need to recompute $w_r^\top x_i$ for $i \in [n]$. Since $w_r(t)^\top x_i$ is already known, in order to compute $w_r(t+1)^\top x_i$, we only need to compute $\delta_{t,r}^\top x_i$. To be specific, the vector $\delta_{t,r}$ has such form $\delta_{t,r} = X_{:,S_t} \cdot c$ with $c \in \mathbb{R}^{|S_t|}$, then we have $\delta_{t,r}^\top x_i = c^\top X_{:,S_t}^\top x_i$, where for $j \in [|S_t|]$, $(X_{:,S_t}^\top x_i)_j = (\overline{X}_{:,S_t}^\top \cdot (\overline{x}_i \overline{x}_i^\top) \cdot \overline{X}_{:,S_t})_{j,j}$.

Hence, the computation of vector $X_{:,S_t}^\top x_i$ is reduced to the pairwise inner products $\overline{x}_i^\top \overline{x}_j$ for $i, j \in [n]$, which can be precomputed at the initialization, and takes time only $O(|S_t|)$. Now the **update** operation can be completed by computing $\delta_{t,r}^\top x_i$ for all $i \in [n]$ and then updating the n trees, and thus takes time $O(n \cdot (|S_t| + \log m))$, which is faster than the fast matrix multiplication.

6 Convergence Analysis

In this section, we present the convergence theorem for training a two-layer fully connected neural network using SGD. We first give the informal version of the training algorithm (see Algorithm 1). Since we analyze the convergence in this section, the formal training algorithm with the asynchronize tree data structure will be shown in the next section. In Section 6.1, we introduce the definition of data-dependent matrix H and give the bound for its smallest eigenvalue which is an important parameter in the proof of convergence theorem. The convergence theorem is presented in Section 6.2.

Algorithm 1 Accelerate computation in each iteration using asynchronize tree data structure (informal version of Algorithm 2)

procedure OurAlgorithm($X = [x_1, \cdots, x_n] \in \mathbb{R}^{d \times n}$, $y \in \mathbb{R}^n$)
  Initialize the weight vector $w_r(0) \sim \mathcal{N}(0, I_d)$ for each $r \in [m]$
  Construct an AsynchronizeTree data structure AT
  Let AT call the procedure Init to build the n trees $T_1, \cdots, T_n$ and compute the pairwise inner products $x_i^\top x_j$ for $i, j \in [n]$
  for $t = 1$ to T **do**
    Sample a set $S_t \subset [n]$ with size S_{batch} uniformly at random
    for each $i \in S_t$ **do**
      Let AT call the procedure Query to return the set of neurons L_i that are activated with respect to x_i
      Compute the prediction value $u_i(t)$
    end for
    Let $\ell(t)$ be the union of L_i for all the $i \in S_t$
    for each $r \in \ell(t)$ **do**
      Initialize the vector v with zero vector and for $i \in S_t$ assign the entry v_i with value $\frac{\eta n}{S_{\mathrm{batch}} \sqrt{m}} \cdot a_r \cdot \mathbb{1}(w_r(t)^\top x_i > \tau)$
      Compute the vector $\delta_{t,r}$ by multiplying matrix $X_{:,S_t}$ with vector $v_{S_t} \odot (y - u(t))_{S_t}$
      Let AT call the procedure Update to update the n trees $T_1, \cdots, T_n$ since $w_r(t)$ increased by $\delta_{t,r}$
    end for
  end for
  return $u(T)$
end procedure

6.1 Essential concepts

We start with the definition of the Gram matrix, which can be found in Du et al. (2019b).

Definition 6.1 (Data-dependent matrix H)**.** Given a collection of data points $\{x_1, \cdots, x_n\} \subset \mathbb{R}^d$ and m weight vectors $\{w_1, \cdots, w_m\} \subset \mathbb{R}^d$, the continuous (resp. discrete) Gram matrix denoted as

H^{cts} (resp. H^{dis}) is defined by

$$H_{i,j}^{\mathrm{cts}} := \mathop{\mathbb{E}}_{w \in \mathcal{N}(0,I_d)}[x_i^\top x_j \cdot \mathbb{1}(w^\top x_i > \tau, w^\top x_j > \tau)],$$

$$H_{i,j}^{\mathrm{dis}} := \frac{1}{m}\sum_{r=1}^{m} x_i^\top x_j \cdot \mathbb{1}(w_r^\top x_i > \tau, w_r^\top x_j > \tau).$$

Let $\lambda := \lambda_{\min}(H^{\mathrm{cts}})$ be the smallest eigenvalue of the matrix H^{cts} and assume $\lambda \in (0, 1]$.

Given the two matrices H^{cts} and H^{dis}, the following lemma gives the bound of $\lambda_{\min}(H^{\mathrm{dis}})$.

Lemma 6.2 (implied in Du et al. (2019b); Song et al. (2021))**.** *For any shift threshold $\tau \geq 0$, let $\lambda := \lambda_{\min}(H^{\mathrm{cts}})$ and $m = \Omega(\lambda^{-1} n \log(n/\alpha))$ be the number of samples in H^{dis}, then $\Pr[\lambda_{\min}(H^{\mathrm{dis}}) \geq \frac{3}{4}\lambda] \geq 1 - \alpha$.*

Besides the two matrices H^{cts} and H^{dis}, each iteration $t \geq 0$ has a data-dependent matrix $H(t)$ defined below.

Definition 6.3 (Dynamic data-dependent matrix $H(t)$)**.** For $t \geq 0$, given the m weight vectors $\{w_1(t), \cdots, w_m(t)\} \subset \mathbb{R}^d$ at iteration t, the corresponding data-dependent matrix $H(t)$ is defined by

$$H(t)_{i,j} := \frac{1}{m}\sum_{r=1}^{m} x_i^\top x_j \cdot \mathbb{1}(w_r(t)^\top x_i > \tau, w_r(t)^\top x_j > \tau).$$

6.2 Convergence Theorem

Finally, we present the convergence theorem for our algorithm.

Theorem 6.4. *Given n training samples $\{(x_i, y_i)\}_{i=1}^n$ and a parameter $\rho \in (0, 1)$. Initialize $w_r \sim \mathcal{N}(0, I_d)$ and sample a_r from $\{-1, +1\}$ uniformly at random for each $r \in [m]$. Set the width of the neural network to be $m = \mathrm{poly}(\lambda^{-1}, S_{\mathrm{batch}}^{-1}, n, \log(n/\rho))$, and the step size $\eta = \mathrm{poly}(\lambda, S_{\mathrm{batch}}, n^{-1})$, where $\lambda = \lambda_{\min}(H^{\mathrm{cts}})$ and S_{batch} is the batch size, then with probability at least $1 - O(\rho)$, the vector $u(t)$ for $t \geq 0$ in Algorithm 1 satisfies that*

$$\mathbb{E}[\|u(t) - y\|_2^2] \leq (1 - \eta\lambda/2)^t \cdot \|u(0) - y\|_2^2. \tag{3}$$

The proof for this theorem is deferred to Section C. Theorem 6.4 means that the error between $u(t)$ and the target y decays exponentially as a function of time. Specifically, the error decreases at a rate determined by the parameter λ, which is linked to the spectrum of the underlying problem, and the step size η, which controls how much the update rule affects $u(t)$ in each iteration. The term $(1 - \eta\lambda/2)^t$ signifies the rate of convergence, showing that after each iteration, the error is reduced by a factor of $(1 - \eta\lambda/2)$. Thus, the theorem guarantees that, under the given conditions, the algorithm will converge to the target with an error that shrinks over time, at least as fast as this exponential rate.

7 Data with Kronecker Structure

In this section, we present the formal training algorithm and analyze the cost for each iteration. In Section 7.1, we show the training algorithm using the asynchronize tree data structure. In particular, when the input data points have special properties, e.g., the Kronecker structure, the cost for each iteration can be obviously improved. We introduce some useful properties when the input data points have Kronecker structure in Section 7.2.

7.1 Training Algorithm Using Asynchronize Tree

The formal algorithm for training a two-layer fully connected neural network using SGD and asynchronize tree data structure (see Section D) is shown in Algorithm 2 which has two main parts: the *initialization* step and the *for* loop of iterations. The initialization step initializes the m weight vectors $w_1, \cdots, w_m \in \mathbb{R}^d$ and computes the inner products $w_r^\top x_i$ for $r \in [m]$ and $i \in [n]$. At each

iteration $t \geq 0$, the *forward* step (Line 6-10) computes the prediction vector $u(t) \in \mathbb{R}^n$; given the sampled set $S_t \subset [n]$, for some x_i with $i \in S_t$, we need to look up the activated neurons such that $w_r^\top x_i > \tau$, which is implemented by the QUERY procedure of the asynchronize tree data structure; in addition, the set of activated neurons L_i has the size bound shown in Lemma 7.2. The *backward* step (Line 11-17) updates the weight vectors; the set of weight vectors that would be changed is denoted as $\ell(t)$, whose size has a relationship with the the size of L_i (see Lemma 7.3); the incremental vector for weight vector w_r is denoted as $\delta_{t,r}$. Since w_r changes, we need to update all the inner products $w_r^\top x_i$ for $i \in [n]$, which is executed by the UPDATE procedure of the asynchronize tree data structure.

Algorithm 2 Accelerate computation in each iteration using asynchronize tree data structure (formal version of Algorithm 1)

1: **procedure** OURALGORITHM($X = [x_1, \cdots, x_n] \in \mathbb{R}^{d \times n}$, $y \in \mathbb{R}^n$)
2: Initialize $w_r(0) \sim \mathcal{N}(0, I_d)$ for each $r \in [m]$
3: Construct a ASYNCHRONIZETREE data structure AT
4: AT.INIT($\{w_r(0)\}_{r \in [m]}$, $\{x_i\}_{i \in [n]}$, n, m, d)
5: **for** $t = 1$ to T **do**
6: Sample a set $S_t \subset [n]$ with $|S_t| = S_{\mathrm{batch}}$ uniformly at random
7: **for** each $i \in S_t$ **do**
8: $L_i \leftarrow$ AT.QUERY(i, τ) ▷ $|L_i| \leq Q$
9: $u_i(t) \leftarrow \frac{1}{\sqrt{m}} \sum_{j \in L_i} a_j \cdot \phi_\tau(j.\text{value})$ ▷ $u_i(t)$ is the i-th entry of vector $u(t)$
10: **end for**
11: $\ell(t) \leftarrow \cup_{i \in S_t} L_i$ ▷ $\ell(t) \subset [m]$ is the index set such that w_r changes for each $r \in \ell(t)$ and $|\ell(t)| \leq K$
12: **for** each $r \in \ell(t)$ **do**
13: $v \leftarrow 0_n$ and $v_i \leftarrow \frac{\eta n}{S_{\mathrm{batch}} \sqrt{m}} \cdot a_r \cdot \mathbb{1}(w_r(t)^\top x_i > \tau)$ for $i \in S_t$
14: ▷ The factor of n is to normalize the sum of the losses across all samples
15: $\delta_{t,r} \leftarrow X_{:,S_t} \cdot (v_{S_t} \odot (y - u(t))_{S_t})$ ▷ $\delta_{t,r} = -\eta \cdot G_{t,r}$
16: AT.UPDATE($\delta_{t,r}$, r)
17: **end for**
18: **end for**
19: **return** $u(T)$
20: **end procedure**

The set of neurons that are activated (i.e., $w_r^\top x_i > \tau$) in Algorithm 2 denoted as L_i is formally defined in the following definition.

Definition 7.1 (Fire set)**.** For each $i \in [n]$ and $0 \leq t \leq T$, let $\mathcal{S}_{i,\mathrm{fire}} \subset [m]$ denote the set of neurons that are "fired" at time t, i.e., $\mathcal{S}_{i,\mathrm{fire}}(t) := \{r \in [m] \mid w_r(t)^\top x_i > \tau\}$.

Let $k_{i,t} := |\mathcal{S}_{i,\mathrm{fire}}(t)|$, then the following lemma gives the upper bound of $k_{i,t}$.

Lemma 7.2 (Lemma C.10 in Song et al. (2021))**.** *For $0 < t \leq T$, with probability at least $1 - n \cdot \exp(-\Omega(m) \cdot \min\{R, \exp(-\tau^2/2)\})$, we have the following holds for all $i \in [n]$ $k_{i,t} = O(m \cdot \exp(-\tau^2/2))$ where R is a parameter that depends on m, n, and λ.*

By Lemma 7.2, in our setting, we have that with high probability, $Q = O(m \cdot \exp(-\tau^2/2))$. Moreover, the set of changed weight vectors denoted as $\ell(t)$ in Algorithm 2 satisfies that the upper bound of its size K has the following relationship with the quantity Q.

Lemma 7.3. *The parameters K and Q in Algorithm 2 satisfy that $K = O(S_{\mathrm{batch}} \cdot Q)$.*

Proof. In Algorithm 2, the weight vector w_r is updated if $G_{t,r} \neq 0$, then there exists at least one $i \in S_t$ such that $w_r(t)^\top x_i > \tau$. Since for each $i \in [n]$, there are at most Q neurons that are activated; in addition, $|S_t| = S_{\mathrm{batch}}$, thus there are at most $S_{\mathrm{batch}} \cdot Q$ weight vectors that are changed. □

7.2 PROPERTIES OF KRONECKER STRUCTURE AND RELATED COMPUTATION FACTS

Before proving the formal version of Theorem 1.1, we provide some background about matrix multiplication and the Kronecker product. Let the time of multiplying two matrices in $\mathbb{R}^{a \times b}$ and $\mathbb{R}^{b \times c}$

be $\mathcal{T}_{\mathrm{mat}}(a, b, c)$. In particular, we use ω to denote the exponent of matrix multiplication, which means that $\mathcal{T}_{\mathrm{mat}}(n, n, n) = n^{\omega}$. Currently, $\omega \approx 2.373$ Williams (2012); Le Gall (2014). For the time of matrix multiplication $\mathcal{T}_{\mathrm{mat}}(a, b, c)$, we have the two following properties.

Fact 7.4 (Bürgisser et al. (1997); Bläser (2013))**.** $\mathcal{T}_{\mathrm{mat}}(a, b, c) = O(\mathcal{T}_{\mathrm{mat}}(a, c, b)) = O(\mathcal{T}_{\mathrm{mat}}(c, a, b))$.

Fact 7.5. *For any $c \geq d > 0$, $\mathcal{T}_{\mathrm{mat}}(a, b, c) \geq \mathcal{T}_{\mathrm{mat}}(a, b, d)$.*

By the property presented in Claim A.2, given $L \subseteq [n]$, we can compute $(Uh)_L$ efficiently using the following lemma.

Lemma 7.6 (Improved running time via tensor trick)**.** *Let U and h be defined same as in Claim A.2, given $L \subseteq [n]$, then computing $(Uh)_L$ takes time*

$$\begin{cases} \mathcal{T}_{\mathrm{mat}}(d, |L|, d) & \text{if } |L| \leq d, \\ (|L|/d) \cdot \mathcal{T}_{\mathrm{mat}}(d, d, d) & \text{otherwise.} \end{cases}$$

Proof. For the case $|L| = d$, it can be computed in $\mathcal{T}_{\mathrm{mat}}(d, d, d) = d^{\omega}$ time. To see that, without loss generality, assume $L = [d]$. By Claim A.2, for each $i \in [d]$, we have $(Uh)_i = (V^{\top} HV)_{i,i}$. Therefore, the computation of $(Uh)_L$ is reduced to computing $V^{\top} HV$ first and then taking the diagonal entries, which takes time d^{ω} that is faster than d^3. In a similar way, For $|L| < d$, we can compute it in $\mathcal{T}_{\mathrm{mat}}(|L|, d, d) + \mathcal{T}_{\mathrm{mat}}(|L|, d, |L|)$ time, which is equal to $\mathcal{T}_{\mathrm{mat}}(d, |L|, d)$ by Fact 7.4 and Fact 7.5. For $|L| > d$, we can divide L into $|L|/d$ groups and each one is reduced to $|L| = d$ case. Thus it can be computed in $|L|/d \cdot \mathcal{T}_{\mathrm{mat}}(d, d, d)$ time.

□

8 Discussion

In this section, we would like to highlight the advantage of our algorithm when input data has Kronecker product structure. The running time of the update stage is $O(n \cdot (d + \log m))$ without Kronecker structure. When one of the m weights changes, say w_r, we need to re-compute the dot product between the updated w_r and each of the n data. Moreover, we need to update all $\log m$ parent nodes of the leaf node with the updated dot product value. With the Kronecker structure, one could utilize the form of the gradient to write the update as learning an $|S_t|$-dimensional coefficients for the diagonal of a covariance matrix of the data, which could be precomputed. Hence, one only needs to read $|S_t|$ entries from the precomputed covariance matrix then linearly combine them with $|S_t|$ coefficients. In contrary, without the Kronecker structure, we no longer have the nice decomposition of gradient updates into a linear combination of diagonals of a covariance matrix, hence we would have to spend d time to recompute the inner product, hence an update time of $O(n(d + \log m))$.

Moreover, we would like to emphasize the importance of getting rid of the factor d in the update time with the Kronecker structure. Note that since we assume the input is from a product space where $d = d_1 \times d_2$, d could be prohibitively large. Even for $d_1 = d_2 = 384$ where there are standard choices of dimensions one encounters in practice, $d = 384^2 \approx 150,000$ which is much larger than individual dimensions. Hence, it's crucial to design an algorithm whose update time does not depend on d at all. On the other hand, $|S_t|$ is the batch size, which could be as small as 1 (for SGD). Thus, in most scenarios, one would prefer an algorithm that depends on $|S_t|$ instead of d. The dependence on d in turn appears at the initialization phase, where we spend $O(mnd^{(\omega-1)/2})$ time to initial the data structure. We would like to examine this runtime under two settings: 1). in practice where $\omega = 3$, our initialization time is $O(mnd)$, we note that this is the size of input data if data is given to us in their Kronecker product form, or the per-iteration runtime of prior works Du et al. (2019b). In contrast, we only need to pay this runtime once instead for each iteration. 2). in theory, the common belief is that $\omega = 2$, in this setting, our initialization time is $O(mnd^{1/2})$, this is precisely the size of input data if we assume $d_1 = d_2 = d^{1/2}$. A simple information theoretical argument would reveal that one has to spend $mnd^{1/2}$ to read the input data. Hence, in this setting, our initialization time is nearly optimal.

REFERENCES

Peyman Afshani and Timothy M Chan. Optimal halfspace range reporting in three dimensions. In *Proceedings of the twentieth annual ACM-SIAM symposium on Discrete algorithms*, pp. 180–186, 2009.

Pankaj K Agarwal, David Eppstein, and Jiri Matousek. Dynamic half-space reporting, geometric optimization, and minimum spanning trees. In *Annual Symposium on Foundations of Computer Science*, volume 33, pp. 80–80, 1992.

Zeyuan Allen-Zhu, Yuanzhi Li, and Zhao Song. A convergence theory for deep learning via over-parameterization. In *ICML*, 2019a.

Zeyuan Allen-Zhu, Yuanzhi Li, and Zhao Song. On the convergence rate of training recurrent neural networks. In *NeurIPS*, 2019b.

Boris Babenko, Ming-Hsuan Yang, and Serge Belongie. Robust object tracking with online multiple instance learning. *IEEE transactions on pattern analysis and machine intelligence*, 33(8):1619–1632, 2010.

Asa Ben-Hur and William Stafford Noble. Kernel methods for predicting protein–protein interactions. *Bioinformatics*, 21(suppl_1):i38–i46, 2005.

Jon Louis Bentley. Multidimensional binary search trees used for associative searching. *Communications of the ACM*, 18(9):509–517, 1975.

Sergei Bernstein. On a modification of chebyshev's inequality and of the error formula of laplace. *Ann. Sci. Inst. Sav. Ukraine, Sect. Math*, 1(4):38–49, 1924.

Markus Bläser. Fast matrix multiplication. *Theory of Computing*, pp. 1–60, 2013.

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

Timothy M Chan. Optimal partition trees. *Discrete & Computational Geometry*, 47(4):661–690, 2012.

Timothy M Chan. Orthogonal range searching in moderate dimensions: kd trees and range trees strike back. *Discrete & Computational Geometry*, 61(4):899–922, 2019.

Beidi Chen, Zichang Liu, Binghui Peng, Zhaozhuo Xu, Jonathan Lingjie Li, Tri Dao, Zhao Song, Anshumali Shrivastava, and Christopher Re. Mongoose: A learnable lsh framework for efficient neural network training. In *International Conference on Learning Representations*, 2020a.

Beidi Chen, Tharun Medini, James Farwell, Charlie Tai, Anshumali Shrivastava, et al. Slide: In defense of smart algorithms over hardware acceleration for large-scale deep learning systems. *Proceedings of Machine Learning and Systems*, 2:291–306, 2020b.

Herman Chernoff. A measure of asymptotic efficiency for tests of a hypothesis based on the sum of observations. *The Annals of Mathematical Statistics*, pp. 493–507, 1952.

Ronan Collobert, Jason Weston, Léon Bottou, Michael Karlen, Koray Kavukcuoglu, and Pavel Kuksa. Natural language processing (almost) from scratch. *Journal of machine learning research*, 12(ARTICLE):2493–2537, 2011.

Navneet Dalal and Bill Triggs. Histograms of oriented gradients for human detection. In *2005 IEEE computer society conference on computer vision and pattern recognition (CVPR'05)*, volume 1, pp. 886–893, 2005.

Amit Daniely. Sgd learns the conjugate kernel class of the network. *Advances in Neural Information Processing Systems*, 30, 2017.

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

Simon S Du, Jason D Lee, Haochuan Li, Liwei Wang, and Xiyu Zhai. Gradient descent finds global minima of deep neural networks. In *International Conference on Machine Learning (ICML)*, 2019a.

Simon S Du, Xiyu Zhai, Barnabas Poczos, and Aarti Singh. Gradient descent provably optimizes over-parameterized neural networks. In *ICLR*, 2019b.

Long Feng and Guang Yang. Deep kronecker network. *arXiv preprint arXiv:2210.13327*, 2022.

Raphael A Finkel and Jon Louis Bentley. Quad trees a data structure for retrieval on composite keys. *Acta informatica*, 4(1):1–9, 1974.

Hongyang Gao, Zhengyang Wang, and Shuiwang Ji. Kronecker attention networks. In *Proceedings of the 26th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, pp. 229–237, 2020.

Jiuxiang Gu, Chenyang Li, Yingyu Liang, Zhenmei Shi, and Zhao Song. Exploring the frontiers of softmax: Provable optimization, applications in diffusion model, and beyond. *arXiv preprint arXiv:2405.03251*, 2024.

Song Han, Huizi Mao, and William J Dally. Deep compression: Compressing deep neural networks with pruning, trained quantization and huffman coding. *arXiv preprint arXiv:1510.00149*, 2015a.

Song Han, Jeff Pool, John Tran, and William Dally. Learning both weights and connections for efficient neural network. *Advances in neural information processing systems*, 28, 2015b.

Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 770–778, 2016.

Wassily Hoeffding. Probability inequalities for sums of bounded random variables. *Journal of the American Statistical Association*, 58(301):13–30, 1963. ISSN 01621459.

Piotr Indyk and Rajeev Motwani. Approximate nearest neighbors: towards removing the curse of dimensionality. In *Proceedings of the thirtieth annual ACM symposium on Theory of computing*, pp. 604–613, 1998.

Arthur Jacot, Franck Gabriel, and Clément Hongler. Neural tangent kernel: Convergence and generalization in neural networks. *arXiv preprint arXiv:1806.07572*, 2018.

Ameya D Jagtap, Yeonjong Shin, Kenji Kawaguchi, and George Em Karniadakis. Deep kronecker neural networks: A general framework for neural networks with adaptive activation functions. *Neurocomputing*, 468:165–180, 2022.

Yuchin Juan, Yong Zhuang, Wei-Sheng Chin, and Chih-Jen Lin. Field-aware factorization machines for ctr prediction. In *Proceedings of the 10th ACM conference on recommender systems*, pp. 43–50, 2016.

Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. *Advances in neural information processing systems*, 25, 2012.

François Le Gall. Powers of tensors and fast matrix multiplication. In *Proceedings of the 39th international symposium on symbolic and algebraic computation (ISSAC)*, pp. 296–303, 2014.

Yann LeCun, Léon Bottou, Yoshua Bengio, and Patrick Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998.

Yuanzhi Li and Yingyu Liang. Learning overparameterized neural networks via stochastic gradient descent on structured data. In *NeurIPS*, 2018.

Jiri Matousek. Efficient partition trees. *Discrete & Computational Geometry*, 8(3):315–334, 1992a.

Jiri Matousek. Reporting points in halfspaces. *Computational Geometry*, 2(3):169–186, 1992b.

Seonwoo Min, Byunghan Lee, and Sungroh Yoon. Deep learning in bioinformatics. *Briefings in bioinformatics*, 18(5):851–869, 2017.

Maxim Naumov, Dheevatsa Mudigere, Hao-Jun Michael Shi, Jianyu Huang, Narayanan Sundaraman, Jongsoo Park, Xiaodong Wang, Udit Gupta, Carole-Jean Wu, Alisson G Azzolini, et al. Deep learning recommendation model for personalization and recommendation systems. *arXiv preprint arXiv:1906.00091*, 2019.

Timo Ojala, Matti Pietikainen, and Topi Maenpaa. Multiresolution gray-scale and rotation invariant texture classification with local binary patterns. *IEEE Transactions on pattern analysis and machine intelligence*, 24(7):971–987, 2002.

Kiran Kumar Patro, Jaya Prakash Allam, Bala Chakravarthy Neelapu, Ryszard Tadeusiewicz, U Rajendra Acharya, Mohamed Hammad, Ozal Yildirim, and Paweł Pławiak. Application of kronecker convolutions in deep learning technique for automated detection of kidney stones with coronal ct images. *Information Sciences*, 640:119005, 2023.

Matthew E. Peters, Mark Neumann, Mohit Iyyer, Matt Gardner, Christopher Clark, Kenton Lee, and Luke Zettlemoyer. Deep contextualized word representations. 2018.

Alec Radford, Karthik Narasimhan, Tim Salimans, and Ilya Sutskever. Improving language understanding by generative pre-training. 2018.

Zhenmei Shi, Junyi Wei, and Yingyu Liang. A theoretical analysis on feature learning in neural networks: Emergence from inputs and advantage over fixed features. In *International Conference on Learning Representations*, 2021.

Zhenmei Shi, Junyi Wei, and Yingyu Liang. Provable guarantees for neural networks via gradient feature learning. *Advances in Neural Information Processing Systems*, 36, 2024.

Chonghao Sima and Yexiang Xue. Lsh-smile: Locality sensitive hashing accelerated simulation and learning. *Advances in Neural Information Processing Systems*, 34, 2021.

Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. In *3rd International Conference on Learning Representations (ICLR)*, 2015.

Aaron Smalter, Jun Huan, and Gerald Lushington. Feature selection in the tensor product feature space. In *2009 Ninth IEEE International Conference on Data Mining*, pp. 1004–1009. IEEE, 2009.

Zhao Song and Xin Yang. Quadratic suffices for over-parametrization via matrix chernoff bound. *arXiv preprint arXiv:1906.03593*, 2019.

Zhao Song, Shuo Yang, and Ruizhe Zhang. Does preprocessing help training over-parameterized neural networks? *Advances in Neural Information Processing Systems*, 34, 2021.

Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, and Andrew Rabinovich. Going deeper with convolutions. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 1–9, 2015.

Virginia Vassilevska Williams. Multiplying matrices faster than coppersmith-winograd. In *Proceedings of the forty-fourth annual ACM symposium on Theory of computing (STOC)*, pp. 887–898, 2012.

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

Yu Zhou, Xiang Bai, Wenyu Liu, and Longin Latecki. Fusion with diffusion for robust visual tracking. *Advances in Neural Information Processing Systems*, 25, 2012.

Appendix

Roadmap. We present the notations and tools used throughout the appendix A. The missing proofs for some technical claims are shown in Section B. In Section C, we complete the proof of convergence theorem (Theorem 6.4). Finally, the asynchronize tree data structure is presented in Section D. We provide the proof of the main theorem in Section E.

A NOTATIONS AND TOOLS

The section organized as follows: In Section A.1, we introduce notations that are used frequently in the appendix. Then, we introduce concentration inequality tools in Section A.2. Finally we present the tensor tools relevant to our research in Section A.3.

A.1 NOTATIONS

For a positive integer n, let $[n]$ represent the set $\{1, 2, \cdots, n\}$. For a matrix $A \in \mathbb{R}^{d \times n}$ and a subset $S \subset [n]$, $A_{i,j}$ is the entry of A at the i-th row and the j-th column, and $A_{:,S}$ represents the matrix whose columns correspond to the columns of A indexed by the set S. Similarly, for a vector $x \in \mathbb{R}^n$, x_S is a vector whose entries correspond to the entries in x indexed by the set S. Let $\|\cdot\|_2$ and $\|\cdot\|_F$ represent the ℓ_2 norm and Frobenius norm respectively. The symbol $\mathbb{1}(\cdot)$ represents the indicator function. For a positive integer d, I_d denotes the $d \times d$ identity matrix. For a random variable X, let $\mathbb{E}[X]$ denote the expectation of X. The symbol $\Pr[\cdot]$ represents probability.

A.2 PROBABILITY TOOL

We state a general concentration inequality tool, which can be viewed as a more general version of Chernoff bound Chernoff (1952) and Hoeffding bound Hoeffding (1963).

Lemma A.1 (Bernstein inequality Bernstein (1924))**.** *Let $X_1, \cdots, X_n$ be independent zero-mean random variables. Suppose that $|X_i| \leq M$ almost surely, for all i. Then, for all positive t,*

$$\Pr[\sum_{i=1}^n X_i > t] \leq \exp(-\frac{t^2/2}{\sum_{j=1}^n \mathbb{E}[X_j^2] + Mt/3}).$$

A.3 TENSOR TOOLS

In this subsection, we state tensor tools utilized in our work. It tells us that given $U \in \mathbb{R}^{n \times d^2}$ and $h \in \mathbb{R}^{d^2}$, the computation of $(Uh)_i$ with $i \in [n]$ has an equivalent way when each row of U has the form $U_{i,:}^\top = \mathrm{vec}(xx^\top)$ for some $x \in \mathbb{R}^d$.

Claim A.2 (Tensor trick)**.** *Given a matrix $H \in \mathbb{R}^{d \times d}$, let $h := \mathrm{vec}(H) \in \mathbb{R}^{d^2}$. Given a matrix $V \in \mathbb{R}^{d \times n}$, the matrix $U \in \mathbb{R}^{n \times d^2}$ is defined satisfying that the i-th row of U is equal to $(\mathrm{vec}(v_i v_i^\top))^\top$, where $v_i \in \mathbb{R}^d$ is the i-th column of V. Then for each $i \in [n]$, it holds that*

$$(V^\top H V)_{i,i} = (U \cdot h)_i.$$

A.4 GENERALIZED TENSOR CASE

More generally, consider the case $x_i = b_i \otimes a_i$ for some $a_i \in \mathbb{R}^p$, $b_i \in \mathbb{R}^q$, and $p \cdot q = d$. Similar to Fact A.2, we have the following statement.

Claim A.3. *Let $A := [a_1, \cdots, a_n] \in \mathbb{R}^{p \times n}$, $B := [b_1, \cdots, b_n] \in \mathbb{R}^{q \times n}$, and $X := [x_1, \cdots, x_n] \in \mathbb{R}^{d \times n}$ such that $x_i = b_i \otimes a_i$ for all $i \in [n]$, then for any $i, j \in [n]$, it holds that $(A^\top \cdot \mathrm{vec}^{-1}(x_i) \cdot B)_{j,j} = (X^\top x_i)_j$.*

Proof. We can show

$$\text{RHS} = x_j^\top x_i$$

$$
\begin{aligned}
&= (b_j \otimes a_j)^\top (b_i \otimes a_i) \\
&= (b_j^\top \otimes a_j^\top)(b_i \otimes a_i) \\
&= (b_j^\top b_i) \otimes (a_j^\top a_i) \\
&= a_j^\top a_i b_i^\top b_j \\
&= \text{LHS},
\end{aligned}
$$

where the third step follows from the fact $(A \otimes B)^\top = A^\top \otimes B^\top$; the fourth step follows from the fact $(A_1 \otimes B_1) \cdot (A_2 \otimes B_2) = (A_1 \cdot A_2) \otimes (B_1 \cdot B_2)$; the last step follows from the fact $x_i = \mathrm{vec}(a_i b_i^\top)$. □

B TECHNICAL PREPARATIONS

This section is organized as follows: In Section B.1, we calculate the upper bound of $\|G_{t,r}\|_2$. Next, we finish the proof of Claim C.4 in Section B.2. Then, We derive bounds for C_1, C_2, C_3, C_4 in Section B.3, B.4, B.5, B.6.

B.1 UPPER BOUND OF $\|G_{t,r}\|_2$

Firstly, we find the upper bound of the norm of $G_{t,r}$.

Lemma B.1. *For any $t \geq 0$ and $r \in [m]$,*

$$
\|G_{t,r}\|_2 \leq \frac{n}{\sqrt{m \cdot S_{\mathrm{batch}}}} \cdot \|u(t) - y\|_2.
$$

Proof. Recall the definition of $G_{t,r}$,

$$
G_{t,r} = \frac{n}{|S_t|} \cdot \frac{1}{\sqrt{m}} \sum_{i \in S_t} (u_i(t) - y_i) \cdot a_r \cdot \mathbb{1}(w_r(t)^\top x_i > \tau) \cdot x_i,
$$

we have

$$
\begin{aligned}
\|G_{t,r}\|_2 &= \frac{n}{S_{\mathrm{batch}}} \cdot \frac{1}{\sqrt{m}} \| \sum_{i \in S_t} (u_i(t) - y_i) \cdot a_r \cdot \mathbb{1}(w_r(t)^\top x_i > \tau) \cdot x_i \|_2 \\
&\leq \frac{n}{S_{\mathrm{batch}}} \cdot \frac{1}{\sqrt{m}} \sum_{i \in S_t} \| (u_i(t) - y_i) \cdot a_r \cdot \mathbb{1}(w_r(t)^\top x_i > \tau) \cdot x_i \|_2 \\
&= \frac{n}{S_{\mathrm{batch}}} \cdot \frac{1}{\sqrt{m}} \sum_{i \in S_t} |u_i(t) - y_i| \cdot |a_r| \cdot |\mathbb{1}(w_r(t)^\top x_i > \tau)| \cdot \|x_i\|_2 \\
&\leq \frac{n}{S_{\mathrm{batch}}} \cdot \frac{1}{\sqrt{m}} \sum_{i \in S_t} |u_i(t) - y_i| \\
&\leq \frac{n}{\sqrt{m}} \cdot \frac{1}{\sqrt{S_{\mathrm{batch}}}} \sqrt{\sum_{i \in S_t} (u_i(t) - y_i)^2} \\
&\leq \frac{n}{\sqrt{m \cdot S_{\mathrm{batch}}}} \cdot \|u(t) - y\|_2,
\end{aligned}
$$

where the second step follows from triangle inequality; the fourth step follows from the facts that $a_r \in \{-1, +1\}$ and $\|x_i\|_2 = 1$; the fifth step follows from Cauchy-Schwarz inequality. □

B.2 PROOF OF CLAIM C.4

Then we finish Claim C.4 deferred to here.

Proof. By the formulations of $v_{1,i}$ and $v_{2,i}$ (see Eq. (16) and Eq. (17)), we have that for each $i \in [n]$,

$$
u_i(t+1) - u_i(t) = v_{1,i} + v_{2,i}.
$$

which has the following vector form

$$u(t+1) - u(t) = v_1 + v_2. \tag{4}$$

For $v_{1,i}$, it holds that

$$\begin{aligned} v_{1,i} &= \frac{1}{\sqrt{m}} \sum_{r \in V_i} a_r (\phi_\tau(w_r(t+1)^\top x_i) - \phi_\tau(w_r(t)^\top x_i)) \\ &= \frac{1}{\sqrt{m}} \sum_{r \in V_i} a_r ((w_r(t+1)^\top x_i - \tau) \cdot \mathbb{1}(w_r(t+1)^\top x_i > \tau) \\ &\quad - (w_r(t)^\top x_i - \tau) \cdot \mathbb{1}(w_r(t)^\top x_i > \tau)). \end{aligned} \tag{5}$$

By Lemma C.1, we can bound the movement of weight vectors, i.e.,

$$\|w_r(t) - w_r(0)\|_2 \leq \gamma.$$

By the definition of the set V_i, for each $r \in V_i$, we have that

$$\mathbb{1}(w_r(t+1)^\top x_i > \tau) = \mathbb{1}(w_r(t)^\top x_i > \tau) = \mathbb{1}(w_r(0)^\top x_i > \tau). \tag{6}$$

Combining Eq. (5) and Eq. (6), we have

$$\begin{aligned} v_{1,i} &= \frac{1}{\sqrt{m}} \sum_{r \in V_i} a_r ((w_r(t+1) - w_r(t))^\top x_i \cdot \mathbb{1}(w_r(t)^\top x_i > \tau)) \\ &= -\frac{1}{\sqrt{m}} \sum_{r \in V_i} a_r \cdot \eta \cdot G_{t,r}^\top x_i \cdot \mathbb{1}(w_r(t)^\top x_i > \tau) \\ &= -\frac{1}{\sqrt{m}} \sum_{r \in V_i} a_r \cdot \eta (\frac{n}{|S_t|} \cdot \frac{1}{\sqrt{m}} \sum_{j \in S_t} (u_j(t) - y_j) \cdot a_r x_j \cdot \mathbb{1}(w_r(t)^\top x_j > \tau))^\top x_i \\ &\quad \cdot \mathbb{1}(w_r(t)^\top x_i > \tau) \\ &= \frac{n}{S_{\text{batch}}} \cdot \frac{\eta}{m} \sum_{r \in V_i} \sum_{j \in S_t} (y_j - u_j(t)) \cdot x_i^\top x_j \cdot \mathbb{1}(w_r(t)^\top x_i > \tau, w_r(t)^\top x_j > \tau) \\ &= \frac{n}{S_{\text{batch}}} \cdot \eta \sum_{j \in S_t} (y_j - u_j(t)) \cdot \frac{1}{m} \sum_{r \in V_i} x_i^\top x_j \cdot \mathbb{1}(w_r(t)^\top x_i > \tau, w_r(t)^\top x_j > \tau) \\ &= \frac{n}{S_{\text{batch}}} \cdot \eta \sum_{j \in S_t} (y_j - u_j(t)) \cdot (H(t)_{i,j} - H(t)_{i,j}^\perp), \end{aligned}$$

where the second step follows from the formulation of $w_r(t+1)$ (see Eq. (1)); the third step follows from the definition of $G_{t,r}$ (see Eq. (2)); the fourth step follows from the fact that a_r is sampled from $\{-1, +1\}$; the last step follows from the definitions of $H(t)_{i,j}$ and $H(t)_{i,j}^\perp$ (see Eq. (18) and Eq. (19)). Then the vector $v_1 \in \mathbb{R}^n$ can be formulated as

$$\begin{aligned} v_1 &= \frac{n}{S_{\text{batch}}} \cdot \eta \cdot [H(t) - H(t)^\perp]_{:,S_t} \cdot [y - u(t)]_{S_t} \\ &= \eta \cdot (H(t) - H(t)^\perp) \cdot D_t \cdot (y - u(t)), \end{aligned} \tag{7}$$

where $D_t \in \mathbb{R}^{n \times n}$ is a diagonal sampling matrix such that the set of indices of nonzero entries is S_t and each nonzero entry is equal to $\frac{n}{S_{\text{batch}}}$.

Now we rewrite $\|y - u(t+1)\|_2^2$ as follows

$$\begin{aligned} &\|y - u(t+1)\|_2^2 \\ &= \|y - u(t) - (u(t+1) - u(t))\|_2^2 \\ &= \|y - u(t)\|_2^2 - 2(y - u(t))^\top (u(t+1) - u(t)) + \|u(t+1) - u(t)\|_2^2. \end{aligned} \tag{8}$$

For the second term in the above equation, it holds that

$$(y - u(t))^\top (u(t+1) - u(t))$$

$$
\begin{aligned}
&= (y - u(t))^\top (v_1 + v_2) \\
&= (y - u(t))^\top v_1 + (y - u(t))^\top v_2 \\
&= \eta (y - u(t))^\top \cdot H(t) \cdot D_t \cdot (y - u(t)) \\
&- \eta (y - u(t))^\top \cdot H(t)^\perp \cdot D_t \cdot (y - u(t)) + (y - u(t))^\top v_2, \qquad (9)
\end{aligned}
$$

where the first step follows from Eq. (4); the third step follows from Eq. (7). Combining Eq. (8) and Eq. (9), and the definitions of quantities C_1, C_2, C_3, and C_4 (see Eq. (20), (21), (22), and (23)), we have

$$\|y - u(t+1)\|_2^2 = \|y - u(t)\|_2^2 + C_1 + C_2 + C_3 + C_4.$$

□

B.3 Bound for C_1

We first introduce the following lemma which is necessary for bounding C_1.

Lemma B.2 (Lemma C.2 in Song et al. (2021))**.** *Let $\tau > 0$ and $\gamma \leq 1/\tau$. Let $c, c' > 0$ be two fixed constants. If $\|w_r(t) - w_r(0)\|_2 \leq \gamma$ holds for each $r \in [m]$, then*

$$\|H(t) - H(0)\|_F \leq n \cdot \min\{c \cdot \exp(-\tau^2/2), 3\gamma\}$$

holds with probability at least $1 - n^2 \cdot \exp(-m \cdot \min\{c' \cdot \exp(-\tau^2/2), \gamma/10\})$.

In addition, the random sampling set S_t has the following fact.

Fact B.3. $\mathbb{E}_{S_t}[D_t] = I$.

Proof. For each $i \in [n]$,

$$\mathbb{E}[(D_t)_{i,i}] = \frac{n}{S_{\text{batch}}} \cdot \frac{\binom{n-1}{S_{\text{batch}}-1}}{\binom{n}{S_{\text{batch}}}} = 1.$$

This completes the proof. □

Now we give the bound for C_1.

Claim B.4. *Let $C_1 = -2\eta (y - u(t))^\top \cdot H(t) \cdot D_t \cdot (y - u(t))$, then*

$$\mathop{\mathbb{E}}_{S_t}[C_1] \leq -2\eta \cdot (3\lambda/4 - 3n\gamma) \cdot \|y - u(t)\|_2^2$$

holds with probability at least $1 - (1/c + \rho + n^2 \cdot \exp(-m \cdot \min\{c' \cdot \exp(-\tau^2/2), \gamma/10\}) + \alpha)$.

Proof. By the definition of C_1, we have

$$
\begin{aligned}
\mathbb{E}[C_1] = & -2\eta (y - u(t))^\top \cdot H(t) \cdot \mathop{\mathbb{E}}_{S_t}[D_t] \cdot (y - u(t)) \\
= & -2\eta (y - u(t))^\top \cdot H(t) \cdot \mathop{\mathbb{E}}_{S_t}[D_t] \cdot (y - u(t)) \\
= & -2\eta (y - u(t))^\top \cdot H(t) \cdot I \cdot (y - u(t)),
\end{aligned}
$$

where the third step follows from Fact B.3.

Lemma C.1 gives us that with probability at least $(1 - 1/c) \cdot (1 - \rho)$ for all $r \in [m]$,

$$\|w_r(t) - w_r(0)\|_2 \leq \gamma. \qquad (10)$$

Combining Eq. (10) and Lemma B.2, we have with probability at least $1 - n^2 \cdot \exp(-m \cdot \min\{c' \cdot \exp(-\tau^2/2), \gamma/10\})$,

$$\|H(0) - H(t)\|_F \leq 3n\gamma. \qquad (11)$$

Moreover, it holds that

$$\|H(0) - H(t)\|_2 \geq \lambda_{\max}(H(0) - H(t)) \geq \lambda_{\min}(H(0)) - \lambda_{\min}(H(t)). \qquad (12)$$

Note that $H(0) = H^{\mathrm{dis}}$, by Lemma 6.2, with probability at least $1 - \alpha$,

$$\lambda_{\min}(H(0)) \geq \frac{3}{4}\lambda. \tag{13}$$

Then Eq. (12) becomes

$$\begin{aligned}\lambda_{\min}(H(t)) &\geq \lambda_{\min}(H(0)) - \|H(0) - H(t)\|_2 \\ &\geq \lambda_{\min}(H(0)) - \|H(0) - H(t)\|_F \\ &\geq 3\lambda/4 - 3n\gamma,\end{aligned}$$

where the second step follows from $\|H(0) - H(t)\|_2 \leq \|H(0) - H(t)\|_F$; the third step follows from Eq. (13) and Eq. (11). Therefore, we have

$$(y - u(t))^\top H(t)(y - u(t)) \geq (3\lambda/4 - 3n\gamma) \cdot \|y - u(t)\|_2^2$$

with probability at least $1 - (1/c + \rho + n^2 \cdot \exp(-m \cdot \min\{c' \cdot \exp(-\tau^2/2), \gamma/10\}) + \alpha)$ by union bound. □

B.4 Bound for C_2

Before bounding C_2, we present the following claim.

Claim B.5 (Claim C.11 in Song et al. (2021)). *Let $\gamma \leq \frac{1}{\tau}$, then for each $r \in [m]$,*

$$\Pr[r \in \overline{V}_i] \leq \min\{\gamma, O(\exp(-\tau^2/2))\}.$$

The following fact gives the upper bound of $\|H(t)^\top\|_F^2$, which is used in the following proof.

Fact B.6.

$$\|H(t)^\perp\|_F^2 \leq \frac{n}{m^2} \sum_{i=1}^n (\sum_{r=1}^m \mathbb{1}(r \in \overline{V}_i))^2.$$

Proof. We have

$$\begin{aligned}\|H(t)^\perp\|_F^2 &= \sum_{i=1}^n \sum_{j=1}^n (H(t)_{i,j}^\perp)^2 \\ &= \sum_{i=1}^n \sum_{j=1}^n (\frac{1}{m} \sum_{r \in \overline{V}_i} x_i^\top x_j \cdot \mathbb{1}(w_r(t)^\top x_i > \tau, w_r(t)^\top x_j > \tau))^2 \\ &= \sum_{i=1}^n \sum_{j=1}^n (\frac{1}{m} \sum_{r=1}^m x_i^\top x_j \cdot \mathbb{1}(w_r(t)^\top x_i > \tau, w_r(t)^\top x_j > \tau) \cdot \mathbb{1}(r \in \overline{V}_i))^2 \\ &= \sum_{i=1}^n \sum_{j=1}^n (\frac{x_i^\top x_j}{m})^2 (\sum_{r=1}^m \mathbb{1}(w_r(t)^\top x_i > \tau, w_r(t)^\top x_j > \tau) \cdot \mathbb{1}(r \in \overline{V}_i))^2 \\ &\leq \frac{1}{m^2} \sum_{i=1}^n \sum_{j=1}^n (\sum_{r=1}^m \mathbb{1}(w_r(t)^\top x_i > \tau, w_r(t)^\top x_j > \tau) \cdot \mathbb{1}(r \in \overline{V}_i))^2 \\ &= \frac{n}{m^2} \sum_{i=1}^n (\sum_{r=1}^m \mathbb{1}(r \in \overline{V}_i))^2.\end{aligned}$$

□

Now we give the bound for C_2.

Claim B.7. *Let $C_2 = 2\eta(y - u(t))^\top \cdot H(t)^\perp \cdot D_t \cdot (y - u(t))$, then*

$$\mathop{\mathbb{E}}_{S_t}[C_2] \leq 6n\eta\gamma \cdot \|y - u(t)\|_2^2$$

holds with probability $1 - n \cdot \exp(-9m\gamma/4)$.

Proof. Similarly as before, we have

$$\begin{aligned}\mathbb{E}[C_2] =& \mathop{\mathbb{E}}_{S_t}[2\eta \cdot (y - u(t))^\top \cdot H(t)^\perp \cdot D_t \cdot (y - u(t))] \\ =& 2\eta \cdot (y - u(t))^\top \cdot H(t)^\perp \cdot \mathop{\mathbb{E}}_{S_t}[D_t] \cdot (y - u(t)) \\ =& 2\eta \cdot (y - u(t))^\top \cdot H(t)^\perp \cdot I \cdot (y - u(t)),\end{aligned}$$

where the last step follows from Fact B.3. Furthermore,

$$\mathbb{E}[C_2] \leq 2\eta \cdot \|H(t)^\perp\|_2 \cdot \|y - u(t)\|_2^2.$$

Therefore, it suffices to upper bound $\|H(t)^\perp\|_2$.

For each $i \in [n]$, we define $Z_i := \sum_{r=1}^m \mathbb{1}(r \in \overline{V}_i)$. Note that the m random variables $\{\mathbb{1}(r \in \overline{V}_i)\}_{r=1}^m$ are mutually independent since $\mathbb{1}(r \in \overline{V}_i)$ only depends on $w_r(0)$. In addition, for each $r \in [m]$, it trivially holds that $|\mathbb{1}(r \in \overline{V}_i)| \leq 1$. By Claim B.5, we have $\mathbb{E}[\mathbb{1}(r \in \overline{V}_i)] \leq \gamma$. In particular,

$$\mathbb{E}[\mathbb{1}(r \in \overline{V}_i)^2] = \mathbb{E}[\mathbb{1}(r \in \overline{V}_i)] \leq \gamma. \tag{14}$$

Applying Bernstein inequality (see Lemma A.1) gives us

$$\begin{aligned}\Pr[Z_i \geq a] \leq& \exp(-\frac{a^2/2}{\sum_{r=1}^m \mathbb{E}[\mathbb{1}(r \in \overline{V}_i)^2] + a/3}) \\ \leq& \exp(-\frac{a^2/2}{mR + a/3}),\end{aligned}$$

where the last step follows from Eq. (14). Setting $a = 3m\gamma$, we have

$$\Pr[Z_i \geq 3m\gamma] \leq \exp(-9m\gamma/4).$$

Moreover, by union bound, we have that

$$\forall i \in [n],\ Z_i \leq 3m\gamma,$$

with probability at least $1 - n \cdot \exp(-9m\gamma/4)$.

By Fact B.6, we know that

$$\begin{aligned}\|H(t)^\perp\|_F^2 \leq& \frac{n}{m^2} \sum_{i=1}^n (\sum_{r=1}^m \mathbb{1}(r \in \overline{V}_i))^2 \\ =& \frac{n}{m^2} \sum_{i=1}^n Z_i^2 \\ \leq& 9n^2\gamma^2,\end{aligned}$$

where the second step follows from the definition of Z_i; the third step follows with probability at least $1 - n \cdot \exp(-9m\gamma/4)$. Furthermore, we have

$$\|H(t)^\perp\|_2 \leq \|H(t)^\perp\|_F \leq 3n\gamma$$

with probability at least $1 - n \cdot \exp(-9m\gamma/4)$. Then we have

$$\mathbb{E}[C_2] \leq 6n\eta\gamma \cdot \|y - u(t)\|_2^2.$$

This completes the proof.

□

B.5 Bound for C_3

In this subsection, we derive the upper bound for C_3 by bounding $\|\mathbb{E}[v_2]\|_2$.

Claim B.8. *Let* $C_3 = -2(y - u(t))^\top v_2$*, then*

$$\mathbb{E}[C_3] \leq \frac{6n^{3/2}\eta\gamma}{\sqrt{S_{\text{batch}}}} \cdot \|y - u(t)\|_2^2$$

with probability at least $1 - n \cdot \exp(-9m\gamma/4)$.

Proof. Using Cauchy-Schwarz inequality, we have

$$\begin{aligned}
\mathbb{E}[C_3] = & -\mathbb{E}[2(y - u(t))^\top v_2] \\
= & -2(y - u(t))^\top \mathbb{E}[v_2] \\
\leq & 2\|y - u(t)\|_2 \cdot \|\mathbb{E}[v_2]\|_2.
\end{aligned}$$

We can upper bound $\|\mathbb{E}[v_2]\|_2$ in the following sense

$$\begin{aligned}
\|\mathbb{E}[v_2]\|_2^2 \leq & \sum_{i=1}^n (\frac{\eta}{\sqrt{m}} \sum_{r \in \overline{V}_i} |G_{t,r}^\top x_i|)^2 \\
= & \frac{\eta^2}{m} \sum_{i=1}^n (\sum_{r=1}^m \mathbb{1}(r \in \overline{V}_i) \cdot |G_{t,r}^\top x_i|)^2 \\
\leq & \frac{\eta^2}{m} \cdot \max_{r \in [m]} \|G_{t,r}\|_2^2 \cdot \sum_{i=1}^n (\sum_{r=1}^m \mathbb{1}(r \in \overline{V}_i))^2 \\
\leq & \frac{\eta^2}{m} \cdot (\frac{n}{\sqrt{m \cdot S_{\text{batch}}}} \cdot \|u(t) - y\|_2)^2 \cdot \sum_{i=1}^n (\sum_{r=1}^m \mathbb{1}(r \in \overline{V}_i))^2 \\
\leq & \frac{\eta^2}{m} \cdot (\frac{n}{\sqrt{m \cdot S_{\text{batch}}}} \cdot \|u(t) - y\|_2)^2 \cdot \sum_{i=1}^n (3m\gamma)^2 \\
= & \frac{9\eta^2 n^3 \gamma^2}{S_{\text{batch}}} \cdot \|u(t) - y\|_2^2,
\end{aligned}$$

where the first step follows from the definition of v_2 (see Eq. (17)) and the property of function ϕ_τ; the third step follows from Cauchy-Schwarz inequality and the fact that $\|x_i\|_2 = 1$; the fourth step follows from Lemma B.1; the fifth step follows from the fact that $\sum_{r=1}^m \mathbb{1}(r \in \overline{V}_i) \leq 3m\gamma$ holds with probability at least $1 - n \cdot \exp(-9m\gamma/4)$ which is proven in Claim B.7. □

B.6 Bound for C_4

Now, we calculate the upper bound of C_4 to prepare for the proof of convergence in Section C.

Claim B.9. *Let* $C_4 = \|u(t+1) - u(t)\|_2^2$*, then*

$$C_4 \leq \frac{n^3\eta^2}{S_{\text{batch}}} \cdot \|y - u(t)\|_2^2.$$

Proof. We need to upper bound

$$\begin{aligned}
& \|u(t+1) - u(t)\|_2^2 \\
= & \sum_{i=1}^n (\frac{1}{\sqrt{m}} \sum_{r=1}^m a_r \cdot (\phi_\tau((w_r(t) - \eta \cdot G_{t,r})^\top x_i) - \phi_\tau(w_r(t)^\top x_i)))^2 \\
\leq & \sum_{i=1}^n (\frac{1}{\sqrt{m}} \sum_{r=1}^m |\eta \cdot G_{t,r}^\top \cdot x_i|)^2 \\
\leq & \eta^2 n \cdot \frac{1}{m} \cdot (\sum_{r=1}^m \|G_{t,r}\|_2)^2
\end{aligned}$$

$$\leq \frac{n^3\eta^2}{S_{\text{batch}}} \cdot \|y - u(t)\|_2^2,$$

where the first step follows from the definition of $w_r(t+1)$; the second step follows from the property of shifted ReLU and $a_r \in \{-1, +1\}$; the fourth step follows from triangle inequality; the last step follows from Lemma B.1. □

C PROOF OF CONVERGENCE

In this section, we give the proof of Theorem 6.4. The proof mainly consists of two parts: (1) showing that the weight vector w_r with $r \in [m]$ does not move too far from initialization; (2) showing that as long as the weight vector does not change too much, then the error $\|u(t)-y\|_2$ decays linearly with extra error term. We proceed the proof via a double induction argument, in which we assume these two conditions hold up to iteration t and prove that they also hold simultaneously for iteration $t+1$.

We prove Theorem 6.4 by induction. The base case is $i = 0$ and it is trivially true. Assume that Eq. (3) is true for $0 \leq i \leq t$, then our goal is to prove that Eq. (3) also holds for $i = t+1$.

From the induction hypothesis, we have the following lemma which states that the weight vectors should not change too much.

Lemma C.1. *If Eq.* (3) *holds for* $0 \leq i \leq t$*, then with probability at least* $(1 - 1/c) \cdot (1 - \rho)$ *where* $c > 1$*, it holds that for all* $r \in [m]$*,*

$$\|w_r(t+1) - w_r(0)\|_2 \leq \frac{2\sqrt{c}n}{\lambda\sqrt{m \cdot S_{\text{batch}}}} \cdot \|u(0) - y\|_2 \leq \gamma,$$

where the parameter γ *is determined later.*

Proof. We first define the two events $\mathcal{E}_1$ and $\mathcal{E}_2$ to be

$$\mathcal{E}_1 : \text{Eq. (3) holds for } 0 \leq i \leq t,$$
$$\mathcal{E}_2 : \|u(t) - y\|_2^2 \leq c \cdot (1 - \eta\lambda/2)^t \cdot \|u(0) - y\|_2^2 \text{ holds for } 0 \leq i \leq t,$$

where $c > 1$ is a constant. By Markov's inequality, we have $\Pr[\mathcal{E}_2 \mid \mathcal{E}_1] \geq 1 - 1/c$. Furthermore, it holds that

$$\Pr[\mathcal{E}_2] \geq \Pr[\mathcal{E}_2 \mid \mathcal{E}_1] \cdot \Pr[\mathcal{E}_1] \geq (1 - 1/c) \cdot (1 - \rho). \tag{15}$$

For $t+1$, we have

$$\begin{aligned}
\|w_r(t+1) - w_r(0)\|_2 &= \|\eta \sum_{i=0}^{t} G_{i,r}\|_2 \\
&\leq \eta \sum_{i=0}^{t} \|G_{i,r}\|_2 \\
&\leq \eta \sum_{i=0}^{t} \frac{n}{\sqrt{m \cdot S_{\text{batch}}}} \cdot \|u(i) - y\|_2 \\
&\leq \frac{\sqrt{c}\eta n}{\sqrt{m \cdot S_{\text{batch}}}} \sum_{i=0}^{t} (1 - \eta\lambda/2)^{i/2} \cdot \|u(0) - y\|_2 \\
&\leq \frac{2\sqrt{c}n}{\lambda\sqrt{m \cdot S_{\text{batch}}}} \cdot \|u(0) - y\|_2,
\end{aligned}$$

where the first step follows from Eq. (1) and Eq. (2); the second step follows from triangle inequality; the third step follows from Lemma B.1; the fourth step follows from Eq. (15); the last step follows from the truncated geometric series. □

For the initial error $\|u(0) - y\|_2$, we have the following claim.

Claim C.2 (Claim D.1 in Song et al. (2021)). *For $\beta \in (0,1)$, with probability at least $1-\beta$,*

$$\|y - u(0)\|_2^2 = O(n(1+\tau^2)\log^2(n/\beta)).$$

Next, we calculate the difference of predictions between two consecutive iterations. For each $i \in [n]$, we have

$$\begin{aligned} & u_i(t+1) - u_i(t) \\ = & \frac{1}{\sqrt{m}} \sum_{r=1}^m a_r \cdot (\phi_\tau(w_r(t+1)^\top x_i) - \phi_\tau(w_r(t)^\top x_i)) \\ = & \frac{1}{\sqrt{m}} \sum_{r=1}^m a_r \cdot (\phi_\tau(\big(w_r(t) - \eta \cdot G_{t,r}\big)^\top x_i) - \phi_\tau(w_r(t)^\top x_i)). \end{aligned}$$

The right hand side can be divided into two parts: $v_{1,i}$ represents one term that does not change and $v_{2,i}$ represents one term that may change. For each $i \in [n]$, we define the set $V_i \subset [m]$ by

$$V_i := \{r \in [m] : \forall w \in \mathbb{R}^d \text{ such that } \|w - w_r(0)\|_2 \leq \gamma,\ \mathbb{1}(w_r(0)^\top x_i > \tau) = \mathbb{1}(w^\top x_i > \tau)\},$$

and $\overline{V}_i := [m] \setminus V_i$. Then the quantities $v_{1,i}$ and $v_{2,i}$ are defined as follows

$$v_{1,i} := \frac{1}{\sqrt{m}} \sum_{r \in V_i} a_r (\phi_\tau(\big(w_r(t) - \eta \cdot G_{t,r}\big)^\top x_i) - \phi_\tau(w_r(t)^\top x_i)), \tag{16}$$

$$v_{2,i} := \frac{1}{\sqrt{m}} \sum_{r \in \overline{V}_i} a_r (\phi_\tau(\big(w_r(t) - \eta \cdot G_{t,r}\big)^\top x_i) - \phi_\tau(w_r(t)^\top x_i)). \tag{17}$$

Given the definition of matrix $H(t) \in \mathbb{R}^{n \times n}$ such that

$$H(t)_{i,j} := \frac{1}{m} \sum_{r=1}^m x_i^\top x_j \cdot \mathbb{1}(w_r(t)^\top x_i > \tau, w_r(t)^\top x_j > \tau), \tag{18}$$

we define the matrix $H(t)^\perp \in \mathbb{R}^{n \times n}$ such that

$$H(t)^\perp_{i,j} := \frac{1}{m} \sum_{r \in \overline{V}_i} x_i^\top x_j \cdot \mathbb{1}(w_r(t)^\top x_i > \tau, w_r(t)^\top x_j > \tau). \tag{19}$$

Given $H(t), H(t)^\perp \in \mathbb{R}^{n \times n}$, we need the following four quantities which are components of $\|y - u(t+1)\|_2^2$.

Definition C.3. Define the quantities C_1, C_2, C_3, and C_4 by

$$C_1 := -2\eta(y - u(t))^\top H(t) \cdot D_t \cdot (y - u(t)), \tag{20}$$

$$C_2 := +2\eta(y - u(t))^\top H(t)^\perp \cdot D_t \cdot (y - u(t)), \tag{21}$$

$$C_3 := -2(y - u(t))^\top v_2, \tag{22}$$

$$C_4 := \|u(t+1) - u(t)\|_2^2, \tag{23}$$

where $D_t \in \mathbb{R}^{n \times n}$ is a diagonal sampling matrix such that the set of indices of the nonzero entries is S_t and each nonzero entry is equal to $\frac{n}{B}$.

Now we can decompose the error term $\|y - u(t+1)\|_2^2$ into the following components and bound them later.

Claim C.4. *The difference between $u(t+1)$ and y can be formulated as*

$$\|y - u(t+1)\|_2^2 = \|y - u(t)\|_2^2 + C_1 + C_2 + C_3 + C_4.$$

The proof for Claim C.4 is deferred to Appendix B.2.

Armed with the above statements, now we prove the convergence theorem. For the sake of completeness, we include Theorem 6.4 below.

Theorem C.5 (Restatement of Theorem 6.4)**.** *Given n training samples $\{(x_i, y_i)\}_{i=1}^n$ and a parameter $\rho \in (0,1)$. Initialize $w_r \sim \mathcal{N}(0, I_d)$ and sample a_r from $\{-1, +1\}$ uniformly at random for each $r \in [m]$. Set the width of neural network to be*

$$m = \mathrm{poly}(\lambda^{-1}, S_{\mathrm{batch}}^{-1}, n, \log(n/\rho)),$$

and the step size $\eta = \mathrm{poly}(\lambda, S_{\mathrm{batch}}, n^{-1})$, where $\lambda = \lambda_{\min}(H^{\mathrm{cts}})$ and S_{batch} is the batch size, then with probability at least $1 - O(\rho)$, the vector $u(t)$ for $t \geq 0$ in Algorithm 2 satisfies that

$$\mathbb{E}[\|u(t) - y\|_2^2] \leq (1 - \eta\lambda/2)^t \cdot \|u(0) - y\|_2^2. \tag{24}$$

Proof. By the linearity of expectation, applying Claim B.4, B.7, B.8, and B.9 gives us

$$\begin{aligned}
&\mathbb{E}[\|y - u(t+1)\|_2^2] \\
=&\|y - u(t)\|_2^2 + \mathbb{E}[C_1] + \mathbb{E}[C_2] + \mathbb{E}[C_3] + \mathbb{E}[C_4] \\
\leq&(1 - 2\eta(3\lambda/4 - 3n\gamma) + 6n\eta\gamma + 6n^{3/2}\eta\gamma/\sqrt{S_{\mathrm{batch}}} + n^3\eta^2/S_{\mathrm{batch}}) \cdot \|y - u(t)\|_2^2.
\end{aligned}$$

Parameter Settings. In order to satisfy Eq. (24) for iteration $t+1$, let

$$1 - 2\eta(3\lambda/4 - 3n\gamma) + 6n\eta\gamma + 6n^{3/2}\eta\gamma/\sqrt{S_{\mathrm{batch}}} + n^3\eta^2/S_{\mathrm{batch}} \leq 1 - \eta\lambda/2. \tag{25}$$

For the probability, we have

$$1/c + n^2 \cdot \exp(-m \cdot \min\{c' \cdot \exp(-\tau^2/2), \gamma/10\}) + \alpha + 2n \cdot \exp(-9m\gamma/4) = O(\rho). \tag{26}$$

Lemma B.2 and Claim B.5 require that

$$\frac{2\sqrt{c}n}{\lambda\sqrt{m \cdot S_{\mathrm{batch}}}} \cdot \|u(0) - y\|_2 \leq \gamma \leq 1/\tau, \tag{27}$$

where Claim C.2 gives that with probability at least $1 - \beta$,

$$\|y - u(0)\|_2^2 = O(n(1 + \tau^2)\log^2(n/\beta)). \tag{28}$$

Eq. (25) implies that the step size η satisfies that

$$\eta = O(\frac{\lambda \cdot S_{\mathrm{batch}}}{n^3})$$

and γ satisfies that

$$\gamma = O(\frac{\lambda}{n}). \tag{29}$$

By setting $\tau = O(\sqrt{\log m})$ and combining Eq. (27), (28) and (29), we have

$$m = \widetilde{\Omega}(\frac{n^5}{\lambda^4 \cdot S_{\mathrm{batch}}}).$$

Taking the probability parameter ρ in Eq. (26) into consideration, we have that

$$m = \widetilde{\Omega}(\frac{n^5 \cdot \log^C(n/\rho)}{\lambda^4 \cdot S_{\mathrm{batch}}}), \tag{30}$$

where $C > 0$ is a constant and the notation $\widetilde{\Omega}(\cdot)$ hides the factors $\log m$ and $\log n$.

Thus, we complete the proof of Theorem 6.4. □

D ASYNCHRONIZE TREE DATA STRUCTURE

In this section, we present the asynchronize tree data structure that has three procedures: INIT, UPDATE, and QUERY, which are shown in Algorithm 3, 4, and 5 respectively.

- The INIT procedure constructs n trees $T_1, \cdots, T_n$ for the n data points $x_1, \cdots, x_n$. The tree T_i with $i \in [n]$ has m leaf nodes and the r-th leaf node with $r \in [m]$ has value $w_r^\top x_i$. The value of each inner node of T_i is the maximum of the values of its two children.
- The UPDATE procedure updates the n trees since the weight vector w_r changes by $\delta_{t,r}$. It starts with the r-th leaf node of each T_i whose value is added by $\delta_{t,r}^\top x_i$, and backtracks until to the root of T_i.
- The QUERY procedure returns the set of activated neurons for data point x_i by searching the values in tree T_i recursively from the root of T_i.

For the asynchronize tree data structure, we have the following theorem and its proof is omitted. Note that the time complexity given in Theorem D.1 is for the general case, i.e., the input data has no special structures. When the data points have Kronecker structure, the time complexity for the initialization step and each iteration can be significantly accelerated.

Theorem D.1 (ASYNCHRONIZETREE data structure)**.** *There exists a data structure with the following procedures:*

- INIT($\{w_1, \cdots, w_m\} \subset \mathbb{R}^d, \{x_1, \cdots, x_n\} \subset \mathbb{R}^d, n \in \mathbb{N}, m \in \mathbb{N}, d \in \mathbb{N}$). *Given a series of weight vectors $w_1, \cdots, w_m$ and data vectors $x_1, \cdots, x_n$ in d-dimensional space, the preprocessing step takes time $O(n \cdot m \cdot d)$.*
- UPDATE($z \in \mathbb{R}^d, r \in [m]$). *Given a vector z and index r, it updates weight vector w_r with z in time $O(n \cdot (d + \log m))$.*
- QUERY($i \in [n], \tau \in \mathbb{R}$). *Given an index i corresponding to point x_i and a threshold τ, it finds all index $r \in [m]$ such that $w_r^\top x_i > \tau$ in time $O(|\widetilde{S}(\tau)| \cdot \log m)$, where $\widetilde{S}(\tau) := \{r : w_r^\top x_i > \tau\}$.*

Algorithm 3 Asynchronize tree data structure

data structure ASYNCHRONIZETREE
members
  $w_1, \cdots, w_m \in \mathbb{R}^d$ ▷ m weight vectors
  $x_1, \cdots, x_n \in \mathbb{R}^d$ ▷ n data points
  Binary trees $T_1, \cdots, T_n$ ▷ n binary search trees
end members

public:
procedure INIT($w_1, \cdots, w_m \in \mathbb{R}^d, x_1, \cdots, x_n \in \mathbb{R}^d, n, m, d$) ▷ INIT in Theorem D.1
  for $i = 1 \to n$ **do**
    $x_i \leftarrow x_i$
  end for
  for $j = 1 \to m$ **do**
    $w_j \leftarrow w_j$
  end for
  for $i = 1 \to n$ **do** ▷ Each data point x_i has a tree T_i
    for $j = 1 \to m$ **do**
      $u_j \leftarrow w_j^\top x_i$
    end for
    $T_i \leftarrow$ MAKETREE($u_1, \cdots, u_m$) ▷ Each node stores the maximum value of its two children
  end for
end procedure
end data structure

Algorithm 4 Asynchronize tree data structure

data structure ASYNCHRONIZETREE
public:
procedure UPDATE($z \in \mathbb{R}^d, r \in [m]$) ▷ UPDATE in Theorem D.1
  for $i = 1$ to n **do**
    $l \leftarrow$ the r-th leaf of tree T_i
    l.value $\leftarrow l$.value $+ z^\top x_i$
    while l is not root **do**
      $p \leftarrow$ parent of l
      p.value $\leftarrow \max\{p$.value, l.value$\}$
      $l \leftarrow p$
    end while
  end for
end procedure
end data structure

Algorithm 5 Asynchronize tree data structure

data structure ASYNCHRONIZETREE
public:
procedure QUERY($i \in [n], \tau \in \mathbb{R}_{\geq 0}$) ▷ QUERY in Theorem D.1
  return QUERYSUB($\tau, \mathrm{root}(T_i)$)
end procedure

private:
procedure QUERYSUB($\tau \in \mathbb{R}_{\geq 0}, r \in T$)
  if r is leaf **then**
    if r.value $> \tau$ **then**
      return r
    end if
  else
    $r_1 \leftarrow$ left child of r, $r_2 \leftarrow$ right child of r
    if r_1.value $> \tau$ **then**
      $S_1 \leftarrow$QUERYSUB(τ, r_1)
    end if
    if r_2.value $> \tau$ **then**
      $S_2 \leftarrow$QUERYSUB(τ, r_2)
    end if
  end if
  return $S_1 \cup S_2$
end procedure
end data structure

E PROOF OF MAIN THEOREM

Theorem E.1 (Formal version of Theorem 1.1)**.** *Given n training samples $\{(x_i, y_i)\}_{i=1}^n$ such that $x_i = \mathrm{vec}(\overline{x}_i \overline{x}_i^\top) \in \mathbb{R}^d$ and $y_i \in \mathbb{R}$ for each $i \in [n]$, there exists an SGD algorithm that can train a two-layer fully connected neural network such that the initialization takes time $O(m \cdot n \cdot d^{\frac{\omega-1}{2}})$ and with high probability each iteration takes time $S_{\mathrm{batch}}^2 \cdot o(m) \cdot n$, where S_{batch} is the batch size, m is the width of the neural network, and ω is the exponent of matrix multiplication.*

Proof. Let $\overline{X} := [\overline{x}_1, \cdots, \overline{x}_n] \in \mathbb{R}^{\sqrt{d} \times n}$ and $X := [\mathrm{vec}(\overline{x}_1 \overline{x}_1^\top), \cdots, \mathrm{vec}(\overline{x}_n \overline{x}_n^\top)] \in \mathbb{R}^{d \times n}$. In the initialization step, there are two main parts: (1) in order to construct the n trees T_i, $i \in [n]$, we need to compute $X^\top w_r$ for all $r \in [m]$; (2) the pairwise inner products $\overline{x}_i^\top \overline{x}_j$ for all $i, j \in [n]$, which will be used in the backward computation.

By Claim A.2, given a fixed $r \in [m]$, we have that $(X^\top w_r)_i = (\overline{X}^\top \cdot \mathrm{vec}^{-1}(w_r) \cdot \overline{X})_{i,i},\ i \in [n]$. By Lemma 7.6, we can compute the matrix $\overline{X}^\top \cdot \mathrm{vec}^{-1}(w_r) \cdot \overline{X}$ and then take its diagonal.

Assume that $n > \sqrt{d}$, then we can compute $X^\top w_r$ in time $(n/\sqrt{d}) \cdot \mathcal{T}_{\mathrm{mat}}(\sqrt{d}, \sqrt{d}, \sqrt{d}) = O(n \cdot d^{\frac{\omega-1}{2}})$. Hence the first part takes time $O(m \cdot n \cdot d^{\frac{\omega-1}{2}})$. In addition, the second part takes time $O(n^2 \cdot \sqrt{d})$. Since $m = \mathrm{poly}(n)$, the initialization step takes time $O(m \cdot n \cdot d^{\frac{\omega-1}{2}})$.

Now we analyze the time complexity of each iteration in Algorithm 2.

Forward Computation. Line 8 takes time $O(Q \cdot \log m)$. Line 9 takes time $O(Q)$. Hence the forward computation takes time $O(S_{\mathrm{batch}} \cdot Q \cdot \log m)$.

Backward Computation. In Line 15, the computation of v_{S_t} and $(y - u(t))_{S_t}$ takes time $O(S_{\mathrm{batch}})$. In Line 16, we need to compute $\delta_{t,r}^\top x_i$ for each $i \in [n]$, in which the core part is to compute the product $X_{:,S_t}^\top x_i$. By Claim A.2, we have that for each $j \in S_t$, $(X_{:,S_t}^\top x_i)_j = (\overline{X}_{:,S_t}^\top \cdot (\overline{x}_i \overline{x}_i^\top) \cdot \overline{X}_{:,S_t})_{j,j}$, which only needs the pairwise products $x_i^\top x_j, i, j \in [n]$ that are already computed in initialization step. Hence, Line 16 takes time $O(n \cdot (S_{\mathrm{batch}} + \log m))$ and the backward computation takes time $O(K \cdot n \cdot (S_{\mathrm{batch}} + \log m))$.

By Lemma 7.3 and setting $\tau = \sqrt{(\log m)/2}$, we have that each iteration takes time

$$\begin{aligned} & O(S_{\mathrm{batch}}^2 \cdot m^{3/4} \cdot n + S_{\mathrm{batch}} \cdot m^{3/4} \cdot n \cdot \log m) \\ = & S_{\mathrm{batch}}^2 \cdot o(m) \cdot n, \end{aligned}$$

which is independent of the data dimensionality d. □

Moreover, we have the following corollary which is a general version of Theorem E.1.

Corollary E.2. *Given n training samples $\{(x_i, y_i)\}_{i=1}^n$ such that for each $i \in [n]$, $x_i = b_i \otimes a_i \in \mathbb{R}^d$ and $a_i \in \mathbb{R}^p$, $b_i \in \mathbb{R}^q$, and $p, q = O(\sqrt{d})$, there exists an SGD algorithm that can train a two-layer fully connected neural network such that the initialization takes time $O(m \cdot n \cdot d^{\frac{\omega-1}{2}})$ and with high probability each iteration takes time $S_{\mathrm{batch}}^2 \cdot o(m) \cdot n$, where S_{batch} is the batch size, m is the width of the neural network, and ω is the exponent of matrix multiplication.*

Proof. Let $A := [a_1, \cdots, a_n] \in \mathbb{R}^{p \times n}$, $B := [b_1, \cdots, b_n] \in \mathbb{R}^{q \times n}$, and $X := [x_1, \cdots, x_n] \in \mathbb{R}^{d \times n}$. Similar to Theorem E.1, in the initialization step, there are two main parts: (1) computing $X^\top w_r$ for all $r \in [m]$ to construct the n trees T_i, $i \in [n]$; (2) computing the pairwise inner products $a_i^\top a_j$, $b_i^\top b_j$ for all $i, j \in [n]$ that will be used in the backward computation.

By Claim A.3, for a fixed $r \in [m]$, it holds that $(X^\top w_r)_i = (A^\top \cdot \mathrm{vec}^{-1}(w_r) \cdot B)_{i,i},\ i \in [n]$. By Lemma 7.6, we can compute the matrix $A^\top \cdot \mathrm{vec}^{-1}(w_r) \cdot B$ and then take its diagonal. Assume that $n = O(\sqrt{d})$, then we can compute $X^\top w_r$ in time

$$O(n/\sqrt{d}) \cdot \mathcal{T}_{\mathrm{mat}}(\sqrt{d}, \sqrt{d}, \sqrt{d}) = O(n \cdot d^{\frac{\omega-1}{2}})$$

since $p, q = O(\sqrt{d})$. Hence the first part takes time $O(m \cdot n \cdot d^{\frac{\omega-1}{2}})$. Furthermore, the second part takes time $O(n^2 \cdot \sqrt{d})$. Since $m = \mathrm{poly}(n)$, the initialization step takes time $O(m \cdot n \cdot d^{\frac{\omega-1}{2}})$.

Now we analyze the time complexity of each iteration in Algorithm 2.

Forward Computation. It is same as the forward computation in Theorem E.1, i.e., $O(S_{\mathrm{batch}} \cdot Q \cdot \log m)$.

Backward Computation. In Line 15, the computation of v_{S_t} and $(y - u(t))_{S_t}$ takes time $O(S_{\mathrm{batch}})$. In Line 16, we need to compute $\delta_{t,r}^\top x_i$ for each $i \in [n]$, in which the core part is to compute the product $X_{:,S_t}^\top x_i$. By Claim A.3, we have that for each $j \in S_t$,

$$(X_{:,S_t}^\top x_i)_j = (A_{:,S_t}^\top \cdot \mathrm{vec}^{-1}(x_i) \cdot B_{:,S_t})_{j,j} = (A_{:,S_t}^\top \cdot a_i \cdot b_i^\top \cdot B_{:,S_t})_{j,j},$$

where the second step follows by $x_i = \mathrm{vec}(a_i b_i^\top)$. Hence we only need to compute the pairwise products $a_i^\top a_j$, $b_i^\top b_j$ for $i, j \in [n]$ which are already computed in initialization step. Then Line 16 takes time $O(n \cdot (S_{\mathrm{batch}} + \log m))$ and the backward computation takes time $O(K \cdot n \cdot (S_{\mathrm{batch}} + \log m))$. Same as Theorem E.1, each iteration takes time $S_{\mathrm{batch}}^2 \cdot o(m) \cdot n$, which is independent of the data dimensionality d. □