Does Preprocessing Help Training Over-parameterized Neural Networks?

Anonymous Author(s)
Affiliation
Address
email

Abstract

Deep neural networks have achieved impressive performance in many areas. Designing a fast and provable method for training neural networks is a fundamental question in machine learning.

The classical training method requires paying $\Omega(mnd)$ cost for both forward computation and backward computation, where m is the width of the neural network, and we are given n training points in d-dimensional space. In this paper, we propose two novel preprocessing ideas to bypass this $\Omega(mnd)$ barrier:

- First, by preprocessing the initial weights of the neural networks, we can train the neural network in $\widetilde{O}(m^{1-\Theta(1/d)}nd)$ cost per iteration.
- Second, by preprocessing the input data points, we can train neural network in $\widetilde{O}(m^{4/5}nd)$ cost per iteration.

From the technical perspective, our result is a sophisticated combination of tools in different fields, greedy-type convergence analysis in optimization, sparsity observation in practical work, high-dimensional geometric search in data structure, concentration and anti-concentration in probability. Our results also provide theoretical insights for a large number of previously established fast training methods.

1 Introduction

During the last decade, deep learning has achieved dominating performance over many areas, e.g., computer vision [LBBH98, KSH12, SLJ+15, HZRS16], natural language processing [CWB+11, DCLT18], automatic driving system, game playing [SHM+16, SSS+17] and beyond. The computational resource requirement for deep neural network training such computation has growing very quickly. Designing a fast and provable training method for neural networks is therefore a fundamental and demanding challenge.

Almost all deep learning models are optimized by gradient descent (or its variants). The total training time can be split in two components, the first one is the number of iterations and the second one is the cost per spent per iteration. Nearly all the iterative algorithms for acceleration can be viewed as two separate lines of research correspondingly, the first line is aiming for an algorithm that has as small as possible number of iterations, the second line is focusing on designing as efficient as possible data structures to improve the cost spent per iteration of the algorithm [Vai89, CLS19, LSZ19, JLSW20, JKL+20]. In this paper, our major focus is on the second line.

There are a number of practical works trying to use a nearest neighbor search data structure to speed up the per-step computation of the deep neural network training [CMF+20, LXJ+20, CLP+21, DMZS21]. However, none of the previous work is able to give a provable guarantee. In this paper, our goal is to develop training algorithms that provably reduce per step time complexity. Let us

Submitted to 35th Conference on Neural Information Processing Systems (NeurIPS 2021). Do not distribute.

consider the ReLU activation neural network and two-layer neural network[1]. Let n denote the number of training data points. Let d denote the dimension of each data point. Let m denote the number of neurons. Suppose we consider the gradient descent algorithm, in each iteration we need to compute prediction for each point in the neural network. Each point $x_i \in \mathbb{R}^d$, requires to compute m inner product in d dimension. Thus, $\Omega(mnd)$ is a natural barrier for cost per iteration in training neural network (in both forward computation and backward computation).

A natural question to ask is

Is it possible to improve the cost per iteration of training neural network algorithm? E.g., is $o(mnd)$ possible?

We list our contributions as follows:

- We provide a new theoretical framework for speeding up neural network training by: 1) adopting the shifted neural tangent kernel; 2) showing that only a small fraction ($o(m)$) of neurons are activated for each input data in each training iteration; 3) identifying the sparsely activated neurons via geometric search; 4) proving that the algorithm can minimize the train loss to zero in a linear convergence rate.
- We provide two theoretical results 1) our first result (Theorem 6.1) builds a dynamic half-space report data structure for the weights of neural network to train neural networks in sublinear cost per iteration; 2) our second result (Theorem 6.2) builds a static half-space report data-structure for the input data points of the training data set for training neural network in sublinear time.

1.1 Related work

Acceleration via high-dimensional search data-structure. High-dimensional search data structures support efficiently finding points in some geometric query regions (e.g., half-spaces, simplices, etc). Currently, there are two main approaches: one is based on Locality Sensitive Hashing (LSH) [IM98], which aims to find the close-by points (i.e., small ℓ_2 distance [DIIM04, AR15, AIL$^+$15, ARN17, Raz17, AIR18, BIW19, DIRW20] or large inner product [SL14, SL15b, SL15a]) of a query $q \in \mathbb{R}^d$ in a given set of points $S \subset \mathbb{R}^d$. This kind of algorithms runs very fast in practice, but most of them only supports approximate queries. Another approach is based on space partitioning data structures, for example, partition trees [Mat92a, Mat92b, AEM92, AC09, Cha12], k-d trees / range trees [CT17, TOG17, Cha19], Voronoi diagrams [ADBMS98, Cha00], which can exactly search the query regions. Recent works have successfully applied high-dimensional geometric data-structure to reduce the complexity of training deep learning models. SLIDE [CMF$^+$20] accelerates the forward pass by retrieving neurons with maximum inner product via a LSH-based data-structure; Reforemer [KKL20] similarly adopts LSH to reduce the memory usage for processing long sequence; MONGOOSE [CLP$^+$21] accelerates the forward pass by retrieving neurons with maximum inner product via a learnable LSH-based data-structure [Cha02] and lazy update framework [CLS19]. Despite the great empirical success, there is no theoretical understanding of such acceleration.

The goal of our paper is to theoretically characterize the acceleration brought by the high-dimensional geometric data-structure. Specifically, our algorithm and analysis are built upon the HSR data structures [AEM92] which can find all the points that have large inner product and support efficient data update. Note that HSR comes with a stronger recovery guarantee than LSH, in the sense that HSR, whereas LSH is guaranteed to find some of those points.

Convergence via over-parameterization. Recently, there has been a tremendous progress in understanding the "small training loss" phenomenon in standard training [LL18, DZPS19, AZLS19a, AZLS19b, DLL$^+$19, ADH$^+$19a, ADH$^+$19b, SY19, OS20, LSS$^+$20, ZPD$^+$20, HLSY21]. A convergence theory has been developed to show that, when randomly initialized, gradient descent and stochastic gradient descent converge to small training loss in polynomially many iterations when the network has polynomial width in terms of the number of training examples.

[1]An alternative name of two-layer neural network is "one-hidden layer neural network"

2 Our techniques

- Empirical works combine high-dimensional search data structures (e.g., LSH) with neural network training, however they do not work theoretically due to the following reasons:
  - Without shifting, the number of activated (and therefore updated) number of neurons is $\Theta(m)$. There is no hope to theoretically prove $o(m)$ complexity (See **Issue 1**).
  - Approximate high-dimensional search data structure might miss some important neurons, which can potentially prevent the training from converge (see **Issue 2**).
- Our solutions are:
  - We propose a shifted ReLU activation that is guaranteed to have $o(m)$ number of activated neurons. Along with the shifted ReLU, we also propose a shifted NTK to rigorously provide convergence guarantee (see **Solution 1**).
  - We adopt an exact high-dimensional search data structure which better couples with the shifted NTK. It takes $o(m)$ time to identify the activated neurons and fits well with the convergence analysis as it avoids missing important neurons (see **Solution 2**).

Issue 1 To speed up the training process, we need the neural network to be "sparse", that is, for each training data $x \in \mathbb{R}^d$, the number of activated neurons is small. Then, in the forward computation, we can just evaluate a small subset of neurons. However, in the previous NTK analysis (e.g., [DZPS19]), the activation function is $\sigma(x) = \max\{\langle w_r, x\rangle, 0\}$, and the weights vectors w_r are initially sampled from a standard d-dimensional Gaussian distribution. Then, by the symmetry of Gaussian distribution, we know that for every input data x, there will be about half of the neurons being activated, which means that we can only obtain a constant factor speedup.

Solution 1 The problem actually comes from the activation function. In practice, people use a shifted ReLU function $\sigma_b(x) = \max\{\langle w_r, x\rangle, b_r\}$ to train neural networks. The main observation of our work is that *threshold implies sparsity*. We consider the setting where all neurons have a unified threshold parameter b. Then, by the concentration of Gaussian distribution, there will be $O(\exp(-b^2) \cdot m)$ activated neurons after the initialization.

The next step is to show that the number of activated neurons will not blow up too much in the following training iterations. In [DZPS19, SY19], they showed that the weights vectors are changing slowly during the training process. In our work, we open the black box of their proof and show a similar phenomenon for the shifted ReLU function. More specifically, a key component is to prove that for each training data, a large fraction of neurons will not change their status (from non-activated to activated and vice versa) in the next iteration with high probability. To achieve this, they showed that this is equivalent to the event that a standard Gaussian random variable in a small centered interval $[-R, R]$, and applied the anti-concentration inequality to upper-bound the probability. In our setting, we need to upper-bound the probability of $z \sim \mathcal{N}(0, 1)$ in a shifted interval $[b - R, b + R]$. On the one hand, we can still apply the anti-concentration inequality by showing that the probability is at most $\Pr[z \in [-R, R]]$. On the other hand, this probability is also upper-bounded by $\Pr[z > b - R]$, and for small R, we can apply the concentration inequality for a more accurate estimation. In the end, by some finer analysis of the probability, we can show that with high probability, the number of activated neurons in each iteration is also $O(\exp(-b^2) \cdot m)$ for each training data. If we take $b = \Theta(\sqrt{\log m})$, we only need to deal with truly sublinear in m of activated neurons in the forward evaluation.

Issue 2: How to find the small subset of activated neurons? A linear scan of the neurons will lead to a time complexity linear in m, which we hope to avoid. Randomly sampling or using LSH for searching can potentially miss important neurons which are important for a rigorous convergence analysis.

Solution 2 Given the shifted ReLU function $\sigma_b(\langle w_r, x\rangle) = \max\{\langle w_r, x\rangle - b, 0\}$, the active neurons are those with weights w_r lying in the half space of $\langle w_r, x\rangle - b > 0$. Finding such neurons is equivalent to a computational geometry problem: given m points in $\mathbb{R}^d$, in each query and a half space $\mathcal{H}$, the goal is to output the points contained in $\mathcal{H}$. Here we use the Half-Space Reporting (HSR) data structure proposed by [AEM92]: after proper initialization, the HSR data structure can return all points lying in the queried half space with complexity as low as $O(\log(n) + k)$, where k

is the number of such points. Note that the HSR data structure well couples with the shifted ReLU, as the number of activated neurons k is truly sublinear in m as per the setting of $b = \Theta(\sqrt{\log m})$.

3 Preliminaries

Notations For an integer n, we use $[n]$ to denote the set $\{1, 2, \cdots, n\}$. For a vector x, we use $\|x\|_2$ to denote the entry-wise ℓ_2 norm of a vector. For a matrix A, we use $\|A\|_F = (\sum_{i,j} A_{i,j}^2)^{1/2}$ to denote its Frobenius norm and use $\|A\|$ to denote the operator/spectral norm of A. We use $x^\top y$ to denote the inner product between vectors x and y. We use I_d to denote d-dimensional identity matrix. We use $\mathcal{N}(0, 1)$ to denote Gaussian distribution with mean 0 and variance 1.

3.1 Problem Formulation

In this section, we introduce the neural network model we study in this paper. We consider a two-layer ReLU activated neural network with m neurons in the hidden layer[2]:

$$f(W, x, a) := \frac{1}{\sqrt{m}} \sum_{r=1}^{m} a_r \sigma_b(\langle w_r, x \rangle),$$

where $x \in \mathbb{R}^d$ is the input, $w_1, \cdots, w_m \in \mathbb{R}^d$ are weight vectors in the first layer, $a_1, \cdots, a_m \in \mathbb{R}$ are weights in the second layer. The ReLU function $\sigma_b(x) := \max\{x - b, 0\}$, where b is the threshold parameter. For simplicity, we only optimize $W \in \mathbb{R}^{d \times m}$ but not $a \in \mathbb{R}^m$.[3] Specifically, we use the following initialization throughout the paper

$$w_r(0) \sim \mathcal{N}(0, I_d); \;\; a_r \sim \mathcal{U}(\{-1, 1\}), \quad \forall r \in [m]. \tag{1}$$

Recall that the ReLU function $\sigma_b(x) = \max\{x - b, 0\}$. Therefore for $r \in [m]$, we have

$$\frac{\partial f(W, x, a)}{\partial w_r} = \frac{1}{\sqrt{m}} a_r x \mathbf{1}_{w_r^\top x \geq b}. \tag{2}$$

We define objective function L as $L(W) = \frac{1}{2} \sum_{i=1}^{n} (y_i - f(W, x_i, a))^2$.

Then, we can apply the gradient descent to optimize the weight matrix W in the following standard way,

$$W(k+1) = W(k) - \eta \frac{\partial L(W(k))}{\partial W(k)}. \tag{3}$$

We can compute the gradient of L in terms of $w_r \in \mathbb{R}^d$

$$\frac{\partial L(W)}{\partial w_r} = \frac{1}{\sqrt{m}} \sum_{i=1}^{n} (f(W, x_i, a) - y_i) a_r x_i \mathbf{1}_{\langle w_r, x_i \rangle \geq 0}. \tag{4}$$

At time t, let $u(t) = (u_1(t), \cdots, u_n(t)) \in \mathbb{R}^n$ be the prediction vector where each $u_i(t)$ is defined as

$$u_i(t) = f(W(t), a, x_i). \tag{5}$$

3.2 Data Structure for Half-Space Reporting

The half-space range reporting problem is an important problem in computational geometry, which is formally defined as follows:

Definition 3.1 (Half-space range reporting)**.** *Given a set S of n points in $\mathbb{R}^d$. There are two operations:*

[2] This is a very standard formulation in the literature, e.g., see [DZPS19, SY19].

[3] We remark, in some previous work, they do choose shift, but their shift is a random shift. In our application, it is important, we fixed the same b for all neurons and never trained.

- QUERY(H): *given a half-space $H \subset \mathbb{R}^d$, output all of the points in S that contain in H, i.e., $S \cap H$.*
- UPDATE: *add or delete a point in S.*
  - INSERT(q): *insert q into S*
  - DELETE(q): *delete q from S*

Let $\mathcal{T}_{\mathsf{init}}$ denote the pre-processing time to build the data structure, $\mathcal{T}_{\mathsf{query}}$ denote the time per query and $\mathcal{T}_{\mathsf{update}}$ time per update.

We use the data-structure proposed in [AEM92] to solve the half-space range reporting problem, which admits the interface summarized in Algorithm 1. Intuitively, the data-structure recursively partition the set S and organizes the points in a tree data-structure. Then for a given query (a, b), all k points of S with $\mathrm{sgn}(\langle a, x\rangle - b) \geq 0$ are reported quickly. Note that the query (a, b) here defines the half-space H in Definition 3.1.

Algorithm 1 Half Space Report Data Structure

```
data structure HALFSPACEREPORT
    procedures:
        INIT(S, n, d)      ▷ Initialize the data structure with a set S of n points in R^d
        QUERY(a, b)        ▷ a, b ∈ R^d. Output the set {x ∈ S : sgn(⟨a, x⟩ − b) ≥ 0}
        ADD(x)             ▷ Add a point x ∈ R^d to S
        DELETE(x)          ▷ Delete the point x ∈ R^d from S
end data structure
```

Adapted from [AEM92], the algorithm comes with the following complexity:

Corollary 3.2 ([AEM92])**.** *Given a set of n points in $\mathbb{R}^d$, the half-space reporting problem can be solved with the following performances:*

- *Part 1.*[4] $\mathcal{T}_{\mathsf{query}}(n, d, k) = O_d(n^{1-1/\lfloor d/2\rfloor} + k)$*, amortized* $\mathcal{T}_{\mathsf{update}} = O_d(\log^2(n))$.
- *Part 2.*[5] $\mathcal{T}_{\mathsf{query}}(n, d, k) = O_d(\log(n) + k)$*, amortized* $\mathcal{T}_{\mathsf{update}} = O_d(n^{\lfloor d/2\rfloor - 1})$.

3.3 Sparsity-based Characterizations

In this section, we consider the ReLU function with a nonzero threshold: $\sigma_b(x) = \max\{0, x - b\}$, which is commonly seen in practise, and also has been considered in theoretical work [ZPD+20].

We first define the set of neurons that are firing at time t.

Definition 3.3 (fire set)**.** *For each $i \in [n]$, for each $t \in \{0, 1, \cdots, T\}$, let $\mathcal{S}_{i,\mathrm{fire}}(t) \subset [m]$ denote the set of neurons that are "fire" at time t, i.e.,*

$$\mathcal{S}_{i,\mathrm{fire}}(t) := \{r \in [m] : \langle w_r(t), x_i\rangle > b\}.$$

We define $k_{i,t} := |\mathcal{S}_{i,\mathrm{fire}}(t)|$, $\forall t \in \{0, 1, \cdots, T\}$.

The following lemma shows that σ_b gives the desired sparsity.

Lemma 3.4 (Sparsity after initialization)**.** *Let $b > 0$ be a tunable parameter. If we use the σ_b as the activation function, then after the initialization, with probability at least $1 - n \cdot \exp(-\Omega(m \cdot \exp(-b^2/2)))$, it holds that for each input data x_i, the number of activated neurons $k_{i,0} \leq O(m \cdot \exp(-b^2/2))$, where m is the total number of neurons.*

Proof. By the concentration of Gaussian distribution, the initial fire probability of a single neuron is

$$\Pr[\sigma_b(\langle w_r(0), x_i\rangle) > 0] = \Pr_{z \sim \mathcal{N}(0,1)}[z > b] \leq \exp(-b^2/2).$$

[4] Used in Theorem 6.1

[5] Used in Theorem 6.2

Algorithm 2 Training Neural Network via building a data structure of weights of the neural network

procedure TrainingWithPreprocessWeights($\{(x_i, y_i)\}_{i\in[n]}$,n,m,d) ▷ Theorem 6.1
  Initialize w_r, a_r for $r \in [m]$ and b according to Equation (1) and Remark 3.5
  HalfSpaceReport hsr.Init($\{w_r(0)\}_{r\in[m]}, m, d$) ▷ Algorithm 1
  for $t = 1 \to T$ **do**
    $S_{i,\text{fire}} \leftarrow$ hsr.Query(x_i, b) for $i \in [n]$
    Forward pass for x_i only on neurons in $S_{i,\text{fire}}$ for $i \in [n]$
    Calculate gradient for x_i only on neurons in $S_{i,\text{fire}}$ for $i \in [n]$
    Gradient update for the neurons in $\cup_{i\in[n]} S_{i,\text{fire}}$
    hsr.Delete($w_r(t)$) for $r \in \cup_{i\in[n]} S_{i,\text{fire}}$
    hsr.Add($w_r(t+1)$) for $r \in \cup_{i\in[n]} S_{i,\text{fire}}$
  end for
  return Trained weights $w_r(T+1)$ for $r \in [m]$
end procedure

Hence, for the indicator variable $\mathbf{1}_{r\in\mathcal{S}_{i,\text{fire}}(0)}$, we have

$$\mathbb{E}[\mathbf{1}_{r\in\mathcal{S}_{i,\text{fire}}(0)}] \le \exp(-b^2/2).$$

By Bernstein inequality (Lemma B.3) we have for all $t > 0$,

$$\Pr\left[|\mathcal{S}_{i,\text{fire}}(0)| > k_0 + t\right] \le \exp\left(-\frac{t^2/2}{k_0 + t/3}\right), \tag{6}$$

where $k_0 := m \cdot \exp(-b^2/2)$. If we choose $t = k_0$, then we have:

$$\Pr\left[|\mathcal{S}_{i,\text{fire}}(0)| > 2k_0\right] \le \exp\left(-3k_0/8\right)$$

Then, by union bound over all $i \in [n]$, we have that with high probability

$$1 - n \cdot \exp(-\Omega(m \cdot \exp(-b^2/2))),$$

the number of initial fire neurons for the sample x_i is bounded by $k_{i,0} \le 2m \cdot \exp(-b^2/2)$. □

The following remark gives an example of setting the threshold b, and will be useful for showing the sublinear complexity in the next section.

Remark 3.5. *If we choose $b = \sqrt{0.4 \log m}$ then $k_0 = m^{4/5}$. For $t = m^{4/5}$, Eq. (6) implies that*

$$\Pr\left[|\mathcal{S}_{i,\text{fire}}(0)| > 2m^{4/5}\right] \le \exp\left(-\min\{mR, O(m^{4/5})\}\right).$$

4 Training Neural Network with Half-Space Reporting Data Structure

In this section, we present two sublinear time algorithms for training over-parameterized neural networks. The first algorithm (Section 4.1) relies on building a high-dimensional search data-structure for the weights of neural network. The second algorithm (Section 4.2) is based on building a data structure for the input data points of the training set. Both of the algorithms use the HSR to quickly identify the fired neurons to avoid unnecessary calculation. The time complexity and the sketch of the proof are provided after each of the algorithms.

4.1 Weights Preprocessing

We first introduce the algorithm that preprocesses the weights w_r for $r \in [m]$, which is commonly used in practice [CLP+21, CMF+20, KKL20]. Recall the two-layer ReLU neural network is defined as $f(W, x, a) := \frac{1}{\sqrt{m}} \sum_{r=1}^m a_r \sigma_b(\langle w_r, x\rangle)$. By constructing a HSR data-structure for w_rs, we can quickly find the set of active neurons $S_{i,\text{fire}}$ for each of the training sample x_i. See pseudo-code in Algorithm 2.

In the remaining part of this section, we focus on the time complexity analysis of Algorithm 2. The convergence proof will be given in Section 5.

Algorithm 3 Training Neural Network via building a data-structure of the training data

1: **procedure** TRAININGWITHPROCESSDATA($\{(x_i, y_i)\}_{i\in[n]}$,n,m,d) ▷ Theorem 6.2
2: Initialize w_r, a_r for $r \in [m]$ and b according to Equation (1) and Remark 3.5
3: HALFSPACEREPORT HSR.INIT($\{x_i\}_{i\in[n]}, n, d$) ▷ Algorithm 1
4: $\widetilde{S}_{r,\text{fire}} \leftarrow$ HSR.QUERY($w_r(0), b$) for $r \in [m]$ ▷ $\widetilde{S}_{r,\text{fire}}$ are samples which neuron r fires for
5: $S_{i,\text{fire}} \leftarrow \{r \mid i \in \widetilde{S}_{r,\text{fire}}\}$ ▷ $S_{i,\text{fire}}$ is the set of neurons, which fire for x_i
6: **for** $t = 1 \to T$ **do**
7: Forward pass for x_i only on neurons in $S_{i,\text{fire}}$ for $i \in [n]$
8: Calculate gradient for x_i only on neurons in $S_{i,\text{fire}}$ for $i \in [n]$
9: Gradient update for the neurons in $\cup_{i\in[n]} S_{i,\text{fire}}$
10: **for** $r \in \cup_{i\in[n]} \mathcal{S}_{i,\text{fire}}$ **do**
11: $S_{i,\text{fire}}$.DEL(r) for $i \in \widetilde{S}_{r,\text{fire}}$
12: $\widetilde{S}_{r,\text{fire}} \leftarrow$ HSR.QUERY($w_r(t+1), b$)
13: $S_{i,\text{fire}}$.ADD(r) for $i \in \widetilde{S}_{r,\text{fire}}$
14: **end for**
15: **end for**
16: **return** Trained weights $w_r(T+1)$ for $r \in [m]$
17: **end procedure**

Lemma 4.1 (Running time part of Theorem 6.1)**.** *Given n data points in d-dimensional space. Running gradient descent algorithm (Algorithm 2) on a two-layer ReLU (over-parameterized) neural network with $b = \sqrt{0.4 \log m}$, the expected per-iteration running time of the gradient descent algorithm is $\widetilde{O}(m^{1-\Theta(1/d)} nd)$.*

Proof. The per-step time complexity is

$$\sum_{i=1}^{n} \mathcal{T}_{\text{QUERY}}(m, d, k_{i,t}) + (\mathcal{T}_{\text{DELETE}} + \mathcal{T}_{\text{INSERT}}) \cdot |\cup_{i\in[n]} S_{i,\text{fire}}(t)| + d \sum_{i\in[n]} k_{i,t}$$

The first term $\sum_{i=1}^{n} \mathcal{T}_{\text{QUERY}}(m, d, k_{i,t})$ corresponds to the running time of querying the active neuron set $S_{i,\text{fire}}(t)$ for all training samples $i \in [n]$. With the first result in Corollary B.7, the complexity is bounded by $\widetilde{O}(m^{1-\Theta(1/d)} nd)$.

The second term $(\mathcal{T}_{\text{DELETE}} + \mathcal{T}_{\text{INSERT}}) \cdot |\cup_{i\in[n]} S_{i,\text{fire}}(t)|$ corresponds to updating w_r in the high-dimensional search data-structure (Lines 9 and 10). Again with the first result in Corollary B.7, we have $\mathcal{T}_{\text{DELETE}} + \mathcal{T}_{\text{INSERT}} = O(\log^2 m)$. Combining with the fact that $|\cup_{i\in[n]} S_{i,\text{fire}}(t)| \leq |\cup_{i\in[n]} S_{i,\text{fire}}(0)| \leq O(nm^{4/5})$, the second term is bounded by $O(nm^{4/5} \log^2 m)$.

The third term is the time complexity of gradient calculation restricted to the set $\mathcal{S}_{i,\text{fire}}(t)$. With the bound on $\sum_{i\in[n]} k_{i,t}$ (Lemma C.9), we have $d \sum_{i\in[n]} k_{i,t} \leq O(m^{4/5} nd)$.

Putting them together completes the proof. □

4.2 Data Preprocessing

While the weights preprcessing algorithm is inspired by the common practise, the dual relationship between the input x_i and model weights w_r inspires us to preprocess the dataset before training (i.e., building HSR data-structure for x_i). This largely improves the per-iteration complexity and avoids the frequent updates of the data structure since the training data is fixed. More importantly, once the training dataset is preprocessed, it can be reused for different models or tasks, thus one does not need to perform the expensive preprocessing for each training.

The corresponding pseudocode is presented in Algorithm 3. With x_i preprocessed, we can query HSR with weights w_r and the result $\widetilde{S}_{r,\text{fire}}$ is the set of training samples x_i for which w_r fires for. Given $\widetilde{S}_{r,\text{fire}}$ for $r \in [m]$, we can easily reconstruct the set $S_{i,\text{fire}}$, which is the set of neurons fired for sample x_i. The forward and backward pass can then proceed similar to Algorithm 2.

At the end of each iteration, we will update $\widetilde{S}_{r,\text{fire}}$ based on the new w_r estimation and update $S_{i,\text{fire}}$ accordingly. For Algorithm 3, the HSR data-structure is static for the entire training process. This is the main difference from Algorithm 2, where the HSR needs to be updated every time step to account for the changing weights w_r.

We defer the convergence analysis to Section 5 and focus on the time complexity analysis of Algorithm 2 in the rest of this section. We consider d being a constant for the rest of this subsection.

Lemma 4.2 (Running time part of Theorem 6.2)**.** *Given n data points in d-dimensional space. Running gradient descent algorithm (Algorithm 2) on a two-layer ReLU (over-parameterized) neural network with $b = \sqrt{0.4 \log m}$, the expected per-iteration running time of initializing $\widetilde{S}_{r,\text{fire}}, S_{i,\text{fire}}$ for $r \in [m], i \in [n]$ is $O(m \log n + m^{4/5} n)$. The cost per iteration of the training algorithm is $O(m^{4/5} n \log n)$.*

Proof. We analyze the initialization and training parts separately.

Initialization In Lines 4 and 5, the sets $\widetilde{S}_{r,\text{fire}}, S_{i,\text{fire}}$ for $r \in [m], i \in [n]$ are initialized. For each $r \in [m]$, we need to query the data structure the set of data points x's such that $\sigma_b(w_r(0)^\top x) > 0$. Hence, the running time of this step is

$$\begin{aligned}\sum_{r=1}^m \mathcal{T}_{\mathsf{query}}(n, d, \widetilde{k}_{r,0}) &= O(m \log n + \sum_{r=1}^m \widetilde{k}_{r,0}) \\ &= O(m \log n + \sum_{i=1}^n k_{i,0}) \\ &= O(m \log n + m^{4/5} n).\end{aligned}$$

where the second step follows from $\sum_{r=1}^m \widetilde{k}_{r,0} = \sum_{i=1}^n k_{i,0}$.

Training Consider training the neural network for T steps. For each step, first notice that the forward and backward computation parts (Line 7 - 9) are the same as previous algorithm. The time complexity is $O(m^{4/5} n \log n)$.

We next show that maintaining $\widetilde{S}_{r,\text{fire}}, r \in [m]$ and $S_{i,\text{fire}}, i \in [n]$ (Line 10-14) takes $O(m^{4/5} n \log n)$ time. For each fired neuron $r \in [m]$, we first remove the indices of data in the sets $S_{i,\mathsf{fire}}$, which takes time

$$O(1) \cdot \sum_{r \in \cup_{i \in [n]} S_{i,\text{fire}}} \widetilde{k}_{r,t} = O(1) \cdot \sum_{r=1}^m \widetilde{k}_{r,t} = O(m^{4/5} n).$$

Then, we find the new set of x's such that $\sigma_b(\langle w_r(t+1), x \rangle) > 0$ by querying the half-space reporting data structure. The total running time for all fired neurons is

$$\sum_{r \in \cup_{i \in [n]} S_{i,\text{fire}}} \mathcal{T}_{\mathsf{query}}(n, d, \widetilde{k}_{r,t+1}) \lesssim m^{4/5} n \log n + \sum_{r \in \cup_{i \in [n]} S_{i,\text{fire}}} \widetilde{k}_{r,t+1} = O(m^{4/5} n \log n)$$

Then, we update the index sets $S_{i,\text{fire}}$ in time $O(m^{4/5} n)$. Therefore, each training step takes $O(m^{4/5} n \log n)$ time, which completes the proof. $\square$

5 Convergence Analysis of Our Algorithm

In this section, we state the convergence result of our training neural network algorithms (Lemma 5.2). An important component in our proof is to show that the smallest eigenvalue of the continuous Hessian matrix $\lambda_{\min}(H^{\text{cts}})$ cannot be too small. It turns out to be an anti-concentration problem of Gaussian random matrix. In [OS20], they gave a lower bound on $\lambda_{\min}(H^{\text{cts}})$ for ReLU function with $b = 0$, assuming the input data are separable. One of our major technical contribution is generalizing it to arbitrary $b \geq 0$.

Proposition 5.1 (Informal version of Theorem F.1). *Given n input data $\{x_1, x_2, \cdots, x_n\} \subseteq \mathbb{R}^d$ such that $\forall i \in [n], \|x_i\|_2 = 1$ and $\delta := \min_{i \neq j}\{\|x_i - x_j\|_2, \|x_i + x_j\|_2\}$. For any $b \geq 0$, we define $H^{\mathrm{cts}} \in \mathbb{R}^{n \times n}$ as follows $H^{\mathrm{cts}}_{i,j} := \mathbb{E}_{w \sim \mathcal{N}(0, I_d)} \left[\langle x_i, x_j \rangle \mathbf{1}_{\langle w, x_i \rangle \geq b, \langle w, x_j \rangle \geq b} \right], \forall i \in [n], j \in [n]$. Then*

$$\lambda_{\min}(H^{\mathrm{cts}}) \geq 0.01 e^{-b^2/2} \delta / n^2.$$

With proposition 5.1, we are ready to show the convergence rate of over-parameterized neural network with shifted ReLU function.

Lemma 5.2 (Convergence part of Theorem 6.1 and Theorem 6.2). *Suppose input data-points are δ-separable, i.e., $\delta := \min_{i \neq j}\{\|x_i - x_j\|_2, \|x_i + x_j\|_2\}$. Let $m = \mathrm{poly}(n, 1/\delta, \log(n/\rho))$, we i.i.d. initialize $w_r \in \mathcal{N}(0, I_d)$, a_r sampled from $\{-1, +1\}$ uniformly at random for $r \in [m]$, $b = \Theta(\sqrt{\log m})$, and we set the step size $\eta = O(\lambda/n^2)$ then with probability at least $1 - \rho$ over the random initialization we have for $k = 0, 1, 2, \cdots, T$* [6],

$$\|u(k) - y\|_2^2 \leq (1 - \eta\lambda/2)^k \cdot \|u(0) - y\|_2^2. \tag{7}$$

This result shows that despite the shifted ReLU and sparsely activated neurons, we can still retain the linear convergence. Combined with the results on per-step complexity in the previous section, it gives our main theoretical results of training deep learning models with sublinear time complexity (Theorem 6.1 and Theorem 6.2).

6 Main Theorems

In this section, we state the main theorems of our work, showing the sublinear running time and linear convergence rate of our two algorithms. The first algorithm is relying on building a high-dimensional geometric search data-structure for the weights of neural network.

Theorem 6.1 (Main result I, informal of Theorem E.2). *Given n data points in d-dimensional space. We preprocess the initialization weights of the neural network. Running gradient descent algorithm (Algorithm 2) on a two-layer ReLU (over-parameterized) neural network with m neurons in the hidden layers is able to minimize the training loss to zero and the expected running time of gradient descent algorithm (per iteration)*

$$\widetilde{O}(m^{1-\Theta(1/d)} nd).$$

The second algorithm is based on building a data structure for the input data points of the training set. Our second algorithm can further reduce the cost per iteration from $m^{1-1/d}$ to truly sublinear in m, e.g. $m^{4/5}$.

Theorem 6.2 (Main result II, informal of Theorem E.2). *Given n data points in d-dimensional space. We preprocess all the data points. Running gradient descent algorithm (Algorithm 3) on a two-layer ReLU (over-parameterized) neural network with m neurons in the hidden layers is able to minimize the training loss to zero, and the expected running time of gradient descent algorithm (per iteration)*

$$\widetilde{O}(m^{4/5} nd).$$

7 Discussion and Limitations

In this paper, we propose two sublinear algorithms to train neural networks. By preprocessing the weights of the neuron networks or preprocessing the training data, we rigorously prove that it is possible to train a neuron network with sublinear complexity, which overcomes the $\Omega(mnd)$ barrier in classical training methods. Our results also offer theoretical insights for many previously established fast training methods.

One limitation of our work is that the current analysis framework does not provide convergence guarantee for combining LSH with gradient descent, which is commonly seen in many empirical works. Our proof breaks as LSH might miss important neurons which potentially ruins the convergence analysis. Instead, we refer to the HSR data-structure, which provides a stronger theoretical guarantee of successfully finding all fired neurons.

[6] Eventually, we choose $T = \lambda^{-2} n^2 \log(n/\epsilon)$ where ϵ is the final accuracy.

References

[AC09] 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*, pages 180–186. SIAM, 2009.

[ADBMS98] Pankaj K Agarwal, Mark De Berg, Jiri Matousek, and Otfried Schwarzkopf. Constructing levels in arrangements and higher order voronoi diagrams. *SIAM journal on computing*, 27(3):654–667, 1998.

[ADH+19a] Sanjeev Arora, Simon Du, Wei Hu, Zhiyuan Li, and Ruosong Wang. Fine-grained analysis of optimization and generalization for overparameterized two-layer neural networks. In *International Conference on Machine Learning*, pages 322–332, 2019.

[ADH+19b] Sanjeev Arora, Simon S Du, Wei Hu, Zhiyuan Li, Ruslan Salakhutdinov, and Ruosong Wang. On exact computation with an infinitely wide neural net. In *NeurIPS*. https://arxiv.org/pdf/1904.11955.pdf, 2019.

[AEM92] Pankaj K Agarwal, David Eppstein, and Jirí Matousek. Dynamic half-space reporting, geometric optimization, and minimum spanning trees. In *Annual Symposium on Foundations of Computer Science (FOCS)*, volume 33, pages 80–80, 1992.

[AIL+15] Alexandr Andoni, Piotr Indyk, TMM Laarhoven, Ilya Razenshteyn, and Ludwig Schmidt. Practical and optimal lsh for angular distance. In *Advances in Neural Information Processing Systems (NIPS)*, pages 1225–1233. Curran Associates, 2015.

[AIR18] Alexandr Andoni, Piotr Indyk, and Ilya Razenshteyn. Approximate nearest neighbor search in high dimensions. *arXiv preprint arXiv:1806.09823*, 7, 2018.

[AR15] Alexandr Andoni and Ilya Razenshteyn. Optimal data-dependent hashing for approximate near neighbors. In *Proceedings of the forty-seventh annual ACM symposium on Theory of computing (STOC)*, pages 793–801, 2015.

[ARN17] Alexandr Andoni, Ilya Razenshteyn, and Negev Shekel Nosatzki. Lsh forest: Practical algorithms made theoretical. In *Proceedings of the Twenty-Eighth Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 67–78. SIAM, 2017.

[AZLS19a] Zeyuan Allen-Zhu, Yuanzhi Li, and Zhao Song. A convergence theory for deep learning via over-parameterization. In *ICML*, 2019.

[AZLS19b] Zeyuan Allen-Zhu, Yuanzhi Li, and Zhao Song. On the convergence rate of training recurrent neural networks. In *NeurIPS*, 2019.

[Ber24] 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.

[BIW19] Arturs Backurs, Piotr Indyk, and Tal Wagner. Space and time efficient kernel density estimation in high dimensions. 2019.

[Cha00] Timothy M Chan. Random sampling, halfspace range reporting, and construction of ($\leq k$)-levels in three dimensions. *SIAM Journal on Computing*, 30(2):561–575, 2000.

[Cha02] Moses S Charikar. Similarity estimation techniques from rounding algorithms. In *Proceedings of the thiry-fourth annual ACM symposium on Theory of computing (STOC)*, pages 380–388, 2002.

[Cha12] Timothy M Chan. Optimal partition trees. *Discrete & Computational Geometry*, 47(4):661–690, 2012.

[Cha19] Timothy M Chan. Orthogonal range searching in moderate dimensions: kd trees and range trees strike back. *Discrete & Computational Geometry*, 61(4):899–922, 2019.

[Che52] Herman Chernoff. A measure of asymptotic efficiency for tests of a hypothesis based on the sum of observations. *The Annals of Mathematical Statistics*, pages 493–507, 1952.

[CLP+21] 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 *ICLR oral*, 2021.

[CLS19] Michael B Cohen, Yin Tat Lee, and Zhao Song. Solving linear programs in the current matrix multiplication time. In *STOC*, 2019.

[CMF+20] Beidi Chen, Tharun Medini, James Farwell, Sameh Gobriel, Charlie Tai, and Anshumali Shrivastava. Slide: In defense of smart algorithms over hardware acceleration for large-scale deep learning systems. In *In Proceedings of the 3rd Conference on Machine Learning and Systems (MLSys)*, 2020.

[CT17] Timothy M Chan and Konstantinos Tsakalidis. Dynamic orthogonal range searching on the ram, revisited. *Leibniz International Proceedings in Informatics, LIPIcs*, 77:281–2813, 2017.

[CWB+11] 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.

[DCLT18] 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.

[DIIM04] Mayur Datar, Nicole Immorlica, Piotr Indyk, and Vahab S Mirrokni. Locality-sensitive hashing scheme based on p-stable distributions. In *Proceedings of the twentieth annual symposium on Computational geometry (SoCG)*, pages 253–262, 2004.

[DIRW20] Yihe Dong, Piotr Indyk, Ilya Razenshteyn, and Tal Wagner. Learning space partitions for nearest neighbor search. In *ICLR*. arXiv preprint arXiv:1901.08544, 2020.

[DLL+19] Simon S Du, Jason D Lee, Haochuan Li, Liwei Wang, and Xiyu Zhai. Gradient descent finds global minima of deep neural networks. In *ICML*. https://arxiv.org/pdf/1811.03804, 2019.

[DMZS21] Shabnam Daghaghi, Nicholas Meisburger, Mengnan Zhao, and Anshumali Shrivastava. Accelerating slide deep learning on modern cpus: Vectorization, quantizations, memory optimizations, and more. *Proceedings of Machine Learning and Systems*, 3, 2021.

[DZPS19] Simon S Du, Xiyu Zhai, Barnabas Poczos, and Aarti Singh. Gradient descent provably optimizes over-parameterized neural networks. In *ICLR*, 2019.

[HLSY21] Baihe Huang, Xiaoxiao Li, Zhao Song, and Xin Yang. Fl-ntk: A neural tangent kernel-based framework for federated learning convergence analysis. In *ICML*, 2021.

[Hoe63] Wassily Hoeffding. Probability inequalities for sums of bounded random variables. *Journal of the American Statistical Association*, 58(301):13–30, 1963.

[HZRS16] 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 (CVPR)*, pages 770–778, 2016.

[IM98] 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 (STOC)*, pages 604–613, 1998.

[JKL+20] Haotian Jiang, Tarun Kathuria, Yin Tat Lee, Swati Padmanabhan, and Zhao Song. A faster interior point method for semidefinite programming. In *FOCS*, 2020.

[JLSW20] Haotian Jiang, Yin Tat Lee, Zhao Song, and Sam Chiu-wai Wong. An improved cutting plane method for convex optimization, convex-concave games and its applications. In *STOC*, 2020.

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

[KSH12] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. *Advances in neural information processing systems*, 25:1097–1105, 2012.

[LBBH98] 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.

[LL18] Yuanzhi Li and Yingyu Liang. Learning overparameterized neural networks via stochastic gradient descent on structured data. In *NeurIPS*, 2018.

[LS01] W.V. Li and Q.-M. Shao. Gaussian processes: Inequalities, small ball probabilities and applications. In *Stochastic Processes: Theory and Methods*, volume 19 of *Handbook of Statistics*, pages 533–597. Elsevier, 2001.

[LSS+20] Jason D Lee, Ruoqi Shen, Zhao Song, Mengdi Wang, and Zheng Yu. Generalized leverage score sampling for neural networks. In *NeurIPS*, 2020.

[LSZ19] Yin Tat Lee, Zhao Song, and Qiuyi Zhang. Solving empirical risk minimization in the current matrix multiplication time. In *Conference on Learning Theory (COLT)*, pages 2140–2157. PMLR, 2019.

[LXJ+20] Zichang Liu, Zhaozhuo Xu, Alan Ji, Jonathan Li, Beidi Chen, and Anshumali Shrivastava. Climbing the wol: Training for cheaper inference. *arXiv preprint arXiv:2007.01230*, 2020.

[Mat92a] Jiří Matoušek. Efficient partition trees. *Discrete & Computational Geometry*, 8(3):315–334, 1992.

[Mat92b] Jiri Matousek. Reporting points in halfspaces. *Computational Geometry*, 2(3):169–186, 1992.

[OS20] Samet Oymak and Mahdi Soltanolkotabi. Towards moderate overparameterization: global convergence guarantees for training shallow neural networks. In *arXiv preprint*. https://arxiv.org/pdf/1902.04674.pdf, 2020.

[Raz17] Ilya Razenshteyn. *High-dimensional similarity search and sketching: algorithms and hardness*. PhD thesis, Massachusetts Institute of Technology, 2017.

[Sch11] Jssai Schur. Bemerkungen zur theorie der beschränkten bilinearformen mit unendlich vielen veränderlichen. 1911.

[SHM+16] David Silver, Aja Huang, Chris J Maddison, Arthur Guez, Laurent Sifre, George Van Den Driessche, Julian Schrittwieser, Ioannis Antonoglou, Veda Panneershelvam, Marc Lanctot, et al. Mastering the game of go with deep neural networks and tree search. *nature*, 529(7587):484–489, 2016.

[SL14] Anshumali Shrivastava and Ping Li. Asymmetric lsh (alsh) for sublinear time maximum inner product search (mips). *Advances in Neural Information Processing Systems (NIPS)*, pages 2321–2329, 2014.

[SL15a] Anshumali Shrivastava and Ping Li. Asymmetric minwise hashing for indexing binary inner products and set containment. In *Proceedings of the 24th international conference on world wide web (WWW)*, pages 981–991, 2015.

[SL15b] Anshumali Shrivastava and Ping Li. Improved asymmetric locality sensitive hashing (alsh) for maximum inner product search (mips). In *Proceedings of the Thirty-First Conference on Uncertainty in Artificial Intelligence (UAI)*, pages 812–821, 2015.

[SLJ+15] 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*, pages 1–9, 2015.

[SSS+17] David Silver, Julian Schrittwieser, Karen Simonyan, Ioannis Antonoglou, Aja Huang, Arthur Guez, Thomas Hubert, Lucas Baker, Matthew Lai, Adrian Bolton, et al. Mastering the game of go without human knowledge. *nature*, 550(7676):354–359, 2017.

[SY19] Zhao Song and Xin Yang. Quadratic suffices for over-parametrization via matrix chernoff bound. *arXiv preprint arXiv:1906.03593*, 2019.

[TOG17] Csaba D Toth, Joseph O'Rourke, and Jacob E Goodman. *Handbook of discrete and computational geometry*. CRC press, 2017.

[Tro15] Joel A Tropp. An introduction to matrix concentration inequalities. *Foundations and Trends in Machine Learning*, 8(1-2):1–230, 2015.

[Vai89] Pravin M Vaidya. Speeding-up linear programming using fast matrix multiplication. In *FOCS*, 1989.

[ZPD+20] Yi Zhang, Orestis Plevrakis, Simon S Du, Xingguo Li, Zhao Song, and Sanjeev Arora. Over-parameterized adversarial training: An analysis overcoming the curse of dimensionality. In *NeurIPS*. arXiv preprint arXiv:2002.06668, 2020.

Checklist

1. For all authors...
   (a) Do the main claims made in the abstract and introduction accurately reflect the paper's contributions and scope?
   A: [Yes] .
   (b) Did you describe the limitations of your work?
   A: [Yes] See Section 7.
   (c) Did you discuss any potential negative societal impacts of your work?
   A: [N/A] . Our theoretical work does not have explicitly negative societal impacts.
   (d) Have you read the ethics review guidelines and ensured that your paper conforms to them?
   A: [Yes] , we conformed.
2. If you are including theoretical results...
   (a) Did you state the full set of assumptions of all theoretical results?
   A: [Yes] , we explictly stated the assumptions.
   (b) Did you include complete proofs of all theoretical results?
   A: [Yes] , we provided the complete proofs in supplementary materials.
3. If you ran experiments...
   (a) Did you include the code, data, and instructions needed to reproduce the main experimental results (either in the supplemental material or as a URL)?
   A: [N/A]
   (b) Did you specify all the training details (e.g., data splits, hyperparameters, how they were chosen)?
   A: [N/A]
   (c) Did you report error bars (e.g., with respect to the random seed after running experiments multiple times)?
   A: [N/A]
   (d) Did you include the total amount of compute and the type of resources used (e.g., type of GPUs, internal cluster, or cloud provider)?
   A: [N/A]
4. If you are using existing assets (e.g., code, data, models) or curating/releasing new assets...

(a) If your work uses existing assets, did you cite the creators?
A: [N/A]

(b) Did you mention the license of the assets?
A: [N/A]

(c) Did you include any new assets either in the supplemental material or as a URL?
A: [N/A]

(d) Did you discuss whether and how consent was obtained from people whose data you're using/curating?
A: [N/A]

(e) Did you discuss whether the data you are using/curating contains personally identifiable information or offensive content?
A: [N/A]

5. If you used crowdsourcing or conducted research with human subjects...

   (a) Did you include the full text of instructions given to participants and screenshots, if applicable?
   A: [N/A]

   (b) Did you describe any potential participant risks, with links to Institutional Review Board (IRB) approvals, if applicable?
   A: [N/A]

   (c) Did you include the estimated hourly wage paid to participants and the total amount spent on participant compensation?
   A: [N/A]