# Appendix (MONGOOSE: A Learnable LSH Framework for Efficient Neural Network Training)

## Table of Contents

## A SLOW CHANGE OBSERVATIONS

In this section, we analyze the relative distance between iterates of model parameters and connect it with LSH updates throughout training. We target at making full use of our slowly change characterization for MONGOOSE.

**Settings.** We report $\Delta W$ throughout training on several benchmark models. We follow the standard training procedures of fully-connected networks on Wiki325k (Partalas et al., 2015) and Transformer (Vaswani et al., 2017) on enwik8. We choose $W$ in the last linear layer for the fully-connected model and $W$ in the projection layer before attention for the transformer model to correspond experiments in section 4. In Appendix A, we provide a detailed investigation on experimental results from different datasets.

**Results.** We plot our results in Figure 3. The left-two and right-most plots show that during the initial steps of the training, $\Delta W$ is relatively high, but quickly drops and flattens out afterwards. The middle and right-most plots exhibit the hash code change of $W$ in the hamming distance along with the training. The pattern matches with $\Delta W$ but has a direct connection with the LSH update overhead. This is also consistent with LSH theory that the hash code collision probability of two vectors equals to the vectors' certain similarity, *e.g.*, angular. Note the above observations are made based on angular distance LSH. The above phenomenon suggests that if there exists an optimal scheduler to update the data structures adaptively based on the actual demand, the overhead by LSH updates can be largely reduced. Furthermore, this opens the door to make LSH learnable in order to improve query efficiency. Since the LSH update time is also closely related to the query time, the overall update overhead might still be reduced after considering learning costs, if the updates are well scheduled.

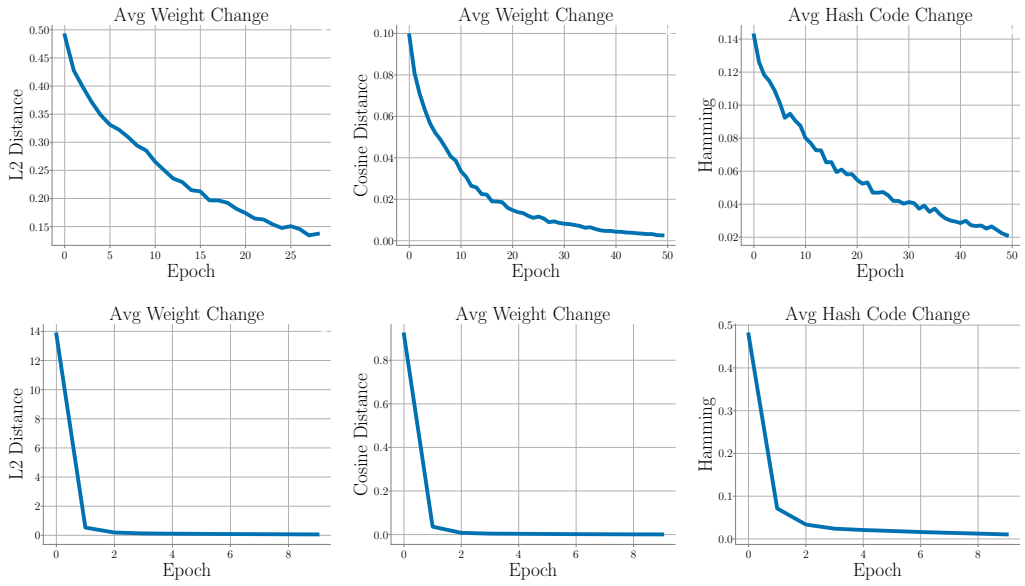


Figure 7: We show the average change of weight (left), cosine similarity (middle), weight's hash code (right). Top row is reformer and the bottom row is for Wiki-325k.

# B DYNAMIC MAINTENANCE DATA STRUCTURE

The idea of dynamic maintenance have been successfully applied to matrix inversion problem, which served as a central component in linear program solvers (Cohen et al., 2019; Jiang et al., 2020c), cutting plane method (Jiang et al., 2020b), and semi-definite program solvers (Jiang et al., 2020a). In this work, we adopt it for dynamic LSH maintenance problem. The rest of this section is organized as follows

- In Section B.1, we state our main results.
- In Section B.2, we give a high-level overview on the correctness and the running time of the algorithm.
- In Section B.3, we give an analyze on the movement of $w$ vector.
- In Section B.4 we give analyze on the movement of $v$ vector.
- In Section B.5, we discussed over the choice of potential function and analysis its property.
- In Section B.6, we compare the performance our algorithm with sequential updating strategy/batch updating strategy.

## B.1 MAIN RESULT

**Theorem B.1** (Locality sensitive hashing maintenance, Formal statement of Theorem 3.3)**.** *For any constant $c_1, c_2$ ($c_1 > c_2$) there is a dynamic data structure (Algorithm 1) that achieves $(c_1, c_2)$-accuracy. The data structure takes $\widetilde{O}(dn^{1+\rho})$ time to initialize and each call of* QUERY$(h)$ *takes time $\widetilde{O}(n^\rho d)$. By taking $a = \min\{\rho, \alpha\}$ and*

$$g_r = \begin{cases} n^{\rho - a}, & r \leq n^a; \\ t_r, & r. \end{cases}$$

*The amortized expected time per call of* UPDATE$(w)$ *is at most*

$$\widetilde{O}((C_1 + C_2) \cdot \|g\|_2). \tag{4}$$

## B.2 PROOF OF THEOREM B.1

We first give a rough explanation on the proof of Theorem 3.3. For intuition, we consider the case $C_1 = \Theta(1)$, $C_2 = \Theta(1)$, and $\epsilon_{\mathrm{mp}} = \Theta(1)$ in this explanation. The amortized time analysis is based on a potential function that measures the distance of the approximate vector $v$ and the target vector $w$. We will show that

- The cost to update the LSH data structure is proportional to the decrease of the potential.
- Each call to query increase the potential by a fixed amount.

Combining both together gives the amortized running time bound of our data structure.

Now, we explain the definition of the potential. Consider the $k$-th round of the algorithm. For all $i \in [n]$, we define $x_i^{(k)} = w_i^{(k)} - v_i^{(k)}$. Note that $\|x_i^{(k)}\|_2$ measures the distance between $w_i^{(k)}$ and $v_i^{(k)}$. Our algorithm fixes the indices with largest error $\|x_i^{(k)}\|_2$. To capture the fact that updating in a larger batch is more efficient, we define the potential as a weighted combination of the error where we put more weight to higher $x_i^{(k)}$. Formally, we sort the coordinates of $x^{(k)}$ such that $\|x_i^{(k)}\|_2 \geq \|x_{i+1}^{(k)}\|_2$ and define the potential by

$$\Psi_k = \sum_{i=1}^{n} g_i \cdot \psi(x_i^{(k)}),$$

where $g_i$ are positive decreasing numbers to be chosen and $\psi$ is a symmetric ($\psi(x) = \psi(-x)$) positive function that increases on both sides. For intuition, one can think $\psi(x)$ behaves roughly like $|x|$.

### B.2.1 PROOF OF CORRECTNESS

We prove the correctness of Theorem 3.3, we will defer some simple calculations into later sections.

**Definition of $x$ and $y$.** Consider the $k$-th round of the algorithm. For all $i \in [n]$, we define $x_i^{(k)} \in \mathbb{R}^d$, $x_i^{(k+1)} \in \mathbb{R}^d$ and $y_i^{(k)} \in \mathbb{R}^d$ as follows:

$$x_i^{(k)} = w_i^{(k)} - v_i^{(k)}, y_i^{(k)} = w_i^{(k+1)} - v_i^{(k)}, x_i^{(k+1)} = w_i^{(k+1)} - v_i^{(k+1)}.$$

Note that the difference between $x_i^{(k)}$ and $y_i^{(k)}$ is that $w$ is changing. The difference between $y_i^{(k)}$ and $x_i^{(k+1)}$ is that $v$ is changing.

**Assume sorting.** Assume the coordinates of vector $x^{(k)} \in \mathbb{R}^{n \times d}$ are sorted such that $\|x_i^{(k)}\|_2 \geq \|x_{i+1}^{(k)}\|_2$. Let $\tau$ and $\pi$ are permutations such that $\|x_{\tau(i)}^{(k+1)}\|_2 \geq \|x_{\tau(i+1)}^{(k+1)}\|_2$ and $\|y_{\pi(i)}^{(k)}\|_2 \geq \|y_{\pi(i+1)}^{(k)}\|_2$.

**Definition of Potential function.** Let $\psi : \mathbb{R} \to \mathbb{R}$ be defined by

$$\psi(x) = \begin{cases} \frac{\|x\|_2^2}{2\epsilon_{\mathrm{mds}}}, & \|x\|_2 \in [0, \epsilon_{\mathrm{mds}}] \\ \epsilon_{\mathrm{mds}} - \frac{(4\epsilon_{\mathrm{mds}}^2 - \|x\|_2^2)^2}{18\epsilon_{\mathrm{mds}}^3}, & \|x\|_2 \in (\epsilon_{\mathrm{mds}}, 2\epsilon_{\mathrm{mds}}] \\ \epsilon_{\mathrm{mds}}, & \|x\|_2 \in (2\epsilon_{\mathrm{mds}}, +\infty) \end{cases} \tag{5}$$

We define the potential at the $k$-th round by

$$\Psi_k = \sum_{i=1}^n g_i \cdot \psi(x_{\tau_k(i)}^{(k)}),$$

where $\tau_k(i)$ is the permutation such that $\|x_{\tau_k(i)}^{(k)}\|_2 \geq \|x_{\tau_k(i+1)}^{(k)}\|_2$.

**Bounding the potential.**

We can express $\Psi_{k+1} - \Psi_k$ as follows:

$$\begin{aligned} \Psi_{k+1} - \Psi_k &= \sum_{i=1}^n g_i \cdot \left( \psi(x_{\tau(i)}^{(k+1)}) - \psi(x_i^{(k)}) \right) \\ &= \sum_{i=1}^n g_i \cdot \underbrace{\left( \psi(y_{\pi(i)}^{(k)}) - \psi(x_i^{(k)}) \right)}_{w \text{ move}} - \sum_{i=1}^n g_i \cdot \underbrace{\left( \psi(y_{\pi(i)}^{(k)}) - \psi(x_{\tau(i)}^{(k+1)}) \right)}_{v \text{ move}}. \end{aligned} \tag{6}$$

Now, using Lemma B.5 and B.7, and the fact that $\Psi_0 = 0$ and $\Psi_T \geq 0$, with Eq. 6, we get

$$\begin{aligned} 0 \leq \Psi_T - \Psi_0 &= \sum_{k=0}^{T-1} (\Psi_{k+1} - \Psi_k) \\ &\leq \sum_{k=0}^{T-1} (O(C_1 + C_2/\epsilon_{\mathrm{mds}}) \cdot \|g\|_2 - \Omega(\epsilon_{\mathrm{mds}} r_k g_{r_k} / \log n)) \\ &= T \cdot O(C_1 + C_2/\epsilon_{\mathrm{mds}}) \cdot \|g\|_2 - \sum_{k=1}^T \Omega(\epsilon_{\mathrm{mds}} r_k g_{r_k} / \log n), \end{aligned}$$

where the third step follows by Lemma B.5 and Lemma B.7 and $r_k$ is the number of coordinates we update during that iteration.

Therefore, we get,

$$\sum_{k=1}^T r_k g_{r_k} = O\left( T \cdot (C_1/\epsilon_{\mathrm{mds}} + C_2/\epsilon_{\mathrm{mds}}^2) \cdot \log n \cdot \|g\|_2 \right).$$

### B.2.2 INITIALIZATION TIME, UPDATE TIME, QUERY TIME

To formalize the amortized runtime proof, we first analyze the initialization time (Lemma B.2), update time (Lemma B.3), and query time (Lemma B.4) of our maintenance data-structure.

**Lemma B.2** (Initialization time)**.** *The initialization time of data-structure* MAINTAIN *(Algorithm 1) is* $O(dn^{1+\rho})$.

**Lemma B.3** (Update time)**.** *The update time of data-structure* MAINTAIN *(Algorithm 1) is* $O(rg_r) + O(Sd + S\log n)$ *where* $r$ *is the number of indices we updated in* $v$*,* $S$ *is the number of weights changed. The later two terms are dominated by the updating time of neural network.*

*Proof.* We change $r$ ($r \geq n^a$) terms each time, and the running time for updating the LSH is $rg_r$. We need to update $v$ and calculate its $\ell_2$ norm every time, and the computational cost is $Sd$. We also need to maintain an order on the error $y_i^{(k)}$, using some standard data structure (like Fibonacci heap), this can be done in $S\log n$. Thus the total running time per update is $O(rg_r) + O(Sd + S\log n)$, the second term is bounded by the updating cost of neural network training, since calculate the gradient takes at least $O(Sd)$ and we usually have $d \gg \log n$. □

**Lemma B.4** (Query time)**.** *The query time of data-structure* MAINTAIN *(Algorithm 1) is* $O(n^\rho d + n^a d)$.

### B.3 BOUNDING $w$ MOVE

**Lemma B.5** ($w$ move)**.** *We have*

$$\sum_{i=1}^n g_i \cdot \mathbb{E}\left[\psi(y_{\pi(i)}^{(k)}) - \psi(x_i^{(k)})\right] \leq O(C_1 + C_2/\epsilon_{\mathrm{mds}}) \cdot \|g\|_2.$$

*Proof.* Observe that since the errors $\|x_i^{(k)}\|_2$ are sorted in descending order, and $\psi(x)$ is symmetric and non-decreasing function, thus $\psi(x_i^{(k)})$ is also in decreasing order. In addition, note that $g$ is decreasing, we have

$$\sum_{i=1}^n g_i \psi(x_{\pi(i)}^{(k)}) \leq \sum_{i=1}^n g_i \psi(x_i^{(k)}). \tag{7}$$

Hence we have

$$\begin{aligned}
\mathbb{E}\left[\sum_{i=1}^n g_i \cdot \left(\psi(y_{\pi(i)}^{(k)}) - \psi(x_i^{(k)})\right)\right] &\leq \mathbb{E}\left[\sum_{i=1}^n g_i \cdot \left(\psi(y_{\pi(i)}^{(k)}) - \psi(x_{\pi(i)}^{(k)})\right)\right] && \text{by Eq. 7} \\
&= \sum_{i=1}^n g_i \cdot \mathbb{E}[\psi(y_{\pi(i)}^{(k)}) - \psi(x_{\pi(i)}^{(k)})] \\
&= O(C_1 + C_2/\epsilon_{\mathrm{mds}}) \cdot \|g\|_2). && \text{by Lemma B.6}
\end{aligned}$$

Thus, we complete the proof of $w$ move Lemma. □

It remains to prove the following Lemma,

**Lemma B.6.**

$$\sum_{i=1}^n g_i \cdot \mathbb{E}[\psi(y_{\pi(i)}^{(k)}) - \psi(x_{\pi(i)}^{(k)})] = O(C_1 + C_2/\epsilon_{\mathrm{mds}}) \cdot \|g\|_2.$$

*Proof.* We separate the term into two:

$$\begin{aligned}
\sum_{i=1}^n g_i \cdot \mathbb{E}[\psi(y_{\pi(i)}^{(k)}) - \psi(x_{\pi(i)}^{(k)})] = &\sum_{i=1}^n g_{\pi^{-1}(i)} \cdot \mathbb{E}[\psi(y_i^{(k)}) - \psi(\mathbb{E}[y_i^{(k)}])] \\
&+ \sum_{i=1}^n g_{\pi^{-1}(i)} \cdot (\psi(\mathbb{E}[y_i^{(k)}]) - \psi(x_i^{(k)})).
\end{aligned} \tag{8}$$

For the first term, we have

$$\begin{aligned}
&\psi(y_i^{(k)}) - \mathbb{E}[\psi(y_i^{(k)})] \\
&= \langle \psi'(\mathbb{E}[y_i^{(k)}]), y_i^{(k)} - \mathbb{E}[y_i^{(k)}] \rangle + \frac{1}{2}(y_i^{(k)} - \mathbb{E}[y_i^{(k)}])^\top \psi''(\zeta)(y_i^{(k)} - \mathbb{E}[y_i^{(k)}]) \\
&\leq \langle \psi'(\mathbb{E}[y_i^{(k)}]), y_i^{(k)} - \mathbb{E}[y_i^{(k)}] \rangle + \frac{1}{2} L_2 \|y_i^{(k)} - \mathbb{E}[y_i^{(k)}]\|_2^2 \\
&= \langle \psi'(\mathbb{E}[y_i^{(k)}]), w_i^{(k+1)} - \mathbb{E}[w_i^{(k+1)}] \rangle + \frac{1}{2} L_2 \|w_i^{(k+1)} - \mathbb{E}[w_i^{(k+1)}]\|_2^2 \qquad (9)
\end{aligned}$$

where the first step follows from the Mean value theorem, the second step follows from the definition of $L_2$ (see Part 4 of potential lemma B.8), the last step follows from the definition of $y_i^{(k)}$.

Next, we denote $\gamma_i = \mathrm{Var}[w_i^{(k+1)}]$. Summing over $i$ and taking conditional expectation given $w^{(k)}$ on both sides, we get

$$\begin{aligned}
\sum_{i=1}^n g_{\pi^{-1}(i)} \cdot \mathbb{E}[\psi(y_i^{(k)}) - \psi(\mathbb{E}[y_i^{(k)}])] &\leq \sum_{i=1}^n g_{\pi^{-1}(i)} \cdot \mathbb{E}[\langle \psi'(\mathbb{E}[y_i^{(k)}]), w_i^{(k+1)} - \mathbb{E}[w_i^{(k+1)}] \rangle] \\
&\quad + \sum_{i=1}^n g_{\pi^{-1}(i)} \cdot \frac{1}{2} L_2 \mathbb{E}[\|w_i^{(k+1)} - \mathbb{E}[w_i^{(k+1)}]\|_2^2] \\
&= 0 + \frac{1}{2} L_2 \sum_{i=1}^n g_{\pi^{-1}(i)} \cdot \gamma_i \\
&\leq \frac{1}{2} L_2 \|g\|_2 (\sum_{i=1}^n \gamma_i^2)^{\frac{1}{2}} \\
&\leq \frac{1}{2} \cdot O(1/\epsilon_{\mathrm{mds}}) \cdot \|g\|_2 \cdot C_2 \\
&= O(C_2/\epsilon_{\mathrm{mds}}) \|g\|_2. \qquad (10)
\end{aligned}$$

The first step follows from Eq. 9, the second step follows from the linearity of expectation and the definition of $\gamma_i$, the third step follows from Cauchy-Shwarz inequality, the fourth step follows from $L_2 = O(1/\epsilon_{\mathrm{mds}})$ (see Lemma B.8) and Eq. 1.

For the second term, conditioning on $w_i^{(k)}$, we have

$$\psi(\mathbb{E}[y_i^{(k)}]) - \psi(x_i^{(k)}) \leq L_1 \cdot \|\mathbb{E}[y_i^{(k)}] - x_i^k\|_2 = L_1 \cdot \|\mathbb{E}[w_i^{(k+1)}] - w_i^k\|_2 \stackrel{\mathrm{def}}{=} L_1 \cdot \beta_i. \qquad (11)$$

The first inequality follows from the $L_1$-Lipschitz continuity of $\Psi$ (see part 4 of Lemma B.8). The second step follows from the definition of $y_i^{(k)}$ and $x_i^{(k)}$.

Summing over $i$, we get

$$\sum_{i=1}^n g_{\pi^{-1}(i)} \cdot (\psi(\mathbb{E}[y_i^{(k)}]) - \psi(x_i^{(k)})) \leq \sum_{i=1}^n g_{\pi^{-1}(i)} \cdot L_1 \beta_i \leq L_1 \cdot \|g\|_2 \cdot (\sum_{i=1}^n \beta_i^2)^{\frac{1}{2}} \leq O(C_1) \cdot \|g\|_2. \qquad (12)$$

The first step follows from Eq. 11, the second step follows from Cauchy Shwarz inequality, the last step follows from $L_1 \leq 2$ (see part 4 of Lemma B.8) and Eq. 1.

Combining Eq. 81012, we have

$$\sum_{i=1}^n g_i \cdot \mathbb{E}[\psi(y_{\pi(i)}^{(k)}) - \psi(x_{\pi(i)}^{(k)})] \leq O(C_1 + C_2/\epsilon_{\mathrm{mds}}) \|g\|_2.$$

Thus completing the proof. □

### B.4 BOUNDING $v$ MOVE

The goal of this section is to prove Lemma B.7.

**Lemma B.7** ($v$ move)**.** *We have,*

$$\sum_{i=1}^{n} g_i \cdot \left(\psi(y^{(k)}_{\pi(i)}) - \psi(x^{(k+1)}_{\tau(i)})\right) \geq \Omega(\epsilon_{\mathrm{mds}} r_k g_{r_k} / \log n).$$

*Proof.* We split the proof into two cases.

We first understand some simple facts which are useful in the later proof. Note that from definition of $x_i^{(k+1)}$, we know that $x^{(k+1)}$ has $r_k$ coordinates are $\vec{0}$. Basically, $\|y^{(k)} - x^{(k+1)}\|_0 = r_k$. The difference between those vectors is, for the largest $r_k$ coordinates in $y^{(k)}$, we erase them in $x^{(k+1)}$. Then for each $i \in [n - r_k]$, $x^{(k+1)}_{\tau(i)} = y^{(k)}_{\pi(i+r_k)}$. For convenience, we define $y^{(k)}_{\pi(n+i)} = \vec{0}, \forall i \in [r_k]$ .

**Case 1.** We exit the while loop when $1.5r_k \geq n$.

Let $u^*$ denote the largest $u$ satisfying $\|y^{(k)}_{\pi(u)}\|_2 \geq \epsilon_{\mathrm{mds}}/2$. If $u^* \geq r_k$, then we have that $\|y^{(k)}_{\pi(r_k)}\|_2 \geq \epsilon_{\mathrm{mds}}/2 \geq \epsilon_{\mathrm{mds}}/100$. Otherwise, the condition of the loop shows that

$$\begin{aligned}
\|y^{(k)}_{\pi(r_k)}\|_2 &\geq (1 - 1/\log n)^{\log_{1.5} r_k - \log_{1.5} u^*} \|y^{(k)}_{\pi(u^*)}\|_2 \\
&\geq (1 - 1/\log n)^{\log_{1.5} n} \epsilon_{\mathrm{mds}}/2 \\
&\geq \epsilon_{\mathrm{mds}}/100.
\end{aligned}$$

where we used that $n \geq 4$.

According to the definition of $x^{(k+1)}_{\tau(i)}$, we have

$$\begin{aligned}
\sum_{i=1}^{n} g_i(\psi(y^{(k)}_{\pi(i)}) - \psi(x^{(k+1)}_{\tau(i)})) &= \sum_{i=1}^{n} g_i(\psi(y^{(k)}_{\pi(i)}) - \psi(y^{(k)}_{\pi(i+r_k)})) \\
&\geq \sum_{i=n/3+1}^{n} g_i(\psi(y^{(k)}_{\pi(i)}) - \psi(y^{(k)}_{\pi(i+r_k)})) \\
&\geq \sum_{i=n/3+1}^{n} g_i \psi(y^{(k)}_{\pi(i)}) \\
&\geq \sum_{i=n/3+1}^{2n/3} g_i f((\epsilon_{\mathrm{mds}}/100)^2) \geq \Omega(r_k g_{r_k} \epsilon_{\mathrm{mds}}),
\end{aligned}$$

where the first step follows from $x^{(k+1)}_{\tau(i)} = y^{(k)}_{\pi(i+r_k)}$, the second step follows from $\psi(x)$ is non-decreasing (see part 2 of Lemma B.8) and $\|y^{(k)}_{\pi(i)}\|_2$ is non-increasing, the third step follows from $1.5r_k > n$ and hence $\psi(y^{(k)}_{\pi(i+r_k)}) = 0$ for $i \geq n/3+1$, the fourth step follows from $\psi(x) = f(\|x\|_2^2)$ is non-decreasing and $\|y^{(k)}_{\pi(i)}\|_2 \geq \|y^{(k)}_{\pi(r_k)}\|_2 \geq \epsilon_{\mathrm{mds}}/100$ for all $i < 2n/3$, and the last step follows by $g$ is non-increasing and the part 3 of Lemma B.8.

**Case 2.** We exit the while loop when $1.5r_k < n$ and $\|y^{(k)}_{\pi(1.5r_k)}\|_2 < (1 - 1/\log n)\|y^{(k)}_{\pi(r_k)}\|_2$.

By the same argument as Case 1, we have that $\|y^{(k)}_{\pi(r_k)}\|_2 \geq \epsilon_{\mathrm{mds}}/100$. Part 3 of Lemma B.8 together with the fact

$$\|y^{(k)}_{\pi(1.5r_k)}\|_2 < \min(\epsilon_{\mathrm{mds}}/2, \|y^{(k)}_{\pi(r_k)}\|_2 \cdot (1 - 1/\log n)),$$

indicates that

$$\psi(y^{(k)}_{\pi(1.5r_k)}) - \psi(y^{(k)}_{\pi(r_k)}) = \Omega(\epsilon_{\mathrm{mds}}/\log n). \tag{13}$$

Putting things together, we have

$$
\begin{aligned}
& \sum_{i=1}^{n} g_i \cdot (\psi(y_{\pi(i)}^{(k)}) - \psi(x_{\tau(i)}^{(k+1)})) \\
= & \sum_{i=1}^{n} g_i \cdot (\psi(y_{\pi(i)}^{(k)}) - \psi(y_{\pi(i+r_k)}^{(k)})) && \text{by } x_{\tau(i)}^{(k+1)} = y_{\pi(i+r_k)}^{(k)} \\
\geq & \sum_{i=r_k/2}^{r_k} g_i \cdot (\psi(y_{\pi(i)}^{(k)}) - \psi(y_{\pi(i+r_k)}^{(k)})) && \text{by } \psi(y_{\pi(i)}^{(k)}) - \psi(y_{\pi(i+r_k)}^{(k)}) \geq 0 \\
\geq & \sum_{i=r_k/2}^{r_k} g_i \cdot (\psi(y_{\pi(r_k)}^{(k)}) - \psi(y_{\pi(1.5r_k)}^{(k)})) \\
\geq & \sum_{i=r_k/2}^{r_k} g_i \cdot \Omega(\frac{\epsilon_{\mathrm{mds}}}{\log n}) && \text{by 13} \\
\geq & \sum_{i=r_k/2}^{r_k} g_{r_k} \cdot \Omega(\frac{\epsilon_{\mathrm{mds}}}{\log n}) && \text{by } g_i \text{ is decreasing} \\
= & \, \Omega\left(\epsilon_{\mathrm{mds}} r_k g_{r_k} / \log n\right),
\end{aligned}
$$

where the third step follows by $\|y_{\pi(i)}^{(k)}\|_2$ is non-increasing and $\psi$ is non-decreasing (see part 2 of Lemma B.8). □

### B.5 POTENTIAL FUNCTION $\psi$

Here we proved a vector version ($\psi(x) = f(\|x\|_2^2)$):

**Lemma B.8** (Properties of potential function $\psi$)**.** *Let function $\psi : \mathbb{R}^d \to \mathbb{R}$ (defined in Eq. 5). Then function $\psi$ satisfies the following properties:*
*1. Symmetric ($\psi(-x) = \psi(x)$) and $\psi(0) = 0$;*
*2. If $\|x\|_2 \geq \|y\|_2$, then $\psi(x) \geq \psi(y)$;*
*3. $|f'(x)| = \Omega(1/\epsilon_{\mathrm{mds}}), \forall |x| \in [(0.01\epsilon_{\mathrm{mds}})^2, \epsilon_{\mathrm{mds}}^2]$;*
*4. $L_1 \stackrel{\mathrm{def}}{=} \max_x \frac{D_x \psi[h]}{\|h\|_2} = 2$ and $L_2 \stackrel{\mathrm{def}}{=} \max_x \frac{D_x^2 \psi[h,h]}{\|h\|_2^2} = 10/\epsilon_{\mathrm{mds}}$;*
*5. $\psi(x)$ is a constant for $\|x\|_2 \geq 2\epsilon_{\mathrm{mds}}$*

*Proof.* Recall $f : \mathbb{R}_+ \to \mathbb{R}$ is defined as

$$
f(x) := \begin{cases} \frac{x^2}{2\epsilon_{\mathrm{mds}}^3}, & x \in [0, \epsilon_{\mathrm{mds}}^2]; \\ \epsilon_{\mathrm{mds}} - \frac{(4\epsilon_{\mathrm{mds}}^2 - x)^2}{18\epsilon_{\mathrm{mds}}^3}, & x \in (\epsilon_{\mathrm{mds}}^2, 4\epsilon_{\mathrm{mds}}^2]; \\ \epsilon_{\mathrm{mds}}, & x \in (4\epsilon_{\mathrm{mds}}^2, +\infty). \end{cases}
$$

We can see that

$$
f(x)' = \begin{cases} \frac{x}{\epsilon_{\mathrm{mds}}^3}, & x \in [0, \epsilon_{\mathrm{mds}}^2]; \\ \frac{4\epsilon_{\mathrm{mds}}^2 - x}{9\epsilon_{\mathrm{mds}}^3}, & x \in (\epsilon_{\mathrm{mds}}^2, 4\epsilon_{\mathrm{mds}}^2]; \\ 0, & x \in (4\epsilon_{\mathrm{mds}}^2, +\infty). \end{cases}
$$

and

$$
f(x)'' = \begin{cases} \frac{1}{\epsilon_{\mathrm{mds}}^3}, & x \in [0, \epsilon_{\mathrm{mds}}^2]; \\ -\frac{1}{9\epsilon_{\mathrm{mds}}^3}, & x \in (\epsilon_{\mathrm{mds}}^2, 4\epsilon_{\mathrm{mds}}^2]; \\ 0, & |x| \in (4\epsilon_{\mathrm{mds}}^2, +\infty) \end{cases}
$$

It implies that $\max_x |f(x)'| \leq \frac{1}{\epsilon_{\mathrm{mds}}}$ and $\max_x |f(x)''| \leq \frac{1}{\epsilon_{\mathrm{mds}}^3}$. Let $\psi(x) = f(\|x\|_2^2)$.

**Proof of Part 1, 2 and 5.** These proofs are pretty standard from definition of $\psi$.

**Proof of Part 3.** This is trivially following from definition of scalar function $f$.

**Proof of Part 4.** By chain rule, we have

$$D_x \psi[h] = 2f'(\|x\|_2^2) \cdot \langle x, h \rangle$$
$$D_x^2 \psi[h,h] = 2f''(\|x\|_2^2) \cdot (\langle x, h \rangle)^2 + 2f'(\|x\|_2^2) \cdot \langle h, h \rangle$$

We can upper bound

$$|D_x \psi[h]| \leq 2|f'(\|x\|_2^2)| \cdot |\langle x, h \rangle| \leq 2|f'(\|x\|_2^2)| \cdot \|x\|_2 \cdot \|h\|_2.$$

Thus, we have

$$|f'(\|x\|_2^2)| \cdot \|x\|_2 = \begin{cases} \|x\|_2^3/\epsilon_{\mathrm{mds}}^3 \leq 1, & \|x\|_2 \in [0, \epsilon_{\mathrm{mds}}]; \\ (4\epsilon_{\mathrm{mds}}^2 - \|x\|_2^2)\|x\|_2/(9\epsilon_{\mathrm{mds}}^3) \leq 2/3, & \|x\|_2 \in (\epsilon_{\mathrm{mds}}, 2\epsilon_{\mathrm{mds}}); \\ 0, & \|x\|_2 \in (2\epsilon_{\mathrm{mds}}, +\infty). \end{cases}$$

It implies that $|D_x \psi[h]| \leq 2\|h\|_2, \forall x$.

By case analysis, we have

$$|f''(\|x\|_2^2)| \cdot \|x\|_2^2 \leq \begin{cases} \frac{1}{\epsilon_{\mathrm{mds}}^3}\|x\|_2^2 \leq 4/\epsilon_{\mathrm{mds}}, & \|x\|_2^2 \in [0, 4\epsilon_{\mathrm{mds}}^2]; \\ 0, & \|x\|_2^2 \in (4\epsilon_{\mathrm{mds}}, +\infty). \end{cases}$$

We can also upper bound

$$\begin{aligned} |D_x^2 \psi[h,h]| &\leq 2|f''(\|x\|_2^2)| \cdot \langle x, h \rangle^2 + 2|f'(\|x\|_2^2)| \cdot \|h\|_2^2 \\ &\leq 2|f''(\|x\|_2^2)| \cdot (\|x\|_2 \cdot \|h\|_2)^2 + 2|f'(\|x\|_2^2)| \cdot \|h\|_2^2 \\ &\leq 2 \cdot \frac{4}{\epsilon_{\mathrm{mds}}}\|h\|_2^2 + 2 \cdot \frac{1}{\epsilon_{\mathrm{mds}}}\|h\|_2^2 \\ &= \frac{10}{\epsilon_{\mathrm{mds}}}\|h\|_2^2. \end{aligned}$$

□

### B.6 EXAMPLES

**Comparing with sequential updating algorithm** We present another version of our main result, which is particularly useful when we compare with the sequential updating algorithm.

**Theorem B.9** (Locality sensitive hashing maintenance, worst case bound)**.** *For any constant $c_1, c_2$ ($c_1 > c_2$) there is a dynamic data structure (Algorithm 1) that achieves $(c_1, c_2)$-accuracy. The data structure takes $\widetilde{O}(dn^{1+\rho})$ time to initialize and each call of* QUERY$(h)$ *takes time $\widetilde{O}(n^\rho d)$. By taking $a = \min\{\rho, \alpha\}$ and $g_r = t_r$, after $T$ iterations, the amortized expected time per call of* UPDATE$(w)$ *is at most*

$$\widetilde{O}\left(\frac{n^\rho}{T}\sum_{k=0}^{T-1}\sum_{i=1}^{n}\left(\left|\mathbb{E}[w_i^{(k+1)}] - w_i^{(k)}\right| + \mathrm{Var}[w_i^{k+1}]\right)\right) \tag{14}$$

The proof is almost indentical to Theorem B.1, we only need to replace Lemma B.5 by the following one.

**Lemma B.10** ($w$ move)**.** *We have*

$$\begin{aligned} \sum_{i=1}^{n} g_i \cdot \mathbb{E}\left[\psi(y_{\pi(i)}^{(k)}) - \psi(x_i^{(k)})\right] &\leq O\left(g_1 \sum_{i=1}^{n}\left|\mathbb{E}[w_i^{(k+1)}] - w_i^{(k)}\right| + \mathrm{Var}[w_i^{k+1}]\right) \\ &= O\left(n^\rho \sum_{i=1}^{n}\left|\mathbb{E}[w_i^{(k+1)}] - w_i^{(k)}\right| + \mathrm{Var}[w_i^{k+1}]\right) \end{aligned}$$

*We omit the dependence on $\epsilon_{\mathrm{mds}}$ since we assume it to be constant in the scheduler.*

*Proof.* Again, the proof is almost identical to Lemma B.5, the only difference is that we use $\ell_\infty/\ell_1$ form of Cauchy-Shwarz instead of the $\ell_2/\ell_2$ form in the previous proof. We only point out the difference here.

We replace Eq. 10 with

$$\begin{aligned}\sum_{i=1}^n g_{\pi^{-1}(i)} \cdot \mathbb{E}[\psi(y_i^{(k)}) - \psi(\mathbb{E}[y_i^{(k)}])] &= 0 + \frac{1}{2} L_2 \sum_{i=1}^n g_{\pi^{-1}(i)} \cdot \gamma_i \\ &\leq \frac{1}{2} L_2 g_1 \cdot \sum_{i=1}^n \gamma_i \\ &\approx O\left( g_1 \sum_{i=1}^n \mathrm{Var}[w_i^{k+1}] \right).\end{aligned}$$

The first step follows from Eq. 10, the second step follows from $\{g_i\}_{i=1,\cdots,n}$ are decreasing and the last step follows from the definition of $\gamma_i$.

Similarly, we replace Eq. 12 with

$$\sum_{i=1}^n g_{\pi^{-1}(i)} \cdot (\psi(\mathbb{E}[y_i^{(k)}]) - \psi(x_i^{(k)})) \leq \sum_{i=1}^n g_{\pi^{-1}(i)} \cdot L_1 \beta_i \lesssim O\left( g_1 \sum_{i=1}^n \left| \mathbb{E}[w_i^{(k+1)}] - w_i^{(k)} \right| \right)$$

The first step follows from Eq. 12, the second step follows from the definition of $\beta_i$, $L_1 \leq 2$ and $\{g_i\}_{i=1,\cdots,n}$ are decreasing.

The rest of the proof are the same and we omit it here. □

The sequential updating time for LSH is $n^\rho$. The total number of updating is (roughly) at least

$$O\left( \sum_{k=1}^T \sum_{i=1}^n \left( \left| \mathbb{E}[w_i^{(k+1)}] - w_i^{(k)} \right| + \mathrm{Var}[w_i^{k+1}] \right) \right)$$

It then easy to see that our scheduler always perform better than naive sequential update.

**Comparing with naive batch updating algorithm** Below we give a concrete example to show the effectiveness of our algorithm. We take $\alpha = \rho$ for simplicity. Remember in Assumption 3.2, we already assume for any $r \leq n^a = n^\rho$, $T_r = n^\rho$, i.e., the average LSH updating time $t_r = n^\rho / r$ decays linearly in $r$ when $r \leq n^\rho$. It remains to specify the rest $t_r$ ($r \geq n^\rho$).

**Example B.11.** *Assuming the average running time decays faster than $1/\sqrt{r}$, i.e. $t_r = n^{-(1-\beta)\rho} r^{-\beta}$ for some $\beta \in [\frac{1}{2}, 1)$ when $r \geq n^\rho$. We then have*

$$\|g\|_2^2 = \sum_{r=1}^n g_r^2 \approx n^{-\rho/2}.$$

*The running time of our scheduler is then at most*

$$(C_1 + C_2)\|g\|_2 \approx (C_1 + C_2) n^{\rho/2}.$$

*As long as the weight changes slowly, say $C_1 \approx C_2 \leq n^{\rho/2}$, we know the amortized updating time of our scheduler is strictly better than the naive approach that updates LSH every time, which has running time $n^\rho$ each iteration.*

# C LEARNABLE LSH

## C.1 LEARNING HASH FUNCTIONS

Denote a set of neurons in a particular layer as $\mathcal{C} = \{v_r \mid 0 \leq r < m\}$, where each neuron $v_r$ has weight and bias. Given an input embedding $q_i$ from the previous hidden layer, the output of a forward pass through neuron $r$ is $\sigma(\langle q_i, v_r \rangle)$, where $\sigma$ is some activation functions. For each input embedding $q_i$, both SLIDE and REFORMER select a set of neurons, denoted as $\mathcal{S}_i$, from the LSH hash tables for the forward pass. We collect the training samples from $q_i$ and its $\mathcal{S}_i$ to improve the performance of LSH. Formally, The pairwise training samples $(q, v)$ are collected according to the following criterion.

- positive pair $\mathcal{P}_+ = (q, v)$ if $v \in S$ and $\langle q, v \rangle > t_+$
- negative pair $\mathcal{P}_- = (q, v)$ if $v \in S$ and $\langle q, v \rangle < t_-$

And the loss is defined as:

$$\mathcal{L}(\mathcal{H}, \mathcal{P}_+, \mathcal{P}_-) = \max(0, \sum_{(q,v)\in\mathcal{P}_+} -\cos(\mathcal{H}(q), \mathcal{H}(v)) + \sum_{(q,v)\in\mathcal{P}_-} \cos(\mathcal{H}(q), \mathcal{H}(v)) + \alpha). \quad (15)$$

Here $\mathcal{H}$ is the hash functions of all $L$ tables. $\mathcal{H}(x)$ generates a $K \cdot L$ vector containing the projection of $x$ in each hash table. $\cos(\mathcal{H}(x), \mathcal{H}(y))$ represents the cosine similarity of two projected vectors. The major contribution of this learning approach is that it first optimize the hash function and the hash table index together. Our method targets at learning a useful indexing for retrieval efficiency while other learn to hash methods Wang et al. (2017) aim for a binary sketching that have higher precision.

### C.2 OBSERVATIONS ON ATTENTION DISTRIBUTION

In this section, we present the visualization with analysis on the distribution of the minimal quantifies of neurons that sum up to have 0.9 softmax values in attention. On Figure 8, we present the distribution of each head of attention in each layer from a transformer model trained on Enwiki8 dataset. We classify the patternsRamsauer et al. (2020) of the distribution into 3 categories by their median values. With this observation, we are able to determine the layers that LSH or learnable LSH can apply in MONGOOSE framework.


Figure 8: The distribution of the minimal quantifies of neurons that sum up to have 0.9 softmax values in attention in each head of attention in each layer of transformer for Enwiki8

# D EXPERIMENTS DETAILS

## D.1 DATA STATISTICS

Table 3: Statistics for our benchmark dataset

| Dataset | Wiki10-31k | Delicious-200K | Wiki-325K |
|---|---|---|---|
| Output Dimension | 30938 | 205443 | 325056 |
| Input Dimension | 101938 | 782585 | 1617899 |
| Training Samples | 14146 | 6616 | 1778351 |
| Testing Samples | 196606 | 100095 | 587084 |

We present statistics on the 3 datasets we test on from the Extreme Classification Repository (Bhatia et al., 2016). While the number of datapoints in each dataset is not large (on the order of 200K at most), the key feature is the sheer size of the input and output dimensions. In particular, each dataset has over 10,000 output classes, which, using a conventional nueral network, requires a matrix multiplication involving over 10,000 neurons at the final layer.

## D.2 ADDITIONAL RESULTS ON THE EXTREME-CLASSIFICATION TASK

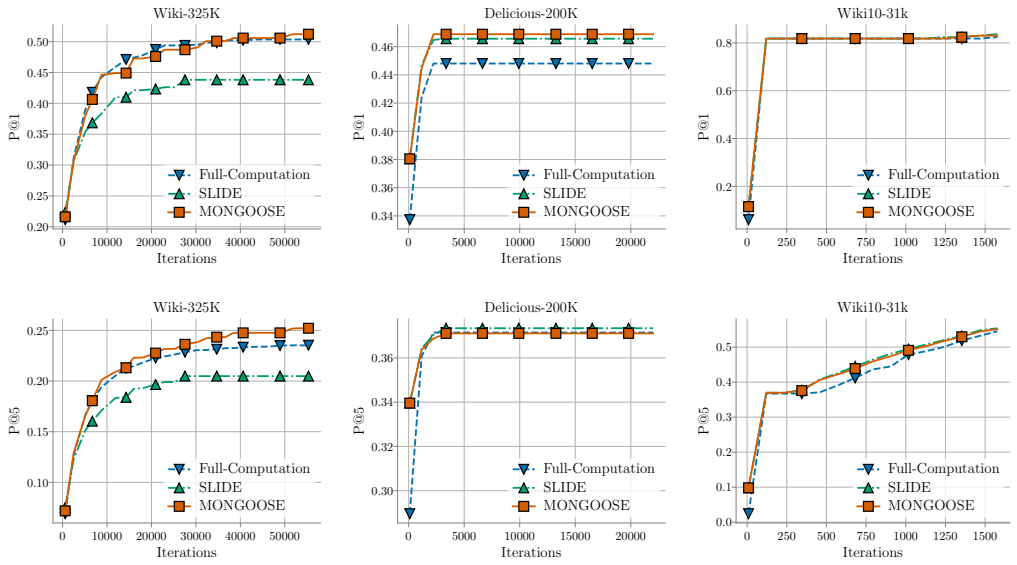


Figure 9: Comparison of MONGOOSE against SLIDE and FULL during the training. The two metrics (P@1 in the top row and P@5 in the bottom row) are the same as in Bhatia et al. (2016).

We present the results of additional experiments comparing the classification performance of MONGOOSE against SLIDE and FULL in Figure 9. The two metrics, P@1 and P@5, are presented in Bhatia et al. (2016) as follows:

$$\text{P@}k = \frac{1}{k} \sum_{l \in \text{rank}_k(\widehat{\mathbf{y}})} \mathbf{y}_l$$

where $\text{rank}_k(\cdot)$ extracts the top $k$ indices from a vector, $\widehat{\mathbf{y}}$ refers to the predicted labels, and $\mathbf{y}$ refers to the ground truth label.

### D.3 ADDITIONAL RESULTS ON THE LANGUAGE MODELING TASK

We present additional experiments comparing the training loss of MONGOOSE and Reformer on the synthetic copy task in Section 4. In Figure 10 we present additional experiments comparing the training loss of MONGOOSE and Reformer on the synthetic copy task in Section 4.1.2. The titles of each graph denote the hyperparameters of the Transformer model being tested: h1_s2048_t16 refers to a model with 1 round of hashing, a sequence length of 2048, and a token size of 16. Overall we can see that MONGOOSE makes a marked improvement in loss over Reformer in every case.

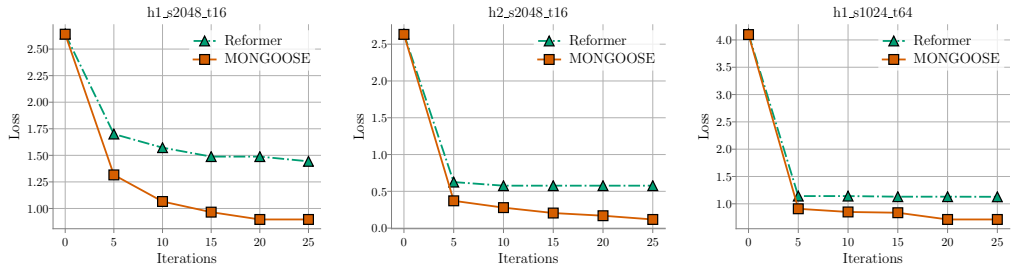


Figure 10: Comparison of MONGOOSE against Reformer during training on the synthetic copy task.

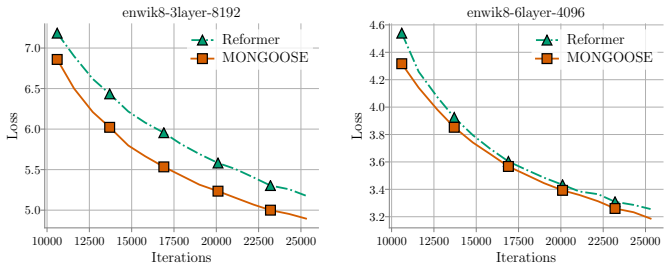


Figure 11: Comparison of MONGOOSE against Reformer during training on enwik8.

In Figure 11, we present further results comparing MONGOOSE to Reformer on the enwik8 character-level language modelling task. In particular, we train two sizes of Transformer (3 and 6 layers) with a maximum sequence length of 8192 and 4096 respectivelyh. In both settings mongoose achieves lower loss than Reformer.

Note that the goal of this experiments is to prove the superiority of learnable LSH over classical LSH in NN training (or MONGOOSE vs. naive LSH-NN framework) rather than improve the state-of-the-art perplexity.

# E RELATED WORK

## E.1 DATA STRUCTURES FOR DYNAMIC SIMILARITY SEARCH

Similarity search is a well-studied problem with wide applications in recommendation system (Xue et al., 2017; Severyn & Moschitti, 2015; Hall & Attenberg, 2015), question answering (Boytsov et al., 2016; Seo et al., 2019; Ahmad et al., 2019; Chang et al., 2020), multi-label classification (Covington et al., 2016; Jain et al., 2016; Tagami, 2017) and natural language processing (Bengio et al., 2003; Gao et al., 2014; Lee et al., 2015). There are two major categories of similarity measures: (1) metric similarity (cosine similarity, euclidean distance), (2) non-metric similarity (inner product, KL divergence, neural network). The brute-force approach to solving similarity search is computationally expensive; in response, researchers have developed novel indexing structures to accelerate the search process, with trade-offs on search accuracy. Based on these indexing structures, similarity search algorithms can be broadly categorized as (1) hashing (Shrivastava & Li, 2014a;b), (2) quantization (Guo et al., 2016; Jegou et al., 2011), (3) tree-based (Ram & Gray, 2012), or (4) graph-based methods (Malkov et al., 2012; 2014; Malkov & Yashunin, 2018).

Most similarity search scenarios studied by these papers are static (search data does not change). In the experiments section, the developed similarity search methods are compared on a fixed dataset such as Sift (Jegou et al., 2011) or Glove (Pennington et al., 2014). However, in current similarity search applications such as the work of Fan et al. (2019), the search distribution (e.g. product vectors) changes over time due to the activation of new products and the expiration of old products. Therefore, some similarity search methods in metric space have been modified for online settings, such as hashing (Coleman et al., 2019) and quantization (Xu et al., 2018) methods; these methods sacrifice search speed for data-adaptiveness.

The training phase of Deep Learning models provides a natural setting for Dynamic LSH. During training, the weight matrices are slowly modified via gradients derived from objective functions. If we consider the weights as the search data and the training sample as queries, we can view DNN training as a Dynamic Similarity Search problem. Recent works take advantage of this view of DNN training by introducing LSH data structures to the NN training process. Chen et al. (2020) propose an algorithm (SLIDE) that retrieves neurons with maximum inner product in each step via an LSH based data structure. In this way, the backward pass of NN training is concentrated on the neurons with estimated large gradients. Their CPU implementation is able to outperform a traditional GPU implementation. Similar hashing based algorithms have also been used in Transformer models: Kitaev et al. (2020a) (Reformer) propose an LSH structure to reduce the memory bottleneck of self-attention modules especially over long sequences in Transformer.

Since DNN weights are the search data, the distribution of the weights among different hash buckets changes throughout the training. This necessitates constantly updating the LSH data structure: failing to update the LSH data structure as the search data changes degrades its nearest-neighbor search performance, which in turn worsens the quality of the DNN approximation. In our experiments, we found that failing to update the LSH data structure in SLIDE caused a 28% decrease in top-1 accuracy.

## E.2 DATA DEPENDENT INDEXING

Data dependent hashing methods (DDH) focus on adapting hashing schemes to the data distribution. They often relate to indexing via an objective function. Most literature concentrates on hashing and quantization methods. Although previous works have achieved promising results in learning B-Trees (Kraska et al., 2018) or Lattice quantization (Sablayrolles et al., 2018), there are two major bottlenecks for DDH: (1) Theoretical insights are few applied in practice, (2) Complex index designs introduce computation overhead. In the theory of data dependent indexing, Andoni et al. (2018) present fruitful insights on hashing-based Approximate Nearest Neighbor search on metric space. However, there is not much literature that puts these insights into practice. Dong et al. (2019) propose a $k$-NN graph based algorithm for efficiently learning data dependent LSH in Euclidean space based on the insights given by Andoni et al. (2018). However, their method requires pre-computing a $k$-NN graph, which is computationally expensive at scale. In the experiments, Dong et al. (2019) only compare their method with $k$-means; their performance compared to major ANN benchmarks (Erik et al., 2018) remains unknown.

Many practical applications of LSH eschew DDH methods for vanilla LSH due to the aforementioned computational bottlenecks. However, despite having sub-linear query time in theory, query complexity for even vanilla LSH is still high in practice Datar et al.(2004); Andoni et al. (2017).

When designing the dynamic data structure, we want to minimize the total running time and balance the overhead and benefit brought by the data structure. In general, the matrix multiplication and backpropagation procedures comes from neural network training, which we can not control, hence, we want to balance the accuracy-efficiency trade-offs by selecting neurons.

This is especially pronounced in the Dynamic LSH regime, when weights are evolving, besides query time, LSH updates incur a more significant overhead that harms the overall efficiency. Chen et al. (2020) introduce a method to control update frequency, but their method requires heavy hyperparameter tuning and is not guaranteed to work for each benchmark. In contrast, our findings in the following section show that the update overhead can be significantly reduced while maintaining nearest-neighbor search quality during NN training. To our knowledge, this is the first time DDH techniques have been successfully applied to the Deep Learning setting.

### E.3 Efficient neural network training

The weights of a neural network dynamically change during the training process, which brings great challenges to the LSH implementation. Since the data are no longer static, we need to constantly update the hash table, which incurs extra computation cost that could harm the overall performance. We present a *dynamic data structure* to handle this issue, and our data structure achieves significant speedup over naive implementation under mild assumptions, without compromising the worst case guarantee. The dynamic data structure (shown in Algorithm 1) borrows ideas from the work of Cohen et al. (2019); Jiang et al. (2020c) (which are originally designed for linear programming), and we adapt it to the LSH setting. Our data structure generalizes several practical insights and turns practical heuristics into rigorous theory, which guarantees significant speed up under natural assumptions for training neural networks and ensure the worst guarantee at the same time. In particular, we generalize and provide the following three practical heuristics (i) only update significantly changed coordinates, (ii) batch updating LSH instead of sequential updating, (iii) using "prediction" on those "marginal" coordinates.

In this work, we primarily study two LSH based efficient NN training methods: SLIDE and REFORMER. SLIDE introduces LSH to select neurons in the forward pass and then only do gradient descent on the chosen neurons. The major trade-off of SLIDE is the rebuild overhead versus the accurate neuron selection. To accurately retrieve the neurons with high inner products, the hash tables are required to be updated. This rehashing and rebuilding will be the major overhead caused by SLIDE. Therefore, the SLIDE's speed over full NN training is determined by the relative magnitude of rebuild overhead compared to the saved back-propagation time. The major goal of REFORMER is to reduce the memory consumption of the transformer so that GPU based hardware can support sequence to sequence tasks with longer sequence length. REFORMER also shares the trade-off when rebuilding the hash tables for better retrieval of attention weights. Besides, learning hash functions of REFORMER is more challenging than SLIDE. From an information retrieval's perspective, in attention settings, each data vector is also a query. This unique setting increases the hardness for learning better hash functions.

## F EFFICIENT GPU IMPLEMENTATION

As noted by Chen et al. (2020), each input in one batch samples a different set of weights(neurons). These irregular operations significantly reduce the benefits of using GPUs: they correspond to different sizes of rows for each input, which makes the forward pass much more difficult to parallelize and thus, more challenging to map to GPUs than regular programs (Burtscher et al., 2012). For this reason, Chen et al. (2020) build their SLIDE system in C++ from scratch on CPU. Even though their implementation achieved remarkable speed up, their impact is limited as they implemented their system from scratch in C++, making it difficult for the community to adopt SLIDE in practice. To verify that such randomized algorithm is not GPU friendly, we implement the exact same algorithm for GPUs. We show in Section 4.1.1 that it indeed fails to gain any speed up.

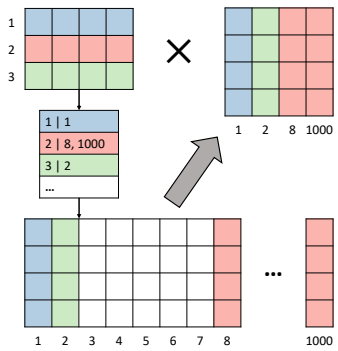


Figure 12: Visualization of how union selection within the batch instead of independent sets of neurons with variant length can avoid irregular memory access.

We design a variant of the algorithm to exploit fast matrix multiplications on GPUs. In the original SLIDE, each training example in a batch retrieves its own subset of weights. Here, we take a union of the retrieved subsets in each batch to avoid irregular and unbalanced memory access, shown in figure 12. Our implementation of this proposal is written in python under Pytorch framework with Cython compiled LSH and CUDA kernels for hashcode efficient computation. We believe this implementation would be more beneficial for the community as it can be easily plugged into any deep learning models.

**Practical Implementation:** In our design, we borrow insights from the theoretical guarantee from above and aim to find the sweet spot between theory limitations and practical challenges. First, instead of detecting weight changes at the cost of an extra copy, we reduce the problem to detect low quality of retrieved weights. In NN training setting, we measure quality as the inner product and low quality indicates a low inner product. We argue these two approaches are similar that they both signal the necessity of updating the data structure. Besides, rather than a soft margin for detecting data changes, which can be ambiguous under dynamic setting, inner product is a better measurement as it clearly indicates the performance of current data structure. More importantly from an efficiency perspective, quality detection comes almost for free because the inner product between the query embedding and retrieved neurons are necessary for the forward pass.

## G DISCUSSIONS

MONGOOSE shed lights on efficient NNS-ds for efficient training of deep neural networks. Especially, the slow change observation along with the smart scheduler demonstrate the possibility of applying NNS-ds with larger indexing overhead for dynamic similarity search where the distribution of data changes slowly. Equipped with both observation and algorithm, more NNS-ds could be involved in the deep learning community to tackle the efficiency issue.

## REFERENCES

Amin Ahmad, Noah Constant, Yinfei Yang, and Daniel Cer. Reqa: An evaluation for end-to-end answer retrieval models. *arXiv preprint arXiv:1907.04780*, 2019.

Alexandr Andoni and Piotr Indyk. Near-optimal hashing algorithms for approximate nearest neighbor in high dimensions. In *2006 47th annual IEEE symposium on foundations of computer science (FOCS'06)*, pp. 459–468. IEEE, 2006.

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)*, pp. 793–801, 2015.

Alexandr Andoni, Assaf Naor, Aleksandar Nikolov, Ilya Razenshteyn, and Erik Waingarten. Data-dependent hashing via nonlinear spectral gaps. In *Proceedings of the 50th Annual ACM SIGACT Symposium on Theory of Computing (STOC)*, pp. 787–800, 2018.

Jimmy Ba and Brendan Frey. Adaptive dropout for training deep neural networks. In *Advances in neural information processing systems (NeurIPS)*, pp. 3084–3092, 2013.

Yoshua Bengio, Réjean Ducharme, Pascal Vincent, and Christian Jauvin. A neural probabilistic language model. *Journal of machine learning research (JMLR)*, 3(Feb):1137–1155, 2003.

K. Bhatia, K. Dahiya, H. Jain, A. Mittal, Y. Prabhu, and M. Varma. The extreme classification repository: Multi-label datasets and code, 2016. URL `http://manikvarma.org/downloads/XC/XMLRepository.html`.

Leonid Boytsov, David Novak, Yury Malkov, and Eric Nyberg. Off the beaten path: Let's replace term-based retrieval with k-nn search. In *Proceedings of the 25th ACM international on conference on information and knowledge management (CIKM)*, pp. 1099–1108, 2016.

M. Burtscher, R. Nasre, and K. Pingali. A quantitative study of irregular programs on gpus. In *2012 IEEE International Symposium on Workload Characterization (IISWC)*, pp. 141–151, 2012.

Sarath Chandar, Sungjin Ahn, Hugo Larochelle, Pascal Vincent, Gerald Tesauro, and Yoshua Bengio. Hierarchical memory networks. *arXiv preprint arXiv:1605.07427*, 2016.

Wei-Cheng Chang, Felix X Yu, Yin-Wen Chang, Yiming Yang, and Sanjiv Kumar. Pre-training tasks for embedding-based large-scale retrieval. *arXiv preprint arXiv:2002.03932*, 2020.

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

Gal Chechik, Varun Sharma, Uri Shalit, and Samy Bengio. Large scale online learning of image similarity through ranking. *Journal of Machine Learning Research*, 11(3), 2010.

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 *MLSys*. `https://arxiv.org/pdf/1903.03129`, 2020.

Wenlin Chen, James Wilson, Stephen Tyree, Kilian Weinberger, and Yixin Chen. Compressing neural networks with the hashing trick. In *International conference on machine learning*, pp. 2285–2294, 2015.

Michael B Cohen, Yin Tat Lee, and Zhao Song. Solving linear programs in the current matrix multiplication time. In *Proceedings of the 51st annual ACM SIGACT symposium on theory of computing (STOC)*, pp. 938–942, 2019.

Benjamin Coleman, Anshumali Shrivastava, and Richard G Baraniuk. Race: Sub-linear memory sketches for approximate near-neighbor search on streaming data. *arXiv preprint arXiv:1902.06687*, 2019.

Paul Covington, Jay Adams, and Emre Sargin. Deep neural networks for youtube recommendations. In *Proceedings of the 10th ACM Conference on Recommender Systems*, pp. 191–198. ACM, 2016.

Yihe Dong, Piotr Indyk, Ilya Razenshteyn, and Tal Wagner. Learning space partitions for nearest neighbor search. In *International Conference on Learning Representations (ICLR)*, 2019.

Bernhardsson Erik, Aumüller Martin, and Faithfull Alexander. ANN Benchmarks. `https://github.com/erikbern/ann-benchmarks`, 2018.

Miao Fan, Jiacheng Guo, Shuai Zhu, Shuo Miao, Mingming Sun, and Ping Li. Mobius: Towards the next generation of query-ad matching in baidu's sponsored search. In *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, pp. 2509–2517. ACM, 2019.

Jianfeng Gao, Xiaodong He, Wen-tau Yih, and Li Deng. Learning continuous phrase representations for translation modeling. In *Proceedings of the 52nd Annual Meeting of the Association for Computational Linguistics (ACL)*, volume 1, pp. 699–709, 2014.

Ruiqi Guo et al. Quantization based fast inner product search. In *Artificial Intelligence and Statistics*, pp. 482–490, 2016.

Rob Hall and Josh Attenberg. Fast and accurate maximum inner product recommendations on map-reduce. In *Proceedings of the 24th International Conference on World Wide Web*, pp. 1263–1268. ACM, 2015.

Himanshu Jain, Yashoteja Prabhu, and Manik Varma. Extreme multi-label loss functions for recommendation, tagging, ranking & other missing label applications. In *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pp. 935–944. ACM, 2016.

Herve Jegou, Matthijs Douze, and Cordelia Schmid. Product quantization for nearest neighbor search. *IEEE transactions on pattern analysis and machine intelligence*, 33(1):117–128, 2011.

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

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*. `https://arxiv.org/pdf/2004.04250.pdf`, 2020b.

Shunhua Jiang, Zhao Song, Omri Weinstein, and Hengjie Zhang. Faster dynamic matrix inverse for faster lps. *arXiv preprint arXiv:2004.07470*, 2020c.

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

Nikita Kitaev, Łukasz Kaiser, and Anselm Levskaya. Reformer: The efficient transformer. In *ICLR*. `https://arxiv.org/pdf/2001.04451`, 2020b.

Tim Kraska, Alex Beutel, Ed H Chi, Jeffrey Dean, and Neoklis Polyzotis. The case for learned index structures. In *SIGMOD*, pp. 489–504, 2018.

Moontae Lee, Xiaodong He, Wen-tau Yih, Jianfeng Gao, Li Deng, and Paul Smolensky. Reasoning in vector space: An exploratory study of question answering. *arXiv preprint arXiv:1511.06426*, 2015.

Yury Malkov, Alexander Ponomarenko, Andrey Logvinov, and Vladimir Krylov. Scalable distributed algorithm for approximate nearest neighbor search problem in high dimensional general metric spaces. In *International Conference on Similarity Search and Applications*, pp. 132–147. Springer, 2012.

Yury Malkov, Alexander Ponomarenko, Andrey Logvinov, and Vladimir Krylov. Approximate nearest neighbor algorithm based on navigable small world graphs. *Information Systems*, 45:61–68, 2014.

Yury A Malkov and Dmitry A Yashunin. Efficient and robust approximate nearest neighbor search using hierarchical navigable small world graphs. *IEEE transactions on pattern analysis and machine intelligence*, 2018.

Ioannis Partalas, Aris Kosmopoulos, Nicolas Baskiotis, Thierry Artieres, George Paliouras, Eric Gaussier, Ion Androutsopoulos, Massih-Reza Amini, and Patrick Galinari. Lshtc: A benchmark for large-scale text classification. *arXiv preprint arXiv:1503.08581*, 2015.

Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Kopf, Edward Yang, Zachary DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. Pytorch: An imperative style, high-performance deep learning library. In *Advances in Neural Information Processing Systems 32*, pp. 8024–8035. Curran Associates, Inc., 2019.

Jeffrey Pennington, Richard Socher, and Christopher D Manning. Glove: Global vectors for word representation. In *Proceedings of the 2014 conference on empirical methods in natural language processing (EMNLP)*, pp. 1532–1543, 2014.

Jack Rae, Jonathan J Hunt, Ivo Danihelka, Timothy Harley, Andrew W Senior, Gregory Wayne, Alex Graves, and Timothy Lillicrap. Scaling memory-augmented neural networks with sparse reads and writes. In *Advances in Neural Information Processing Systems*, pp. 3621–3629, 2016.

Parikshit Ram and Alexander G Gray. Maximum inner-product search using cone trees. In *Proceedings of the 18th ACM SIGKDD international conference on Knowledge discovery and data mining (KDD)*, pp. 931–939. ACM, 2012.

Hubert Ramsauer, Bernhard Schäfl, Johannes Lehner, Philipp Seidl, Michael Widrich, Lukas Gruber, Markus Holzleitner, Milena Pavlović, Geir Kjetil Sandve, Victor Greiff, et al. Hopfield networks is all you need. *arXiv preprint arXiv:2008.02217*, 2020.

Alexandre Sablayrolles, Matthijs Douze, Cordelia Schmid, and Hervé Jégou. Spreading vectors for similarity search. *arXiv preprint arXiv:1806.03198*, 2018.

Minjoon Seo, Jinhyuk Lee, Tom Kwiatkowski, Ankur P Parikh, Ali Farhadi, and Hannaneh Hajishirzi. Real-time open-domain question answering with dense-sparse phrase index. *arXiv preprint arXiv:1906.05807*, 2019.

Aliaksei Severyn and Alessandro Moschitti. Learning to rank short text pairs with convolutional deep neural networks. In *Proceedings of the 38th International ACM SIGIR Conference on Research and Development in Information Retrieval (SIGIR)*, pp. 373–382, Santiago, Chile, 2015.

Anshumali Shrivastava and Ping Li. Asymmetric lsh (alsh) for sublinear time maximum inner product search (mips). In *Advances in Neural Information Processing Systems*, pp. 2321–2329, 2014a.

Anshumali Shrivastava and Ping Li. Improved asymmetric locality sensitive hashing (alsh) for maximum inner product search (mips). *arXiv preprint arXiv:1410.5410*, 2014b.

Ryan Spring and Anshumali Shrivastava. Scalable and sustainable deep learning via randomized hashing. In *Proceedings of the 23rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pp. 445–454, 2017.

Yukihiro Tagami. Annexml: Approximate nearest neighbor search for extreme multi-label classification. In *Proceedings of the 23rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pp. 455–464. ACM, 2017.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. In *Advances in neural information processing systems (NeurIPS)*, pp. 5998–6008, 2017.

Jingdong Wang, Ting Zhang, Nicu Sebe, Heng Tao Shen, et al. A survey on learning to hash. *IEEE transactions on pattern analysis and machine intelligence*, 40(4):769–790, 2017.

Donna Xu, Ivor W Tsang, and Ying Zhang. Online product quantization. *IEEE Transactions on Knowledge and Data Engineering*, 30(11):2185–2198, 2018.

Hong-Jian Xue, Xinyu Dai, Jianbing Zhang, Shujian Huang, and Jiajun Chen. Deep matrix factorization models for recommender systems. In *IJCAI*, pp. 3203–3209, 2017.