Evaluating the Feasibility of Sampling-Based Techniques for Training Multilayer Perceptrons

Sana Ebrahimi Rishi Advani Abolfazl Asudeh

University of Illinois Chicago
{sebrah7, radvani2, asudeh}@uic.edu

28th International Conference on Extending Database Technology
March 28, 2025 – Barcelona, Spain

Outline

Motivation

A scalability issue

- DNNs are often very large in scale:
    - Computationally expensive to train
    - Requiring powerful hardware, including expensive GPUs
    - CPUs are widely available! why not using them instead?

Motivation

A scalability issue

- DNNs are often very large in scale:
  - Computationally expensive to train
  - Requiring powerful hardware, including expensive GPUs
  - CPUs are widely available! why not using them instead?

Extended Database Technologies for Machine Learning

- Feasibility of the extension of sampling-based techniques for efficient training of deep neural networks (DNNs)
- On CPU machines with limited resources

Motivation

A scalability issue

- DNNs are often very large in scale:
  - Computationally expensive to train
  - Requiring powerful hardware, including expensive GPUs
  - CPUs are widely available! why not using them instead?

Extended Database Technologies for Machine Learning

- Feasibility of the extension of sampling-based techniques for efficient training of deep neural networks (DNNs)
- On CPU machines with limited resources

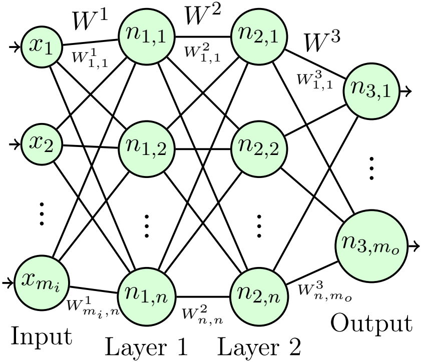
Two scalability directions

1. Sampling-based approaches based on locality-sensitive hashing (LSH) [GIM99]
2. Monte-carlo (MC) estimations [Rob16]

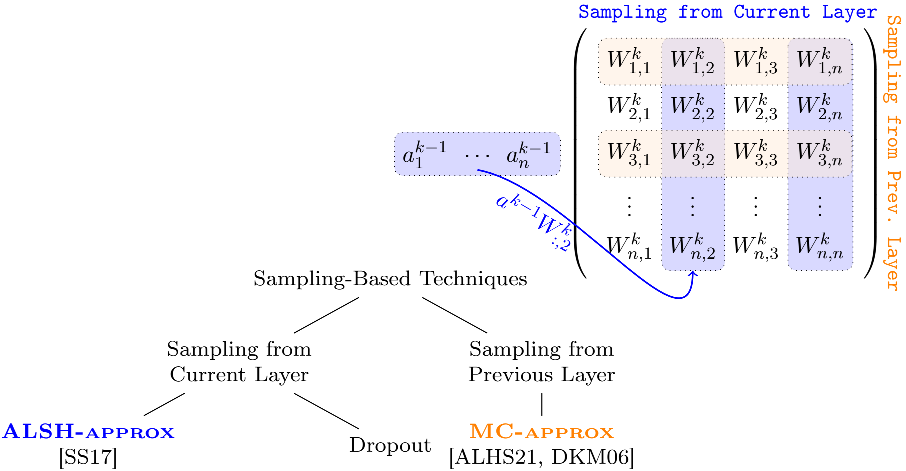
Outline

Review: multi-layer perceptron (MLP)

$$z^k = a^{k-1} W^k + b^k$$
$$a^k = f(z^k)$$



Taxonomy of Sampling-Based Techniques



Outline

Asymmetric Locality-Sensitive Hashing

A family $\mathcal{H}$ is $(\tau, c, p_1 > p_2)$-sensitive for c-NNS, if for all $h \in \mathcal{H}$:

$$sim(w,a) \geq \tau \quad \implies \Pr[h(Q(a)) = h(P(w))] \geq p_1$$
$$sim(w,a) \leq c\tau \quad \implies \Pr[h(Q(a)) = h(P(w))] \leq p_2$$

For w, $a \in \mathbb{R}^n$ with $\|w\| \leq C$, where C is a constant less than 1, and $\|a\| = 1$, they define the transformations P and Q for the inner product as follows.

$$\begin{aligned} P\colon \mathbb{R}^n \to \mathbb{R}^{n+m}, &\quad w \mapsto \left[w; \|w\|^{2^1}, \ldots, \|w\|^{2^m}\right] \\ Q\colon \mathbb{R}^n \to \mathbb{R}^{n+m}, &\quad a \mapsto \left[a; 1/2, \ldots, 1/2\right] \end{aligned} \tag{1}$$

ALSH-approx

- Only computes the high-value elements of $a^{k-1}W^k$ (called active nodes), skipping the computation for the small values.

Sampling from Current Layer

$$\begin{pmatrix} a_1^{k-1} & \cdots & a_n^{k-1} \end{pmatrix} \begin{pmatrix} W_{1,1}^k & \boxed{W_{1,2}^k} & W_{1,3}^k & \boxed{W_{1,n}^k} \\ W_{2,1}^k & W_{2,2}^k & W_{2,3}^k & W_{2,n}^k \\ W_{3,1}^k & W_{3,2}^k & W_{3,3}^k & W_{3,n}^k \\ \vdots & \vdots & \vdots & \vdots \\ W_{n,1}^k & W_{n,2}^k & W_{n,3}^k & W_{n,n}^k \end{pmatrix}$$

$a^{k-1}W_{:,2}^k$

Uses ALSH to find active nodes: the columns $W_{:,j}^k$ that collide with a^{k-1} into the same bin.

$$\text{argmax}_j \langle W_{:,j}^k, a^{k-1} \rangle \approx \text{argmin}_j \| Q(a^{k-1}) - P(W_{:,j}^k) \|$$

Outline

Approximate Matrix Multiplication $A \times B$

- $AB_{i,j} = \sum_{t=1}^{n} A_{i,t} B_{t,j}$
- Goal: A Monte-carlo method to estimate the sum over a subset of values of t
- Standard approach: uniformly select a sample of columns (high estimation error)
- Idea (weighted sampling): compute the values that are expected to be larger

$$p_t = \frac{\|A_{:,t}\| \cdot \|B_{t,:}\|}{\sum_{k=1}^{n} \|A_{:,k}\| \cdot \|B_{k,:}\|}$$

For a set C of c samples:

$$AB_{i,j} \approx \sum_{t \in C} \frac{1}{c p_t} A_{i,t} B_{t,j}$$

MC-approx

- Computes all values of $a^{k-1}W^k$, but instead of exact computation, it estimates each value.
- It is more appropriate for mini-batch settings, where a matrix of activation vectors is multiplied to W^k.
- For single-row settings, it may not accurately estimate p_t values $\Rightarrow$ its estimations are not accurate.

$$\begin{pmatrix} a_1^{k-1} & \cdots & a_n^{k-1} \end{pmatrix} \xrightarrow{a^{k-1}W^k_{:,2}} \begin{pmatrix} W^k_{1,1} & W^k_{1,2} & W^k_{1,3} & W^k_{1,n} \\ W^k_{2,1} & W^k_{2,2} & W^k_{2,3} & W^k_{2,n} \\ W^k_{3,1} & W^k_{3,2} & W^k_{3,3} & W^k_{3,n} \\ \vdots & \vdots & \vdots & \vdots \\ W^k_{n,1} & W^k_{n,2} & W^k_{n,3} & W^k_{n,n} \end{pmatrix}$$

Sampling from Prev. Layer

Outline

Negative Result: ALSH-APPROX does not scale

Theorem

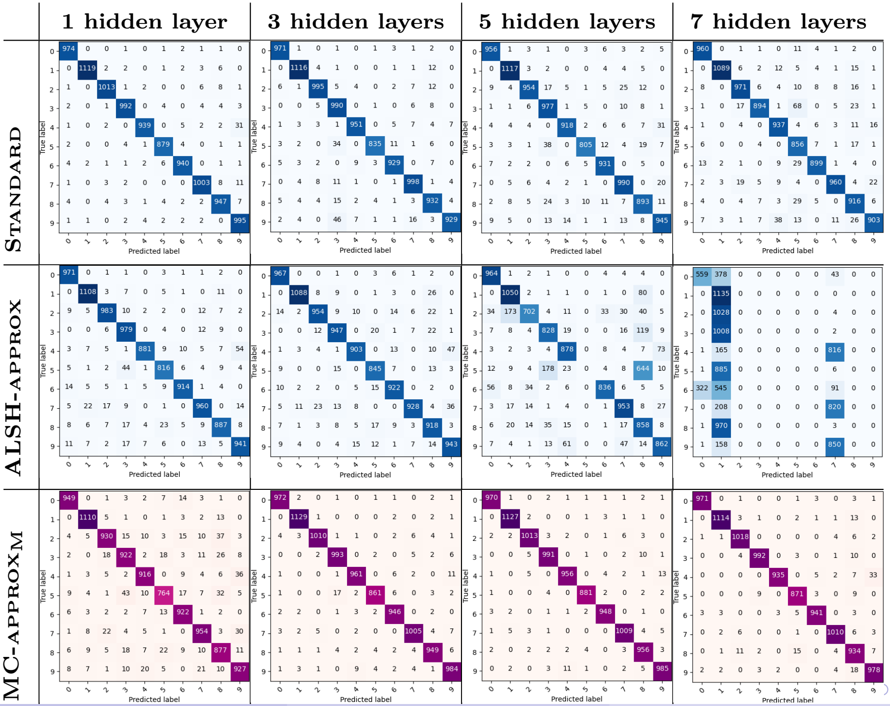
Let f be a linear activation function such that $a = f(z) = z$. Suppose for any node n_p^l, $\sum_{i \in \uparrow_p^l} a_i^{l-1} W_{i,p} = c \sum_{i \notin \uparrow_p^l} a_i^{l-1} W_{i,p}$. Then, $a_j^k = \bar{a}_j^k \left(\frac{c+1}{c}\right)^k$. That is,

$$e_j^k = \bar{a}_j^k \left(\left(\frac{c+1}{c} \right)^k - 1 \right)$$

Example: suppose $c = 5$ (i.e., the weighted sum for the active nodes is five times that of the inactive nodes). Then,

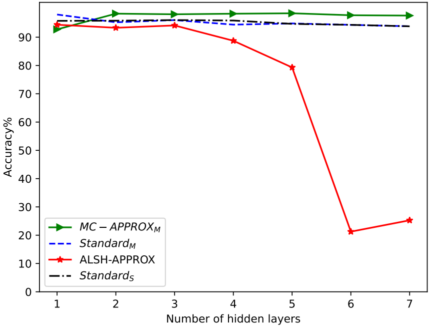
k (number of layers)	1	2	3	4	5	6
$\mathbf{e_j^k / \bar{a}_j^k}$	0.2	0.44	0.72	1.07	1.48	1.98

Outline

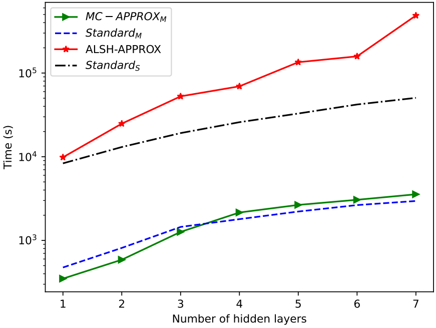

1 hidden layer
3 hidden layers
5 hidden layers
7 hidden layers
STANDARD
ALSH-APPROX
MC-APPROXM
True label
Predicted label

Time & Accuracy Comparison

Test accuracy

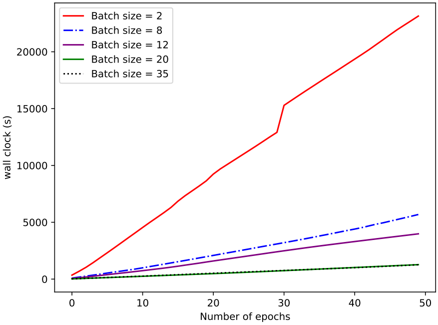


Training time

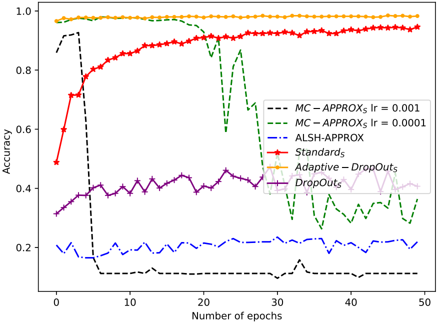


Time & Accuracy Comparison: MC-APPROX$_s$ failed

MC-APPROX training time



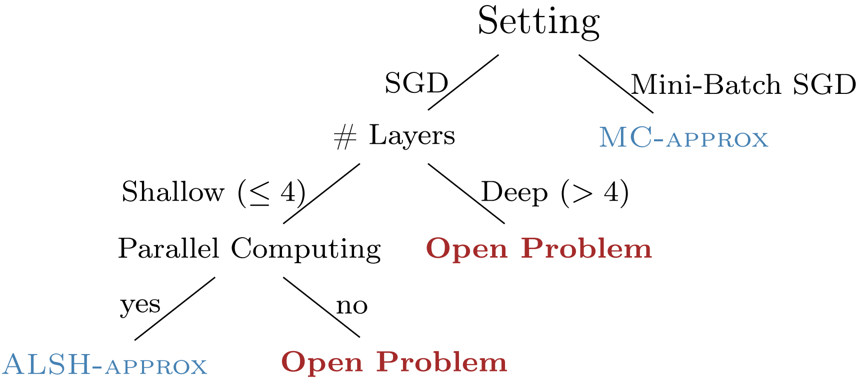
Validation accuracies (7 hidden layers)



Outline

Summary

Sampling-based Techniques for Training DNNs on CPU machines:



Next Step Advertisement!

Efficient Matrix Multiplication for Accelerating Inference in Binary and Ternary Neural Networks

- Complexity: $O(\frac{n^2}{\log n})$ time and memory
- Experiments: up to 6X less memory
- On CPU. C++: up to 30X faster; NumPy: 24X faster
- LLM Inference (Llama3.0, Falcon3, etc.), without system-level optimizations:
  - on CPU: up to 5.24X faster; on GPU: up to 2.5X faster

ArXiv Preprint

Github Repo.

Thank you!

- InDeX Lab: cs.uic.edu/∼indexlab/

Sana Ebrahimi

Rishi Advani

References

Menachem Adelman, Kfir Levy, Ido Hakimi, and Mark Silberstein.
Faster neural network training with approximate tensor operations.
In *NeurIPS*, 2021.

Petros Drineas, Ravi Kannan, and Michael W. Mahoney.
Fast Monte Carlo algorithms for matrices i: Approximating matrix multiplication.
SIAM Journal on Computing, 36(1):132–157, January 2006.

Aristides Gionis, Piotr Indyk, and Rajeev Motwani.
Similarity search in high dimensions via hashing.
In *25th International Conference on Very Large Data Bases*, 1999.

Christian P. Robert.
Monte Carlo Methods.
Wiley, August 2016.

Ryan Spring and Anshumali Shrivastava.
Scalable and sustainable deep learning via randomized hashing.
In *ACM SIGKDD*, pages 445–454, 2017.