SLIDE : In Defense of Smart Algorithms over Hardware Acceleration for Large-Scale Deep Learning Systems

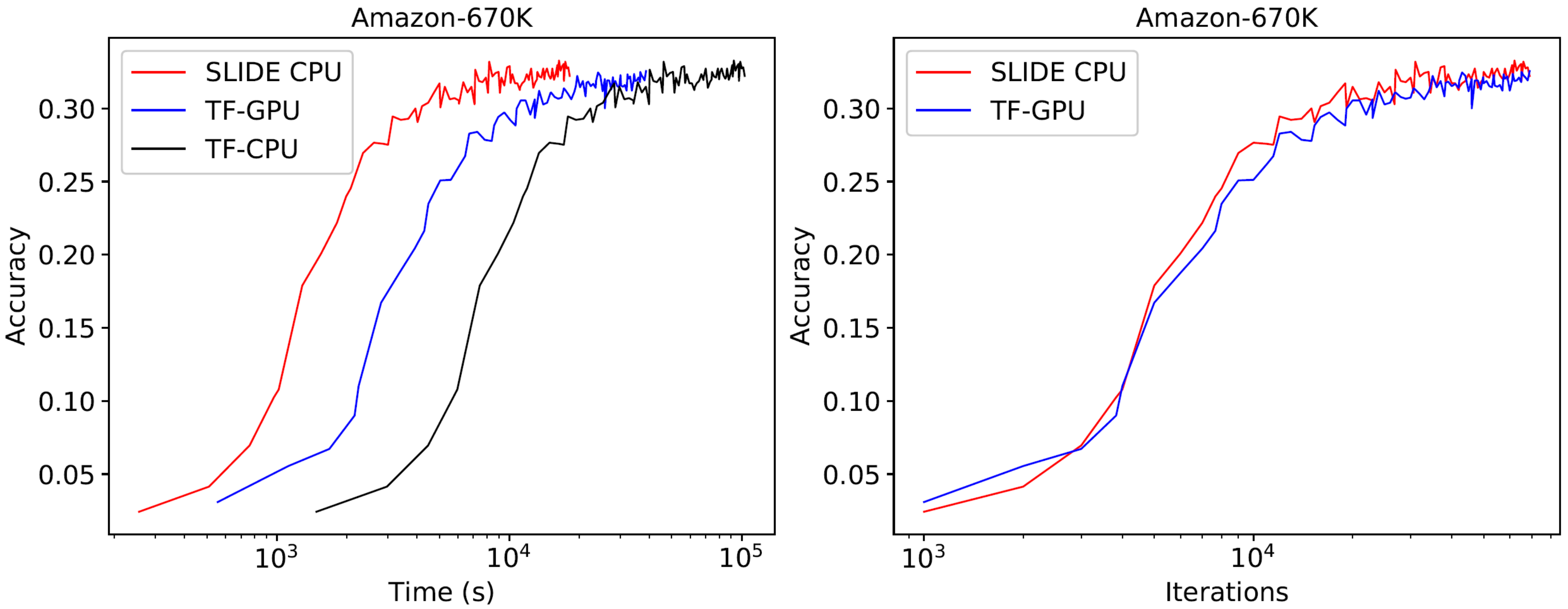
Beidi Chen

Collaborators: **Tharun Medini**[*], **James Farwell**[†] , **Sameh Gobriel**[†], **Charlie Tai** [†], **Anshumali Shrivastava**[*]

[*] Rice University, [†]Intel

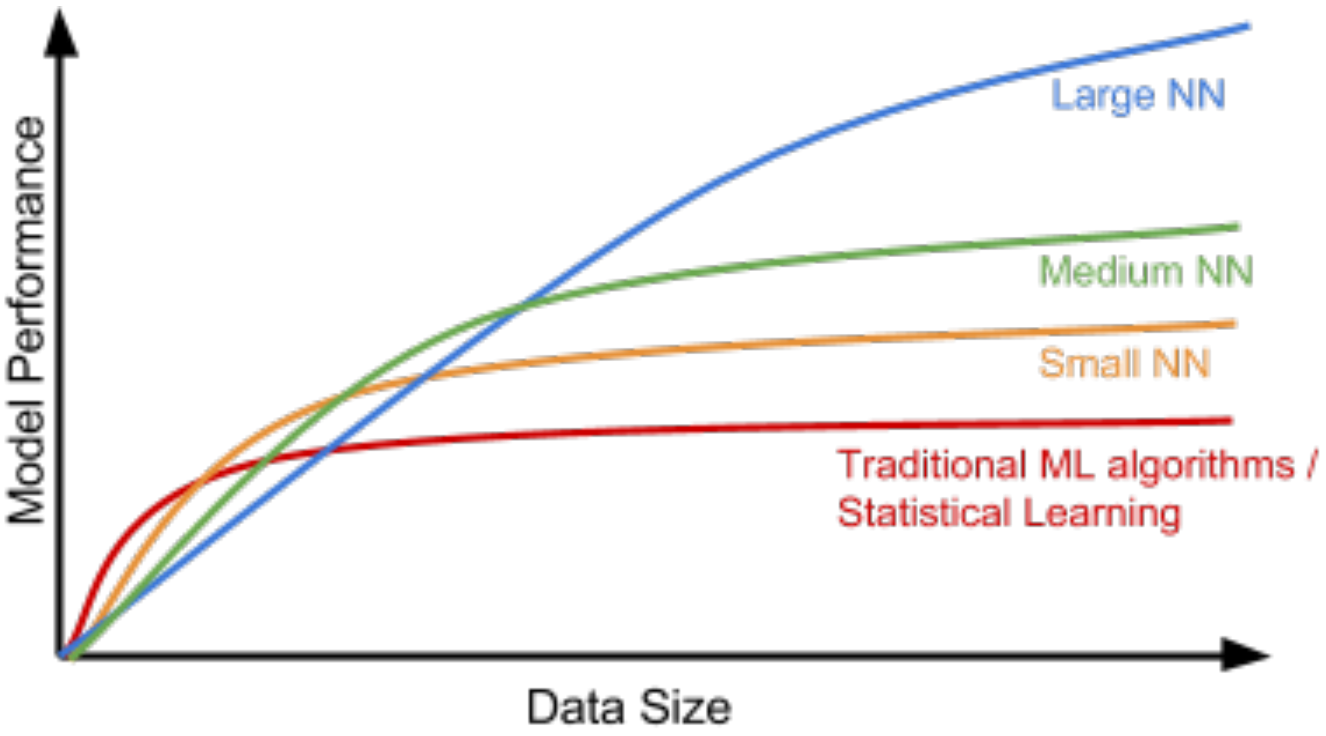
MLSys 2020

Our SLIDE System (C++ from scratch) on a **44 core CPU beats TF on V100 (1 hours vs 3.5 hours).** 100+ million parameter networks. TF on same CPU is 16 hours with all HPC optimization (Intel MKL-DNN).



3.5x faster on CPU than TF on V100 (Log Scale in Time)

The Age of Large Networks



- More Data
- Large Models
- Tons of Engineering
- Backpropagation

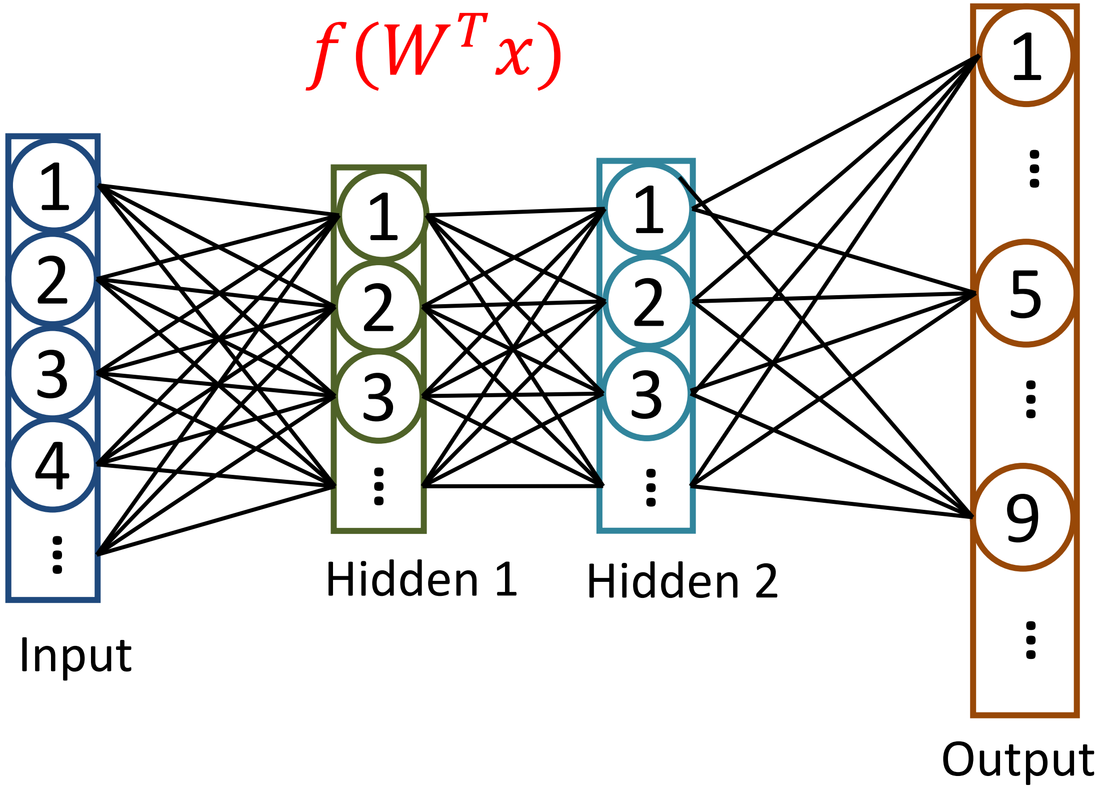
(Aka Simple Gradient Descent)

Fully Connected NN

Giant Matrix Multiplication for **every** data point in **each** epoch

(Forward + Backward)

$f(W^T x)$



Challenges

Do we really need all the computations?

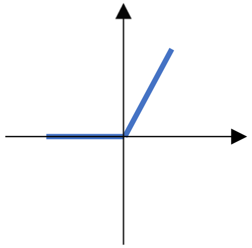
No!!

Good News: Only high activations are important

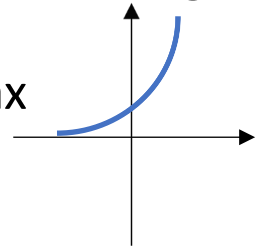
- Sampling few neurons in proportion of activations is enough (**Adaptive Dropouts**)

  (Ba et al. Neurips 13 , Makhzani et al. Neurips 15)

- Relu filtered negative activations (50% sparsity by design)



- Softmax



Bad News: We need to compute all to identify (or sample) the high activation neurons.

NO SAVINGS

The Fundamental Sampling Puzzle

Given N fixed sampling weights, $\{w_1, w_2, \dots, w_N\}$.

- Task: Sample x_i with probability w_i
  - Cost of 1 sample $O(N)$.
  - Cost of K samples $O(N)$.

Given N time-varying sampling weights (activations) $\{w_1^t, w_2^t, \dots, w_N^t\}$.

- Task: At time t, sample x_i with probability w_i^t
  - Cost of sampling O(N), at every time t.
  - **Last Few years of work in Locality Sensitive Hashing:** If $w_i^t = f(sim(\theta_t, x_i))$, for a specific set of f and sim, then $O(1)$ every time after and initial preprocessing cost of $O(N)$.

Textbook Hashing (Dictionary)

Hashing: Function h that maps a given data point ($x \in R^D$) to an integer key $h : R^D \mapsto \{0, 1, 2, \dots, N\}$. $h(x)$ serves as a discrete fingerprint.

Property (Ideal Hash Functions):

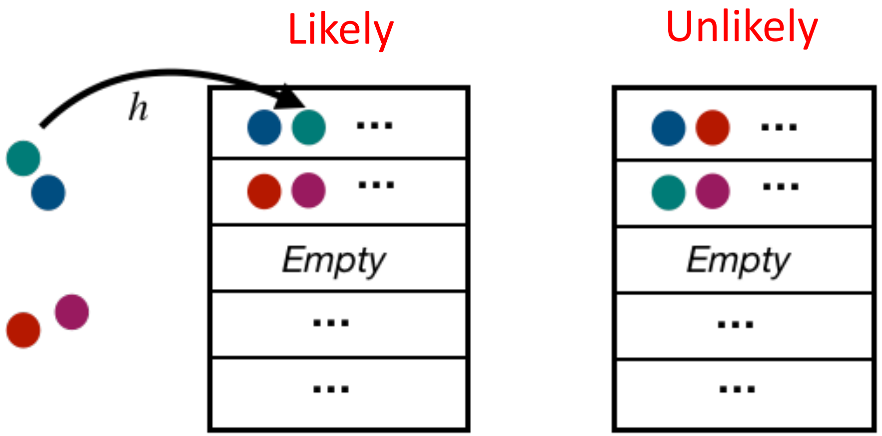
- If $\mathrm{x} = y$, then $h(x) = h(y)$
- If $\mathrm{x} \neq y$, then $h(x) \neq h(y)$

Probabilistic Fingerprinting (Hashing) (late 90s)

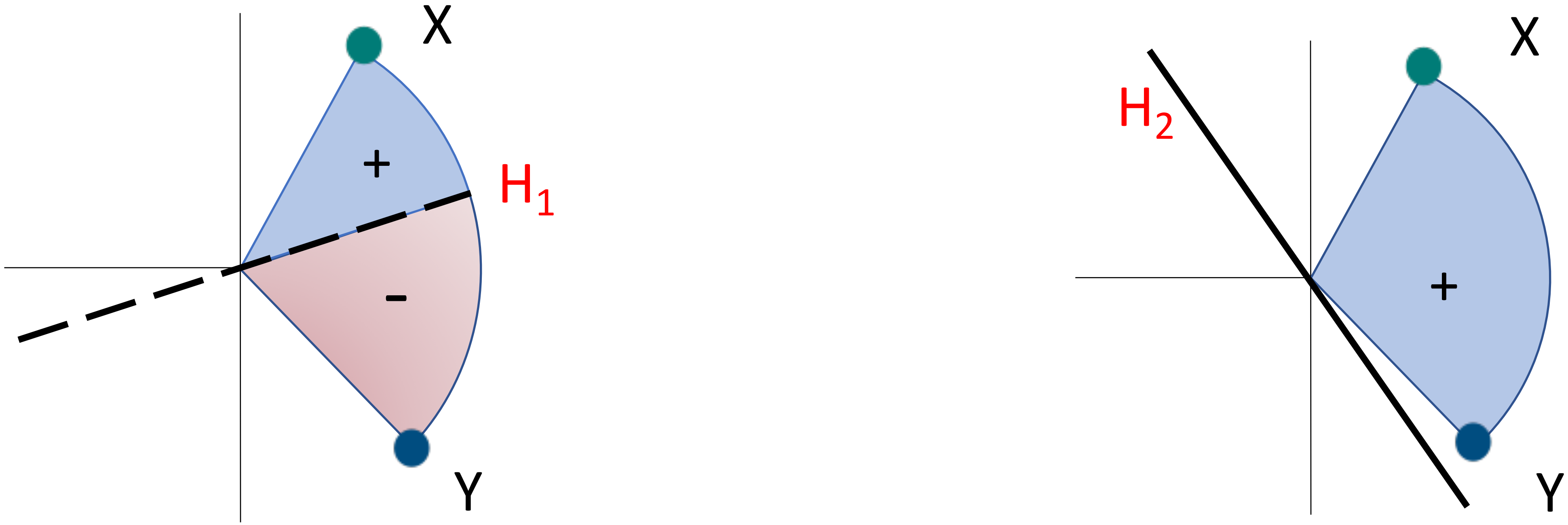
Hashing: Function **(Randomized)** h that maps a given data point ($x \in R^D$) to an integer key $h : R^D \mapsto \{0, 1, 2, \ldots, N\}$. $h(x)$ serves as a discrete fingerprint.

Locality Sensitive Property:

- If ~~$x = y$~~ $Sim(x, y)$ is high, then ~~$h(x) = h(y)$~~ $\Pr(h(x) = h(y))$ is high
- If ~~$x \neq y$~~ $Sim(x, y)$ is low, then ~~$h(x) \neq h(y)$~~ $\Pr(h(x) = h(y))$ is low



Example 1: Signed Random Projection (SRP)



$$Pr(h(x) = h(y)) = 1 - \frac{1}{\pi}\cos^{-1}(\theta)$$ monotonic in θ

A classical result from Goemans-Williamson (95)

Example 2: (Densified) Winner Take All

Original Vectors:

x	0, 0, 5, 0, 0, 7, 6, 0, 0
y	0, 0, 1, 0, 0, 0, 0, 0, 0

K=3

WTA hash codes:
(ICCV 2011)

	Bin 1	Bin 2	Bin 3	Bin 4	Bin 5	Bin 6
Θ	2, 1, 8	5, 3, 9	6, 2, 4	8, 9, 1	1, 7, 3	2, 4, 5
$\Theta(x)$	0, 0, 0 (E)	0, 5, 0	7, 0, 0	0, 0, 0 (E)	0, 6, 5	0, 0, 0 (E)
$\Theta(y)$	0, 0, 0 (E)	0, 1, 0	0, 0, 0 (E)	0, 0, 0 (E)	0, 0, 1	0, 0, 0 (E)
$\mathcal{H}_{wta}(x)$	1 (E)	2	1	1 (E)	2	1 (E)
$\mathcal{H}_{wta}(y)$	1 (E)	2	1 (E)	1 (E)	3	1 (E)

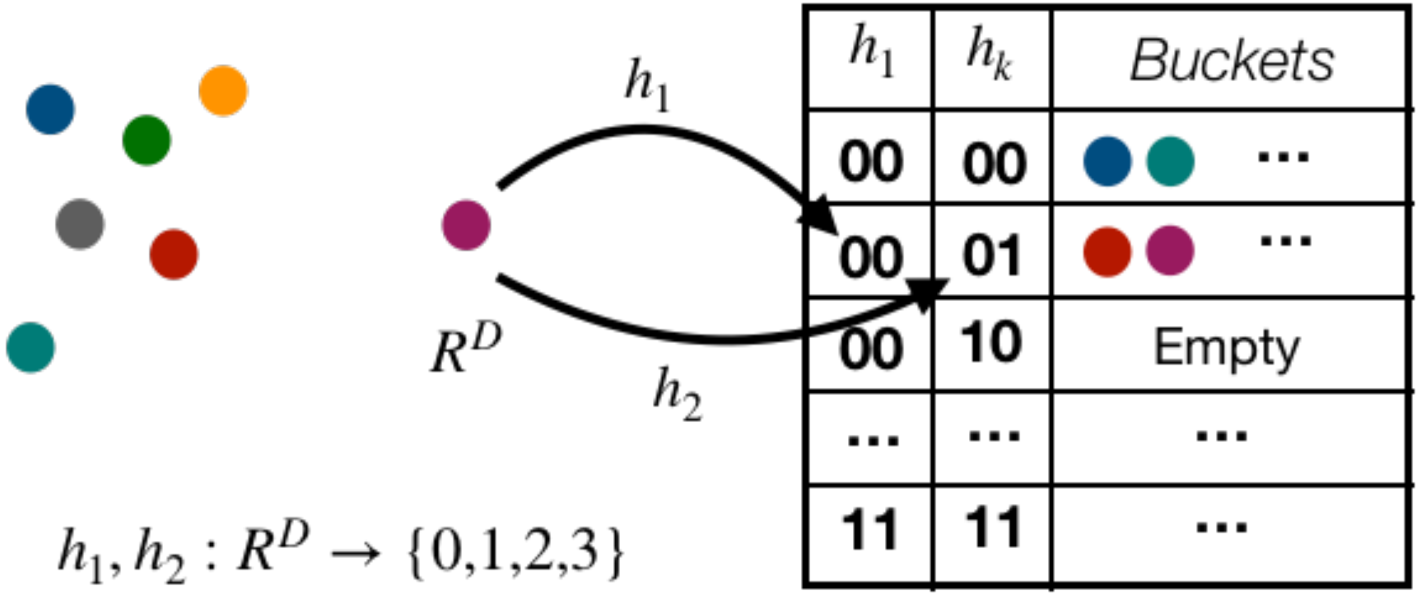
DWTA hash codes:
(UAI 2018)

$\mathcal{H}_{Dwta}(x)$	1+3*C	2	1	2+1*C	2	2+2*C
$\mathcal{H}_{Dwta}(y)$	3+3*C	2	2+3*C	3+4*C	3	2+2*C

Yagnik (ICCV11), Chen and Shrivastava (UAI 18)

Probabilistic Hash Tables

Given: $Pr_h\left[h(x) = h(y)\right] = f(sim(x,y)),\ f$ is monotonic.



- Given query, if $h_1(q) = 11$ **and** $h_2(q) = 01$, then probe bucket with index **1101**. It is a good bucket !!
- (Locality Sensitive) $h_i(q) = h_i(x)$ noisy indicator of high similarity.
- Doing better than random !!

LSH for Search (Known)

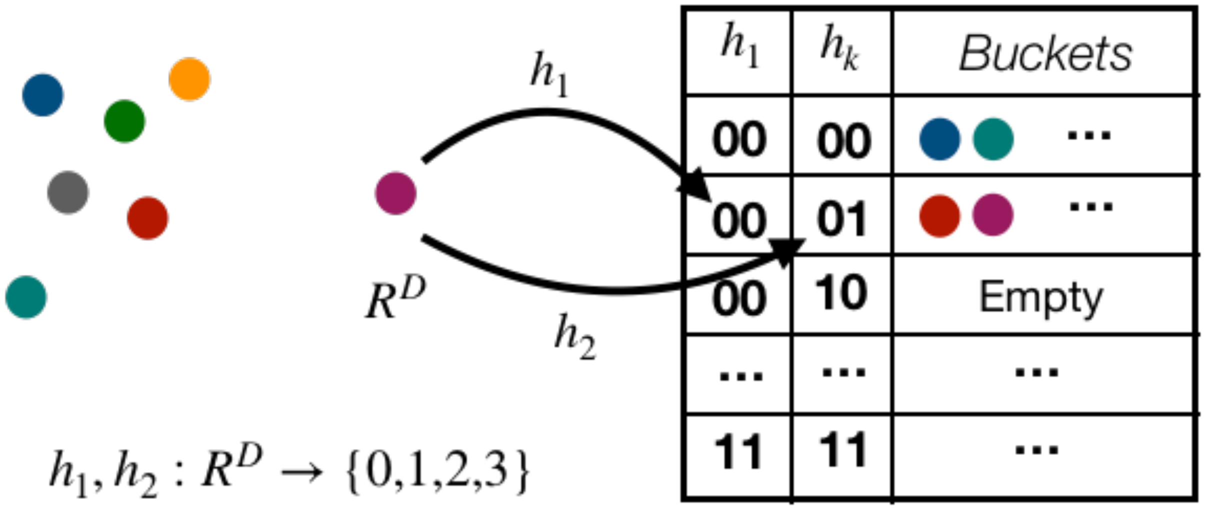
Theory

- Super-linear $O(N^{1+\rho})$ memory
- Sub-linear query time, O(N^{ρ})
- $\rho < 1$ but generally large (close to 1) and often hard to determine

Practical Issues

- Needs lot of hash tables and distance computations for good accuracy on near-neighbors
- Buckets can be quite heavy. Poor randomness, or unfavorable data distributions

New View: Data Structures for Efficient Sampling!



Is LSH really a search algorithm?

- Given the query θ_t, LSH samples x_i from the dataset, with probability $w_i^t = 1 - \left(1 - \mathrm{p}(\mathrm{x_i}, \theta_t)^{\mathrm{K}}\right)^{\mathrm{L}}$
- w_i^t is proportional to $\mathrm{p}(\mathrm{x_i}, \theta_t)^{\mathrm{K}}$ and the some similarity of $\mathrm{x_i}, \theta_t$
- LSH is considered a black box for nearest-neighbor search. It is not!!

LSH as Samplers

We can pre-process the dataset D, such that

- Given any query q, we can sample $x \in D$ with probability $Const \times [1-(1-p(q,x)^K)^L]$ in KL hash computation and L bucket probes.
- Even K = 1, L =1 is adaptive. So O(1) time adaptive.
- **Adaptive**: x is sampled with higher probability than y
  - if and only if sim(q,x) > sim(q,y)

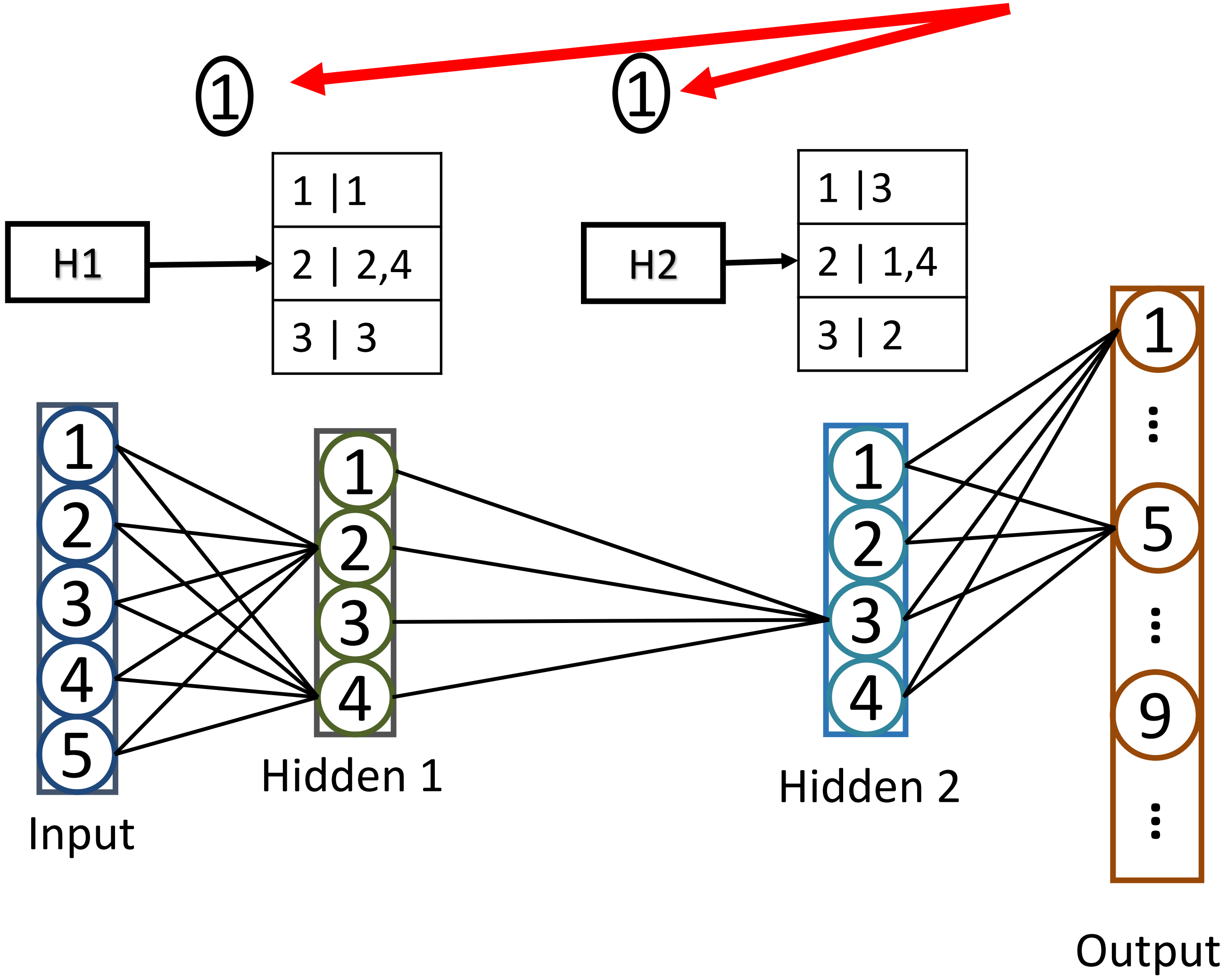
We can exactly compute the sampling probability.

- Const = No of elements sampled/ No of elements in Buckets

  (Chen et al. NeurIPS 2019)

Sufficient for Importance Sampling Estimations. Sampling cost O(1).

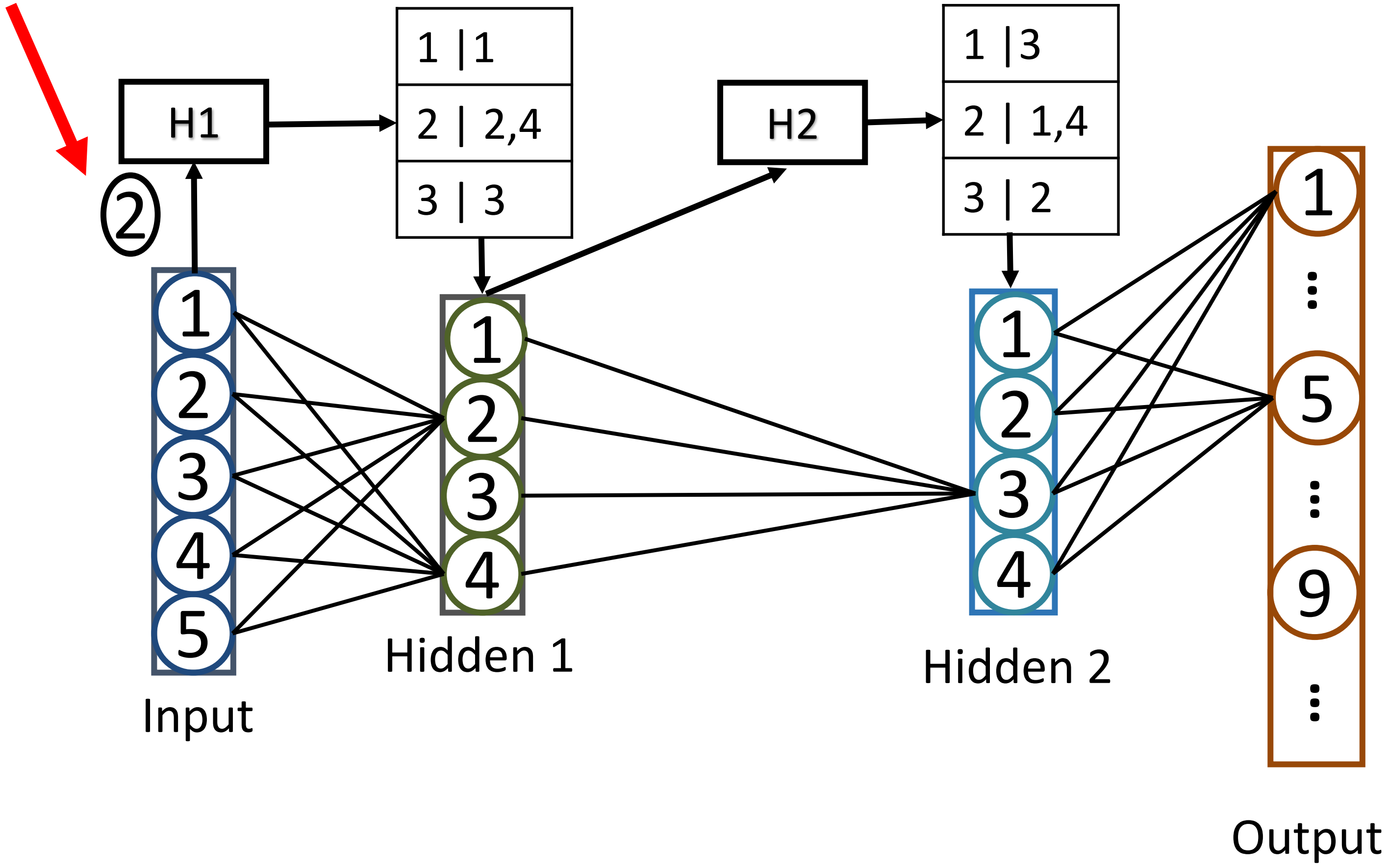
SLIDE: Sub-LInear Deep learning Engine



Step 1 – Build the hash tables by processing the weights of the hidden layers (initialization).

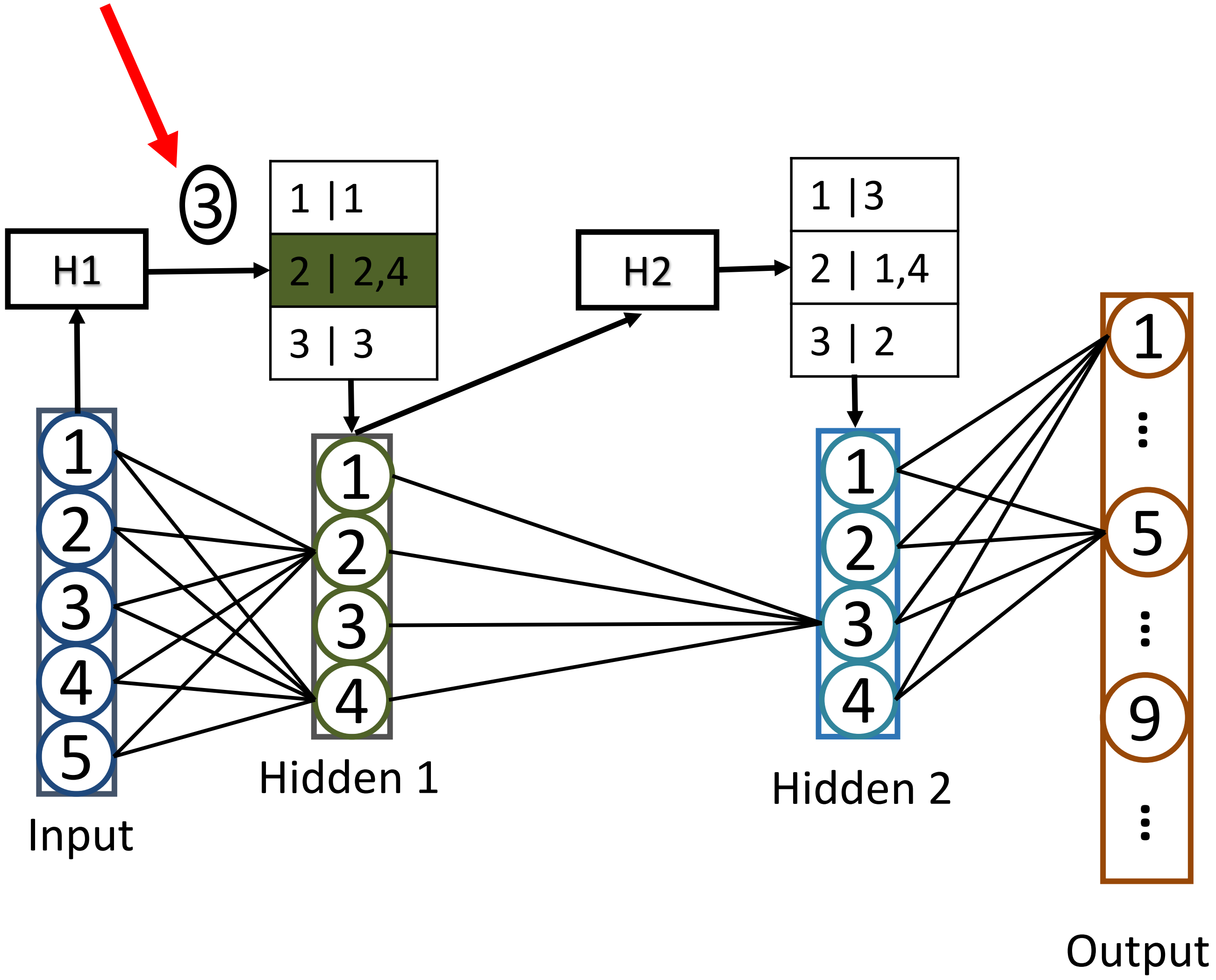
Subtlety: Neurons (vectors) in hash tables are not the data vectors. Reorganizing neurons.

SLIDE: Sub-LInear Deep learning Engine



Step 2 – Hash the input to any given layer using its randomized hash function.

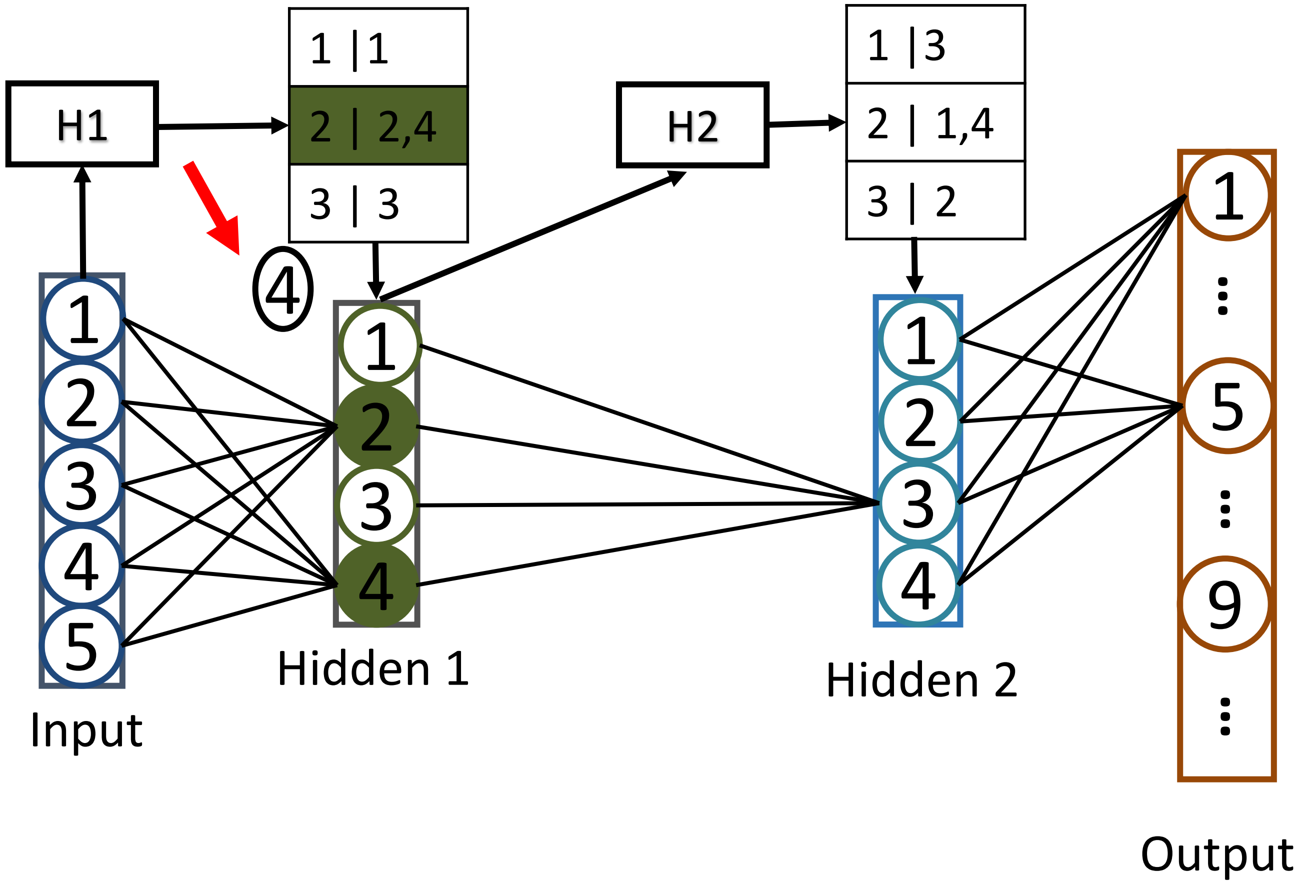
SLIDE: Sub-LInear Deep learning Engine



Step 3 – Query the hidden layer's hash table(s) for the active set using integer fingerprint.

Sample neurons in proportion to their activations.

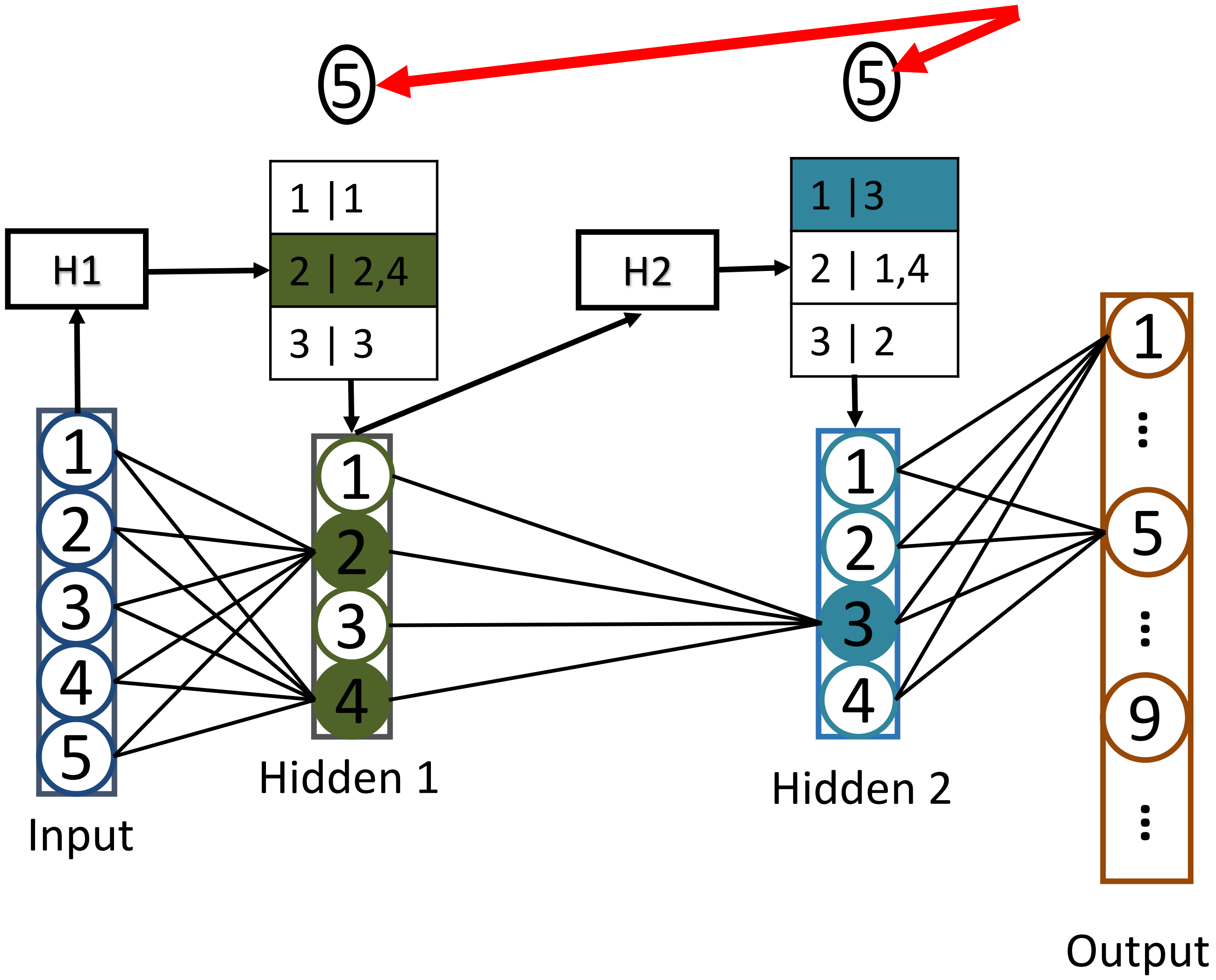
SLIDE: Sub-LInear Deep learning Engine



Step 4 – Perform forward and back propagation only on the nodes in the active set.

Computation is in the same order of active neurons.

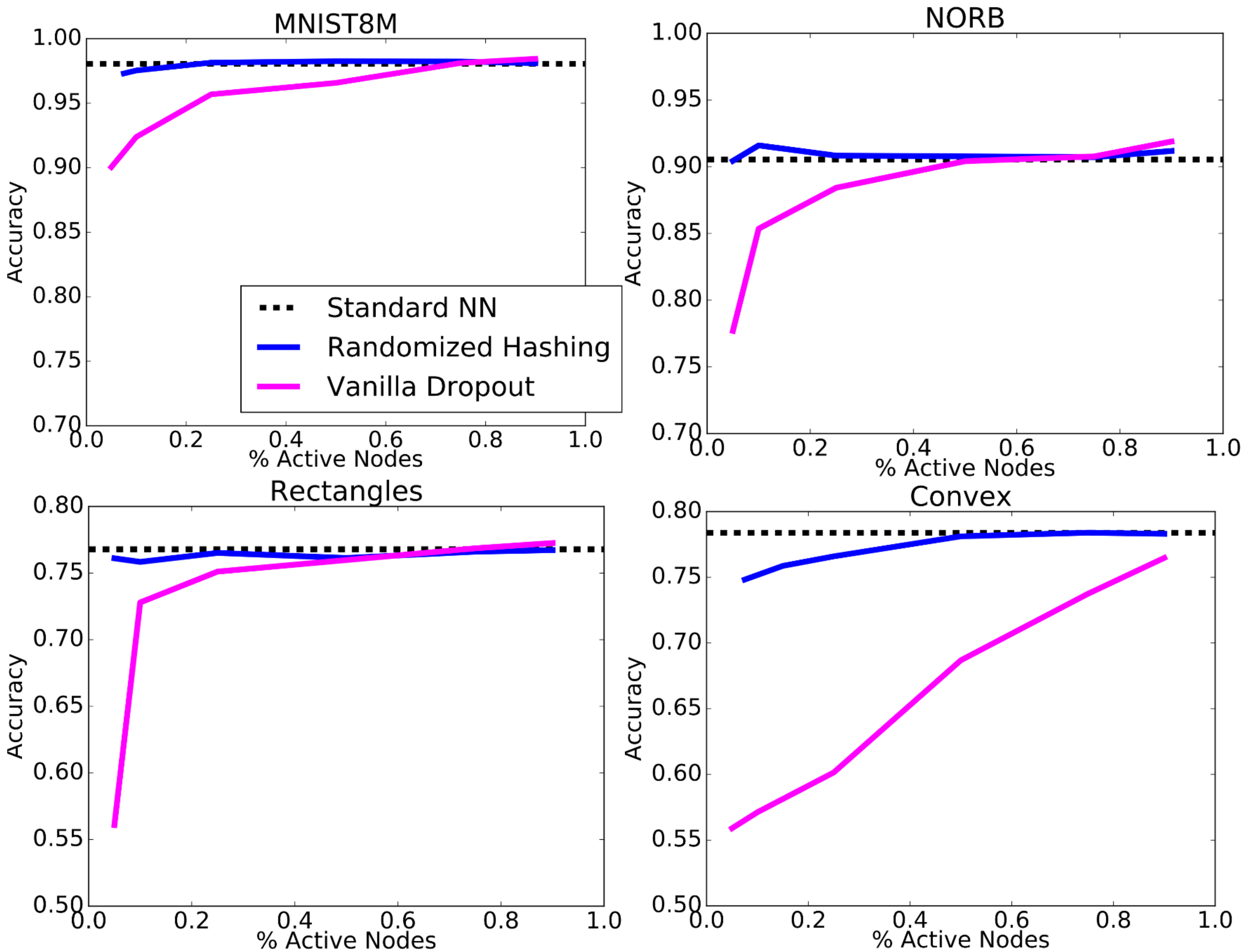
SLIDE: Sub-LInear Deep learning Engine



Step 5 – Update hash tables by rehashing the updated node weights.

Computation is in the same order of active neurons.

We can go very sparse if Adaptive

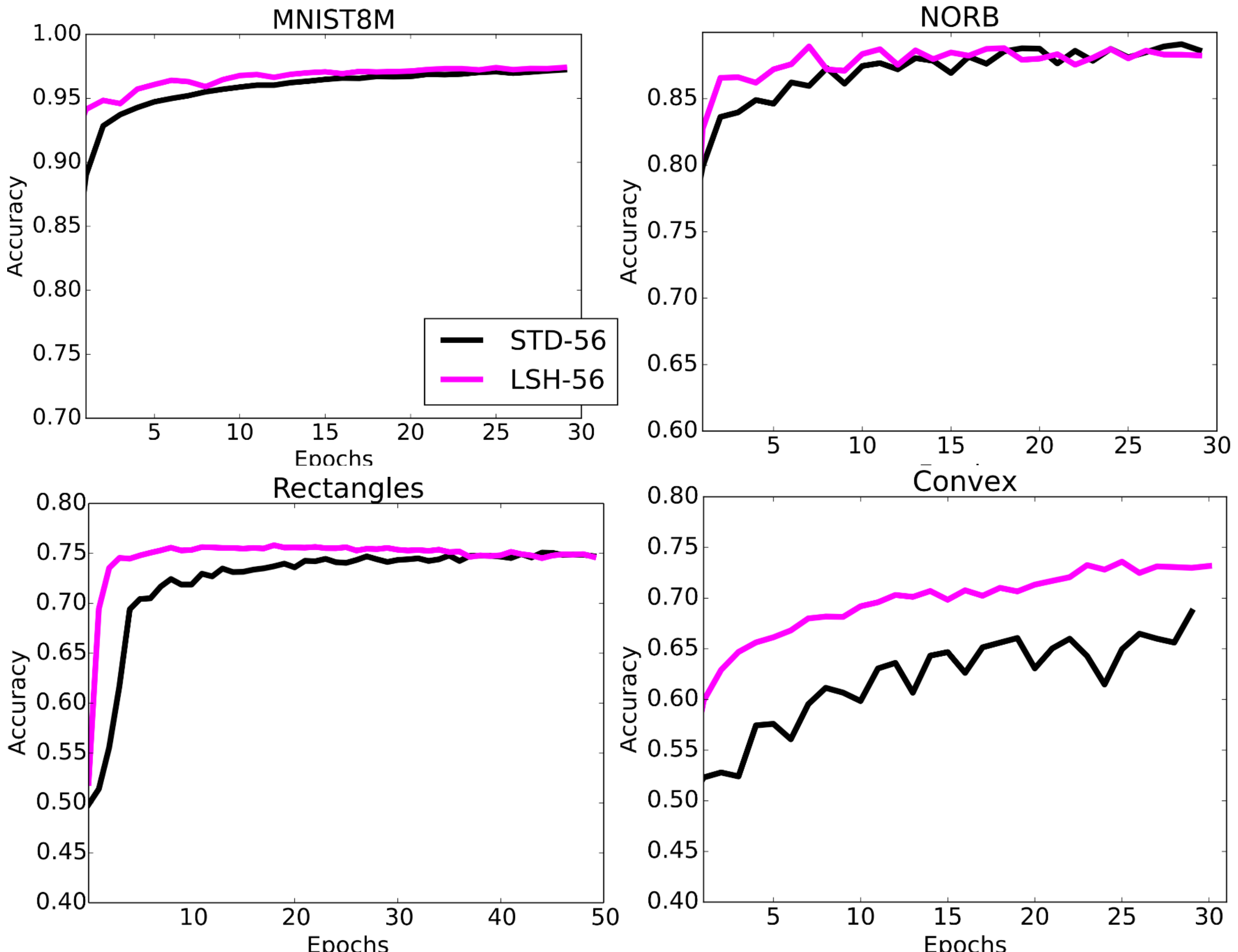


- Reduce both training and inference cost by 95%!
- Significantly more for larger networks.

(The wider the better)

- 2 Hidden Layers
- 1000 Nodes Per Layer

Sparsity + Randomness → Asynchronous Updates



- 3 Hidden Layers
- 1000 Nodes Per Layer

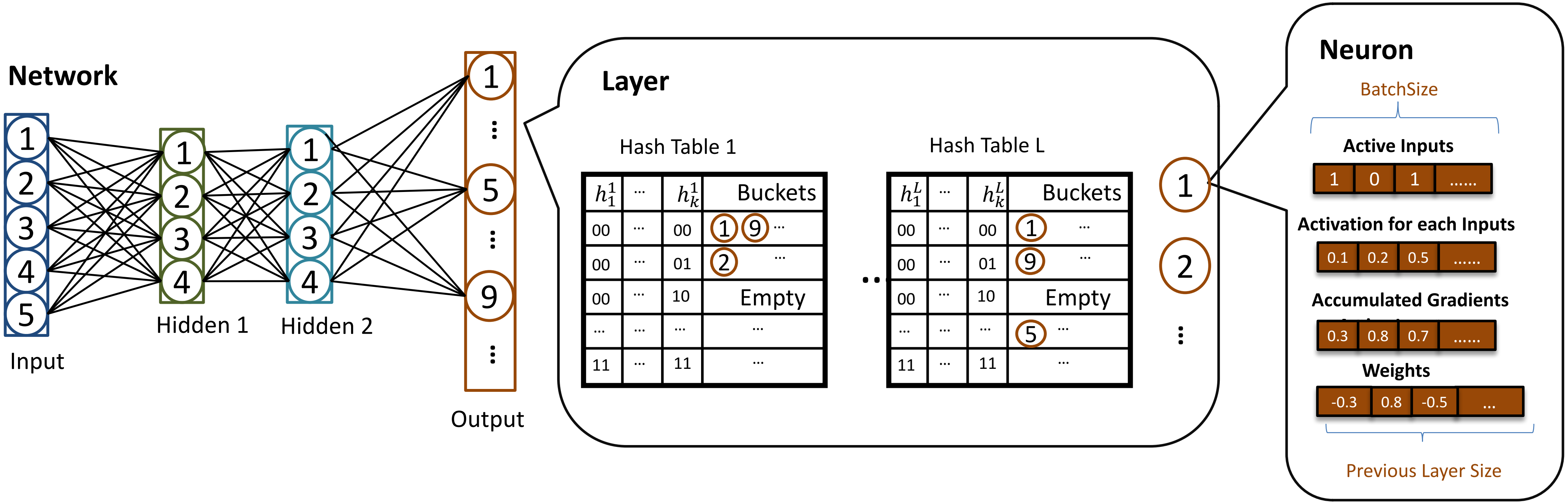
Less Computations + Asynchronous Parallelism

- Each update is computationally very small (100x+ reduction in computation and energy)

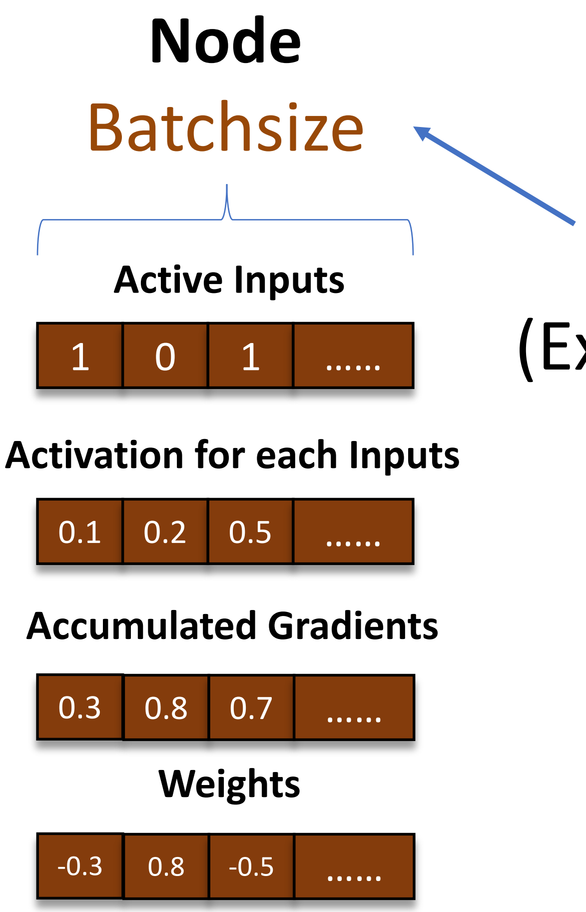
- Updates are near-independent, very low chance of conflict. Hence, parallel SGD!

RICE

SLIDE: Sub-LInear Deep learning Engine



Parallelism with OpenMP



Parallel across training samples in a batch
(Extreme sparsity and randomness in gradient updates)

Thanks to the theory of **HOGWILD**!

(Recht et al. Neurips 11)

Flexible choices of Hash Functions

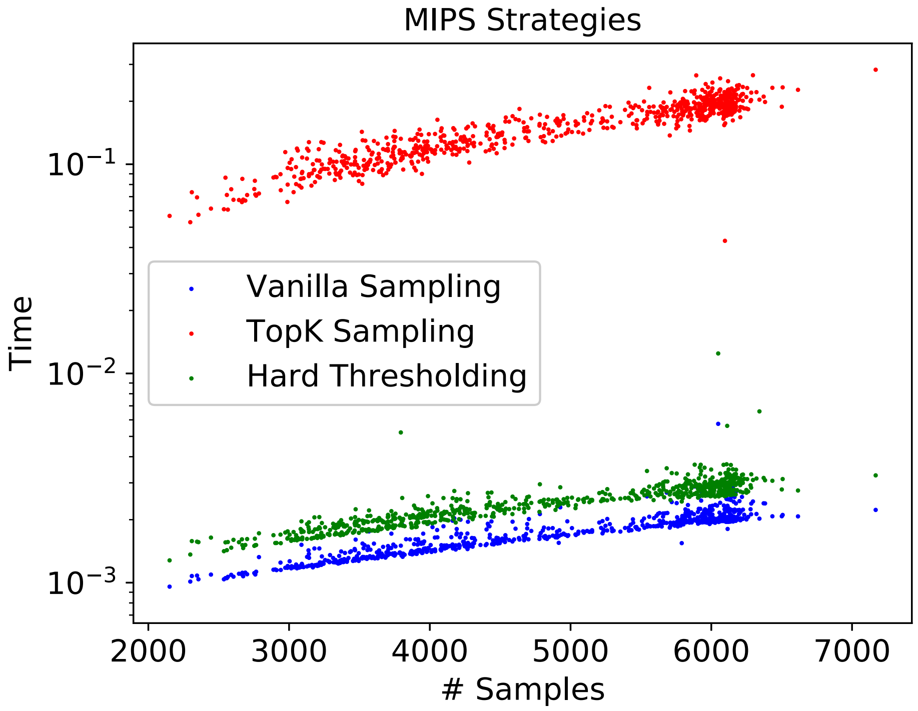
SLIDE supports four different LSH hash functions

- Simhash (cosine similarity)
- Winner-take-all Hashing (order)
- Densified Winner-take-all Hashing (for sparse data)*
- Minhash (jaccard similarity)

Easily add more!

Design Choices for Speed

- Vanilla sub-sampling:
  - choose sub-samples uniformly
- Top K sub-sampling:
  - rank samples and choose topk
- Hard Thresholding sub-sampling:
  - choose sub-samples that occur > *threshold* times



Micro-Architecture Optimization

Cache Optimization

Transparent Hugepages

Vector Processing

Software Pipelining and Prefetching

Looks Good on Paper. Does it change anything?

Baseline

State-of-the-art optimized Implementations

- TF on Intel Xeon E5-2699A v4 @ 2.40GHz CPU (FMA,AVX, AVX2, SSE4.2)
- TF on NVIDIA Tesla V100 (32GB)

VS.

SLIDE on Intel Xeon E5-2699A v4 @ 2.40GHz CPU (FMA,AVX, AVX2, SSE4.2)

- TF on NVIDIA Tesla V100 (32GB)

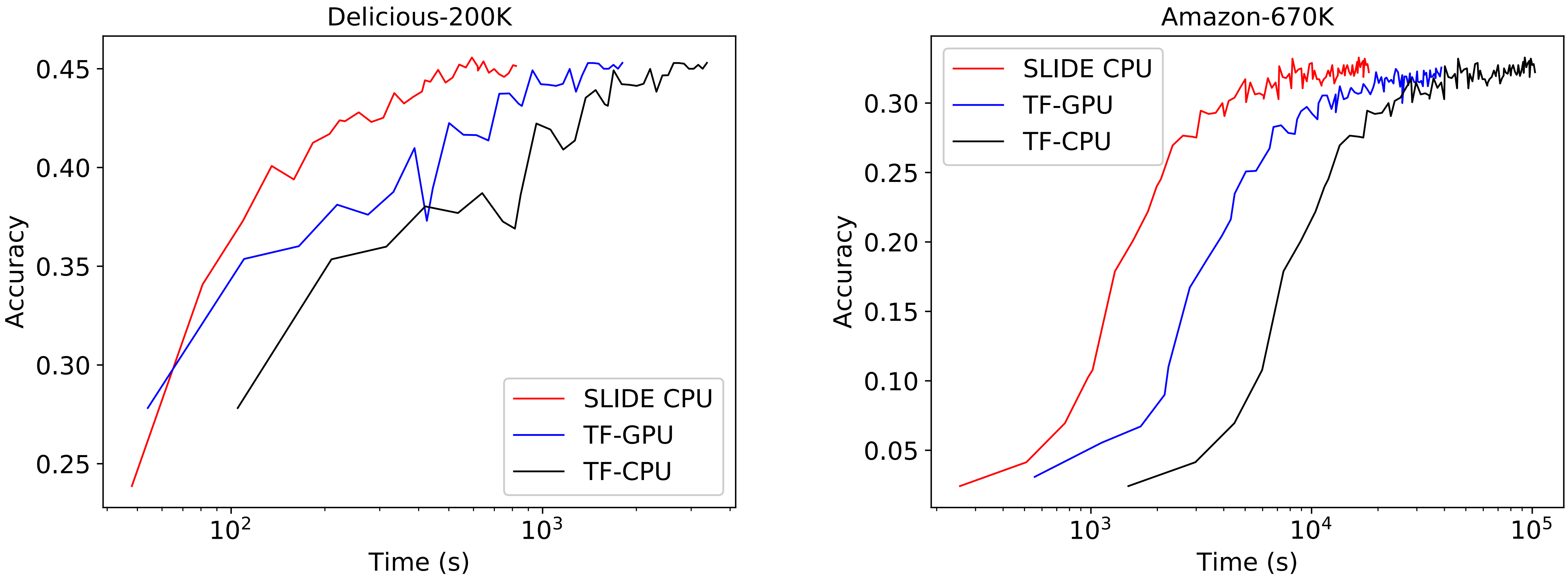
Datasets

	Delicious-200K	Amazon-670K
Feature Dim	782,585	135,909
Feature Sparsity	0.038 %	0.055 %
Label Dim	205,443	670,091
Training Size	196,606	490,449
Testing Size	100,095	153,025

Network Architectures (Fully Connected)

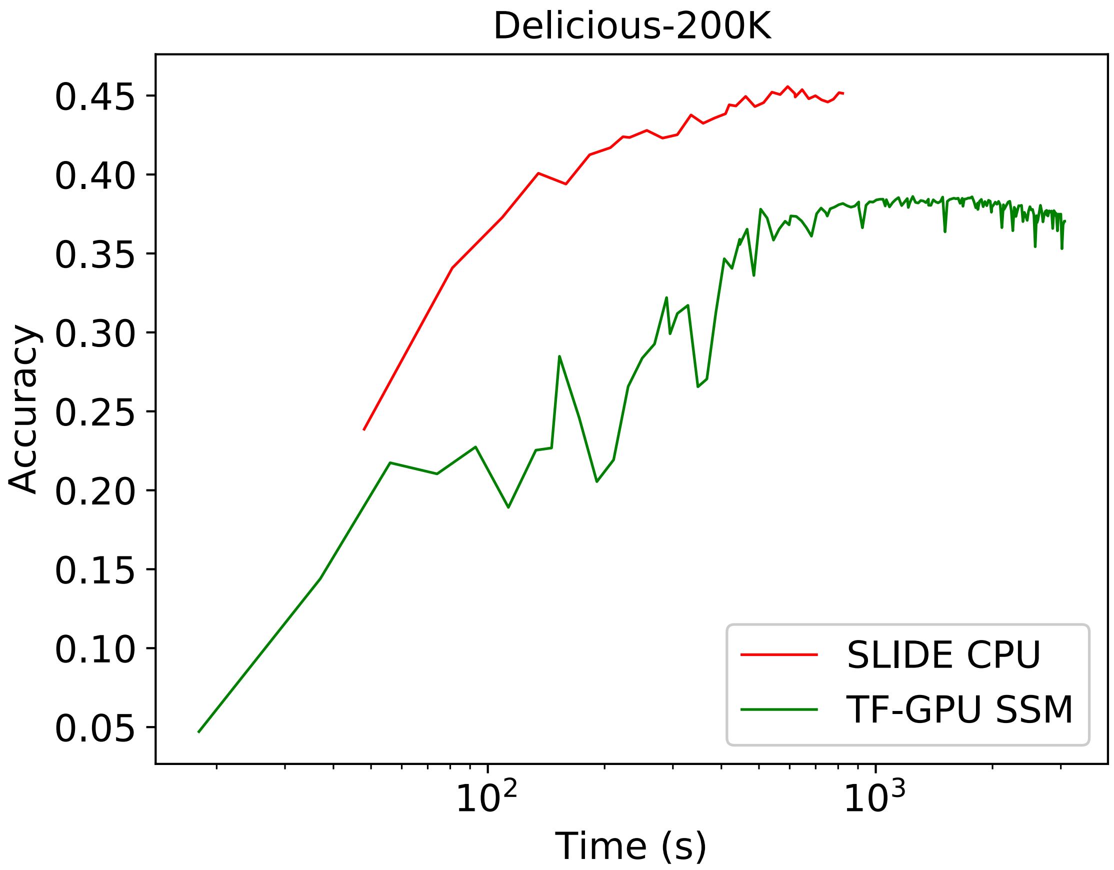
- Delicious-200K $782,585 \Rightarrow 128 \Rightarrow 205,443$ (126 million parameters)
- Amazon-670K $135,909 \Rightarrow 128 \Rightarrow 670,091$ (103 million parameters)

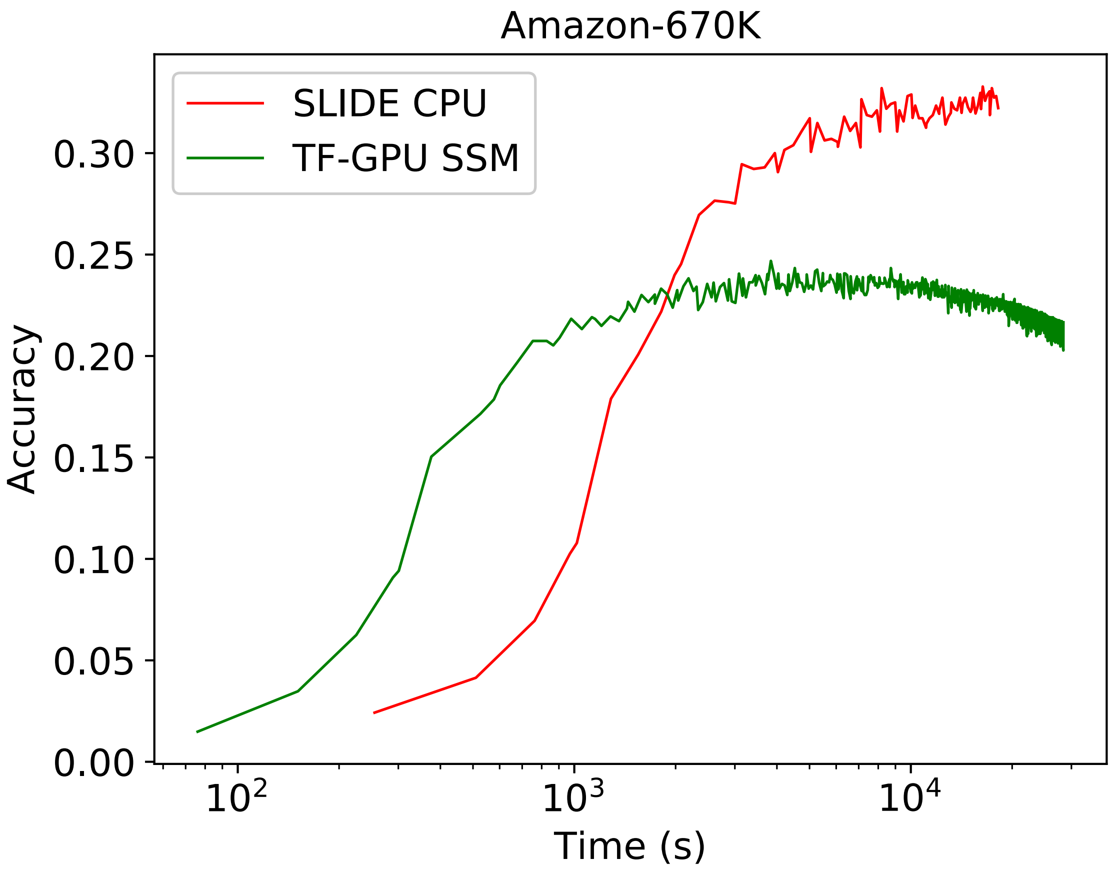
Performance



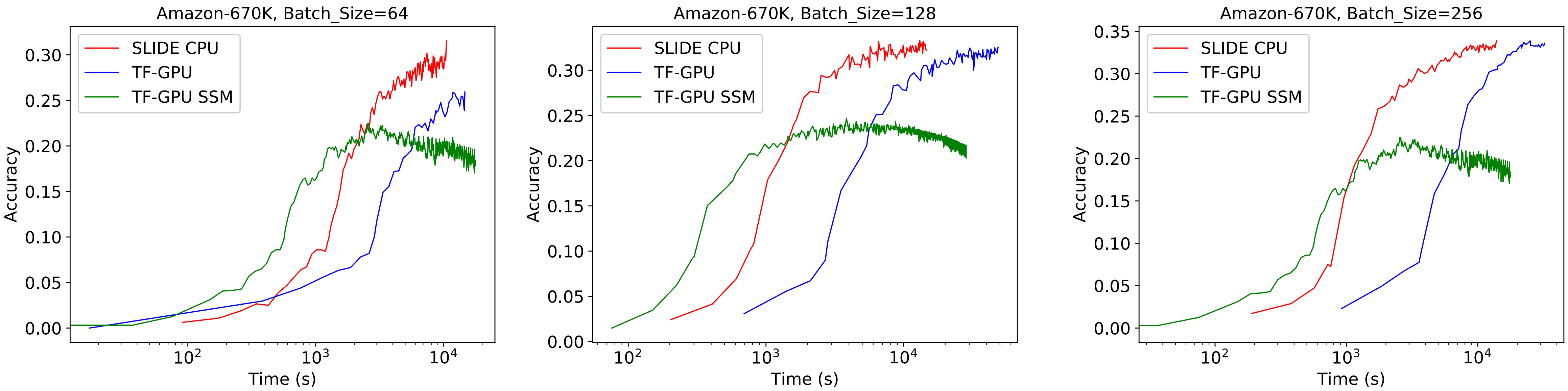
RICE

Performance compared to sampled softmax





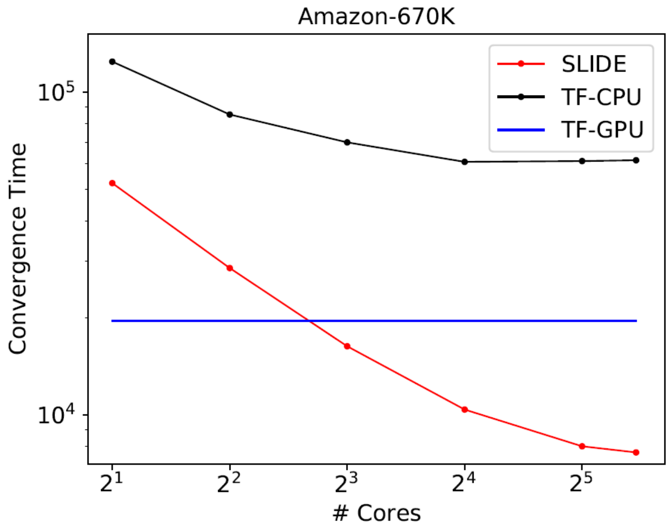
Performance @ Different Batchsizes

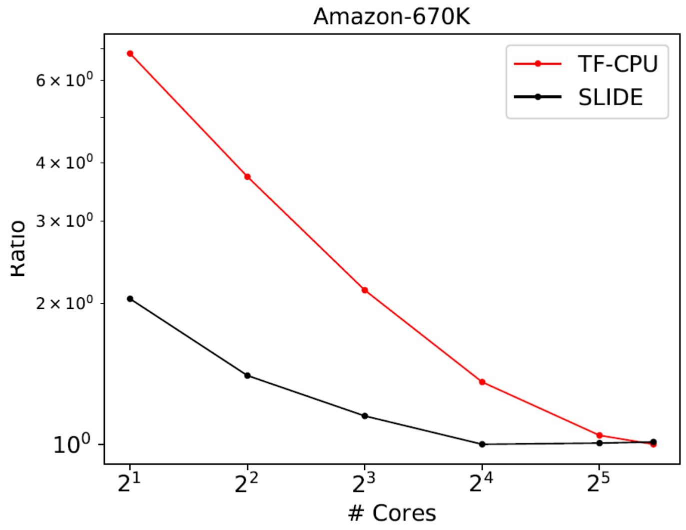


Asynchronous Parallelism gets best scalability

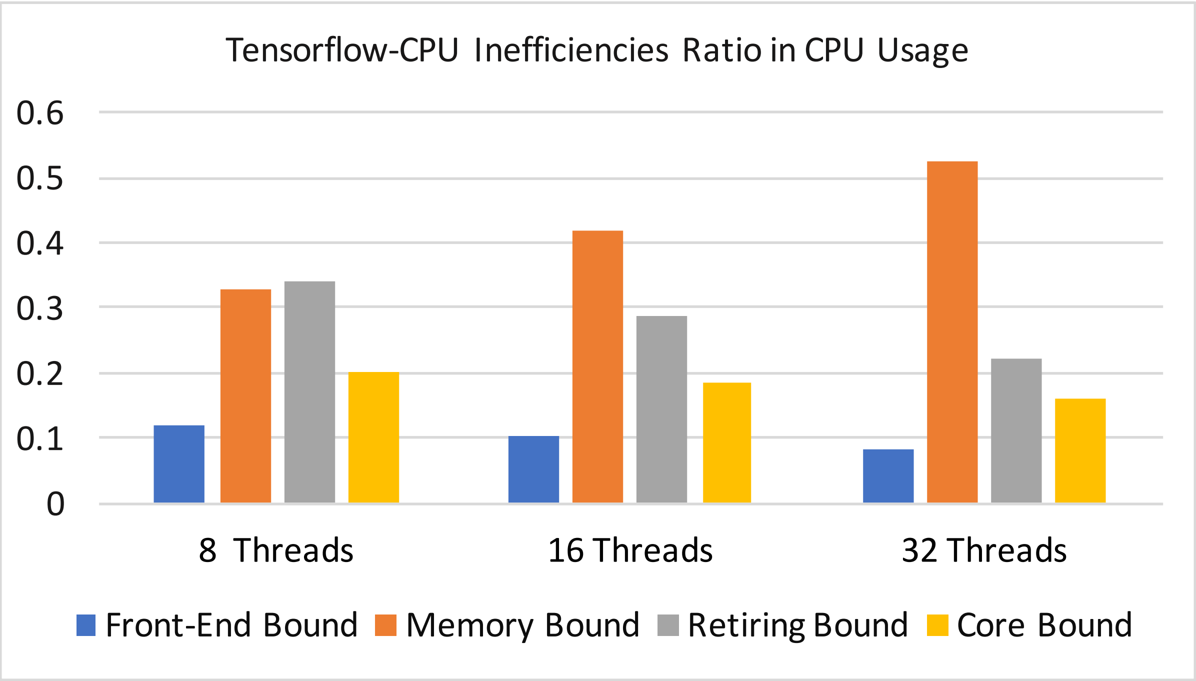
Table: Core Utilization

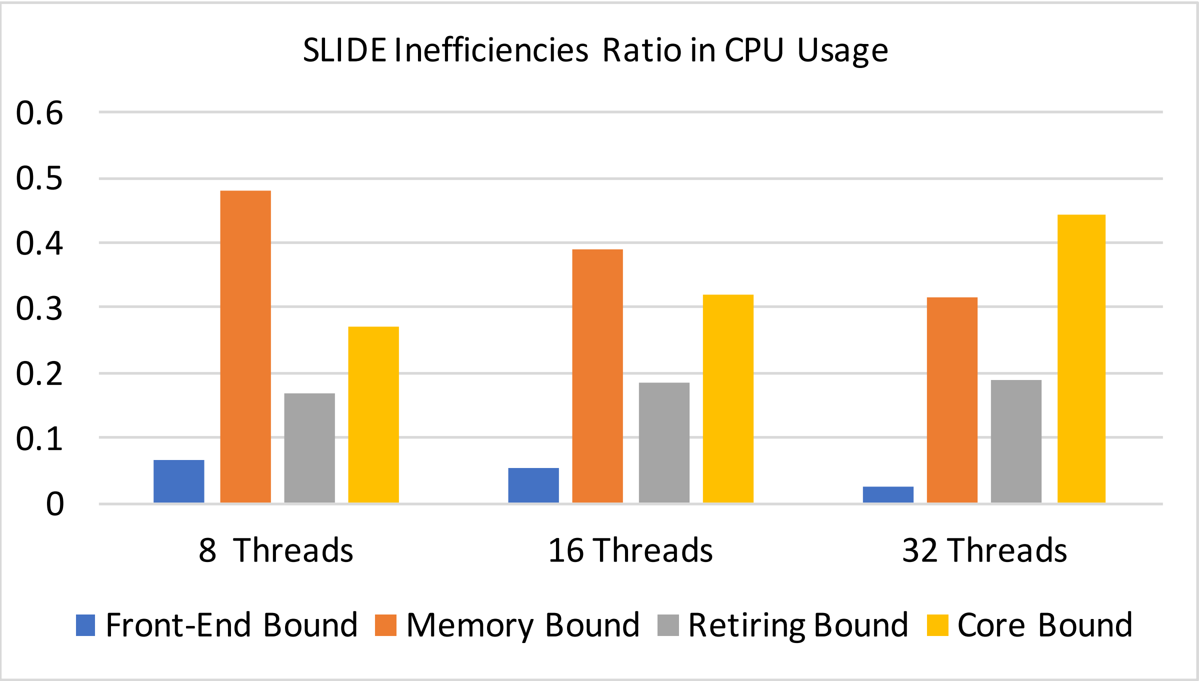
	8	16	32
Tensorflow-CPU	45%	35%	32%
SLIDE	82%	81%	85%





Inefficiency Diagnosis





Impact of HugePages

Metric	Without Hugepages	With Hugepages
dTLB load miss rate	5.12%	0.25%
iTLB load miss rate	56.12%	20.96%
PTW dTLB-miss	7.74%	0.72%
PTW iTLB-miss	0.02%	0.015%
RAM read dTLB-miss	$3,062,039/s$	$749,485/s$
RAM read iTLB-miss	$12,060/s$	$11,580/s$
PageFault	$32,548/s$	$26,527/s$

Conclusion: From Matrix Multiplication to (few) Hash Lookups

- Standard
  - **Operation**
    - Matrix Multiply
  - **Pros**
    - Hardware Support
  - **Cons**
    - Expensive O(N^3)
    - Can only scale with hardware.
    - Energy
- SLIDE
  - **Operations**
    - Compute Random Hashes of Data
    - Hash lookups, Sample and Update. (Decades of work in Databases)
    - Very Few Multiplication (100x+ reduction)
  - **Pros**
    - Energy (IoT), Latency
    - Asynchronous Parallel Gradient updates
    - Simple Hash Tables
    - Larger Network → More Savings
  - **Cons**
    - Random Memory Access (**but parallel SGD**)

Future Work

- Distributed SLIDE
- SLIDE on more complex architectures like CNN/RNN

RICE

References

[1] Beidi Chen, Tharun Medini, Anshumali Shrivastava “SLIDE : In Defense of Smart Algorithms over Hardware Acceleration for Large-Scale Deep Learning Systems”. Proceedings of the 3rd MLSys Conference (2020).

[2] Ryan Spring, Anshumali Shrivastava. "Scalable and sustainable deep learning via randomized hashing". Proceedings of the 23rd ACM SIGKDD (2017).

[3] Makhzani, A. and Frey, B. J. “Winner-take-all autoencoders”. In Advances in neural information processing systems (2015).

[4] Beid Chen, Anshumali Shrivastava. “Densified Winner Take All (WTA) Hashing for Sparse Datasets”. In Uncertainty in artificial intelligence (2018).

[5] Beidi Chen, Yingchen Xu, and Anshumali Shrivastava. "LGD: Fast and Accurate Stochastic Gradient Estimation”. In Neurips, Dec. 2019. Vancouver.

[6] Benjamin Recht, Christopher Re, Stephen Wright, and Feng Niu. “Hogwild: A lock-free approach to parallelizing stochastic gradient descent”. In Advances in neural information processing systems (2011).

Thanks!!!
Welcome to stop by Poster #7

PAPER LINK

CODE LINK