

Hashing Meets Statistical Estimation and Inference: Adaptive Sampling at the cost of random sampling.



Anshumali Shrivastava
with Ryan, Beidi, Chen, Tharun, Ben (Ph.D.),
Yiqiu, Jonathan, (Masters)
Frankie, Scarlett (Undergrads).

anshumali@rice.edu

IIT Bombay

28th June 2019

Motivating Problem: Stochastic Gradient Descent

$$\theta^* = \arg \min_{\theta} F(\theta) = \arg \min_{\theta} \frac{1}{N} \sum_{i=1}^N f(x_i, \theta) \quad (1)$$

Standard GD

$$\theta_t = \theta_{t-1} - \eta^t \frac{1}{N} \sum_{i=1}^N \nabla f(x_j, \theta_{t-1}) \quad (2)$$

SGD, pick a random x_j , and

$$\theta_t = \theta_{t-1} - \eta^t \nabla f(x_j, \theta_{t-1}) \quad (3)$$

SGD Preferred over GD in Large-Scale Optimization.

- Slow Convergence per epoch.
- Faster Epoch, $O(N)$ times and hence overall faster convergence.

Better SGD?

Why SGD Works? (It is Unbiased Estimator)

$$\mathbb{E}(\nabla f(x_j, \theta_{t-1})) = \frac{1}{N} \sum_{i=1}^N \nabla f(x_i, \theta_{t-1}). \quad (4)$$

Are there better estimators? YES!!

- Pick x_i , with probability proportional to w_i
- Optimal Variance (Alain et. al. 2015): $w_i = \|\nabla f(x_i, \theta_{t-1})\|_2$
- Many works on other Importance Weights (e.g. works by Rachel Ward)

The Chicken-and-Egg Loop

- Maintaining w_i , requires $O(N)$ work.
- For Least Squares, $w_i = \|\nabla f(x_i, \theta_t)\|_2 = |2(\theta_t \cdot x_i - y_i)| \|x_i\|_2$, changes in every iteration.

Better SGD?

Why SGD Works? (It is Unbiased Estimator)

$$\mathbb{E}(\nabla f(x_j, \theta_{t-1})) = \frac{1}{N} \sum_{i=1}^N \nabla f(x_i, \theta_{t-1}). \quad (4)$$

Are there better estimators? YES!!

- Pick x_i , with probability proportional to w_i
- Optimal Variance (Alain et. al. 2015): $w_i = \|\nabla f(x_i, \theta_{t-1})\|_2$
- Many works on other Importance Weights (e.g. works by Rachel Ward)

The Chicken-and-Egg Loop

- Maintaining w_i , requires $O(N)$ work.
- For Least Squares, $w_i = \|\nabla f(x_i, \theta_t)\|_2 = |2(\theta_t \cdot x_i - y_i)| \|x_i\|_2$, changes in every iteration.

Can we Break this Chicken-and-Egg Loop? Can we get adaptive sampling in constant time $O(1)$ per Iterations, similar to cost of

Outline of the Talk

- 1 Algorithmic Perspective of Probabilistic Hashing
 - Fast Near Neighbor Search (Classical LSH algorithm)
- 2 Hashing as Efficient Adaptive Sampling.
 - A New Efficient Class of Samplers and Unbiased Estimators.
 - Speeding up Stochastic Gradient Estimation.
 - Scalable and Sustainable Deep Learning via Hashing
- 3 What more?
 - Sub-Linear Memory Anomaly Detection and Near-Neighbors
- 4 Other Relevant Projects

Textbook Hashing (Dictionary)

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

Textbook Hashing (Dictionary)

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

Property (Ideal Hash Functions):

- if $x = y$ then $h(x) = h(y)$
- if $x \neq y$ then $h(x) \neq h(y)$

Textbook Hashing (Dictionary)

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

Property (Ideal Hash Functions):

- if $x = y$ then $h(x) = h(y)$
- if $x \neq y$ then $h(x) \neq h(y)$

Think of Java HashMaps (dictionary).

- **Problem:** Given an array of n integers. Remove duplicates.
- Naive Solution: $O(n^2)$, with sorting $O(n \log n)$
- With HashMaps (or Dictionary): $O(n)$

Probabilistic Fingerprinting (Hashing)

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

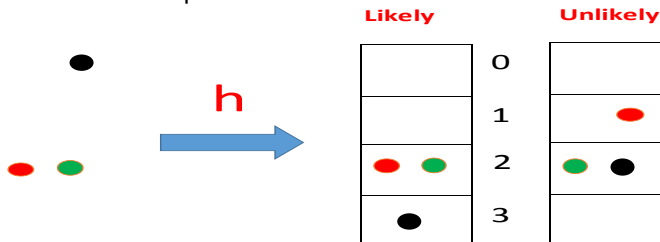
Probabilistic Fingerprinting (Hashing)

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

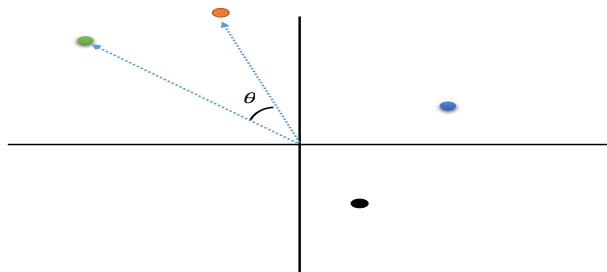
Locality Sensitive Property:

- if $x \approx y$ $\text{Sim}(x,y)$ is high then $h(x) \approx h(y)$ $Pr(h(x) = h(y))$ is high.
- if $x \neq y$ $\text{Sim}(x,y)$ is low then $h(x) \neq h(y)$ $Pr(h(x) = h(y))$ is low.

Similar points are more likely to have the same hash value (hash collision) compared to dissimilar points.



Popular Hashing Scheme: SimHash (SRP)

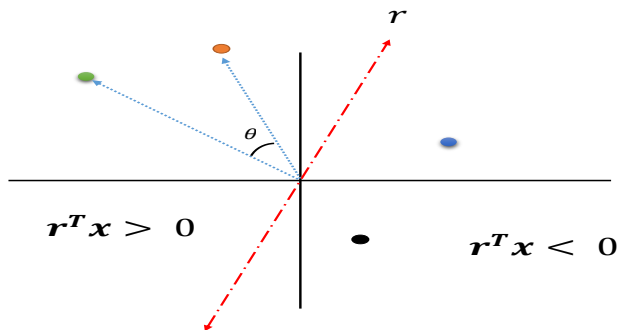


$$h_r(x) = \begin{cases} 1 & \text{if } r^T x \geq 0 \\ 0 & \text{otherwise} \end{cases} \quad r \in \mathbb{R}^D \sim N(0, \mathcal{I})$$

$$Pr_r(h_r(x) = h_r(y)) = 1 - \frac{1}{\pi} \cos^{-1}(\theta), \quad \text{monotonic in } \theta \quad (\text{Cosine Similarity})$$

A classical result from Goemans-Williamson (95)

Popular Hashing Scheme: SimHash (SRP)



$$h_r(x) = \begin{cases} 1 & \text{if } r^T x \geq 0 \\ 0 & \text{otherwise} \end{cases} \quad r \in \mathbb{R}^D \sim N(0, \mathcal{I})$$

$$Pr_r(h_r(x) = h_r(y)) = 1 - \frac{1}{\pi} \cos^{-1}(\theta), \quad \text{monotonic in } \theta \quad (\text{Cosine Similarity})$$

A classical result from Goemans-Williamson (95)

Some Popular Measures that are Hashable

Many Popular Measures.

- Jaccard Similarity (MinHash)
- Cosine Similarity (Simhash and also MinHash if Data is Binary)
- Euclidian Distance
- Earth Mover Distance, etc.

Recently, Un-normalized Inner Products¹

- 1 With bounded norm assumption.
- 2 Allowing Asymmetry.

¹SL [NIPS 14 (Best Paper), UAI 15, WWW 15], APRS [PODS 16].

Sub-linear Near-Neighbor Search

Given a query $q \in \mathbb{R}^D$ and a **giant** collection \mathcal{C} of N vectors in \mathbb{R}^D , search for $p \in \mathcal{C}$ s.t.,

$$p = \arg \max_{x \in \mathcal{C}} \text{sim}(q, x)$$

- Worst case $O(N)$ for any query. N is huge.
- Querying is a very frequent operation.

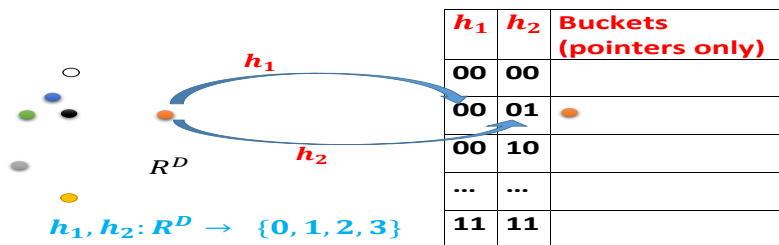
Our goal is to find sub-linear query time algorithm.

Probabilities Hash Tables

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

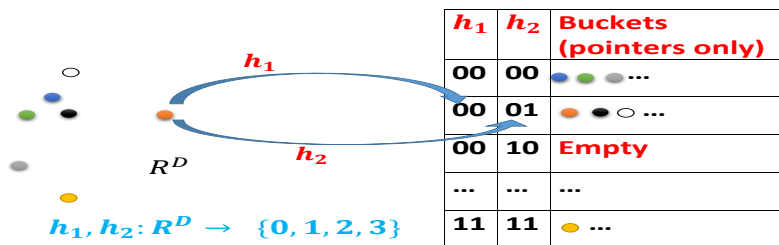
Probabilities Hash Tables

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



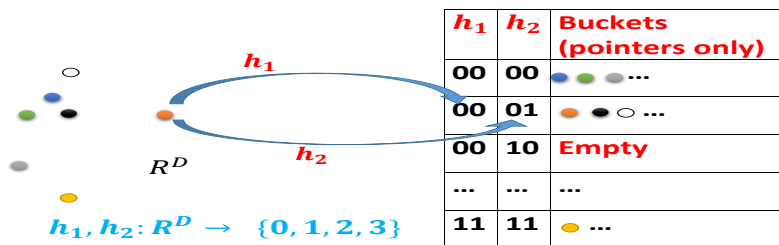
Probabilities Hash Tables

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



Probabilities Hash Tables

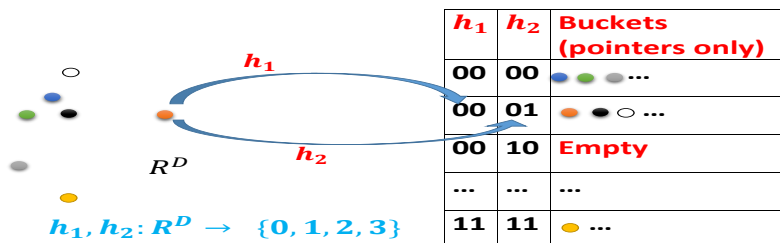
Given: $Pr_h[h(x) = h(y)] = f(sim(x, y))$, f is monotonic.



- Given query q , if $h_1(q) = 11$ **and** $h_2(q) = 01$, then probe bucket with index **1101**. **It is a good bucket !!**

Probabilities Hash Tables

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



- Given query q , 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 !!

The Classical LSH Algorithm





Table 1

h_1^1	...	h_K^1	Buckets
00	...	00	● ● ...
00	...	01	● ○ ...
00	...	10	Empty
...
11	...	11	...

- We use K concatenation.

The Classical LSH Algorithm

Table 1

h_1^1	...	h_K^1	Buckets
00	...	00	  ...
00	...	01	  ...
00	...	10	Empty
...
11	...	11	...

...





Table L

h_1^L	...	h_K^L	Buckets
00	...	00	  ...
00	...	01	  ...
00	...	10	   ...
...
11	...	11	Empty

- We use K concatenation.
- Repeat the process L times. (L Independent Hash Tables)






The Classical LSH Algorithm

Table 1

h_1^1	...	h_K^1	Buckets
00	...	00	  ...
00	...	01	  ...
00	...	10	Empty
...
11	...	11	...

...





Table L

h_1^L	...	h_K^L	Buckets
00	...	00	  ...
00	...	01	  ...
00	...	10	   ...
...
11	...	11	Empty

- We use K concatenation.
- Repeat the process L times. (L Independent Hash Tables)
- **Querying** : Probe one bucket from each of L tables. Report union.



The Classical LSH Algorithm

Table 1

h_1^1	...	h_K^1	Buckets
00	...	00	  ...
00	...	01	  ...
00	...	10	Empty
...
11	...	11	...

...

Table L

h_1^L	...	h_K^L	Buckets
00	...	00	  ...
00	...	01	  ...
00	...	10	   ...
...
11	...	11	Empty

- We use K concatenation.
 - Repeat the process L times. (L Independent Hash Tables)
 - **Querying** : Probe one bucket from each of L tables. Report union.
- 1 Two knobs K and L to control.
 - 2 **Theory says we have a sweet spot.** Provable sub-linear algorithm. (Indyk & Motwani 98)

Success of LSH

Similarity Search or Related (Reduce n)

- Similarity Search or related.
- Plenty of Applications.

Similarity Estimation and Embedding (Reduce dimensionality d)

- Basically JL (Johnson-Lindenstrauss) or Random Projections does most of the job!!
- Similarity Estimation. (Usually not optimal in Fisher Information Sense)
- Non-Linear SVMs in Learning Linear Time ².

Result: Won 2012 ACM Paris Kanellakis Theory and Practice Award.

²Li et. al. NIPS 2011

Success of LSH

Similarity Search or Related (Reduce n)

- Similarity Search or related.
- Plenty of Applications.

Similarity Estimation and Embedding (Reduce dimensionality d)

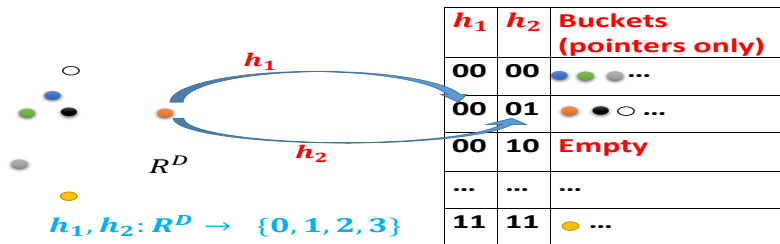
- Basically JL (Johnson-Lindenstrauss) or Random Projections does most of the job!!
- Similarity Estimation. (Usually not optimal in Fisher Information Sense)
- Non-Linear SVMs in Learning Linear Time ².

Result: Won 2012 ACM Paris Kanellakis Theory and Practice Award.

Are there other Fundamental Problems?

²Li et. al. NIPS 2011

A Step Back



Is LSH really a search algorithm?

- Given the query q , LSH samples x from the dataset, with probability exactly $p_y = 1 - (1 - p(q, x)^K)^L$.
- LSH is considered a black box for near-neighbor search.
- Adaptive Sampling is being converted into an algorithm for high similarity search.

New View: Hashing is an Efficient Adaptive Sampling in Disguise.
With Ryan Spring, Beidi Chen, and Scarlett Xu.

Partition Function in Log-Linear Models

$$P(y|x, \theta) = \frac{e^{\theta_y \cdot x}}{Z_\theta}$$

- θ_y is the weight vector
- x is the (current context) feature vector (word2vec).
- $Z_\theta = \sum_{y \in Y} e^{\theta_y \cdot x}$ is the partition function

Issues:

- Z_θ is expensive. $|Y|$ is huge. (billion word2vec)
- Change in context x requires to recompute Z_θ .

Question: Can we reduce the amortized cost of estimating Z_θ ?

Importance Sampling (IS)

Summation by expectation: But sampling $y \propto e^{\theta y \cdot x} = f(y)$ is equally harder.

Importance Sampling

- Given a normalized proposal distribution $g(y)$ where $\sum_y g(y) = 1$.

- We have an unbiased estimator

$$\mathbb{E} \left[\frac{f(y)}{g(y)} \right] = \sum_y g(y) \frac{f(y)}{g(y)} = \sum_y f(y) = Z_\theta$$

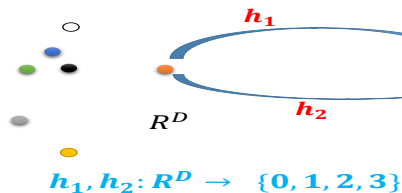
- Draw N samples $y_i \sim g(y)$ for $i = 1 \dots N$. we can estimate

$$Z_\theta = \frac{1}{N} \text{sum}_{i=1}^N \frac{f(y_i)}{g(y_i)}.$$

Chicken and Egg Loop:

- Does not really work if $g(y)$ is not close to $f(y)$.
- Getting $g(y)$ which is efficient and close to $f(y)$ is not known.
- No efficient choice in literature. Random sampling or other heuristics.

Detour: LSH as Samplers

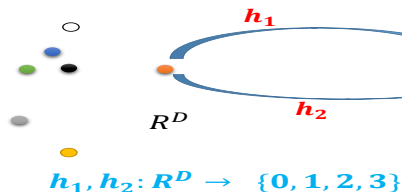


h_1	h_2	Buckets (pointers only)
00	00	● ● ● ...
00	01	● ● ○ ...
00	10	Empty
...
11	11	● ...

(K, L) parameterized LSH algorithm is an efficient sampling:

- Given the query x , LSH samples θ_y from the dataset, with probability exactly $p_y = 1 - (1 - p(x, \theta_y)^K)^L$.

Detour: LSH as Samplers



h_1	h_2	Buckets (pointers only)
00	00	● ● ● ...
00	01	● ● ○ ...
00	10	Empty
...
11	11	● ...

(K, L) parameterized LSH algorithm is an efficient sampling:

- Given the query x , LSH samples θ_y from the dataset, with probability exactly $p_y = 1 - (1 - p(x, \theta_y)^K)^L$.

Not Quite Importance Sampling:

- It is not normalized $\sum_y p_y \neq 1$
- Samples are correlated.

It turns out, we can still make them work!

Beyond IS: The Unbiased LSH Based Estimator

Procedure:

- For context x , report all the retrieved y_i s from the (K, L) parameterized LSH Algorithm. (just one NN query)
- Report $\hat{Z}_\theta = \sum_i \frac{e^{\theta y_i \cdot x}}{1 - (1 - p(x, \theta y_i))^K)^L}$

Properties:

- $E[\hat{Z}_\theta] = Z_\theta$ (**Unbiased**)
-

$$\begin{aligned} \text{Var}[\hat{Z}_\theta] &= \sum_i \frac{f(y_i)^2}{p_i} - \sum_{i=1}^N f(y_i)^2 \\ &\quad + \sum_{i \neq j} \frac{f(y_i)f(y_j)}{p_i p_j} \text{Cov}(\mathbf{1}_{[y_i \in S]} \cdot \mathbf{1}_{[y_j \in S]}) \end{aligned}$$

- Correlations are mostly negative (favorable) with LSH.

MIPS Hashing is Adaptive for Log-Linear Models

Theorem

For any two states y_1 and y_2 :

$$P(y_1|x; \theta) \geq P(y_2|x; \theta) \iff p_1 \geq p_2$$

where

$$p_i = 1 - (1 - p(\theta_{y_i} \cdot x))^K$$

$$P(y|x, \theta) \propto e^{\theta_y \cdot x}$$

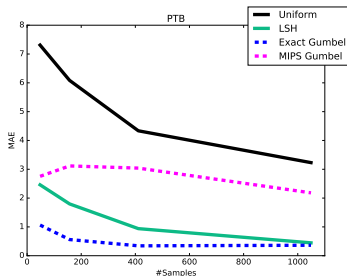
Corollary

The modes of both the sample and the target distributions are identical.

For the first time we break the Chicken-and-Egg-Loop! Sampling time is near-constant.

Efficient as well as similar to target (Adaptive).

How does it work? (PTB and Text8 Datasets)



Running Time:

Samples	Uniform	LSH	Exact Gumbel	MIPS Gumbel
50	0.13	0.23	531.37	260.75
400	0.92	1.66	3,962.25	1,946.22
1500	3.41	6.14	1,4686.73	7,253.44
5000	9.69	17.40	42,034.58	20,668.61

Final Perplexity of Language Models

Standard	LSH	Uniform	Exact Gumbel	MIPS Gumbel
91.8	98.8	524.3	91.9	Diverged
140.7	162.7	1347.5	152.9	

Note the Efficiency of Sampling

Sampling

- Even 1 table will work.
- No Candidate filtering. Random Sampling From Buckets.

Near-Neighbors

- Many tables, Large L
- Bucket aggregation, Candidate Generation, and Candidate Filtering.

Sampling is significantly efficient (1-2 memory lookups). Candidate Filtering is wasteful.

Exact Same Story with Adaptive Sampling for SGD

$$\theta^* = \arg \min_{\theta} F(\theta) = \arg \min_{\theta} \frac{1}{N} \sum_{i=1}^N f(x_i, \theta) \quad (5)$$

Standard GD

$$\theta_t = \theta_{t-1} - \eta^t \frac{1}{N} \sum_{i=1}^N \nabla f(x_j, \theta_{t-1}) \quad (6)$$

SGD, pick a random x_j , and

$$\theta_t = \theta_{t-1} - \eta^t \nabla f(x_j, \theta_{t-1}) \quad (7)$$

SGD Preferred over GD in Large-Scale Optimization.

- Slow Convergence per epoch.
- Faster Epoch, $O(N)$ times and hence overall faster convergence.

Better SGD?

Why SGD Works? (It is Unbiased Estimator)

$$\mathbb{E}(\nabla f(x_j, \theta_{t-1})) = \frac{1}{N} \sum_{i=1}^N \nabla f(x_i, \theta_{t-1}). \quad (8)$$

Are there better estimators? YES!!

- Pick x_i , with probability proportional to w_i
- Optimal Variance (Alain et. al. 2015): $w_i = \|\nabla f(x_i, \theta_{t-1})\|_2$
- Many works on other Importance Weights.

The Chicken-and-Egg Loop

- Maintaining w_i , requires $O(N)$ work.
- For Least Squares, $w_i = \|\nabla f(x_i, \theta_t)\|_2 = |2(\theta_t \cdot x_i - y_i)| \|x_i\|_2$, changes in every iteration.

Better SGD?

Why SGD Works? (It is Unbiased Estimator)

$$\mathbb{E}(\nabla f(x_j, \theta_{t-1})) = \frac{1}{N} \sum_{i=1}^N \nabla f(x_i, \theta_{t-1}). \quad (8)$$

Are there better estimators? YES!!

- Pick x_i , with probability proportional to w_i
- Optimal Variance (Alain et. al. 2015): $w_i = \|\nabla f(x_i, \theta_{t-1})\|_2$
- Many works on other Importance Weights.

The Chicken-and-Egg Loop

- Maintaining w_i , requires $O(N)$ work.
- For Least Squares, $w_i = \|\nabla f(x_i, \theta_t)\|_2 = |2(\theta_t \cdot x_i - y_i)| \|x_i\|_2$, changes in every iteration.

LSH Sampling Breaks this Chicken-and-Egg Loop. We get adaptive sampling in constant time $O(1)$ per Iterations, similar to cost of SGD

LSH Gradient Estimators

One Time Cost

- Preprocess $\langle x_i || x_i ||, y_i || x_i || \rangle$ into Inner Product Hash Tables. (Data Reading Cost)

Per Iteration

- Query hash tables with $\langle \theta_{t-1}, -1 \rangle$ for sample x_i . (1-2 Hash Lookups)
- Estimate Gradient as $\frac{\nabla f(x_i, \theta_{t-1})}{N \times \text{SamplingProbability}}$
- Can show: Unbiased and better variance than SGD.

LSH Gradient Estimators

One Time Cost

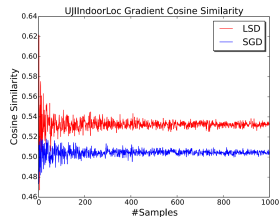
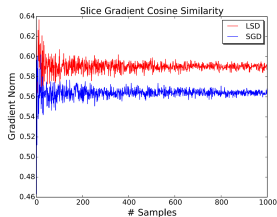
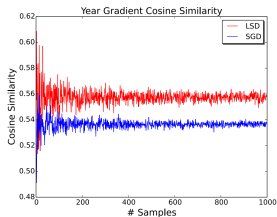
- Preprocess $\langle x_i || x_i ||, y_i || x_i || \rangle$ into Inner Product Hash Tables. (Data Reading Cost)

Per Iteration

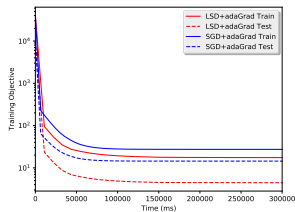
- Query hash tables with $\langle \theta_{t-1}, -1 \rangle$ for sample x_i . (1-2 Hash Lookups)
- Estimate Gradient as $\frac{\nabla f(x_i, \theta_{t-1})}{N \times \text{SamplingProbability}}$
- Can show: Unbiased and better variance than SGD.

Per iterations cost is 1.5 times that of SGD, but superior variance.

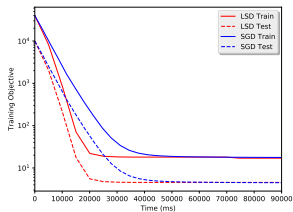
Quality of Samples



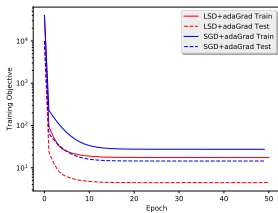
Beating SGD on Wall Clock Time



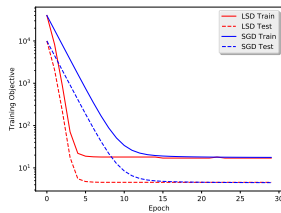
(a) Ada Time



(b) Plain Time



(c) Ada Epoch



(d) Plain Epoch

What Fundamental Problem Did We Solve?

Problem: Given N time evolving weights, $w_1^t, w_2^t, \dots, w_N^t$, we want to sample x_i in proportion to w_i^t at time t .

- $O(N)$ cost every time

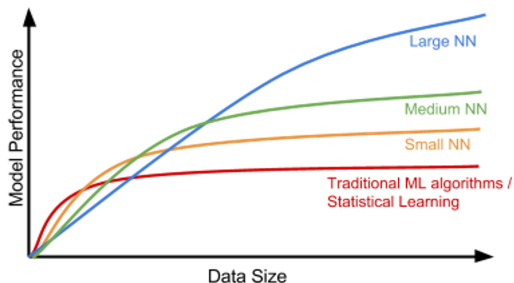
If w_i^t is a specific monotonic function of $\theta_t^T \times c_i$, where θ_t is changing and c_i is fixed, then something significantly efficient than $O(N)$!

Efficient Deep Networks Using Asymmetric Hashing

With Ryan Spring.

- ① *Scalable and Sustainable Deep Learning via Randomized Hashing.*
SIGKDD 2017

The age of Large Networks



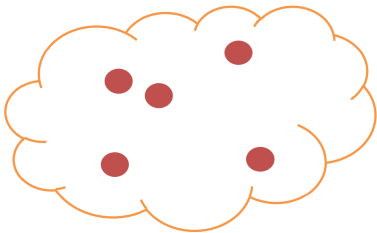
- More Data
- Large Models
- Tons of Engineering
- Backpropagation
(Aka Simple Gradient Descent)

Backpropagation is not Sustainable

Backpropagation with Big-Models and Big-Data.

- **Slow Training and Validation:** Stalls the scientific progress
- **Requires Expensive Clusters:** Not everyone can afford it. Increased dependency on computation services. (only few winners)
- **Memory and Energy:** Out of reach for IoT and other embedded devices.
- **Too Slow for Latency Critical Application:** Hard to do inference in few milliseconds with very large networks.

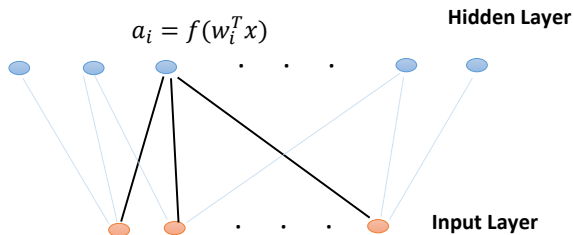
Key Observation: Information Sparsity



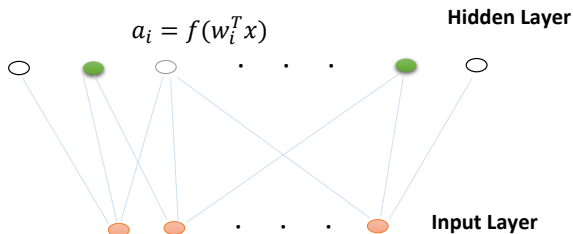
Given any input, out of potentially billions of neurons, it will affect only few neurons. Information in the sparsity pattern.

Note: Different from Parameter Sparsity!!

Deep Networks



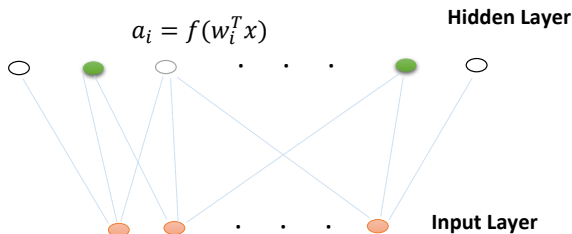
Deep Networks



Adaptive Dropouts³: Sample very few Nodes (neurons) with probability proportional to activations. It works as well as original with extremely sparse updates.

³Ba and Frey NIPS 2014

Deep Networks



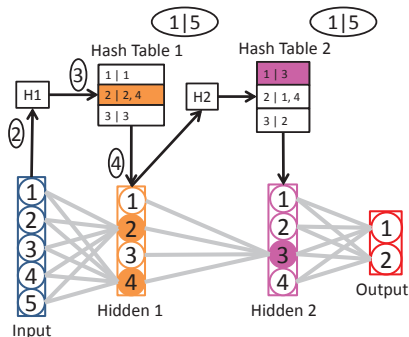
Adaptive Dropouts³: Sample very few Nodes (neurons) with probability proportional to activations. It works as well as original with extremely sparse updates.

- For every data point, compute activations, pick (very few nodes) with high activations (using Bernoulli Sampling).
- Need few nodes but Identifying which ones requires all computations.

Full Cost of Training and Testing

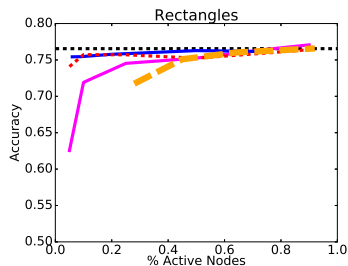
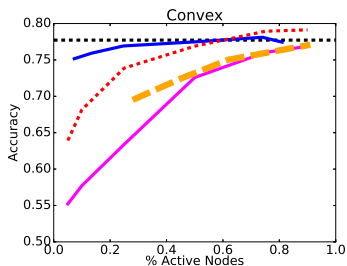
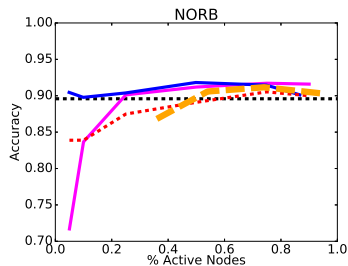
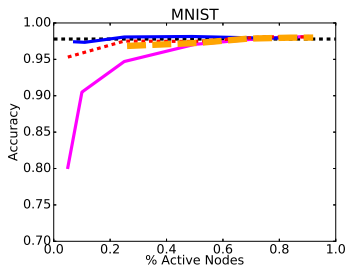
³Ba and Frey NIPS 2014

Hash Lookups for Adaptively Sampling Active Nodes

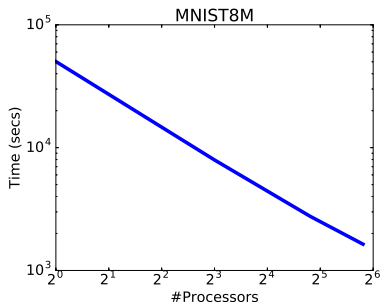
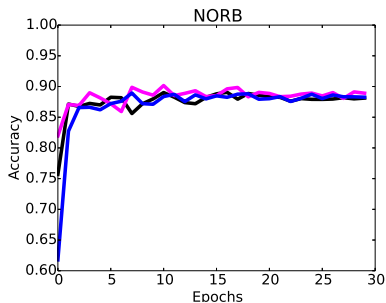
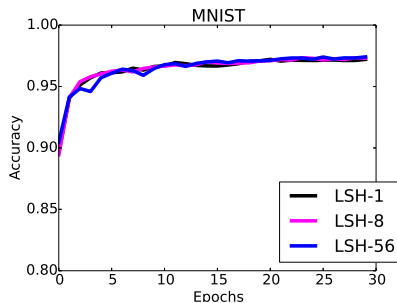


- **Initialization:** Hash all nodes (weights) to LSH tables.
- Output of every layer is query for then next. (Initially the query is x).
- The retrieved nodes serves as active sets. Nodes Sampled with Probability $1 - (1 - p^K)^L$ (**Unusual Distribution**)
- Update active sets and the hash tables. **Significantly less multiplications**

Significantly Less Computations



Bonus: Asynchronous (SGD) for Very Sparse Updates



Summary: Cheaper and Parallel Updates

- Every gradient update is significantly cheaper in terms of arithmetic operation.
- Gradient update parallelizable across data due to sparsity.

Can we beat hardware acceleration (like V100 GPU) with limited multi-core CPU?

Baselines

State-of-the-art optimized implementations

- Tensorflow on Intel(R) Xeon(R) Platinum 8175M CPU (48 cores)
 - Tensorflow-CPU 1.12 from source with GCC5.4 in order to support FMA, AVX, AVX2, SSE4.1, and SSE4.2 instructions.
- Tensorflow on NVIDIA Tesla V100 (32GB)

-VS-

SLIDE: Sub-Linear Deep Learning Engine.

The algorithm we just saw over Intel(R) Xeon(R) Platinum 8175M CPU (48 cores)

Metrics

- Full spectrum of accuracy climb with Wall clock time.
- Effect of changes in batch size
- Scalability with Increasing Cores.

Dataset and Architectures

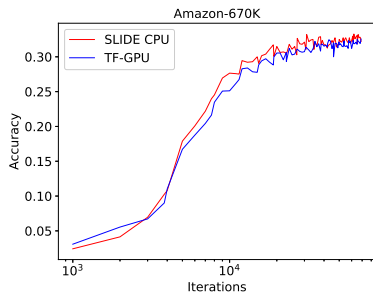
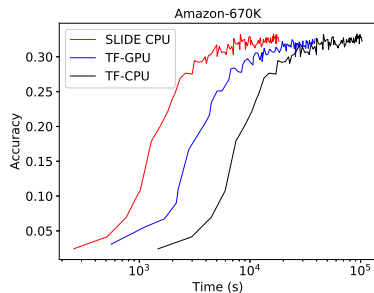
Table: Statistics of the datasets

	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)

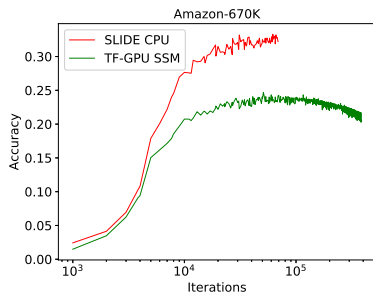
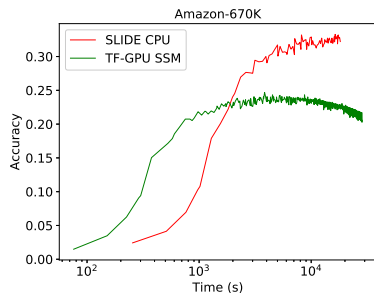
- Delicious-200K $782,585 \Rightarrow 128 \Rightarrow 205,443$ (126 million parameters)
- Amazon-670K $135,909 \Rightarrow 128 \Rightarrow 670,091$ (103 million parameters)

Result: Straight Comparison (2 hours Vs 5.5 hours)



SLIDE on a 44 core CPU is more than 2.7 times (2 hours vs. 5.5 hours) faster than the same network trained using Tensorflow on Tesla V100. Reaches any given accuracy level faster. **Note the log scale on time.**

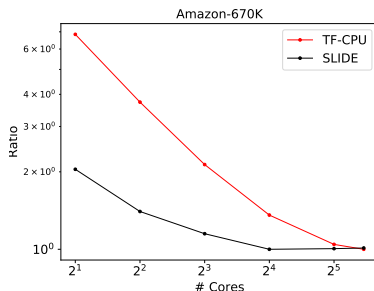
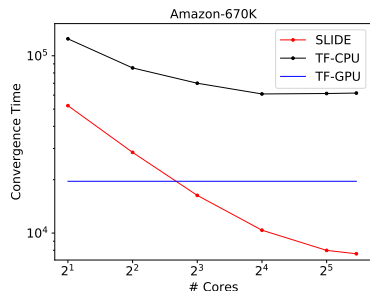
Result 2: Sampled Softmax Tricks are Bad



Result 3: Scalability with Cores

Table: Core Utilization

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



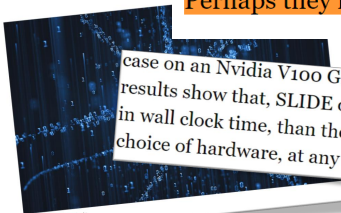
New Results: 100x faster than Tensorflow on CPU system after optimizing for cache thrashing. (Thanks to Intel)



Hash Your Way To a Better Neural Network

Industry has been focused on speeding the many matrix multiplications involved, but a search algorithm may provide better performance

By David Sch



Deep learning breakthrough could slash computation and time by 95 percent

on: June 05, 2017 In: deep learning, Faces of Discovery No Comments

Science
ATAAS

Print Email

Perhaps they're all making a giant mistake.

case on an Nvidia V100 GPU. What they report was pretty stunning: "Our results show that, SLIDE on a modest CPU can be orders of magnitude faster, in wall clock time, than the best possible alternative with the best possible choice of hardware, at any accuracy."

Compressed Estimations

Good old Near-Neighbor but this time sub-linear memory and on streams!

- **Problem Setting:** We have a stream of vectors, x_t at time t .
 - Assume we observe n vectors in total. $D = \{x_1, x_2, \dots, x_N\}$
 - We are not allowed $O(N)$ storage.
 - We are not allowed second pass.
- **Goal:** Given any out of sample query q ($q \notin D$), find the **index** of approximate near-neighbors of q in D .
- **$\ll O(N)$ storage!!**
 - JL lemma gives $O(N \log(N))$
 - We don't know q in advance? So best hope is random sampling? But that loses recall in proportion to the fraction sampled! (Not sub-linear)
- LSH, KD trees, etc. all super-linear in memory.
 - They all requiring storing the data vector or projection of the vector.

Motivating Applications

- Who does not need low communication near-neighbor?
 - Recommendation systems.
 - Compressing Friends graphs for recommending friends.
 - Almost any large scale or memory constrained near-neighbor application.
- Distributed, low memory, IoT
 - Small devices independently sketch what they see. We add the sketch to get the capability to do near-neighbor search!
- Robust Caching
 - Caching is widely deployed in practice for search and related network application.
 - What if caches are robust to perturbation in query?

A Hard Problem in General.

- Information Lower bound (A Negative Result)
 - There cannot exist an algorithm (even randomized) that can solve near-neighbor in stream with $< O(N)$ memory on **generic inputs**
 - Reduction to INDEX problem
- We need a completely different characterization to get sub-linear memory!
 - What kind of inputs can we get sub-linear memory?
 - Are such inputs practical?
 - What does a sub-linear algorithm even look like?

RACE (Repeated ACE Algorithm): A Weird Sketching that Works!

Initialize M (some number) Arrays of size R with zeros.

While Get x_i

 Choose a sparse sample out of M Arrays (universal hashing on i)

 For each sampled array A ,

 compute the associated locality sensitive hash $l(x_i)$ and
 increment the location, i.e. $A[l(x_i)] + +$

DONE!!

These M arrays of counts are the required sketches

- No storing of any attribute of any kind.
- One Pass and mergeable (Just add the arrays)

1	0	2	3	1	2	0
0	12	41	3	0	21	0
2	3	0	10	13	16	44
17	0	11	4	65	44	57
0	9	0	1	0	0	81

Based on i , sample few arrays, let say 2, 3, 6. (Sparse Design matrix.)

x_i

Each array i has LSH h_i associated with it, Compute

$$h_2(x_i) = 0, h_3(x_i) = 3, h_6(x_i) = 3$$

Increment all the counts.

1	0	2	3	1	2	0
0	12	41	3	0	21	0
2	3	0	10	13	16	44
17	0	11	4	65	44	57
0	9	0	1	0	0	81

Based on i , sample few arrays, let say 2, 3, 6. (Sparse Design matrix.)

x_i

Each array i has LSH h_i associated with it, Compute

$$h_2(x_i) = 0, h_3(x_i) = 3, h_6(x_i) = 3$$

Increment all the counts.

1	0	2	3	1	2	0
0	13	41	3	0	21	0
2	3	0	10	13	16	44
17	0	12	4	65	45	57
0	9	0	1	0	0	81

Based on i , sample few arrays, let say 2, 3, 6. (Sparse Design matrix.)

x_i

Each array i has LSH h_i associated with it, Compute

$$h_2(x_i) = 0, h_3(x_i) = 3, h_6(x_i) = 3$$

Increment all the counts.

Proceed to reading x_{i+1}

1	0	2	3	1	2	0
0	12	41	3	0	21	0
2	3	0	10	13	16	44
17	0	11	4	65	44	57
0	9	0	1	0	0	81

q

For array i , get the count of $h(q)$.

Say $h_1(q) = 2, h_2(q) = 4, h_3(q) = 0, \dots, h_7(q) = 0$

Their counts are 2, 9, 2, 10, 65, 16, 0. We take average of these counts over some repetitions.

These average counts are compressed sensing measurements of vector V , whose i^{th} component (V_i) is collision probability of query q , with x_i in database (under LSH). V is a sparse vector, so just recover heavy entries!!

RACE: Querying

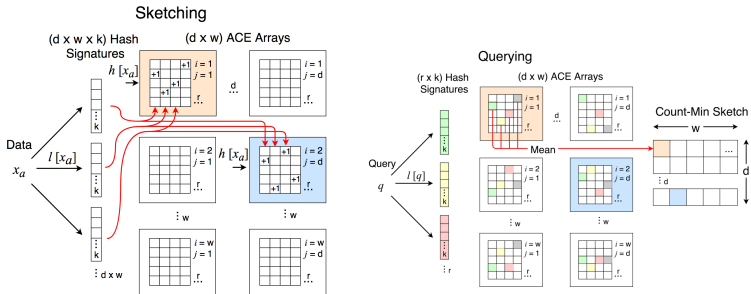
Given a query q .

For each M arrays of counts

Get the value of $A[l(q)]$ which will be a compressed sensing measurement.

Enough arrays give enough measurements.

From these measurements, recover the heavy entries and they are guaranteed to be near-neighbors!!



For any query we can get (estimate sharply) the count-min Sketch of vector V where the i^{th} component of V is given by $V_i = p(q, x_i)^K$, where $p(x_i, q)$ is the collision probability of x_i (in database) with query. K is any parameter we can choose to control sparsity.

Main Theorem

Theorem 1. Main Theorem¹

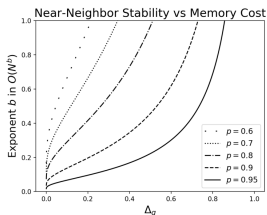
Given a query q , a dataset \mathcal{D} in a metric space $(\mathcal{M}, d(\cdot, \cdot))$, and a $(d(q, x_v), d(q, x_{v+1}), p_1, p_2)$ -sensitive hash function with a finite range r , we can construct a sketch \mathcal{S} that solves the exact v -nearest neighbor problem with probability $1 - \delta$ using

$$O\left((v+1)^3 N^b \log^3\left(\frac{N}{\delta}\right)\right)$$

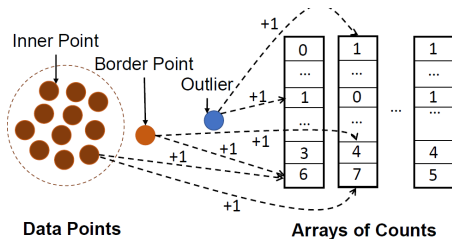
bits, where

$$b = \frac{6|\log p_1| + 2 \log r}{\log \frac{1}{\Delta}}$$

x_v is the v^{th} nearest neighbor of q in \mathcal{D} , x_{v+1} is the $(v+1)^{\text{th}}$ nearest neighbor of q in \mathcal{D} , and $\Delta = \frac{p_2}{p_1} \leq 1$.



Key Hammer: ACE Algorithm (WWW 2018)



- **Online Addition Phase**

- Generate few (hash) arrays of counters. (**No hash tables, no buckets.**)
- **Addition:** only increment the count, (no adding element to buckets).
- The global mean of anomaly score can also be updated online. (later)

ACE: Arrays of Locality Sensitive Counts: Anomaly Detection on the Edge (by Luo and Shrivastava in WWW 2018)

ACE Estimates

- Given a data point q , ACE estimates the following unbiasedly

$$S(q, \mathcal{D}) = \sum_{x_i \in \mathcal{D}} p(q, x_i)^K$$

Parameter (points to K)

Collision Probability (points to $p(q, x_i)$)

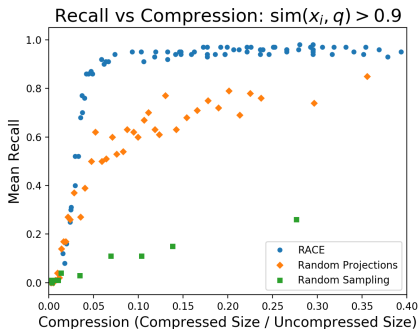
- We modify ACE to instead estimate $\sum r_i \times p(q, x_i)^K$ for any $-1 \leq r_i \leq 1$

Hammer 2: Compressed Sensing (we use CMS sketch)

- We modify ACE to instead estimate
 - $\sum r_i \times p(q, x_i)^K$ for any $-1 \leq r_i \leq 1$
- These are compressed sensing estimate of N dimensional vector (signal) for RIP choices of r_i
- Signal = $[p(q, x_1)^K, p(q, x_2)^K \dots, p(q, x_N)^K]$
- **RACE can estimate these measurements for any query (In Sub-linear ($\ll O(N)$) space!!)**
- We can choose large K to satisfy *sparsity*.
 - Notion of stable near-neighbor (Data Dependent)

New Connection: Hardness of near-neighbor reduces to sparsity of similarity with query and associated hardness in compressed sensing!!

Experimental Results

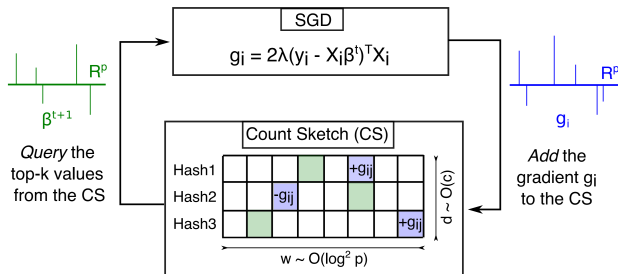


- Google+ graph with around 100k nodes
- Every nodes sparsely represented by direct friends (0-1 vector)
- Compress these vectors.
- **Goal:** Friends recommendation
 - Given a node vector, find identity of nodes with most similar vectors.
- **Metric:** Compression-Accuracy Trade-off

Other Related Projects.

- FLASH System For Search and Sampling. (SIGMOD 18)
- Large Scale Feature Selection with $\log d$ working memory. (ICML 18)
- Parallel and distributed Count-Min Sketch avoiding Heaps (NIPS 18)
- SSH (Sketch, Shingle, & Hash) for Indexing Massive-Scale Time Series. (JMLR 2017)
- Bayesian Estimation with Hashing. (AAAI 2019)
- Sub-Linear Memory Search (Compressed Sensing Meets LSH) (arxiv)

Quick Peek: Sketching Optimizations



Why sketching?

- All updates linear
- Heavy items dominates the computation

Conclusion

- Perfect Time for Algorithmic Disruption in large scale machine learning.
- Machine Learning techniques were not designed keeping in mind computations, and the future with current algorithms such as backpropagation is hopeless.
- We need to revisit old algorithms with a new perspective.
- Probabilistic hashing seems to be a very promising techniques.