IIT KHARAGPUR

IIT GANDHINAGAR

NPTEL ONLINE
CERTIFICATION COURSES

NPTEL

# Scalable Data Science

## Lecture 14c: Fast LSH + Sparse Random Projection


**Anirban Dasgupta**

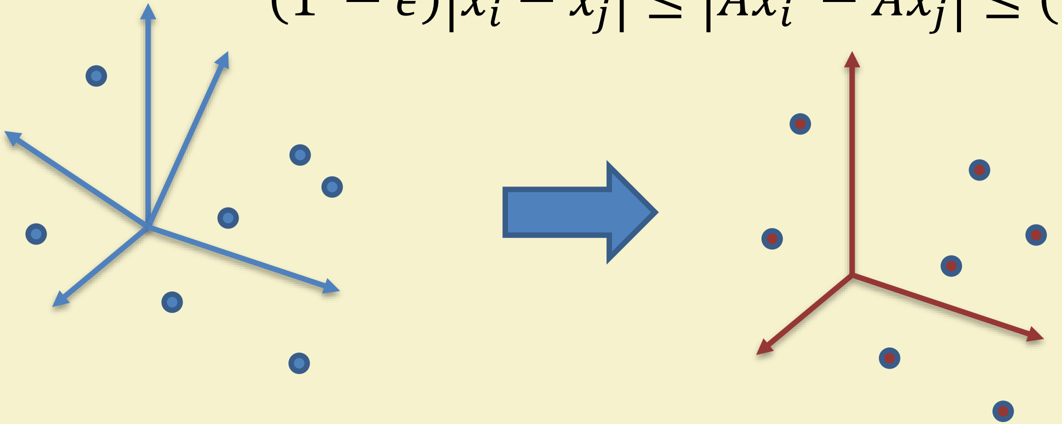
Computer Science and Engineering

IIT GANDHINAGAR

# Johnson Lindenstrauss Lemma [JL84]

$\epsilon > 0$ , $k \geq \frac{C}{\epsilon^2} \log(n)$ . There exists a linear mapping $A$ such that whp, for all $(i, j)$

$$(1 - \epsilon)|x_i - x_j| \leq |Ax_i - Ax_j| \leq (1 + \epsilon)|x_i - x_j|$$



$x_i \in \Re^d$

# Time taken for projection

- Projection matrix out of Gaussian or iid $\pm 1$ = $O(kd)$
  - $k = \Omega\left(\frac{1}{\epsilon^2}\right)$
- FJLT : projection matrix is $\frac{1}{\sqrt{d}}PHD$
  - $H$ is the $d \times d$ Hadamard matrix
  - $D$ is a random $\pm 1$ diagonal matrix
  - $P$ is a sparse Gaussian matrix
  - Time = $O(d \log d + k \log(nd))$

Notice the normalization $\frac{1}{\sqrt{d}}$

# Densification claim

$x \in \Re^d$, $|x|_2 = 1$

<u>Claim:</u> $\max_i |(HDx)_i| \leq O\left(\frac{\log(nd)}{d}\right)^{1/2}$

Application of Chernoff style tail inequality per coordinate and union bound

# Projecting a dense vector

$y = HDx,\ \max_i |y_i| \approx O(\sqrt{\log(nd)\,/d}\,)$

$$P = \begin{cases} 0, w.p.\ 1-q \\ N\left(0, \frac{1}{\sqrt{q}}\right), w.p.\ q \end{cases},\ P \in \Re^{k\times d}, k = O\left(\frac{1}{\epsilon^2}\log\left(\frac{1}{\delta}\right)\right)$$

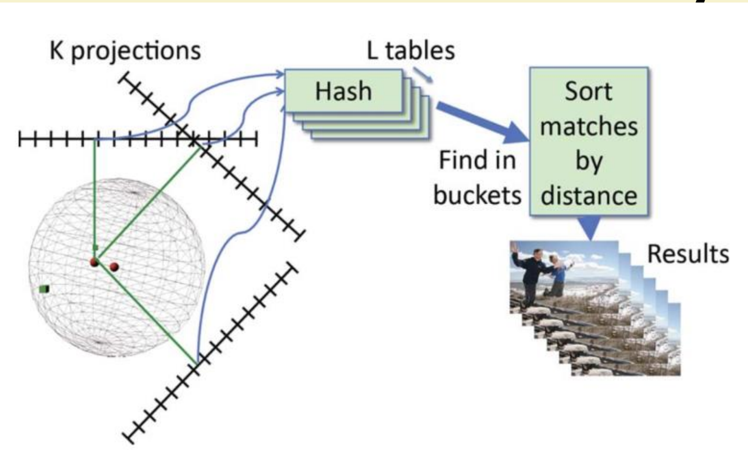
$z = \frac{1}{\sqrt{d}} PHDx$ is the final projected vector

# Fast JL Transform [AC09]

If $q = O(|x|_\infty^2) = O\left(\frac{\log(nd)}{d}\right)$, $\frac{1}{\sqrt{d}} PHD$ satisfies JL property

Calculating $y = PHDx$ takes time $O(d \log d \ + klog(nd))$, potentially much faster than original Gaussian construction

# Locality Sensitive Hashing




Picture courtesy Slaney et al.

Given input data, radius r, approx factor c and confident $\delta$

Output: if there is any point at distance $\leq r$ then w.p. $1-\delta$ return one at distance $\leq cr$

$$h_i(x) = \left\lfloor \frac{x \cdot v_i + b_i}{w} \right\rfloor$$

# Time taken

- Total query time = time to hash + time to check all candidates
  - Calculating k-hash indices takes time $O(kd)$
  - Calculating indices for $L$ buckets takes time $O(kdL)$

- Can we reduce query time?

# Creating hash indices

- Looking at LSH as random projection + quantization
  - $A \in R^{k \times d}$ is a Gaussian JL matrix, $b \in R^k$
  - We first project and then bucketize
  - calculate $\left\lfloor \frac{Ax+b}{w} \right\rfloor$, k-index key calculated at once

- Time taken by matrix-vector multiplication = $O(kd)$ per hash table

# Collision Probability

- $p(u) = \Pr[h_i(q) = h_i(q)]$ when $|p - q| = u$
- $p(u) = \int_0^w \frac{1}{u} f(t/u)\left(1 - \frac{t}{w}\right) dt$ , $f(v) = \text{pdf of } |N(0,1)|$
- This is decreasing with increasing $u$

# ACHash [DKS11]

- When calculating a k-tuple hash bucket index
  - $\left\lfloor \frac{Ax+b}{w} \right\rfloor$ , use $A = PHD$
  - $q \approx O\left(\frac{\log(d)}{d}\right)$
  - Projection time = $O(d\ log\ d + kL \log^2 d\ )$

# ACHash

$p_{AC}(u) =$ probability that a k-tuple hash bucket has same value for two points at distance $u$

We can show that

$$-(k+1)\delta + p^k((1+\epsilon)u) \le p_{\mathrm{AC}}(u) \le p^k((1-\epsilon)u) + (k+1)\delta.$$

i.e. collision prob does not change much

# DHHash

We can make the projection faster

$D$ = random diagonal matrix of $\pm 1$

$G$ = random diagonal matrix, $G_{ii} \sim N(0, 1)$

$M$ = random permutation matrix

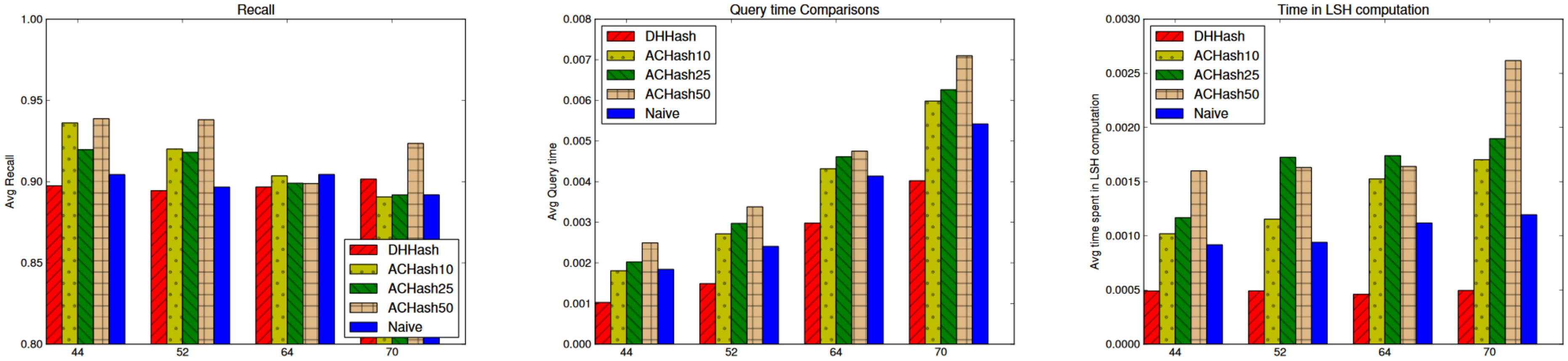
Hash value calculated as $\left\lfloor \frac{HGMHDx + b}{w} \right\rfloor$

# DHHash

- The above creates $d$ bits
- Sample $kL$ indices and create $L$ hash bucket ids, each of size $k$

- Total calculation time for all $L$ bucket-ids = $O(d \log d + kL)$
- Also performs nicely in practice

# Experiments: faster query time with more or less same recall




(d) Recall, LSH query time, and LSH computation time for P53.

# FJLT is fast, but…

- Sparse vectors are prevalent in large scale ML
  - e.g. document representations
- What happens when a sparse vector is
  - multiplied by a dense Gaussian matrix
  - multiplied by $PHD$ of FJLT

# Effect on sparsity

- Sparse vectors are prevalent in large scale ML
  - e.g. document representations
- What happens when a sparse vector is
  - multiplied by a dense Gaussian matrix
  - multiplied by $PHD$ of FJLT
- Both result in dense vector!
  - much more expensive in terms of storage and computation

# Preserving sparsity

- Can we design a linear transformation $A \in \Re^{k \times d}$ such that
  - $|Ax| \approx |x|$ w.h.p. for any fixed $x \in \Re^d$
  - $nnz(x) \approx nnz(Ax)$
- Existing iid constructions do not satisfy this
  - As we saw before, we cannot make the projection matrix very sparse if the elements are chosen independently

# Hashing as projection

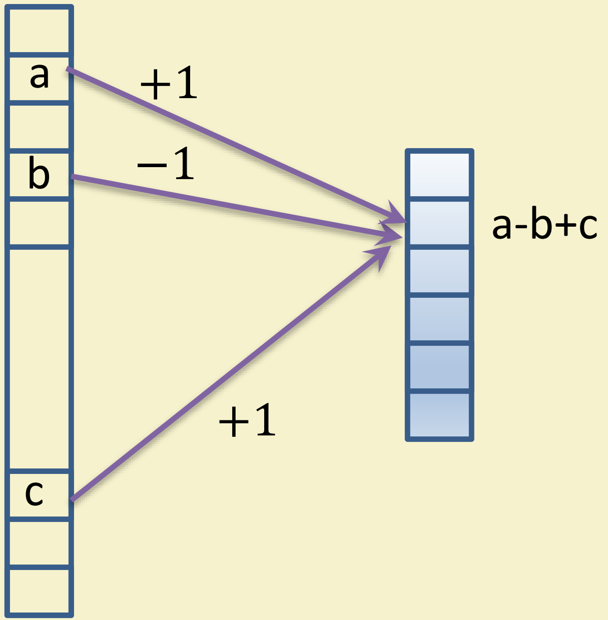
Input $x \in \Re^d$

Target $y \in \Re^k$

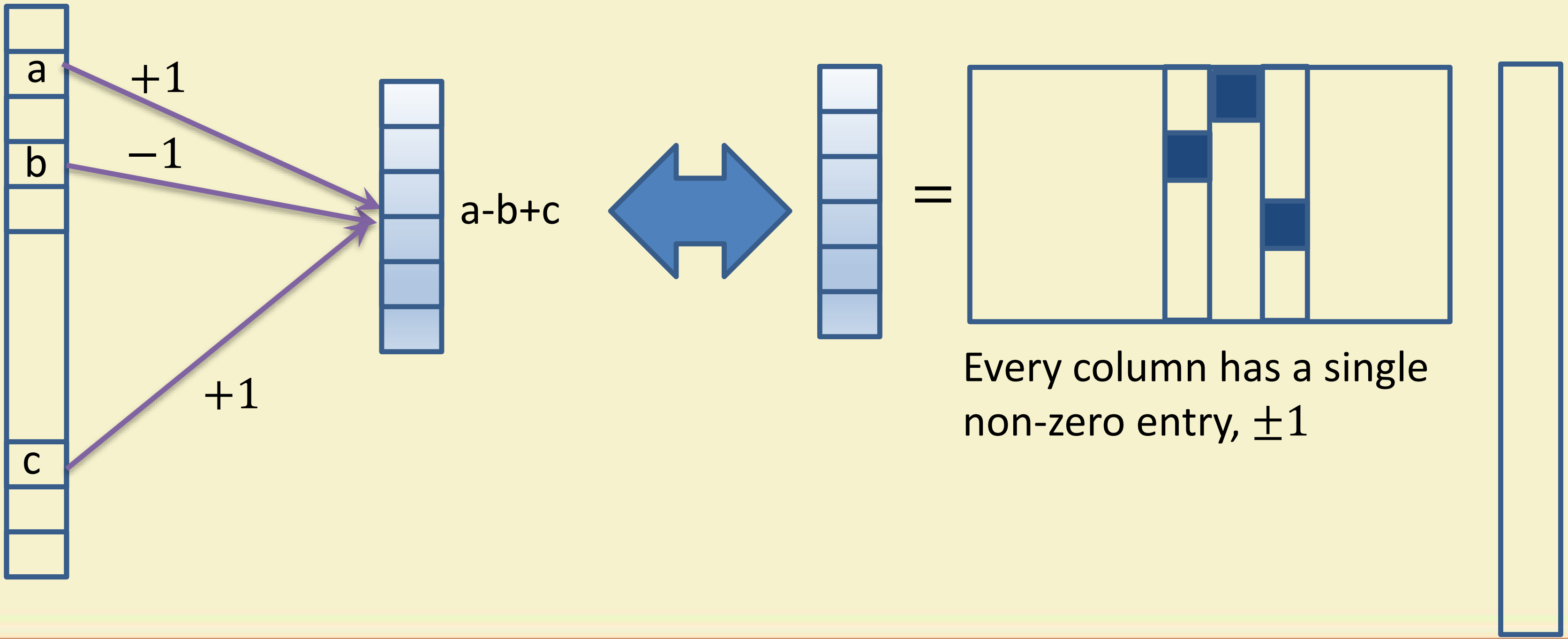
Hash function $h: [d] \rightarrow [k]$

Sign hash: $s: [h] \rightarrow \{-1, +1\}$

$$y[j] = \sum_{i:h(i)=j} s(i)\, x_i$$



# Hashing as projection




Every column has a single non-zero entry, $\pm 1$

# Sparsity

- $nnz(y) = nnz(x)$
- Norm preservation whp does not happen with only one hash function
  - Repeat the construction

# Sparse random projection matrix

For every column, choose a fixed number, $\ell$, positions

For each position chosen, fill up with uar $\pm 1$ random variable

# Formalization

$A \in \Re^{k \times d}$

[DKS10] Choose $\ell$ positions from each column <u>with replacement</u>

[KN11] Choose $\ell$ positions <u>without replacement</u>

For each nonzero position $A_{ij} = \begin{cases} +1 \; w.p. \frac{1}{2} \\ -1 \; w.p. \frac{1}{2} \end{cases}$

# Guarantee

<u>Claim</u>: For $\ell = \tilde{O}\left(\frac{1}{\epsilon}\right)$, $k = O\left(\frac{1}{\epsilon^2}\log\left(\frac{1}{\delta}\right)\right)$,

$$\Pr[(1-\epsilon) \leq |Ax| \leq (1+\epsilon)] \geq 1-\delta$$

So a vector that initially has $nnz(x)$ nonzeros, now will have at most $\frac{nnz(x)}{\epsilon}$ non-zeros

# References:

- Primary references for this lecture
  - Fast Johnson Lindenstrauss Transform, Ailon and Chazelle, SIAM J Computing 2009.
  - Sparse Johnson Lindenstrauss Transformation, Dasgupta, Kumar, Sarlos, STOC 2010.
  - Fast Locality Sensitive Hashing, Dasgupta, Kumar, Sarlos, KDD 2011.

# Summary

In this lecture, we saw

- An application of FJLT in improving the query time of LSH
- A different construction for random projection that, in addition to preserving length, preserves sparsity better than original

# Thank You!!