Structured Spinners: Approach for fast and large-scale Machine Learning computations

Anne Morvan [1,2]

Joint work with: M. Bojarski[3], A. Choromanska[4], K. Choromanski[5], F. Fagan[6], C. Gouy-Pailler[2], N. Sakr[6], T. Sarlós[4], J. Atif[1]

[1]Université Paris-Dauphine, PSL Research University, CNRS, LAMSADE

[2]CEA, LIST

[3]NVIDIA

[4]NYU

[5]Google Brain Robotics

[6]Columbia University

April 6, 2018

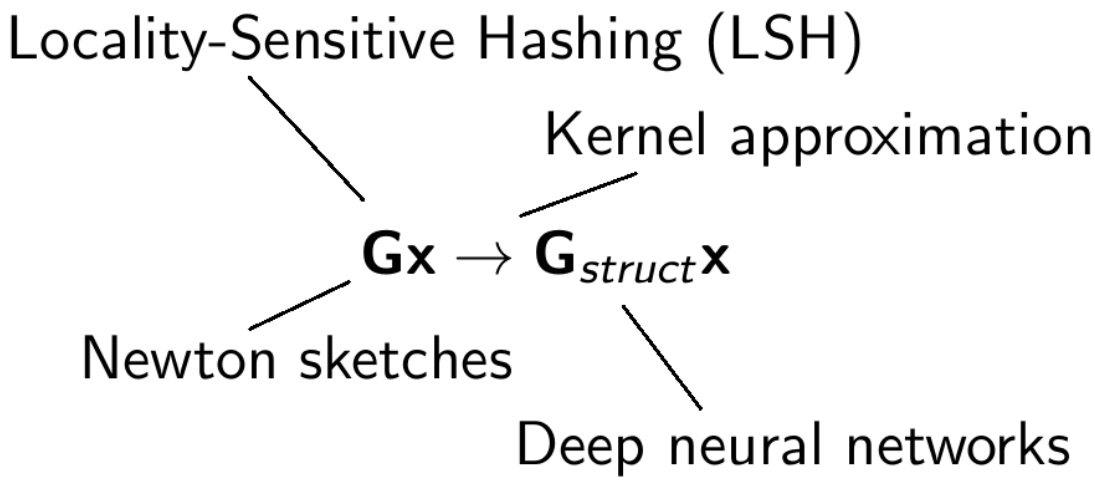
Plan

Plan

Approach for fast and large-scale Machine Learning computations

Structured Spinners' principle

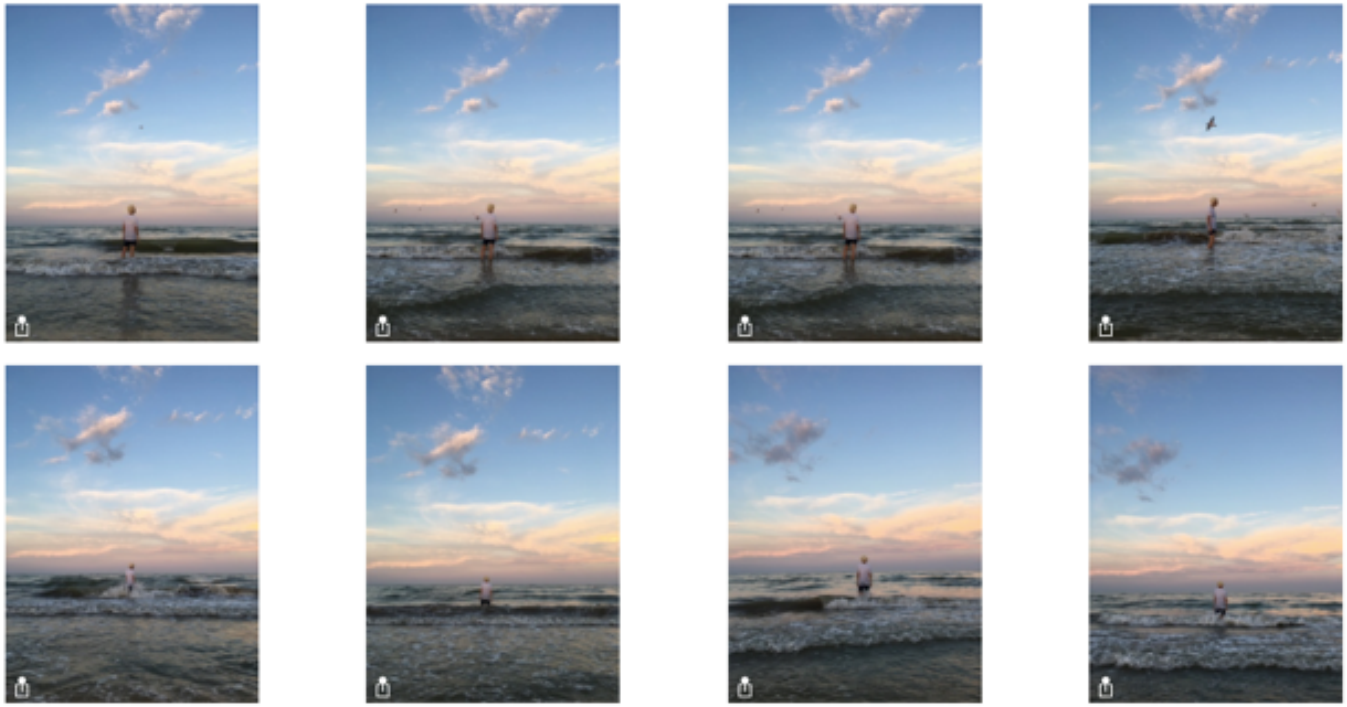
- Replace $\mathbf{Gx}$ by $\mathbf{G}_{struct}\mathbf{x}$ in large-scale ML applications containing random projections, $\mathbf{G}$, $\mathbf{G}_{struct} \in \mathbb{R}^{m \times n}$
- Significant speedups $O(mn) \rightarrow O(n \log m)$
- Memory space savings $O(mn) \rightarrow O(n \log m)$
- Almost no loss of accuracy



Plan

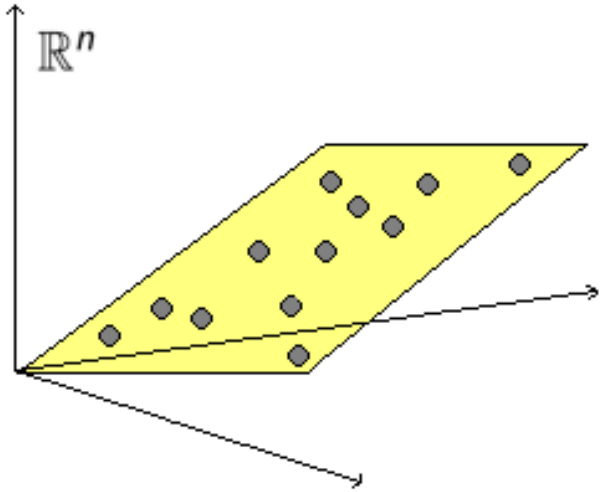
When all your data do not fit into memory...

- Massive data + high dimensionality
- Ex: finding near duplicates in holiday photos



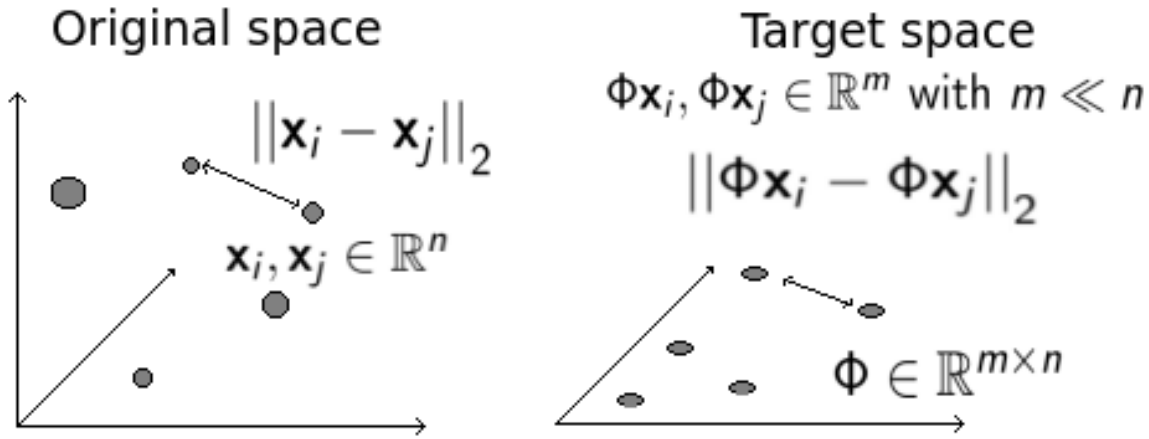
Observation

- Lot of high dimensional data lie on a lower-dimensional manifold
- Concept of intrinsic dimension



- Perform dimensionality reduction / find a suitable projection onto a lower dimensional space

Dimensionality reduction



$$||\Phi\mathbf{x}_i - \Phi\mathbf{x}_j||_2 \approx ||\mathbf{x}_i - \mathbf{x}_j||_2, \; ||\Phi\mathbf{x}_i||_2 \approx ||\mathbf{x}_i||_2$$

Desirable properties of the projection

- Near isometric embedding
- $(1 \pm \epsilon)$ distorsion
- Distance and angle preserved between points

Classical projections

- Principal Component Analysis (PCA)
- Random Projection (RP)

Random projections: theoretical justification

Founder Lemma: [Johnson and Lindenstrauss, 1984]:

Let $\epsilon \in]0,1[$, $\mathcal{X} = \{\mathbf{x}_1, \ldots, \mathbf{x}_N\} \subset \mathbb{R}^n$.

Let $m \in \mathbb{N}$, s.t. $m \geq C\epsilon^{-2} \log N$.

Then there exists a linear map $\Phi : \mathbb{R}^n \to \mathbb{R}^m$ s.t. :

$$\forall \mathbf{x}_i, \mathbf{x}_j \in \mathcal{X}, \ (1-\epsilon)||\mathbf{x}_i - \mathbf{x}_j||_2 \leq ||\Phi\mathbf{x}_i - \Phi\mathbf{x}_j||_2 \leq (1+\epsilon)||\mathbf{x}_i - \mathbf{x}_j||_2.$$

- One can take $\Phi =$ **Random** (near orthonormal) which works with high probability.

Random projections applications

- Linear embedding / Dimensionality reduction,
- Approximate nearest neighbor algorithms, e.g.:
  - Random Projection Trees,
  - Locality Sensitive Hashing-based algorithms.
- Compressed sensing,
- Efficient kernel computations via random feature maps,
- Convex optimization algorithms,
- Quantization techniques,
- etc.

$\Longrightarrow$ information retrieval, similarity search, classification, clustering.

Brief random projections evolution

Φ: Dense i.i.d. distribution

- [Frankl and Maehara, 1987]: $\Phi_{i,j} \sim \mathcal{N}(0, \frac{1}{\sqrt{m}})$
- [Achlioptas, 2003]: $\Phi_{i,j} \sim \{-1, 1\}$ uniformly

Φ: Sparse i.i.d. distribution

- [Kane and Nelson, 2010]: #nonzero entries in $\Phi = O(n \log N/\epsilon)$,
- Fast Johnson-Lindenstrauss Transform - FJLT [Ailon and Chazelle, 2006]: $\Phi = \mathbf{PHD}$
  - $\mathbf{P}_{i,j} = \begin{cases} \sim \mathcal{N}(0, \frac{1}{q}) & \text{with probability } q \\ 0 & \text{with probability } 1-q \end{cases}$,
  - $\mathbf{H}$ normalized Hadamard,
  - $\mathbf{D}$ with independent Rademacher (± 1) entries.

$$\mathbf{H}_0 = 1$$

$$\mathbf{H}_1 = \frac{1}{\sqrt{2}} \begin{pmatrix} 1 & 1 \\ 1 & -1 \end{pmatrix}$$

$$\mathbf{H}_m = \frac{1}{\sqrt{2}} \begin{pmatrix} \mathbf{H}_{m-1} & \mathbf{H}_{m-1} \\ \mathbf{H}_{m-1} & -\mathbf{H}_{m-1} \end{pmatrix}$$

Classical structured matrices

$$\mathbf{H}_0 = 1$$

$$\mathbf{H}_1 = \frac{1}{\sqrt{2}} \begin{pmatrix} 1 & 1 \\ 1 & -1 \end{pmatrix}$$

$$\mathbf{H}_m = \frac{1}{\sqrt{2}} \begin{pmatrix} \mathbf{H}_{m-1} & \mathbf{H}_{m-1} \\ \mathbf{H}_{m-1} & -\mathbf{H}_{m-1} \end{pmatrix}$$

$$\mathbf{G}_{skew-circ} = \begin{pmatrix} \mathbf{a} & -\mathbf{b} & -\mathbf{c} & -\mathbf{d} & -\mathbf{e} \\ e & a & -b & -c & -d \\ d & e & a & -b & -c \\ c & d & e & a & -b \\ b & c & d & e & a \end{pmatrix}$$

$$\mathbf{G}_{circ} = \begin{pmatrix} \mathbf{a} & \mathbf{b} & \mathbf{c} & \mathbf{d} & \mathbf{e} \\ e & a & b & c & d \\ d & e & a & b & c \\ c & d & e & a & b \\ b & c & d & e & a \end{pmatrix}$$

$$\mathbf{G}_{Toeplitz} = \begin{pmatrix} \mathbf{a} & \mathbf{b} & \mathbf{c} & \mathbf{d} & \mathbf{e} \\ \mathbf{f} & a & b & c & d \\ \mathbf{g} & f & a & b & c \\ \mathbf{h} & g & f & a & b \\ \mathbf{j} & h & g & f & a \end{pmatrix}$$

$$\mathbf{G}_{Hankel} = \begin{pmatrix} \mathbf{a} & \mathbf{b} & \mathbf{c} & \mathbf{d} & \mathbf{e} \\ b & c & d & e & \mathbf{f} \\ c & d & e & f & \mathbf{g} \\ d & e & f & g & \mathbf{h} \\ e & f & g & h & \mathbf{i} \end{pmatrix}$$

$$\mathbf{K} = \mathbf{R}_1 \otimes \mathbf{R}_2 \otimes \ldots \otimes \mathbf{R}_m \in \mathbb{R}^{2^m \times 2^m}$$

$$\mathbf{R}_i \in \mathbb{R}^{2\times 2} \text{ or } \mathbf{R}_i \in \{-1, 1\}^{2\times 2}$$

$$\mathbf{R}_i \mathbf{R}_i{}^T = \mathbf{R}_i{}^T \mathbf{R}_i = \mathbf{I}_2$$

And what is about our *Structured Spinners*-family?

Main purpose of *Structured Spinners*-family

Speed up several machine learning algorithms relying on unstructured random matrices with almost no loss of accuracy!

Arguments

- Speedups:
  - Fast Fourier Transform (FFT) or Fast Hadamard Transform (FHT): $O(n \log m)$ instead of $O(mn)$ for matrix-vector product.
- Less storage:
  - $\mathbf{H}$ is not stored
  - Sparse matrices: diagonal ones
  - Structured matrices: $n \times n$-circulant one $\Longrightarrow$ only n parameters (linear)
  - Structured matrices with ± 1 entries: only bits.

Some of state-of-the-art for structured matrices in applications (1/2)

Approximate Nearest Neighbor search (ANN), e.g.:

- [Andoni et al., 2015]: Locality-Sensitive Hashing (LSH), $\mathbf{HD}_3\mathbf{HD}_2\mathbf{HD}_1$.

Quantization, e.g.:

- [Yu et al., 2014]: $\mathbf{G}_{circulant}$

$$\mathbf{G}_{circulant} = \begin{pmatrix} \mathbf{a} & \mathbf{b} & \mathbf{c} & \mathbf{d} & \mathbf{e} \\ e & a & b & c & d \\ d & e & a & b & c \\ c & d & e & a & b \\ b & c & d & e & a \end{pmatrix}$$

Some of state-of-the-art for structured matrices in applications (2/2)

Kernel approximation via random feature maps
[Rahimi and Recht, 2007, Rahimi and Recht, 2009]

- [Le et al., 2013]: "FastFood", $\frac{1}{\sqrt{n}}\mathbf{SHGPHB}$,
- [Feng et al., 2015]: $\pm 1\mathbf{G}_{circulant}$,
- [Choromanski and Sindhwani, 2016]: "$\mathcal{P}$-model", and Toeplitz-like semi-Gaussian matrices,

  $\sum_{i=1}^{r} \mathrm{Circ}[\mathbf{g}^i]\,\mathrm{SkewCirc}[\mathbf{h}^i]$ for some $\{\mathbf{g}^i, \mathbf{h}^i\}_{i=1}^{r} \in \mathbb{R}^n$.

Plan

Definition of *Structured Spinners* family

Structured Spinners for 3 blocks

$$\mathbf{G} \rightarrow \mathbf{G}_{struct}$$

$$\mathbf{G}_{struct} = \mathbf{M}_3\mathbf{M}_2\mathbf{M}_1 \in \mathbb{R}^{n \times n},$$

where matrices $\mathbf{M}_1, \mathbf{M}_2$ and $\mathbf{M}_3$ satisfy 3 conditions.

Examples

- $[\mathbf{G}_{circ} \mid \mathbf{G}_{skew-circ} \mid \mathbf{G}_{Toeplitz} \mid \mathbf{G}_{Hankel}]\mathbf{D}_2\mathbf{H}\mathbf{D}_1$,
- $\sqrt{n}\,\mathbf{H}\mathbf{D}_{g_1,\ldots,g_n}\mathbf{H}\mathbf{D}_2\mathbf{H}\mathbf{D}_1$,
- $\sqrt{n}\,\mathbf{H}\mathbf{D}_3\mathbf{H}\mathbf{D}_2\mathbf{H}\mathbf{D}_1$.

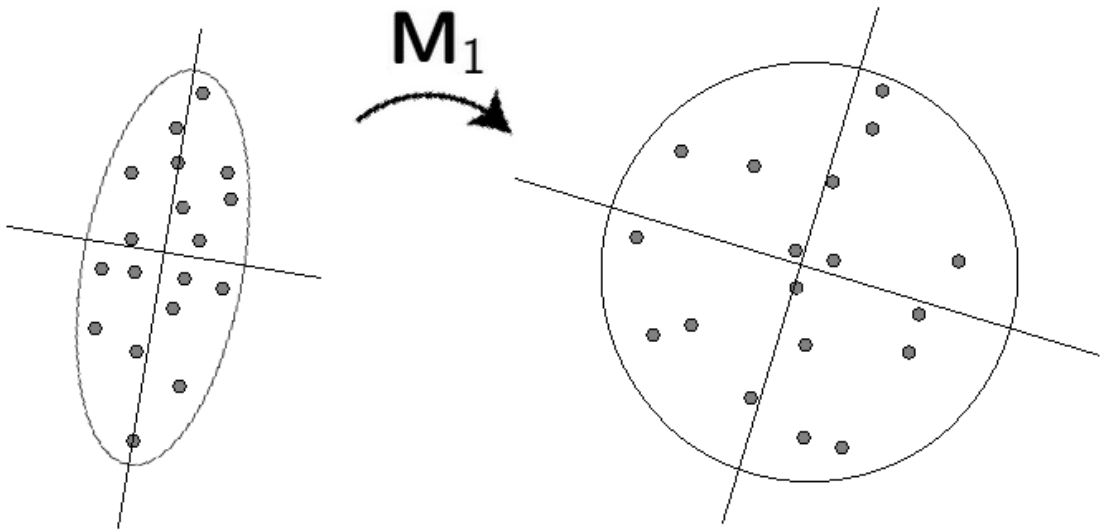
Role of each *Structured Spinner* block

$$\mathbf{G}_{struct} = \mathbf{M}_3\mathbf{M}_2\mathbf{M}_1$$

Role of each *Structured Spinner* block - $\mathbf{M}_1$

$$\mathbf{G}_{struct} = \mathbf{M}_3\mathbf{M}_2\mathbf{M}_1$$

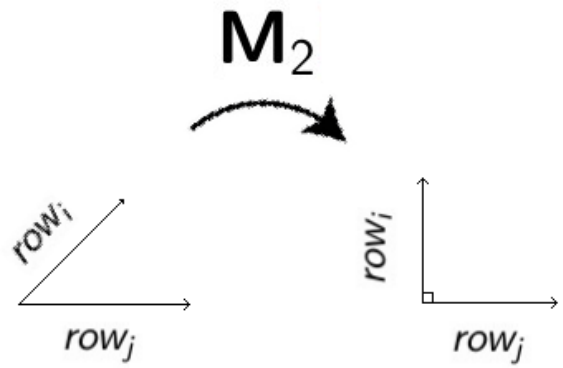
$\hookrightarrow$ Balances data.



Role of each *Structured Spinner* block - $\mathbf{M}_2$

$$\mathbf{G}_{struct} = \mathbf{M}_3\mathbf{M}_2\mathbf{M}_1$$

$\hookrightarrow$ Makes the rows of the final matrix almost independent.



Role of each *Structured Spinner* block - $\mathbf{M}_3$

$$\mathbf{G}_{struct} = \mathbf{M}_3\mathbf{M}_2\mathbf{M}_1$$

$\hookrightarrow$ Budget of randomness.

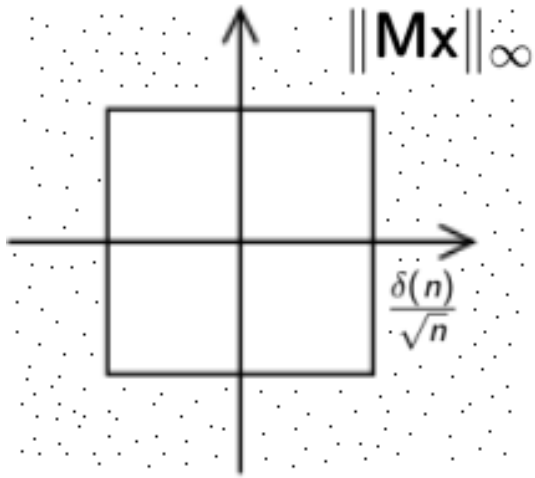
Condition 1 - Balanceness

Condition 1: $\mathbf{M}_1$ and $\mathbf{M}_2\mathbf{M}_1$ are $(\delta(n), p(n))$-balanced isometries.

Definition: $(\delta(n), p(n))$-balanced matrices

A randomized matrix $\mathbf{M} \in \mathbb{R}^{n \times n}$ is $(\delta(n), p(n))$-balanced if it represents an isometry and for every $\mathbf{x} \in \mathbb{R}^n$ with $\|\mathbf{x}\|_2 = 1$ we have:

$$\mathbb{P}[\|\mathbf{M}\mathbf{x}\|_\infty > \frac{\delta(n)}{\sqrt{n}}] \leq p(n).$$



Example

$\mathbf{M}_1 = \mathbf{H}\mathbf{D}_1$, since $\mathbf{H}\mathbf{D}_1$ is $(\log(n), 2ne^{-\frac{\log^2(n)}{8}})$-balanced.

Condition 2 - Decorrelation (1/2)

Condition 2: $\mathbf{M}_2 = \mathbf{V}(\mathbf{W}^1, ..., \mathbf{W}^n)\mathbf{D}_{\rho_1,...,\rho_n}$ for some (Λ_F, Λ_2)-smooth set $\mathbf{W}^1, ..., \mathbf{W}^n \in \mathbb{R}^{k \times n}$ and some i.i.d sub-Gaussian random variables $\rho_1, ..., \rho_n$ with sub-Gaussian norm K.

$$\mathbf{V}(\mathbf{W}^1, ..., \mathbf{W}^n) = \begin{pmatrix} \mathbf{W}^1 \\ \mathbf{W}^2 \\ ... \\ \mathbf{W}^n \end{pmatrix} \qquad \mathbf{D}_{\rho_1,...,\rho_n} = \begin{pmatrix} \rho_1 & 0 & \dots & 0 \\ 0 & \rho_2 & \ddots & \vdots \\ \vdots & \ddots & \ddots & \vdots \\ 0 & \dots & 0 & \rho_n \end{pmatrix}$$

Typically, $K = 1$.

Condition 2 - Decorrelation (2/2)

Definition: (Λ_F, Λ_2)-smooth sets

A deterministic set of matrices $\mathbf{W} = \{\mathbf{W}^1, ..., \mathbf{W}^n\}$, where $\mathbf{W}^i = \{w^i_{k,l}\}_{k,l \in \{1,...,n\}}$ is (Λ_F, Λ_2)-smooth if:

- for $i = 1, ..., n$:

$$\mathbf{W}^i = \begin{pmatrix} \vdots & & \vdots & & \vdots \\ \mathbf{W}^i_1 & \dots & \mathbf{W}^i_l & \dots & \mathbf{W}^i_n \\ \vdots & & \vdots & & \vdots \end{pmatrix}$$

$$\|\mathbf{W}^i_1\|_2 = .. = \|\mathbf{W}^i_l\|_2 = .. = \|\mathbf{W}^i_n\|_2$$

- for $i \neq j$ and $l = 1, ..., n$:

$$\mathbf{W}^i = \begin{pmatrix} \dots & \vdots & \dots \\ \dots & \mathbf{W}^i_l & \dots \\ \dots & \vdots & \dots \end{pmatrix} \quad \mathbf{W}^j = \begin{pmatrix} \dots & \vdots & \dots \\ \dots & \mathbf{W}^j_l & \dots \\ \dots & \vdots & \dots \end{pmatrix}$$

$$(\mathbf{W}^i_l)^T \cdot \mathbf{W}^j_l = 0$$

- $\max_{i,j} \|(\mathbf{W}^j)^T \mathbf{W}^i\|_F \leq \Lambda_F$ and $\max_{i,j} \|(\mathbf{W}^j)^T \mathbf{W}^i\|_2 \leq \Lambda_2$.

Condition 3 - Budget of randomness

Condition 3: $\mathbf{M}_3 = \mathbf{C}(\mathbf{r}, n)$ for $\mathbf{r} \in \mathbb{R}^k$, where $\mathbf{r}$ is random Rademacher (± 1 entries) or Gaussian.

$$M_3 = \begin{pmatrix} \mathbf{r}_1 & \dots & \mathbf{r}_k & 0 & \dots & \dots & \dots & \dots & \dots & 0 \\ 0 & \dots & 0 & \mathbf{r}_1 & \dots & \mathbf{r}_k & 0 & \dots & \dots & 0 \\ & & & & \vdots & \vdots & & & & \\ & & & & \vdots & \vdots & & & & \\ 0 & \dots & \dots & \dots & \dots & \dots & 0 & \mathbf{r}_1 & \dots & \mathbf{r}_k \end{pmatrix}$$

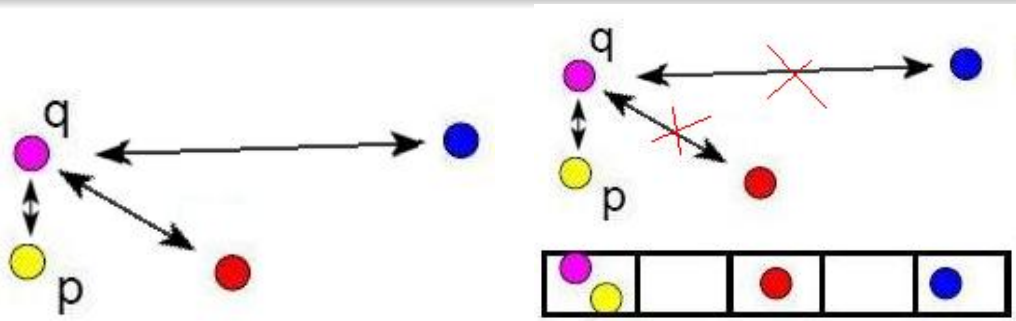
Plan

Plan

Locality-Sensitive Hashing (LSH) for Nearest Neighbor (NN) search

NN search naive approach

- Linear search.
- Prohibitive cost when lots of high dimensional data.
- Solution: Approximate Nearest Neighbor (ANN) search with LSH algorithm in sublinear time.



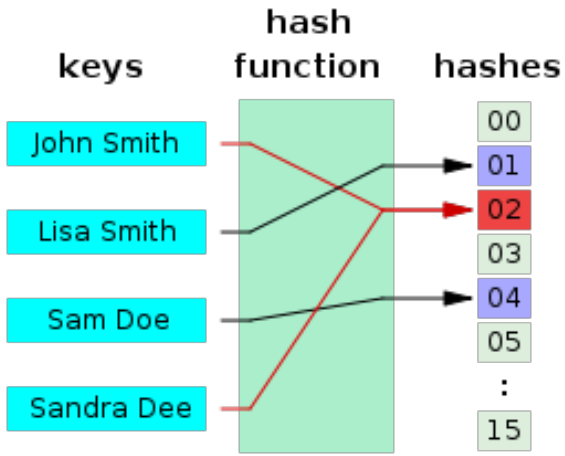
LSH : Two phases

- Build a data structure (hash table) for fast lookup.
- NN search phase: query the database with query point q.

Hashing vs LSH

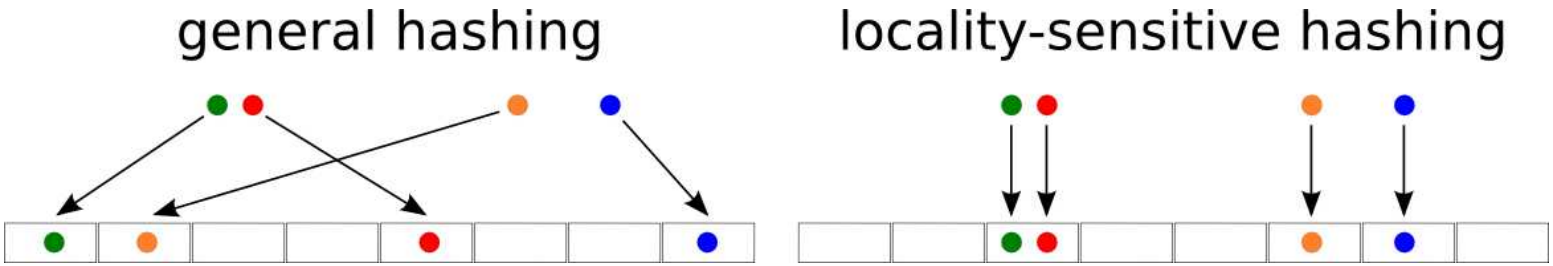
Hashing principle

- Mapping data from a potential high dimensionality to a fixed-size hash value.
- Fast lookup in a database.



LSH principle

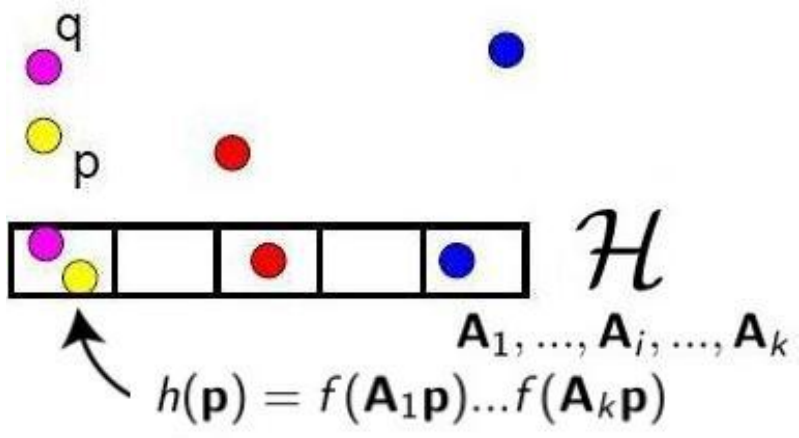
- Exploiting collision probabilities.



LSH in details

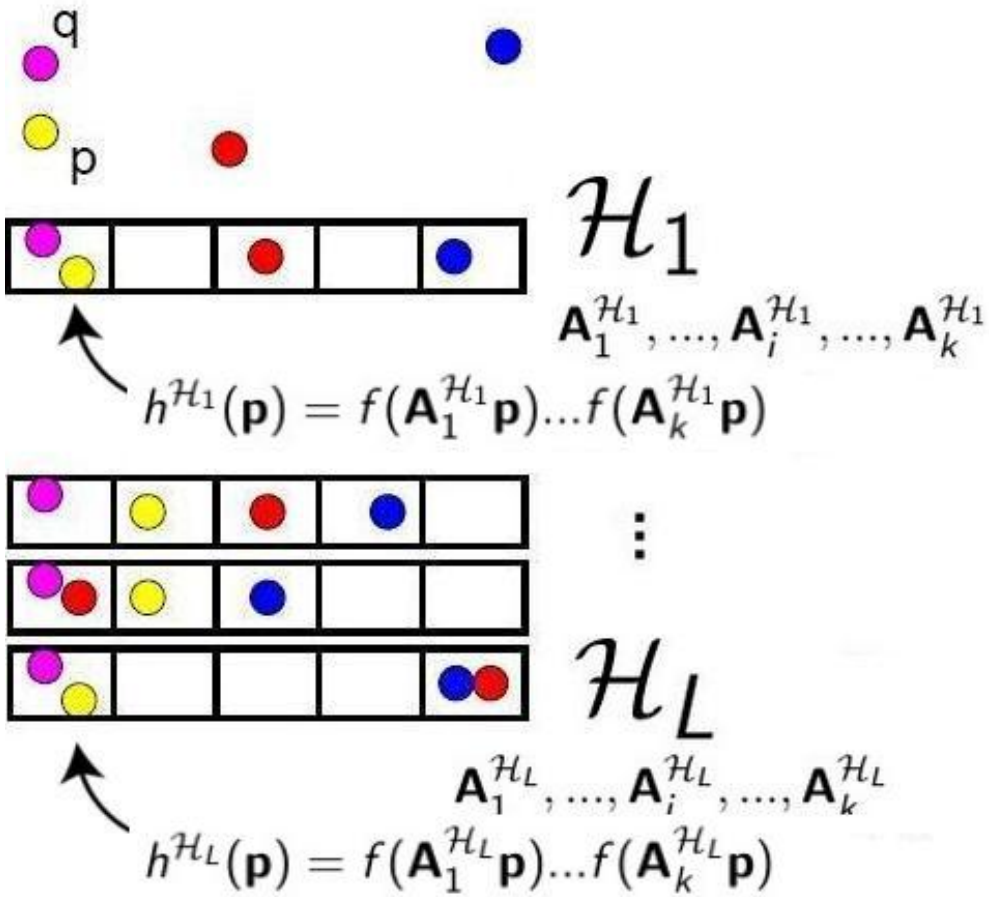
Hash value computation

- Hash value h of a point $\mathbf{x} \in \mathbb{R}^n$ is a combination of k hash function results $h_i, i = 1...k$ s.t. $h_i = f(\mathbf{A}_i\mathbf{x})$ with $\mathbf{A}_i \in \mathbb{R}^{m \times n}$ a projection matrix s.t. $m \ll n$.
- Example: Concatenation: $h = h_1 h_2 ... h_k$.



LSH in details

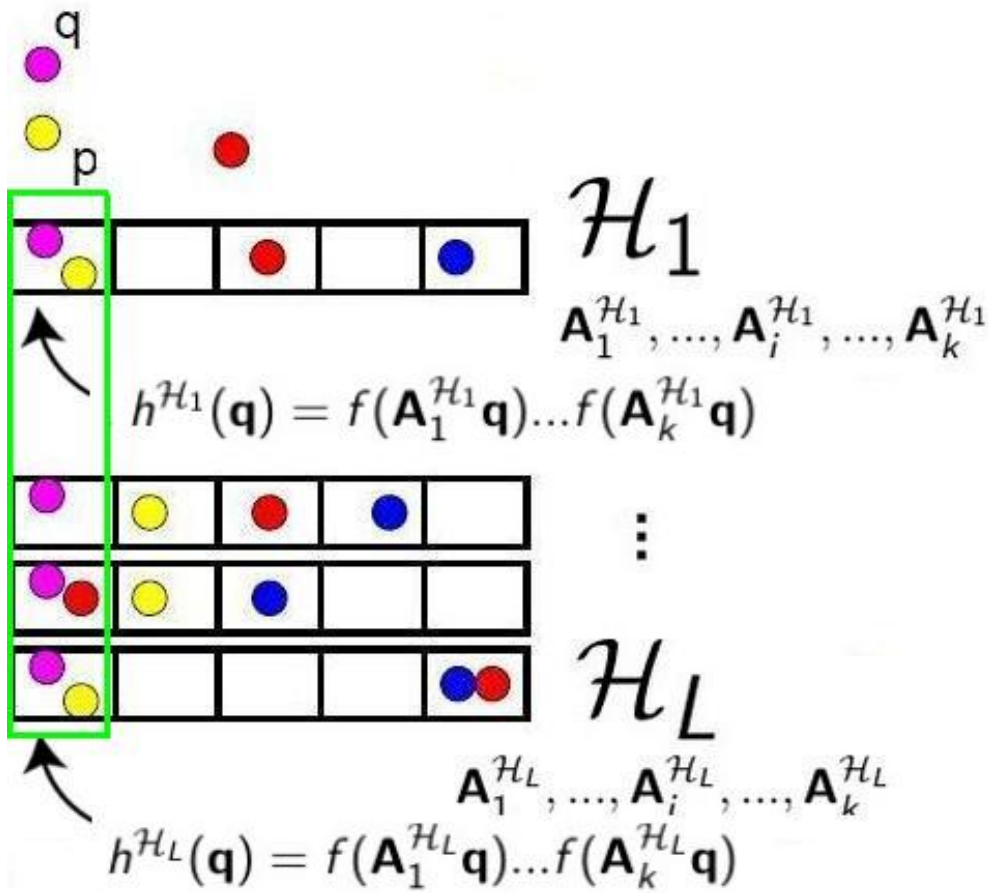
L hash tables



ANN search with LSH

ANN search

- Hash query q.
- Determine pool of candidates (in green).
- Linear scan in the pool of candidates.

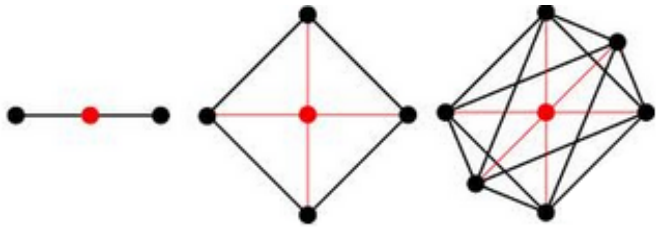


Cross-polytope LSH

Cross-polytope from [Terasawa and Tanaka, 2007]

$$h_i(\mathbf{x}) = f\left(\frac{\mathbf{Gx}}{||\mathbf{Gx}||_2}\right)$$

- $h = (2m)^{k-1}h_1 + ... + h_k$.
- $\mathbf{G} \in \mathbb{R}^{m \times n}$ a random matrix with i.i.d. Gaussian entries.
- $f(\mathbf{y})$ returns the closest vector to $\mathbf{y}$ from the set $\{\pm 1\mathbf{e}_i\}_{1 \leq i \leq m}$, where $\{\mathbf{e}_i\}_{1 \leq i \leq m}$ stands for the canonical basis.



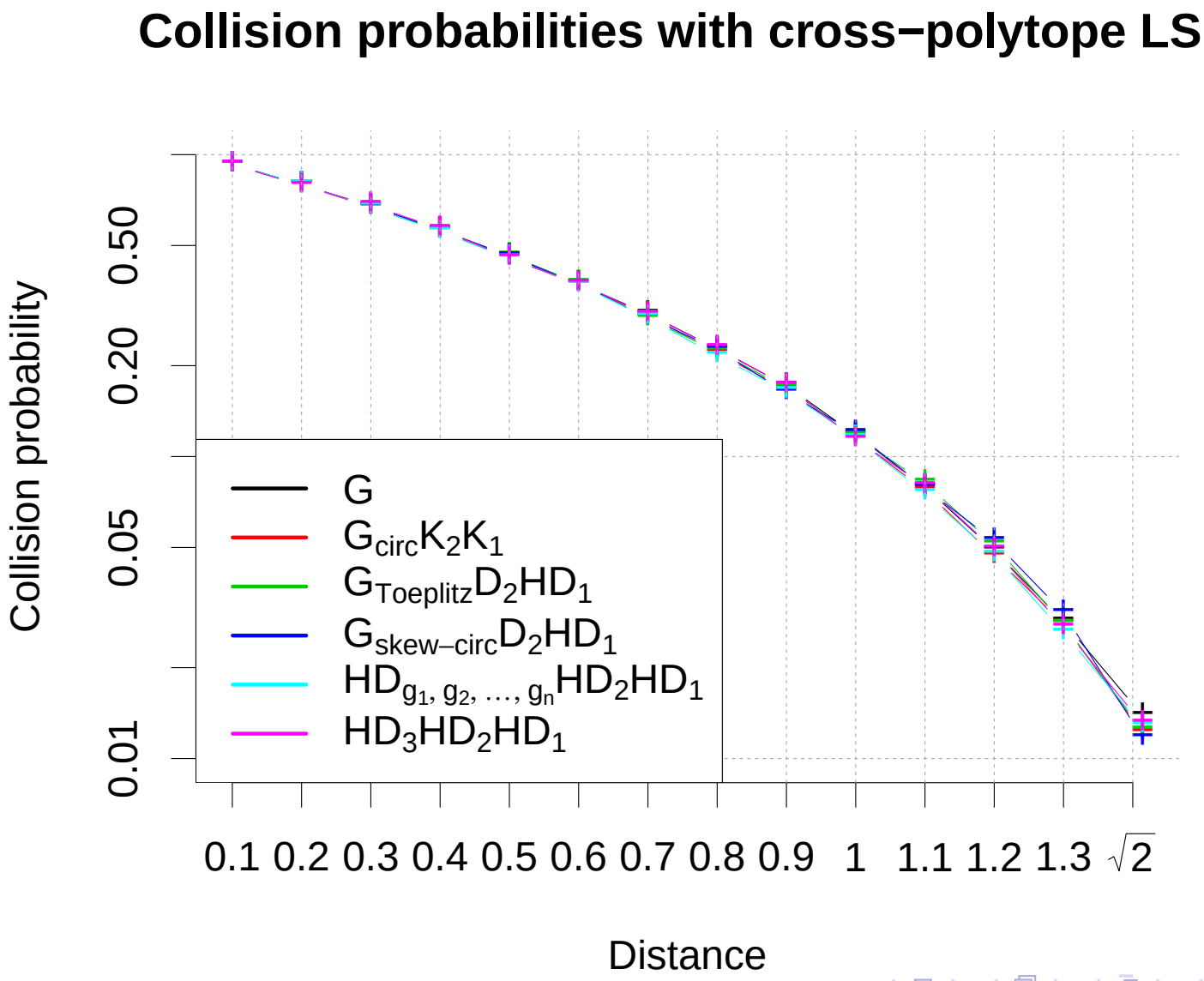
- State-of-the-art cross-polytope LSH [Andoni et al., 2015] $\mathbf{G} \rightarrow \mathbf{HD}_3\mathbf{HD}_2\mathbf{HD}_1$.
- Our variant: $\mathbf{G}_{struct} = \mathbf{M}_3\mathbf{M}_2\mathbf{M}_1$ + theoretical guarantees.

Cross-polytope LSH experiment with *Structured Spinners*

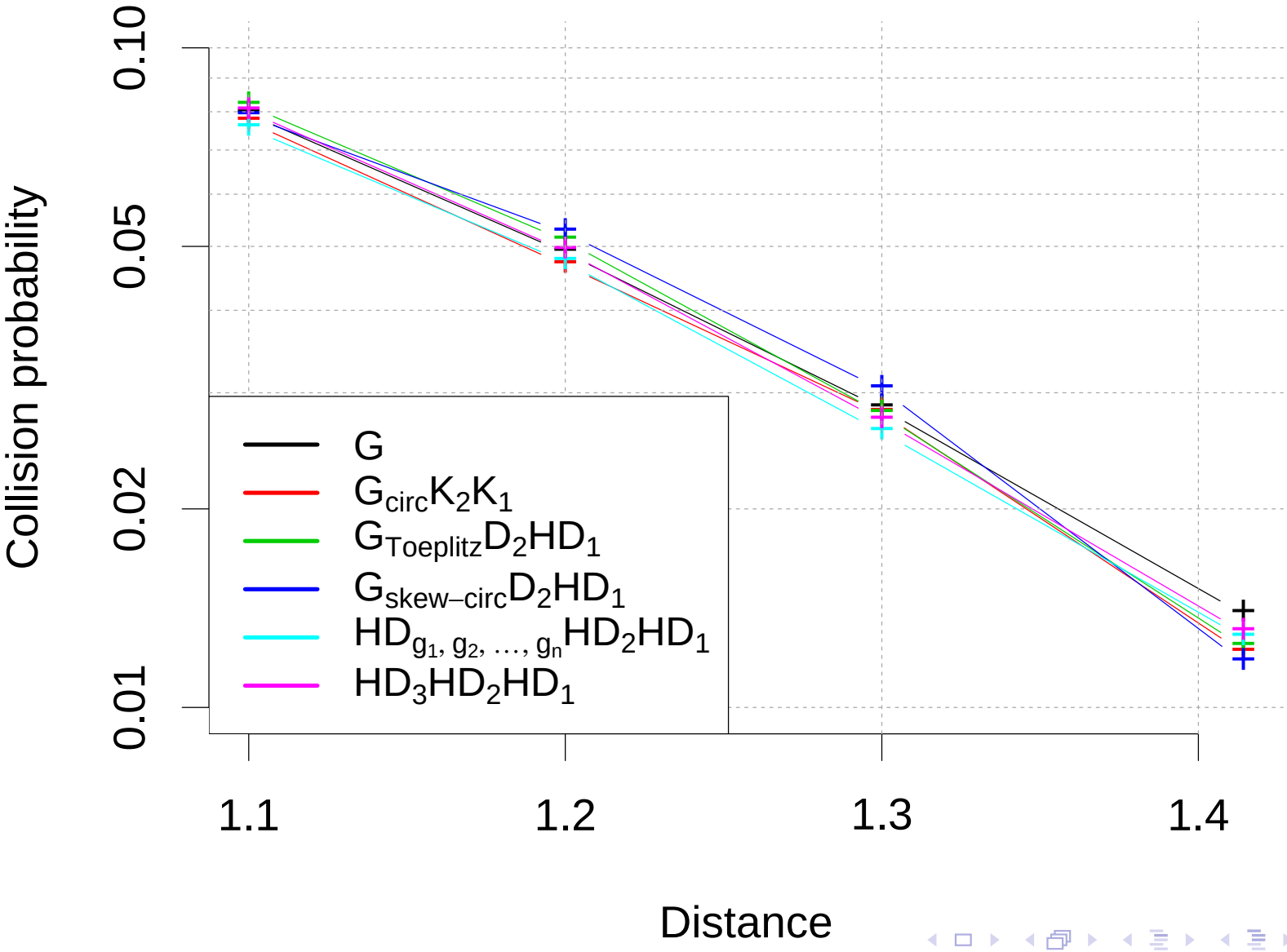
Experimental protocol

- Plot $Pr[h(p) = h(q)]$ as a function of $dist(p, q)$,
- 100 runs,
- $k = 1$,
- Draw points from the hypersphere $\implies \max_{p,q} dist(p, q) = \sqrt{2}$,
- 20000 points per interval of distance:
  $[0, 0.2), [0.2, 0.4), [0.4, 0.6), [0.6, 0.8), [0.8, 1.2), [1.2, \sqrt{2}]$,
- $n = 256$,
- $m = 64$.

Cross-polytope LSH experiment with *Structured Spinners*



Cross-polytope LSH experiment with *Structured Spinners*



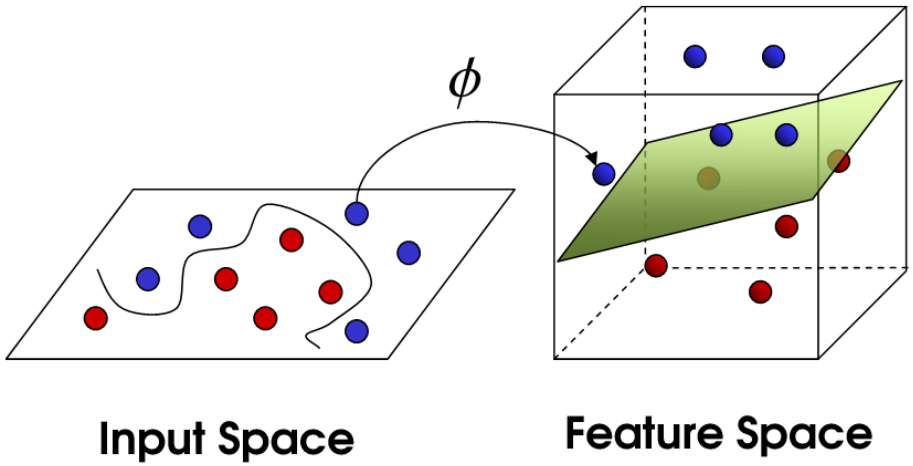
Plan

Kernel methods

Principle

- Goal: To solve nonlinear problems with linear methods.
- How? Map all data into a higher dimensional (possibly infinite) dot product space ν with feature map $\phi : \chi \to \nu$.
- Access to mapped data:

$$\kappa(\mathbf{x}, \mathbf{y}) = \langle \phi(\mathbf{x}), \phi(\mathbf{y}) \rangle$$



Kernel approximation in Support Vector Machines (SVM)

Decision evaluation in SVM: the "kernel trick"

$$f(x) = \langle \mathbf{w},\ \phi(\mathbf{x}) \rangle = \left\langle \sum_{i=1}^{N'} \alpha_i\ \phi(\mathbf{x}_i),\ \phi(\mathbf{x}) \right\rangle = \sum_{i=1}^{N'} \alpha_i\ \kappa(\mathbf{x}_i, \mathbf{x})$$

N' : number of nonzero α_i = number of "support vectors"

Why approximation ?

- Problem: evaluating f cost inscreases as the dataset grows
  N number of training samples.
- Kernel or Gram matrix K:

$$K_{ij} = \kappa(x_i, x_j)$$

$\Longrightarrow$ storage cost: $O(N^2)$

Kernel approximation via random feature maps

Random Kitchen Sinks [Rahimi and Recht, 2007, Rahimi and Recht, 2009]

- $\left\langle \underbrace{z(\mathbf{x})}_{\in\mathbb{R}^k}, z(\mathbf{y}) \right\rangle \approx \left\langle \underbrace{\phi(\mathbf{x})}_{\in\mathbb{R}^D}, \phi(\mathbf{y}) \right\rangle = \kappa(\underbrace{\mathbf{x}}_{\in\mathbb{R}^n}, \mathbf{y})$

  where $k \gg n$; D high, possibly infinite.

- $z(\mathbf{x}) = \frac{1}{\sqrt{k}}\, s(\mathbf{G}\mathbf{x})$,
- random Gaussian matrix $\mathbf{G} \in \mathbb{R}^{k\times n}$ with $k \gg n$, $k = O(n\epsilon^{-2}\log\frac{1}{\epsilon^2})$,
- s is a nonlinearity function.

Still a problem...

- Storage of $\mathbf{G}$: $O(kn)$,
- Computation of $\mathbf{G}\mathbf{x}$: $O(kn)$.

Solution

- Storage of $\mathbf{G}_{struct}$: $O(k\log n)$,
- Computation of $\mathbf{G}_{struct}\mathbf{x}$: $O(k\log n)$.

Experimental protocol for kernel approximation (1/2)

$\mathbf{A} \in \mathbb{R}^{k \times n}$ with $k \gg n$,

Gaussian kernel

- $\kappa_G(\mathbf{x}, \mathbf{y}) = e^{\frac{-||\mathbf{x}-\mathbf{y}||_2^2}{2\sigma^2}}$,
- $\tilde{\kappa}_G(\mathbf{x}, \mathbf{y}) = \frac{1}{k}\, s(\mathbf{Ax})^T s(\mathbf{Ay})$ with $s(x) = e^{\frac{-ix}{\sigma}}$ applied pointwise.

Angular kernel

- $\kappa_0(\mathbf{x}, \mathbf{y}) = 1 - \frac{\theta}{\pi}$ with $\theta = \cos^{-1}\left(\frac{\mathbf{x}^T\mathbf{y}}{||\mathbf{x}||||\mathbf{y}||}\right)$,
- $\tilde{\kappa}_0(\mathbf{x}, \mathbf{y}) = 1 - \frac{dist_{Hamming}(s(\mathbf{Ax}), s(\mathbf{Ay}))}{k}$

with $s(x) = sign(x)$ applied pointwise.

Experiments for kernel approximation (1/4)

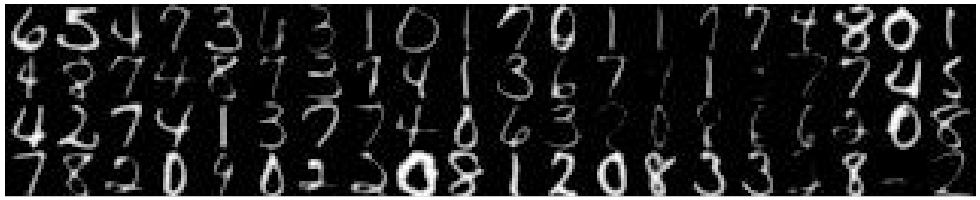
Speedups with Gaussian kernel

$Time(\mathbf{G})/Time(\mathbf{G}_{struct})$

Matrix dimensions	2^9	2^{10}	2^{11}	2^{12}	2^{13}	2^{14}	2^{15}
$\mathbf{G}_{Toeplitz}\mathbf{D}_2\mathbf{H}\mathbf{D}_1$	x1.4	x3.4	x6.4	x12.9	x28.0	x42.3	x89.6
$\mathbf{G}_{skew-circ}\mathbf{D}_2\mathbf{H}\mathbf{D}_1$	x1.5	x3.6	x6.8	x14.9	x31.2	x49.7	x96.5
$\mathbf{H}\mathbf{D}_{g_1,\ldots,g_n}\mathbf{H}\mathbf{D}_2\mathbf{H}\mathbf{D}_1$	x2.3	x6.0	x13.8	x31.5	x75.7	x137.0	x308.8
$\mathbf{H}\mathbf{D}_3\mathbf{H}\mathbf{D}_2\mathbf{H}\mathbf{D}_1$	x2.2	x6.0	x14.1	x33.3	x74.3	x140.4	x316.8

ex: $\mathbf{H}\mathbf{D}_3\mathbf{H}\mathbf{D}_2\mathbf{H}\mathbf{D}_1$, $k = 2^{15}$, $1.382s \rightarrow 4363\mu s$ in comparison with $\mathbf{G}$

Experimental protocol for kernel approximation (2/2)



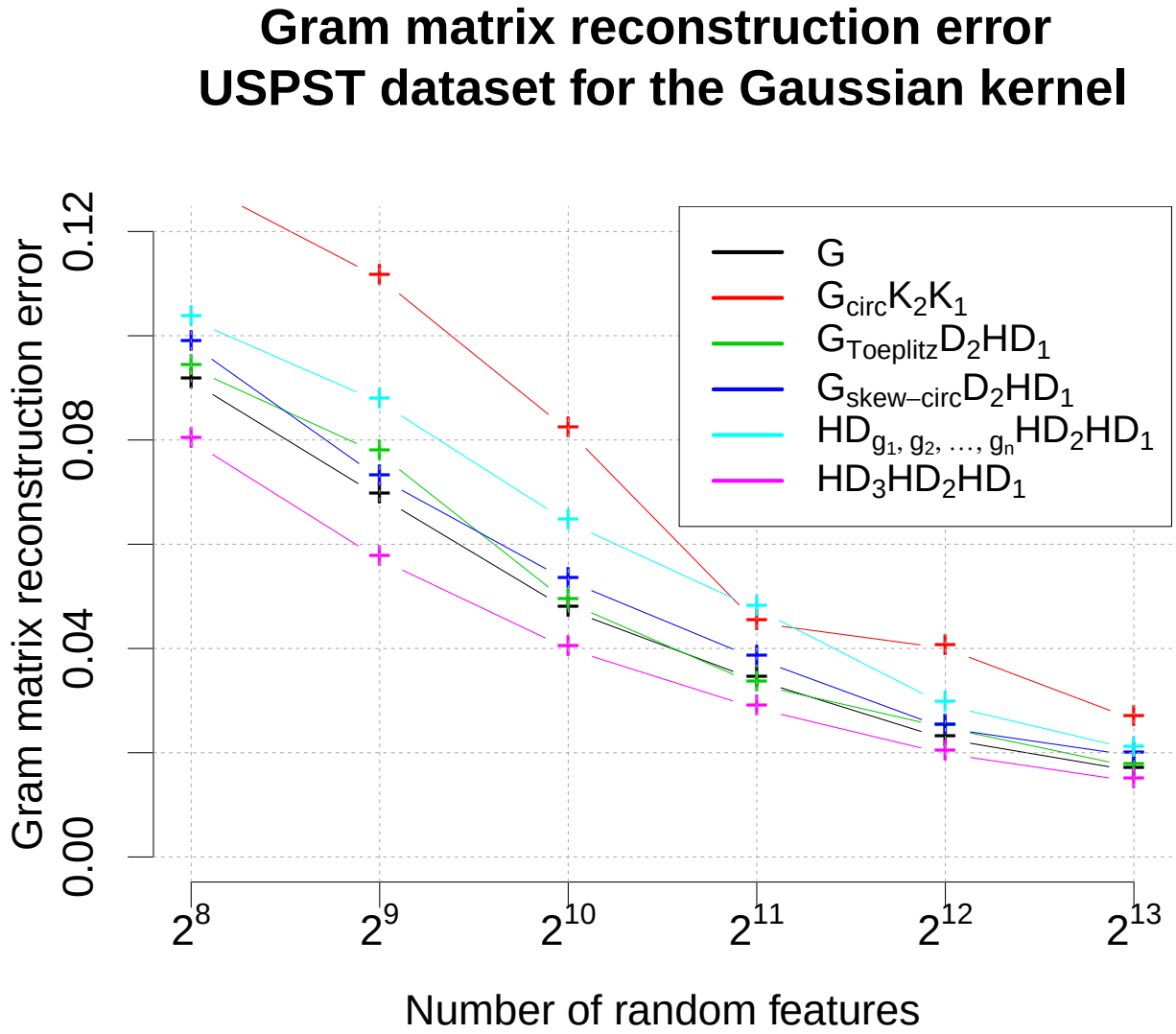
Measure of accuracy

- 10 runs,
- Dataset: USPST,
- 16×16 grayscale images,
- 2007 points of dimensionality 256 ($n = 256$),
- $\sigma = 9.4338$,
- Plots Gram reconstruction error: $\frac{||\mathbf{K} - \tilde{\mathbf{K}}||_F}{||\mathbf{K}||_F}$,
- $\mathbf{K}_{i,j} = \kappa(x_i, x_j)$.

Experiments for kernel approximation (2/3)

$$\frac{||\mathbf{K}-\tilde{\mathbf{K}}||_F}{||\mathbf{K}||_F}$$

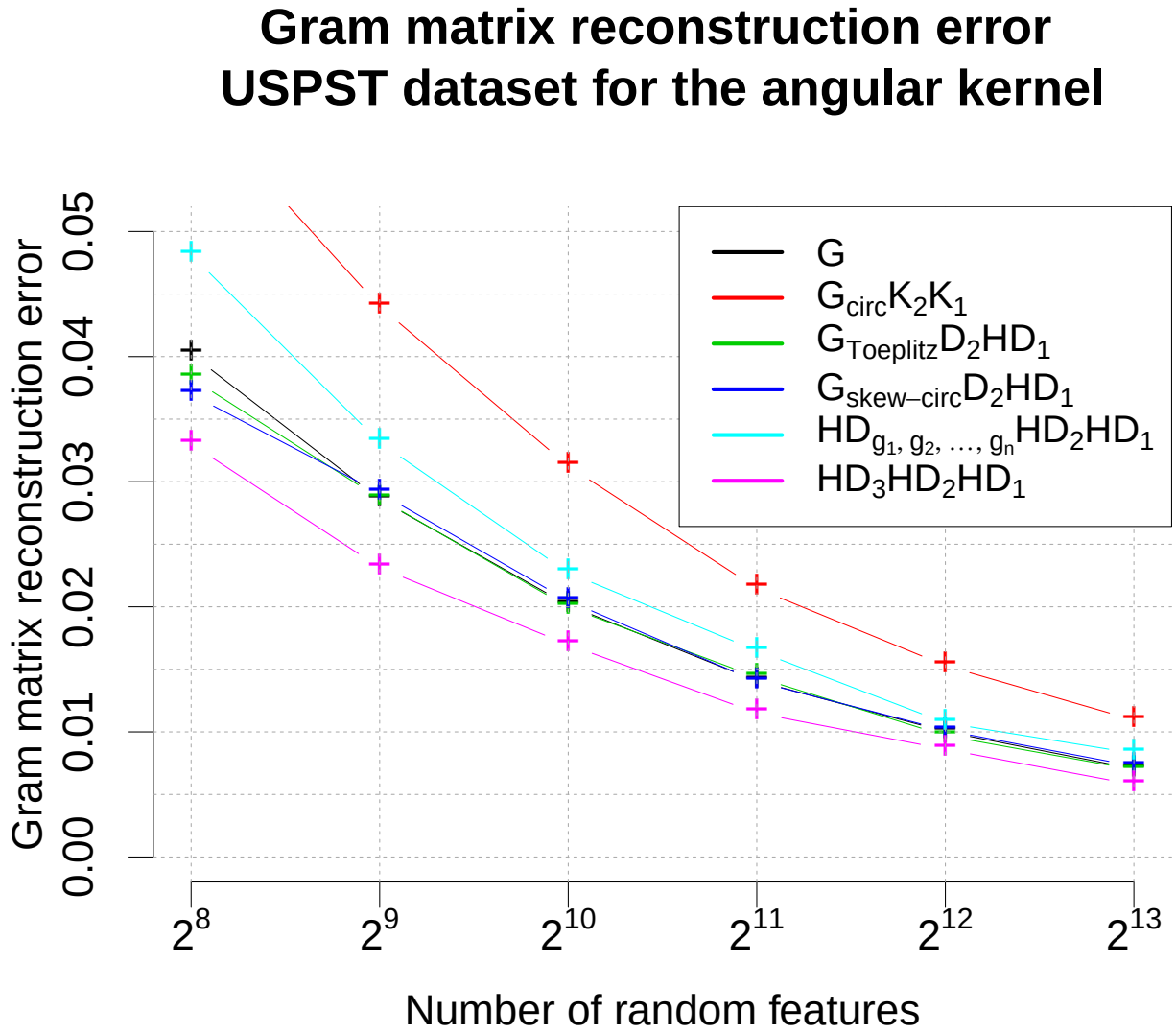
$$\mathbf{K}_{i,j} = \kappa(x_i, x_j)$$



Experiments for kernel approximation (3/3)

$$\frac{||\mathbf{K}-\tilde{\mathbf{K}}||_F}{||\mathbf{K}||_F}$$

$$\mathbf{K}_{i,j} = \kappa(x_i, x_j)$$



Plan

Unconstrained convex optimization with Newton step gradient descent

$$\text{minimize } f(x)$$

where $f : \mathbb{R} \to \mathbb{R}$ is convex and twice continuously differentiable.

- $x^{(t+1)} = x^{(t)} - \mu^{(t)} \nabla^2 f(x)^{-1} \nabla f(x)$
- *Newton decrement:* $\lambda = (\nabla f(x)^T \nabla^2 f(x)^{-1} \nabla f(x))^{1/2}$

  ↳ Used as stopping criterion + in backtracking line search:

  $$\text{while } \lambda^2 = -\nabla f(x)^T \Delta x > \epsilon$$

Principle of Newton sketch's algorithm [Pilanci and Wainwright, 2015]

Newton sketch's algorithm [Pilanci and Wainwright, 2015]

If analytic expression for the square root of the Hessian matrix:

$$x^{(t+1)} = x^{(t)} - \mu \, (\underbrace{(S^{(t)} \, (\nabla^2 f(x^{(t)}))^{1/2})^T}_{(SM)^T} \underbrace{S^{(t)} (\nabla^2 f(x^{(t)}))^{1/2}}_{SM})^{-1} \nabla f(x^{(t)})$$

where $S^{(t)} \in \mathbb{R}^{m \times n}$ is a sequence of isotropic sketchs matrices.

Example for Newton sketch's algorithm (1/2)

Large scale logistic regression problem

$$\min_{x \in \mathbb{R}^n} f(x)$$

$$\text{with } f(x) = \sum_{i=1}^{N} \log(1 + \exp(-y_i a_i^T x))$$

$$N \text{ observations } (a_i, y_i)_{i=1 \dots N}$$

$$\text{s.t. } a_i \in \mathbb{R}^n,$$

$$y_i \in \{-1, 1\}.$$

Example for Newton sketch's algorithm (2/2)

Analytic expressions for the gradient and the Hessian matrix

- $\nabla f(x^{(t)}) = \sum_{i=1}^{n} (\frac{1}{1+\exp(-y_i a_i^T x)} - 1) y_i a_i \ \in \mathbb{R}^n$,
- $\nabla^2 f(x^{(t)}) = A^T diag(\frac{1}{1+\exp(-a_i^T x)}(1 - \frac{1}{1+\exp(-a_i^T x)}))A \ \in \mathbb{R}^{n \times n}$,

  $A = [a_1^T ... a_N^T] \in \mathbb{R}^{N \times n}$, with $N \gg n$,
- We set

  $\nabla^2 f(x^{(t)})^{1/2} = diag(\frac{1}{1+\exp(-a_i^T x)}(1 - \frac{1}{1+\exp(-a_i^T x)}))^{1/2} A \in \mathbb{R}^{N \times n}$.

Newton sketch's algorithm, complexity analysis (1/2)

Comparison

- Exact Newton:

$$\nabla^2 f(x)^{-1}$$

$$\nabla^2 f(x^{(t)}) = A^T diag(\frac{1}{1+\exp(-a_i^T x)}(1 - \frac{1}{1+\exp(-a_i^T x)}))A$$

$$Cost = O(Nn^2 + n^3) \ (n \ll N)$$

Newton sketch's algorithm, complexity analysis (2/2)

Comparison

- Exact Newton:

$$Cost = O(Nn^2 + n^3) \ (n \ll N)$$

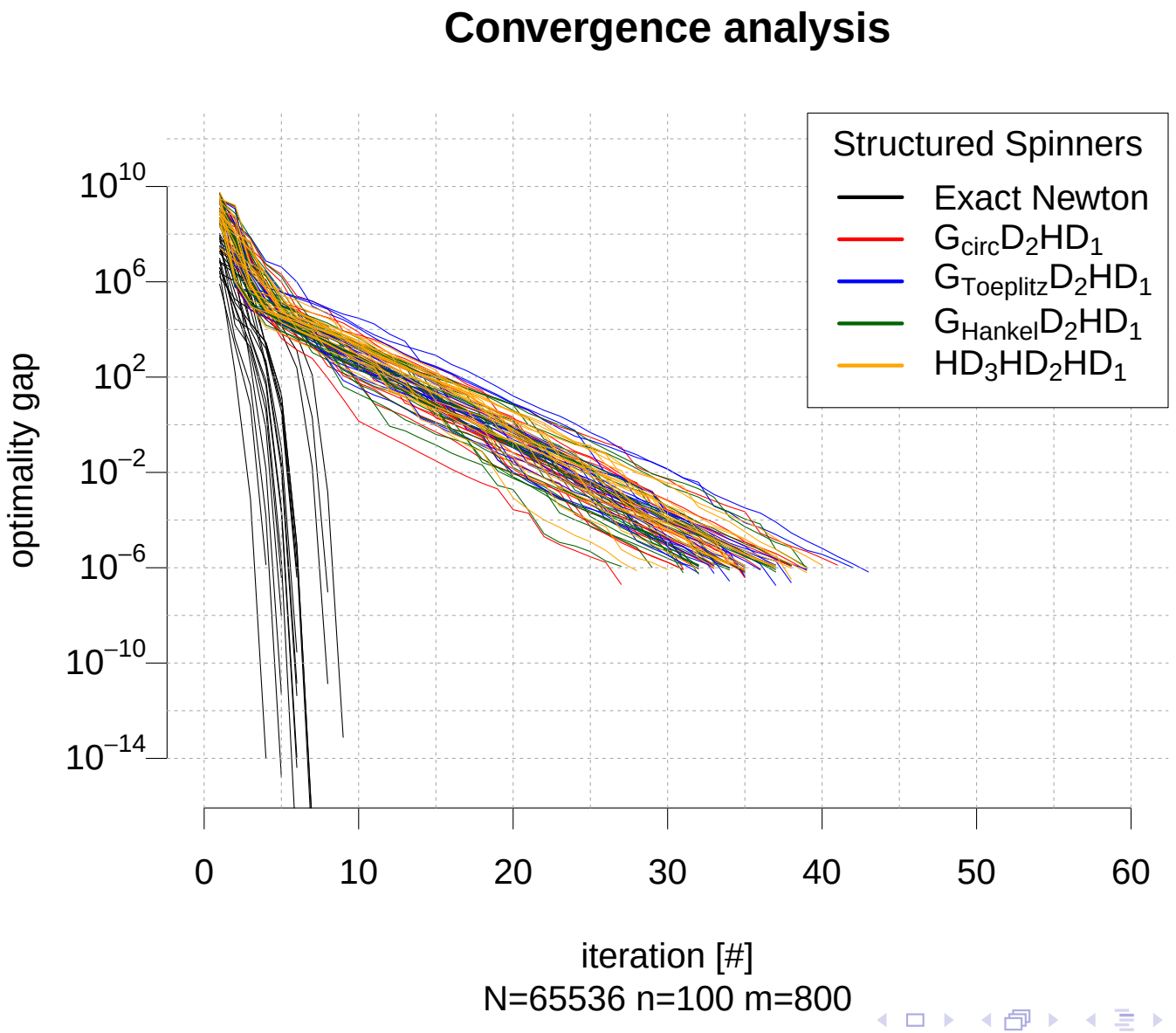
- Sketching: $((\underbrace{S^{(t)} \ (\nabla^2 f(x^{(t)}))^{1/2})^T}_{(SM)^T} \ \underbrace{S^{(t)}(\nabla^2 f(x^{(t)}))^{1/2}}_{SM} \ )^{-1}$

$$\nabla^2 f(x^{(t)})^{1/2} = diag(\frac{1}{1+\exp(-a_i^T x)}(1 - \frac{1}{1+\exp(-a_i^T x)}))^{1/2} A \in \mathbb{R}^{N \times n}$$

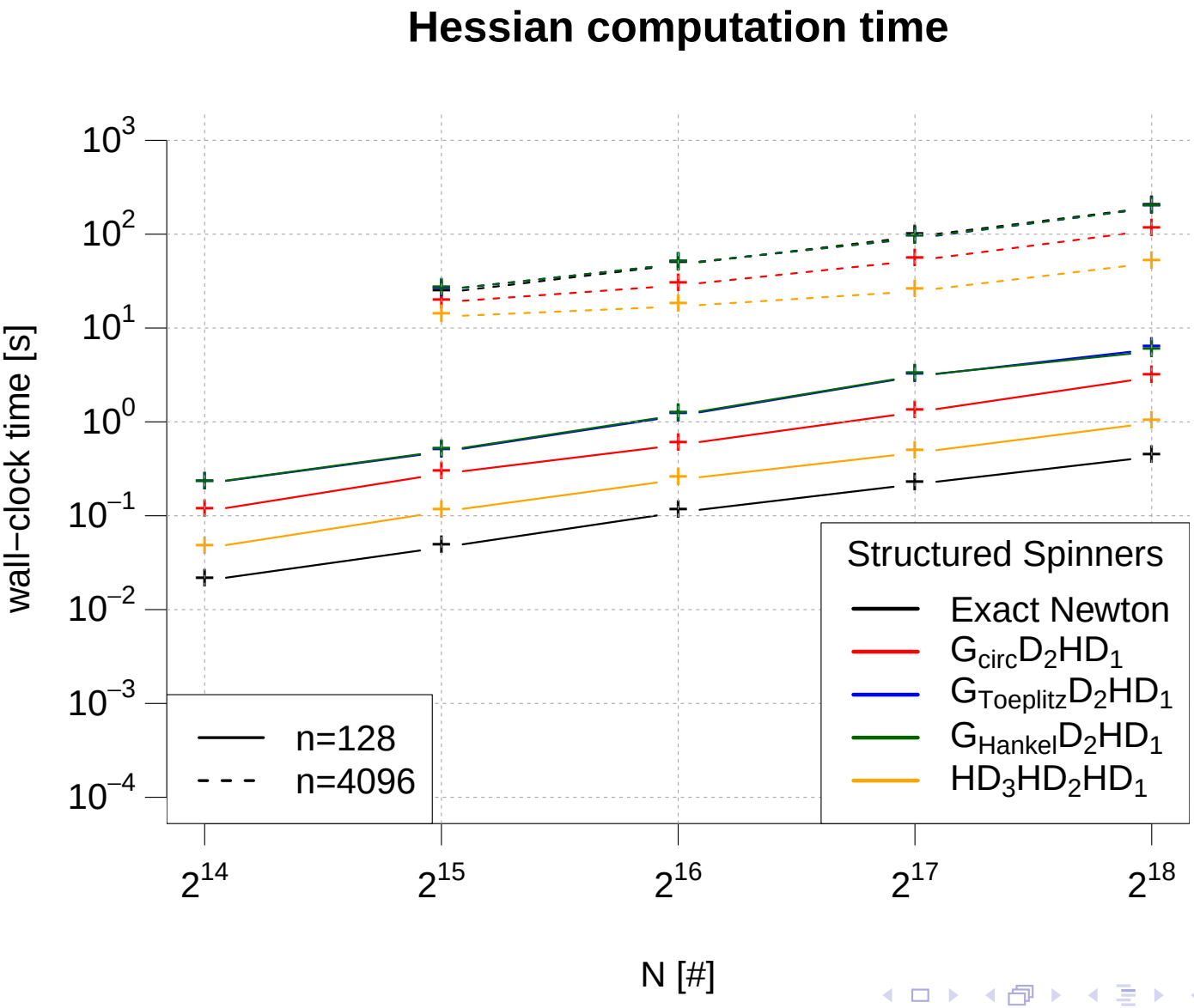
$$Cost = O(3nN \log N + mn^2 + n^3) \text{ with } m \ll N$$

Critical issue: when is $O(3nN \log N + mn^2)$ better than $O(Nn^2)$?

Experimental results (1/2)



Experimental results (2/2)

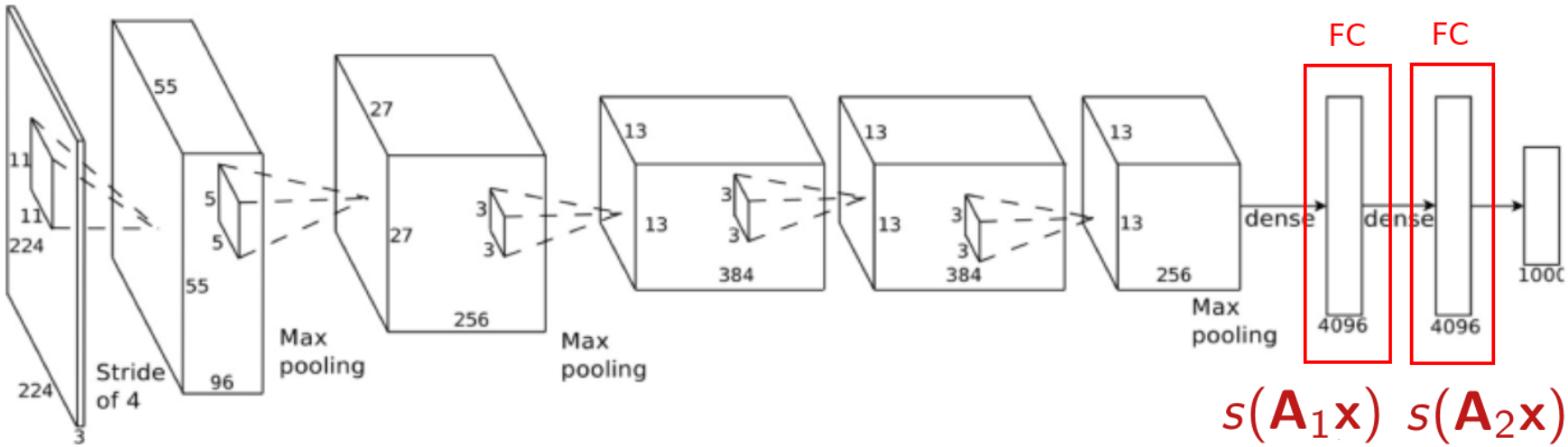


Plan

Plan

Deep neural networks and parameters

- Explosion of deep neural networks (e.g. convolutional networks) applications: with billions of parameters!
- Standard architecture: convolutional and fully-connected layers
- Convolutional layers: most of the computational effort
- Fully-connected layers: 90% of the parameters!
- Necessity to reduce the number of parameters for deployment on embedded mobile devices (speed up train + test time)



Plan

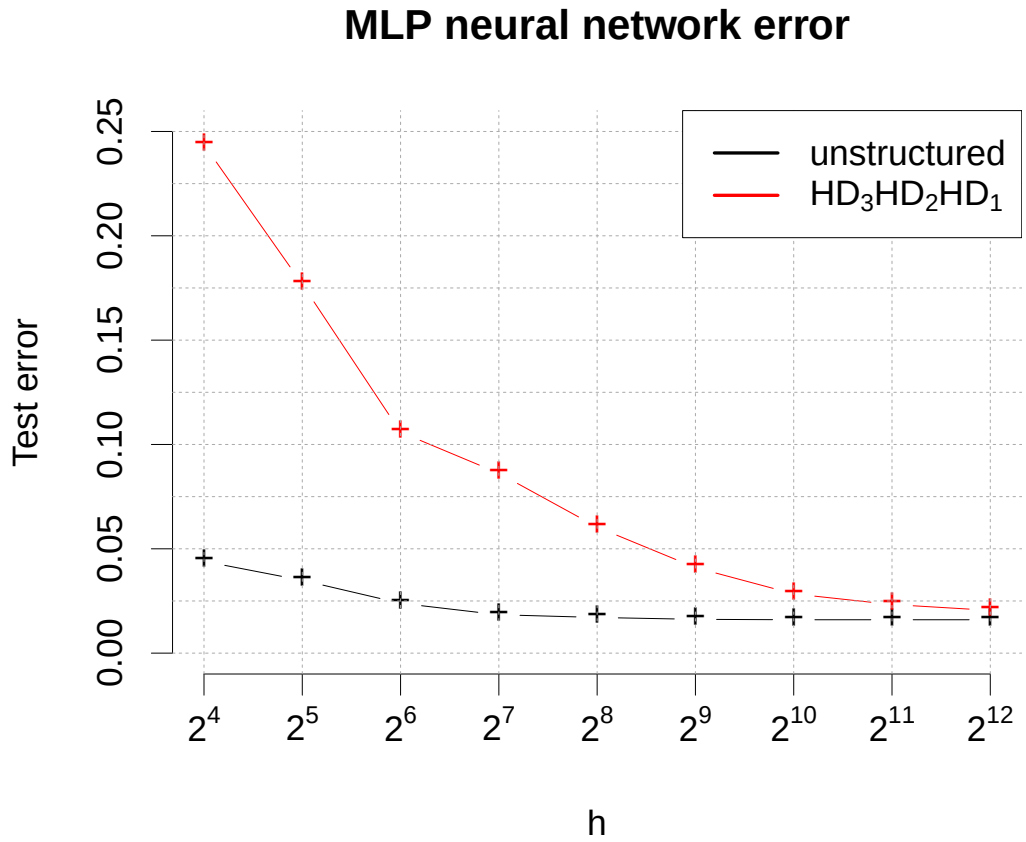
Related work

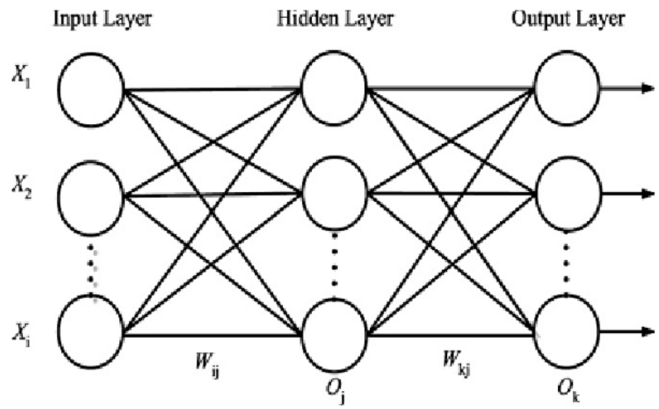
Some structured neural networks

- Deep Fried Convnets [Yang et al., 2015]: **SHGPHB** (Fastfood)
- [Moczulski et al., 2016]: $\mathbf{ACDC}^{-1}$, $\mathbf{A}$, $\mathbf{D}$ diagonal, $\mathbf{C}$ is the discrete cosine transform
- [Denil et al., 2013]: **UV**, fix $\mathbf{U} \in \mathbb{R}^{m \times r}$ and learn $\mathbf{V} \in \mathbb{R}^{r \times n}$, $r \ll m, n$
- [Sainath et al., 2013]: low-rank matrix factorization **UV**
- [Xue et al., 2013]: $\mathbf{U}(\mathbf{\Sigma}\mathbf{V}^T)$, $\mathbf{U} \in \mathbb{R}^{m \times r}$, $\mathbf{\Sigma} \in \mathbb{R}^{r \times r}$, $\mathbf{V} \in \mathbb{R}^{n \times r}$ (after training)

Plan

Structured MLP with 2 fully-connected layers on MNIST



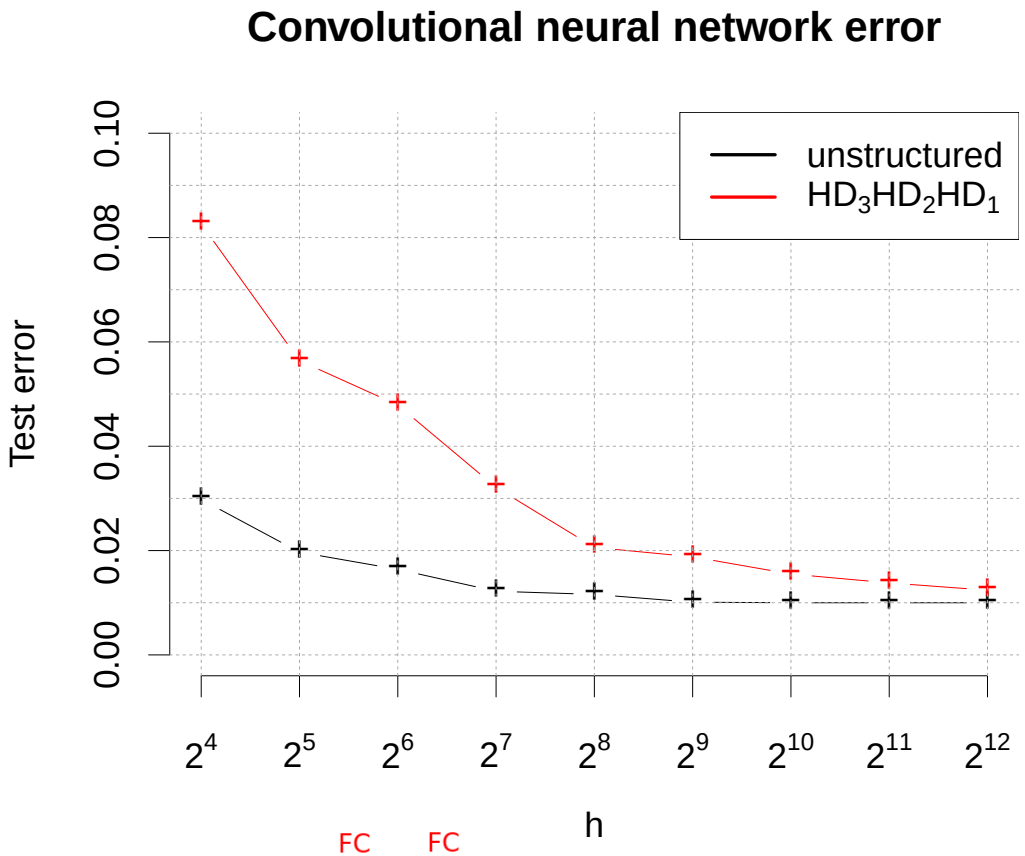


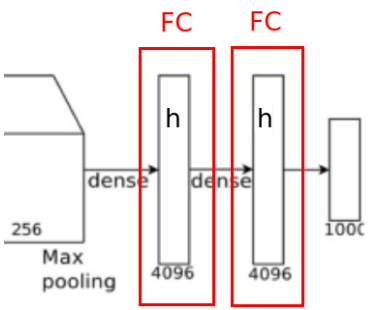
$s(\mathbf{A}_1\mathbf{x})$ $s(\mathbf{A}_2\mathbf{x})$

Running time (in μs), h: size of the hidden layers

h	2^4	2^5	2^6	2^7	2^8	2^9	2^{10}	2^{11}	2^{12}
unstructured	42.9	51.9	72.7	99.9	163.9	350.5	716.7	1271.5	2317.4
$HD_3HD_2HD_1$	109.2	121.3	109.7	114.2	117.4	123.9	130.6	214.3	389.8

Structured convolutional network on MNIST





$s(\mathbf{A}_1\mathbf{x})$ $s(\mathbf{A}_2\mathbf{x})$

- Conv. layer with filter size 5×5, 4 feature maps + ReLU + Max Pooling (region 2×2 and step 2×2)
- Conv. layer with filter size 5×5, 6 feature maps + ReLU + Max Pooling (region 2×2 and step 2×2)
- FC layer (h outputs) + ReLU
- FC layer (10 outputs)
- LogSoftMax.

Plan

Conclusion

Structured Spinners paper brings:

- a general structured paradigm for large scale machine learning computations with random matrices, providing computational speedups and storage compression with various applications:
  - kernel approximations via random feature maps
  - dimensionality reduction algorithms
  - deep learning
  - convex optimization via Newton sketches
  - quantization wit random projection trees
- theoretical guarantees on the effectiveness of the structured approach.

Open question

Can one obtain computation speedups for these matrices from the *Structured Spinners* model for which the Fast Fourier Transform trick does not work ?

Thank you for your attention!

References I

Achlioptas, D. (2003).
Database-friendly random projections: Johnson-Lindenstrauss with binary coins.
J. Comput. Syst. Sci., 66(4):671–687.

Ailon, N. and Chazelle, B. (2006).
Approximate nearest neighbors and the fast Johnson-Lindenstrauss transform.
In Proceedings of the 38th STOC, pages 557–563. ACM.

Andoni, A., Indyk, P., Laarhoven, T., Razenshteyn, I. P., and Schmidt, L. (2015).
Practical and optimal LSH for angular distance.
In NIPS, pages 1225–1233.

Choromanski, K. and Sindhwani, V. (2016).
Recycling randomness with structure for sublinear time kernel expansions.
In Proceedings of the 33rd International Conference on International Conference on Machine Learning - Volume 48, ICML'16, pages 2502–2510.

Denil, M., Shakibi, B., Dinh, L., Ranzato, M., and Freitas, N. D. (2013).
Predicting parameters in deep learning.
In NIPS.

Feng, C., Hu, Q., and Liao, S. (2015).
Random feature mapping with signed circulant matrix projection.
In Proceedings of the 24th IJCAI, pages 3490–3496.

References II

Frankl, P. and Maehara, H. (1987).
The Johnson-Lindenstrauss lemma and the sphericity of some graphs.
J. Comb. Theory Ser. A, 44(3):355–362.

Johnson, W. and Lindenstrauss, J. (1984).
Extensions of Lipschitz mappings into a Hilbert space.
In Conference in modern analysis and probability (New Haven, Conn., 1982), volume 26 of Contemporary Mathematics, pages 189–206. American Mathematical Society.

Kane, D. M. and Nelson, J. (2010).
A sparser Johnson-Lindenstrauss transform.
CoRR, abs/1012.1577.

Le, Q., Sarlós, T., and Smola, A. (2013).
Fastfood-computing hilbert space expansions in loglinear time.
In Proceedings of the 30th ICML, pages 244–252.

Moczulski, M., Denil, M., Appleyard, J., and de Freitas, N. (2016).
ACDC: A structured efficient linear layer.

Pilanci, M. and Wainwright, M. J. (2015).
Newton sketch: A linear-time optimization algorithm with linear-quadratic convergence.
CoRR, abs/1505.02250.

Rahimi, A. and Recht, B. (2007).
Random features for large-scale kernel machines.
In NIPS, pages 1177–1184.

References III

Rahimi, A. and Recht, B. (2009).
Weighted sums of random kitchen sinks: Replacing minimization with randomization in learning.
In Advances in Neural Information Processing Systems 21, pages 1313–1320.

Sainath, T. N., Kingsbury, B., Sindhwani, V., Arisoy, E., and Ramabhadran, B. (2013).
Low-rank matrix factorization for deep neural network training with high-dimensional output targets.
In ICASSP.

Terasawa, K. and Tanaka, Y. (2007).
Spherical LSH for approximate nearest neighbor search on unit hypersphere.
In WADS, pages 27–38.

Xue, J., Li, J., and Gong, Y. (2013).
Restructuring of deep neural network acoustic models with singular value decomposition.
In Bimbot, F., Cerisara, C., Fougeron, C., Gravier, G., Lamel, L., Pellegrino, F., and Perrier, P., editors, Interspeech, pages 2365–2369. ISCA.

Yang, Z., Moczulski, M., Denil, M., d. Freitas, N., Smola, A., Song, L., and Wang, Z. (2015).
Deep fried convnets.
In 2015 IEEE International Conference on Computer Vision (ICCV), pages 1476–1483.

Yu, F. X., Kumar, S., Gong, Y., and Chang, S. (2014).
Circulant binary embedding.
In Proceedings of the 31th International Conference on Machine Learning, ICML 2014, Beijing, China, 21-26 June 2014, pages 946–954.

References IV

Zhang, X., Yu, F. X., Guo, R., Kumar, S., Wang, S., and Chang, S. F. (2015).
Fast orthogonal projection based on kronecker product.
In 2015 IEEE International Conference on Computer Vision (ICCV), pages 2929–2937.

Kronecker matrices

Gaussian or discrete Kronecker matrix

$$\mathbf{K} = \mathbf{R}_1 \otimes \mathbf{R}_2 \otimes \ldots \otimes \mathbf{R}_m \in \mathbb{R}^{2^m \times 2^m}$$

$$\mathbf{R}_i \in \mathbb{R}^{2\times 2} \text{ or } \mathbf{R}_i \in \{-1, 1\}^{2\times 2}$$

$$\mathbf{R}_i \mathbf{R}_i^T = \mathbf{R}_i^T \mathbf{R}_i = \mathbf{I}_2$$

Kronecker product [Zhang et al., 2015]

$$\text{For } \mathbf{A} \in \mathbb{R}^{k_1, \times d_1}, \ \mathbf{B} \in \mathbb{R}^{k_1, \times d_1},$$

$$\mathbf{A} \otimes \mathbf{B} = \begin{pmatrix} \mathbf{A}_{11}\mathbf{B} & \ldots & \mathbf{A}_{1d_1}\mathbf{B} \\ \mathbf{A}_{21}\mathbf{B} & \ldots & \mathbf{A}_{2d_1}\mathbf{B} \\ \vdots & \ddots & \vdots \\ \mathbf{A}_{k_1 1}\mathbf{B} & \ldots & \mathbf{A}_{k_1 d_1}\mathbf{B} \end{pmatrix} \in \mathbb{R}^{k_1 k_2 \times d_1 d_2}$$

Thank you for your attention!