Random Projections

Lopez–Paz & Duvenaud

November 7, 2013
Random Outline


Random Kitchen Sinks (Rahimi and Recht, NIPS 2008)

Fastfood (Le et al., ICML 2013)
Why random projections?

Fast, efficient and $\mathcal{O}'$ distance-preserving **dimensionality reduction**!

$$w \in \mathbb{R}^{40500 \times 1000}$$

$$x_1 \xrightarrow{\delta} y_1 \xleftarrow{\delta(1 \pm \epsilon)} x_2 \xrightarrow{\delta} y_2$$

$$w \in \mathbb{R}^{40500 \times 1000}$$

$$\mathbb{R}^{40500} \xrightarrow{\delta} \mathbb{R}^{1000}$$

$$(1 - \epsilon) \|x_1 - x_2\|^2 \leq \|y_1 - y_2\|^2 \leq (1 + \epsilon) \|x_1 - x_2\|^2$$

This result is formalized in the **Johnson-Lindenstrauss Lemma**
The Johnson-Lindenstrauss Lemma

The proof is a great example of Erdös’ *probabilistic method* (1947).

§12.5 of *Foundations of Machine Learning* (Mohri et al., 2012)
Auxiliary Lemma 1

Let $Q$ be a random variable following a $\chi^2$ distribution with $k$ degrees of freedom. Then, for any $0 < \epsilon < 1/2$:

$$\Pr[(1 - \epsilon)k \leq Q \leq (1 + \epsilon)k] \geq 1 - 2e^{-(\epsilon^2 - \epsilon^3)k/4}.$$ 

Proof: we start by using Markov’s inequality $\left( \Pr[X > a] \leq \frac{E[X]}{a} \right)$:

$$\Pr[Q \geq (1 + \epsilon)k] = \Pr[e^{\lambda Q} \geq e^{\lambda(1+\epsilon)k}] \leq \frac{E[e^{\lambda Q}]}{e^{\lambda(1+\epsilon)k}} = \frac{(1 - 2\lambda)^{-k/2}}{e^{\lambda(1+\epsilon)k}},$$

where $E[e^{\lambda Q}] = (1 - 2\lambda)^{-k/2}$ is the m.g.f. of a $\chi^2$ distribution, $\lambda < \frac{1}{2}$.

To tighten the bound we minimize the right-hand side with $\lambda = \frac{\epsilon}{2(1+\epsilon)}$:

$$\Pr[Q \geq (1 + \epsilon)k] \leq \frac{(1 - \frac{\epsilon}{1+\epsilon})^{-k/2}}{e^{\epsilon k/2}} = \frac{(1 + \epsilon)^{k/2}}{(e^\epsilon)^{k/2}} = \left(\frac{1 + \epsilon}{e^\epsilon}\right)^{k/2}. $$
Auxiliary Lemma 1

Using $1 + \epsilon \leq e^{\epsilon - (\epsilon^2 - \epsilon^3)/2}$ yields

$$\Pr[Q \geq (1 + \epsilon)k] \leq \left( \frac{1 + \epsilon}{e^\epsilon} \right)^{k/2} \leq \left( \frac{e^{\epsilon - \frac{\epsilon^2 - \epsilon^3}{2}}}{e^\epsilon} \right) = e^{-\frac{k}{4}(\epsilon^2 - \epsilon^3)}.$$

$\Pr[Q \leq (1 - \epsilon)k]$ is bounded similarly, and the lemma follows by applying the union bound:

$$\Pr[(1 - \epsilon)k \leq Q \leq (1 + \epsilon)k] \leq \Pr[Q \geq (1 + \epsilon)k \cup Q \leq (1 - \epsilon)k] \leq \Pr[Q \geq (1 + \epsilon)k] + \Pr[Q \leq (1 - \epsilon)k] = 2e^{-\frac{k}{4}(\epsilon^2 - \epsilon^3)}$$

Then,

$$\Pr[(1 - \epsilon)k \leq Q \leq (1 + \epsilon)k] \geq 1 - 2e^{-\frac{k}{4}(\epsilon^2 - \epsilon^3)}$$
Let $\boldsymbol{x} \in \mathbb{R}^N$, $k < N$ and $\mathbf{A} \in \mathbb{R}^{k \times N}$ with $A_{ij} \sim \mathcal{N}(0, 1)$. Then, for any $0 \leq \epsilon \leq 1/2$:

$$
\Pr[(1 - \epsilon)\|\boldsymbol{x}\|^2 \leq \|\frac{1}{\sqrt{k}}\mathbf{A}\boldsymbol{x}\|^2 \leq (1 + \epsilon)\|\boldsymbol{x}\|^2] \geq 1 - 2e^{-(\epsilon^2 - \epsilon^3)k/4}.
$$

**Proof:** let $\hat{\boldsymbol{x}} = \mathbf{A}\boldsymbol{x}$. Then,

$$
E[\hat{x}_j^2] = E \left[ \left( \sum_{i}^{N} A_{ji}x_i \right)^2 \right] = E \left[ \sum_{i}^{N} A_{ji}^2x_i^2 \right] = \sum_{i}^{N} x_i^2 = \|\boldsymbol{x}\|^2.
$$

Note that $T_j = \hat{x}_j/\|\boldsymbol{x}\| \sim \mathcal{N}(0, 1)$. Then, $Q = \sum_{i}^{k} T_j^2 \sim \chi_k^2$.

Remember the previous lemma?
Remember: $\hat{x} = Ax, T_j = \hat{x}_j/\|\hat{x}\| \sim N(0, 1), Q = \sum_{i}^{k} T_j^2 \sim \chi_k^2$:

$$\Pr[(1 - \epsilon)\|x\|^2 \leq \|\frac{1}{\sqrt{k}}Ax\|^2 \leq (1 + \epsilon)\|x\|^2] =$$

$$\Pr[(1 - \epsilon)\|x\|^2 \leq \frac{\|\hat{x}\|^2}{k} \leq (1 + \epsilon)\|x\|^2] =$$

$$\Pr[(1 - \epsilon)k \leq \frac{\|\hat{x}\|^2}{\|x\|^2} \leq (1 + \epsilon)k] =$$

$$\Pr \left[(1 - \epsilon)k \leq \sum_{i}^{k} T_j^2 \leq (1 + \epsilon)k \right] =$$

$$\Pr \left[(1 - \epsilon)k \leq Q \leq (1 + \epsilon)k \right] \geq 1 - 2e^{-(\epsilon^2 - \epsilon^3)k/4}$$
The Johnson-Lindenstrauss Lemma

For any $0 < \epsilon < 1/2$ and any integer $m > 4$, let $k = \frac{20 \log m}{\epsilon^2}$. Then, for any set $V$ of $m$ points in $\mathbb{R}^N \ni f : \mathbb{R}^N \to \mathbb{R}^k$ s.t. $\forall u, v \in V$:

$$(1 - \epsilon)\|u - v\|^2 \leq \|f(u) - f(v)\|^2 \leq (1 + \epsilon)\|u - v\|^2.$$

**Proof:** Let $f = \frac{1}{\sqrt{k}}A$, $A \in \mathbb{R}^{k \times N}$, $k < N$ and $A_{ij} \sim \mathcal{N}(0, 1)$.

- For fixed $u, v \in V$, we apply the previous lemma with $x = u - v$ to lower bound the success probability by $1 - 2e^{-(\epsilon^2 - \epsilon^3)k/4}$.

- Union bound again! This time over the $m^2$ pairs in $V$ with $k = \frac{20 \log m}{\epsilon^2}$ and $\epsilon < 1/2$ to obtain:

$$\Pr[success] \geq 1 - 2m^2e^{-(\epsilon^2 - \epsilon^3)k/4} = 1 - 2m^{5\epsilon - 3} > 1 - 2m^{-1/2} > 0.$$
JL Experiments

Data: 20-newsgroups, from 100,000 features to 300 (0.3\%)
JL Experiments

Data: 20-newsgroups, from 100,000 features to 1,000 (1%)
JL Experiments

Data: 20-newsgroups, from 100,000 features to 10,000 (10%)

MATLAB implementation: $1 / \sqrt{k} \cdot \text{randn}(k, N) \cdot X$. 
JL Conclusions

- Do you have a huge feature space?
- Are you wasting too much time with PCA?
- Random Projections are fast, compact & efficient!
- Monograph (Vampala, 2004)
- Sparse Random Projections (Achlioptas, 2003)
- Random Projections can Improve MoG! (Dasgupta, 2000)

But... What about non-linear random projections?

Ali Rahimi

Ben Recht
Random Outline


Random Kitchen Sinks (Rahimi and Recht, NIPS 2008)

Fastfood (Le et al., ICML 2013)
A Familiar Creature

\[ f(x) = \sum_{i=1}^{T} \alpha_i \phi(x; w_i) \]

- Gaussian Process
- Kernel Regression
- SVM
- AdaBoost
- Multilayer Perceptron
- ...

Same model, different training approaches!

Things get interesting when \( \phi \) is non-linear...
A Familiar Solution
A Familiar Solution
A Familiar Monster: The Kernel Trap

$k(x_i, x_j)$
Approx. \( f(x) = \sum_{i=1}^{\infty} \alpha_i \phi(x; w_i) \) with \( f_T(x) = \sum_{i=1}^{T} \alpha_i \phi(x; w_i) \).

Greedy Fitting

\[
(\alpha^*, W^*) = \min_{\alpha, W} \left\| \sum_{i=1}^{T} \alpha_i \phi(; w_i) - f \right\|_\mu
\]
Greedy Approximation of Functions

\[ \mathcal{F} \equiv \{ f(x) = \sum_{i=1}^{\infty} \alpha_i \phi(x; w_i), w_i \in \Omega, \|\alpha\|_1 \leq C \} \]

\[ f(x) = \sum_{i=1}^{T} \alpha_i \phi(x; w_i), w_i \in \Omega, \|\alpha\|_1 \leq C \]

\[ \|f_T - f\|_\mu = \sqrt{\int_X (f_T(x) - f(x))^2 \mu(dx)} = O \left( \frac{C}{\sqrt{T}} \right) \] (Jones, 1992)
RKS Approximation of Functions

Approx. \( f(x) = \sum_{i=1}^{\infty} \alpha_i \phi(x; w_i) \) with \( f_T(x) = \sum_{i=1}^{T} \alpha_i \phi(x; w_i) \).

Greedy Fitting

\[
(\alpha^*, W^*) = \min_{\alpha, W} \left\| \sum_{i=1}^{T} \alpha_i \phi(; w_i) - f \right\|_{\mu}
\]

Random Kitchen Sinks Fitting

\( w_i^*, \ldots, w_T^* \sim p(w), \quad \alpha^* = \min_{\alpha} \left\| \sum_{i=1}^{T} \alpha_i \phi(; w_i^*) - f \right\|_{\mu} \)
The Perceptron: A Model for Brain Functioning. I*

H. D. Block

Cornell University, Ithaca, New York

The Perceptron is a self-organizing or adaptive system proposed by Rosenblatt. Its primary purpose is to shed some light on the problem of explaining brain function in terms of brain structure. It also has technological applications as a pattern-recognizing device, but here our emphasis is on the brain function-structure problem. The technological aspects are not completely irrelevant however, since a model, no matter how appealing it may appear from the point of view of structural similarity, must also be judged on the basis of its performance.

In brief a perceptron consists of a retina of sensory units (for example photocells); these are connected (for example by wires) to associator units. The connections are many to many and random. The associator units may be connected to each other or to response units. When a stimulus is presented to the retina
Just an old idea?

Liquid state machine

From Wikipedia, the free encyclopedia

A liquid state machine (LSM) is a computational construct like a neural network. An LSM consists of a large collection of units (called nodes, or neurons). Each node receives time varying input from external sources (the inputs) as well as from other nodes. Nodes are randomly connected to each other. The recurrent nature of the connections turns the time varying input into a spatio-temporal pattern of activations in the network nodes. The spatio-temporal patterns of activation are read out by linear discriminant units.

The soup of recurrently connected nodes will end up computing a large variety of nonlinear functions on the input. Given a large enough variety of such nonlinear functions, it is theoretically possible to obtain linear combinations (using the read out units) to perform whatever mathematical operation is needed to perform a certain task, such as speech recognition or computer vision.

How does RKS work?

For functions \( f(x) = \int_{\Omega} \alpha(w) \phi(x; w) dw \), define the \( p \)-norm as:

\[
\|f\|_p = \sup_{w \in \Omega} \frac{|\alpha(w)|}{p(w)}
\]

and let \( \mathcal{F}_p \) be all \( f \) with \( \|f\|_p \leq C \). Then, for \( w_1, \ldots, w_T \sim p(w) \), w.p. at least \( 1 - \delta \), \( \delta > 0 \), there exist some \( \alpha \) s.t. \( f_T \) satisfies:

\[
\|f_T - f\|_\mu = O \left( \frac{C}{\sqrt{T}} \sqrt{1 + 2 \log \frac{1}{\delta}} \right)
\]

Why? Set \( \alpha_i = \frac{1}{T} \alpha(w_i) \). Then, the discrepancy is given by \( \|f\|_p \):

\[
f_T(x) = \frac{1}{T} \sum_{i}^{T} a_i(w_i) \phi(x; w_i)
\]

With a dataset of size \( N \), an error \( O \left( \frac{1}{\sqrt{N}} \right) \) is added to all bounds.
RKS Approximation of Functions

\[ \mathcal{F}_p \equiv \{ f(x) = \int_\Omega \alpha(w) \phi(x; w) \, dw, |\alpha(w)| \leq Cp(w) \} \]

\[ f(x) = \sum_{i=1}^{T} \alpha_i \phi(x; w_i) \]

\[ w_i \sim p(w) \]

\[ \alpha_i \leq \frac{C}{K} \]

\[ O \left( \frac{\|f\|_p}{\sqrt{T}} \right) \]
Random Kitchen Sinks: Auxiliary Lemma

Let $X = \{x_1, \cdots, x_K\}$ be i.i.d. r.v.s drawn from a centered $C$-radius ball of a Hilbert Space $\mathcal{H}$. Let $\overline{X} = \frac{1}{K} \sum_{k}^{K} x_k$. Then, for any $\delta > 0$, with probability at least $1 - \delta$:

$$\| \overline{X} - E[\overline{X}] \| \leq \frac{C}{\sqrt{K}} \left( 1 + \sqrt{2 \log \frac{1}{\delta}} \right)$$

Proof: Show that $f(X) = \| \overline{X} - E[\overline{X}] \|$ is stable w.r.t. perturbations:

$$|f(X) - f(\tilde{X})| \leq \| \overline{X} - \tilde{X} \| \leq \frac{\|x_i - x'_i\|}{K} \leq \frac{2C}{K}.$$ 

Second, the variance of the average of i.i.d. random variables is:

$$E[\| \overline{X} - E[\overline{X}] \|^2] = \frac{1}{K} (E[\|x\|^2] - \|E[x]\|^2).$$

Third, using Jensen’s inequality and given that $\|x\| \leq C$:

$$E[f(X)] \leq \sqrt{E[f^2(X)]} = \sqrt{E[\| \overline{X} - E[\overline{X}] \|^2]} \leq \frac{C}{\sqrt{K}}$$

Fourth, use McDiarmid’s inequality and rearrange.
Let $\mu$ be a measure on $\mathcal{X}$, and $f^* \in \mathcal{F}_p$. Let $\mathbf{w}_1, \ldots, \mathbf{w}_T \sim p(\mathbf{w})$. Then, w.p. at least $1 - \delta$, with $\delta > 0$, $\exists f_T(x) = \sum_i^T \beta_i \phi(x; \mathbf{w}_i)$ s.t.:

$$\|f_T - f^*\|_{\mu} \leq \frac{C}{\sqrt{T}} \left(1 + \sqrt{2 \log \frac{1}{\delta}}\right)$$

**Proof:**

Let $f_i = \beta_i \phi(x; \mathbf{w}_i)$, $1 \leq k \leq T$ and $\beta_i = \frac{\alpha(\mathbf{w}_i)}{p(\mathbf{w}_i)}$. Then, $E[f_i] = f^*$:

$$E[f_i] = E_{\mathbf{w}} \left[ \frac{\alpha(\mathbf{w}_i)}{p(\mathbf{w}_i)} \phi(\mathbf{w}_i) \right] = \int_\Omega p(\mathbf{w}) \frac{\alpha(\mathbf{w})}{p(\mathbf{w})} \phi(\mathbf{w}) d\mathbf{w} = f^*$$

The claim is mainly completed by describing the concentration of the average $f_T = \frac{1}{T} \sum f_i$ around $f^*$ with the previous lemma.
Approximating Kernels with RKS

Bochner’s Theorem: A kernel \( k(\mathbf{x} - \mathbf{y}) \) on \( \mathbb{R}^d \) is PSD if and only if \( k(\mathbf{x} - \mathbf{y}) \) is the Fourier transform of a non-negative measure \( p(\mathbf{w}) \).

\[
    k(\mathbf{x} - \mathbf{y}) = \int_{\mathbb{R}^d} p(\mathbf{w}) e^{j\mathbf{w}'(\mathbf{x} - \mathbf{y})} d\mathbf{w}
\]

\[
    \approx \frac{1}{T} \sum_{i=1}^{T} e^{j\mathbf{w}_i'(\mathbf{x} - \mathbf{y})} \quad \text{(Monte-Carlo, } O(T^{-1/2})\text{)}
\]

\[
    = \frac{1}{T} \sum_{i=1}^{T} e^{j\mathbf{w}_i'\mathbf{x}} e^{-j\mathbf{w}_i'\mathbf{y}}
\]

\[
    = \frac{1}{\sqrt{T}} \phi(\mathbf{x}; \mathbf{W})^* \frac{1}{\sqrt{T}} \phi(\mathbf{y}; \mathbf{W})
\]

Now solve least squares in the primal in \( O(n) \) time!
Random Kitchen Sinks: Implementation

```matlab
function ytest = kitchen_sinks( X, y, Xtest, T, noise)

Z = randn(T, size(X,1)); % Sample feature frequencies
phi = exp(i*Z*X); % Compute feature matrix

% Linear regression with observation noise.
w = (phi*phi' + eye(T)*noise)\(phi*y);

% testing
ytest = w'*exp(i*Z*xtest);
```

from http://www.keysduplicated.com/~ali/random-features/

- That’s fast, approximate GP regression! (with a sq-exp kernel)
- Or linear regression with [sin(zx), cos(zx)] feature pairs
- Show demo
Random Kitchen Sinks and Gram Matrices

- How fast do we approach the exact Gram matrix?
- \( k(X, X) = \Phi(X)^T \Phi(X) \)
Random Kitchen Sinks and Gram Matrices

- How fast do we approach the exact Gram matrix?
- \( k(X, X) = \Phi(X)^T \Phi(X) \)
Random Kitchen Sinks and Gram Matrices

- How fast do we approach the exact Gram matrix?
- $k(X, X) = \Phi(X)^T \Phi(X)$
Random Kitchen Sinks and Gram Matrices

- How fast do we approach the exact Gram matrix?
- \( k(X, X) = \Phi(X)^T \Phi(X) \)
Random Kitchen Sinks and Gram Matrices

- How fast do we approach the exact Gram matrix?
- \( k(\mathbf{X}, \mathbf{X}) = \Phi(\mathbf{X})^T \Phi(\mathbf{X}) \)
Random Kitchen Sinks and Gram Matrices

- How fast do we approach the exact Gram matrix?
- $k(X, X) = \Phi(X)^T\Phi(X)$
Random Kitchen Sinks and Gram Matrices

- How fast do we approach the exact Gram matrix?
- $k(\mathbf{X}, \mathbf{X}) = \Phi(\mathbf{X})^T \Phi(\mathbf{X})$
Random Kitchen Sinks and Posterior

- How fast do we approach the exact Posterior?
Random Kitchen Sinks and Posterior

- How fast do we approach the exact Posterior?

\[ x \quad f(x) \]

4 features

\[ f(x) \]

\[ x \]

GP Posterior Mean

GP Posterior Uncertainty

Data
Random Kitchen Sinks and Posterior

- How fast do we approach the exact Posterior?

5 features

- $f(x)$

GP Posterior Mean
GP Posterior Uncertainty
Data
Random Kitchen Sinks and Posterior

• How fast do we approach the exact Posterior?

10 features

\[ f(x) \]

\( f(x) \) and its approximations using Random Kitchen Sinks. The graph shows the GP Posterior Mean, GP Posterior Uncertainty, and Data points.

- GP Posterior Mean
- GP Posterior Uncertainty
- Data
Random Kitchen Sinks and Posterior

- How fast do we approach the exact Posterior?

20 features

$\mathbf{x}$

$f(\mathbf{x})$

GP Posterior Mean
GP Posterior Uncertainty
Data

- Blue: GP Posterior Mean
- Purple: GP Posterior Uncertainty
- Diamond: Data
Random Kitchen Sinks and Posterior

• How fast do we approach the exact Posterior?

200 features
Random Kitchen Sinks and Posterior

- How fast do we approach the exact Posterior?

400 features
Kitchen Sinks in multiple dimensions

D dimensions, T random features, N datapoints

- First: Sample $T \times D$ random numbers: $Z \sim \mathcal{N}(0, \sigma^{-2})$
- each $\phi_j(x) = \exp(i[Zx]_j)$
- or: $\Phi(x) = \exp(iZx)$

Each feature $\phi_j(\cdot)$ is a sine, cos wave varying along the direction given by one row of $Z$, with varying periods.

Can be slow for many features in high dimensions. For example, computing $\Phi(x)$ is $O(NTD)$. 
But isn’t linear regression already fast?

D dimensions, T features, N Datapoints

- Computing features: $\Phi(x) = \frac{1}{\sqrt{T}} \exp(iZx)$
- Regression: $w = [\Phi(x)^T \Phi(x)]^{-1} \Phi(x)^T y$
- Train time complexity:
  $\mathcal{O}(DTN + T^3 + T^2N)$
  - Computing Features
  - Inverting Covariance Matrix
  - Multiplication

- Prediction: $y^* = w^T \Phi(x^*)$
- Test time complexity: $\mathcal{O}(DTN^* + T^2N^*)$
  - Computing Features
  - Multiplication

- For images, $D$ is often $> 10,000$. 
Random Outline


Random Kitchen Sinks (Rahimi and Recht, NIPS 2008)

Fastfood (Le et al., ICML 2013)
Main idea: Approximately compute $Zx$ quickly, by replacing $Z$ with some easy-to-compute matrices.

- Uses the Subsampled Randomized Hadamard Transform (SRHT) (Sarlós, 2006)
- Train time complexity: $\mathcal{O}(\log(D)TN + T^3 + T^2N)$
  - Computing Features
  - Inverting Feature Matrix
  - Multiplication

- Test time complexity: $\mathcal{O}(\log(D)TN^* + T^2N^*)$
  - Computing Features
  - Multiplication

- For images, if $D = 10,000$, $\log_{10}(D) = 4$. 

---

**Fastfood (Q. Le, T. Sarlós, A. Smola, 2013)**
Main idea: Compute $Vx \approx Zx$ quickly, by building $V$ out of easy-to-compute matrices. $V$ has similar properties to the Gaussian matrix $Z$.

$$V = \frac{1}{\sigma \sqrt{d}} SHG\Pi HB$$

- $\Pi$ is a $D \times D$ permutation matrix
- $G$ is diagonal random Gaussian.
- $B$ is diagonal random $\{+1, -1\}$.
- $S$ is diagonal random scaling.
- $H$ is Walsh-Hadamard Matrix.
Main idea: Compute $Vx \approx Zx$ quickly, by building $V$ out of easy-to-compute matrices.

$$V = \frac{1}{\sigma \sqrt{d}} SHG \Pi HB$$  \hspace{1cm} (2)

- $H$ is Walsh-Hadamard Matrix. Multiplication in $\mathcal{O}(T \log(D))$. $H$ is orthogonal, can be built recursively.

$$H_3 = \frac{1}{2^3 \cdot 2}$$

\begin{align*}
H_3 &= \left(\begin{array}{cccccccccc}
1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\
1 & -1 & 1 & -1 & 1 & -1 & 1 & -1 & 1 \\
1 & 1 & -1 & -1 & 1 & 1 & -1 & -1 & 1 \\
1 & -1 & -1 & -1 & 1 & 1 & -1 & -1 & 1 \\
1 & 1 & 1 & 1 & -1 & -1 & -1 & -1 & 1 \\
1 & -1 & 1 & -1 & -1 & -1 & -1 & -1 & 1 \\
1 & 1 & -1 & -1 & -1 & -1 & -1 & -1 & 1 \\
1 & -1 & -1 & -1 & -1 & -1 & -1 & -1 & 1 \\
1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\
1 & -1 & -1 & -1 & -1 & -1 & -1 & -1 & 1 \\
1 & 1 & -1 & -1 & -1 & -1 & -1 & -1 & 1 \\
1 & -1 & -1 & -1 & -1 & -1 & -1 & -1 & 1 \\
1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\
1 & -1 & -1 & -1 & -1 & -1 & -1 & -1 & 1 \\
1 & 1 & -1 & -1 & -1 & -1 & -1 & -1 & 1 \\
1 & -1 & -1 & -1 & -1 & -1 & -1 & -1 & 1 \\
1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\
1 & -1 & -1 & -1 & -1 & -1 & -1 & -1 & 1 \\
1 & 1 & -1 & -1 & -1 & -1 & -1 & -1 & 1 \\
1 & -1 & -1 & -1 & -1 & -1 & -1 & -1 & 1 \\
1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1
\end{array}\right) \hspace{1cm} (3)
\end{align*}
Main idea: Compute $Vx \approx Zx$ quickly, by building $V$ out of easy-to-compute matrices.

$$Vx = \frac{1}{\sigma \sqrt{d}} SHG \Pi HBx$$

(4)

Intuition: Scramble a single Gaussian random vector many different ways, to create a matrix with similar properties to $Z$. [Draw on board]

- $HG \Pi HB$ produces pseudo-random Gaussian vectors (of identical length)
- $S$ fixes the lengths to have the correct distribution.
## Fastfood results

**Computing $Vx$:**

<table>
<thead>
<tr>
<th>$d$</th>
<th>$T$</th>
<th>Fastfood</th>
<th>RKS</th>
<th>Speedup</th>
<th>RAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>1024</td>
<td>16384</td>
<td>0.00058s</td>
<td>0.0139s</td>
<td>24x</td>
<td>256x</td>
</tr>
<tr>
<td>4096</td>
<td>32768</td>
<td>0.00136s</td>
<td>0.1224s</td>
<td>90x</td>
<td>1024x</td>
</tr>
<tr>
<td>8192</td>
<td>65536</td>
<td>0.00268s</td>
<td>0.5360s</td>
<td>200x</td>
<td>2048x</td>
</tr>
</tbody>
</table>

We never store $V$!
## Fastfood results

Regression MSE:

<table>
<thead>
<tr>
<th>Dataset</th>
<th>$m$</th>
<th>$d$</th>
<th>Exact RBF</th>
<th>Nystrom RBF</th>
</tr>
</thead>
<tbody>
<tr>
<td>Insurance Company</td>
<td>5,822</td>
<td>85</td>
<td>0.231</td>
<td>0.232</td>
</tr>
<tr>
<td>Wine Quality</td>
<td>4,080</td>
<td>11</td>
<td>0.819</td>
<td>0.797</td>
</tr>
<tr>
<td>Parkinson Telemonitor</td>
<td>4,700</td>
<td>21</td>
<td>0.059</td>
<td>0.058</td>
</tr>
<tr>
<td>CPU</td>
<td>6,554</td>
<td>21</td>
<td>7.271</td>
<td>6.758</td>
</tr>
<tr>
<td>Location of CT slices (axial)</td>
<td>42,800</td>
<td>384</td>
<td>n.a.</td>
<td>60.683</td>
</tr>
<tr>
<td>KEGG Metabolic Network</td>
<td>51,686</td>
<td>27</td>
<td>n.a.</td>
<td>17.872</td>
</tr>
<tr>
<td>Year Prediction MSD</td>
<td>463,715</td>
<td>90</td>
<td>n.a.</td>
<td>0.113</td>
</tr>
<tr>
<td>Forest</td>
<td>522,910</td>
<td>54</td>
<td>n.a.</td>
<td>0.837</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Random Kitchen Sinks (RBF)</th>
<th>Fastfood FFT</th>
<th>Fastfood RBF</th>
<th>Exact Matern</th>
<th>Fastfood Matern</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>0.266</td>
<td>0.266</td>
<td>0.234</td>
<td>0.235</td>
</tr>
<tr>
<td></td>
<td>0.740</td>
<td>0.721</td>
<td>0.753</td>
<td>0.720</td>
</tr>
<tr>
<td></td>
<td>0.054</td>
<td>0.052</td>
<td>0.053</td>
<td>0.052</td>
</tr>
<tr>
<td></td>
<td>7.103</td>
<td>4.544</td>
<td>4.345</td>
<td>4.211</td>
</tr>
<tr>
<td></td>
<td>49.491</td>
<td>58.425</td>
<td>43.858</td>
<td>n.a.</td>
</tr>
<tr>
<td></td>
<td>17.837</td>
<td>17.826</td>
<td>17.818</td>
<td>n.a.</td>
</tr>
<tr>
<td></td>
<td>0.123</td>
<td>0.106</td>
<td>0.115</td>
<td>n.a.</td>
</tr>
<tr>
<td></td>
<td>0.840</td>
<td>0.838</td>
<td>0.840</td>
<td>n.a.</td>
</tr>
</tbody>
</table>
“The Trouble with Kernels” (Smola)

- Kernel Expansion: \( f(x) = \sum_{i=1}^{m} \alpha_i k(x_i, x) \)
- Feature Expansion: \( f(x) = \sum_{i=1}^{T} w_i \phi_i(x) \)

D dimensions, T features, N samples

<table>
<thead>
<tr>
<th>Method</th>
<th>Train Time</th>
<th>Test Time</th>
<th>Train Mem</th>
<th>Test Mem</th>
</tr>
</thead>
<tbody>
<tr>
<td>Naive</td>
<td>( \mathcal{O}(N^2D) )</td>
<td>( \mathcal{O}(ND) )</td>
<td>( \mathcal{O}(ND) )</td>
<td>( \mathcal{O}(ND) )</td>
</tr>
<tr>
<td>Low Rank</td>
<td>( \mathcal{O}(NTD) )</td>
<td>( \mathcal{O}(TD) )</td>
<td>( \mathcal{O}(TD) )</td>
<td>( \mathcal{O}(TD) )</td>
</tr>
<tr>
<td>Kitchen Sinks</td>
<td>( \mathcal{O}(NTD) )</td>
<td>( \mathcal{O}(TD) )</td>
<td>( \mathcal{O}(TD) )</td>
<td>( \mathcal{O}(TD) )</td>
</tr>
<tr>
<td>Fastfood</td>
<td>( \mathcal{O}(NT \log(D)) )</td>
<td>( \mathcal{O}(T \log(D)) )</td>
<td>( \mathcal{O}(T \log(D)) )</td>
<td>( \mathcal{O}(T) )</td>
</tr>
</tbody>
</table>

“Can run on a phone”
Feature transforms of more interesting kernels

In high dimensions, all Euclidian distances become the same, unless data lie on manifold. Usually need structured kernel.

- can do any stationary kernel (Matérn, rational-quadratic) with Fastfood
- sum of kernels is concatenation of features:
  \[ k^{(+)}(x, x') = k^{(1)}(x, x') + k^{(2)}(x, x') \implies \Phi^{(+)} = \begin{bmatrix} \Phi^{(1)}(x) \\ \Phi^{(2)}(x) \end{bmatrix} \]
- product of kernels is outer product of features:
  \[ k^{(\times)}(x, x') = k^{(1)}(x, x')k^{(2)}(x, x') \implies \Phi^{(\times)}(x) = \Phi^{(1)}(x)\Phi^{(2)}(x)^T \]

For example, can build translation-invariant kernel:

\[ f\begin{pmatrix} \mathbf{1} \end{pmatrix} = f\begin{pmatrix} \mathbf{1} \end{pmatrix} \quad (5) \]

\[ k(((x_1, x_2, \ldots, x_D), (x'_1, x'_2, \ldots, x'_D)) = \sum_{i=1}^{D} \prod_{j=1}^{D} k(x_j, x'_{i+j \text{mod } D}) \quad (6) \]
## Takeaways

### Random Projections
- Preserve Euclidian distances
- while reducing dimensionality
- Allow for nonlinear mappings

### Random Features
- RKS can approximate GP posterior quickly
- Fastfood can compute $T$ nonlinear basis functions in $O(T \log D)$ time.
- Can operate on structured kernels.

---

Gourmet cuisine (exact inference) is nice, but often fastfood is good enough.