Random Projections

Lopez–Paz & Duvenaud

November 7, 2013

Random Outline

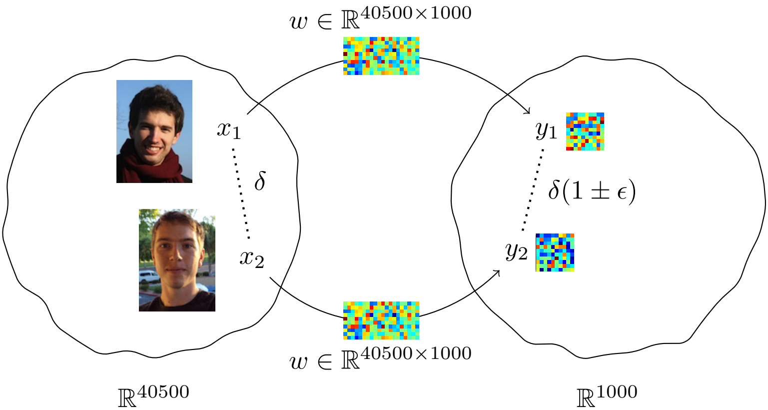
The Johnson-Lindenstrauss Lemma (1984)

Random Kitchen Sinks (Rahimi and Recht, NIPS 2008)

Fastfood (Le et al., ICML 2013)

Why random projections?

Fast, efficient and & distance-preserving **dimensionality reduction**!

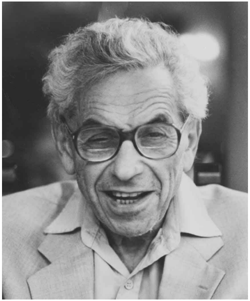


$$(1-\epsilon)\|x_1 - x_2\|^2 \leq \|y_1 - y_2\|^2 \leq (1+\epsilon)\|x_1 - x_2\|^2$$

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

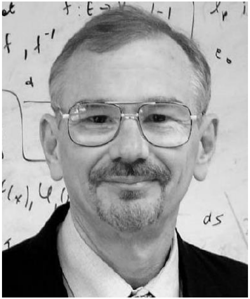
The Johnson-Lindenstrauss Lemma

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



Paul Erdös
1913-1996

Joram Lindenstrauss
1936-2012



William B. Johnson
1944-

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

Auxiliary Lemma 1

Let Q be a random variable following a χ^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 χ^2 distribution, $\lambda < \frac{1}{2}$.

To tight 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 $1+\epsilon \le e^{\epsilon-(\epsilon^2-\epsilon^3)/2}$ yields

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

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

$$\begin{aligned} \Pr[\overline{(1-\epsilon)k \le Q \le (1+\epsilon)k}] \le \\ \Pr[Q \ge (1+\epsilon)k \cup Q \le (1-\epsilon)k] \le \\ \Pr[Q \ge (1+\epsilon)k] + \Pr[Q \le (1-\epsilon)k] = \\ 2e^{-\frac{k}{4}(\epsilon^2-\epsilon^3)} \end{aligned}$$

Then,

$$\Pr[(1-\epsilon)k \le Q \le (1+\epsilon)k] \ge 1 - 2e^{-\frac{k}{4}(\epsilon^2-\epsilon^3)}$$

$\square$

Auxiliary Lemma 2

Let $\boldsymbol{x} \in \mathbb{R}^N$, $k < N$ and $\boldsymbol{A} \in \mathbb{R}^{k\times N}$ with $A_{ij} \sim \mathcal{N}(0,1)$. Then, for any $0 \leq \epsilon \leq 1/2$:

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

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

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

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?

Auxiliary Lemma 2

Remember: $\hat{\boldsymbol{x}} = \boldsymbol{A}\boldsymbol{x}$, $T_j = \hat{x}_j/\|\boldsymbol{x}\| \sim \mathcal{N}(0,1)$, $Q = \sum_i^k T_j^2 \sim \chi_k^2$:

$$\begin{aligned}
\Pr[(1-\epsilon)\|\boldsymbol{x}\|^2 \leq \|\frac{1}{\sqrt{k}}\boldsymbol{A}\boldsymbol{x}\|^2 \leq (1+\epsilon)\|\boldsymbol{x}\|^2] &= \\
\Pr[(1-\epsilon)\|\boldsymbol{x}\|^2 \leq \frac{\|\hat{\boldsymbol{x}}\|^2}{k} \leq (1+\epsilon)\|\boldsymbol{x}\|^2] &= \\
\Pr[(1-\epsilon)k \leq \frac{\|\hat{\boldsymbol{x}}\|^2}{\|\boldsymbol{x}\|^2} \leq (1+\epsilon)k] &= \\
\Pr\left[(1-\epsilon)k \leq \sum_i^k T_j^2 \leq (1+\epsilon)k\right] &= \\
\Pr\left[(1-\epsilon)k \leq Q \leq (1+\epsilon)k\right] &\geq \\
1 - 2e^{-(\epsilon^2-\epsilon^3)k/4} &
\end{aligned}$$

$\square$

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 $\mathbb{R}^N$ $\exists$ $f : \mathbb{R}^N \to \mathbb{R}^k$ s.t. $\forall$ $\boldsymbol{u}, \boldsymbol{v} \in V$:

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

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

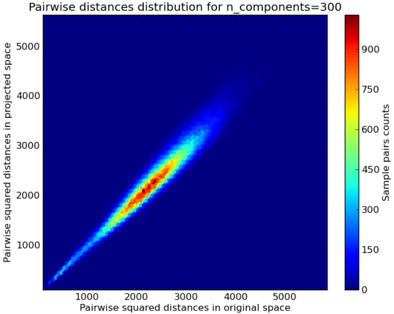
- For fixed $\boldsymbol{u}, \boldsymbol{v} \in V$, we apply the previous lemma with $\boldsymbol{x} = \boldsymbol{u} - \boldsymbol{v}$ to lower bound the *success* probability by $1 - 2e^{-(\epsilon^2 - \epsilon^3)k/4}$.
- Union bound again! This time over the m^2 pairs in V with $k = \frac{20 \log m}{\epsilon^2}$ and $\epsilon < 1/2$ to obtain:

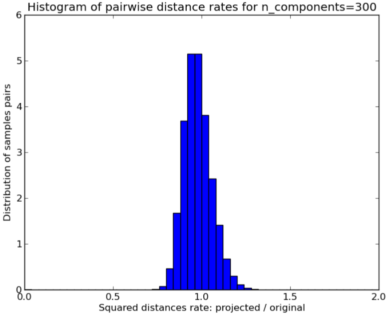
$$\Pr[success] \geq 1 - 2m^2 e^{-(\epsilon^2-\epsilon^3)k/4} = 1 - 2m^{5\epsilon-3} > 1 - 2m^{-1/2} > 0.$$

$\square$

JL Experiments

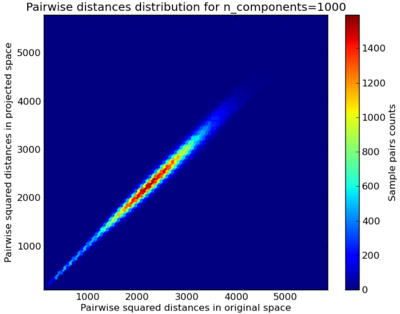
Data: 20-newsgroups, from 100.000 features to 300 (0.3%)

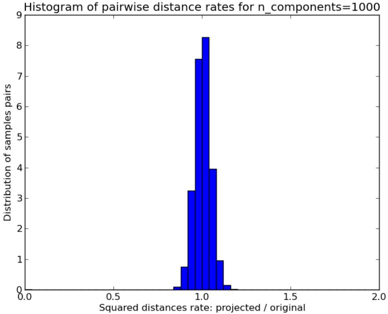




JL Experiments

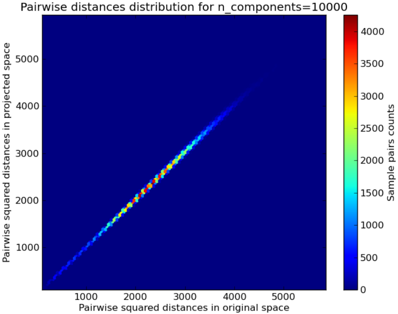
Data: 20-newsgroups, from 100.000 features to 1.000 (1%)

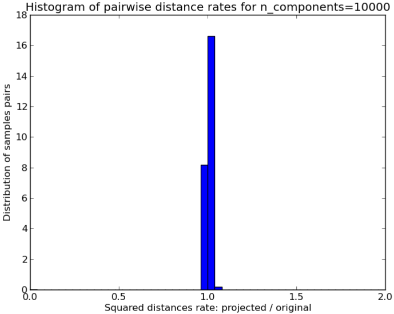




JL Experiments

Data: 20-newsgroups, from 100.000 features to 10.000 (10%)





MATLAB implementation: `1/sqrt(k).*randn(k,N)%*%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)
- Code for previous experiments: `http://bit.ly/17FTfbH`

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

Ali Rahimi

Ben Recht

Random Outline

The Johnson-Lindenstrauss Lemma (1984)

Random Kitchen Sinks (Rahimi and Recht, NIPS 2008)

Fastfood (Le et al., ICML 2013)

A Familiar Creature

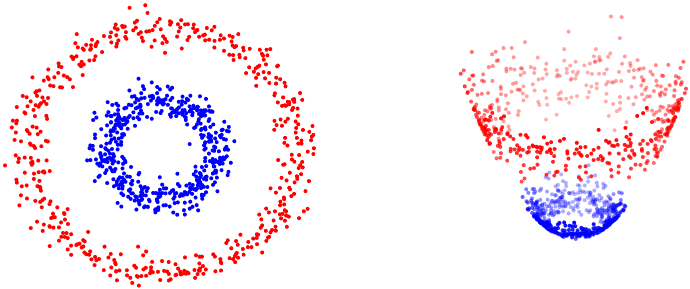
$$f(\boldsymbol{x}) = \sum_{i=1}^{T} \alpha_i \phi(\boldsymbol{x}; \boldsymbol{w}_i)$$

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

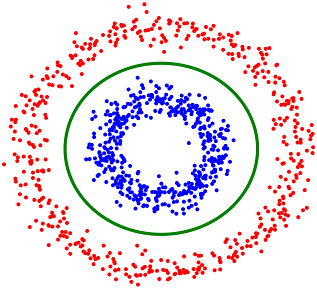
Same model, different training approaches!

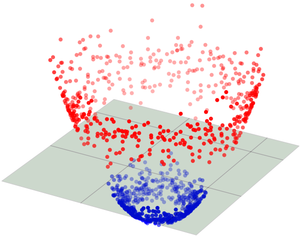
Things get interesting when ϕ is non-linear...

A Familiar Solution

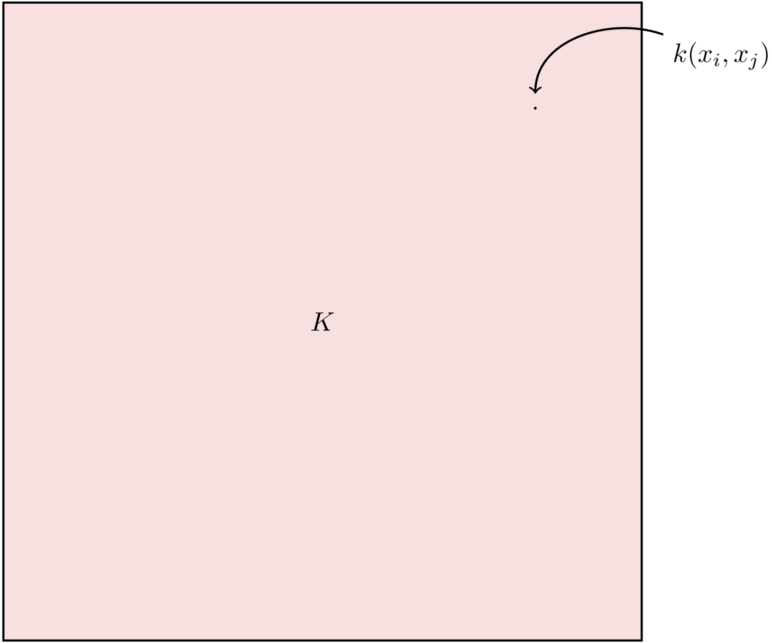


A Familiar Solution





A Familiar Monster: The Kernel Trap



Greedy Approximation of Functions

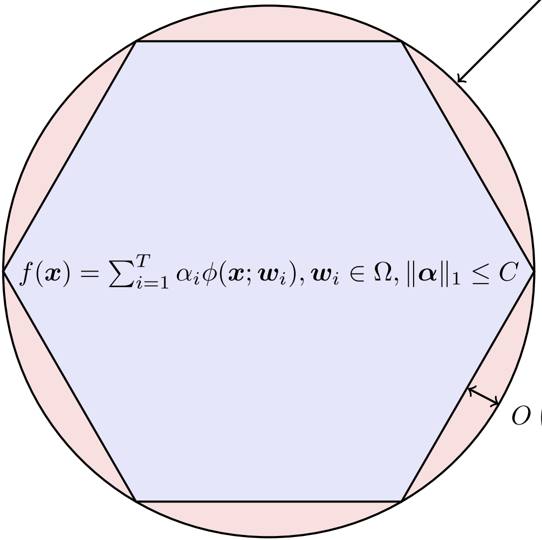
Approx. $f(\boldsymbol{x}) = \sum_{i=1}^{\infty} \alpha_i \phi(\boldsymbol{x}; \boldsymbol{w}_i)$ with $f_T(\boldsymbol{x}) = \sum_{i=1}^{T} \alpha_i \phi(\boldsymbol{x}; \boldsymbol{w}_i)$.

Greedy Fitting

$$(\boldsymbol{\alpha}^\star, \boldsymbol{W}^\star) = \min_{\boldsymbol{\alpha}, \boldsymbol{W}} \left\| \sum_{i=1}^{T} \alpha_i \phi(; \boldsymbol{w}_i) - f \right\|_\mu$$

Greedy Approximation of Functions

$$\mathcal{F} \equiv \{f(\boldsymbol{x}) = \sum_{i=1}^{\infty} \alpha_i \phi(\boldsymbol{x}; \boldsymbol{w}_i), \boldsymbol{w}_i \in \Omega, \|\boldsymbol{\alpha}\|_1 \leq C\}$$



$$\|f_T - f\|_\mu = \sqrt{\int_{\mathcal{X}} (f_T(x) - f(x))^2 \mu(dx)} = O\left(\frac{C}{\sqrt{T}}\right)$$ (Jones, 1992)

RKS Approximation of Functions

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

Greedy Fitting

$$(\boldsymbol{\alpha}^\star, \boldsymbol{W}^\star) = \min_{\boldsymbol{\alpha}, \boldsymbol{W}} \left\| \sum_{i=1}^{T} \alpha_i \phi(; \boldsymbol{w}_i) - f \right\|_\mu$$

Random Kitchen Sinks Fitting

$$\boldsymbol{w}_i^\star, \ldots, \boldsymbol{w}_T^\star \sim p(\boldsymbol{w}), \quad \boldsymbol{\alpha}^\star = \min_{\boldsymbol{\alpha}} \left\| \sum_{i=1}^{T} \alpha_i \phi(; \boldsymbol{w}_i^\star) - f \right\|_\mu$$

Just an old idea?

REVIEWS OF MODERN PHYSICS VOLUME 34, NUMBER 1 JANUARY, 1962

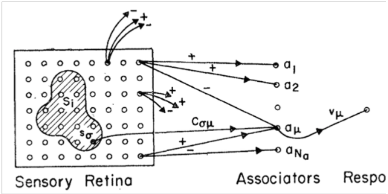
The Perceptron: A Model for Brain Functioning. I*

H. D. Block

Cornell University, Ithaca, New York

THE Perceptron is a self-organizing or adaptive system proposed by Rosenblatt.[1] 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.

W. Maasss, T. Natschlaeger, H. Markram, Real-time computing without stable states: A new framework for neural computation based on perturbations, *Neural Computation*. **14**(11), 2531-2560, (2002)

How does RKS work?

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

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

and let $\mathcal{F}_p$ be all f with $\|f\|_p \leq C$. Then, for $\boldsymbol{w}_1, \ldots, \boldsymbol{w}_T \sim p(\boldsymbol{w})$, w.p. at least $1-\delta$, $\delta > 0$, there exist some $\boldsymbol{\alpha}$ s.t. f_T satisfies:

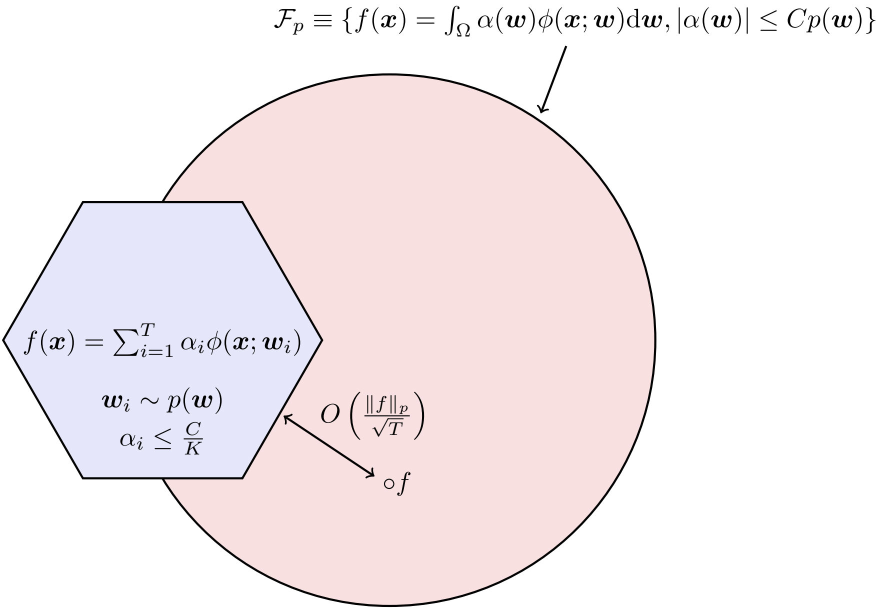
$$\|f_T - f\|_\mu = O\left(\frac{C}{\sqrt{T}}\sqrt{1 + 2\log\frac{1}{\delta}}\right)$$

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

$$f_T(\boldsymbol{x}) = \frac{1}{T}\sum_i^T a_i(\boldsymbol{w}_i)\phi(\boldsymbol{x};\boldsymbol{w}_i)$$

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

RKS Approximation of Functions



Random Kitchen Sinks: Auxiliary Lemma

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

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

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

$$|f(\boldsymbol{X}) - f(\tilde{\boldsymbol{X}})| \leq \|\bar{\boldsymbol{X}} - \bar{\tilde{\boldsymbol{X}}}\| \leq \frac{\|x_i - x_i'\|}{K} \leq \frac{2C}{K}.$$

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

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

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

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

Fourth, use McDiarmid's inequality and rearrange.

Random Kitchen Sinks Proof

Let μ be a measure on $\mathcal{X}$, and $f^* \in \mathcal{F}_p$. Let $\boldsymbol{w}_1, \ldots, \boldsymbol{w}_T \sim p(\boldsymbol{w})$. Then, w.p. at least $1-\delta$, with $\delta > 0$, $\exists\ f_T(x) = \sum_i^T \beta_i \phi(\boldsymbol{x}; \boldsymbol{w}_i)$ s.t.:

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

Proof:

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

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

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 $k(\boldsymbol{x}-\boldsymbol{y})$ on $\mathbb{R}^d$ is PSD if and only if $k(\boldsymbol{x}-\boldsymbol{y})$ is the Fourier transform of a non-negative measure $p(\boldsymbol{w})$.

$$\begin{aligned}
k(\boldsymbol{x}-\boldsymbol{y}) &= \int_{\mathbb{R}^d} p(\boldsymbol{w}) e^{j\boldsymbol{w}'(\boldsymbol{x}-\boldsymbol{y})} d\boldsymbol{w} \\
&\approx \frac{1}{T}\sum_{i=1}^{T} e^{j\boldsymbol{w}_i'(\boldsymbol{x}-\boldsymbol{y})} \quad \text{(Monte-Carlo, } O(T^{-1/2})\text{)} \\
&= \frac{1}{T}\sum_{i=1}^{T} \underbrace{e^{j\boldsymbol{w}_i'\boldsymbol{x}}}_{\boldsymbol{\phi}(\boldsymbol{x};\boldsymbol{W})} \underbrace{e^{-j\boldsymbol{w}_i'\boldsymbol{y}}}_{\boldsymbol{\phi}(\boldsymbol{y};\boldsymbol{W})} \\
&= \frac{1}{\sqrt{T}}\boldsymbol{\phi}(\boldsymbol{x};\boldsymbol{W})^* \frac{1}{\sqrt{T}}\boldsymbol{\phi}(\boldsymbol{y};\boldsymbol{W})
\end{aligned}$$

Now solve least squares in the primal in $O(n)$ time!

Random Kitchen Sinks: Implementation

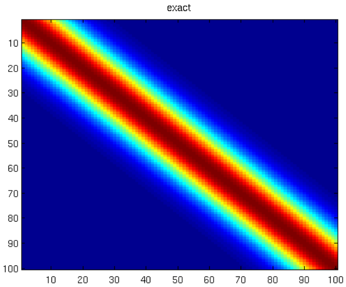
```
 % Training
function ytest = kitchen_sinks( X, y, Xtest, T, noise)

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

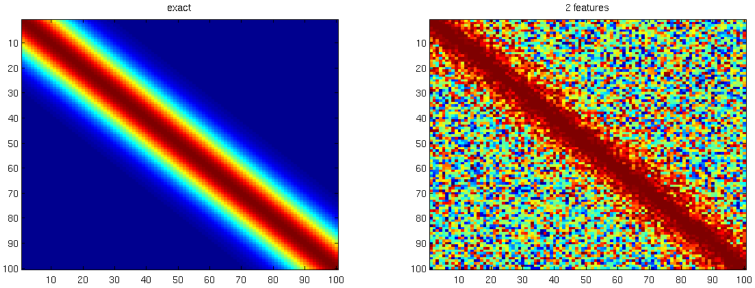
Random Kitchen Sinks and Gram Matrices

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



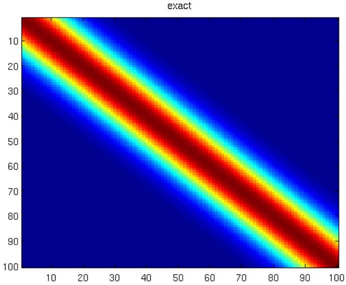
Random Kitchen Sinks and Gram Matrices

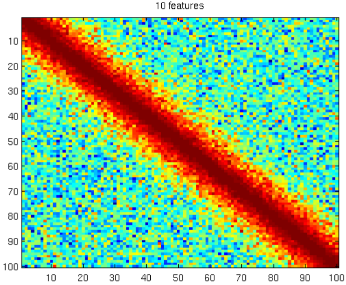
- How fast do we approach the exact Gram matrix?
- $k(\mathbf{X}, \mathbf{X}) = \Phi(\mathbf{X})^\mathsf{T}\Phi(\mathbf{X})$



Random Kitchen Sinks and Gram Matrices

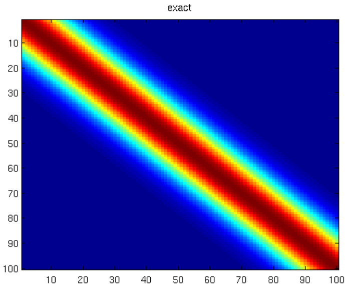
- How fast do we approach the exact Gram matrix?
- $k(\mathbf{X}, \mathbf{X}) = \Phi(\mathbf{X})^\top \Phi(\mathbf{X})$

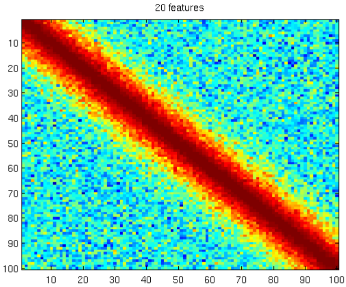




Random Kitchen Sinks and Gram Matrices

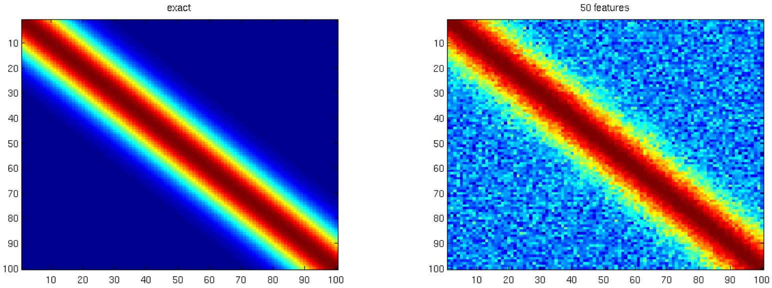
- How fast do we approach the exact Gram matrix?
- $k(\mathbf{X}, \mathbf{X}) = \Phi(\mathbf{X})^{\mathsf{T}} \Phi(\mathbf{X})$





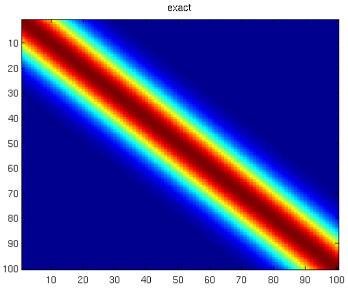
Random Kitchen Sinks and Gram Matrices

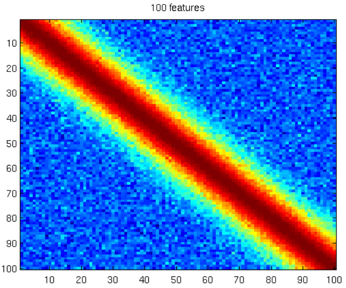
- How fast do we approach the exact Gram matrix?
- $k(\mathbf{X}, \mathbf{X}) = \Phi(\mathbf{X})^\top \Phi(\mathbf{X})$



Random Kitchen Sinks and Gram Matrices

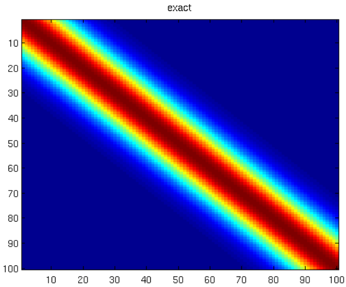
- How fast do we approach the exact Gram matrix?
- $k(\mathbf{X}, \mathbf{X}) = \Phi(\mathbf{X})^{\mathsf{T}}\Phi(\mathbf{X})$

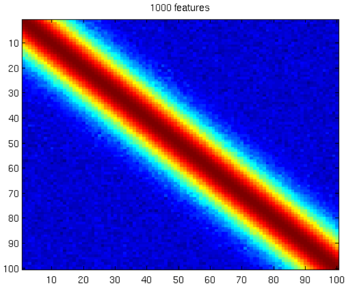




Random Kitchen Sinks and Gram Matrices

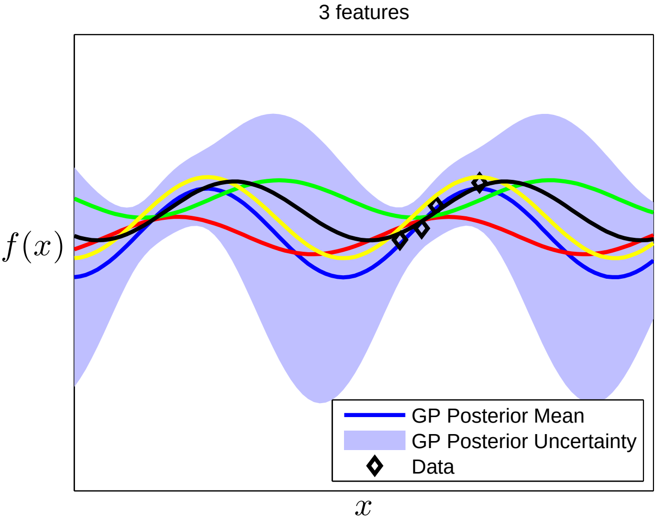
- How fast do we approach the exact Gram matrix?
- $k(\mathbf{X}, \mathbf{X}) = \Phi(\mathbf{X})^\top \Phi(\mathbf{X})$





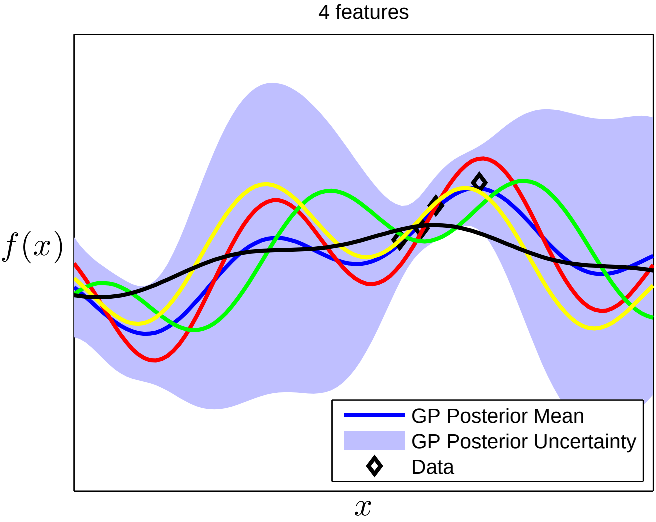
Random Kitchen Sinks and Posterior

- How fast do we approach the exact Posterior?



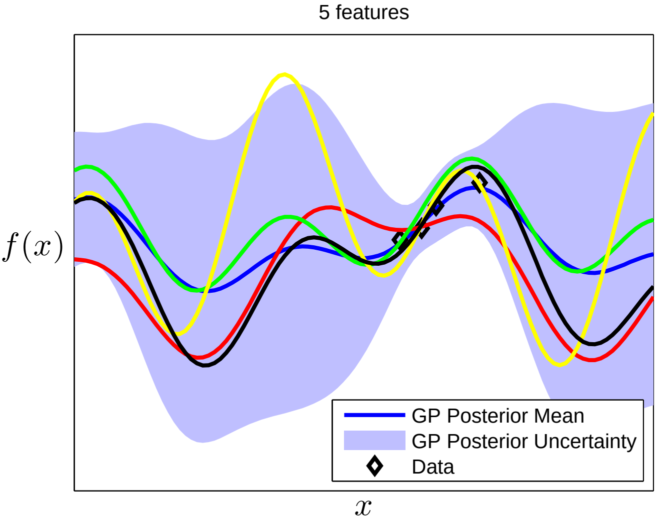
Random Kitchen Sinks and Posterior

- How fast do we approach the exact Posterior?



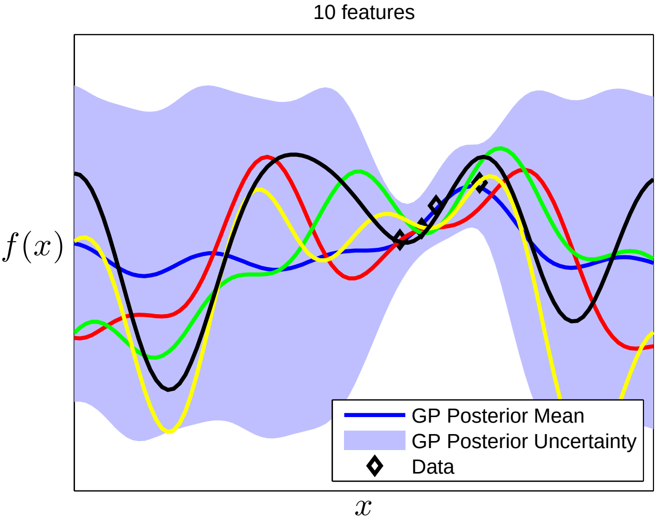
Random Kitchen Sinks and Posterior

- How fast do we approach the exact Posterior?



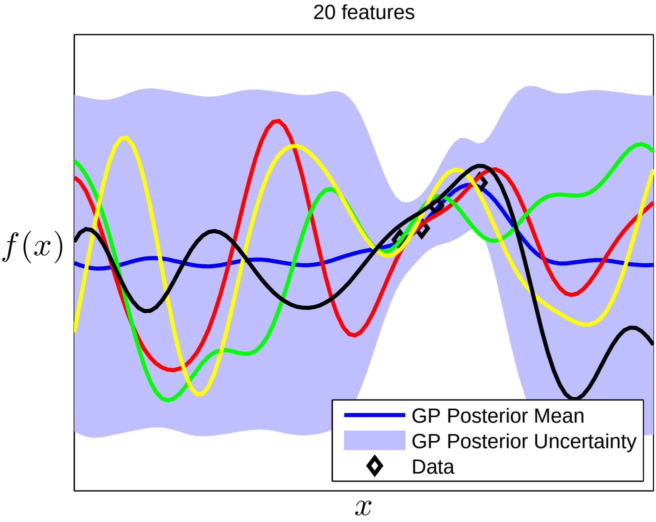
Random Kitchen Sinks and Posterior

- How fast do we approach the exact Posterior?



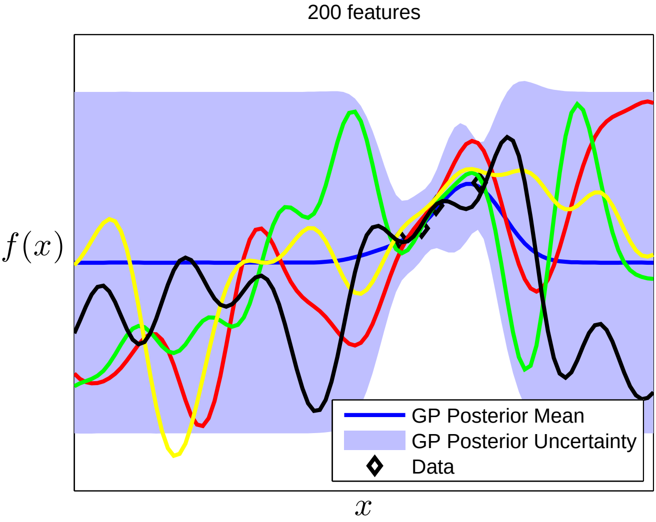
Random Kitchen Sinks and Posterior

- How fast do we approach the exact Posterior?



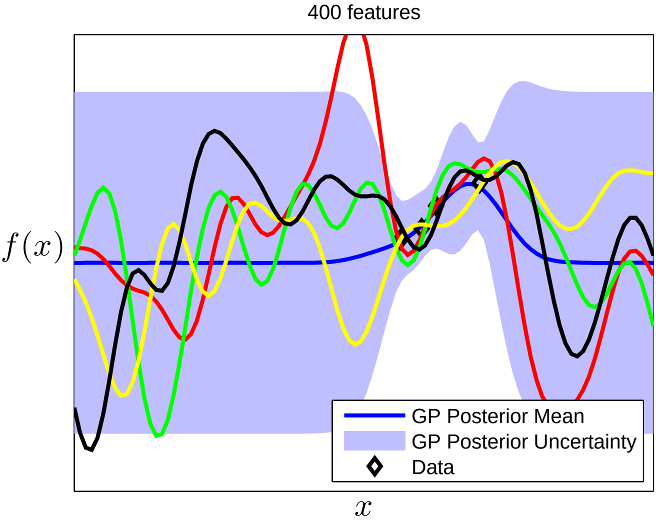
Random Kitchen Sinks and Posterior

- How fast do we approach the exact Posterior?



Random Kitchen Sinks and Posterior

- How fast do we approach the exact Posterior?

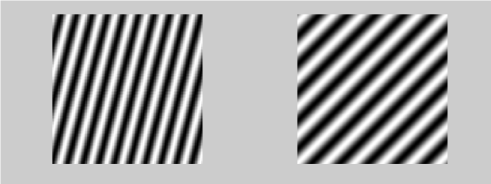


Kitchen Sinks in multiple dimensions

D dimensions, T random features, N datapoints

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

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 $\Phi(\mathbf{x})$ is $\mathcal{O}(NTD)$.

But isn't linear regression already fast?

D dimensions, T features, N Datapoints

- Computing features: $\Phi(\mathbf{x}) = \frac{1}{\sqrt{T}} \exp(iZx)$
- Regression: $\mathbf{w} = \left[\Phi(\mathbf{x})^{\mathsf{T}} \Phi(\mathbf{x})\right]^{-1} \Phi(\mathbf{x})^{\mathsf{T}} \mathbf{y}$
- Train time complexity:
  $\mathcal{O}(\underbrace{DTN}_{\text{Computing Features}} + \underbrace{T^3}_{\text{Inverting Covariance Matrix}} + \underbrace{T^2 N}_{\text{Multiplication}})$
- Prediction: $\mathbf{y}^* = \mathbf{w}^{\mathsf{T}} \Phi(\mathbf{x}^*)$
- Test time complexity: $\mathcal{O}(\underbrace{DTN^*}_{\text{Computing Features}} + \underbrace{T^2 N^*}_{\text{Multiplication}})$
- For images, D is often $> 10,000$.

Random Outline

The Johnson-Lindenstrauss Lemma (1984)

Random Kitchen Sinks (Rahimi and Recht, NIPS 2008)

Fastfood (Le et al., ICML 2013)

Fastfood (Q. Le, T. Sarlós, A. Smola, 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:
  $$\mathcal{O}(\underbrace{log(D)TN}_{\text{Computing Features}} + \underbrace{T^3}_{\text{Inverting Feature Matrix}} + \underbrace{T^2N}_{\text{Multiplication}})$$
- Test time complexity: $\mathcal{O}(\underbrace{log(D)TN^*}_{\text{Computing Features}} + \underbrace{T^2N^*}_{\text{Multiplication}})$
- For images, if $D = 10,000$, $\log_{10}(D) = 4$.

Fastfood (Q. Le, T. Sarlós, A. Smola, 2013)

Main idea: Compute $Vx \approxeq Zx$ quickly, by building V out of easy-to-compute matrices.
V has similar properties to the Gaussian matrix Z.

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

- Π 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.

Fastfood (Q. Le, T. Sarlós, A. Smola, 2013)

Main idea: Compute $Vx \approxeq Zx$ quickly, by building V out of easy-to-compute matrices.

$$V = \frac{1}{\sigma\sqrt{d}} SHG\Pi HB \tag{2}$$

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

$$H_3 = \frac{1}{2^{\frac{3}{2}}} \begin{pmatrix} 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & -1 & 1 & -1 & 1 & -1 & 1 & -1 \\ 1 & 1 & -1 & -1 & 1 & 1 & -1 & -1 \\ 1 & -1 & -1 & 1 & 1 & -1 & -1 & 1 \\ 1 & 1 & 1 & 1 & -1 & -1 & -1 & -1 \\ 1 & -1 & 1 & -1 & -1 & 1 & -1 & 1 \\ 1 & 1 & -1 & -1 & -1 & -1 & 1 & 1 \\ 1 & -1 & -1 & 1 & -1 & 1 & 1 & -1 \end{pmatrix} \tag{3}$$

Fastfood (Q. Le, T. Sarlós, A. Smola, 2013)

Main idea: Compute $Vx \approxeq Zx$ quickly, by building V out of easy-to-compute matrices.

$$Vx = \frac{1}{\sigma\sqrt{d}} SHG\Pi HBx \tag{4}$$

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 pseduo-random Gaussian vectors (of identical length)
- S fixes the lengths to have the correct distribution.

Fastfood results

Computing Vx:

d	T	Fastfood	RKS	Speedup	RAM
1024	16384	0.00058s	0.0139s	24x	256x
4096	32768	0.00136s	0.1224s	90x	1024x
8192	65536	0.00268s	0.5360s	200x	2048x

We never store V!

Fastfood results

Regression MSE:

Dataset	m	d	Exact RBF	Nystrom RBF	Random Kitchen Sinks (RBF)	Fastfood FFT	Fastfood RBF	Exact Matern	Fastfood Matern
Insurance Company	5,822	85	0.231	0.232	0.266	0.266	0.264	0.234	0.235
Wine Quality	4,080	11	0.819	0.797	0.740	0.721	0.740	0.753	0.720
Parkinson Telemonitor	4,700	21	0.059	0.058	0.054	0.052	0.054	0.053	0.052
CPU	6,554	21	7.271	6.758	7.103	4.544	7.366	4.345	4.211
Location of CT slices (axial)	42,800	384	n.a.	60.683	49.491	58.425	43.858	n.a.	14.868
KEGG Metabolic Network	51,686	27	n.a.	17.872	17.837	17.826	17.818	n.a.	17.846
Year Prediction MSD	463,715	90	n.a.	0.113	0.123	0.106	0.115	n.a.	0.116
Forest	522,910	54	n.a.	0.837	0.840	0.838	0.840	n.a.	0.976

“The Trouble with Kernels” (Smola)

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

D dimensions, T features, N samples

Method	Train Time	Test Time	Train Mem	Test Mem
Naive	$\mathcal{O}(N^2 D)$	$\mathcal{O}(ND)$	$\mathcal{O}(ND)$	$\mathcal{O}(ND)$
Low Rank	$\mathcal{O}(NTD)$	$\mathcal{O}(TD)$	$\mathcal{O}(TD)$	$\mathcal{O}(TD)$
Kitchen Sinks	$\mathcal{O}(NTD)$	$\mathcal{O}(TD)$	$\mathcal{O}(TD)$	$\mathcal{O}(TD)$
Fastfood	$\mathcal{O}(NT\log(D))$	$\mathcal{O}(T\log(D))$	$\mathcal{O}(T\log(D))$	$\mathcal{O}(T)$

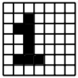
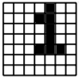
“Can run on a phone”

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:
  $k^{(+)}(\mathbf{x}, \mathbf{x}') = k^{(1)}(\mathbf{x}, \mathbf{x}') + k^{(2)}(\mathbf{x}, \mathbf{x}') \implies \Phi^{(+)} = \begin{bmatrix} \Phi^{(1)}(\mathbf{x}) \\ \Phi^{(2)}(\mathbf{x}) \end{bmatrix}$
- product of kernels is outer product of features:
  $k^{(\times)}(\mathbf{x}, \mathbf{x}') = k^{(1)}(\mathbf{x}, \mathbf{x}')k^{(2)}(\mathbf{x}, \mathbf{x}') \implies \Phi^{(\times)}(\mathbf{x}) = \Phi^{(1)}(\mathbf{x})\Phi^{(2)}(\mathbf{x})^\mathsf{T}$

For example, can build translation-invariant kernel:

$f\Bigg($  $\Bigg) = f\Bigg($  $\Bigg)$ (5)

$$k\left((x_1, x_2, \ldots, x_D), (x'_1, x'_2, \ldots, x'_D)\right) = \sum_{i=1}^{D} \prod_{j=1}^{D} k(x_j, x'_{i+j \bmod D}) \qquad (6)$$

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 $\mathcal{O}(T \log D)$ time.
- Can operate on structured kernels.

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