# Kernel Approximation Methods for Speech Recognition

Erfan Loweimi

Centre for Speech Technology Research (CSTR)

# Kernel Approximation Methods for Speech Recognition

Avner May[1†], Alireza Bagheri Garakani[2‡], Zhiyun Lu[2‡], Dong Guo[2‡], Kuan Liu[2‡], Aurélien Bellet[3], Linxi Fan[4], Michael Collins[1*], Daniel Hsu[1], Brian Kingsbury[5], Michael Picheny[5], Fei Sha[2]

[1]Dept. of Computer Science, Columbia University, New York, NY 10027, USA
{avnermay, mcollins, djhsu}@cs.columbia.edu, lf2422@columbia.edu

[2]Dept. of Computer Science, University of Southern California, Los Angeles, CA 90089, USA
{bagherig, zhiyunlu, dongguo, kuanl, feisha}@usc.edu

[3]INRIA, 40 Avenue Halley, 59650 Villeneuve d'Ascq, France
aurelien.bellet@inria.fr

[4]Dept. of Computer Science, Stanford University, Stanford, CA 94305, USA
jimfan@cs.stanford.edu

[5]IBM T. J. Watson Research Center, Yorktown Heights, NY 10598, USA
{bedk, picheny}@us.ibm.com

† ‡: *Contributed equally as the first and second co-authors, respectively*

# Kernel Approximation Methods for Speech Recognition

Avner May

Submitted in partial fulfillment of the
requirements for the degree
of Doctor of Philosophy
in the Graduate School of Arts and Sciences

**COLUMBIA UNIVERSITY**

2018

# A COMPARISON BETWEEN DEEP NEURAL NETS AND KERNEL ACOUSTIC MODELS FOR SPEECH RECOGNITION

*Zhiyun Lu*[1†] *Dong Guo*[2†] *Alireza Bagheri Garakani*[2†] *Kuan Liu*[2†]

*Avner May*[3‡] *Aurélien Bellet*[4‡] *Linxi Fan*[2]

*Michael Collins*[3*] *Brian Kingsbury*[5] *Michael Picheny*[5] *Fei Sha*[1]

[1]U. of California (Los Angeles) [2]U. of Southern California [3]Columbia U.
[4]Team Magnet, INRIA Lille - Nord Europe [5]IBM T. J. Watson Research Center (USA)

† ‡: *contributed equally as the first and second co-authors, respectively*

# COMPACT KERNEL MODELS FOR ACOUSTIC MODELING VIA RANDOM FEATURE SELECTION

*Avner May*[⋆] *Michael Collins*[⋆1] *Daniel Hsu*[⋆] *Brian Kingsbury*[†]

[⋆] Department of Computer Science, Columbia University, New York, NY 10025, USA
[†]IBM T. J. Watson Research Center, Yorktown Heights, NY 10598, USA

# Outlines

- Kernel Methods for Pattern Recognition
- How to Scale-up
- Application in ASR $\rightarrow$ Acoustic Modelling
- Novelties
- Experimental Results
- Conclusion

# Kernel Methods

# Kernel Methods for Pattern Recognition

- Advantages:
  - Handle Non-linear data, Interpretable, learning guarantees

# Kernel Methods for Pattern Recognition

- Advantages:
  - Handle Non-linear data, Interpretable, learning guarantees
- Representer Theorem

$$f^* = \underset{f}{argmin} \frac{1}{N} \sum_{n=1}^{N} L(y_n, f(x_n)) + \Phi(\|f\|^2)$$

# Kernel Methods for Pattern Recognition

- Advantages:
  - Handle Non-linear data, Interpretable, learning guarantees
- Representer Theorem

$$f^* = \underset{f}{argmin} \frac{1}{N} \sum_{n=1}^{N} L(y_n, f(x_n)) + \Phi(\|f\|^2)$$

$$f^*(x) = \sum_{n=1}^{N} \alpha_n \; K(x, x_n) \Big|_{K(x,x_n)=\langle \phi(x), \phi(x_n) \rangle}$$

# Kernel Methods for Pattern Recognition

- Advantages:
  - Handle Non-linear data, Interpretable, learning guarantees
- Representer Theorem

$$f^* = \underset{f}{argmin} \underbrace{\frac{1}{N}\sum_{n=1}^{N} L(y_n, f(x_n))}_{\text{Empirical Risk}} + \underbrace{\Phi(\|f\|^2)}_{\text{Regulariser}}$$

$$f^*(x) = \sum_{n=1}^{N} \alpha_n \, K(x, x_n)\Big|_{K(x,x_n)=\langle\phi(x),\phi(x_n)\rangle}$$

Kernel function: $K(x,x_n)$; Feature map: $\phi$

# Kernel Methods for Pattern Recognition

- Advantages:
  - Handle Non-linear data, Interpretable, learning guarantees
- Representer Theorem

Empirical Risk

Regulariser

$$f^* = \underset{f}{argmin} \frac{1}{N} \sum_{n=1}^{N} L(y_n, f(x_n)) + \Phi(\|f\|^2)$$

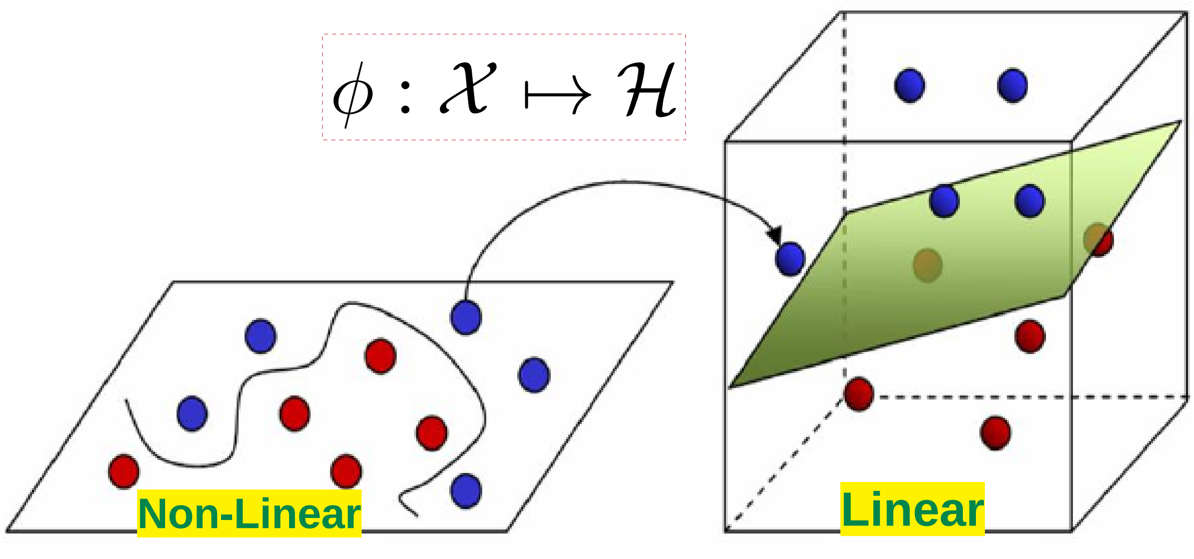
Kernel function

Feature map

$$f^*(x) = \sum_{n=1}^{N} \alpha_n K(x, x_n) \Big|_{K(x,x_n)=\langle \phi(x), \phi(x_n) \rangle}$$

Optimise for $\alpha$

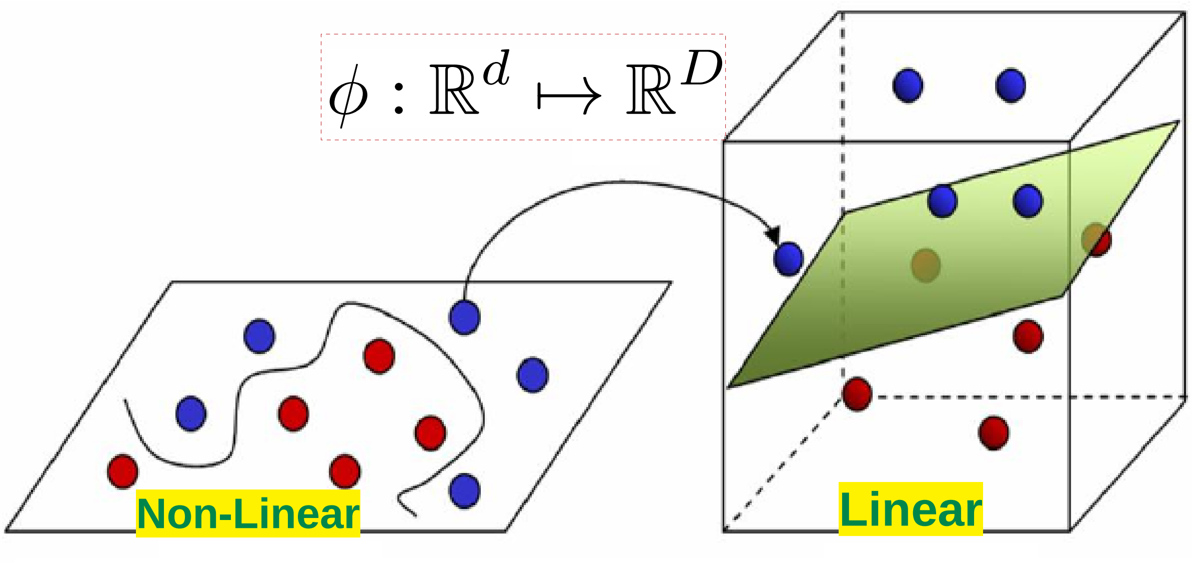
# Linearity in high-dimensional Space




Input Space, **X**　　　　Feature Space, ***ϕ(x)***

$$f(\mathbf{x}) = \sum_{n=1}^{N} \alpha_n \, K(\mathbf{x}, \mathbf{x}_n) = W^T \phi(\mathbf{x})$$

# Linearity in high-dimensional Space




Input Space, **x**

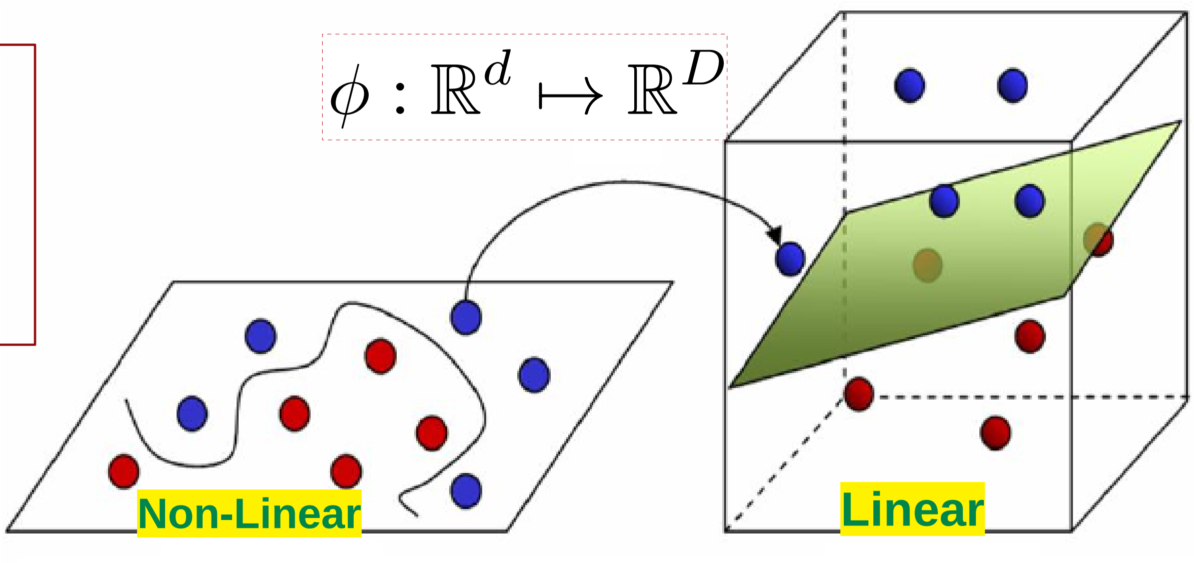
Feature Space, ***ϕ(x)***

$$f(\mathbf{x}) = \sum_{n=1}^{N} \alpha_n \, K(\mathbf{x}, \mathbf{x}_n) = W^T \phi(\mathbf{x})$$

# Linearity in high-dimensional Space

-- D may tend to ∞

-- Higher D => better linearly separability



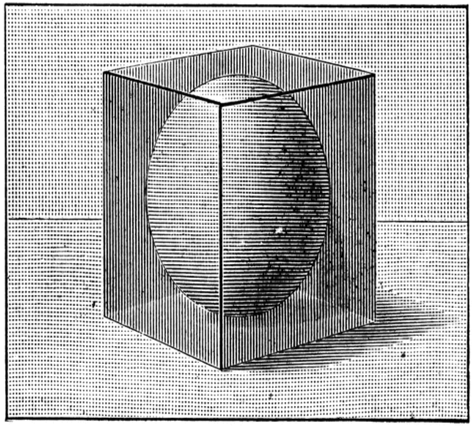

Input Space, **X**

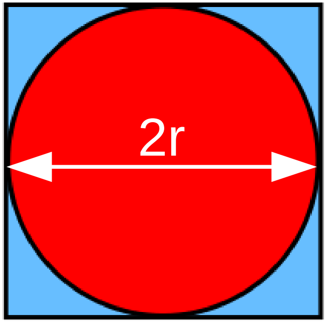
Feature Space, ***ϕ(x)***

$$f(\mathbf{x}) = \sum_{n=1}^{N} \alpha_n \, K(\mathbf{x}, \mathbf{x}_n) = W^T \phi(\mathbf{x})$$

# Linearity in high-dimensional Space

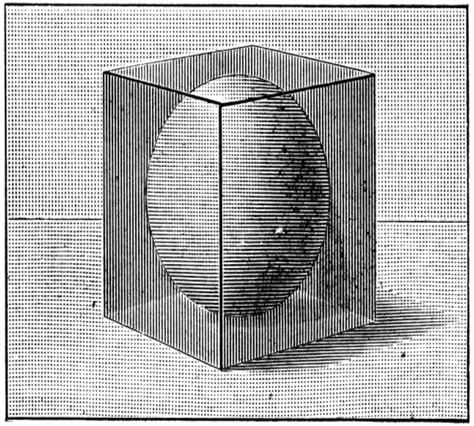
- $\mathbb{R}^{D=3}$
  - Hyper-cube Volume = $(2r)^3$
  - Hyper-sphere Volume = $\frac{4}{3}\pi r^3$

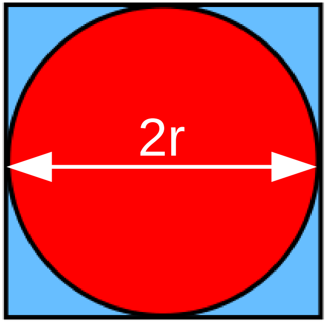




# Linearity in high-dimensional Space

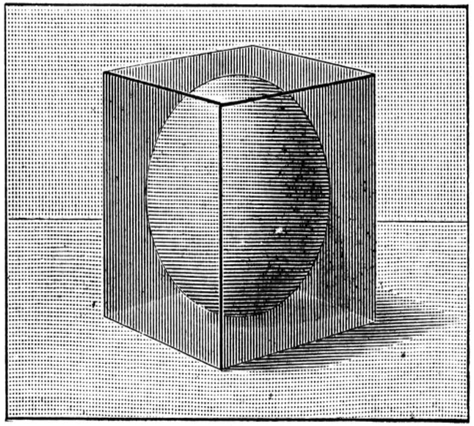
- $\mathbb{R}^{D=3}$
  - Hyper-cube Volume = $(2r)^3$
  - Hyper-sphere Volume = $\frac{4}{3}\pi r^3$

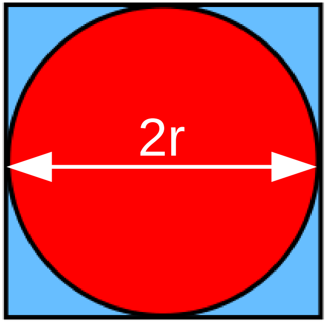




# Linearity in high-dimensional Space

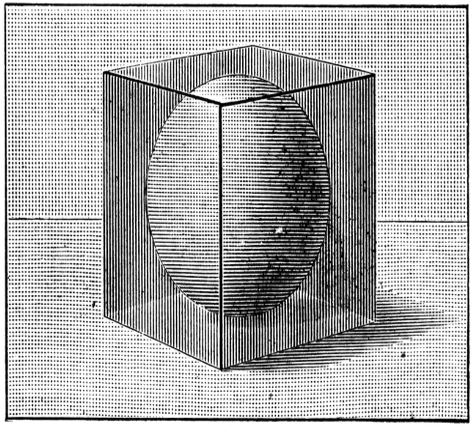
- $\mathbb{R}^{D \to \infty}$
  - Hyper-cube Volume = ?
  - Hyper-sphere Volume = ?

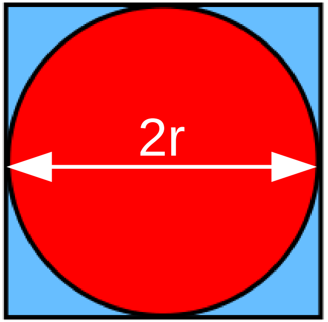




# Linearity in high-dimensional Space

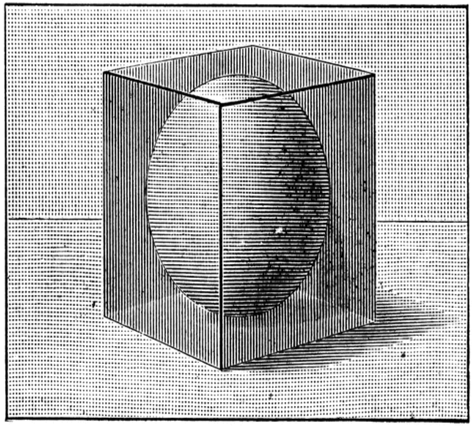
- $\mathbb{R}^{D \to \infty}$
  - Hyper-cube Volume $\to$ **∞**
  - Hyper-sphere Volume $\to$ **0**
  - Proof in Appendix 1

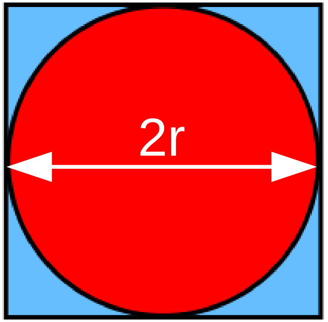




# Linearity in high-dimensional Space

- $\mathbb{R}^{D \rightarrow \infty}$
  - Hyper-cube Volume $\rightarrow$ **∞**
  - Hyper-sphere Volume $\rightarrow$ **0**
  - Proof in Appendix 1
- Linear separability ↑





# Kernel Trick

$$f(\mathbf{x}_i) = W^T \phi(\mathbf{x}_i) = \sum_{n=1}^{N} \alpha_n \phi^T(\mathbf{x}_i)\phi(\mathbf{x}_n)$$

# Kernel Trick

$$f(\mathbf{x}_i) = W^T\phi(\mathbf{x}_i) = \sum_{n=1}^{N} \alpha_n \phi^T(\mathbf{x}_i)\phi(\mathbf{x}_n)$$

Gaussian Kernel (RBF)

$$K(x,y) = \langle \phi(x), \phi(y) \rangle = \sum_{k=0}^{\infty} \phi_k(x)\phi_k(y) = exp(\|x-y\|^2)$$

$$\phi_k(x) = exp(-x^2)\sqrt{\frac{2^k}{k!}}x^k, \quad k = 0, 1, ..., \to \infty$$

Proof $\rightarrow$ Taylor Series Expansion

# Kernel Trick

$$f(\mathbf{x}_i) = W^T\phi(\mathbf{x}_i) = \sum_{n=1}^{N} \alpha_n \phi^T(\mathbf{x}_i)\phi(\mathbf{x}_n)$$

Gaussian Kernel (RBF)

$$K(x,y) = \langle\phi(x),\phi(y)\rangle = \sum_{k=0}^{\infty} \phi_k(x)\phi_k(y) = exp(\|x-y\|^2)$$

$$\phi_k(x) = exp(-x^2)\sqrt{\frac{2^k}{k!}}x^k, \quad k = 0, 1, ..., \to \infty$$

$$\phi : \mathcal{X} \mapsto \mathcal{H}$$

$$x \in \mathbb{R}^{d<\infty}$$

$$\phi(x) \in \mathbb{R}^{D\to\infty}$$

# Kernel Trick

$$\mathbb{R}^D (D \text{ may} \to \infty)$$

Go to H

dot prod

$$f(\mathbf{x}_i) = W^T \phi(\mathbf{x}_i) = \sum_{n=1}^{N} \alpha_n \phi^T(\mathbf{x}_i)\phi(\mathbf{x}_n)$$

$$\phi : \mathcal{X} \mapsto \mathcal{H}$$

$$x \in \mathbb{R}^{d<\infty}$$

$$\phi(x) \in \mathbb{R}^{D\to\infty}$$

# Kernel Trick

$\mathbb{R}^D (D \text{ may} \to \infty)$

Go to H

dot prod

Kernel Trick:
Bypass the inner product

$$f(\mathbf{x}_i) = W^T \phi(\mathbf{x}_i) = \sum_{n=1}^{N} \alpha_n \phi^T(\mathbf{x}_i)\phi(\mathbf{x}_n) = \sum_{n=1}^{N} \alpha_n K(\mathbf{x}_i, \mathbf{x}_n)$$

# Kernel Trick

$\mathbb{R}^D (D \text{ may} \to \infty)$

Go to H

dot prod

Kernel Trick:
Bypass the inner product

$$f(\mathbf{x}_i) = W^T \phi(\mathbf{x}_i) = \sum_{n=1}^{N} \alpha_n \phi^T(\mathbf{x}_i)\phi(\mathbf{x}_n) = \sum_{n=1}^{N} \alpha_n K(\mathbf{x}_i, \mathbf{x}_n)$$

$$K(x, y) = \langle \phi(x), \phi(y) \rangle = \sum_{k=0}^{\infty} \phi_k(x)\phi_k(y) = exp(\frac{\|x - y\|^2}{2\sigma^2})$$

RBF

# Kernel Trick

$$\mathbb{R}^D (D \text{ may} \to \infty)$$

Go to H

dot prod

Kernel Trick:
Bypass the inner product

$$f(\mathbf{x}_i) = W^T \phi(\mathbf{x}_i) = \sum_{n=1}^{N} \alpha_n \phi^T(\mathbf{x}_i)\phi(\mathbf{x}_n) = \sum_{n=1}^{N} \alpha_n K(\mathbf{x}_i, \mathbf{x}_n)$$

$$K(x, y) = \langle \phi(x), \phi(y) \rangle = \sum_{k=0}^{\infty} \phi_k(x)\phi_k(y) = exp(\frac{\|x - y\|^2}{2\sigma^2})$$

RBF

**Kernel Trick**: Instead of D (may D $\to \infty$) products/sums, simply use the kernel function K(x,y) to compute the inner product in H space.

# Kernel Trick

$\mathbb{R}^D (D \text{ may} \to \infty)$

Go to H

dot prod

Kernel Trick:
Bypass the inner product

$$f(\mathbf{x}_i) = W^T \phi(\mathbf{x}_i) = \sum_{n=1}^{N} \alpha_n \phi^T(\mathbf{x}_i)\phi(\mathbf{x}_n) = \sum_{n=1}^{N} \alpha_n K(\mathbf{x}_i, \mathbf{x}_n)$$

$$\phi : \mathcal{X} \mapsto \mathcal{H}$$

$$x \in \mathbb{R}^{d<\infty}$$

$$\phi(x) \in \mathbb{R}^{D \to \infty}$$

No need to visit the feature space (H)!

# Kernel Methods do NOT Scale Well

$$\phi : \mathcal{X} \to \mathcal{H}$$

$$f(x) = W^T \phi(x) = \sum_{n=1}^{N} \alpha_n \; K(x, x_n)$$

$$K_{ij} = \phi^T(x_i) \; \phi(x_j)$$

$$\begin{bmatrix} K_{11} & \cdots & K_{1N} \\ \vdots & \ddots & \vdots \\ K_{N1} & \cdots & K_{NN} \end{bmatrix}$$

Kernel matrix

# Kernel Methods do NOT Scale Well

- Training complexity
  - Time: $O(N^2) < \ < O(N^3)$
  - Space: $O(N^2)$

$$\phi : \mathcal{X} \rightarrow \mathcal{H}$$

$$f(x) = W^T \phi(x) = \sum_{n=1}^{N} \alpha_n \ K(x, x_n)$$

$$K_{ij} = \phi^T(x_i) \ \phi(x_j)$$

$$\begin{bmatrix} K_{11} & \cdots & K_{1N} \\ \vdots & \ddots & \vdots \\ K_{N1} & \cdots & K_{NN} \end{bmatrix}$$

Kernel matrix

# Kernel Methods do NOT Scale Well

- Training complexity
  - Time: $O(N^2) < < O(N^3)$
  - Space: $O(N^2)$
  - One Hour Speech
    - N = 360,000
    - Kernel mat size = 230Mbit (16x40xN)

$$\phi : \mathcal{X} \rightarrow \mathcal{H}$$

$$f(x) = W^T \phi(x) = \sum_{n=1}^{N} \alpha_n \; K(x, x_n)$$

$$K_{ij} = \phi^T(x_i) \; \phi(x_j)$$

$$\begin{bmatrix} K_{11} & \cdots & K_{1N} \\ \vdots & \ddots & \vdots \\ K_{N1} & \cdots & K_{NN} \end{bmatrix}$$

Kernel matrix

# Kernel Methods do NOT Scale Well

- Training complexity
  - Time: O($N^2$) < < O($N^3$)
  - Space: O($N^2$)
  - One Hour Speech
    - N = 360,000
    - Kernel mat size = 230Mbit (16x40xN)
- Test Complexity
  - O($N$)

$$\phi : \mathcal{X} \rightarrow \mathcal{H}$$

$$f(x) = W^T \phi(x) = \sum_{n=1}^{N} \alpha_n \, K(x, x_n)$$

$$K_{ij} = \phi^T(x_i) \, \phi(x_j)$$

$$\begin{bmatrix} K_{11} & \cdots & K_{1N} \\ \vdots & \ddots & \vdots \\ K_{N1} & \cdots & K_{NN} \end{bmatrix}$$

Kernel matrix

# Kernel Methods do NOT Scale Well

- Training complexity
  - Time: O($N^2$) < < O($N^3$)
  - Space: O($N^2$)
  - One Hour Speech
    - N = 360,000
    - Kernel mat size = 230Mbit (16x40xN)
- Test Complexity
  - O($N$)
  - #SVs increases linearly by $N$ (*Steinwart et al, 2008*)

$$\phi : \mathcal{X} \to \mathcal{H}$$

$$f(x) = W^T \phi(x) = \sum_{n=1}^{N} \alpha_n \, K(x, x_n)$$

$$K_{ij} = \phi^T(x_i)\ \phi(x_j)$$

$$\begin{bmatrix} K_{11} & \cdots & K_{1N} \\ \vdots & \ddots & \vdots \\ K_{N1} & \cdots & K_{NN} \end{bmatrix}$$

Kernel matrix

# Scaling Up the Kernel Machines

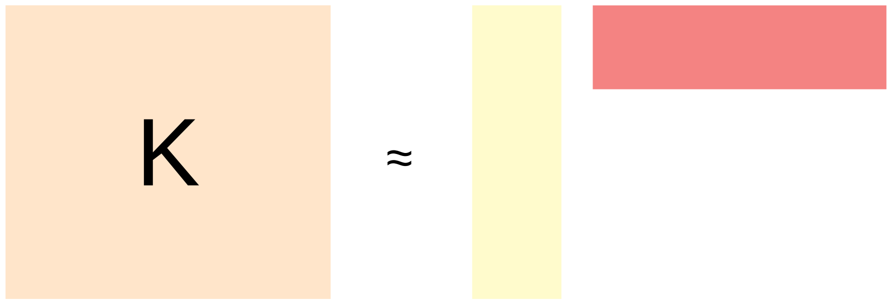
# How to Scale-up -- Kernel Approximation

- Kernel matrix approximation

- Kernel function approximation

# How to Scale-up -- Kernel Approximation

- Kernel matrix approximation

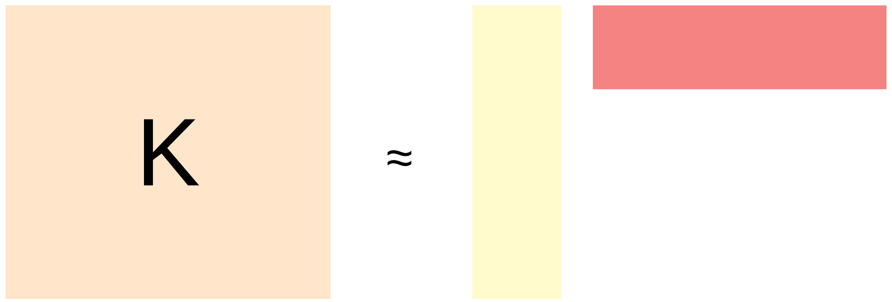



- Kernel function approximation

$$K(\mathbf{x}_i, \mathbf{x}_j) = \phi^T(\mathbf{x}_i)\phi(\mathbf{x}_j)$$
$$\approx \hat{\phi}^T(\mathbf{x}_i)\hat{\phi}(\mathbf{x}_j)$$

# How to Scale-up -- Kernel Approximation

- Kernel matrix approximation
  - Nyström approximation

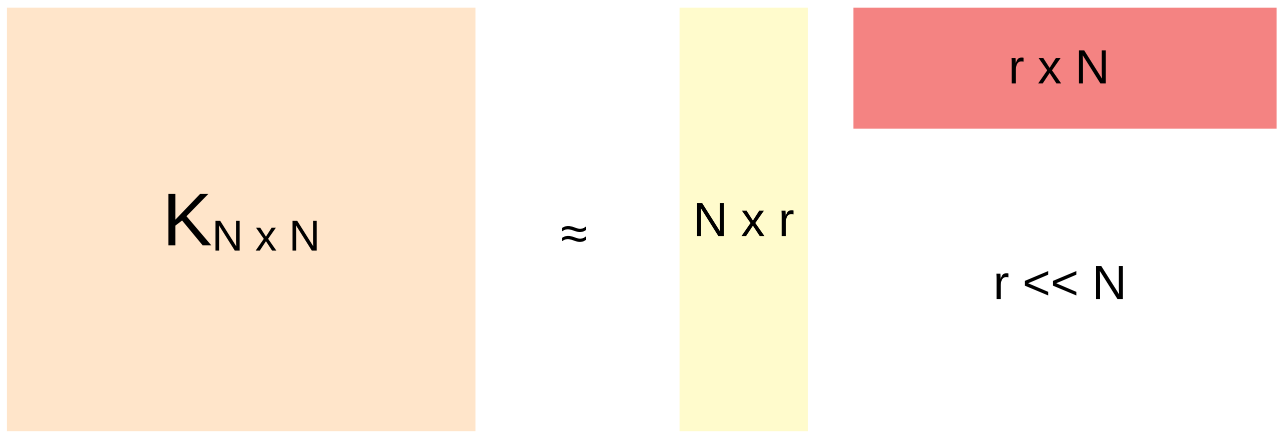



- Kernel function approximation
  - Random Fourier Features

$$K(\mathbf{x}_i, \mathbf{x}_j) = \phi^T(\mathbf{x}_i)\phi(\mathbf{x}_j)$$
$$\approx \hat{\phi}^T(\mathbf{x}_i)\hat{\phi}(\mathbf{x}_j)$$

# Nyström Approximation

- Consider low-rank matrix decomposition, e.g. SVD: $K = U\Sigma V^T$

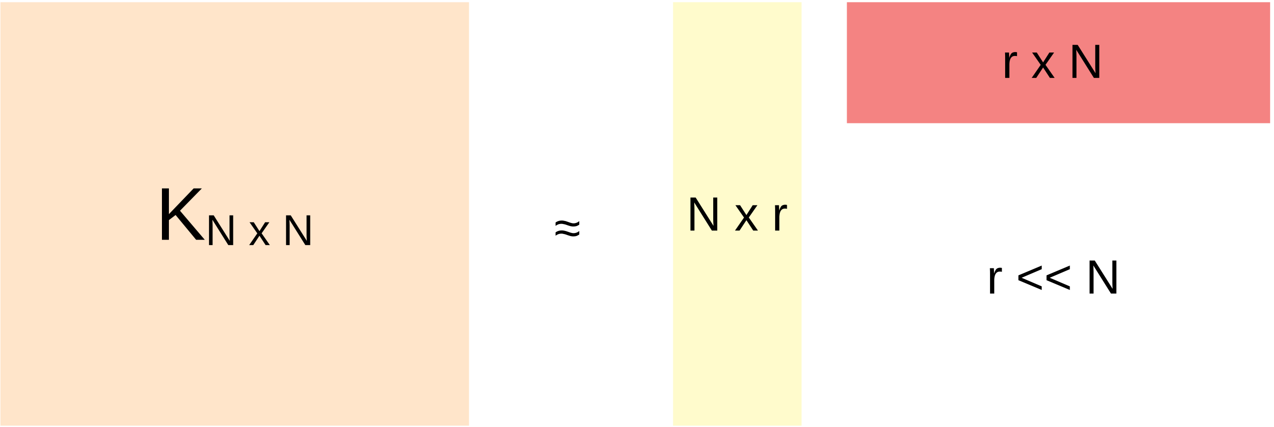



#parameters: N x N

#parameters: 2rN

# Nyström Approximation

- Consider low-rank matrix decomposition, e.g. SVD: $K = U\Sigma V^T$
- K must be formed explicitly, challenging when $N \to \infty$

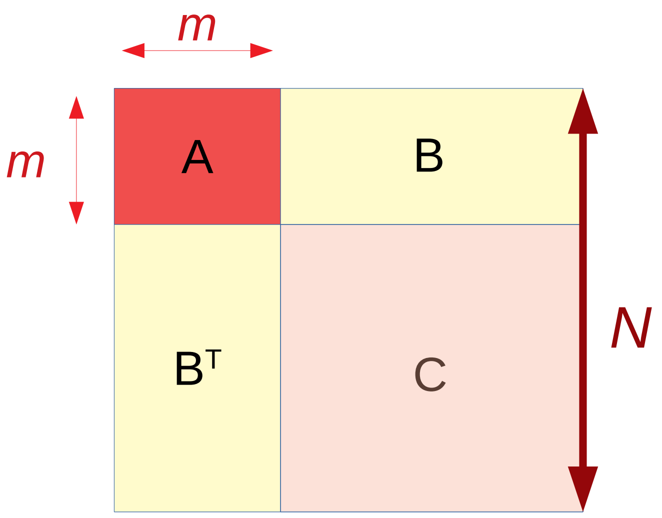



#parameters: N x N

#parameters: 2rN

# Nyström Approximation

- Nyström Approximation → no need to form K explicitly
- ONLY save A and B!



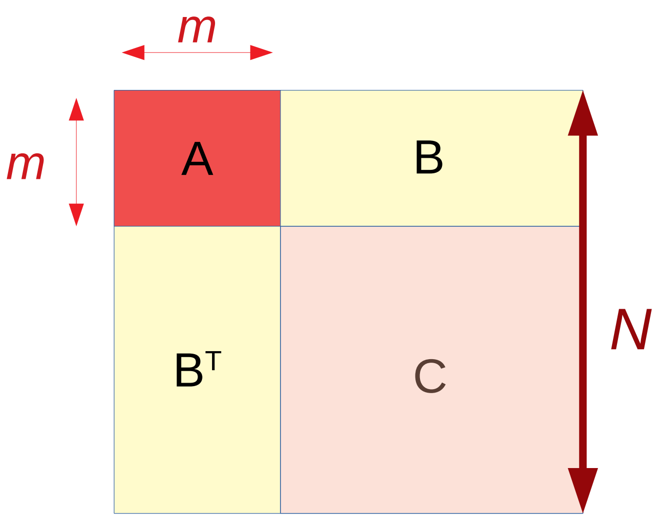

$K = K^T$

$$m \ll N$$

#parameters: $N^2 \rightarrow mN$

# Nyström Approximation

- Nyström Approximation → no need to form K explicitly
- ONLY save A and B!



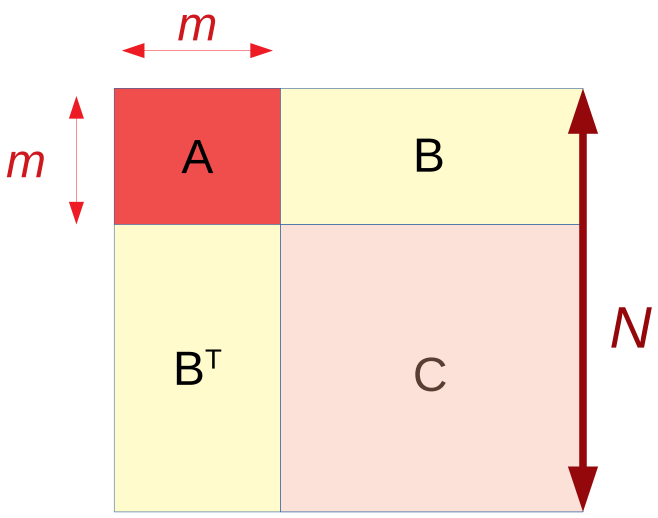

$$C \approx B^T A^{-1} B$$

$$m \ll N$$

$K = K^T$

# Nyström Approximation

- Nyström Approximation → no need to form K explicitly
- ONLY save A and B!




K = K^T

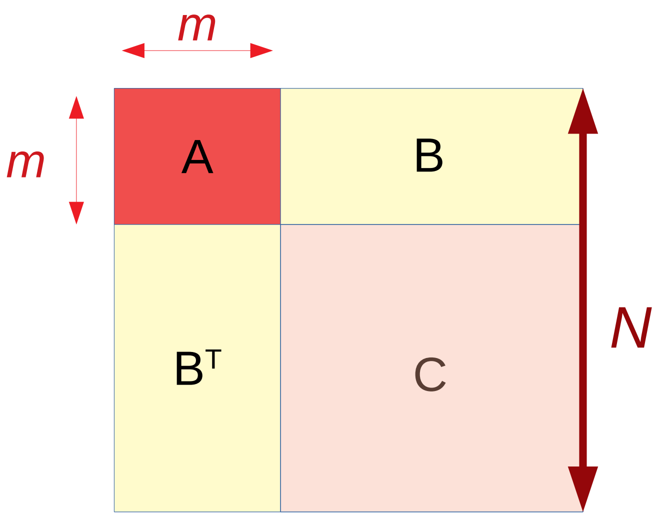
$$C \approx B^T A^{-1} B$$

$$m \ll N$$

$$m \geq r$$

# Nyström Approximation

- Nyström Approximation → no need to form K explicitly
- Challenges: choosing *m* value and *m* landmark points




$$C \approx B^T A^{-1} B$$

$$m \ll N$$

$$m \geq r$$

$K = K^T$

# Random Fourier Features

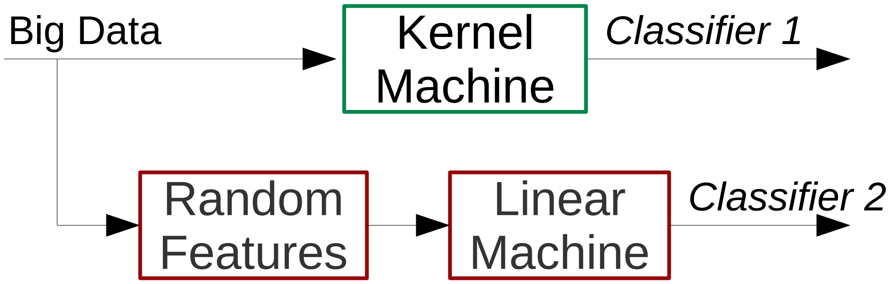
$$K(\mathbf{x}_i, \mathbf{x}_j) = \langle \phi(x_i), \phi(x_j) \rangle$$
$$\approx \langle \hat{\phi}(\mathbf{x}_i), \hat{\phi}(\mathbf{x}_j) \rangle$$

$$\begin{cases} \phi : \mathbb{R}^d \to \mathbb{R}^D \\ \hat{\phi} : \mathbb{R}^d \to \mathbb{R}^{\hat{D}} \end{cases}, \quad \hat{D} \ll D$$

# Random Fourier Features

$$K(\mathbf{x}_i, \mathbf{x}_j) = \langle \phi(x_i), \phi(x_j) \rangle$$
$$\approx \langle \hat{\phi}(\mathbf{x}_i), \hat{\phi}(\mathbf{x}_j) \rangle$$

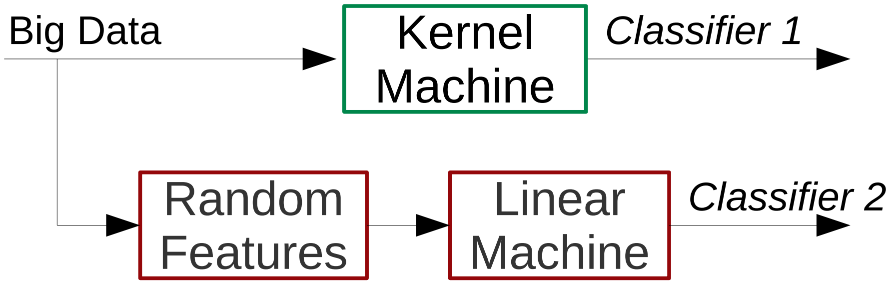
$$\begin{cases} \phi : \mathbb{R}^d \rightarrow \mathbb{R}^D \\ \hat{\phi} : \mathbb{R}^d \rightarrow \mathbb{R}^{\hat{D}} \end{cases} , \quad \hat{D} \ll D$$



# Random Fourier Features

$$K(\mathbf{x}_i, \mathbf{x}_j) = \langle \phi(x_i), \phi(x_j) \rangle \approx \langle \hat{\phi}(\mathbf{x}_i), \hat{\phi}(\mathbf{x}_j) \rangle$$

$$\begin{cases} \phi : \mathbb{R}^d \to \mathbb{R}^D \\ \hat{\phi} : \mathbb{R}^d \to \mathbb{R}^{\hat{D}} \end{cases}, \quad \hat{D} \ll D$$

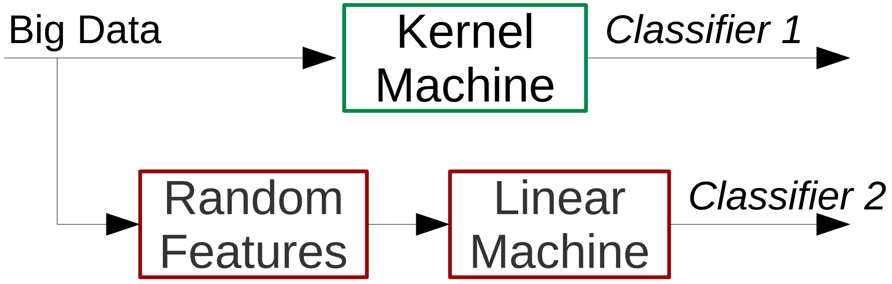



$$\text{Classifier 1} = W^T \phi(x) = \sum_{n=1}^{N} \alpha_n \, K(x, x_n)$$

$$\text{Classifier 2} = \hat{W}^T \hat{\phi}(x)$$

# Random Fourier Features

$$K(\mathbf{x}_i, \mathbf{x}_j) = \langle \phi(x_i), \phi(x_j) \rangle$$
$$\approx \langle \hat{\phi}(\mathbf{x}_i), \hat{\phi}(\mathbf{x}_j) \rangle$$




$$\begin{cases} \phi : \mathbb{R}^d \rightarrow \mathbb{R}^D \\ \hat{\phi} : \mathbb{R}^d \rightarrow \mathbb{R}^{\hat{D}} \end{cases}, \quad \hat{D} \ll D$$

GOAL: Find $\hat{\phi}$

such that

Classifier1 $\equiv$ Classifier2

$$\text{Classifier 1} = W^T \phi(x) = \sum_{n=1}^{N} \alpha_n \, K(x, x_n)$$

$$\text{Classifier 2} = \hat{W}^T \hat{\phi}(x)$$

# Random Fourier Features – Theory

- ***Bochner's Theorem:***
  - *A continuous shift-invariant kernel function* $K(x,y)=K(x-y,0)=K(\delta)$

# Random Fourier Features – Theory

- ***Bochner's Theorem:***
  - *A continuous shift-invariant kernel function K(x,y)=K(x-y,0)=K(δ)*

$$K(x,y) = exp(-\frac{\|x-y\|^2}{2\sigma^2}) \rightarrow K(\delta) = exp(-\frac{\|\delta\|^2}{2\sigma^2})$$

# Random Fourier Features – Theory

- ***Bochner's Theorem:***
  - *A continuous shift-invariant kernel function K(x,y)=K(x-y,0)=K(δ)*
  - *is positive-definite (satisfies Mercer's condition) iff*

$$K(x,y) = exp(-\frac{\|x-y\|^2}{2\sigma^2}) \rightarrow K(\delta) = exp(-\frac{\|\delta\|^2}{2\sigma^2})$$

# Random Fourier Features – Theory

- ***Bochner's Theorem:***
  - *A continuous shift-invariant kernel function K(x,y)=K(x-y,0)=K(δ)*
  - *is positive-definite (satisfies Mercer's condition) iff*
  - *K(δ) is the Fourier transform of a non-negative measure k(ω).*

$$K(x,y) = exp(-\frac{\|x-y\|^2}{2\sigma^2}) \rightarrow K(\delta) = exp(-\frac{\|\delta\|^2}{2\sigma^2})$$

# Random Fourier Features – Theory

- ***Bochner's Theorem:***
  - *A continuous shift-invariant kernel function K(x,y)=K(x-y,0)=K(δ)*
  - *is positive-definite (satisfies Mercer's condition) iff*
  - *K(δ) is the Fourier transform of a non-negative measure k(ω).*

$$K(x,y) = exp(-\frac{\|x-y\|^2}{2\sigma^2}) \rightarrow K(\delta) = exp(-\frac{\|\delta\|^2}{2\sigma^2})$$

$$K(\delta) = \int_{\mathbb{R}^d} k(\omega) e^{-j\delta^T\omega} d\omega \Big|_{k(\omega)\geq 0}$$

# Random Fourier Features – Approxmation

$$k(\omega) \geq 0$$

# Random Fourier Features – Approxmation

$$k(\omega) \geq 0 \Longrightarrow \frac{k(\omega)}{Z} = p(\omega)$$

# Random Fourier Features – Approxmation

$$k(\omega) \geq 0 \Longrightarrow \frac{k(\omega)}{Z} = p(\omega)$$

$$K(\delta) = \int_{\mathbb{R}^d} k(\omega) e^{-j\delta^T \omega} \, d\omega = Z \int_{\mathbb{R}^d} p(\omega) \; e^{-j\delta^T \omega} \, d\omega$$

# Random Fourier Features – Approxmation

$$k(\omega) \geq 0 \Longrightarrow \frac{k(\omega)}{Z} = p(\omega)$$

$$K(\delta) = \int_{\mathbb{R}^d} k(\omega) e^{-j\delta^T\omega} \; d\omega = Z \int_{\mathbb{R}^d} p(\omega) \; e^{-j\delta^T\omega} \; d\omega$$

Without loss of generality
assume Z=1

# Random Fourier Features – Approxmation

$$k(\omega) \geq 0 \Longrightarrow \frac{k(\omega)}{Z} = p(\omega)$$

$$K(\delta) = \int_{\mathbb{R}^d} k(\omega) e^{-j\delta^T \omega} \, d\omega = \boxed{\int_{\mathbb{R}^d} p(\omega) \; e^{-j\delta^T \omega} \; d\omega}$$

# Random Fourier Features – Approxmation

$$k(\omega) \geq 0 \Longrightarrow \frac{k(\omega)}{Z} = p(\omega)$$

$$K(\delta) = \int_{\mathbb{R}^d} k(\omega) e^{-j\delta^T\omega}\, d\omega = \int_{\mathbb{R}^d} p(\omega)\; e^{-j\delta^T\omega}\, d\omega = \mathbb{E}_\omega \left[ e^{-j\delta^T\omega} \right]$$

# Random Fourier Features – Approxmation

$$k(\omega) \geq 0 \Longrightarrow \frac{k(\omega)}{Z} = p(\omega)$$

$$K(\delta) = \int_{\mathbb{R}^d} k(\omega) e^{-j\delta^T \omega} \, d\omega = \int_{\mathbb{R}^d} p(\omega) \; e^{-j\delta^T \omega} \, d\omega = \mathbb{E}_\omega \left[ e^{-j\delta^T \omega} \right]$$

$$\mathrm{K}(\delta) = \mathbb{E}_\omega \left[ e^{-j\delta^T \omega} \right] \approx \tfrac{1}{\hat{D}} \textstyle\sum_{i=1}^{\hat{D}} e^{-j\delta^T \omega_i} \Big|_{\omega_i \sim p(\omega)}$$

# Random Fourier Features – Approxmation

$$k(\omega) \geq 0 \implies \frac{k(\omega)}{Z} = p(\omega)$$

$$K(\delta) = \int_{\mathbb{R}^d} k(\omega) e^{-j\delta^T \omega} \, d\omega = \int_{\mathbb{R}^d} p(\omega) \; e^{-j\delta^T \omega} \, d\omega = \mathbb{E}_\omega \left[ e^{-j\delta^T \omega} \right]$$

$$\mathrm{K}(\delta) = \mathbb{E}_\omega \left[ e^{-j\delta^T \omega} \right] \approx \frac{1}{\hat{D}} \sum_{i=1}^{\hat{D}} e^{-j\delta^T \omega_i} \Big|_{\omega_i \sim p(\omega)}$$

Monte Carlo Method (MC)

Draw D̂ iid samples from $p(\omega)$

# Random Fourier Features – Kernelisation

$$\mathrm{K}(\vec{x},\vec{y}) = \mathrm{K}(\overbrace{\vec{x}-\vec{y}}^{\delta},0) \approx \frac{1}{\hat{D}}\sum_{i=1}^{\hat{D}} e^{-j(\vec{x}-\vec{y})^T\omega_i}\Big|_{\vec{\omega}_i\sim p(\vec{\omega})}$$

# Random Fourier Features – Kernelisation

$$\mathrm{K}(\vec{x},\vec{y}) = \mathrm{K}(\overbrace{\vec{x}-\vec{y}}^{\delta},0) \approx \frac{1}{\hat{D}}\sum_{i=1}^{\hat{D}} e^{-j(\vec{x}-\vec{y})^T\omega_i}\Big|_{\vec{\omega}_i\sim p(\vec{\omega})}$$

↓

Turn it into an inner product

↓

$$K(\mathbf{x}_i,\mathbf{x}_j) \approx \hat{\phi}^T(\mathbf{x}_i)\hat{\phi}(\mathbf{x}_j)$$

# Random Fourier Features – Kernelisation

$$\mathrm{K}(\vec{x},\vec{y}) = \mathrm{K}(\overbrace{\vec{x}-\vec{y}}^{\delta},0) \approx \frac{1}{\hat{D}}\sum_{i=1}^{\hat{D}} e^{-j(\vec{x}-\vec{y})^T\omega_i}\Big|_{\vec{\omega}_i\sim p(\vec{\omega})}$$

Turn it into an inner product

$$K(\mathbf{x}_i,\mathbf{x}_j) \approx \hat{\phi}^T(\mathbf{x}_i)\hat{\phi}(\mathbf{x}_j)$$

$$\vec{\Omega} = \begin{bmatrix} \vec{\omega}_1 \\ \vdots \\ \vec{\omega}_m \\ \vdots \\ \vec{\omega}_{\hat{D}} \end{bmatrix} \rightarrow \hat{\phi}_m(\mathbf{x}) = \sqrt{\frac{2}{\hat{D}}} cos(\vec{\omega}_m^T\mathbf{x} + b_m) \rightarrow \hat{\phi}(x) = \begin{bmatrix} \hat{\phi}_1(x) \\ \vdots \\ \hat{\phi}_m(x) \\ \vdots \\ \hat{\phi}_{\hat{D}}(x) \end{bmatrix}$$

$$\vec{\omega}_m \sim p(\vec{\omega}) \qquad b \sim \mathcal{U}(0,2\pi)$$

Uniform

# Random Fourier Features – Kernelisation

$$\mathrm{K}(\vec{x},\vec{y}) = \mathrm{K}(\overbrace{\vec{x}-\vec{y}}^{\delta},0) \approx \frac{1}{\hat{D}}\sum_{i=1}^{\hat{D}} e^{-j(\vec{x}-\vec{y})^T\omega_i}\Big|_{\vec{\omega}_i\sim p(\vec{\omega})}$$

Turn it into an inner product

$$K(\mathbf{x}_i,\mathbf{x}_j) \approx \hat{\phi}^T(\mathbf{x}_i)\hat{\phi}(\mathbf{x}_j)$$

$$\vec{\Omega} = \begin{bmatrix} \vec{\omega}_1 \\ \vdots \\ \vec{\omega}_m \\ \vdots \\ \vec{\omega}_{\hat{D}} \end{bmatrix} \rightarrow \quad \hat{\phi}_m(\mathbf{x}) = \sqrt{\frac{2}{\hat{D}}} cos(\vec{\omega}_m^T\mathbf{x} + b_m) \quad \rightarrow \hat{\phi}(x) = \begin{bmatrix} \hat{\phi}_1(x) \\ \vdots \\ \hat{\phi}_m(x) \\ \vdots \\ \hat{\phi}_{\hat{D}}(x) \end{bmatrix}$$

$$\vec{\omega}_m \sim p(\vec{\omega}) \qquad b \sim \mathcal{U}(0,2\pi)$$

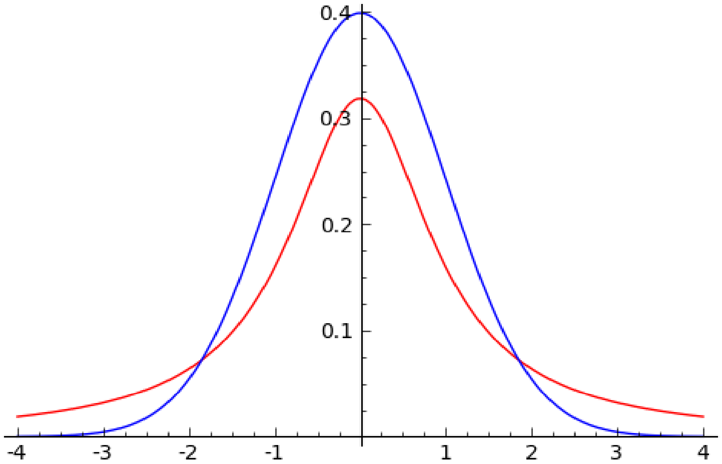
Details in the Appendix A of the paper

# Kernels Associated PDFs

- Kernel PDF = Inverse Fourier transform of K(x-y,0)

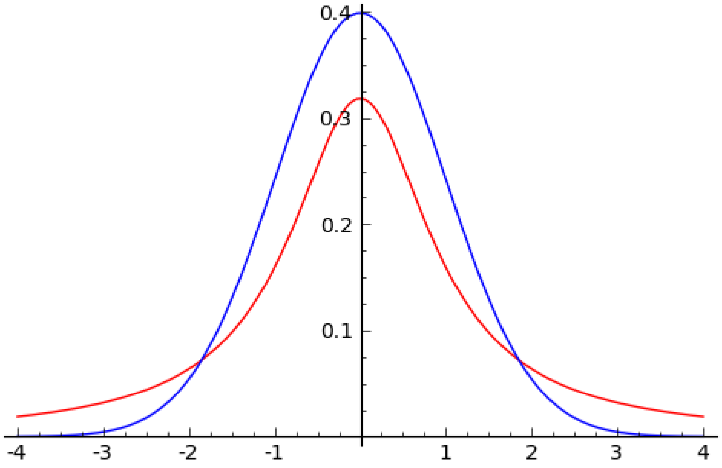
# Kernels Associated PDFs

- Kernel PDF = Inverse Fourier transform of K(x-y,0)
  - Gaussian kernel → Normal($0_d$,$\sigma^{-2}I_d$)
  - Laplacian kernel → Cauchy($0_d$,λ)

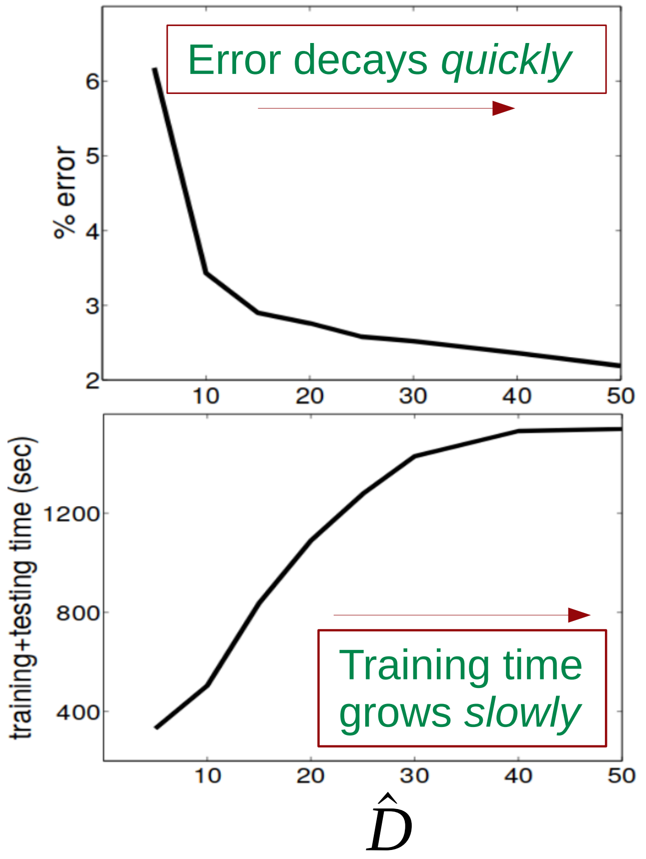


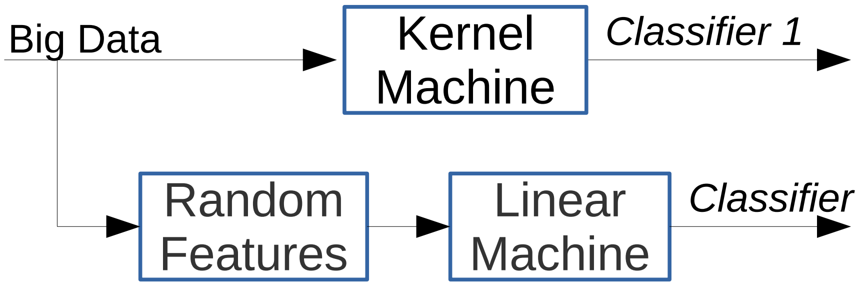
# Kernels Associated PDFs

- Kernel PDF = Inverse Fourier transform of K(x-y,0)
  - Gaussian kernel → Normal($0_d$,$\sigma^{-2}I_d$) → thin-tailed
  - Laplacian kernel → Cauchy($0_d$,λ) → fat-tailed



# Computational Gain



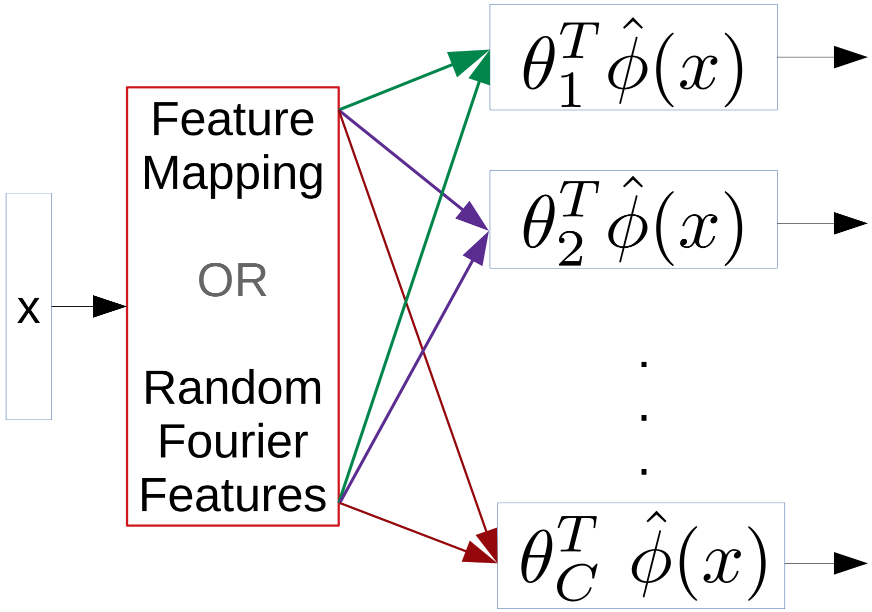

$$\text{Classifier 1} = \underbrace{W^T\phi(x)}_{O(D)} = \underbrace{\sum_{n=1}^{N} \alpha_n \; k(x, x_n)}_{O(N)}$$

$$\text{Classifier 2} = \underbrace{\hat{W}^T\hat{\phi}(x)}_{O(\hat{D})}$$

*A. Rahimi, B. Recht, "Random Features for Large-Scale Kernel Machines", NIPS 2007*

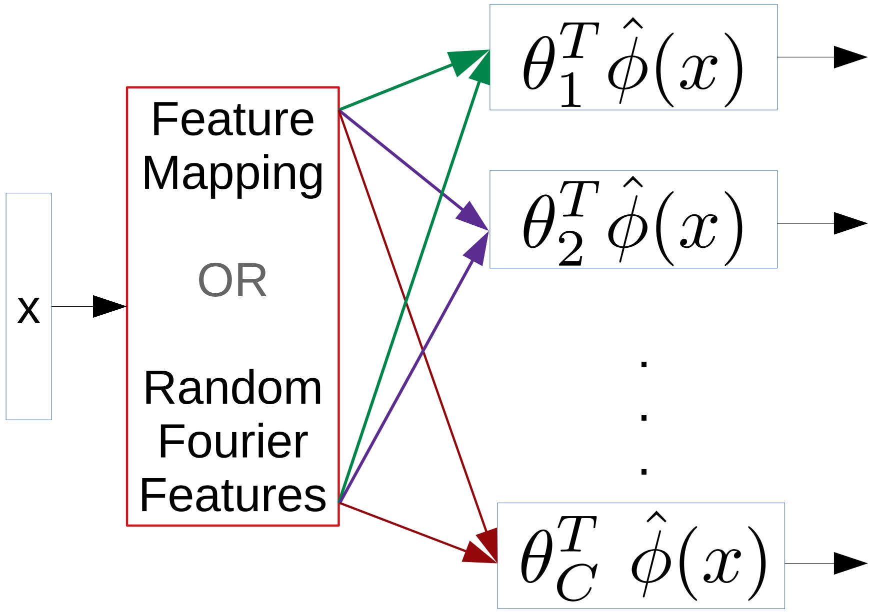
# Acoustic Modelling Using Kernel Methods

# Kernel Machine as a Classifier



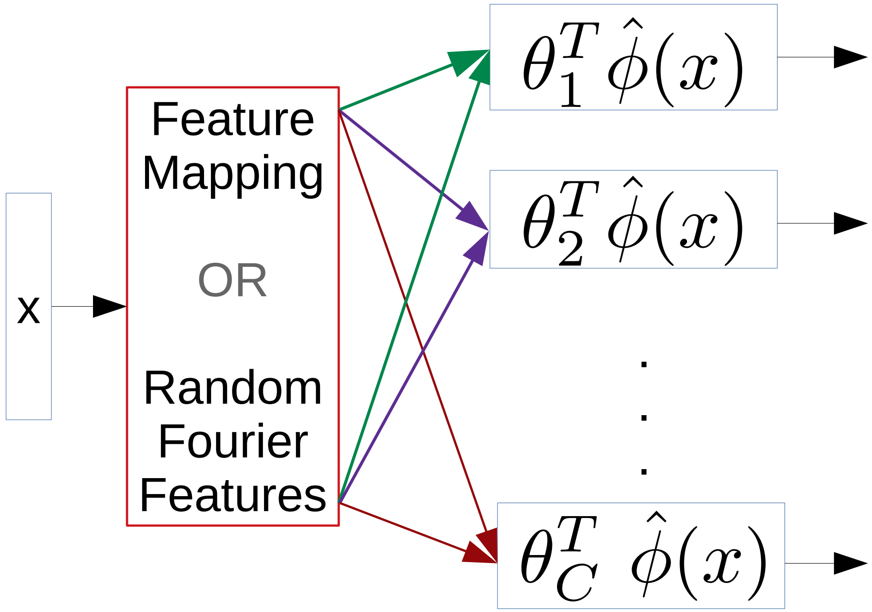
# Kernel Machine as a Classifier

$$p(y = c|x) \propto exp(\theta_c^T \hat{\phi}(x))$$



# Kernel Machine as a Classifier

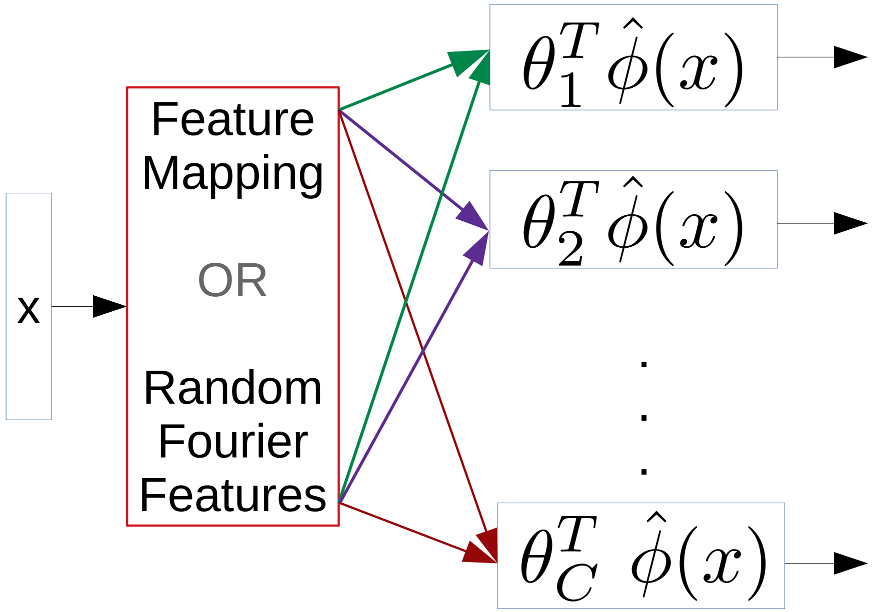
$$p(y = c|x) = \frac{exp(-E(\theta_c))}{Z}$$



# Kernel Machine as a Classifier

$$p(y = c|x) = \frac{exp(-E(\theta_c))}{Z}$$

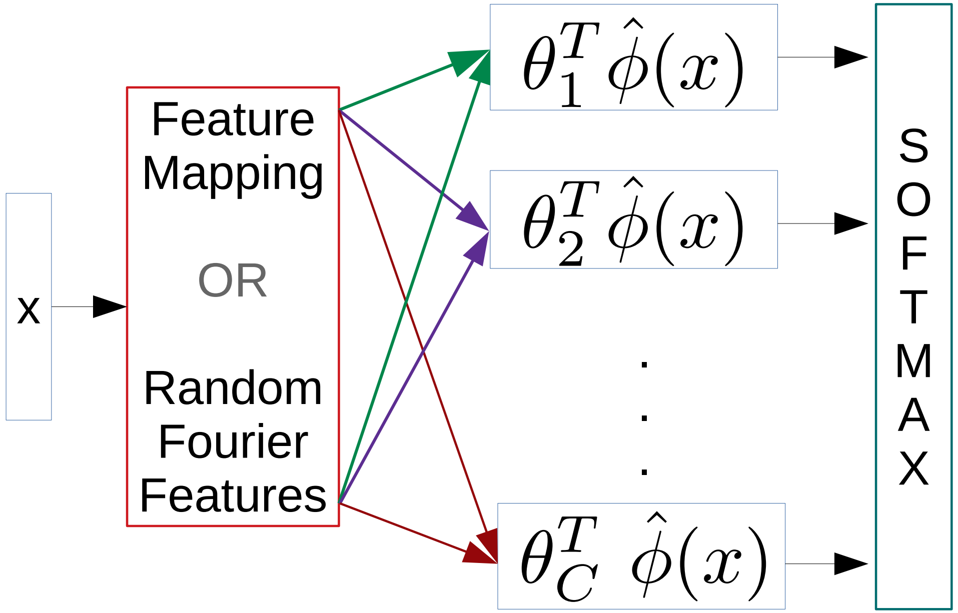
$$p(y = c|x) \propto exp(\underbrace{\theta_c^T \hat{\phi}(x)}_{-E})$$



# Kernel Machine as a Classifier

$$p(y=c|x) \propto exp(\theta_c^T \hat{\phi}(x))$$

$$p(y=c|x) = \frac{exp(\theta_c^T \hat{\phi}(x))}{\sum_{c'} exp(\theta_{c'}^T \hat{\phi}(x))}$$



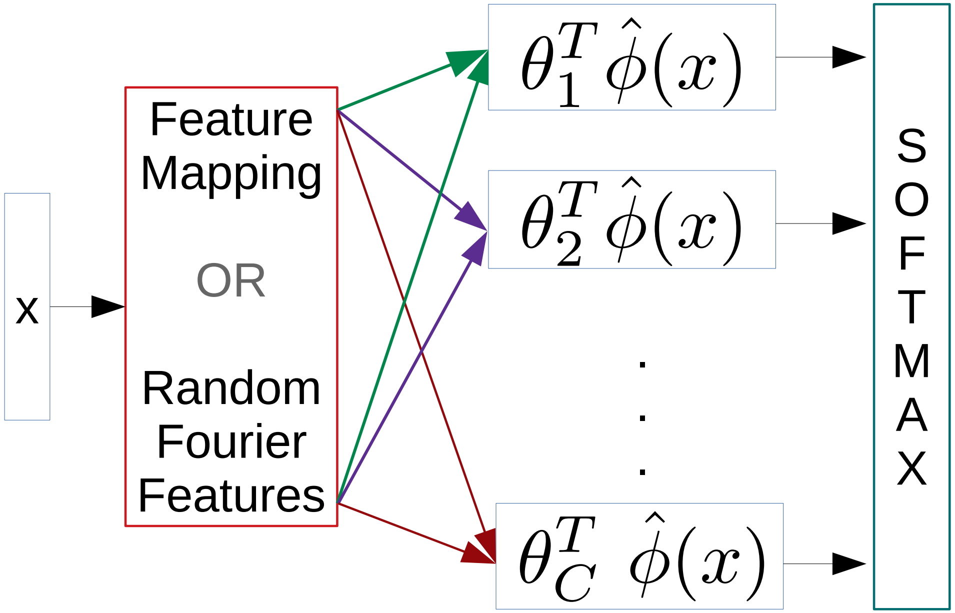
# Kernel Machine as a Classifier

$$p(y=c|x) \propto exp(\theta_c^T \hat{\phi}(x))$$

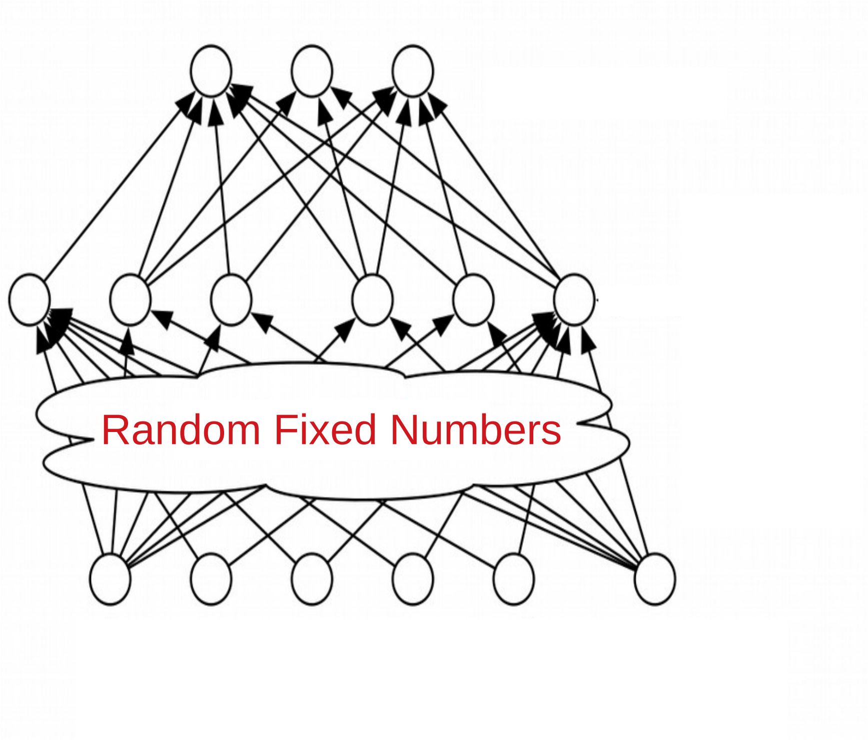
$$p(y=c|x) = \frac{exp(\theta_c^T \hat{\phi}(x))}{\sum_{c\prime} exp(\theta_{c\prime}^T \hat{\phi}(x))}$$

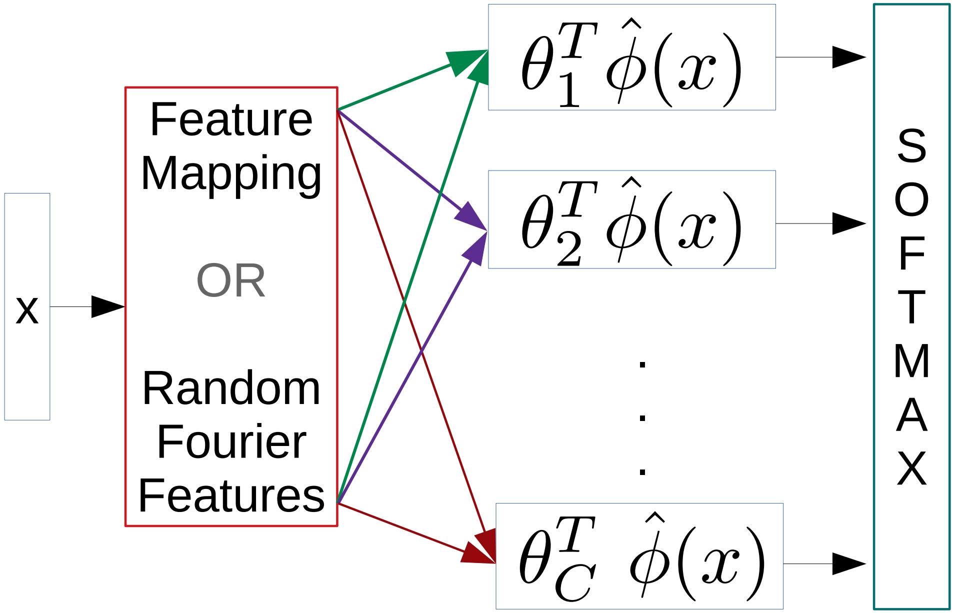
$$L(\Theta;(x,y)) = -log(p(y|x;\Theta))$$

$$= -\Theta_y^T \hat{\phi}(x) + log \sum_{c=1}^{C} exp(\Theta_c^T \hat{\phi}(x))$$



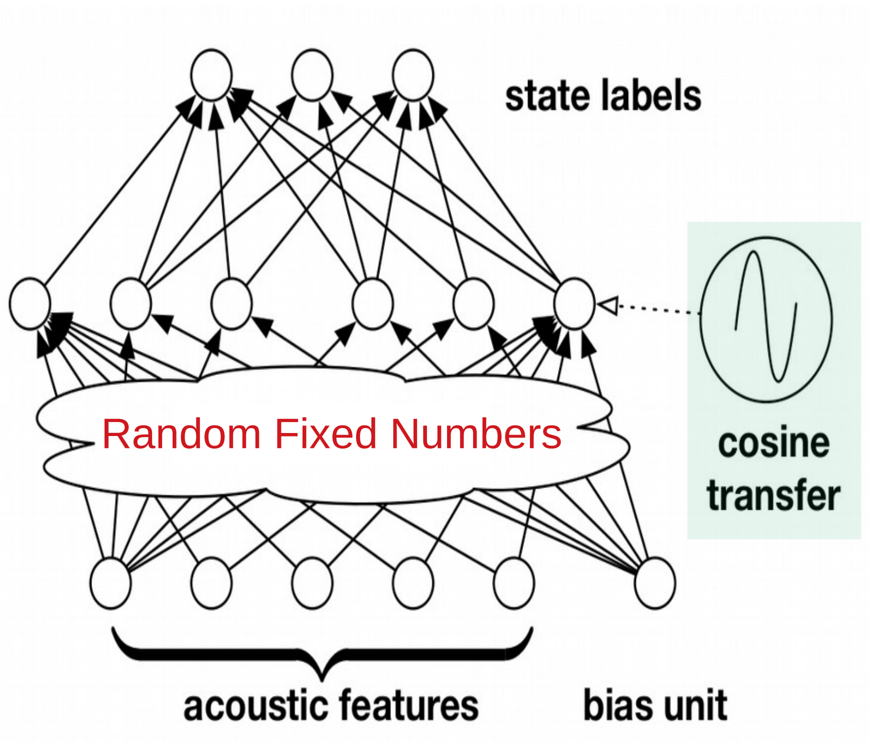
# Kernel Machines as a Shallow NN



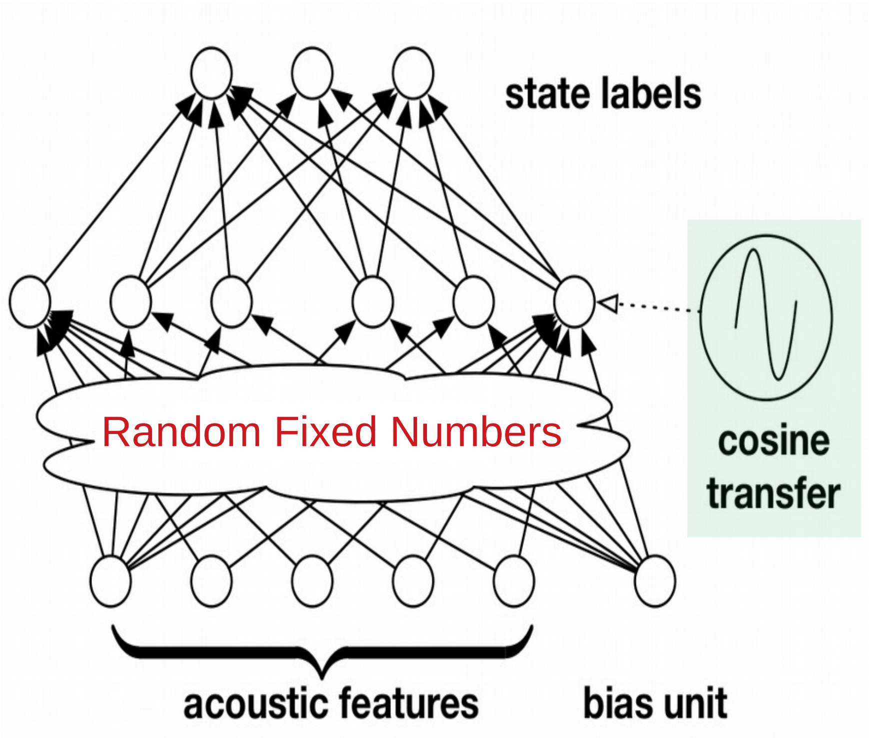
# Kernel Machines as a Shallow NN

- Objective function → Convex
- Optimisation → SGD



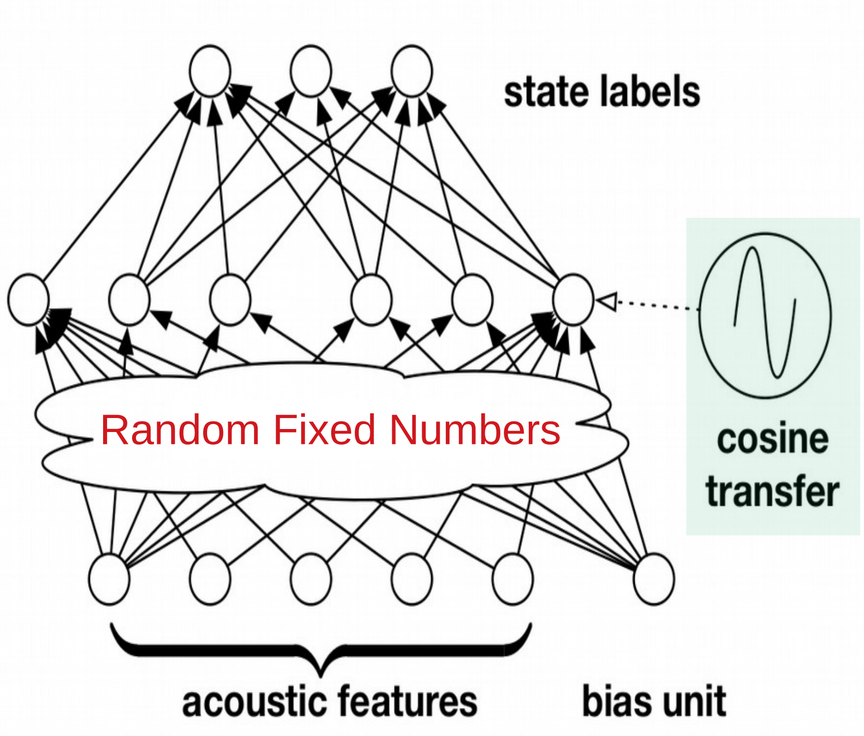
# Kernel Machines as a Shallow NN

- Objective function → Convex
- Optimisation: SGD
- Kernel Machines are discriminative → Posterior
  - Likelihood?



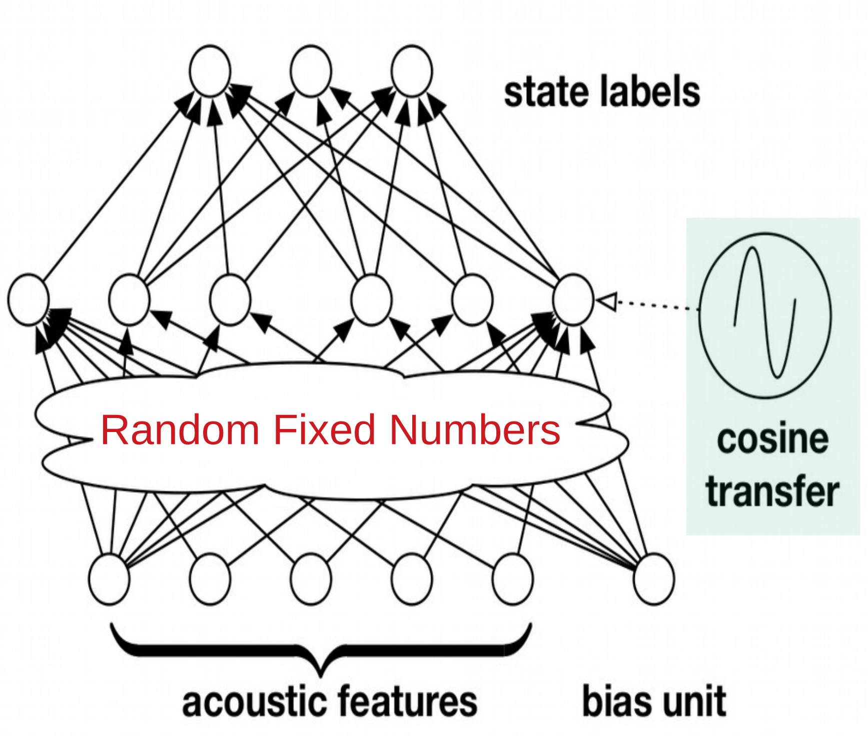
# Kernel Machines as a Shallow NN

- Objective function → Convex
- Optimisation: SGD
- Kernel Machines are discriminative → Posterior
- Bayes' rule + forced alignment → *scaled-likelihood*



# Kernel Machines as a Shallow NN

- Objective function → Convex
- Optimisation: SGD
- Kernel Machines are discriminative → Posterior
- Bayes' rule + forced alignment → *scaled-likelihood*
- Classes: context-dependent phonetic states



# Kernel Machines as a Shallow NN

$$\mathbb{R}^C$$

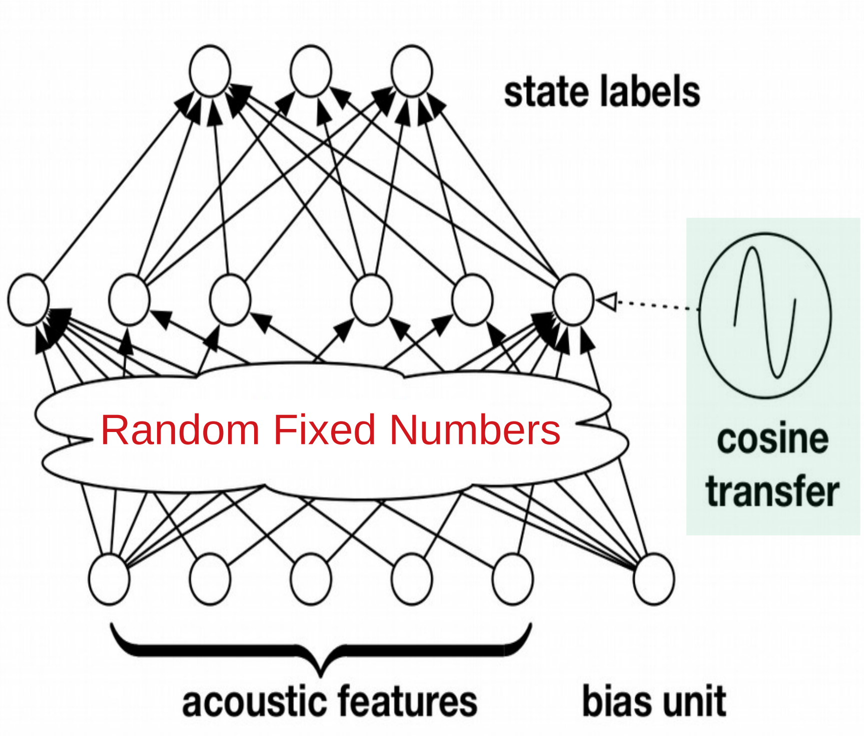
$$\Theta \in \mathbb{R}^{\hat{D}\times C}$$

$$\mathbb{R}^{\hat{D}}$$

$$\Theta_{FS} \in \mathbb{R}^{d\times \hat{D}}$$

$$\mathbb{R}^d$$

Feature Selection

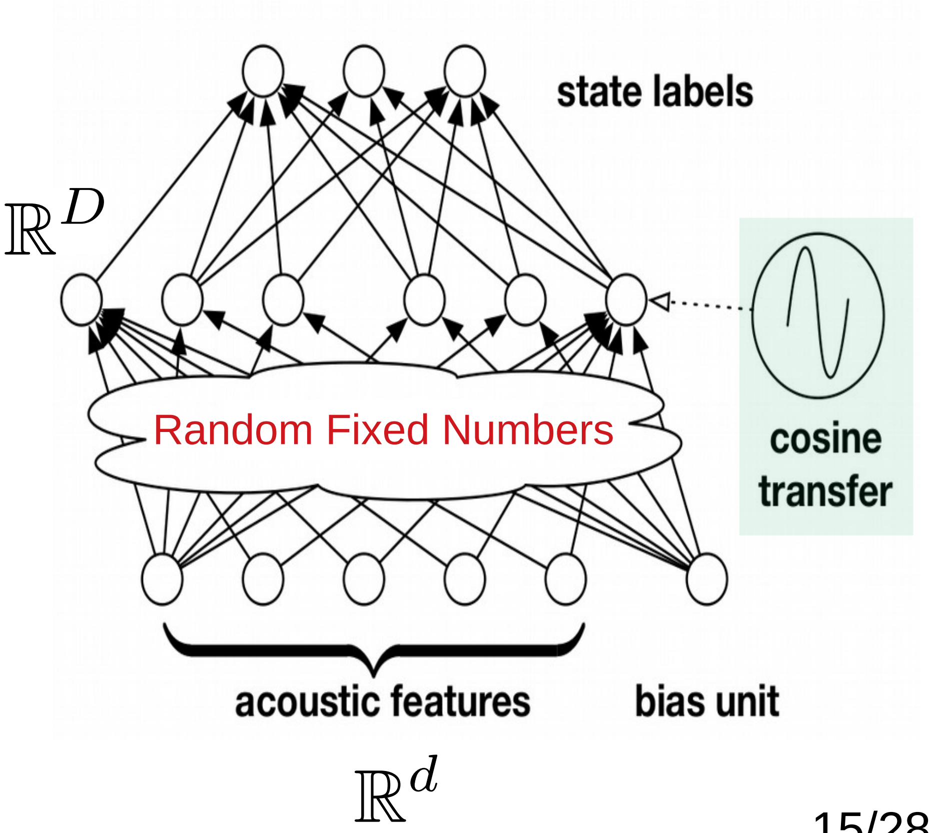


# Kernel Machines as a Shallow NN

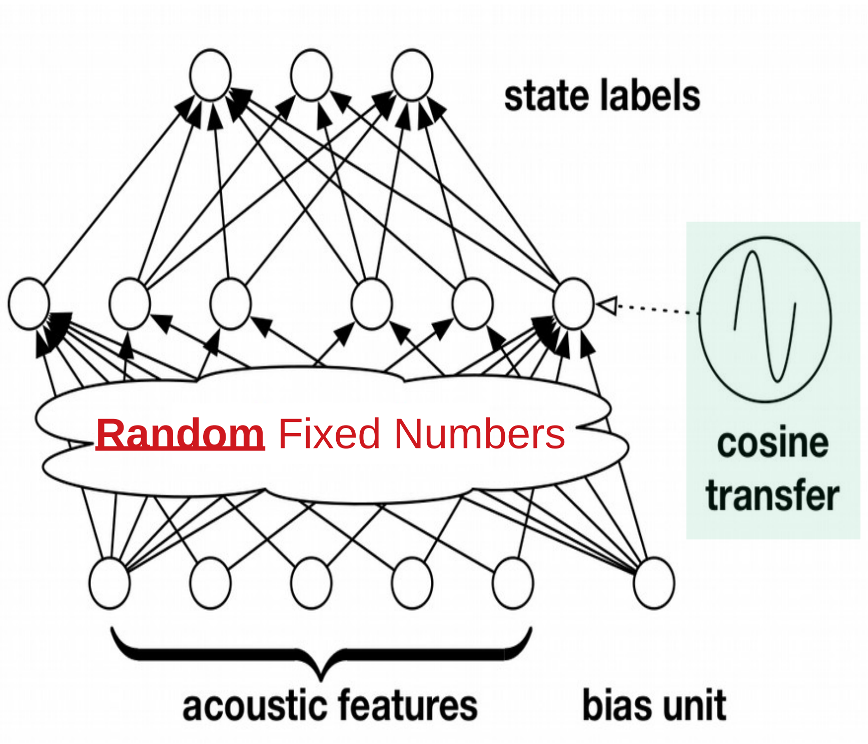
Without Random Features and *D* >> *d*

Cybenko Theorem ≡ Representer Theorem

$$f(\mathbf{x}) \approx \sum_i \alpha_i \; \sigma(\mathbf{x})$$

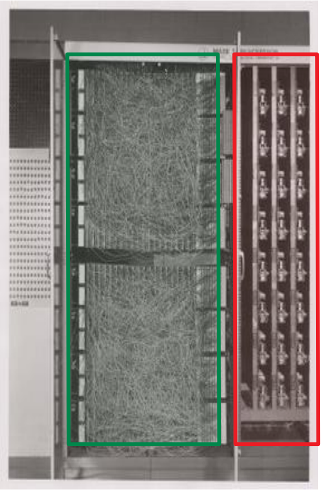


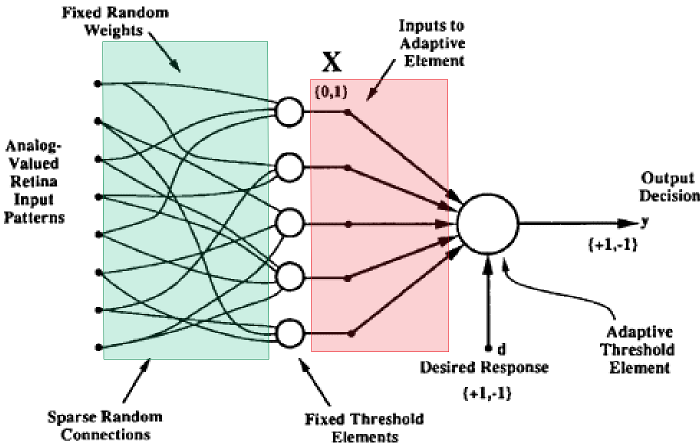
# Randomness in NNs

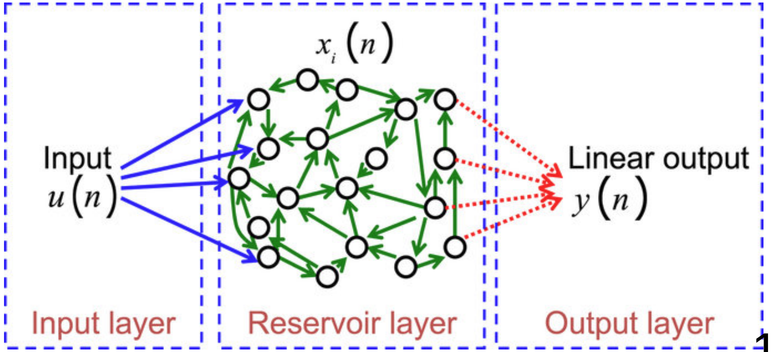


# Randomness in NNs

- Examples
  - FFNN → Perceptron
  - RNN → Reservoir Computing

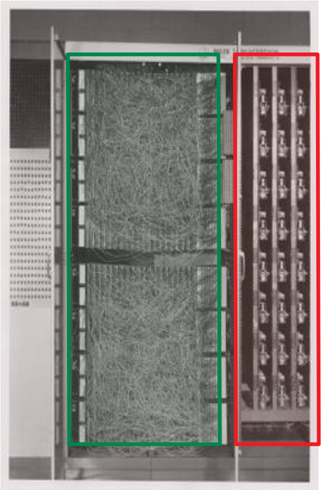


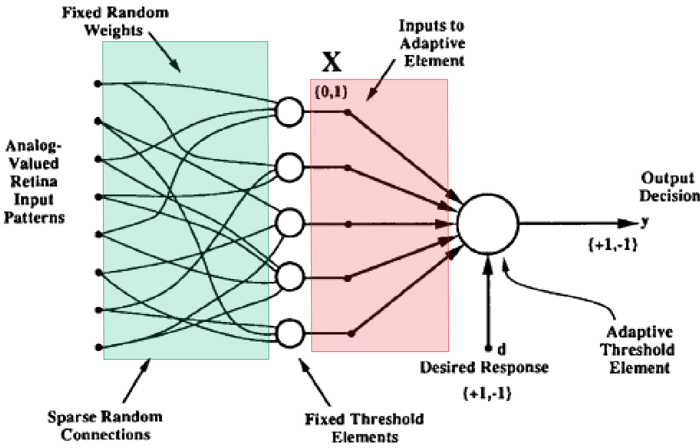


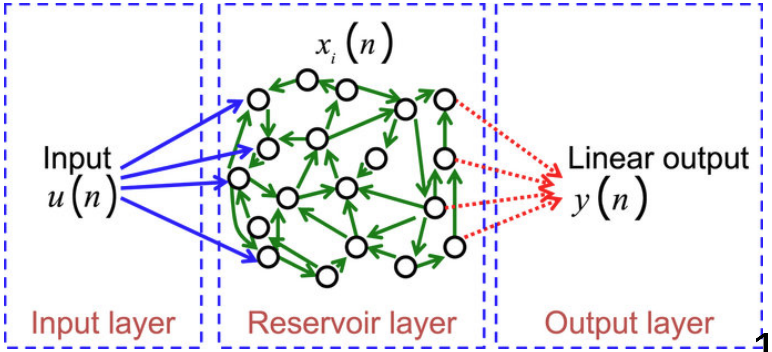



# Randomness in NNs

- Examples
  - FFNN → Perceptron
  - RNN → Reservoir Computing
- Advantages
  - Sparse high-dim feature space → better learning
  - Easier/scalable optimisation

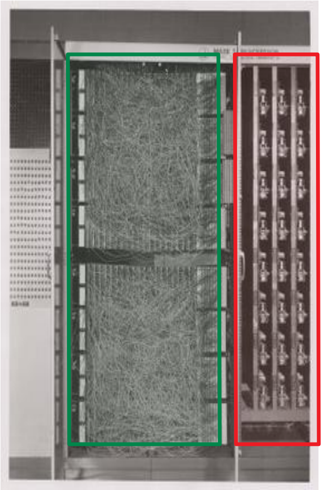


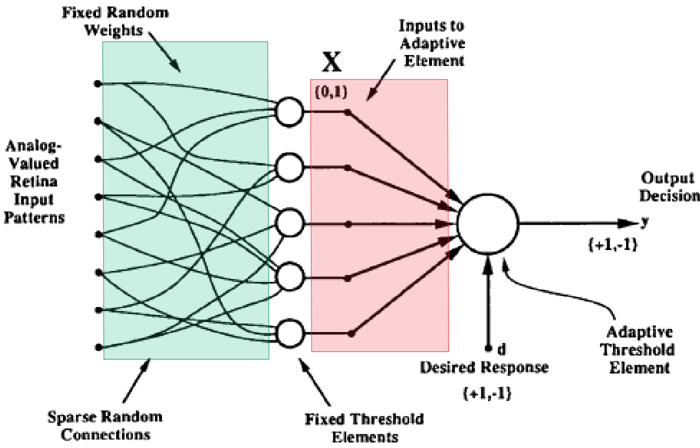


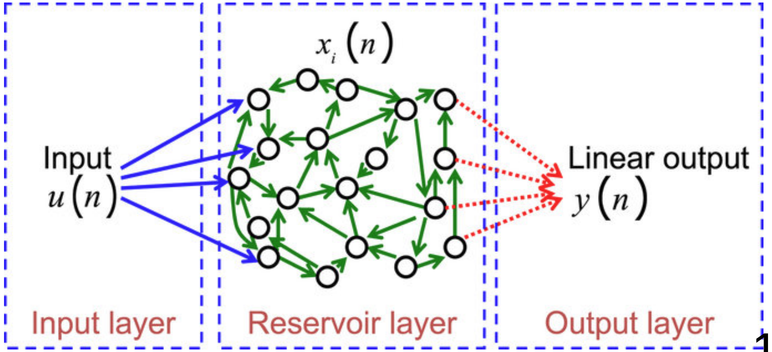



# Randomness in NNs

- Examples
  - FFNN → Perceptron
  - RNN → Reservoir Computing
- Advantages
  - Sparse high-dim feature space → better learning
  - Easier/scalable optimisation





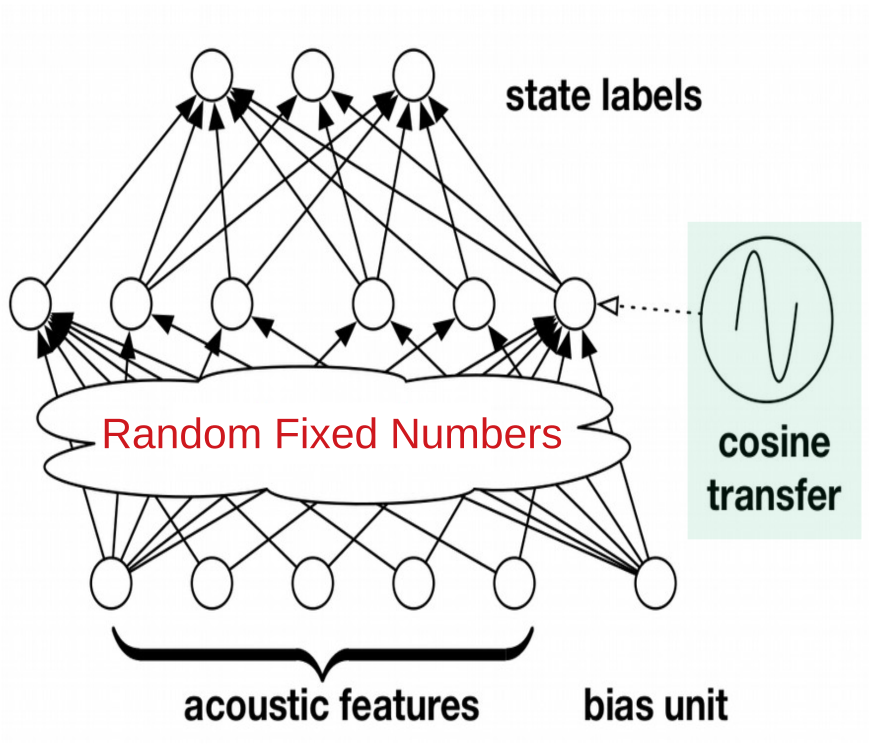




Rahimi and Recht "*randomisation is […] cheaper than optimisation*."

Advances in Neural Information Processing Systems (2009)

# Linear Bottlenecks

- #parameters: D x C
  - $10^4$ x $10^3$



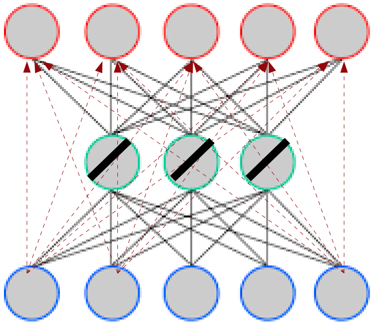
# Linear Bottlenecks

- GOAL: Reducing #parameters ($D \times C \geq 10^7$)

T. Sainath et al, *Low-rank matrix factorization for Deep Neural Network training with high-dimensional output targets*, ICASSP 2013

# Linear Bottlenecks

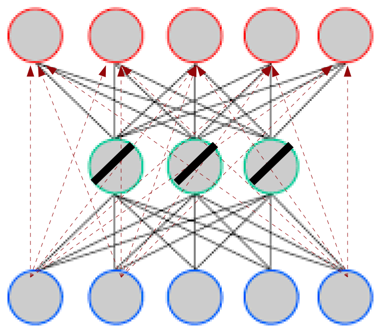
- GOAL: Reducing #parameters ($\mathbf{D} \times C \geq 10^7$)
- HOW: Low-rank matrix factorisation → $\Theta_{D \times C} \approx U_{D \times r} V_{r \times C}$



T. Sainath et al, Low-rank matrix factorization for Deep Neural Network training with high-dimensional output targets, ICASSP 2013

# Linear Bottlenecks

- GOAL: Reducing #parameters ($D \times C \geq 10^7$)
- HOW: Low-rank matrix factorisation → $\Theta_{D \times C} \approx U_{D \times r} V_{r \times C}$
- ADVANTAGE: Less parameters $D \times C$ → $r(D+C)$



*T. Sainath et al, Low-rank matrix factorization for Deep Neural Network training with high-dimensional output targets, ICASSP 2013*

# Linear Bottlenecks

- GOAL: Reducing #parameters ($D \times C \geq 10^7$)
- HOW: Low-rank matrix factorisation $\rightarrow \Theta_{D \times C} \approx U_{D \times r} V_{r \times C}$
- ADVANTAGE: Less parameters $D \times C \rightarrow r(D+C)$
- DISADVANTAGE: Less modelling power + non-convex optim.

T. Sainath et al, *Low-rank matrix factorization for Deep Neural Network training with high-dimensional output targets*, ICASSP 2013

# Linear Bottlenecks

- GOAL: Reducing #parameters ($D \times C \geq 10^7$)
- HOW: Low-rank matrix factorisation $\rightarrow$ $\Theta_{D \times C} \approx U_{D \times r} V_{r \times C}$
- ADVANTAGE: Less parameters $D \times C \rightarrow r(D+C)$
- DISADVANTAGE: Less modelling power + non-convex optim.
  - Success depends on weights correlation
  - NOT useful for low layers

*T. Sainath et al, Low-rank matrix factorization for Deep Neural Network training with high-dimensional output targets, ICASSP 2013*

# (Iterative) Random Feature Selection

$$\vec{\Omega} = \begin{bmatrix} \vec{\omega}_1 \\ \vdots \\ \vec{\omega}_m \\ \vdots \\ \vec{\omega}_{\hat{D}} \end{bmatrix}$$

$$\vec{\omega}_m \sim p(\vec{\omega})$$

# (Iterative) Random Feature Selection

Random Fourier feature is too random!

How to draw/find better random samples/features?

$$\vec{\Omega} = \begin{bmatrix} \vec{\omega}_1 \\ \vdots \\ \vec{\omega}_m \\ \vdots \\ \vec{\omega}_{\hat{D}} \end{bmatrix} \rightarrow$$

$$K(\mathbf{x}_i, \mathbf{x}_j) \approx \hat{\phi}^T(\mathbf{x}_i)\hat{\phi}(\mathbf{x}_j)$$

$$\hat{\phi}_m(\mathbf{x}) = \sqrt{\frac{2}{\hat{D}}} cos(\vec{\omega}_m^T \mathbf{x} + b_m)$$

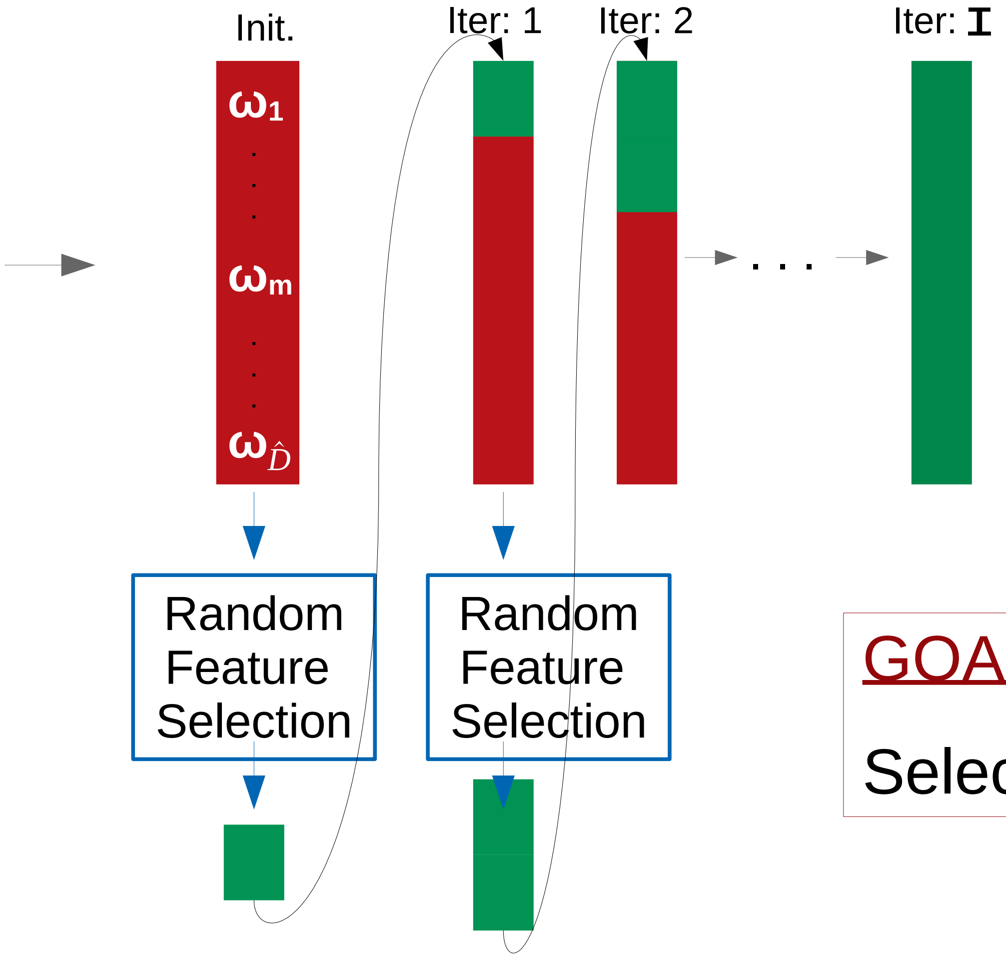
$$\vec{\omega}_m \sim p(\vec{\omega}) \qquad b \sim \mathcal{U}(0, 2\pi)$$

$$\rightarrow \hat{\phi}(x) = \begin{bmatrix} \hat{\phi}_1(x) \\ \vdots \\ \hat{\phi}_m(x) \\ \vdots \\ \hat{\phi}_{\hat{D}}(x) \end{bmatrix}$$

# (Iterative) Random Feature Selection

$$\vec{\Omega} = \begin{bmatrix} \vec{\omega}_1 \\ \vdots \\ \vec{\omega}_m \\ \vdots \\ \vec{\omega}_{\hat{D}} \end{bmatrix}$$

$$\vec{\omega}_m \sim p(\vec{\omega})$$



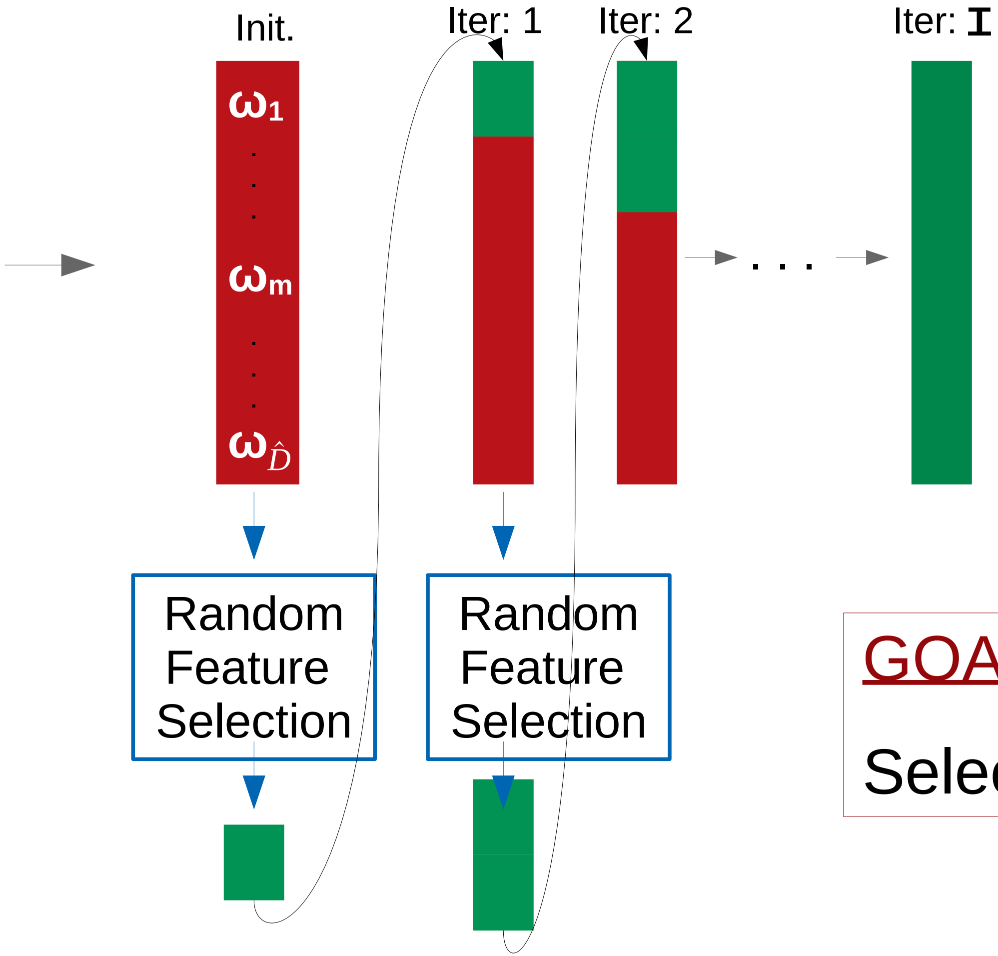

GOAL:

Select/Find best-$\hat{D}$ $\vec{\omega}_i$

# (Iterative) Random Feature Selection

$$\vec{\Omega} = \begin{bmatrix} \vec{\omega}_1 \\ \vdots \\ \vec{\omega}_m \\ \vdots \\ \vec{\omega}_{\hat{D}} \end{bmatrix}$$

$$\vec{\omega}_m \sim p(\vec{\omega})$$





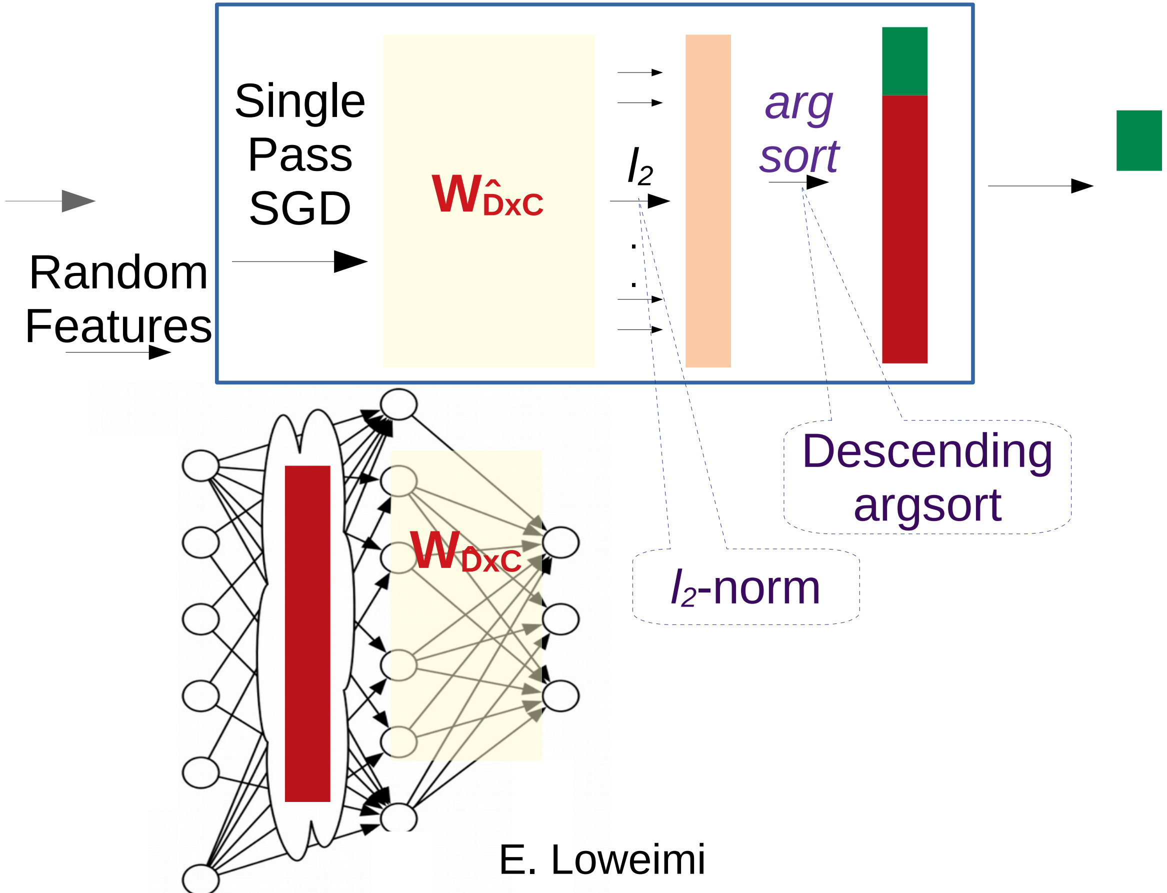
**GOAL:**

Select/Find best-$\hat{D}$ $\vec{\omega}_i$

# (Iterative) Random Feature Selection

$$\vec{\Omega} = \begin{bmatrix} \vec{\omega}_1 \\ \vdots \\ \vec{\omega}_m \\ \vdots \\ \vec{\omega}_{\hat{D}} \end{bmatrix}$$

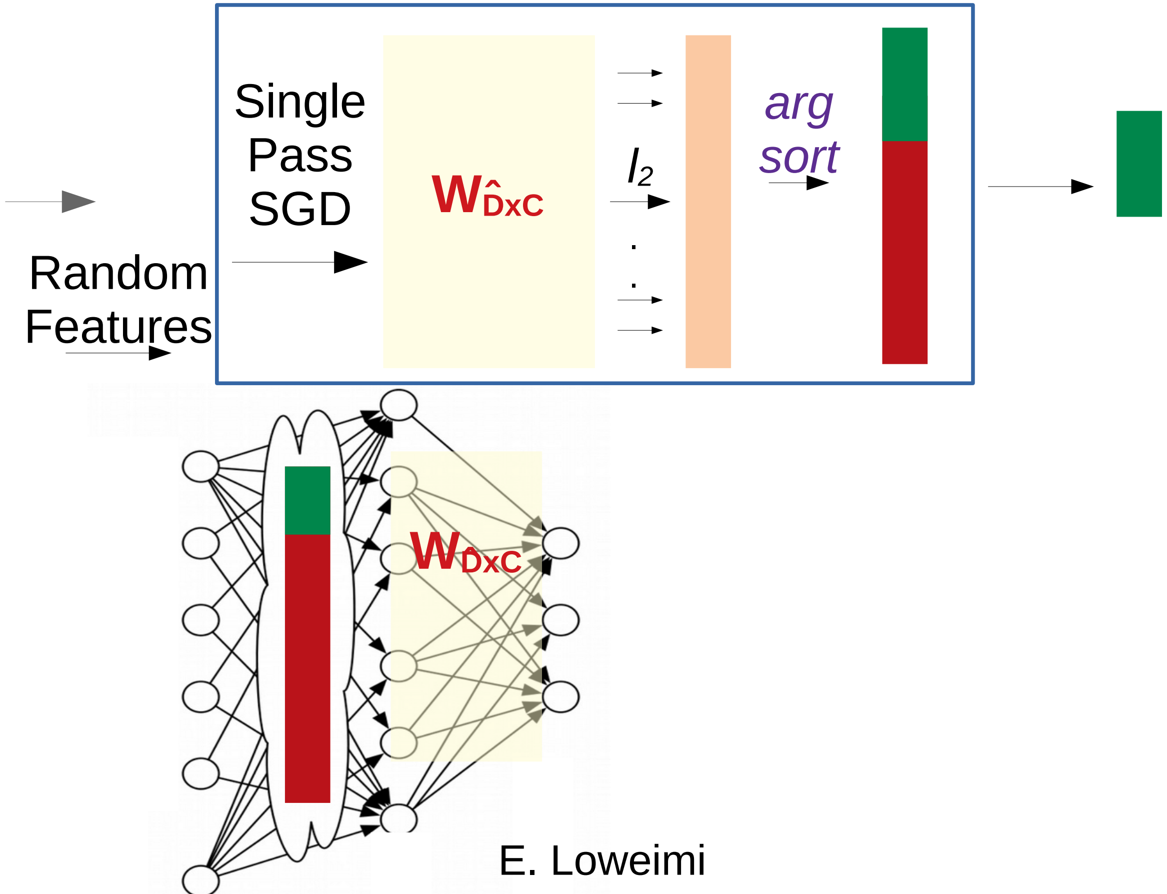
$$\vec{\omega}_m \sim p(\vec{\omega})$$



# (Iterative) Random Feature Selection

$$\vec{\Omega} = \begin{bmatrix} \vec{\omega}_1 \\ \vdots \\ \vec{\omega}_m \\ \vdots \\ \vec{\omega}_{\hat{D}} \end{bmatrix}$$

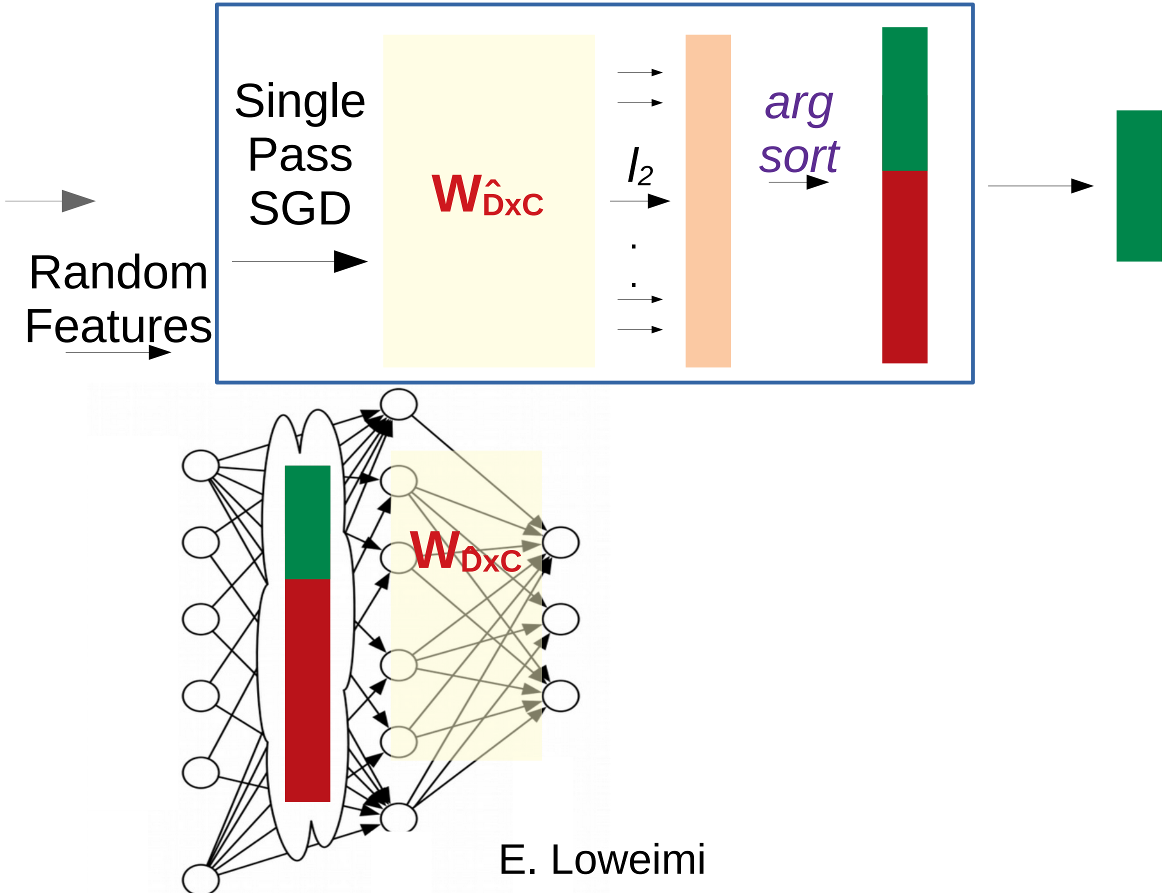
$$\vec{\omega}_m \sim p(\vec{\omega})$$



# (Iterative) Random Feature Selection

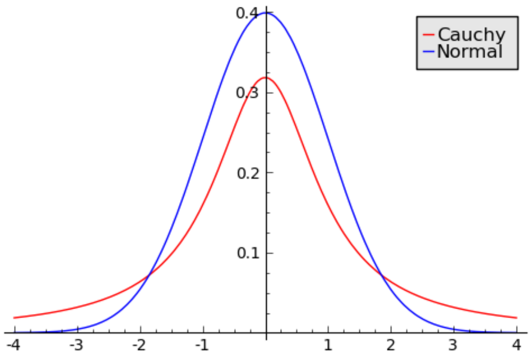
$$\vec{\Omega} = \begin{bmatrix} \vec{\omega}_1 \\ \vdots \\ \vec{\omega}_m \\ \vdots \\ \vec{\omega}_{\hat{D}} \end{bmatrix}$$

$$\vec{\omega}_m \sim p(\vec{\omega})$$



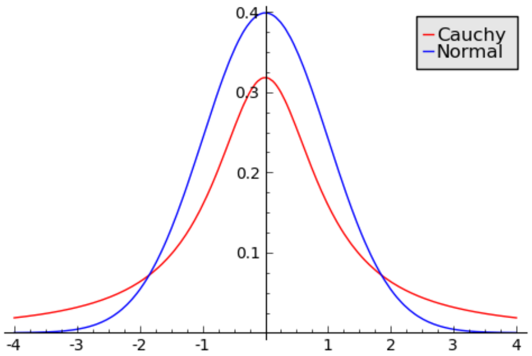
# Sparse Gaussian Kernel

- Laplacian Kernel ↔ Cauchy density → Fat tail



# Sparse Gaussian Kernel

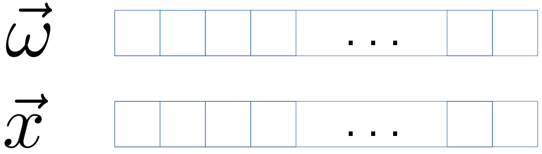
- Laplacian Kernel ↔ Cauchy density → Fat tail
- Fat tail → extreme events occur

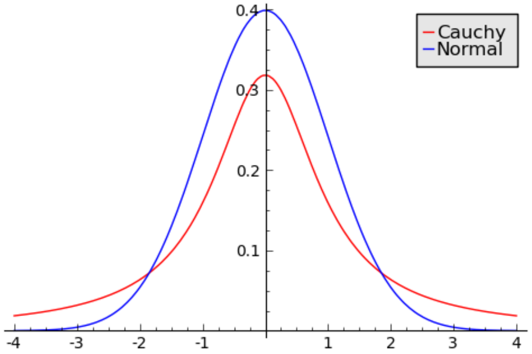


# Sparse Gaussian Kernel

- Laplacian Kernel $\leftrightarrow$ Cauchy density $\rightarrow$ Fat tail
- Fat tail $\rightarrow$ extreme events occur

$$\hat{\phi}_m(\mathbf{x}) = \sqrt{\frac{2}{\hat{D}}} cos(\vec{\omega}_m^T \mathbf{x} + b_m)$$

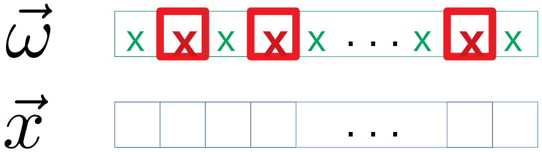


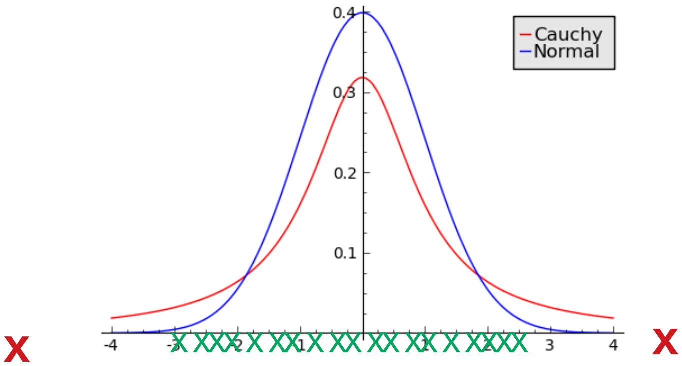


# Sparse Gaussian Kernel

- Laplacian Kernel $\leftrightarrow$ Cauchy density $\rightarrow$ Fat tail
- Fat tail $\rightarrow$ extreme events occur

$$\hat{\phi}_m(\mathbf{x}) = \sqrt{\frac{2}{\hat{D}}} cos(\vec{\omega}_m^T \mathbf{x} + b_m)$$

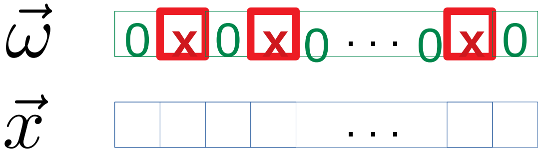


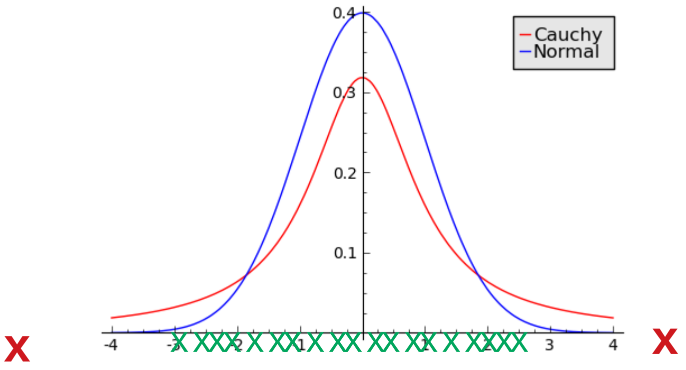



# Sparse Gaussian Kernel

- Laplacian Kernel ↔ Cauchy density → Fat tail
- Fat tail → extreme events occur → Implicit sparsity

$$\hat{\phi}_m(\mathbf{x}) = \sqrt{\frac{2}{\hat{D}}} cos(\vec{\omega}_m^T \mathbf{x} + b_m)$$

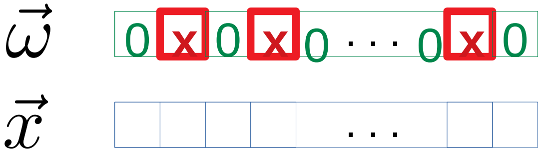


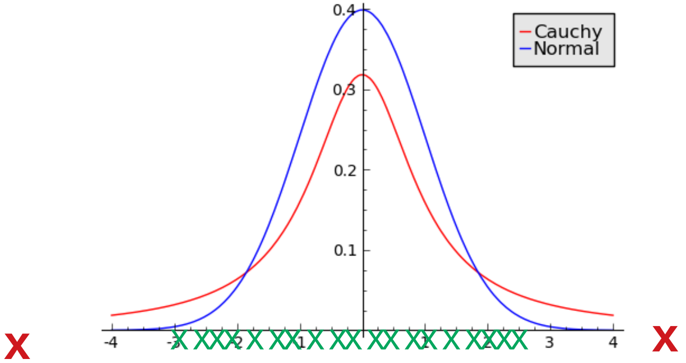



# Sparse Gaussian Kernel

- Laplacian Kernel ↔ Cauchy density → Fat tail
- Fat tail → extreme events occur → Implicit sparsity
- Explicitly impose sparsity
  - Draw *k* samples from {*1, 2, …, d*}, set rest indices to zero

$$\hat{\phi}_m(\mathbf{x}) = \sqrt{\frac{2}{\hat{D}}} cos(\vec{\omega}_m^T \mathbf{x} + b_m)$$



# CE as Early Stopping Criterion

- CE doesn't perfectly correlate with TER
  - e.g. DNNs return better TER than kernel models but worse CE

Cross Entropy

Token Error Rate

# CE as Early Stopping Criterion

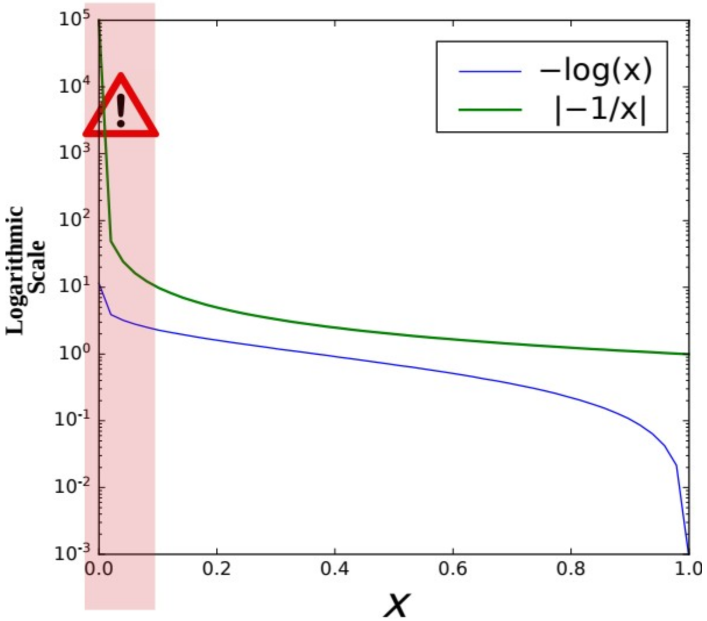
- CE doesn't perfectly correlate with TER
  - e.g. DNNs return better TER than kernel models but worse CE
- Better proxies for TER → better training

# CE as Early Stopping Criterion

- CE doesn't perfectly correlate with TER
  - e.g. DNNs return better TER than kernel models but worse CE
- Better proxies for TER → better training
- One point to look at
  - CE over-penalise very incorrect classification

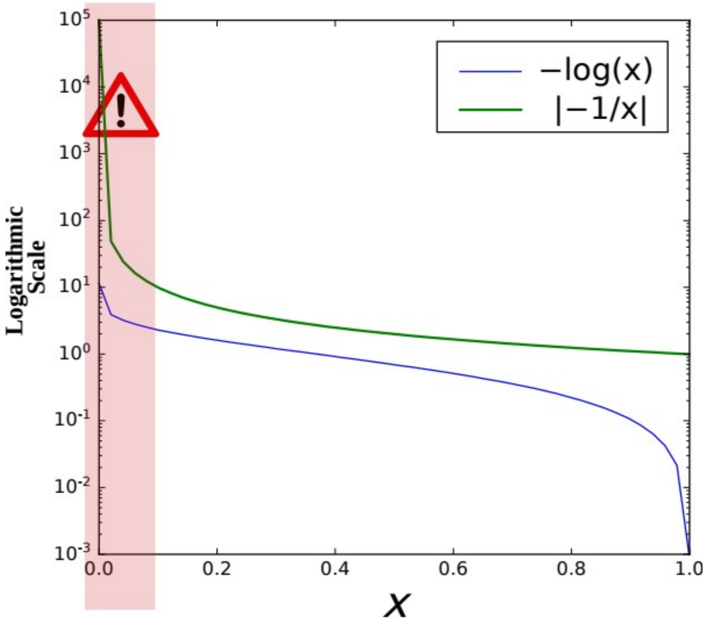
# CE as Early Stopping Criterion

- CE doesn't perfectly correlate with TER
  - e.g. DNNs return better TER than kernel models but worse CE
- Better proxies for TER → better training
- One point to look at
  - CE over-penalise very incorrect classification
  - Miss is more costly than False Alarm



# CE as Early Stopping Criterion

- CE doesn’t perfectly correlate with TER
  - e.g. DNNs return better TER than kernel models but worse CE
- Better proxies for TER → better training
- One point to look at
  - CE over-penalise very incorrect classification
  - Example → Incorrect labels

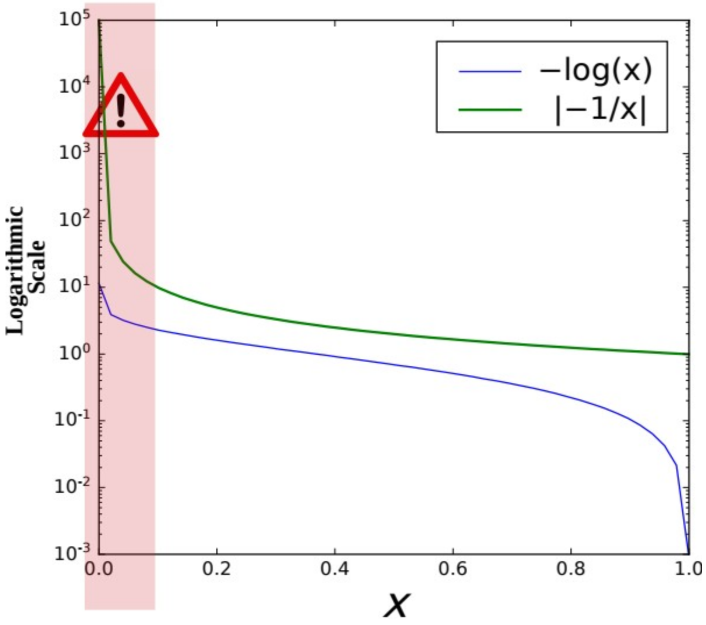


# Proposed Early Stopping Criteria

Entropy Regularised Log Loss

$$ERLL = CE + \beta \; ENT$$

$$= -\frac{1}{N}\sum_{i=1}^{N}\sum_{y=1}^{C} \left[\mathbb{I}(y = y_i) + \beta p(y|x_i)\right] log(p(y|x_i))$$



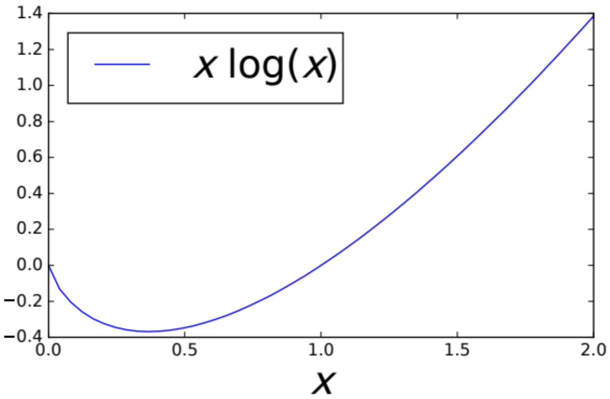
# Proposed Early Stopping Criteria

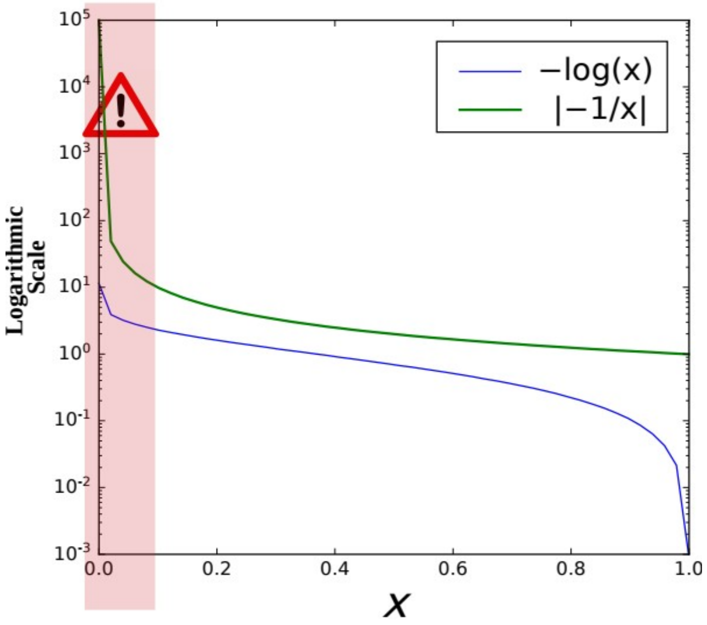
Entropy Regularised Log Loss

$$ERLL = CE + \beta\ ENT$$

$$= -\frac{1}{N}\sum_{i=1}^{N}\sum_{y=1}^{C}\left[\mathbb{I}(y = y_i) + \beta p(y|x_i)\right] log(p(y|x_i))$$

Avoids over-penalisation when $p(y|x_i) \rightarrow 0$



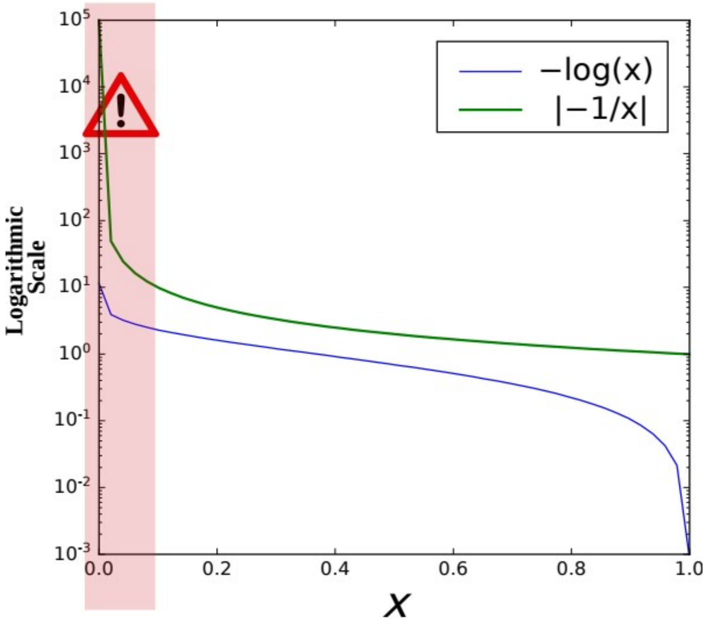
# Proposed Early Stopping Criteria

Entropy Regularised Log Loss

$$ERLL = CE + \beta\ ENT$$

$$= -\frac{1}{N}\sum_{i=1}^{N}\sum_{y=1}^{C}\left[\mathbb{I}(y = y_i) + \beta p(y|x_i)\right] log(p(y|x_i))$$

$$\text{Capped Log Loss} = -\frac{1}{N}\sum_{i=1}^{N} log(p(y_i|x_i) + \lambda)$$

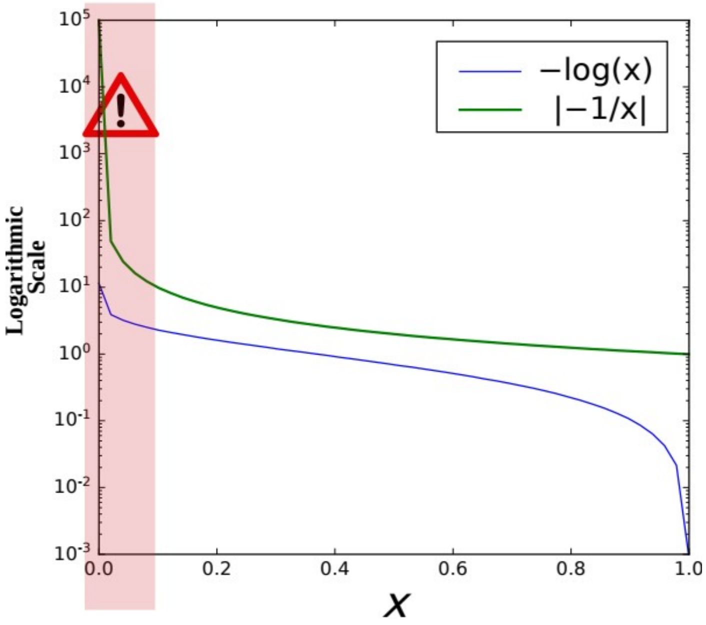


# Proposed Early Stopping Criteria

Entropy Regularised Log Loss

$$ERLL = CE + \beta \ ENT$$

$$= -\frac{1}{N}\sum_{i=1}^{N}\sum_{y=1}^{C}\left[\mathbb{I}(y = y_i) + \beta p(y|x_i)\right] log(p(y|x_i))$$




$$\text{Capped Log Loss} = -\frac{1}{N}\sum_{i=1}^{N} log(p(y_i|x_i) + \lambda)$$

$$\text{Top-k Log Loss} = -\frac{1}{k}\sum_{i=1}^{k} log(p(y_i|\mathbf{x}_i))$$

# Experimental Results

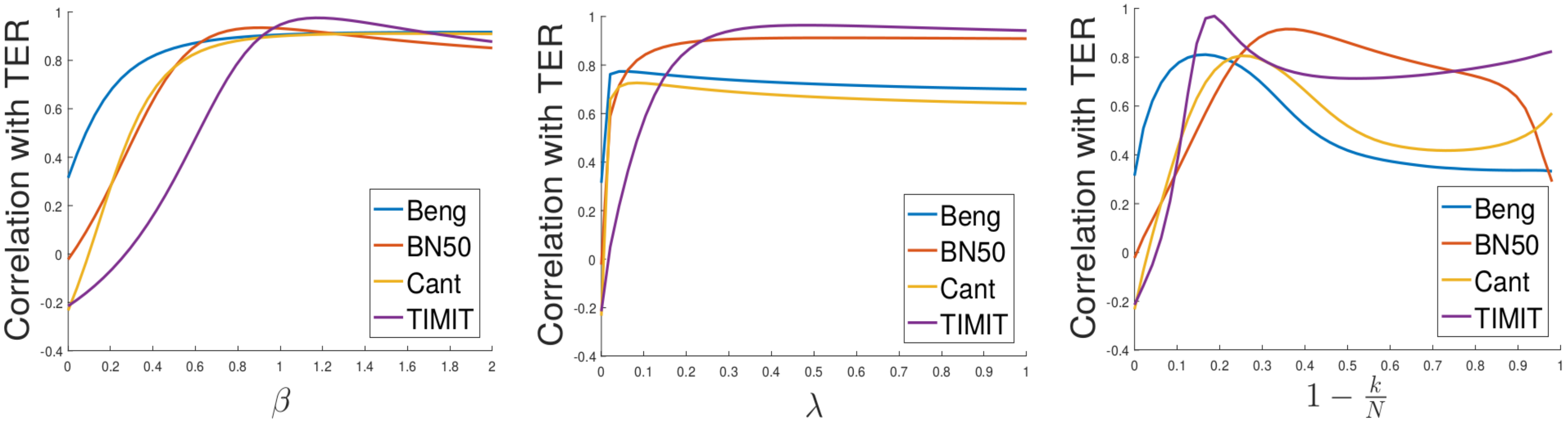
# Experimental Setup

- Initialisation → Glorot and Bengio (2010)
  - Biases: zero, Weights: random uniform ($n_j$: #nodes in layer $j$)

$$\mathcal{U}[-b, b] \leftarrow b = \frac{\sqrt{6}}{\sqrt{n_j + n_{j+1}}}$$

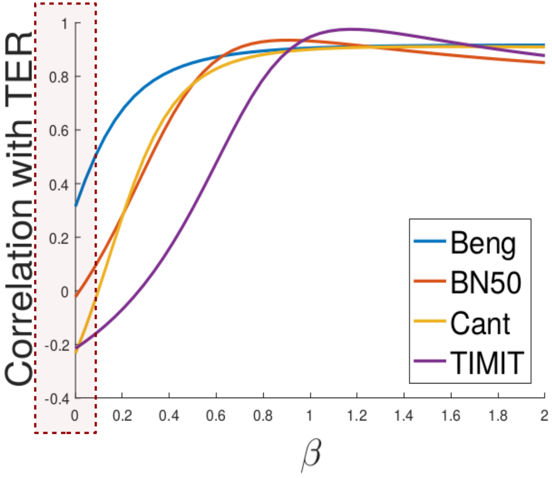
- DNN architecture: 4 hidden layers (#nodes: 1k→4k), tanh activation
- Trainig: SGD, mini-batch size: 250, learning rate annealing (halve it at the end of epoch if ΔCE{Heldout} < 1%)
- Each test set divided into training set, held-out (hyper-parameter adjustment), dev set (LMSF and WIP adjustment) and test set (no speaker overlap between sets)
- Decoding→ IBM'S Attila speech recognition toolkit
- Feature extraction:
  - 25 ms, 10 ms [TIMIT 5 ms], 13-dim PLP
  - speaker-based MVN, splice 9 frames → LDA → 40D → STC transform
  - Final feature: 360 (4x2+1 x 40) [TIMIT: 440 5x2+1 x 40]
- #Classes: context-dependent HMM state-clustered quinphones
  - Bengali and Cantonese = 1k, BN = 5k
  - TIMIT = 147 = 3 x 49 ↔ beginning, middle and end of 49 phonemes

# Correlation with TER

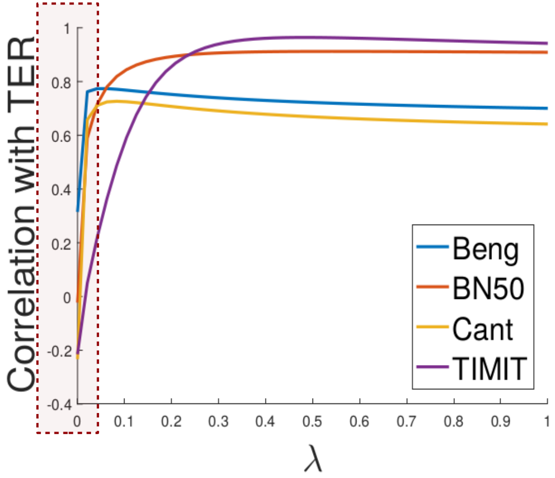



ERLL | Capped log loss | Top-k

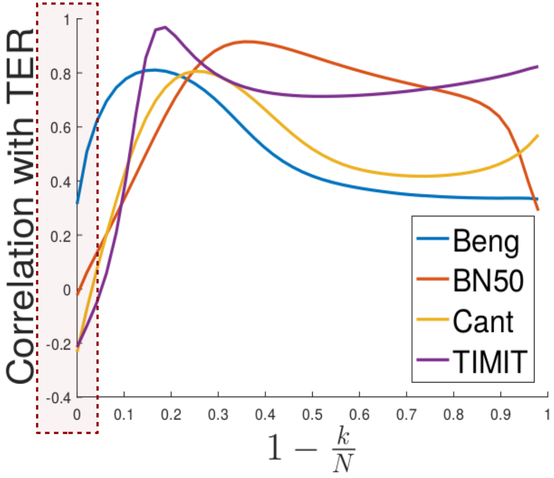
# Correlation with TER




ERLL



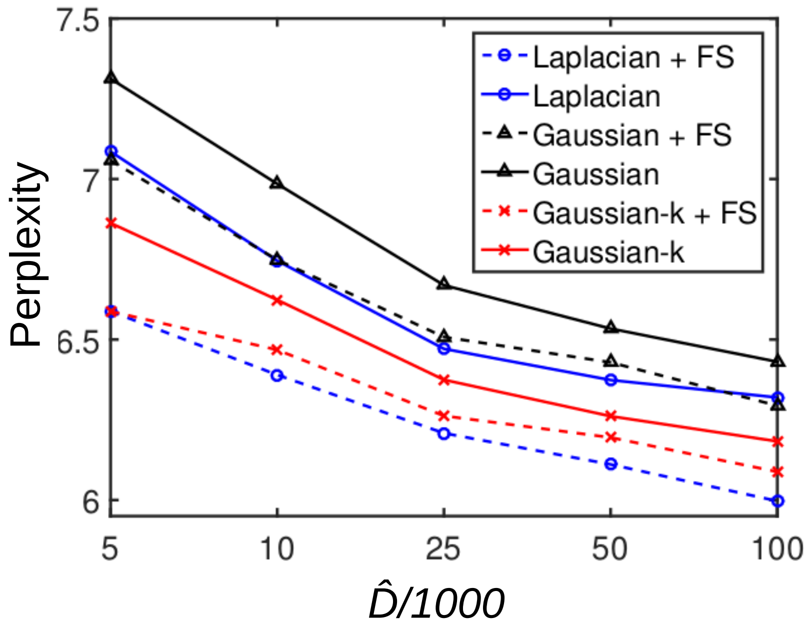

Capped log loss

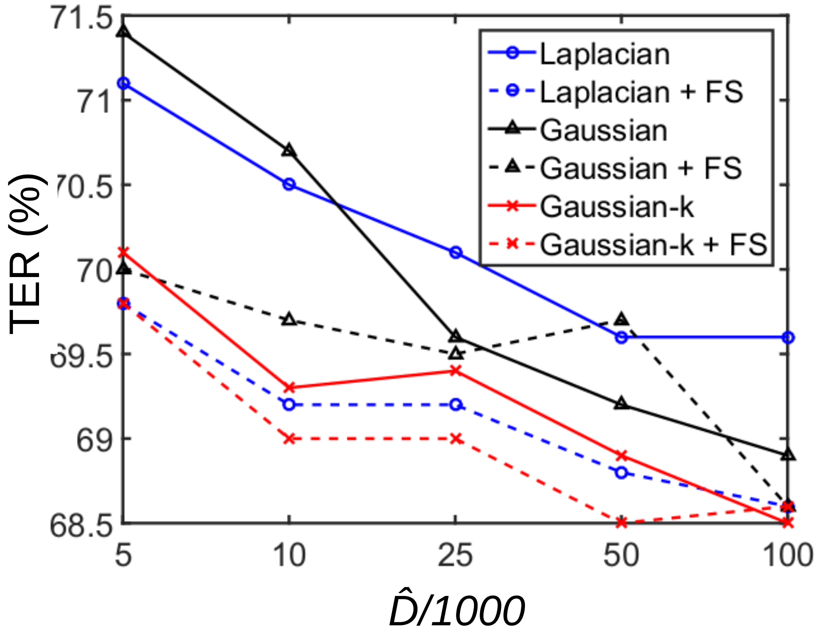



Top-k

Cross entropy and TER correlation
→ metric parameter $\{\beta, \lambda, \kappa\}$ → 0

# Effects of Kernel Type and FS

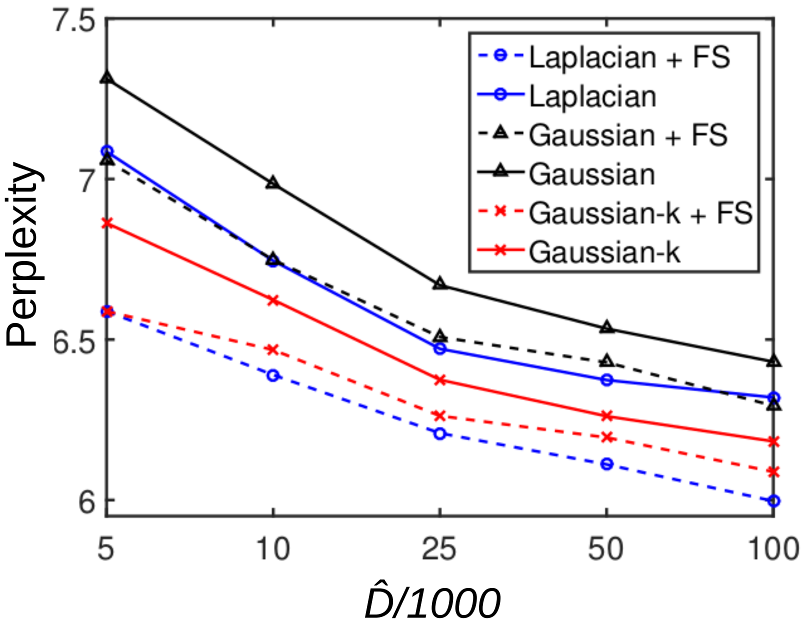


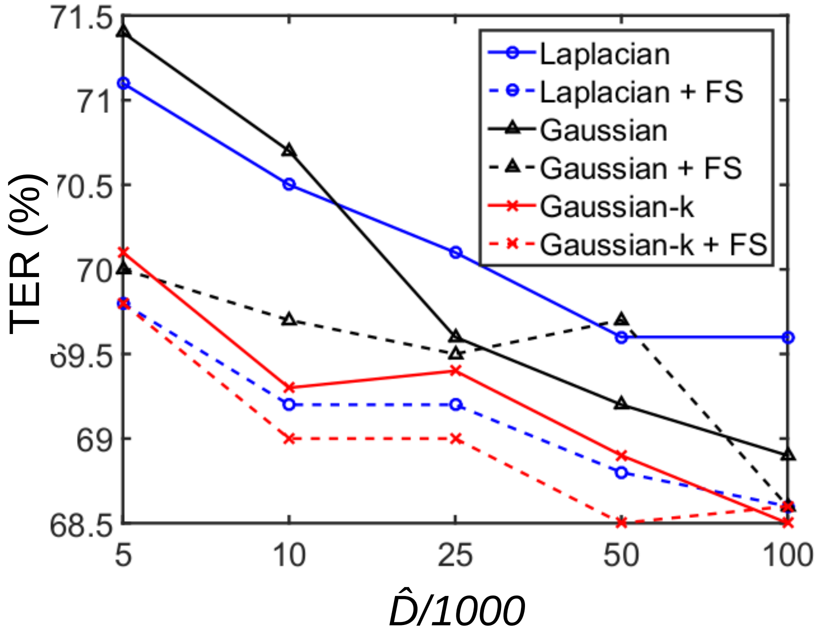




$$Perplexity = exp(-\frac{1}{M}\sum_{m=1}^{M} log(p(y_m|x_m))$$

-- M: size of the held-out set

# Effects of Kernel Type and FS




$$Perplexity = exp(-\frac{1}{M}\sum_{m=1}^{M} log(p(y_m|x_m)))$$

-- M: size of the held-out set

Better than

-- Gaussian-k > Lapalacian > Gaussian

-- FS helps

# Experimental Results (TER)

| Dataset | Method | Perplexity | Collapsed | TER |
|---|---|---|---|---|
| Cant. | DNN | 6.127 | 4.316 | **67.3%** |
| | Lap+FS | **5.997** | **4.176** | 68.6% |
| Beng. | DNN | **3.616** | 3.256 | **71.3%** |
| | Lap+FS | 3.678 | **3.233** | 72.7% |

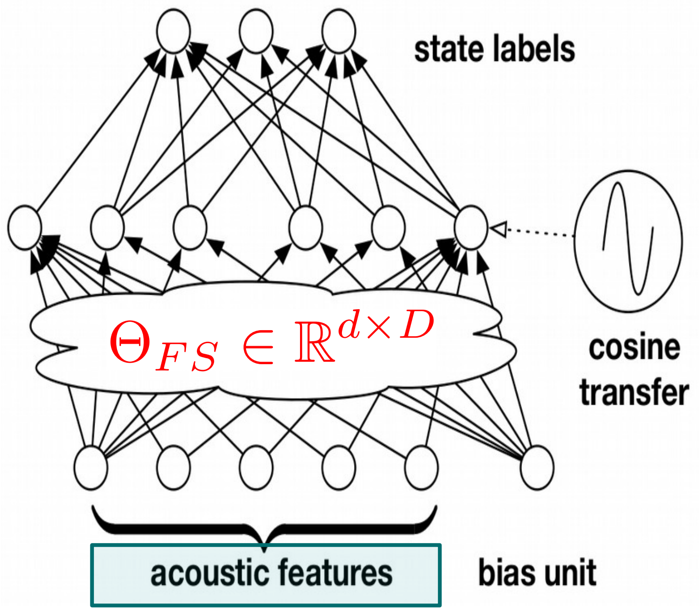
| | | Test TER (DNN) | Test TER (Kernel) |
|---|---|---|---|
| TIMIT | *Huang et al* | 20.5 | 21.3 |
| | Lap+FS | 20.5 | **20.4** |

- FS: proposed feature selection
- Lap: Laplace kernel
- Collapsed: treat all silence states as one

*Huang et al, Kernel methods match deep neural networks on TIMIT, 2014*

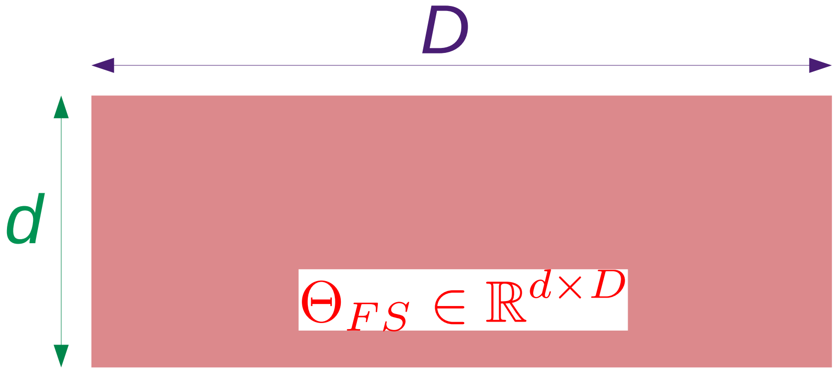
# Relative Weight of Input Features in Random Matrix

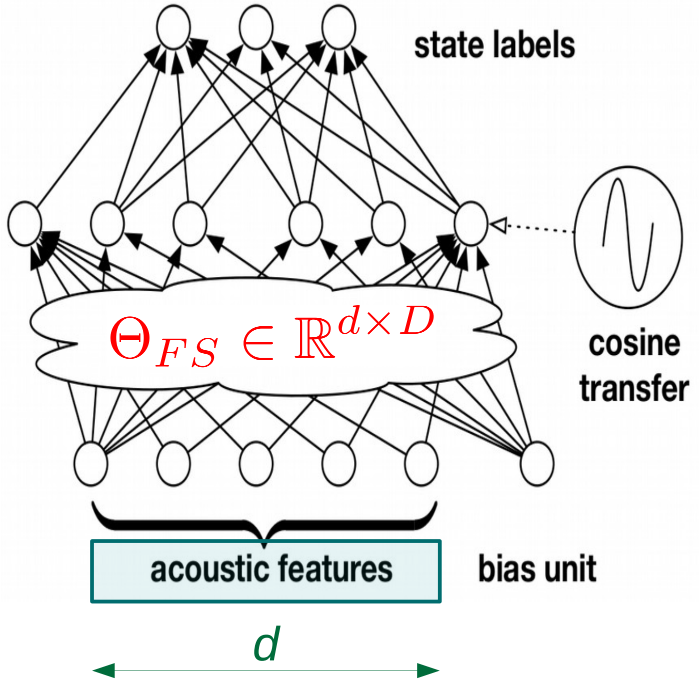
$$R_i = \frac{\sum_{j=1}^{D} |\Theta_{FS}[i,j]|}{\sum_i \sum_j |\Theta_{FS}[i,j]|}$$



# Relative Weight of Input Features in Random Matrix

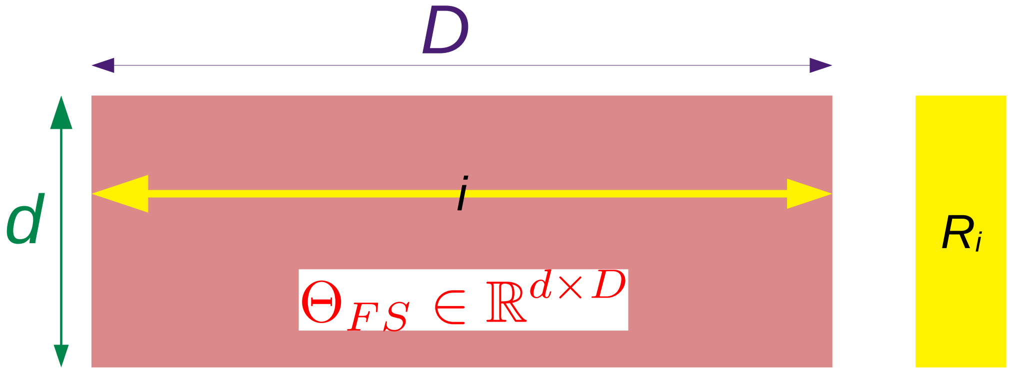
$$R_i = \frac{\sum_{j=1}^{D} |\Theta_{FS}[i,j]|}{\sum_i \sum_j |\Theta_{FS}[i,j]|}$$

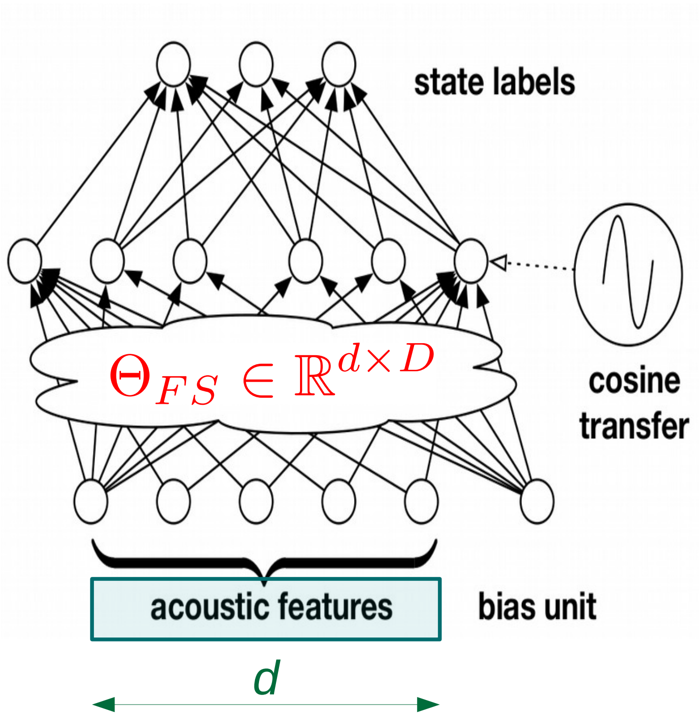


# Relative Weight of Input Features in Random Matrix

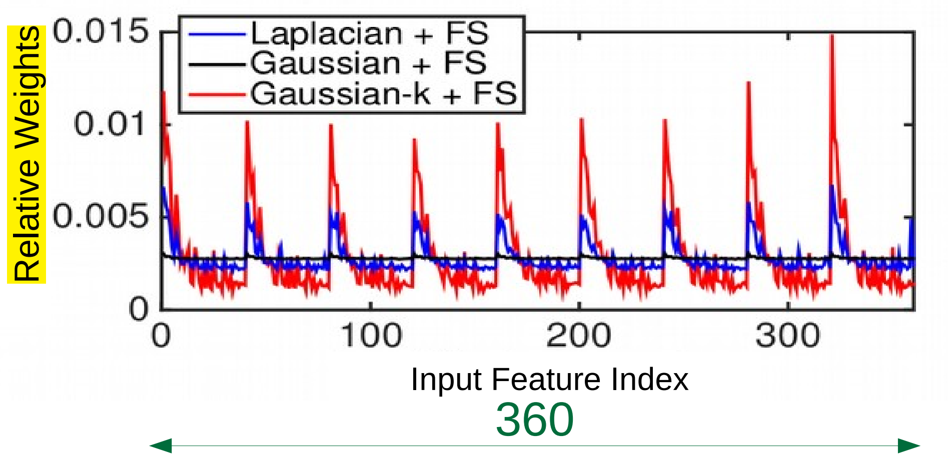
$$R_i = \frac{\sum_{j=1}^{D} |\Theta_{FS}[i,j]|}{\sum_i \sum_j |\Theta_{FS}[i,j]|}$$

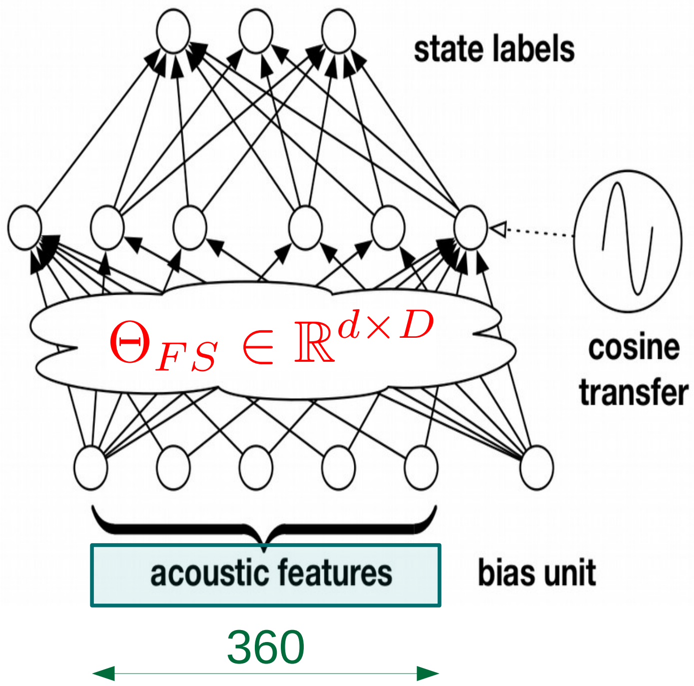


# Relative Weight of Input Features in Random Matrix

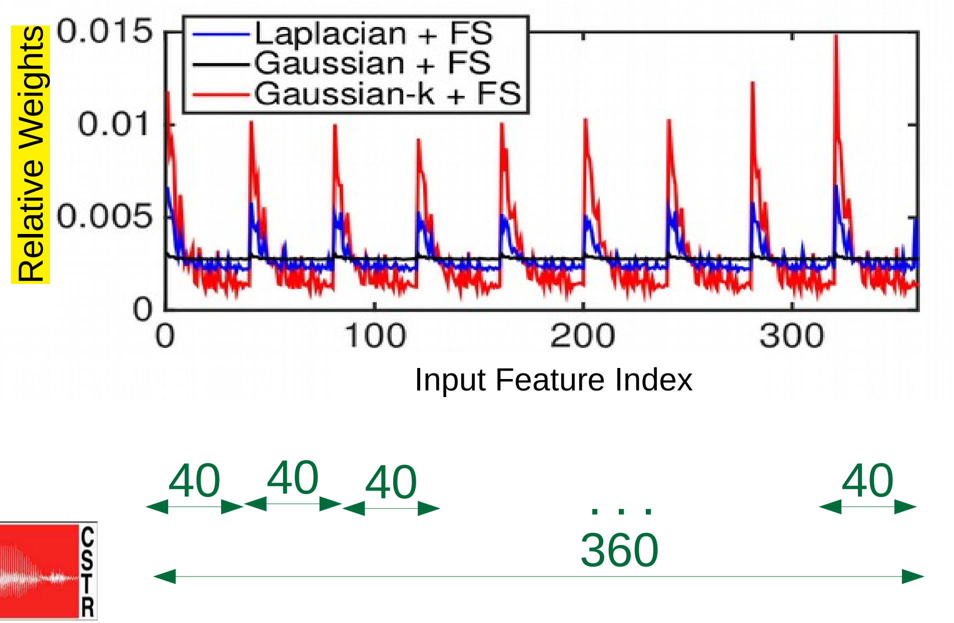
$$R_i = \frac{\sum_{j=1}^{D} |\Theta_{FS}[i,j]|}{\sum_i \sum_j |\Theta_{FS}[i,j]|}$$

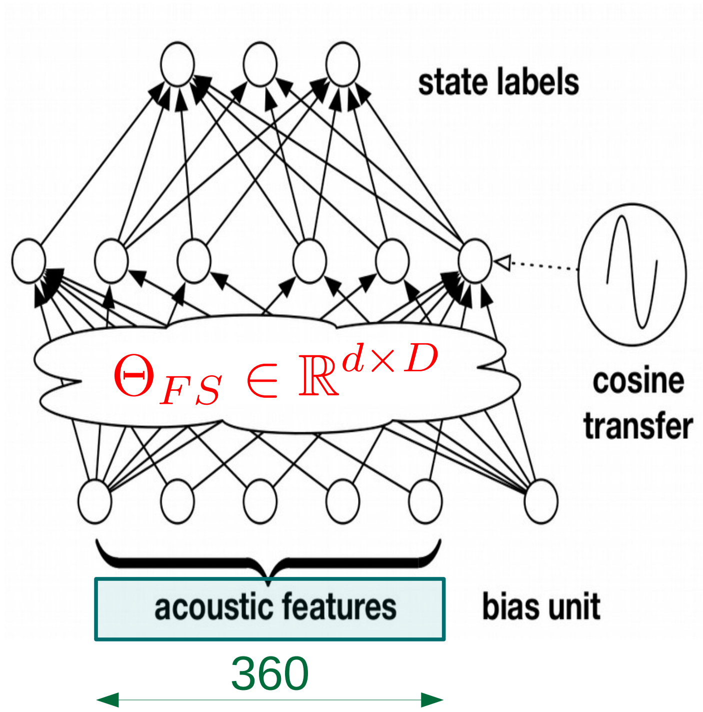


# Relative Weight of Input Features in Random Matrix

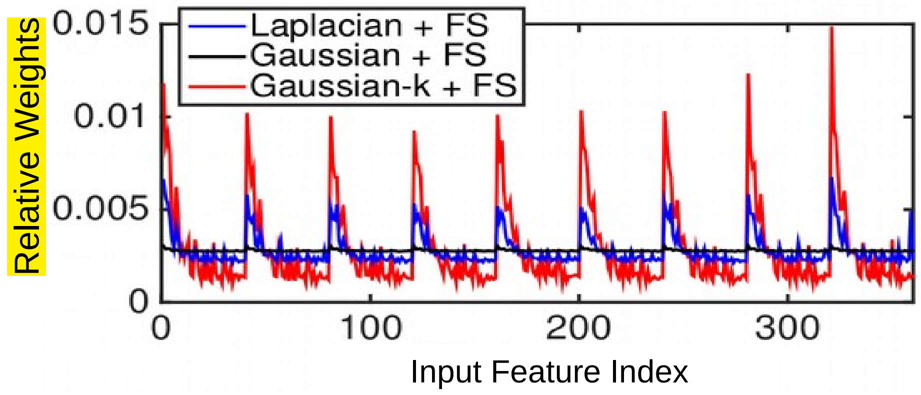
$$R_i = \frac{\sum_{j=1}^{D} |\Theta_{FS}[i,j]|}{\sum_i \sum_j |\Theta_{FS}[i,j]|}$$

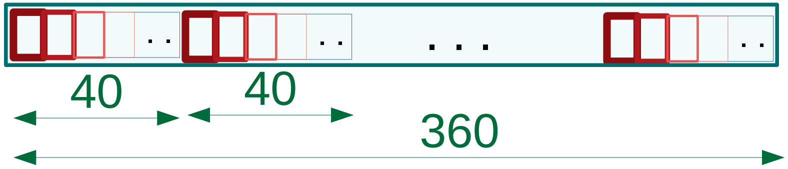


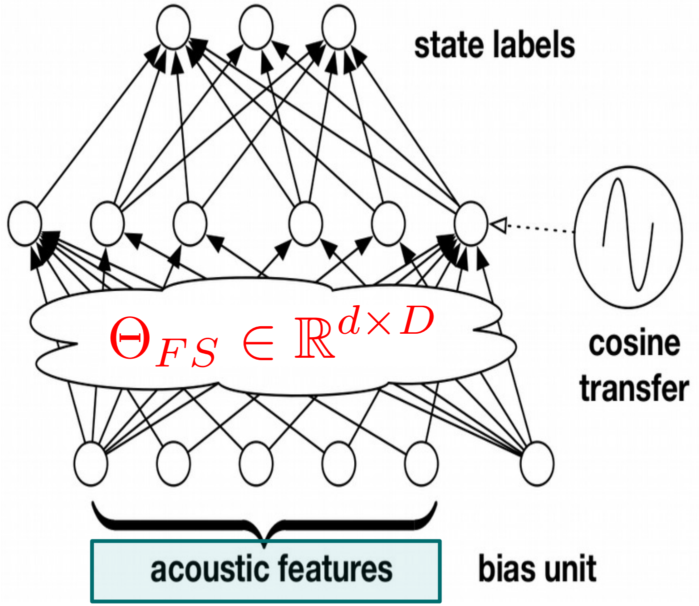



# Relative Weight of Input Features in Random Matrix

$$R_i = \frac{\sum_{j=1}^{D} |\Theta_{FS}[i,j]|}{\sum_i \sum_j |\Theta_{FS}[i,j]|}$$




LDA **ranks** its axes, like PCA

# Wrap-up

- Kernel machines
  - ADVANTAGES: handles non-linear data, interpretable, learning guarantees
  - DISADVANTAGE: Do not scale well
  - SOLUTIONS: Approximate kernel matrix or kernel function
- Novelties
  - Scale-up kernel methods to LVCSR level + comparable results with DNN
  - Random feature selection (0.2 → 1.6 WER↓)
  - Frame-level metrics (0→ 0.7 WER↓)
  - Linear bottleneck (0.9 → 2.4 WER↓)

# That's it!

- Thanks for your attention
- Q & A

# Appendices

- Volume/Surface of Hyper-Sphere
- Dataset
- Kernel Results
- DNN Results
- Nyström vs Random Fourier Features

# Volume of Hypersphere (n-ball)

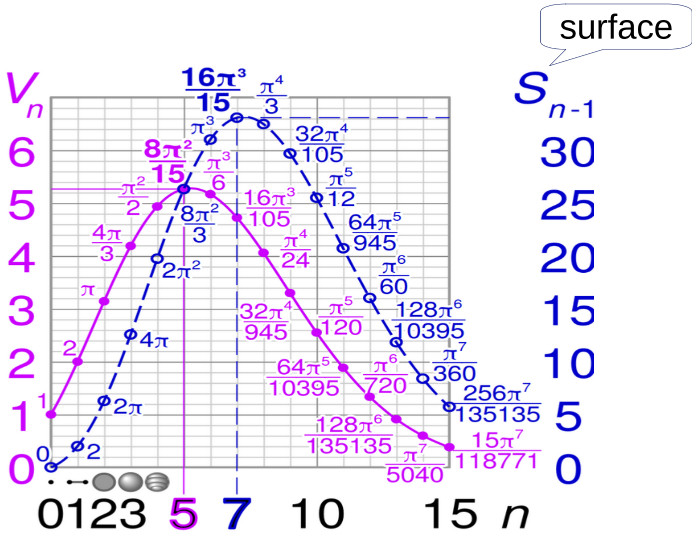
Volume

Radius

Gamma function (factorial)
Growth faster than exponential

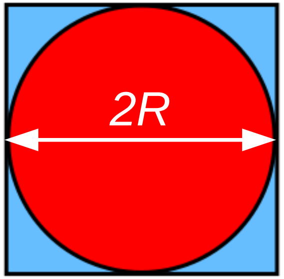
$$V_n(R) = \frac{\pi^{\frac{n}{2}}}{\Gamma(\frac{n}{2}+1)} R^n$$

dimension




$$\lim_{n\to\infty} \frac{\text{Volume of hypersphere in } \mathbb{R}^n}{\text{Volume of hypercube in } \mathbb{R}^n} = \lim_{n\to\infty} \frac{\frac{\pi^{\frac{n}{2}}}{\Gamma(\frac{n}{2}+1)} R^n}{(2R)^n}$$

$$= \lim_{n\to\infty} \frac{\pi^{\frac{n}{2}}}{2^n\,\Gamma(\frac{n}{2}+1)} = 0$$




$n \to \infty$
**Red** $\to 0$

https://en.wikipedia.org/wiki/Volume_of_an_n-ball

# Experimental Setup – Dataset/TER

| Dataset | Train | Heldout | Dev | Test | # Features | # Classes |
|---|---|---|---|---|---|---|
| Beng. | 21 hr (7.7M) | 2.8 hr (1.0M) | 20 hr (7.1M) | 5 hr (1.7M) | 360 | 1000 |
| BN-50 | 45 hr (16M) | 5 hr (1.8M) | 2 hr (0.7M) | 2.5 hr (0.9M) | 360 | 5000 |
| Cant. | 21 hr (7.5M) | 2.5 hr (0.9M) | 20 hr (7.2M) | 5 hr (1.8M) | 360 | 1000 |
| TIMIT | 3.2 hr (2.3M) | 0.3 hr (0.2M) | 0.15 hr (0.1M) | 0.15 hr (0.1M) | 440 | 147 |

- Performance Measure → Token Error Rate (TER)
  - WER for Bengali and BN-50
  - CER (character error rate) for Cantonese
  - PER (phone error rate) for TIMIT

# Experimental Results -- Kernel

| | Laplacian | | | | Gaussian | | | | Sparse Gaussian | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | NT | B | R | BR | NT | B | R | BR | NT | B | R | BR |
| Beng. | 74.5 | 72.1 | 74.5 | 71.4 | 72.6 | 72.0 | 72.6 | 71.8 | 73.0 | 71.5 | 73.0 | **70.9** |
| +FS | 72.9 | 71.1 | 72.8 | 70.4 | 74.1 | 71.4 | 74.2 | **70.3** | 72.9 | 71.2 | 72.8 | 70.7 |
| BN-50 | N/A | 17.9 | N/A | 17.7 | N/A | 17.3 | N/A | 17.1 | N/A | 17.3 | N/A | **17.0** |
| +FS | N/A | 17.1 | N/A | **16.7** | N/A | 17.5 | N/A | 17.0 | N/A | 17.1 | N/A | **16.7** |
| Cant. | 69.9 | 68.2 | 69.2 | 67.4 | 70.2 | 67.6 | 70.0 | **67.1** | 68.6 | 67.5 | 68.1 | **67.1** |
| +FS | 68.4 | 67.5 | 68.5 | **66.7** | 69.9 | 67.7 | 69.8 | 66.9 | 68.6 | 67.4 | 68.5 | 66.8 |
| TIMIT | 20.6 | 19.2 | 20.4 | 18.9 | 19.8 | 18.9 | 19.6 | 18.6 | 19.9 | 18.8 | 19.6 | **18.4** |
| +FS | 19.5 | 18.6 | 19.3 | 18.4 | 19.5 | 18.6 | 19.4 | 18.4 | 19.3 | 18.4 | 19.1 | **18.2** |

-- NT: No Trick
-- B: linear Bottleneck
-- R: ERLL
-- BR: using B & R

# Experimental Results -- DNN

#nodes hidden layer →

| | 1000 | | | | 2000 | | | | 4000 | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | NT | B | R | BR | NT | B | R | BR | NT | B | R | BR |
| Beng. | 72.3 | 71.6 | 71.7 | 70.9 | 71.5 | 71.1 | 70.7 | 70.3 | 71.1 | 70.6 | 70.5 | **70.2** |
| BN-50 | 18.0 | 17.3 | 17.8 | 17.1 | 17.4 | 16.7 | 17.1 | **16.4** | 16.8 | 16.7 | 16.7 | 16.5 |
| Cant. | 68.4 | 68.1 | 67.9 | 67.5 | 67.7 | 67.7 | 67.2 | **67.1** | 67.7 | **67.1** | 67.2 | 67.2 |
| TIMIT | 19.5 | 19.3 | 19.4 | 19.2 | 19.0 | 18.9 | 19.2 | 19.2 | **18.6** | **18.6** | 18.7 | 18.9 |

– #hidden-layers: 4
– NT: No Trick
– B: linear Bottleneck
– R: Entorpy Regularised Log Loss
– BR: using both B & R

# Nyström vs Random Fourier Features

- Kernel matrix approximation
  - Nyström approximation
    - Data-dependent

- Kernel function approximation
  - Random Fourier Features
    - Data independent

- Large eigengap ($\lambda_{max}$ - $\lambda_{min}$) in kernel matrix → Difference is highlighted
  - Nyström → lower generalisation error
  - Random Fourier method requires many sample to discover subspace spanned by top eigenvectors