Being *Shallow* and *Random* is (almost) as *Good* as Being *Deep* and *Thoughtful*

Fei Sha

Joint work with collaborators from IBM and Columbia

Learning representation

Motivation

Deep neural networks

Kernel methods

Motivating toy example: face recognition

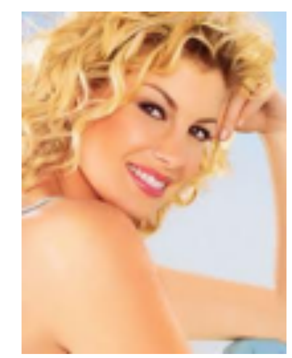
Dude

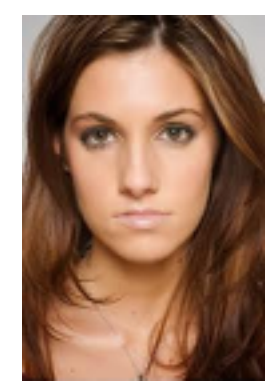
Lady

Step 1: collect labeled images as ***training*** data

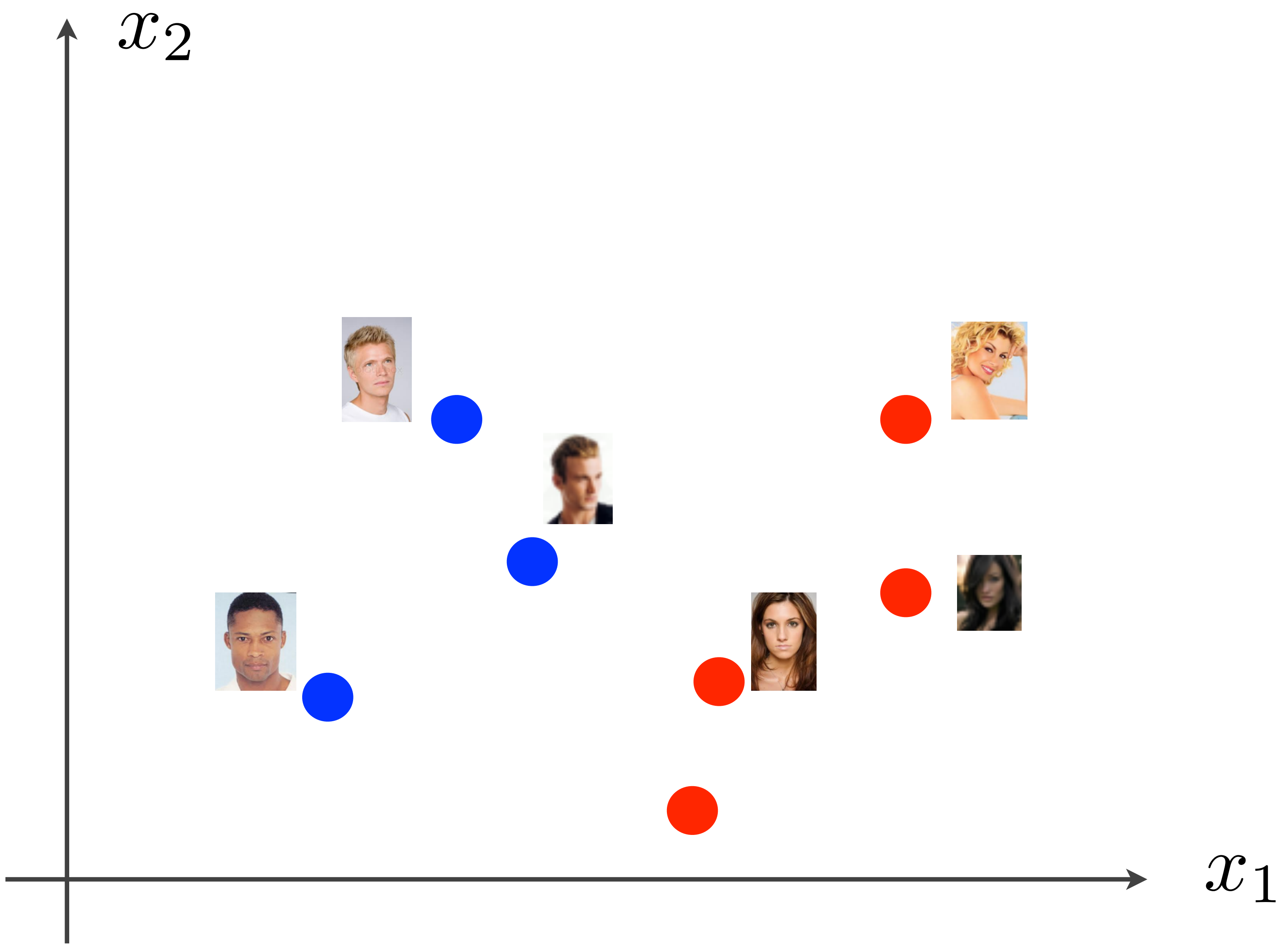
Dude

Lady

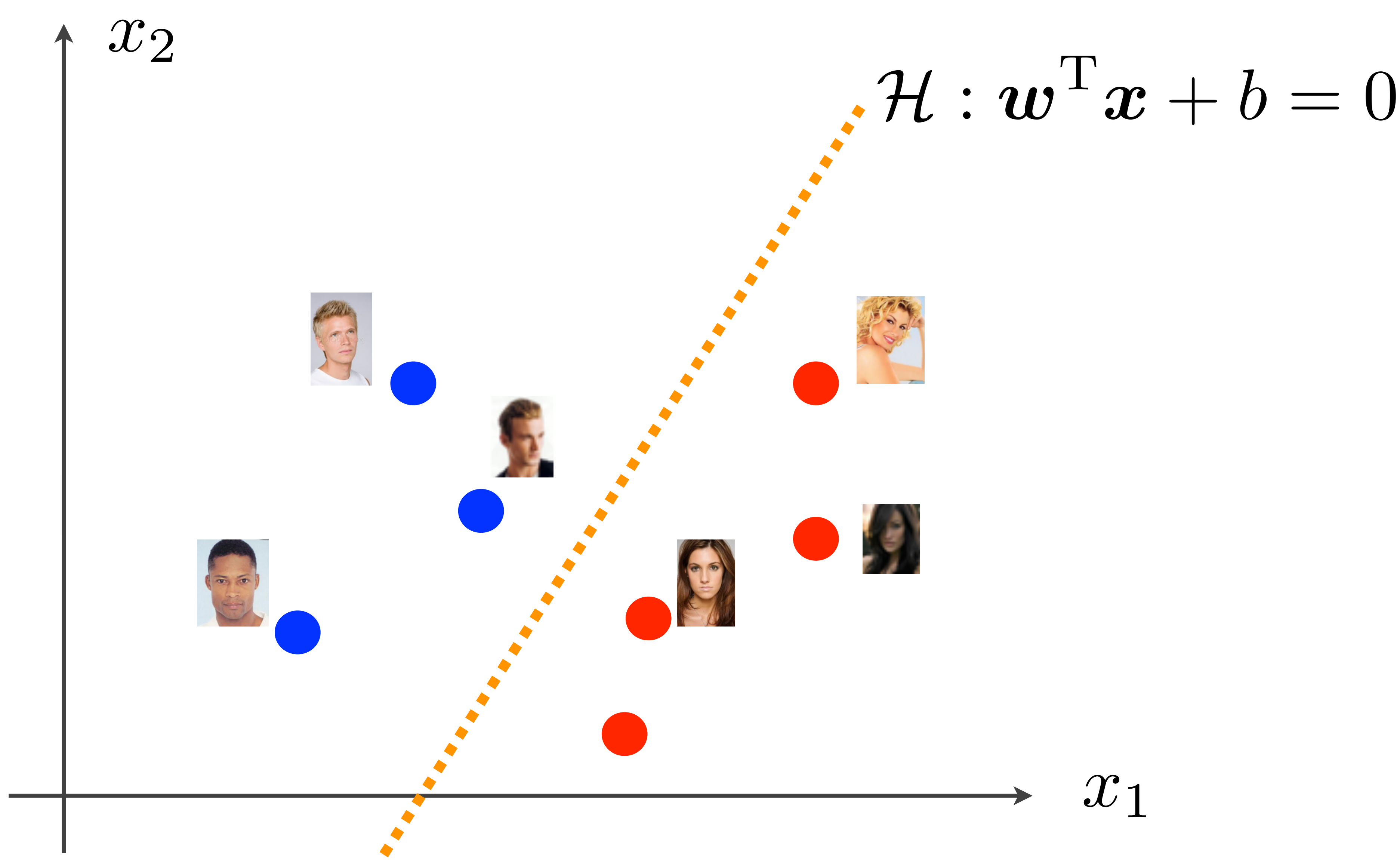




Step 2: ***represent*** each image as a point



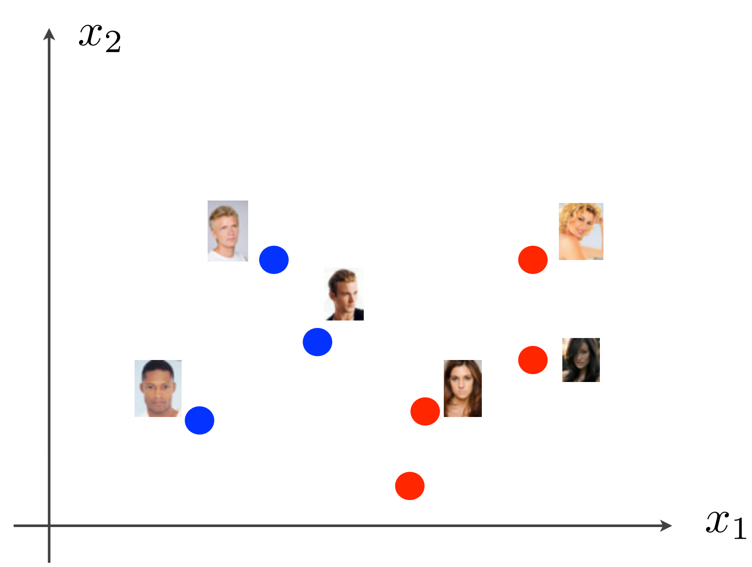
Step 3: ***fit*** a model (decision boundary)



Not so simple: key question *unanswered*!

How?

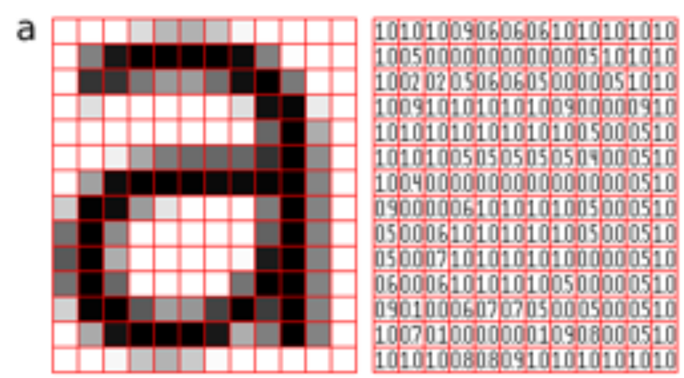
Step 2: ***represent*** each image as a point



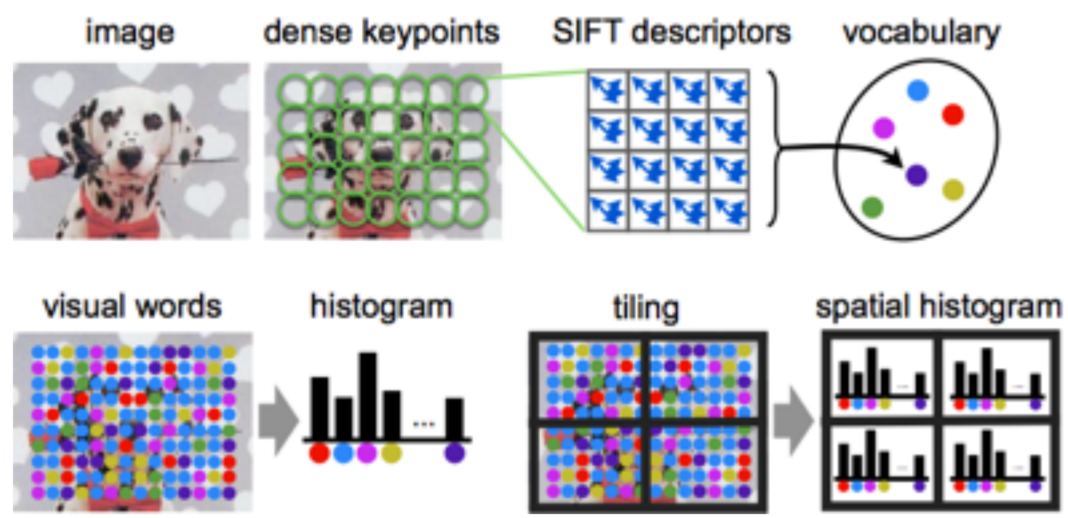
U. of Texas, Austin, April 12, 2016

Many choices, but which is ***better*** ?

Simple: raw image pixel values



Complex: Bag of visual words from SIFT descriptors



[Visual Geometry Group, Oxford]

Hard to get good (or optimal) representation

Past: ***major bottleneck***

An art known as “feature engineering”

Often laborious and manual process

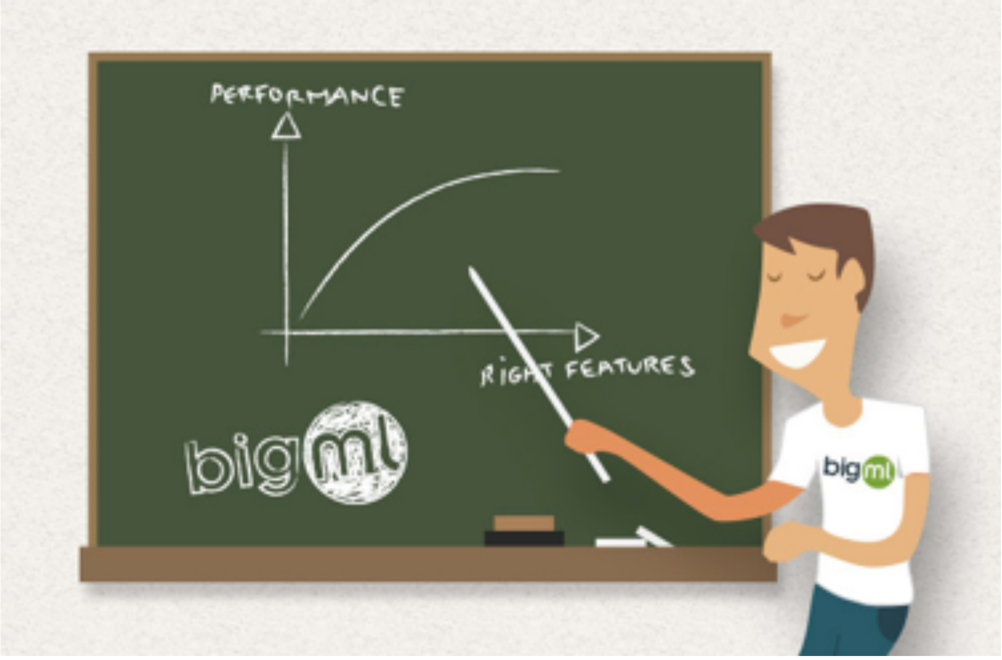
Present: ***even more so***

as intuition and manual inspection fail

facing a large amount of data

modeling high-dimensional data

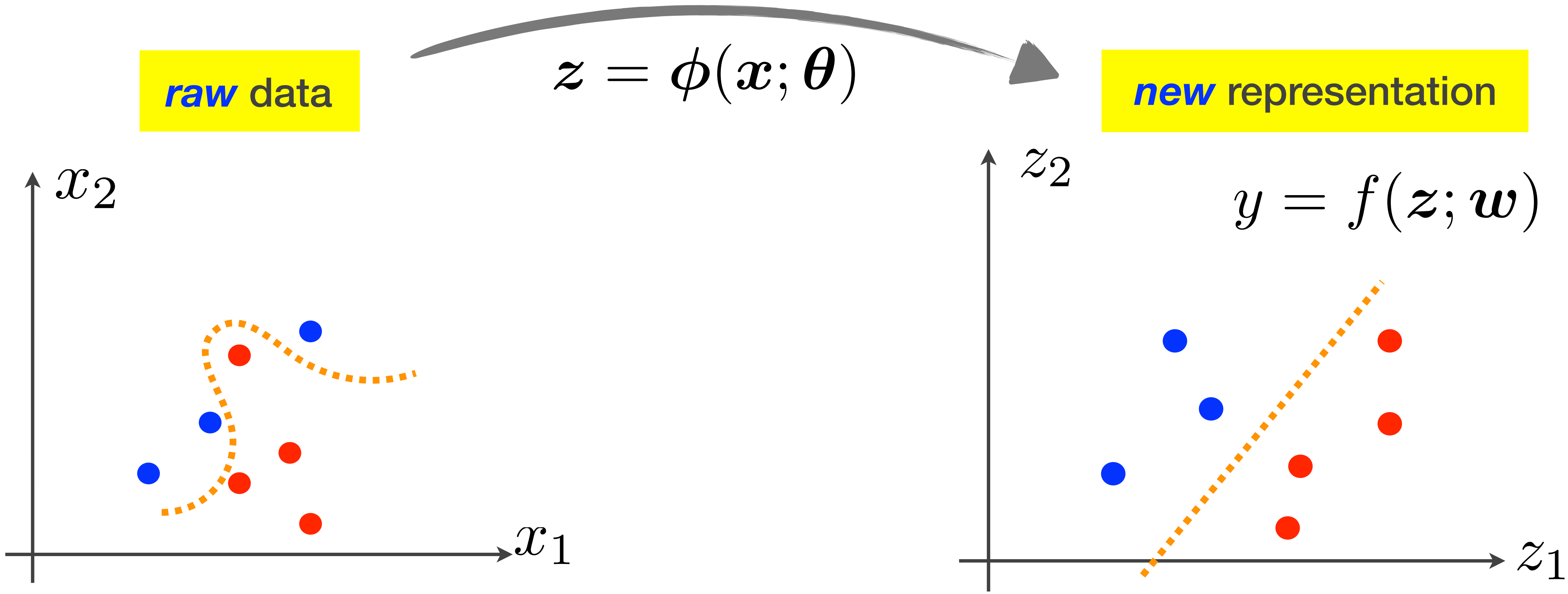
disentangling many latent factors



“easily the most important factor”

[P. Domingos]

Representation as a learning problem



Classifier in the ***old*** space can be ***complex***

Classifier in the ***new*** space can be ***simple***

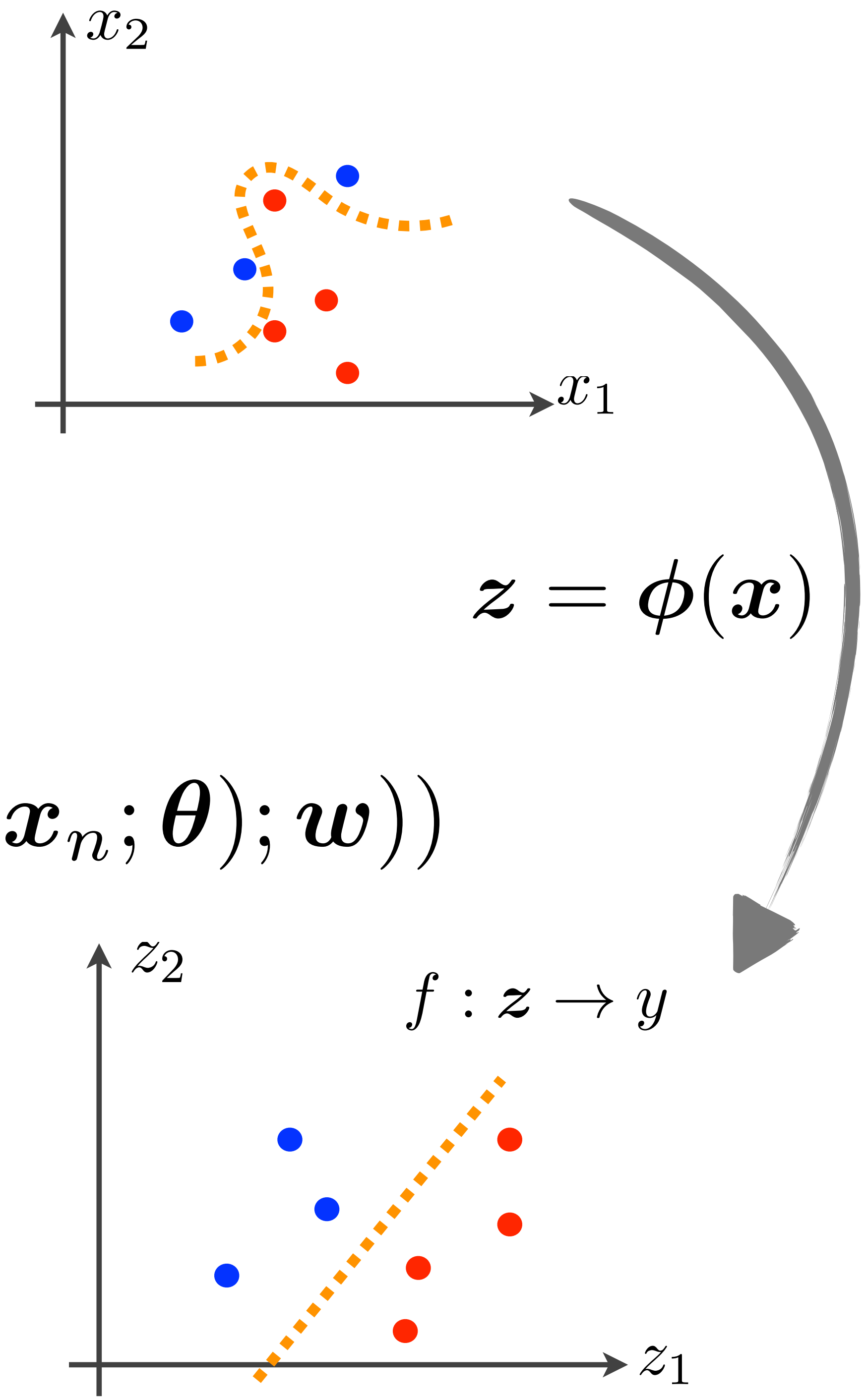
Representation learning (abstractly)

Training data

$$\{(\boldsymbol{x}_n, y_n), n = 1, 2, \cdots, \mathsf{N}\}$$

Jointly empirical risk minimization

$$\boldsymbol{\theta}^*, \boldsymbol{w}^* = \arg\min \frac{1}{\mathsf{N}} \sum_n \ell(\boldsymbol{x}_n, y_n, f(\boldsymbol{\phi}(\boldsymbol{x}_n; \boldsymbol{\theta}); \boldsymbol{w}))$$



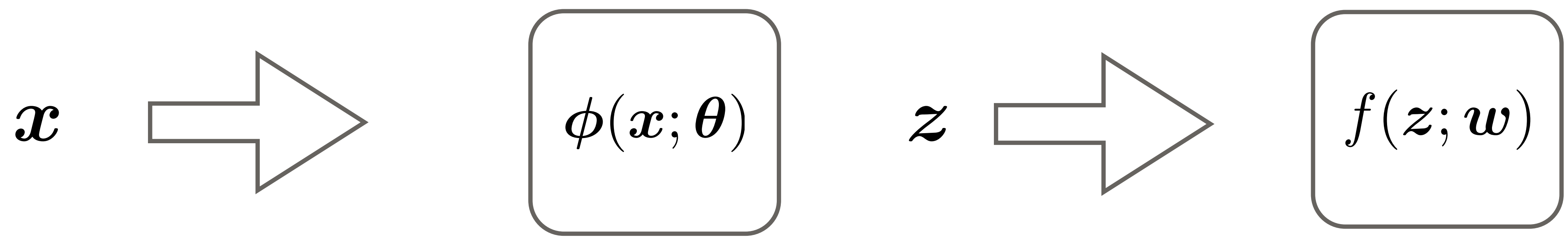
Learning representation

Motivation

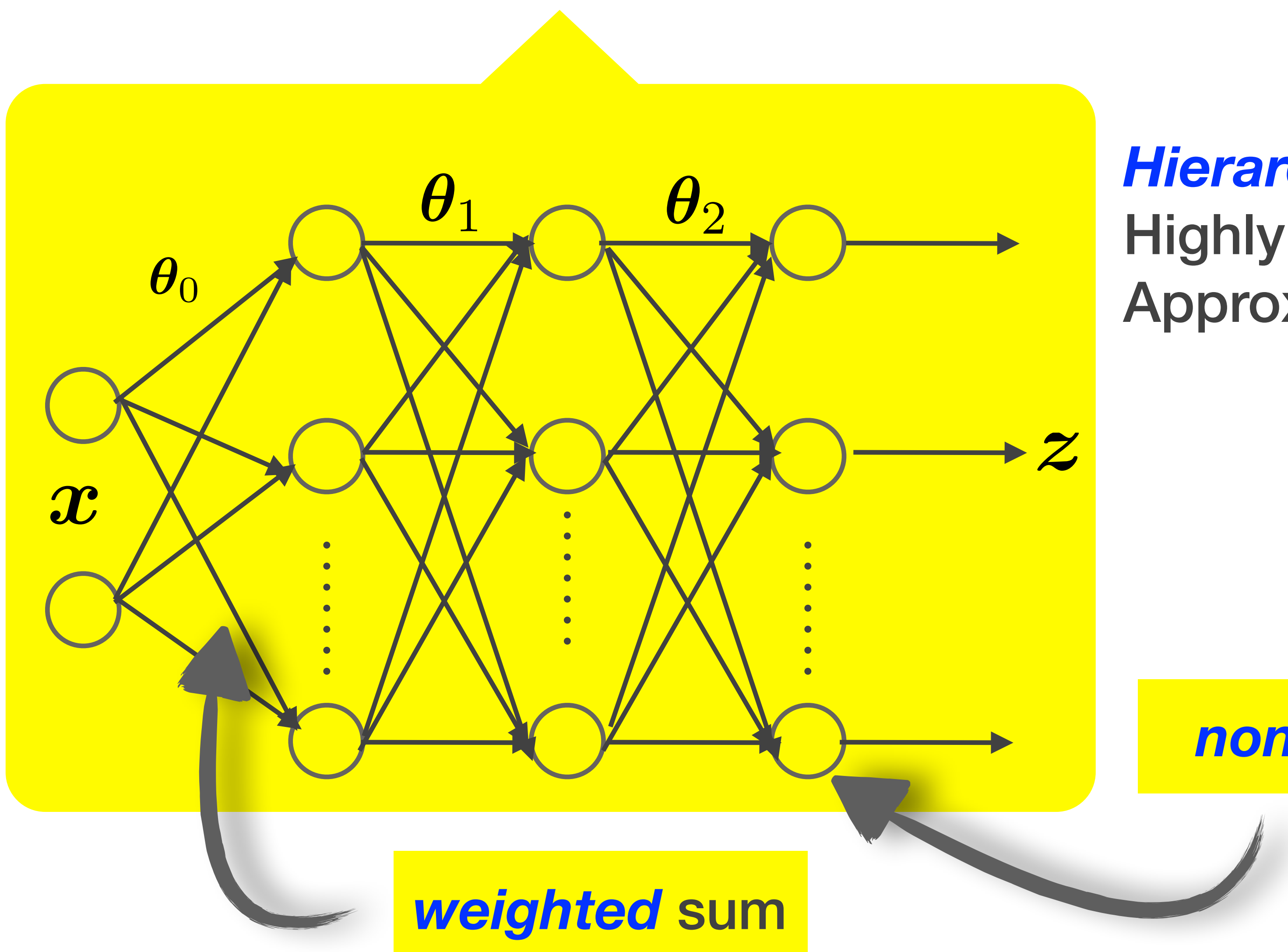
Deep neural networks

Kernel methods

Deep neural networks for learning representation



Hierarchical (deep) transformation
Highly ***nonlinear mapping***
Approximate any ***smooth*** function



nonlinear processing units

The success of deep learning/DNN

Automatic speech recognition

The community has heavily used DNN since 2011

Computer vision

Tasks: object recognition, face detection, street number recognition

Attain the best result on ImageNet (a challenging benchmark)

Langauge processing

Tasks: language model, generating captions for images, machine translations

Board games

AlphaGo

Many more and more

….

What is ***not so ideal*** about DNN?

Practical concerns

Intensive development cost due to many hidden knobs

Design and architecture: *how many layers? how many hidden units in each layer? what are the types of hidden units?*

Algorithm: *step size, momentum, step size decay rate, regularization coefficients, etc*

Resources demanding

Data: *what if we do not have a lot of data?*

Computing: *what of we do not have a lot of GPUs and CPUs?*

Theoretical concerns

Rely very much on ***intuition and heuristics and trial-and-error***

Gap between rich empirical success and scarce theoretical underpinning

Then, any alternatives?

Learning representation

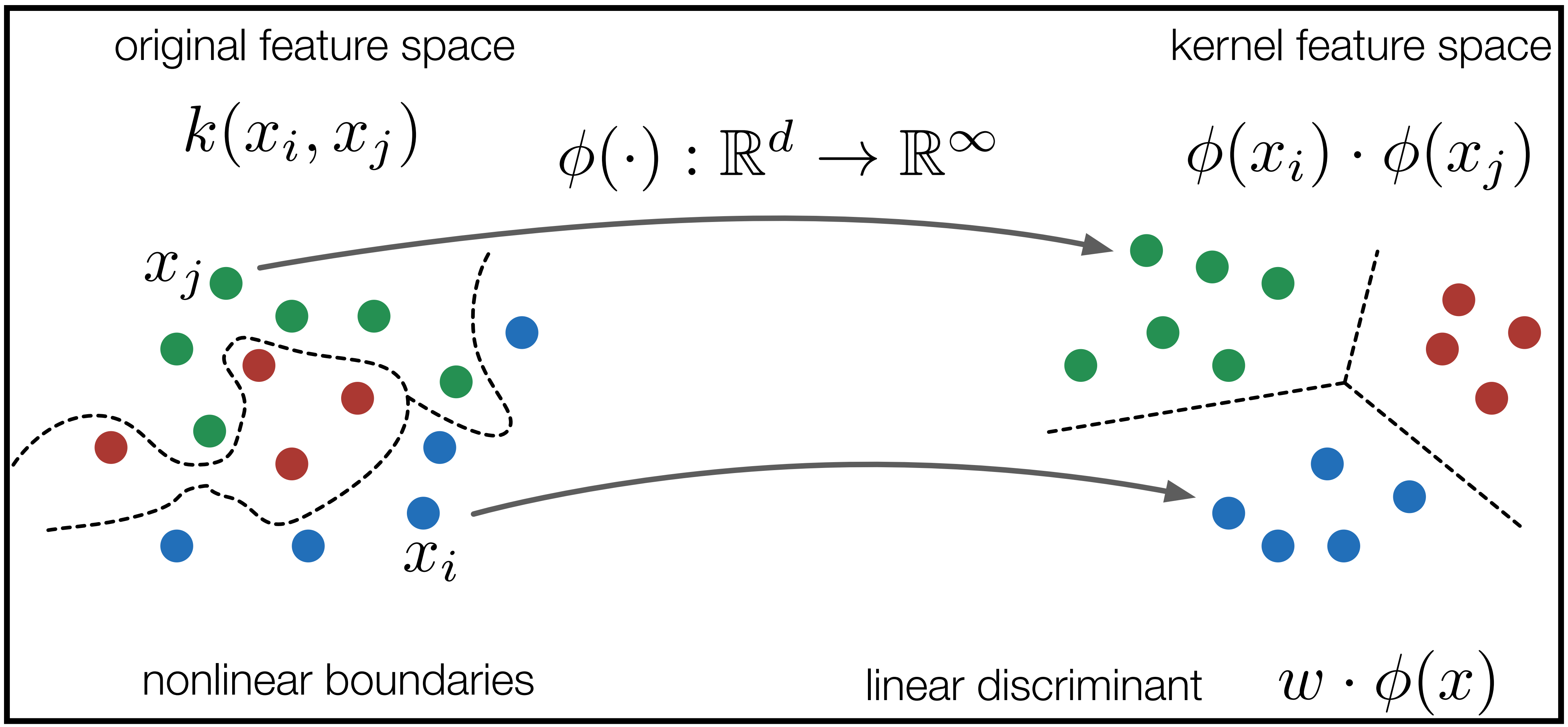
Motivation

Deep neural networks

Kernel methods

Kernel methods

Insights: classifiers use ***inner products*** between features



Kernel trick

Definition

A Mercer (or positive definite) kernel function is a bivariate function

$$k(\boldsymbol{x}_i, \boldsymbol{x}_j) = \boldsymbol{\phi}(\boldsymbol{x}_i)^{\mathrm{T}} \boldsymbol{\phi}(\boldsymbol{x}_j)$$

Implications

Kernel function ***implicitly defines*** a feature mapping, ie, a ***new*** representation of data

$$\boldsymbol{\phi} : \boldsymbol{x} \rightarrow k(\boldsymbol{x}, \cdot) \in \mathcal{H}$$

Selecting the right kernel will give us the ***right*** representation

Example

Gaussian kernel function

$$k_1(\boldsymbol{x}_i, \boldsymbol{x}_j) = e^{-\|\boldsymbol{x}_i - \boldsymbol{x}_j\|_2^2/\sigma^2}$$

Mapping

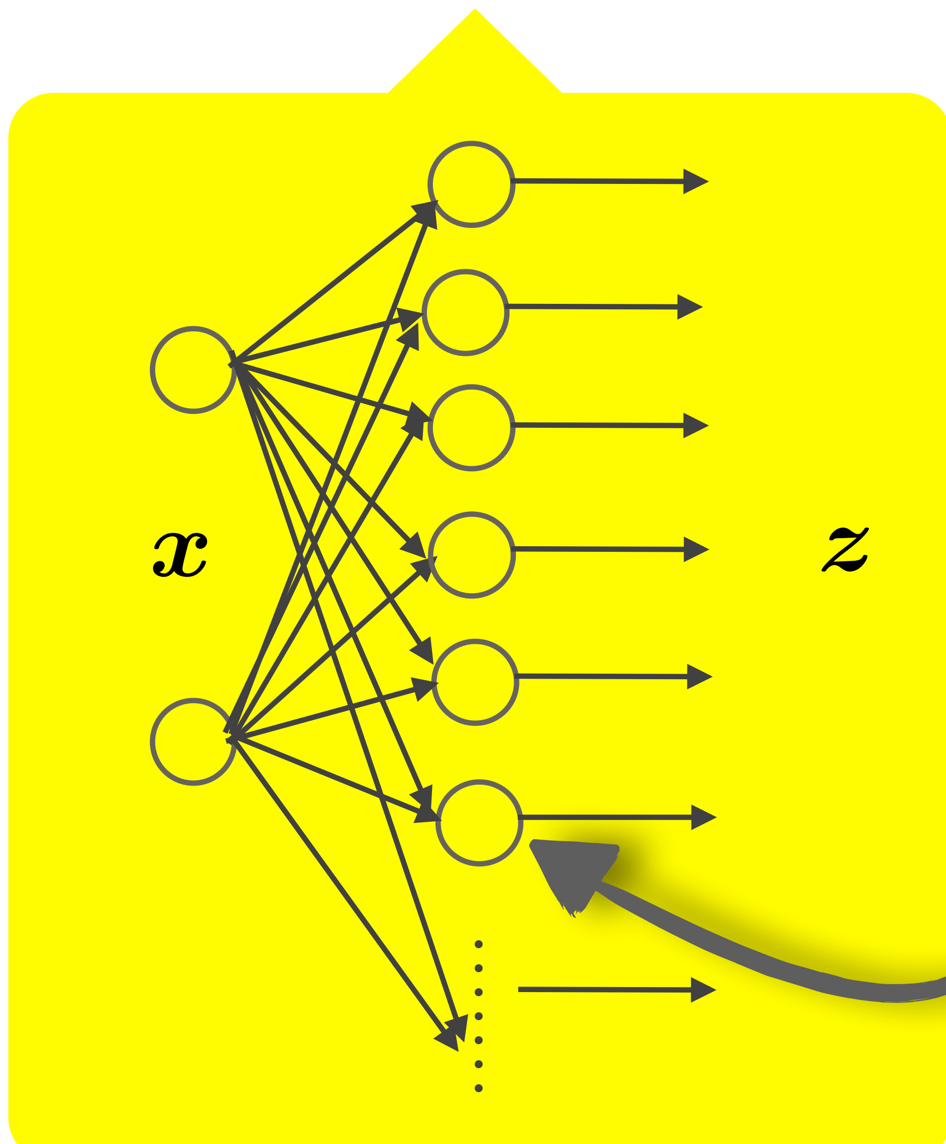
$$\boldsymbol{\phi} : \boldsymbol{x} \to (\sqrt{\lambda_j}\psi_j(\boldsymbol{x}))_{j=1,2,3,\cdots,\infty}$$

with eigenfunction and eigenvalues from

$$\int_{\mathcal{X}} e^{-\|\boldsymbol{x}-\boldsymbol{x'}\|_2^2/\sigma^2} \psi_j(\boldsymbol{x}')d\mu(\boldsymbol{x}') = \lambda_j \psi_j(\boldsymbol{x})$$

Kernel methods are ***shallow***

$\boldsymbol{x} \Rightarrow \boxed{\phi(\boldsymbol{x};\boldsymbol{\theta})} \quad \boldsymbol{z} \Rightarrow \boxed{f(\boldsymbol{z};\boldsymbol{w})}$



Shallow transformation
Highly ***nonlinear mapping***
Infinite-dimension representation
Approximate any ***smooth*** function

nonlinear processing units

What is ***nice*** about kernel methods?

Extensively studied and well-understood theoretical properties

Ex: regularization, generalization error bound

Strong computational advantages (at least in theory)

Most time, convex optimization

Not many hidden tuning knobs

Kernel methods are clean

Transparent

It is relatively easier to explain a kernel model

What is not ***so great*** about them?

Computational complexity in practice

Kernel trick is a double-bladed sword

Need to evaluate kernel functions: ***second-order*** *in the number of training samples*

Difficult to handle large-scale datasets: limited often at millions of samples

How to choose the right kernel?

Infinitely many kernel functions

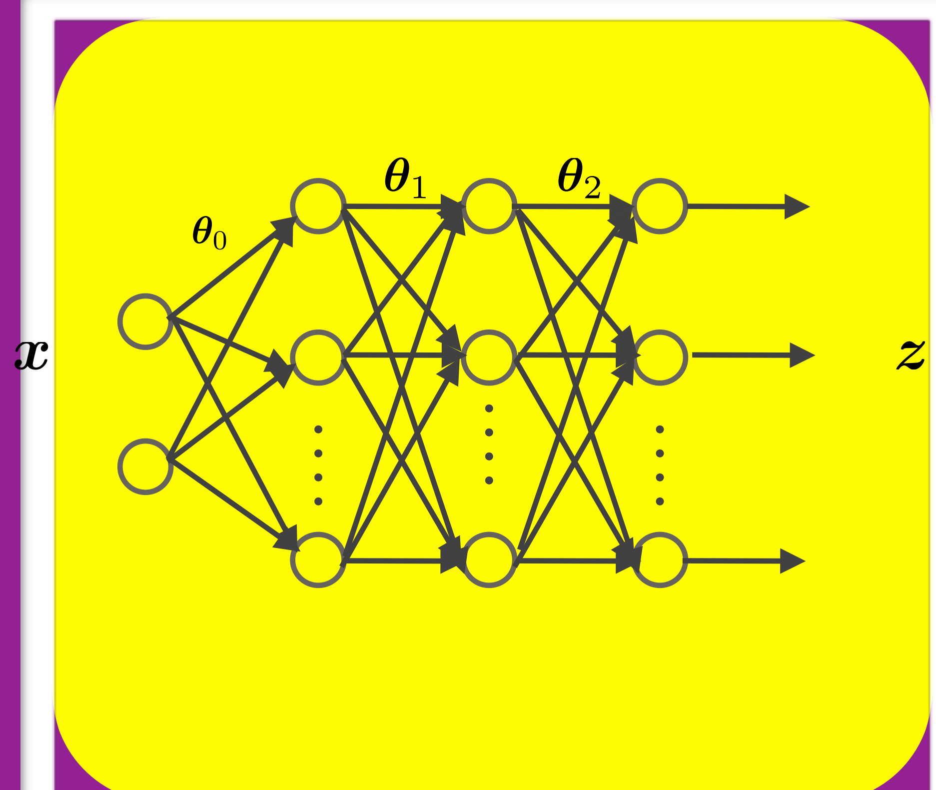
Learning ***optimal*** kernel function from data is an open problem

[NB: a large body of work on overcoming this challenge. Eg. Bouttou, Chapelle, DeCoste, and Weston' 07 (eds). Das et al, '14, Huang et al, 14, Le, Sarlos and Smola, '13, Yen et al' 14, Hsien, Si and Dhillon, '13]

No method is ***perfect***

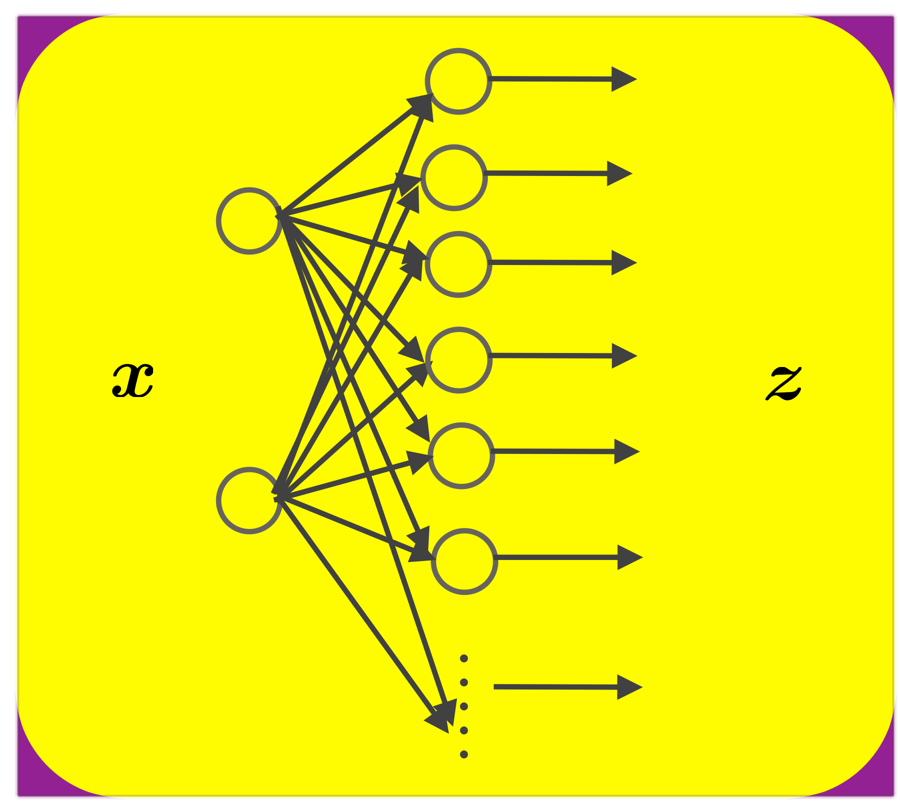
Deep neural networks

deep
scale to big data
strong empirical success



Kernel methods

shallow
does not scale
strong theoretical results



Then, why deep learning is so ***hot***?

Myth #1: being deep is theoretically necessary*

There exists functions that are implementable with d-layer deep learning, but requires $O(e^d)$ nodes for shallow learning.

But, ***do real tasks we care really need those types of functions?***

Myth #2: kernel methods are empirically intractable

Implementing kernel methods exactly does require quadratic-ordered complexity.

But, ***can real tasks we care be solved approximately?***

[: Montufar et a' 14, Montufar and Morton '14, Telgarsky '15]*

Shall we try to demystify the myths?

Scientific merits

Reveal the ***true differences*** between two paradigms after ***teasing the power of data out***: eg. *are the successes largely attributed to the volume of data?*

Understand the ***nature*** of different tasks: *eg. are certain tasks inherently far more difficult than others, thus entailing deep learning?*

How: head-on empirical comparison

On **large-scale datasets** from real-world applications

With **task-specific** evaluation metrics

Equally enthusiastic in **tuning** both paradigms

[NB: Huang et al, ICASSP 2013, 2014]

Let us fill the void

Learning representation

Optimizing billion-parameter models

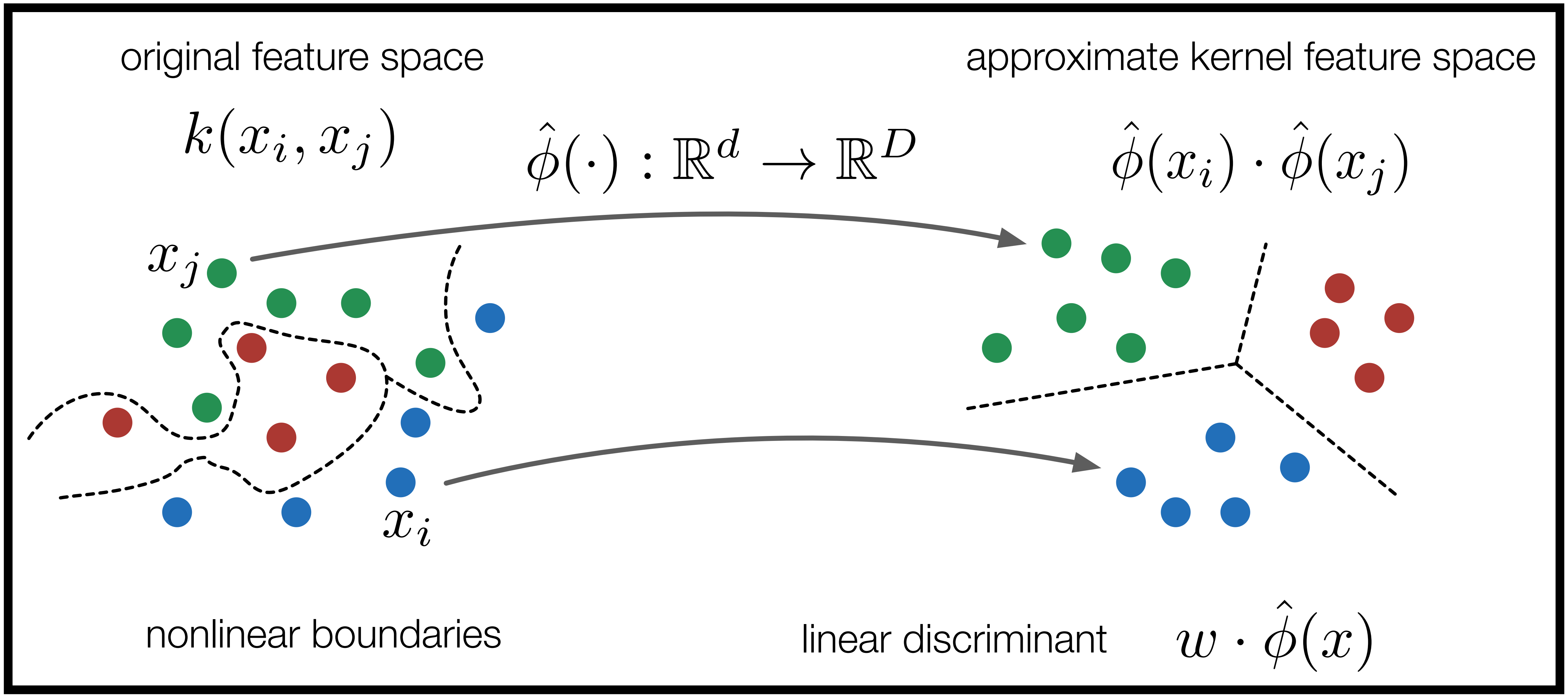
Scaling up Kernel methods

Kernel Garbage Compactor

How to scale kernel methods?

Key ideas: approximate kernel features

$$\boldsymbol{\phi}(\boldsymbol{x}_i)^{\mathrm{T}}\boldsymbol{\phi}(\boldsymbol{x}_j) \approx \hat{\boldsymbol{\phi}}(\boldsymbol{x}_i)^{\mathrm{T}}\hat{\boldsymbol{\phi}}(\boldsymbol{x}_j)$$



Monte Carlo approximation of kernel

[Rahimi & Recht, NIPS 2007, 2009]

Bochner's Theorem

$k(\boldsymbol{x}, \boldsymbol{z}) = k(\boldsymbol{x} - \boldsymbol{z})$ is a positive definite if and only if $k(\boldsymbol{\delta})$ is the Fourier transform of a non-negative measure. Specifically, the kernel function can be expanded with harmonic basis, namely

$$k(\boldsymbol{x} - \boldsymbol{z}) = \int_{R^d} p(\boldsymbol{\omega}) e^{j\boldsymbol{\omega}^{\mathrm{T}}(\boldsymbol{x}-\boldsymbol{z})d\boldsymbol{\omega}} = \int_{R^d} p(\boldsymbol{\omega}) e^{j\boldsymbol{\omega}^{\mathrm{T}}\boldsymbol{x}} e^{-j\boldsymbol{\omega}^{\mathrm{T}}\boldsymbol{z}}$$

$$= \mathbb{E}_{\boldsymbol{\omega}} e^{j\boldsymbol{\omega}^{\mathrm{T}}\boldsymbol{x}} e^{-j\boldsymbol{\omega}^{\mathrm{T}}\boldsymbol{z}}$$

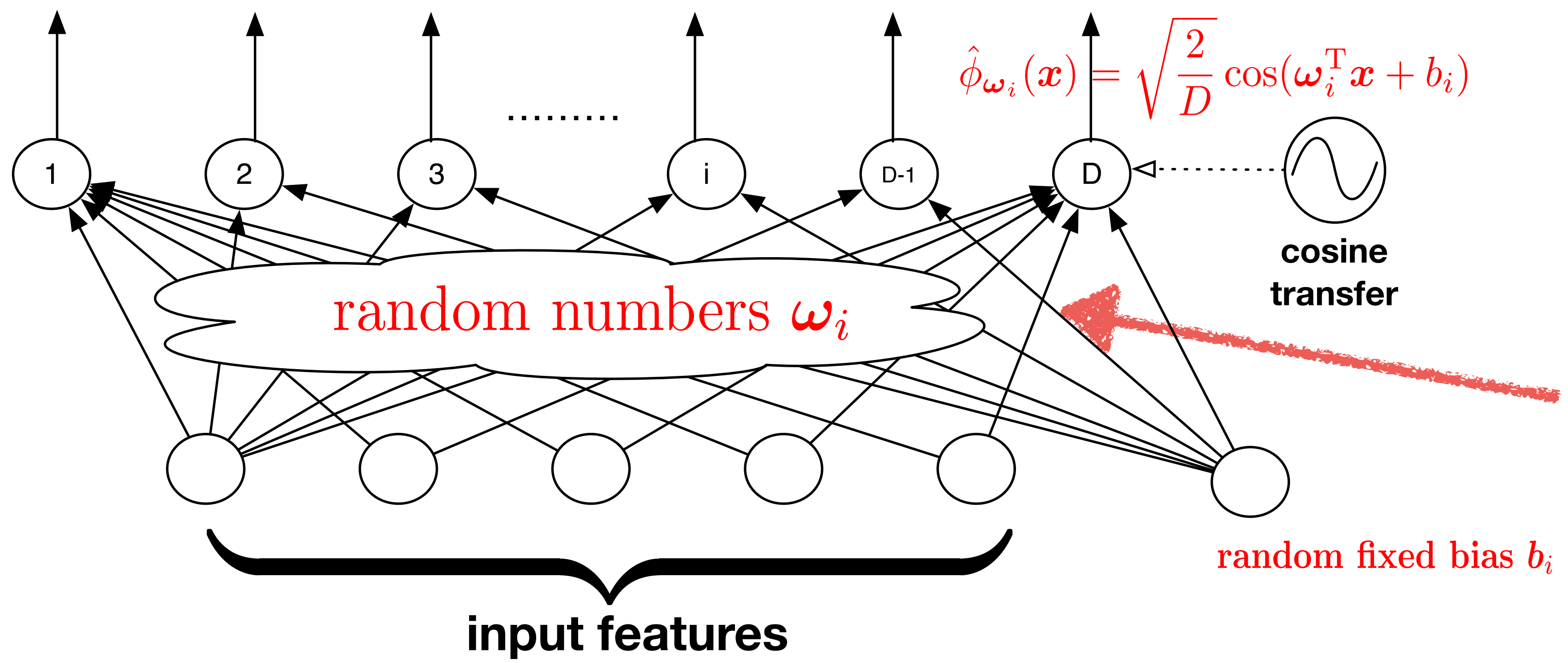
Implication

We can ***sample*** from the (probability) measure

Use the random samples to generate the ***approximate*** features

$$k(\boldsymbol{x}, \boldsymbol{z}) \approx \frac{1}{D} \sum_{i=1, \boldsymbol{\omega}_i \sim p(\boldsymbol{\omega})}^{D} e^{j\boldsymbol{\omega}_i^{\mathrm{T}}\boldsymbol{x}} e^{-j\boldsymbol{\omega}_i^{\mathrm{T}}\boldsymbol{z}} = \frac{1}{D} \sum_{i=1, \boldsymbol{\omega}_i \sim p(\boldsymbol{\omega})}^{D} \hat{\phi}_{\boldsymbol{\omega}_i}(\boldsymbol{x}) \hat{\phi}_{\boldsymbol{\omega}_i}(\boldsymbol{z})$$

From kernel to random and shallow features



***Ex:* Gaussian distributed for Gaussian kernels**

For $i = 1, 2, \cdots$ to D

- Draw $\boldsymbol{\omega}_i$ from the distribution $p(\boldsymbol{\omega})$
- Construct a random feature

$$\phi_{\boldsymbol{\omega}_i} = \sqrt{2} \cos(\boldsymbol{\omega}_i^{\mathrm{T}} \boldsymbol{x} + b_i)$$

where b_i is a random number, uniformly sampled from $[0,\ 2\pi]$

Unlike DNN, those features are not adapted to data

Make the random feature vector

$$\hat{\phi}(\boldsymbol{x}) = \frac{1}{\sqrt{D}} [\phi_{\boldsymbol{\omega}_1}\ \phi_{\boldsymbol{\omega}_2}\ \cdots\ \phi_{\boldsymbol{\omega}_D}]$$

How to use those randomly generated features?

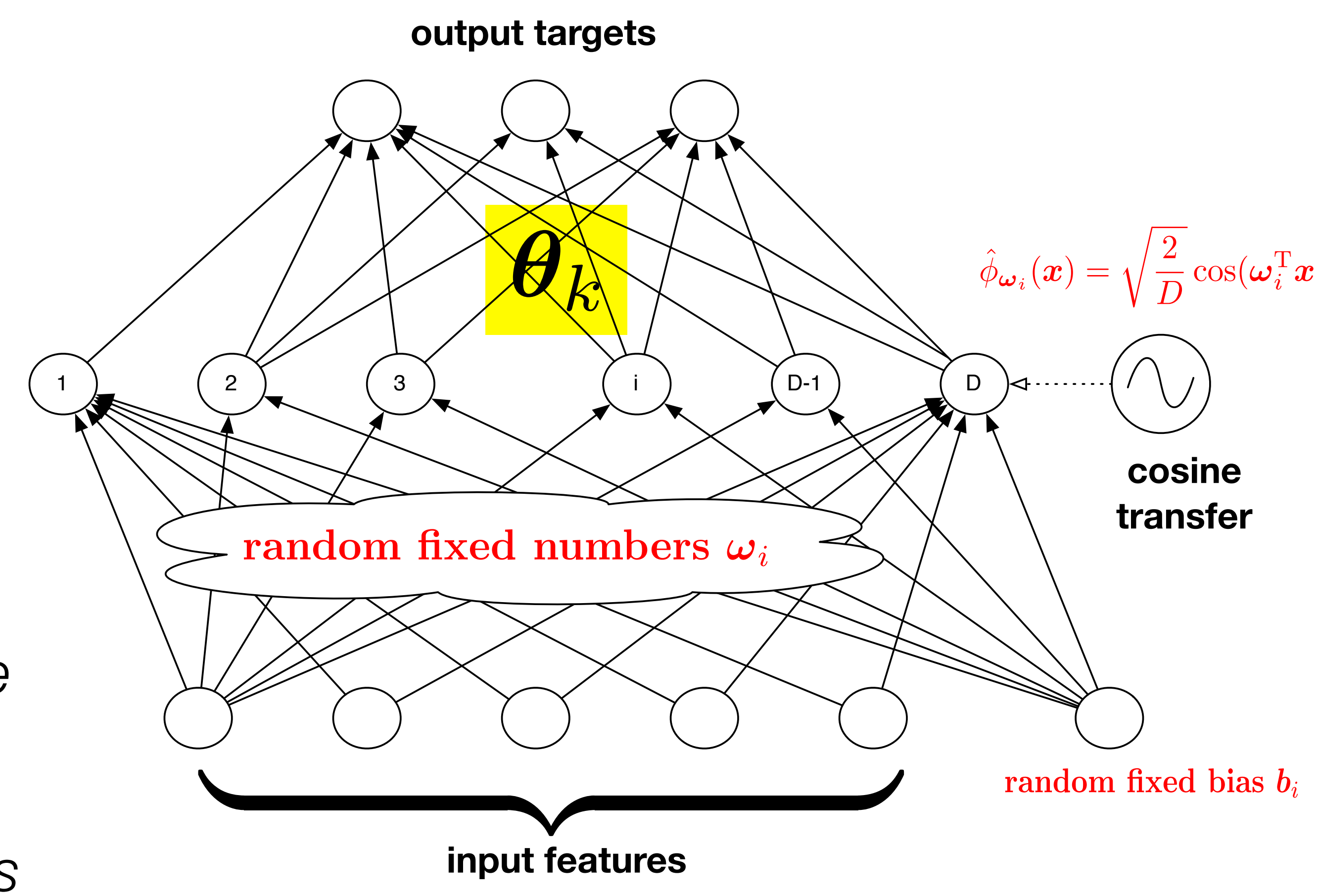
[Rahimi & Recht, NIPS 2007, 2009]

Random kitchen sink

Build linear classifiers on top of those features

Ex: multinomial logistic regression

$$P(y = k|\boldsymbol{x}) \propto \exp(\boldsymbol{\theta}_k^{\mathrm{T}} \boldsymbol{\phi}(\boldsymbol{x}))$$



Properties

Computational complexity

No longer *depends quadratically on the number of training samples.*

The number of random features provides ***speed and accuracy tradeoff****.*

Optimization

Convex optimization

Flashback: connection between shallow and deep

[Neal, 1994; Williams, 1996; Cho and Saul, 2009]

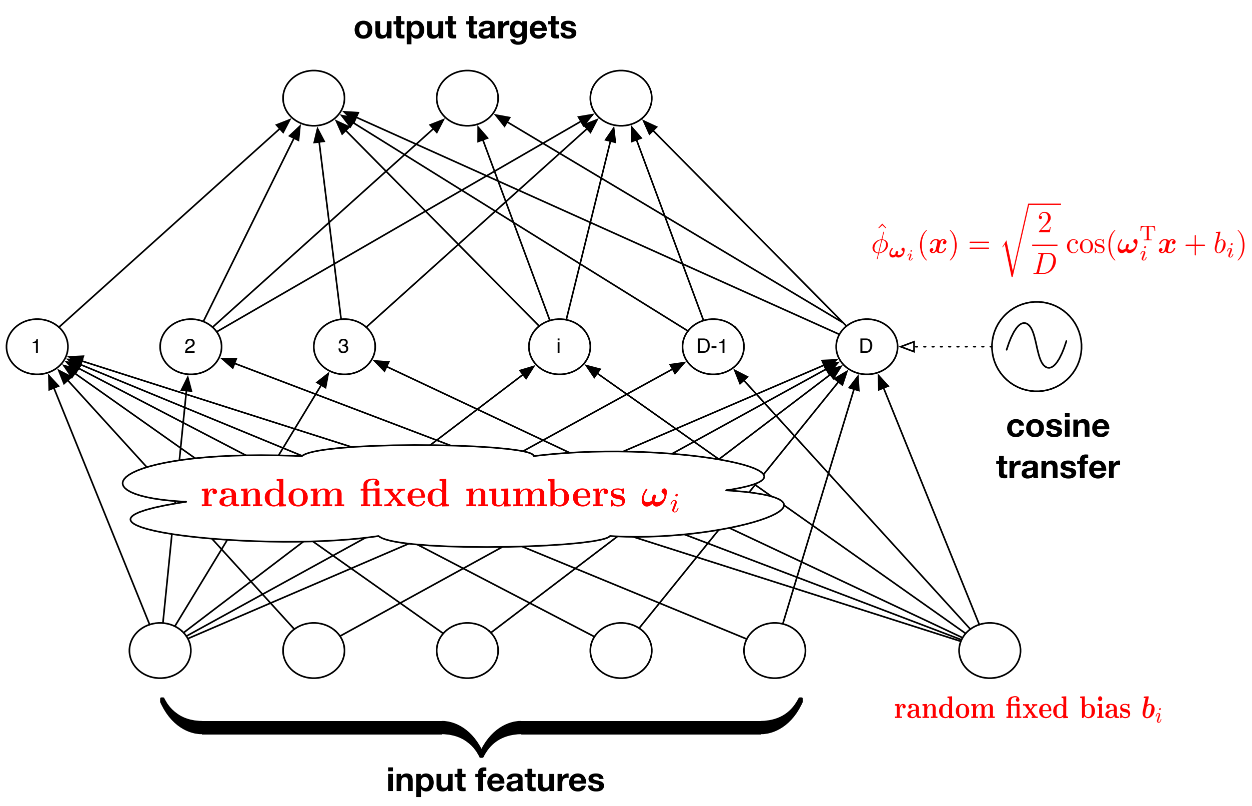
Kernel machines can be seen as a neural network

Shallow and infinitely-wide

Simpler to construct and learn

random projection in bottom

optimize only in the top



Interestingly, kernel machines can be very big!

Number of random features

~200,000

Number of classes

~5000

Total number of parameters

1 billion learnable

72 million random numbers

In many of our experiments, the kernel machines have *significantly more parameters* than typical DNN systems.

Learning representation

Optimizing billion-parameter models

Scaling up Kernel methods

Kernel Garbage Compactor

Kernel Garbage Compactor

Main idea

Inject a ***linear*** layer between random features and outputs

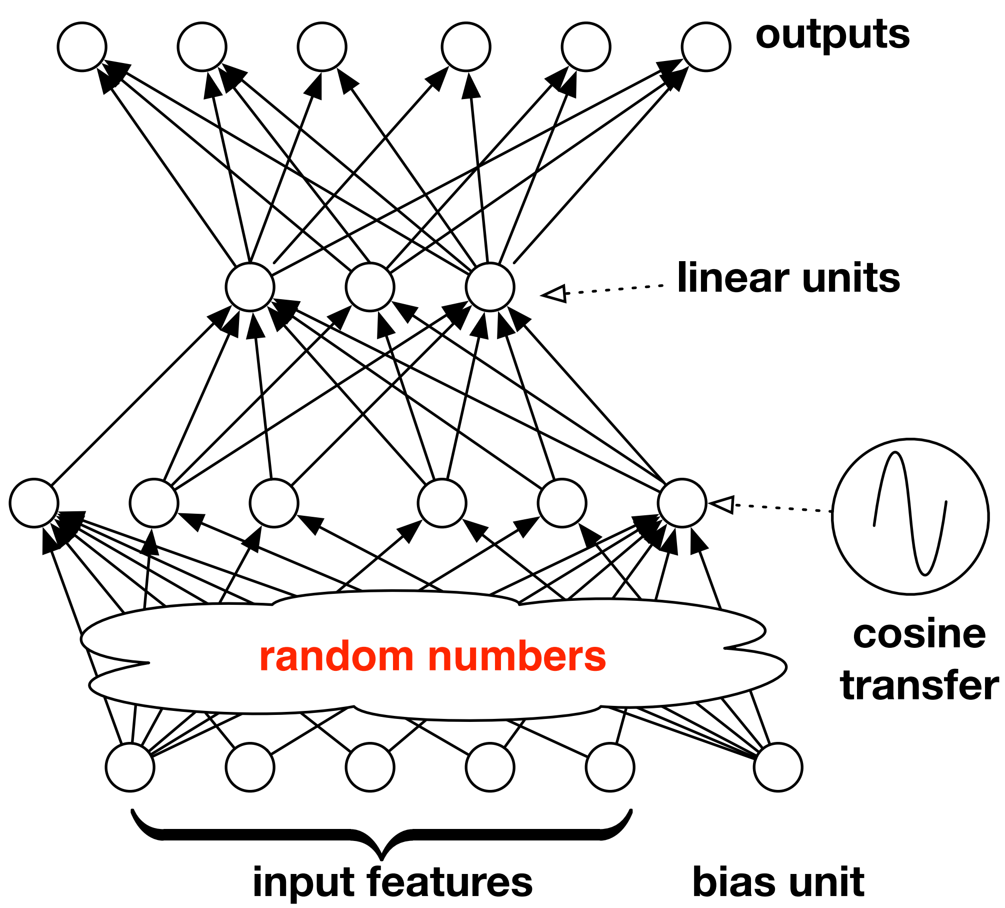
Demand ***bottleneck***: fewer number of linear units than random features

Properties

Compact random features: not all random features are equally useful

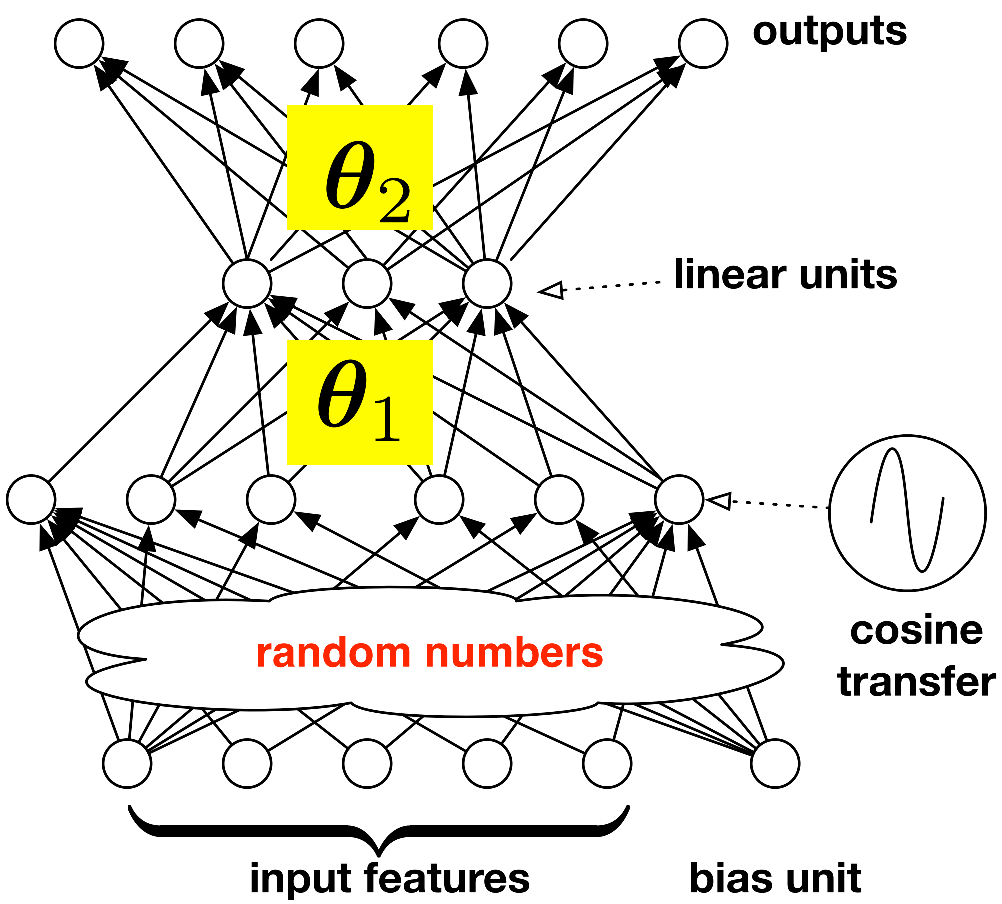
Prevent ***overfitting***: reduce the expressiveness of the model

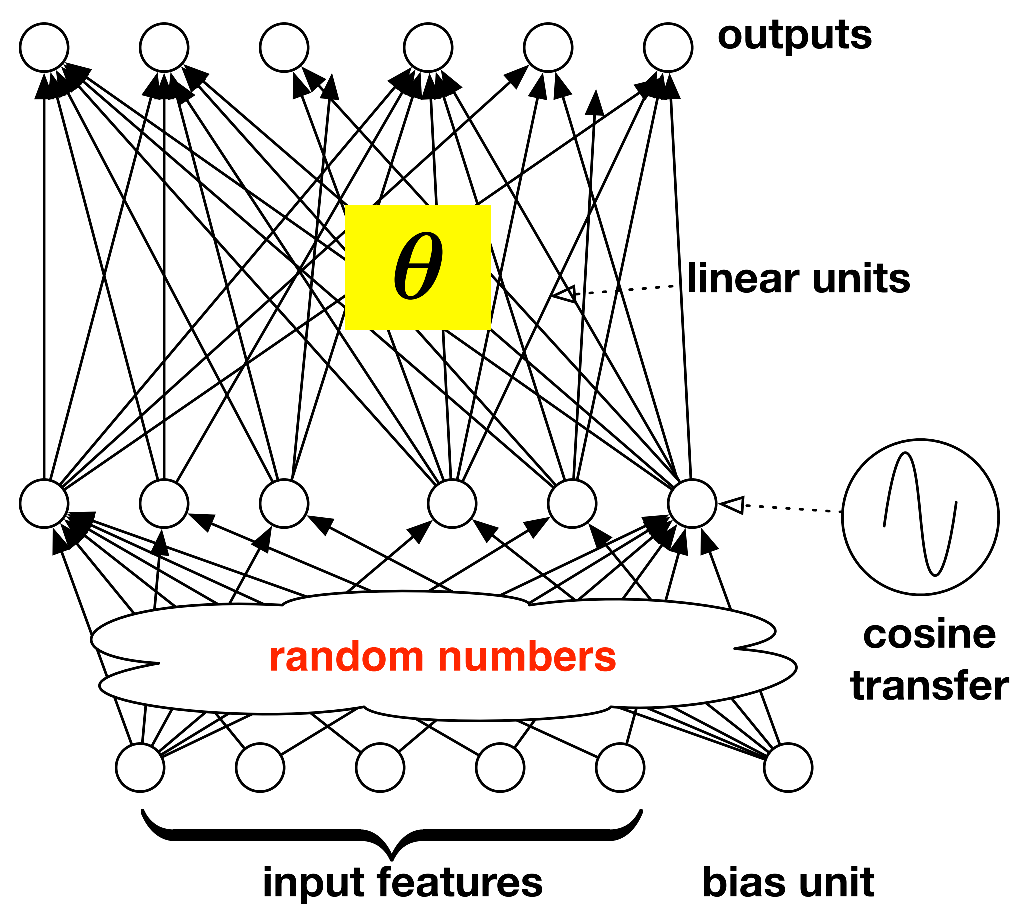
Encourage ***multi-tasking***: reuse outputs of linear units



[cf. Yen, Lin, Lin, Ravikumar, and Dhillon, '14]

Mathematically, low-rank regularization





$$\min \ \ell(\mathcal{D}; \boldsymbol{\theta} = \boldsymbol{\theta}_1 \boldsymbol{\theta}_2)$$

$$\min \ \ell(\mathcal{D}; \boldsymbol{\theta}) \text{ s.t } \operatorname{rank}(\boldsymbol{\theta}) \leq r$$

$$\min \ \ell(\mathcal{D}; \boldsymbol{\theta}) + \lambda \|\boldsymbol{\theta}\|_*$$

[cf. Yen, Lin, Lin, Ravikumar, and Dhillon, '14]

Automatic speech recognition

Massive experimentation

Setup

Results

Acoustic modeling for ASR

Tasks

Estimate the conditional probability of phone (state) labels at any given time t

$$P(y = k | \boldsymbol{x}_t)$$

Model is optimized for *lowest cross-entropy error (or perplexity)*, proxy to *classification accuracy*

Data

2 language packs from IARPA BABEL Program: Bengali & Cantonese

Challenging: *bad acoustic conditions, limited language resources*

Large-scale: *each with 1000 classes, 7-8 million training samples*

Broadcast News (50 hours): commonly used in ASR community

Large-scale: *5000 classes, 16 million training samples*

System details

Kernels

Gaussian, Laplacian kernels and their combinations

# of random features: up to 500,000 (model size: > 1 billion params)

hyperparameters: 4 (bandwidth, # features, gradient step size, bottleneck size)

Deep neural networks

Industry: provided by IBM Research Speech Group (using greedily layer-wise discriminative training), 4 hidden layers, very well tuned

Home-brew: our own training recipe (with unsupervised pre-training)

Evaluation criteria

Follow industry standard: word error rate

Assessed by IBM's proprietary ASR engine (including decoder)

Automatic speech recognition

Massive experimentation

Setup

Results

Sanity check: handwritten digit recognition

Dataset

Classification error (%)	Kernel (150K features)		DNN (4 hidden layers)	
Augmented training data	no	yes	no	yes
Validation	0.97	0.79	0.71	0.62
Test	1.09	**0.85**	**0.69**	0.77

Difference is statistically ***insignificant***
(McNemar test p-value $= 0.45$)

MNIST-6.7 (a variants of MNIST) with ***6.75 million training examples***

10 classes

Performance on real task of ASR

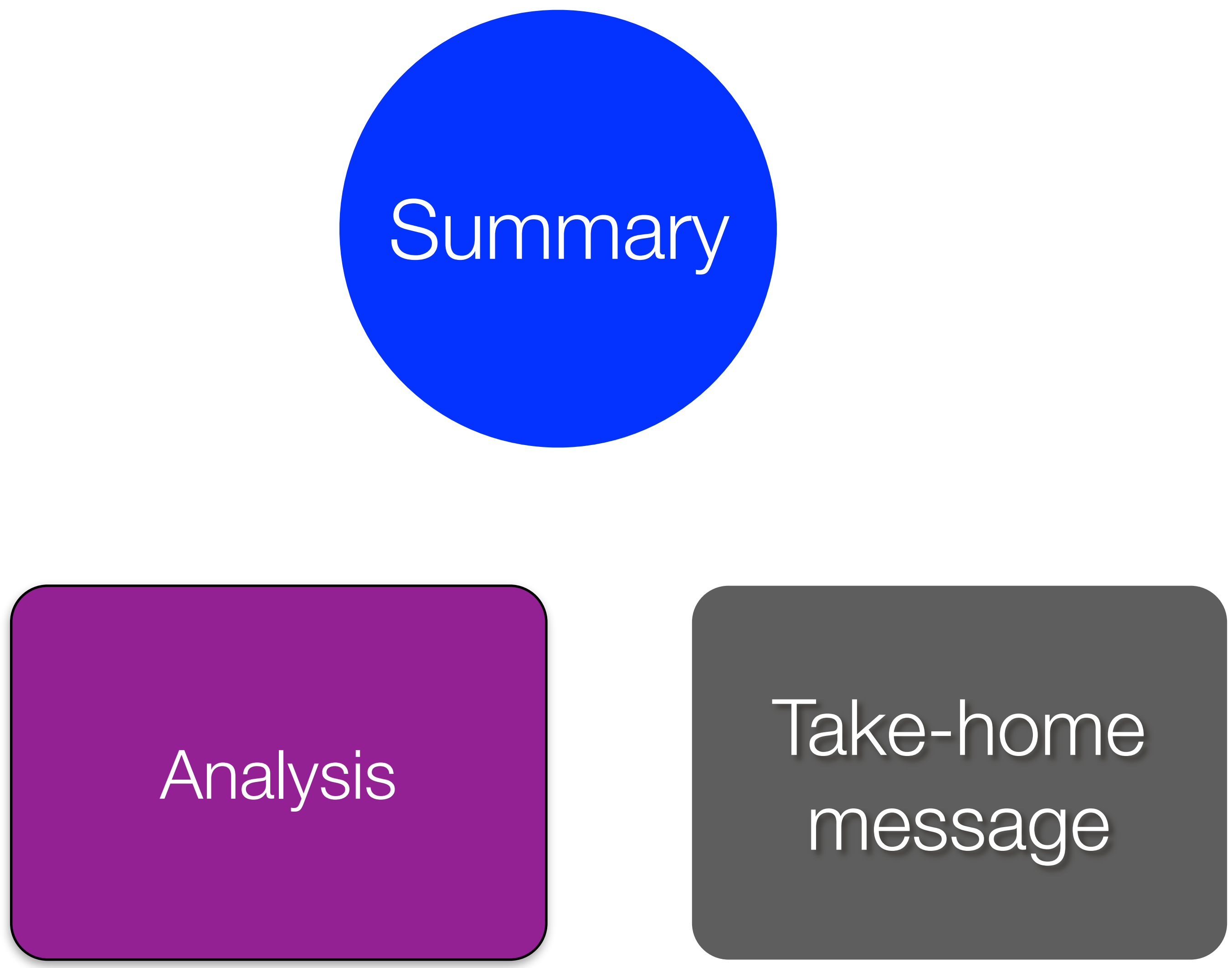
Word error rate (%)

	Bengali	Cantonese	Broadcast
IBM DNN	70.4	67.3	16.7
Our / Columbia) DNN (1)	**69.5**	66.3	16.6
Our DNN (2)	**-**	-	15.5
Kernel (200K)	70	**65.7**	16.7

Kernel and DNN are complementary

Word error rate (%)

	Best single	Combined
MNIST	0.69	**0.61**
Bengali	69.5	**69.1**
Cantonese	65.7	**64.9**
Broadcast	16.6	**-**

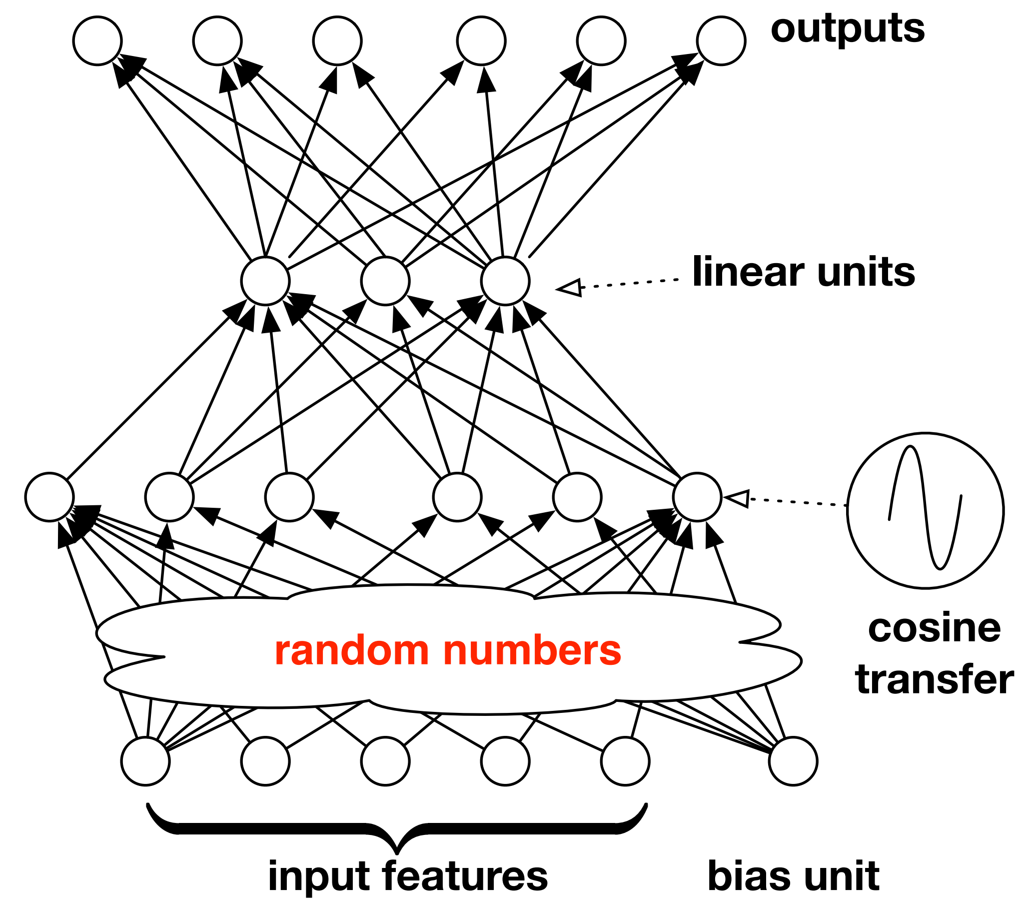

Summary
Analysis
Take-home message

Details of our kernel systems

Initial stage

Train a kernel garbage compactor

Take the output of the linear units as ***new representations*** of data



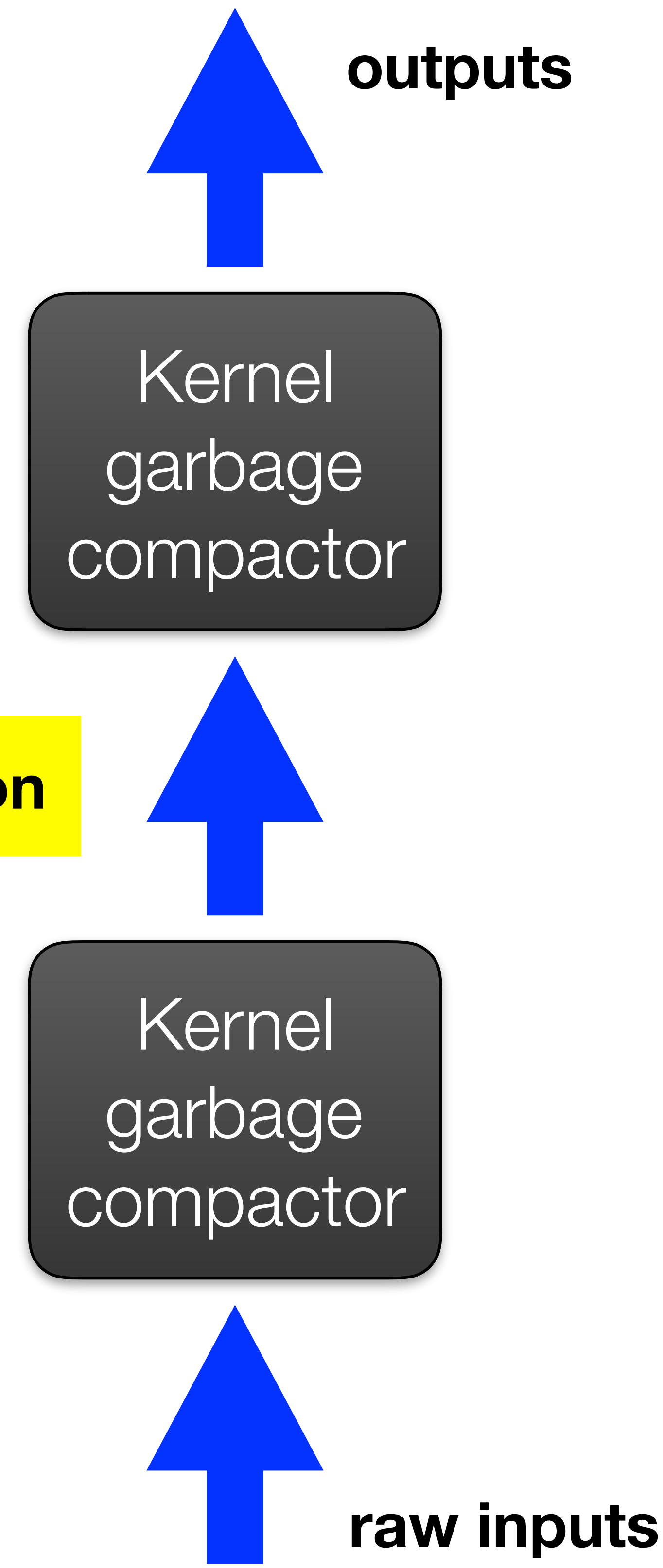
Details of our kernel systems

Final stage

Train ***another*** kernel garbage compactor

Keep stage-1's representation unchanged, ie, ***no back-propagation***

New **representation**

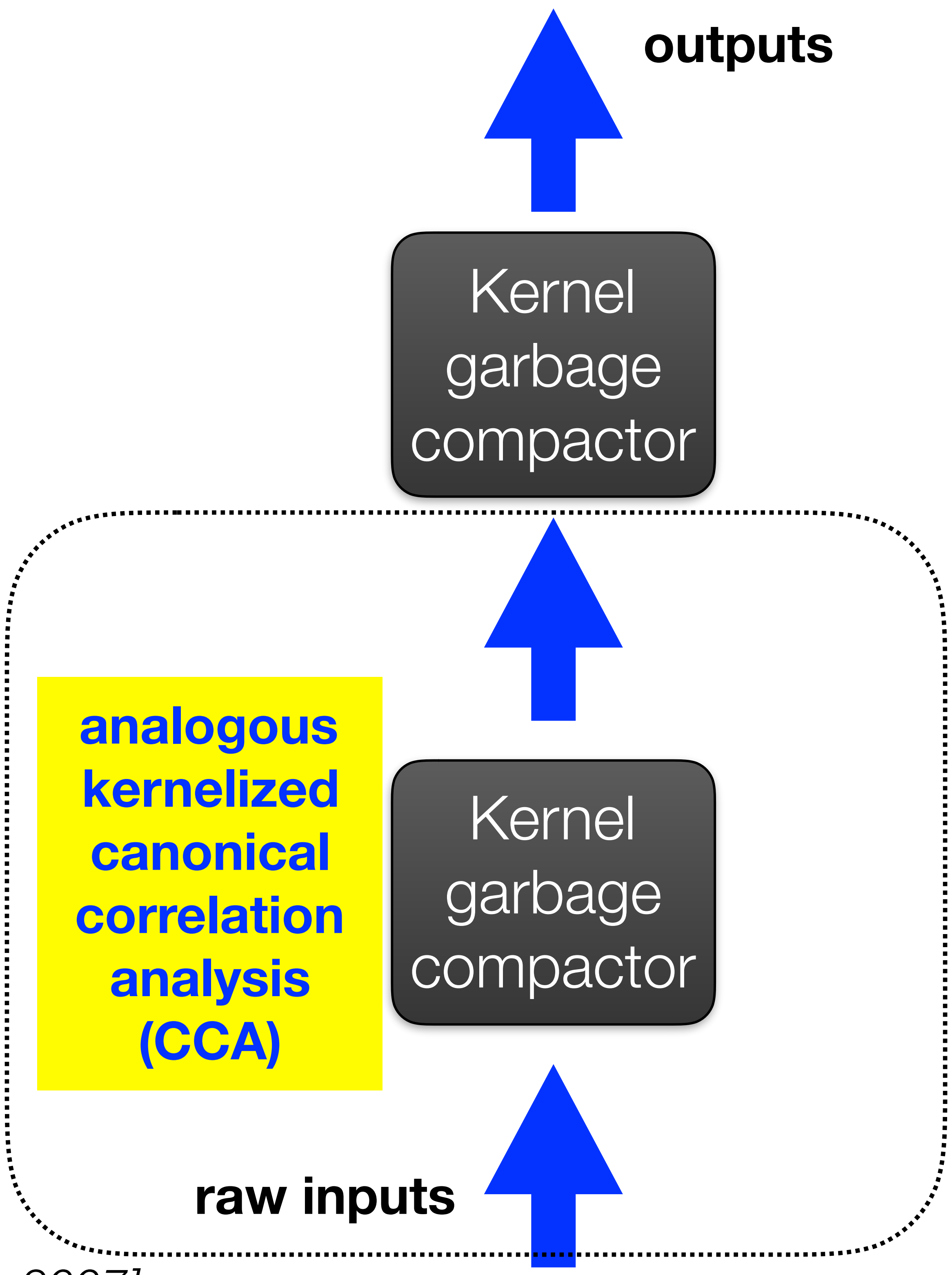


A somewhat shocking (re)discovery

Classic machine learning recipe works well

Feature extraction: PCA, CCA, Fisher discriminant analysis, kernel PCA, kernel CCA, manifold learning, etc

Model fitting: linear classification, kernel SVM, boosting, neural networks, etc



[Bach and Jordan, 2012. Fukumizu, Bach and Gretton, 2007]

In a similar spirit

[May et al, ICASSP 2016]

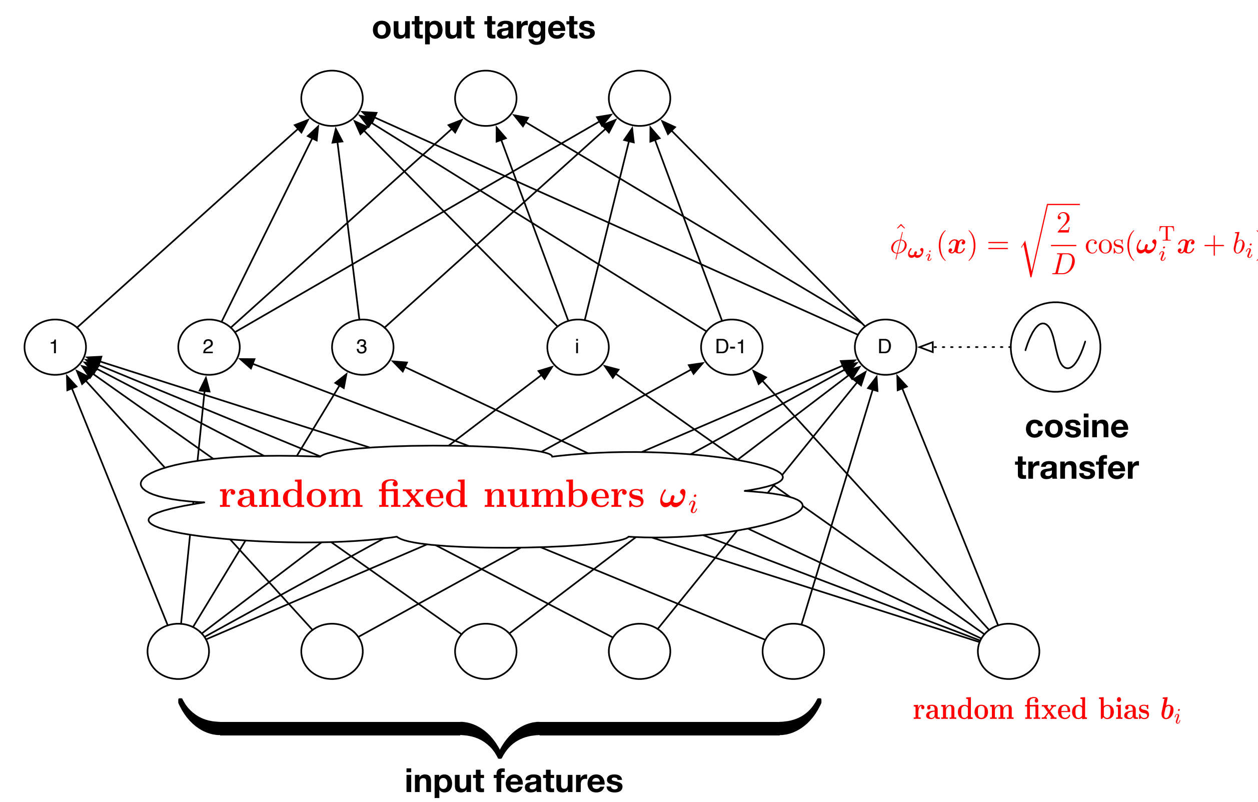
Random feature selection

Train a kernel machine

Delete "weak" features

Add more random features

Retrain the kernel machine



In the end, the idea of learning kernels!

Optimal kernel needs to be adapted to data

Combine base kernels (cf. Lancrkiet et al JMLR, 2014)

Use neural network to do back propagation (cf Salakhutdinov & Hinton' 08, Wilson et 2015)

Sequential selection (kernel CCA, random feature selection)

Kernel features via random projections are too dirty

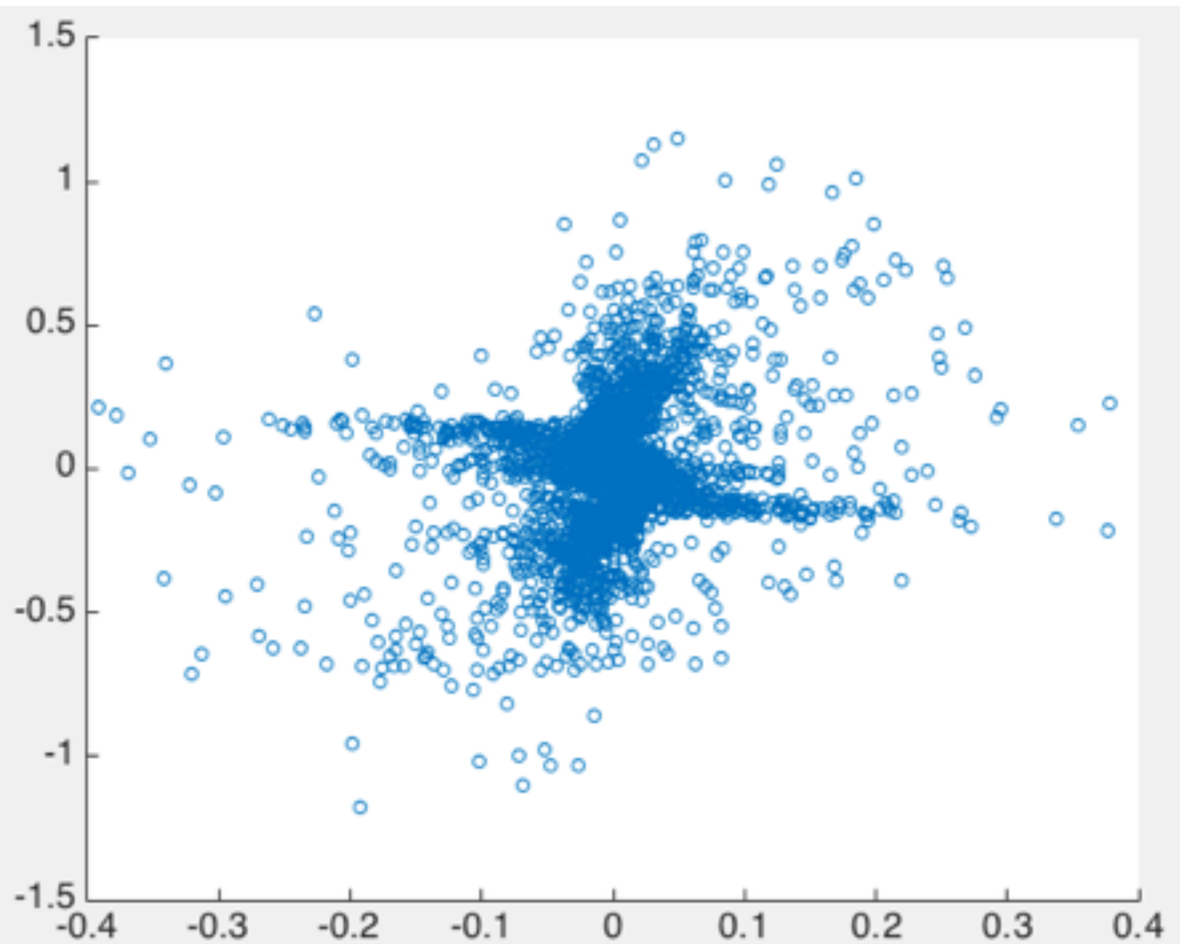
Minus: learning from wrong features

Plus: likely more robust

Detailed analysis using MNIST

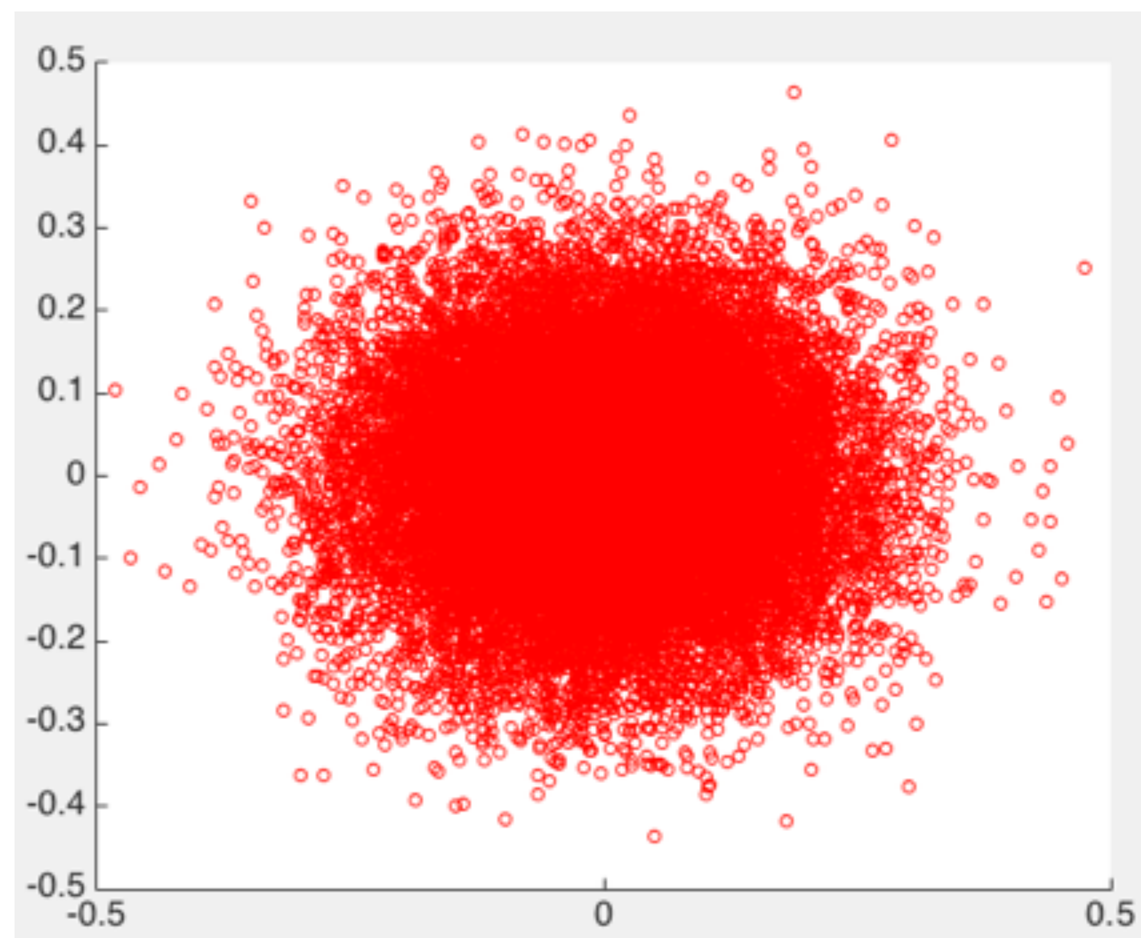
How random are they?

Neural network's has more interesting (non-Gaussian) structures!!

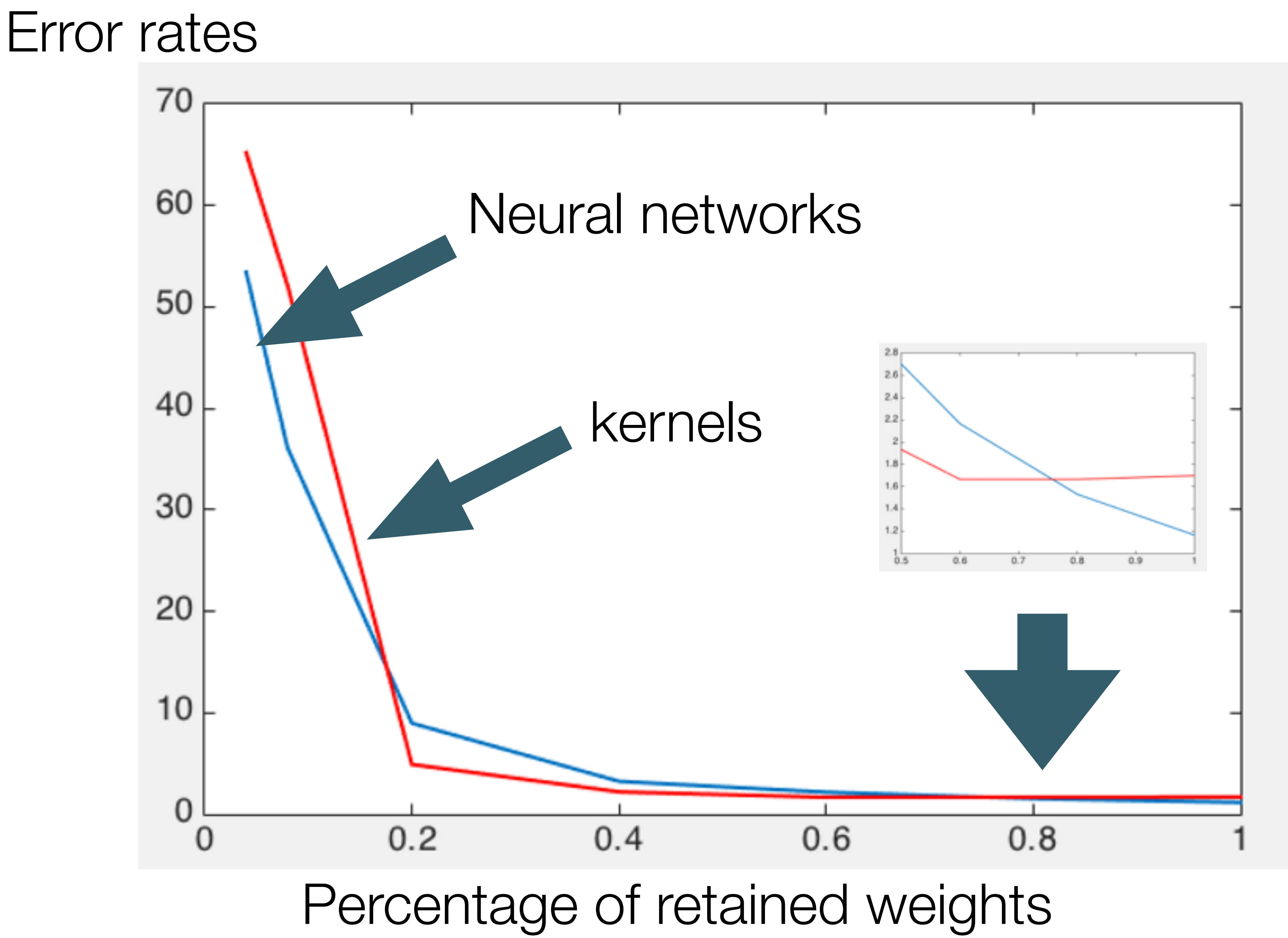


Bottom weights for NN w/ cosine activation

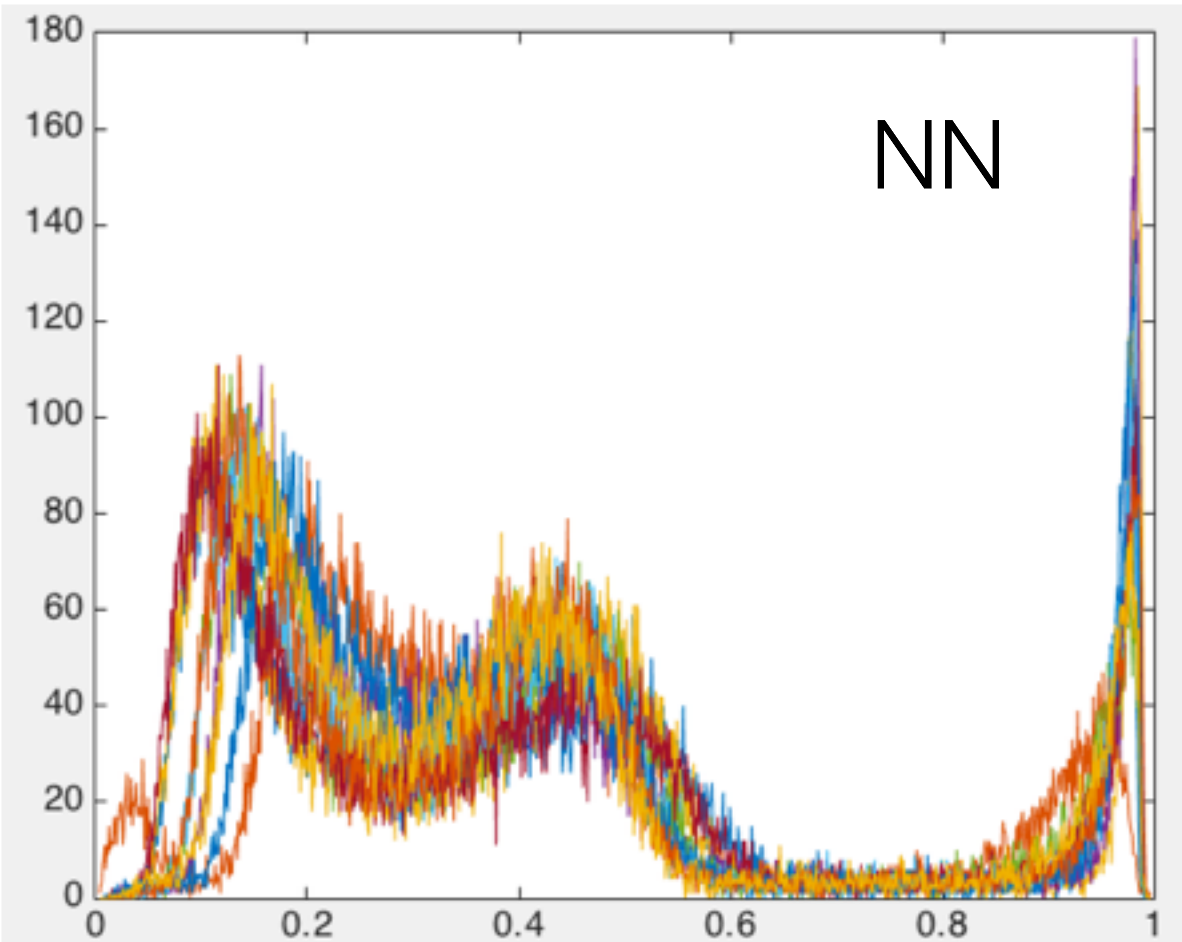
Bottom weights for RBF kernel



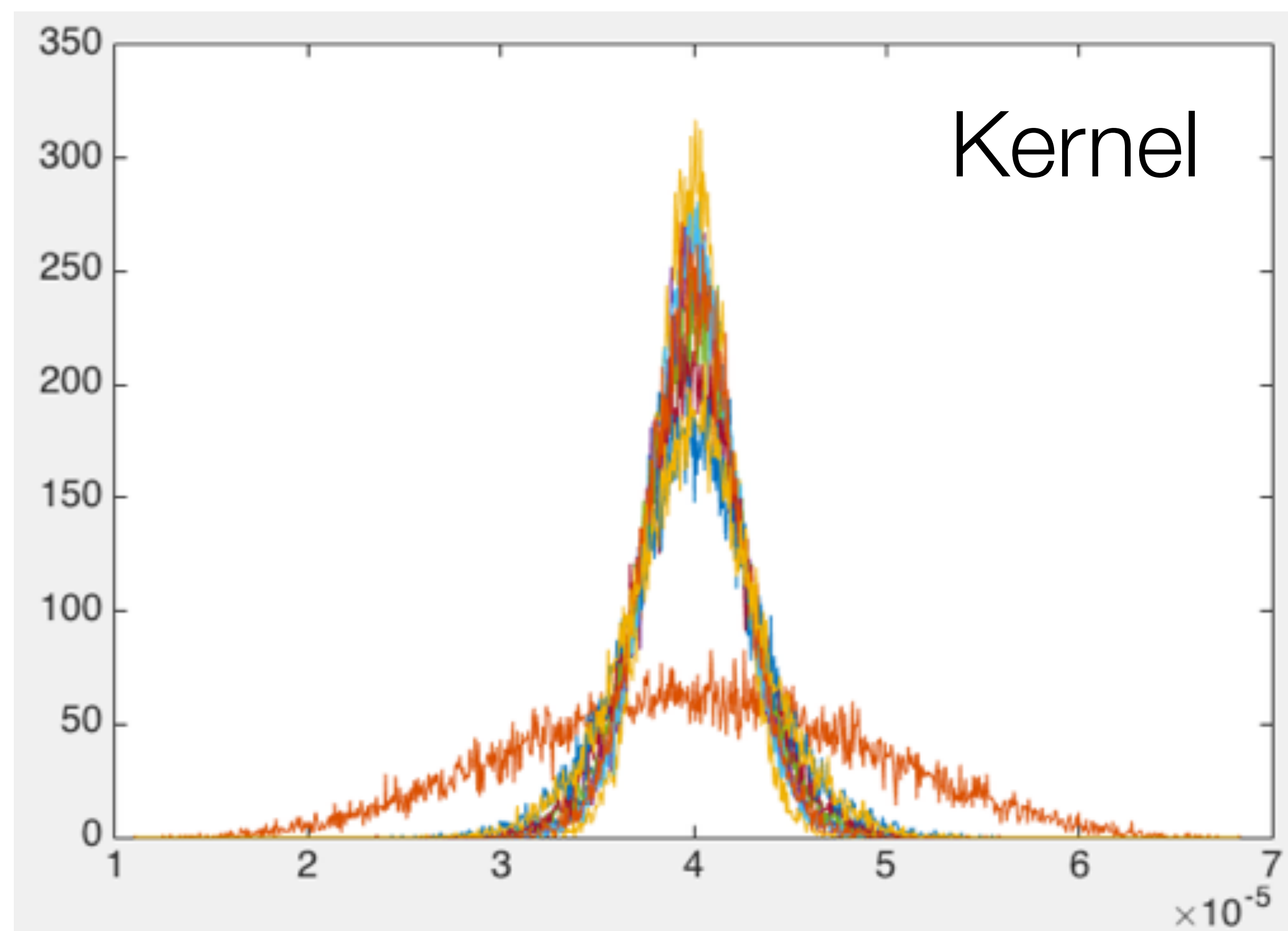
How robust of using random features?



How different random vs. non-random features?



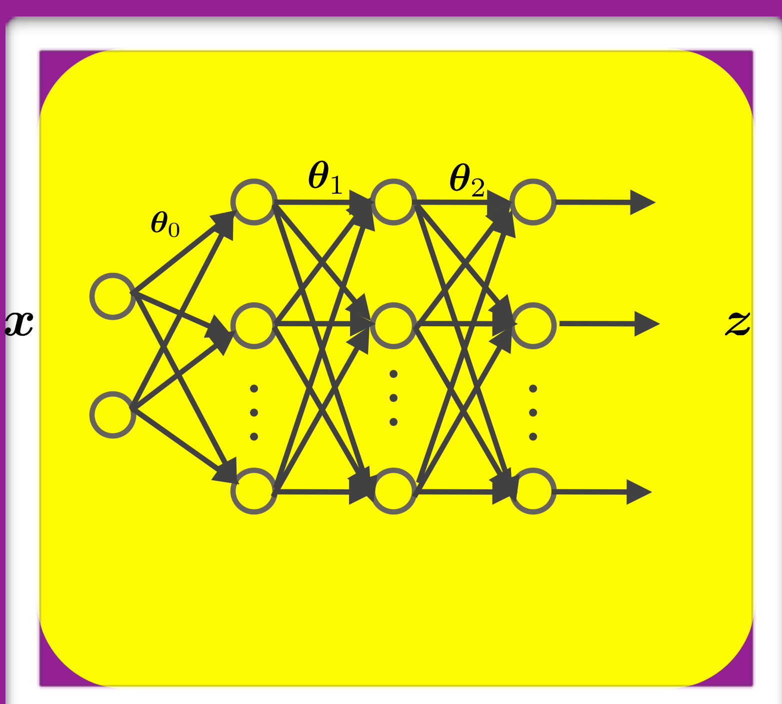
Histogram of average features per category



Take-home message: no method is magic or panacea

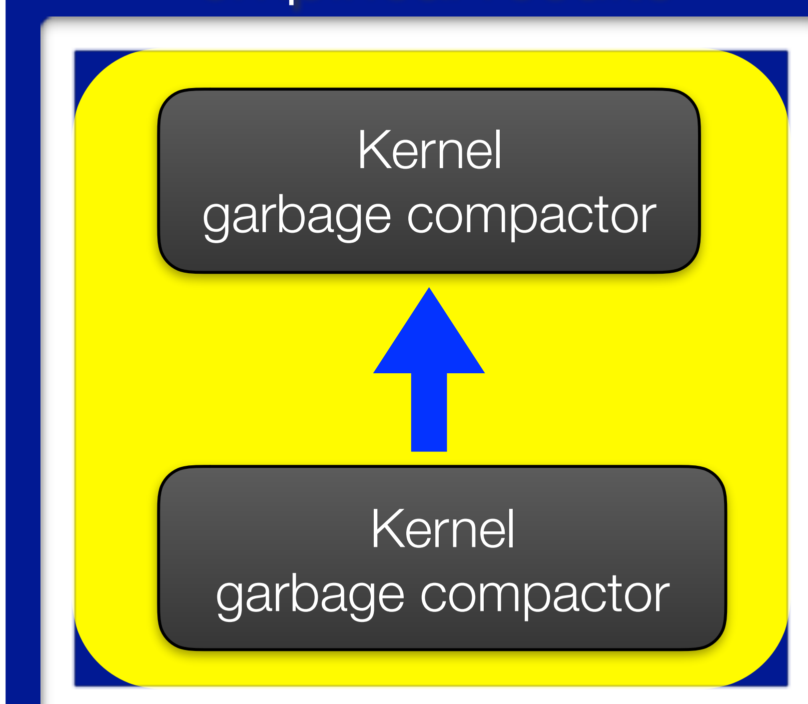
Deep neural networks

deep
scalable
strong empirical success



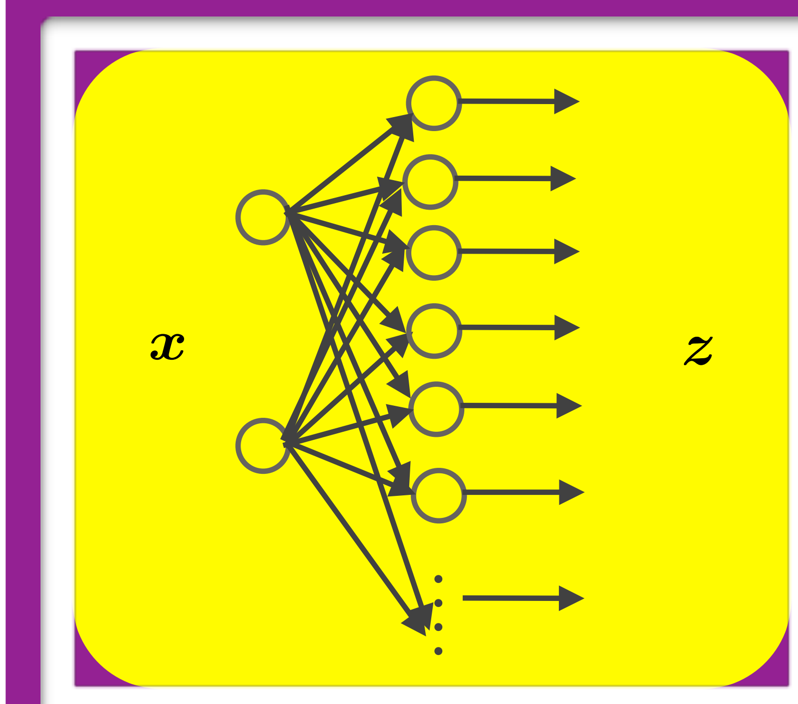
Garbage compactors

not too shallow or deep
scalable
strong theoretical and
empirical results



Kernel methods

shallow
does not scale
strong theoretical results



Details of the work in this talk

arXiv.org > cs > arXiv:1411.4000

Search or

Computer Science > Learning

How to Scale Up Kernel Methods to Be As Good As Deep Neural Nets

Zhiyun Lu, Avner May, Kuan Liu, Alireza Bagheri Garakani, Dong Guo, Aurélien Bellet, Linxi Fan, Michael Collins, Brian Kingsbury, Michael Picheny, Fei Sha

(Submitted on 14 Nov 2014)

In this paper, we investigate how to scale up kernel methods to take on large-scale problems, on which deep neural networks have been prevailing. To this end, we leverage existing techniques and develop new ones. These techniques include approximating kernel functions with features derived from random projections, parallel training of kernel models with 100 million parameters or more, and new schemes for combining kernel functions as a way of learning representations. We demonstrate how to muster those ideas skillfully to implement large-scale kernel machines for challenging problems in automatic speech recognition. We valid our approaches with extensive empirical studies on real-world speech datasets on the tasks of acoustic modeling. We show that our kernel models are equally competitive as well-engineered deep neural networks (DNNs). In particular, kernel models either attain similar performance to, or surpass their DNNs counterparts. Our work thus avails more tools to machine learning researchers in addressing large-scale learning problems.

[Arxiv 2014]

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*

[ICASSP 2016]

Acknowledgemens

Collaborators

U. of Southern California

Zhiyun Lu, Kuan Liu, Alireza Bagheri Garakani, Dong Guo, Aurelien Bellet (now at INRIA)

IBM Research Speech Group

Brian Kingsbury, Michael Picheny

Columbia

Michael Collins, Avner May, Linxi Fan

Funding

IARPA BABEL Program, ARO, Google Research Award