The limits and power of kernels

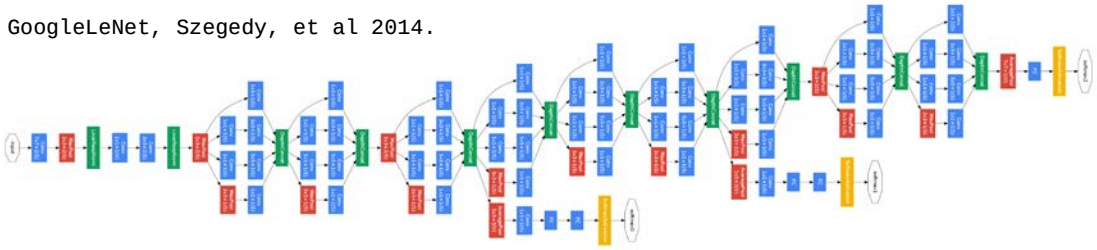
Simons Institute, Nov 2017
Optimization, statistics and uncertainty

Mikhail Belkin, Ohio State University,
Department of Computer Science and Engineering,
Department of Statistics,

Collaborators: Siyuan Ma, Raef Bassily, Chaoyue Liu

Machine Learning/AI is becoming a backbone of commerce and society.

GoogleLeNet, Szegedy, et al 2014.



The fog of war:

What is new and what is key?

Goal: a model for modern ML

- competitive on modern data
- analytically tractable
- convex

This talk

1. Limits of kernels.
2. The power of kernels: making kernels competitive on large data. [Ma, **B.** NIPS 2017]
3. Why is SGD so effective? Overfitting: a modern innovation and a puzzle. [Ma, Bassily, **B.** NIPS 2017]
4. Modern behavior of kernels (once computation is addressed).
   - SGD
   - Overfitting
   - Acceleration/momentum infinite condition number.

   [Liu, **B.** 2017]

Modern ML

Computation is key. f^{ERM} is found algorithmically.

Large data:

Map to (fast!) GPU (matrix x vector products)

→ limits algorithms available

→ limits # matrix x vector products

ERM algorithmic requirements:

small # of matrix x vector products

“Shallow”/kernel architectures

Feature map $\phi\colon \mathbb{R}^d \to \mathcal{H}$ (Hilbert space)
Followed by linear regression/classification.

$$w^* = arg\min_{w\in\mathcal{H}} \frac{1}{n}\sum L\left(\langle w, \phi(x_i)\rangle, y_i\right)$$

Classifier: $y(x) = \langle w, \phi(x_i)\rangle$ (regression / sign for classification)

Kernel methods. RKHS $\mathcal{H}$ is infinite dimension:

$\phi\colon x \to K(x,\cdot) \in \mathcal{H}$ (K is psd kernel, e.g., Gaussian)

Output: $y(x) = \sum_i \alpha_i K(x_i, x)$

Kernel learning for modern ML

Beautiful classical statistical/mathematical theory.

RKHS Theory [Aronszajn,…, 50s]

Splines [Parzen, Wahba,…, 1970-80s]

Kernel machines [Vapnik,…, 90s]

Perform well on small data/not as well on large data.

Intrinsic architectural limitation?

Issue: standard methods practical for large data/GPU have low computational reach/fitting capacity for fixed computational budget.

Addressing computational reach results in major speed/accuracy improvements.

Kernel methods for big data

Regression/classification, square loss.

$$K\,\alpha^* = y$$

Direct inversion: cost n^3 (does not map to GPU).

Small data:

$\mathrm{n} = 10^4$: $\mathrm{n}^3 = 10^{12}$ FLOPs easy.

Big data:

$\mathrm{n} = 10^7$: $\mathrm{n}^3 = 10^{21}$ FLOPS impossible! (Modern GPU $\sim 10^{13}$ FLOPs/ CPU $\sim 10^{11}$ max)

Iterative: $\alpha^{(t)} = \alpha^{(t-1)} - \eta\left(K\alpha^{(t-1)} - y\right)$. [Richarso

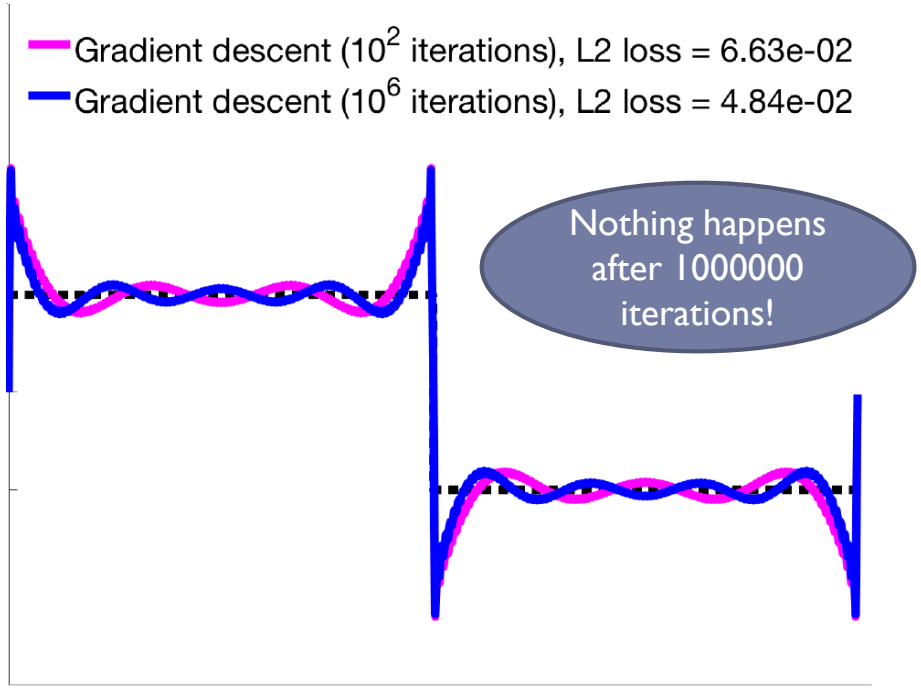
SGD is much cheaper

Cost n^2 per iteration. GPU compatible.

$\mathrm{n} = 10^7$: $\mathrm{n}^2 = 10^{14}$ FLOPS per iteration feasible.

But how many iterations?

Simple 1-D example: Heaviside function



Real Data: gradient descent

N_{iter}	MNIST-10k		HINT-M-10k	
	train	test	train	test
10240	2.36e-3	3.64e-2	1.83e-2	**3.14e-2**
81920	2.17e-5	**3.55e-2**	4.21e-3	3.42e-2

Need >10k iterations on 10k point dataset.
Worse than a direct method (n^3)!

The limits of kernels

Theorem 1: Let $K(x,z)$ be a smooth radial and let

$$\mathcal{K}f(x) := \int K(x,z)f(z)\,d\mu$$

Then $\lambda_i(\mathcal{K}) < C\,e^{-C' i^{1/d}}$, where C, C' do not depend on μ.

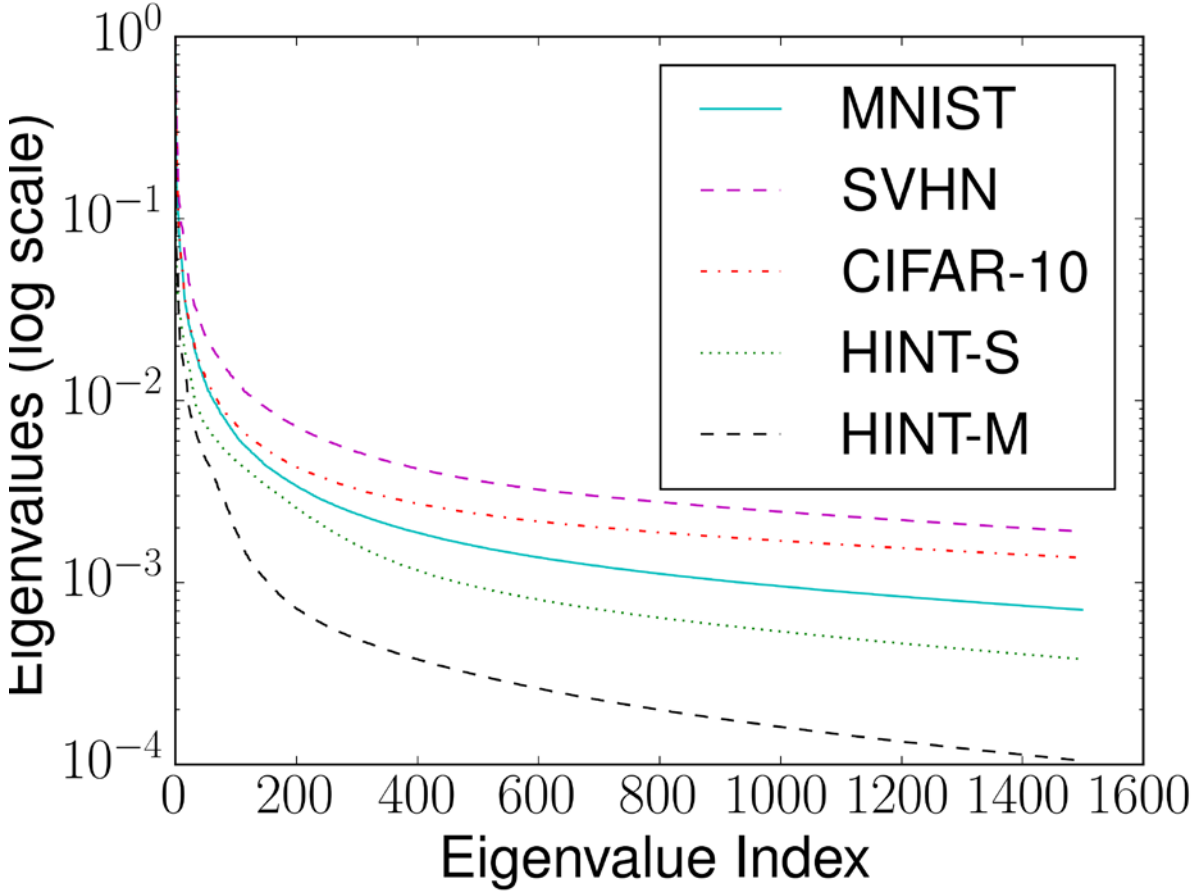
Corollary (approximation beats concentration): If μ is supported on a finite set of points (e.g., sampled from density), eigenvalues of the corresponding kernel matrix decay nearly exponentially.

Theorem 2: Fat shattering (V_γ)-dimension of function reachable by t iterations of gradient descent is at most $O(\log^d(t/\gamma))$. (Does not require square loss).

[Ma, **B.** NIPS 2017, **B.** 2017.]

Related work: [Santin, Schaback, 16], [Schaback, Wendland, 02]

Eigenvalue decay



Computational reach of Kernel GD

Theorem 3: Only very smooth functions can be $\epsilon -$ approximated by a smooth kernel in $t = Poly(1/\epsilon)$ number of iterations of gradient descent.

Contradicts classical $1/\epsilon^2$ rate for GD?

Classification functions are generally not that smooth.

Need $\frac{\lambda_1}{\lambda_i} \approx e^{i^{1/d}}$ iterations for i'th direction.

EigenPro Kernel

Problem: fast eigenvalue decay.

Solution: construct a kernel with flatter spectrum.

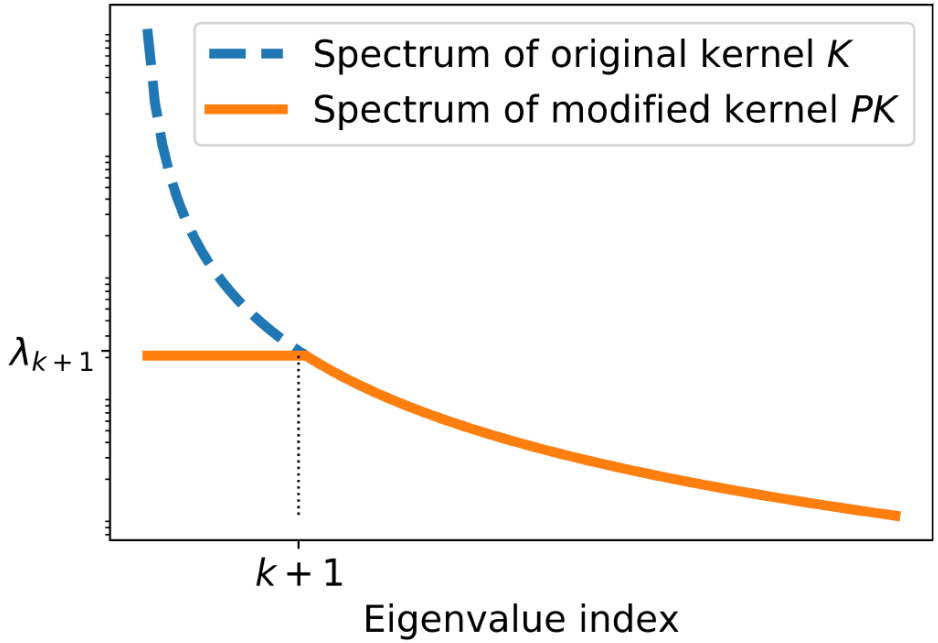
Original kernel:

$$K(x,z) = \sum_{i=1}^{\infty} \lambda_i e_i(x) e_i(z)$$

EigenPro kernel:

$$K_{EiP}(x,z) = \sum_{i=1}^{k} \lambda_{k+1} e_i(x) e_i(z) + \sum_{i=k+1}^{\infty} \lambda_i e_i(x) e_i(z)$$

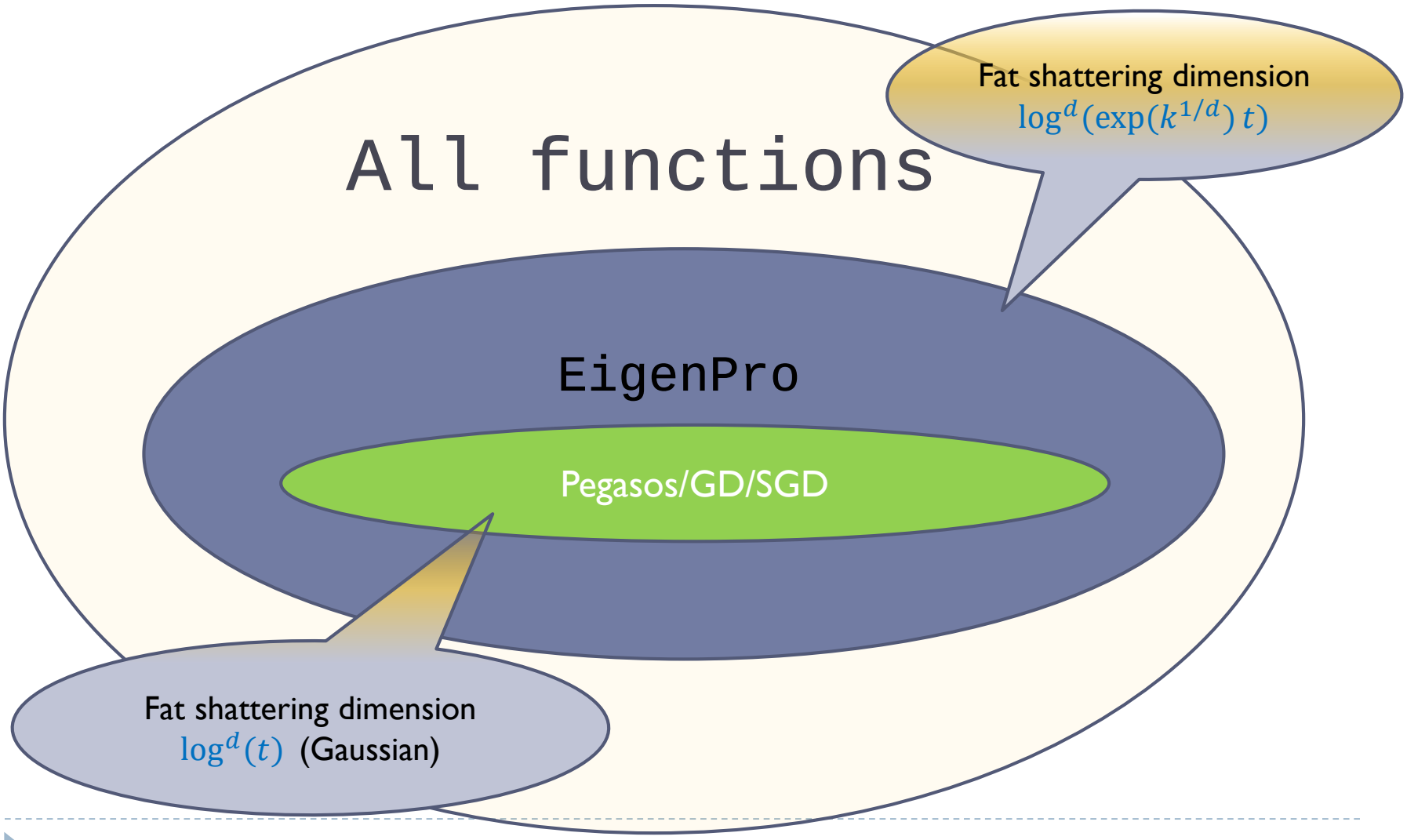
EigenPro kernel learning



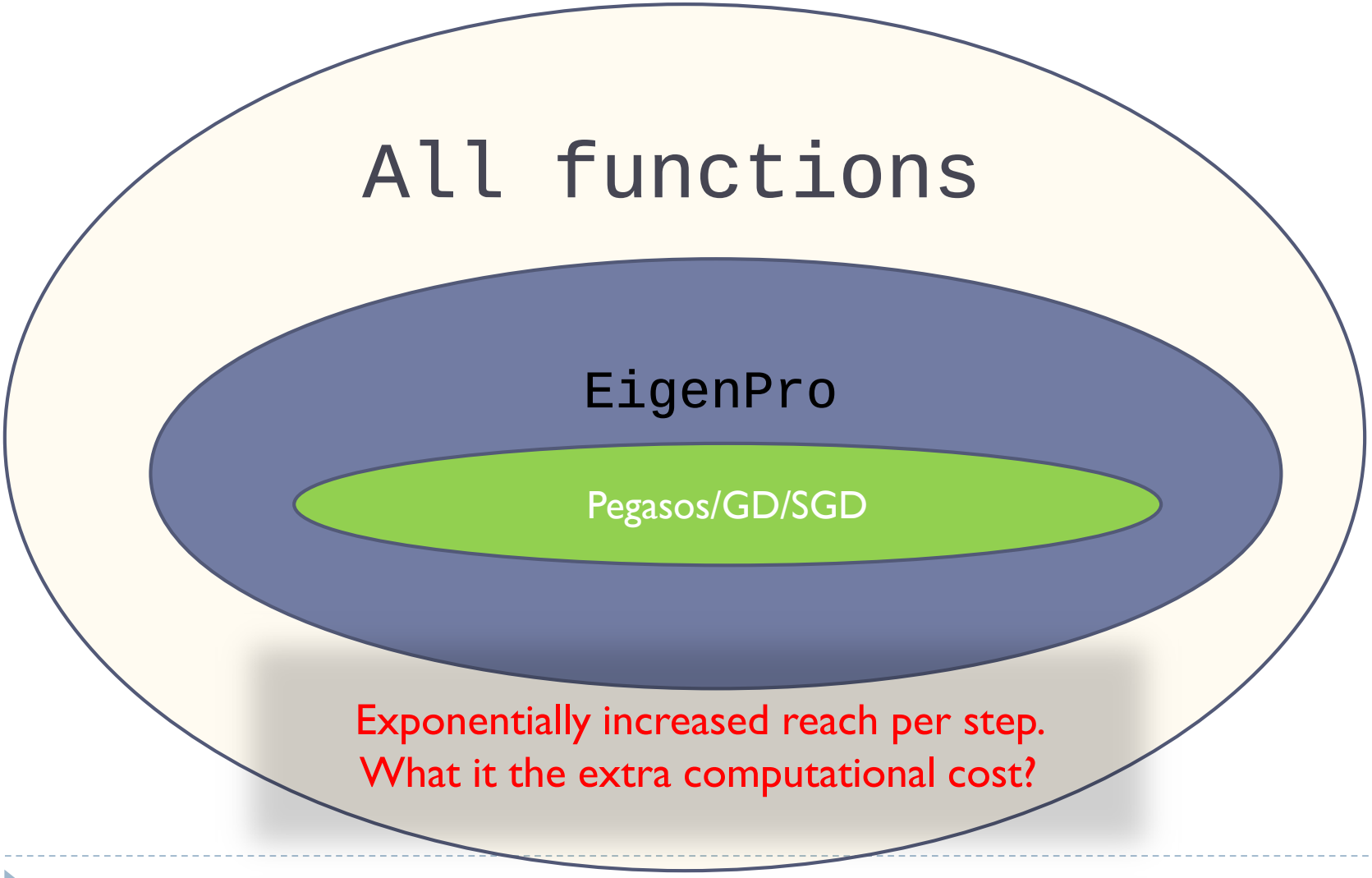
Fits first $e_1, \ldots, e_k$ in one iteration.

Approximately λ_1/λ_{k+1} acceleration for each $i > k$.

Computational reach after t iterations



Computational reach after t iterations



EigenPro: practical implementation

Preconditioned gradient descent.

$$P = I - \sum_{i=1}^{k} \left(1 - \frac{\lambda_{k+1}}{\lambda_i}\right) e_i e_i^T$$

$$w^{(t)} = w^{(t-1)} - \eta P(Kw^{(t-1)} - y)$$

Use RSVD or Nystrom to compute first k eigenvectors of K.

Preconditioner P is formed only once.
(No regularization!)

Related work: [Fasshauer, McCourt, 12], [Erdogdu, Montanari, 15], [Gonen, et al, 16].

EigenPro: practical implementation II

$$P = I - \sum_{i=1}^{k} \left(1 - \frac{\lambda_{k+1}}{\lambda_i}\right) e_i e_i^T$$

$$w^{(t)} = w^{(t-1)} - \eta P(Kw^{(t-1)} - y)$$

Important points:

- Low initial cost: P is estimated from a small subsample.
- Low overhead/iteration (~15-20% in practice).

Robustness: converges to the correct solution for any P.

Potentially exponential acceleration $\frac{\lambda_1}{\lambda_{k+1}}$.

EigenPro: Stochastic Gradient Descent

Key to effective implementation.

Have to sacrifice some acceleration.

Theorem: minibatch size m.

$$\lambda_1(PK_m) \lesssim \lambda_{k+1} + O\left(\sqrt{\frac{\lambda_{k+1}}{m}}\right)$$

- When minibatch size m is small $\sqrt{\frac{\lambda_{k+1}}{m}}$ is the dominant term.
- Hence acceleration factor $\frac{\lambda_1\sqrt{m}}{\sqrt{\lambda_{k+1}}}$.

EigenPro acceleration

epochs for different kernels

Dataset	Size	Gaussian Kernel		Laplace Kernel		Cauchy Kernel	
		EigenPro	Pegasos	EigenPro	Pegasos	EigenPro	Pegasos
MNIST	6×10^4	**7**	77	**4**	143	**7**	78
CIFAR-10	5×10^4	**5**	56	**13**	136	**6**	107
SVHN	7×10^4	**8**	54	**14**	297	**17**	191
HINT-S	5×10^4	**19**	164	**15**	308	**13**	126

6x-35x acceleration factor.

Comparison with state-of-the-art

Dataset	Size	EigenPro (use 1 GTX Titan X)		Reported results for other methods		
		error	GPU hours	error	description	source
MNIST	$1 \cdot 10^6$	**0.70%**	4.8	0.72%	1.1 hours (189 epochs) on 1344 AWS vCPUs	PCG [ACW16]
	$6.7 \cdot 10^6$	**0.80%**†	0.8	0.85%	less than 37.5 hours on 1 Tesla K20m	[LML+14]
TIMIT	$2 \cdot 10^6$	**31.6%**	3.9	33.5%	512 IBM BlueGene/Q cores	Ensemble [HAS+14]
				33.5%	7.5 hours on 1024 AWS vCPUs	BCD [TRVR16]
				30.9%	multiple AWS g2.2xlarge instances	SparseKernel [MGL+17] (learned features)
				32.4%		DNN [MGL+17]
SUSY	$4 \cdot 10^6$	**19.8%**	0.1	$\approx$ 20%	0.6 hours on IBM POWER8	Hierarchical [CAS16]

Better performance with (far) less computational budget.

This talk

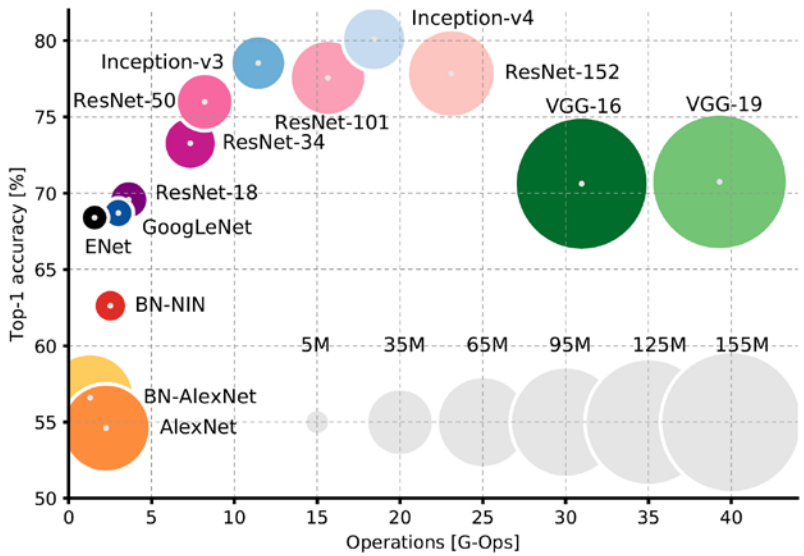
1. Algorithmic requirements of modern ML.
2. Making kernels competitive on large data.
3. Why is SGD so effective? Overfitting: a modern innovation and a puzzle.
4. Modern behavior of kernels (once computation is addressed).
   - SGD
   - Overfitting
   - Acceleration

Modern ML

Major innovation: systematic overfitting

parameters >> # training data

From Canziani, et al., 2017.



The best way to solve the problem from practical standpoint is you build a very big system. If you remove any of these regularizations like dropout or L2, basically you want to make sure you hit the zero training error. Because if you don't, you somehow waste the capacity of the model.

Ruslan Salakhutdinov's Simons tutorial, 2017.

Over-parametrization → interpolation.

All local minima (for the training data) are global?

[Kawaguchi, 16] [Soheil, et al, 16] [Bartlett, et al, 17] [Soltanolkotabi, et al, 17]…

Modern ML

1. Why are large models easy to optimize?

Very large models → over-parametrization → interpolation.

Will show small batch SGD is highly effective in the interpolated regime.

[Ma, **B.**, Bassily, 2017]

2. Why do large models perform well?

Seems to contradict classical generalization results.

Cf. [Zhang, et al, 2017].

model	# params	random crop	weight decay	train accuracy	test accuracy
Inception	1,649,402	yes	yes	100.0	89.05
		yes	no	100.0	89.31
		no	yes	100.0	86.03
		no	no	100.0	85.75

[CIFAR 10, from Zhang, et al, 2017]

$$E(L(f^{ERM}, y)) \leq \frac{1}{n}\sum L\left(f^{ERM}(x_i), y_i\right)) + \sqrt{c/n}$$

We don't know why (margins are probably not the whole story).

Will show parallel experimental results for kernels in the convex setting.

Stochastic Gradient Descent

All major architectures use SGD.

$$w^* = \operatorname*{argmin}_{\mathrm{w}} L(w) = \operatorname*{argmin}_{\mathrm{w}} \frac{1}{n} \sum L_i(w), \quad L_i(w) = (f_w(x_i) - y_i)^2 \text{ (e.g.)}$$

SGD Idea: optimize $L_i(w)$ sequentially (instead of $L(w)$).
However: each $L_i(w)$ only weakly related to $L(w)$.

General analysis is complex. Need to control variance.

[Moulines, Bach, 2011], [Nedell, Srebro, Ward 2014]

But $\forall_i \, L_i(w^*) = 0 \rightarrow$ exponential convergence.

(cf. original Perceptron analysis, Kaczmarz 37)

Understanding SGD

Key: Interpolation → fast (exponential) convergence!
But how fast is fast? Can be analyzed explicitly.

Quadratic case (or close to a minimum):

$$E\left(||w_{t+1} - w^*||^2\right) \leq g(m,\eta) E\left(||w_t - w^*||^2\right)$$

$$g^*(m) = \arg\min_{\eta} g(m,\eta)$$

Theorem 1 [optimality of $m = 1$ for sequential computation].

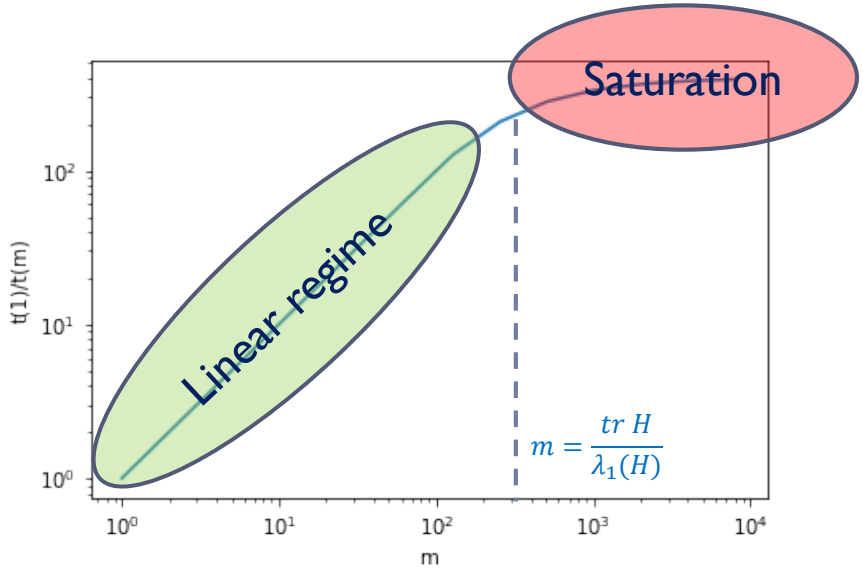
$$g^*(1) \leq g^*(m)^{\frac{1}{m}}$$

[Ma, Bassily, **B.**, 2017]

Batch size for parallel computation

Theorem 2 [optimality for parallel computation]:

Minibatch size $m = \frac{tr\ H}{\lambda_1(H)}$ is (nearly) optimal in low cost parallel computation model.



Consistent with "linear scaling rule" observed empirically in neural nets [Goyal, et al, 17].

Empirical results: MNIST-10k.

N_{iter}	$m=1$			$m=16$			$m=256$			$m=n=10^4$		
	L2 loss		c-error	L2 loss		c-error	L2 loss		c-error	L2 loss		c-error
	train	test	(test)	train	test	(test)	train	test	(test)	train	test	(test)
1	3.93e-1	3.92e-1	89.91%	4.38e-1	4.36e-1	63.86%	4.24e-1	4.24e-1	50.62%	4.07e-1	4.07e-1	38.50%
80	2.02e-1	2.00e-1	28.29%	1.08e-1	1.08e-1	10.92%	9.76e-2	9.91e-2	7.66%	9.61e-2	9.74e-2	7.60%
1280	8.65e-2	8.93e-2	7.53%	2.75e-2	4.78e-2	3.30%	2.64e-2	4.62e-2	3.26%	2.60e-2	4.59e-2	3.26%
10240	3.44e-2	5.29e-2	3.13%	2.42e-3	3.63e-2	2.49%	2.48e-3	3.64e-2	2.48%	2.36e-3	3.64e-2	**2.39%**
81920	**2.27e-3**	**3.64e-2**	**2.40%**	**5.41e-5**	**3.55e-2**	**2.42%**	**2.86e-5**	**3.55e-2**	**2.41%**	**2.17e-5**	**3.55e-2**	2.49%

Gradient computations to reach **optimum** (Gaussian kernel):

$m = 1 \quad \rightarrow \quad 8 * 10^4$

$m = 16 \quad \rightarrow \quad 1.6 * 10^5$

$m = 256 \quad \rightarrow \quad 2.6 * 10^6$

$m = 10^4 \quad \rightarrow \quad 1 * 10^7$

Can we build architectures for parallel computation?

Saturation $m = 5{\sim}10.$ Close to theoretical bound $m = \frac{tr\ H}{\lambda_1(H)} \approx 3.5.$

This talk

1. Algorithmic requirements of modern ML.
2. Making kernels competitive on large data.
3. Why is SGD so effective? Overfitting: a modern innovation and a puzzle.
4. Modern behavior of kernels (once computation is addressed).
   - SGD
   - Overfitting
   - Acceleration

Overfitting with kernels

parameters = # training data

N_{Epochs}	Primal							
	EigenPro (k = 160)				Pegasos			
	c-error		L2 loss		c-error		L2 loss	
	train	test	train	test	train	test	train	test
1	0.92%	2.03%	2.4e-2	3.2e-2	5.12%	5.21%	6.9e-2	6
5	0.10%	1.44%	8.6e-3	2.4e-2	2.36%	2.84%	4.0e-2	5e-2
10	0.01%	1.23%	4.3e-3	2.2e-2	1.58%	2.32%	3.1e-2	3.6e-2
20	0.0%	**1.20%**	1.8e-3	2.1e-2	0.90%	1.93%	2.3e-2	3.1e-2
40	0.0%	1.20%	6.1e-4	2.1e-2	0.39%	1.65%	1.6e-2	2.7e-2
80	0.0%	1.23%	2.2e-4	2.1e-2	0.14%	1.41%	9.7e-3	2.4e-2
160	0.0%	1.21%	8.0e-5	2.1e-2	0.03%	1.24%	5.1e-3	2.2e-2
320	0.0%	1.23%	3.0e-5	2.1e-2	0.01%	**1.23%**	2.2e-3	2.1e-2

Hard to interpolate using standard methods

Overfitting

Performs well!

Interpolation

[MNIST, Ma, B., 2017]

Kernel overfitting/interpolation

Why do overfitted (interpolated) models perform so well?

Dataset	Size	Kernel	(EigenPro) epochs	Result			
				c-error		L2 loss	
				train	test	train	test
MNIST	$1 \cdot 10^6$	Gaussian	10	0.02%	0.76%	9.5e-4	4.1e-3
	$2 \cdot 10^6$	Laplace	3	0.0%	0.77%	7.3e-4	3.8e-3
			10	0.0%	0.80%	6.6e-6	3.7e-3
TIMIT	$1 \cdot 10^6$	Gaussian	10	1.6%	31.6%	9.0e-4	-
		Laplace	3	1.1%	32.0%	8.0e-4	-
			10	0.0%	31.6%	8.2e-5	-
			20	0.0%	31.6%	2.3e-5	-

We still don't know.

However:

1. Not a unique feature of deep architectures.
2. Can be examined in a convex analytical setting.

Moreover, Laplace kernel take ~3x iterations to fit random labels.

Same as reported for ReLU nets.

[Zhang, et al, 17].

This talk

1. Algorithmic requirements of modern ML.
2. Making kernels competitive on large data.
3. Why is SGD so effective? Overfitting: a modern innovation and a puzzle.
4. Modern behavior of kernels (once computation is addressed).
   - SGD
   - Overfitting
   - Acceleration

Accelerated/momentum/Nesterov methods

Almost as widely used as SGD.

$$w^{(t)} = w^{(t-1)} - \eta \nabla f\left(w^{(t-1)}\right) - \eta_1 \nabla f\left(w^{(t-2)}\right)$$

Nesterov acceleration [Nesterov, 83].

Far easier to analyze in the kernel case!
Reduces to optimality of certain polynomials.

Richarson second-order, Chebyshev semi-iterative method, etc…
[Golub, Varga, 1961]

Accelerated methods for kernels

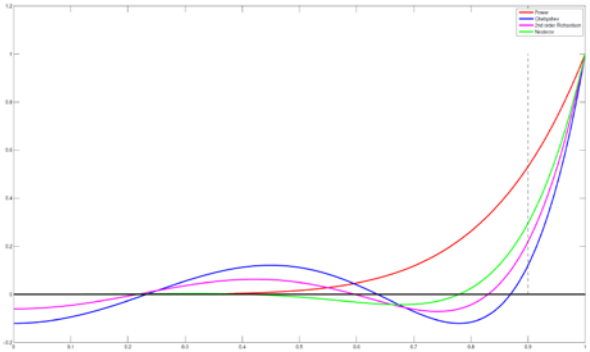
Classical analyses do not quite work: assume finite condition number. Infinite theoretically, beyond numerical precision in practice.

Actually often true even for linear regression.

Theorem:

1. Nesterov, Richardson^2, Chebyshev converge for mis-specified condition parameter.
2. For small eigenvalues

Chebyshev > (faster) Richardson^2 > Nesterov > GD.



[Liu, **B.** 2017]

Parting Thoughts

- Classical kernel methods as a convex model for modern ML.
  - Once computation is addressed, competitive performance and “modern” behavior.
  - Design kernels for (parallel?) computation.
- SGD very effective for over-parametrized methods.
  - But why do over-parametrized methods generalize?
- Infinite condition numbers are everywhere.
- Approximation vs optimization vs statistics?