An Interpolation Perspective on Modern Machine Learning

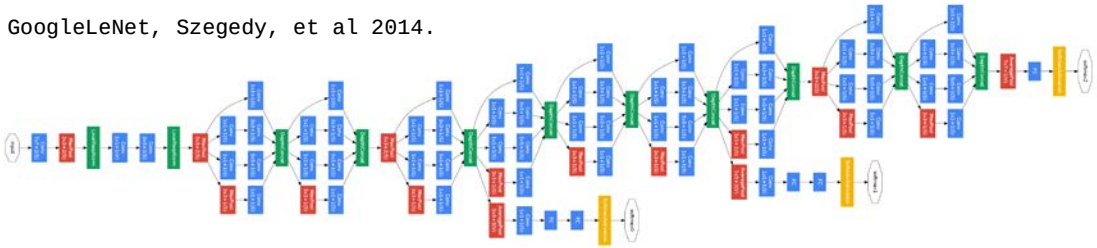
Sept 2018
Purdue Workshop on Approximation and ML

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

Collaborators: Siyuan Ma, Soumik Mandal, Raef Bassily, Daniel Hsu, Partha Mitra, Alexander Rakhlin, Alexandre Tsybakov

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 important?

Isolate and analyze components.

Supervised ML

Data $(x_i, y_i),\ i = 1..n\,, x_i \in \mathbb{R}^d, y_i \in \{-1,1\}$

Goal: construct $f\colon \mathbb{R}^d \to \mathbb{R}$, that "generalizes" to unseen data.

Empirical risk minimization (basis for most algorithms):

$(f(\mathrm{x_i}) - \mathrm{y_i})^2$, e.g.

$$f^* = \underset{f \in \mathcal{H}}{argmin} \frac{1}{n} \sum L\left(f(x_i), y_i\right)$$

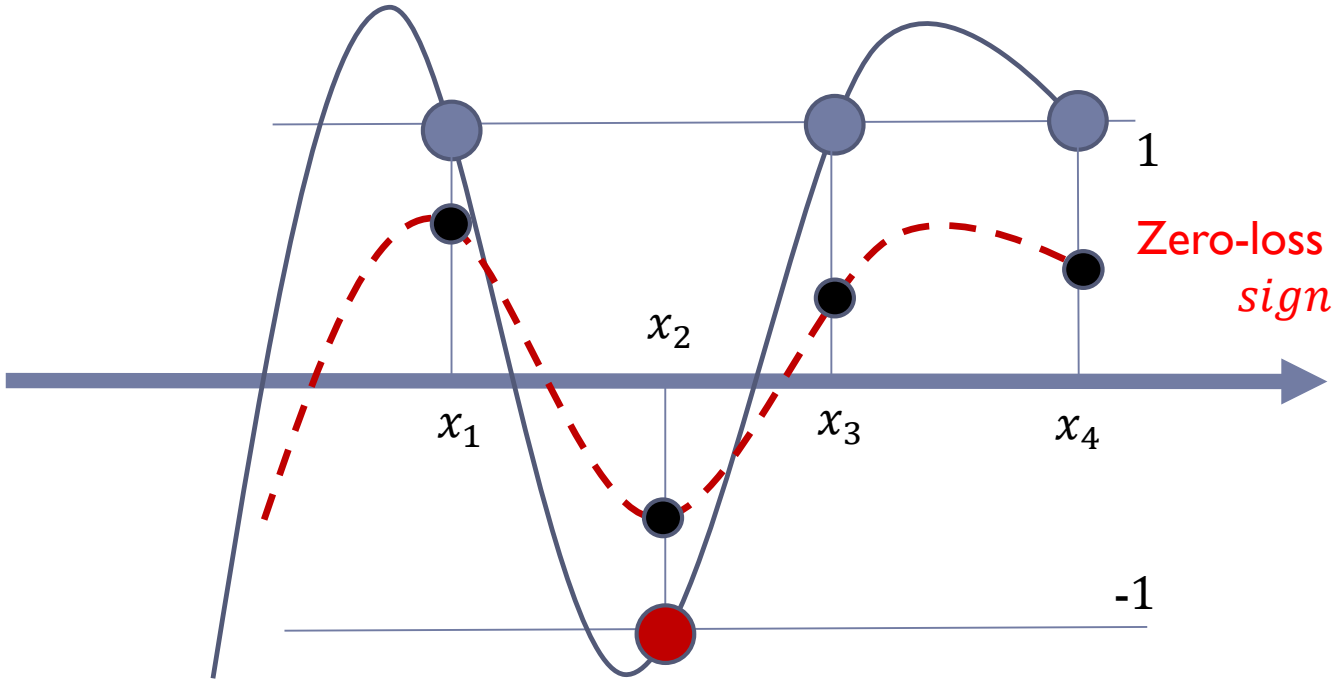
Optimized using SGD.

…

Interpolation

Data $(x_i, y_i),\ i = 1..n\ , x_i \in \mathbb{R}^d, y_i \in \{-1,1\}$

Interpolation: $f(x_i) = y_i$



Zero-loss fitting:
$sign(f(x_i)) = y_i$

Accepted wisdom

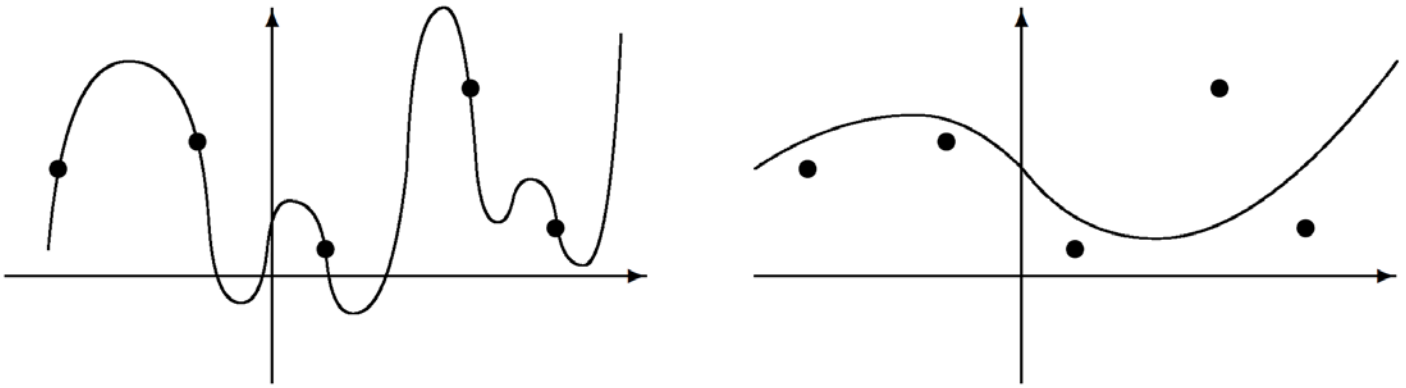
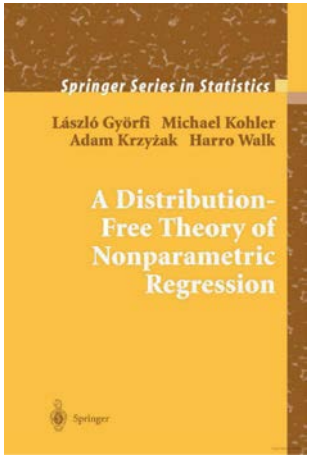


Figure 2.3. The estimate on the right seems to be more reasonable than the estimate on the left, which interpolates the data.



Who is afraid of over-fitting?

Zero loss classifiers produce near optimal results.

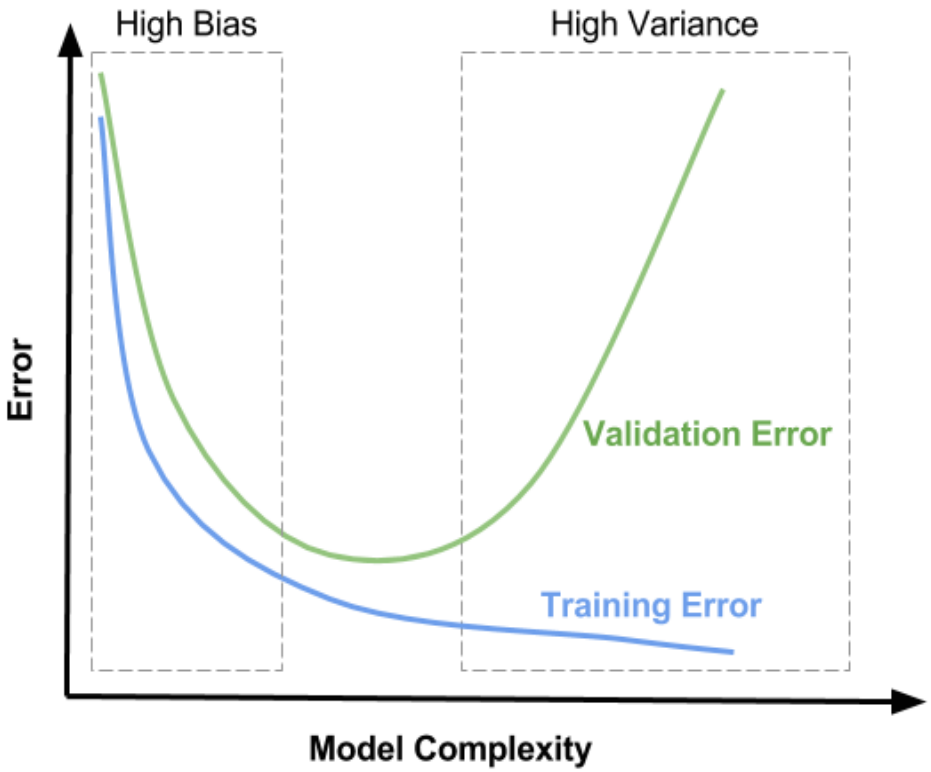
model	# params	random crop	weight decay	train accuracy	test accuracy
Inception	1,649,402	yes	yes	100.0	89.05
		yes	no	100.0	89.31
		no	yes	100.0	86.03
		no	no	100.0	85.75

[CIFAR 10, from *Understanding deep learning requires rethinking generalization*, Zhang, et al, 2017]

Ruslan Salakhutdinov's tutorial on deep learning (Simons Institute, Berkeley, 2017):

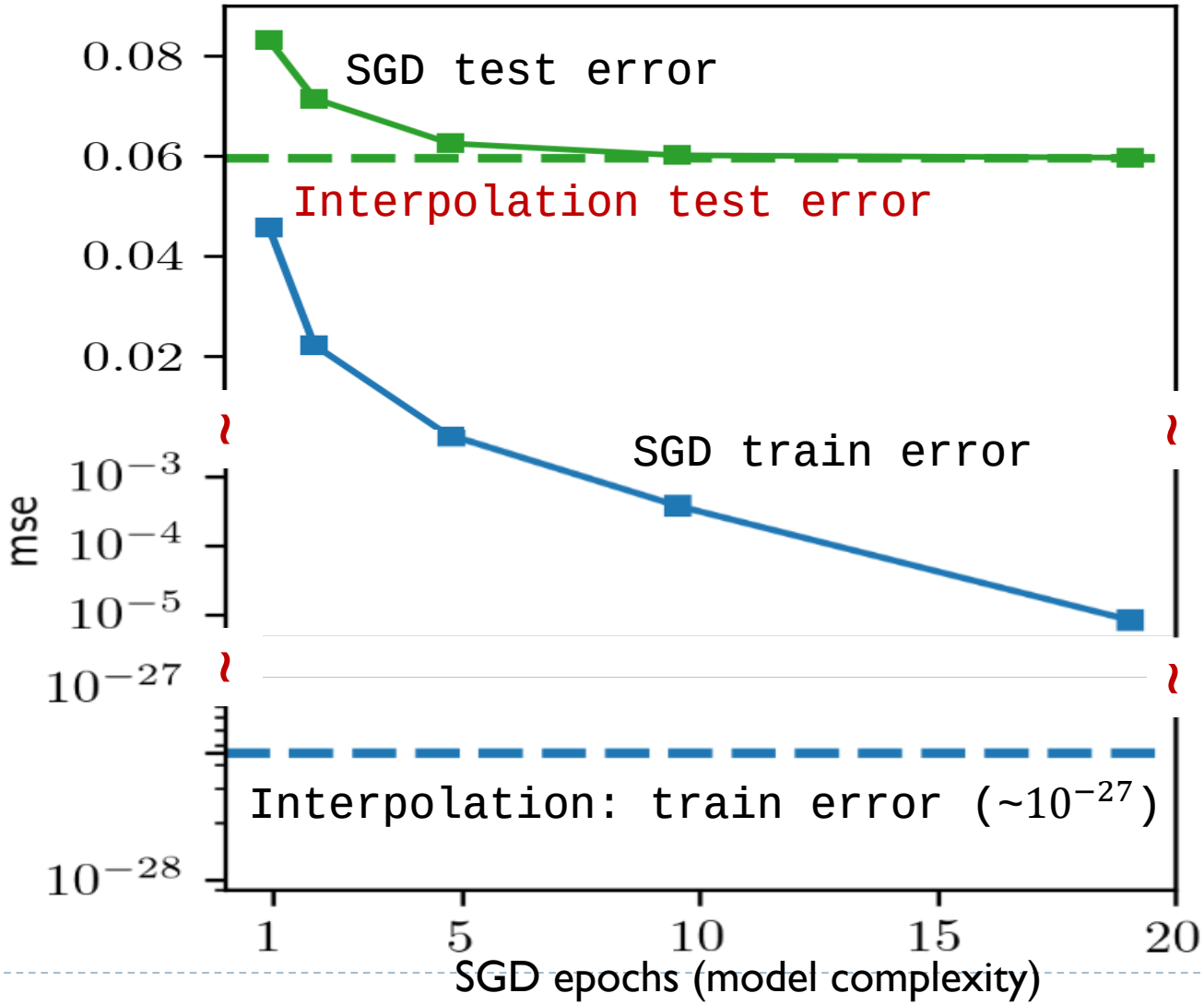
The best way to solve the problem from practical standpoint is you build a very big system. If you remove any of these regularizations like dropout or L2, basically you want to make sure you hit the zero training error. Because if you don't, you somehow waste the capacity of the model.

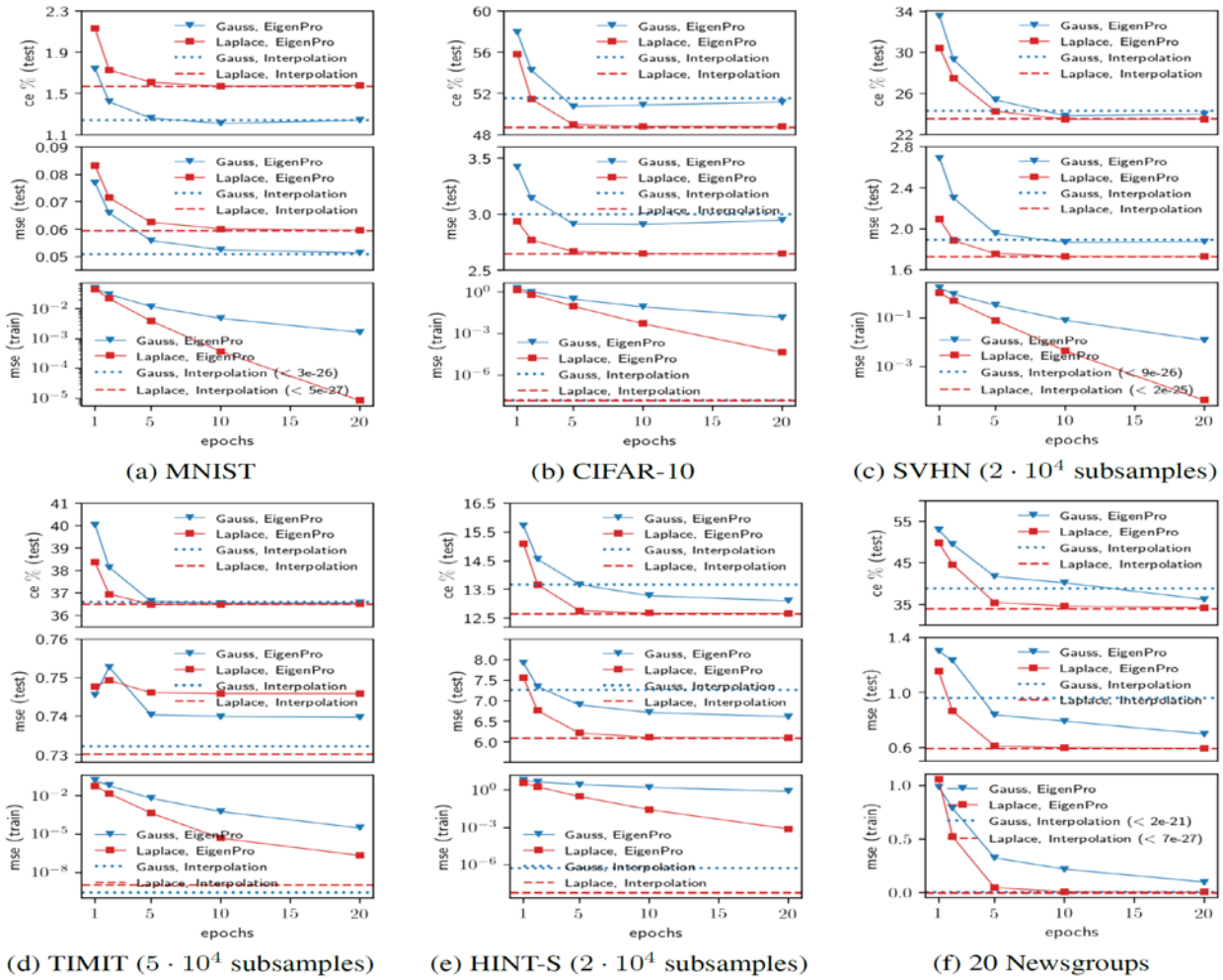
Theory



From https://www.learnopencv.com/bias-variance-tradeoff-in-machine-learning/

Practice (SGD, kernel machine)





(a) MNIST (b) CIFAR-10 (c) SVHN ($2 \cdot 10^4$ subsamples)

(d) TIMIT ($5 \cdot 10^4$ subsamples) (e) HINT-S ($2 \cdot 10^4$ subsamples) (f) 20 Newsgroups

[B., Ma, Mandal, ICML 18]

A new phenomenon?

Interpolated classifiers produce near optimal results.

Deep Neural Networks (Zhang, et al, 17, but also much earlier)
Kernel Machines (our work)
Random Forests (PERT, Cutler, Zhao, 2001)
Adaboost (Schapire, et, al. 1998)

Not always recognized as such
(e.g., regularization with very small λ).

A new phenomenon?

Leo Breiman, 1995

From Reflections after refereeing papers for NIPS:

- *Why don't heavily parametrized networks overfit the data?*
- *What is the effective number of parameters?*
- *When doesn't backpropagation head for poor local minima?*
- *When should one stop backpropagation and use the current parameters?*

We are finally closing on some answers.

Two key questions

1. Why do interpolated classifiers generalize?

2. Why interpolate?

The challenge of interpolation (approximation/statistics)

- Do we have theory?
  - Not much help from existing theory.
  - Interpolated classifiers are robust to label noise.
  - Unlikely to understand neural networks until (convex) kernel machines are understood.

  [**B.**, Ma, Mandal ICML 18]

- Moving forward: provable (near-optimal) generalization for methods that interpolate.

  [**B.**, Hsu, Mitra, 18] [**B.**, Rakhlin, Tsybakov, 18]

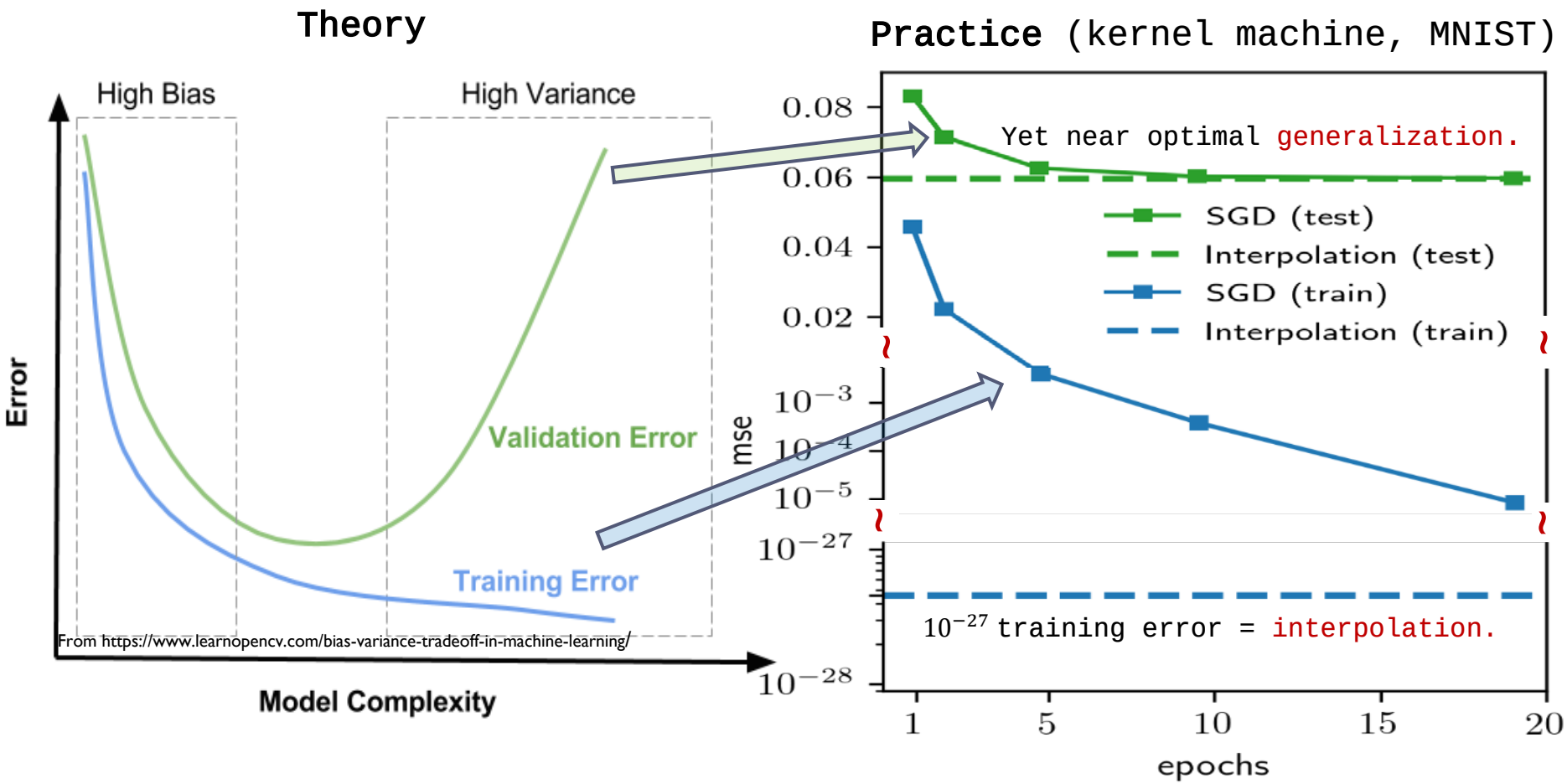
This talk: the power of interpolation (optimization)

- Optimization under interpolation
  - Why is SGD so efficient in modern learning?
  - Exponential convergence of mini-batch SGD.

  [Ma, Bassily, **B.**, ICML 18]

- Application: Fast and simple kernel machines for large data.
  - Simplicity: no regularization or loss function
  - Learning kernels that adapt to GPU.
  - Automatic mini-batch/step size selection.

  EigenPro 2.0 [Ma, **B.**, NIPS 17, 18]


Theory
High Bias
High Variance
Error
Validation Error
Training Error
Model Complexity
From https://www.learnopencv.com/bias-variance-tradeoff-in-machine-learning/
Practice (kernel machine, MNIST)
Yet near optimal generalization.
SGD (test)
Interpolation (test)
SGD (train)
Interpolation (train)
mse
0.08
0.06
0.04
0.02
10^{-3}
10^{-4}
10^{-5}
10^{-27}
10^{-28}
10^{-27} training error = interpolation.
1
5
10
15
20
epochs

Generalization bounds (interpolation)

Basic bound:

VC-dim, fat shattering, Rademacher, Covering numbers, etc…

Model or function complexity, e.g., VC or $\|f\|_{\mathcal{H}}$

Expected loss Empirical loss

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

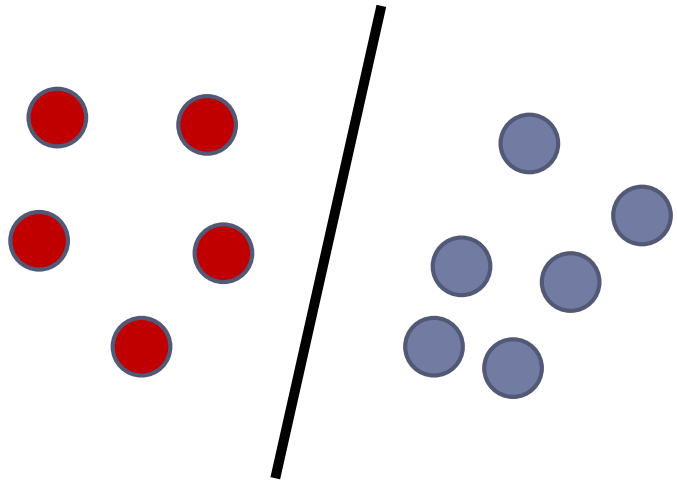
$= 0$

Interpolation

Can such bounds be useful?

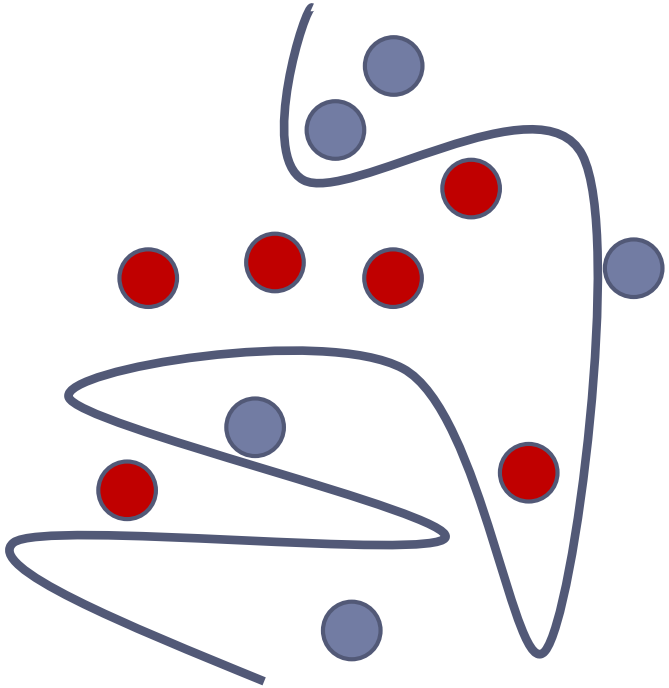
Model complexity of interpolation?

Low complexity, does not grow with data size.



Many classical results available. Margin bounds, [Schapire, et al 98], etc.

High complexity, grows linearly with data size.

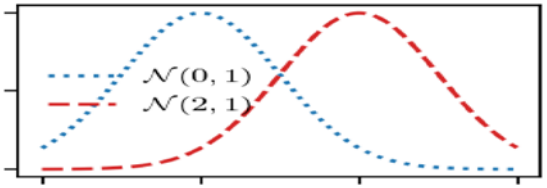


???

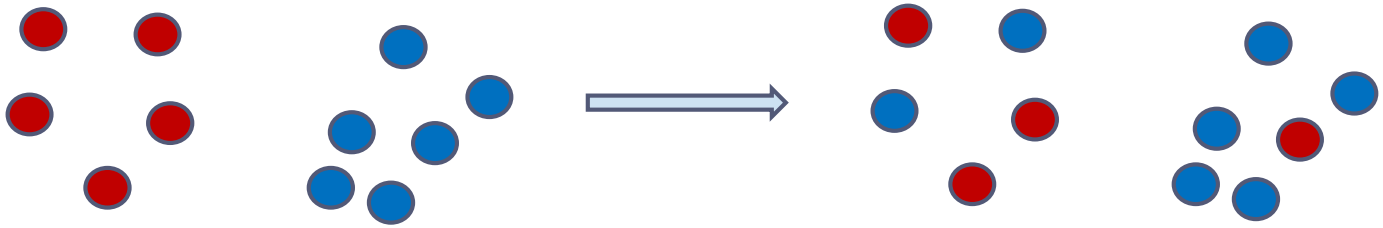
How to test model complexity?

Two ways:

1. Synthetic data.



2. Add label noise.



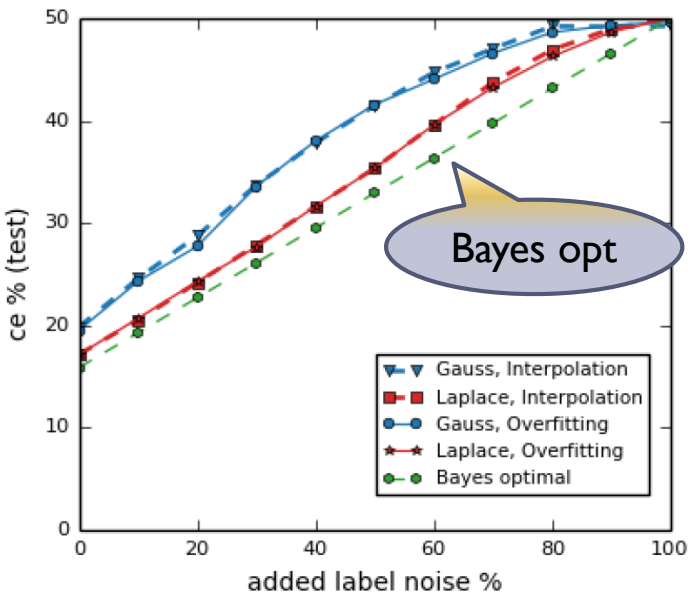
Model complexity grows but Bayes opt. does not change!

Expect overfitting to become severe as model complexity grows.

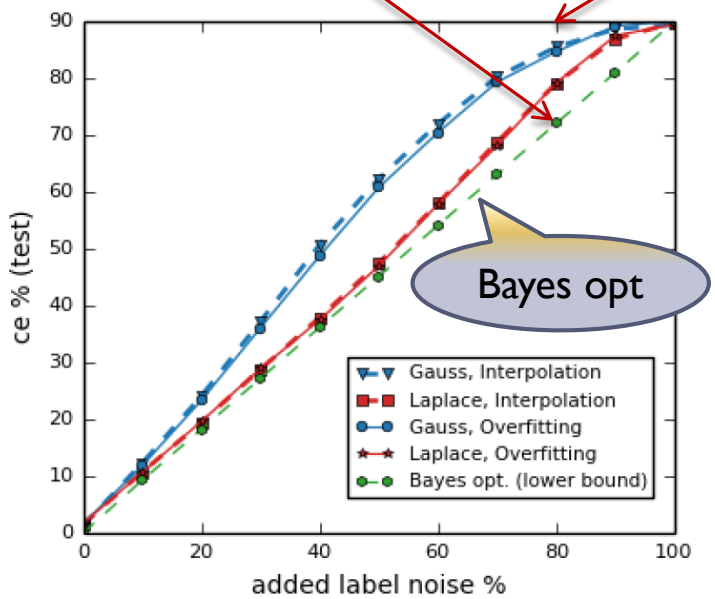
Robustness to noise

What kind of generalization bound could work here?
(hopefully correct but nontrivial)

$$0.7 < O^*\left(\sqrt{\frac{c(n)}{n}}\right) < 0.9$$



(a) Synthetic-2



(b) MNIST

$$0.7 < O^*\left(\sqrt{\frac{c(n)}{n}}\right) < 0.9 \qquad n \gg 1$$

There are no bounds like this!
Not clear whether they can exist mathematically.

e.g., can it be true if $c(n) = \phi(\|f\|_{\mathcal{H}})$?

The challenge of interpolation

- Do we have theory?
  - Not much help from existing theory.
  - Interpolated classifiers are robust to label noise.
  - Unlikely to understand neural networks until (convex) kernel machines are understood.

  [**B.**, Ma, Mandal ICML 18]

- Moving forward: provable (near-optimal) generalization for methods that interpolate.

  [**B.**, Hsu, Mitra, 18] [**B.**, Rakhlin, Tsybakov, 18]

Theoretical analyses fall short

- VC-dimension/Rademacher complexity/covering bounds.
  - Cannot deal with interpolated classifiers when Bayes risk is non-zero.
  - Empirical risk is zero – hard to bound the gap.
- Regularization-type analyses (Tikhonov, early stopping, etc.)
  - Diverge as $\lambda \to 0$ for fixed n.
- Algorithmic stability.
  - Does not apply when empirical risk is zero, expected risk non-zero.
- Classical smoothing methods (i.e., Nadaraya–Watson).
  - Most classical analyses do not support interpolation.

  (But Hilbert Regression Scheme [Devroye, et al, 1998], 1-NN, our recent results)

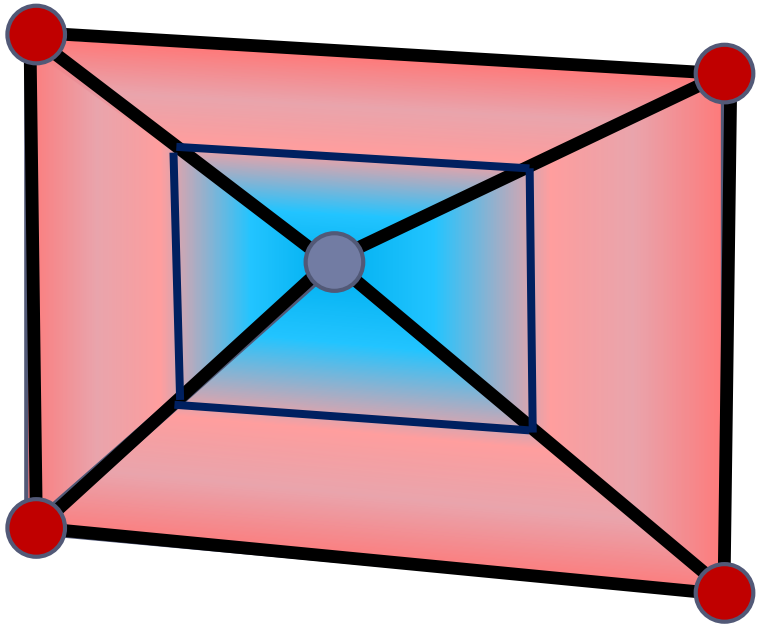
A way forward?

1-nearest neighbor classifier is very suggestive.

Interpolating classifier with a non-trivial (sharp!) performance guarantee (twice the Bayes risk).

- No margin assumptions.
- Analysis not based on uniform bounds.
- Estimate generalization, not the generalization gap.

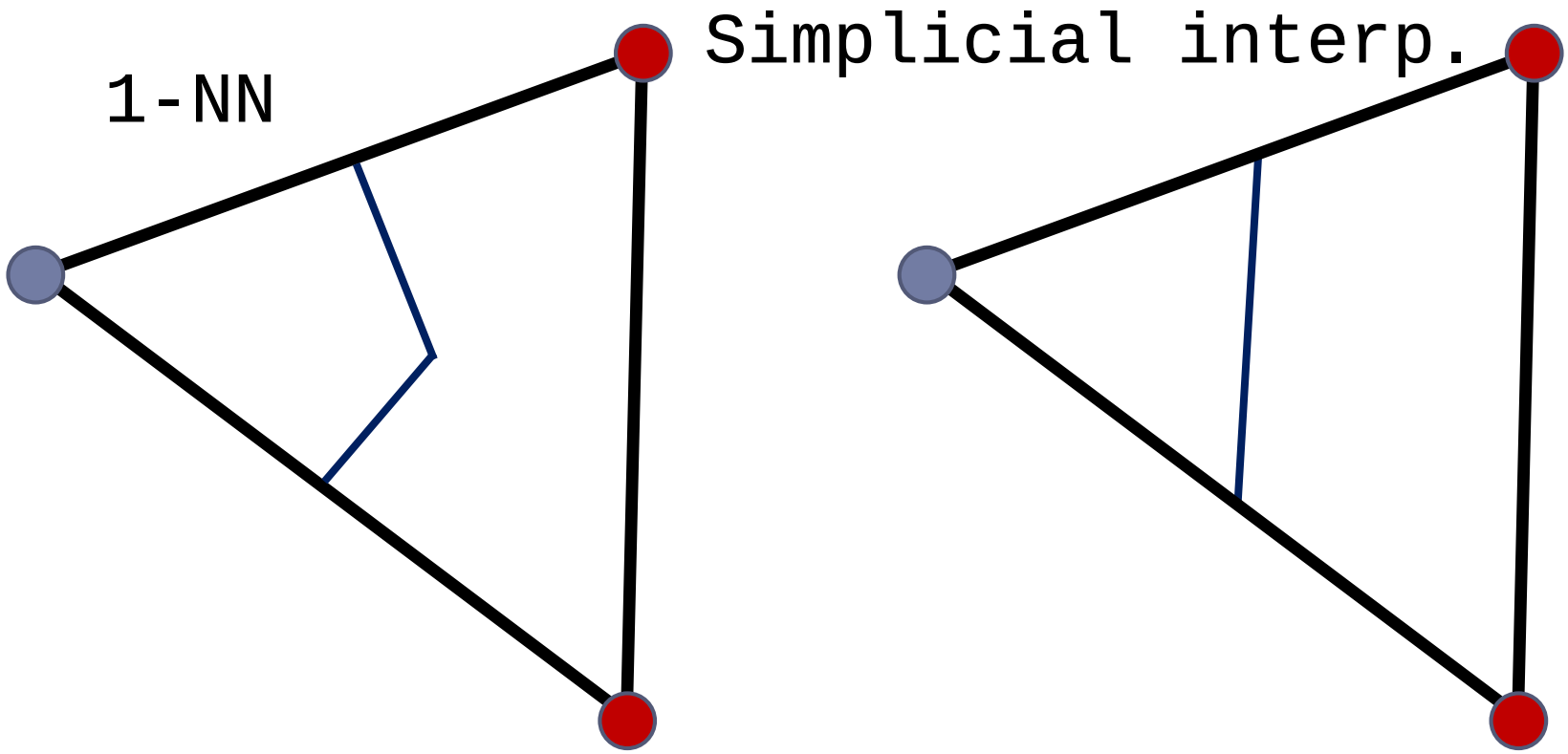
Simplicial interpolation



1. Triangulate.
2. Linearly interpolate
3. Threshold

[B., Hsu, Mitra, 18]

1-NN vs simplicial interpolation



Nearly Bayes optimal

Theorem: (dimension d)

$$E\big(L(SI)\big) < \left(1 + \frac{1}{2^d}\right) \times Bayes\ Risk$$

Classical bound:

$$E\left(L\left(1_{NN}\right)\right) < 2 \times Bayes\ Risk$$

The blessing of dimensionality!

[B., Hsu, Mitra, 18]

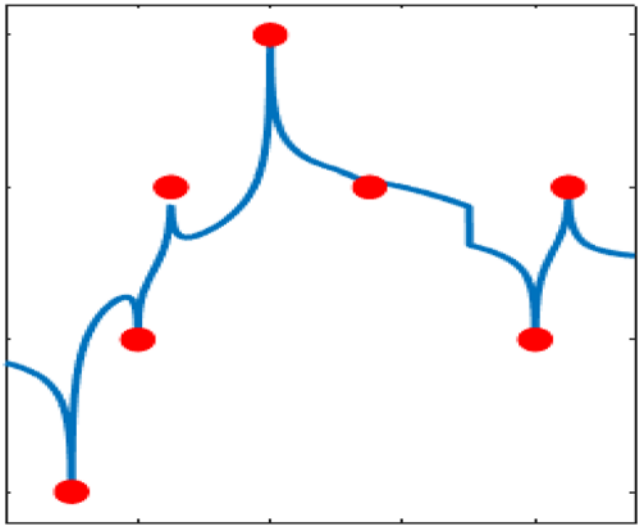
Interpolated k-NN schemes

$$y(x) = \text{sign}\left(\sum w(x, x_i) y_i\right)$$

Singular kernel, e.g. $w(x,z) = -\ln ||x - z||$

Theorem:

Weighted (interpolated) k-nn schemes with certain singular kernels are consistent.



Moreover, statistically optimal!

[B., Hsu, Mitra, 18] [B., Rakhlin, Tsybakov, 18]

This talk: the power of interpolation (optimization)

- Optimization under interpolation
  - Why is SGD so efficient in modern learning?
  - Exponential convergence of mini-batch SGD.

  [Ma, Bassily, **B.**, ICML 18]

- Application: Fast and simple kernel machines for large data.
  - Simplicity: no regularization or loss function
  - Learning kernels that adapt to GPU.
  - Automatic mini-batch/step size selection.

  EigenPro 2.0 [Ma, **B.**, NIPS 17, 18]

Stochastic Gradient Descent

$= (f_{\mathrm{w}}(\mathrm{x_i}) - \mathrm{y_i})^2$, e.g.

$$\mathrm{w}^* = \underset{\mathrm{w}}{\operatorname{argmin}}\, L(w) = \underset{\mathrm{w}}{\operatorname{argmin}} \frac{1}{n} \sum L_i(w)$$

SGD Idea: optimize $\sum L_i(w)$, m at a time.

Error after t steps SGD: $1/\mathrm{t}$

GD: e^{-t}

Why use SGD?

All major neural network architectures use SGD.

The Power of Interpolation

Key observation:

Interpolation $f_{w^*}(x_i) = y_i \;\Rightarrow\; \forall_i\, L_i(w^*) = 0$

Implies exponential convergence.

$(w \cdot 1 - 1)^2 + (\mathrm{w} \cdot 1 + 1)^2$ [non-interpolation]

[SGD oscillates, adaptive step size/slow convergence]

$(w_1 \cdot 1 - 1)^2 + (\mathrm{w}_2 \cdot 1 + 1)^2$ [interpolation]

[fixed step size, exponential convergence]

Exponential convergence of m-SGD

Convex loss function L (λ-smooth, α-strongly convex), L_i(β-smooth).

Theorem 1 [exponential convergence of m-SGD]

$$E\, L(w_{t+1}) \leq \frac{\lambda}{2}(1-\eta^*(m)\alpha)^t\, ||w_1 - w^*||$$

$$\eta^*(m) = \frac{m}{\beta + \lambda(m-1)}$$

[Ma, Bassily, **B**., ICML 18]

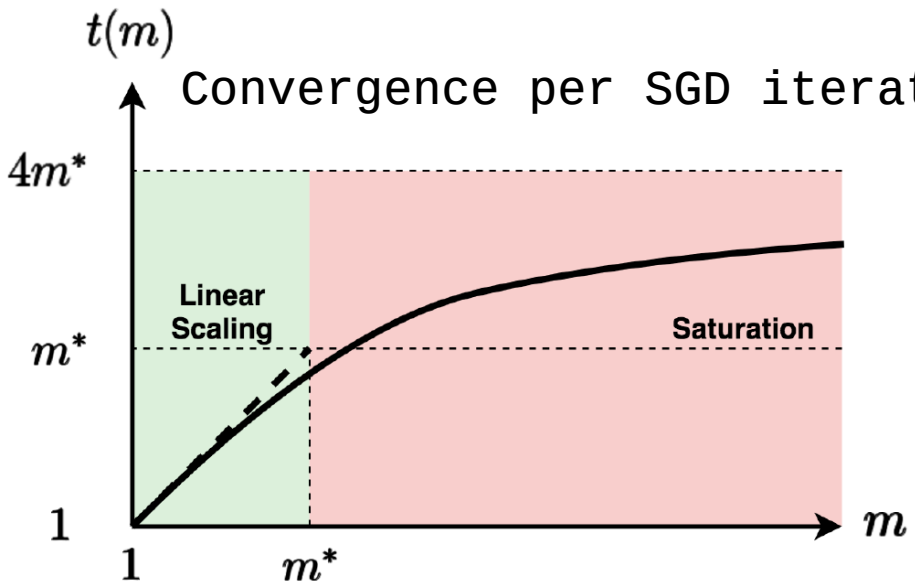
Related work ($m = 1$): [Strohmer, Vershynin 09] [Moulines, Bach, 11] [Needell, Srebro, Ward, 14]

Minibatch size?

Theorem 2: Critical size $m^* = \frac{\beta}{\lambda}$ [optimal fixed step size]

[1. linear scaling: $m \leq m^*$] One step m-SGD $\approx$ m steps of 1-SGD

[2. saturation: $m \geq m^*$] One step m-SGD $\sim$ one step of full GD



$m^* = \frac{\beta}{\lambda}$ (nearly) data independent. Quadratic loss function:

$$m^* \sim \frac{tr\ H}{\lambda_1(H)}$$

$O(n)$ computational gain over GD

Why minibatch?

GPU: fast highly parallel matrix x matrix products

→ limits algorithms available

Nvidia Titan X GPU (from nvidia.com)

Algorithmic requirements:
matrix x matrix products +
limited amount of other computation (CPU)

Full parallel minibatch computation $1 \ll m \ll n$
(much larger than 1-SGD, much smaller than full GD)

Interpolation in modern ML

How do you interpolate?

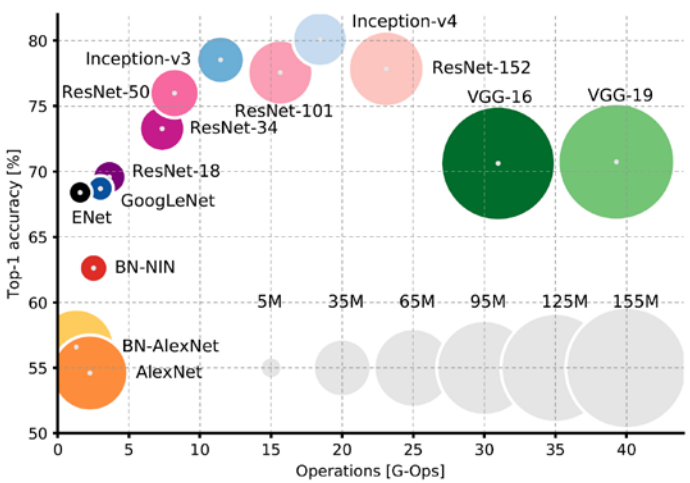
Think about a $m \times n$ linear system: $Ax = b$.

Need rank $\operatorname{rank} A \geq n$ (at least as many parameters as equations).

Over-parametrization # parameters $\geq$ # training data.

More parameters → easier to interpolate (even non-convex).

From Canziani, et al., 2017.



Ruslan Salakhutdinov's tutorial on deep learning (Simons Institute, Berkeley, 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.

SGD in modern ML

Systematic over-parametrization.

parameters >> # training data

Over-parametrization → interpolation → fast SGD:

SGD $O(n)$ computational gain ($n \sim 10^5$) over GD

+ GPU implementation ~ 100 over CPU.

We used to do that..

Combined: SGD on GPU $\sim 10^7$ faster than GD on CPU!

This talk: the power of interpolation

- Optimization under interpolation
  - Why is SGD so efficient in modern learning?
  - Exponential convergence of mini-batch SGD.

  [Ma, Bassily, **B**., ICML 18]

- Application: Fast and simple kernel machines for large data.
  - Simplicity: no regularization or loss function
  - Learning kernels that adapt to GPU.
  - Automatic mini-batch/step size selection.

  EigenPro 2.0 [Ma, **B**., NIPS 17, 18]

Kernel machines

$\mathcal{H}$ (Reproducing Kernel Hilbert space),

p.d. kernel $K(x,z)$, (e.g., $K(x,y) = e^{-\frac{\|x-y\|^2}{\sigma^2}}$)

$$f^* = argmin_{f \in \mathcal{H}, f(x_i)=y_i} \ \|f\|_{\mathcal{H}}$$

Representer Theorem ($K(x,y) = e^{-\frac{\|x-y\|^2}{\sigma^2}}$, for example)

$$f^*(x) = \sum_i \alpha_i K(x_i, x), \qquad \alpha = K^{-1} y$$

Minimum norm interpolation.

Kernel learning

Beautiful classical statistical/mathematical theory

RKHS Theory [Aronszajn,…, 50s]
Potential functions method [Izerman,…, 60s]
Splines [Parzen, Wahba,…, 1970-80s]
Kernel machines [Vapnik,…, 90s]

Very attractive setting:

- Convex
- Analytically tractable

Can be viewed as a 2-layer neural net.

Kernel Interpolation

No loss functions -- no regularization:

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

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

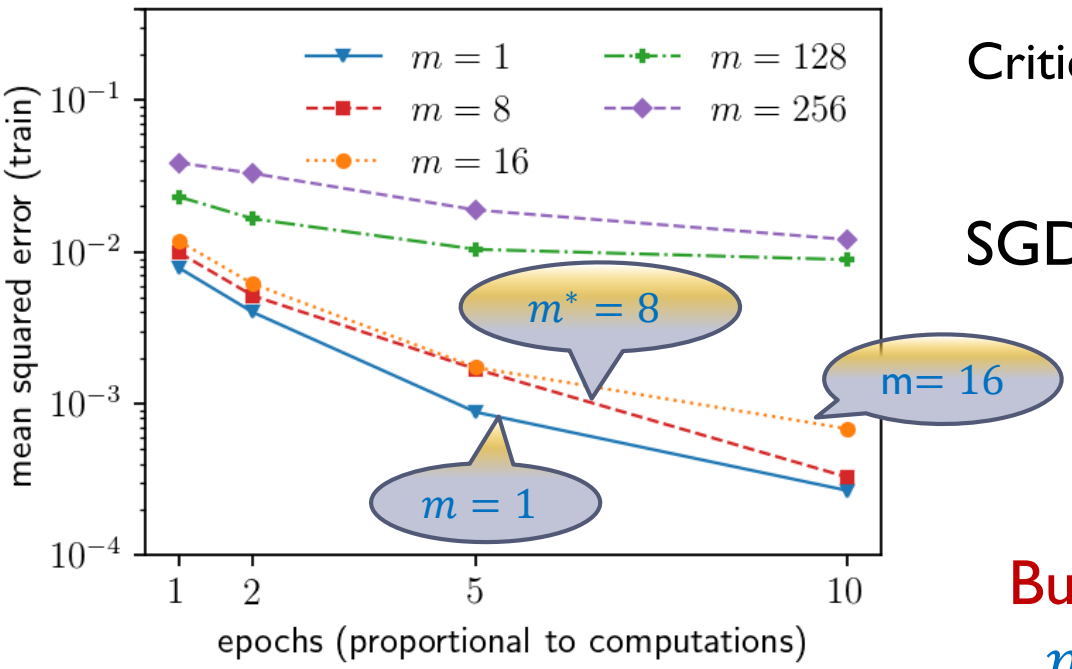
Gradient descent: $\alpha^{(t)} = \alpha^{(t-1)} - \eta\left(K\alpha^{(t-1)} - y\right)$

[Richarson, Landweber,…]

Cost n^2 per iteration. GPU compatible.

How much gain from SGD?

Real data example (Gaussian kernel)



(a) MNIST (Gaussian, $\sigma = 5$)
$\beta = 1, \lambda_1 = 0.15, m^* \approx 8$

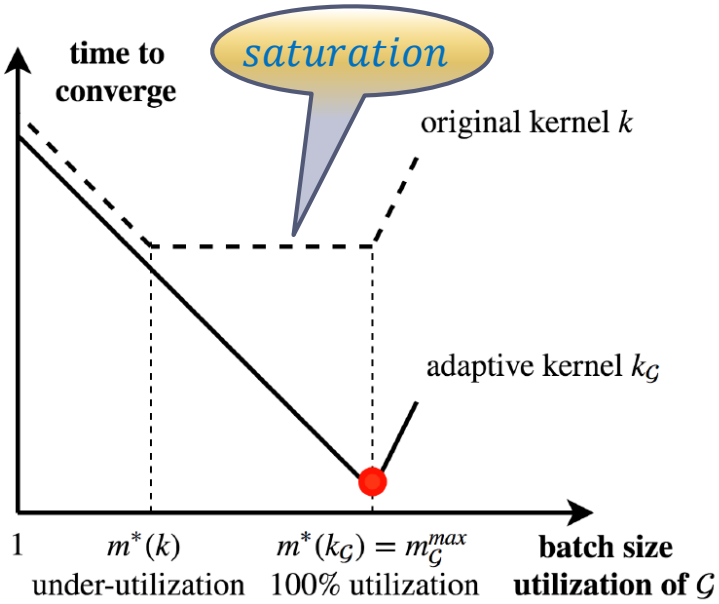
Critical size $m^* \approx \frac{\beta(K)}{\lambda_1(K)} + 1 \approx 8$.

SGD acceleration factor over GD ~10^4 (sequential computation)

But: parallelization beyond $m^* = 8$ has little effect.

Controlling parallelism

Problem: Parallelization is controlled by $\frac{1}{\lambda_1}$. (cf. convergence of gradient descent)



Idea: Make λ_1 smaller to fully utilize parallel resource (GPU) without changing the original solution.

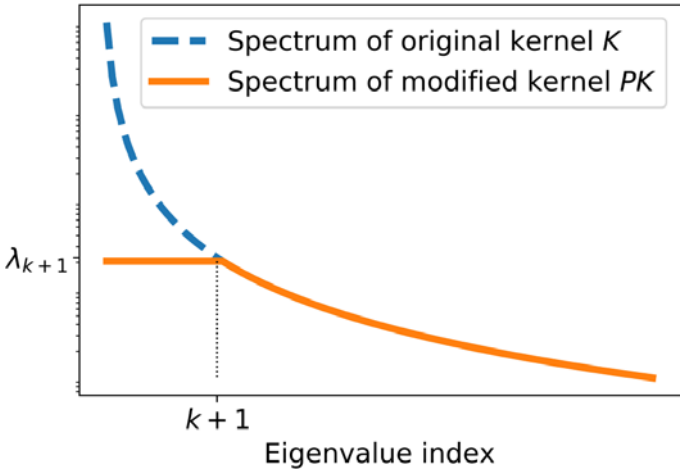
Eigenvalue control

Idea: Change λ_1 to fully utilize parallel resource (GPU).

We have a tool: EigenPro kernel.

Original kernel:

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



EigenPro kernel:

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

[Ma, B. NIPS 2017]

Comparisons to state-of-the-art

Dataset	Size	Our Method (use 1 GTX Titan Xp)		Results of Other Methods		
		error	GPU time	resource time	error	reference
MNIST	$1 \cdot 10^6$ / 6.7×10^6	0.67%	**21 m**	4.8 h on 1 GTX Titan X	0.70%	EigenPro [MB17]
				1.1 h on 1344 AWS vCPUs	0.72%	PCG [ACW16]
				less than 37.5 hours on 1 Tesla K20m	0.85%	[LML+14]
ImageNet†	1.3×10^6	20.6%	**40 m**	4 h on 1 Tesla K40c	20.7%	FALKON [RCR17]
				-	19.9%	Inception-ResNet-v2 [SIVA17]
TIMIT	$1.1 \cdot 10^6$ / $2 \cdot 10^6$	31.6%	34 m (4 epochs)	3.2 h on 1 GTX Titan X	31.7%	EigenPro [MB17]
				1.5 h on 1 Tesla K40c	32.3%	FALKON [RCR17]
				512 IBM Blue Gene/Q cores	33.5%	Ensemble [HAS+14]
		32.0%	**17 m** (2 epochs)	7.5 h on 1024 AWS vCPUs	33.5%	BCD [TRVR16]
				multiple AWS g2.2xlarge instances	32.4%	DNN [MGL+17]
				multiple AWS g2.2xlarge instances	30.9%	SparseKernel [MGL+17] (use learned features)
SUSY	$4 \cdot 10^6$	19.7%	**48 s**	6 m on 1 GTX Titan X	19.8%	EigenPro [MB17]
				4 m on 1 Tesla K40c	19.6%	FALKON [RCR17]
				36 m on IBM POWER8	$\approx 20\%$	Hierarchical [CAS16]

Interactive ML

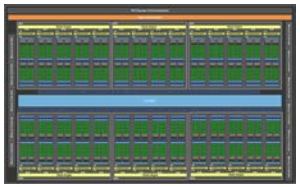
Dataset	Size	Our method	LibSVM
TIMIT	$1 \cdot 10^5$	**15 s**	1.6 h
SVHN	$7 \cdot 10^4$	**13 s**	3.8 h
MNIST	$6 \cdot 10^4$	**6 s**	9 m
CIFAR-10	$5 \cdot 10^4$	**8 s**	3.4 h

Smaller datasets take seconds.
No optimization parameters to select.

Take away messages

The Power of Interpolation:

1. Explains fast SGD in modern ML.
2. Allows for much simpler analysis.
3. Leads to very efficient kernel machines, adaptive to modern hardware.



Deep learning:

overparametrization
interpolation → generalizes well!
fast SGD
GPU

+ convolutional structures

The challenge of interpolation

Why do interpolated classifiers generalize?

Ubiquitous: Deep Neural Networks, Kernel Machines, Random Forests, Adaboost

Generalization is probably not (primarily) determined by:

- Non-convexity
- Regularization
- Loss functions
- Deep architectures
- Specific properties of optimization algorithms

Inductive bias is clearly important.

Time to revisit high-dimensional statistics!