Applying Statistical Learning Theory to Deep Learning:

What we understand, what we need understand, and what we need to re-think

Nati Srebro (TTIC)

Plan

Today:

- Supervised Learning, Free Lunch and Inductive Bias
- What ~~is~~ do I mean by "Inductive Bias"?
- Understanding Deep Learning from a Learning Theory perspective:
  What we do and don't understand

Next Four Lectures:

- Tutorial on (Stochastic) Optimization and Learning
  —mostly convex
- Implicit Bias of Optimization in Deep Learning
- Benign Overfitting and Interpolation Learning

- **Supervised Learning:** find $h: \mathcal{X} \to \mathcal{Y}$ with small *generalization error*

$$L(h) = \mathbb{E}_{(x,y)\sim\mathcal{D}}[loss(h(x); y)]$$

based on samples S (hopefully $S \sim \mathcal{D}^m$) using learning rule:

$$A: S \mapsto h \quad \text{(i.e. } A: (\mathcal{X} \times \mathcal{Y})^* \to \mathcal{Y}^{\mathcal{X}} \text{)}$$

- **No Free Lunch:** For any learning rule, there exists a source $\mathcal{D}$ (i.e. reality), for which the learning rule yields expected error ½
- More formally for any A, m there exists $\mathcal{D}$ s.t. $\exists_{h^*} L(h^*) = 0$ but

$$\mathbb{E}_{S\sim\mathcal{D}^m}\left[L\left(A(S)\right)\right] \geq \frac{1}{2} - \frac{m}{2|\mathcal{X}|}$$

- **Inductive Bias:**
  - Some realities (sources $\mathcal{D}$) are less likely; design A to work well on more likely realities
    e.g., by preferring certain $y|x$ (i.e. $h(x)$) over others
  - Assumption or property of reality $\mathcal{D}$ under which A ensures good generalization error
    e.g., $\exists h \in \mathcal{H}$ with low $L(h)$
    e.g., $\exists h$ with low "complexity" $c(h)$ and low $L(h)$

Flat Inductive Bias

- **"Flat" inductive bias**: $\exists h^* \in \mathcal{H}$ with low $L(h^*)$
- (Almost) optimal learning rule:

$$ERM_{\mathcal{H}}(S) = \hat{h} = \arg\min_{h \in \mathcal{H}} L_S(h)$$

- Guarantee (in expectation over $S \sim \mathcal{D}^m$):

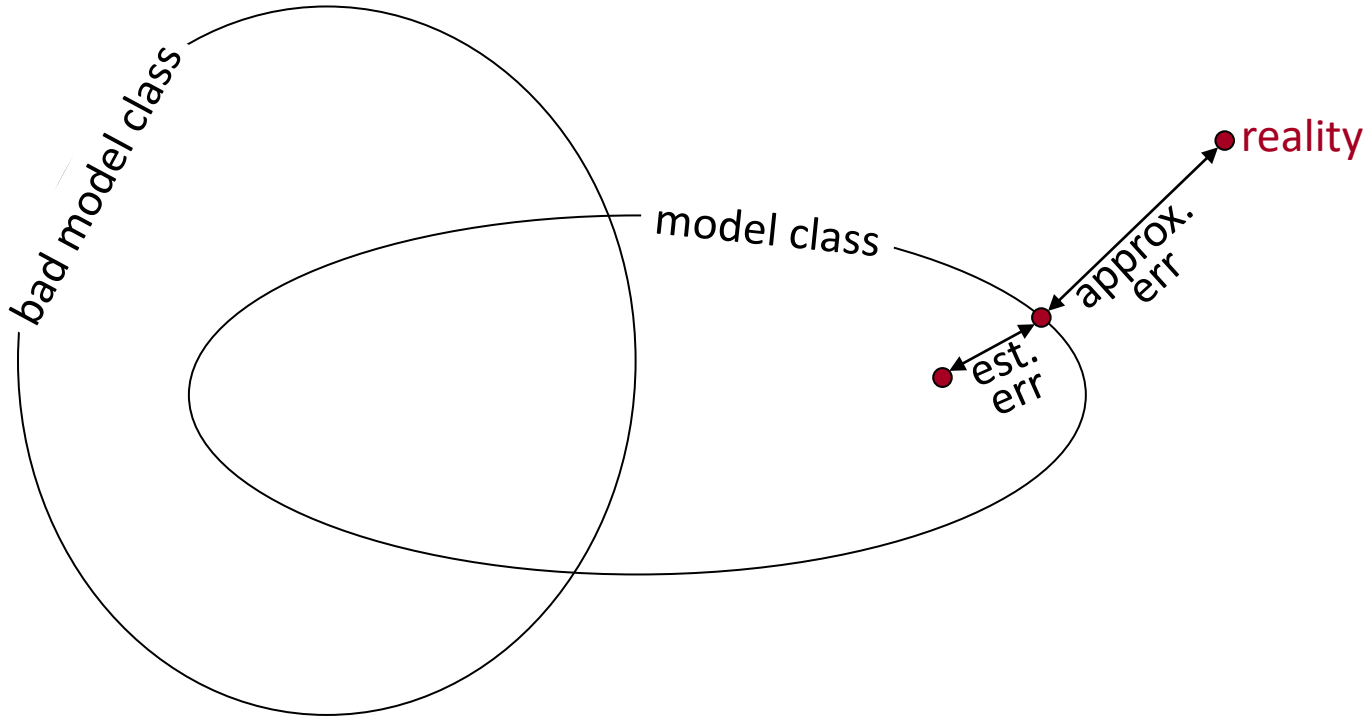
$$L\left(ERM_{\mathcal{H}}(S)\right) \leq L(h^*) + \mathcal{R}_m(\mathcal{H}) \approx L(h^*) + \sqrt{\frac{O\left(capacity(\mathcal{H})\right)}{m}}$$

➔ can learn with $O(capacity(\mathcal{H}))$

- E.g.
  - For binary classification, $capacity(\mathcal{H}) = VCdim(\mathcal{H})$

  Vapnik-Chrvonenkis (VC) dimension: largest number of points D that can be labeled (by some $h \in \mathcal{H}$) in every possible way (i.e. for which the inductive bias is uninformative)
  - For linear classifiers over d features, $VCdim(\mathcal{H}) = d$
  - Usually with d parameters, $VCdim(\mathcal{H}) \approx \tilde{O}(\#params)$
  - Always: $VCdim(\mathcal{H}) \leq \log|\mathcal{H}| \leq \#bits = \#params \cdot \#bits/param$
  - For linear predictors with $\|w\|_2 \leq B$, with logistic loss and normalized data: $capacity(\mathcal{H}) = B^2$

Machine Learning



- We want model classes (hypothesis classes) that:
  - Are expressive enough to capture reality well
  - Have small enough capacity to allow generalization

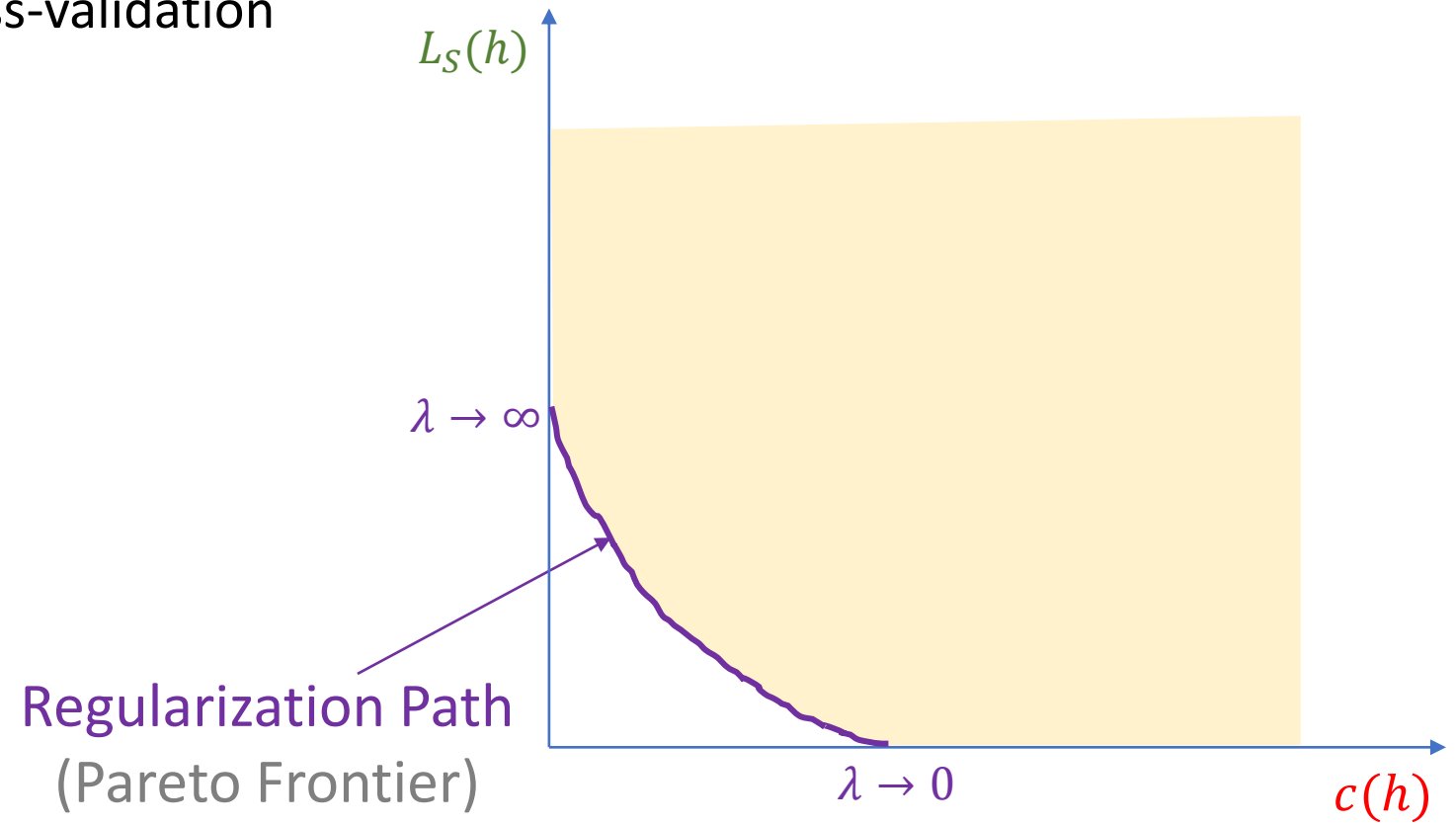
Complexity Measure as Inductive Bias

- **Complexity measure**: mapping $c: \mathcal{Y}^{\mathcal{X}} \to [0, \infty]$
- Associated inductive bias: $\exists h^*$ with small $c(h^*)$ and small $L(h^*)$
- Learning rule: $SRM_{\mathcal{H}}(S) = \arg\min \; L(h) \;,\; c(h)$

  e.g. $\arg\min L(h) + \boldsymbol{\lambda}\, c(h)$ or $\arg\min L(h)$ s.t. $c(h) \leq \boldsymbol{B}$

  and choose λ or B using cross-validation

- E.g.:
  - Degree of poly
  - Sparsity
  - $\|w\|$



Complexity Measure as Inductive Bias

- **Complexity measure**: mapping $c: \mathcal{Y}^{\mathcal{X}} \to [0, \infty]$
- Associated inductive bias: $\exists h^*$ with small $c(h^*)$ and small $L(h^*)$
- Learning rule: $SRM_{\mathcal{H}}(S) = \arg\min \; L(h) \;,\; c(h)$

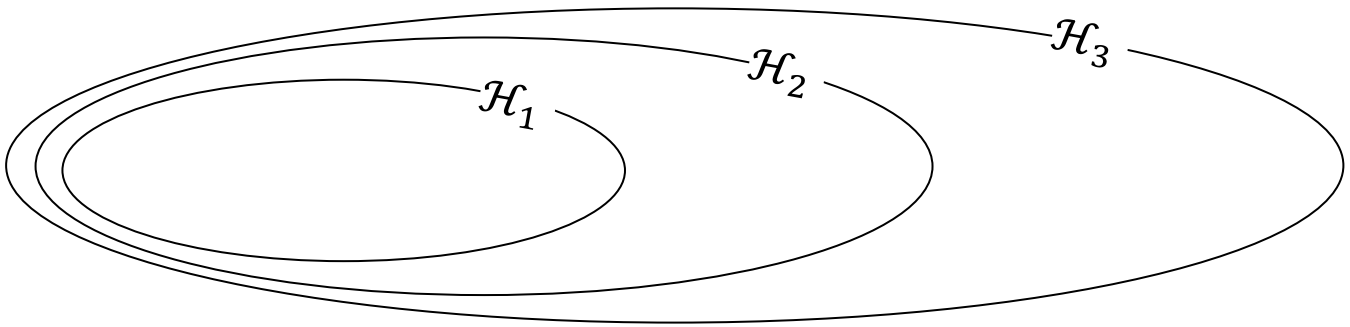
  e.g. $\arg\min L(h) + \lambda\, c(h)$ or $\arg\min L(h)$ s.t. $c(h) \leq B$

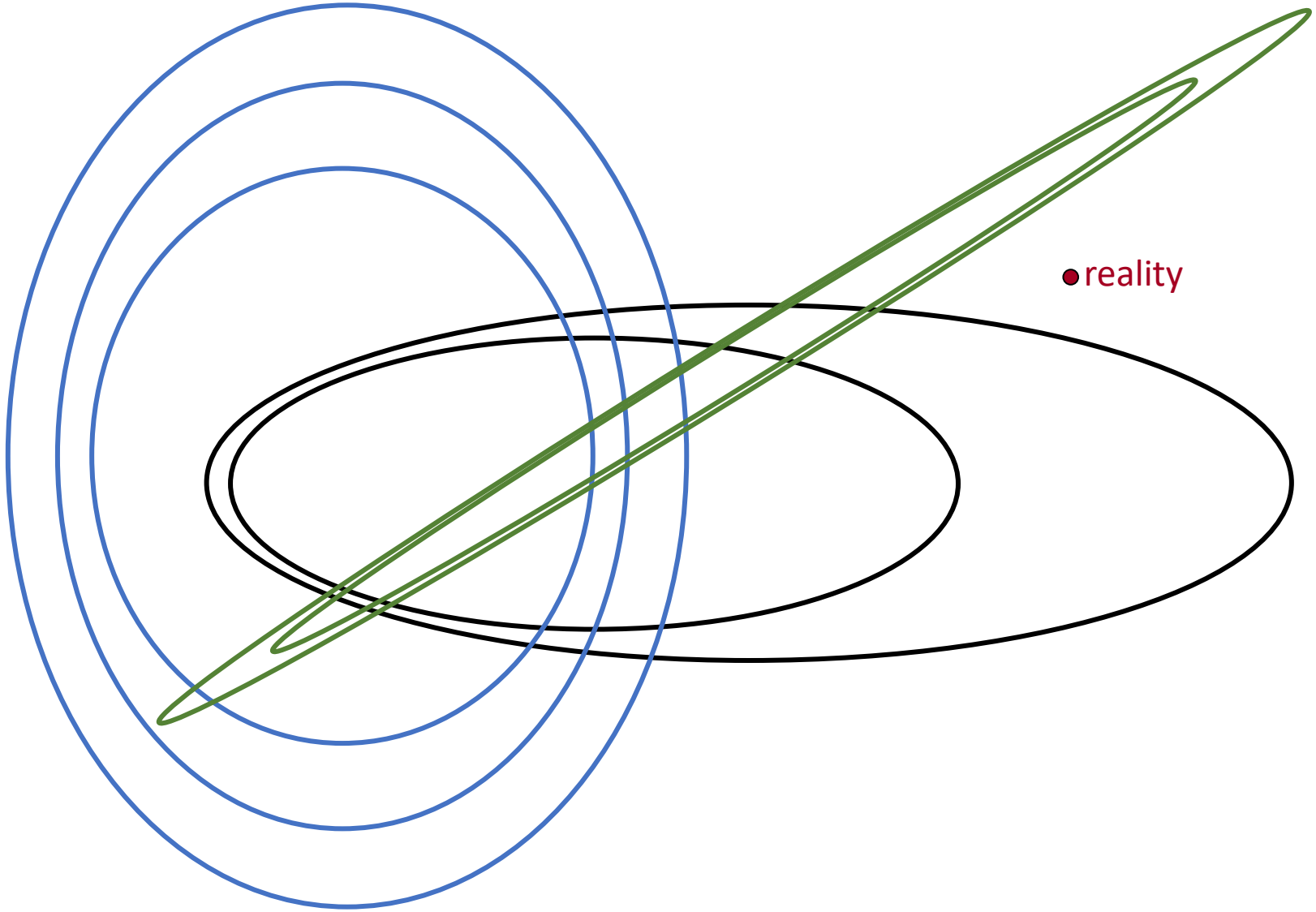
  and choose λ or B using cross-validation
- Guarantee:

$$L\left(SRM_{\mathcal{H}}(S)\right) \leq\approx L(h^*) + \sqrt{\frac{capacity\left(\mathcal{H}_{c(h^*)}\right)}{m}}$$

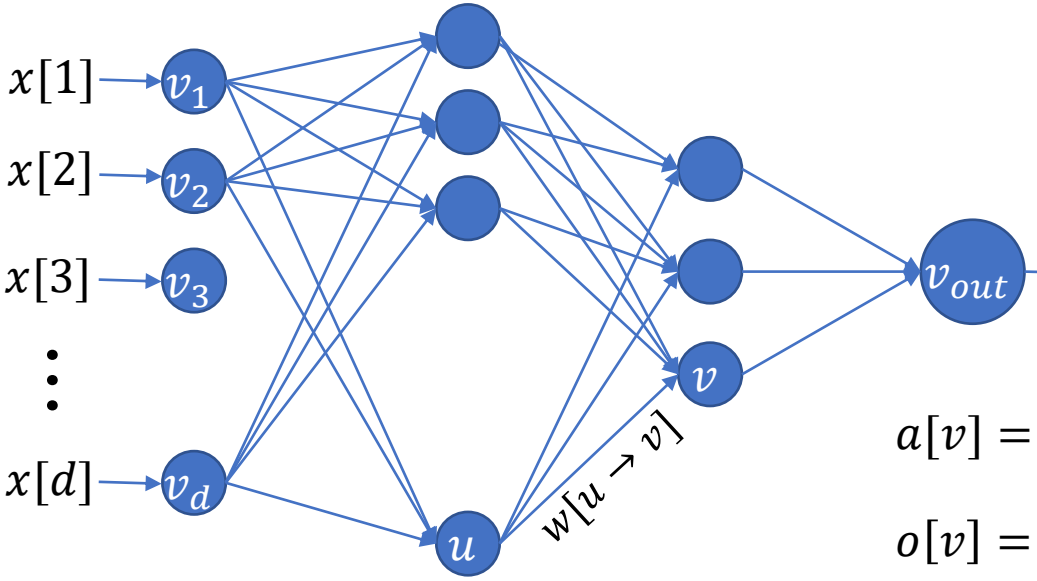
$\mathcal{H}_B = \{h | c(h) \leq B\}$

- E.g.:
  - Degree of poly
  - Sparsity
  - $\|w\|$




reality

Feed-Forward Neural Networks (The Multilayer Perceptron)



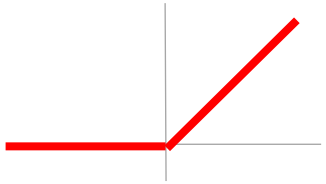
$h_{G(V,E),\sigma,w}(x)$

$$a[v] = \sum_{u \to v \in E} w[u \to v]\, o[u]$$

$$o[v] = \sigma(\, a[v]\,)$$

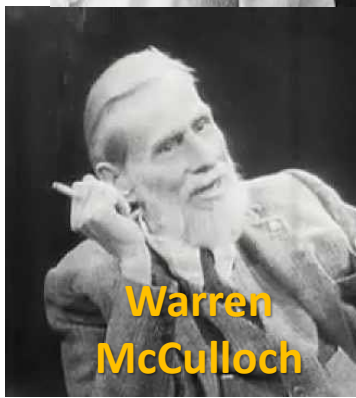
Architecture:

- Directed Acyclic Graph G(V,E). Units (neurons) indexed by vertices in V.
  - "Input Units" $v_1 \ldots v_d \in V$, with no incoming edges and $o[v_i] = x[i]$
  - "Output Unit" $v_{out} \in V$, $h_w(x) = o[v_{out}]$
- "Activation Function" σ: $\mathbb{R} \to \mathbb{R}$. E.g. $\sigma_{RELU}(z) = [z]_+$



Parameters:

- Weight $w[u \to v]$ for each edge $u \to v \in E$



Feed Forward Neural Networks

- Fix architecture (connection graph $G(V, E)$, transfer σ)

$$\mathcal{H}_{G(V,E),\sigma} = \{ f_w(x) = output\ of\ net\ with\ weights\ w \}$$

- Capacity / Generalization ability / Sample Complexity

- Expressive Power / Approximation

Capacity (Sample Complexity) of NN

- #params = $|E|$ (number of weights we need to learn)
- More formally: $VCdim\left(\mathcal{H}_{G(V,E),sign}\right) = O(|E|\log|E|)$
- Other activation functions?
  - $VCdim\left(\mathcal{H}_{G(V,E),\sin}\right) = \infty$ even with single unit and single real-valued input
  - For $\sigma(z) = sigmoid(z) = \frac{1}{1+e^{-z}}$:

$$\Omega(|E|^2) \leq VCdim(\mathcal{H}_{G(V,E),\text{sigmoid}}) \leq O(|E|^4)$$

  - For piecewise linear, e.g. $ramp(z) = clip_{[-1,1]}(z)$ or $ReLU(z) = \max(0,z)$:

$$\Omega\left(|E|L\log {}^{|E|}/_{L}\right) \leq VCdim(\mathcal{H}_{G,\sigma}) \leq O(|E|L\log|E|)$$

L=depth

- With integer weights $\in [-B,..,B]$:

$$VCdim\left(\mathcal{H}_{G(V,E),\sigma}\right) \leq \log\left|\mathcal{H}_{G(V,E),\sigma}\right| \leq 2|E|\log B$$

Feed Forward Neural Networks

- Fix architecture (connection graph $G(V, E)$, transfer σ)

$$\mathcal{H}_{G(V,E),\sigma} = \{\, f_w(x) = output\ of\ net\ with\ weights\ w \,\}$$

- Capacity / Generalization ability / Sample Complexity
  - $\widetilde{\boldsymbol{O}}(|\boldsymbol{E}|)$ (number of edges, i.e. number of weights)
    (with threshold σ, or with RELU and finite precision; RELU with inf precision: $\widetilde{\Theta}(|E| \cdot depth)$)

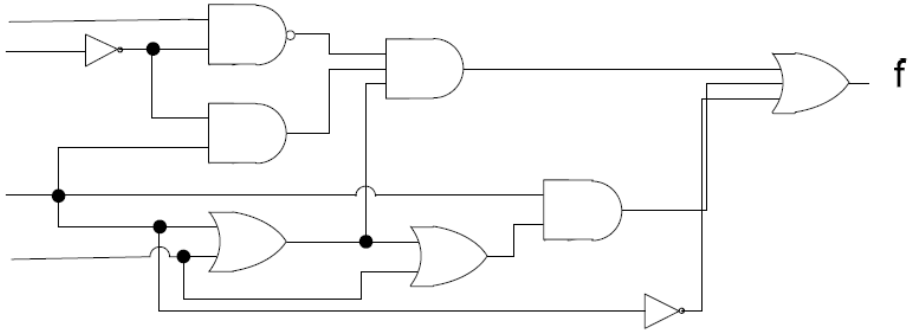
- Expressive Power / Approximation

What can Feed-Forward Networks Represent?

- ANDs (using a single unit)
- ORs (using a single unit)
- XORs (parities) (using $|E| = d^2$ with depth 2, or $|E| = O(d)$ with depth $\log(d)$)
- NOT (using a single weight)

- Any function over $\mathcal{X} = \{\pm 1\}^d$

Learning Circuits as Neural Networks



$CIRCUIT_n[depth, size]$ = functions $f: \{\pm 1\}^n \to \{0,1\}$ that can be implemented with logical circuits with at most $size$ unlimited-fan-in AND, OR and NOT gates, and longest path from input to output at most $depth$

$(AC^i \approx CIRCUIT[O(\log^i n), poly(n)])$

Learning a circuit (ie learning with the class $CIRCUIT$): learning the *architecture*

Claim: $CIRCUIT_n[depth, size] \subseteq \mathcal{H}_{G_{n,L=depth,k=size}, sign}$

Fully connected layer graph, with L(=*depth*) layers and k(=size) nodes in each layers.

- Weights are ± 1 if connected in the circuit (with/without a NOT gate in between), 0 otherwise;
- Bias terms are *fanin-1* for AND, *1-fanin* for OR

What can Feed-Forward Networks Represent?

- Any function over $\mathcal{X} = \{\pm 1\}^d$
  - As a circuit
  - E.g. using DNF (OR of ANDS), with a single hidden layer of ANDs, output output unit implementing OR
  - $|V| = 2^d, |E| = d2^d$
  - Like representing the truth table directly…

- Universal Representation Theorem: Any continuous functions $f: [0,1]^d \rightarrow \mathbb{R}$ can be approximated to to within ϵ (for any ϵ) by a feed-forward network with sigmoidal (or almost any other) activation and a single hidden layer.
  - Size of layer exponential in d

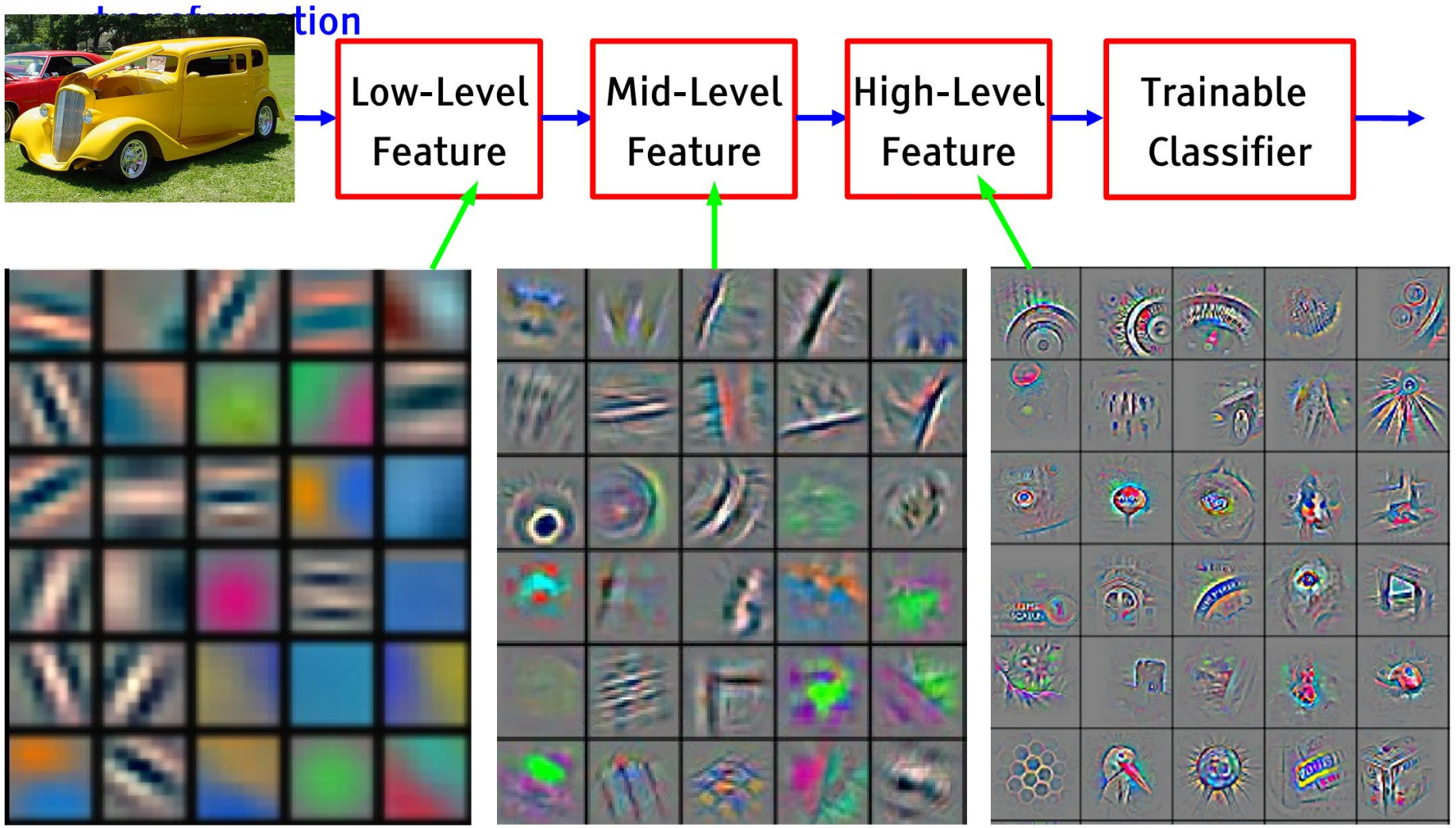
What can SMALL Networks Represent?

- Intersection of halfspaces
  - Using single hidden layer (the halfspaces; output unit does AND)

- Union of intersection of halfspaces
  - Using two hidden layers (halfspaces→OR→AND)

- Feature learning:
  Linear predictors over (small number of) features,
  in turn represented as linear predictors over more basic features,
  that in turn are also represented as linear predictors

Multi-Layer Feature Learning



Feed Forward Neural Networks

- Fix architecture (connection graph $G(V,E)$, transfer σ)

$$\mathcal{H}_{G(V,E),\sigma} = \{\, f_w(x) = output\ of\ net\ with\ weights\ w \,\}$$

- Capacity / Generalization ability / Sample Complexity
  - $\widetilde{\boldsymbol{O}}(|\boldsymbol{E}|)$ (number of edges, i.e. number of weights)
    (with threshold σ, or with RELU and finite precision; RELU with inf precision: $\widetilde{\Theta}(|E| \cdot depth)$)
- Expressive Power / Approximation
  - Any continuous function with huge network
  - Lots of interesting things naturally with small networks
  - **Any time T computable function with network of size $\widetilde{\boldsymbol{O}}(\boldsymbol{T})$**

    Using a depth-T network, since anything computable in time T is also computable using a logical circuit of size $\tilde{O}(T)$

Free Lunches

- **ML as an Engineering Paradigm**: Use data and examples, instead of expert knowledge and tedious programming, to automatically create efficient systems that perform complex tasks
- We only care about $\{h|h \text{ is an efficient system}\}$
- **Free Lunch**: $\boldsymbol{TIME_T} = \{h|h \text{ comp. in time } T\}$ has capacity $O(T)$ and hence learnable with $O(T)$ samples, e.g. using ERM
- Even better: $\boldsymbol{PROG_T} = \{\text{program of length T}\}$ has capacity $O(T)$
- Problem: ERM for above is not computable!
- Modified ERM for $\boldsymbol{TIME_T}$ (truncating exec. time) is NP-complete
- P=NP ➔ **Universal Learning is possible! (Free Lunch)**
- Crypto is possible (one-way functions exist)
  ➔ No poly-time learning algorithm for $\boldsymbol{TIME_T}$
  (that is: no poly-time A and uses $poly(T)$ samples s.t. if $\exists h^* \in TIME_T$ with $L(h^*) = 0$ then $\mathbb{E}\left[L\left(A(S)\right)\right] \leq 0.4$)

No Free (Computational) Lunch

- **Statistical No-Free Lunch**: For any learning rule A, there exists a source $\mathcal{D}$ (i.e. reality), s.t. $\exists h^*$ with $L(h^*) = 0$ but $\mathbb{E}\left[L\left(A(S)\right)\right] \approx \frac{1}{2}$.
- **Cheating Free Lunch**: There exists A, s.t. for any reality $\mathcal{D}$ and any **efficiently computable $\boldsymbol{h}^*$**, A learns a predictor almost as good as h^* (with #samples=O(runtime of h^*), but *a lot* of time).
- **Computational No-Free Lunch**: For every **computationally efficient** learning ***algorithm*** $\boldsymbol{A}$, there is a reality $\mathcal{D}$ s.t. there is some comp. efficient (poly-time) h^* with $L(h^*) = 0$ but $\mathbb{E}\left[L\left(A(S)\right)\right] \approx \frac{1}{2}$.
- **Inductive Bias**: Assumption or property of reality $\mathcal{D}$ under which a learning ***algorithm*** A **runs efficiently** and ensures **good generalization error**.
- $\mathcal{H}$ or $c(h)$ are *not* sufficient inductive bias if ERM/SRM not efficiently implementable, or implementation doesn't always work (runs quickly and returns actual ERM/SRM).

Feed Forward Neural Networks

- Capacity / Generalization ability / Sample Complexity
  - $\widetilde{\boldsymbol{O}}(|\boldsymbol{E}|)$ (number of edges, i.e. number of weights)
    (with threshold σ, or with RELU and finite precision; RELU with inf precision: $\widetilde{\Theta}(|E| \cdot depth)$)

- Expressive Power / Approximation
  - Any continuous function with huge network
  - Lots of interesting things naturally with small networks
  - **Any time T computable function with network of size $\widetilde{\boldsymbol{O}}(\boldsymbol{T})$**

- Computation / Optimization
  - Non-convex
  - No known algorithm guaranteed to work
  - NP-hard to find weights even with 2 hidden units

  - Even if function exactly representable with single hidden layer with $\Theta(\log d)$ units, even with no noise, and even if we train a much larger network or use any other method when learning: no poly-time algorithm can ensure better-than-chance prediction
    [Kearns Valiant 94; Klivans Sherstov 06; Daniely Linial Shalev-Shwartz '14]

Choose your universal learner:

Short (or short runtime) Programs

- Universal
- Captures anything we want with reasonable sample complexity
- ERM is NP-hard
- Provably hard to learn even improperly, with any rule (subject to crypto)
- Hard to optimize in practice
  Short programs: Incomputable.
  Even if we limit to bounded-time:
  - No practical local search
  - Highly non-continuous, disconnected discrete space
  - Not much success

Deep Networks

- Universal
- Captures anything we want with reasonable sample complexity
- ERM is NP-hard
- Provably hard to learn even improperly, with any rule (subject to crypto)
- Often easy to optimize
  - **Continuous**
  - Amenable to local search with Grad Descent, or SGD
  - **Lots of empirical success**

Feed Forward Neural Networks

- Fix architecture (connection graph $G(V,E)$, transfer σ)

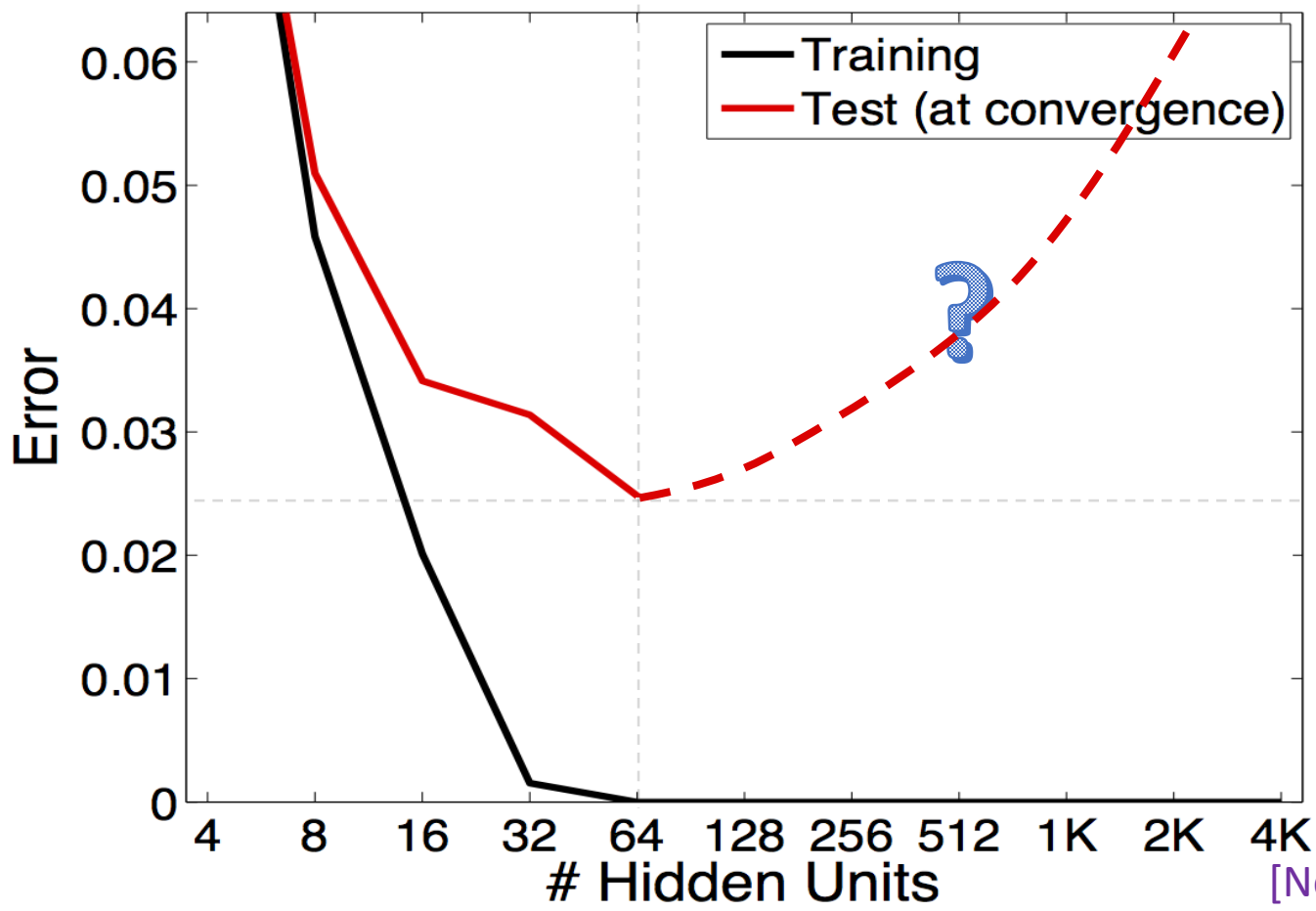
$$\mathcal{H}_{G(V,E),\sigma} = \{\, f_w(x) = output\ of\ net\ with\ weights\ w \,\}$$

- Capacity / Generalization ability / Sample Complexity
  - $\widetilde{O}(|E|)$ (number of edges, i.e. number of weights)
    (with threshold σ, or with RELU and finite precision; RELU with inf precision: $\widetilde{\Theta}(|E| \cdot depth)$)

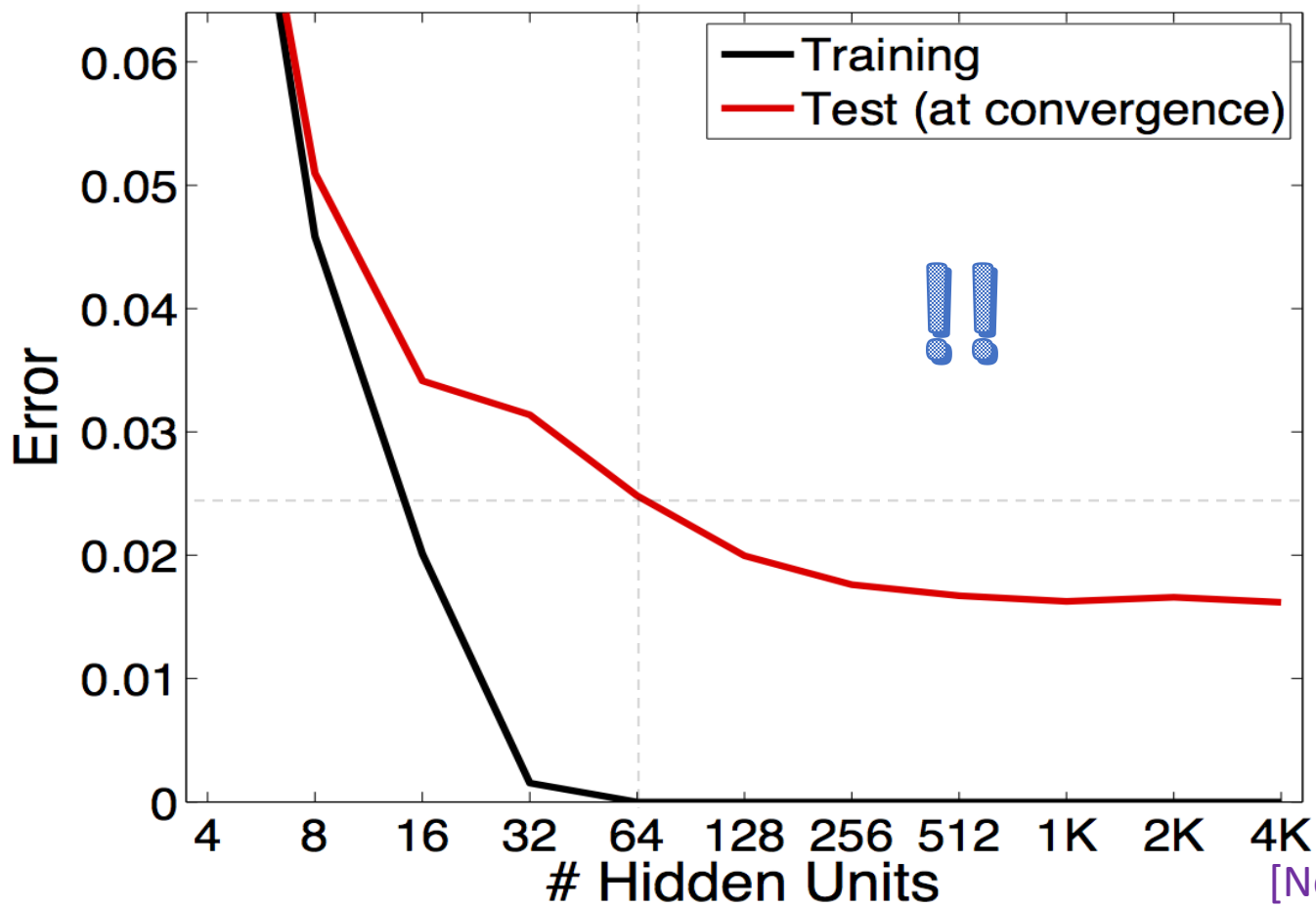
- Expressive Power / Approximation
  - Any continuous function with huge network
  - Lots of interesting things naturally with small networks
  - **Any time T computable function with network of size $\widetilde{O}(T)$**

- Computation / Optimization
  - Even if function exactly representable with single hidden layer with $\Theta(\log d)$ units, even with no noise, and even if we allow a much larger network when learning: no poly-time algorithm always works
    [Kearns Valiant 94; Klivans Sherstov 06; Daniely Linial Shalev-Shwartz '14]

  - Magic property of reality that makes local search "work"



[Neyshabur Tomioka S ICLR'15]



[Neyshabur Tomioka S ICLR'15]

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
(fitting random labels)		no	no	100.0	9.78
Inception w/o BatchNorm	1,649,402	no	yes	100.0	83.00
		no	no	100.0	82.00
(fitting random labels)		no	no	100.0	10.12
Alexnet	1,387,786	yes	yes	99.90	81.22
		yes	no	99.82	79.66
		no	yes	100.0	77.36
		no	no	100.0	76.07
(fitting random labels)		no	no	99.82	9.86
MLP 3x512	1,735,178	no	yes	100.0	53.35
		no	no	100.0	52.39
(fitting random labels)		no	no	100.0	10.48
MLP 1x512	1,209,866	no	yes	99.80	50.39
		no	no	100.0	50.51
(fitting random labels)		no	no	99.34	10.61

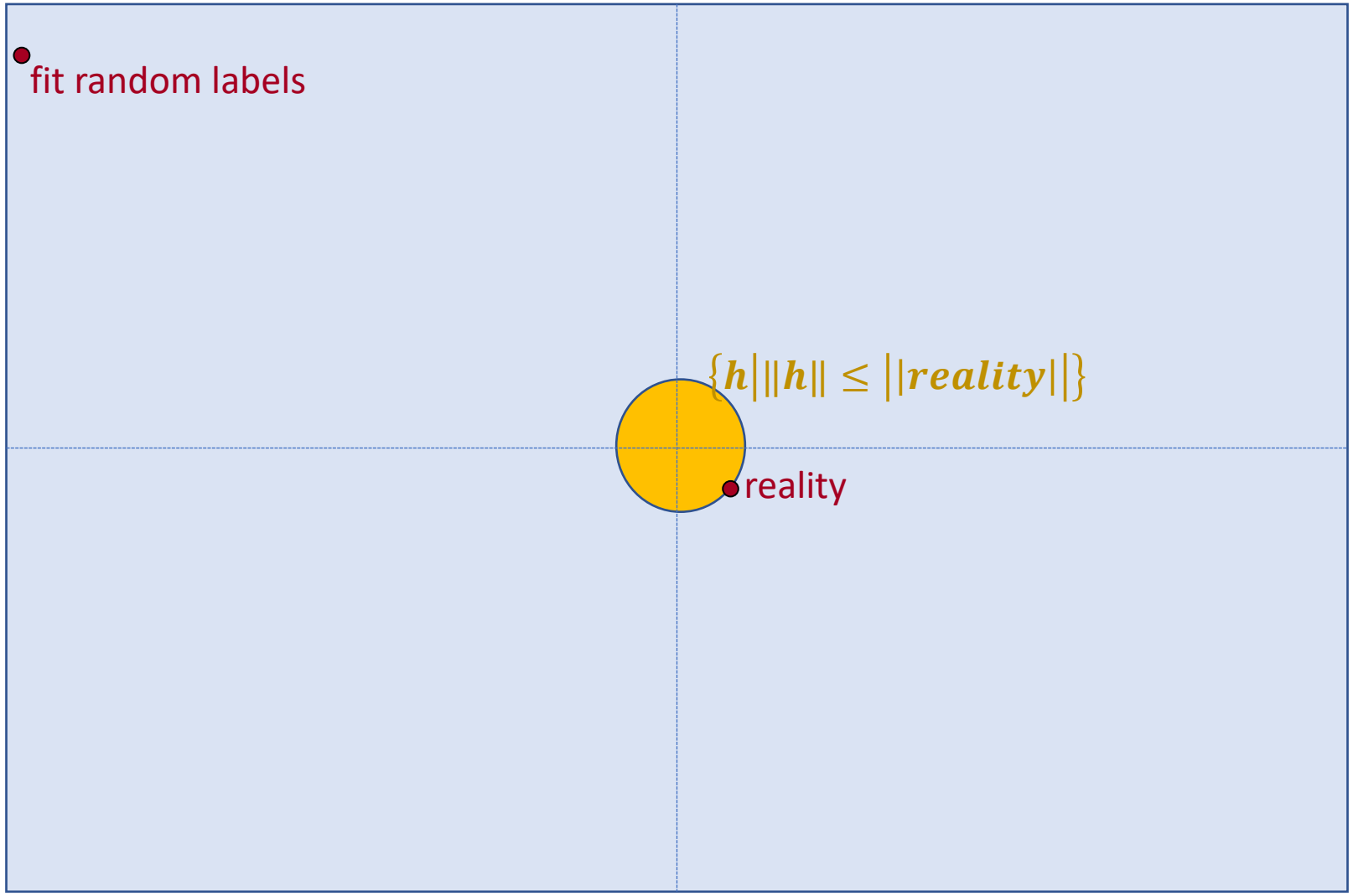
UNDERSTANDING DEEP LEARNING REQUIRES RE-THINKING GENERALIZATION

Zhang, Bengio, Hardt, Recht, Vinyals 2017

Learning with a Rich Function Class

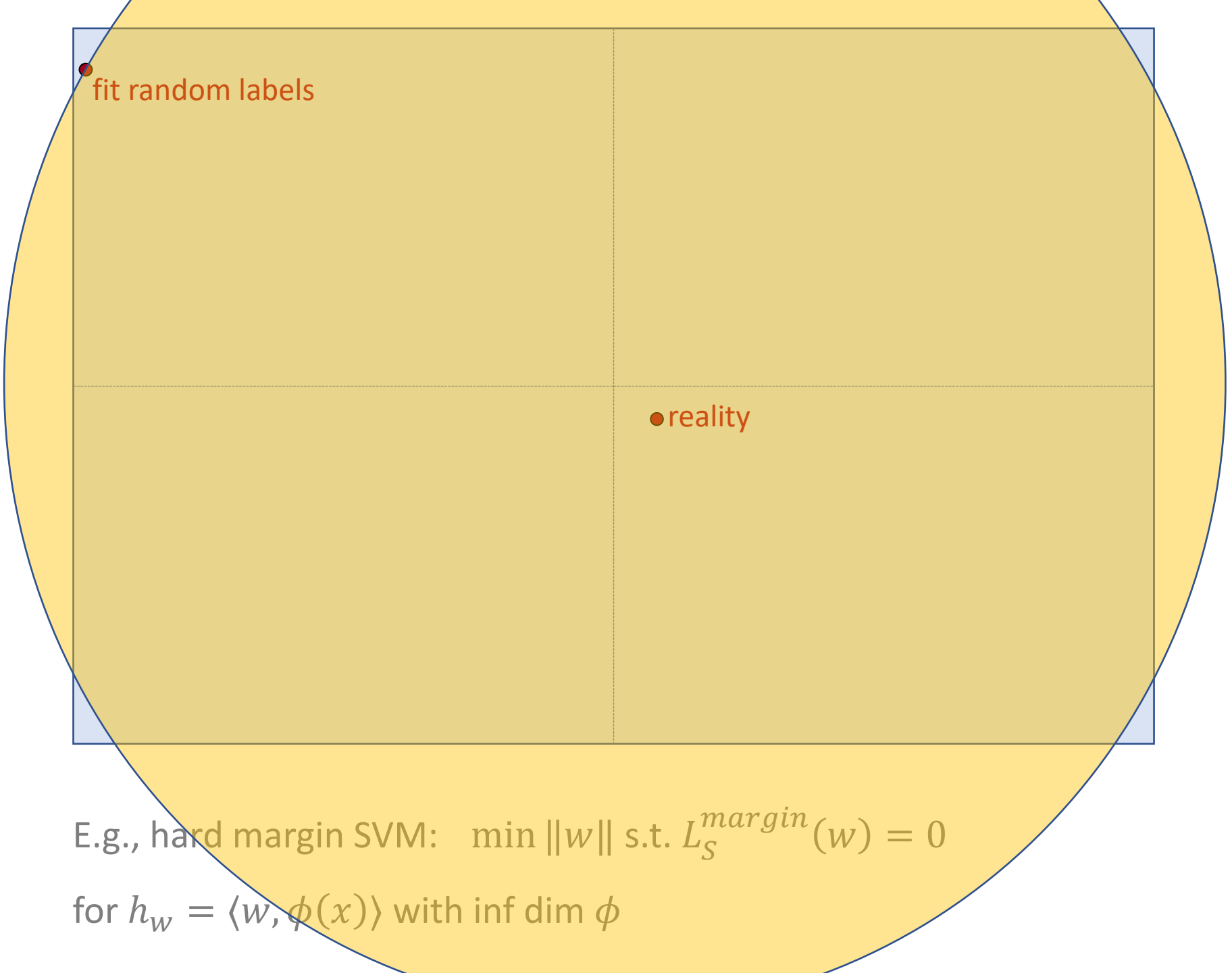
- Learning rule $A(S)$ s.t.
  - For any data set, even with random labels, can fit data: $L_S\big(A(S)\big) = 0$
  - For "real" data $S \sim \mathcal{D}^m$ sampled from a reasonable reality $\mathcal{D}$, we can generalize: $L\big(A(S)\big)$ is low
- Examples:
  - 1-Nearest Neighbor: if realizable by some continuous h^* (ie $L(h^*) = 0$), then consistent: $L(1NN(S)) \xrightarrow{|S|\to\infty} 0$
  - Hard Margin SVM with Gaussian Kernel (or other universal kernel) or more generally min norm consistent solution: $\arg\min \|h\|_K \;\; s.t. \; L_S(h) = 0$

$$\equiv \arg\min_{\langle w,\phi(x_i)\rangle = y_i} \|w\|_2$$



E.g., hard margin SVM: $\min \|w\|$ s.t. $L_S^{margin}(w) = 0$

for $h_w = \langle w, \phi(x) \rangle$ with inf dim ϕ



E.g., hard margin SVM: $\min \|w\|$ s.t. $L_S^{margin}(w) = 0$

for $h_w = \langle w, \phi(x) \rangle$ with inf dim ϕ

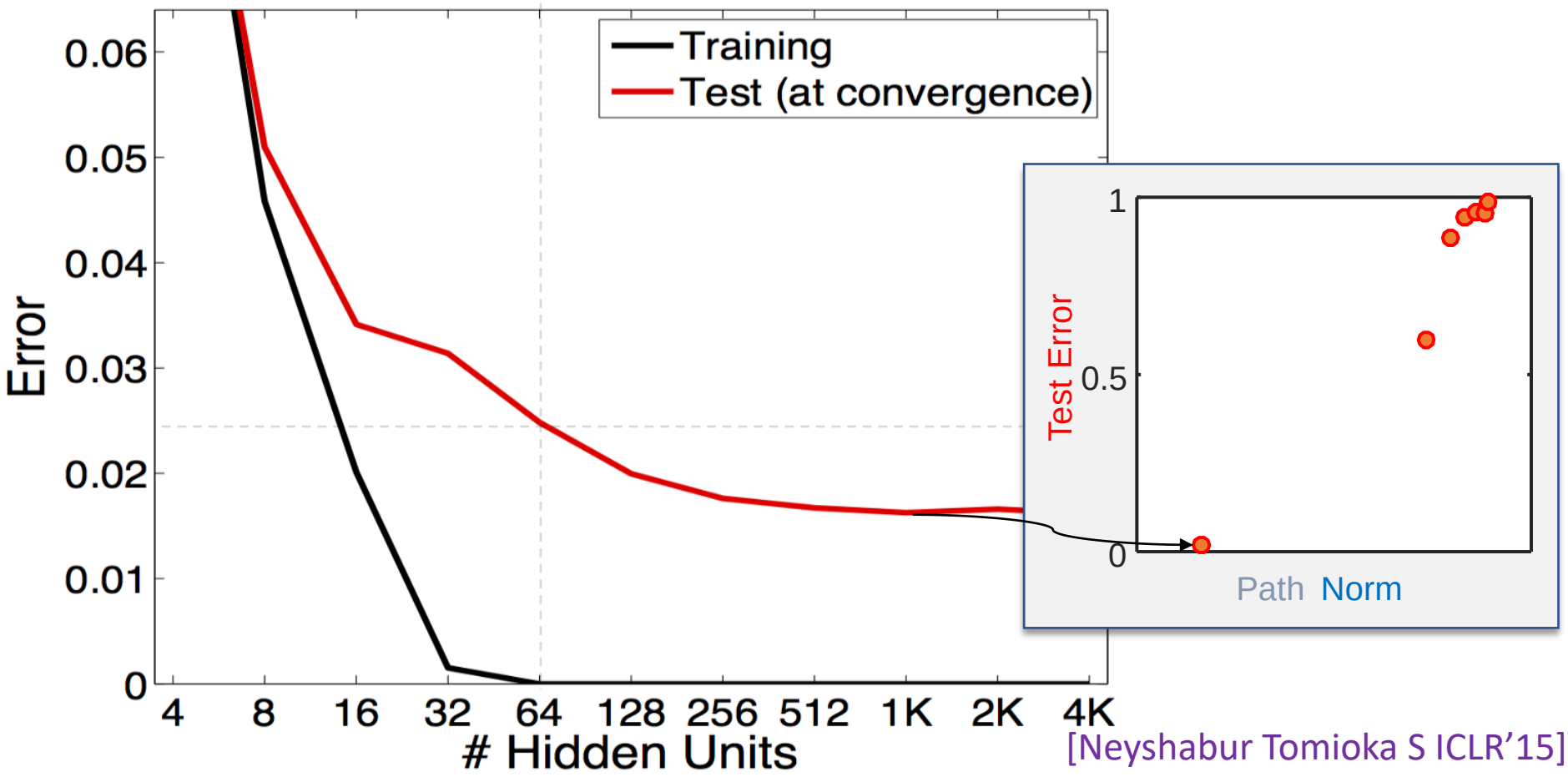
Learning with a Rich Function Class

- Learning rule $A(S)$ s.t.
  - For any data set, even with random labels, can fit data: $L_S\big(A(S)\big) = 0$
  - For "real" data $S \sim \mathcal{D}^m$ sampled from a reasonable reality $\mathcal{D}$, we can generalize: $L\big(A(S)\big)$ is low
- Examples:
  - 1-Nearest Neighbor: if realizable by some continuous h^* (ie $L(h^*) = 0$), then consistent: $L(1NN(S)) \xrightarrow{|S|\to\infty} 0$
  - Hard Margin SVM with Gaussian Kernel (or other universal kernel) or more generally min norm consistent solution: $\arg\min \|h\|_K \;\; s.t.\; L_S(h) = 0$

$$\equiv \arg\min_{\langle w,\phi(x_i)\rangle = y_i} \|w\|_2$$

    - Can always get $L_S(h) = 0$
    - If $\exists h^*, L_{\mathcal{D}}(h^*) = 0$, generalizes with sample complexity $|S| = O\big(\|h\|_K^2\big)$
  - MDL: $\arg\min |program| \;\; s.t.\; L(program) = 0$

    $L\big(MDL(S)\big) \le O\left(\frac{|prog^*|}{|S|}\right)$ if realizable by $prog^*$



[Neyshabur Tomioka S ICLR'15]

For valid generalization, the size of the weights is more important than the size of the network

1997

Peter L. Bartlett

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
(fitting random labels)		no	no	100.0	9.78
Inception w/o BatchNorm	1,649,402	no	yes	100.0	83.00
		no	no	100.0	82.00
(fitting random labels)		no	no	100.0	10.12
Alexnet	1,387,786	yes	yes	99.90	81.22
		yes	no	99.82	79.66
		no	yes	100.0	77.36
		no	no	100.0	76.07
(fitting random labels)		no	no	99.82	9.86
MLP 3x512	1,735,178	no	yes	100.0	53.35
		no	no	100.0	52.39
(fitting random labels)		no	no	100.0	10.48
MLP 1x512	1,209,866	no	yes	99.80	50.39
		no	no	100.0	50.51
(fitting random labels)		no	no	99.34	10.61

UNDERSTANDING DEEP LEARNING REQUIRES RE-THINKING GENERALIZATION

Zhang, Bengio, Hardt, Recht, Vinyals 2017

Where is the regularization?

- Consider an under-constraint least-squares problem ($n < m$):

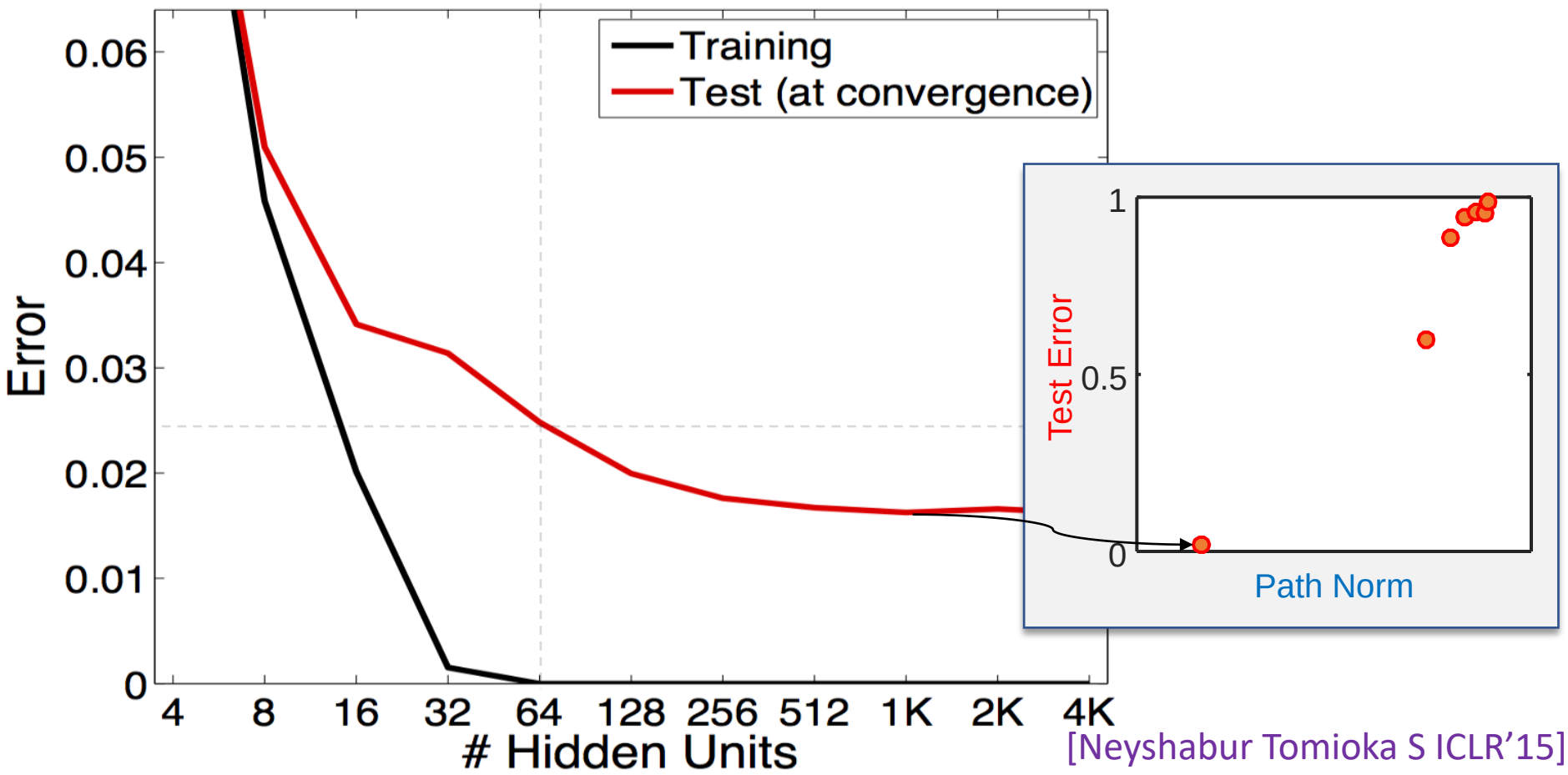
$$\min_{w \in \mathbb{R}^n} \| Aw - b \|^2$$

$A \in \mathbb{R}^{m \times n}$

- Claim: Gradient Descent (or SGD, or conjugate gradient descent, or BFGS) converges to the least norm solution

$$\min_{Aw=b} \|w\|_2$$

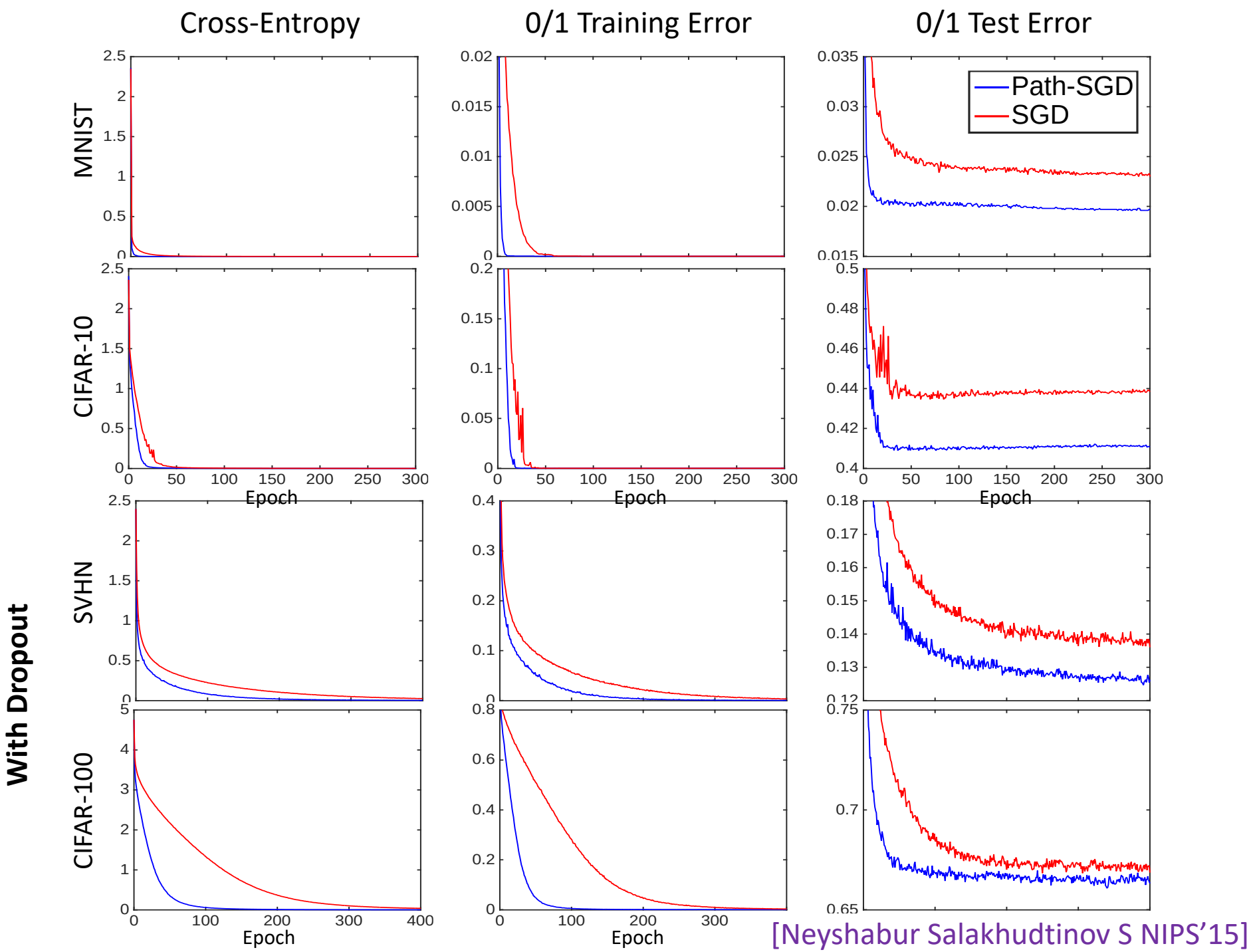
➢ Proof: iterates always spanned by rows of A (more details soon)



[Neyshabur Tomioka S ICLR'15]

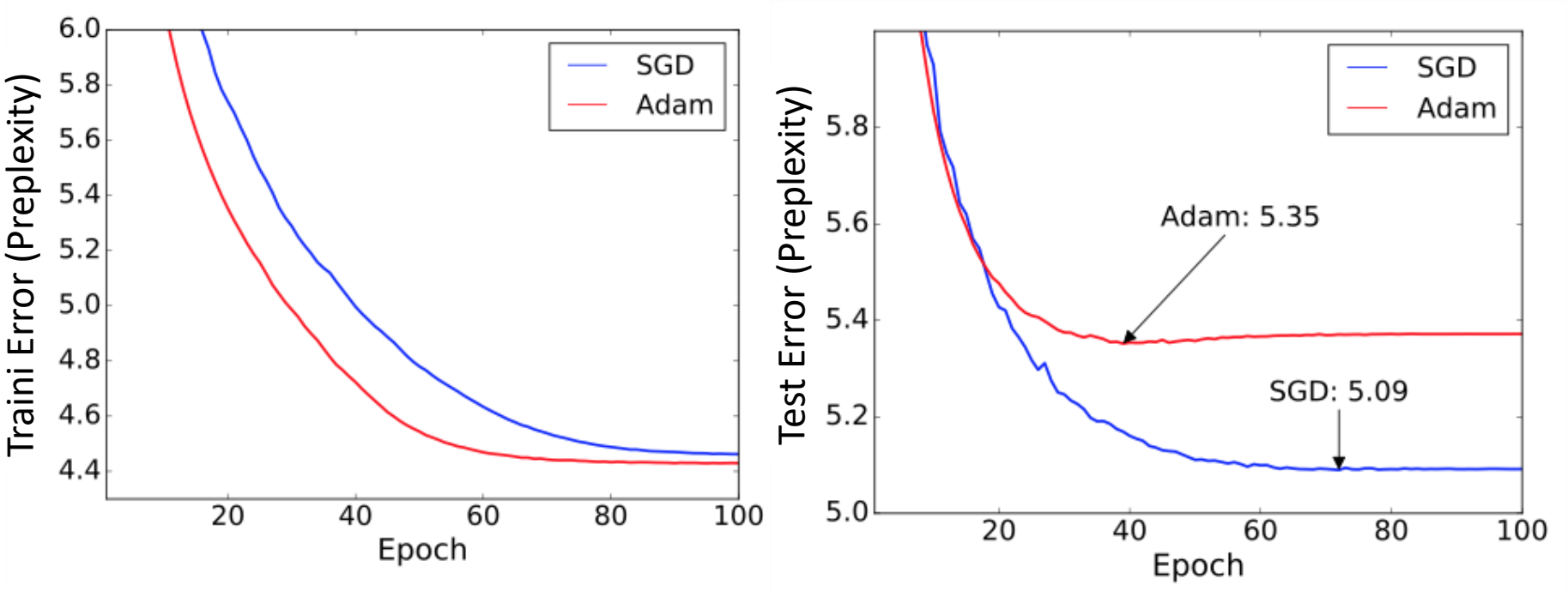
Peter L. Bartlett

- What is the relevant "complexity measure" (eg norm)?
- How is this minimized (or controlled) by the opt algorithm?
- How does it change if we change the opt algorithm?



[Neyshabur Salakhudtinov S NIPS'15]

SGD vs ADAM



Results on Penn Treebank using 3-layer LSTM

[Wilson Roelofs Stern S Recht, “The Marginal Value of Adaptive Gradient Methods in Machine Learning”, NIPS’17]

Different optimization algorithm

➔ Different bias in optimum reached

➔ Different Inductive bias

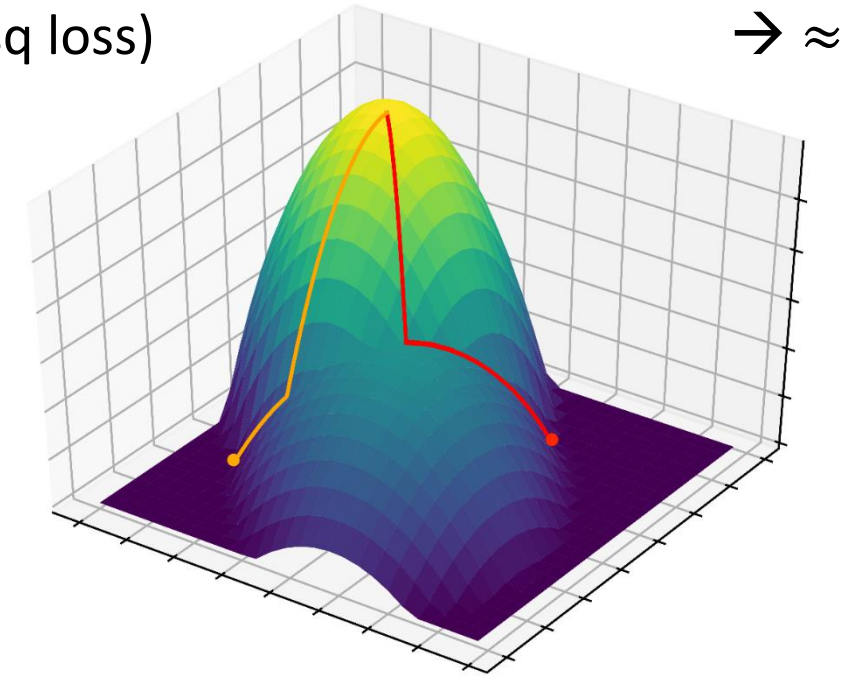
➔ Different generalization properties

Grad Descent

→ $\min_{Xw=y} \|w\|_2$ (sq loss)

Coordinate Descent

→ $\approx \min_{Xw=y} \|w\|_1$ (sq loss)



Need to understand optimization alg. not just as reaching ***some*** (global) optimum, but as reaching a ***specific*** optimum