Towards an Empirical Theory of Deep Learning

Preetum Nakkiran

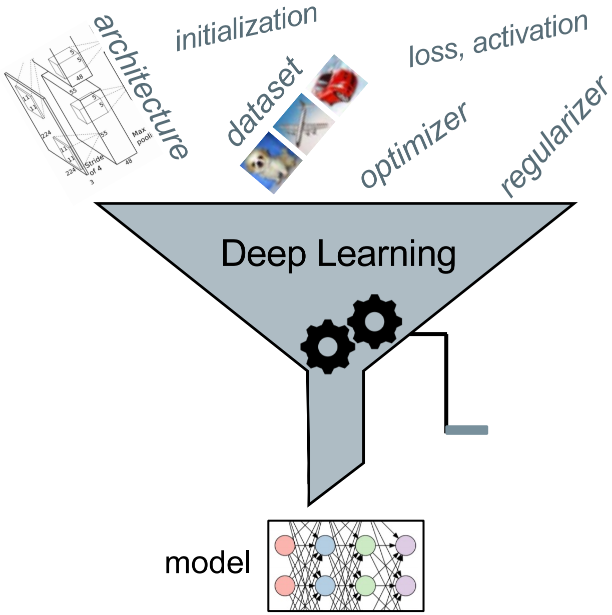
Thesis Defense. July 12, 2021

Advisors: Boaz Barak & Madhu Sudan

What is Deep Learning?

Deep Learning (informal):

A set of *ingredients* that can be combined to solve a certain *learning problems*



What is Deep Learning?

Very successful in practice:

- Solved “hard” problems
- Solved “new” problems

ARTICLE

doi:10.1038/nature24270

Mastering the game of Go without human knowledge

David Silver[1]*, Julian Schrittwieser[1]*, Karen Simonyan[1]*, Ioannis Antonoglou[1], Aja Huang[1], Arthur Guez[1], Thomas Hubert[1], Lucas Baker[1], Matthew Lai[1], Adrian Bolton[1], Yutian Chen[1], Timothy Lillicrap[1], Fan Hui[1], Laurent Sifre[1],

ImageNet Classification with Deep Convolutional Neural Networks

Alex Krizhevsky
University of Toronto
kriz@cs.utoronto.ca

Ilya Sutskever
University of Toronto
ilya@cs.utoronto.ca

Geoffrey E. Hinton
University of Toronto
hinton@cs.utoronto.ca

‘IT WILL CHANGE EVERYTHING’: AI MAKES GIGANTIC LEAP IN SOLVING PROTEIN STRUCTURES

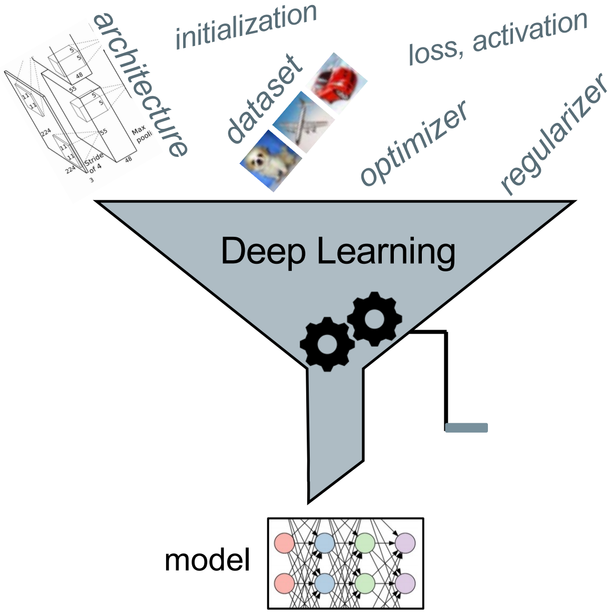
DeepMind’s program for determining the 3D shapes of proteins stands to transform biology, say scientists.

The New York Times

Meet GPT-3. It Has Learned to Code (and Blog and Argue).

The latest natural-language system generates tweets, pens poetry, summarizes emails, answers trivia questions, translates languages and even writes its own computer programs.

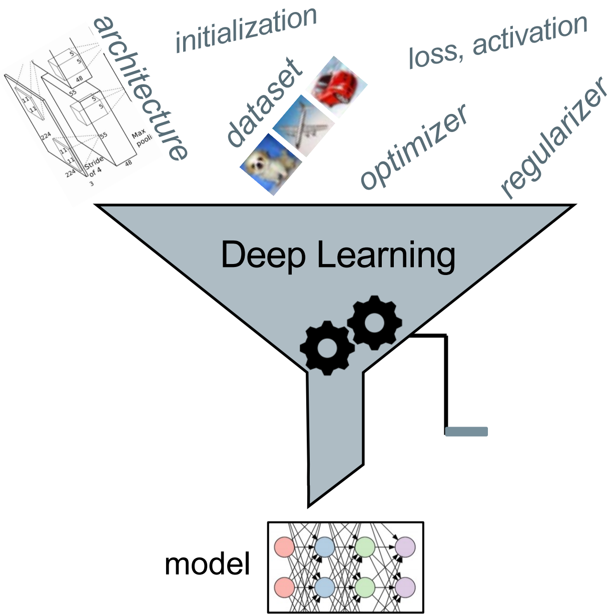
Advances are *Unpredictable*



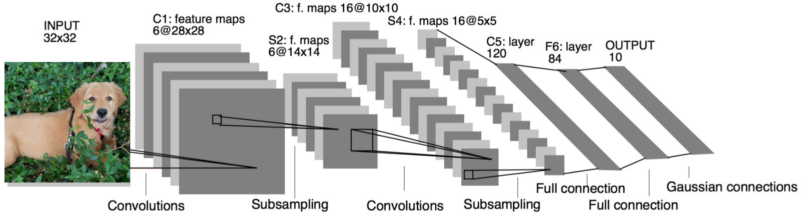
Every advance = new choice of "ingredients"

Surprised by which choices work!

Advances are *Unpredictable*

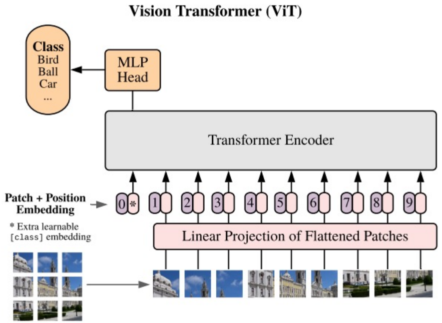


~1998-2020: ConvNets dominate vision



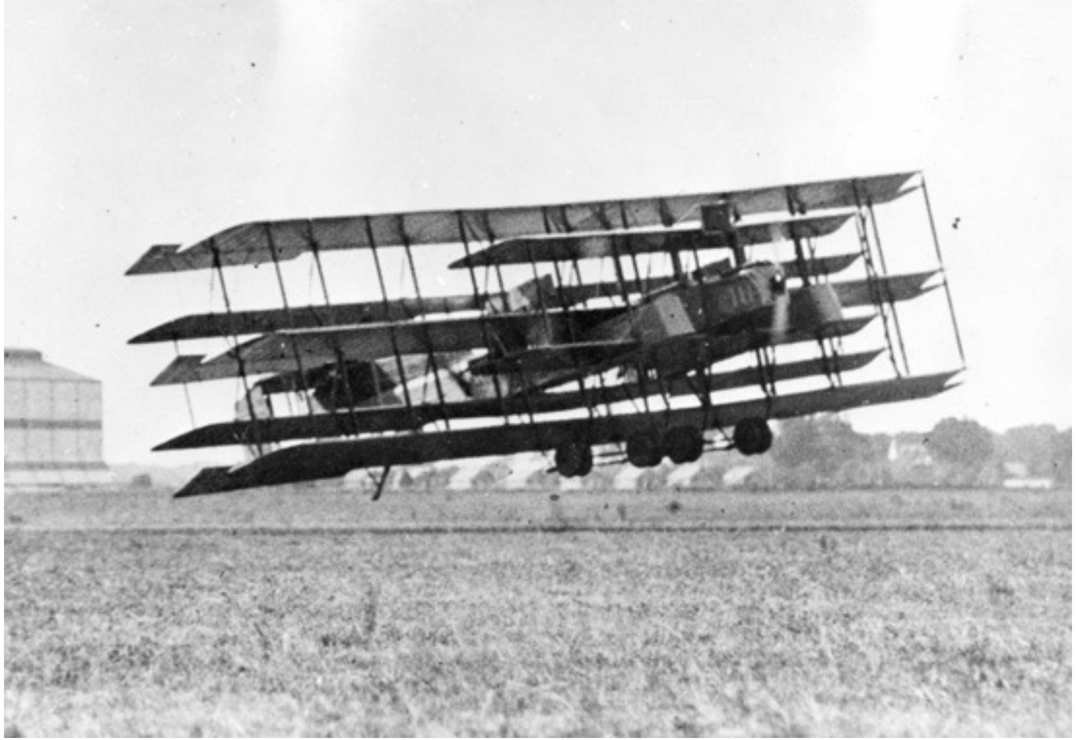
[LeCun et al 1998]

2020: *Transformers* (from NLP) dominate vision



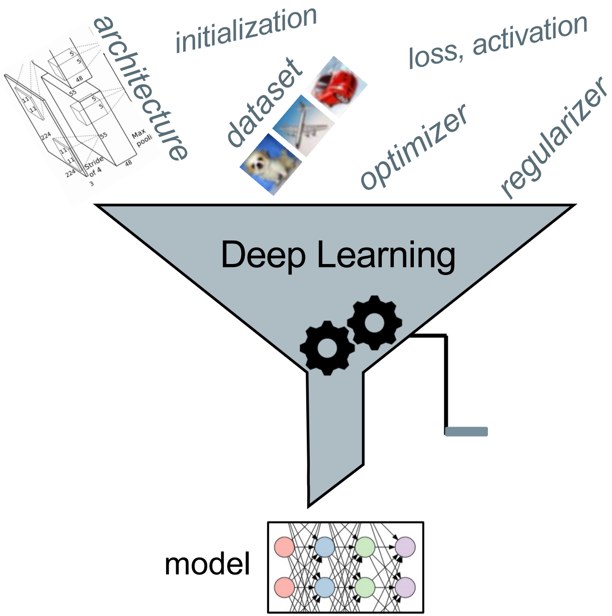
[Dosovitskiy et al 2020]

Deep Learning Practice



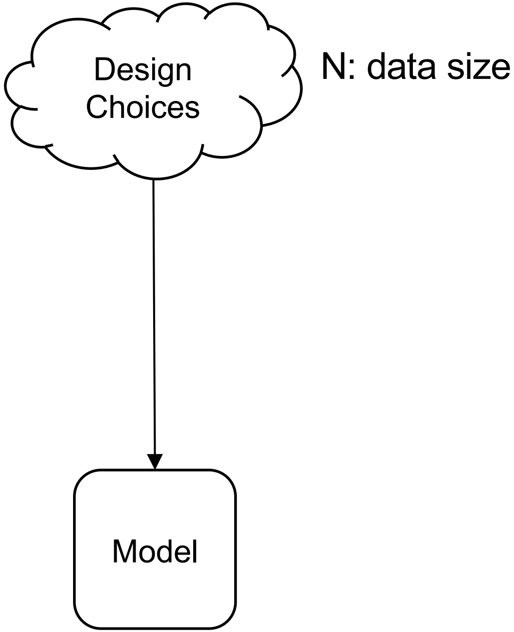
[https://www.flickr.com/photos/sdasmarchives/4590501514]

Guiding Question



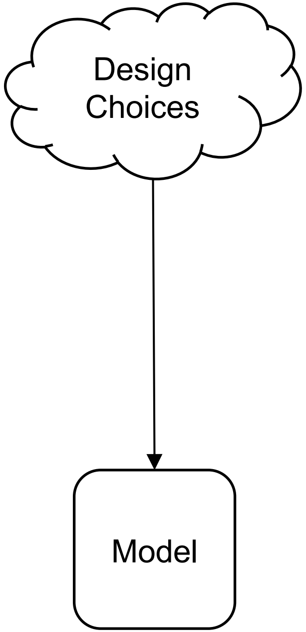
" How does what we ***do*** affect what we ***get?*** "

Guiding Question



“ How does what we ***do*** affect what we ***get?*** ”

Obstacles to Mathematical Rigor



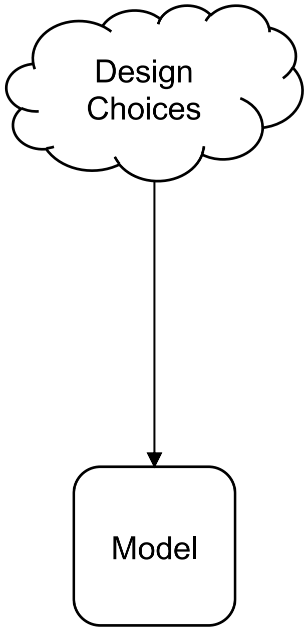
"<u>Theorem</u>: **Deep neural nets** with design choices X, on **task Y**, have performance F(X, Y)"

Not even too hard. Too ill-defined!

1. Can't define "deep neural nets"
   (big enough to include practice,
   small enough to exclude P/poly)

2. Can't define the tasks they solve
   (Vision, NLP,…)

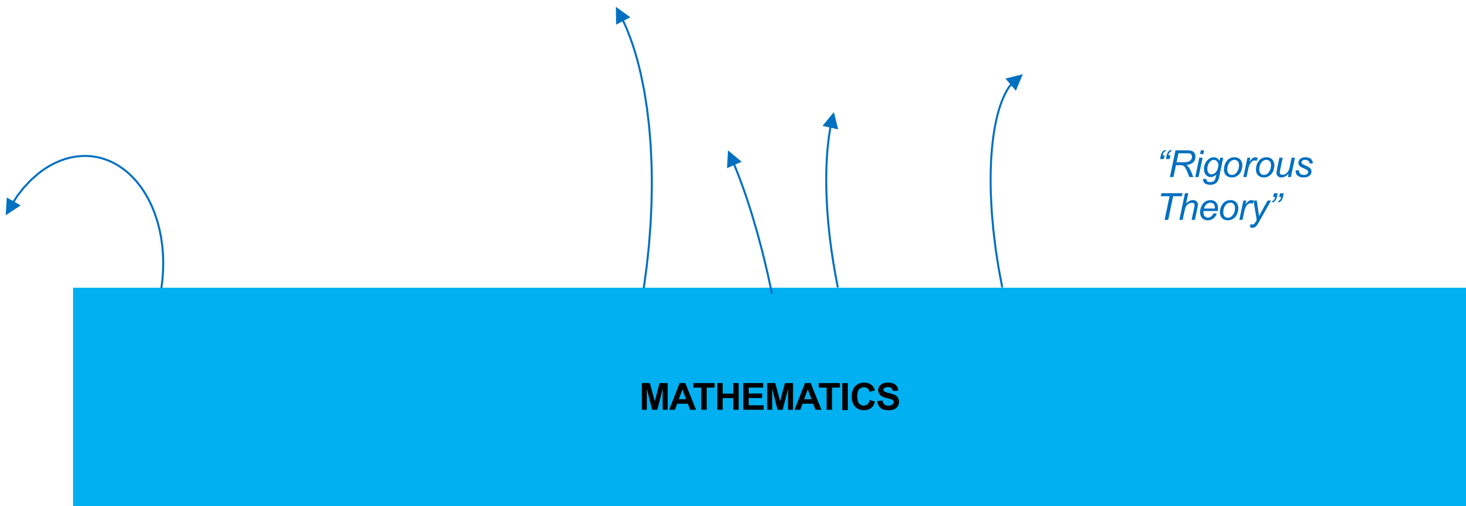
The Two Cultures



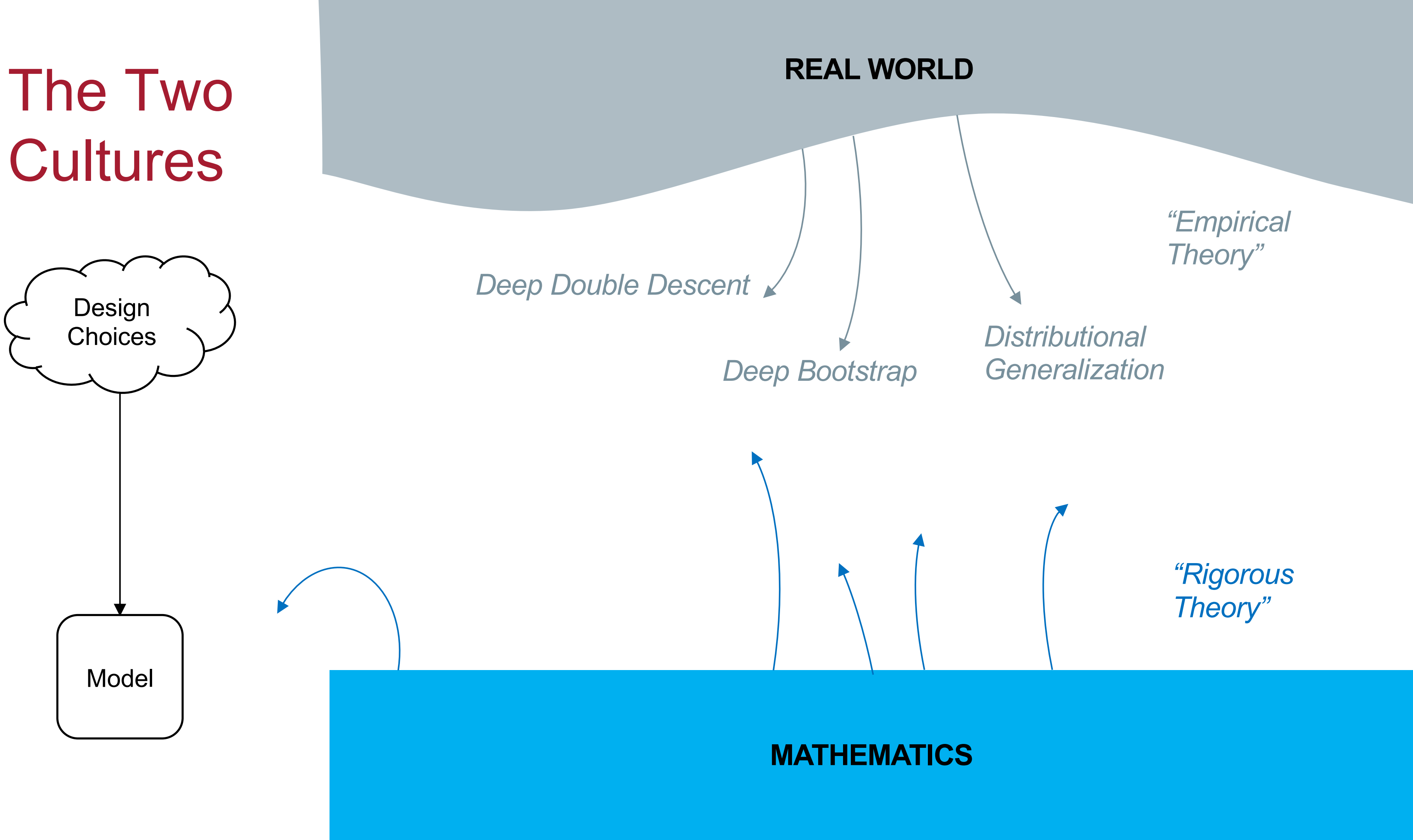
<u>Empirical Theory in Physics:</u>

- Kepler's **Laws**
- Ideal Gas **Law**
- Hooke's **Law**
- **…**

characterize first; prove later!



The Two Cultures



BACKGROUND

“what do we do?”

Supervised Classification

Setup:

Distribution D over pairs (input, label): $D \in \Delta(\mathcal{X} \times \mathcal{Y})$

Ex: Image Classification

$\mathcal{X}$ = { images of cats/dogs }

$\mathcal{Y}$ = { 'cat', 'dog' }

( , 'cat')

( , 'dog')

( , 'cat')

( , 'dog')

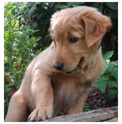
Given:

IID samples from distribution $(x_i, y_i) \sim D$

Want:

Find function $f: \mathcal{X} \rightarrow \mathcal{Y}$ with small test error:

$$\text{TestError}(f) := \Pr_{x,y \sim D}[f(x) \neq y]$$

$f($  $)$ = 'dog'

What we want:

Function $f: \mathcal{X} \to \mathcal{Y}$ with small: $\text{TestError}(f) \coloneqq \Pr_{x,y \sim D}[f(x) \neq y]$

What we do:

1. Pick a parametric family of functions $\mathcal{F}$ ("*neural network architecture*")

   search for $f_\theta \in \mathcal{F}$

2. Draw *N* samples from distribution: $\{(x_i, y_i)\}$ *("train set")*

3. Try to "fit" the train set. Find θ to minimize:

$$\text{L}(\theta) = \text{TrainError}(f_\theta) \coloneqq \frac{1}{N} \sum_i \mathbb{I}[f_\theta(x_i) \neq y_i]$$

Minimize $L(\theta)$ via a local optimizer (e.g. Stochastic Gradient Descent)

What we want:

Small $\text{TestError}(f) \coloneqq \Pr_{x,y \sim D}[f(x) \neq y]$

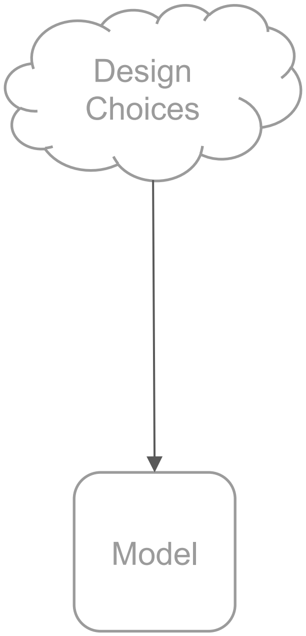
"The Generalization Problem"

What we do:

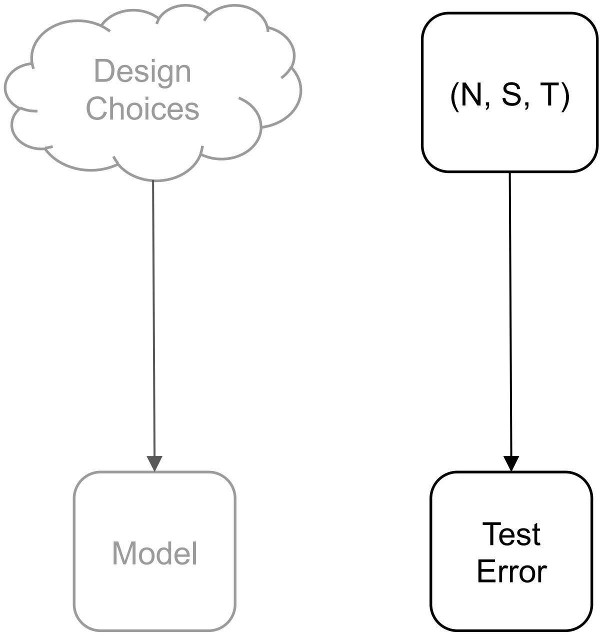
Minimize (via SGD) $\text{TrainError}(f_\theta) \coloneqq \frac{1}{N} \sum_i \mathbb{I}[f_\theta(x_i) \neq y_i]$

PART I:
DEEP DOUBLE DESCENT

[**N.**, Kaplun*, Bansal*, Yang, Barak, Sutskever ICLR 2020]

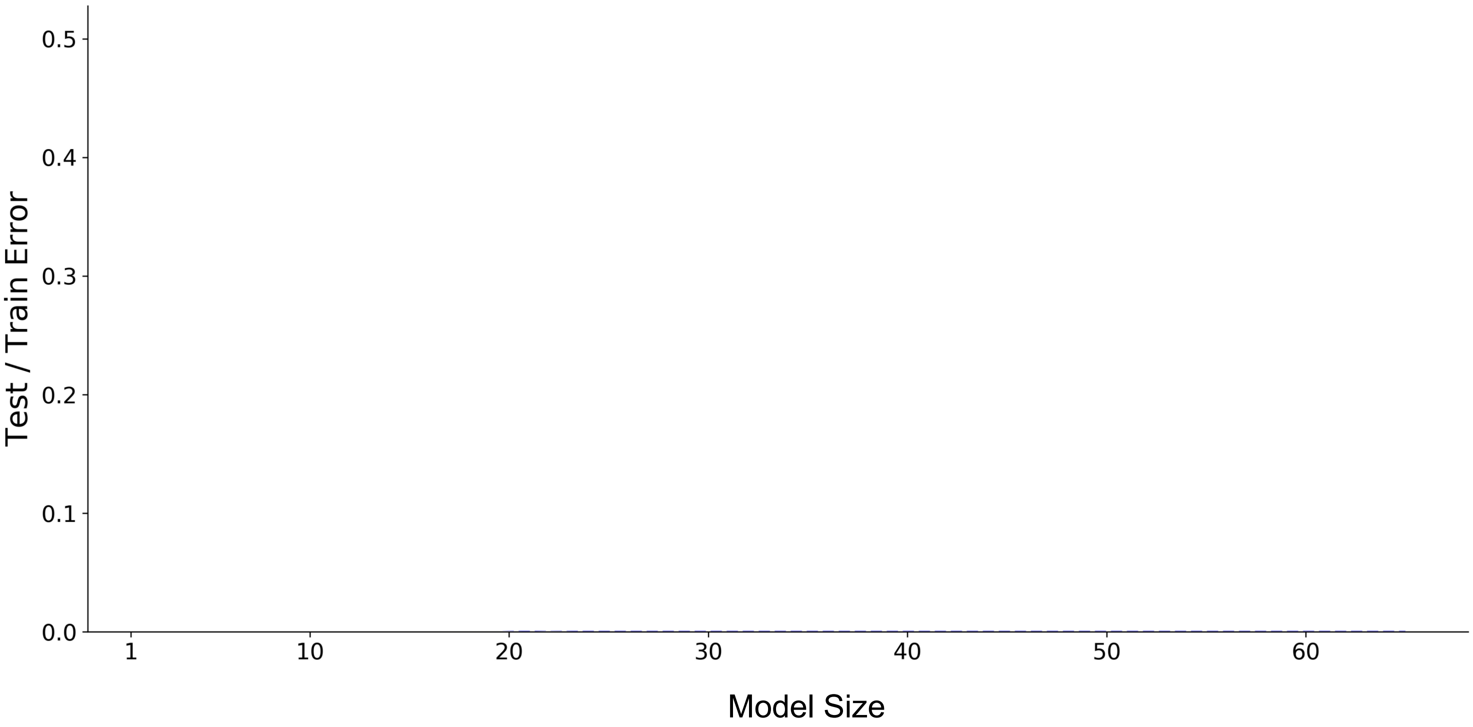

Design Choices
Model

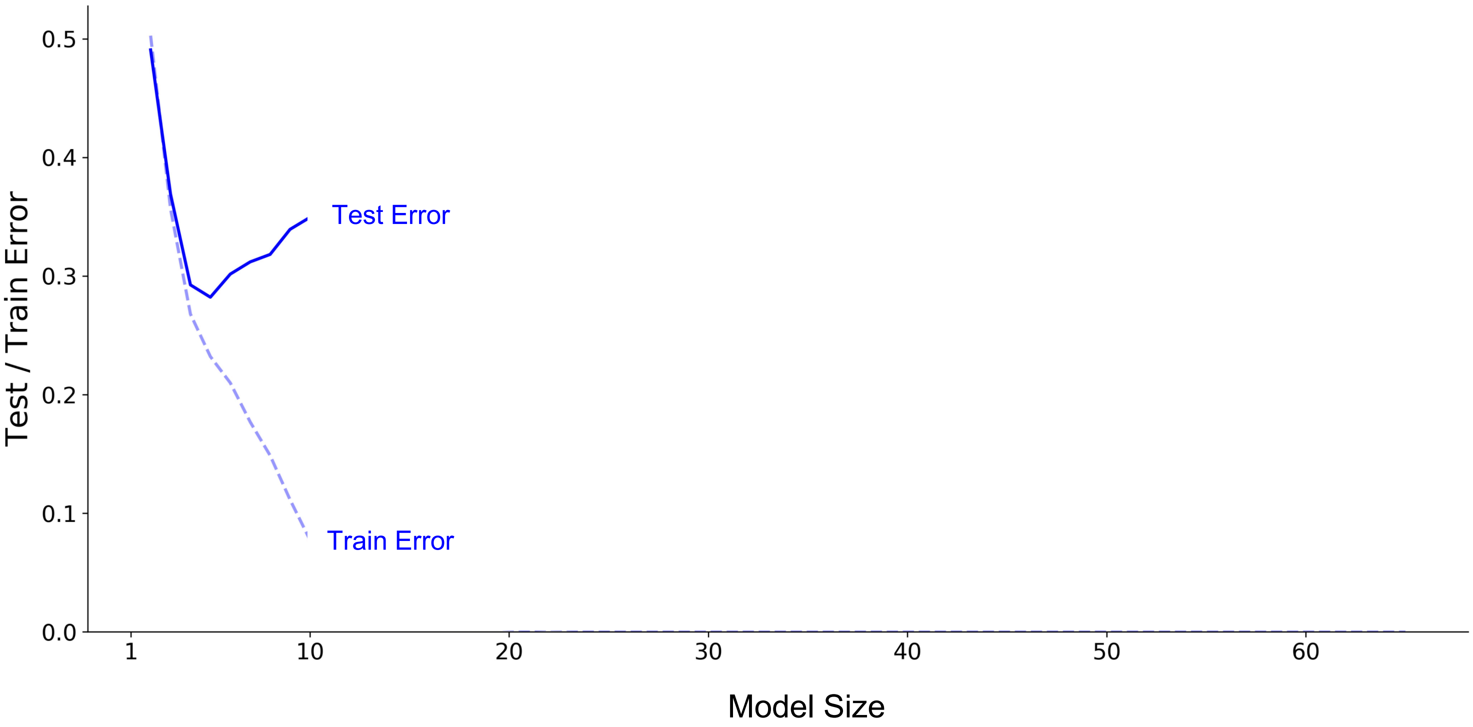
N: Num. samples
S: Model size
T: Train time (optimization steps)

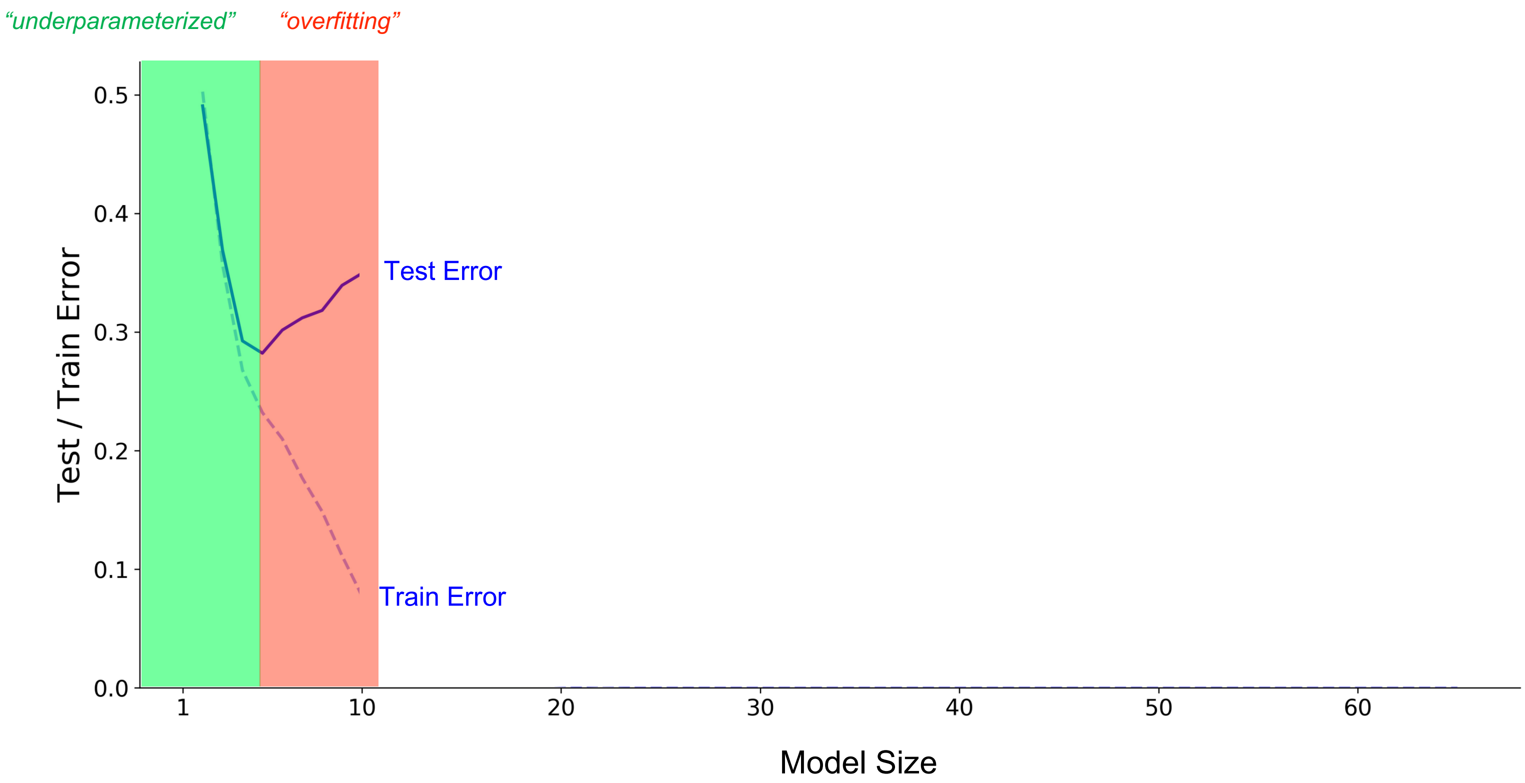


~~Monotonic in (N, S, T)?~~
- ~~More data better~~
- ~~Larger models better~~
- ~~Longer train time better~~

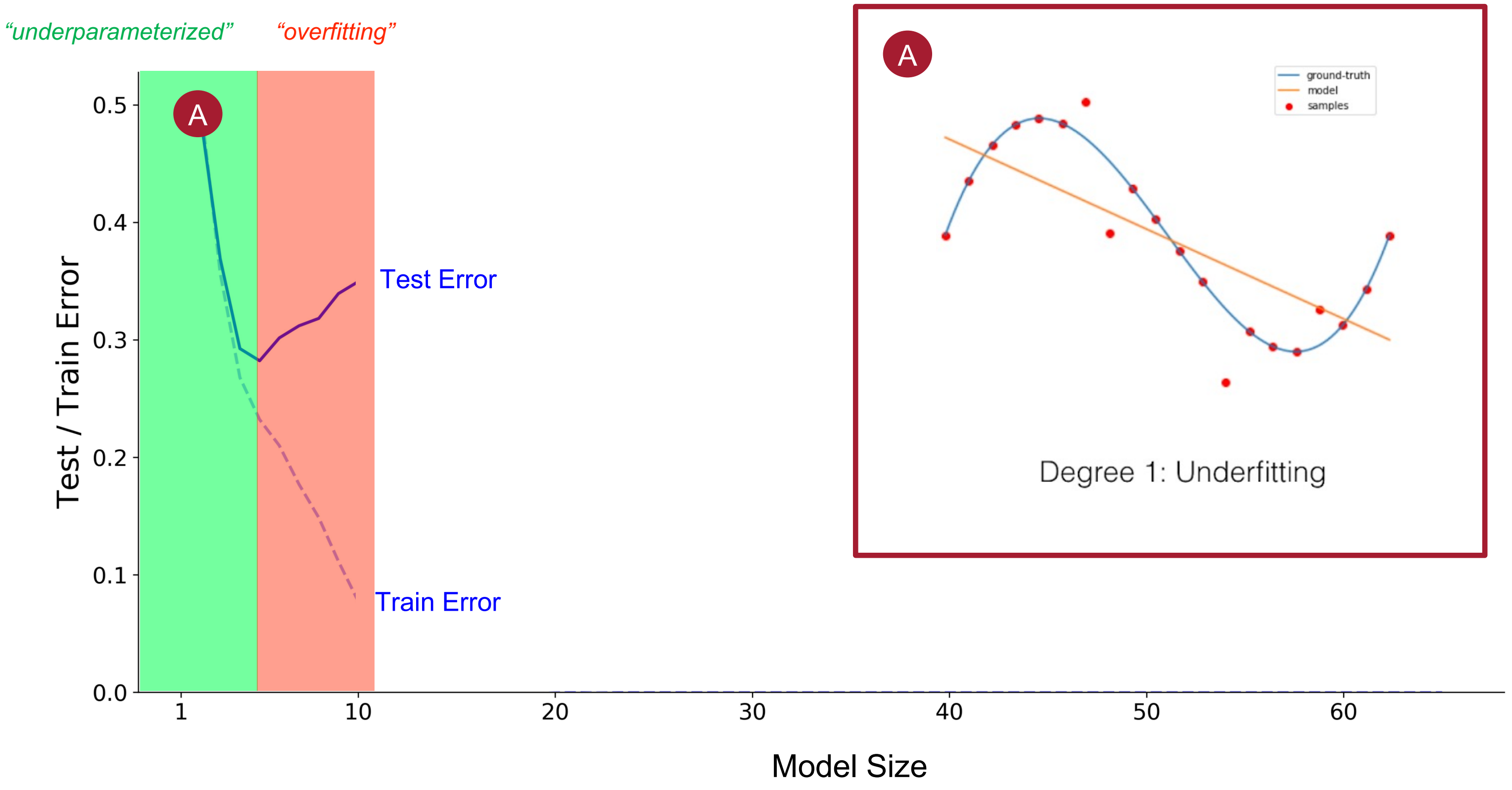
Fails for the same reason: systematic “obstruction”


Test / Train Error
0.5
0.4
0.3
0.2
0.1
0.0
1
10
20
30
40
50
60
Model Size

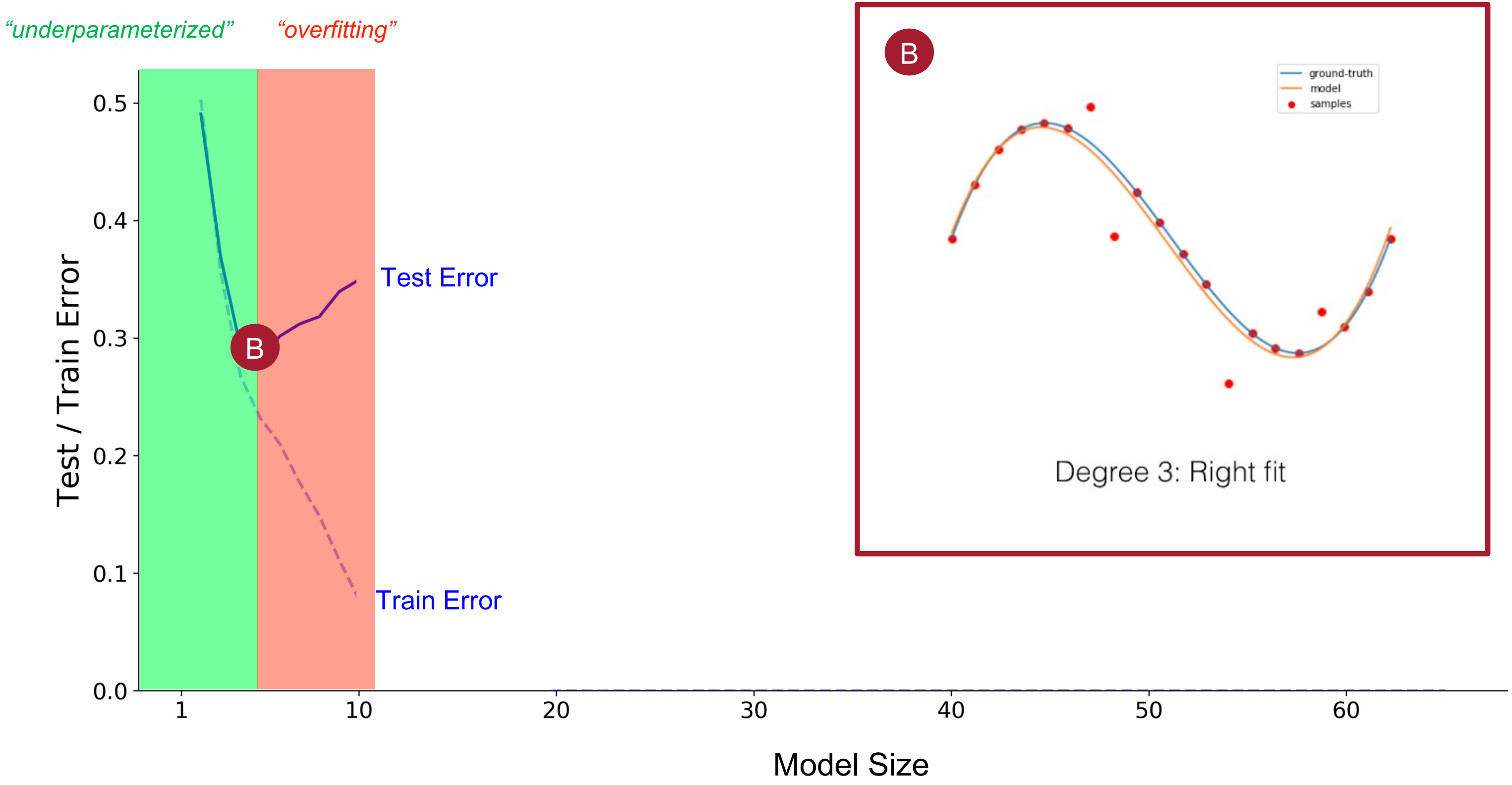

Test / Train Error
0.5
0.4
0.3
0.2
0.1
0.0
Test Error
Train Error
1
10
20
30
40
50
60
Model Size



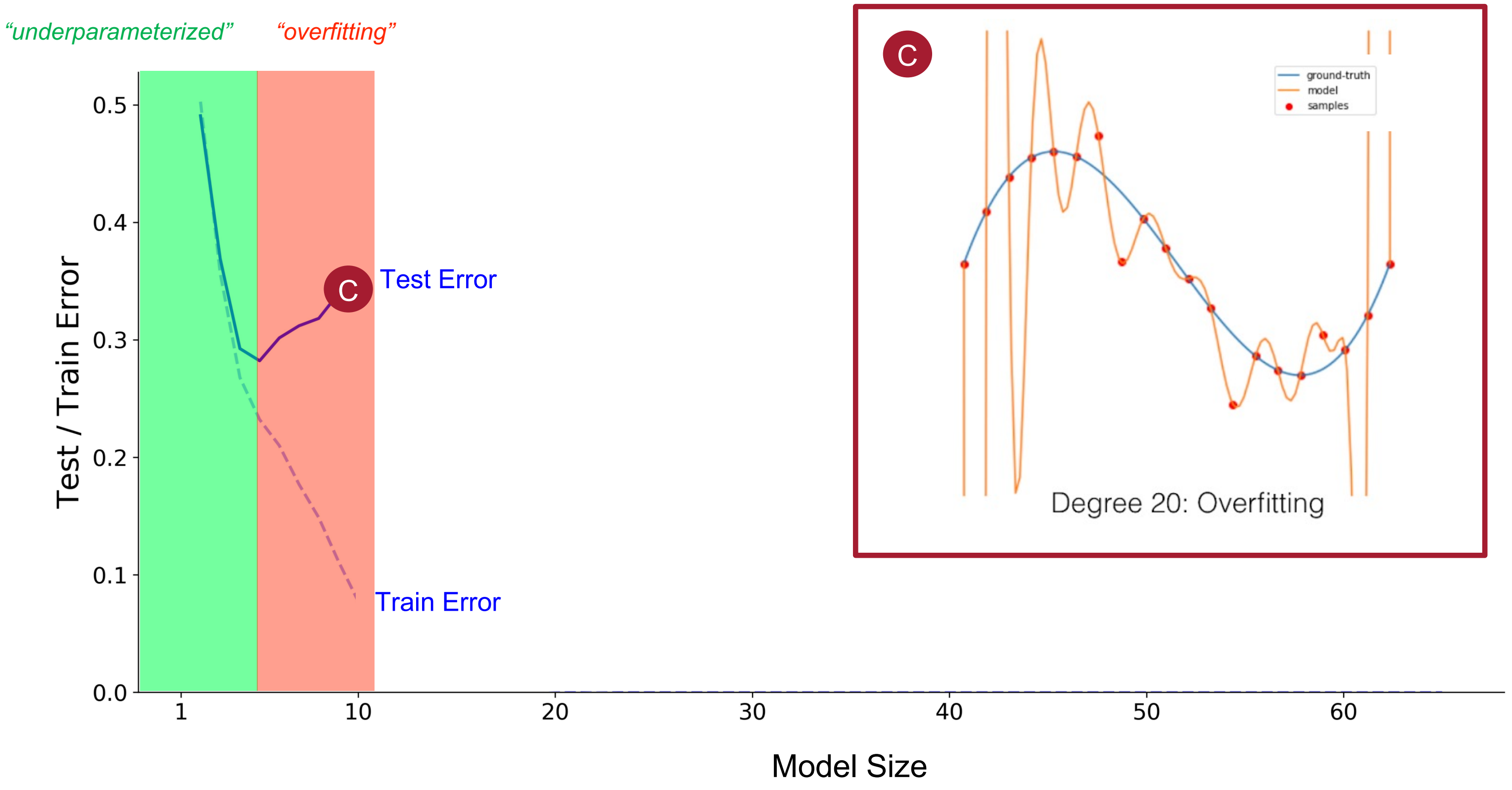
“underparameterized”
“overfitting”
Test Error
Train Error
Test / Train Error
0.0
0.1
0.2
0.3
0.4
0.5
1
10
20
30
40
50
60
Model Size



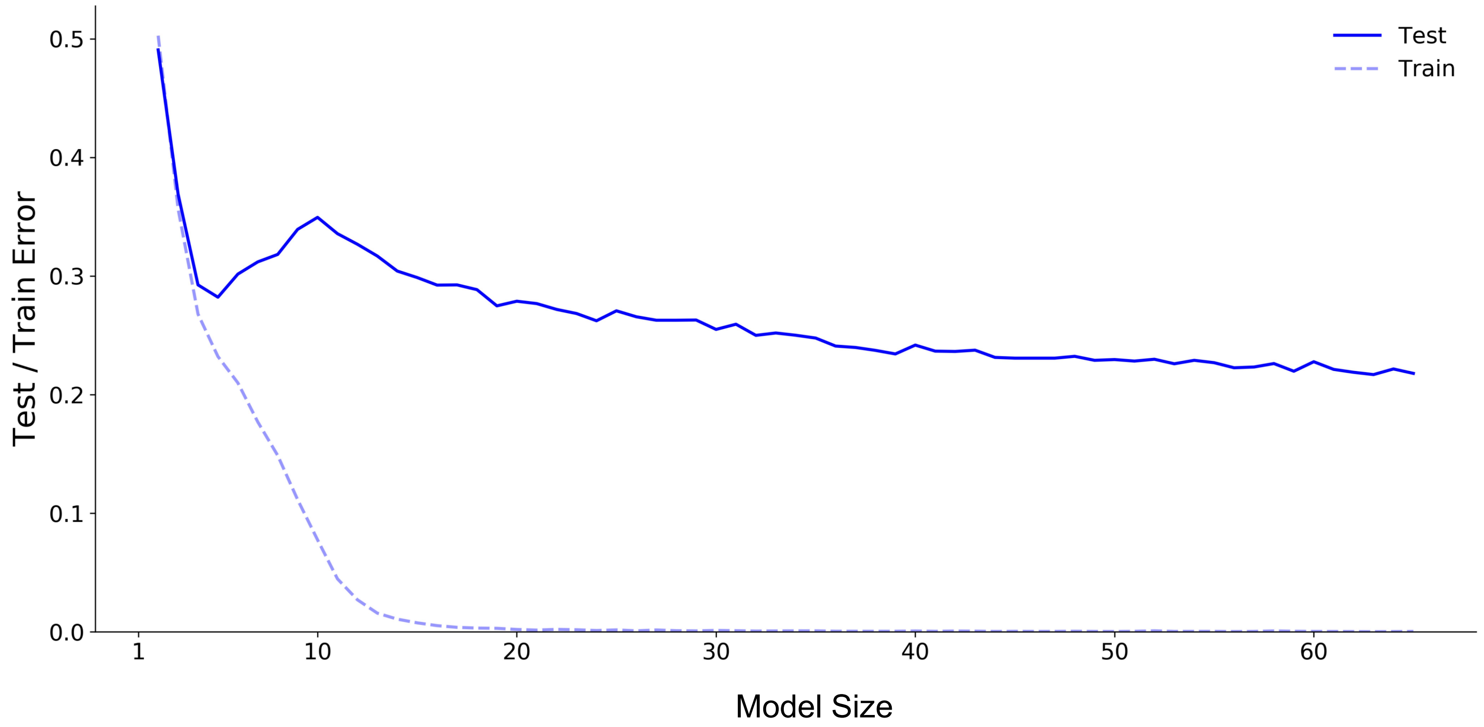
“underparameterized”
“overfitting”
A
0.5
0.4
0.3
0.2
0.1
0.0
Test / Train Error
Test Error
Train Error
1
10
20
30
40
50
60
Model Size
A
ground-truth
model
samples
Degree 1: Underfitting

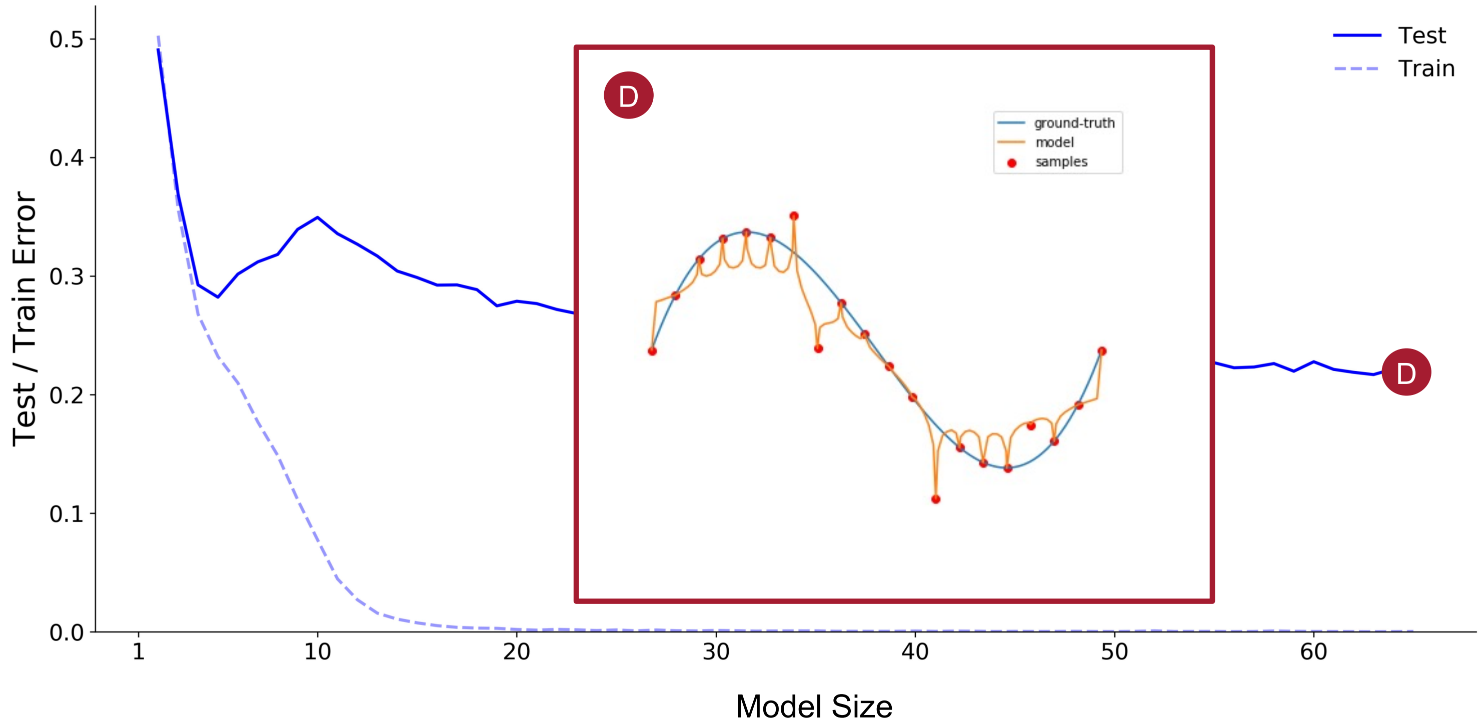


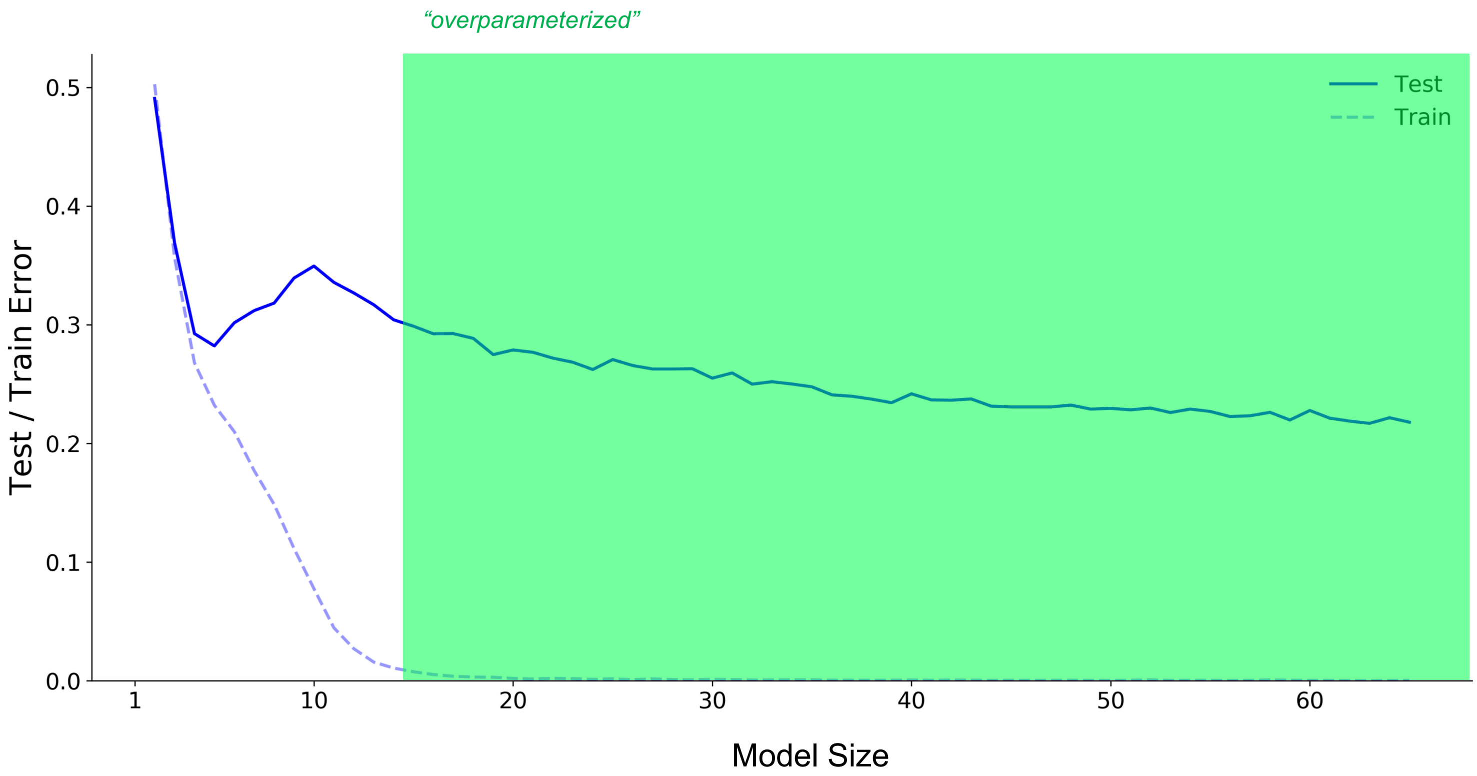
"underparameterized"
"overfitting"
0.5
0.4
0.3
0.2
0.1
0.0
Test / Train Error
B
Test Error
Train Error
1
10
20
30
40
50
60
Model Size
B
ground-truth
model
samples
Degree 3: Right fit



“underparameterized”
“overfitting”
Test / Train Error
0.5
0.4
0.3
0.2
0.1
0.0
C
Test Error
Train Error
1
10
20
30
40
50
60
Model Size
C
ground-truth
model
samples
Degree 20: Overfitting


Test
Train
Test / Train Error
0.0
0.1
0.2
0.3
0.4
0.5
1
10
20
30
40
50
60
Model Size

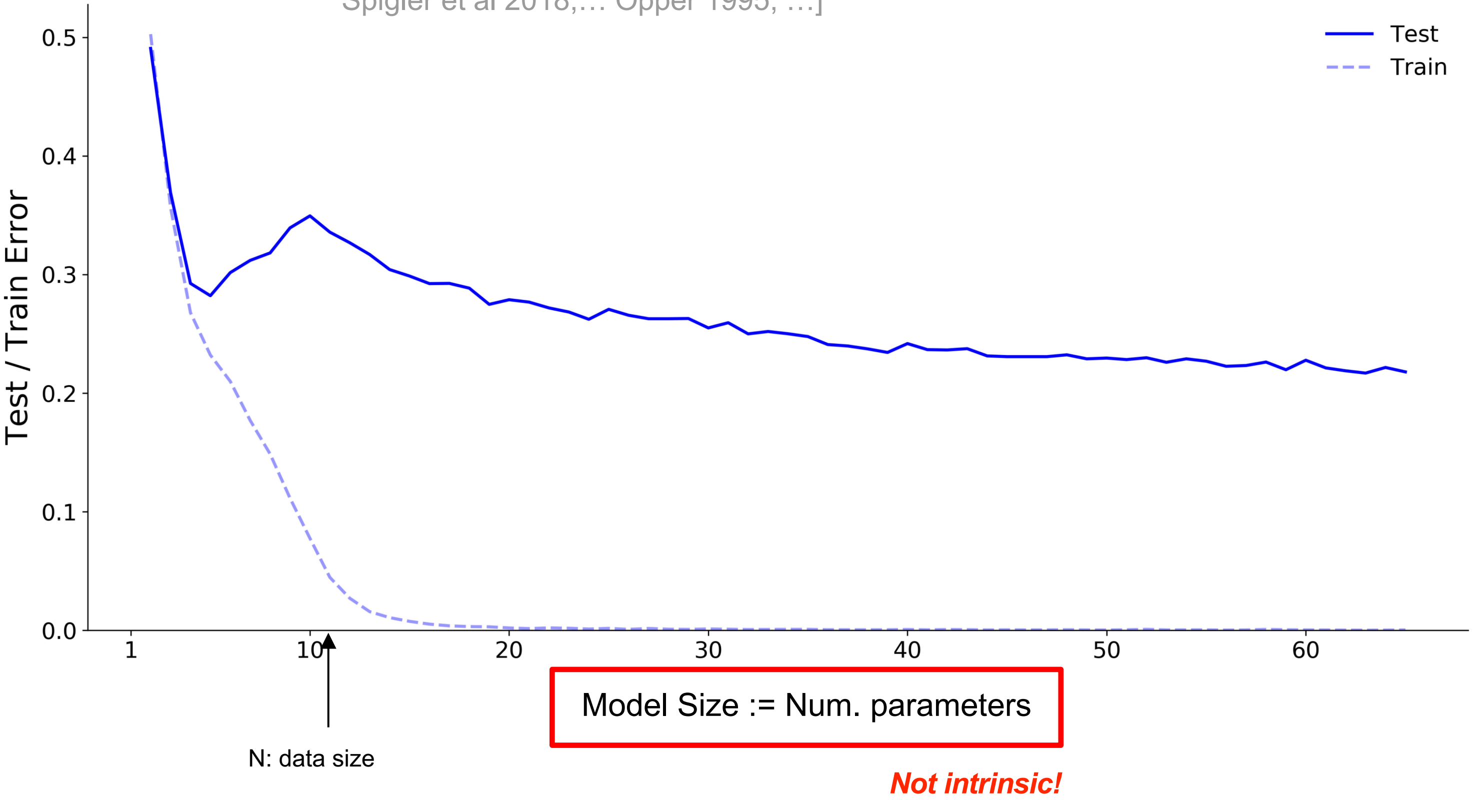

Test / Train Error
0.5
0.4
0.3
0.2
0.1
0.0
1
10
20
30
40
50
60
Model Size
Test
Train
D
ground-truth
model
samples
D



“overparameterized”
Test
Train
0.5
0.4
0.3
0.2
0.1
0.0
Test / Train Error
1
10
20
30
40
50
60
Model Size

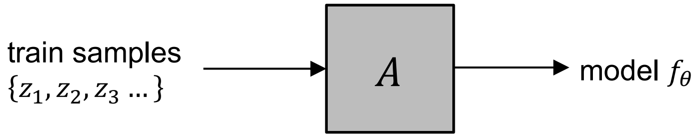
“Double Descent”

[Belkin et al 2019, Advani & Saxe 2017, Geiger et al 2019,
Spigler et al 2018,… Opper 1995, …]



Our Contributions

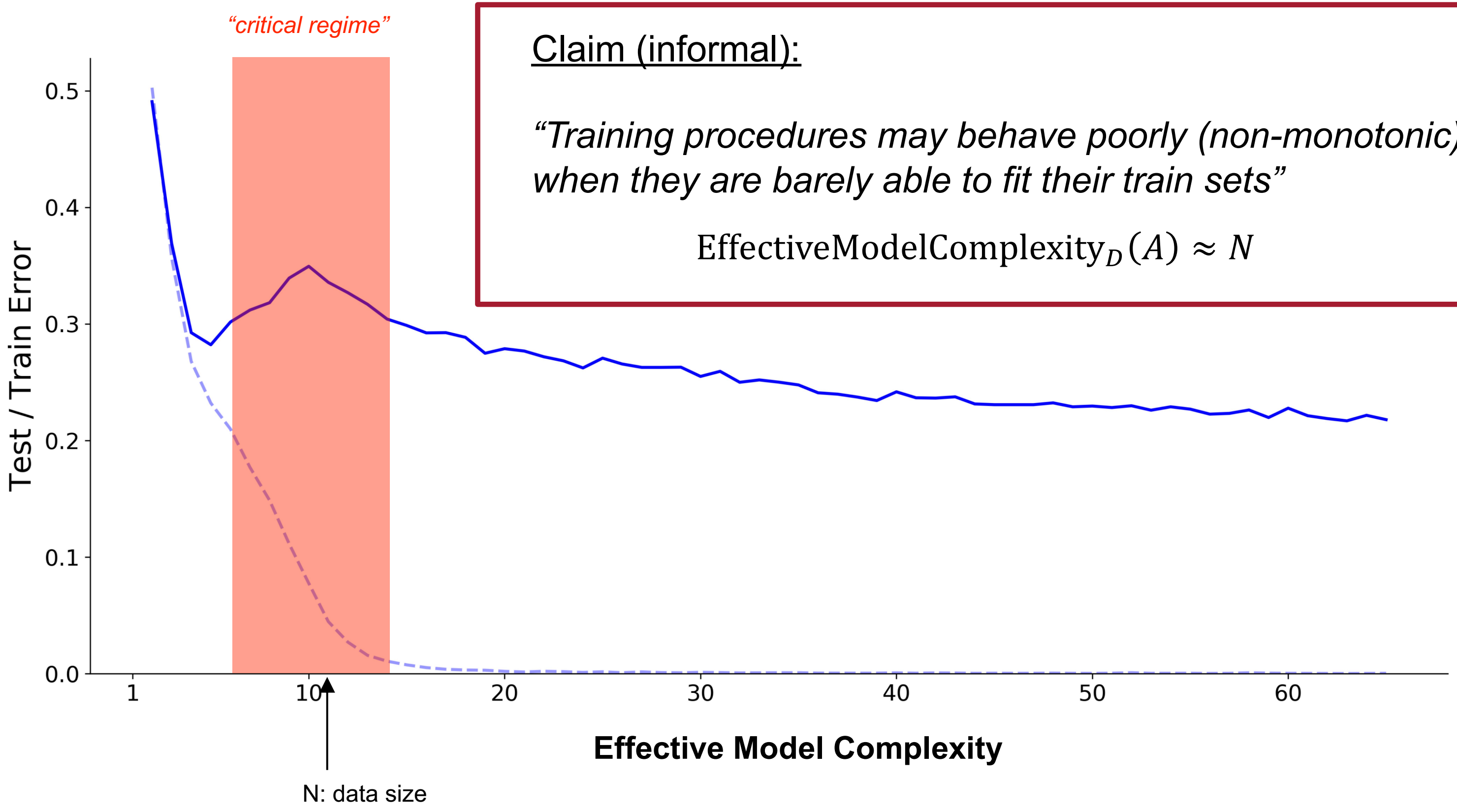
1. Generalized "model size" to the *entire training procedure* A



$$\text{EffectiveModelComplexity}_D(A) \coloneqq$$

"max num. samples $\{z_i\} \sim D$ *that* A *fits to* ≈ 0 *train error"*

Generalized Double Descent



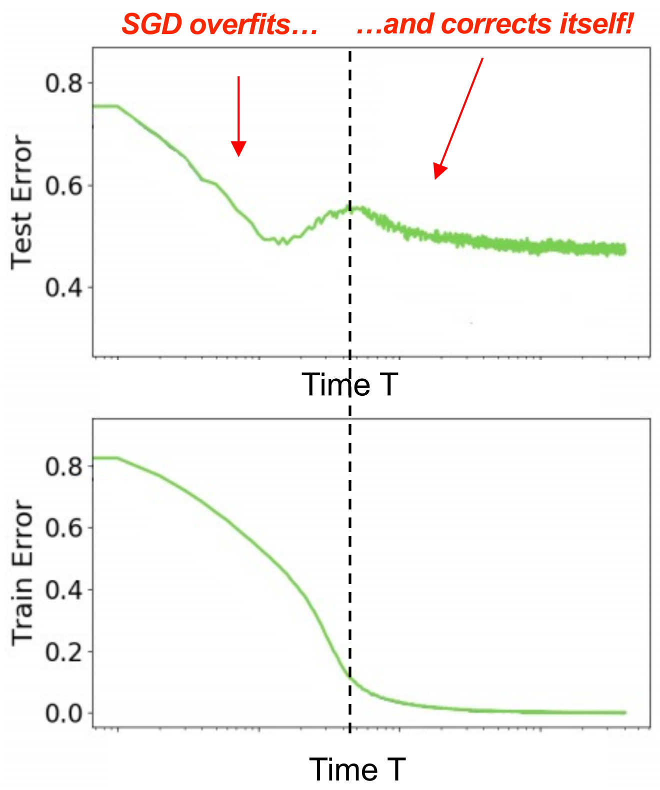
Claim (informal):

"Training procedures may behave poorly (non-monotonic) when they are barely able to fit their train sets"

$$\text{EffectiveModelComplexity}_D(A) \approx N$$

New Behaviors

Fix large model, increase optimization steps (T): <u>Epoch-wise double descent</u>

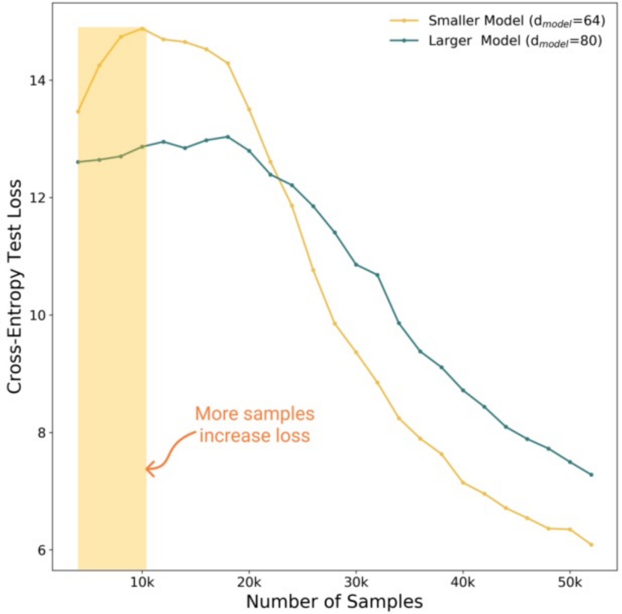


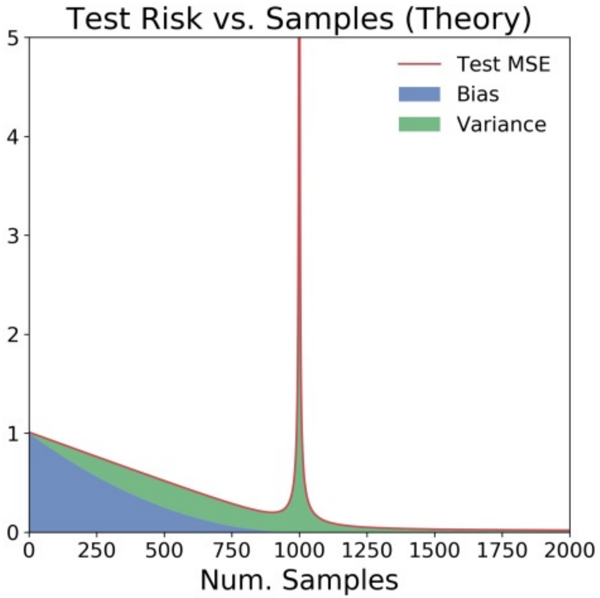
New Behaviors

Fix model size, train steps. Increase data-size (N).

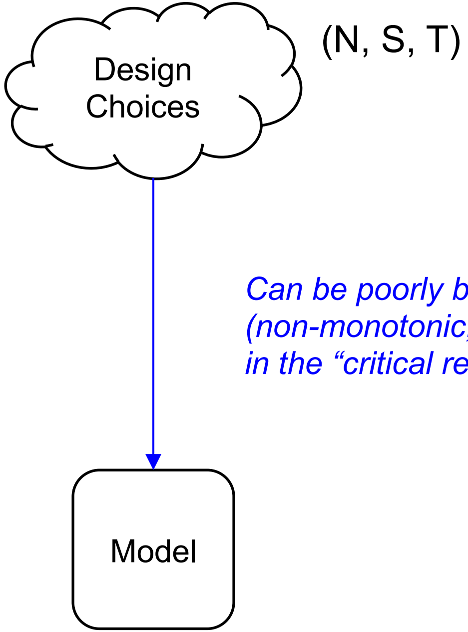
Training on more data can hurt performance!

Sample-wise double descent

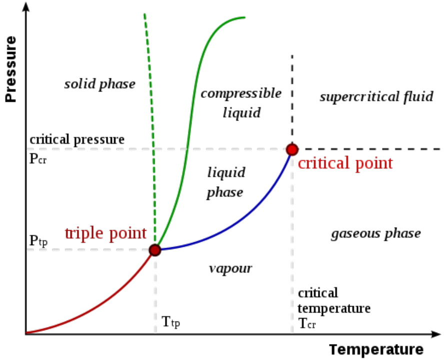


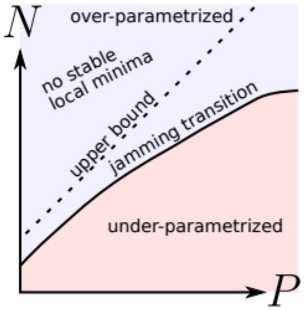


[N. 2019]

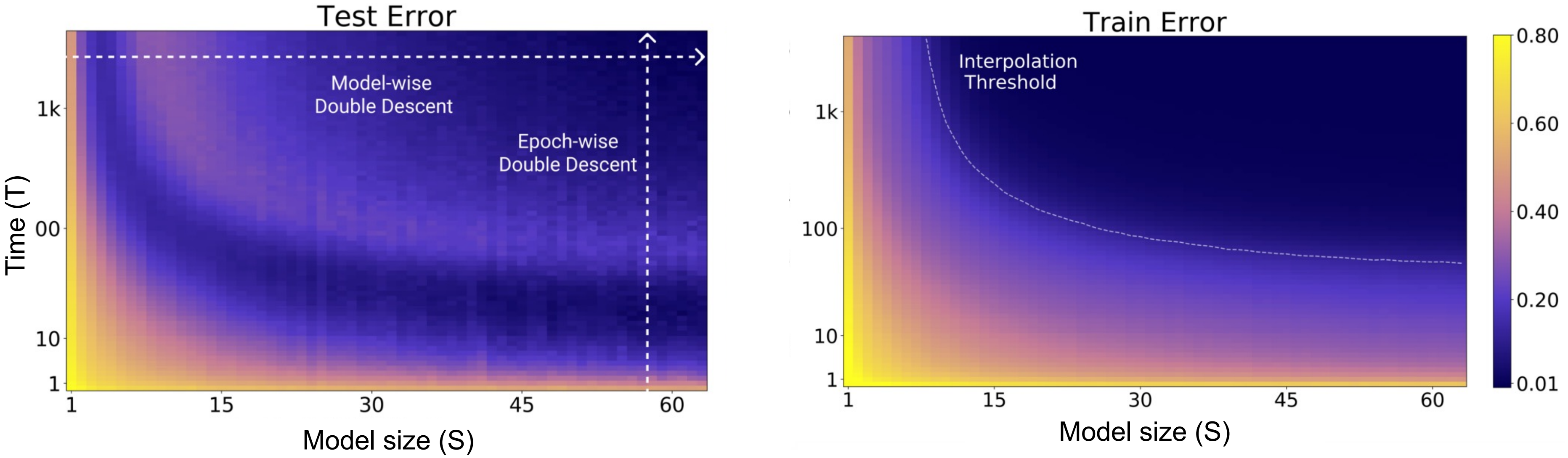


Can be poorly behaved (non-monotonic, discontinuous) in the "critical regime"





[Spigler, Geiger et al 2019, …]


Test Error
Model-wise
Double Descent
Epoch-wise
Double Descent
Time (T)
1k
00
10
1
1
15
30
45
60
Model size (S)
Train Error
Interpolation
Threshold
1k
100
10
1
1
15
30
45
60
Model size (S)
0.80
0.60
0.40
0.20
0.01

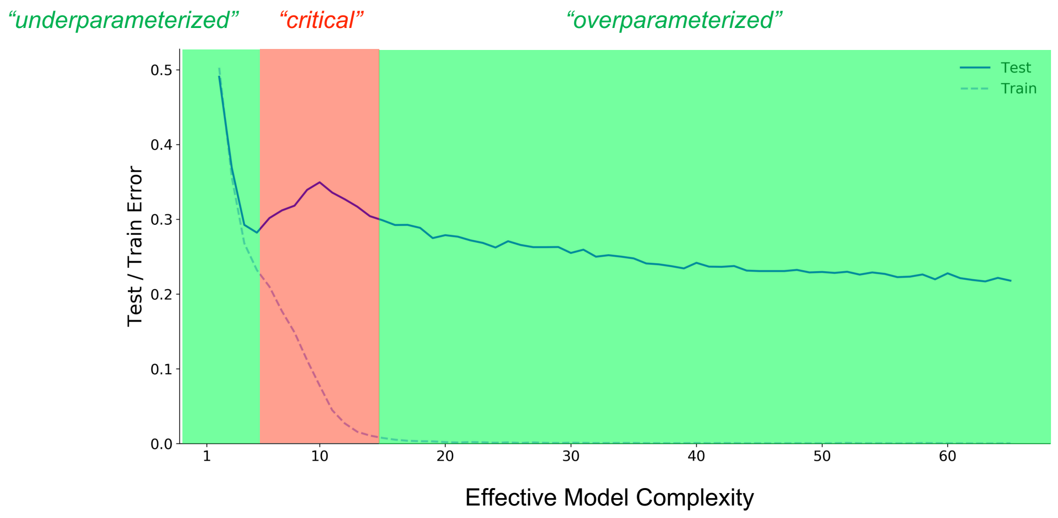
Lessons for Theory

Tight generalization bound must either:

1. Non-monotonic in {data size, model size, train time}

2. Not apply in the “critical regime”

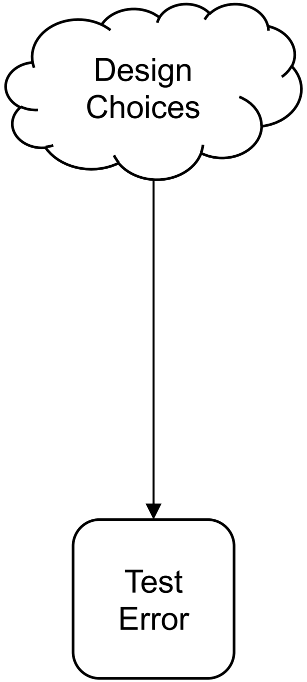
    - Stay in “overparameterized” or “underparameterized”



PART II:
THE DEEP BOOTSTRAP FRAMEWORK

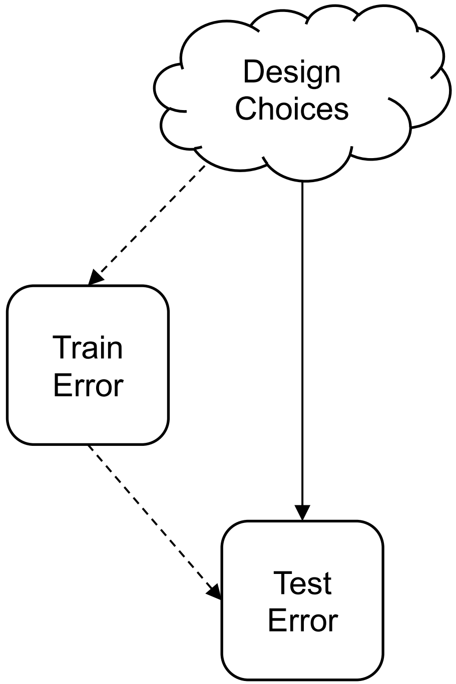
[**N.**, Neyshabur, Sedghi ICLR 2021]

Generalization Frameworks



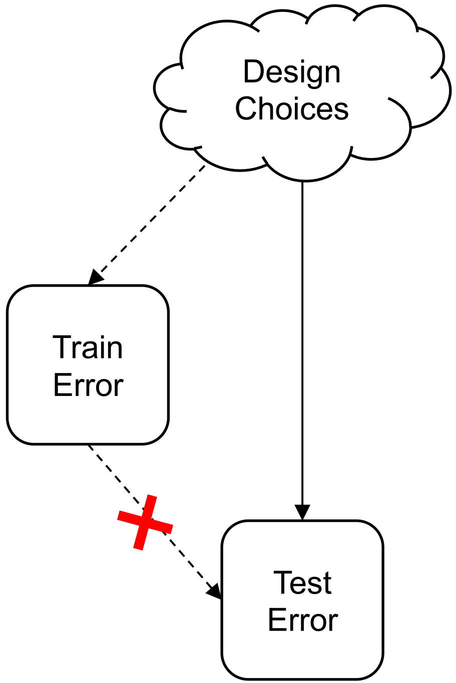
Generalization Frameworks

“Empirical Risk Minimization Framework”



Generalization Frameworks

“Empirical Risk Minimization Framework”



Any “big enough” network can have Train Error ≈ 0

[Zhang et al. 2016]

Our Framework

Main Idea: compare Real World vs. Ideal World

Fix distribution D, architecture $\mathcal{F}$, num samples n.
Then, for all steps $t \in \mathbb{N}$ define:

Real World(n, t)

Our Framework

Main Idea: compare Real World vs. Ideal World

Fix distribution D, architecture $\mathcal{F}$, num samples n.
Then, for all steps $t \in \mathbb{N}$ define:

Real World(n, t)

- Sample train set $S \sim D^n$
- Initialize architecture f_0 from $\mathcal{F}$
- For t steps:
    - Sample minibatch from S
    - Gradient step on minibatch
- Output f_t

Ideal World(t)

Our Framework

Main Idea: compare Real World vs. Ideal World

Fix distribution D, architecture $\mathcal{F}$, num samples n.
Then, for all steps $t \in \mathbb{N}$ define:

Real World(n, t)

- Sample train set $S \sim D^n$
- Initialize architecture f_0 from $\mathcal{F}$
- For t steps:
  - Sample minibatch from S
  - Gradient step on minibatch
- Output f_t

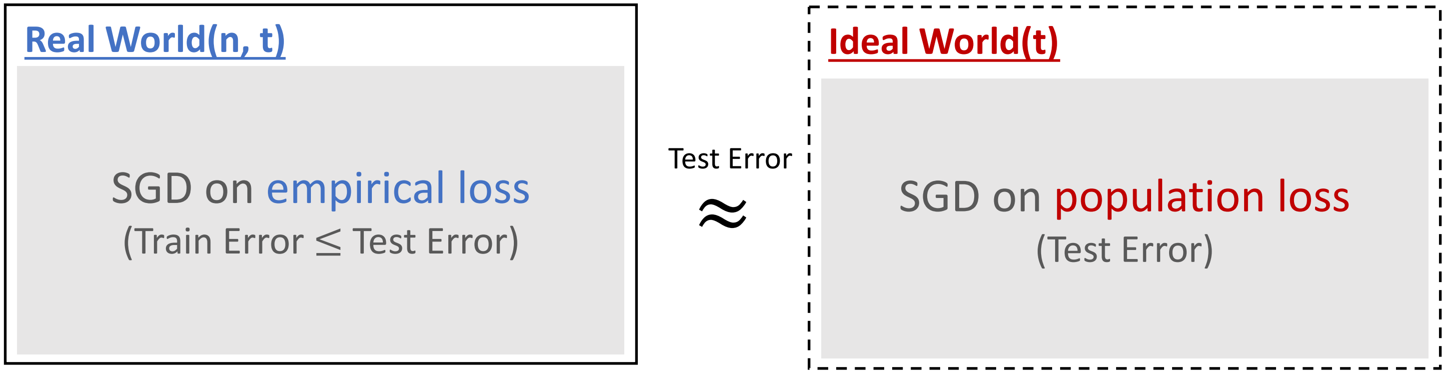
Ideal World(t)

- Initialize architecture f_0 from $\mathcal{F}$
- For t steps:
  - Sample minibatch from D
  - Gradient step on minibatch
- Output f_t^{iid}

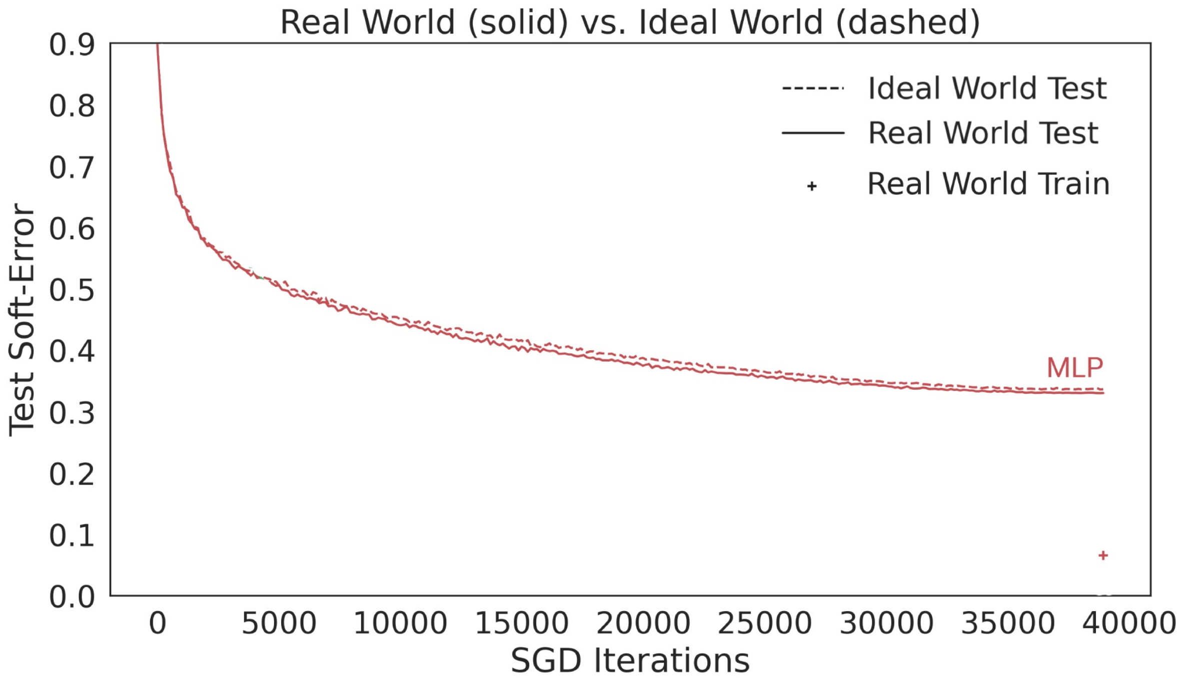
Our Framework

Main Idea: compare Real World vs. Ideal World

Fix distribution D, architecture $\mathcal{F}$, num samples n.
Then, for all steps $t \in \mathbb{N}$ define:



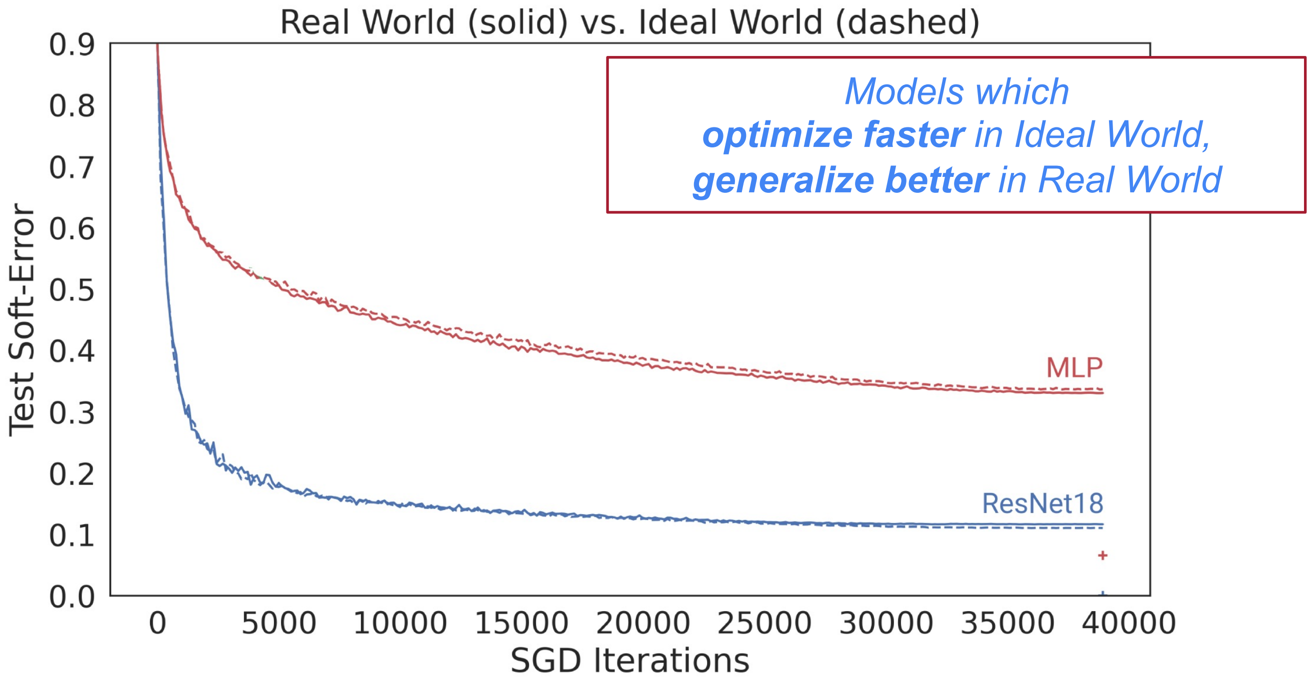
Experiment



Real World: 50K samples, 100 epochs.

Ideal World: 5M samples, 1 epoch.

Experiment



Real World: 50K samples, 100 epochs.

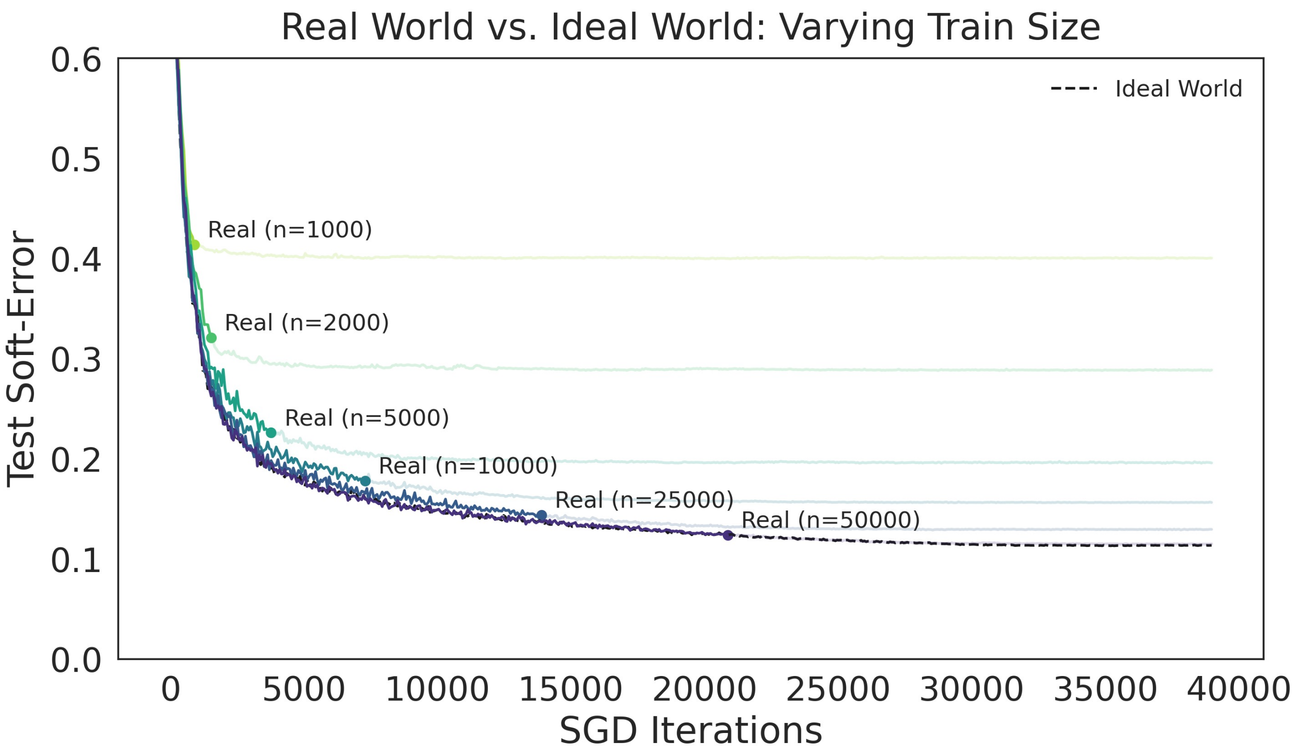
Ideal World: 5M samples, 1 epoch.

$T(n)$: “Stopping time”. Real World time to converge on **n** samples (< 1% train error)

Deep Bootstrap:

$$\forall t \leq T(n): \qquad \text{RealWorld}(n, t) \approx_{\epsilon} \text{IdealWorld}(t)$$

*“SGD on deep nets behaves similarly whether trained on **re-used samples** or **fresh samples** ...up until the Real World has converged”*



Real World vs. Ideal World: Varying Train Size
Ideal World
Real (n=1000)
Real (n=2000)
Real (n=5000)
Real (n=10000)
Real (n=25000)
Real (n=50000)
Test Soft-Error
SGD Iterations
0.0
0.1
0.2
0.3
0.4
0.5
0.6
0
5000
10000
15000
20000
25000
30000
35000
40000

Deep Bootstrap:

$$\text{FinalError}(n) \approx_{\epsilon} \text{IdealWorld}\big(T(n)\big)$$

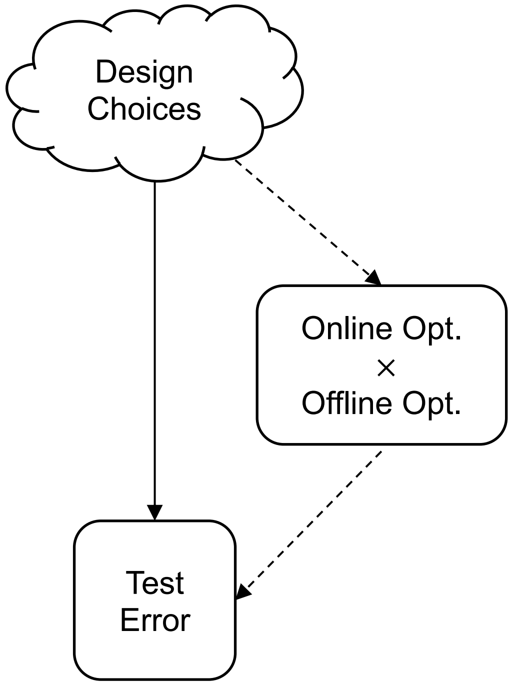
$T(n)$: Time to converge on **n** samples

LHS: Generalization

RHS: Optimization
(Online optimization & Empirical Optimization)

Deep Bootstrap:

$$\text{FinalError}(n) \approx_\epsilon \text{IdealWorld}\big(T(n)\big)$$

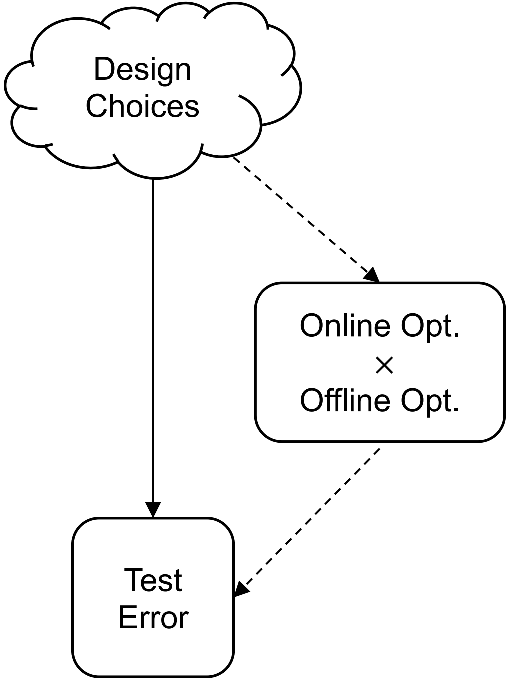


Empirically verified for varying:

- Architectures
- Model size
- Data size
- Optimizers (SGD/Adam/etc)
- Pretraining
- Data-augmentation
- Learning rate
- …

Deep Bootstrap:

$$\text{FinalError}(n) \approx_{\epsilon} \text{IdealWorld}\big(T(n)\big)$$



Good design choices:

1. **Optimize quickly** in online setting
(large models, skip-connections, pretraining,…)

2. **Don't optimize too** quickly on finite samples
(regularization, data-aug,…)

Alternate Perspectives

Generalization Perspective:	→	**Optimization Perspective:**
"ConvNets *generalize better* than MLPs"	→	"ConvNets *optimize faster* than MLPs"
"Pretraining helps *generalization*"	→	"Pretraining helps *optimization"* (a la *preconditioning)*

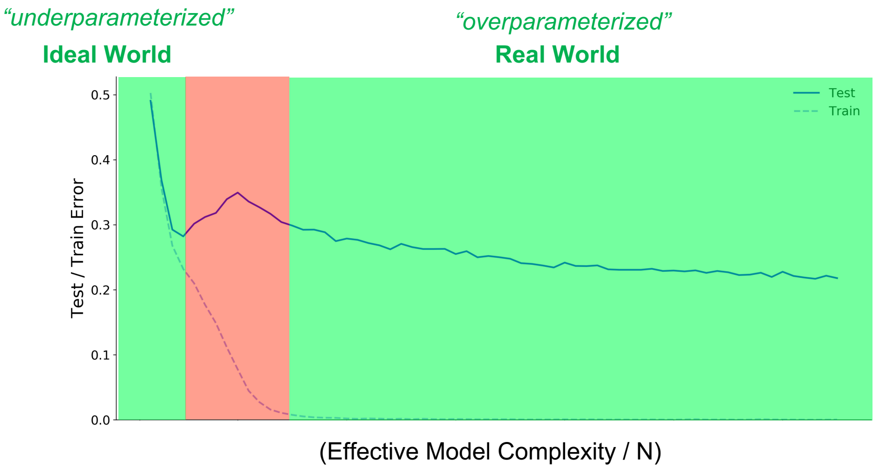
Significance

Assuming bootstrap claim: Reduces *generalization* to *optimization*.

Hope: Refocus attention on online optimization aspects of deep learning

Connects *overparametrized* and *underparameterized* regimes:

Models which fit their train sets “behave like” models trained on infinite data



A Practical Mystery

Two regimes in practice:

1. Effectively infinite data (e.g. train on internet, 1B+ samples)

   want architectures which optimize quickly

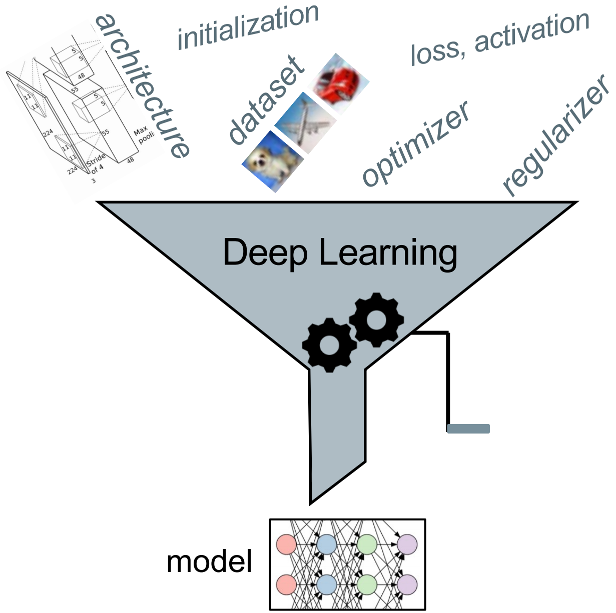
2. Small finite data (e.g. 50K samples)

   want architectures which generalize well

Mystery: Why do we use the same architectures in both regimes?

Deep Bootstrap: Not a coincidence…

Significance



Many arbitrary choices in deep learning.

Want theory of generalization that is *not sensitive* to irrelevant choices.

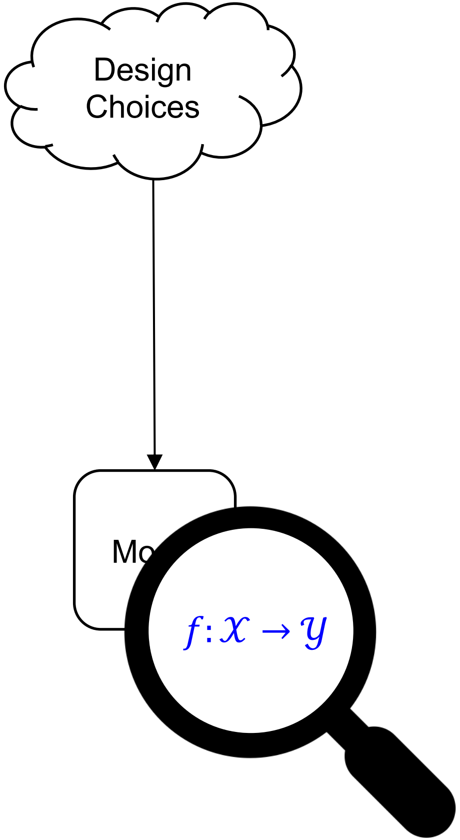
<u>Deep Bootstrap:</u>

“Any choice that works for online optimization will work for offline generalization.”

PART III
DISTRIBUTIONAL GENERALIZATION: A NEW KIND OF GENERALIZATION

(warning: technical & imprecise)

[**N.**, Bansal 2020]

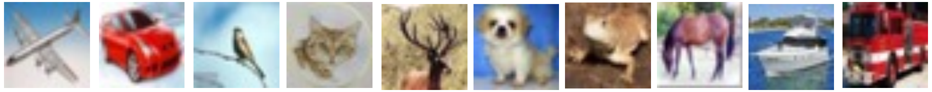


Suppose test error of f = 40%

Many such f! Which one did we get?

Experiment

Type: 0 1 2 3 … 9



Distribution on (x, y):

x ~ { random image, random type }

$y|x$ ~ Bernoulli(type(x) / 10)

Train Set (x, y)

y	plane	auto-mobile	bird	cat	deer	dog	frog	horse	ship	truck
0	0.10	0.09	0.08	0.07	0.06	0.05	0.04	0.03	0.02	0.01
1	0.00	0.01	0.02	0.03	0.04	0.05	0.06	0.07	0.08	0.09

Sample from this distribution.

Train a neural-net to predict $f: \mathcal{X} \rightarrow \mathcal{Y}$

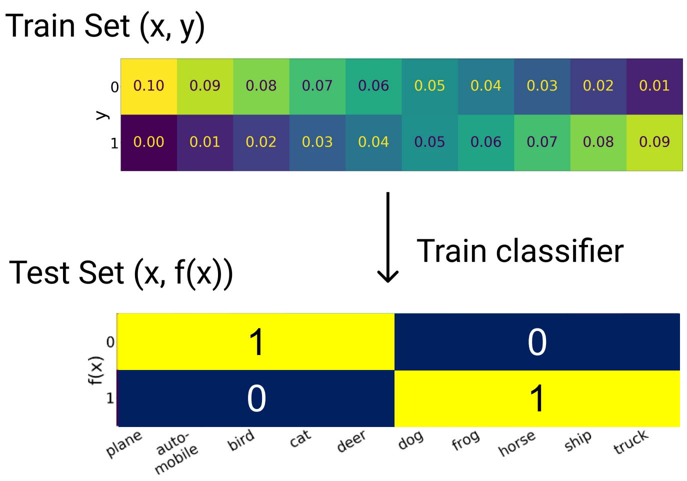
Q: What happens at test time?

A: ~Same distribution!

We use a method for **classification.**
We **don't get** a good classifier: high test error!

We get an approximate **sampler**:

$$f(x) \sim p(y\,|x)$$



<u>Happens for:</u>

- Interpolating **neural networks**
- Interpolating **kernel regressors**
- Interpolating **decision trees**

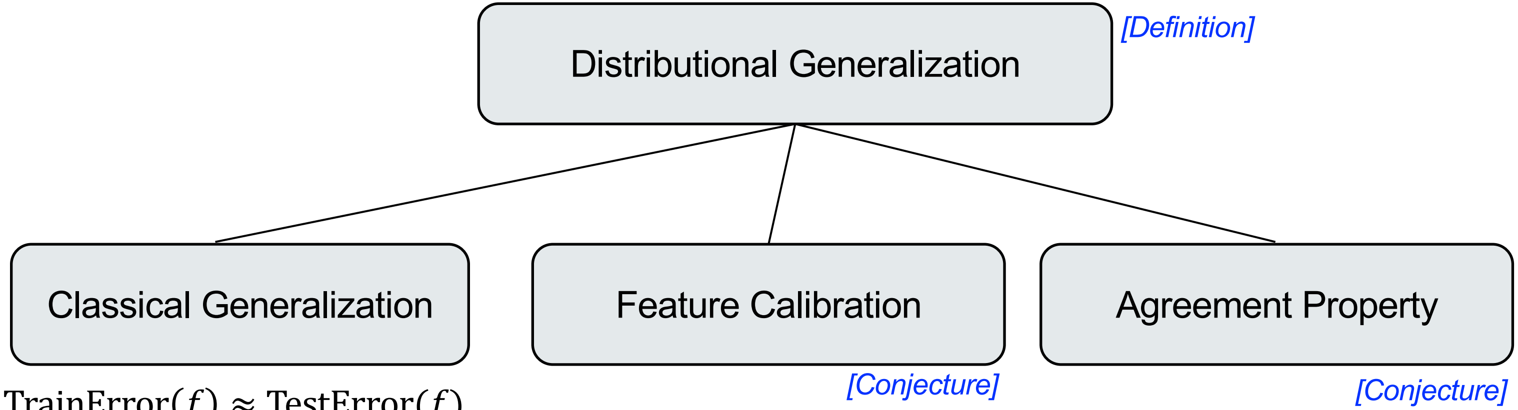
} *Best thought of as samplers.*

Classical generalization is insufficient language

Main Idea:

“ Test and train outputs of classifiers are close as ***distributions*** *”*

$$(x, f(x))_{x \in \mathrm{TrainSet}} \approx (x, f(x))_{x \in \mathrm{TestSet}}$$

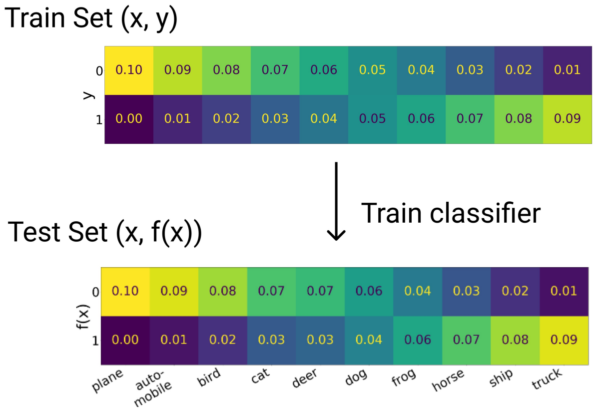


$$\mathrm{TrainError}(f) \approx \mathrm{TestError}(f)$$

Roadmap

We want to formalize the closeness:

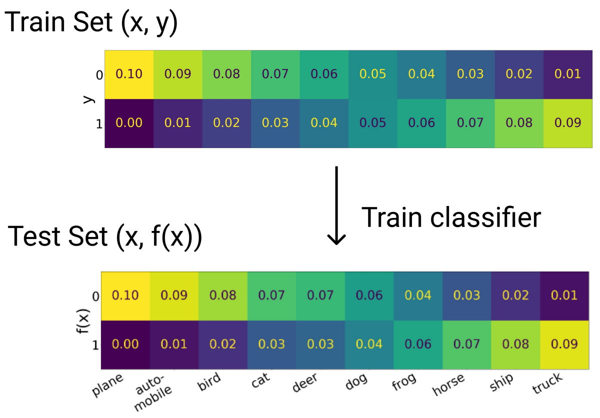
$$(x, f(x))_{x,y\sim D} \approx (x, y)_{x,y\sim D}$$



Roadmap

We want to formalize the closeness:

$$(x, f(x)) \approx (x, y)$$



<ins>Claim</ins>: For some partitions $L: \mathcal{X} \to [M]$,

$$(L(x), f(x)) \approx_{TV} (L(x), y)$$

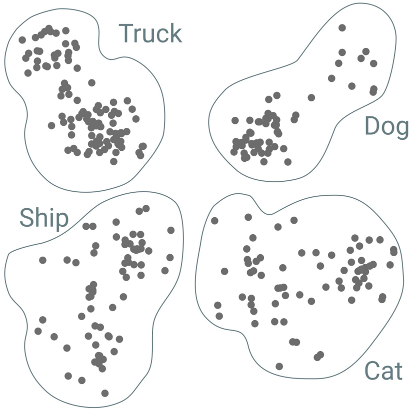
x is "coarsened" into a partition L(x)

Which partitions?

- Depends on architecture, distribution, num samples…
- Intuitively, "partitions which can be learnt"

Distinguishable Feature

Given: Training procedure $\mathcal{F}$, distribution $(x, y) \sim \mathcal{D}$, num train samples n.



Defn (informal): *A* **distinguishable feature** *is a labeling $L: \mathcal{X} \to [M]$ of the domain that is* ***learnable*** *to high test accuracy, from samples*

$$\left(x_i, L(x_i)\right)_{x_i \sim D}$$

eg: L: X → {cat, dog, plane…}
is a distinguishable feature for ResNets
with n=50K samples.

…for training-procedures $\mathcal{F}$, with n samples from $\mathcal{D}$.

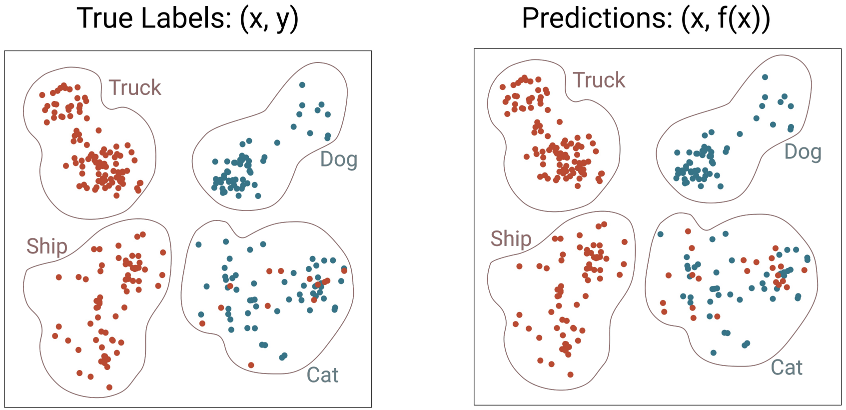
Main Conjecture: Feature Calibration

Conjecture: *For all natural* $\mathcal{D}, \mathcal{F}, n$:

For all *distinguishable features* L:

$$(L(x), f(x)) \approx_{TV} (L(x), y)$$

Test Set



Main Conjecture: Feature Calibration

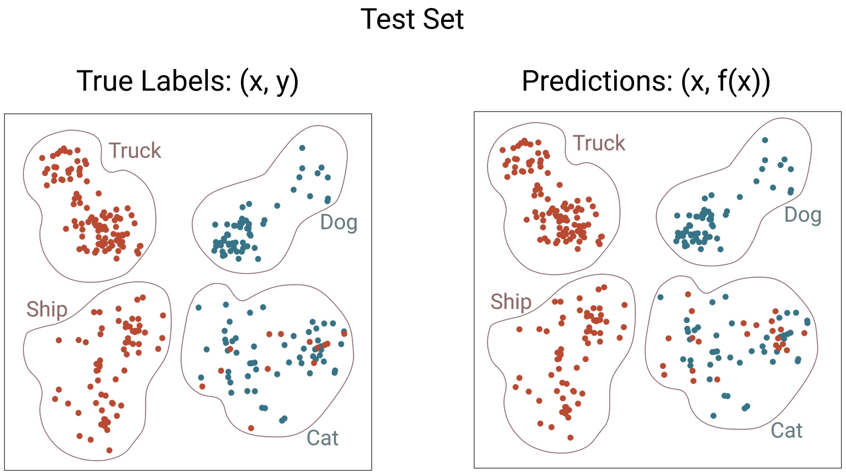
Conjecture: *For all natural* $\mathcal{D}, \mathcal{F}, n$:

*"Marginal distributions of **f(x)** and **y** match, when conditioned on any distinguishable-feature **L**"*

Eg:

$$p(f(x) \mid x \in CAT) \approx p(y \mid x \in CAT)$$

"subgroup calibration property"



Example Application

ImageNet: Image classification. 1000-classes, 116 dogs.

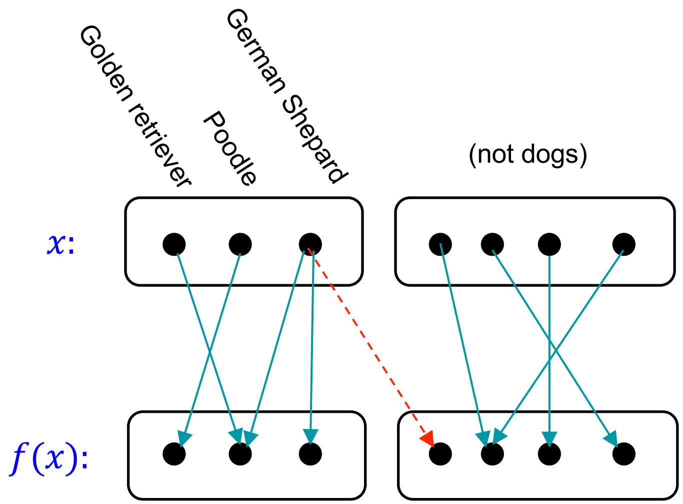
ImageNet is “hard”: AlexNet (f) gets 56% test accuracy.

Does it at least classify dogs as *some type* of dog?

- Yes! (98% acc). Not 56% accuracy on all groups.

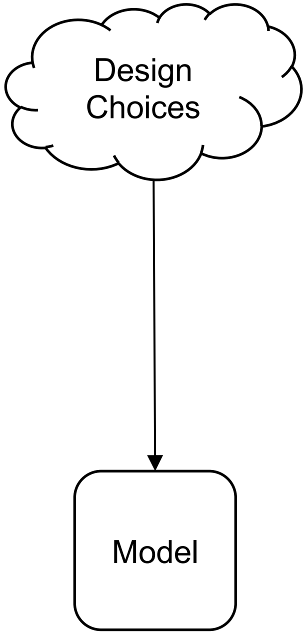
- Predicted by our conjecture

- *Even “bad” classifiers (w.r.t. test error), can have “good” hidden structure*



CONCLUSIONS

Conclusions



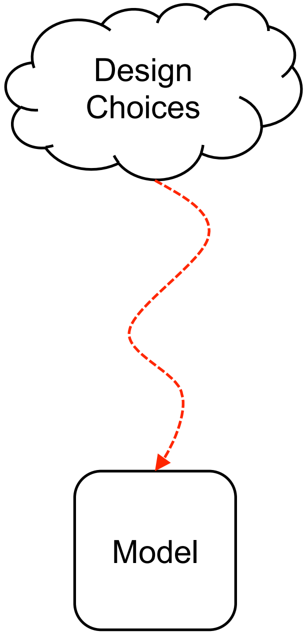
Several ways to understand map
between what we *do* & what we *get:*

Deep Double Descent:
- Definition of “over/under-parameterized regimes”
- Map poorly behaved in “critical regime”

Deep Bootstrap:
- Factorize map via (online × offline) optimization
- Connection between over/underparam regimes

Distributional Generalization:
- Structural properties of model, beyond test error
- Separation between over/underparam regimes

Conclusions



Several ways to understand map
between what we *do* & what we *get:*

Deep Double Descent:

- Definition of “over/under-parameterized regimes”
- Map poorly behaved in “critical regime”

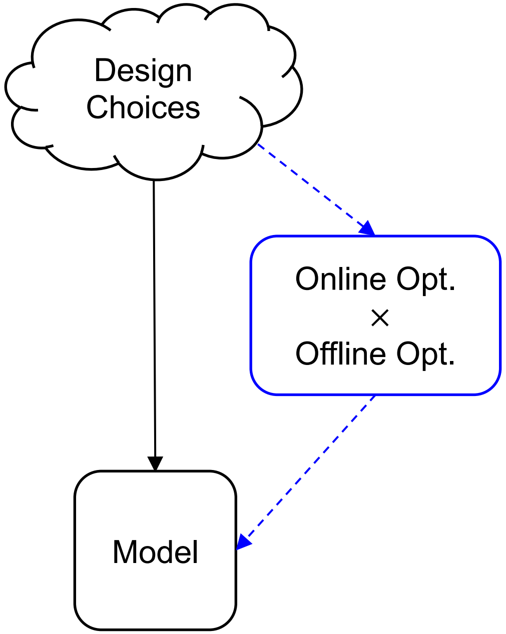
Deep Bootstrap:

- Factorize map via (online × offline) optimization
- Connection between over/underparam regimes

Distributional Generalization:

- Structural properties of model, beyond test error
- Separation between over/underparam regimes

Conclusions



Several ways to understand map between what we *do* & what we *get:*

<u>Deep Double Descent</u>:

- Definition of "over/under-parameterized regimes"
- Map poorly behaved in "critical regime"

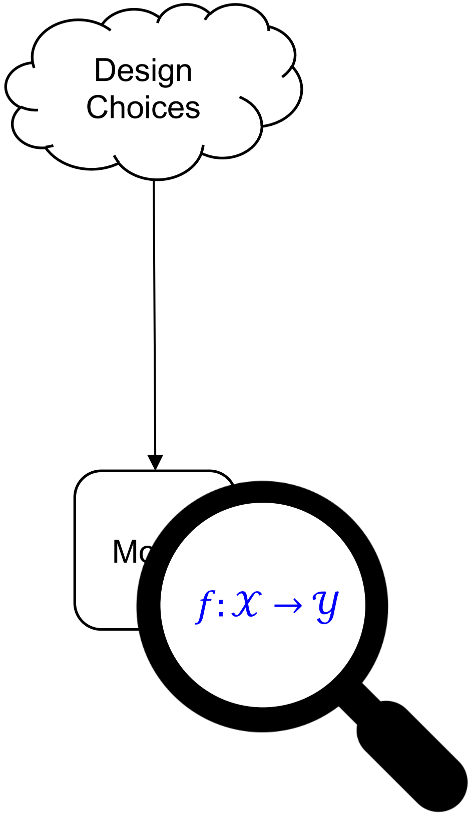
<u>Deep Bootstrap:</u>

- Factorize map via (online × offline) optimization
- Connection between over/underparam regimes

<u>Distributional Generalization:</u>

- Structural properties of model, beyond test error
- Separation between over/underparam regimes

Conclusions



Several ways to understand map
between what we *do* & what we *get:*

Deep Double Descent:

- Definition of “over/under-parameterized regimes”
- Map poorly behaved in “critical regime”

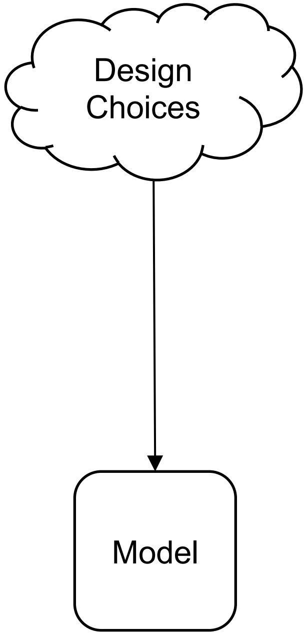
Deep Bootstrap:

- Factorize map via (online $\times$ offline) optimization
- Connection between over/underparam regimes

Distributional Generalization:

- Structural properties of model, beyond test error
- Separation between over/underparam regimes

Conclusions



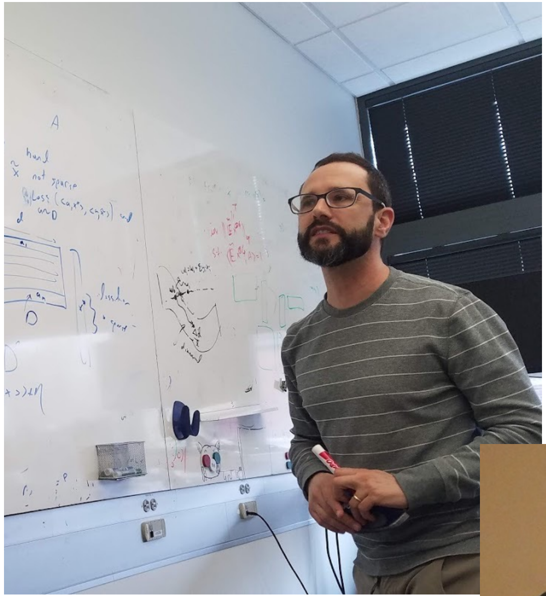
<u>Methodology:</u>

Experiments → New behaviors → Conjectures

<u>Hope:</u> Results weave into general theory of learning

What can we learn (deeply, or otherwise)?

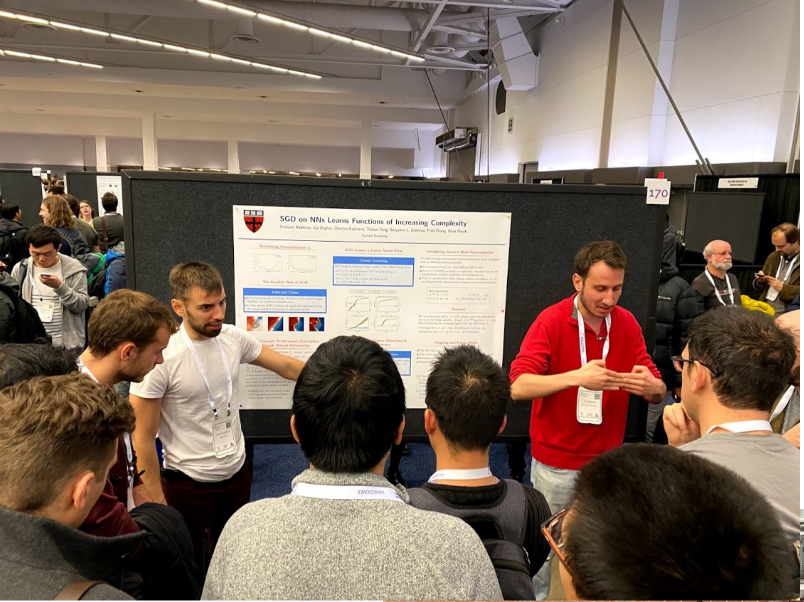
ACKNOWLEDGEMENTS

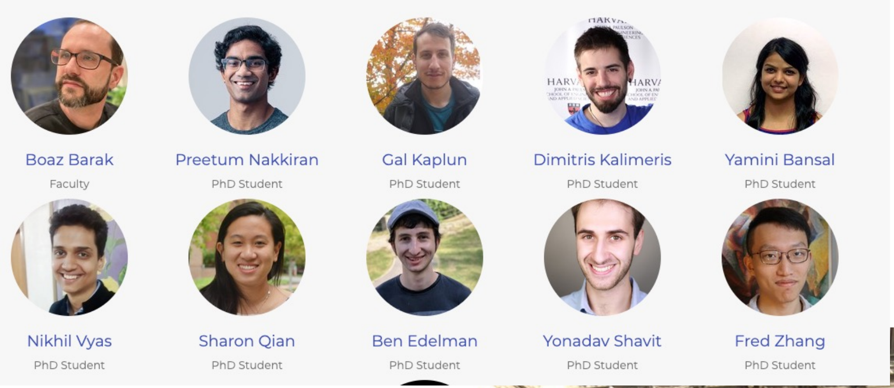


Advisors: Boaz & Madhu

Harvard Theory Group

& Friends





ML Theory Group

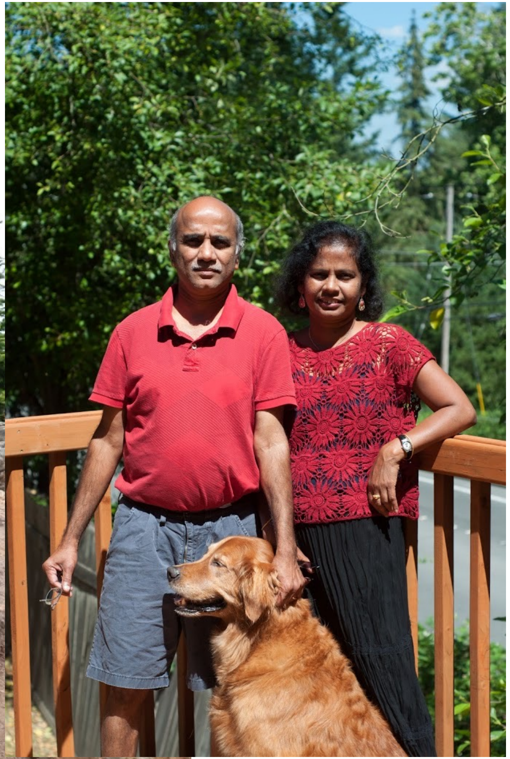
All my teachers:

Salil Vadhan, Jelani Nelson, Scott Kominers,…
Luca Trevisan, Sangam Garg, Anant Sahai,…
Peter Saxby, John Frank,…

Senior Collaborators:

Ilya Suskever, Chris Olah, Sham Kakade, Tengyu Ma, Jacob Steinhardt, Behnam Neyshabur, Hanie Sedghi

My Family



END