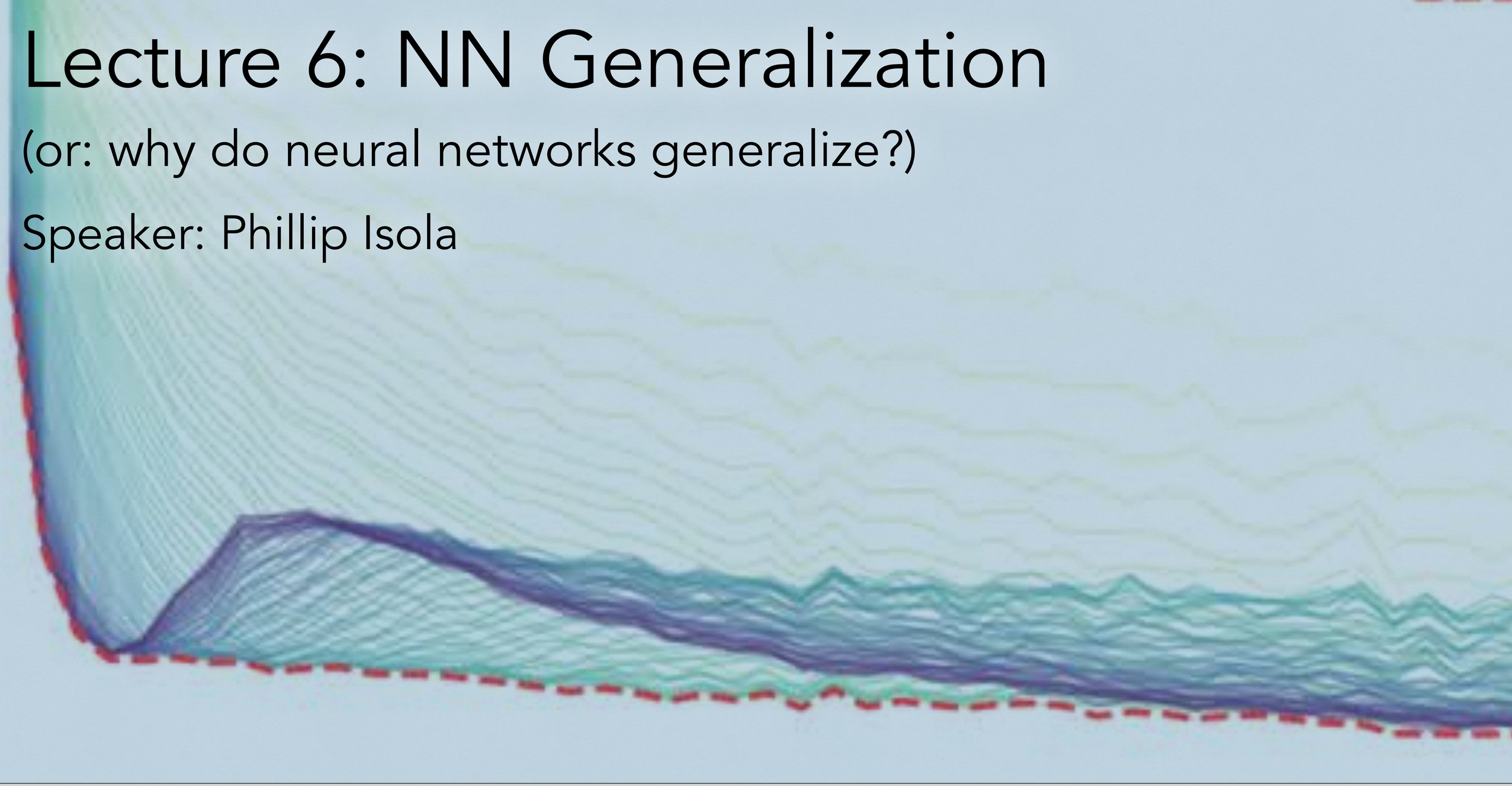
Lecture 6: NN Generalization

(or: why do neural networks generalize?)

Speaker: Phillip Isola



Outline

- What is generalization
- Do deep nets generalize?
- Deep nets violate certain classical generalization theory
- Why do they generalize? → What are their inductive biases?

Approximation vs Generalization

- How well will our trained neural network do on new data?
- We minimize the *empirical risk*: $\widehat{\mathcal{R}}(\theta) = \frac{1}{N}\sum_{i=1}^{N} \mathcal{L}(f_\theta(\mathbf{x}_i), \mathbf{y}_i)$
- We actually want the *population risk (test error)* to be small:

$$\mathcal{R}(\theta) = \mathbb{E}_{(\mathbf{x},\mathbf{y})\sim\mathcal{P}}\, \mathcal{L}(f_\theta(\mathbf{x}), \mathbf{y})$$

Important questions:

Approximation: what is the best $\mathscr{R}(\theta^*)$ we can achieve with our model?

Optimization: how well are we minimizing $\widehat{\mathscr{R}}(\theta)$?

Generalization: how different is $\widehat{\mathscr{R}}(\theta)$ from $\mathscr{R}(\theta)$?

Intuitive ideas about generalization

Bad data

Suppose we want to train a cats vs dogs classifier.

But our training data only contains cats.

Can we do it? ….. No

Bad models: the filing cabinet

Every time we see a new training point (x,y), we put it in the cabinet.

```
def predict(x):
  if x in cabinet:
    return cabinet[x]
  else:
    return 0
```

What will the approximation error be?

What will the generalization error be?

Bad models: Paul the octopus

The Amazing Tale of Paul the Psychic Octopus: Germany's World Cup Soothsayer

NEVER FORGET

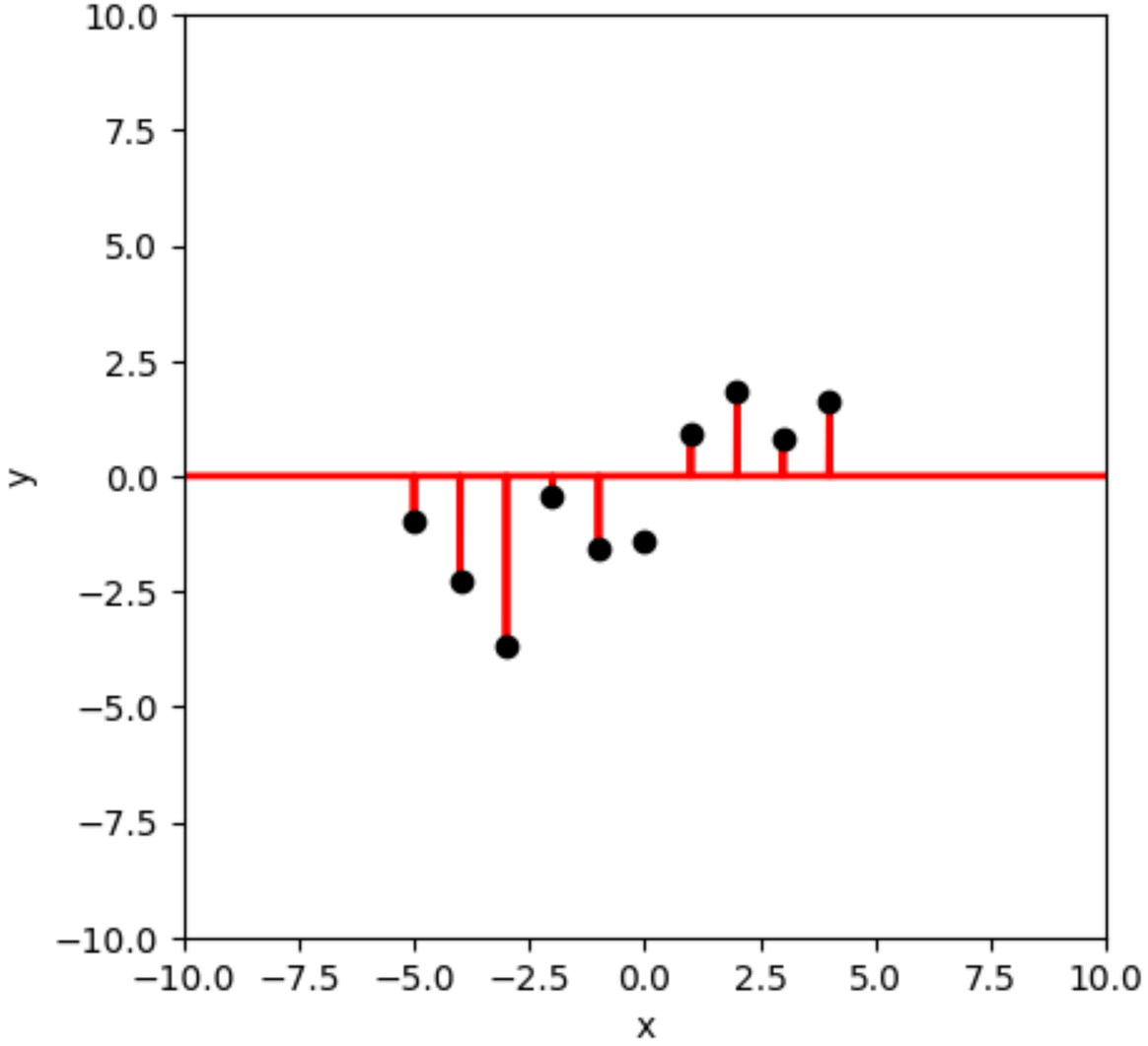
Sure, Germany is back in the World Cup final. But it'll have to beat Argentina without Paul, the cephalopod that correctly predicted the results of all eight (!) German matches last go-around.

Emily Shire | Updated Apr. 14, 2017 3:21PM EDT
Published Jul. 12, 2014 12:00AM EDT

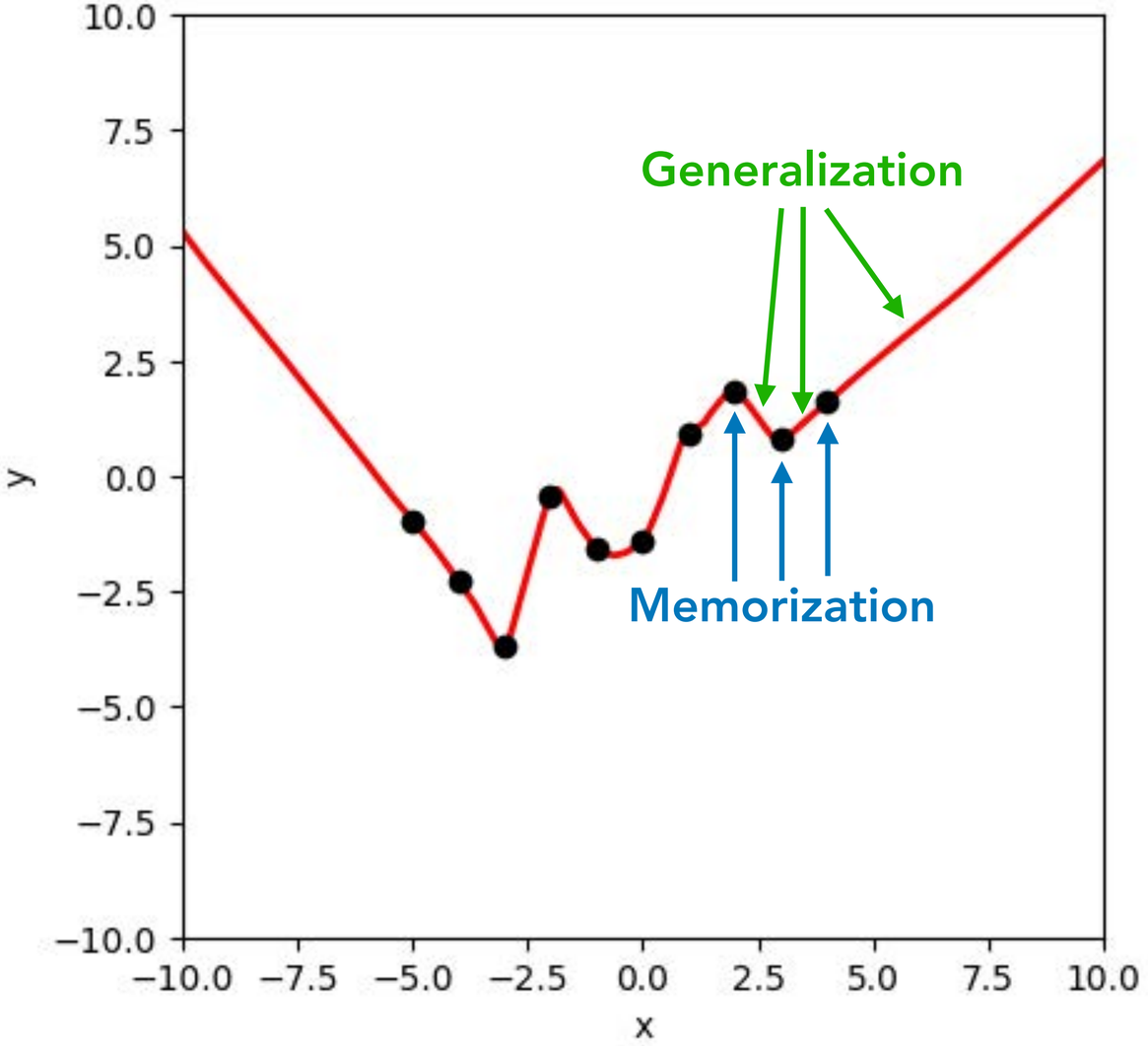
Would you trust Paul?

Do deep nets generalize?

Filing cabinet



3-layer relu-MLP



Do deep nets generalize?

```
def predict(x):
  if x in cabinet:
    return cabinet[x]
  else:
    return 0
```

What about much big fancy modern nets?

1. Suppose an LLM is a filing cabinet (no generalization)
2. How big does the cabinet need to be?
3. Simple counting experiment:
   - Sample random sequences of **n** words from a vocab of size **m**
   - There are **m^n** such sequences
   - Input a set of such sequences into your favorite model that answers some question about a text input; estimate percent of time, **p**, that the output is correct (or at least “non-zero”)
   - Filing cabinet (and training data) needs to be size **s** = **p*m^n**

Vocab:

```
fruits = [
    'Apple', 'Banana', 'Orange', 'Grapes', 'Mango', 'Peach', 'Pear', 'Pineapple', 'Strawberry', 'Blueberry',
    'Raspberry', 'Watermelon', 'Cantaloupe', 'Cherry', 'Coconut', 'Fig', 'Guava', 'Kiwi', 'Lemon', 'Lime',
    'Lychee', 'Mandarin', 'Nectarine', 'Papaya', 'Passion Fruit', 'Plum', 'Pomegranate', 'Tangerine', 'Dragonfruit', 'Durian'
]
```

n = 30, **m** = 10

prompt: “How many citrus fruits are in this list?”

My estimated **p** for GPT-4o: 1.0 (5/5)

s = 100 trillion

How many citrus fruits are in this list: Cantaloupe, Fig, Raspberry, Blueberry, Papaya, Lemon, Tangerine, Cherry, Grapes, Orange

In the list you provided, the citrus fruits are:

- Lemon
- Tangerine
- Orange

So, there are **3 citrus fruits** in the list.

Created with ChatGPT.

n = 30, **m** = 10

prompt: Draw these fruits

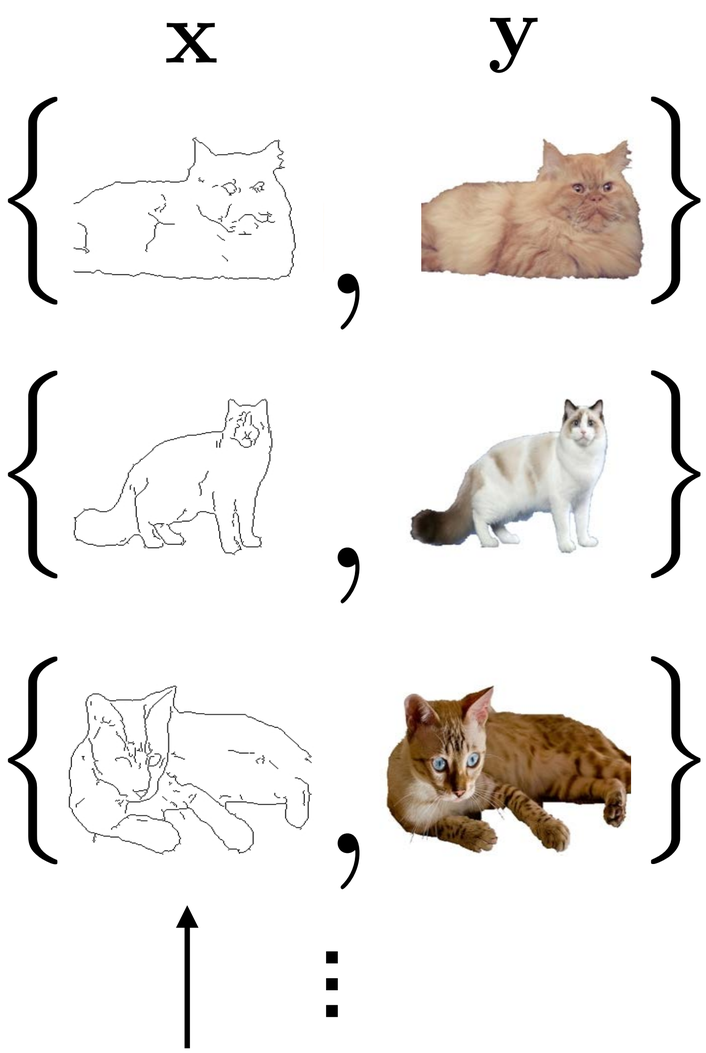
My estimated **p** for GPT-4o: 0.125 (1/8)

s = 12.5 trillion

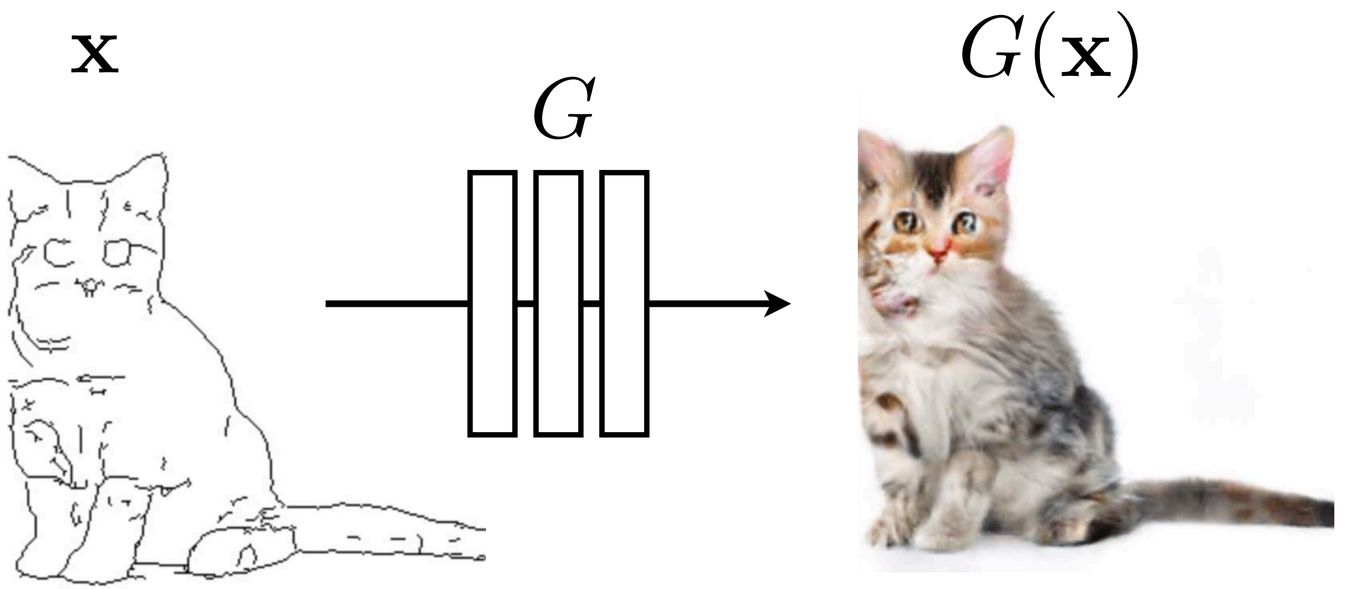
Draw these fruits: Blueberry, Mandarin, Kiwi, Cantaloupe, Peach, Dragonfruit, Coconut, Cherry, Passion Fruit, Lychee

Created with ChatGPT.

Training data

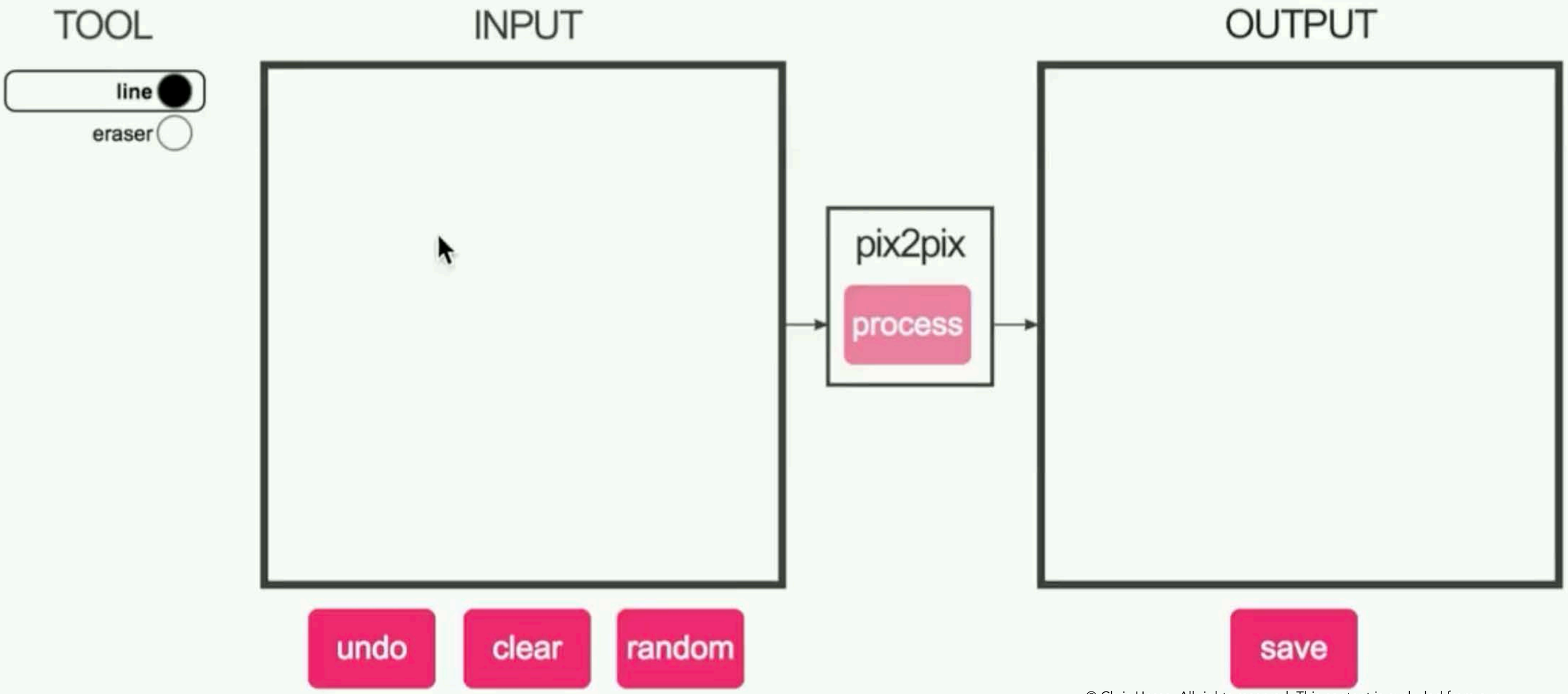


[HED: Xie & Tu, 2015]

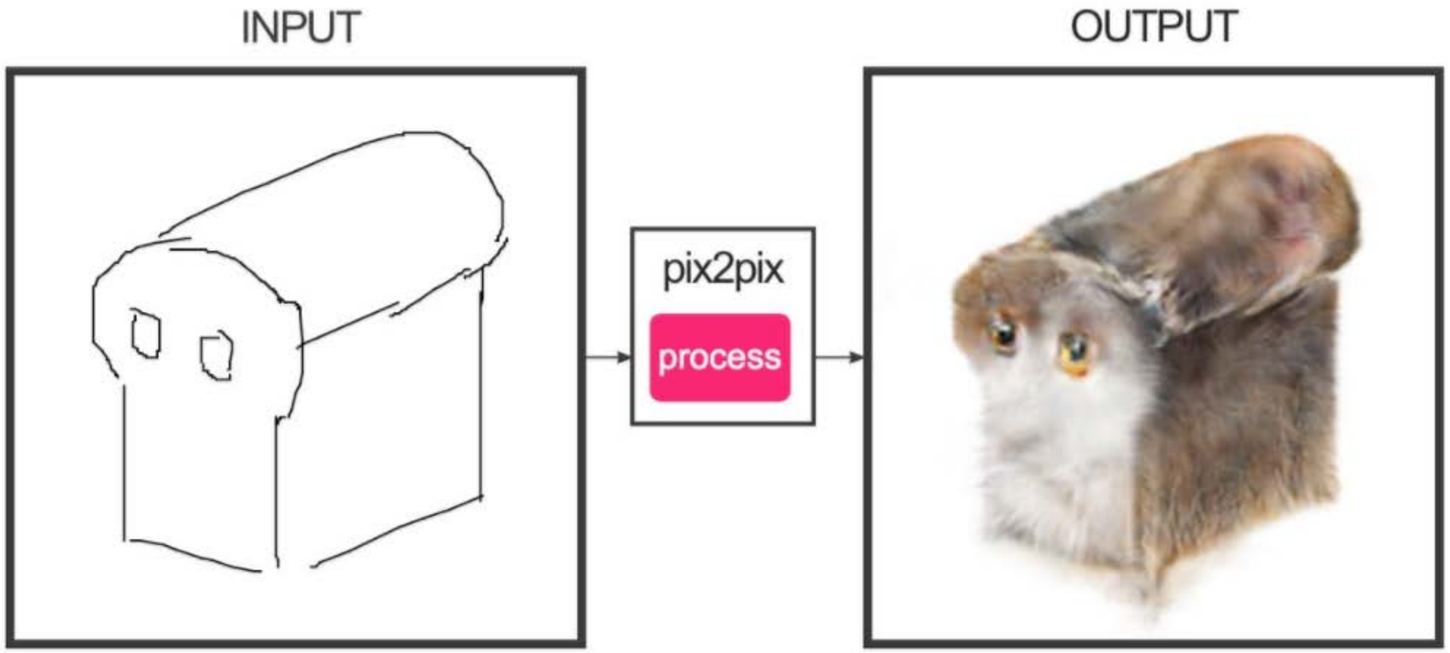


[pix2pix: Isola, Zhu, Zhou, Efros, 2017]

edges2cats



[#edges2cats: Chris Hesse, 2017]



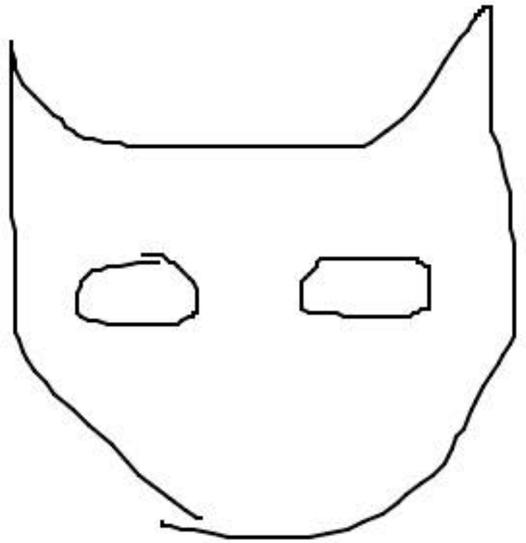
Ivy Tasi @ivymyt

Vitaly Vidmirov @vvid

Images created using pix2pix and edges2cats.

INPUT

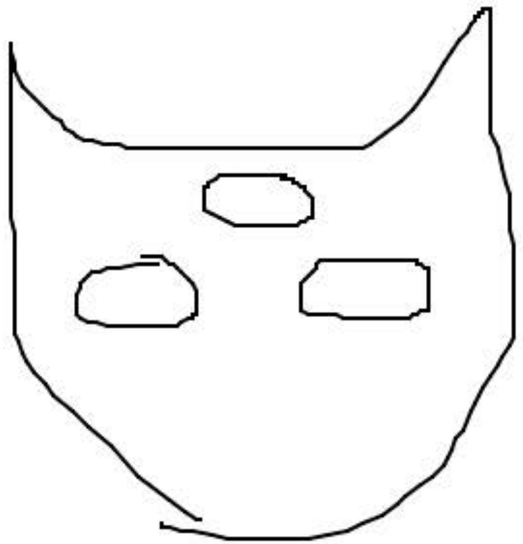
OUTPUT



pix2pix

process

INPUT



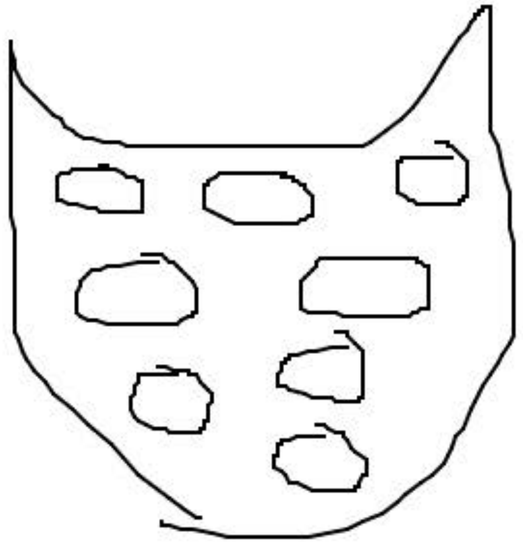
pix2pix

process

OUTPUT

INPUT

OUTPUT

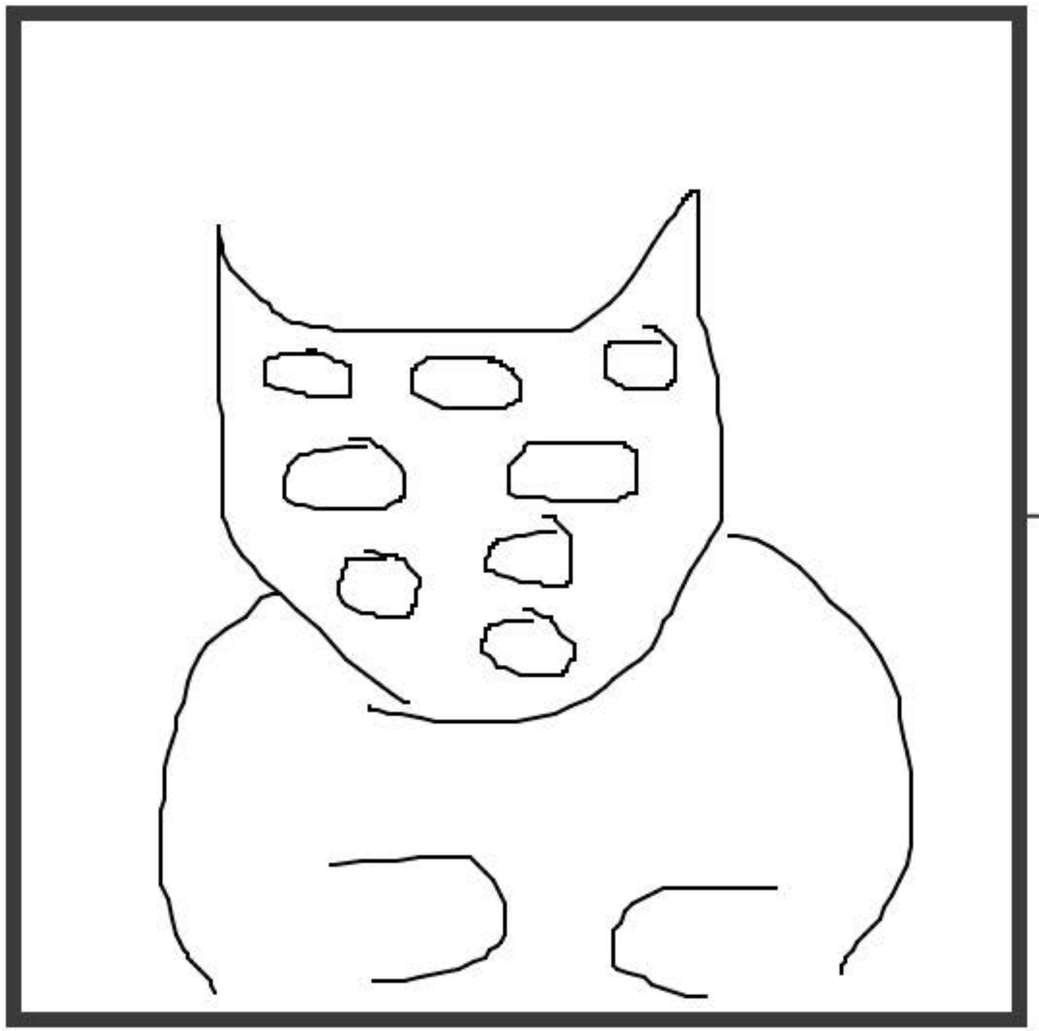


pix2pix

process

INPUT

OUTPUT

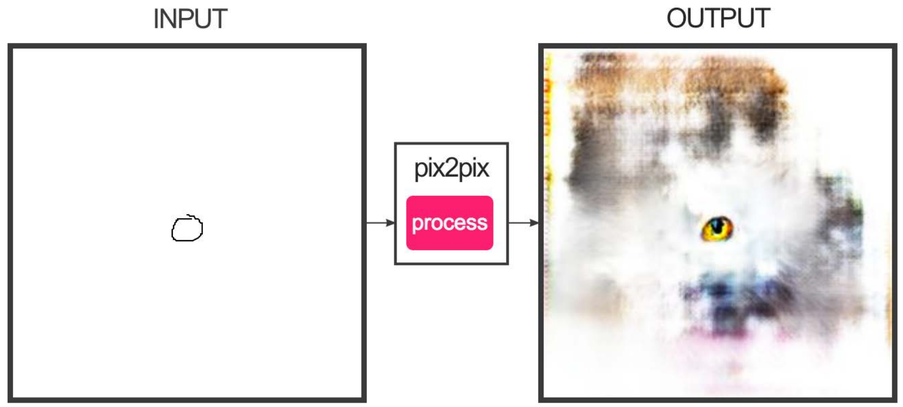


pix2pix

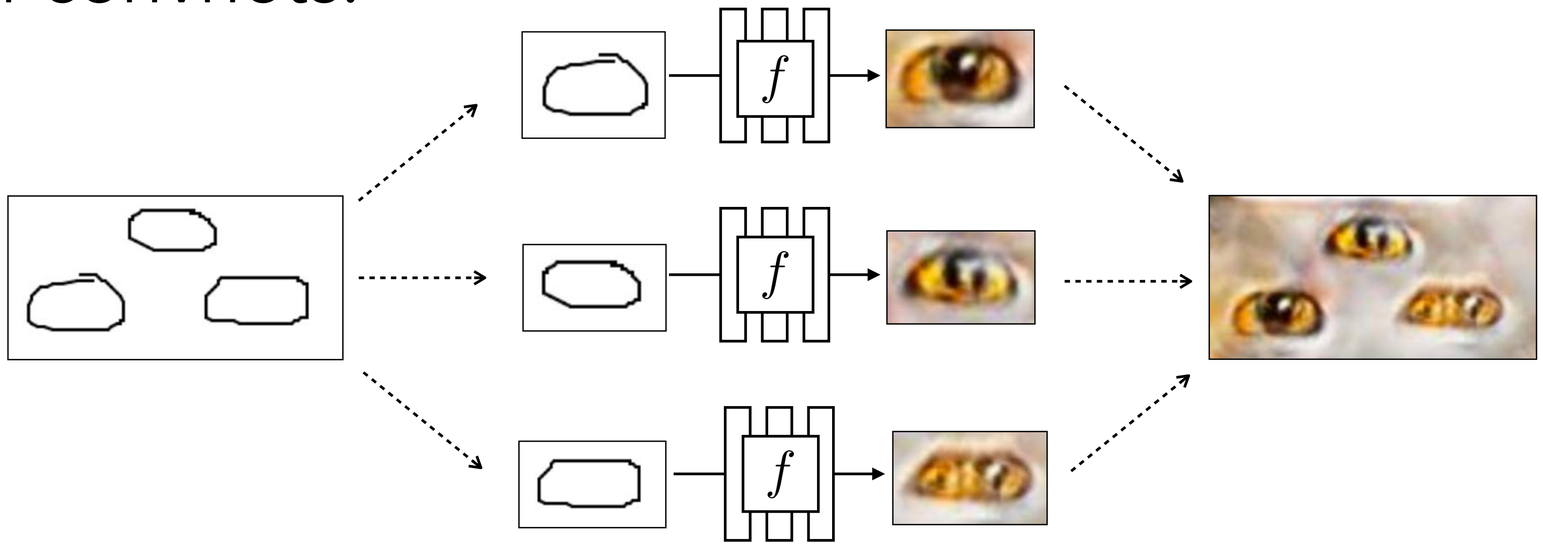
process

Inductive bias toward simple modular processing

"When I see an oval, draw an eye."



Why? Bias of convnets!



Generalization theory

Occam's Razor

The simplest model that fits the data will generalize best.

How do we measure "simpler"?

Image is in the public domain.

How should we measure model complexity?

Theory answer:

The shortest program that fits the data is the one that will generalize best.

Intractable, but good to keep in mind…

[Solomonoff 1964]

Review: Overfitting and the bias-variance tradeoff

test error = train error + (test error - training error)

bias ***variance***

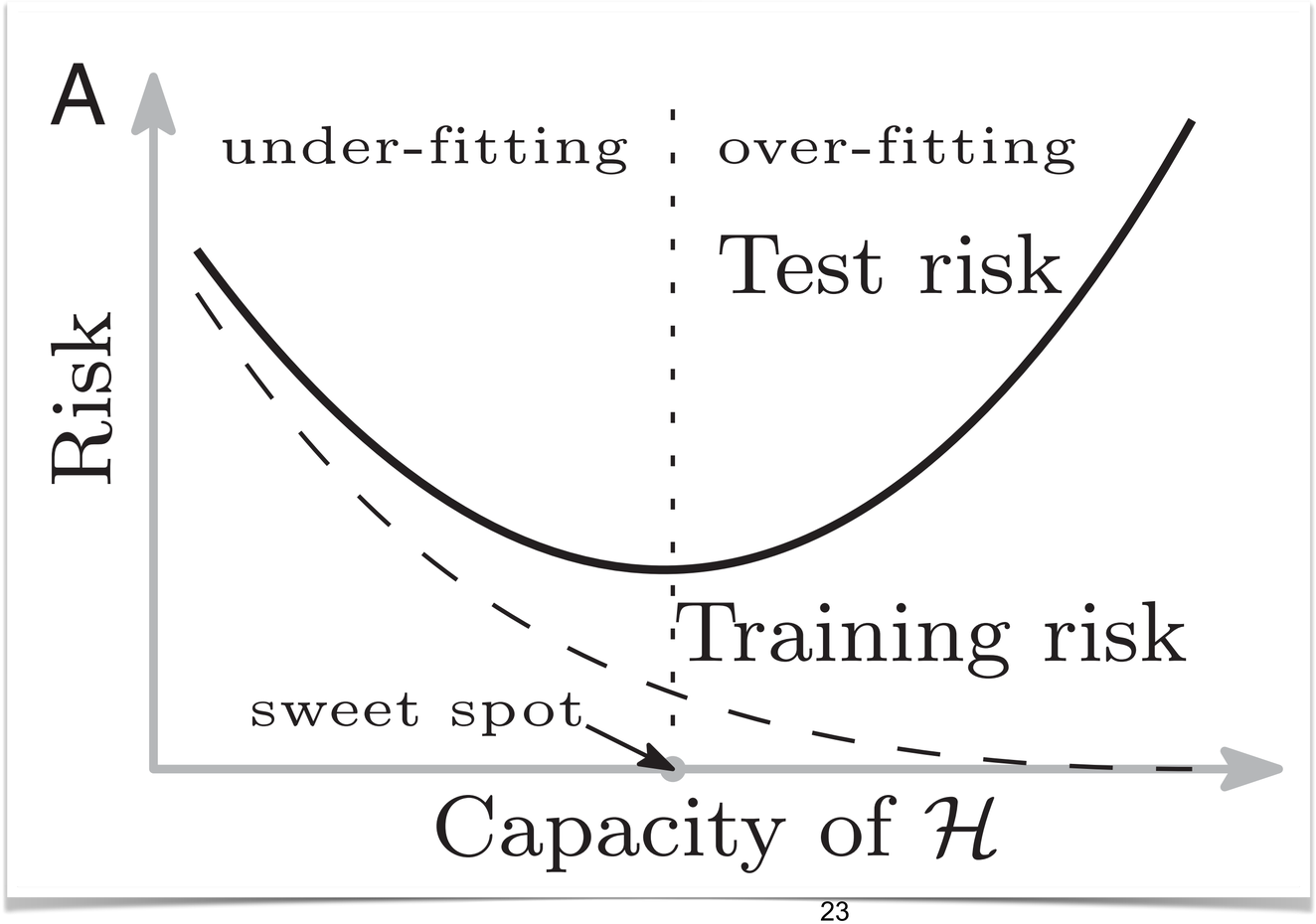
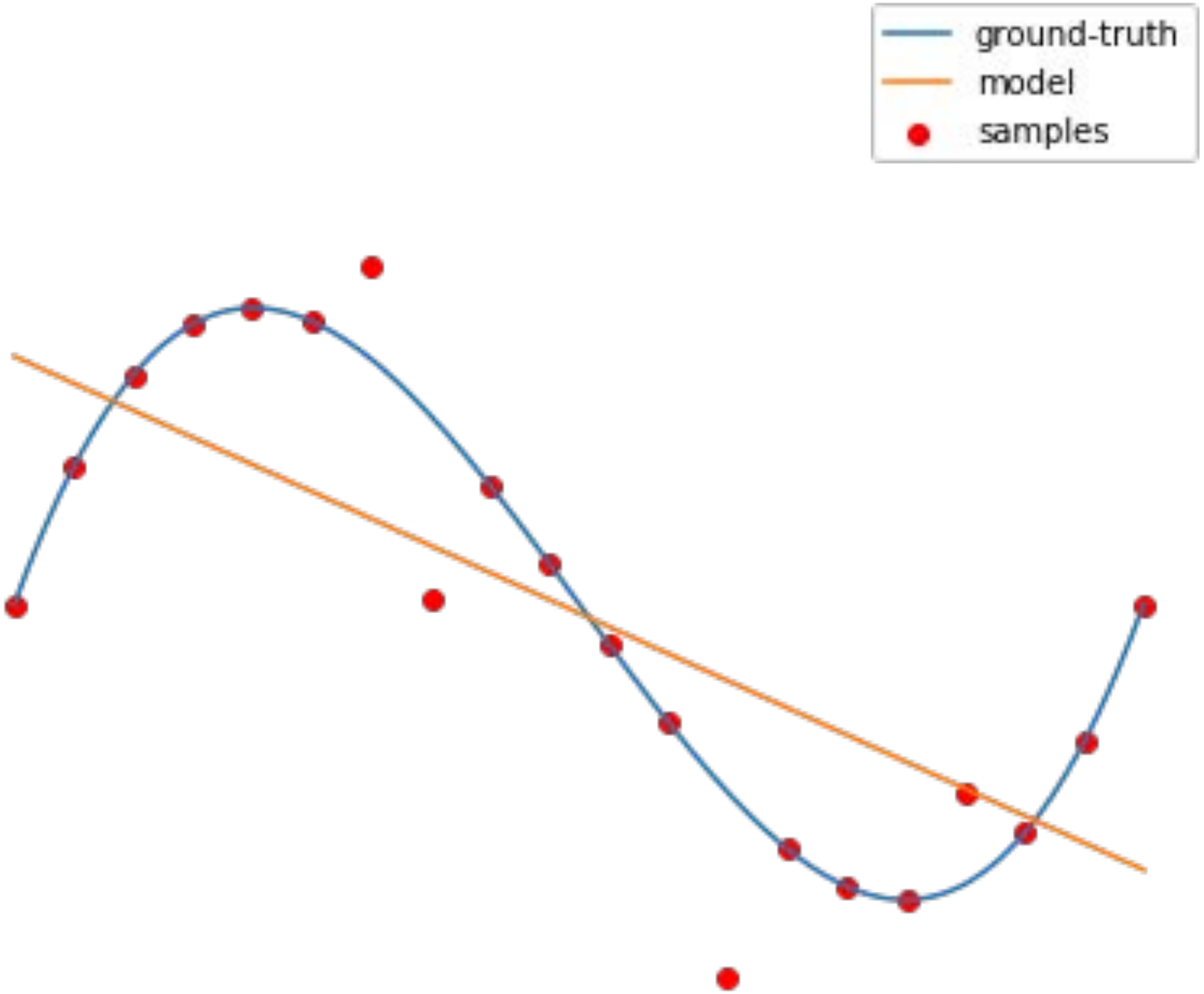


image: Belkin et al, 2019

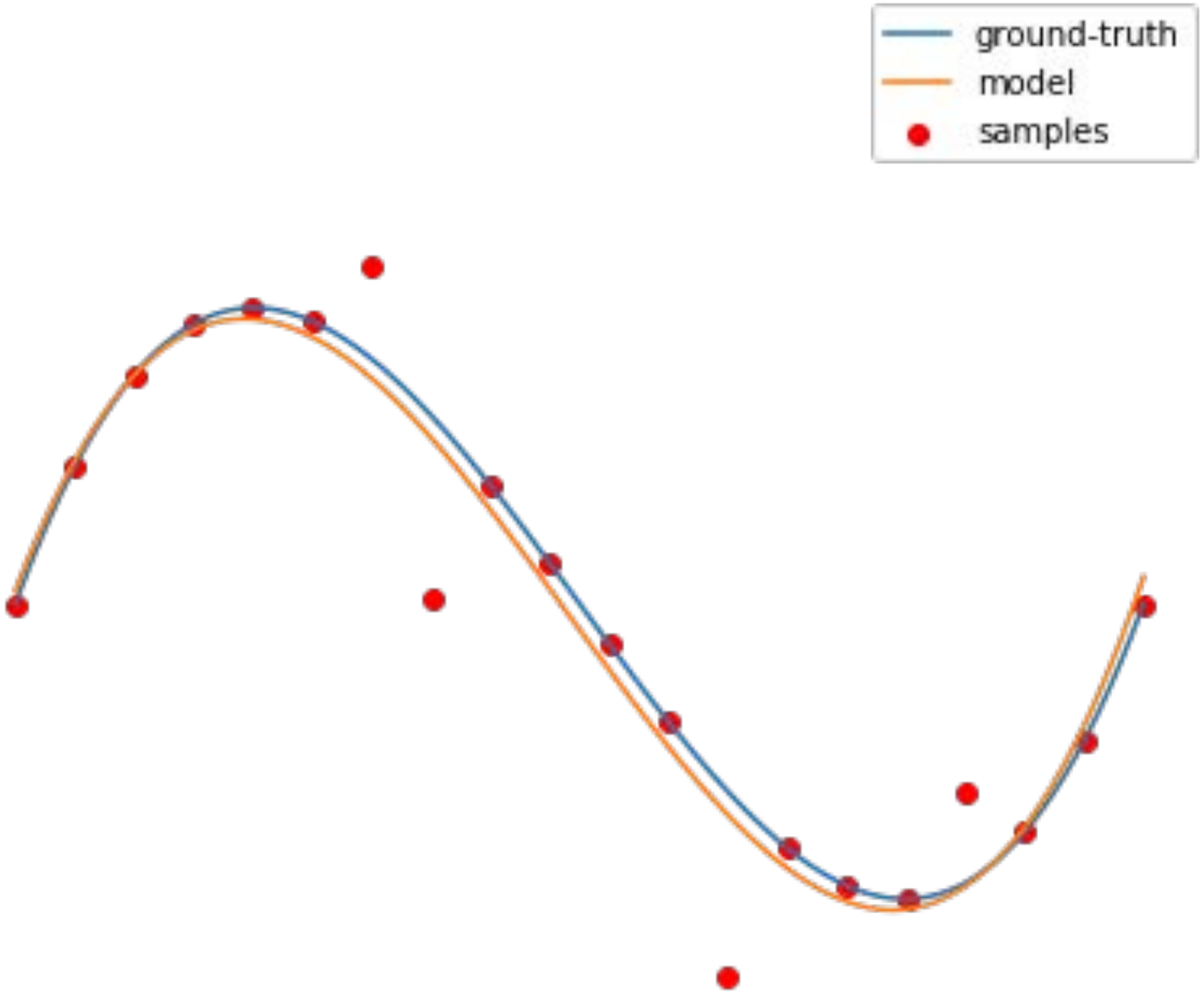
How should we measure model complexity?

parameters?

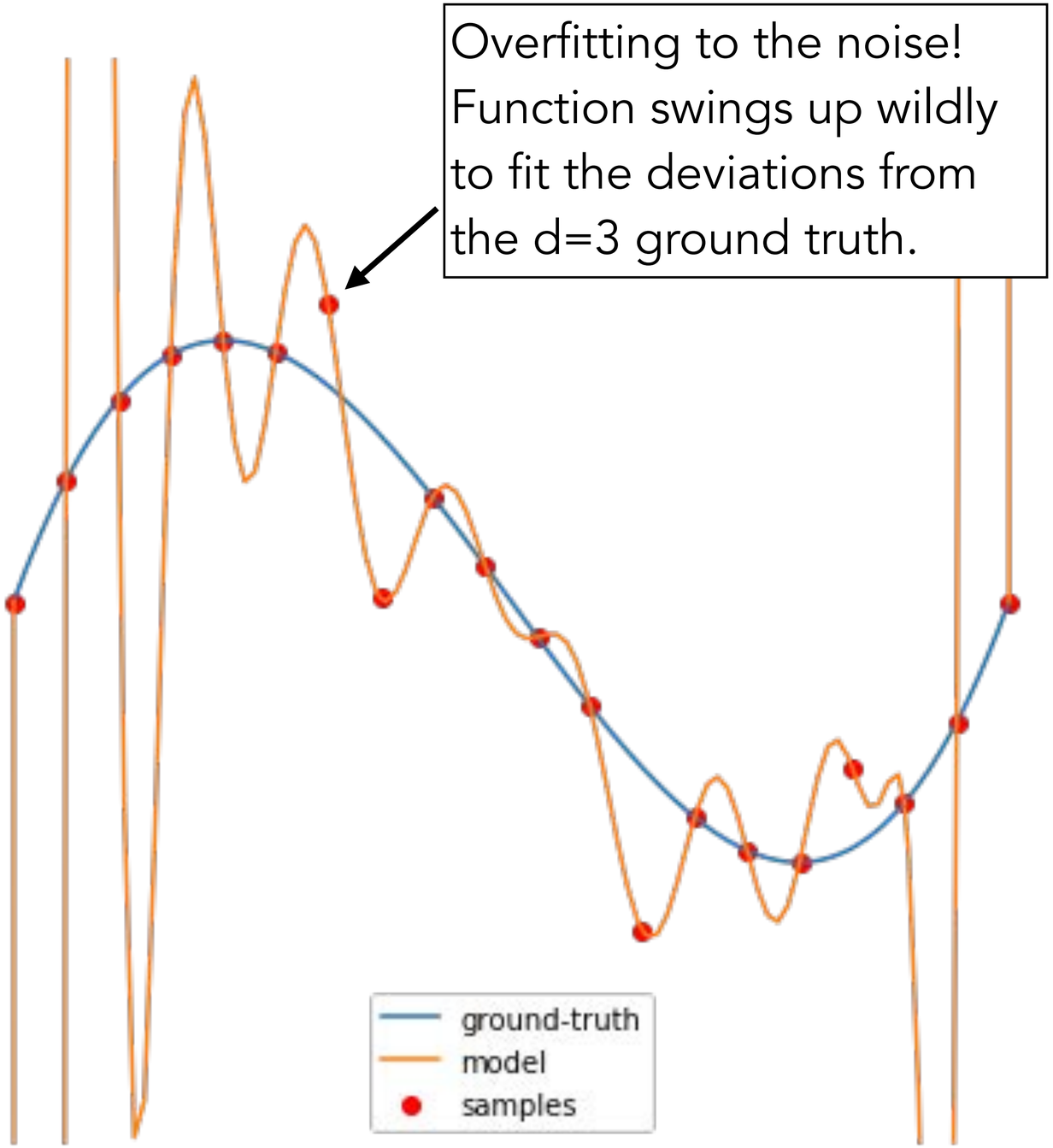
d=1



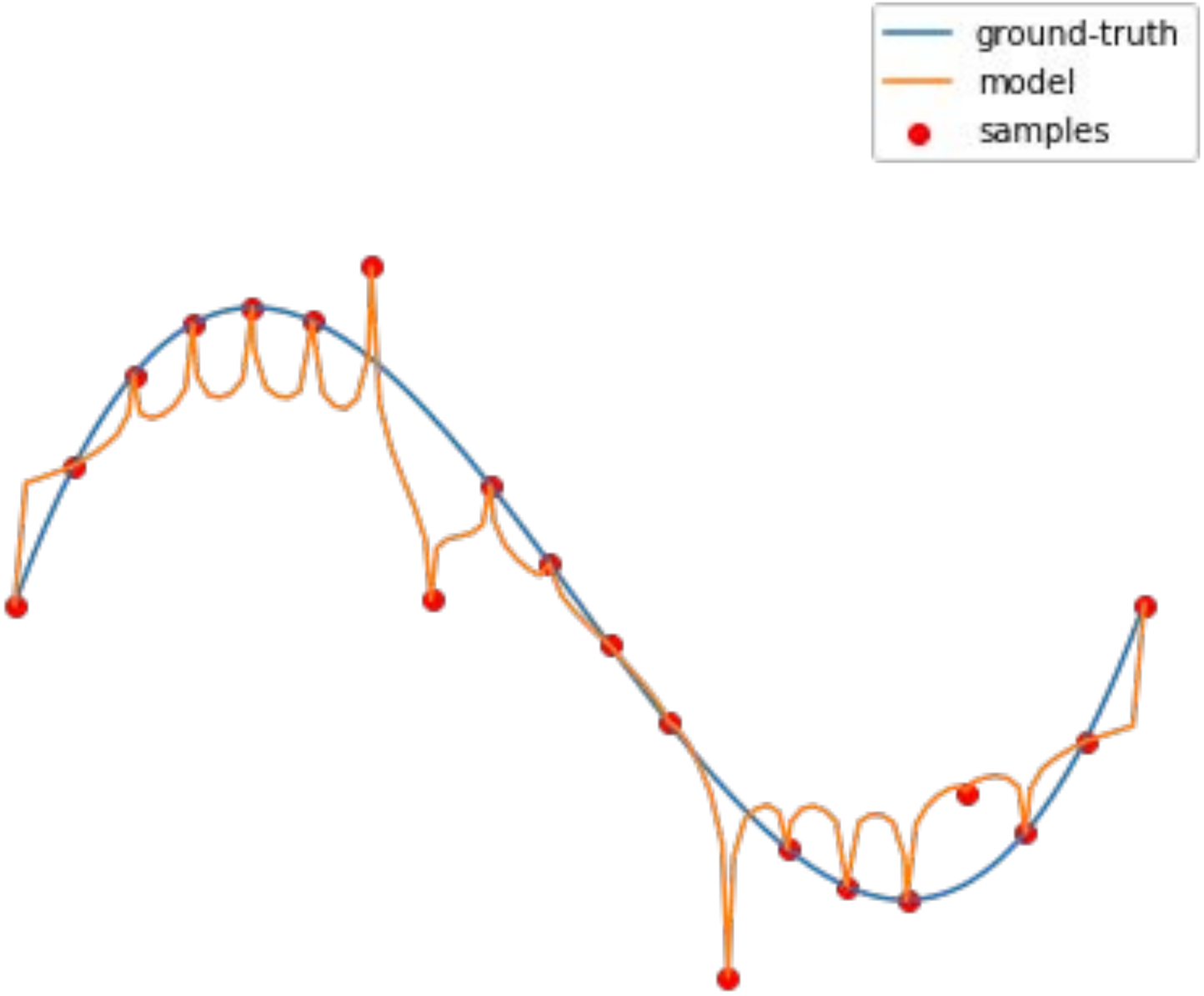
d=3



d=20

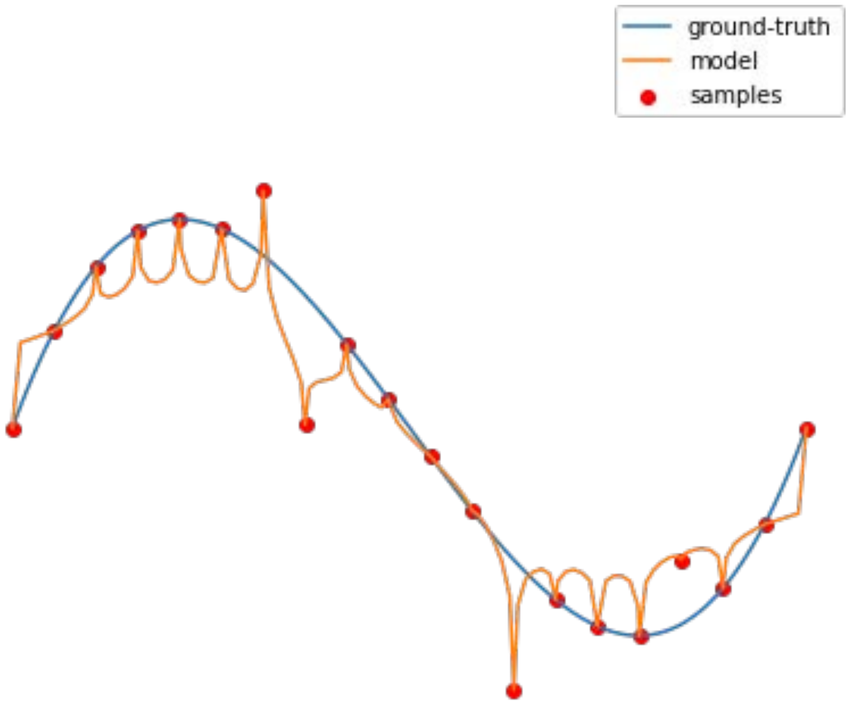

Overfitting to the noise!
Function swings up wildly
to fit the deviations from
the d=3 ground truth.
ground-truth
model
samples

d=1000



The simple + spiky hypothesis

How can deep nets perfectly fit noisy training data but also make good predictions on test data?



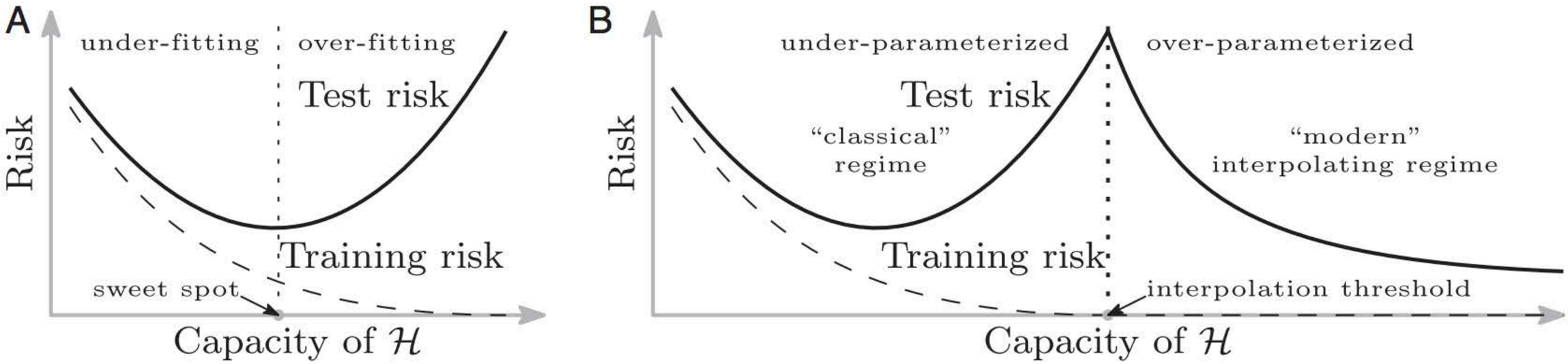
learned model = "simple" + "spiky"

predictive component *overfitting component*

[Belkin, Rakhlin, Tsybakov 2018]

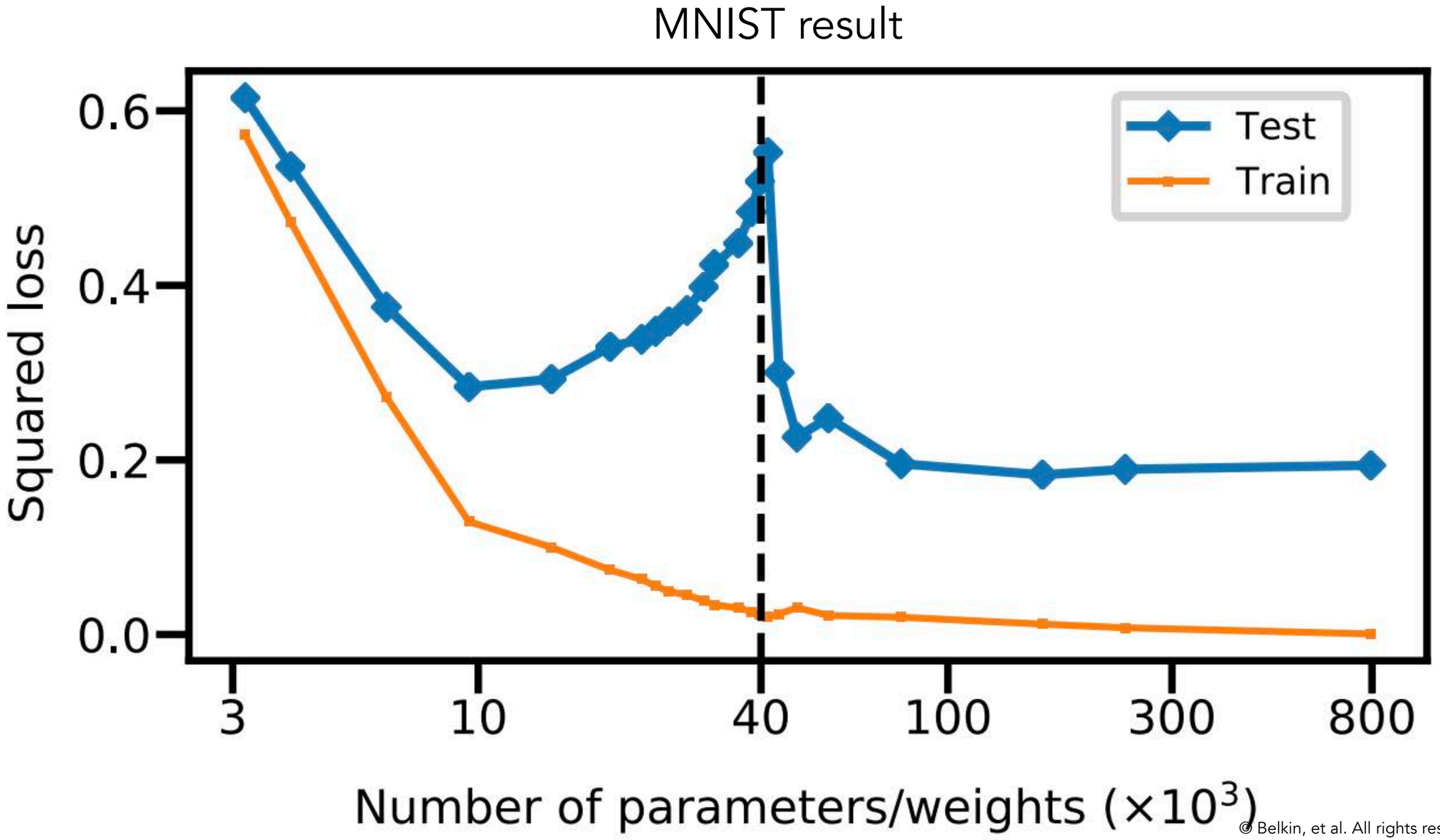
Visualization for an MLP:
https://www.youtube.com/watch?v=Kih-VPHL3gA

Double descent



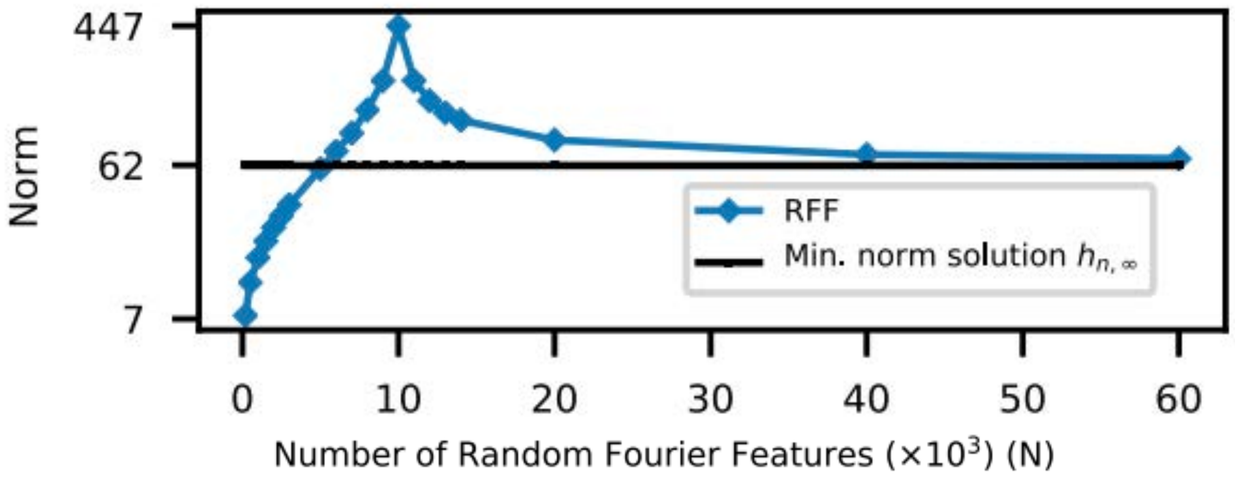
[Double-descent: Belkin, Hsu, Ma, Mandal, PNAS 2019]

Double descent



[Double-descent: Belkin, Hsu, Ma, Mandal, PNAS 2019]

The more features, the lower norm the learned function



← More features —> smoother solutions

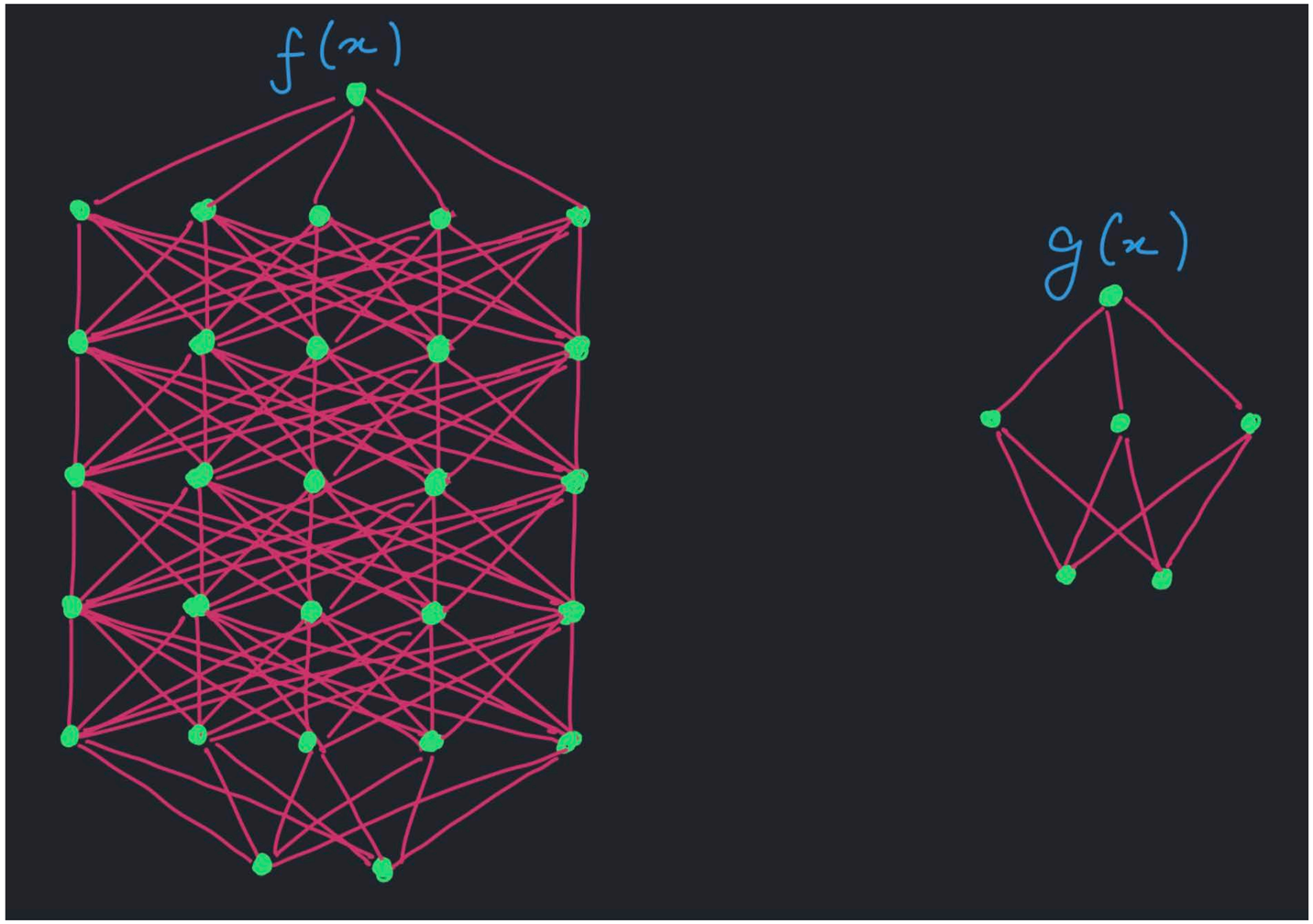
[Double-descent: Belkin, Hsu, Ma, Mandal, PNAS 2019]

How should we measure model complexity?

parameters?

No!

Parameter norm? maybe



Consider $h(x) = 10^{-100} f(x) + (1 - 10^{-100}) g(x)$

How many parameters does it have? Does it matter?

How should we measure model complexity?

Number of distinct functions the model can represent?

The classical picture: Vapnik-Chervonenkis theory

Remember: Generalization error = population error - training error

IF size of training set dwarfs number of functions in our function class.

THEN training error matches population error with high probability.

Intuition:

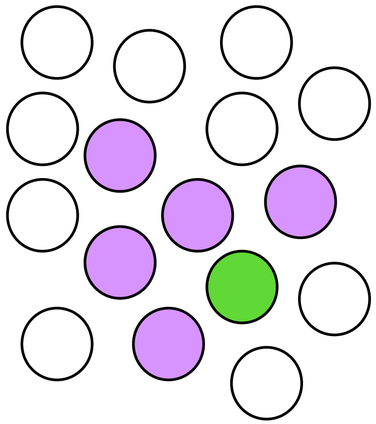
- "False positive" = function that fits training data but does not generalize
- Chance of a false positive is:
  - **higher** if we have more candidate functions (one could get lucky)
  - **lower** if we have more data (each additional datapoint rules out more candidate functions)
- Therefore, **if function class is small** and **training data is big**, chance of false positive is low

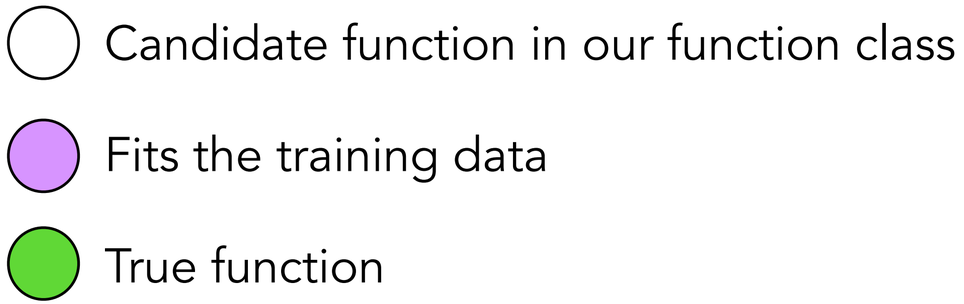
The classical picture: Vapnik-Chervonenkis theory

Remember: Generalization error = population error - training error

IF size of training set dwarfs number of functions in our function class.

THEN training error matches population error with high probability.





(Assumptions: noise-free training data, true function is in function class, optimizer just picks one of the purple points at random)

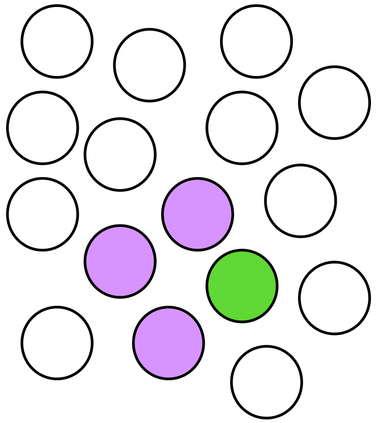
The classical picture: Vapnik-Chervonenkis theory

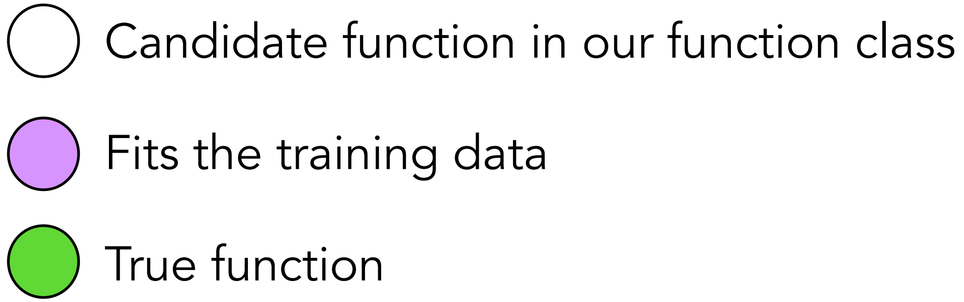
Remember: Generalization error = population error - training error

IF size of training set dwarfs number of functions in our function class.

THEN training error matches population error with high probability.

Increase training data





(Assumptions: noise-free training data, true function is in function class, optimizer just picks one of the purple points at random)

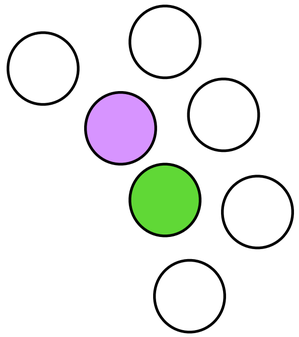
The classical picture: Vapnik-Chervonenkis theory

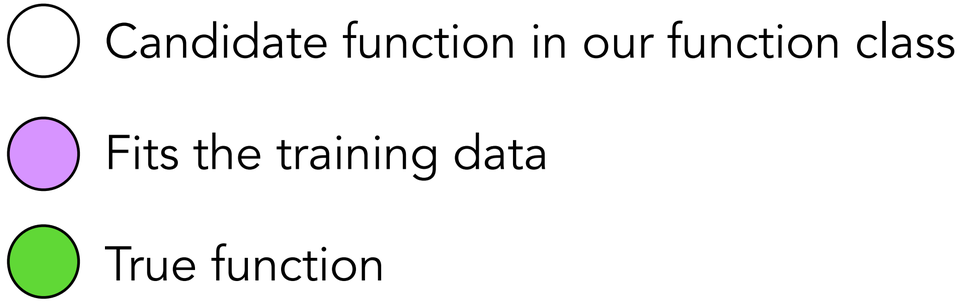
Remember: Generalization error = population error - training error

IF size of training set dwarfs number of functions in our function class.

THEN training error matches population error with high probability.

Reduce capacity *(i.e. decrease number of functions in our function class)*





(Assumptions: noise-free training data, true function is in function class, optimizer just picks one of the purple points at random)

The classical picture: Vapnik-Chervonenkis theory

How to count number of functions in a function class (architecture)?

It turns out you can just count the number of "dichotomies" (i.e. binary labelings) that your function class can realize on your data.

	dichotomy 1	dichotomy 2	⋯	dichotomy d
img_001.jpg	+1	-1		-1
img_002.jpg	+1	-1		+1
⋮	⋮	⋮		⋮
img_999.jpg	-1	-1		-1

The classical picture: Vapnik-Chervonenkis theory

VC dimension (d):

d = # dichotomies

n = # training points

Generalization bound:

Generalization error is bounded by $\sqrt{\frac{d}{n}}$

Empirical observation: NNs can fit random labels

Given a dataset of cats and dogs:

img_001.jpg	cat
img_002.jpg	cat
⋮	⋮
img_999.jpg	dog

Assign each image a random label.

Often, the neural net can still fit the random labeling… they can fit noise!

The classical picture: Vapnik-Chervonenkis theory

VC dimension (d):

d = # dichotomies

n = # training points

Generalization bound:

Generalization error is bounded by $\sqrt{\frac{d}{n}}$

For neural nets, often we can fit any dichotomy!

There are d = 2^n dichotomies of n datapoints.

Therefore, generalization bound is extremely loose (in fact, vacuous).

Zhang, et al., 2017 Understanding deep learning requires rethinking generalization.

NNs empirically

Table 1: The training and test accuracy (in percentage) of various models on the CIFAR10 dataset Performance with and without data augmentation and weight decay are compared. The results o fitting random labels are also included.

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

- NN "memorize"/ interpolate the data, even random labels
- can still generalize
- with and without explicit regularization

How should we measure model complexity?

Number of distinct functions the model can represent?

No!

How should we measure model complexity???

… for deep learning, it’s still an open question!

Recap so far

Deep nets generalize. They can make reasonable predictions on inputs they have never seen during training.

Generalization requires *inductive biases*. Can't be explained by just fitting the training data (we have to rule out the filing cabinet!).

These inductive biases can't just be about classical notions of complexity (# parameters, VC-dimension, etc don't work).

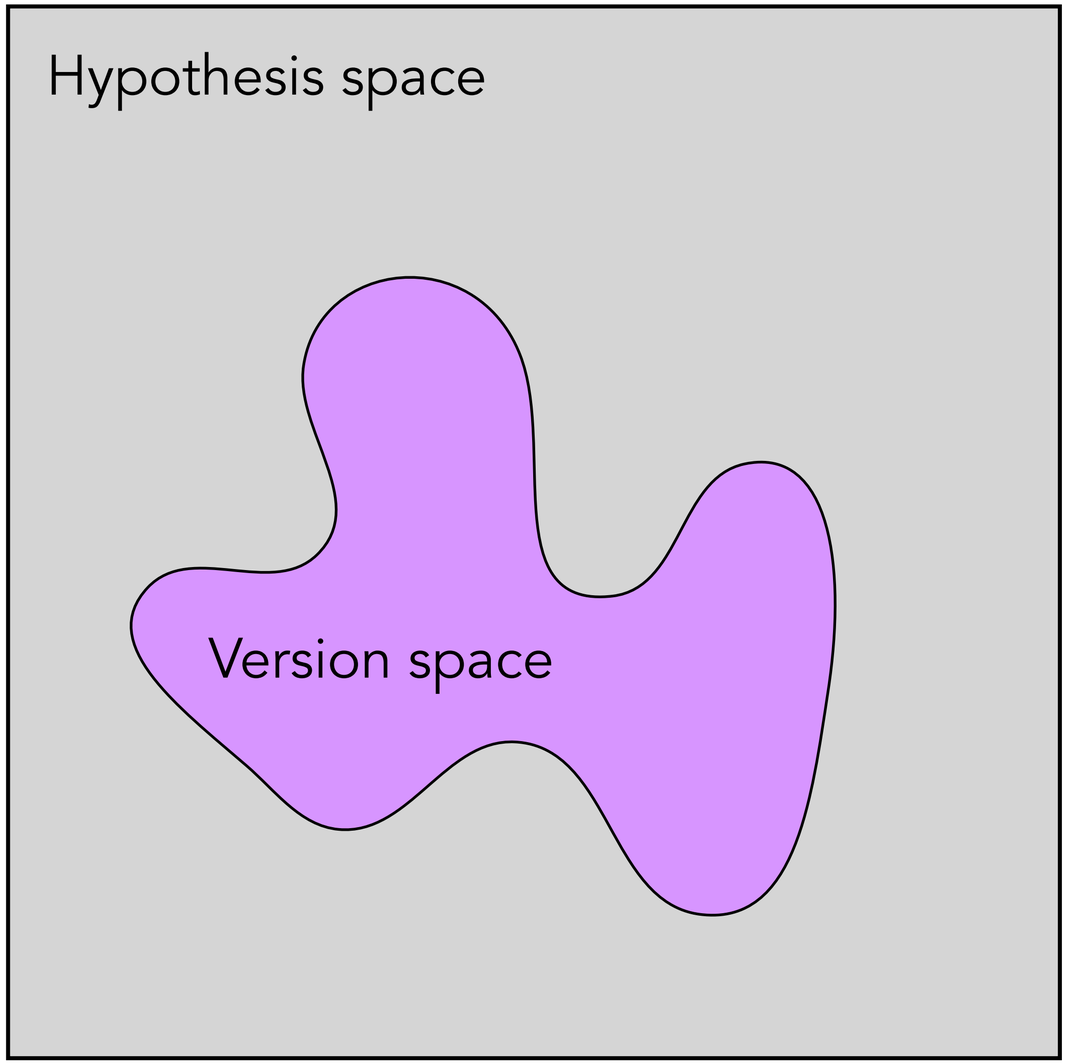
Therefore, deep learning must have some nice inductive biases that control complexity in ways we don't fully know how to characterize!

Next: **what are they?**

Why do deep nets learn functions that generalize?

Other than fitting the training data, what are the other pressures that affect the solution deep learning arrives at?

If we fit the data, what's left to consider?

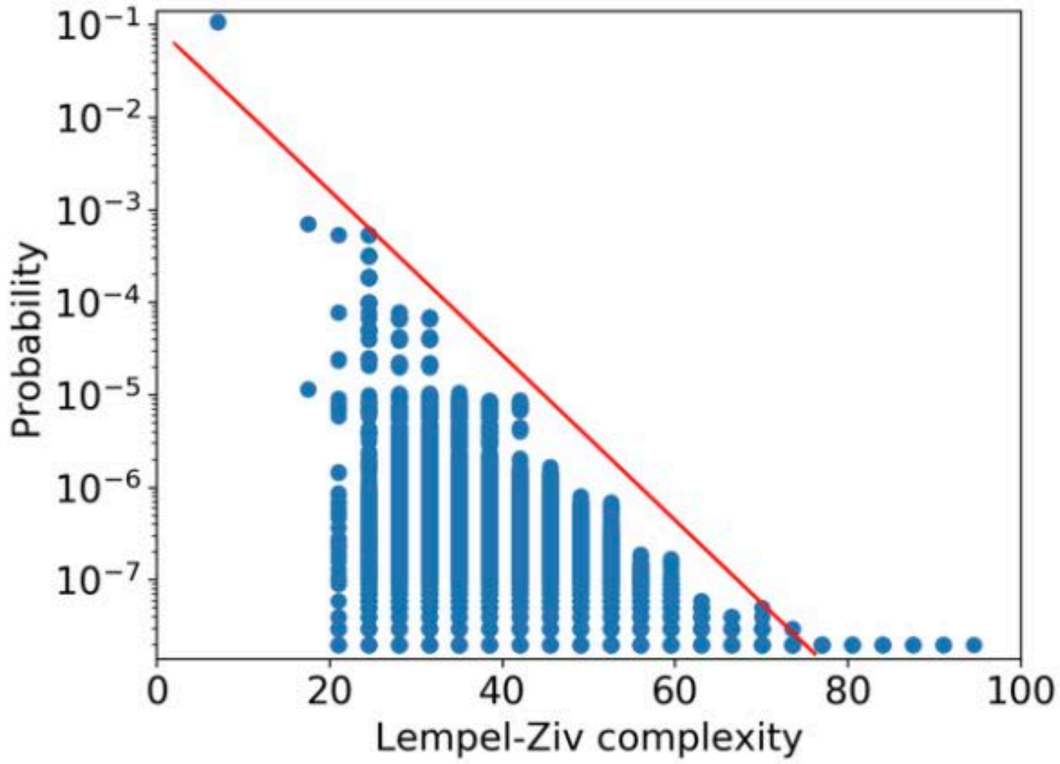


Version space: set of all mappings that achieve zero training error.

Simplicity bias in the parameter-function map

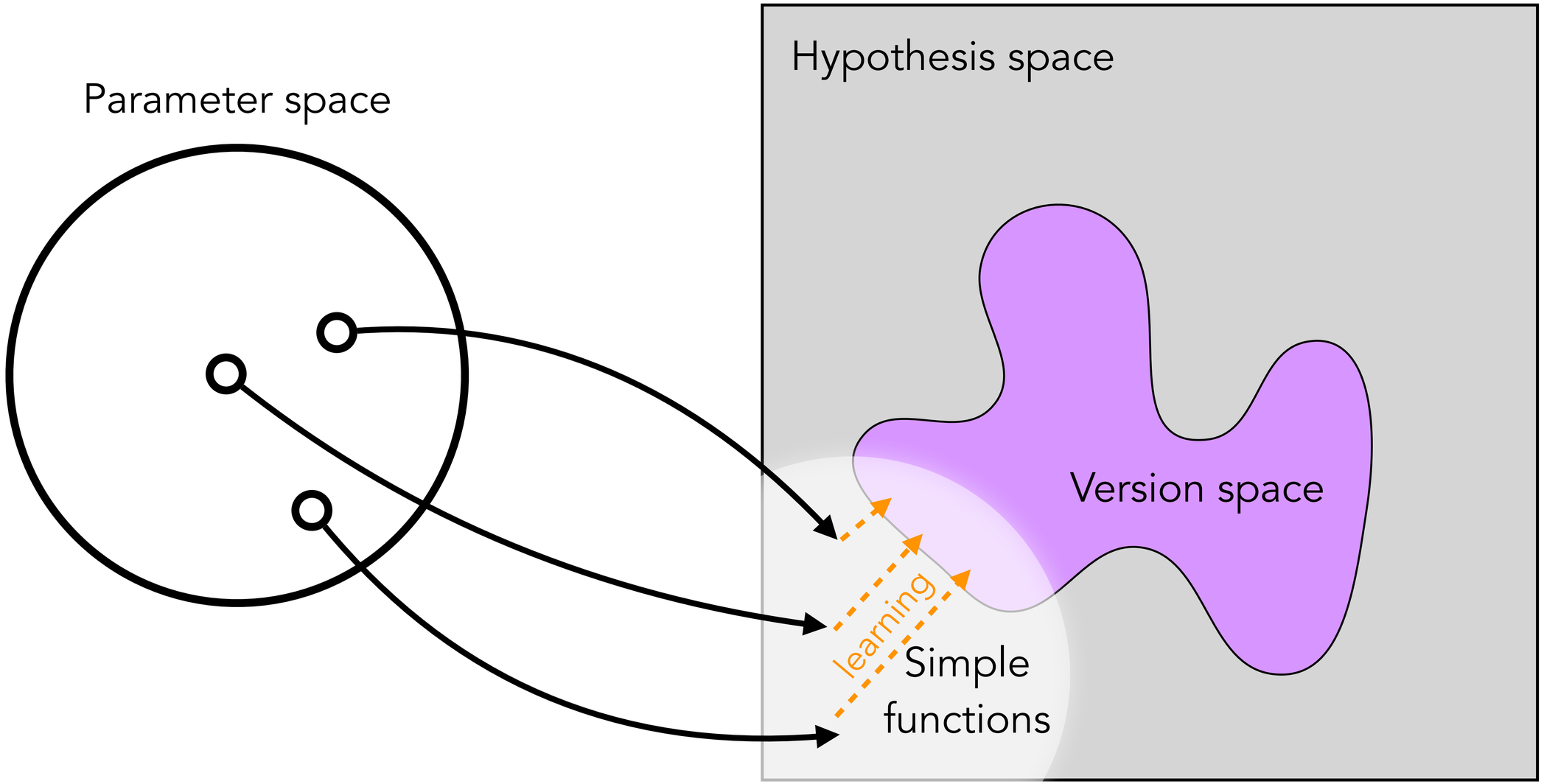
Parameter-function map: $\mathcal{M} : \Theta \to \mathcal{F}$

Most random settings of the weights in biases in a neural net map to simple functions.



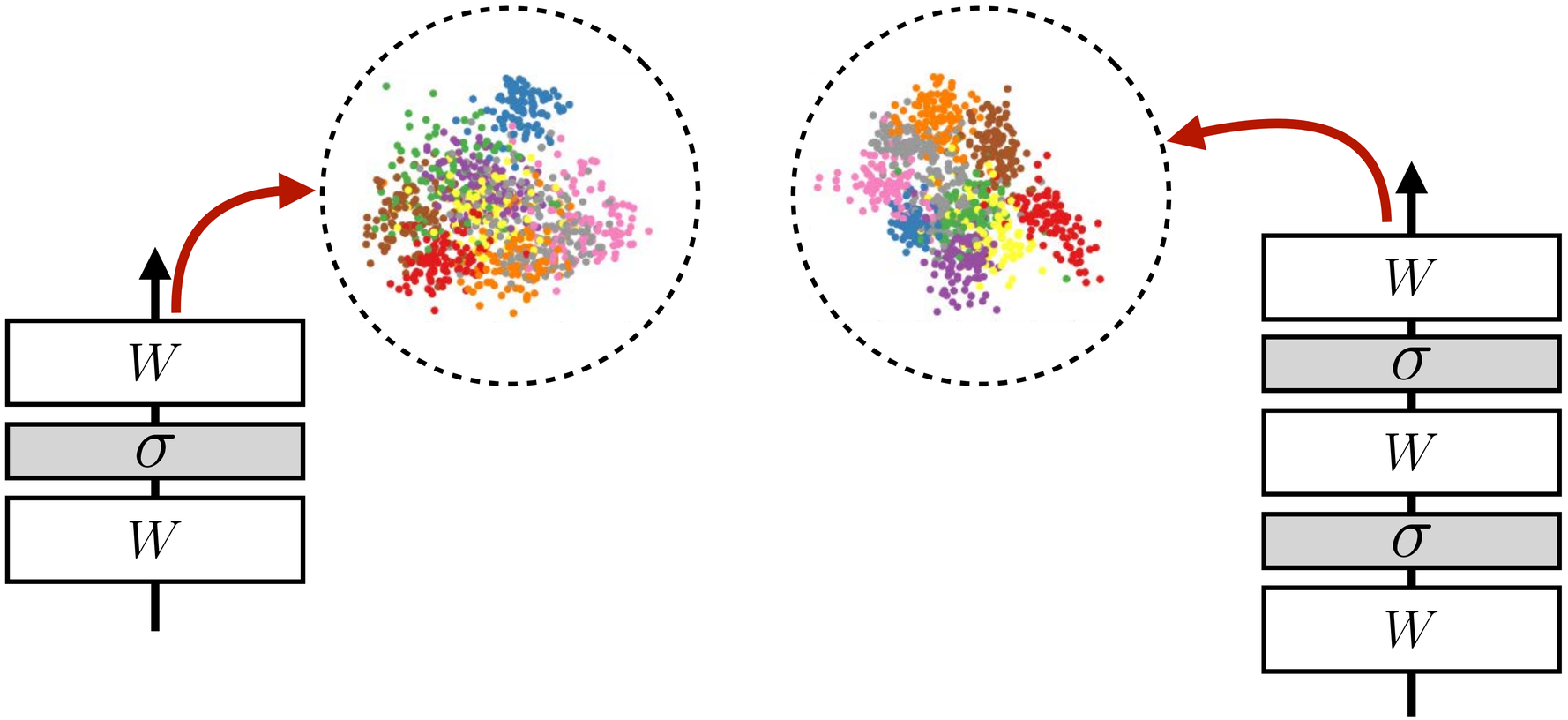
[Valle Pérez, Camargo, Louis, ICLR 2019]

Simplicity bias in the parameter-function map



[Valle Pérez, Camargo, Louis, ICLR 2019]

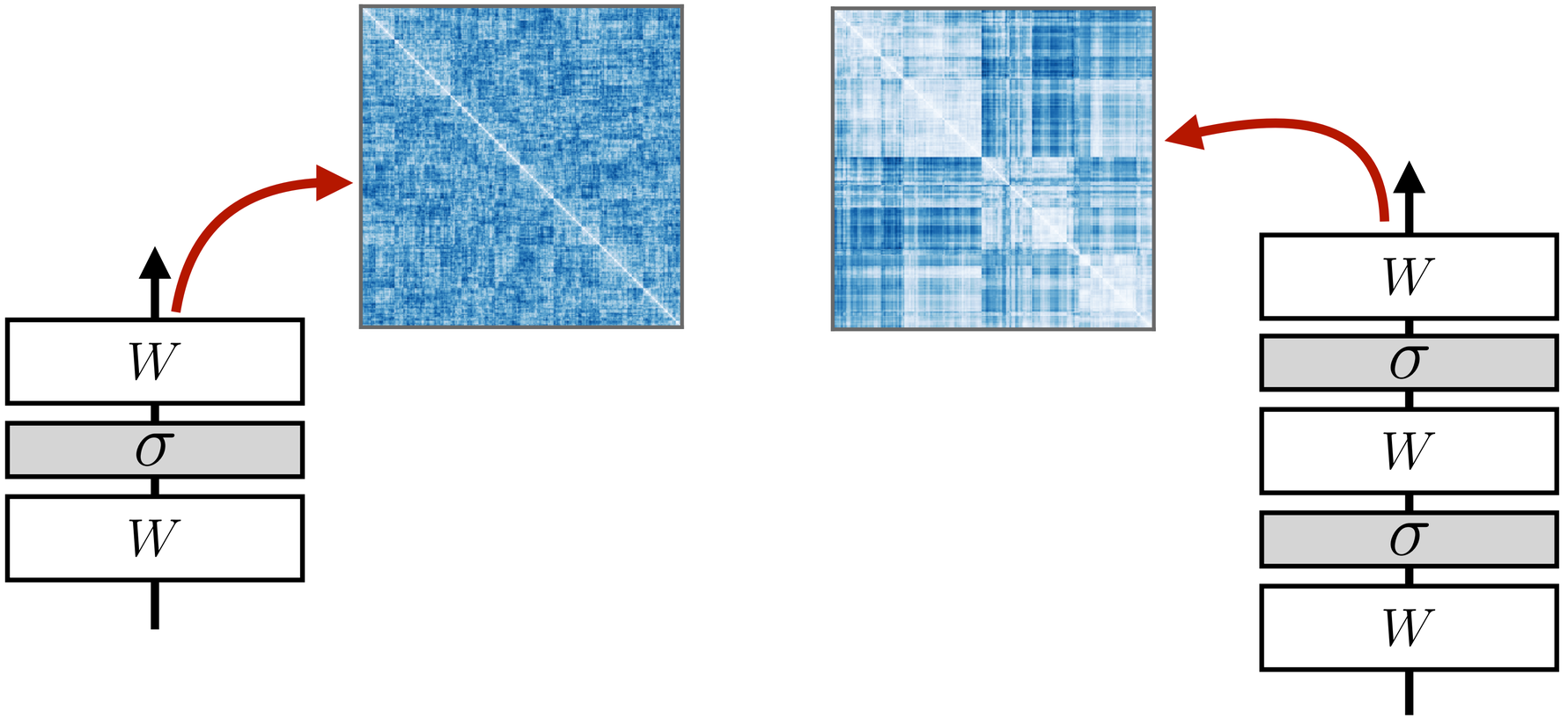
Low-rank bias of depth



[Huh, Mobahi, Zhang, Cheung, Agrawal, Isola, TMLR 2023]

Low-rank bias of depth

Similarity matrices between different inputs (aka **kernel**)

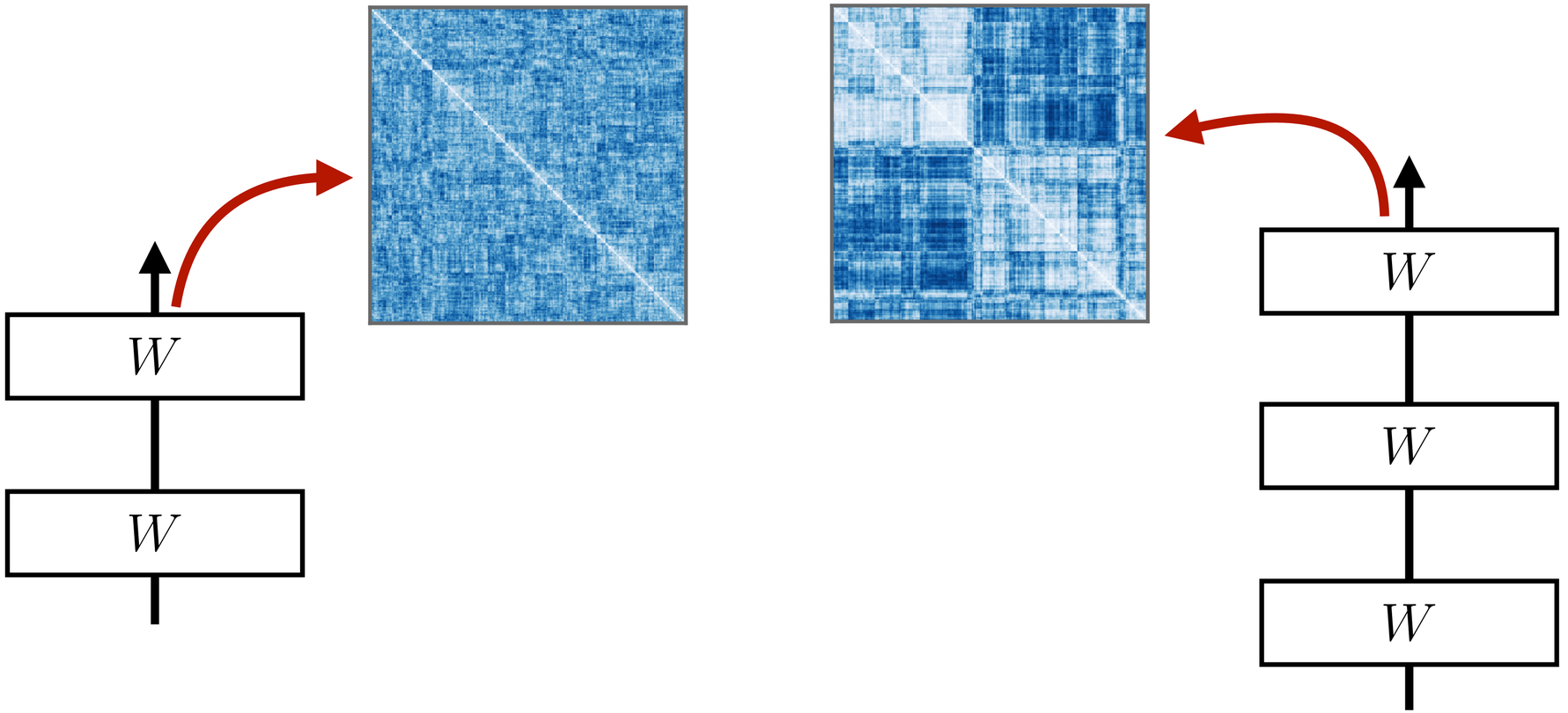


Common understanding: deeper nets have greater capacity to organize the data

[Huh, Mobahi, Zhang, Cheung, Agrawal, Isola, TMLR 2023]

Low-rank bias of depth

Similarity matrices between different inputs (aka **kernel**)



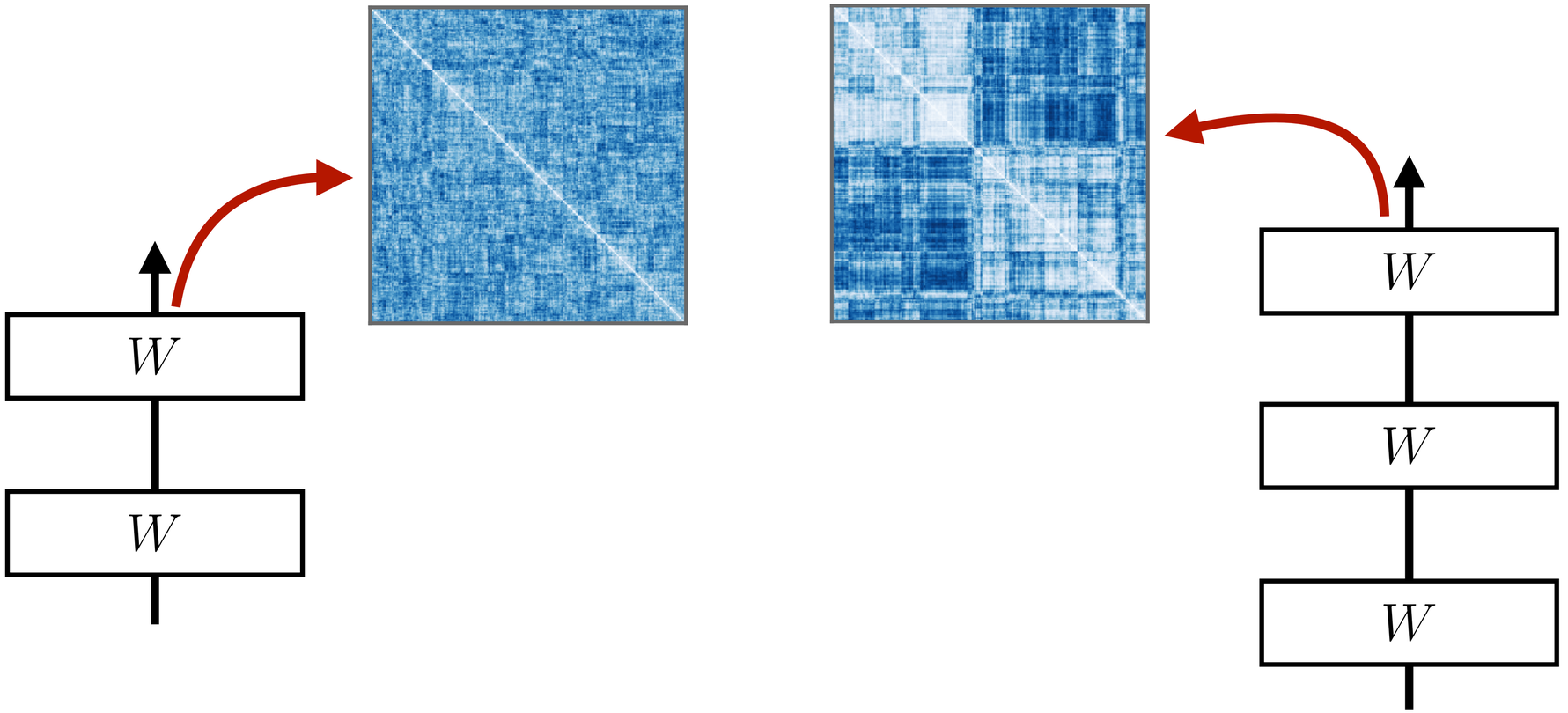
Same phenomenon with deep *linear* nets

(depth does not increase modeling capacity in this case)

[Huh, Mobahi, Zhang, Cheung, Agrawal, Isola, TMLR 2023]

Low-rank bias of depth

Similarity matrices between different inputs (aka **kernel**)

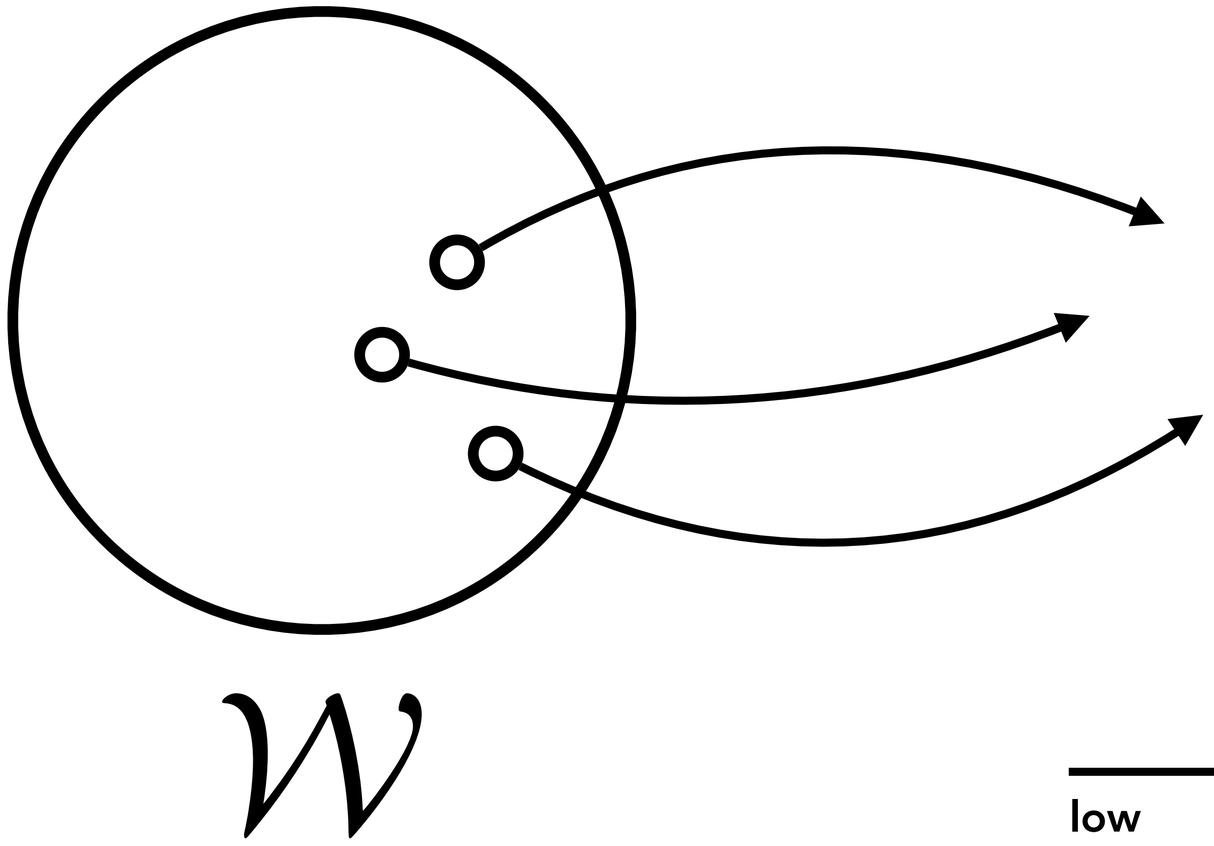


Why? Products of matrices tend to be low rank.

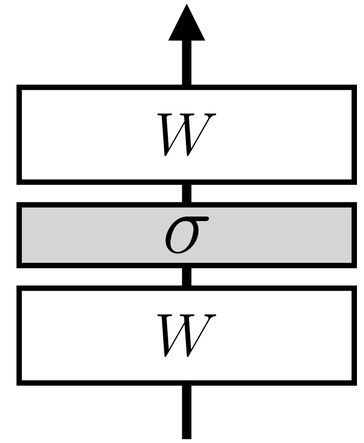
[Huh, Mobahi, Zhang, Cheung, Agrawal, Isola, TMLR 2023]

Parameter-kernel map

Sample a random set of network weights



K

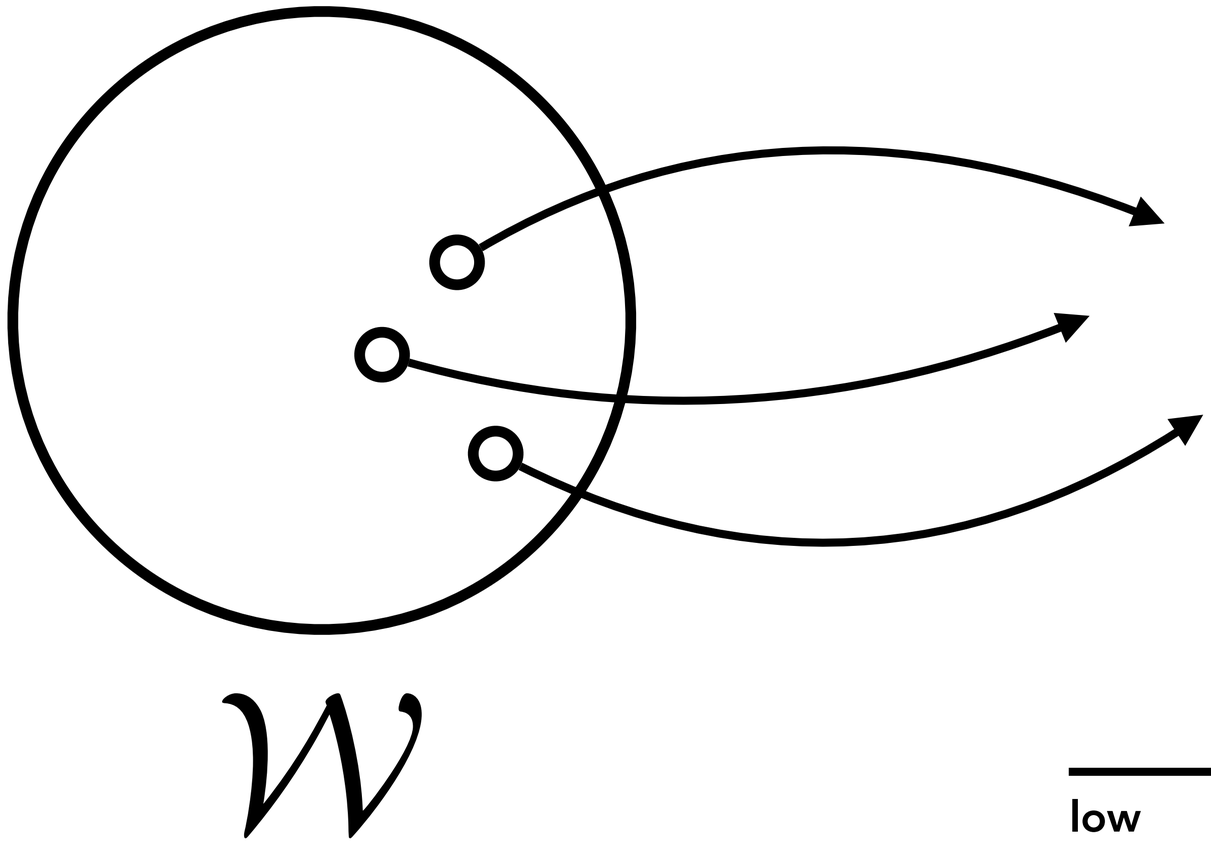


effective rank

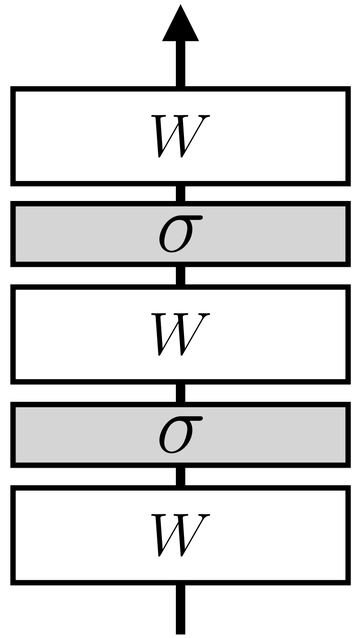
[Huh, Mobahi, Zhang, Cheung, Agrawal, Isola, TMLR 2023]

Parameter-kernel map

Sample a random set of network weights



K

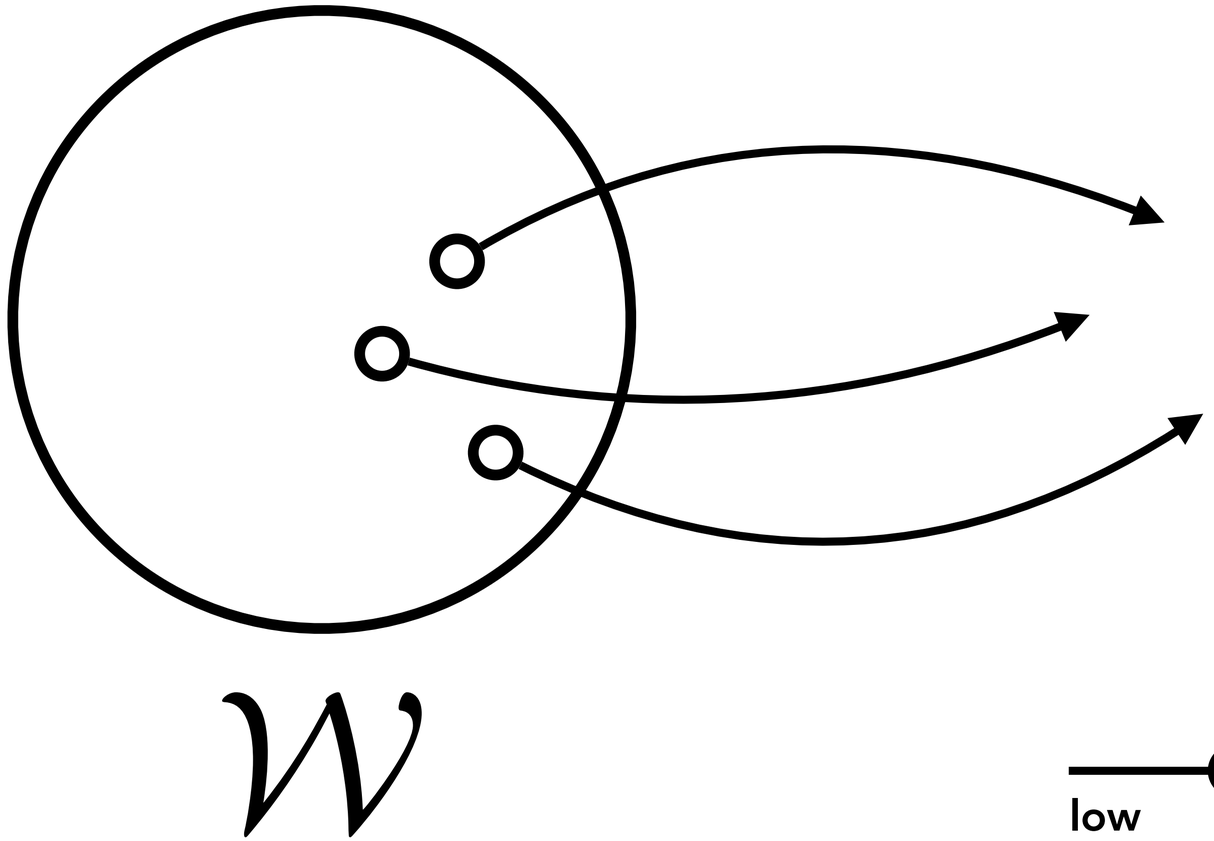


effective rank

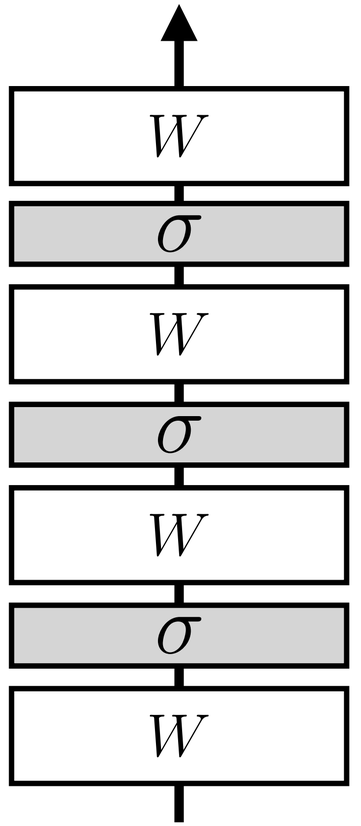
[Huh, Mobahi, Zhang, Cheung, Agrawal, Isola, TMLR 2023]

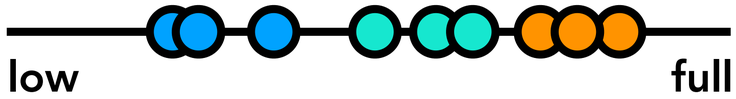
Parameter-kernel map

Sample a random set of network weights



K

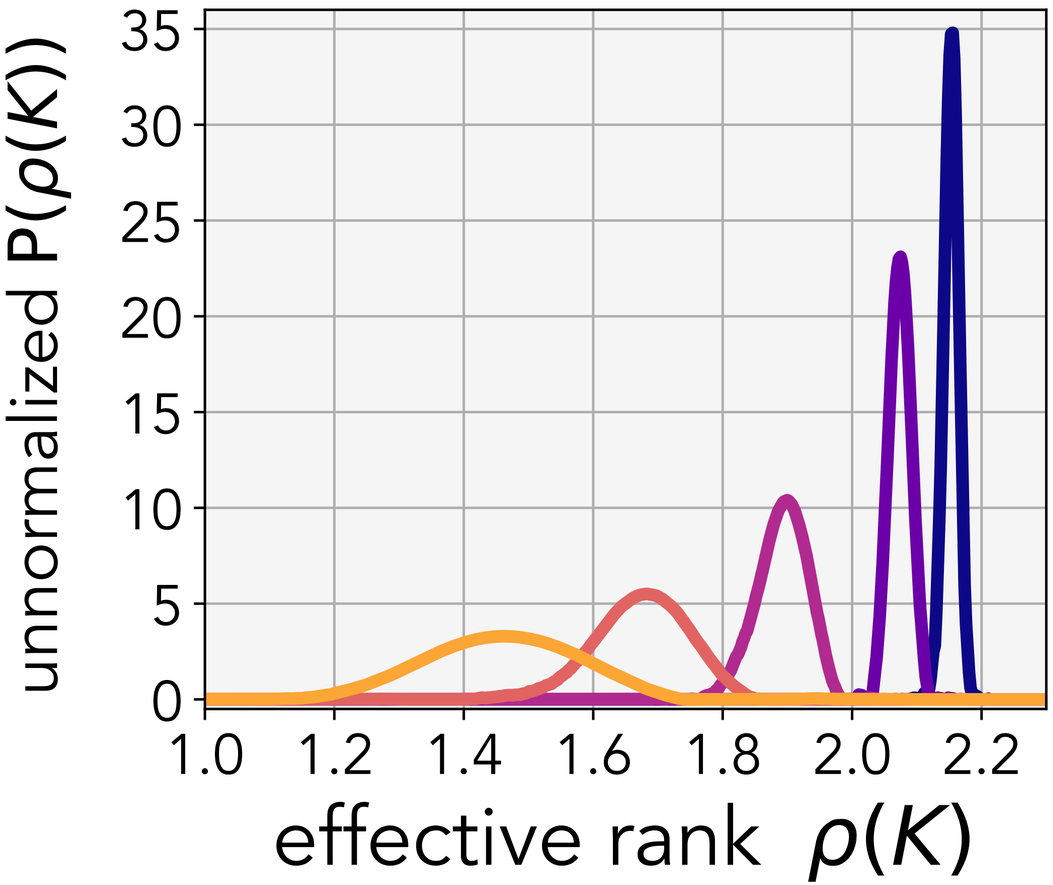




effective rank

[Huh, Mobahi, Zhang, Cheung, Agrawal, Isola, TMLR 2023]

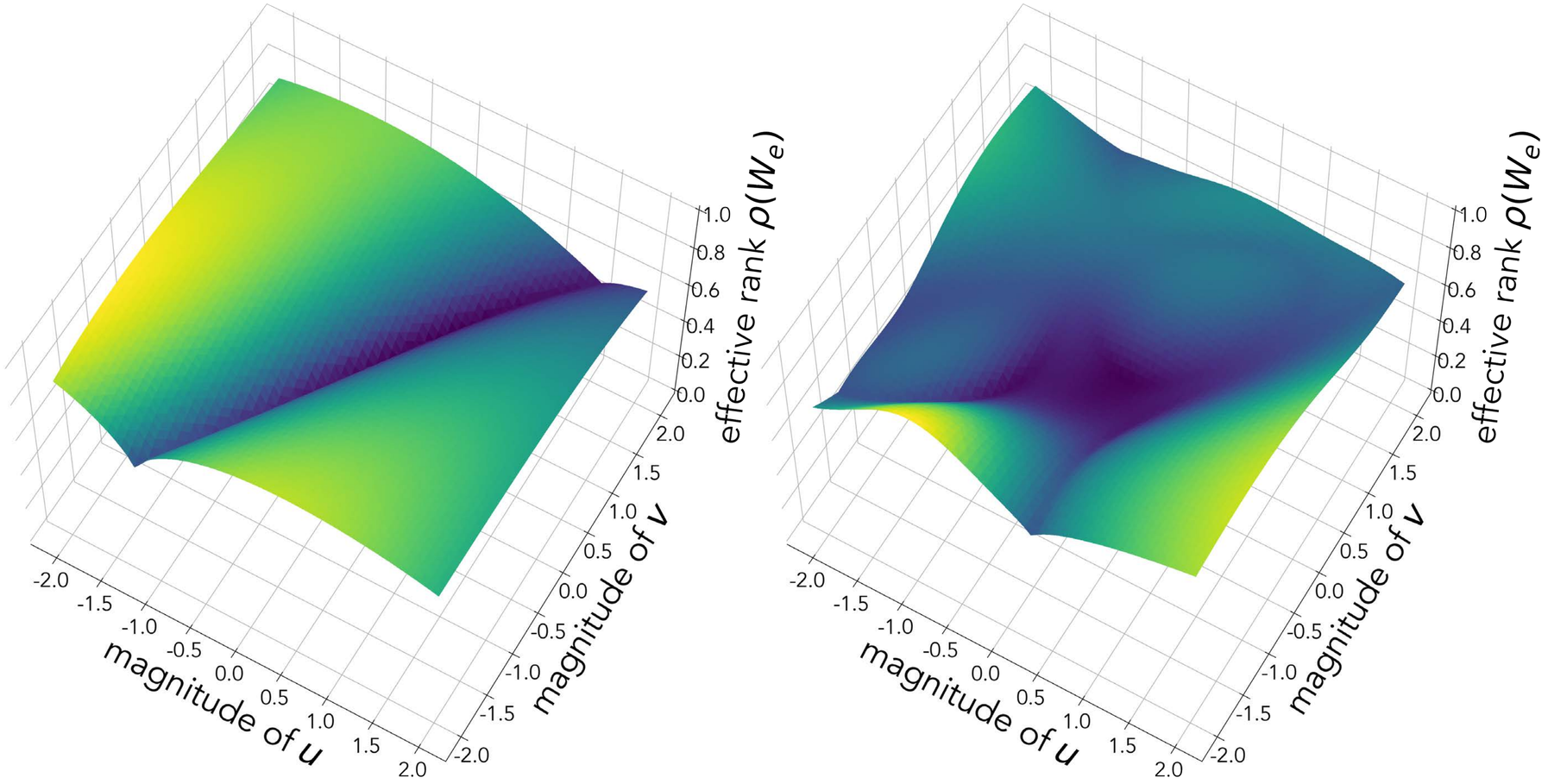
Deeper nets are *biased* toward lower-rank embeddings



Deeper networks have a greater proportion of parameter space that maps the input data to lower-rank embeddings.

[Huh, Mobahi, Zhang, Cheung, Agrawal, Isola, TMLR 2023]

Visualizing how depth reshapes the parameter landscape



single-layer

two-layer

[Huh, Mobahi, Zhang, Cheung, Agrawal, Isola, TMLR 2023]

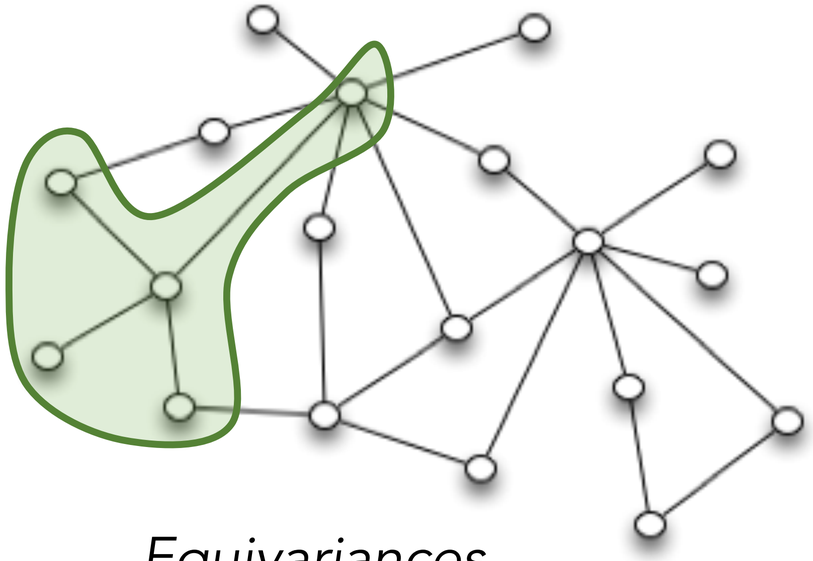
Implicit regularization of optimizers

- **Weight decay** acts like an L2 regularizer on weights, shrinking them toward zero all else being equal.

- **Initialization** near zero biases solutions toward low norm. GD initialized near zero converges to minimum norm solution for linear models [Zhang et al. 2017, Gunasekar et al. 2017]

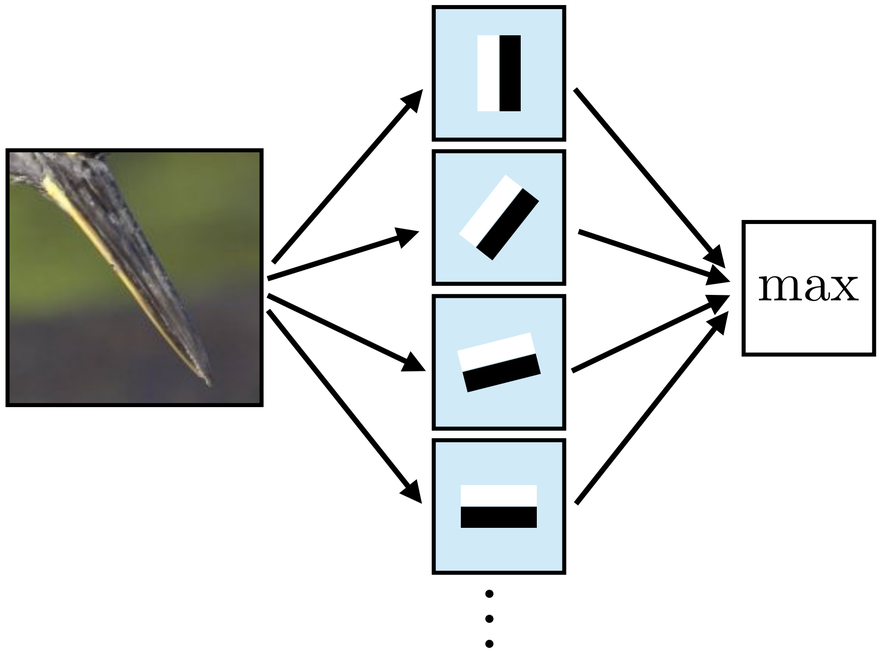
- **SGD**, and **GD** with finite step size, converge to “flat” minima; they will tend to overshoot or bounce out of minima that are too narrow. [see Vardi 2022 for a review]

Architectural symmetries

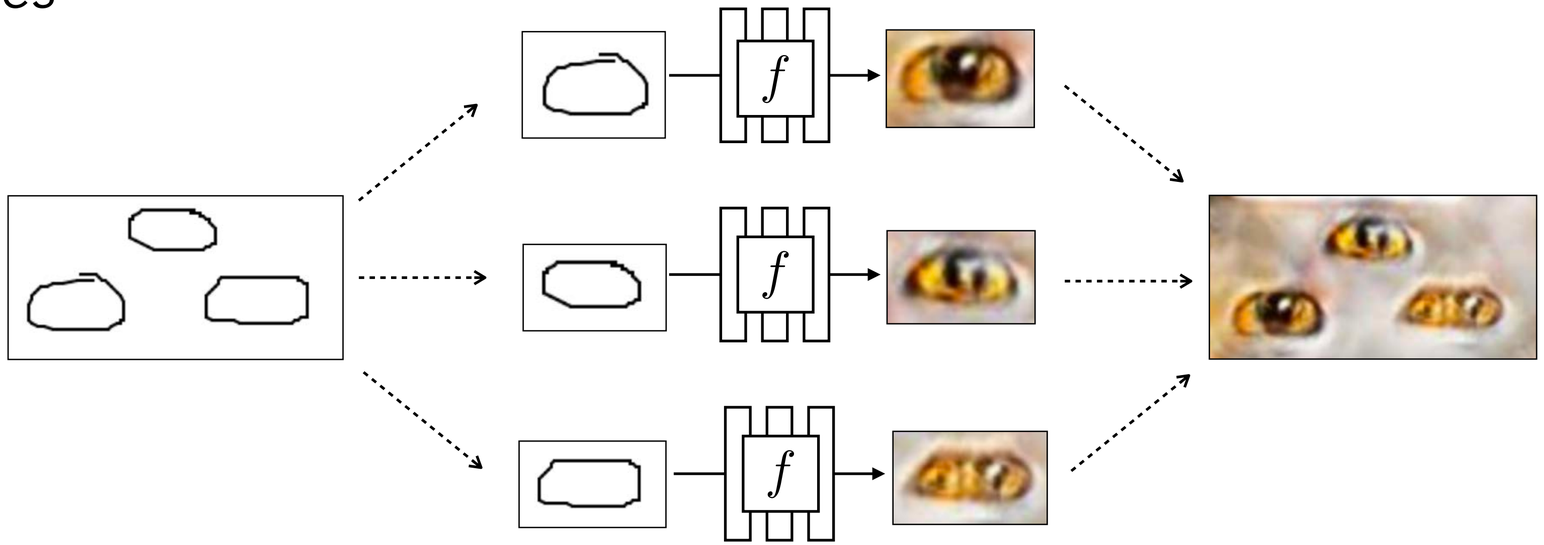


Equivariances

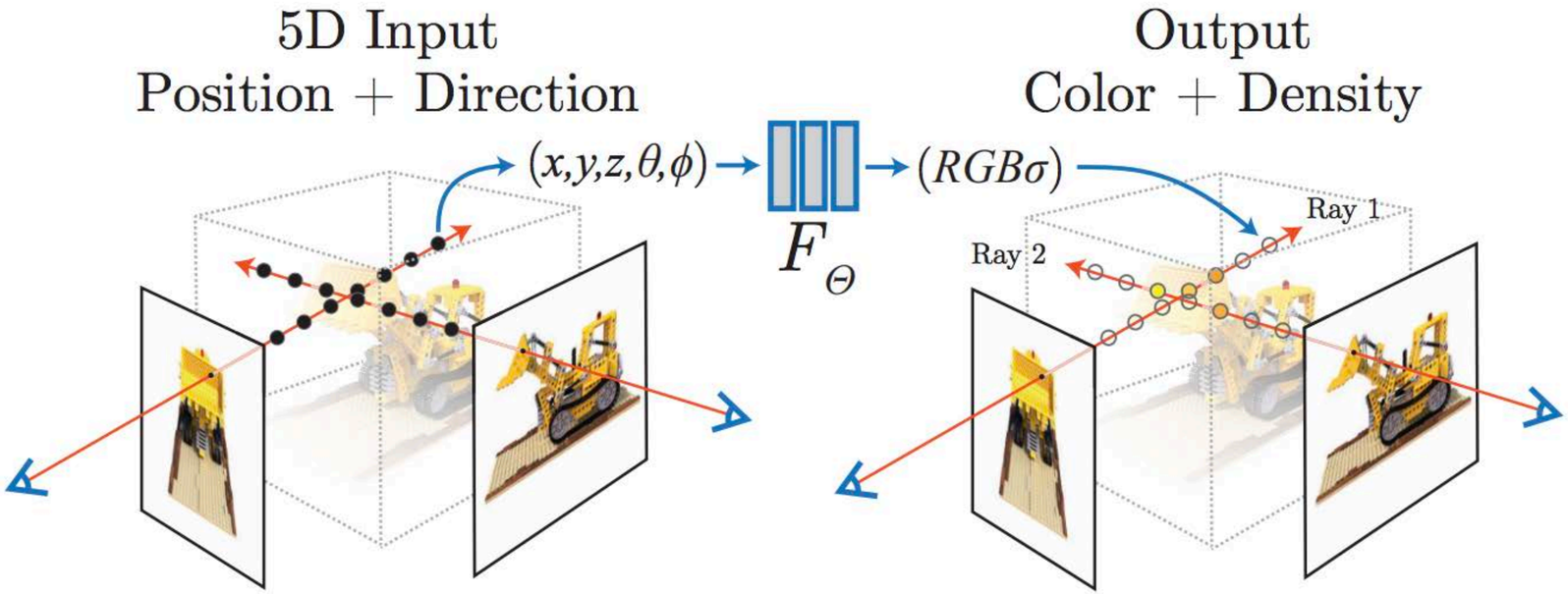
Invariances

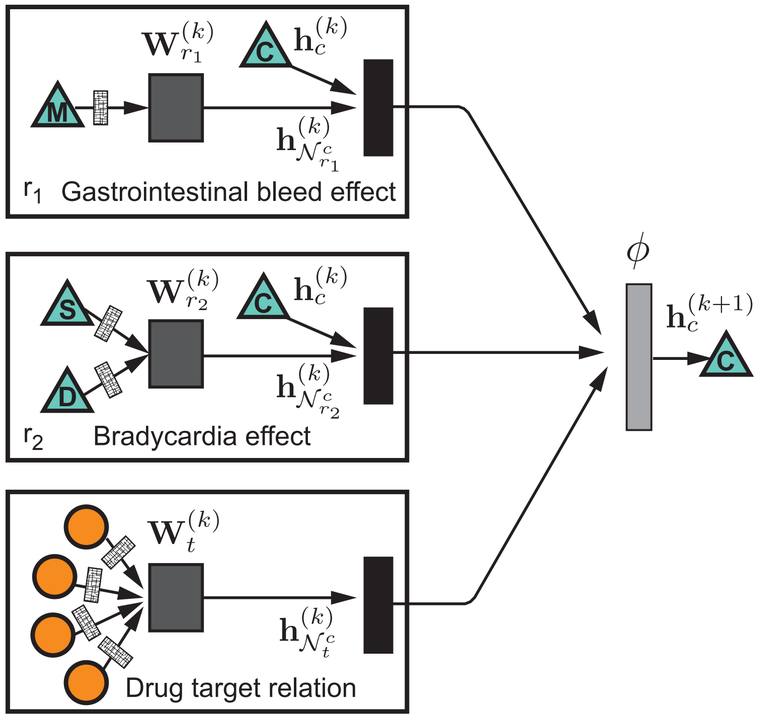


Compositionality



Domain specific constraints





How should we measure model complexity?

Theory answer:

The shortest program that fits the data is the one that will generalize best.

Intractable, but good to keep in mind…

Or… do we actually have to optimize for shortest? How about just short enough?

[Solomonoff 1964]

Finite models, infinite data

(rough memory of a conversation with Ilya Sutskever around 2018)

- Ilya: Deep nets generalize because they find small circuits that fit the data.
- Me: *Small* circuits? But deep nets are big! Don't we need something else to bias toward small circuits?
- Ilya: No. Deep nets are finite; that is enough. Anything finite will look small once you have enough data.

6.7960 Deep Learning
Fall 2024