Measuring Performance

DL4DS – Spring 2024

Where we are

=== Foundational Concepts ===

- ✓ 02 -- Supervised learning refresher
- ✓ 03 -- Shallow networks and their representation capacity
- ✓ 04 -- Deep networks and depth efficiency
- ✓ 05 -- Loss function in terms of maximizing likelihoods
- ✓ 06 – Fitting models with different optimizers
- ✓ 07a – Gradients on deep models and backpropagation
- ✓ 07b – Initialization to avoid vanishing and exploding weights & gradients
- 08 – Measuring performance, test sets, overfitting and double descent
- 09 – Regularization to improve fitting on test sets and unseen data

=== Network Architectures and Applications ===

- 10 – Convolutional Networks
- 11 – Residual Networks
- 12 – Transformers
- Large Language and other Foundational Models
- Generative Models
- Graph Neural Networks
- …

Measuring performance

- MNIST1D dataset model and performance
- Noise, bias, and variance
- Reducing variance
- Reducing bias & bias-variance trade-off
- Double descent
- Curse of dimensionality & weird properties of high dimensional space
- Choosing hyperparameters

MNIST1D

Scaling down Deep Learning

Sam Greydanus [1]

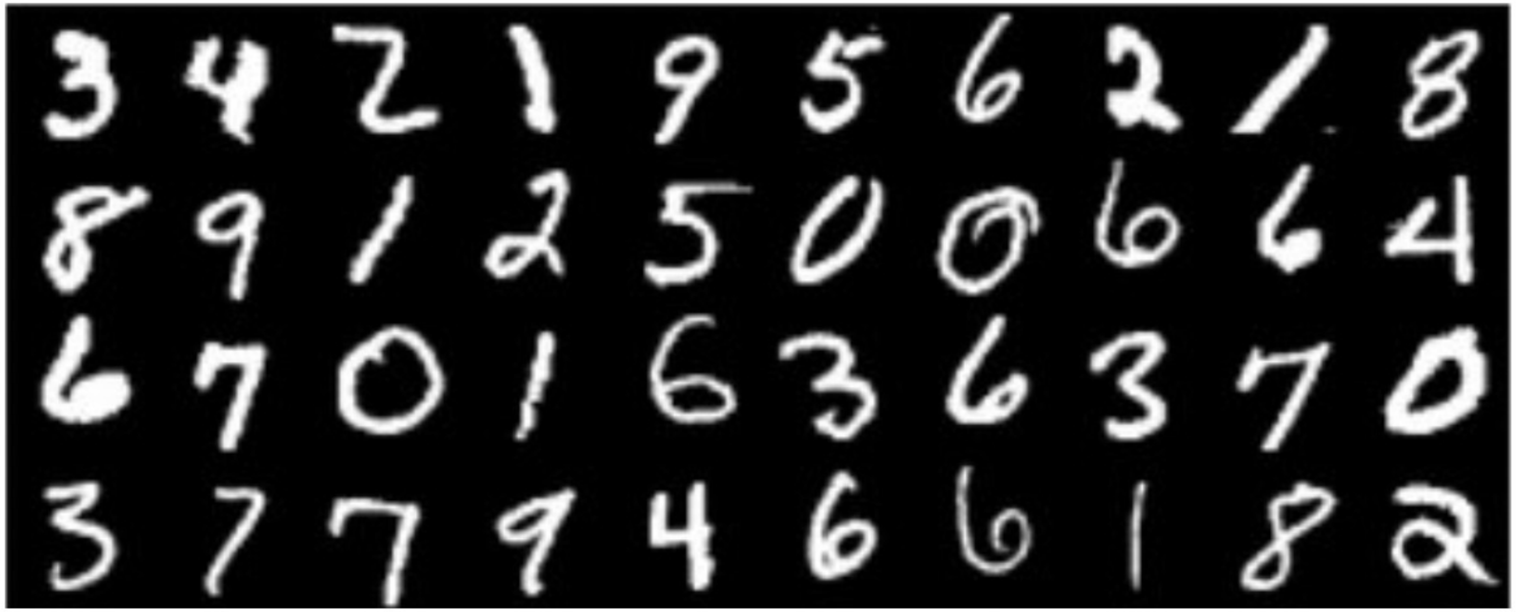
"A large number of deep learning innovations including dropout, Adam, convolutional networks, generative adversarial networks, and variational autoencoders began life as MNIST experiments. Once these innovations proved themselves on small-scale experiments, scientists found ways to scale them to larger and more impactful applications."

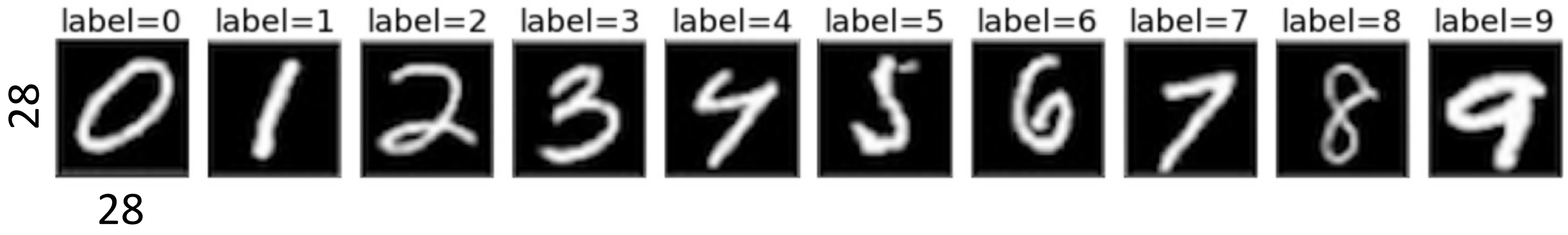
S. Greydanus, "Scaling down Deep Learning." arXiv, Dec. 04, 2020. doi: 10.48550/arXiv.2011.14439.

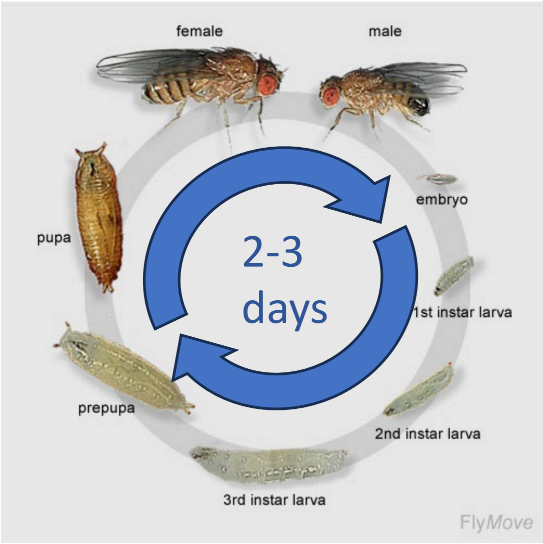
https://github.com/greydanus/mnist1d

MNIST Dataset

- 28x28x1 grayscale images
- 60K Training, 10K Test
- “Is to Deep Learning what fruit flies are to genetics research”



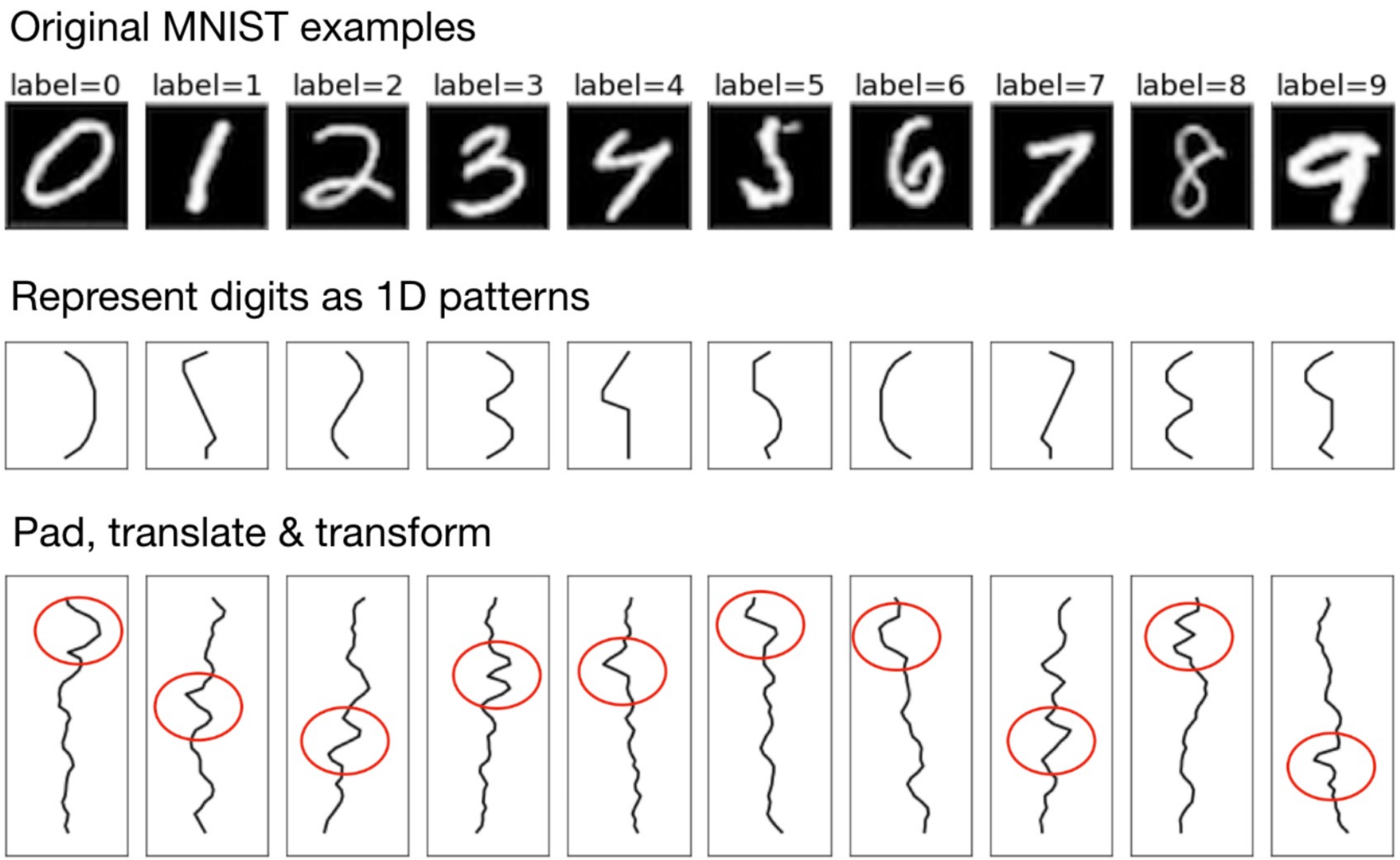




But poorly differentiates model performance:

Model Type	Accuracy
Logistic Regression	94%
MLP	99+%
CNN	99+%

MNIST 1D Dataset



Fixed, 1-D, length-12 templates for each of 10 digit classes

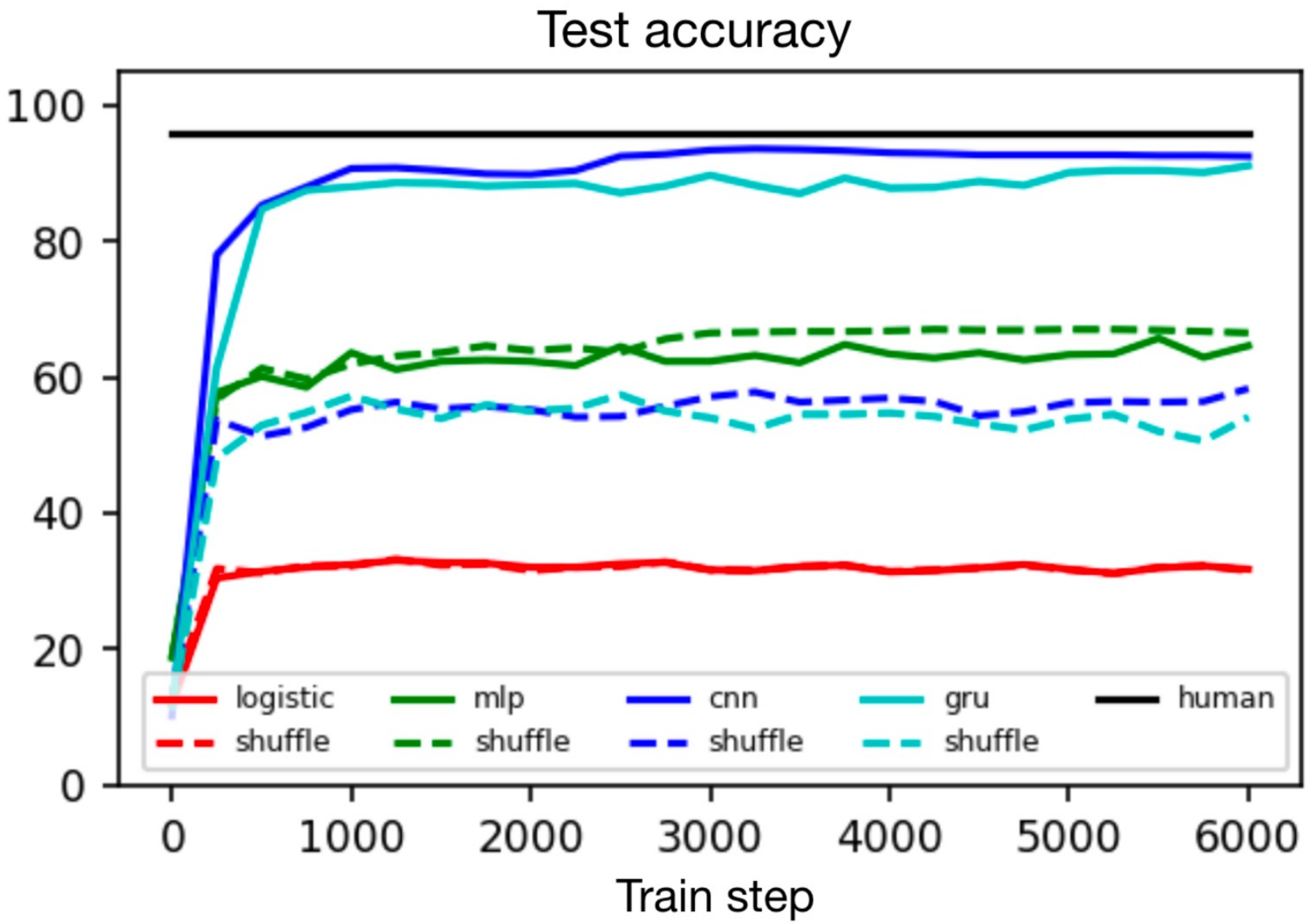
Generate dataset by programmatically applying 6 parametric transformations.

E.g. pad, shear, translate, correlated noise, i.i.d. noise, interpolation.

See https://github.com/greydanus/mnist1d/blob/master/building_mnist1d.ipynb

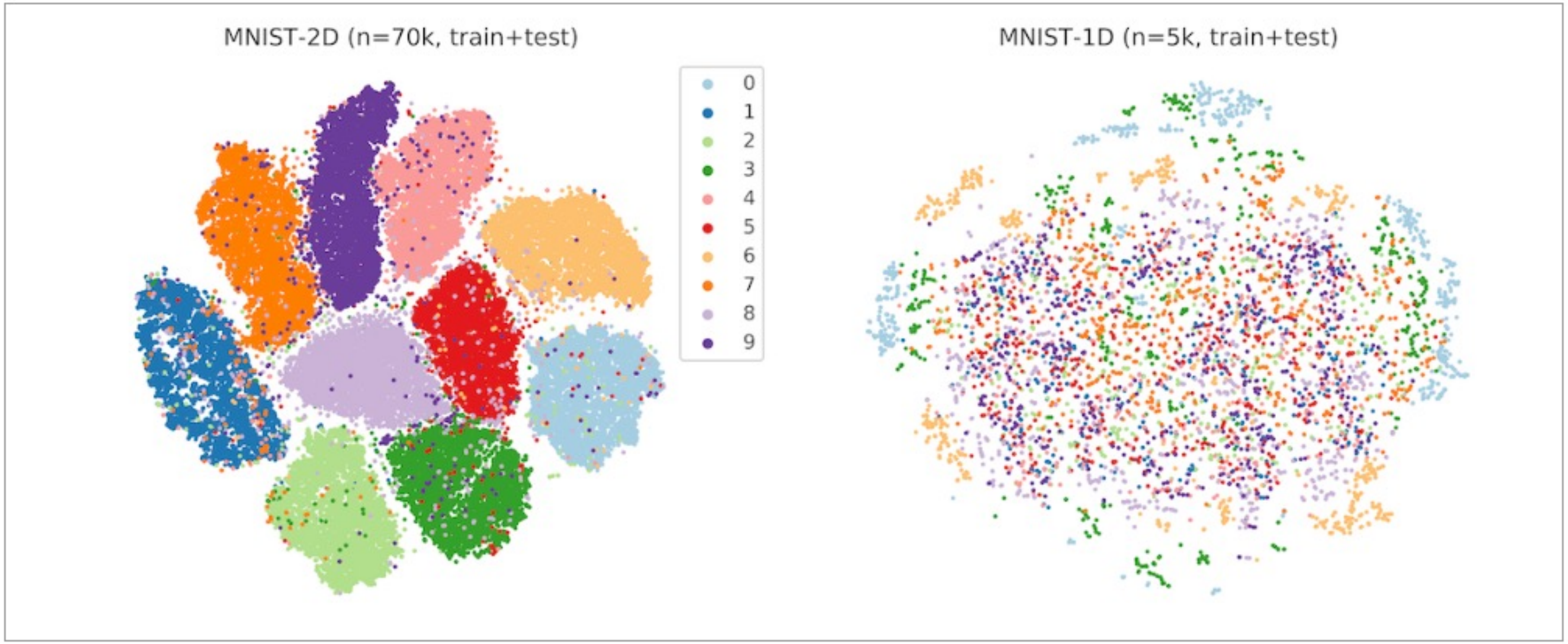
MNIST 1D

Differentiates performance of different model types much more than MNIST



Dataset	Logistic regression	Fully connected model	Convolutional model	GRU model	Human expert
MNIST	94 ± 0.5	> 99	> 99	> 99	> 99
MNIST-1D	32 ± 1	68 ± 2	94 ± 2	91 ± 2	96 ± 1
MNIST-1D (shuffled)	32 ± 1	68 ± 2	56 ± 2	57 ± 2	$\approx 30 \pm 10$

Visualizing MNIST and MNIST-1D with tSNE



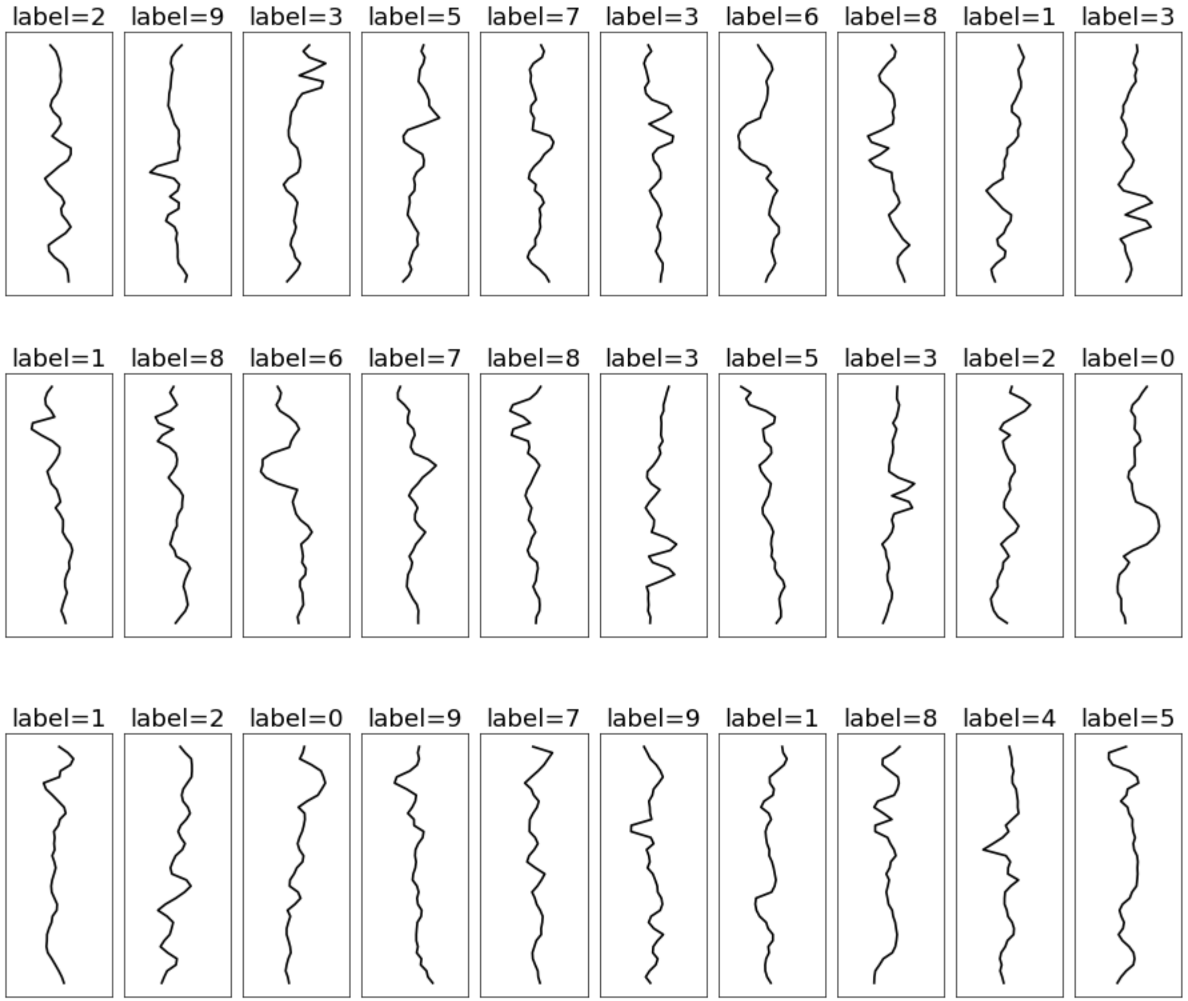
Visualizing the MNIST and MNIST-1D datasets with tSNE. The well-defined clusters in the MNIST plot indicate that the majority of the examples are separable via a kNN classifier in pixel space. The MNIST-1D plot, meanwhile, reveals a lack of well-defined clusters which suggests that learning a nonlinear representation of the data is much more important to achieve successful classification. Thanks to Dmitry Kobak for making this plot.

https://twitter.com/hippopedoid

MNIST1D Train and Test Set

- 1D, Length 40 samples
- 4,000 training samples
- 1,000 test samples (80/20 split)

Dataset Samples



Network

- 40 inputs
- 10 outputs
- Two hidden layers
  - 100 hidden units each
- SGD with batch size 100, learning rate 0.1
- 6000 steps (?? Epochs)

```
model = torch.nn.Sequential(
    torch.nn.Linear(40, 100),
    torch.nn.ReLU(),
    torch.nn.Linear(100, 100),
    torch.nn.ReLU(),
    torch.nn.Linear(100, 10))
```

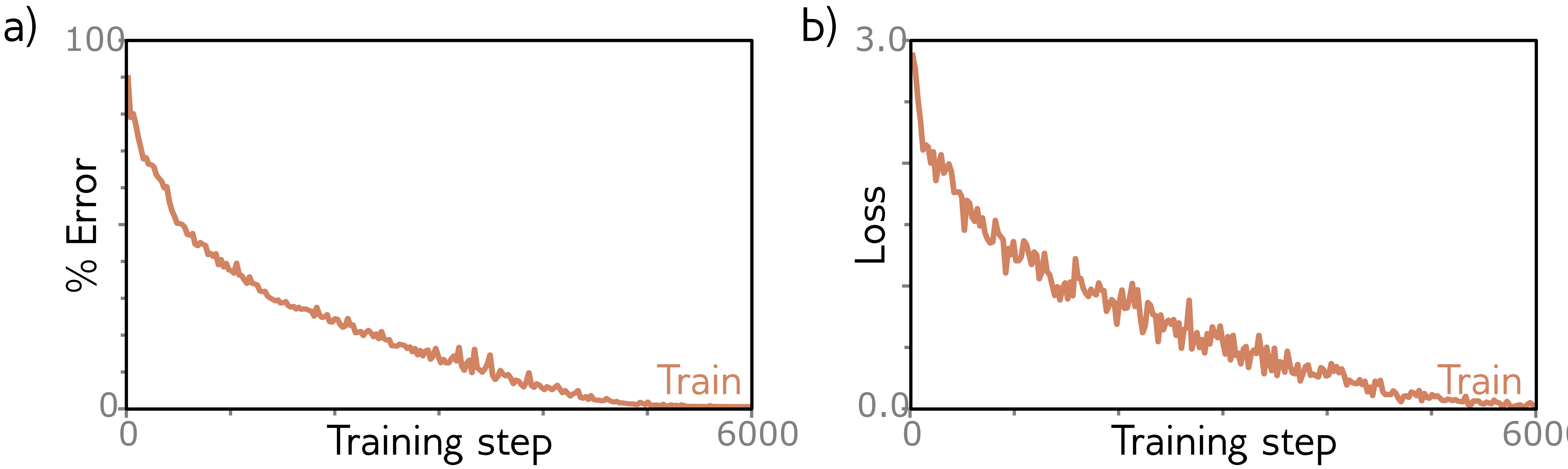
```
==========================================================
Layer (type:depth-idx)  Output Shape     Param #
==========================================================
Sequential              [1, 10]          –
├─Linear: 1-1           [1, 100]         4,100
├─ReLU: 1-2             [1, 100]         –
├─Linear: 1-3           [1, 100]         10,100
├─ReLU: 1-4             [1, 100]         –
├─Linear: 1-5           [1, 10]          1,010
==========================================================
Total params: 15,210
Trainable params: 15,210
Non-trainable params: 0
Total mult-adds (Units.MEGABYTES): 0.02
==========================================================
Input size (MB): 0.00
Forward/backward pass size (MB): 0.00
Params size (MB): 0.06
Estimated Total Size (MB): 0.06
==========================================================
```

```
# choose cross entropy loss function
loss_function = torch.nn.CrossEntropyLoss()

# construct SGD optimizer and initialize learning rate and momentum
optimizer = torch.optim.SGD(model.parameters(), lr = 0.1)

# object that decreases learning rate by half every 10 epochs
scheduler = StepLR(optimizer, step_size=10, gamma=0.5)

# load the data into a class that creates the batches
data_loader = DataLoader(TensorDataset(x_train,y_train), batch_size=100, shuffle=True)
```

...

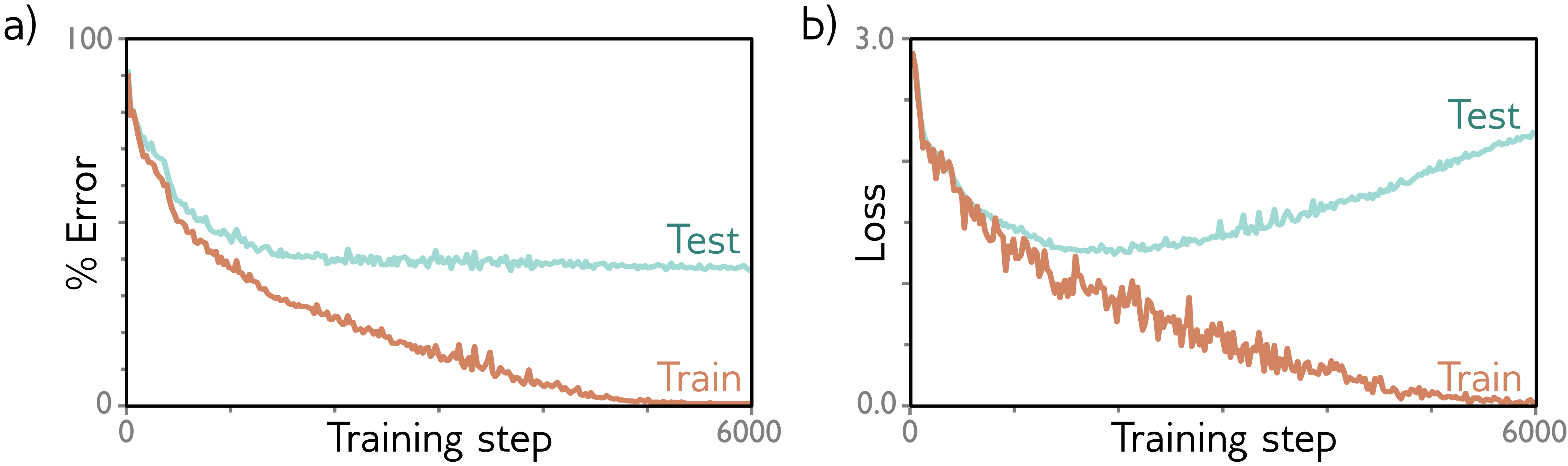
```
# inference – just choose the max
pred_train = model(x_train)
pred_test = model(x_test)
_, predicted_train_class = torch.max(pred_train.data, 1)
_, predicted_test_class = torch.max(pred_test.data, 1)
```

Results



a)
100
% Error
0
0
Training step
6000
Train
b)
3.0
Loss
0.0
0
Training step
600
Train

Need to use separate test data

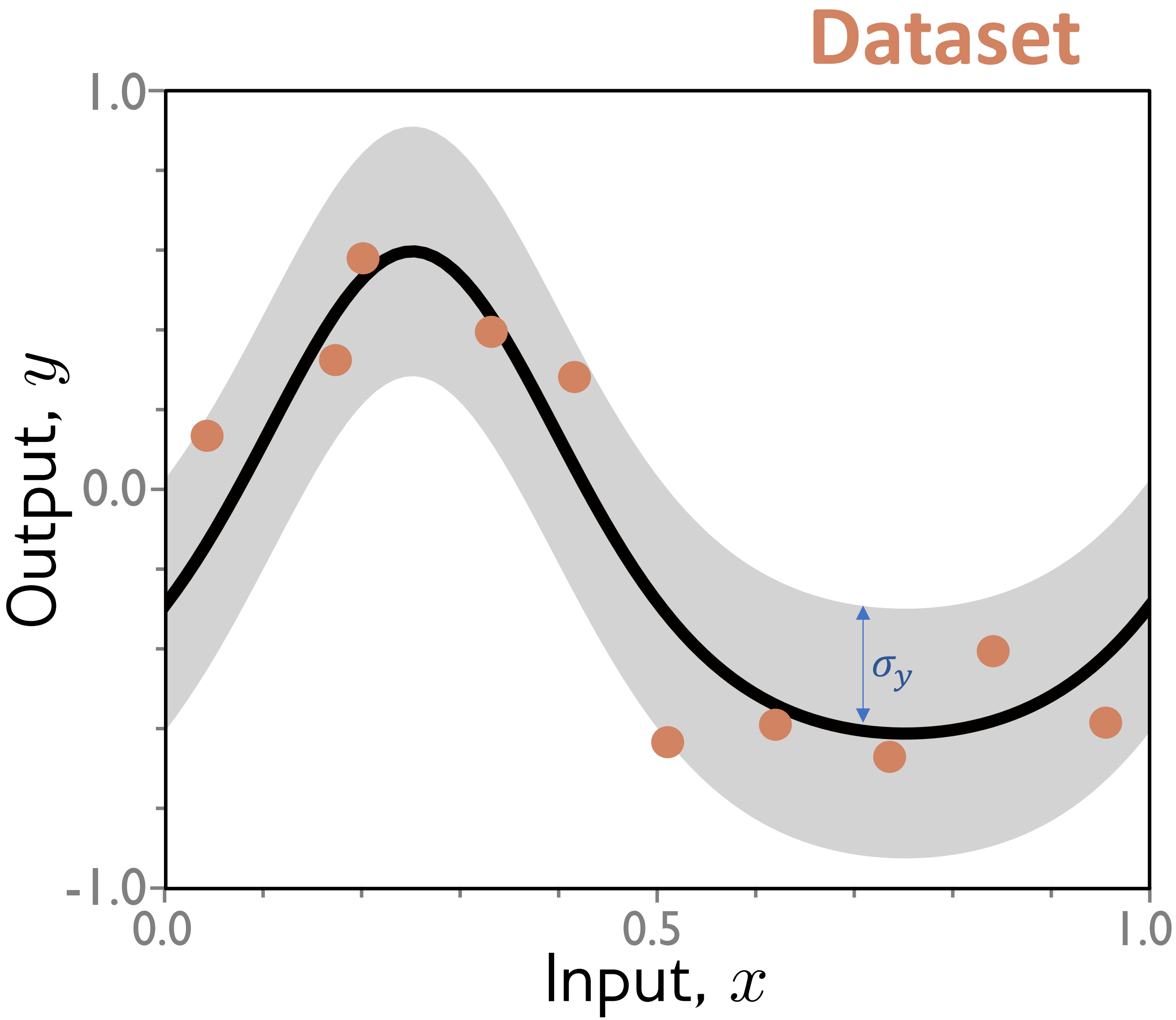


The model has not generalized well to the new data

Measuring performance

- MNIST1D dataset model and performance
- Noise, bias, and variance
- Reducing variance
- Reducing bias & bias-variance trade-off
- Double descent
- Curse of dimensionality & weird properties of high dimensional space
- Choosing hyperparameters

Regression example with Toy Model



"True" function:

$$y = e^{\sin(2\pi x)}$$

Add small uniform noise to x:

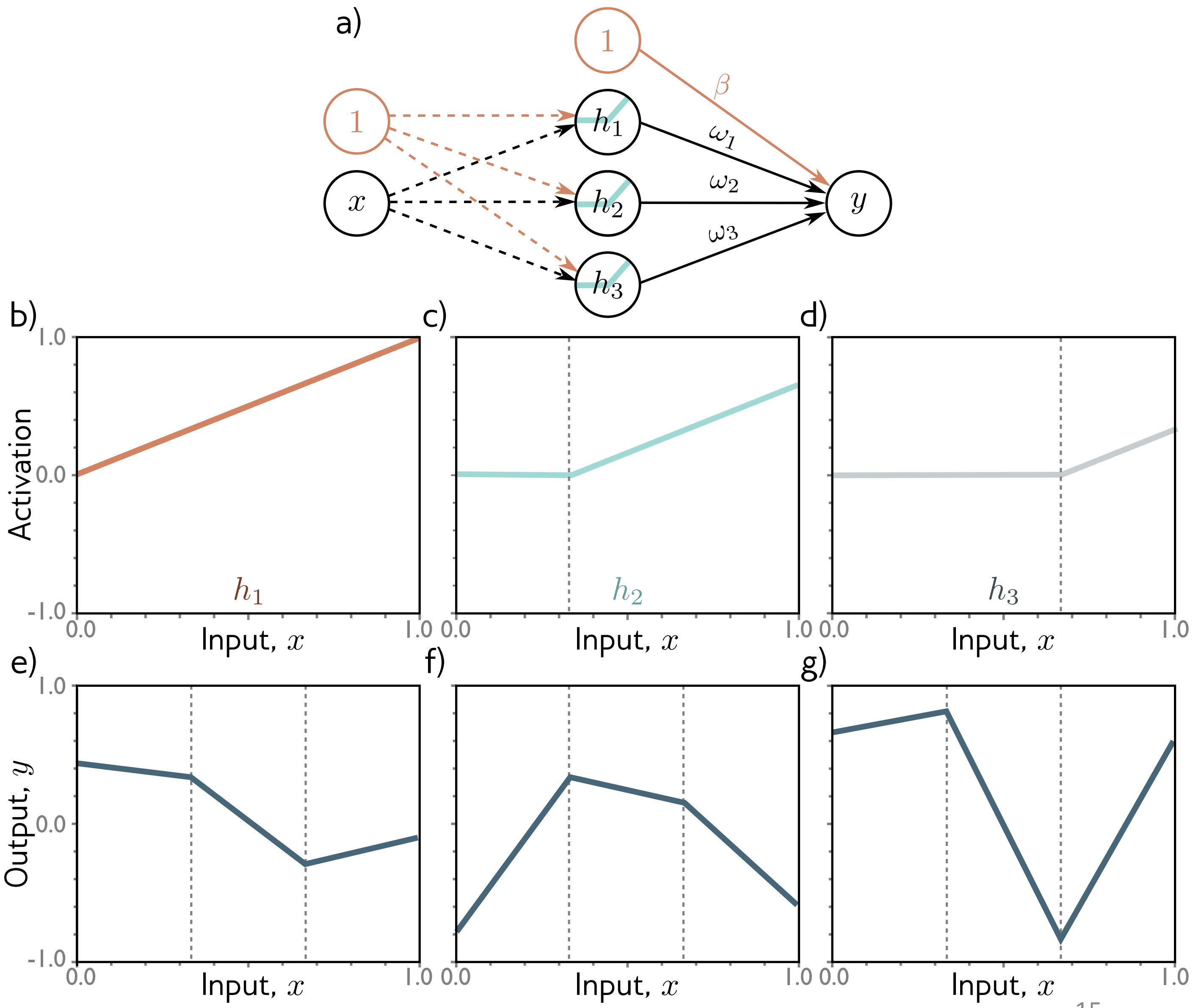
$$x = x + \mathcal{U}(\pm 1/\mathrm{num\_data})$$

Add small Gaussian noise to y:

$$y = y + \mathcal{N}(0, \sigma_y)$$

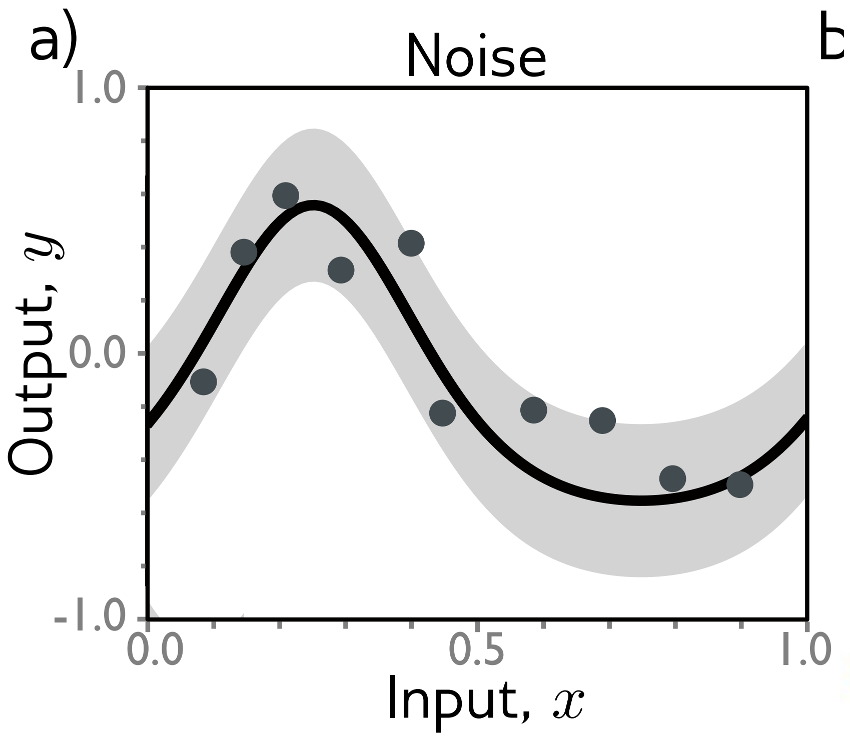
Toy model

- D hidden units
- First layer fixed so "joints" divide interval evenly, e.g. 0, 1/D, 2/D, …, (D-1)/D
- Second layer trained
- But… now linear in **h**
  - so convex cost function
  - can find best soln in closed-form
- A piecewise linear model with D regions.

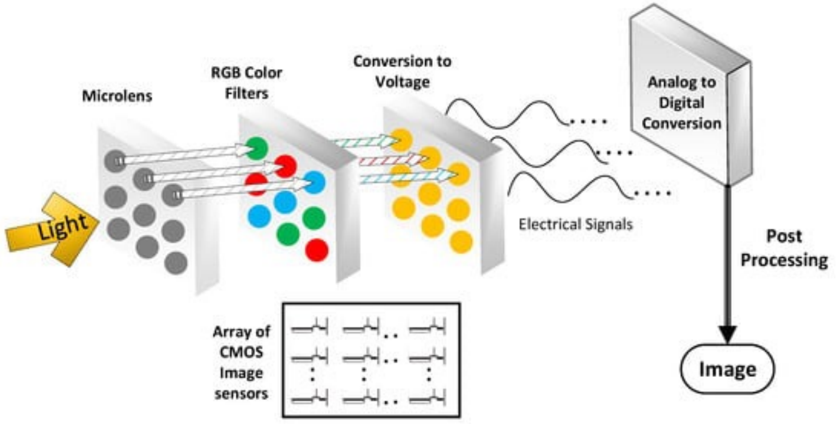


Three possible sources of error: *noise*, *bias* and *variance*

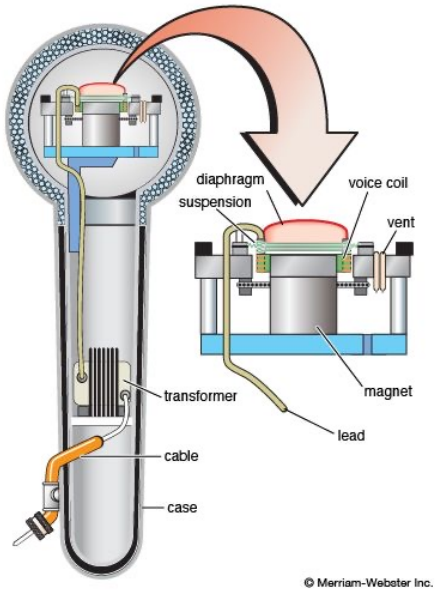
Noise, bias, and variance



- Genuine stochastic nature of the underlying model
- Noise in measurements, e.g. from sensors
- Some variables not observed
- Data mislabeled

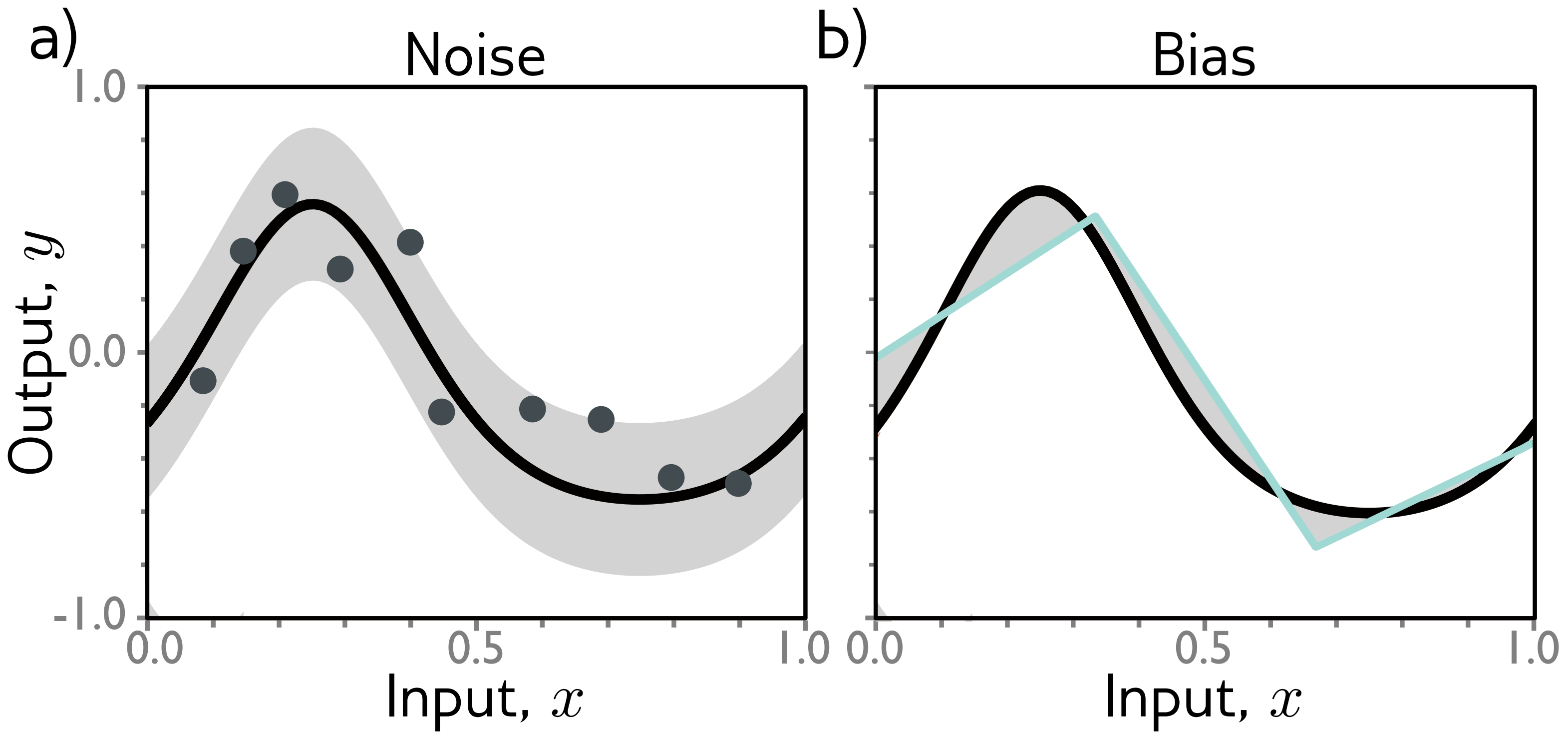


https://images.app.goo.gl/2PuBhaFpfdL9Pyjb8



https://images.app.goo.gl/CMDaXSCdX4pqN8Yx7

Noise, bias, and variance



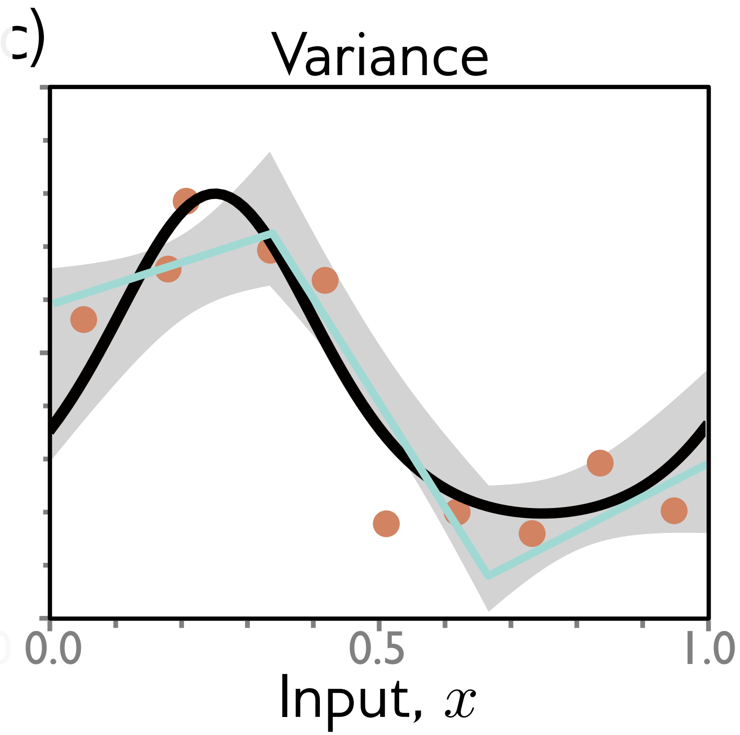
Bias occurs because the model lacks precision or capacity to accurately match the underlying function.

E.g. optimal fit with 3 hidden units and 3 line segments

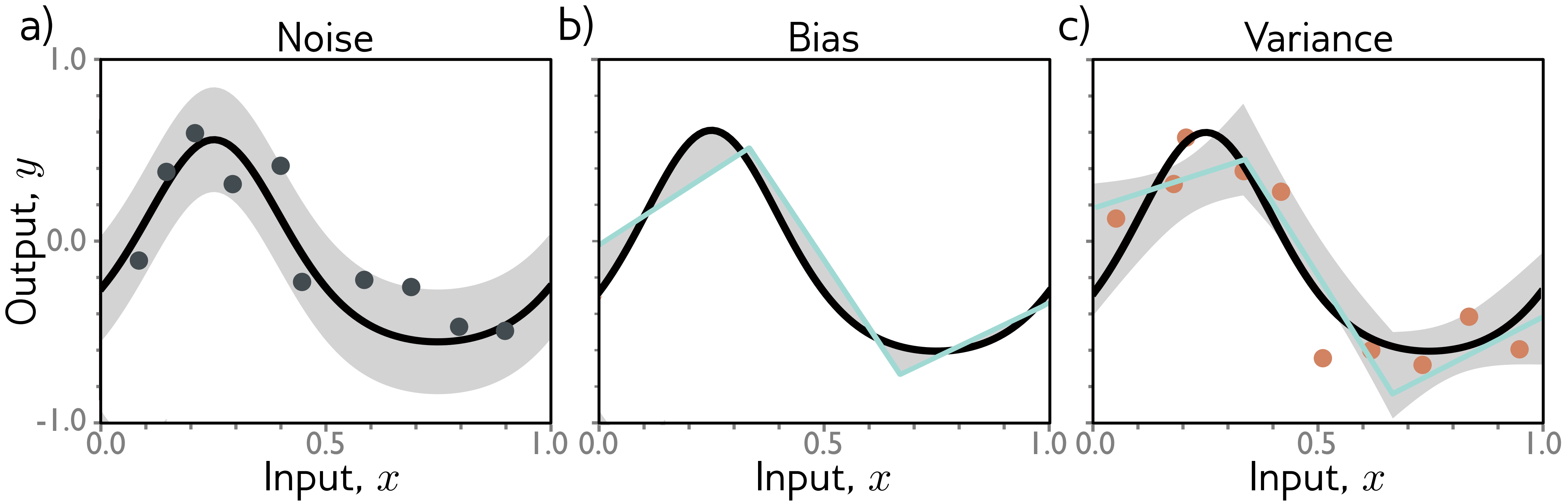
Noise, bias, and variance

No way to distinguish change in the true underlying function from noise in the data.

Variability every time we capture training data and also from stochastic learning algorithms.



Noise, bias, and variance



For derivation see Section 8.2.2 in UDL.

Least squares regression only

$$L[x] = \left(\mathrm{f}[x, \boldsymbol{\phi}] - y[x]\right)^2$$

- We can show that:

$$\mathbb{E}_y\big[L[x]\big] = \left(\mathrm{f}[x, \boldsymbol{\phi}] - \mu[x]\right)^2 + \sigma^2$$

- And then:

$$\mathbb{E}_{\mathcal{D}}\Big[\mathbb{E}_y[L[x]]\Big] = \underbrace{\mathbb{E}_{\mathcal{D}}\Big[\left(\mathrm{f}[x, \boldsymbol{\phi}[\mathcal{D}]] - f_\mu[x]\right)^2\Big]}_{\text{variance}} + \underbrace{\left(f_\mu[x] - \mu[x]\right)^2}_{\text{bias}} + \underbrace{\sigma^2}_{\text{noise}}$$

Expectation over noise in training data

Expectation over noise in test data

Actual model

Best possible model if we had infinite data

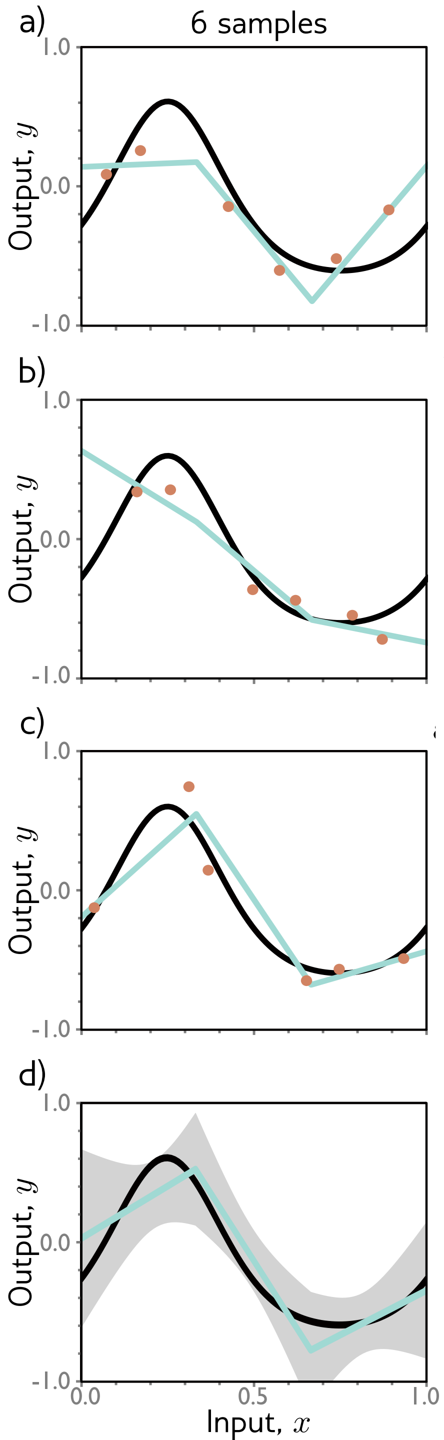
True function

More complex interactions between noise, bias and variance in more complex models.

Measuring performance

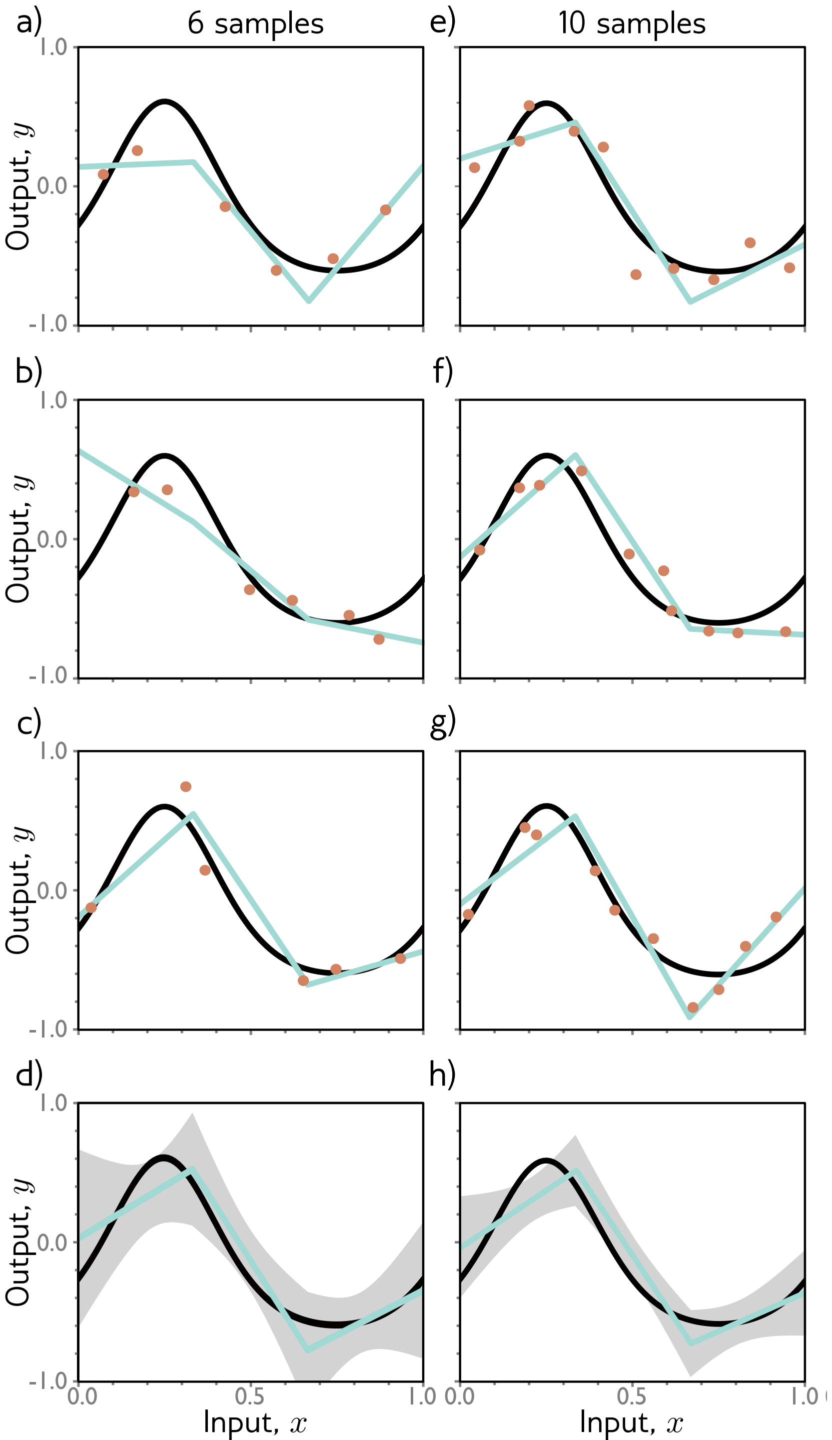
- MNIST1D dataset model and performance
- Noise, bias, and variance
- Reducing variance
- Reducing bias & bias-variance trade-off
- Double descent
- Curse of dimensionality & weird properties of high dimensional space
- Choosing hyperparameters

Variance



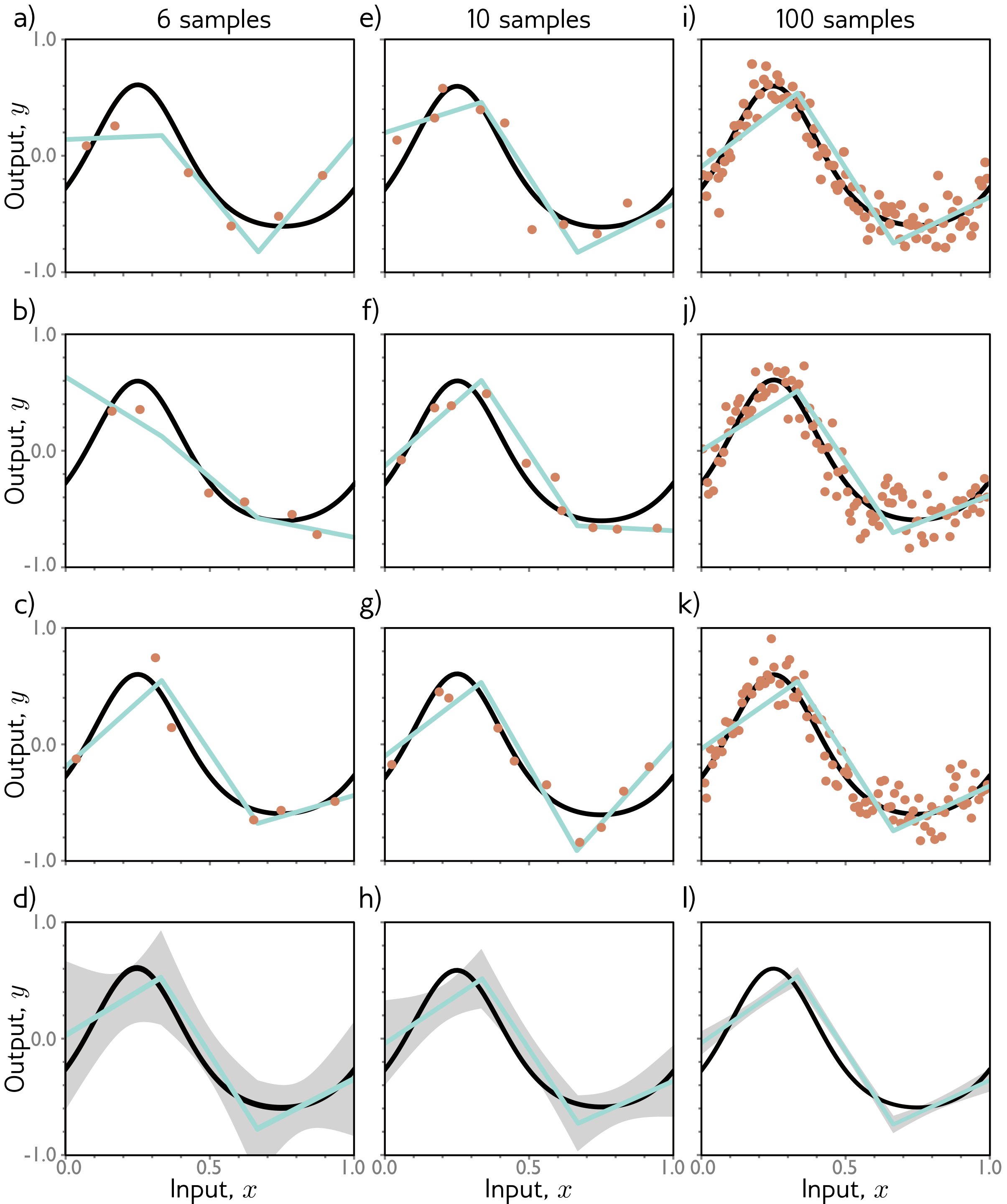
When measuring (capturing) 6 different data samples with a fixed model (e.g. 3 hidden units), we get different optimal fits every time.

Variance



Can reduce variance by adding more samples

Variance



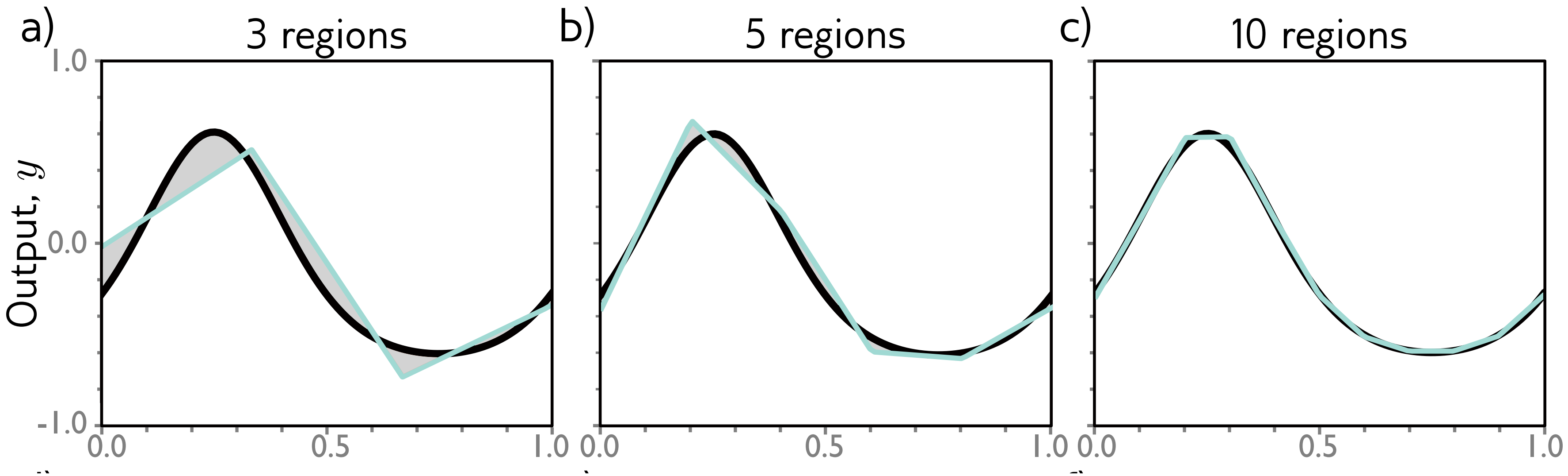
Can reduce variance by adding more samples

Measuring performance

- MNIST1D dataset model and performance
- Noise, bias, and variance
- Reducing variance
- Reducing bias & bias-variance trade-off
- Double descent
- Curse of dimensionality & weird properties of high dimensional space
- Choosing hyperparameters

Reducing bias
(example with the true function)

Bias



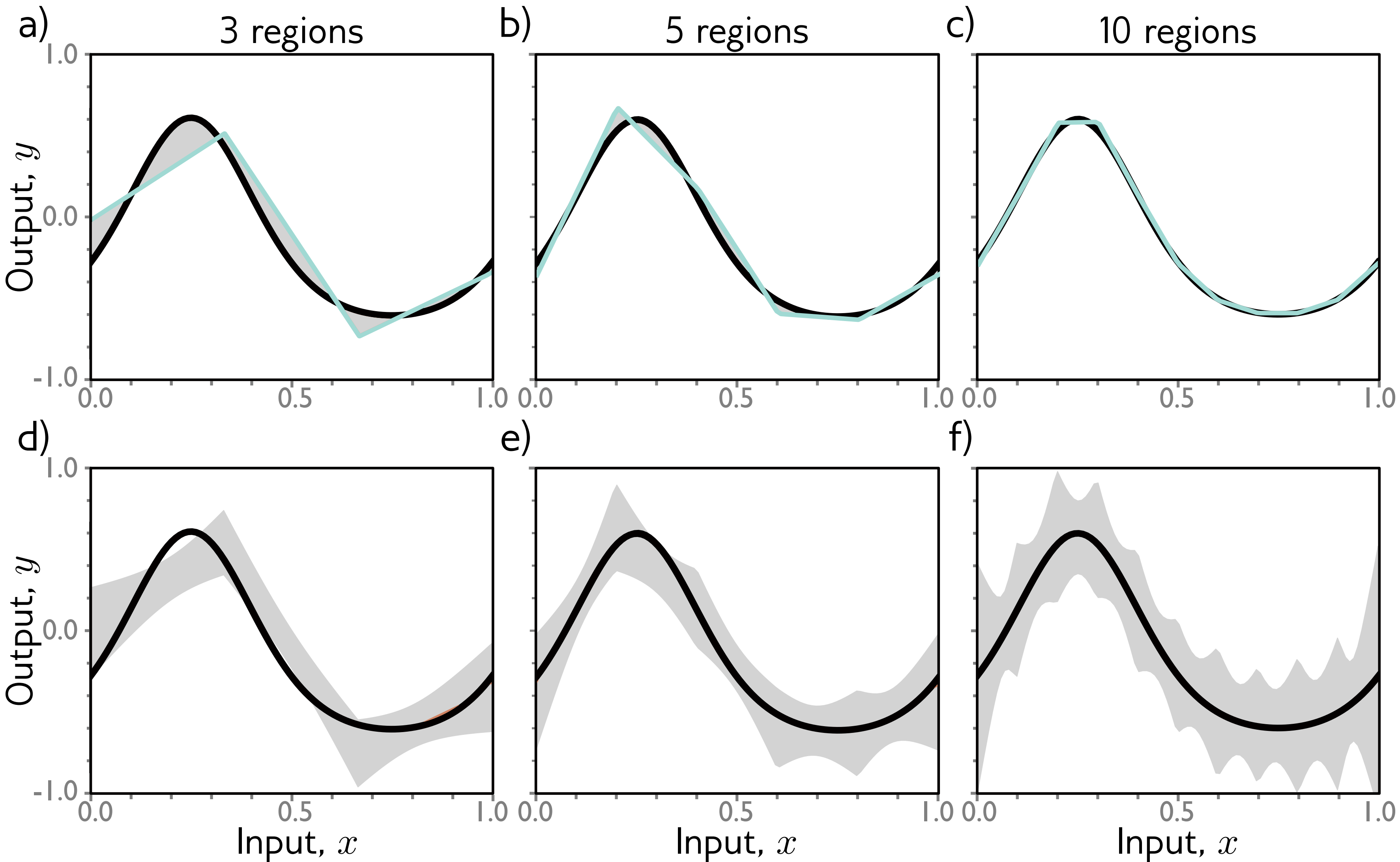
We can reduce bias by adding more model capacity.

In this case, adding more hidden units.

Reducing bias ➔ Increases variance!!

Bias

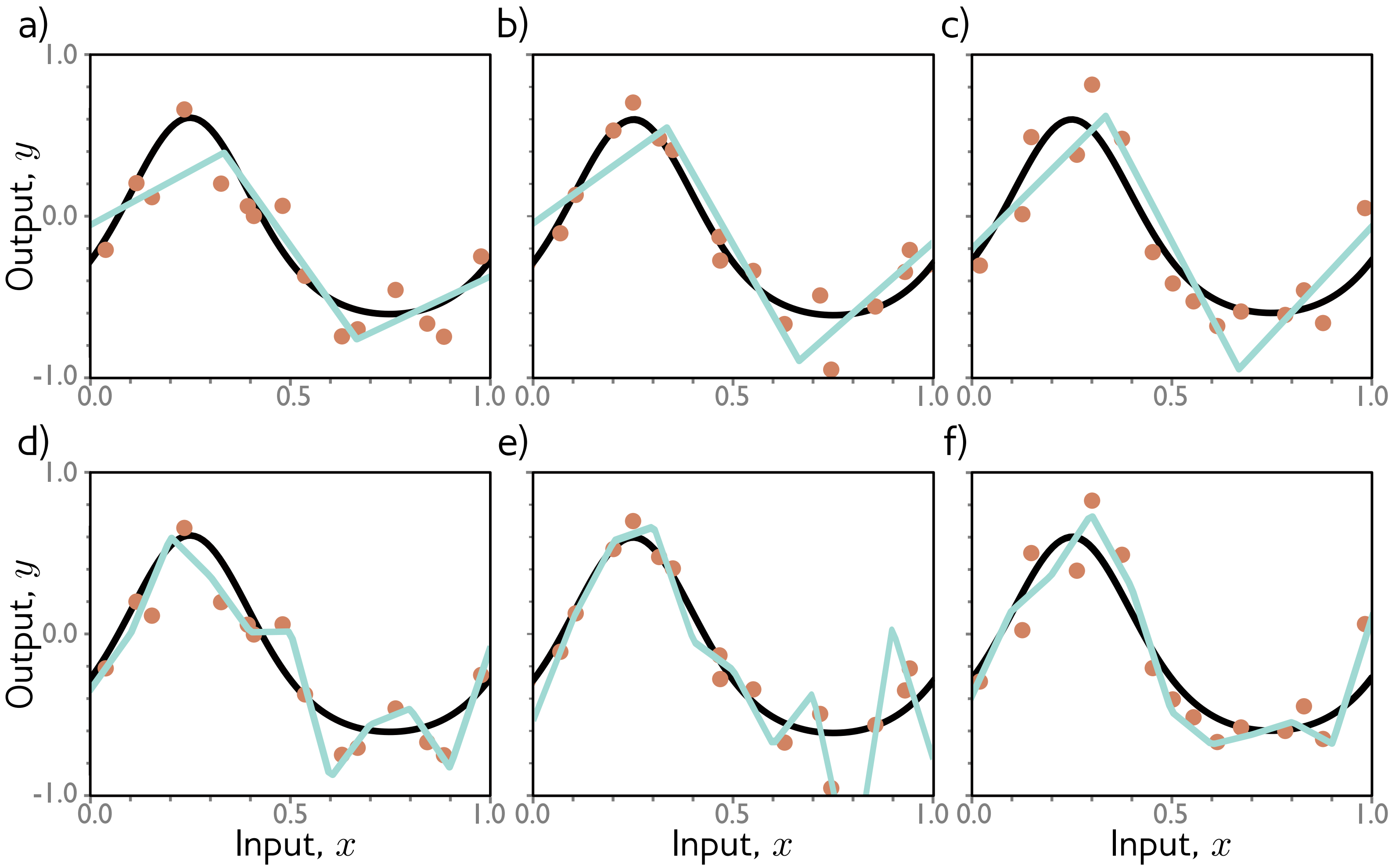
Variance



Why does variance increase? Overfitting

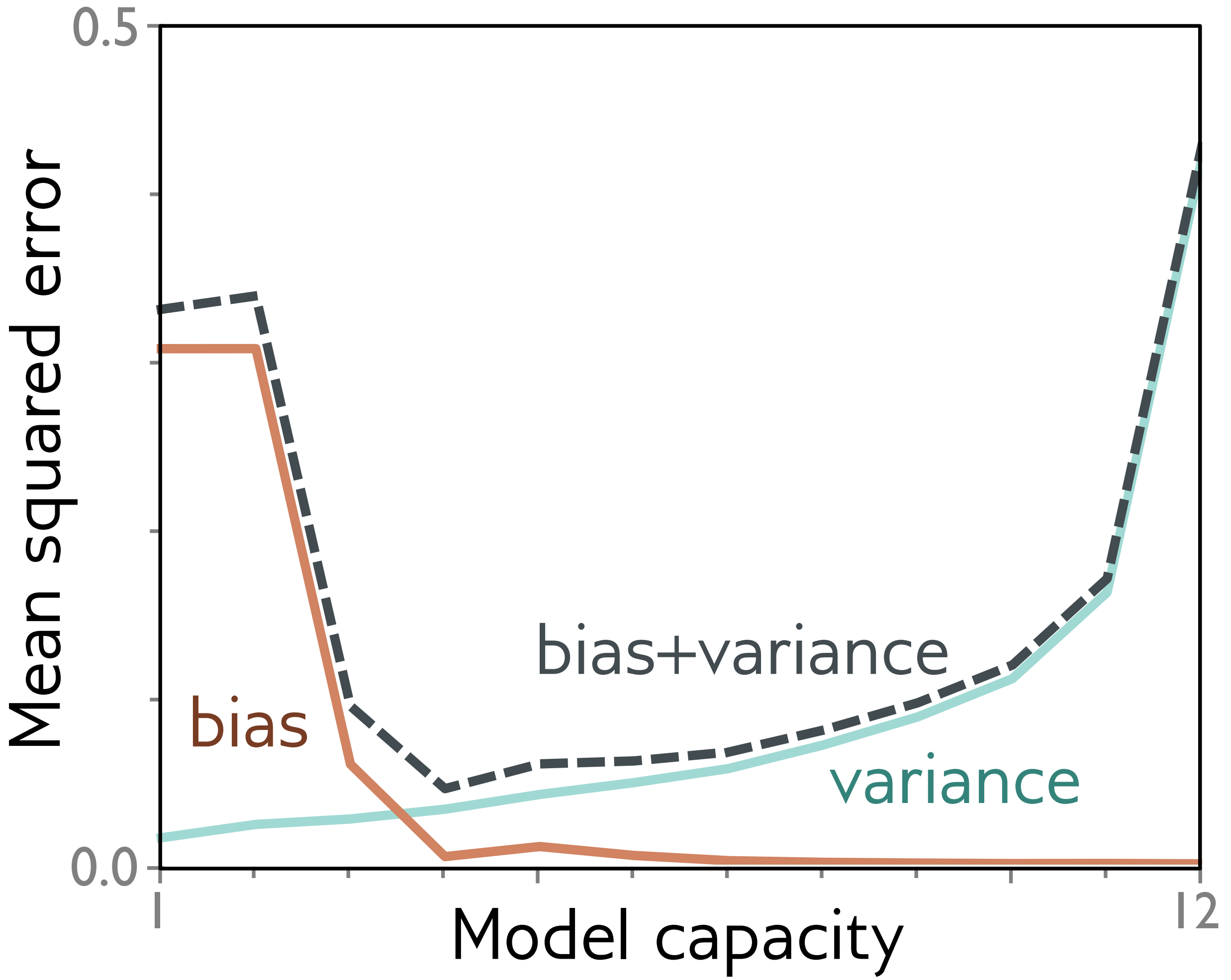
3 Regions

10 Regions

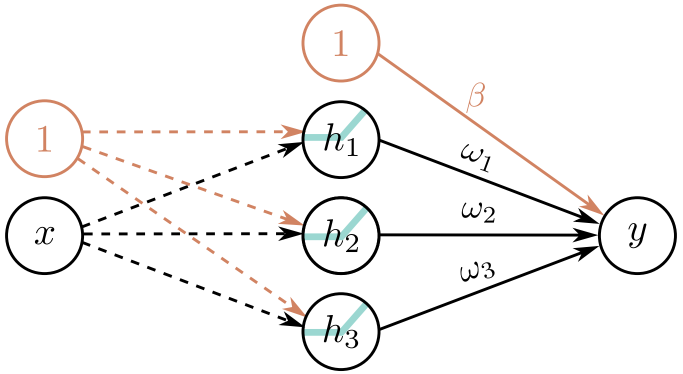


Describes the training data better, but not the true underlying function (black curve)
Many ways to fit a sample of 15 data points

Bias and variance trade-off for the simple linear model



Number of hidden units



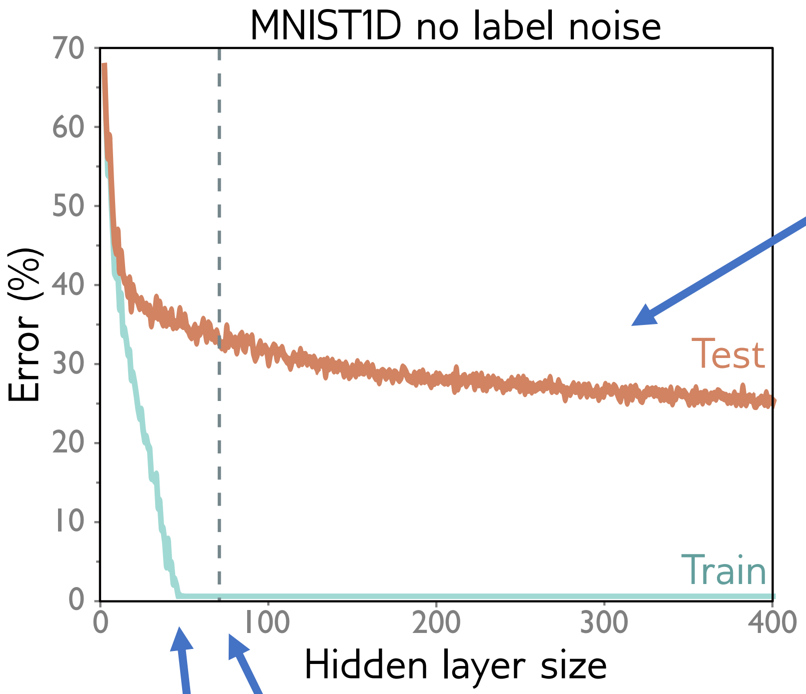
$$\mathbb{E}_{\mathcal{D}}\Big[\mathbb{E}_y[L[x]]\Big] = \underbrace{\mathbb{E}_{\mathcal{D}}\Big[\big(\mathrm{f}[x,\boldsymbol{\phi}[\mathcal{D}]] - f_\mu[x]\big)^2\Big]}_{\text{variance}} + \underbrace{\big(f_\mu[x]-\mu[x]\big)^2}_{\text{bias}} + \underbrace{\sigma^2}_{\text{noise}}$$

But does picking model capacity to minimize bias & variance hold for more complex data and models?

Measuring performance

- MNIST1D dataset model and performance
- Noise, bias, and variance
- Reducing variance
- Reducing bias & bias-variance trade-off
- Double descent
- Curse of dimensionality & weird properties of high dimensional space
- Choosing hyperparameters

Train and Test Error versus # of Hidden Layers



Test error keep decreasing even as we keep increasing model capacity!

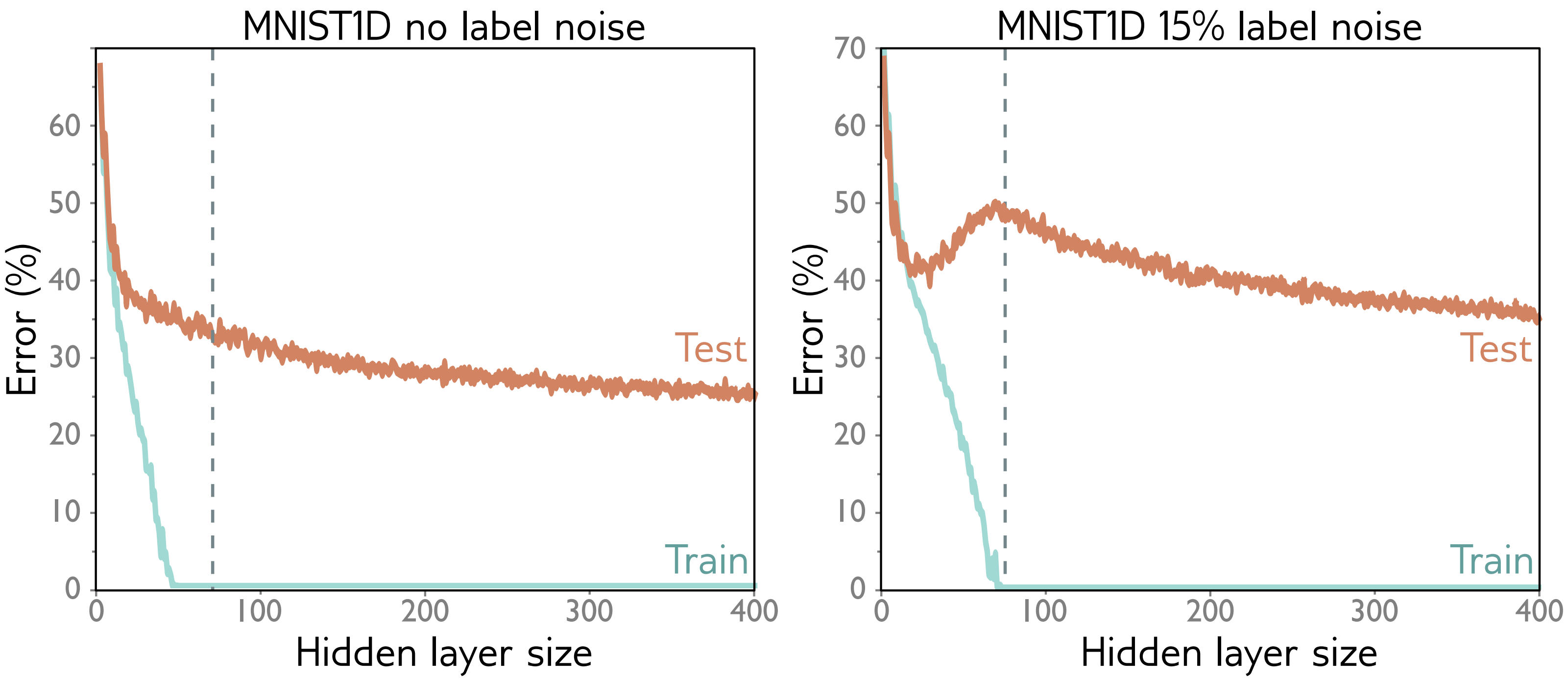
Training parameters = Training examples

Model has *memorized* the training set

Why do we say that?

- 10,000 training examples
- 5,000 test examples
- Two hidden layers
- Adam optimizer
- Step size of 0.005
- Full batch
- 4000 training steps

Now randomize 15% of the training labels

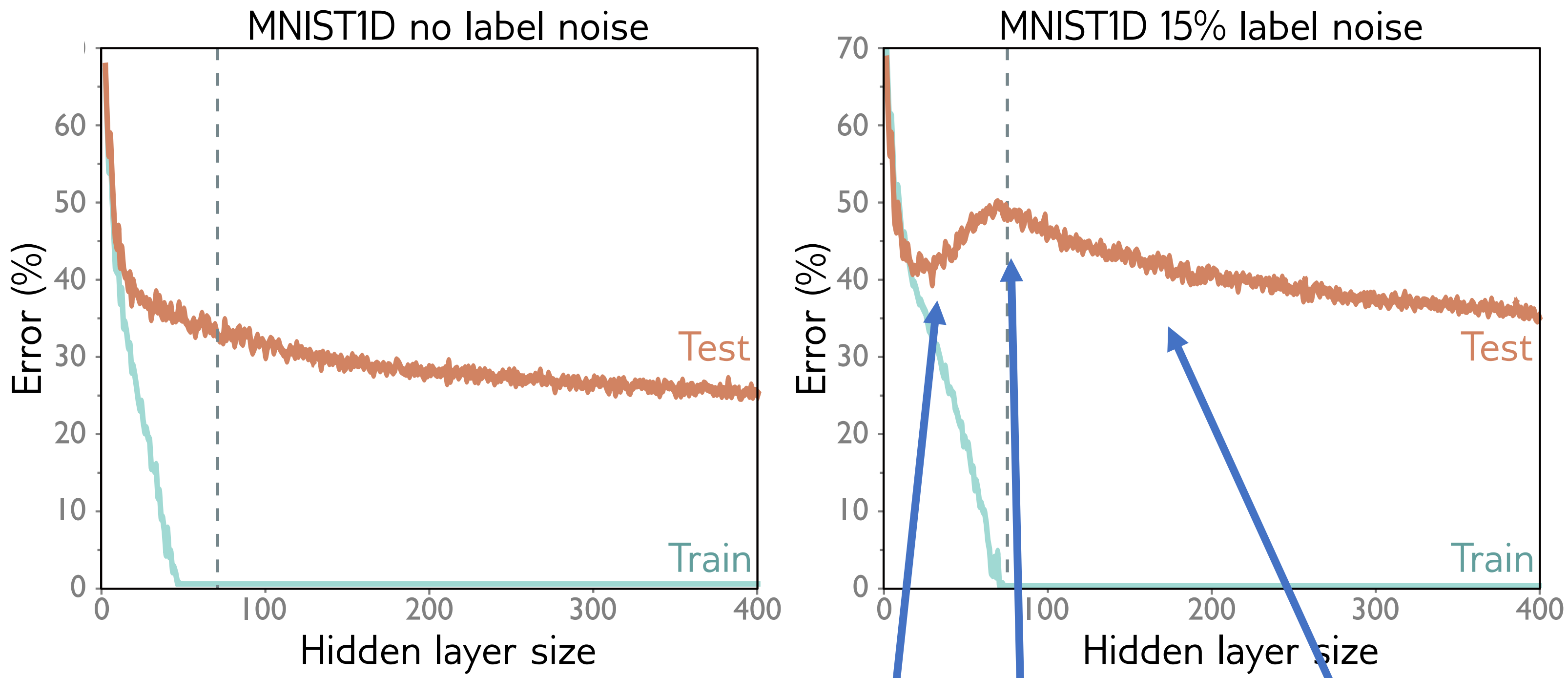


Now we see what looks like bias-variance trade-off as we increase capacity to the point where the model fits training data.

But then???

Reminder: vertical dashed line is where:
training parameters = # training samples

Double Descent



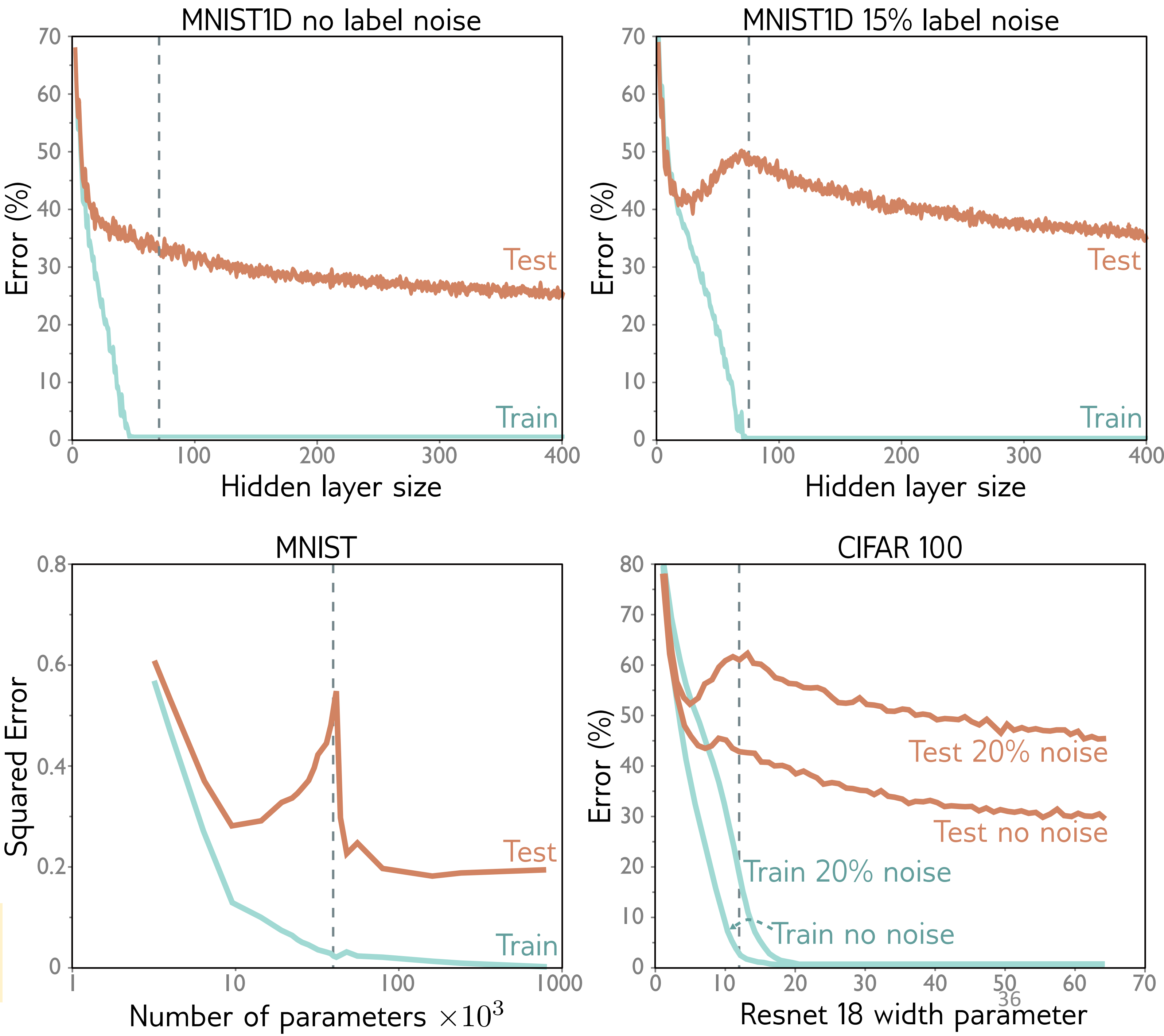
Classical or under-parameterized regime

Critical regime

Modern or over-parameterized regime

Reminder: vertical dashed line is where:
training parameters = # training samples

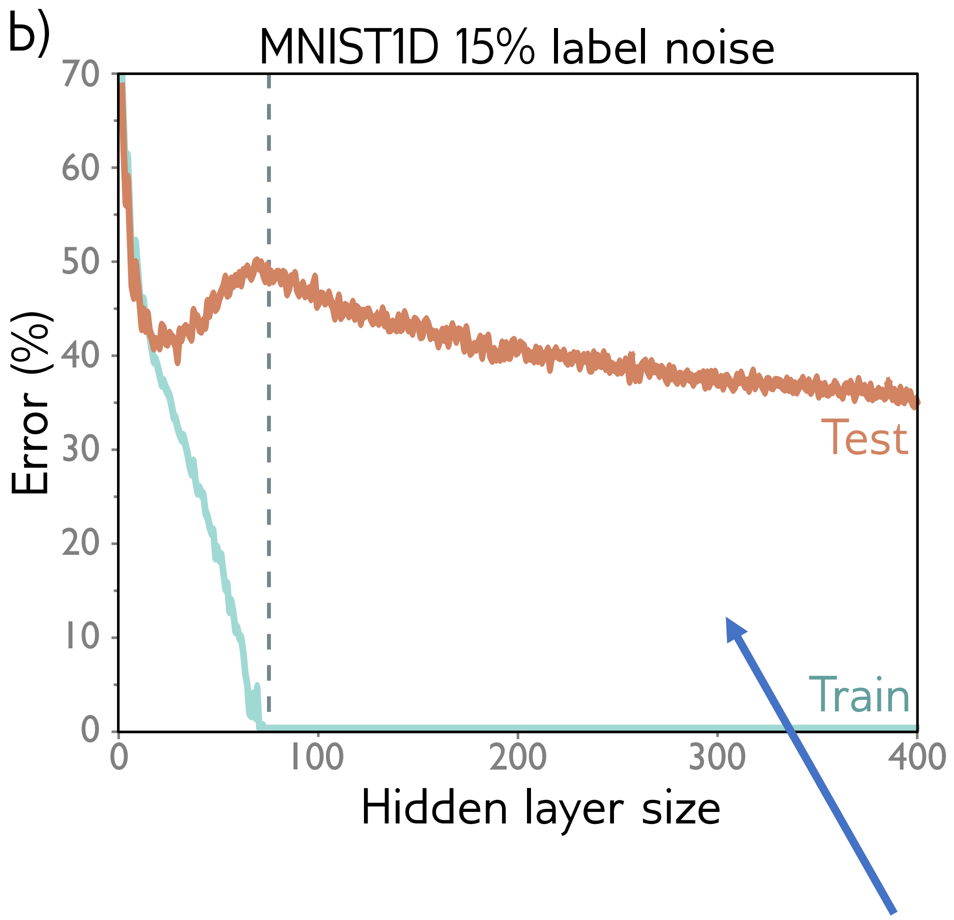
Same phenomenon shows up on MNIST and CIFAR100

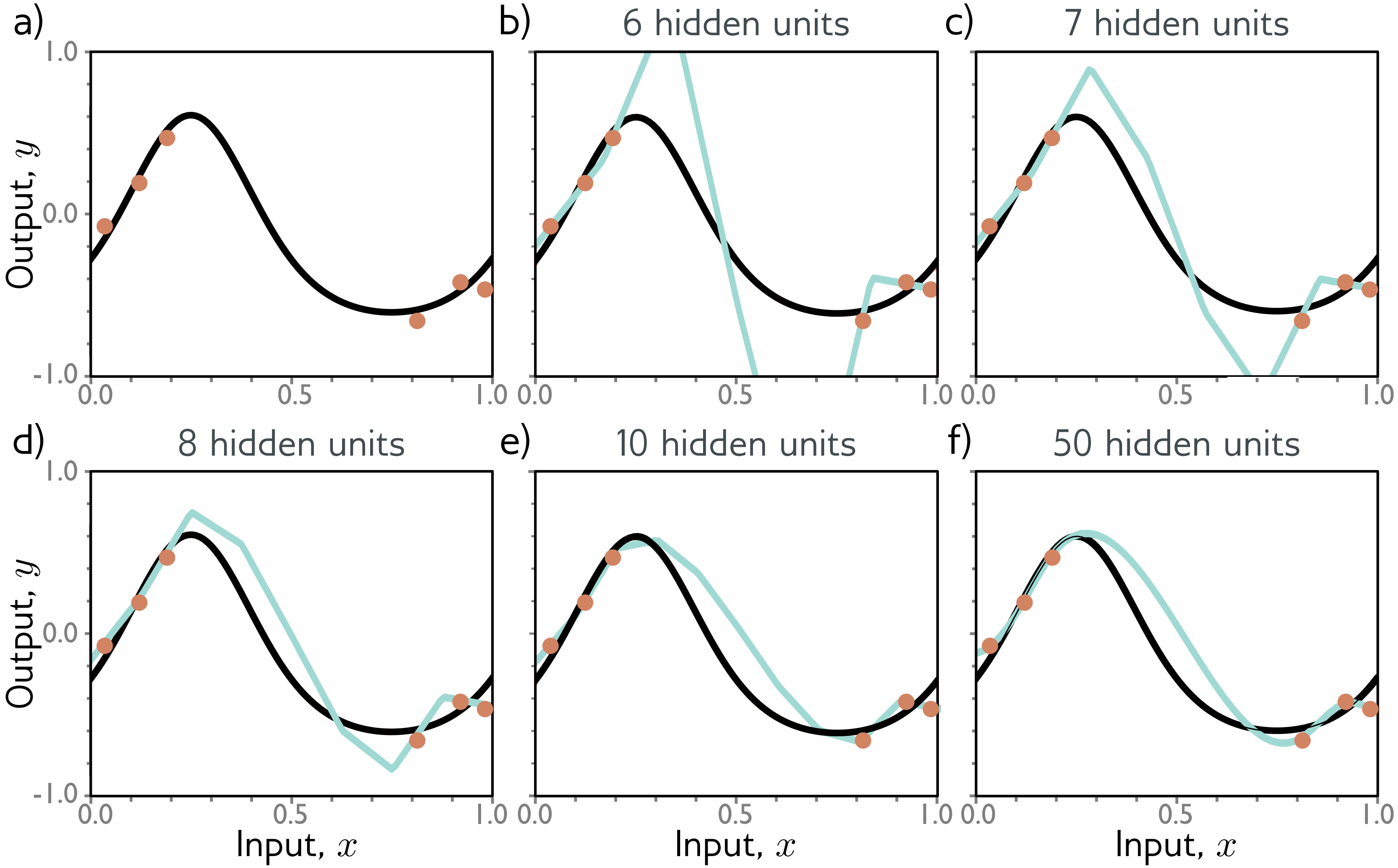


Reminder: vertical dashed line is where:
training parameters = # training samples

Double Descent

- Note that training loss is very close to zero.
- Whatever is happening isn't happening at training data points
- Model never sees test set during training
- Must be happening between the data points??

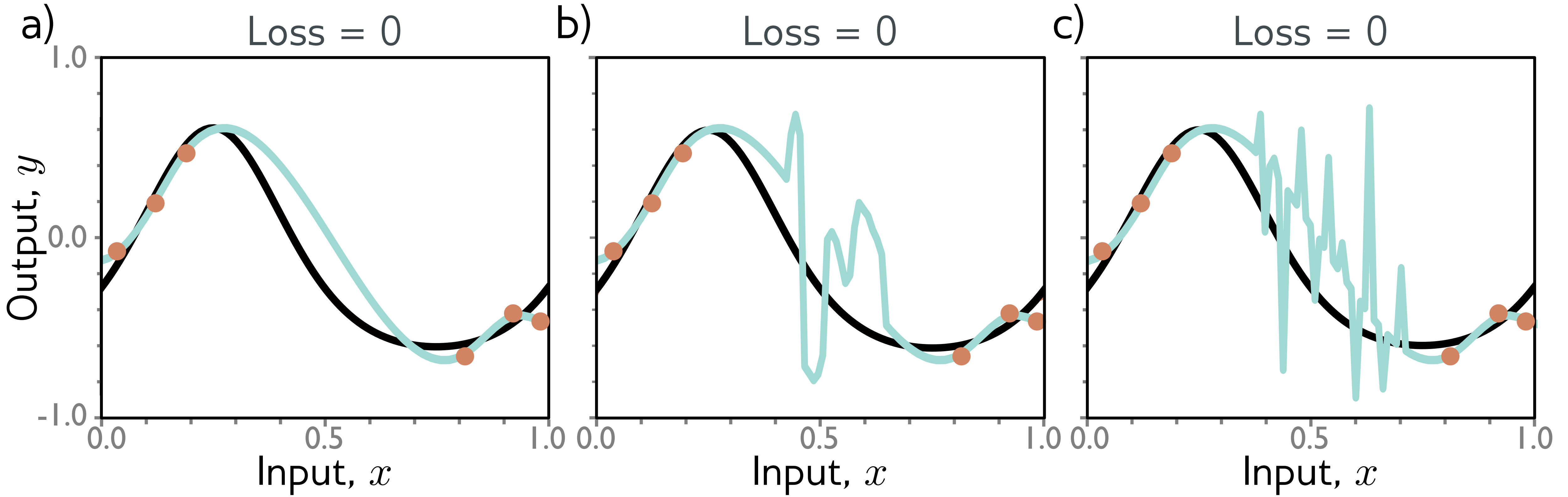




Potential explanation:

- can make smoother functions with more hidden units
- being smooth between the datapoints is a reasonable thing to do

But why?



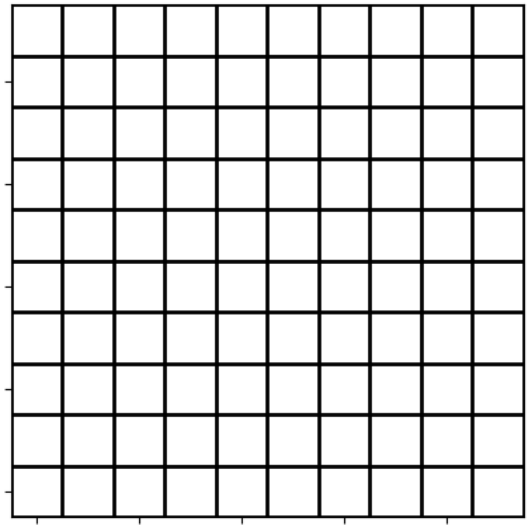
- All of these solutions are equivalent in terms of loss.
- Why should the model choose the smooth solution?
- Tendency of model to choose one solution over another is inductive bias

Measuring performance

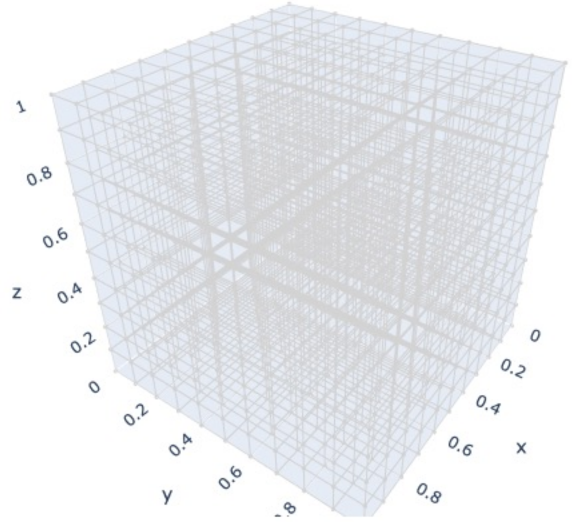
- MNIST1D dataset model and performance
- Noise, bias, and variance
- Reducing variance
- Reducing bias & bias-variance trade-off
- Double descent
- Curse of dimensionality & weird properties of high dimensional space
- Choosing hyperparameters

Curse of dimensionality

- 40-dimensional data
- 10,000 data points
- Consider quantizing each dimension into 10 bins
- 10^{40} bins
- 1 data point per 10^{35} bins
- The tendency of high-dimensional space to overwhelm the number of data points is called the curse of dimensionality



2D: 10x10=100 bins

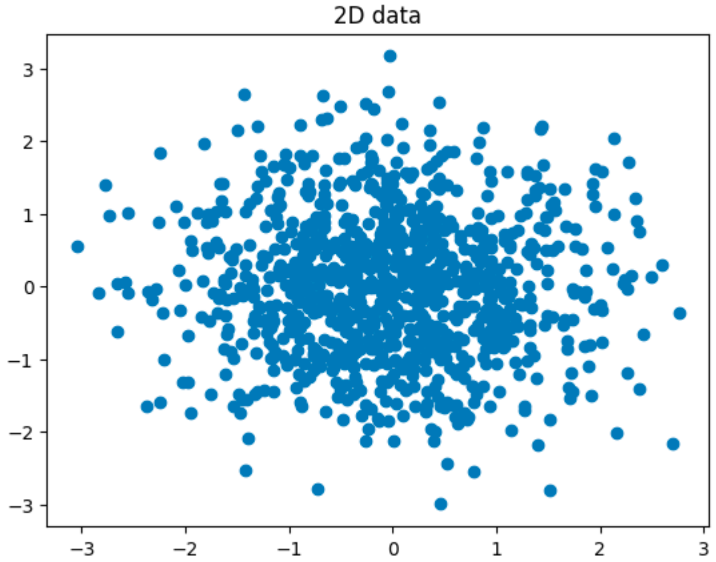


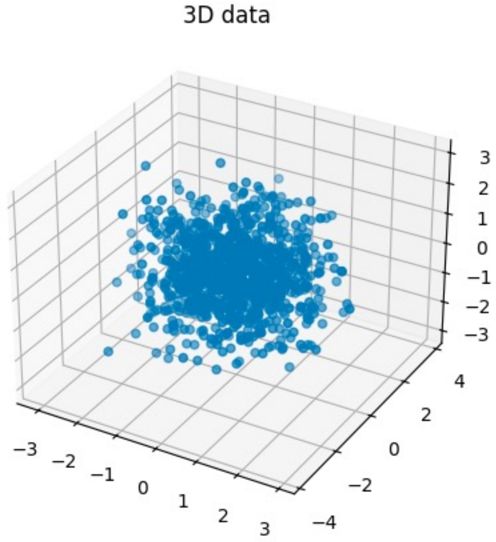
3D: 10x10x10=1000 bins

Curse: Distances collapse

Generate 1,000 normally distributed samples in:

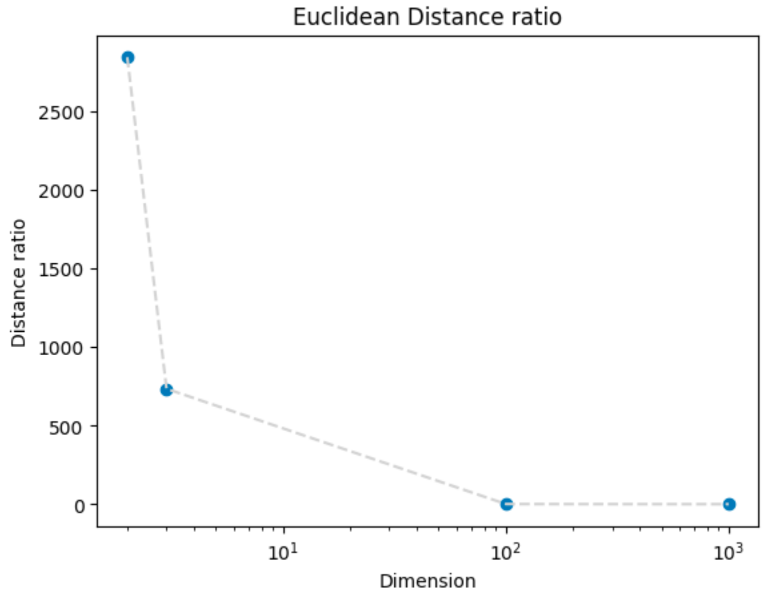
- 2D
- 3D
- 100D
- 1000D





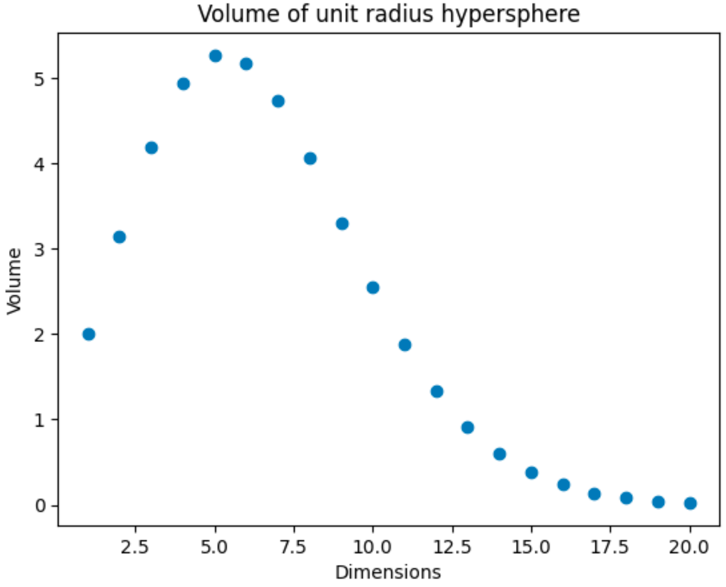
...

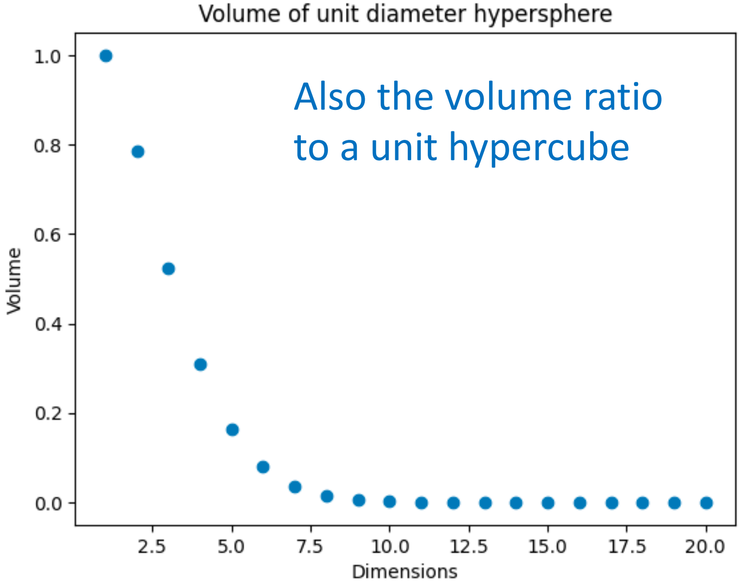
Calculate the ratio of distances between the farthest and closest points.

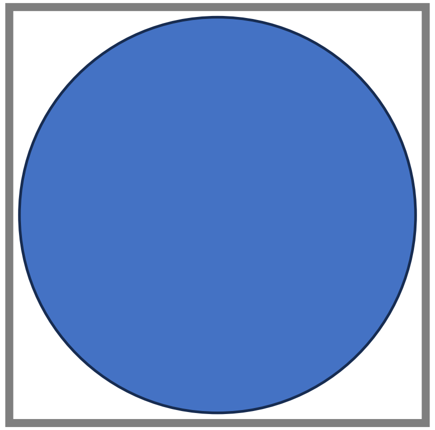


Approaches 1!!

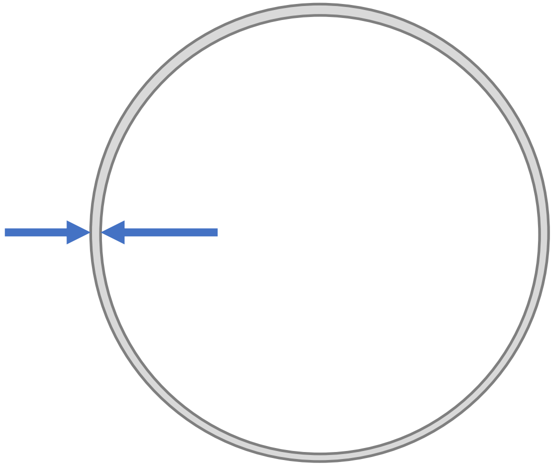
Curse: Volumes of a hyperspheres

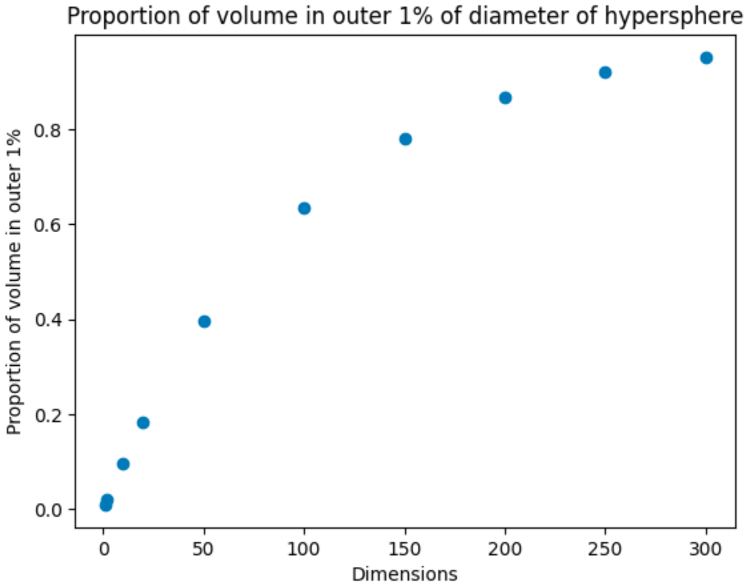






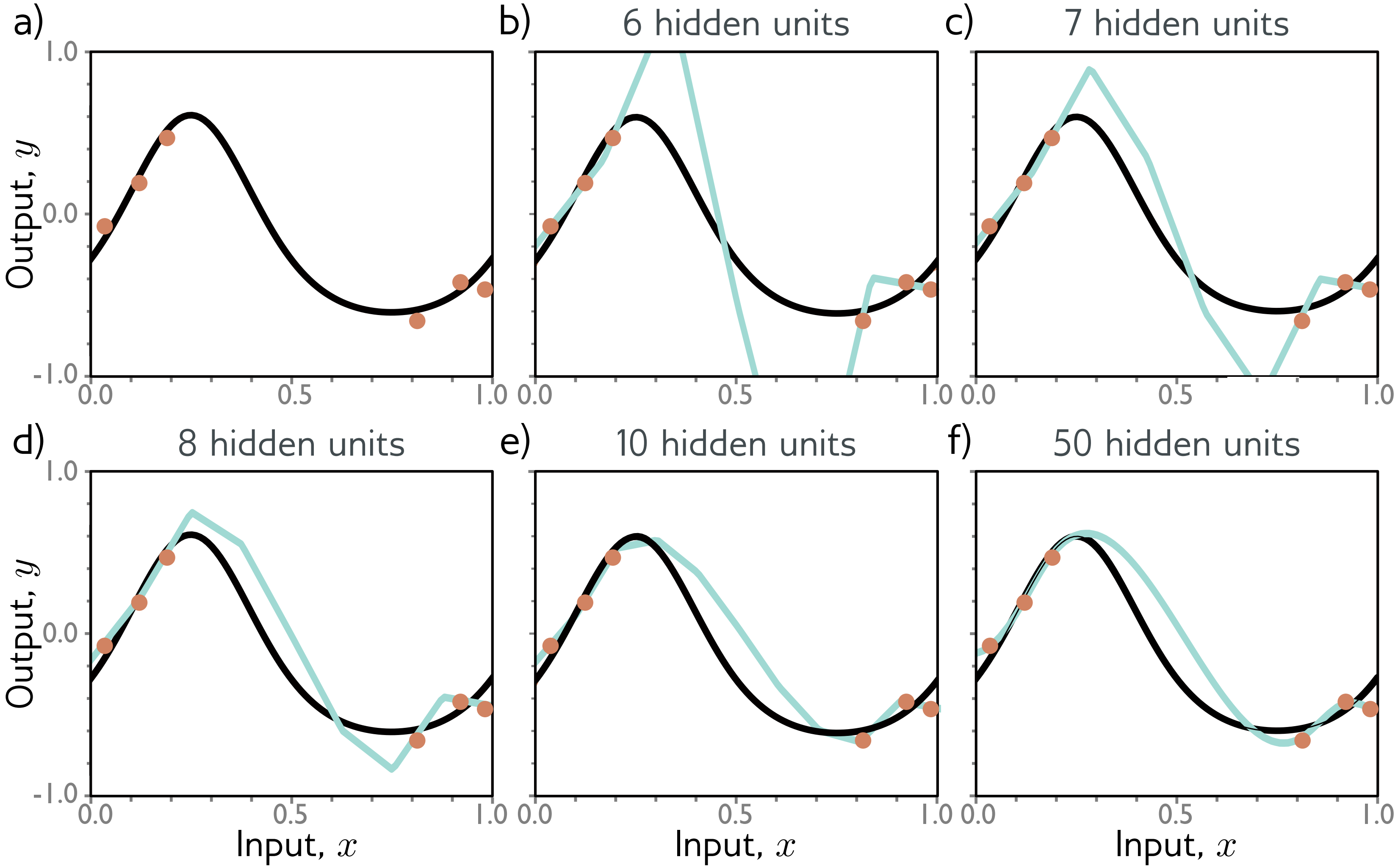
Unit diameter hypersphere in a unit hypercube.





"All the volume goes to the peel of the orange, not the pulp."

See also [“An Adventure in the Nth Dimension”, American Scientist]



Potential explanation:

- It seems that through implicit and explicit regularization (next lecture!) the (well trained) model tends to interpolate smoothly between training data points.

Measuring performance

- MNIST1D dataset model and performance
- Noise, bias, and variance
- Reducing variance
- Reducing bias & bias-variance trade-off
- Double descent
- Curse of dimensionality & weird properties of high dimensional space
- Choosing hyperparameters

Choosing hyperparameters

- Don't know bias or variance
- Don't know how much capacity to add
- How do we choose capacity in practice?
  - Or model structure
  - Or training algorithm
  - Or learning rate
- Third data set – validation set
  - Train models with different hyperparameters on training set
  - Choose best hyperparameters with validation set
  - Test once with test set

Feedback?