BOSTON UNIVERSITY

Deep Learning for Data Science DS 542

Lecture 08
Gradients, Initializations, and Measuring Performance

Slides originally by Thomas Gardos.

Images from Understanding Deep Learning unless otherwise cited.

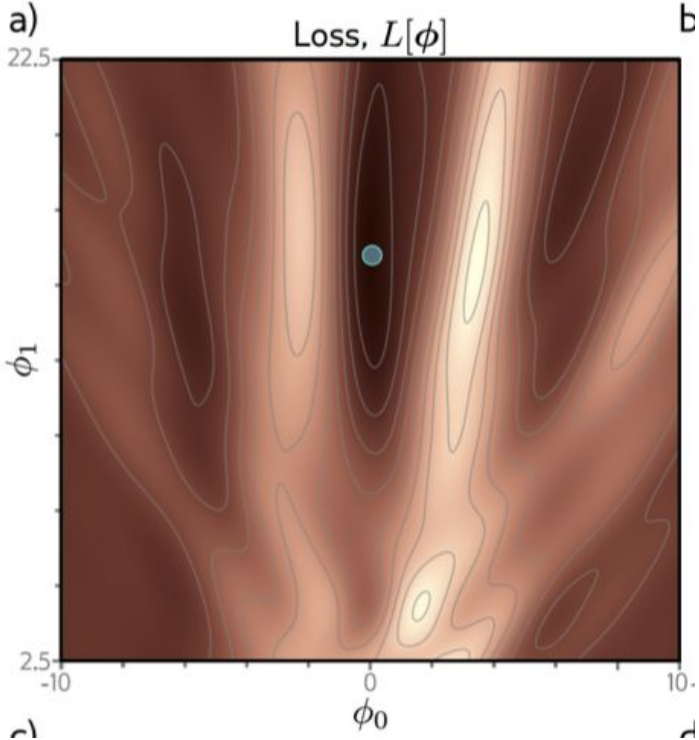
Today's Plan

- Gradients Recap
  - Visualizing 2 Parameter Models
  - Backpropagation again
- Initialization
  - Forward and backward behavior
  - Derivation of decent initializations
- Measuring Performance
  - Train/test splits
  - Bias/variance trade offs

Reminder: you should be reading the book chapters too. There are extra readings for Wednesday - see course web site or end of slides for links.

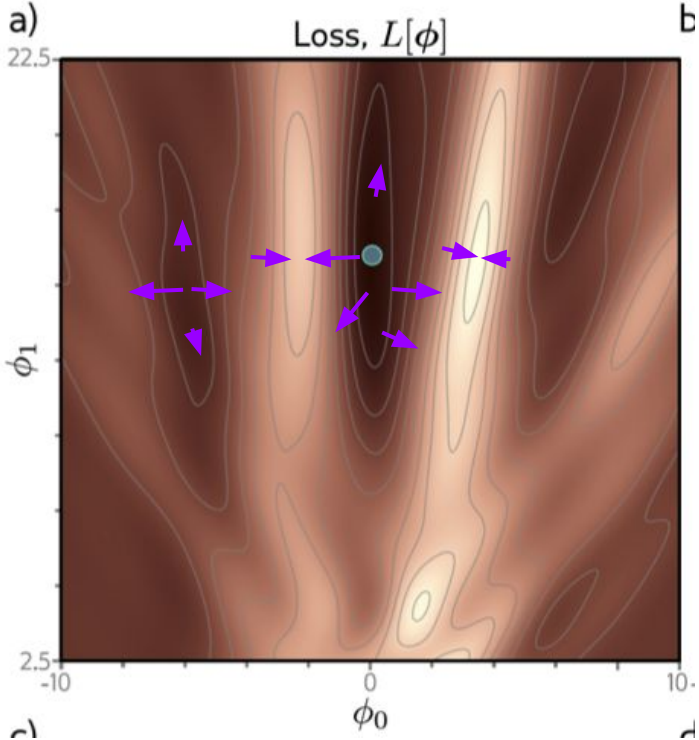
Visualizing Two Parameter Models

Loss function of the Gabor model from last time



Visualizing Two Parameter Models

Gradients of the loss function



Backpropagation again

Repeating a couple key points

- Basic mechanics of efficient gradient computation
- Critical to understand initialization

Backpropagation and the Chain Rule

If $v_j = f(v_i)$,

$$\frac{\partial L}{\partial v_i} = \frac{\partial L}{\partial v_j}\frac{\partial v_j}{\partial v_i} = \frac{\partial L}{\partial v_j}\frac{\partial f(v_i)}{\partial v_i}$$

Most common operation

And if $f(v_i) = \phi_i v_i$,

$$\frac{\partial L}{\partial v_i} = \frac{\partial L}{\partial v_j}\frac{\partial v_j}{\partial v_i} = \frac{\partial L}{\partial v_j}\frac{\partial f(v_i)}{\partial v_i} = \frac{\partial L}{\partial v_j}\frac{\partial \phi_i v_i}{\partial v_i} = \phi_i \frac{\partial L}{\partial v_j}$$

What if we have several layers?

Backpropagation with Matrix Operations

If

$$\mathbf{f}_0 = \beta_0 + \boldsymbol{\Omega}_0 \mathbf{x}_i$$

$$\mathbf{h}_k = \mathrm{a}[\mathbf{f}_{k-1}]$$

$$\mathbf{f}_k = \beta_k + \boldsymbol{\Omega}_k \mathbf{h}_k$$

Then,

$$\frac{\partial l_i}{\partial \mathbf{f}_{k-1}} = \mathbf{I}[\mathbf{f}_{k-1} > 0] \odot \left(\boldsymbol{\Omega}_k^T \frac{\partial l_i}{\partial \mathbf{f}_k} \right)$$

(k-1) more of these when fully unwound

Initialization

Perhaps an obvious point -

- Initializing all parameters to zero is degenerate.
  - All units within a layer will see the same gradients.
  - All units within a layer will get the same updates.
  - All units within a layer will represent the same function.
  - All layers effectively become one wide.
- Generally do not want to start with any symmetries within layers
  - Different initializations are opportunities to learn different useful things.
  - Motivates random initializations.

Initialize weights to different variances

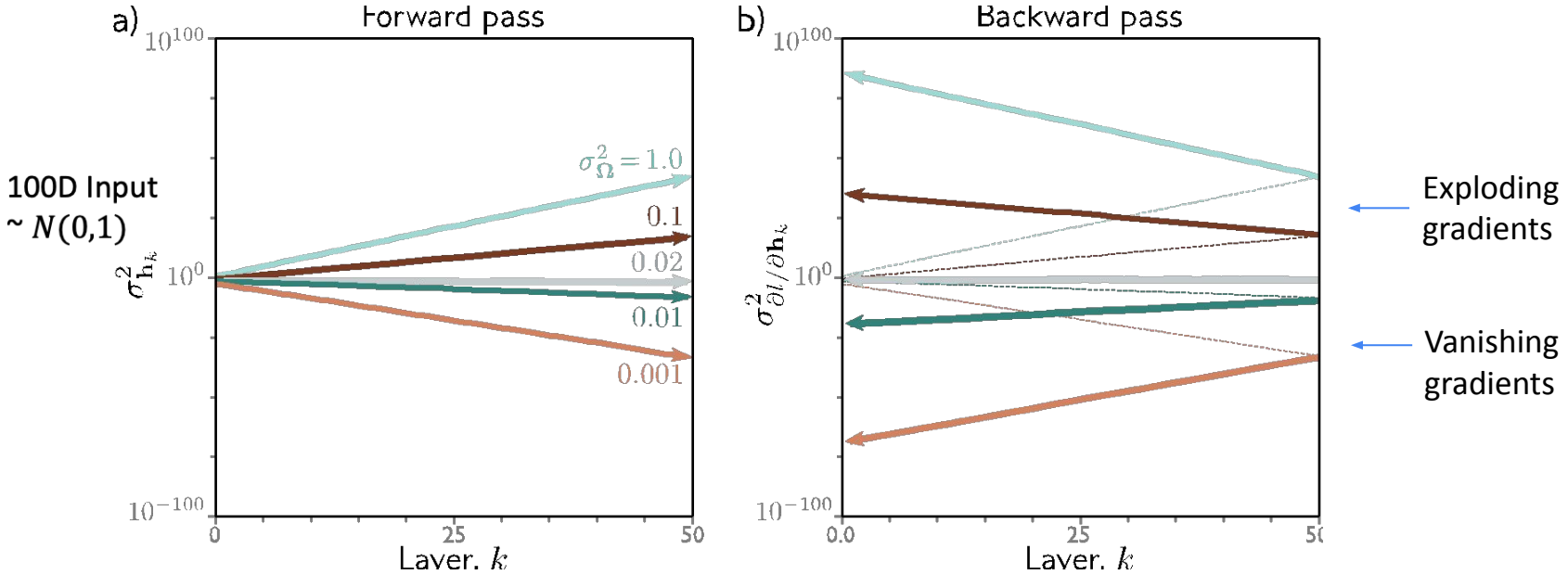


Figure 7.4 Weight initialization. Consider a deep network with 50 hidden layers and $D_h = 100$ hidden units per layer. The network has a 100 dimensional input $\mathbf{x}$ initialized with values from a standard normal distribution, a single output fixed at $y = 0$, and a least squares loss function. The bias vectors $\boldsymbol{\beta}_k$ are initialized to zero and the weight matrices $\boldsymbol{\Omega}_k$ are initialized with a normal distribution with mean zero and five different variances $\sigma^2_\Omega \in \{0.001, 0.01, 0.02, 0.1, 1.0\}$. a)

Initialization Theory

Any math to guide us to better initializations?

- Exponential growth and shrinkage are predictable from chain rule.
- Can we scale the weights to avoid exponential behavior?
  - Goldilocks zone - not too big or too small
  - We should be able to calculate a more useful range of initial weights
- Perfect value management not needed
  - Want a variety of values and gradients…
  - Some hidden units will be inactive sometimes…

Aim: keep variance same between two layers

$$\mathbf{f}' = \boldsymbol{\beta} + \boldsymbol{\Omega}\mathbf{h}$$
$$\mathbf{h} = \mathbf{a}[\mathbf{f}],$$

Definition of variance:

$$\sigma_{f'}^2 = \mathbb{E}[(f_i' - \mathbb{E}[f_i'])^2]$$

Agenda

- The need for weights initialization
- Expectations Refresher
- The He (Kaiming) Initialization

Expectations

$$\mathbb{E}\Big[\mathrm{g}[x]\Big] = \int \mathrm{g}[x] Pr(x) dx,$$

Interpretation: what is the average value of g[x] when taking into account the probability of x?

Consider discrete case and assume uniform probability so calculating g[x] reduces to taking average:

$$\mathbb{E}\Big[\mathrm{g}[x]\Big] \approx \frac{1}{N}\sum_{n=1}^{N} \mathrm{g}[x_n^*] \qquad \text{where} \qquad x_n^* \sim Pr(x)$$

Common Expectation Functions

Function g[•]	Expectation
x	mean, μ
x^k	kth moment about zero
$(x-\mu)^k$	kth moment about the mean
$(x-\mu)^2$	variance
$(x-\mu)^3$	skew
$(x-\mu)^4$	kurtosis

Table B.1 Special cases of expectation. For some functions $g[x]$, the expectation $\mathbb{E}[g[x]$ is given a special name. Here we use the notation μ_x to represent the mean with respect to random variable x.

Rules for manipulating expectation

$$\mathbb{E}\Big[k\Big] = k$$

$$\mathbb{E}\Big[k \cdot \mathrm{g}[x]\Big] = k \cdot \mathbb{E}\Big[\mathrm{g}[x]\Big]$$

$$\mathbb{E}\Big[\mathrm{f}[x] + \mathrm{g}[x]\Big] = \mathbb{E}\Big[\mathrm{f}[x]\Big] + \mathbb{E}\Big[\mathrm{g}[x]\Big]$$

$$\mathbb{E}\Big[\mathrm{f}[x]g[y]\Big] = \mathbb{E}\Big[\mathrm{f}[x]\Big]\mathbb{E}\Big[\mathrm{g}[y]\Big] \qquad \text{if} \quad x, y \quad \text{independent}$$

Agenda

- The need for weights initialization
- Expectations Refresher
- The He (Kaiming) Initialization

Aim: keep variance same between two layers

$$\mathbf{h} = \mathbf{a}[\mathbf{f}],$$
$$\mathbf{f}' = \boldsymbol{\beta} + \boldsymbol{\Omega}\mathbf{h}$$

Definition of variance:

$$\sigma_{f_i'}^2 = \mathbb{E}[(f_i' - \mathbb{E}[f_i'])^2]$$

Aim: keep variance same between two layers

$$\mathbf{f}' = \boldsymbol{\beta} + \boldsymbol{\Omega}\mathbf{h}$$
$$\mathbf{h} = \mathbf{a}[\mathbf{f}],$$

$$\sigma_{f'}^2 = \mathbb{E}[(f_i' - \mathbb{E}[f_i'])^2]$$

$$\sigma_{f'}^2 = \mathbb{E}[f_i'^2] - \mathbb{E}[f_i']^2$$

Aim: keep variance same between two layers

$$\mathbf{f}' = \boldsymbol{\beta} + \boldsymbol{\Omega}\mathbf{h}$$
$$\mathbf{h} = \mathbf{a}[\mathbf{f}],$$

$$\sigma_{f'}^2 = \mathbb{E}[(f_i' - \mathbb{E}[f_i'])^2]$$

$$\sigma_{f'}^2 = \mathbb{E}[f_i'^2] - \mathbb{E}[f_i']^2$$

Aim: keep variance same between two layers

$$\mathbf{f}' = \boldsymbol{\beta} + \boldsymbol{\Omega}\mathbf{h}$$

Consider the mean of the pre-activations:

$$\mathbb{E}[f_i'] = \mathbb{E}\left[\beta_i + \sum_{j=1}^{D_h} \Omega_{ij} h_j\right]$$

Rule 1: $$\mathbb{E}\Big[k\Big] = k$$

Rule 2: $$\mathbb{E}\Big[k \cdot \mathrm{g}[x]\Big] = k \cdot \mathbb{E}\Big[\mathrm{g}[x]\Big]$$

Rule 3: $$\mathbb{E}\Big[\mathrm{f}[x] + \mathrm{g}[x]\Big] = \mathbb{E}\Big[\mathrm{f}[x]\Big] + \mathbb{E}\Big[\mathrm{g}[x]\Big]$$

Rule 4: $$\mathbb{E}\Big[\mathrm{f}[x]g[y]\Big] = \mathbb{E}\Big[\mathrm{f}[x]\Big]\mathbb{E}\Big[\mathrm{g}[y]\Big] \quad \text{if} \quad x, y \quad \text{independent}$$

$$\mathbb{E}[f_i'] = \mathbb{E}\left[\beta_i + \sum_{j=1}^{D_h} \Omega_{ij} h_j\right]$$

$$= \mathbb{E}\left[\beta_i\right] + \sum_{j=1}^{D_h} \mathbb{E}\left[\Omega_{ij} h_j\right]$$

Rule 1: $\mathbb{E}\Big[k\Big] = k$

Rule 2: $\mathbb{E}\Big[k \cdot \mathrm{g}[x]\Big] = k \cdot \mathbb{E}\Big[\mathrm{g}[x]\Big]$

Rule 3: $\mathbb{E}\Big[\mathrm{f}[x] + \mathrm{g}[x]\Big] = \mathbb{E}\Big[\mathrm{f}[x]\Big] + \mathbb{E}\Big[\mathrm{g}[x]\Big]$

Rule 4: $\mathbb{E}\Big[\mathrm{f}[x]g[y]\Big] = \mathbb{E}\Big[\mathrm{f}[x]\Big]\mathbb{E}\Big[\mathrm{g}[y]\Big] \quad \text{if} \quad x, y \quad \text{independent}$

$$\mathbb{E}[f_i'] = \mathbb{E}\left[\beta_i + \sum_{j=1}^{D_h} \Omega_{ij} h_j\right]$$

$$= \mathbb{E}\left[\beta_i\right] + \sum_{j=1}^{D_h} \mathbb{E}\left[\Omega_{ij} h_j\right]$$

$$= \mathbb{E}\left[\beta_i\right] + \sum_{j=1}^{D_h} \mathbb{E}\left[\Omega_{ij}\right] \mathbb{E}\left[h_j\right]$$

Rule 1: $\mathbb{E}\Big[k\Big] = k$

Rule 2: $\mathbb{E}\Big[k \cdot \mathrm{g}[x]\Big] = k \cdot \mathbb{E}\Big[\mathrm{g}[x]\Big]$

Rule 3: $\mathbb{E}\Big[\mathrm{f}[x] + \mathrm{g}[x]\Big] = \mathbb{E}\Big[\mathrm{f}[x]\Big] + \mathbb{E}\Big[\mathrm{g}[x]\Big]$

Rule 4: $\mathbb{E}\Big[\mathrm{f}[x]g[y]\Big] = \mathbb{E}\Big[\mathrm{f}[x]\Big]\mathbb{E}\Big[\mathrm{g}[y]\Big]$ if x, y independent

$$\mathbb{E}[f_i'] = \mathbb{E}\left[\beta_i + \sum_{j=1}^{D_h} \Omega_{ij} h_j\right]$$

$$= \mathbb{E}\left[\beta_i\right] + \sum_{j=1}^{D_h} \mathbb{E}\left[\Omega_{ij} h_j\right]$$

Set all the biases to 0

$$= \mathbb{E}\left[\beta_i\right] + \sum_{j=1}^{D_h} \mathbb{E}\left[\Omega_{ij}\right]\mathbb{E}\left[h_j\right]$$

Weights normally distributed
- mean 0
- variance σ_Ω^2

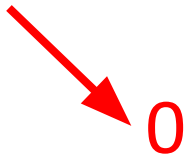
$$= 0 + \sum_{j=1}^{D_h} 0 \cdot \mathbb{E}\left[h_j\right] = 0$$

Aim: keep variance same between two layers

$$\mathbf{f}' = \boldsymbol{\beta} + \boldsymbol{\Omega}\mathbf{h}$$
$$\mathbf{h} = \mathbf{a}[\mathbf{f}],$$

$$\sigma_{f'}^2 = \mathbb{E}[(f_i' - \mathbb{E}[f_i'])^2]$$

$$\sigma_{f'}^2 = \mathbb{E}[f_i'^2] - \mathbb{E}[f_i']^2 = \mathbb{E}[f_i'^2]$$



Rule 1: $\mathbb{E}\Big[k\Big] = k$

Rule 2: $\mathbb{E}\Big[k \cdot \mathrm{g}[x]\Big] = k \cdot \mathbb{E}\Big[\mathrm{g}[x]\Big]$

Rule 3: $\mathbb{E}\Big[\mathrm{f}[x] + \mathrm{g}[x]\Big] = \mathbb{E}\Big[\mathrm{f}[x]\Big] + \mathbb{E}\Big[\mathrm{g}[x]\Big]$

Rule 4: $\mathbb{E}\Big[\mathrm{f}[x]g[y]\Big] = \mathbb{E}\Big[\mathrm{f}[x]\Big]\mathbb{E}\Big[\mathrm{g}[y]\Big] \quad \text{if} \quad x, y \quad \text{independent}$

$$\sigma_{f'}^2 = \mathbb{E}[f_i'^2] - \mathbb{E}[f_i']^2$$

$$= \mathbb{E}\left[\left(\beta_i + \sum_{j=1}^{D_h} \Omega_{ij} h_j\right)^2\right] - 0$$

Set all the biases to 0

Weights normally distributed

- mean 0
- variance σ_Ω^2

Rule 1: $\mathbb{E}\left[k\right] = k$

Rule 2: $\mathbb{E}\left[k \cdot \mathrm{g}[x]\right] = k \cdot \mathbb{E}\left[\mathrm{g}[x]\right]$

Rule 3: $\mathbb{E}\left[\mathrm{f}[x] + \mathrm{g}[x]\right] = \mathbb{E}\left[\mathrm{f}[x]\right] + \mathbb{E}\left[\mathrm{g}[x]\right]$

Rule 4: $\mathbb{E}\left[\mathrm{f}[x]g[y]\right] = \mathbb{E}\left[\mathrm{f}[x]\right]\mathbb{E}\left[\mathrm{g}[y]\right]$ if x, y independent

$$\begin{aligned}\sigma_{f'}^2 &= \mathbb{E}[f_i'^2] - \mathbb{E}[f_i']^2 \\ &= \mathbb{E}\left[\left(\beta_i + \sum_{j=1}^{D_h} \Omega_{ij} h_j\right)^2\right] - 0 \\ &= \mathbb{E}\left[\left(\sum_{j=1}^{D_h} \Omega_{ij} h_j\right)^2\right]\end{aligned}$$

Set all the biases to 0

Weights normally distributed

- mean 0
- variance σ_Ω^2

Rule 1: $\mathbb{E}\left[k\right] = k$

Rule 2: $\mathbb{E}\left[k \cdot \mathrm{g}[x]\right] = k \cdot \mathbb{E}\left[\mathrm{g}[x]\right]$

Rule 3: $\mathbb{E}\left[\mathrm{f}[x] + \mathrm{g}[x]\right] = \mathbb{E}\left[\mathrm{f}[x]\right] + \mathbb{E}\left[\mathrm{g}[x]\right]$

Rule 4: $\mathbb{E}\left[\mathrm{f}[x]g[y]\right] = \mathbb{E}\left[\mathrm{f}[x]\right]\mathbb{E}\left[\mathrm{g}[y]\right] \quad \text{if} \quad x, y \quad \text{independent}$

$$\sigma_{f'}^2 = \mathbb{E}[f_i'^2] - \mathbb{E}[f_i']^2$$

$$= \mathbb{E}\left[\left(\beta_i + \sum_{j=1}^{D_h} \Omega_{ij} h_j\right)^2\right] - 0$$

$$= \mathbb{E}\left[\left(\sum_{j=1}^{D_h} \Omega_{ij} h_j\right)^2\right]$$

$$= \sum_{j=1}^{D_h} \mathbb{E}\left[\Omega_{ij}^2\right] \mathbb{E}\left[h_j^2\right]$$

For all the cross terms, $E[\Omega_{ij}] = 0$ so only the squared terms are left, then use independence.

Set all the biases to 0

Weights normally distributed
- mean 0
- variance σ_Ω^2

Rule 1: $\mathbb{E}\Big[k\Big] = k$

Rule 2: $\mathbb{E}\Big[k \cdot \mathrm{g}[x]\Big] = k \cdot \mathbb{E}\Big[\mathrm{g}[x]\Big]$

Rule 3: $\mathbb{E}\Big[\mathrm{f}[x] + \mathrm{g}[x]\Big] = \mathbb{E}\Big[\mathrm{f}[x]\Big] + \mathbb{E}\Big[\mathrm{g}[x]\Big]$

Rule 4: $\mathbb{E}\Big[\mathrm{f}[x]g[y]\Big] = \mathbb{E}\Big[\mathrm{f}[x]\Big]\mathbb{E}\Big[\mathrm{g}[y]\Big]$ if x, y independent

$$\begin{aligned}
\sigma_{f'}^2 &= \mathbb{E}[f_i'^2] - \mathbb{E}[f_i']^2 \\
&= \mathbb{E}\left[\left(\beta_i + \sum_{j=1}^{D_h} \Omega_{ij} h_j\right)^2\right] - 0 \\
&= \mathbb{E}\left[\left(\sum_{j=1}^{D_h} \Omega_{ij} h_j\right)^2\right] \\
&= \sum_{j=1}^{D_h} \mathbb{E}\left[\Omega_{ij}^2\right] \mathbb{E}\left[h_j^2\right] \\
&= \sum_{j=1}^{D_h} \sigma_\Omega^2 \mathbb{E}\left[h_j^2\right] = \sigma_\Omega^2 \sum_{j=1}^{D_h} \mathbb{E}\left[h_j^2\right]
\end{aligned}$$

Set all the biases to 0

Weights normally distributed
- mean 0
- variance σ_Ω^2

Because the Ω's are zero mean, this is the variance.

$$\sigma_{f'}^2 = \sigma_\Omega^2 \sum_{j=1}^{D_h} \mathbb{E}\left[h_j^2\right]$$

$$= \sigma_\Omega^2 \sum_{j=1}^{D_h} \mathbb{E}\left[\mathrm{ReLU}[f_j]^2\right]$$

$$= \sigma_\Omega^2 \sum_{j=1}^{D_h} \int_{-\infty}^{\infty} \mathrm{ReLU}[f_j]^2 Pr(f_j) df_j$$

From the definition of expectation.

$$= \sigma_\Omega^2 \sum_{j=1}^{D_h} \int_{-\infty}^{\infty} (\mathbb{I}[f_j > 0] f_j)^2 Pr(f_j) df_j$$

$$= \sigma_\Omega^2 \sum_{j=1}^{D_h} \int_{0}^{\infty} f_j^2 Pr(f_j) df_j$$

Only positive integral limits because of ReLU

$$= \sigma_\Omega^2 \sum_{j=1}^{D_h} \frac{\sigma_f^2}{2} = \frac{D_h \sigma_\Omega^2 \sigma_f^2}{2}$$

½ of the variance for zero mean distribution

Aim: keep variance same between two layers

Since:

$$\sigma_{f'}^2 = \frac{D_h \sigma_\Omega^2 \sigma_f^2}{2}$$

Should choose:

$$\sigma_\Omega^2 = \frac{2}{D_h}$$

To get:

$$\sigma_{f'}^2 = \sigma_f^2$$

This is called He initialization or Kaiming initialization after Kaiming He (何恺明).

K. He, X. Zhang, S. Ren, and J. Sun, “Delving Deep into Rectifiers: Surpassing Human-Level Performance on ImageNet Classification,” Proc. IEEE International Conference on Computer Vision, 2015, pp. 1026–1034. Accessed: Feb. 11, 2024.

He initialization (assumes ReLU)

- Forward pass: want the variance of hidden unit activations in layer k+1 to be the same as variance of activations in layer k:

$$\sigma_{\Omega}^{2} = \frac{2}{D_h}$$

Number of units at layer k

- Backward pass: want the variance of gradients at layer k to be the same as variance of gradient in layer k+1:

$$\sigma_{\Omega}^{2} = \frac{2}{D_{h'}}$$

Number of units at layer k+1

Initialization with Non-Square Matrices

$\frac{2}{D_h}$ or $\frac{2}{D_{h'}}$?

Compromise:

$$\frac{4}{D_h + D_{h'}}$$

Not quite the average. Maybe an argument to keep hidden layer sizes uniform?

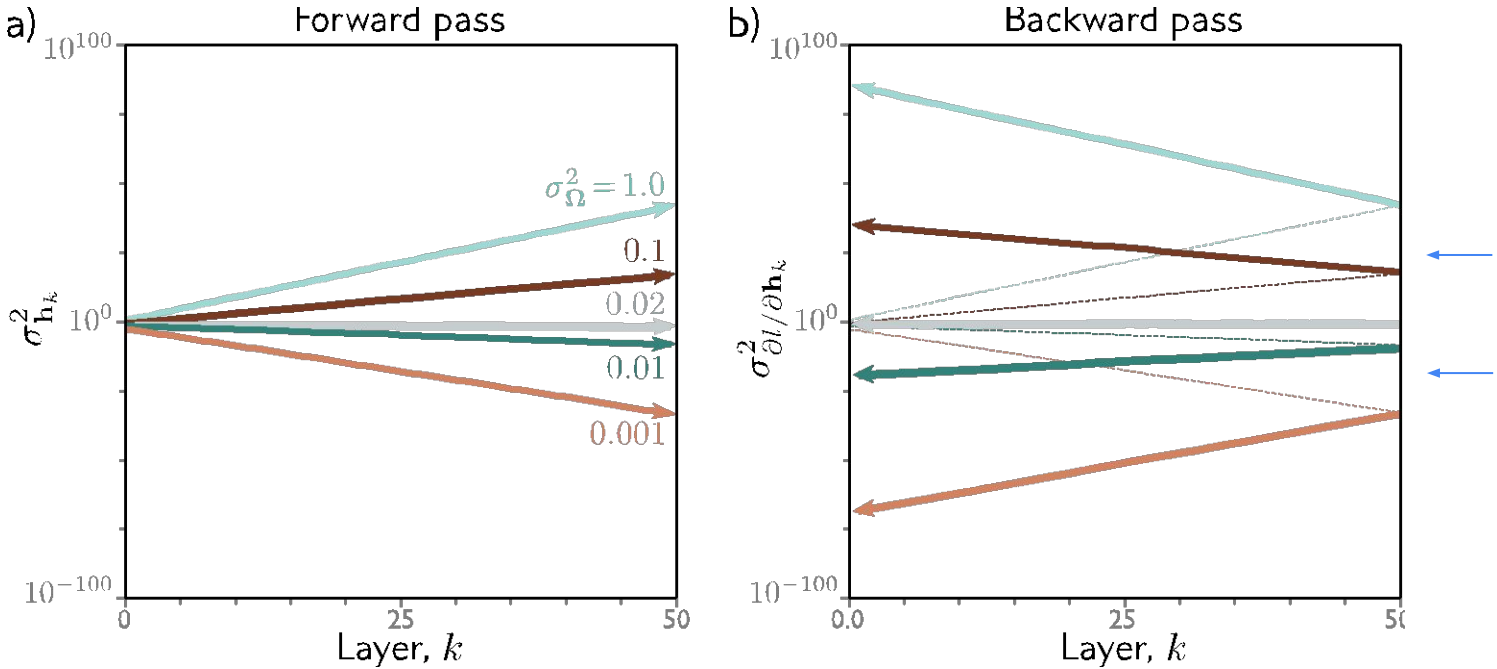


Figure 7.4 Weight initialization. Consider a deep network with 50 hidden layers and $D_h = 100$ hidden units per layer. The network has a 100 dimensional input $\mathbf{x}$ initialized with values from a standard normal distribution, a single output fixed at $y = 0$, and a least squares loss function. The bias vectors $\boldsymbol{\beta}_k$ are initialized to zero and the weight matrices $\boldsymbol{\Omega}_k$ are initialized with a normal distribution with mean zero and five different variances $\sigma^2_\Omega \in \{0.001, 0.01, 0.02, 0.1, 1.0\}$. a)

$$\sigma^2_\Omega = \frac{2}{D_h} = \frac{2}{100} = 0.02$$

Default Initialization in PyTorch

https://pytorch.org/docs/stable/nn.init.html#torch.nn.init.kaiming_uniform_

torch.nn.init.kaiming_uniform_(*tensor*, *a=0*, *mode='fan_in'*, *nonlinearity='leaky_relu'*, *generator=None*) [SOURCE]

Fill the input *Tensor* with values using a Kaiming uniform distribution.

The method is described in *Delving deep into rectifiers: Surpassing human-level performance on ImageNet classification* - He, K. et al. (2015). The resulting tensor will have values sampled from $\mathcal{U}(-\text{bound}, \text{bound})$ where

$$\text{bound} = \text{gain} \times \sqrt{\frac{3}{\text{fan\_mode}}}$$

Also known as He initialization.

https://arxiv.org/abs/1502.01852

Initialization Note

A good initialization does not prevent gradient descent from changing the weights a lot.

- A good initialization keeps the initial gradients modestly sized,
- And modest gradients reduce wild swings in parameters with gradient descent
- Smaller learning rates also help with this.
- Next week’s topic, regularization, will directly address this.

Limitations of Initialization

- No guarantees that the model will train to low losses
- No guarantees that training process won’t lead to large values or gradients
- No guarantees that the model won’t have lots of inactive units
  - In fact, the estimates adjusted for half being inactive!

- In fact, much of the network is often useless, and could be pruned away!

The Lottery Ticket Hypothesis: Finding Sparse, Trainable Neural Networks

Neural network pruning techniques can reduce the parameter counts of trained networks by over 90%, decreasing storage requirements and improving computational performance of inference without compromising accuracy. However, contemporary experience is that the sparse architectures produced by pruning are difficult to train from the start, which would similarly improve training performance.

We find that a standard pruning technique naturally uncovers subnetworks whose initializations made them capable of training effectively. Based on these results, we articulate the "lottery ticket hypothesis:" dense, randomly-initialized, feed-forward networks contain subnetworks ("winning tickets") that - when trained in isolation - reach test accuracy comparable to the original network in a similar number of iterations. The winning tickets we find have won the initialization lottery: their connections have initial weights that make training particularly effective.

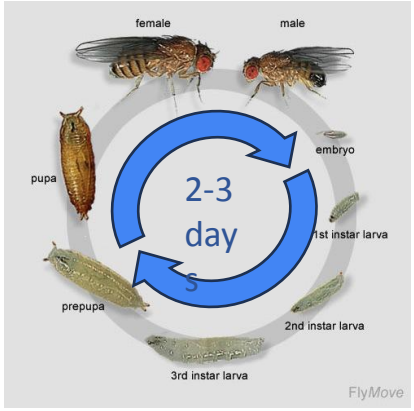
We present an algorithm to identify winning tickets and a series of experiments that support the lottery ticket hypothesis and the importance of these fortuitous initializations. We consistently find winning tickets that are less than 10-20% of the size of several fully-connected and convolutional feed-forward architectures for MNIST and CIFAR10. Above this size, the winning tickets that we find learn faster than the original network and reach higher test accuracy.

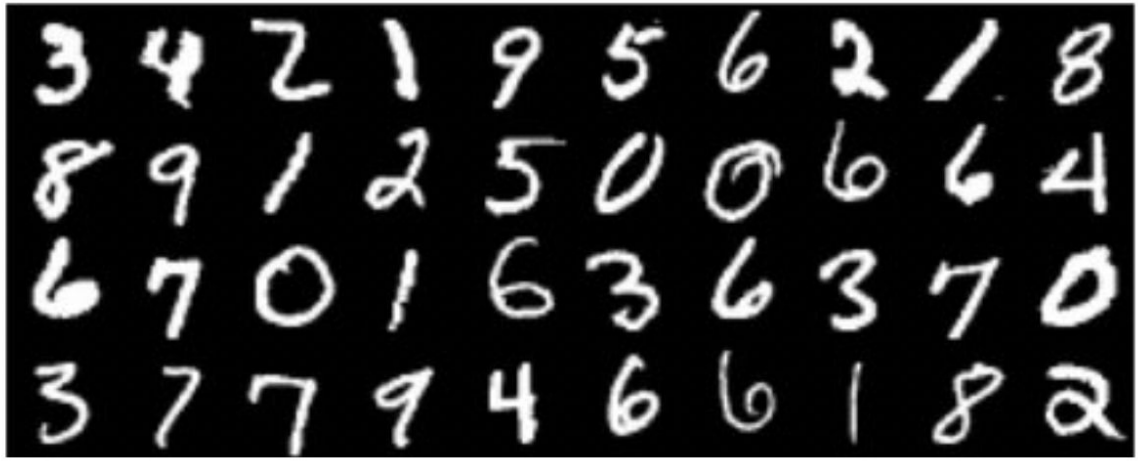
Measuring Performance

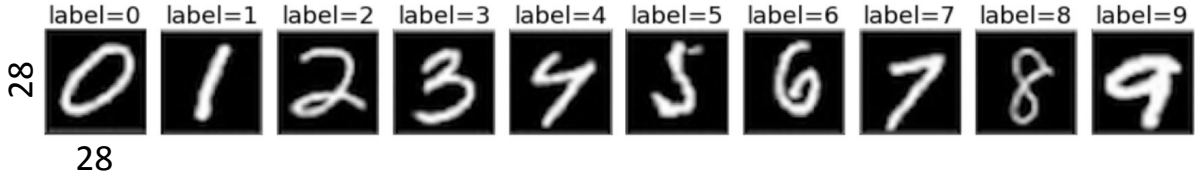
How do we do it?

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+%

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."

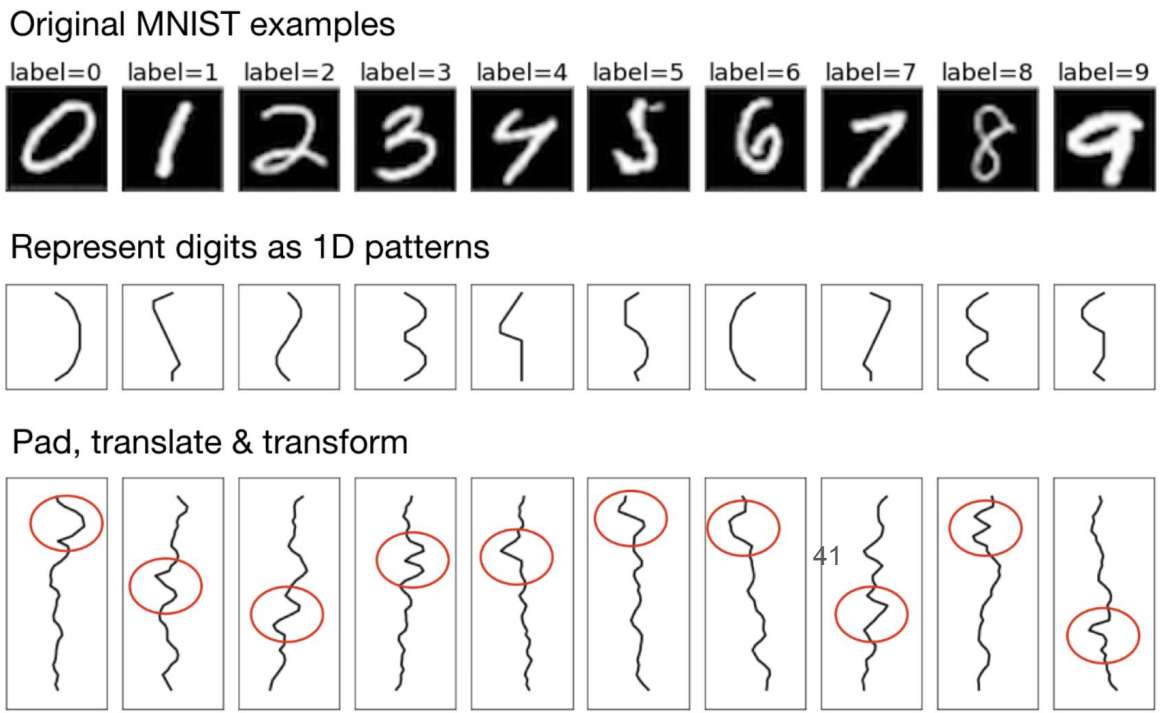
S. Greydanus, "Scaling down Deep Learning." arXiv, Dec. 04, 2020. doi: 10.48550/arXiv.2011.14439.

https://github.com/greydanus/mnist1d

MNIST 1D Dataset

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

Generate dataset by programmatically applying 6 parametric transformations.

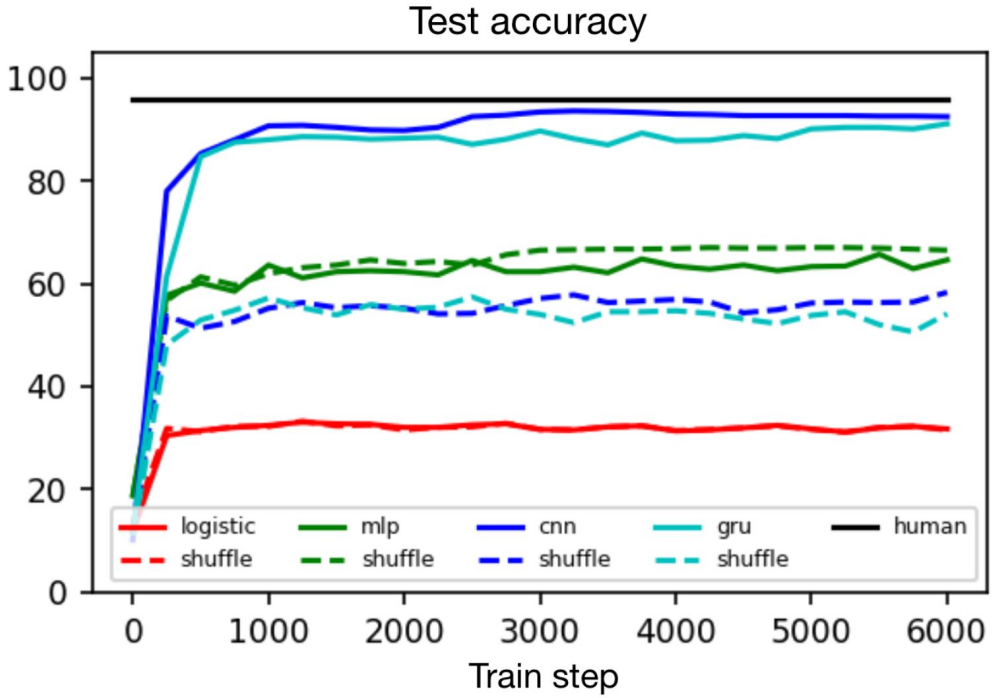


E.g. pad, shear, translate, correlated noise, i.i.d. noise, interpolation.

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

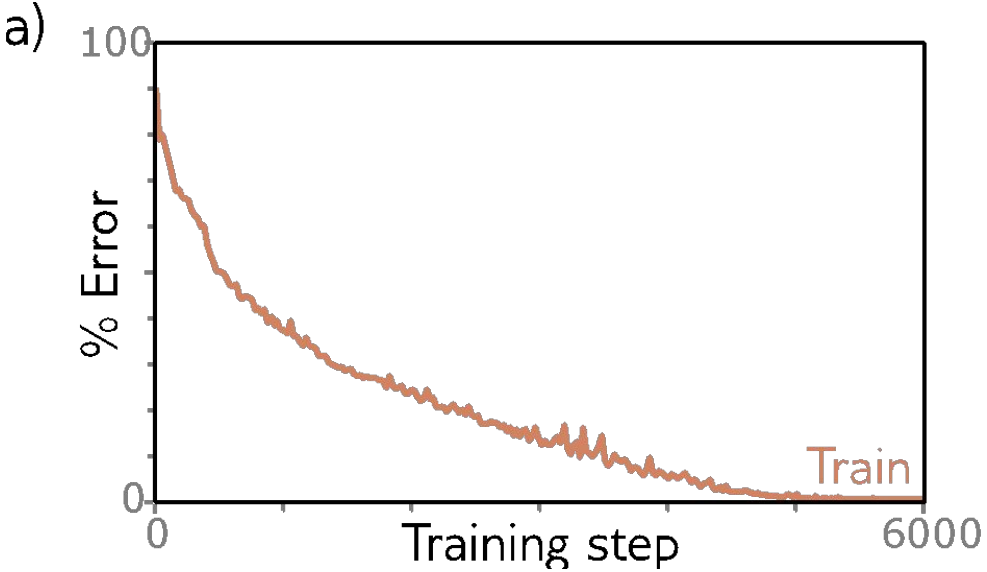
MNIST 1D

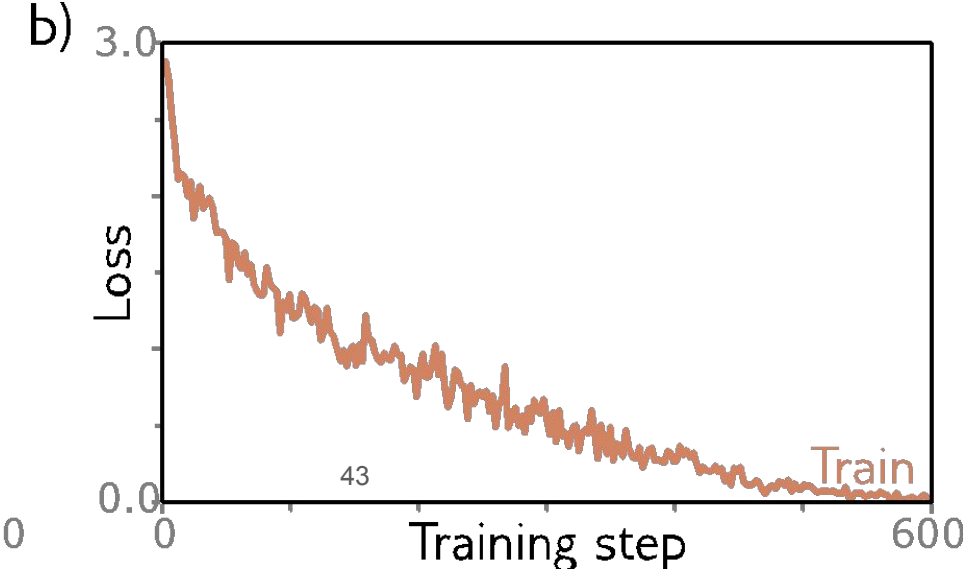
Differentiates performance of different model types much more than MNIST



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	$\approx 30 \pm 10$

Results


a)
100
% Error
0
Train
0
Training step
6000


b)
3.0
Loss
0.0
Train
0
Training step
600

Train / Test Splits

When training a model, reserve a sample to test the performance when you are done training.

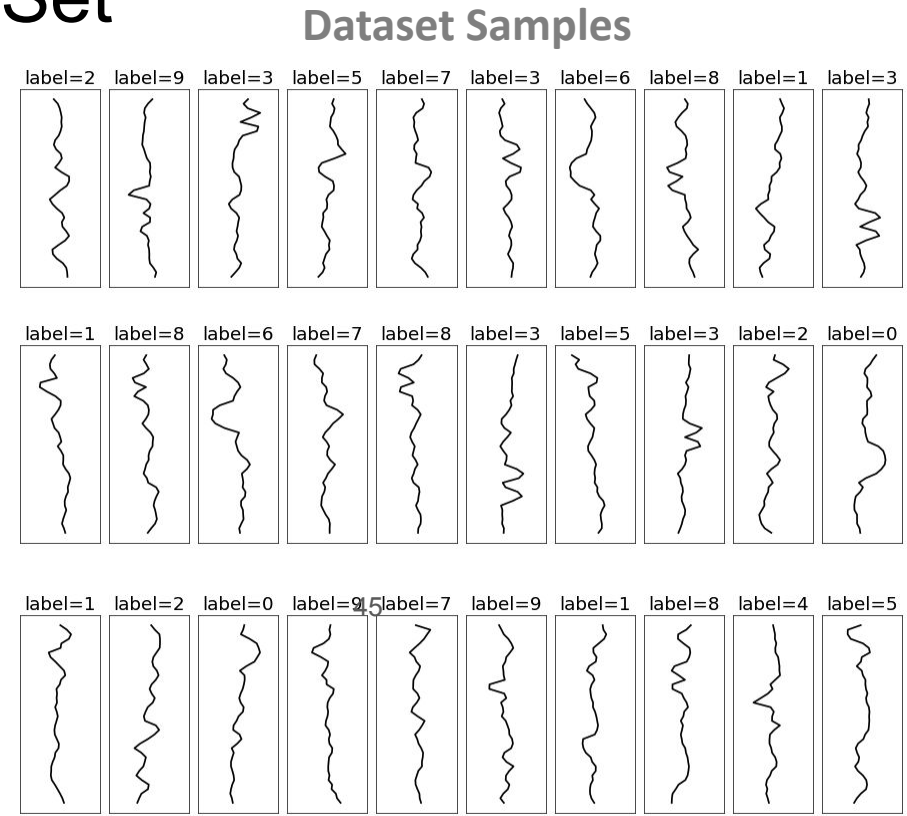
- Original training data → train + test sets.

- Use train set to fit the model.
- Use test set to check the final result.

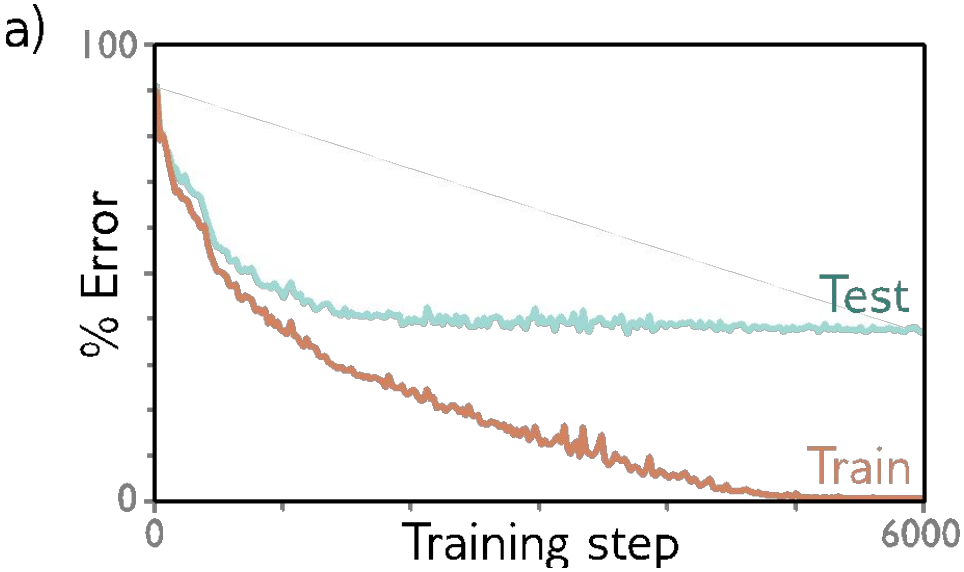
- Adapting to errors in the test is a good way to overfit your model.

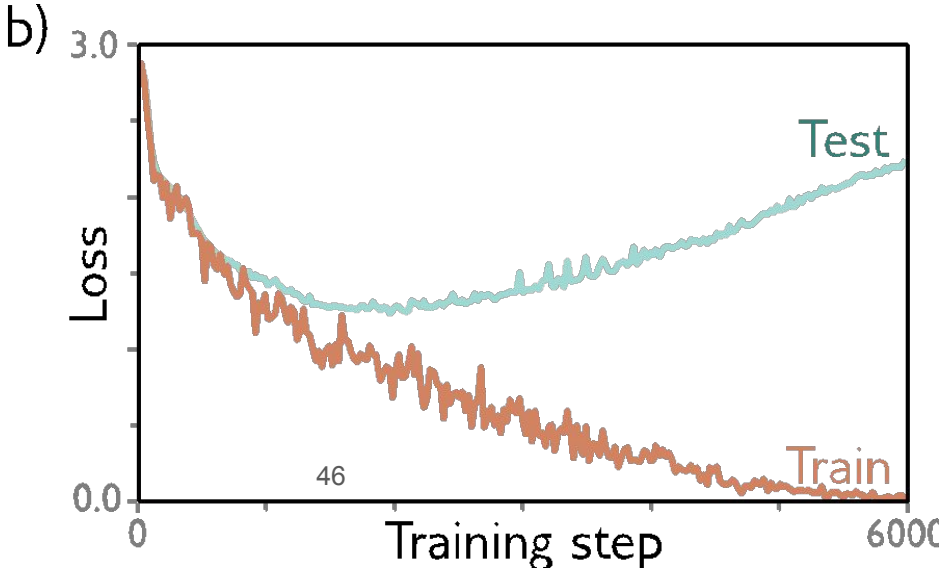
MNIST1D Train and Test Set

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



Need to use separate test data





The model has not generalized well to the new data

Train / Validation / Test Splits

More advanced methodology

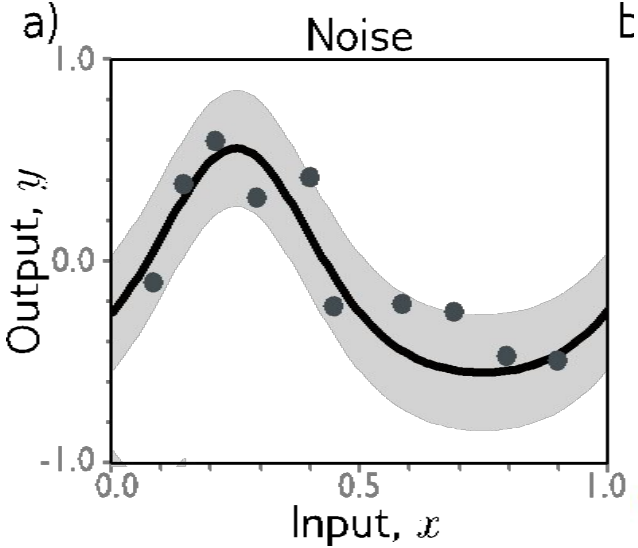
- Original training data → train + validation + test sets.

- Use train set to fit the model.
- Use validation set to make decisions about fitting the model.
- Use test set to check the final result.

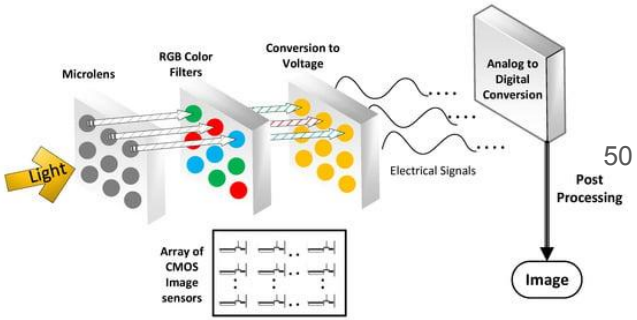
Sources of Error

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

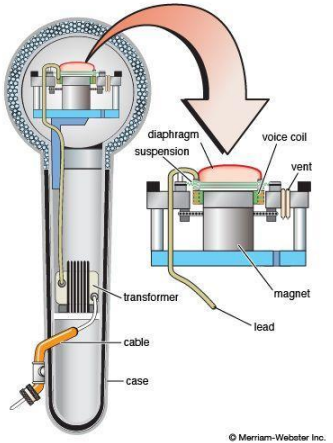
Noise, bias, and variance



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

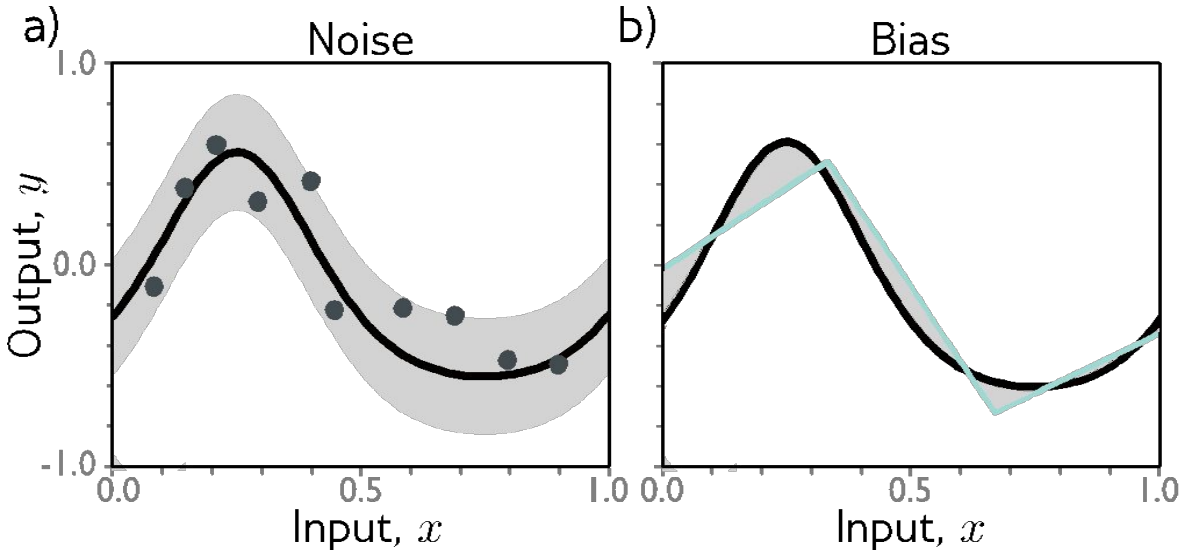


https://images.app.goo.gl/2PuBhaFpfdL9Pyjb8



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

Noise, bias, and variance



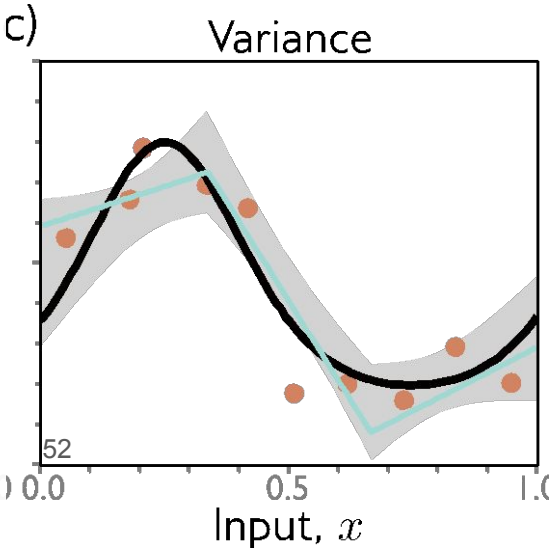
Bias occurs because the model lacks precision or capacity to accurately match the underlying function.

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

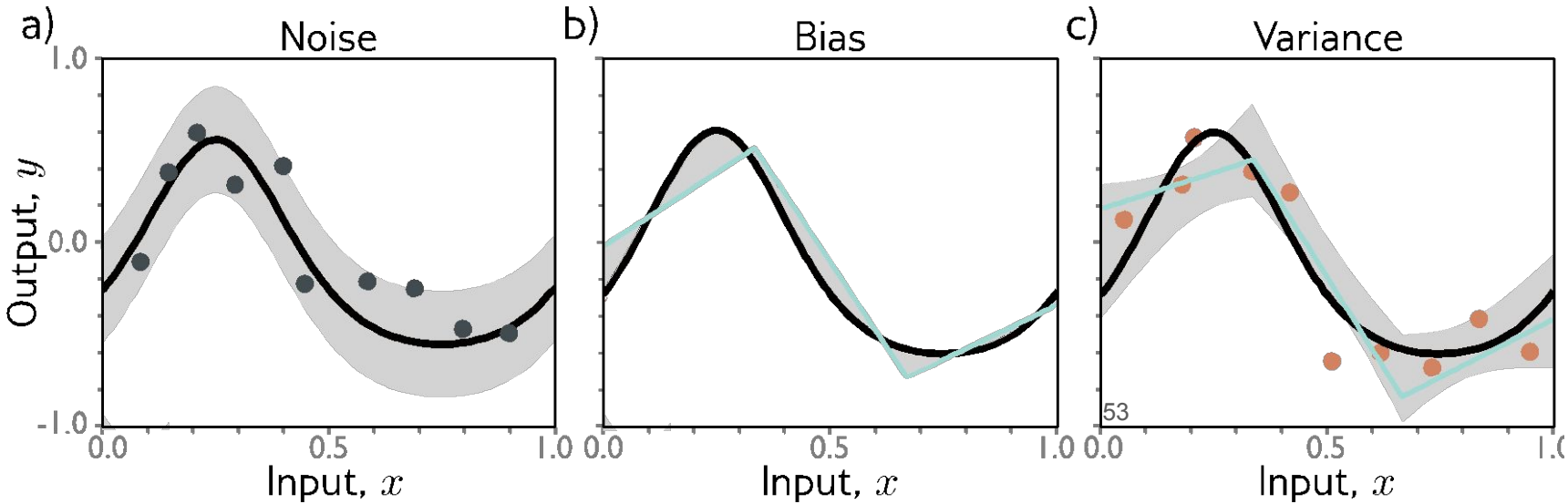
Noise, bias, and variance

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



Least squares regression only

For derivation see Section 8.2.2 in UDL.

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

- We can show that:

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

- And then:

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

Expectation over noise in training data

Expectation over noise in test data

Actual model

Best possible model if we had infinite data

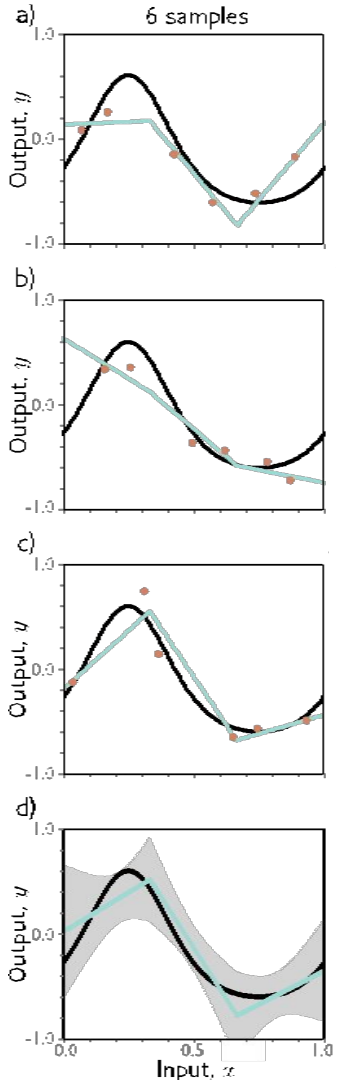
True function

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

Measuring performance

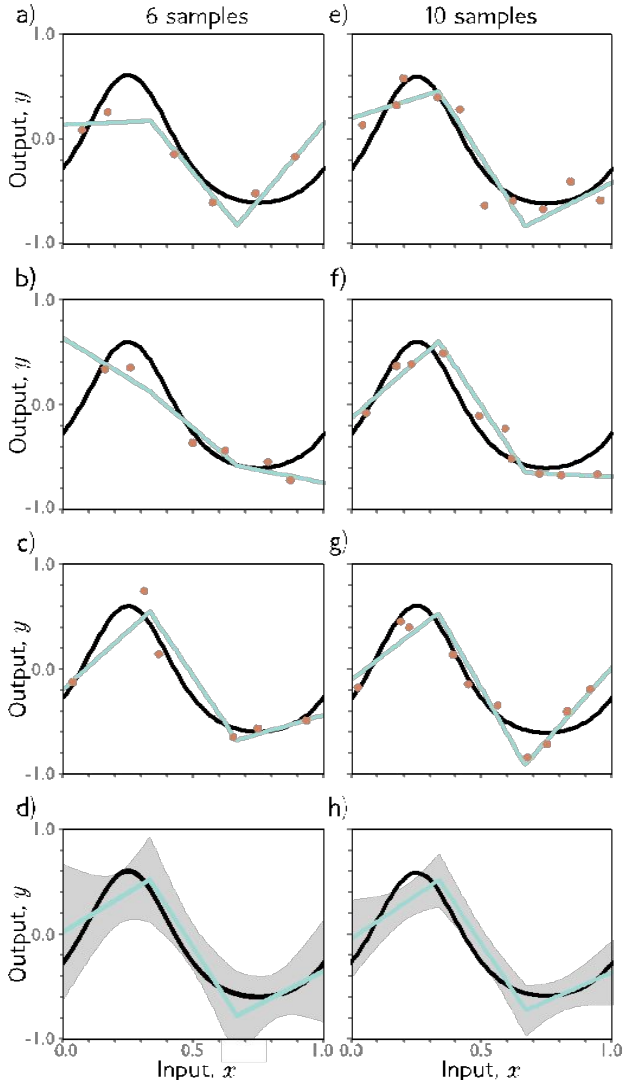
- MNIST1D dataset model and performance
- Noise, bias, and variance
- Reducing variance
- Reducing bias & bias-variance trade-off
- Double descent
- Curse of dimensionality & weird properties of high dimensional space
- Choosing hyperparameters

Variance



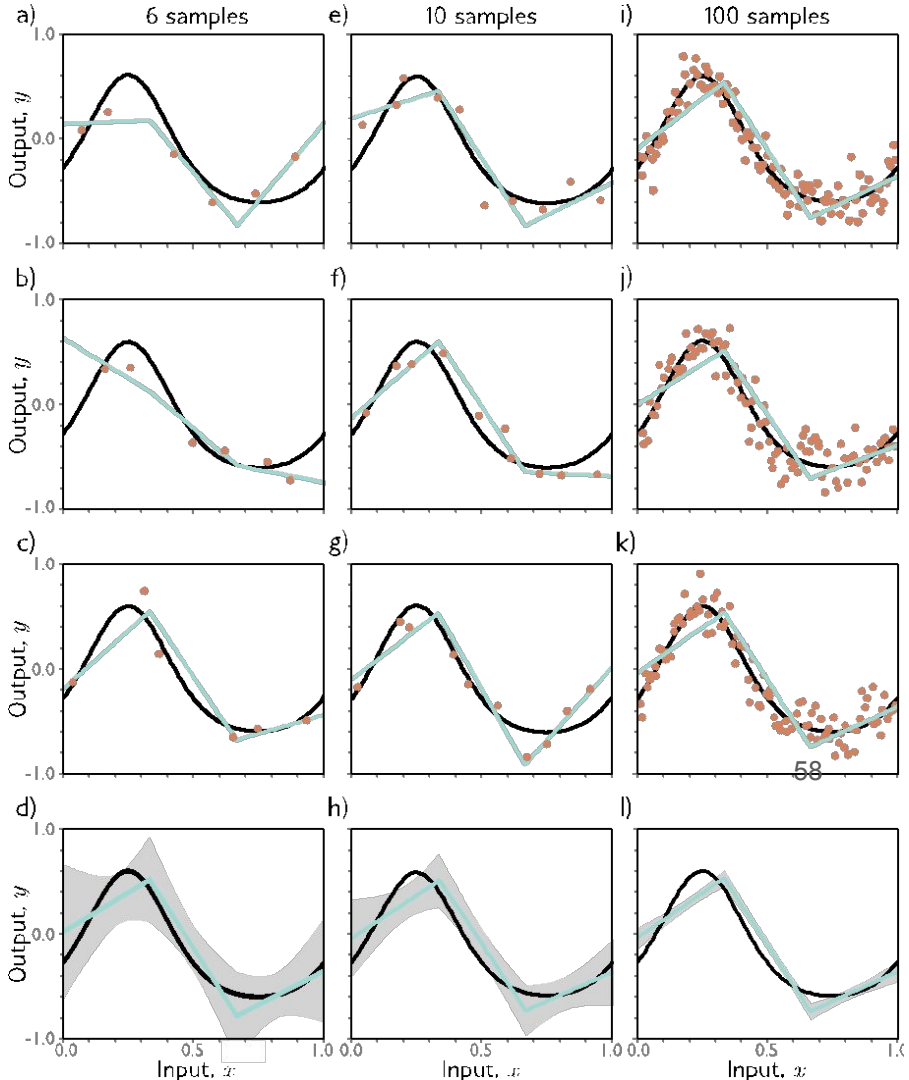
When measuring (capturing) 6 different data samples with a fixed model (e.g. 3 hidden units), we get different optimal fits every time.

Variance



Can reduce variance by adding more samples

Variance



Can reduce variance by adding more samples.

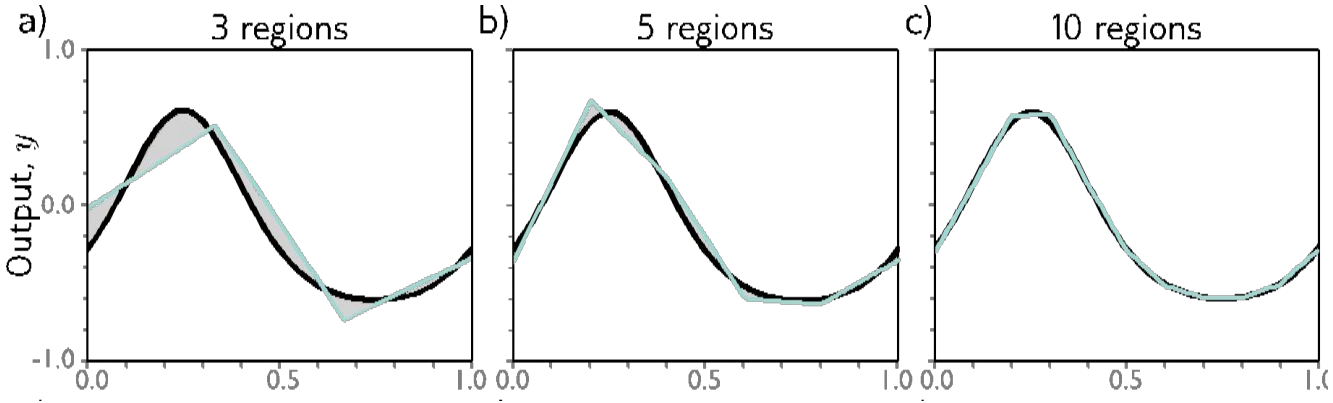
Eventually hits bias limit.

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



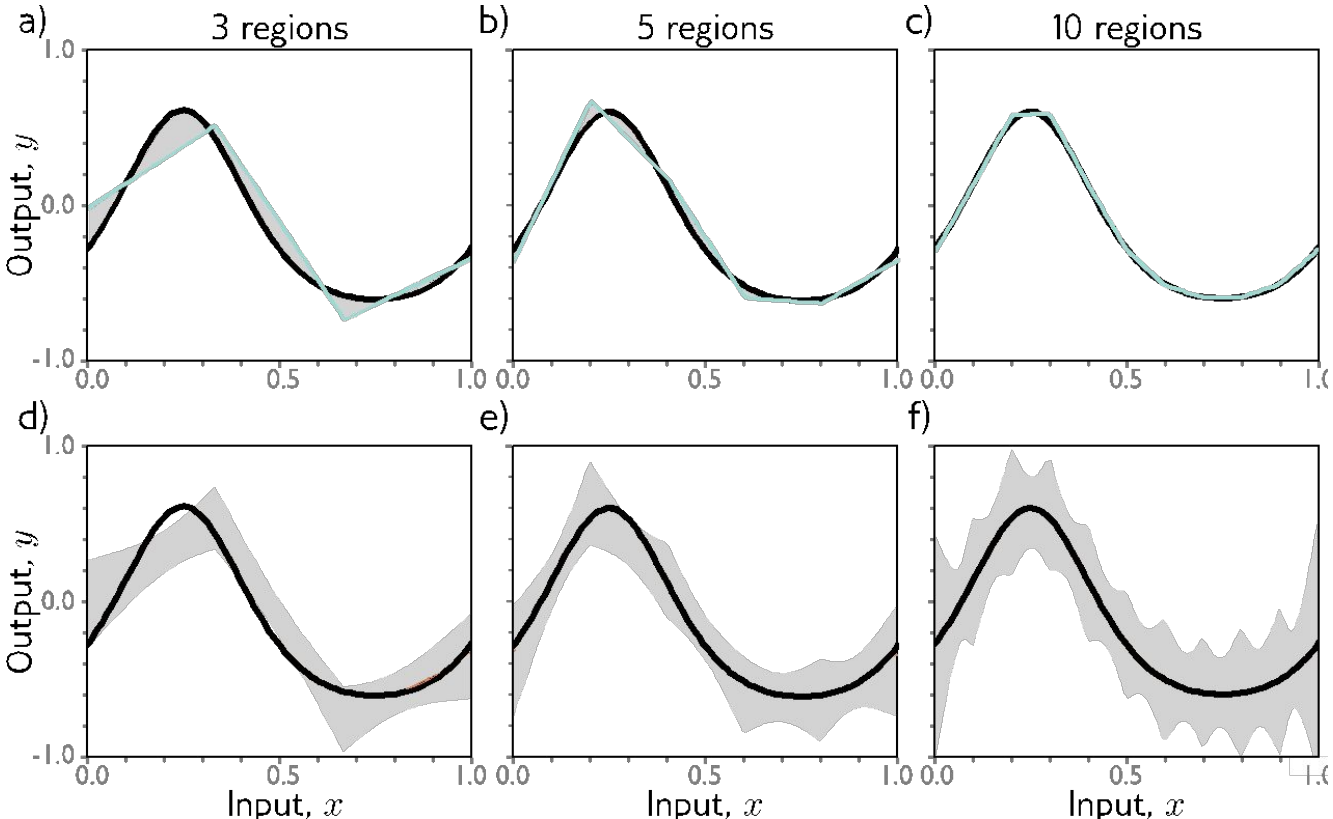
We can reduce bias by adding more model capacity.

In this case, adding more hidden units.

Reducing bias Increases variance!!

Bias

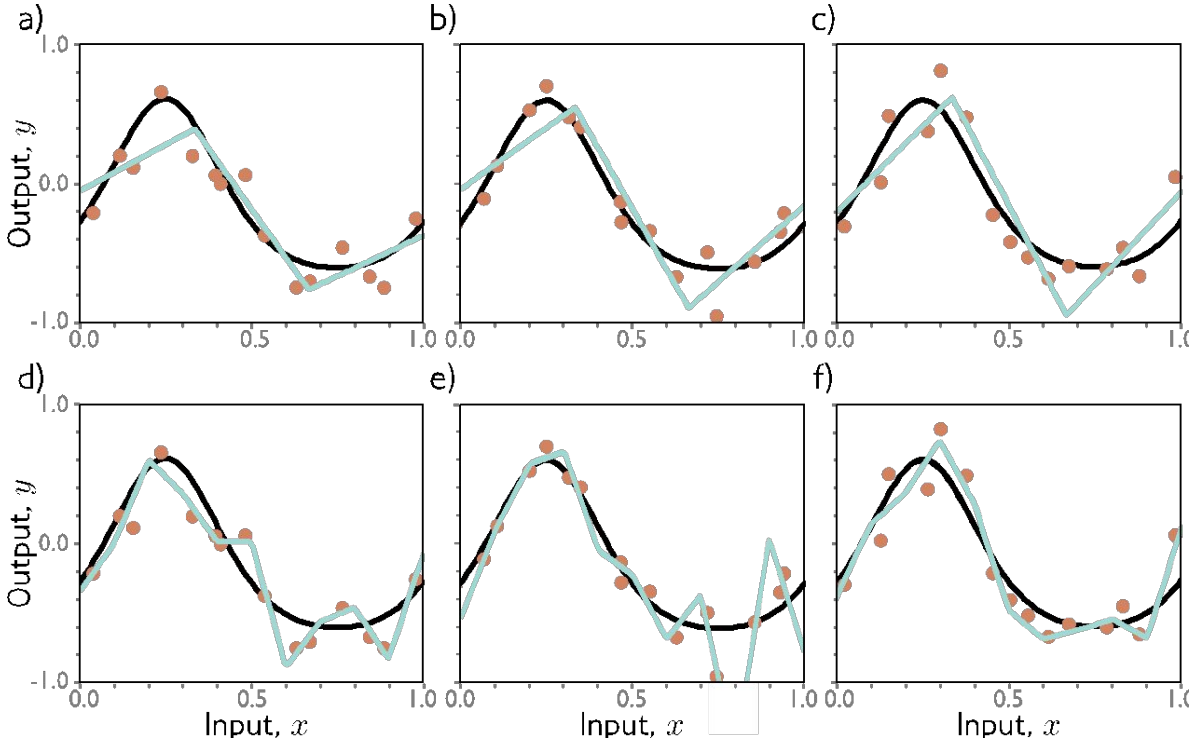
Variance



Why does variance increase? Overfitting

3 Regions

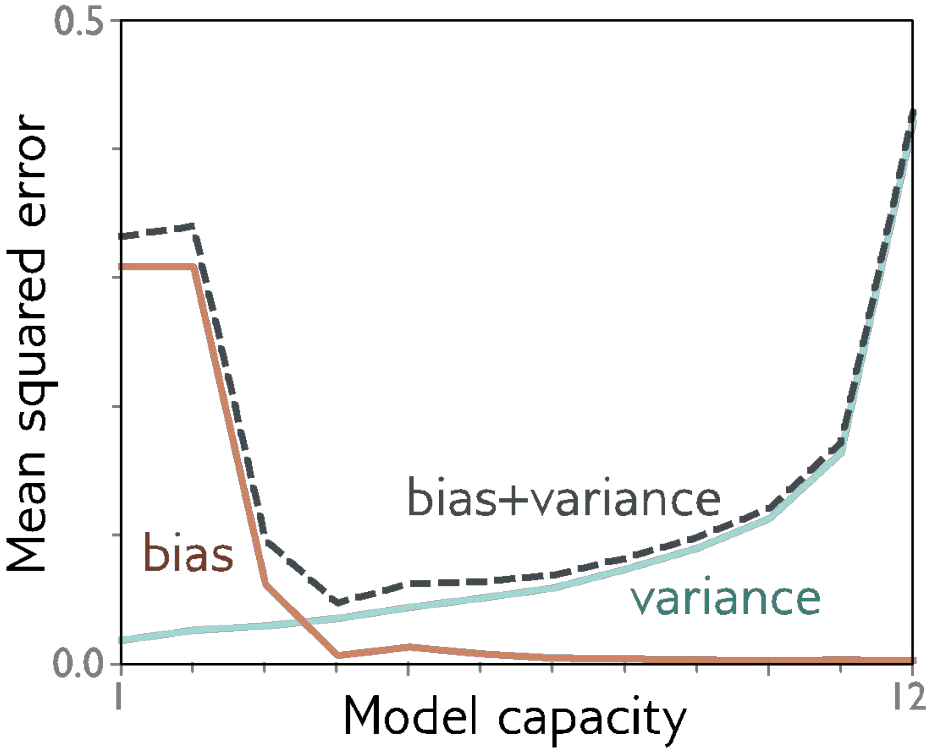
10 Regions

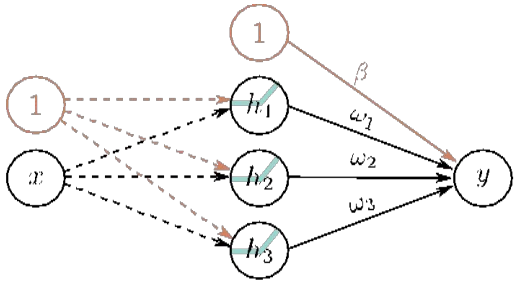


Describes the training data better, but not the true underlying function (black curve)

Many ways to fit a sample of 15 data points

Bias and variance trade-off for the simple linear model





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

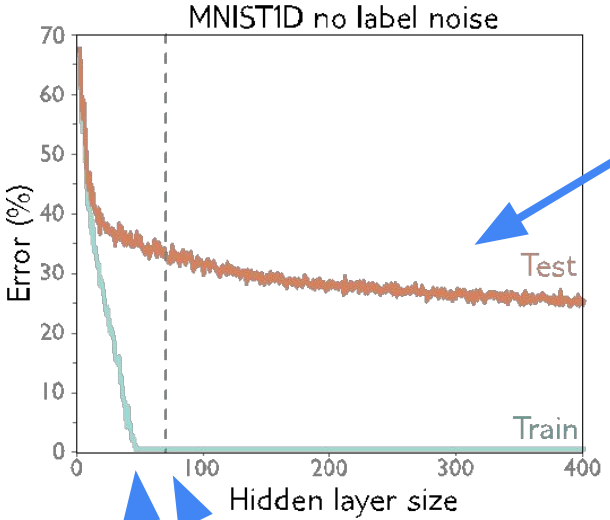
Number of hidden units

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
- Preview for next time

Train and Test Error versus # of Hidden Layers



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

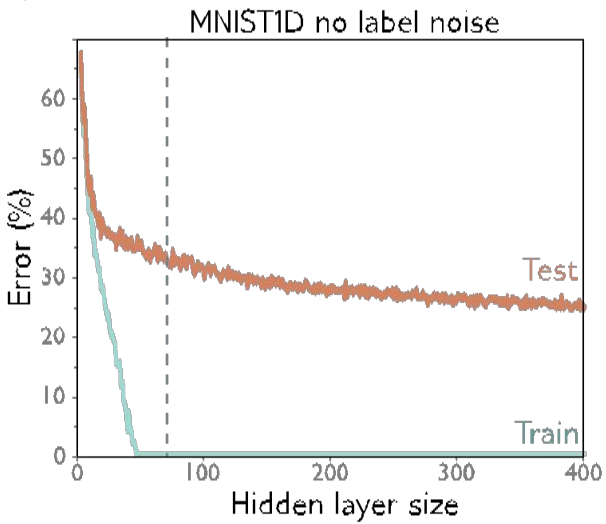
- 10,000 training examples
- 5,000 test examples
- Two hidden layers
- Adam optimizer
- Step size of 0.005
- Full batch
- 4000 training steps

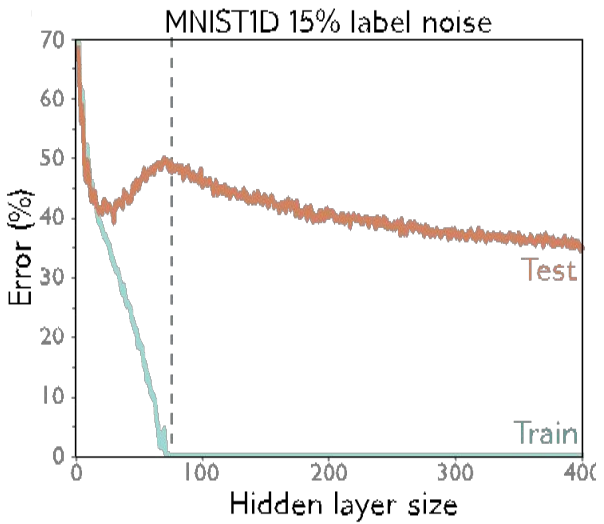
Training parameters = Training examples

Model has *memorized* the training set

Why do we say that?

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

Measuring performance

- MNIST1D dataset model and performance
- Noise, bias, and variance
- Reducing variance
- Reducing bias & bias-variance trade-off

Next Time

- Double descent
- Choosing hyperparameters

Next Time

Measuring Performance

- Double descent - intriguing patterns of test performance getting worse then better
- Hyperparameter selection

Regularization

- Heuristics to reduce generalization gap (e.g. test vs training error)

Reading

Today:

- Understanding Deep Learning, Chapter 7 Gradients and initialization
- Understanding Deep Learning, Chapter 8 Measuring Performance

For Wednesday:

- Understanding Deep Learning, Chapter 9 Regularization
- For Valid Generalization the Size of the Weights is More Important than the Size of the Network (skim)
- Train faster, generalize better: Stability of stochastic gradient descent (skim)

Feedback?