Lecture 09

# Multilayer Perceptrons

STAT 479: Deep Learning, Spring 2019

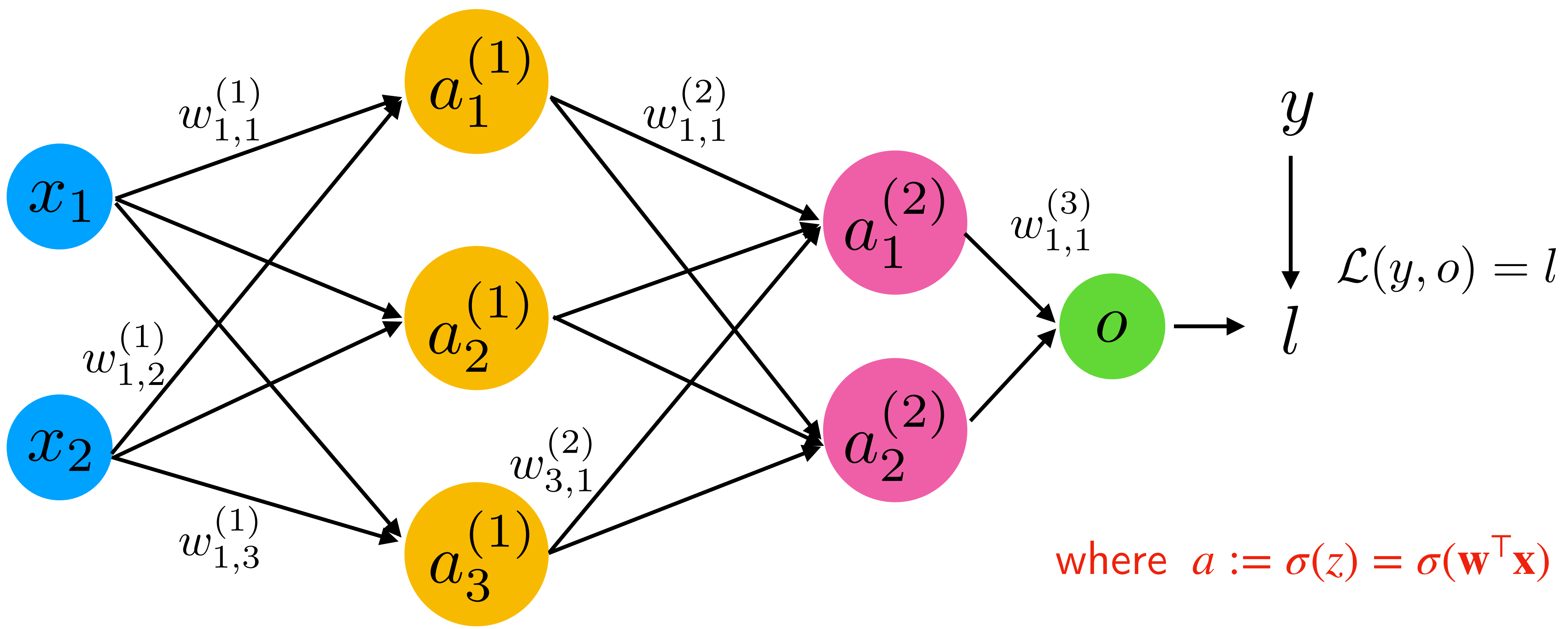
Sebastian Raschka

http://stat.wisc.edu/~sraschka/teaching/stat479-ss2019/

# Project Proposal

# Graph with Fully-Connected Layers = Multilayer Perceptron

Nothing new, really

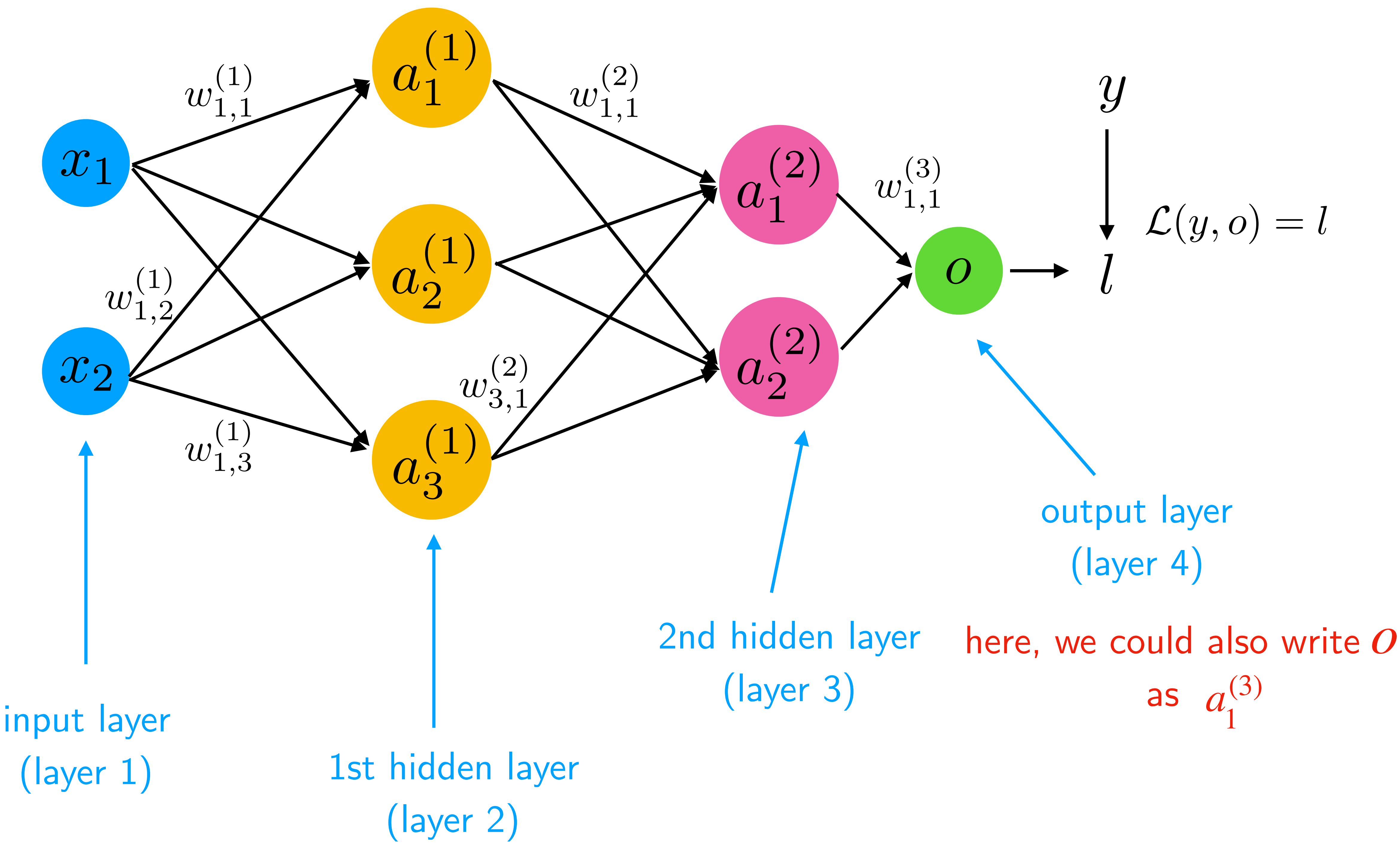



where $a := \sigma(z) = \sigma(\mathbf{w}^\top \mathbf{x})$

$$\frac{\partial l}{\partial w_{1,1}^{(1)}} = \frac{\partial l}{\partial o} \cdot \frac{\partial o}{\partial a_1^{(2)}} \cdot \frac{\partial a_1^{(2)}}{\partial a_1^{(1)}} \cdot \frac{\partial a_1^{(1)}}{\partial w_{1,1}^{(1)}} + \frac{\partial l}{\partial o} \cdot \frac{\partial o}{\partial a_2^{(2)}} \cdot \frac{\partial a_2^{(2)}}{\partial a_1^{(1)}} \cdot \frac{\partial a_1^{(1)}}{\partial w_{1,1}^{(1)}}$$

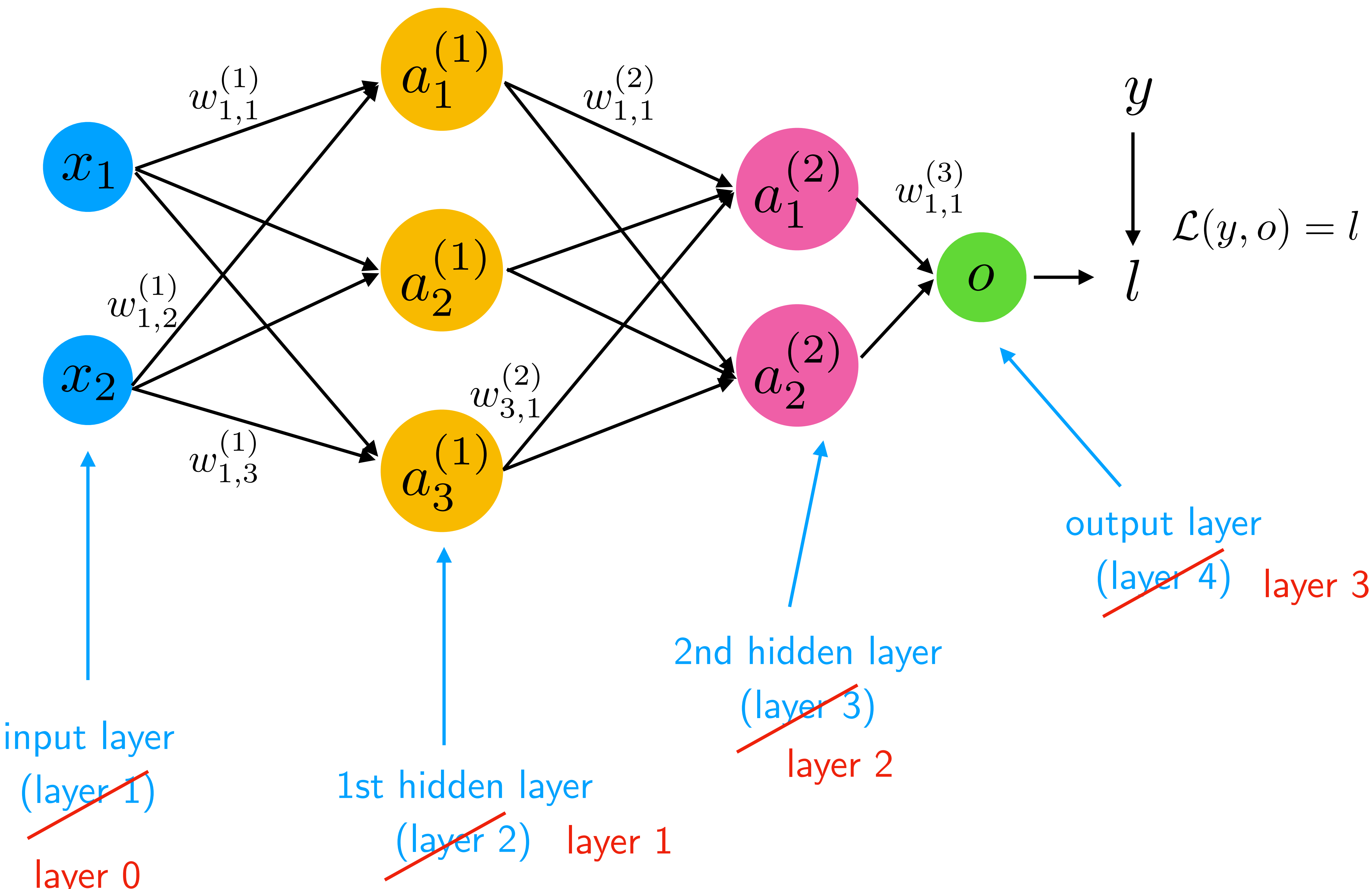
(Assume network for binary classification)

# Graph with Fully-Connected Layers = Multilayer Perceptron

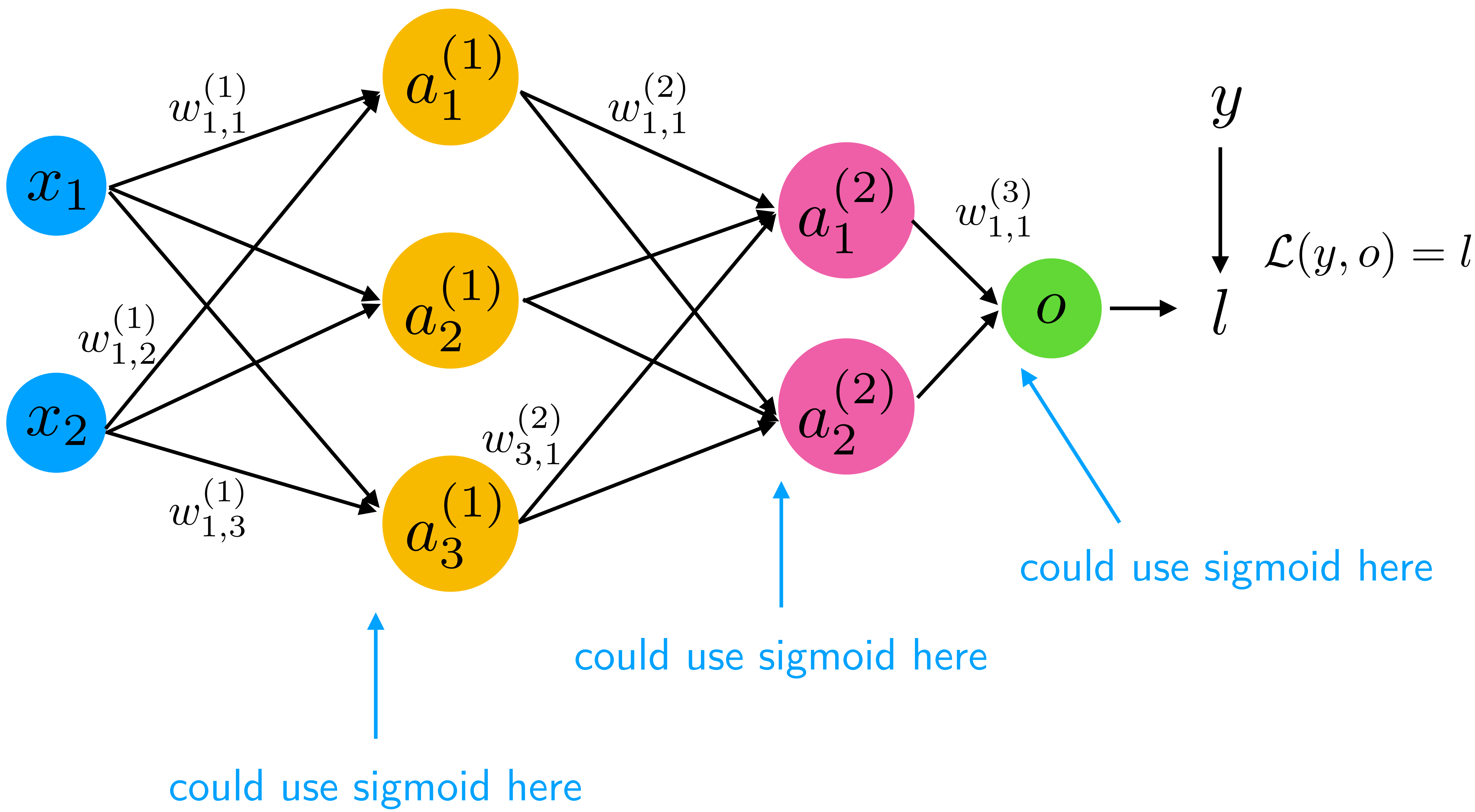


## Graph with Fully-Connected Layers = Multilayer Perceptron

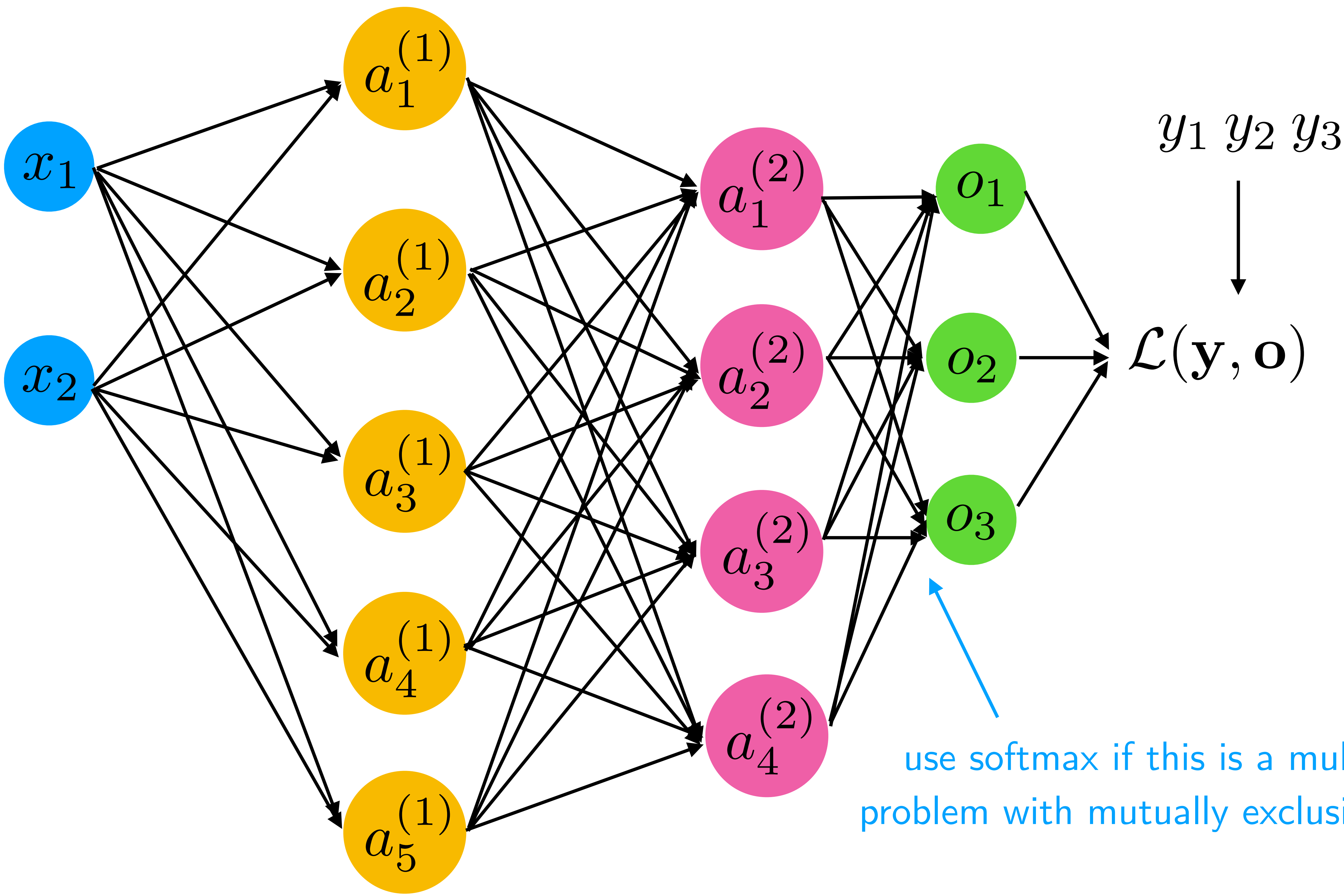
A more common counting/naming scheme, because then a perceptron/Adaline/ logistic regression model can be called a "1-layer neural network"



# Graph with Fully-Connected Layers = Multilayer Perceptron



# Graph with Fully-Connected Layers
# = Multilayer Perceptron



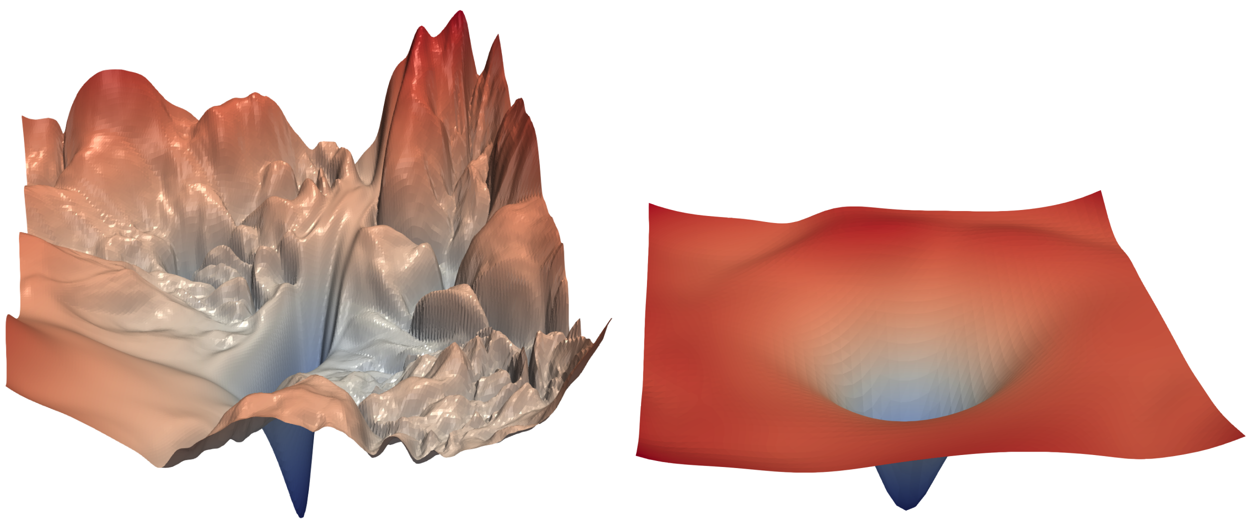

use softmax if this is a multi-class problem with mutually exclusive classes

# Note That the Loss is Not Convex Anymore

- Linear regression, Adaline, Logistic Regression, and Softmax Regression had convex loss functions with respect to the weights

- This is not the case anymore; in practice, we usually end up at different local minima if we repeat the training (e.g., by changing the random seed for weight initialization or shuffling the dataset while leaving all setting the same)

- In practice though, we WANT to explore different starting weights, however, because some lead to better solutions than others



(a) without skip connections (b) with skip connections

Figure 1: The loss surfaces of ResNet-56 with/without skip connections. The proposed filter normalization scheme is used to enable comparisons of sharpness/flatness between the two figures.

32nd Conference on Neural Information Processing Systems (NIPS 2018), Montréal, Canada.

Image Source: Li, H., Xu, Z., Taylor, G., Studer, C. and Goldstein, T., 2018. Visualizing the loss landscape of neural nets. In *Advances in Neural Information Processing Systems* (pp. 6391-6401).

# About Softmax & Sigmoid in the Output Layer and Issues with MSE

- Sigmoid activation + MSE has the problem of very flat gradients when the output is very wrong i.e., $10^{-5}$ probability and class label 1

$$\frac{\partial \mathcal{L}}{\partial w_j} = -\frac{2}{n}(\mathbf{y} - \mathbf{a}) \odot \sigma(\mathbf{z}) \odot (1 - \sigma(\mathbf{z}))\mathbf{x}_j^\top$$ (from HW2: sigmoid + MSE neuron)

- Softmax (forces network to learn probability distribution over labels) in output layer is better than sigmoid because of the mutually exclusive labels as discussed in the Softmax lecture; hence, in output layer, better than sigmoid

# Your 3rd Homework

- Experiment with a multi-layer perceptron on a subset of the QuickDraw dataset (due next Friday, March 8th 11:59 pm)

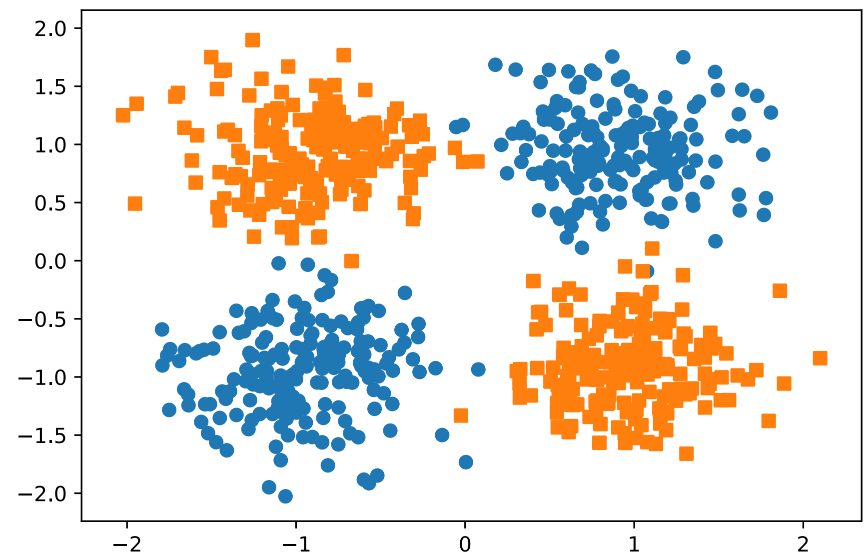
  https://github.com/rasbt/stat479-deep-learning-ss19/tree/master/L09_mlp/code/custom-dataloader

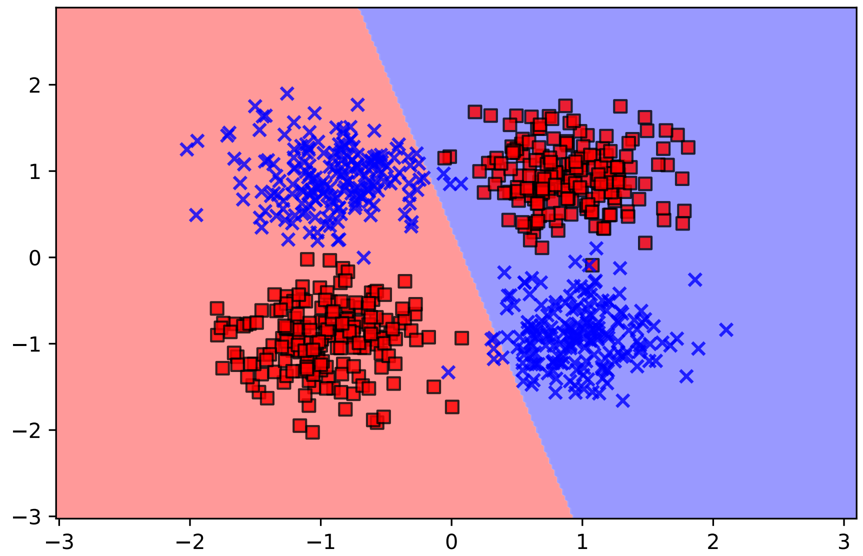
**What happens if we initialize the multi-layer perceptron to all-zero weights?**

# Activation Functions

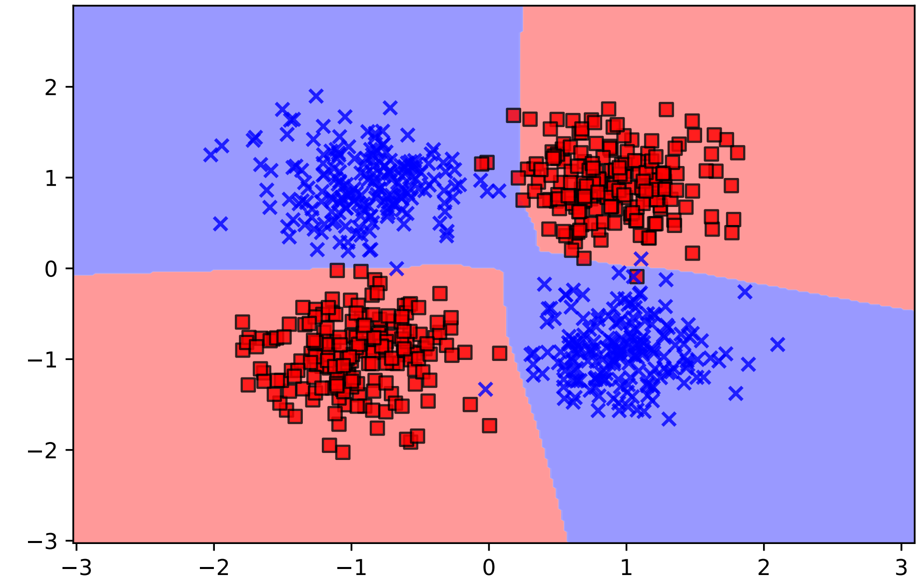
Question: What happens if we don't use non-linear activation functions?

# Solving the XOR Problem with Non-Linear Activations




1-hidden layer MLP
with linear activation function



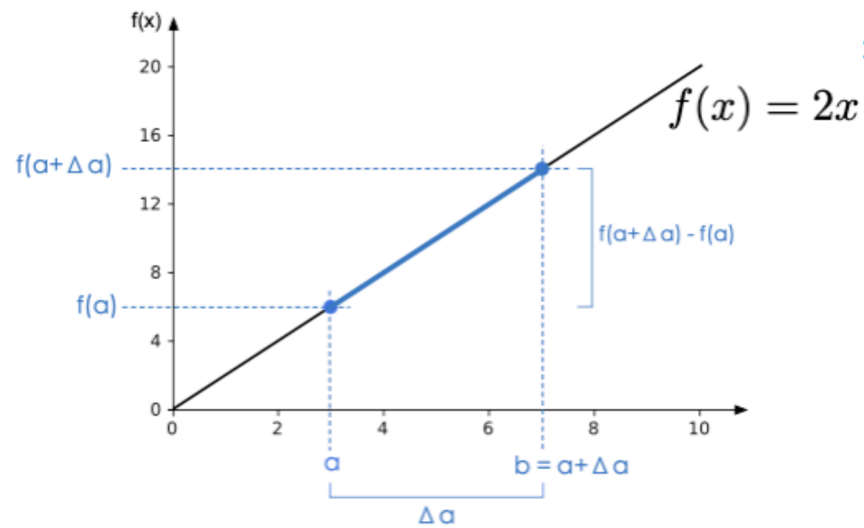

1-hidden layer MLP
with non-linear activation function (ReLU)

https://github.com/rasbt/stat479-deep-learning-ss19/blob/master/L09_mlp/code/xor-problem.ipynb

# Gradient Checking

- Back in the day, we usually checked our gradients manually during debugging (note that this is super slow!)

Derivative of a function = "rate of change" = "slope"




$$\text{Slope} = \frac{f(a+\Delta a) - f(a)}{a + \Delta a - a} = \frac{f(a+\Delta a) - f(a)}{\Delta a}$$

(remember this from the calculus refresher section?)

Usually, a centered version works better, where epsilon is a very small value:

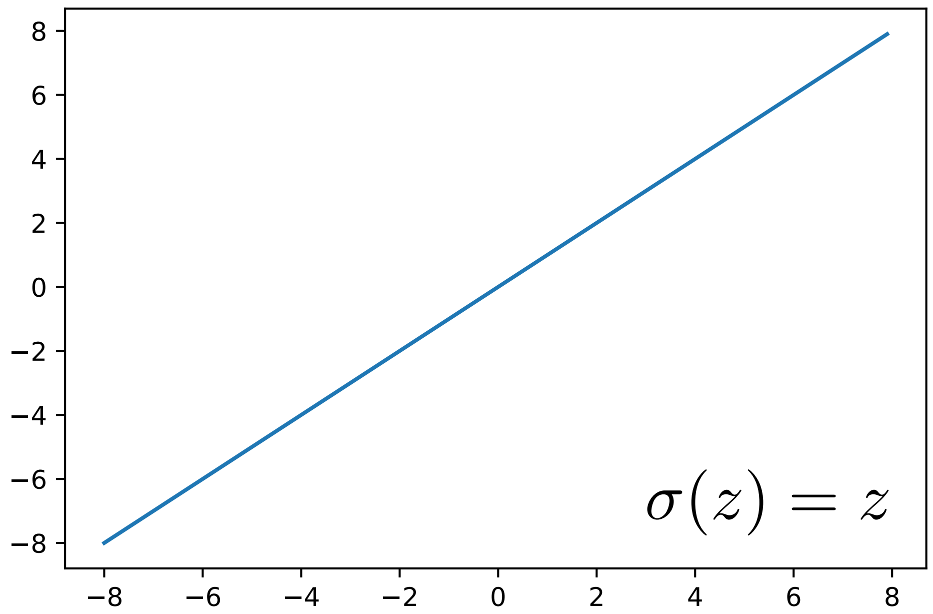
$$\frac{\mathcal{L}\left(w_{i,j}^{(l)} + \varepsilon\right) - \mathcal{L}\left(w_{i,j}^{(l)} - \varepsilon\right)}{2\varepsilon}$$

(then compare this with the symbolic gradient and compute the difference, e.g., via L2 norm)

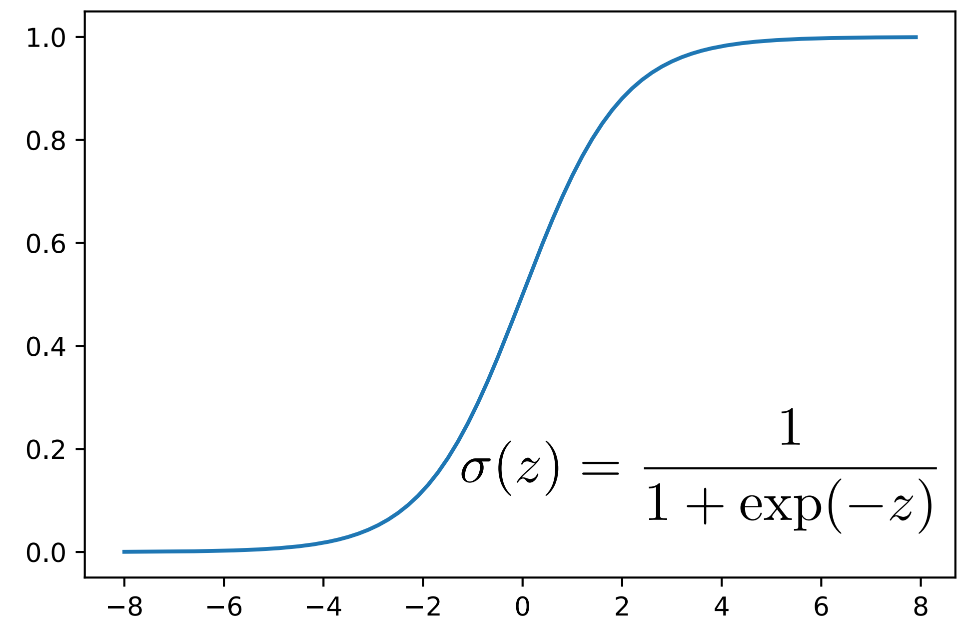
- Rarely done in practice anymore because we usually nowadays use autograd anyway, due to the complexity of deep neural networks

# A Selection of Common Activation Functions (1)

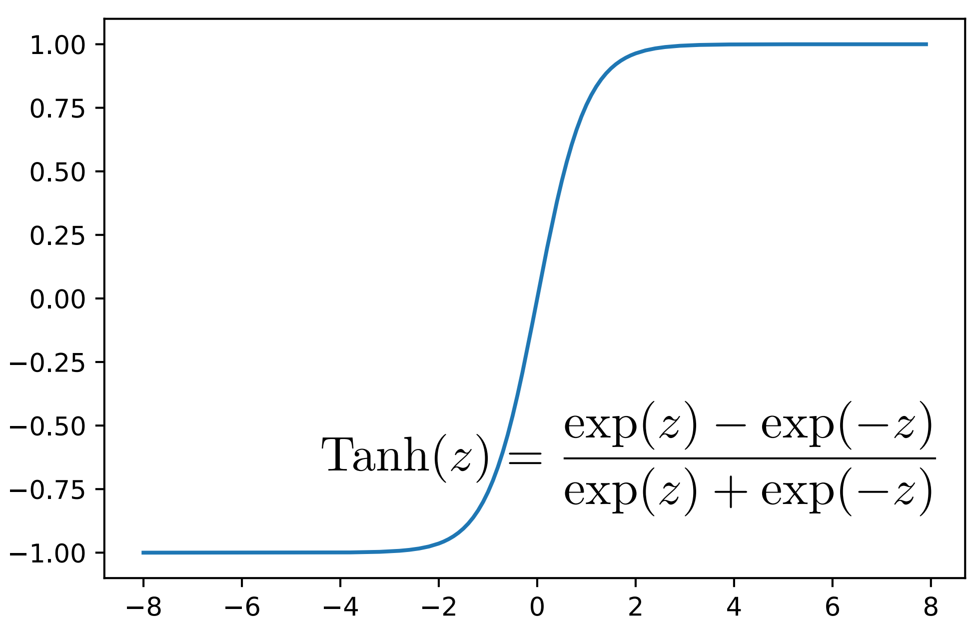
## Identity



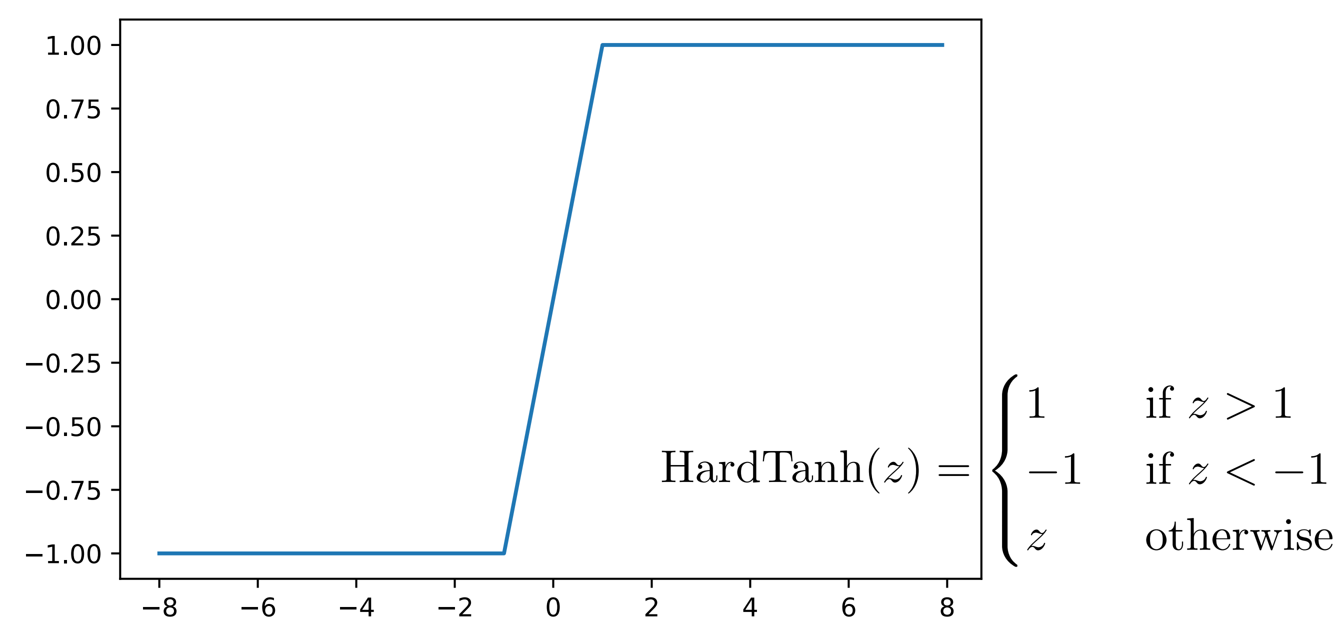

## (Logistic) Sigmoid




## Tanh ("tanH")

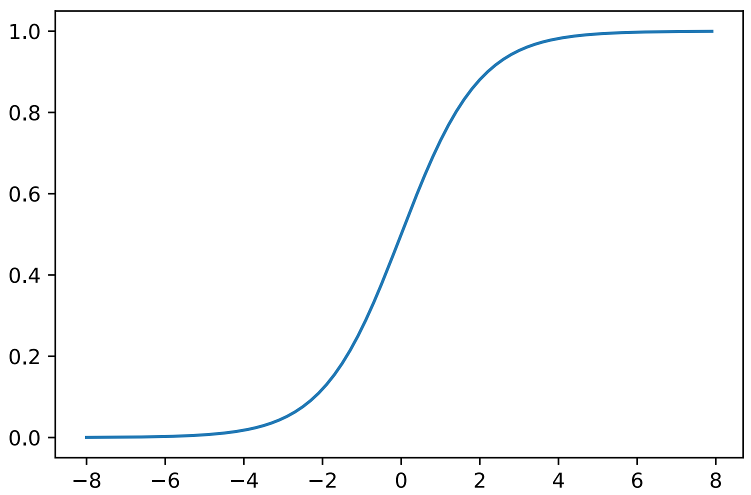



## Hard Tanh



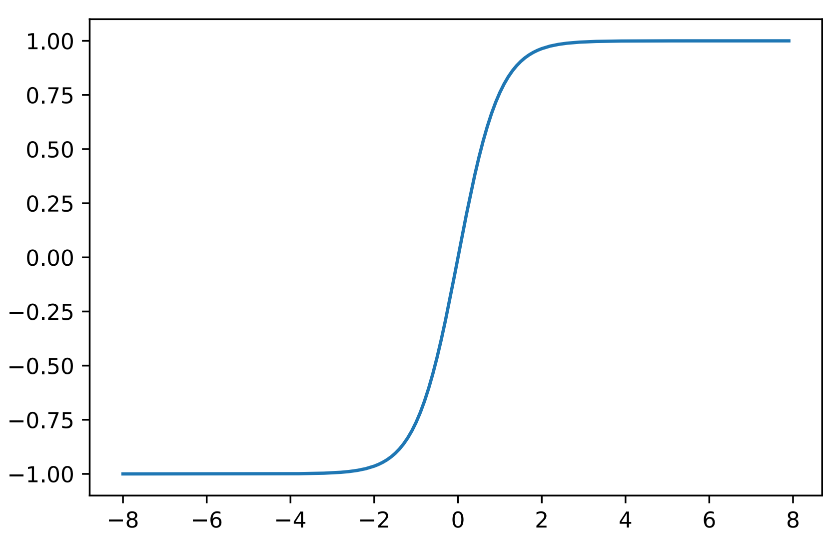
# A Selection of Common Activation Functions (1)

(Logistic) Sigmoid




- Advantages of Tanh
- Mean centering
- Positive and negative values
- Larger gradients

Tanh ("tanH")



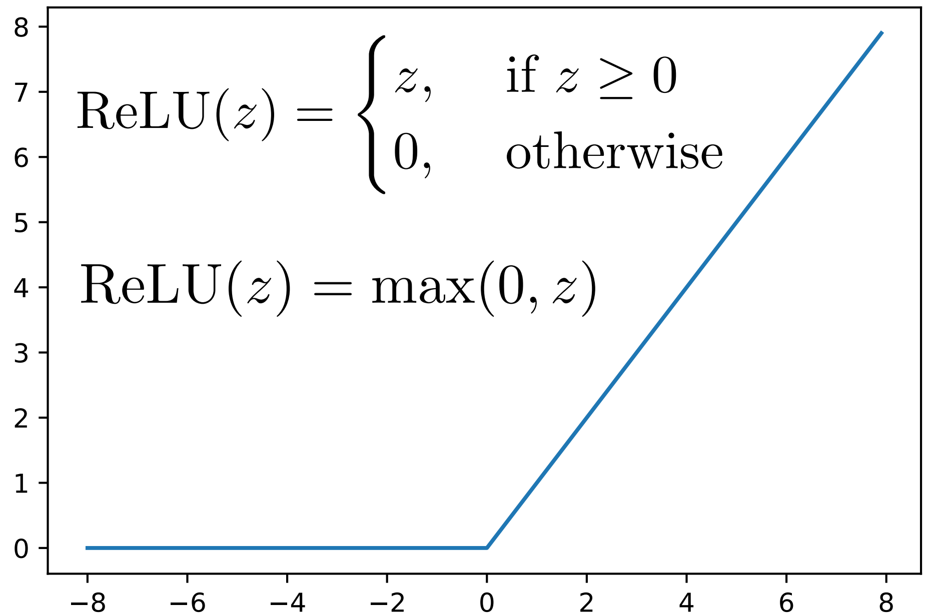

Additional tip: Also good to normalize inputs to mean zero and use random weight initialization with avg. weight centered at zero

Also simple derivative:

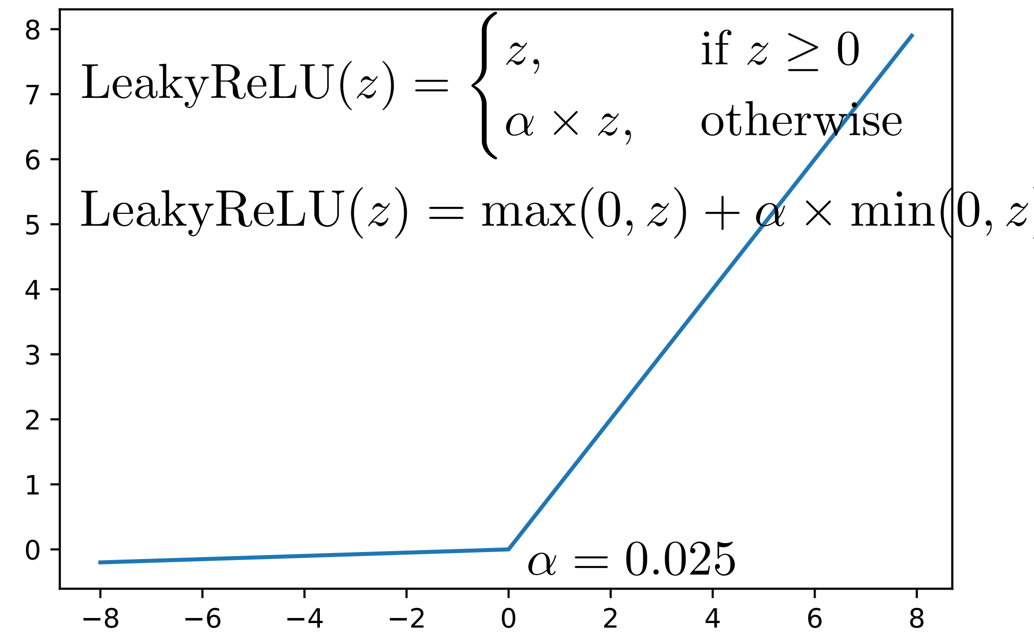
$$\frac{d}{dz}Tanh(z) = 1 - Tanh(z)^2$$

# A Selection of Common Activation Functions (2)

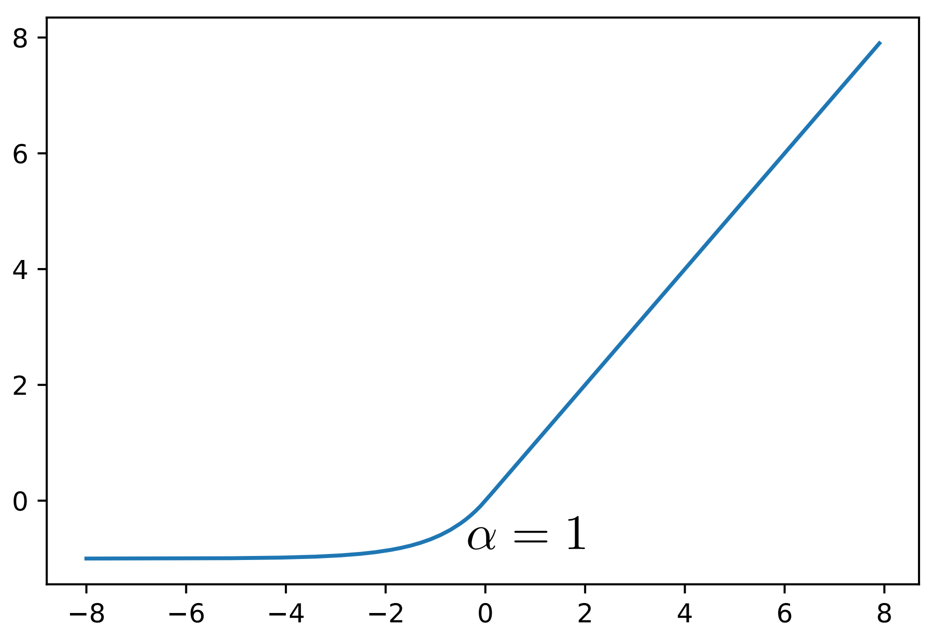
## ReLU (Rectified Linear Unit)




## Leaky ReLU




## ELU (Exponential Linear Unit)




$$\mathrm{ELU}(z) = \max(0, z) + \min(0, \alpha \times (\exp(z) - 1))$$

## PReLU (Parameterized Rectified Linear Unit)

here, alpha is a trainable parameter

$$\mathrm{PReLU}(z) = \begin{cases} z, & \text{if } z \geq 0 \\ \alpha z, & \text{otherwise} \end{cases}$$

$$\mathrm{PReLU}(z) = \max(0, z) + \alpha \times \min(0, z)$$

# Ungraded HW Exercise

Compute/draw the derivatives of these activation functions

# Multilayer Perceptron with Sigmoid Activation and MSE Loss (from scratch)

https://github.com/rasbt/stat479-deep-learning-ss19/blob/master/L09_mlp/code/mlp-fromscratch__sigmoid-mse.ipynb

# Multilayer Perceptron with Sigmoid Activation and MSE Loss

# vs

# Multilayer Perceptron with Softmax Activation and Cross Entropy Loss (in PyTorch)

https://github.com/rasbt/stat479-deep-learning-ss19/blob/master/L09_mlp/code/mlp-pytorch.ipynb

# Dead Neurons

- ReLU is probably the most popular activation function (simple to compute, fast, good results)
- But esp. ReLU neurons might "die" during training
- Can happen if, e.g., input is so large/small that net input is so small that ReLUs never recover (gradient 0 at $x < 0$)
- Not necessarily bad, can be considered as a form of regularization
- (compared to sigmoid/Tanh, ReLU suffers less from vanishing gradient problem but can more easily "explode")

# White vs Deep Architectures (Breadth vs Depth)

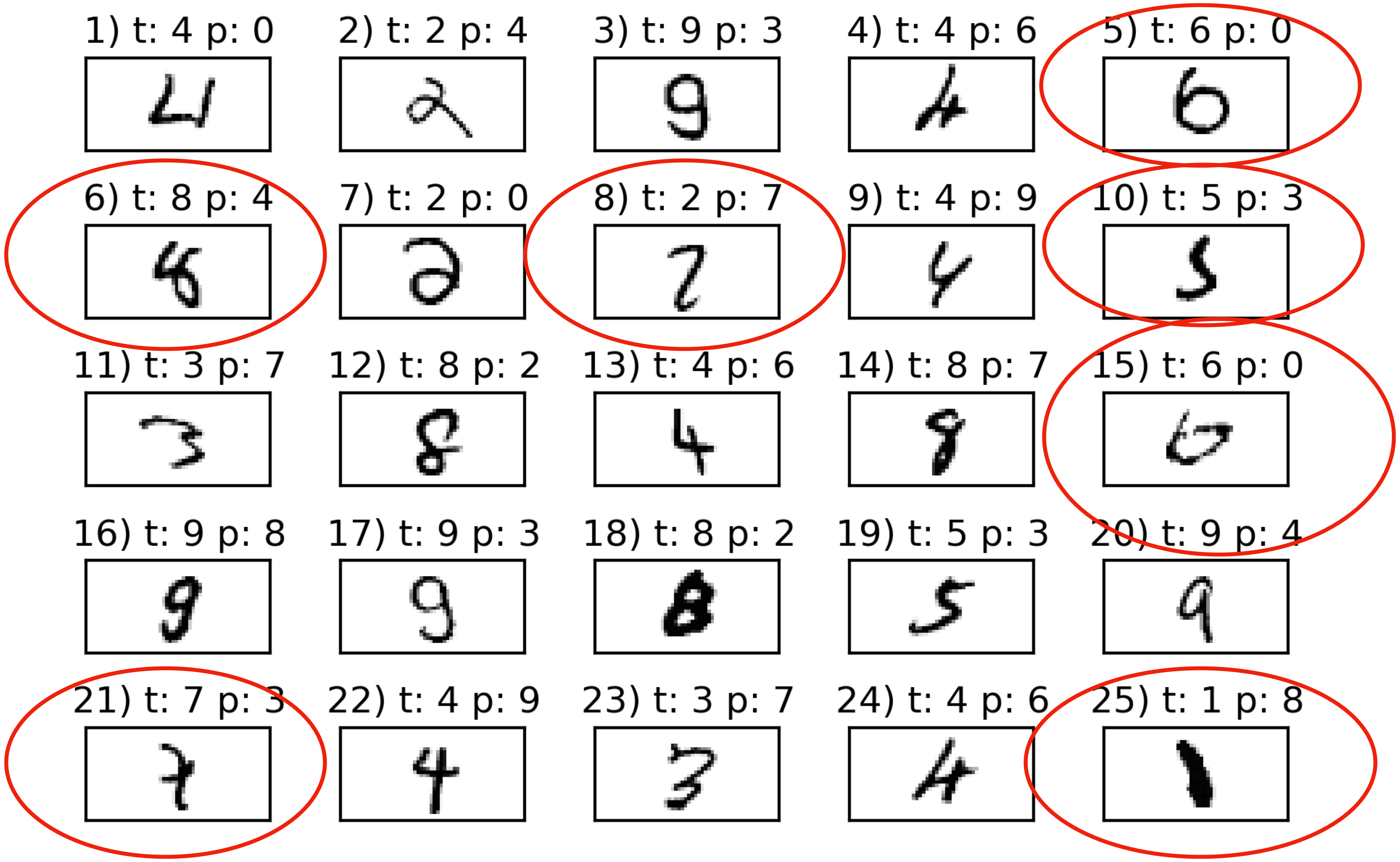
MLP's with one (large) hidden unit are universal function approximators [1-3] already why do we want to use deeper architectures?

[1] Balázs Csanád Csáji (2001) Approximation with Artificial Neural Networks; Faculty of Sciences; Eötvös Loránd University, Hungary
[2] Cybenko, G. (1989) "Approximations by superpositions of sigmoidal functions", Mathematics of Control, Signals, and Systems, 2(4), 303–314. doi:10.1007/BF02551274
[3] Hornik, K., Stinchcombe, M., & White, H. (1989). Multilayer feedforward networks are universal approximators. Neural networks, *2*(5), 359-366.

# White vs Deep Architectures (Breadth vs Depth)

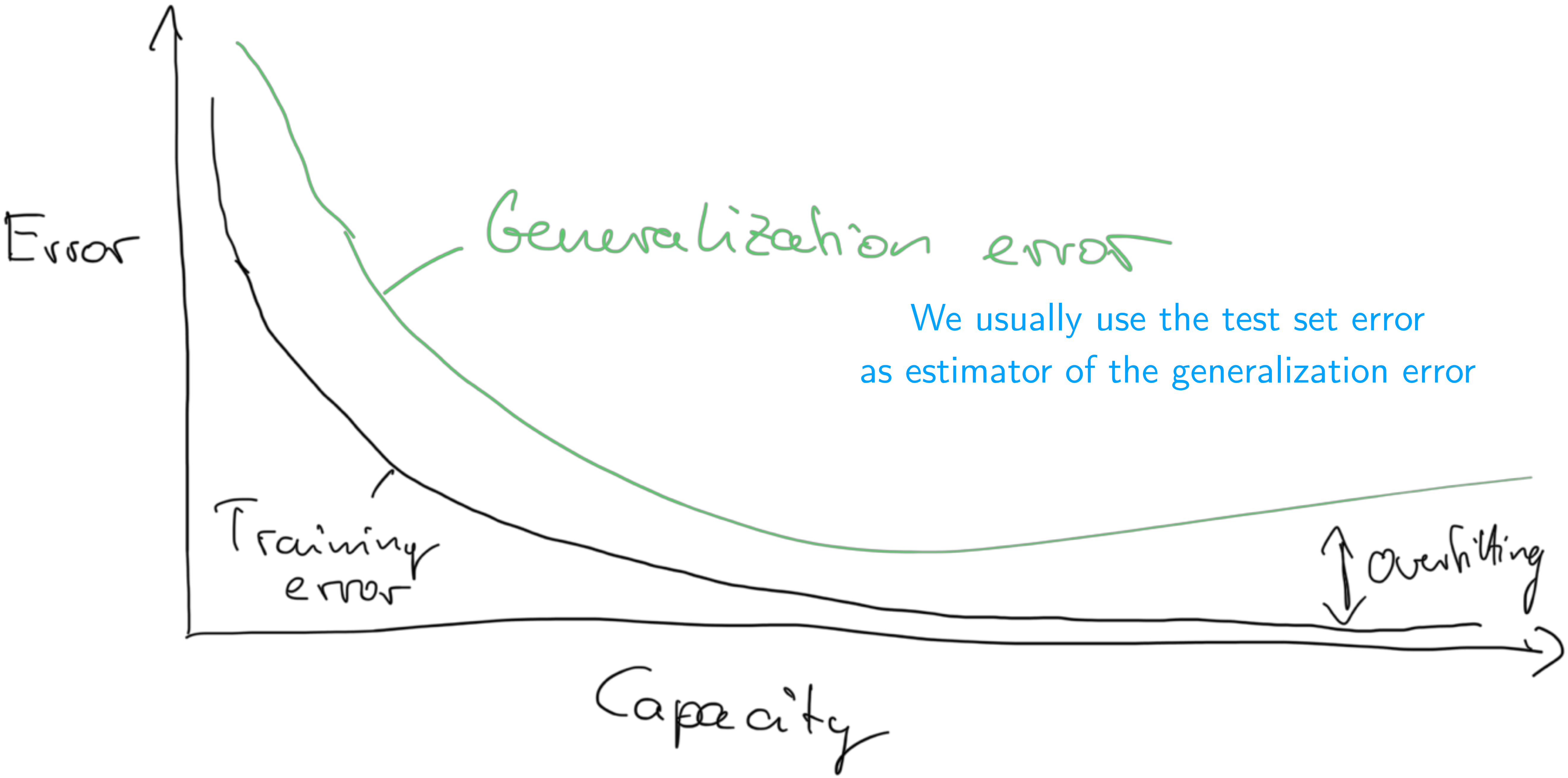
- Can achieve the same expressiveness with more layers but fewer parameters (combinatorics); fewer parameters => less overfitting
- Also, having more layers provides some form of regularization: later layers are constrained on the behavior of earlier layers
- However, more layers => vanishing/exploding gradients
- Later: different layers for different levels of feature abstraction (DL is really more about feature learning than just stacking multiple layers)

# Recommended Practice: Looking at Some Failure Cases




Failure cases of a ~93% accuracy (not very good, but beside the point)
2-layer (1-hidden layer) MLP on MNIST
(where *t=target* class and *p=predicted class*)

# Overfitting and Underfitting




We usually use the test set error
as estimator of the generalization error

# Bias-Variance Decomposition

Details in https://github.com/rasbt/stat479-machine-learning-fs18/blob/master/08_eval-intro/08_eval-intro_notes.pdf

General Definition:

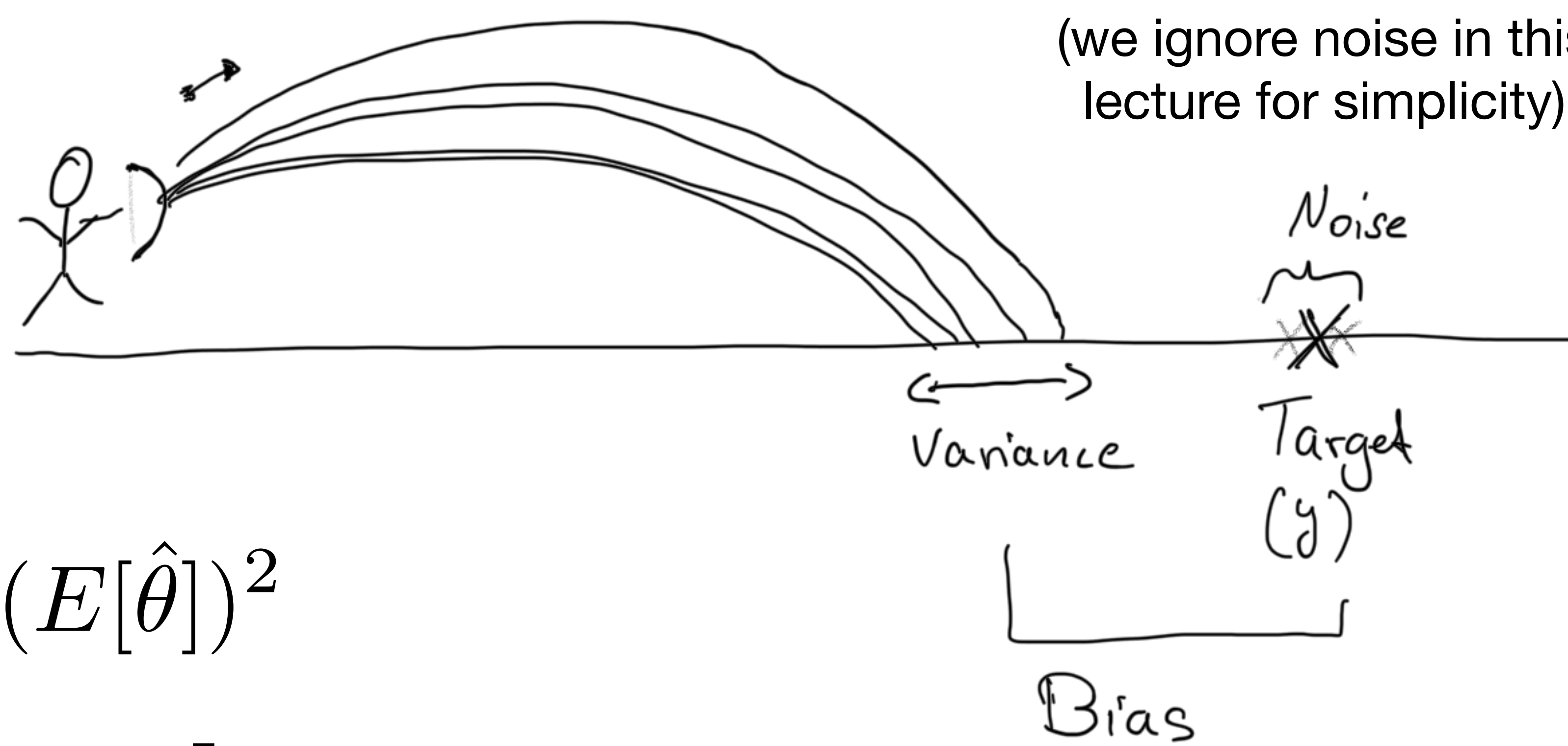
$$\mathrm{Bias}_{\theta}[\hat{\theta}] = E[\hat{\theta}] - \theta$$

$$\mathrm{Var}_{\theta}[\hat{\theta}] = E\left[\hat{\theta}^2\right] - (E[\hat{\theta}])^2$$

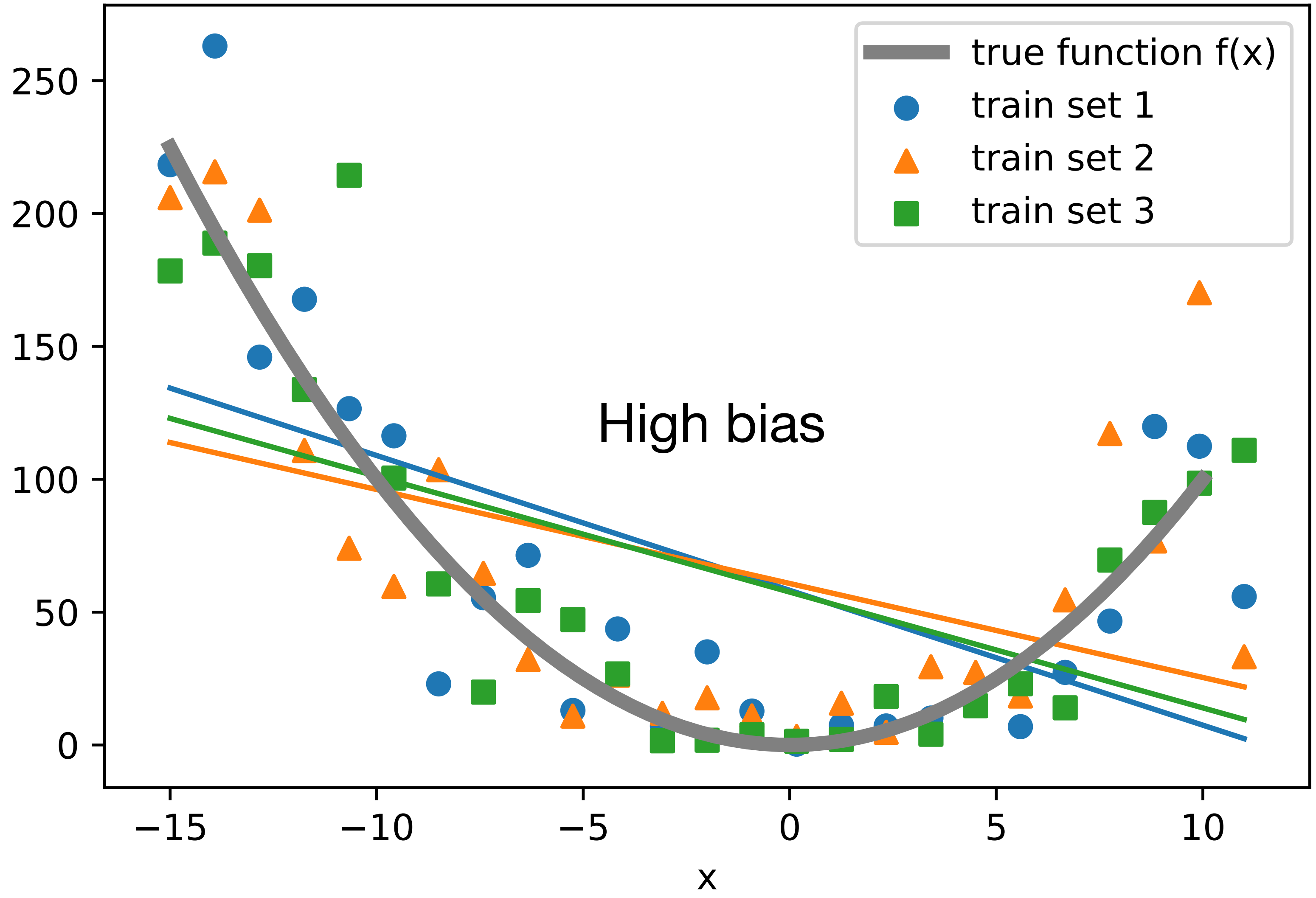
$$\mathrm{Var}_{\theta}[\hat{\theta}] = E\left[(E[\hat{\theta}] - \hat{\theta})^2\right]$$

Intuition:

(we ignore noise in this lecture for simplicity)

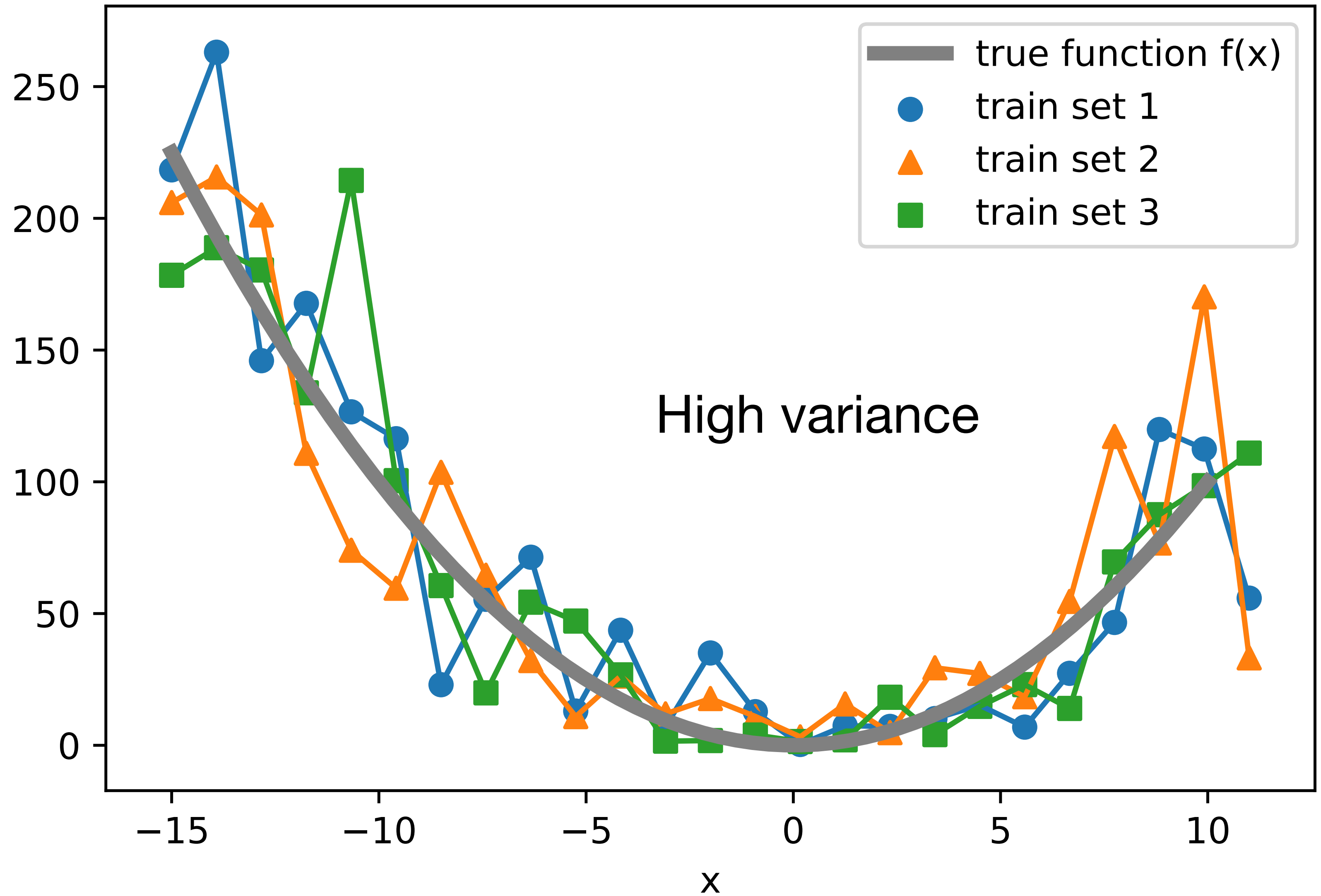


# High Bias Example



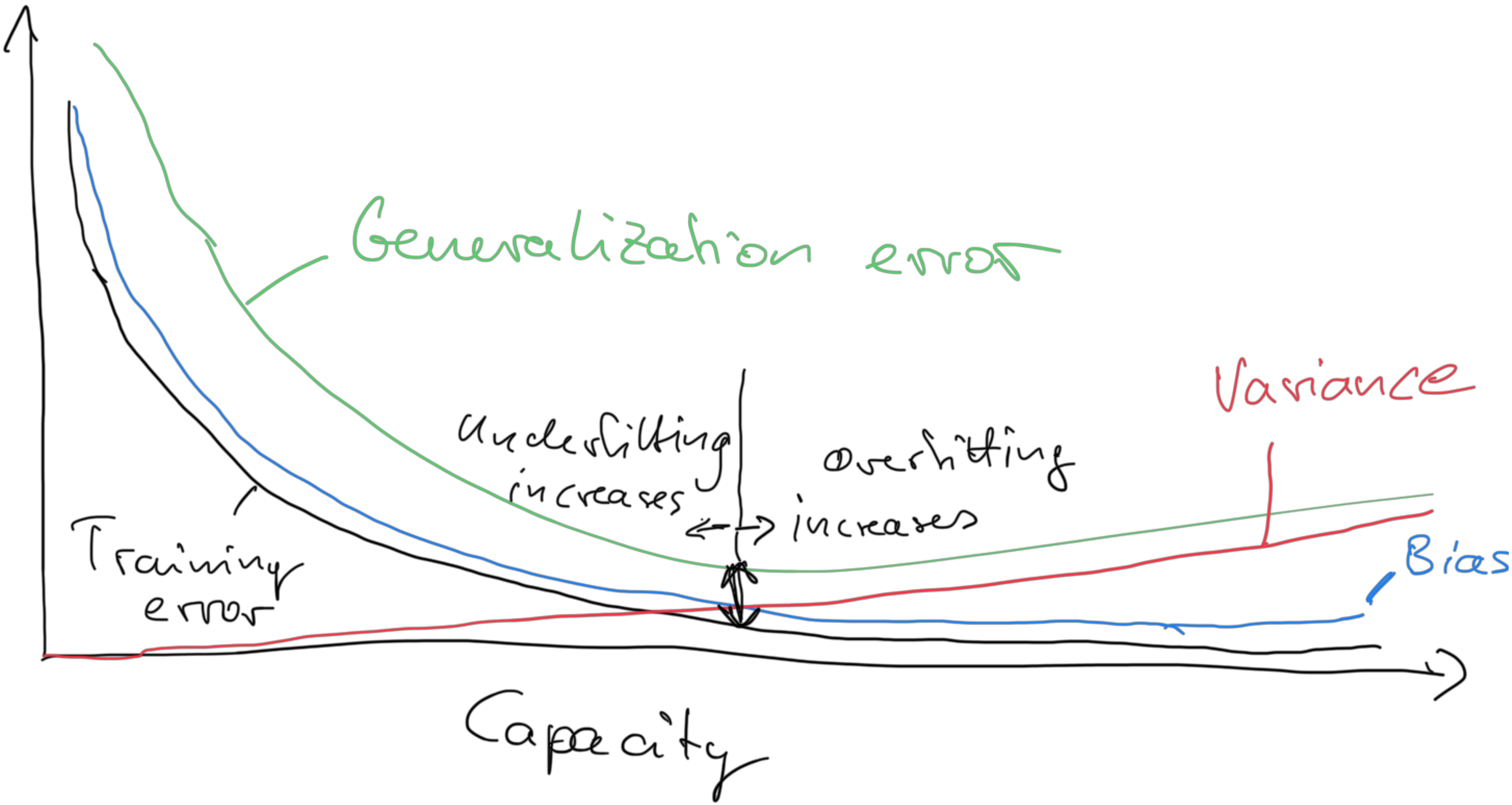

$$\text{Bias}_\theta[\hat{\theta}] = E[\hat{\theta}] - \theta$$

# High Variance Example

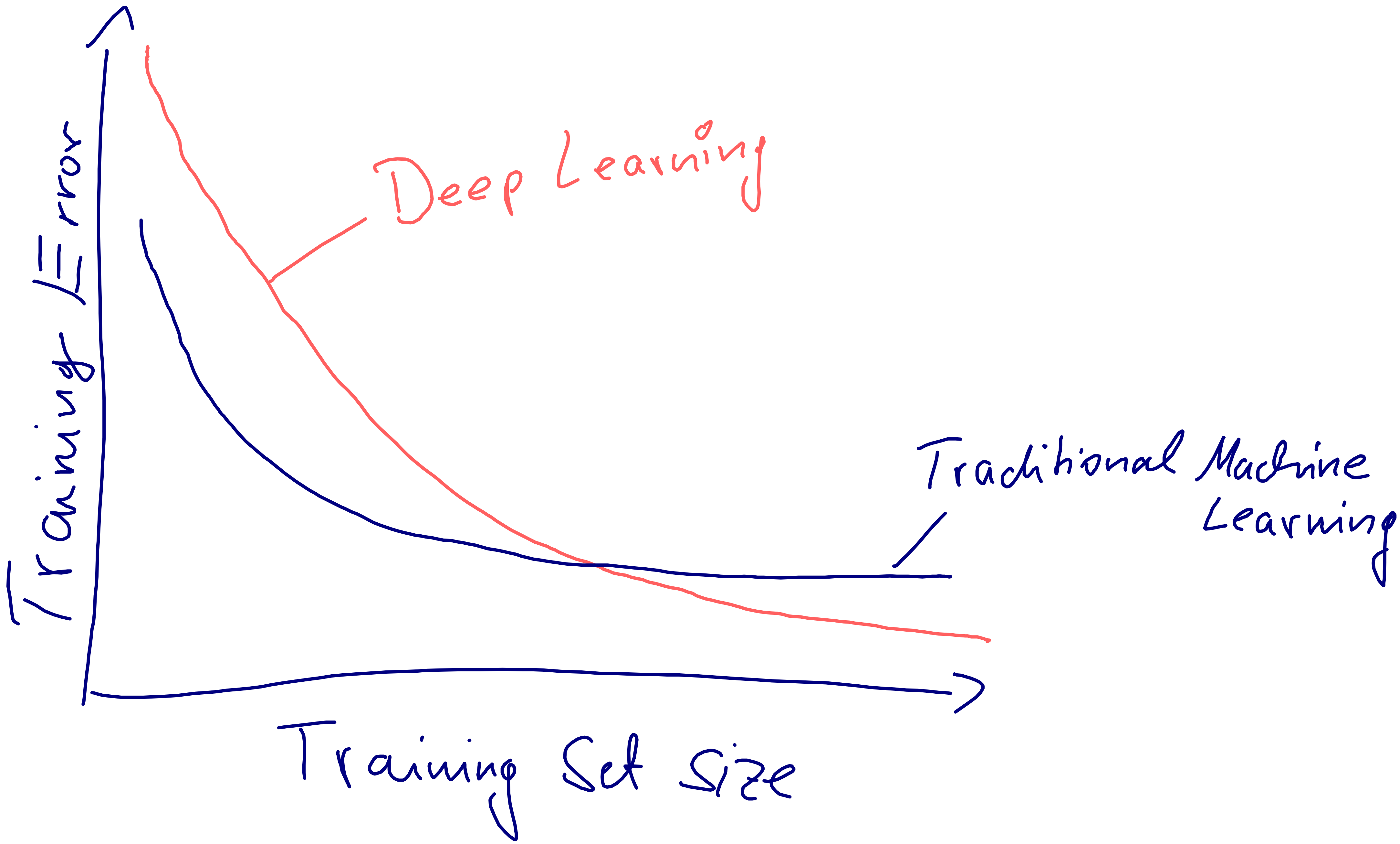



$$\mathrm{Var}_\theta[\hat{\theta}] = E\left[\left(E[\hat{\theta}] - \hat{\theta}\right)^2\right]$$

# Bias & Variance vs Overfitting & Underfitting



# Deep Learning Works Best with Large Datasets



# Bias & Variance vs Overfitting & Underfitting

Be aware when reading DL resources that many researchers use *bias* and *variance* as jargon terms for *underfitting* and *overfitting* (they are related but not the same!)

# Parameters vs Hyperparameters

## Parameters

- weights (weight parameters)
- biases (bias units)

## Hyperparameters

- minibatch size
- data normalization schemes
- number of epochs
- number of hidden layers
- number of hidden units
- learning rates
- (random seed, why?)
- loss function
- various weights (weighting terms)
- activation function types
- regularization schemes (more later)
- weight initialization schemes (more later)
- optimization algorithm type (more later)
- ...

(Mostly no scientific explanation, mostly engineering; need to try many things -> "graduate student descent")

# What does Deep Learning have to do with the Human Brain now?

# About the DataLoader Class ...

- Example showing how you can create your own data loader to efficiently iterate through your own collection of images
(pretend the MNIST images there are some custom image collection)

  https://github.com/rasbt/stat479-deep-learning-ss19/tree/master/L09_mlp/code/custom-dataloader

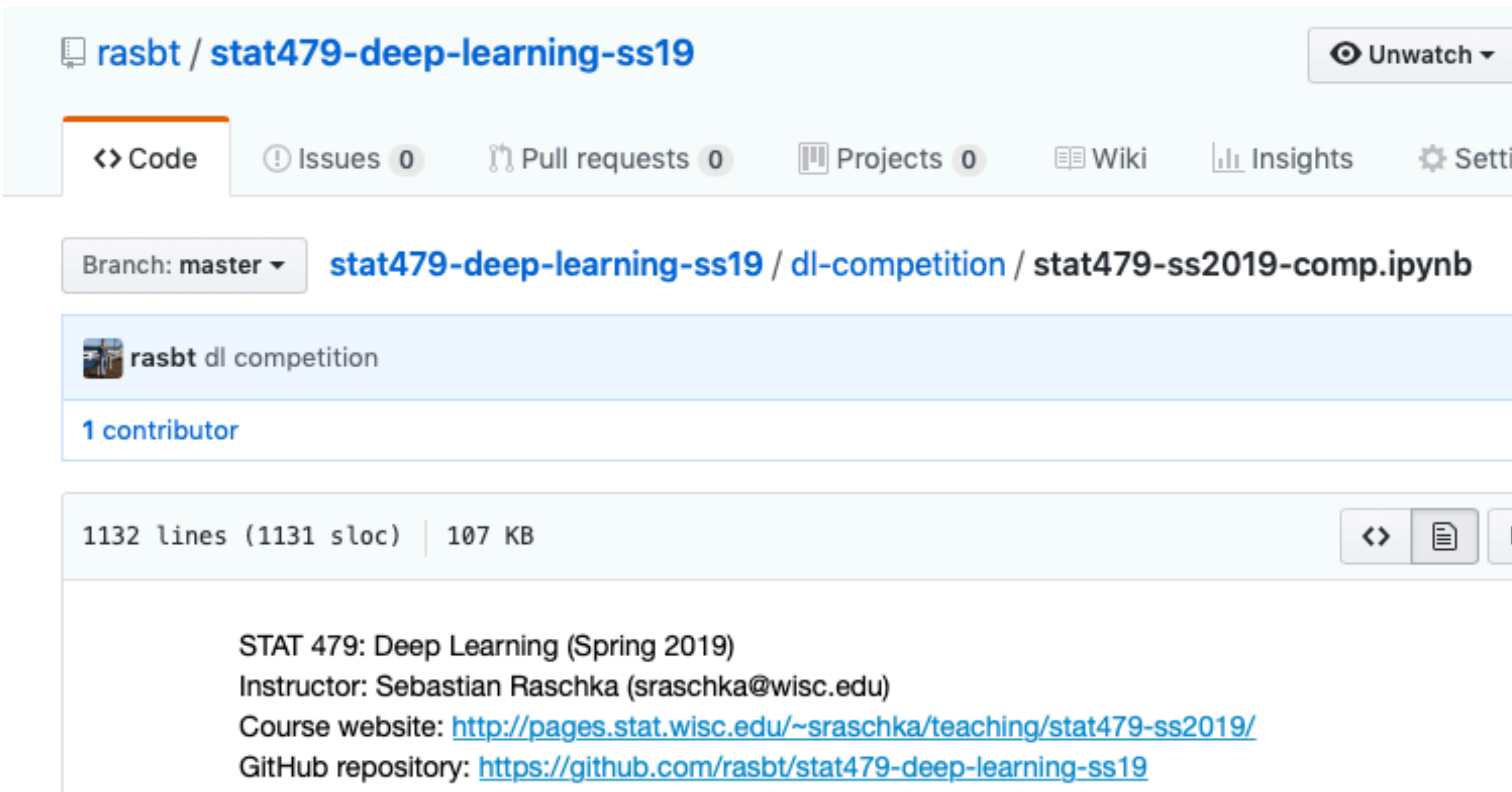
# Reading Assignments

Model Evaluation, Model Selection, and Algorithm Selection in Machine Learning, *pp. 1-15*

https://arxiv.org/pdf/1811.12808.pdf

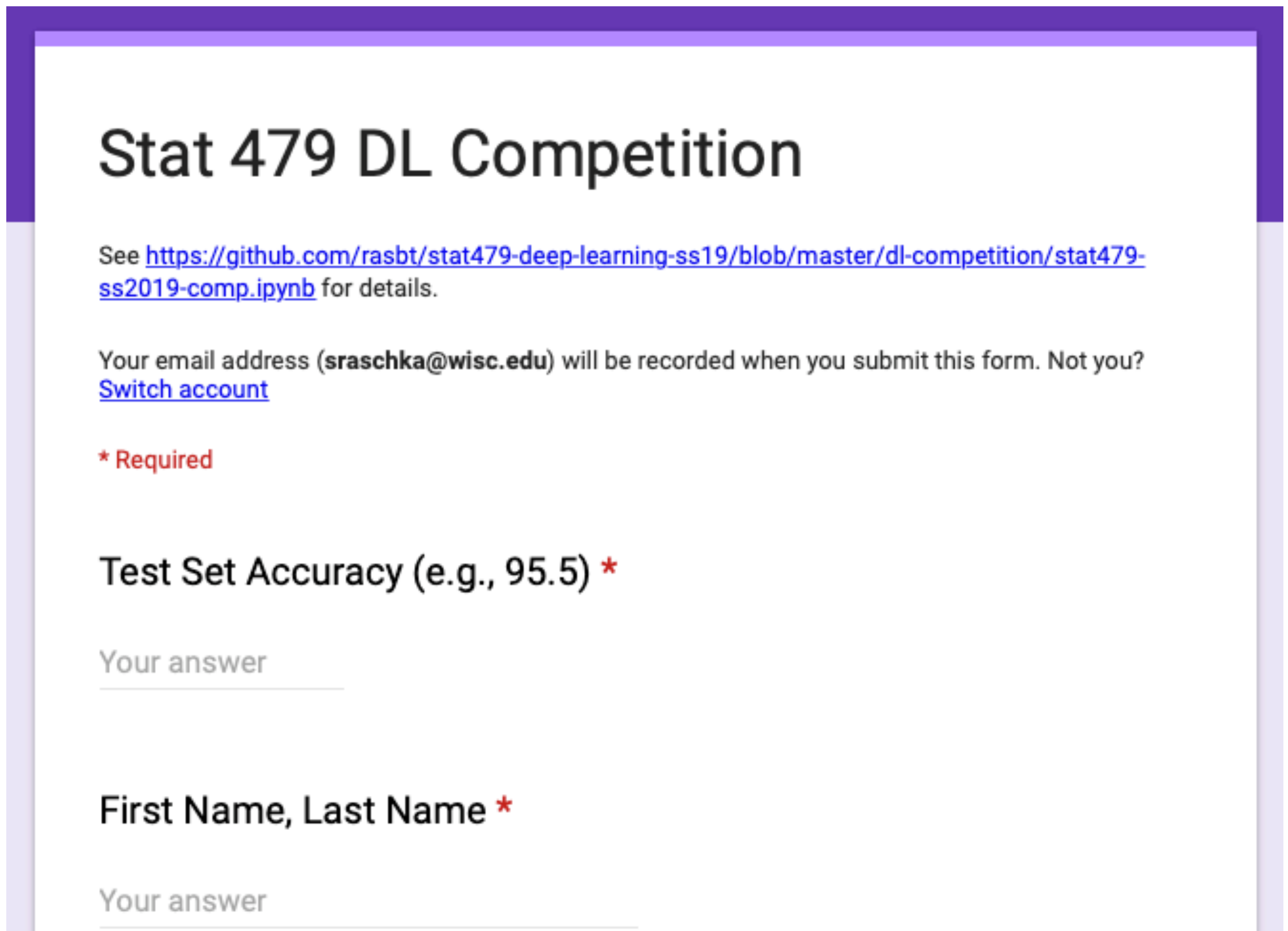
# DL Competition

- Highest accuracy wins (needs to be reproducible)
- \$50 Amazon Gift Card
- Participate alone or in group (up to 5)
- Details in: https://github.com/rasbt/stat479-deep-learning-ss19/blob/master/dl-competition/stat479-ss2019-comp.ipynb



# DL Competition

- Accuracy score submission Form: https://docs.google.com/forms/d/e/1FAIpQLSfvw_JNsImfW0fZbQhUsM5XYeLGEUOCcKrN1Zyb1R0wQ0hd7g/viewform?usp=sf_link (link in Notebook)



## Stat 479 DL Competition

See https://github.com/rasbt/stat479-deep-learning-ss19/blob/master/dl-competition/stat479-ss2019-comp.ipynb for details.

Your email address (**sraschka@wisc.edu**) will be recorded when you submit this form. Not you? Switch account

* Required

Test Set Accuracy (e.g., 95.5) *

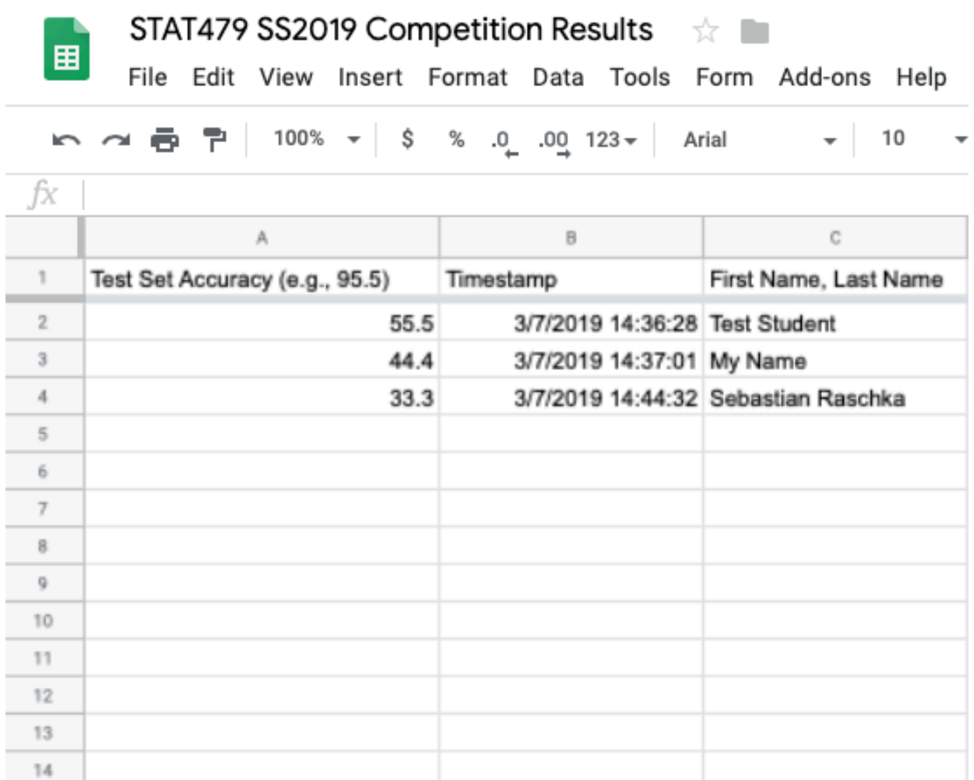
Your answer

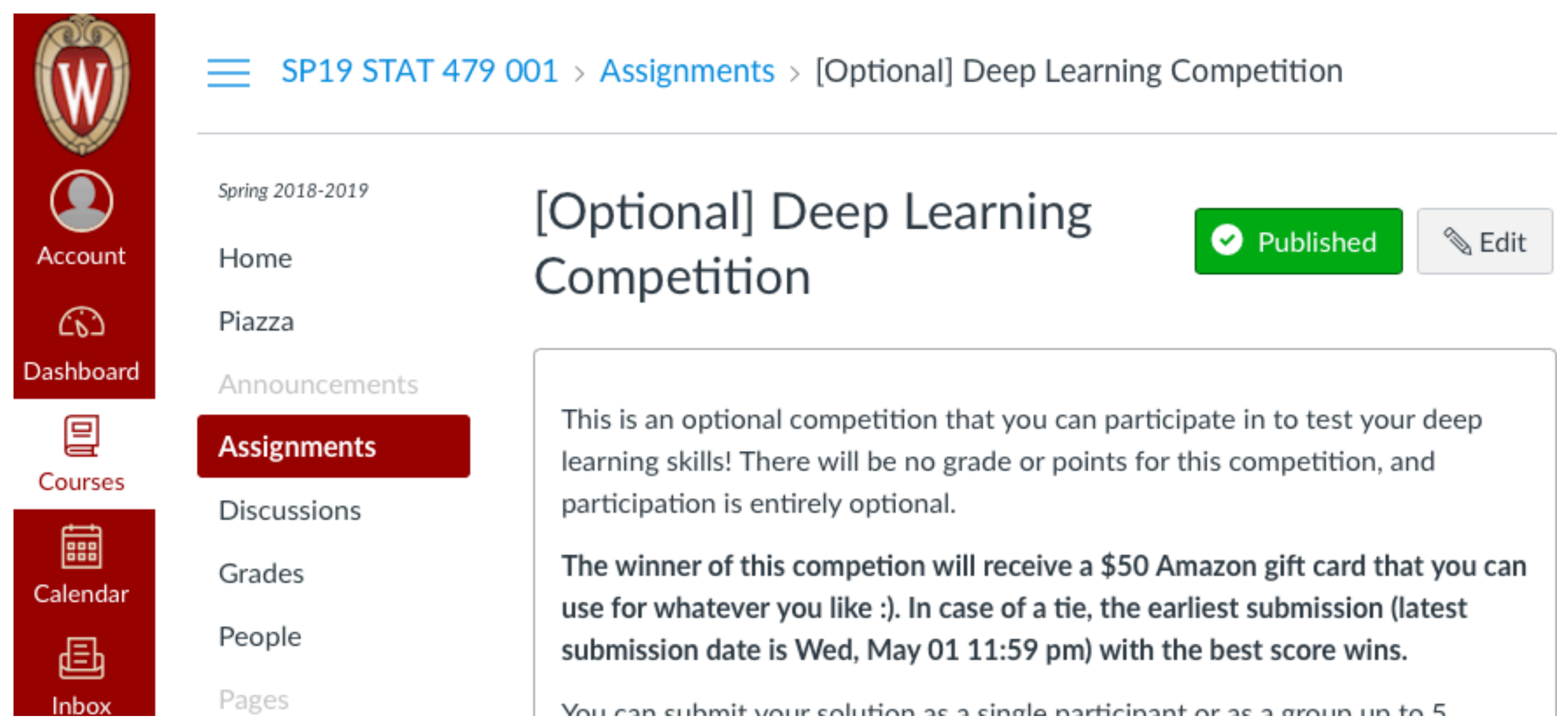
First Name, Last Name *

Your answer

# DL Competition

- Live Leaderboard: https://docs.google.com/spreadsheets/d/11lsz5AT0p6pkYh9Az8ZWxKPD8SleUkq32mv0keIHnEw/edit#gid=1372722537 (link in Notebook)
- Submit code to Canvas until May 1st 11:59 pm




(private, automatically updated, viewing only)