STAT 453: Introduction to Deep Learning and Generative Models

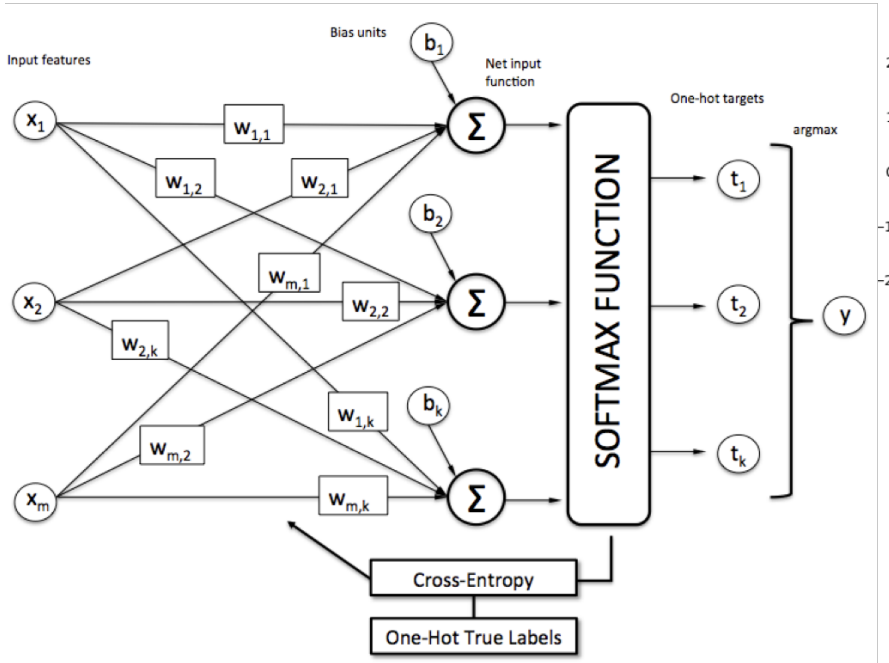
Ben Lengerich

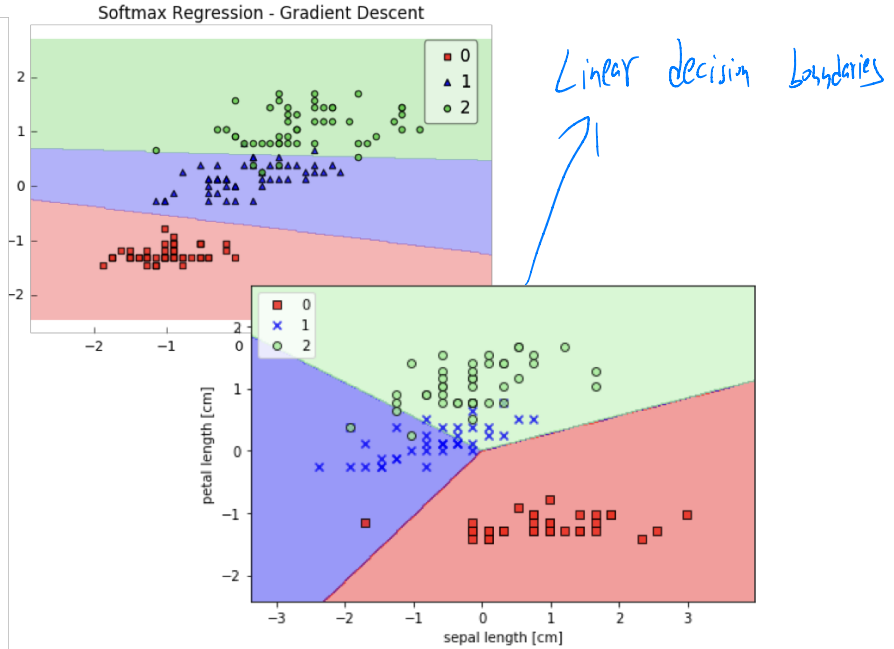
Lecture 09: Multi-layer Perceptrons, Backprop

October 1, 2025

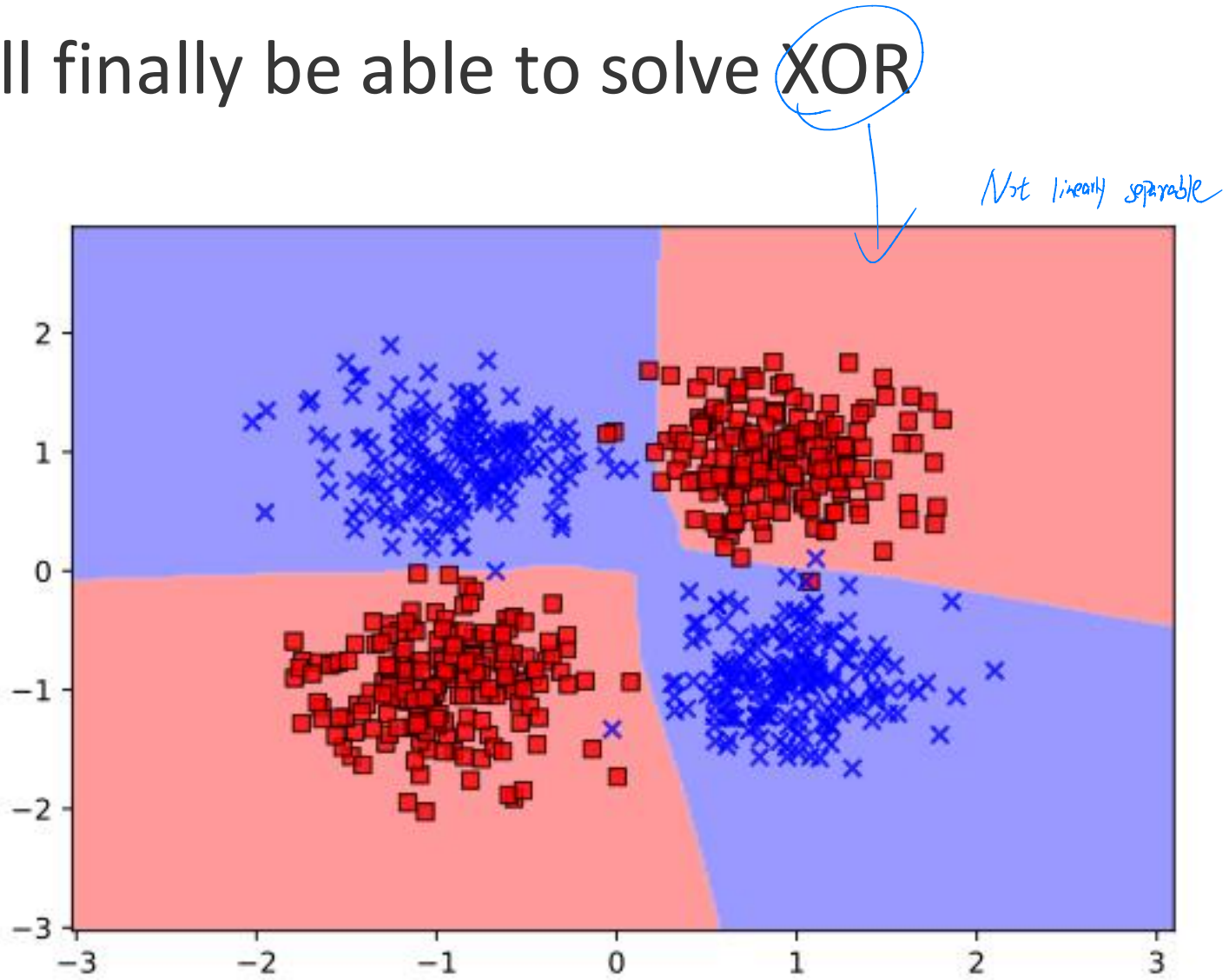
Last Time: Our old friend logistic regression…

1. A better loss function for classification (cross entropy instead of MSE)
2. Extending neurons to multi-classification (multiple output nodes + softmax)





Today: We will finally be able to solve XOR

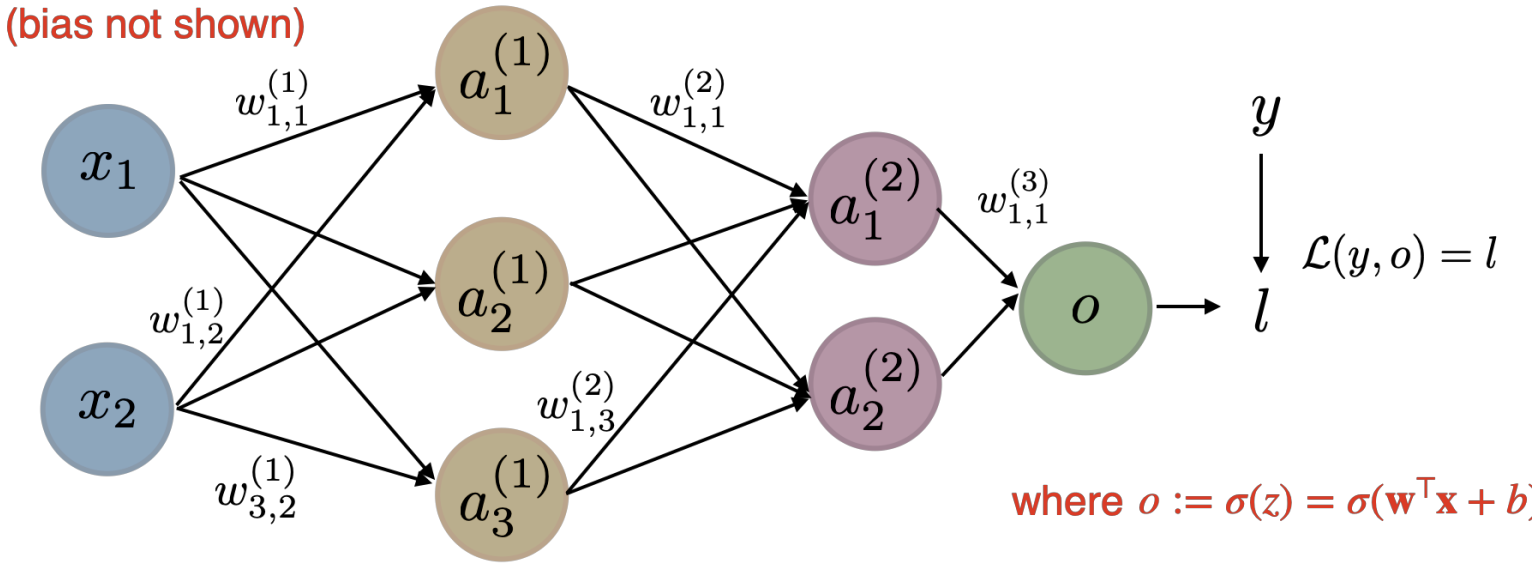


Today: Multilayer Perceptrons & Backpropagation

1. Multilayer Perceptron Architecture
2. Nonlinear Activation Functions
3. Multilayer Perceptron Code Examples
4. Overfitting and Underfitting (intro)
5. Cats & Dogs and Custom Data Loaders

Multilayer Perceptron

- Computation Graph with Multiple Fully-Connected Layers



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

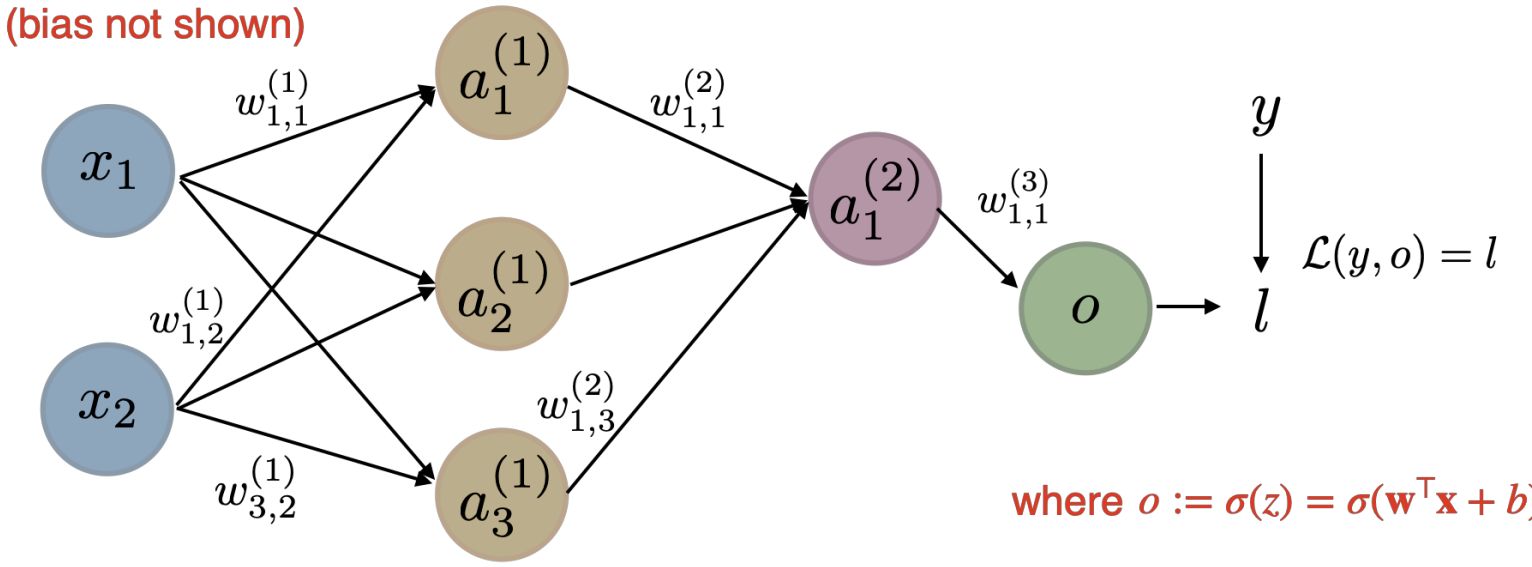
(Assume network for binary classification)

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

Nothing new, really

Multilayer Perceptron

- Computation Graph with Multiple Fully-Connected Layers



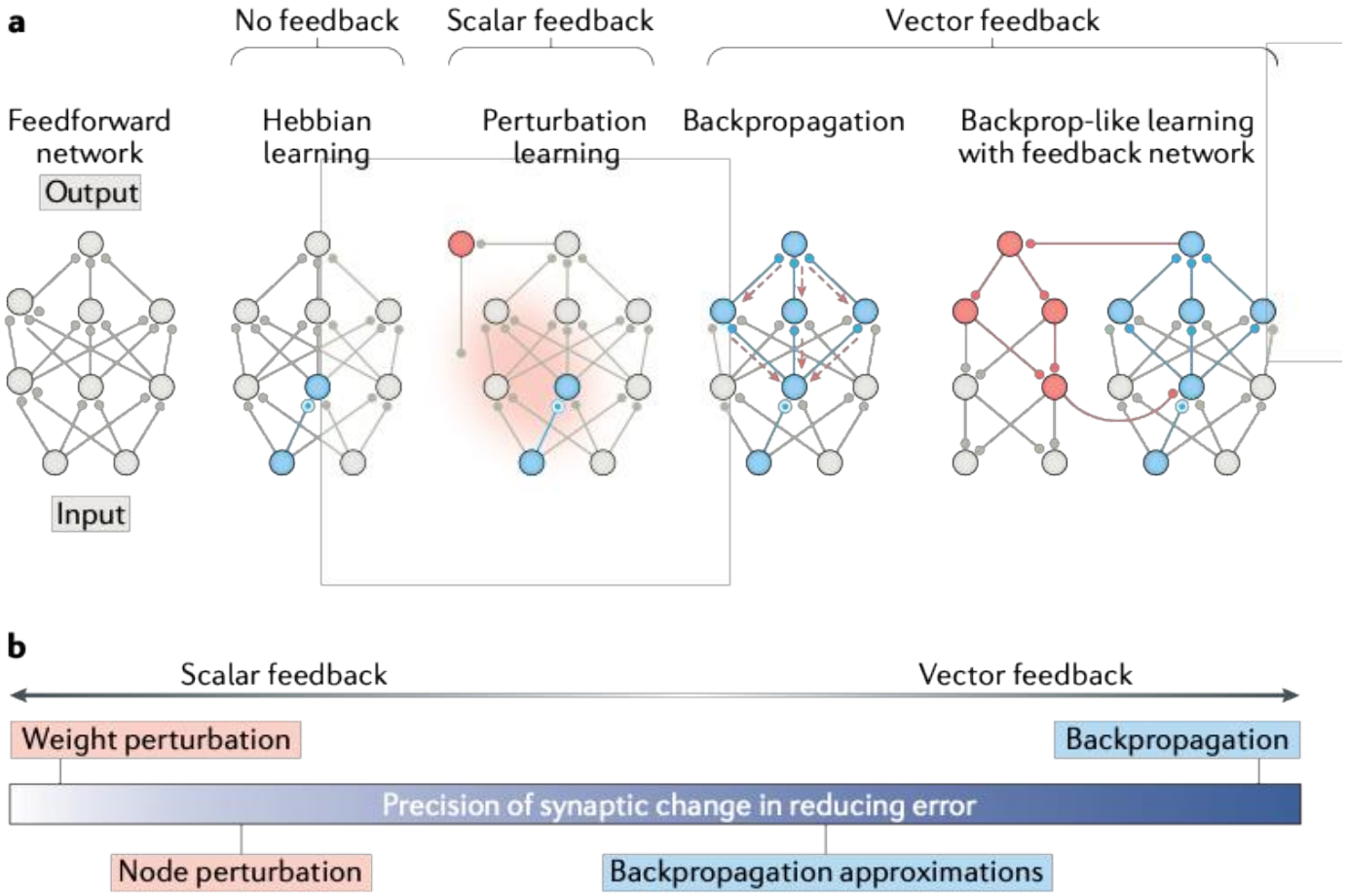
where $o := \sigma(z) = \sigma(\mathbf{w}^\top \mathbf{x} + b)$

(Assume network for binary classification)

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

Nothing new, really

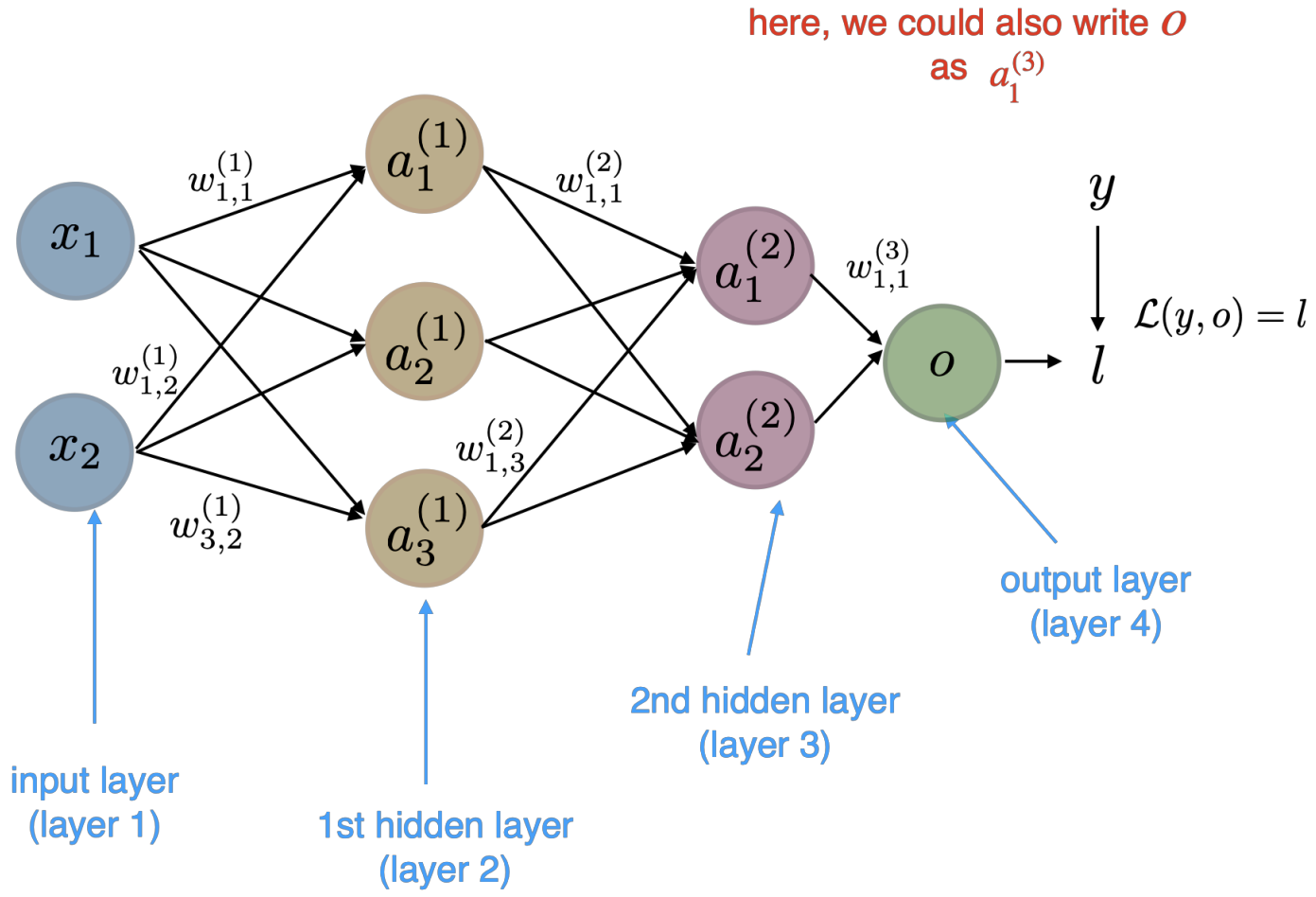
Progress over many decades



Lillicrap, T. P., Santoro, A., Marris, L., & Akerman, C. (n.d.). Backpropagation and the brain. *Nature Reviews Neuroscience*, 1–12. https://doi.org/10.1038/s41583-020-0277-3

Multilayer Perceptron

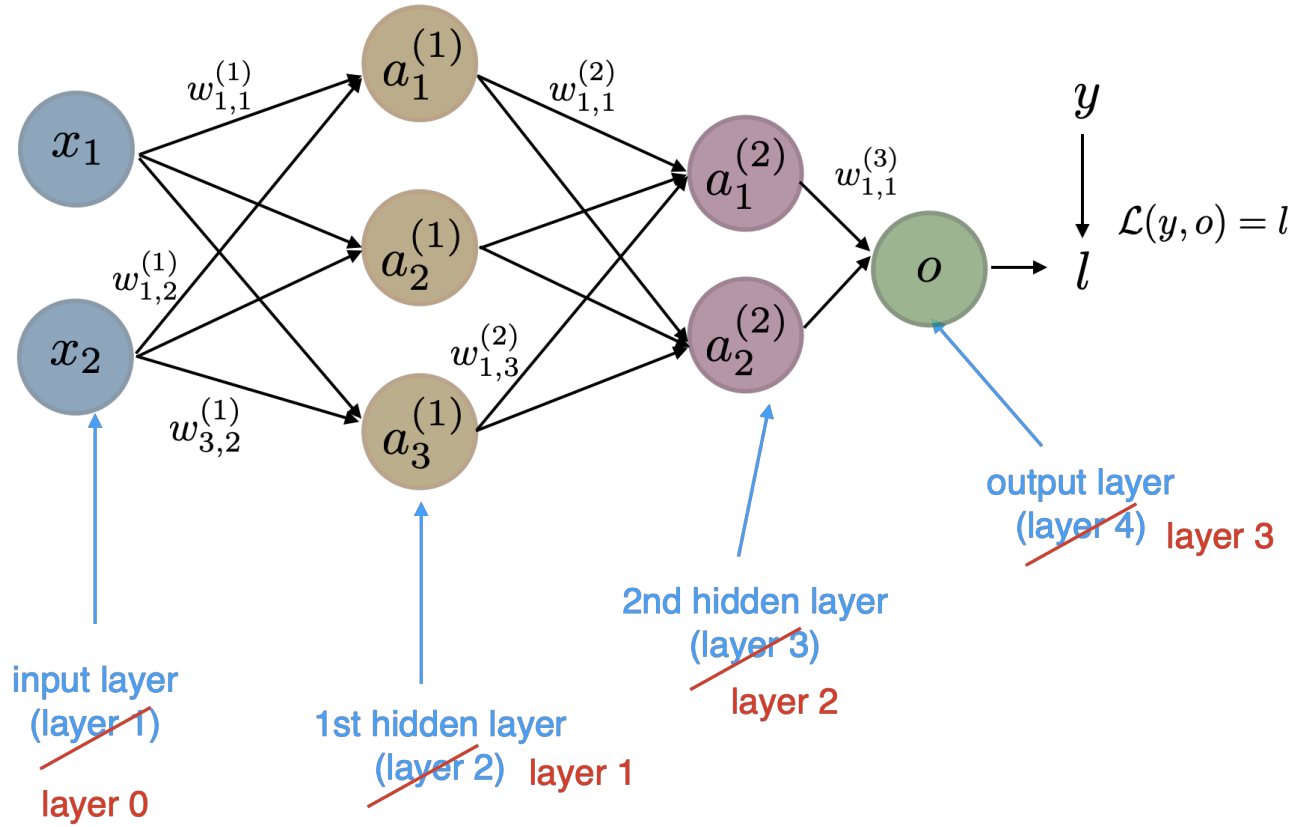
- Computation Graph with Multiple Fully-Connected Layers



Multilayer Perceptron

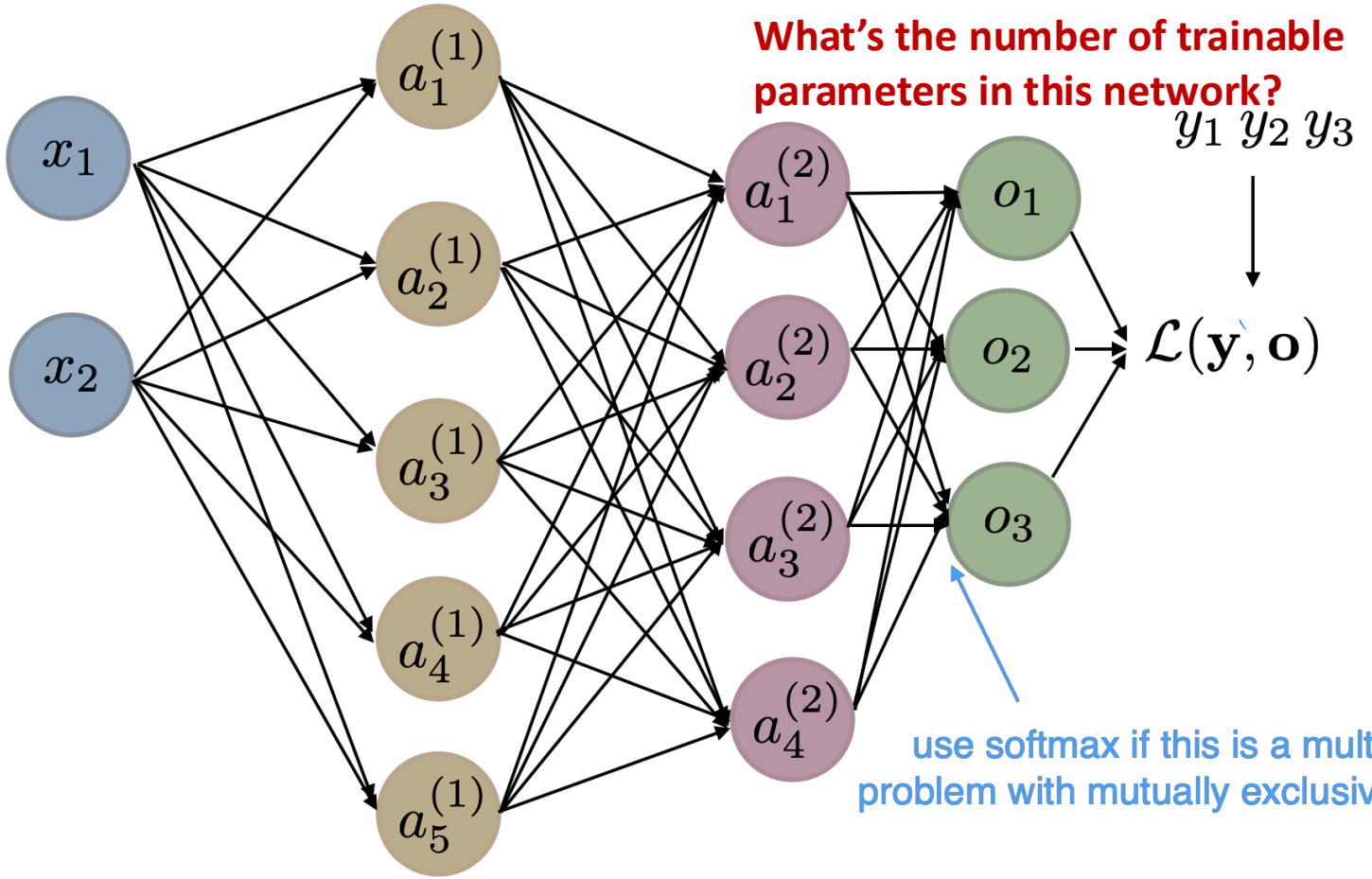
- Computation Graph with Multiple Fully-Connected Layers

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



Multilayer Perceptron

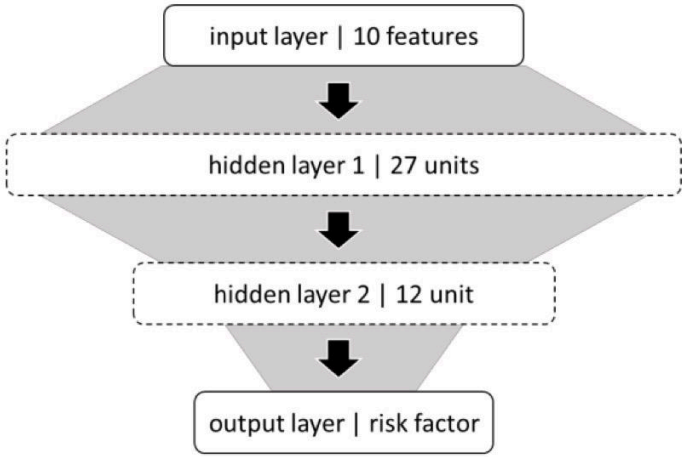
- Computation Graph with Multiple Fully-Connected Layers



$2\times5 + 4\times5 + 4\times3$

$= 42$

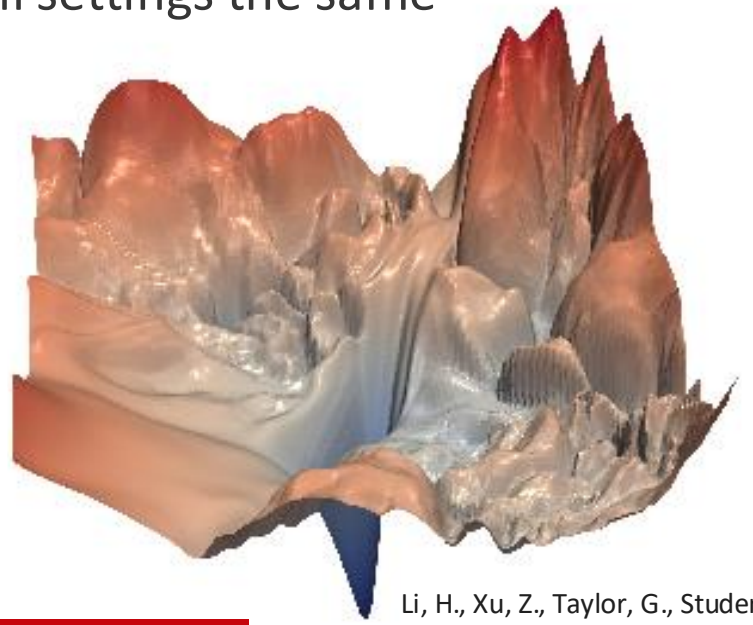
“Deep” Learning



Supplementary Figure 2. Illustration of the feed forward neural network architecture of the proposed deep survival model.

Note that our Loss is Not Convex Anymore

- Linear regression, Adaline, Logistic Regression, and Softmax Regression have convex loss functions
- But our deep loss is no longer convex (most of the time)
  - 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 settings the same



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).

Death Valley National Park

Formally

Convex function

$$\forall \lambda \in [0,1], \qquad f(\lambda \mathbf{x} + (1-\lambda)\boldsymbol{y}) \leq \lambda f(\mathbf{x}) + (1-\lambda) f(\boldsymbol{y})$$

Strictly convex function

$$\forall \lambda \in]0,1[, \qquad f(\lambda \mathbf{x} + (1-\lambda)\boldsymbol{y}) < \lambda f(\mathbf{x}) + (1-\lambda) f(\boldsymbol{y})$$

Strongly convex function

$$\exists \mu > 0, \text{ s.t.} \quad \mathbf{x} \mapsto f(\mathbf{x}) - \mu \|\mathbf{x}\|^2 \quad \text{is convex}$$

Equivalently:

$$\forall \lambda \in [0,1], \quad f(\lambda \mathbf{x} + (1-\lambda)\boldsymbol{y}) \leq \lambda f(\mathbf{x}) + (1-\lambda) f(\boldsymbol{y}) - \mu \lambda(1-\lambda)\|\mathbf{x}-\boldsymbol{y}\|^2$$

The largest possible μ is called the strong convexity constant.

Formal definition of convexity

Convex function

$$\forall \lambda \in [0,1], \qquad f(\lambda \mathbf{x} + (1-\lambda)\boldsymbol{y}) \leq \lambda f(\mathbf{x}) + (1-\lambda) f(\boldsymbol{y})$$

Strictly convex function

$$\forall \lambda \in]0,1[, \qquad f(\lambda \mathbf{x} + (1-\lambda)\boldsymbol{y}) < \lambda f(\mathbf{x}) + (1-\lambda) f(\boldsymbol{y})$$

Strongly convex function

$$\exists \mu > 0, \text{ s.t.} \quad \mathbf{x} \mapsto f(\mathbf{x}) - \mu \|\mathbf{x}\|^2 \quad \text{is convex}$$

Equivalently:

$$\forall \lambda \in [0,1], \quad f(\lambda \mathbf{x} + (1-\lambda)\boldsymbol{y}) \leq \lambda f(\mathbf{x}) + (1-\lambda) f(\boldsymbol{y}) - \mu \lambda(1-\lambda)\|\mathbf{x}-\boldsymbol{y}\|^2$$

The largest possible μ is called the strong convexity constant.

Formal reason why we like convexity

If f is convex and differentiable at $\mathbf{x}$ then

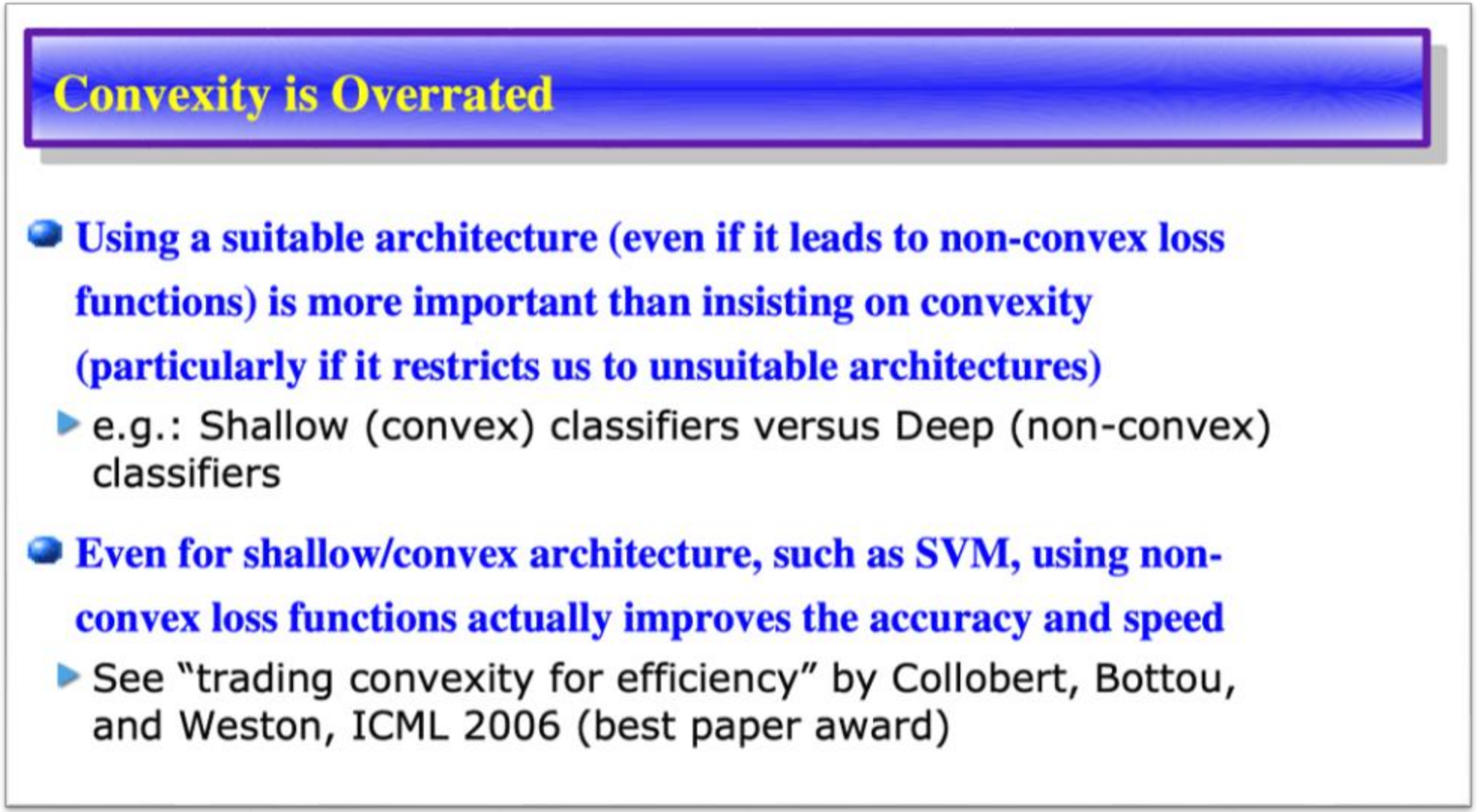
$$f(\boldsymbol{y}) \geq f(\mathbf{x}) + \nabla f(\mathbf{x})^\top (\boldsymbol{y} - \mathbf{x})$$

Convex function
All local minima are global minima.

Strictly convex function
If there is a local minimum, then it is unique and global.

Strongly convex function
There exists a unique local minimum which is also global.

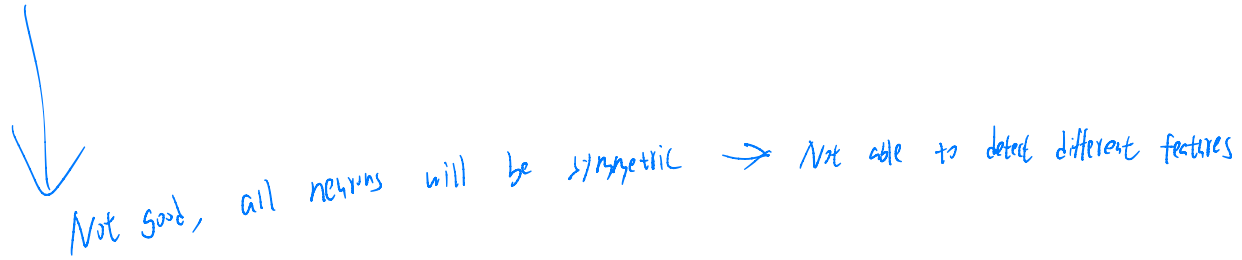
But…



- Yann LeCun, "Who's afraid of Non-convex loss functions?" – **2007**

Hmm…

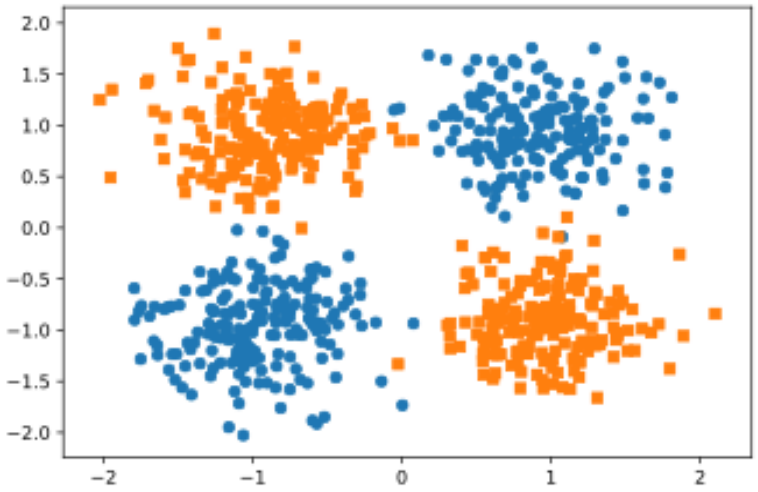
What happens if we initialize the multilayer perceptron to all-zero weights?



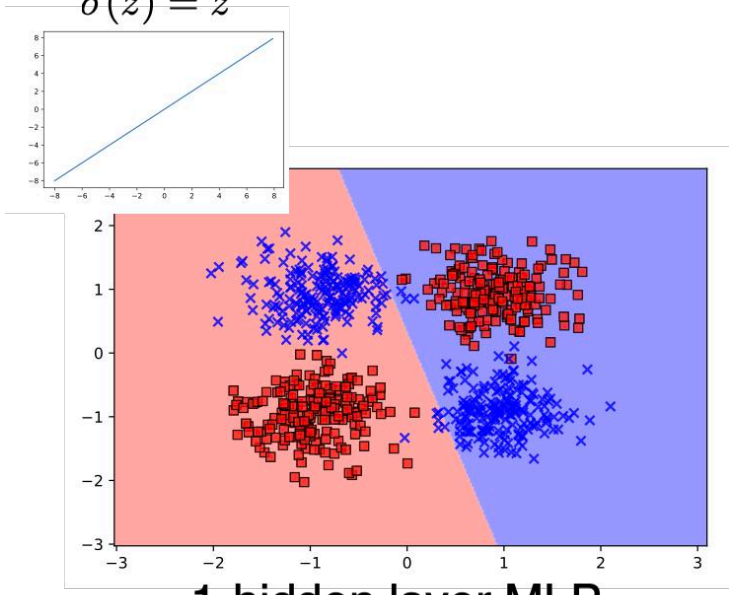
Today: Multilayer Perceptrons & Backpropagation

1. Multilayer Perceptron Architecture
2. **Nonlinear Activation Functions**
3. Multilayer Perceptron Code Examples
4. Overfitting and Underfitting (intro)
5. Cats & Dogs and Custom Data Loaders

Solving the XOR Problem with Non-Linear Activations

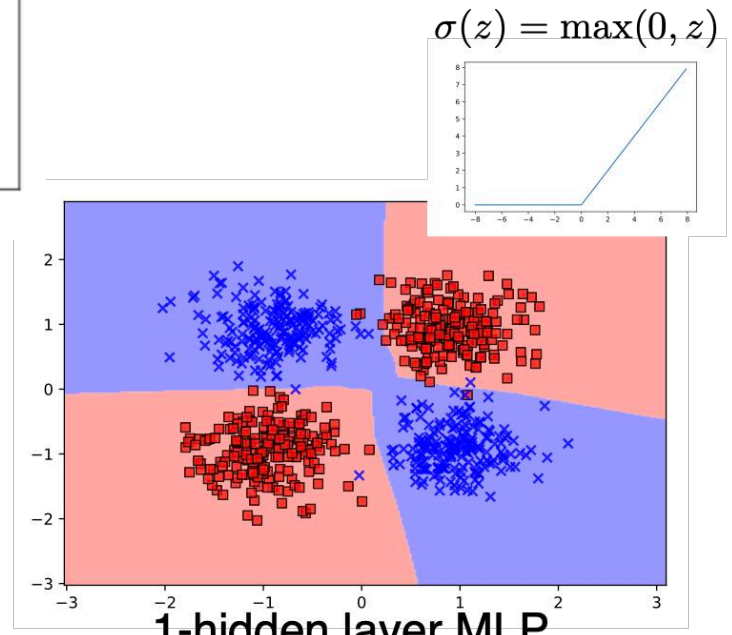


$\sigma(z) = z$



1-hidden layer MLP
with linear activation function

$\sigma(z) = \max(0, z)$

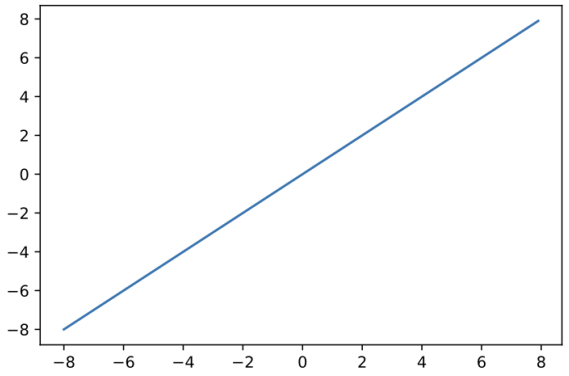


1-hidden layer MLP
with non-linear activation function (ReLU)

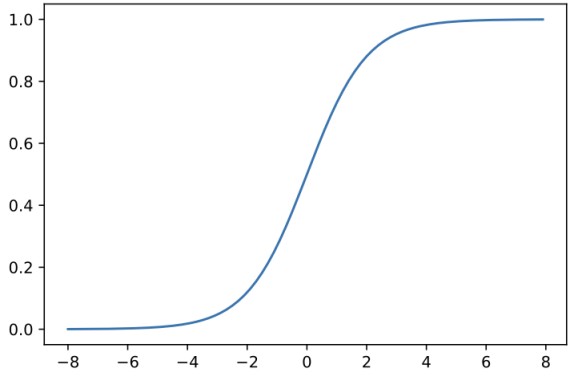
https://github.com/rasbt/stat453-deep-learning-ss21/blob/master/L09/code/xor-problem.ipynb

A Selection of Common Activation Functions

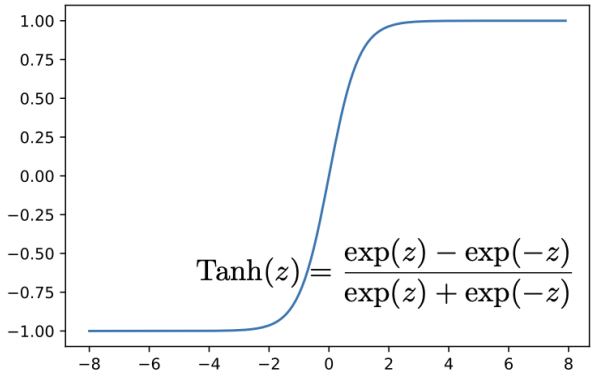
Identity



(Logistic) Sigmoid

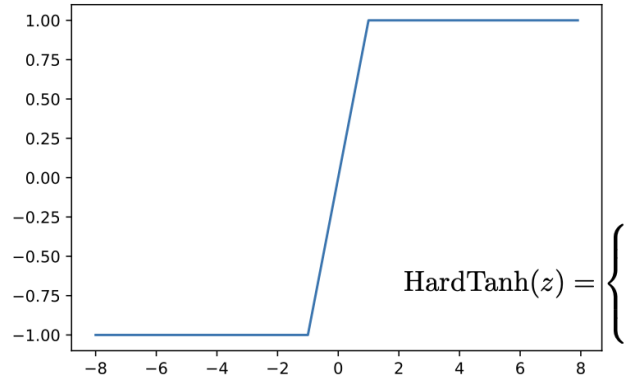


Tanh ("tanH")



$$\text{Tanh}(z) = \frac{\exp(z) - \exp(-z)}{\exp(z) + \exp(-z)}$$

Hard Tanh

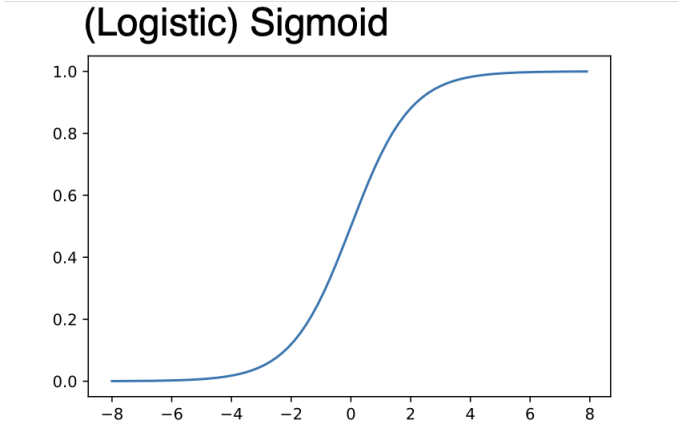


$$\text{HardTanh}(z) = \begin{cases} 1 & \text{if } z > 1 \\ -1 & \text{if } z < -1 \\ z & \text{otherwise} \end{cases}$$

A Selection of Common Activation Functions

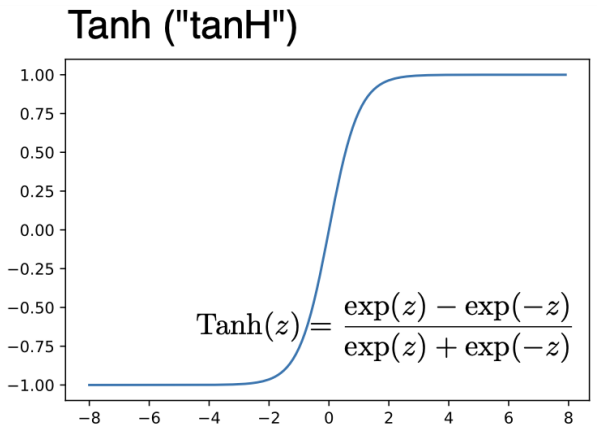
Advantages of Tanh

- Mean centering
- Positive and negative values
- Larger gradients



Also simple derivative:

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

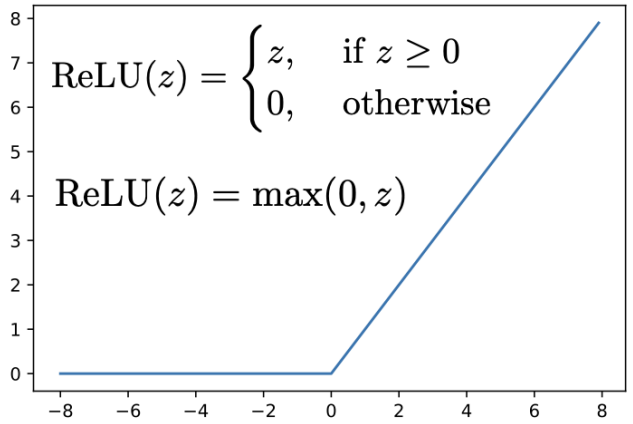


Important to normalize inputs to mean zero and use random weight initialization with **avg. weight centered at zero**

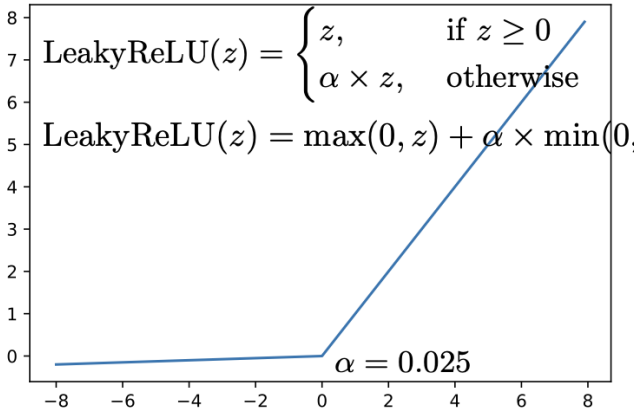
The scale of different features is similar → can be trained in the same learning rate and steadily step to the optimal point.

A Selection of Common Activation Functions (cont.)

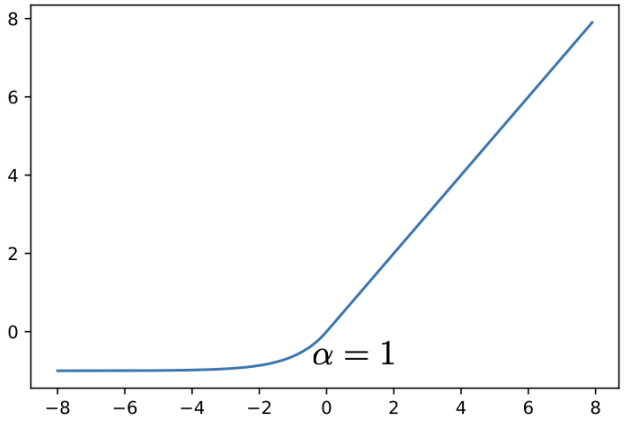
ReLU (Rectified Linear Unit)



Leaky ReLU



ELU (Exponential Linear Unit)



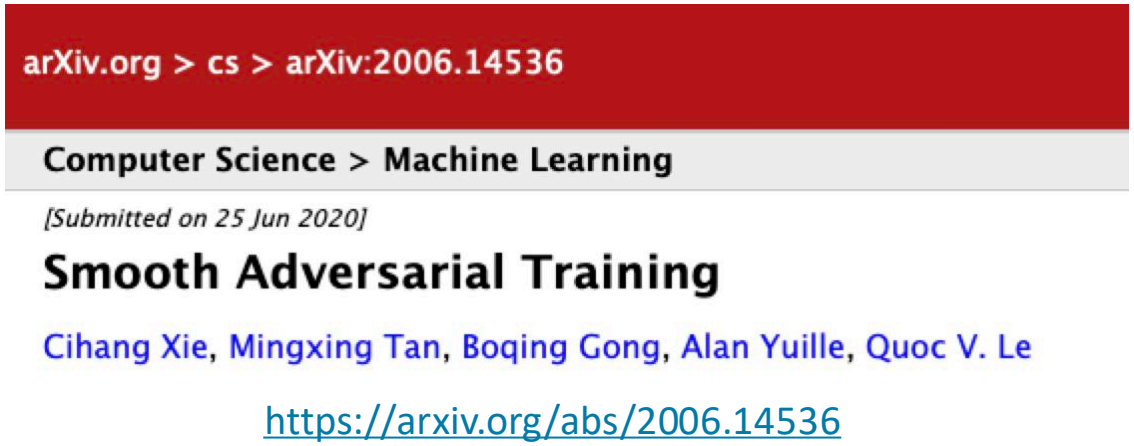
PReLU (Parameterized Rectified Linear Unit)

here, alpha is a trainable parameter

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

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

Activation Functions Influence Robustness?



https://arxiv.org/abs/2006.14536

It is commonly believed that networks cannot be both accurate and robust, that gaining robustness means losing accuracy. [...] Our key observation is that the widely-used ReLU activation function significantly weakens adversarial training due to its non-smooth nature. Hence we propose smooth adversarial training (SAT), in which we replace ReLU with its smooth approximations to strengthen adversarial training.

Activation Functions Influence Robustness?

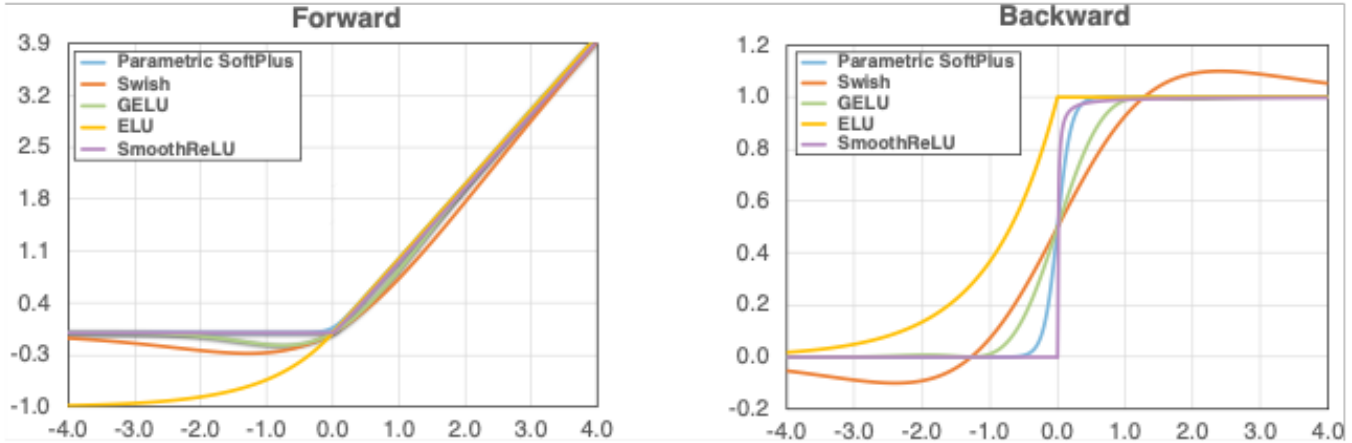


Figure 2: Visualizations of smooth activation functions and their derivatives.

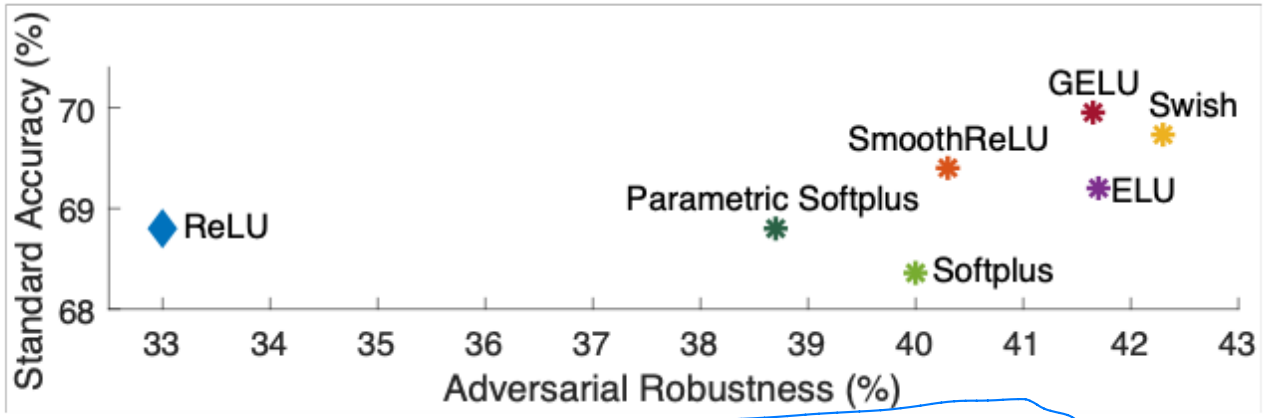
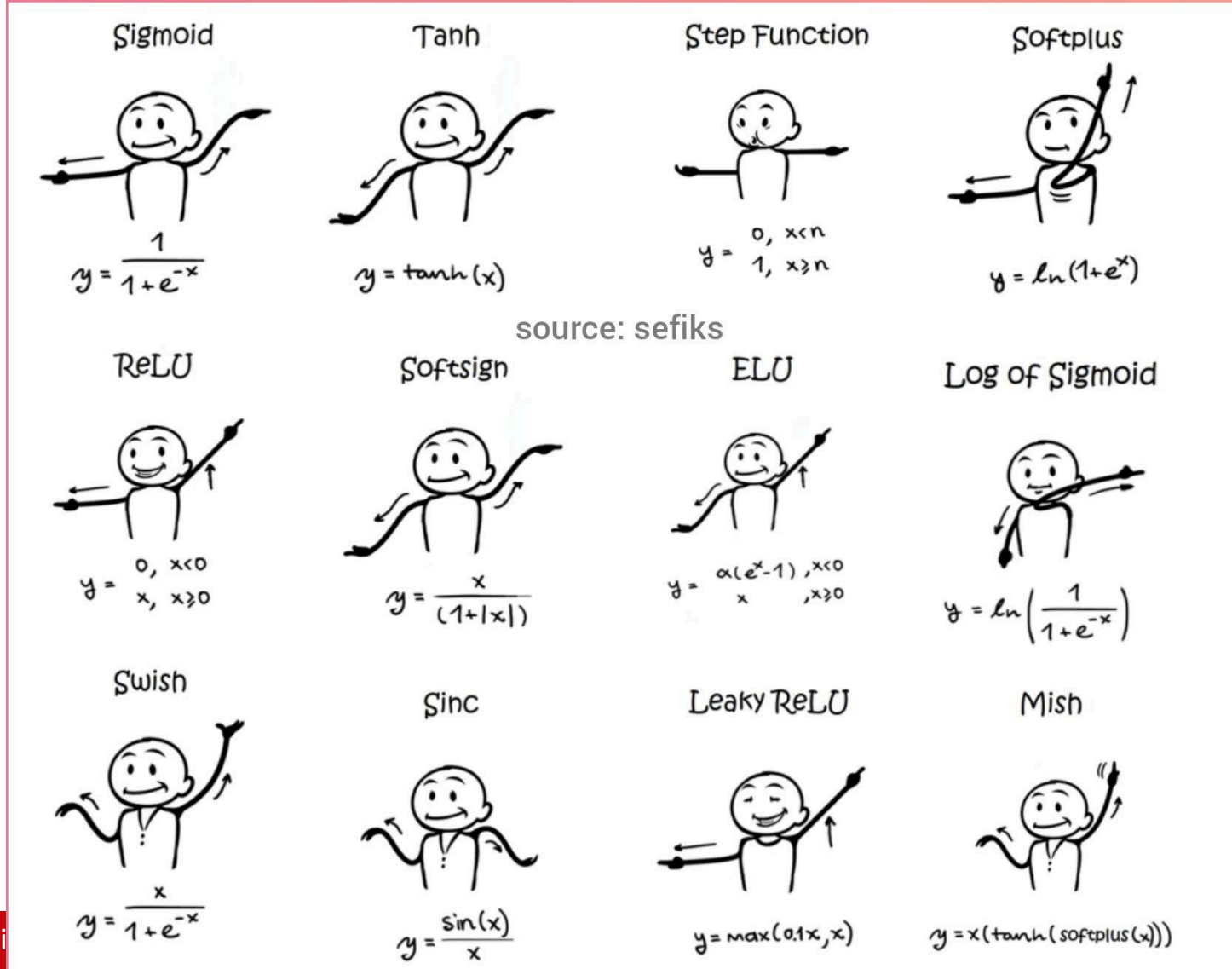


Figure 3: Smooth activation functions improve adversarial training. Compared to ReLU, all smooth activation functions significantly boost robustness, while keeping accuracy almost the same.

Dance Moves of Deep Learning Activation Functions



https://twitter.com/TheInsaneApp/status/1366324846976659461?s=20

Today: Multilayer Perceptrons & Backpropagation

1. Multilayer Perceptron Architecture
2. Nonlinear Activation Functions
3. **Multilayer Perceptron Code Examples**
4. Overfitting and Underfitting (intro)
5. Cats & Dogs and Custom Data Loaders

MLP Code Examples

Multilayer Perceptron with Sigmoid Activation and MSE Loss

https://github.com/rasbt/stat453-deep-learning-ss20/blob/master/L08-mlp/code/mlp-pytorch_sigmoid-mse.ipynb

Multilayer Perceptron with Softmax Activation and Cross-Entropy Loss

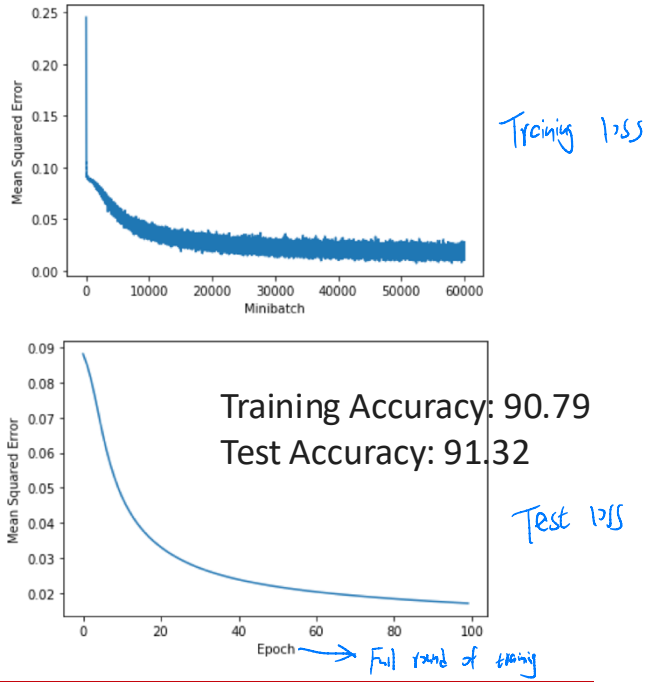
https://github.com/rasbt/stat453-deep-learning-ss20/blob/master/L08-mlp/code/mlp-pytorch_sigmoid-crossentr.ipynb

- Good practice:
  - Set random seed for reproducibility
  - Check if datasets, input data are as expected before training
  - Set timer, generator log, and print intermediate values.
  - During training, if the loss behaves strangely (e.g. going up prematurely), stop and start diagnosis/debugging
  - Change training hyperparameters and repeat experiments

MLP Code Examples

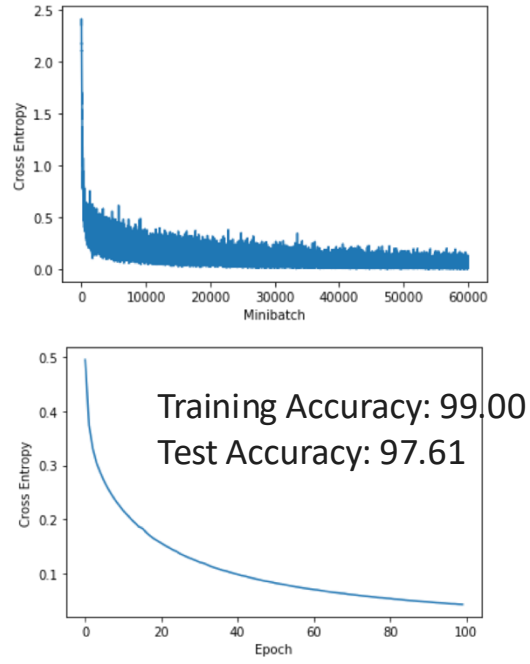
Multilayer Perceptron with Sigmoid Activation and MSE Loss

https://github.com/rasbt/stat453-deep-learning-ss20/blob/master/L08-mlp/code/mlp-pytorch_sigmoid-mse.ipynb



Multilayer Perceptron with Softmax Activation and Cross-Entropy Loss

https://github.com/rasbt/stat453-deep-learning-ss20/blob/master/L08-mlp/code/mlp-pytorch_sigmoid-crossentr.ipynb



Diagnosing loss curves

- Checking the train/test loss/error is one simple diagnosis.
  - Tools such as Weights & Biases (https://wandb.ai) are helpful
- Test/val. error plateau or increases may suggest overfitting
  - Regularization / early stopping is needed (we will discuss next time)
- Train loss may be smoothed over mini-batches
- Some models such as Transformers may be more difficult to train, sharp transition followed by long plateau, e.g. “grokking” (https://arxiv.org/abs/2201.02177)
- Learning rate (LR), LR scheduler, initialization all effect training curves. Lots of trial and error.
  - “Grad student descent”

Parameters vs Hyperparameters

weights (weight parameters)
biases (bias units)

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)
…

Wide vs Deep Architectures

- MLPs with one (large) hidden layer are universal functions approximators [1-4]
- So why do we want to use deeper architectures?

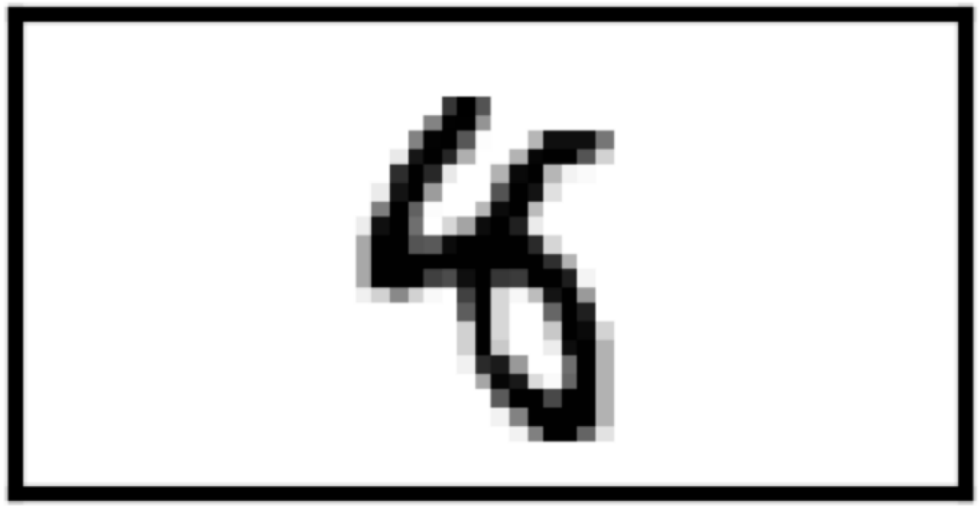
[1] Balázs Csanád Csáji (2001) Approximation with Artificial Neural Networks; Faculty of Sciences; Eötvös Loránd University, Hungary
[2] Barron, Andrew R. "Universal approximation bounds for superpositions of a sigmoidal function." IEEE Transactions on Information theory 39.3 (1993): 930-945.
[3] Cybenko, G. (1989) "Approximations by superpositions of sigmoidal functions", Mathematics of Control, Signals, and Systems, 2(4), 303–314. doi:10.1007/BF02551274
[4] Hornik, K., Stinchcombe, M., & White, H. (1989). Multilayer feedforward networks are universal approximators. Neural networks, 2(5), 359-366.

Wide vs Deep Architectures

- MLPs with one (large) hidden layer are universal functions approximators [1-4]
- So why do we want to use deeper architectures?

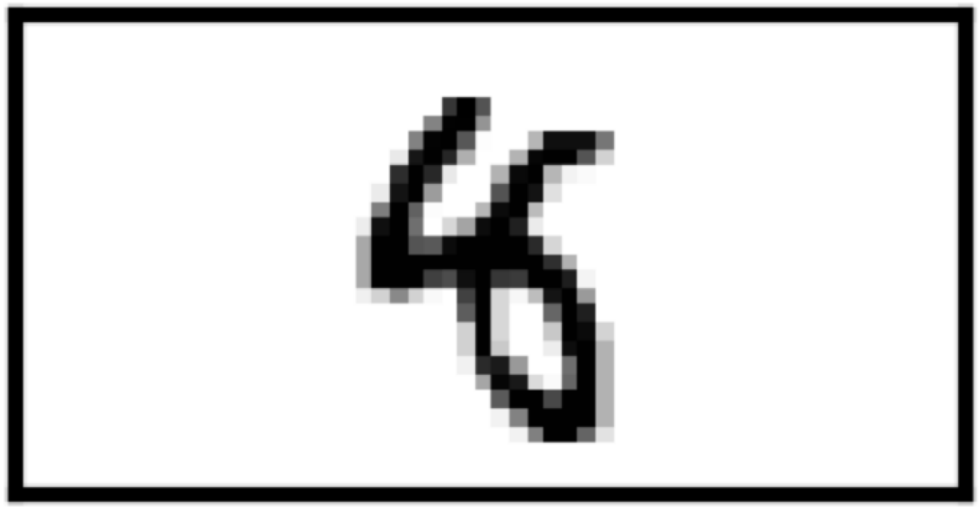
  - Can achieve the same expressiveness with more layers but fewer parameters (combinatorics); fewer parameters => less overfitting?, faster computation
  - Also, having more layers provides some form of regularization: later layers are constrained on the behavior of earlier layers
  - However, more layers => vanishing/exploding gradients
  - Later: different layers for different levels of feature abstraction (DL is really more about feature learning than just stacking multiple layers)
  - Inductive bias? [https://arxiv.org/abs/1608.08225]

When diagnosing: look at your actual data!

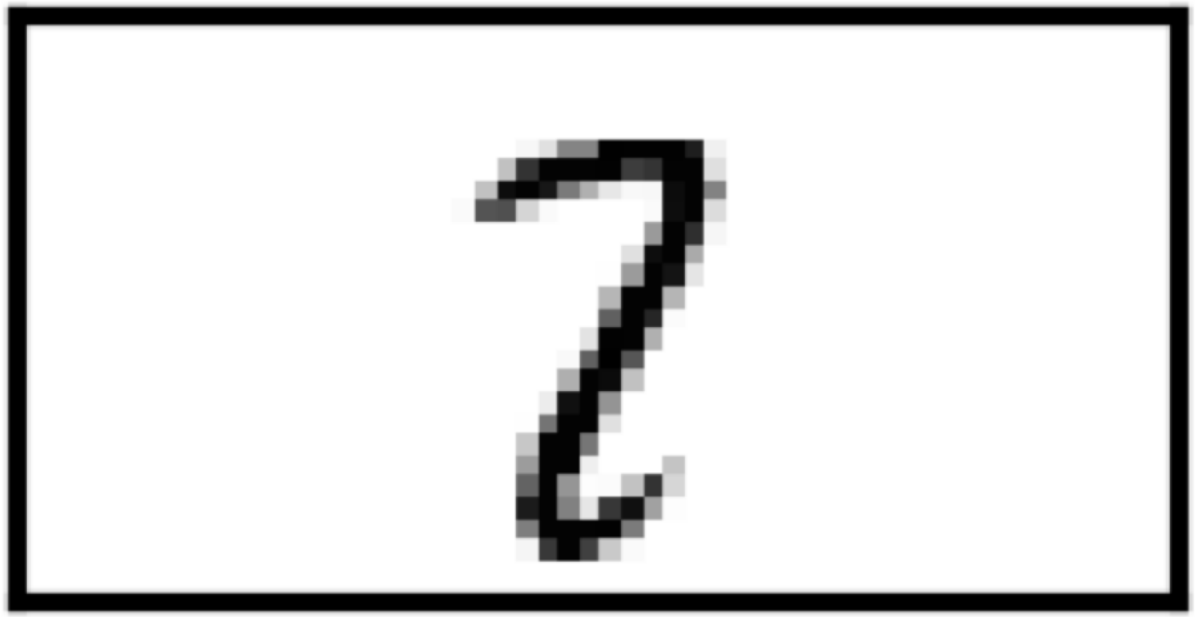


When diagnosing: look at your actual data!

6) t: 8 p: 4

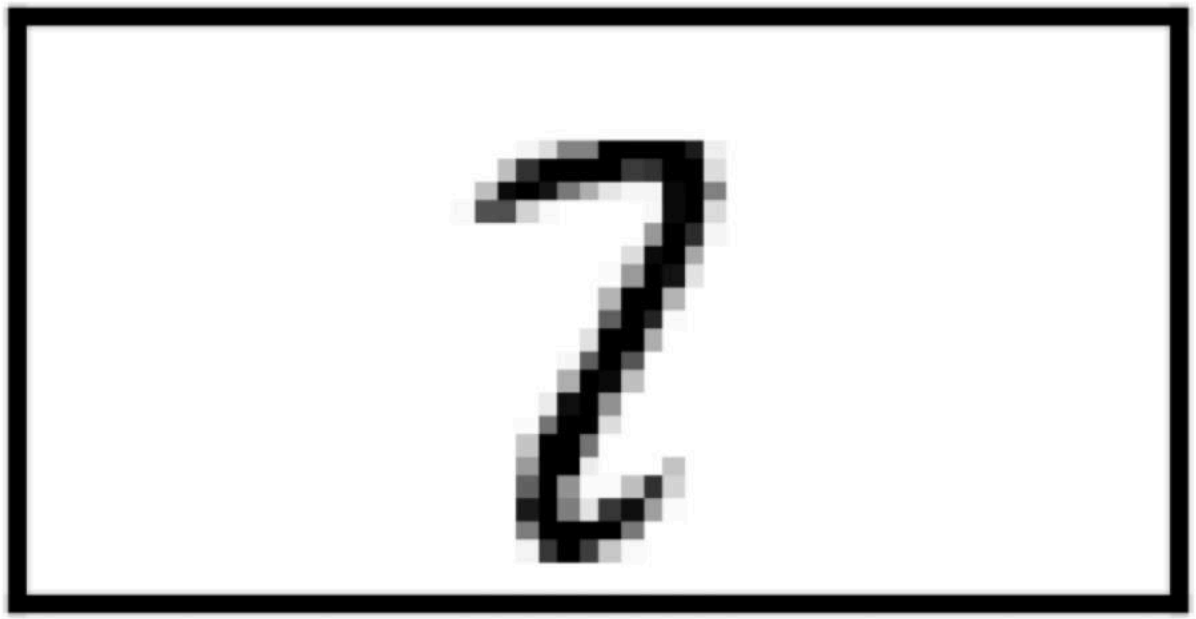


When diagnosing: look at your actual data!

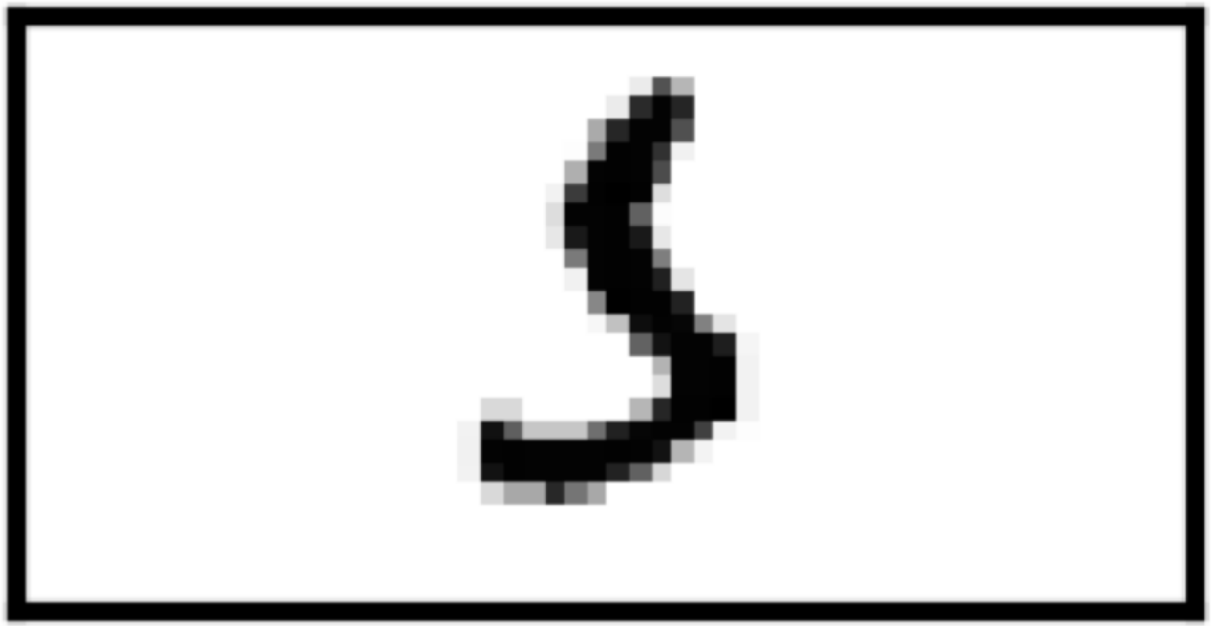


When diagnosing: look at your actual data!

8) t: 2 p: 7

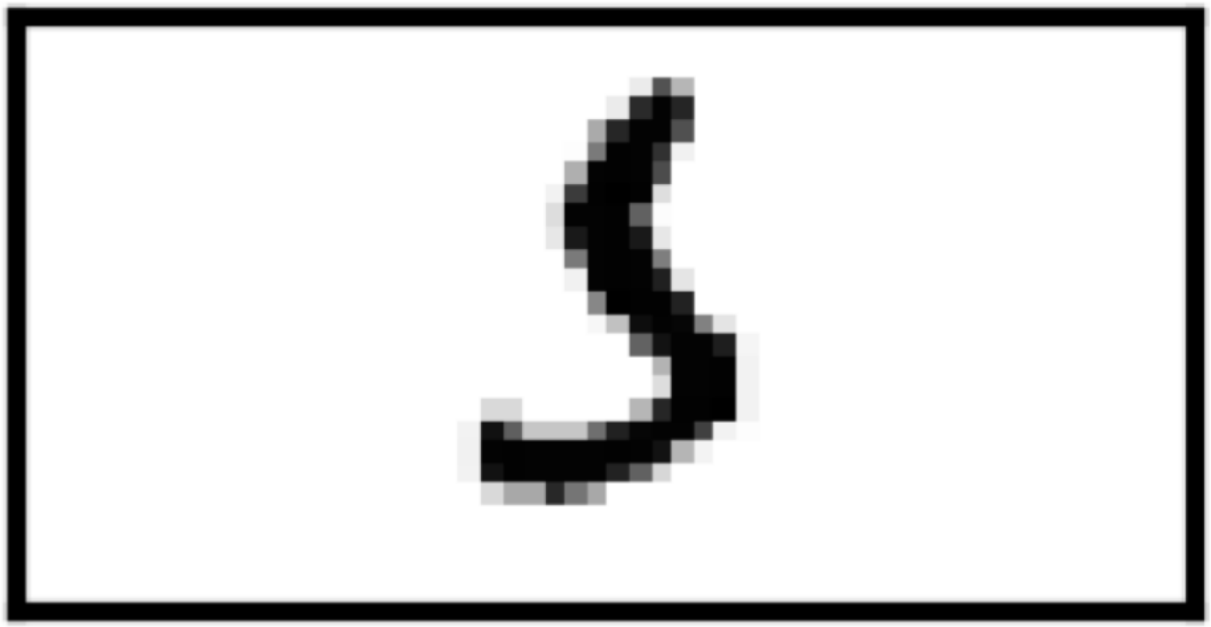


When diagnosing: look at your actual data!

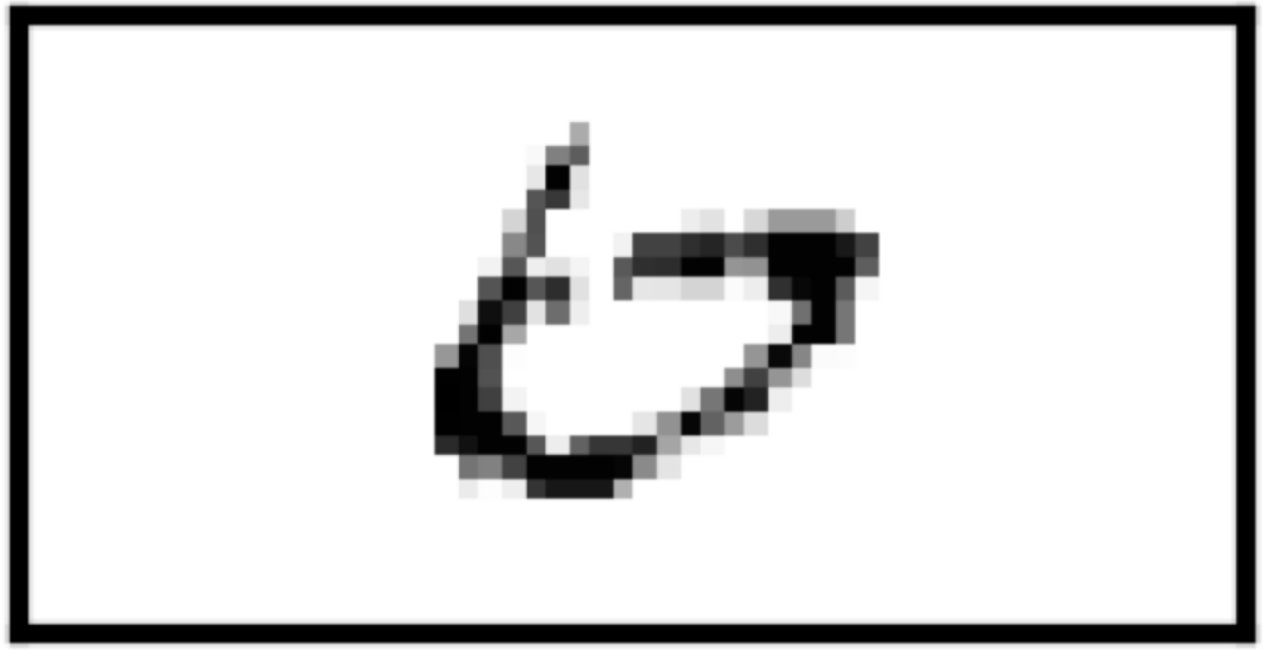


When diagnosing: look at your actual data!

10) t: 5 p: 3



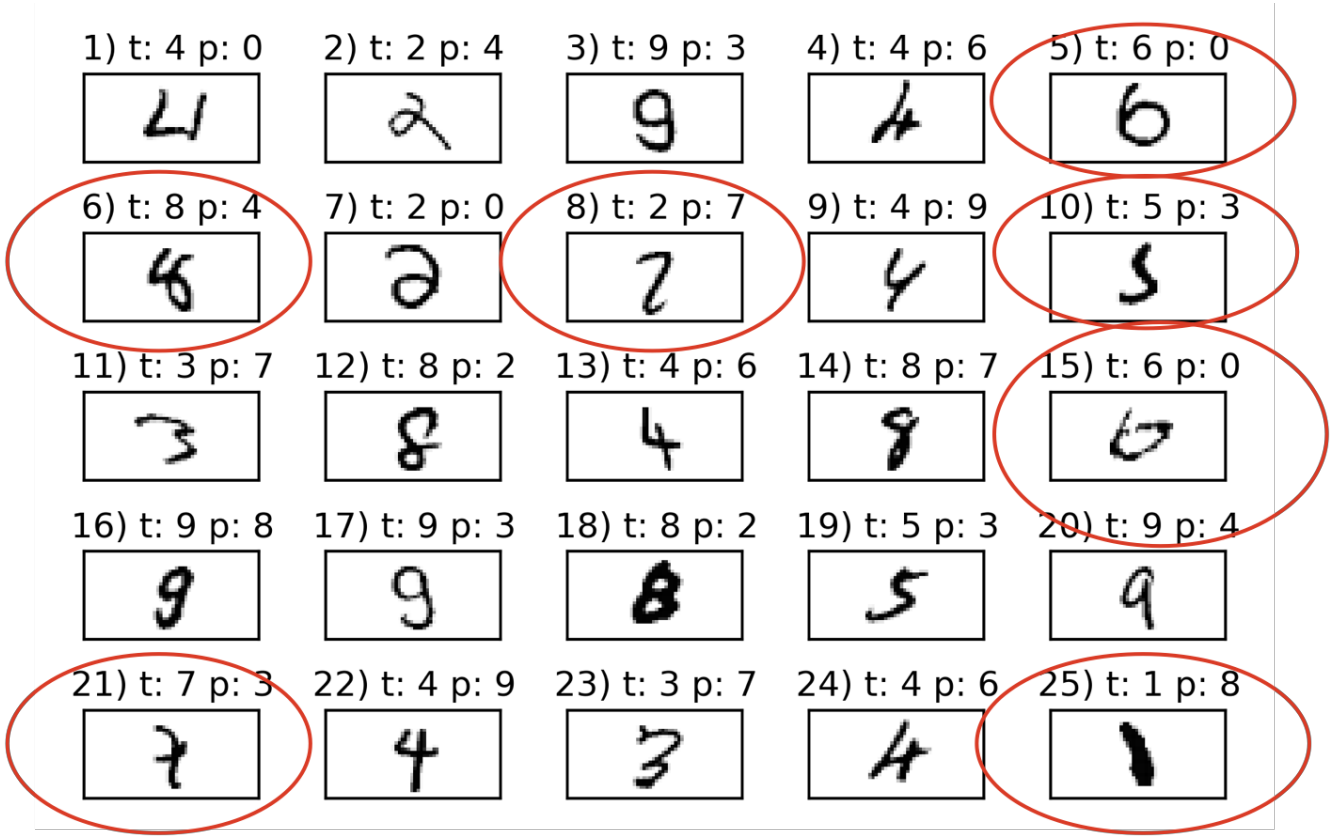
When diagnosing: look at your actual data!



When diagnosing: look at your actual data!

15) t: 6 p: 0

When diagnosing: look at your actual data!

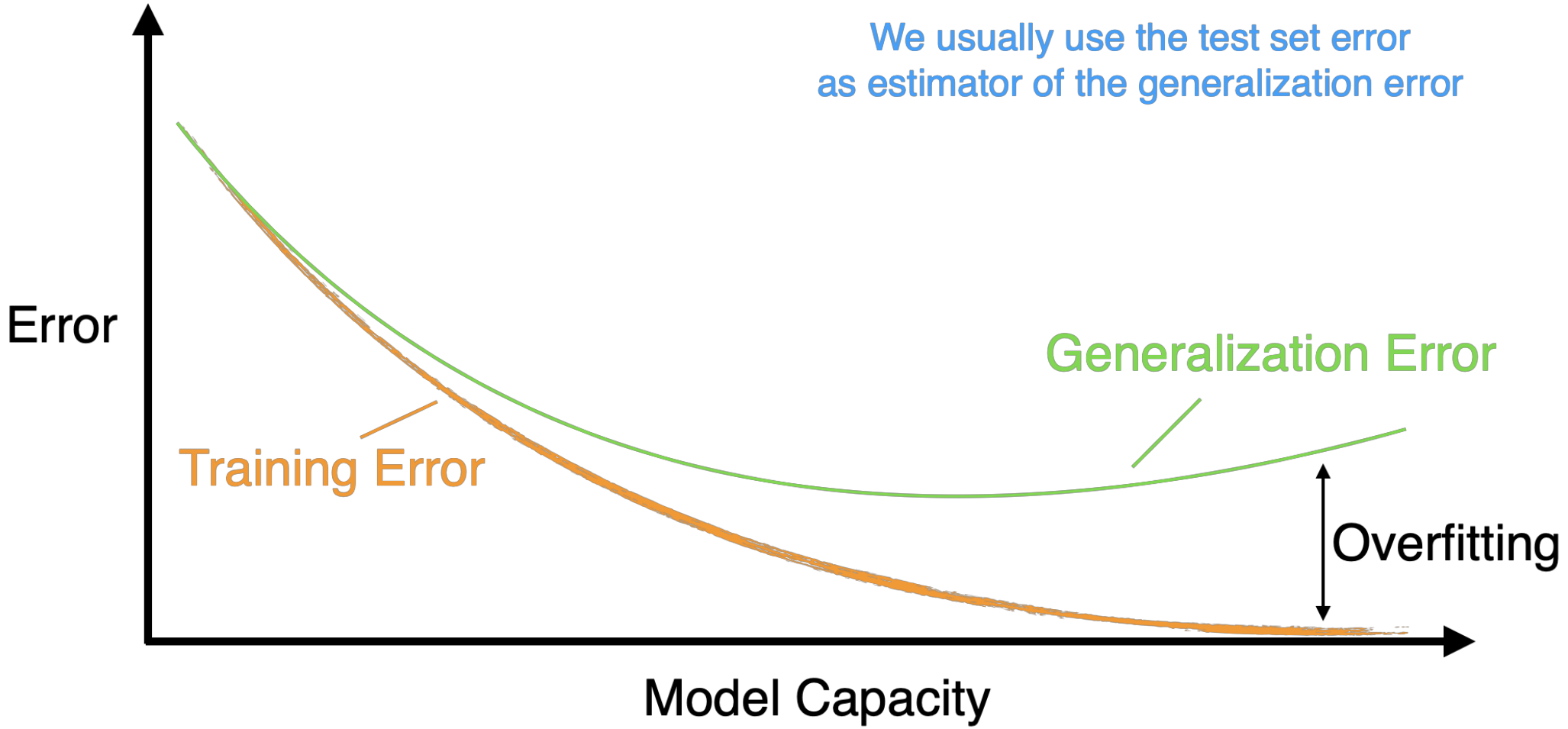


Failure cases of a ~93% accuracy (not very good, but beside the point) 2-layer (1-hidden layer) MLP on MNIST
(where t=target class and p=predicted class)

Today: Multilayer Perceptrons & Backpropagation

1. Multilayer Perceptron Architecture
2. Nonlinear Activation Functions
3. Multilayer Perceptron Code Examples
4. **Overfitting and Underfitting (intro)**
5. Cats & Dogs and Custom Data Loaders

Overfitting and Underfitting



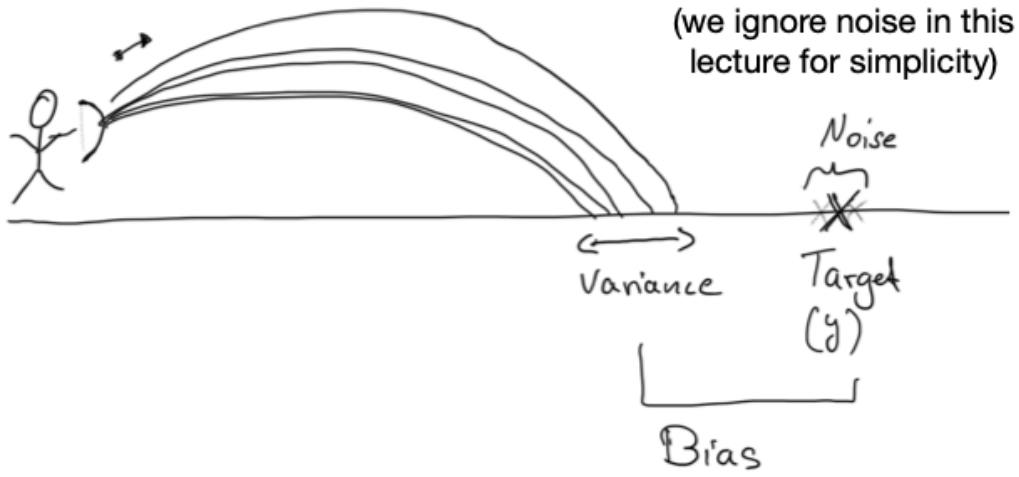
Bias-Variance Decomposition

General Definition:

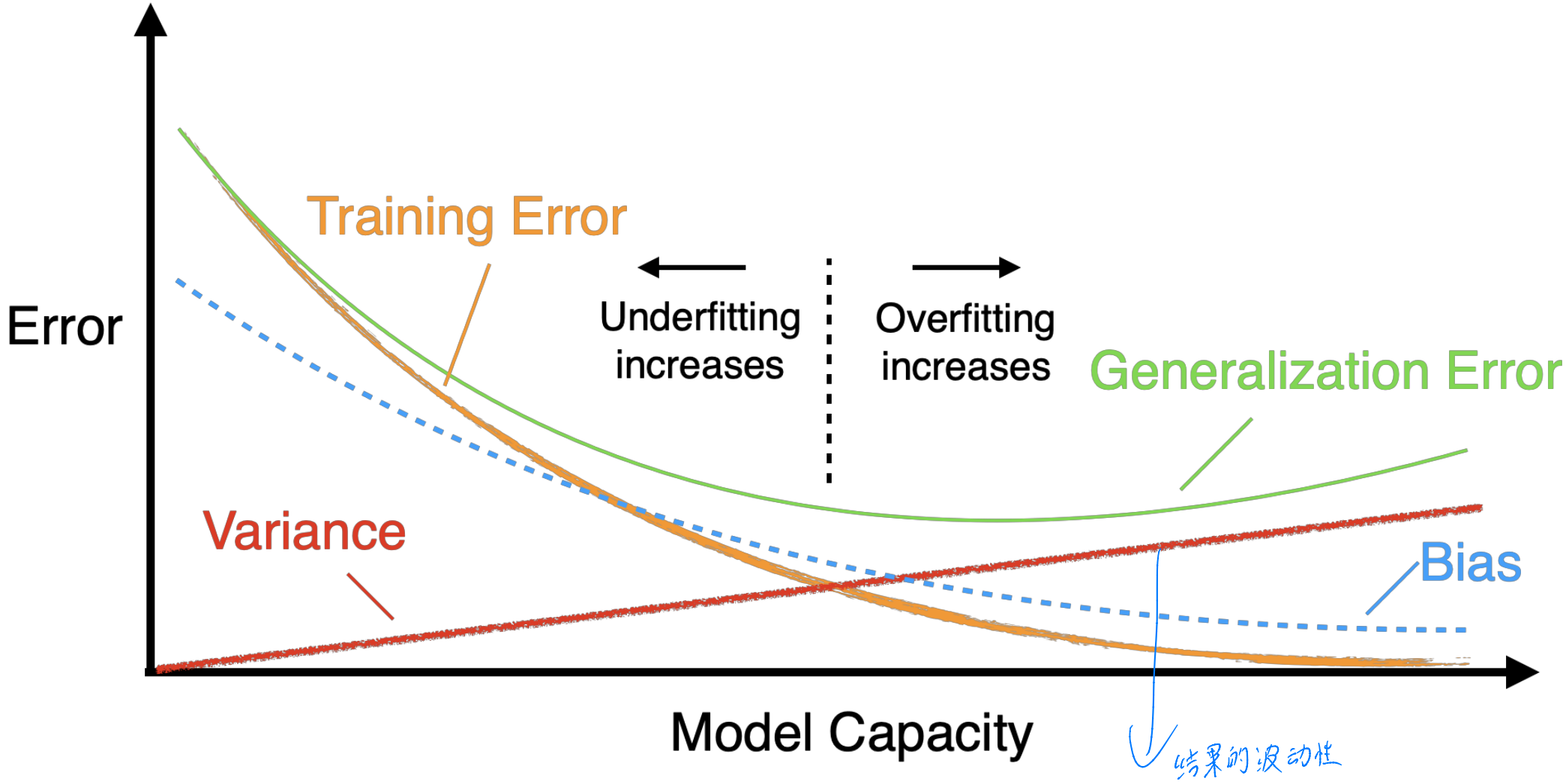
$$\text{Bias}_\theta[\hat{\theta}] = E[\hat{\theta}] - \theta$$

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

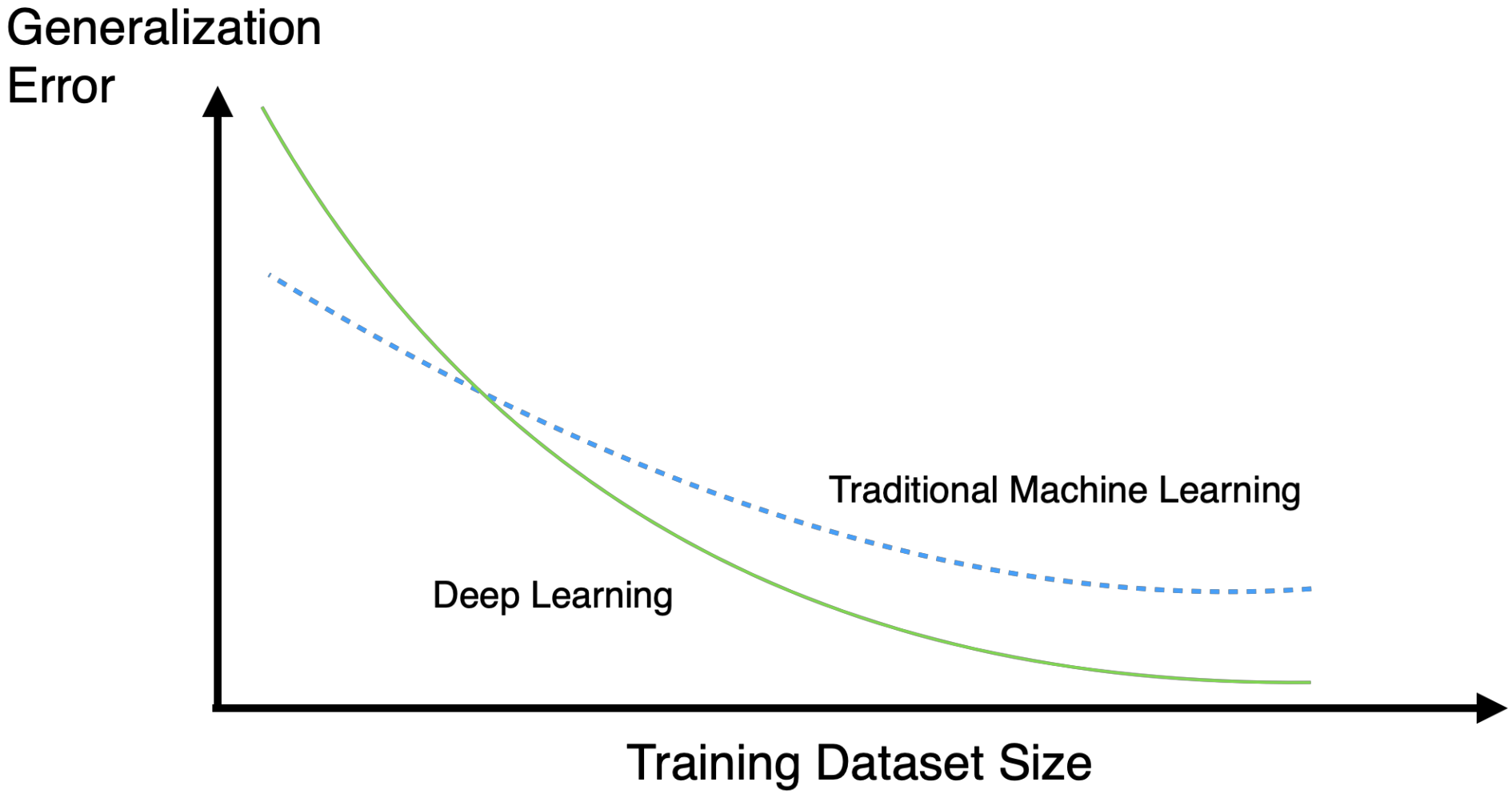
Intuition:



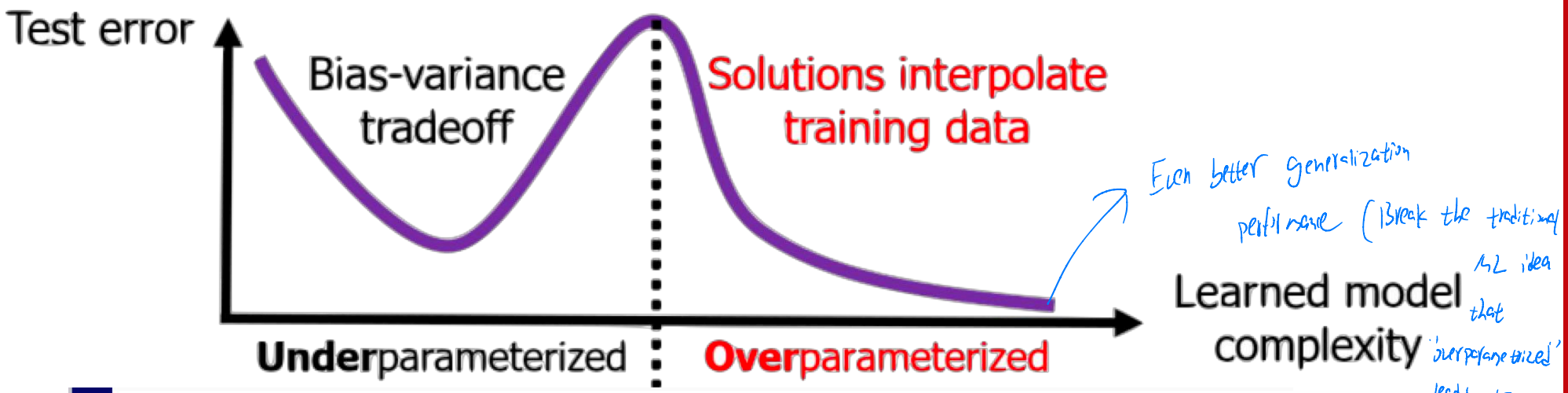
Bias-Variance & Overfitting-Underfitting



Deep Learning works best with large datasets



“Double descent” phenomena



PNAS

Reconciling modern machine-learning practice and the classical bias–variance trade-off

Mikhail Belkin[a,b,1], Daniel Hsu[c], Siyuan Ma[a], and Soumik Mandal[a]

[a]Department of Computer Science and Engineering, The Ohio State University, Columbus, OH 43210; [b]Department of Statistics, The Ohio State University, Columbus, OH 43210; and [c]Computer Science Department and Data Science Institute, Columbia University, New York, NY 10027

Edited by Peter J. Bickel, University of California, Berkeley, CA, and approved July 2, 2019 (received for review February 21, 2019)

Breakthroughs in machine learning are rapidly changing science and society, yet our fundamental understanding of this technology has lagged far behind. Indeed, one of the central tenets of the

ing data (i.e., have large empirical risk) and hence predict poorly on new data. 2) If $\mathcal{H}$ is too large, the empirical risk minimizer may overfit spurious patterns in the training data, resulting in

“Double descent” phenomena

arXiv.org > cs > arXiv:1912.02292

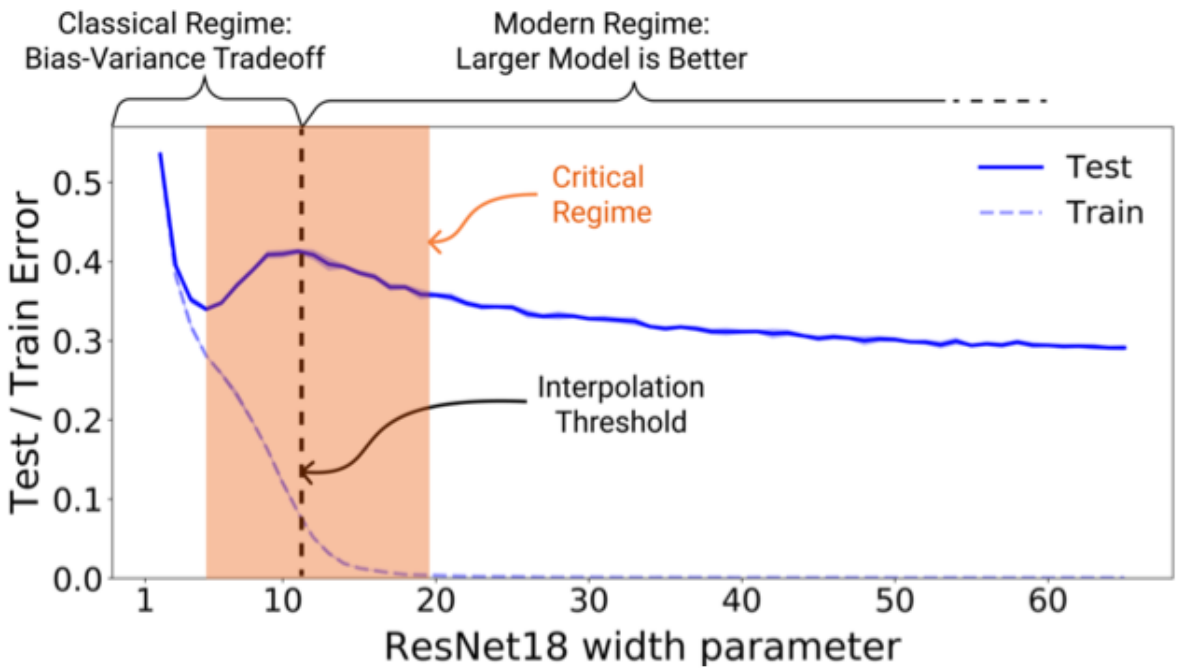
Computer Science > Machine Learning

[Submitted on 4 Dec 2019]

Deep Double Descent: Where Bigger Models and More Data Hurt

Preetum Nakkiran, Gal Kaplun, Yamini Bansal, Tristan Yang, Boaz Barak, Ilya Sutskever

https://arxiv.org/abs/1912.02292



“Double descent” phenomena

arXiv.org > cs > arXiv:1912.02292

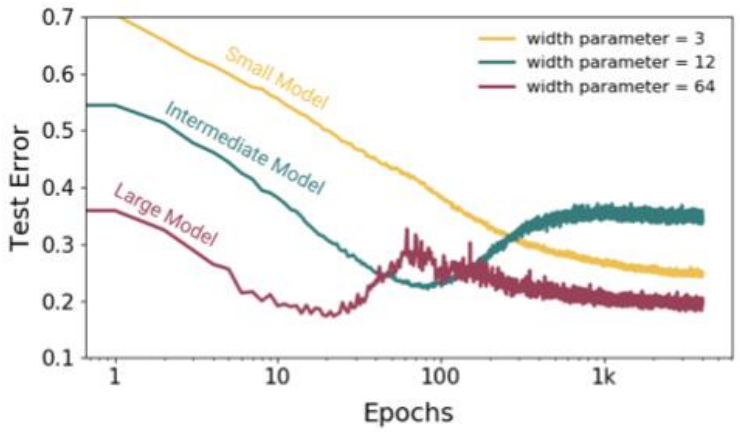
Computer Science > Machine Learning

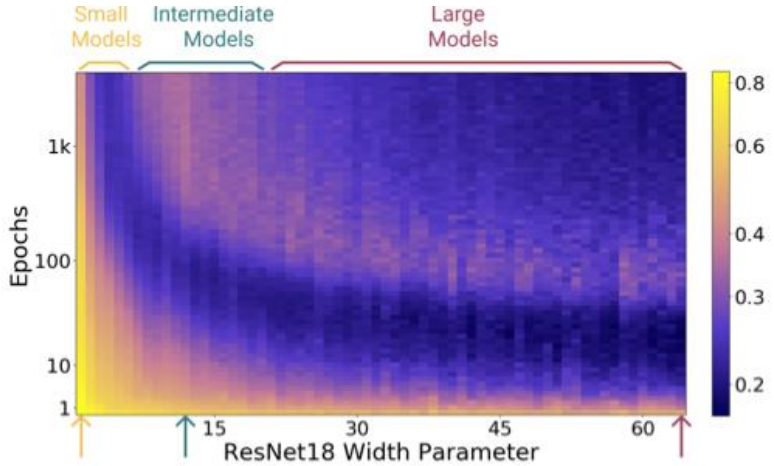
[Submitted on 4 Dec 2019]

Deep Double Descent: Where Bigger Models and More Data Hurt

Preetum Nakkiran, Gal Kaplun, Yamini Bansal, Tristan Yang, Boaz Barak, Ilya Sutskever

https://arxiv.org/abs/1912.02292

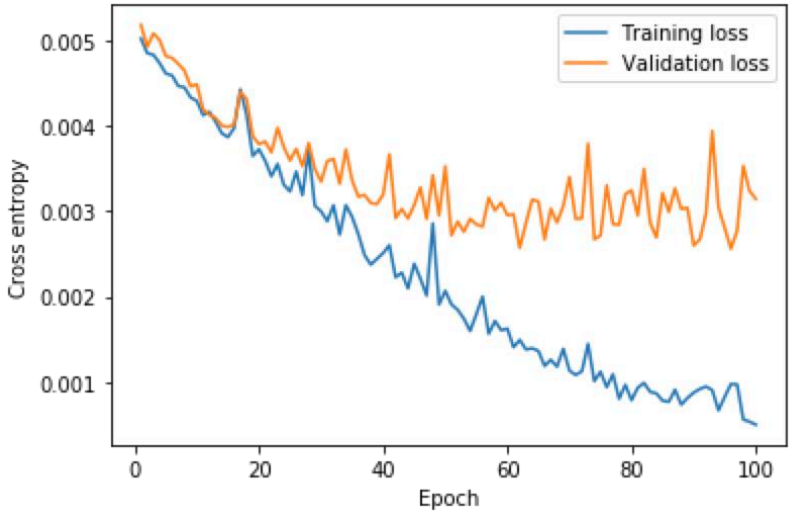


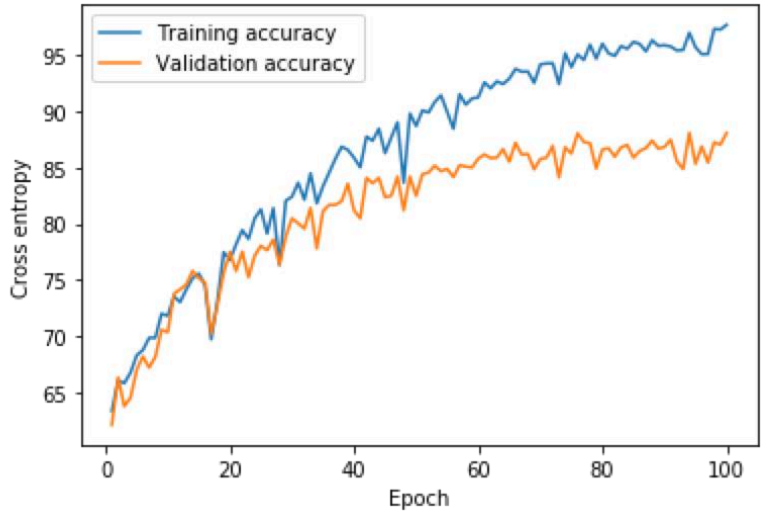


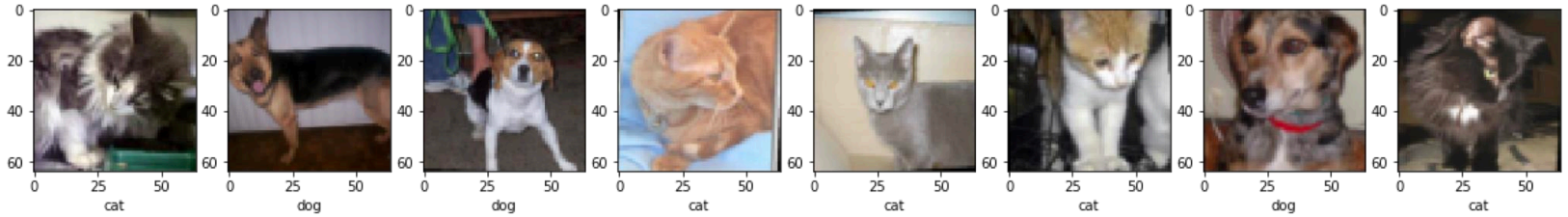
Today: Multilayer Perceptrons & Backpropagation

1. Multilayer Perceptron Architecture
2. Nonlinear Activation Functions
3. Multilayer Perceptron Code Examples
4. Overfitting and Underfitting (intro)
5. **Cats & Dogs and Custom Data Loaders**

VGG16 CNN for Kaggle's Cats & Dogs Images





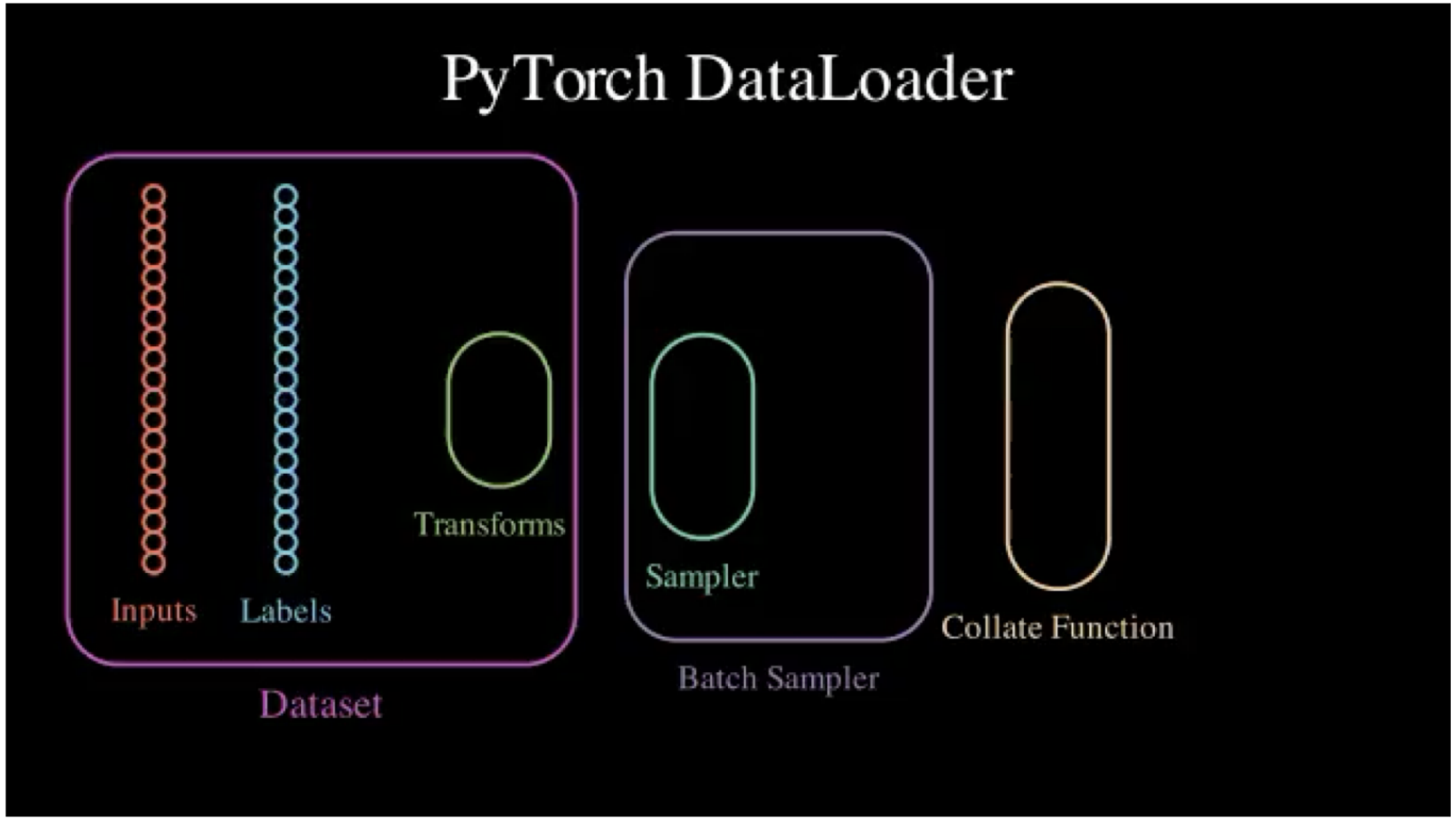



```
model.eval()
with torch.set_grad_enabled(False): # save memory during inference
    test_acc, test_loss = compute_accuracy_and_loss(model, test_loader, DEVICE)
    print(f'Test accuracy: {test_acc:.2f}%')
```

```
Test accuracy: 88.28%
```

https://github.com/rasbt/deeplearning-models/blob/master/pytorch_ipynb/cnn/cnn-vgg16-cats-dogs.ipynb

Loading Data

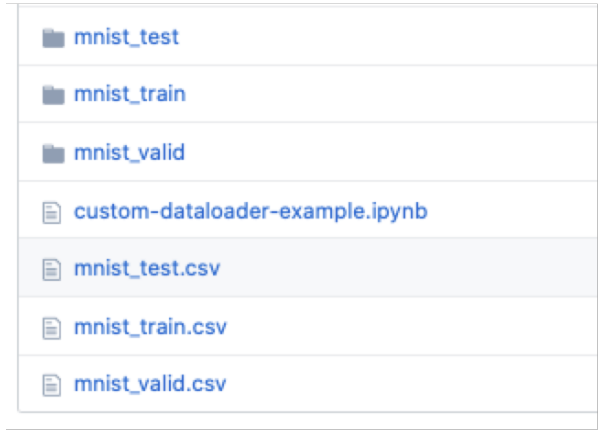


https://x.com/_ScottCondron/status/1363494433715552259

Custom DataLoader Classes

- 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/stat453-deep-learning-ss20/blob/master/L08-mlp/code/custom-dataloader/custom-dataloader-example.ipynb

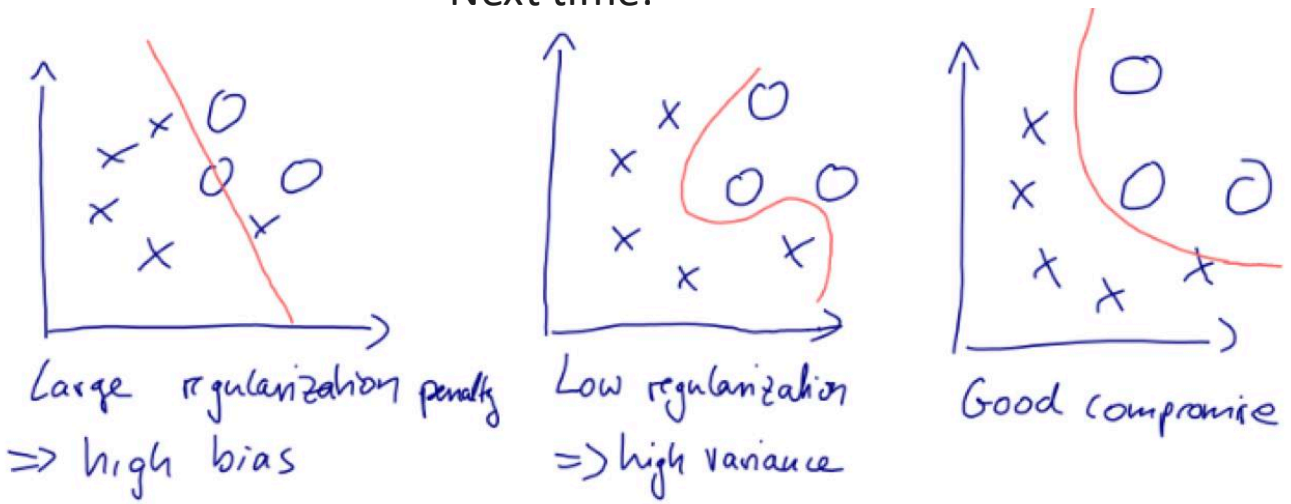



```python
import torch
from PIL import Image
from torch.utils.data import Dataset
import os


class MyDataset(Dataset):

    def __init__(self, csv_path, img_dir, transform=None):

        df = pd.read_csv(csv_path)
        self.img_dir = img_dir
        self.img_names = df['File Name']
        self.y = df['Class Label']
        self.transform = transform

    def __getitem__(self, index):
        img = Image.open(os.path.join(self.img_dir,
                                      self.img_names[index]))

        if self.transform is not None:
            img = self.transform(img)

        label = self.y[index]
        return img, label

    def __len__(self):
        return self.y.shape[0]
```

Where we are…

- Good news: We can solve non-linear problems!
- Bad news: Our multilayer neural networks have lots of parameters and it's easy to overfit the data…

Next time:



Questions?