CS 540 Introduction to Artificial Intelligence

Neural Networks (III)

Yudong Chen
University of Wisconsin-Madison

Oct 26, 2021

Slides created by Sharon Li [modified by Yudong Chen]

Reminder:

Midterm this Thursday

Today's outline

- Deep neural networks
  - Computational graph (forward and backward propagation)
- Numerical stability in training
  - Gradient vanishing/exploding
- Generalization and regularization
  - Overfitting, underfitting
  - Weight decay and dropout

computational/practical considerations.

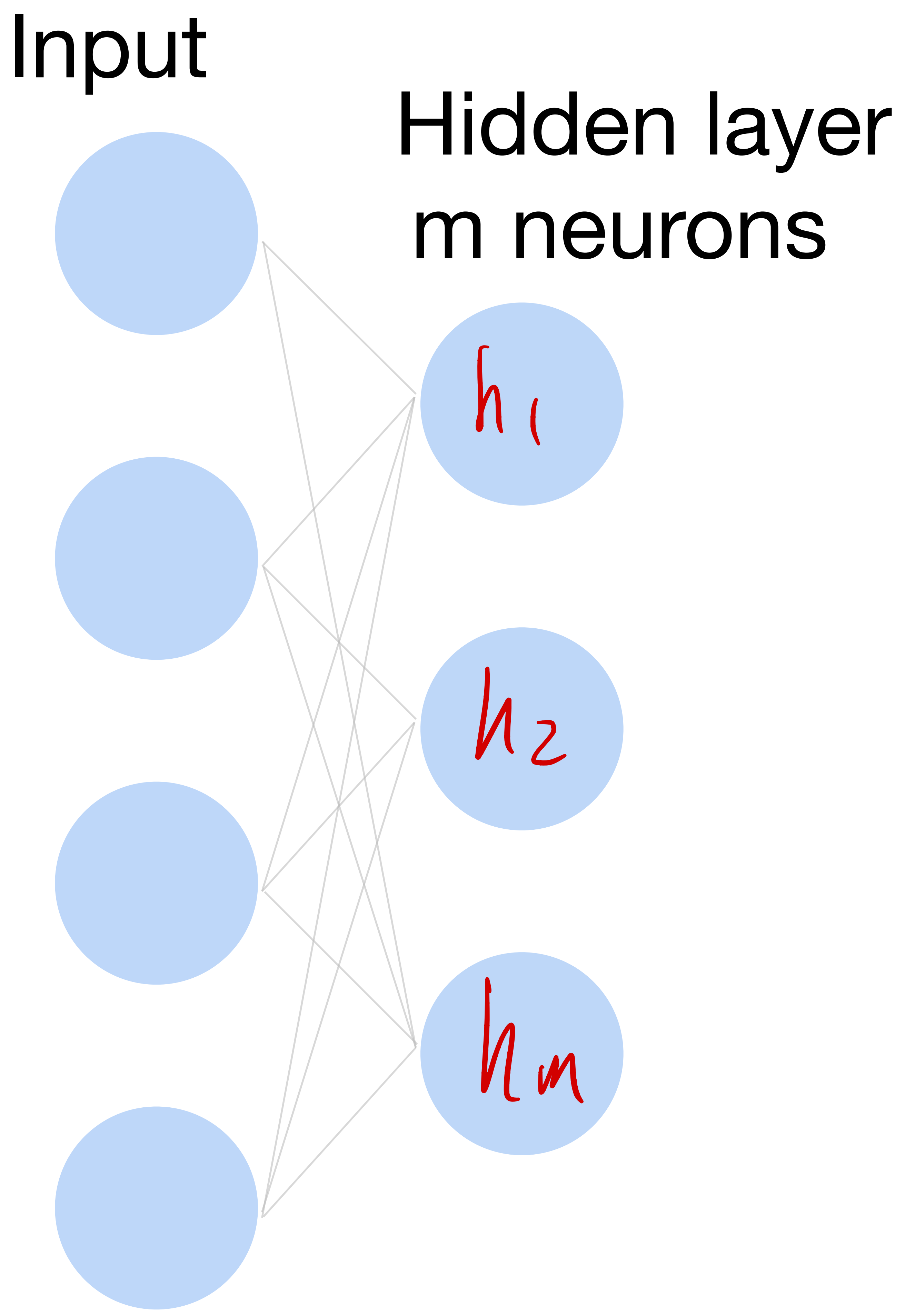
Part I: Neural Networks as a Computational Graph

Review: Neural networks with one hidden layer

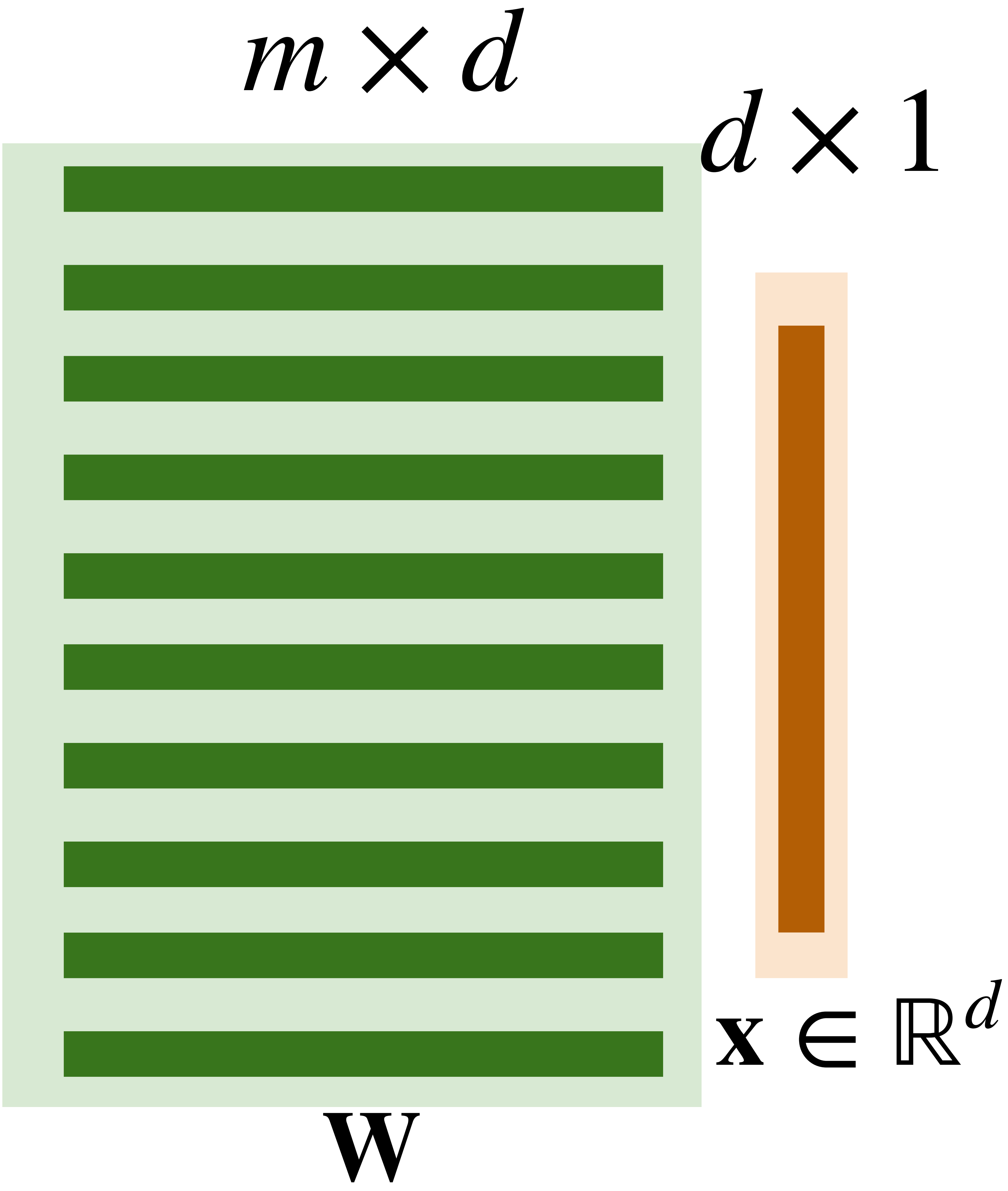
- Input $\mathbf{x} \in \mathbb{R}^d$
- Hidden $\mathbf{W}^{(1)} \in \mathbb{R}^{m \times d}, \mathbf{b} \in \mathbb{R}^m$
- Intermediate output

$$\mathbf{h} = \sigma(\mathbf{W}^{(1)}\mathbf{x} + \mathbf{b})$$

$$\mathbf{h} \in \mathbb{R}^m$$



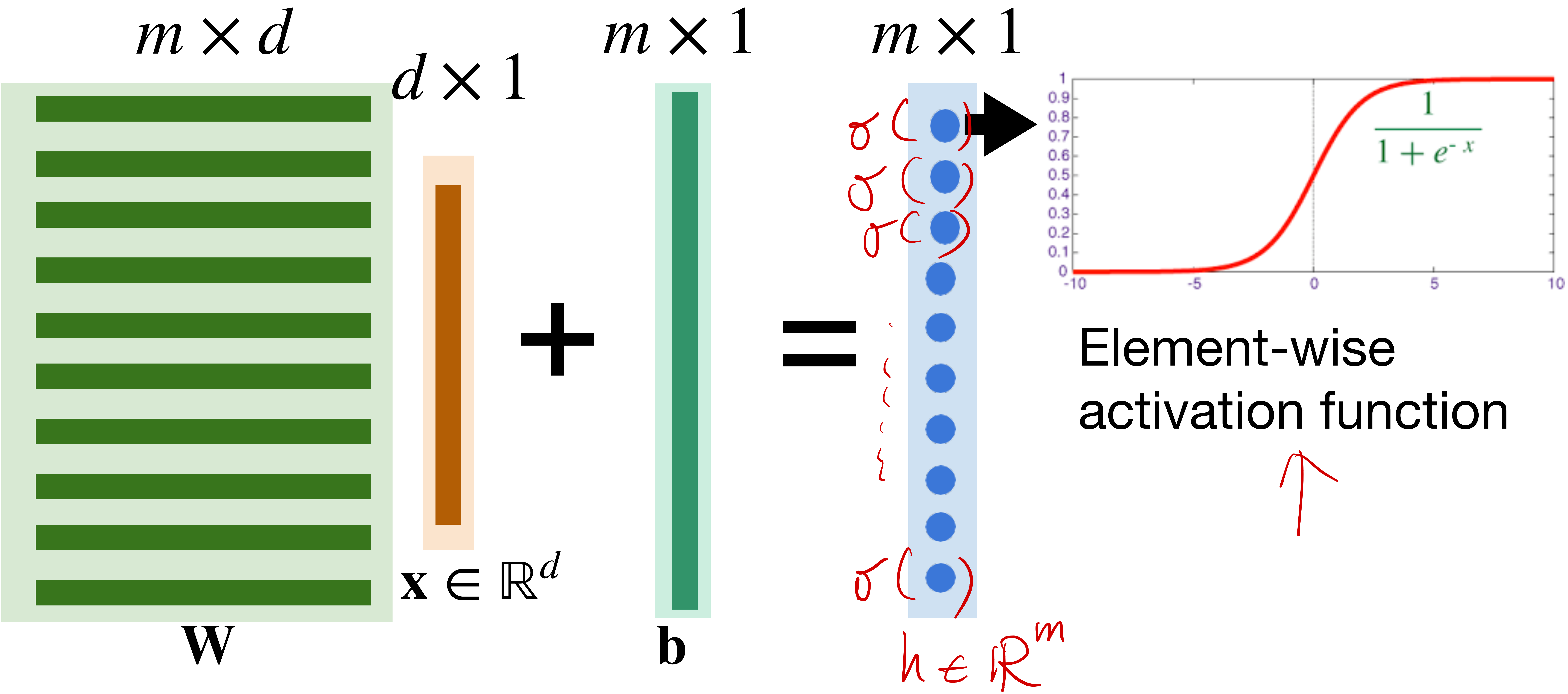
Review: Neural networks with one hidden layer



Review: neural networks with one hidden layer

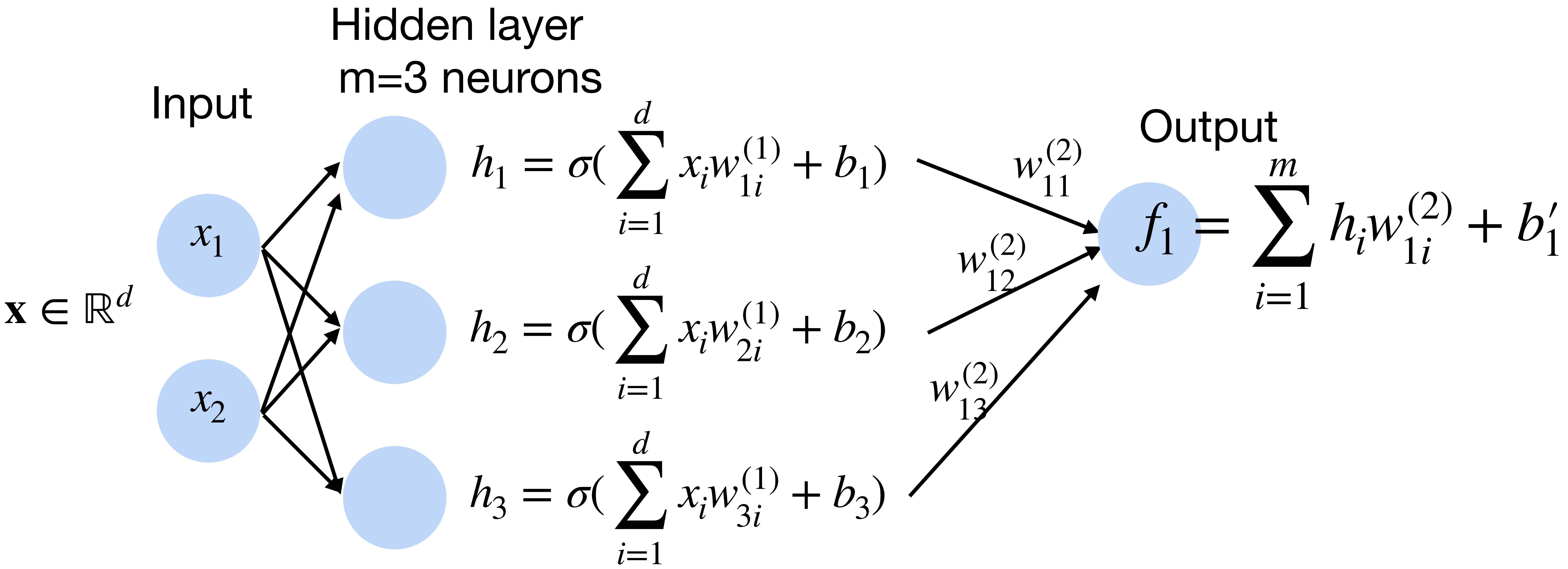
$\sigma(\vec{h})$

Key elements: linear operations + Nonlinear activations



Review: Neural network for *k*-way classification

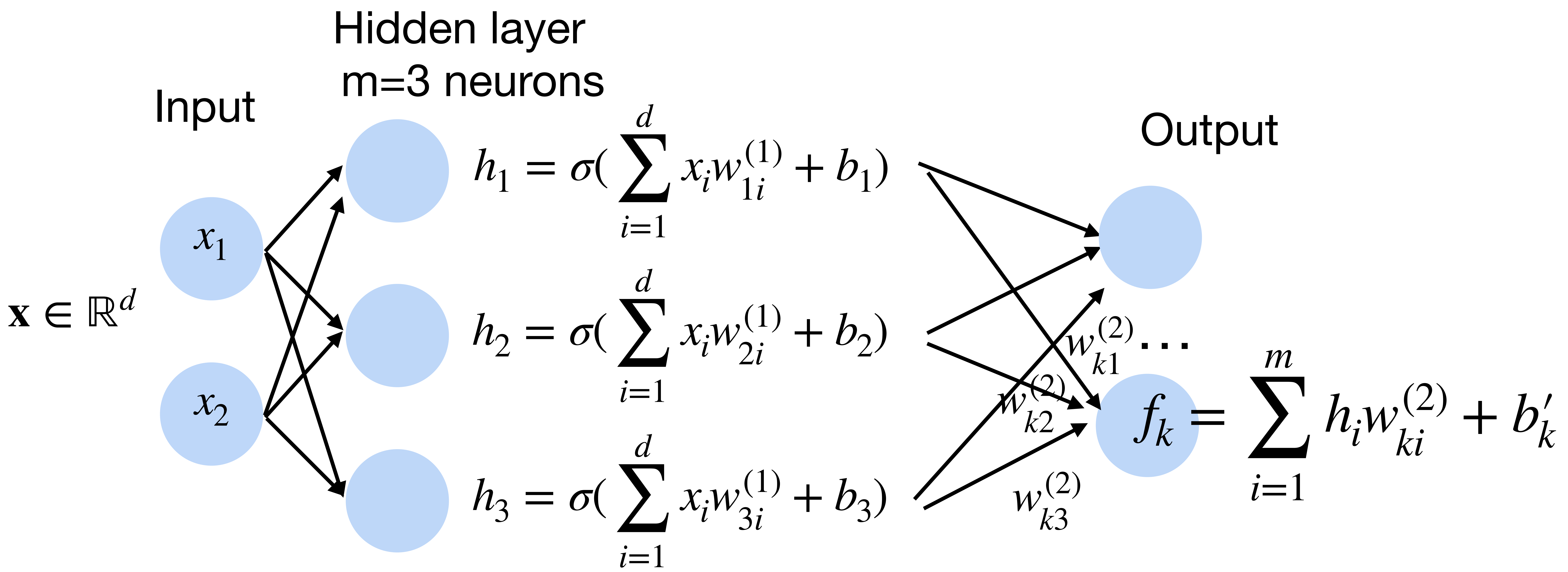
- *k* outputs in the final layer



Review: Neural network for *k*-way classification

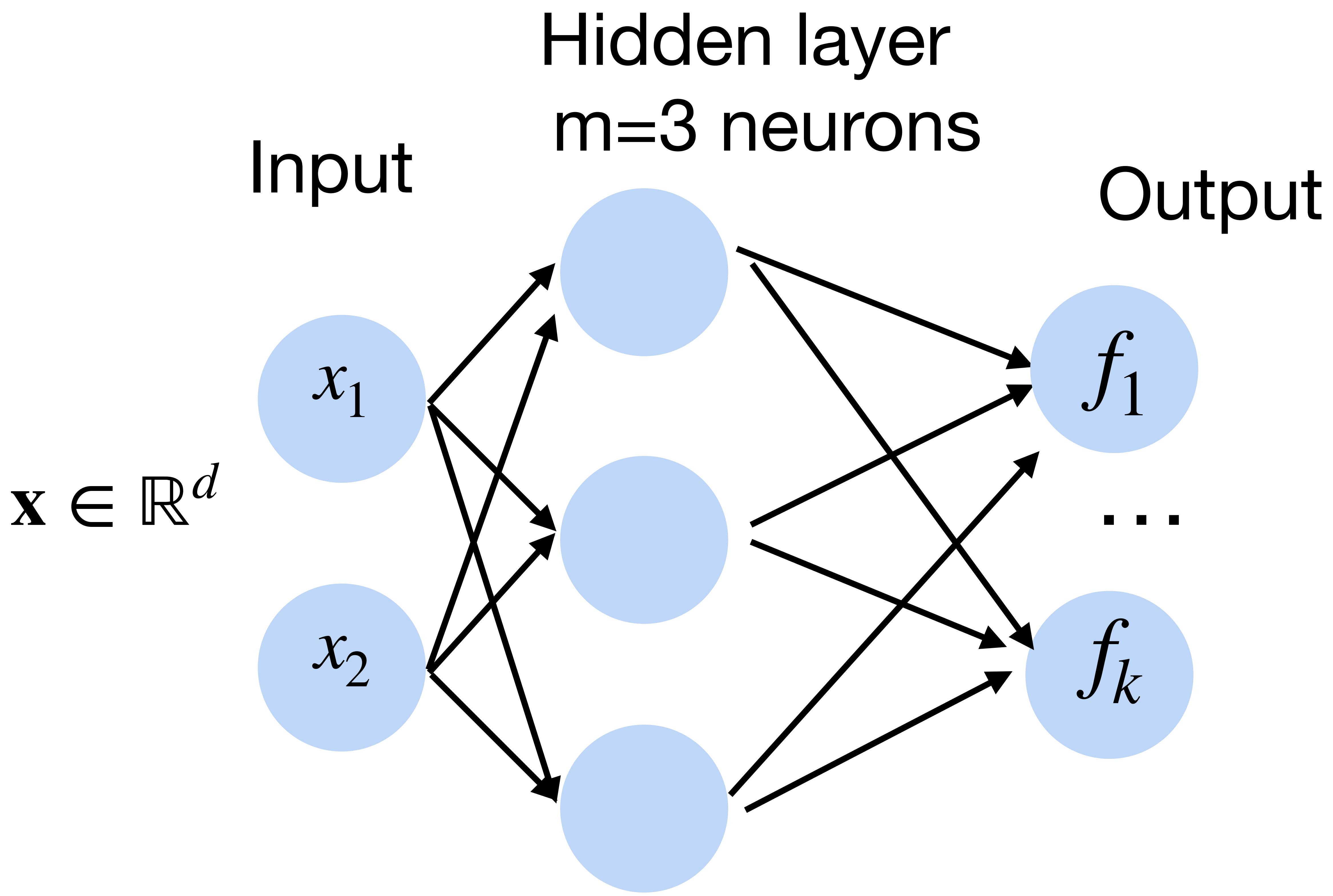
- *k* outputs units in the final layer

k-class classification (e.g., ImageNet has k=1000)



Review: Softmax

Turns outputs f into probabilities (sum up to 1 across k classes)

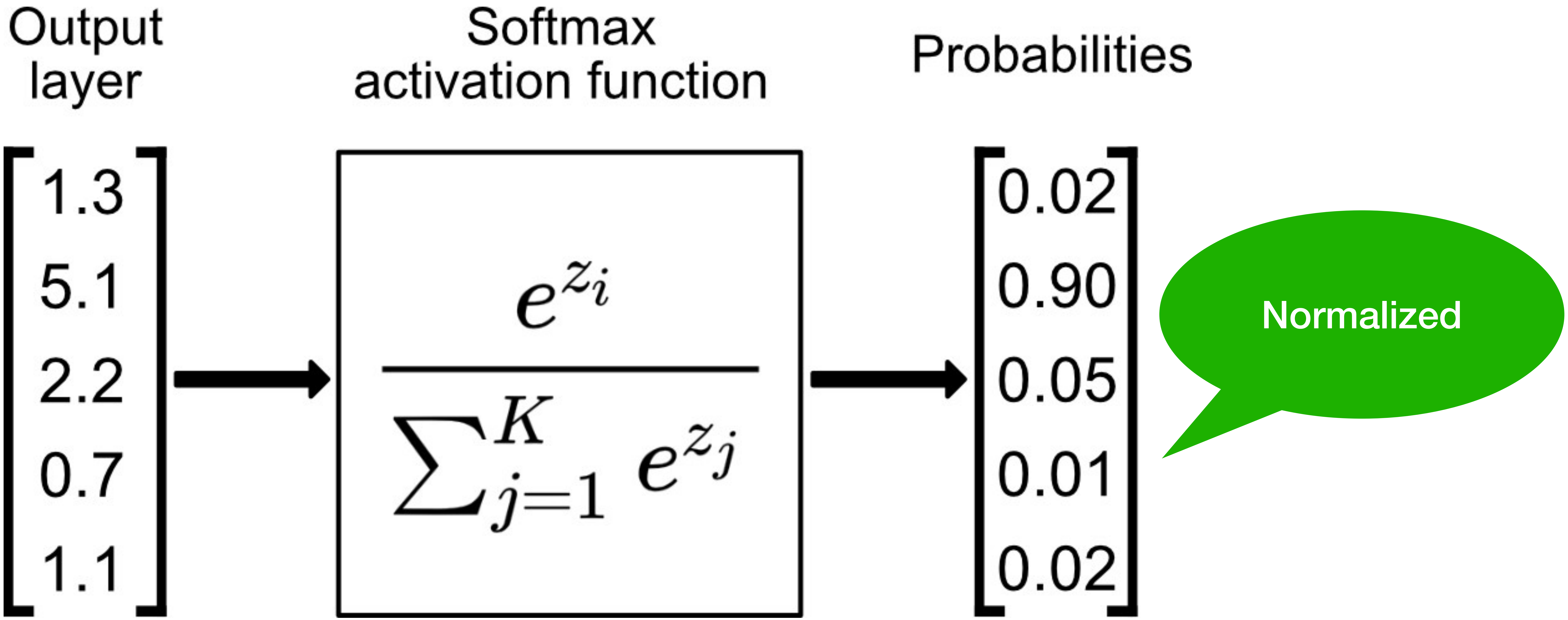


$$p(y \mid \mathbf{x}) = \text{softmax}(f)$$

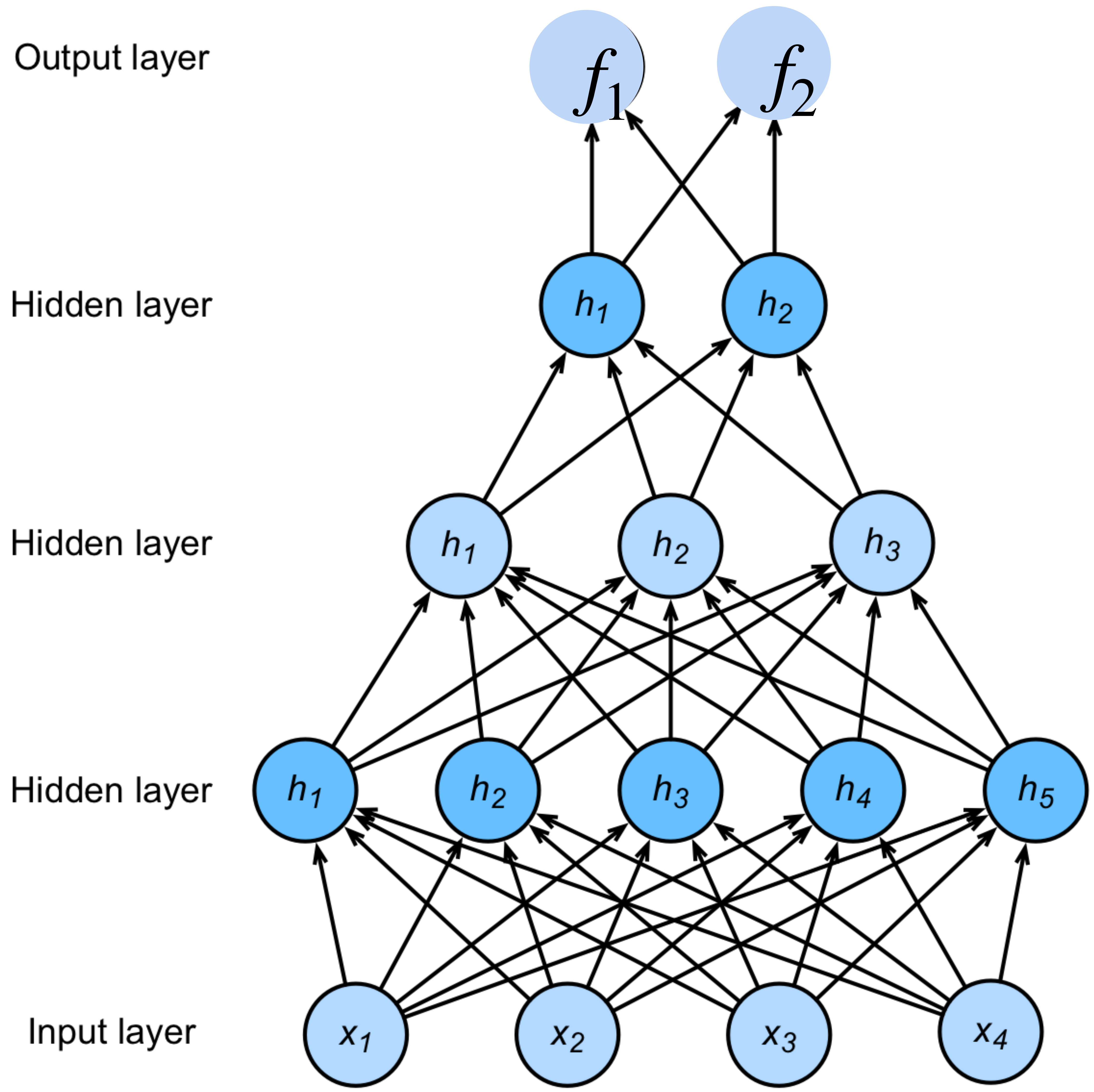
$$= \frac{\exp f_y(x)}{\sum_i^k \exp f_i(x)}$$

Review: Softmax

Turns outputs f into probabilities (sum up to 1 across k classes)



Deep neural networks (DNNs)



$$\mathbf{h}_1 = \sigma(\mathbf{W}^{(1)}\mathbf{x} + \mathbf{b}^{(1)})$$

$$\mathbf{h}_2 = \sigma(\mathbf{W}^{(2)}\mathbf{h}_1 + \mathbf{b}^{(2)})$$

$$\mathbf{h}_3 = \sigma(\mathbf{W}^{(3)}\mathbf{h}_2 + \mathbf{b}^{(3)})$$

$$\mathbf{f} = \mathbf{W}^{(4)}\mathbf{h}_3 + \mathbf{b}^{(4)}$$

$$\mathbf{y} = \mathrm{softmax}(\mathbf{f})$$

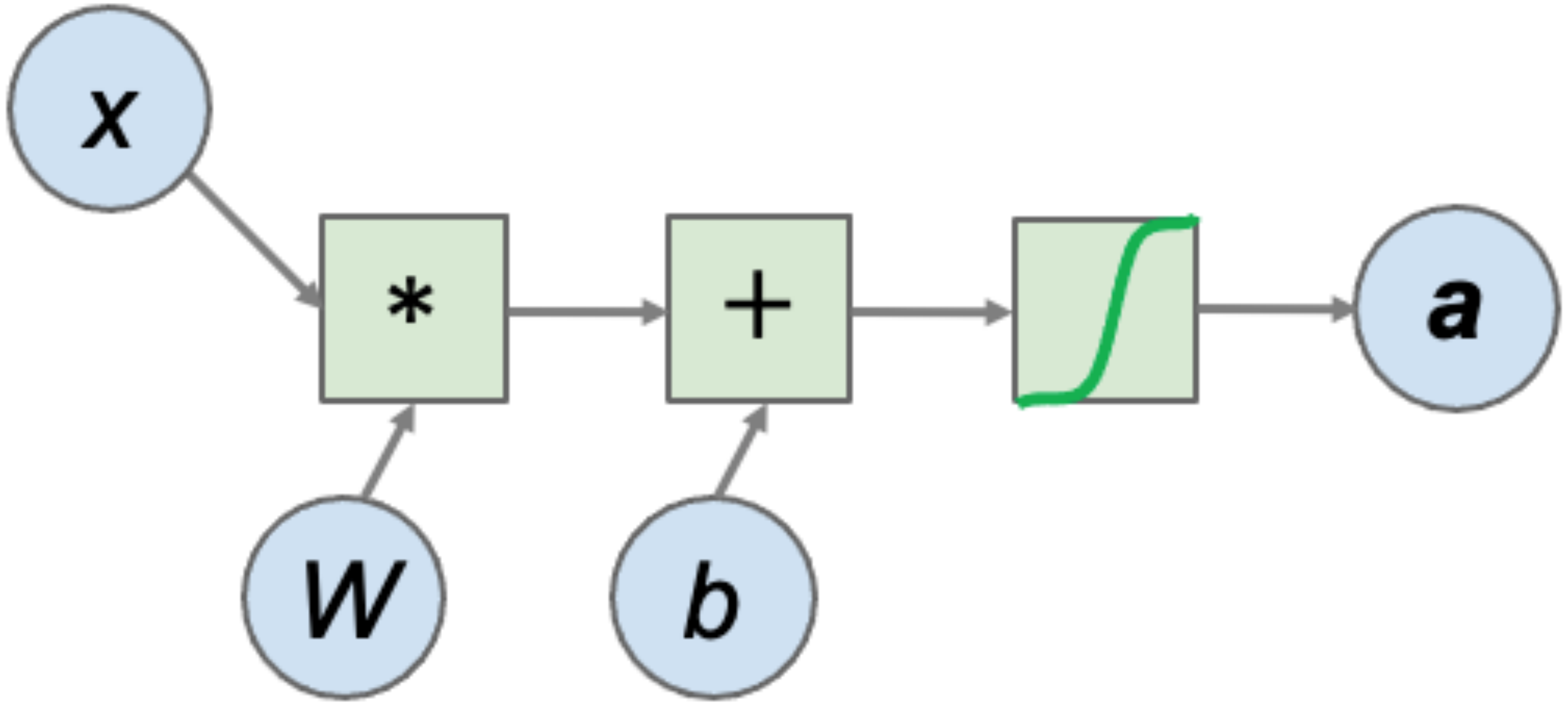
NNs are composition of nonlinear functions

Neural networks as variables + operations

$$\mathbf{a} = \text{sigmoid}(\mathbf{W}\mathbf{x} + \mathbf{b})$$

logistic regression.

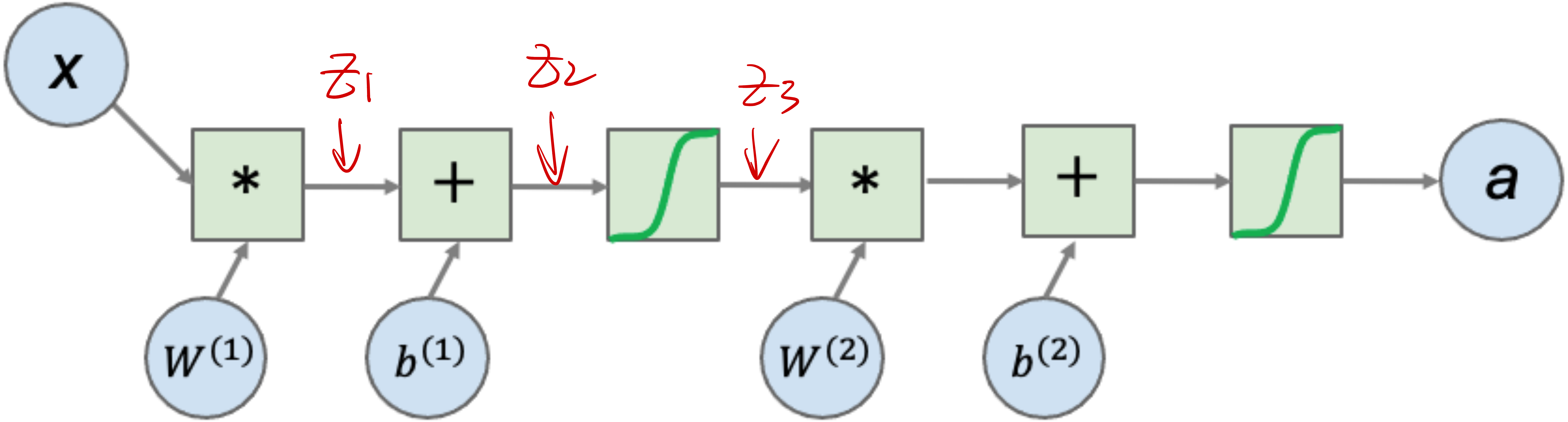
- Decompose functions into atomic operations
- Separate data (variables) and computing (operations)
- Known as a **computational graph**



Neural networks as a computational graph

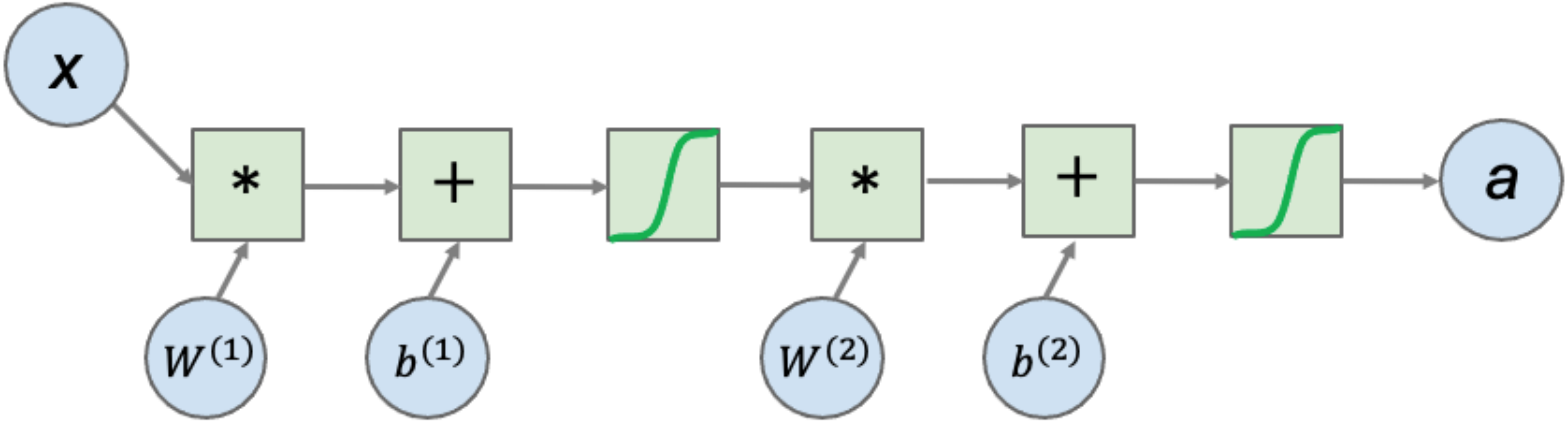
- A two-layer neural network

$z_1 = W^{(1)} x \qquad z_2 = z_1 + b^{(1)} \qquad z_3 = \sigma(z_2)$



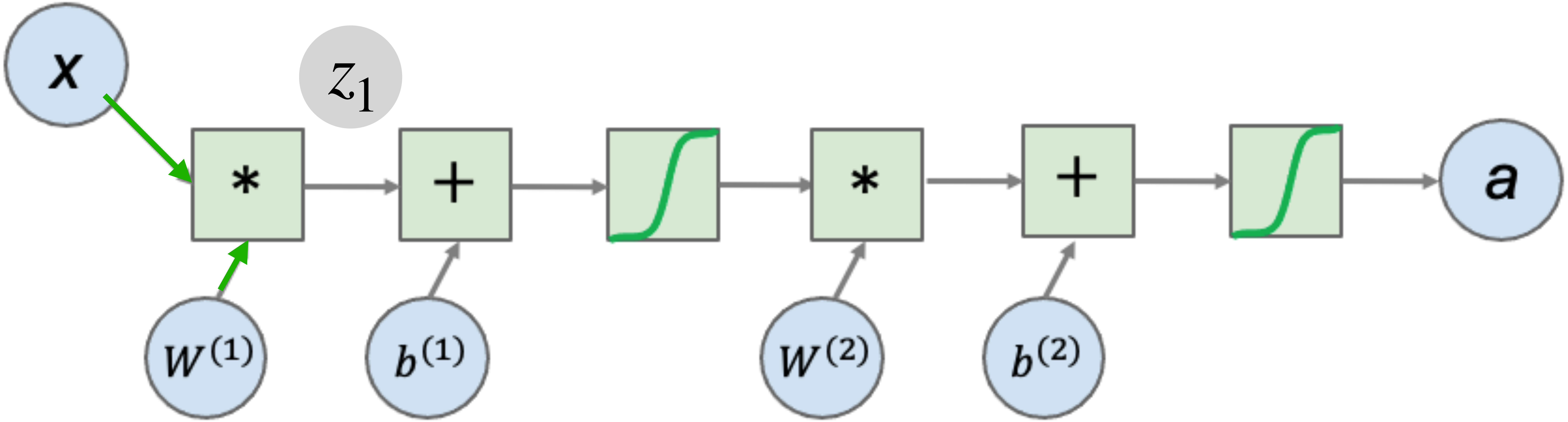
Neural networks as a computational graph

- A two-layer neural network
- Forward propagation vs. backward propagation



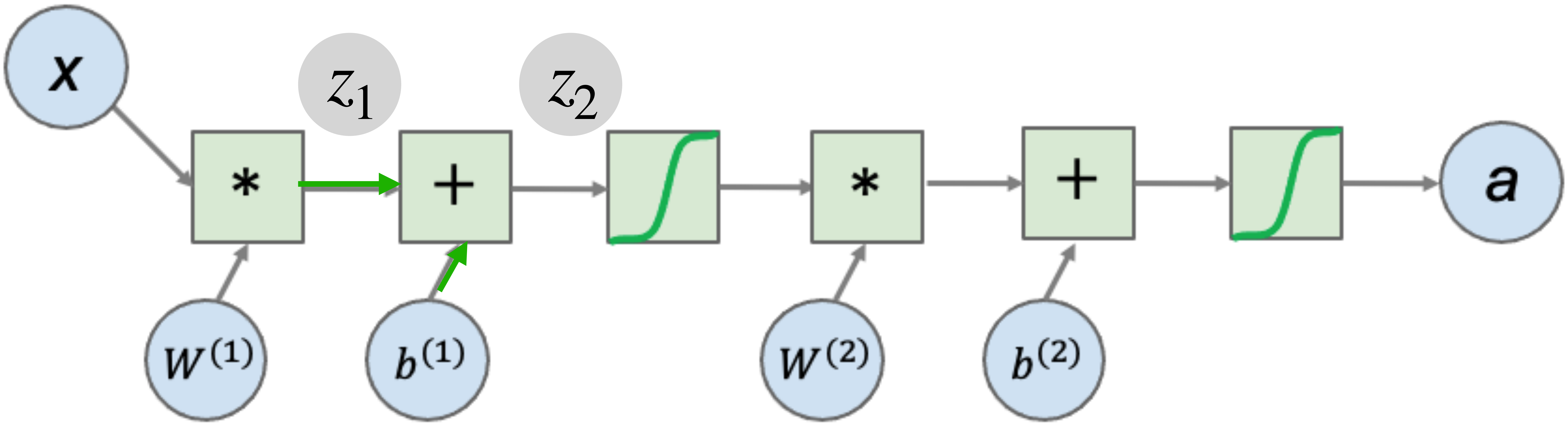
Neural networks: forward propagation

- A two-layer neural network
- Intermediate variables Z



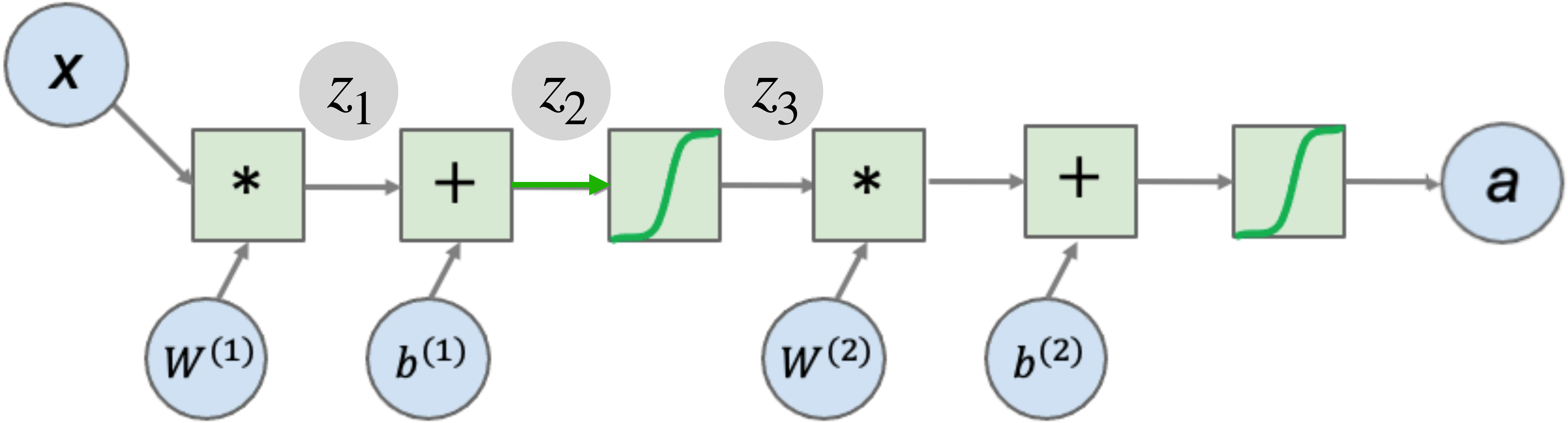
Neural networks: forward propagation

- A two-layer neural network
- Intermediate variables Z



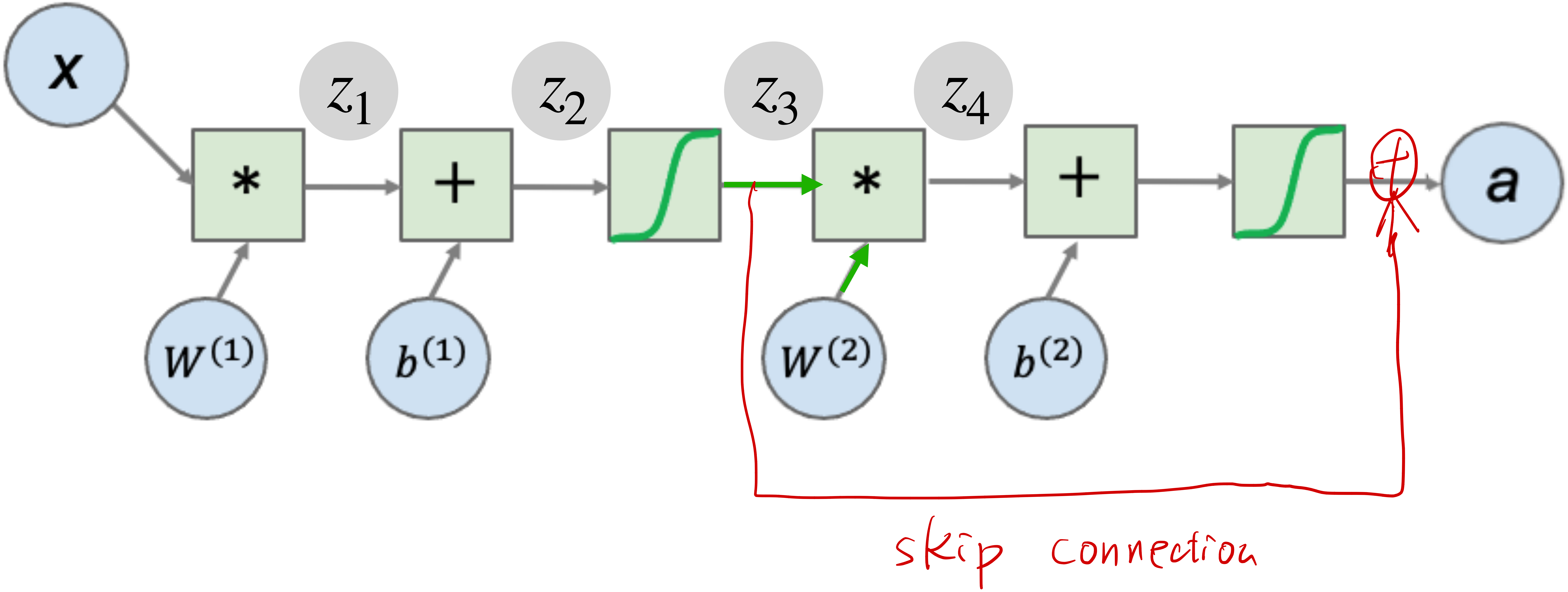
Neural networks: forward propagation

- A two-layer neural network
- Intermediate variables Z



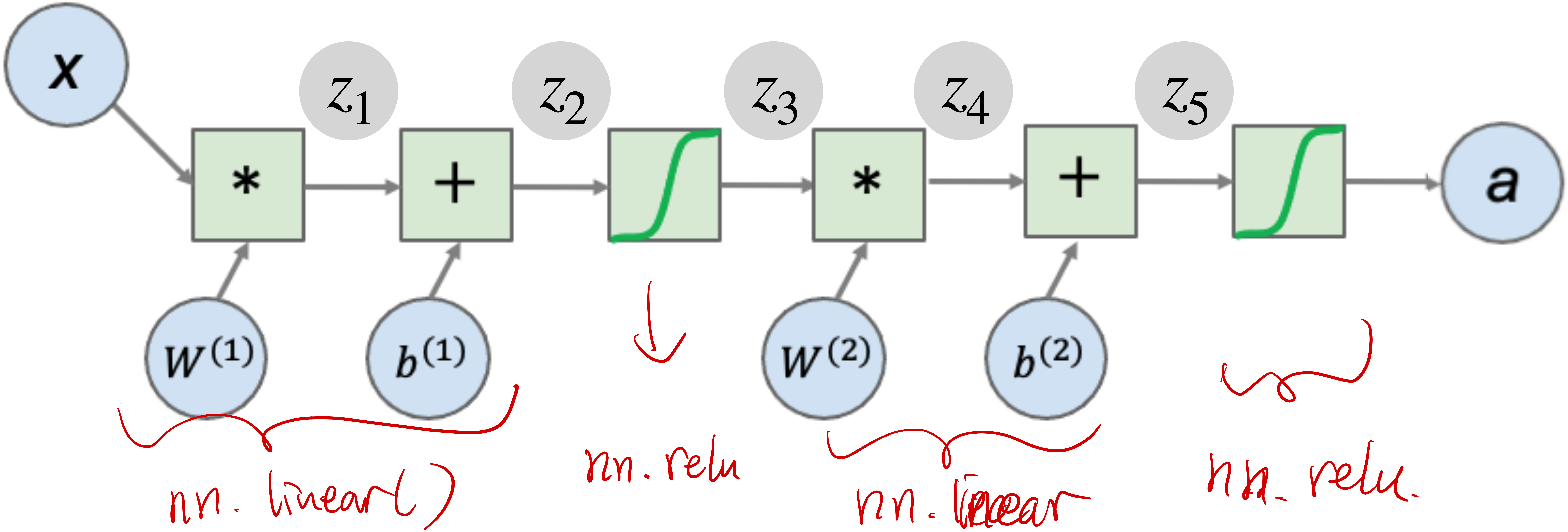
Neural networks: forward propagation

- A two-layer neural network $a = \sigma\left(W^{(2)} z_3 + b^{(2)}\right) + z_3$
- Intermediate variables Z



Neural networks: forward propagation

- A two-layer neural network
- Intermediate variables Z

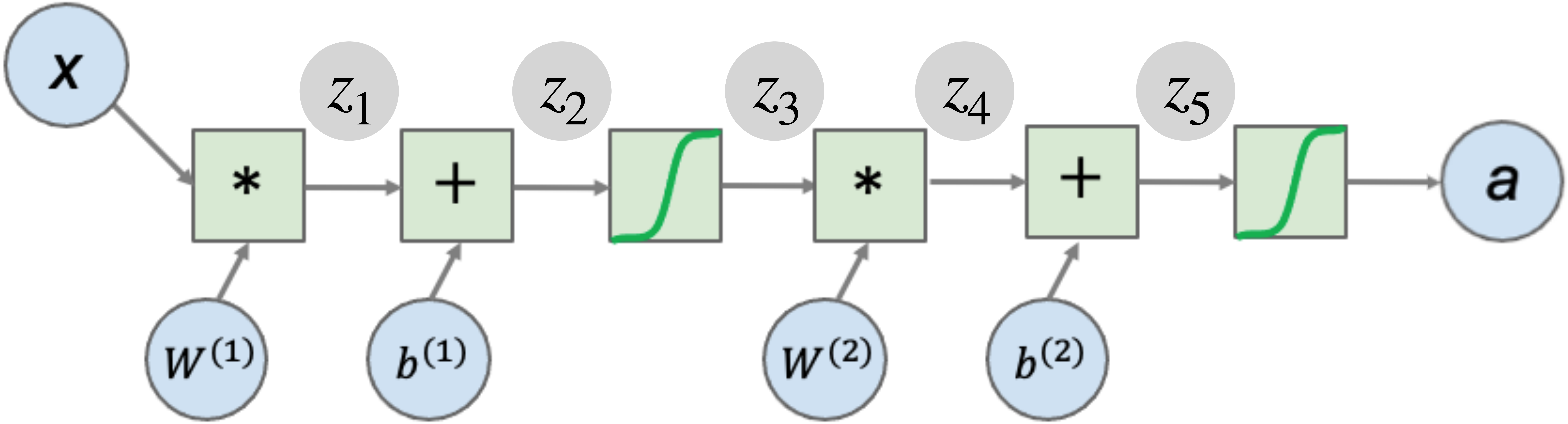


Neural networks: backward propagation

- A two-layer neural network
- Assuming forward propagation is done
- Minimize a **loss function** *L*

$\min_{w} L(y, \hat{y}_w)$

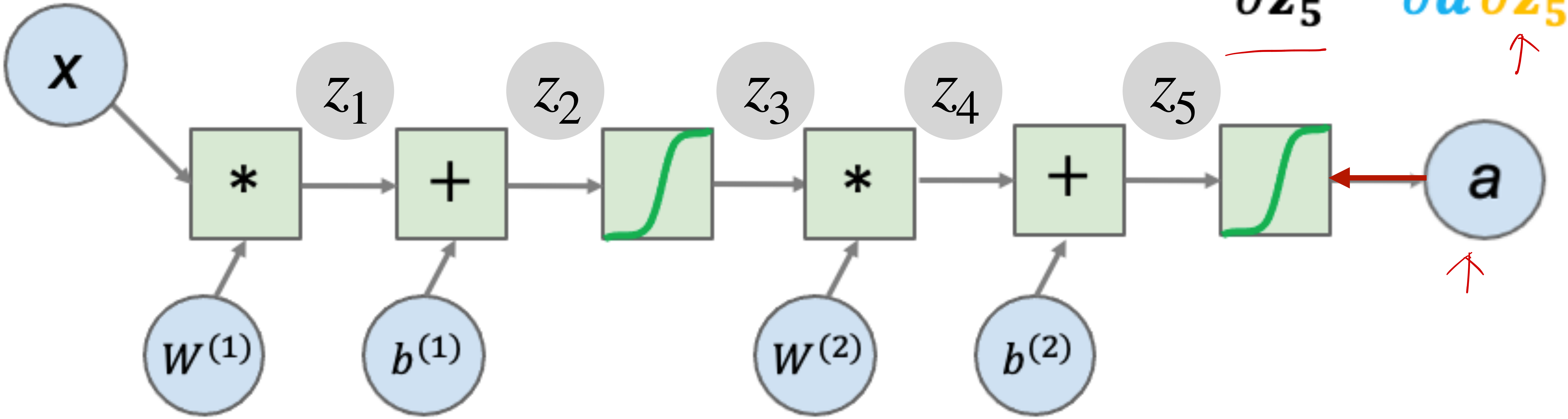
$\frac{\partial L}{\partial w}$



Neural networks: backward propagation

- A two-layer neural network
- Assuming forward propagation is done
- Minimize a **loss function** *L*

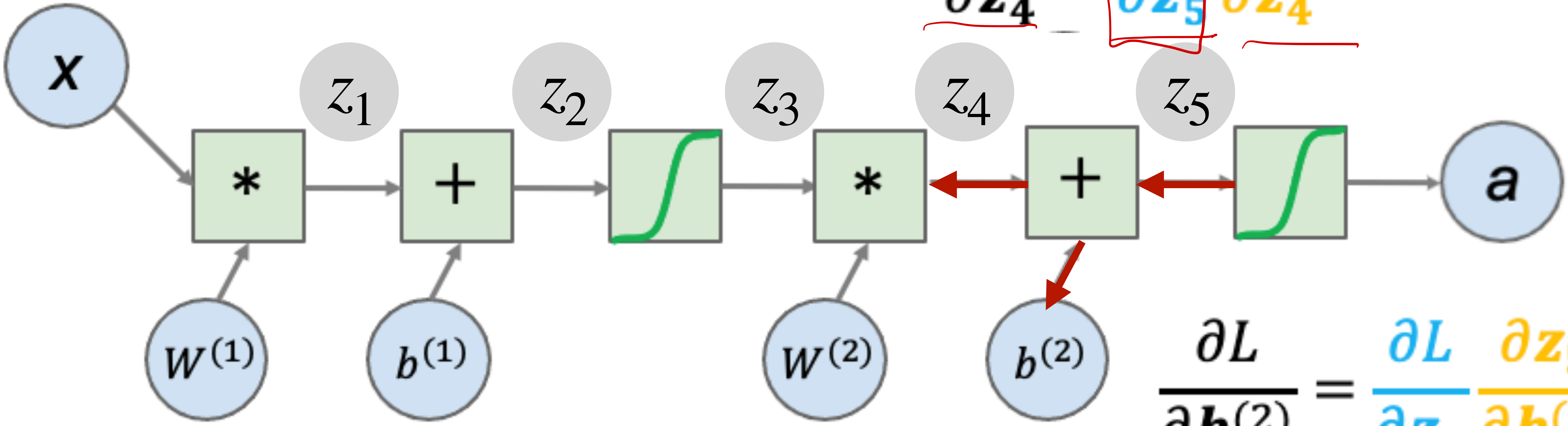
$$\frac{\partial L}{\partial \boldsymbol{z_5}} = \frac{\partial L}{\partial \boldsymbol{a}} \frac{\partial \boldsymbol{a}}{\partial \boldsymbol{z_5}}$$



Neural networks: backward propagation

- A two-layer neural network
- Assuming forward propagation is done
- Minimize a **loss function** L

$$\frac{\partial L}{\partial \boldsymbol{z}_4} = \frac{\partial L}{\partial \boldsymbol{z}_5}\frac{\partial \boldsymbol{z}_5}{\partial \boldsymbol{z}_4}$$



$$\frac{\partial L}{\partial \boldsymbol{b}^{(2)}} = \frac{\partial L}{\partial \boldsymbol{z}_5}\frac{\partial \boldsymbol{z}_5}{\partial \boldsymbol{b}^{(2)}}$$

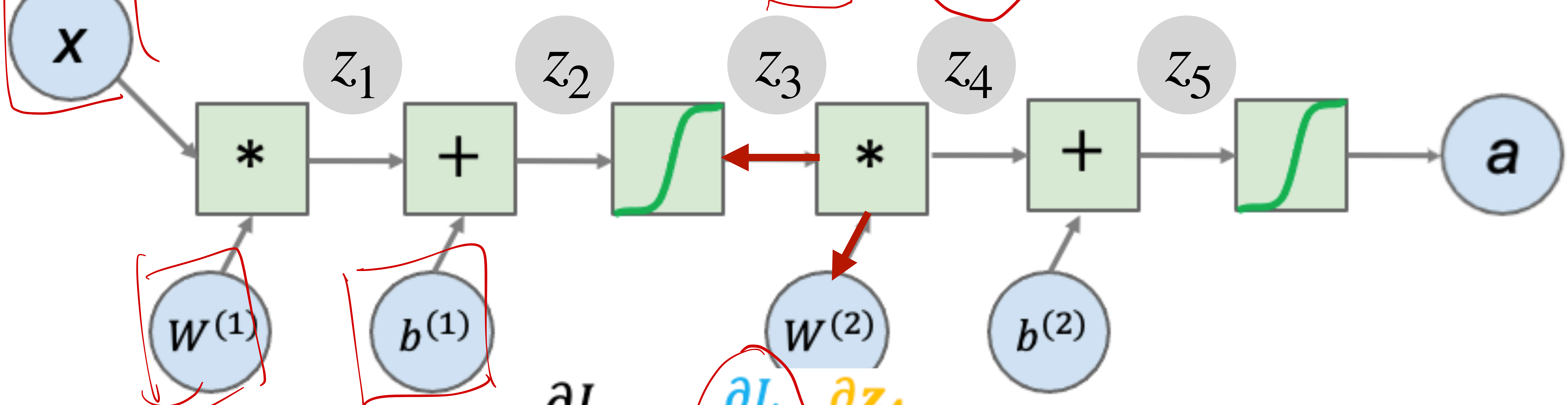
Neural networks: backward propagation

PyTorch

- A two-layer neural network
- Assuming forward propagation is done

TensorFlow

$$\frac{\partial L}{\partial \mathbf{z}_3} = \frac{\partial L}{\partial \mathbf{z}_4} \frac{\partial \mathbf{z}_4}{\partial \mathbf{z}_3}$$



Tensor

$$\frac{\partial L}{\partial \boldsymbol{W}^{(2)}} = \frac{\partial L}{\partial \mathbf{z}_4} \frac{\partial \mathbf{z}_4}{\partial \boldsymbol{W}^{(2)}}$$

Backward propagation: A modern treatment

- Define a neural network as a computational graph
- Must be a directed graph
- Nodes as variables and operations
- All operations must be **differentiable**

- Facilitate *automatic differentiation*

Part II: Numerical Stability

Gradients for Neural Networks

- Compute the gradient of the loss ℓ w.r.t. $\mathbf{W}_t$

$$\frac{\partial \ell}{\partial \mathbf{W}^t} = \frac{\partial \ell}{\partial \mathbf{h}^d} \underbrace{\frac{\partial \mathbf{h}^d}{\partial \mathbf{h}^{d-1}} \cdots \frac{\partial \mathbf{h}^{t+1}}{\partial \mathbf{h}^t}}_{\text{Multiplication of \textit{many} matrices}} \frac{\partial \mathbf{h}^t}{\partial \mathbf{W}^t}$$

$$\frac{\partial \ell}{w^1} = \frac{\partial \ell}{h^d} \cdot \frac{\partial h^d}{\partial h^{d-1}} \cdots \frac{\partial h^1}{\partial w^1}$$

1.5 1.5 1.5

Wikipedia

Two Issues for Deep Neural Networks

$$\prod_{i=t}^{d-1} \frac{\partial \mathbf{h}^{i+1}}{\partial \mathbf{h}^{i}}$$

$w^{t+1} \Leftarrow w^t - \alpha \cdot \frac{\partial L}{\partial w^t}$

10^{-10}

Gradient Exploding	Gradient Vanishing
	
$1.5^{100} \approx 4 \times 10^{17}$	$0.8^{100} \approx 2 \times 10^{-10}$

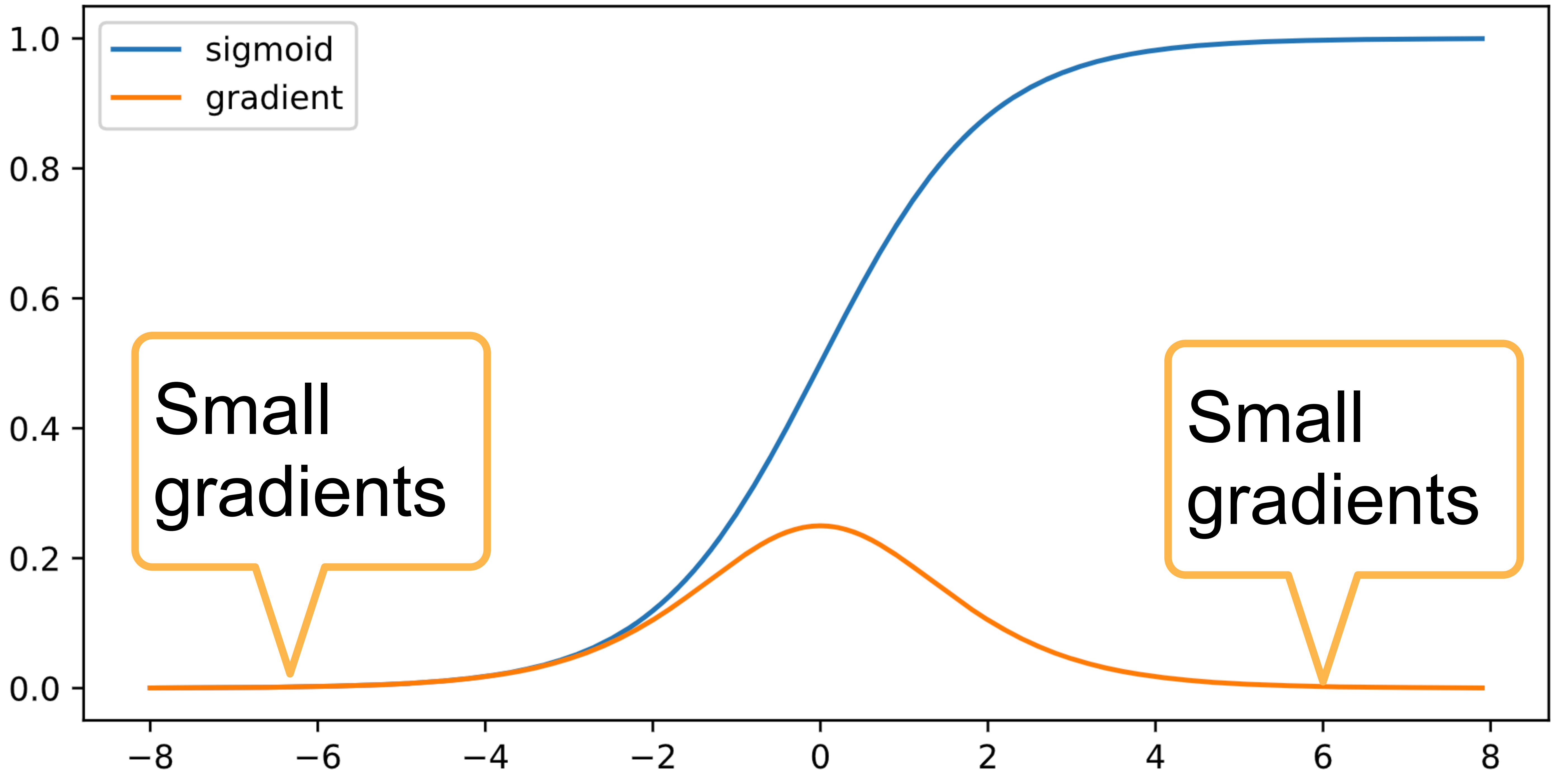
Issues with Gradient Exploding

- Value out of range: infinity value (NaN)
- Sensitive to learning rate (LR)
  - Not small enough LR -> larger gradients
  - Too small LR -> No progress
  - May need to change LR dramatically during training

Gradient Vanishing

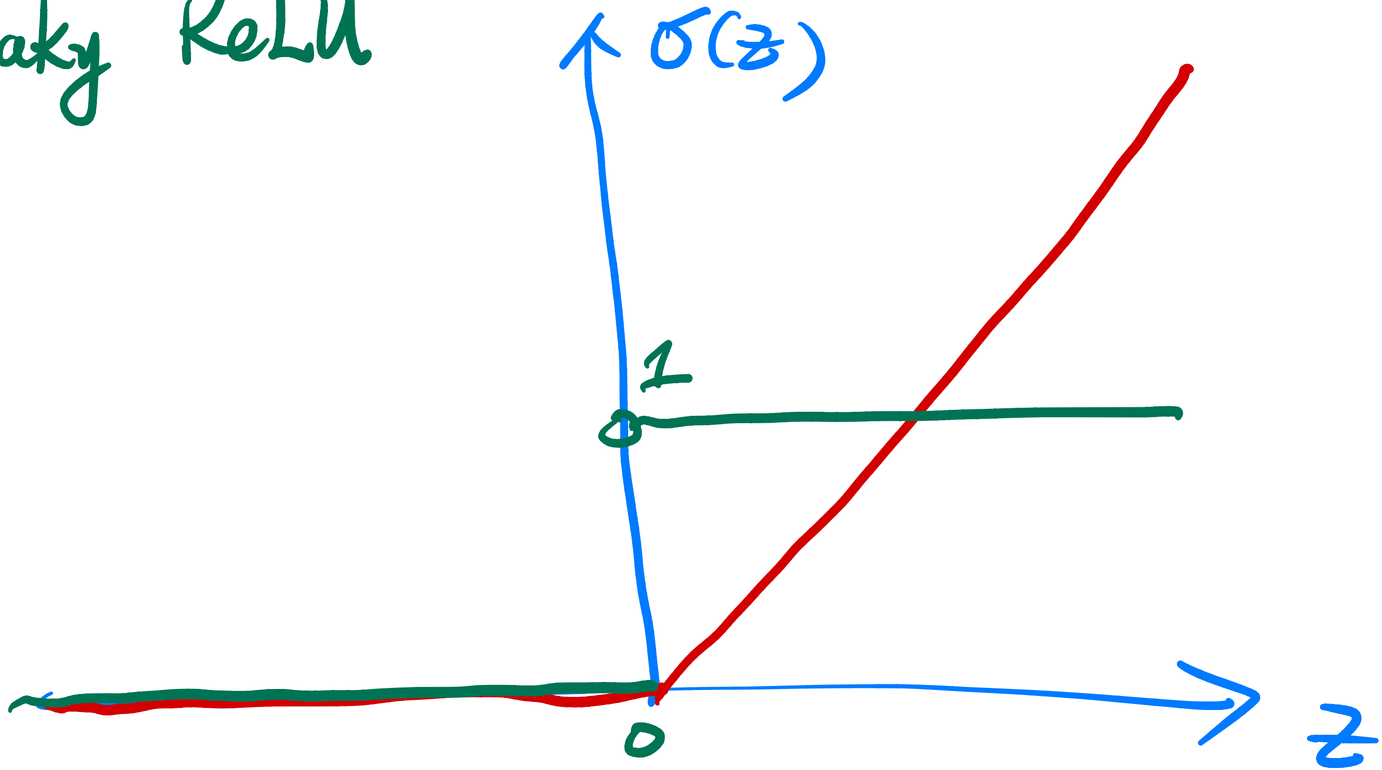
- Use sigmoid as the activation function

$$\sigma(x) = \frac{1}{1 + e^{-x}} \qquad \sigma'(x) = \sigma(x)(1 - \sigma(x))$$

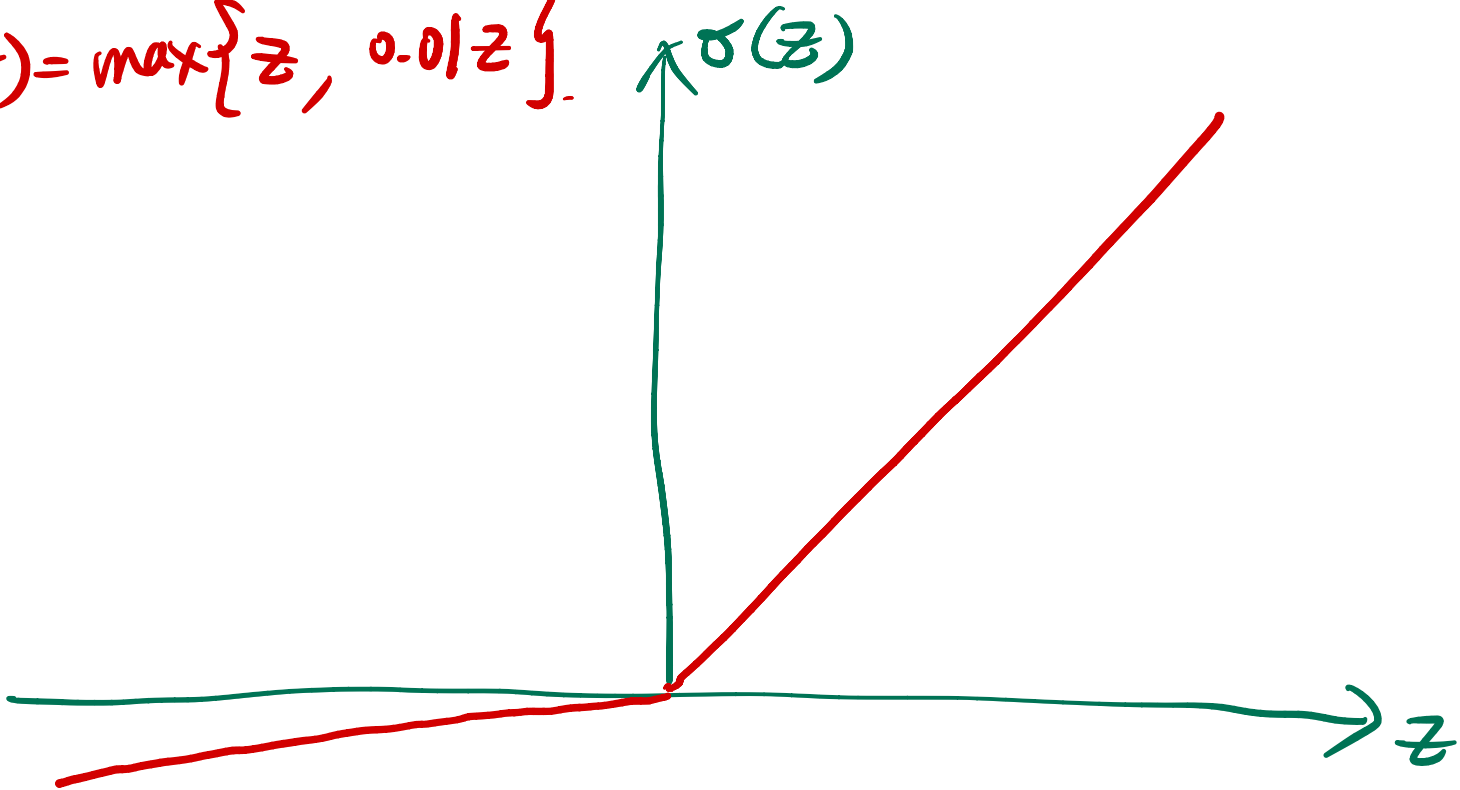


ReLU: $\sigma(z) = \max\{z, 0\}$.

Leaky ReLU



$$\sigma(z) = \max\{z, 0.01z\}.$$



Leaky ReLU.

Issues with Gradient Vanishing

- Gradients with value 0
- No progress in training
  - No matter how to choose learning rate
- Severe with bottom layers
  - Only top layers are well trained
  - No benefit to make networks deeper

How to stabilize training?

Stabilize Training: Practical Considerations

~~a~~ avoid. NaN

- Goal: make sure gradient values are in a proper range
  - E.g. in [1e-6, 1e3]

by clipping

- Multiplication -> plus
  - Architecture change (e.g., ResNet)
- Normalization
  - Batch Normalization, Gradient clipping
- Proper activation functions

$$\hat{h}^{(t)} = \sigma(W^t h^{(t-1)} + b^{(t)})$$

$$h^{(t)} = \frac{\hat{h}^{(t)}}{\|\hat{h}^{(t)}\|_2}$$

Part III: Generalization & Regularization

How good are the models?

Training Error and Generalization Error

- Training error: prediction error on the training data
- **Generalization/test error**: prediction error on new data
- Example: practice for a future exam with past exams
  - Should practice and do well on past exams (training is needed)
  - However, doing well on past exams (training error) doesn’t guarantee a good score on the future exam (generalization error)

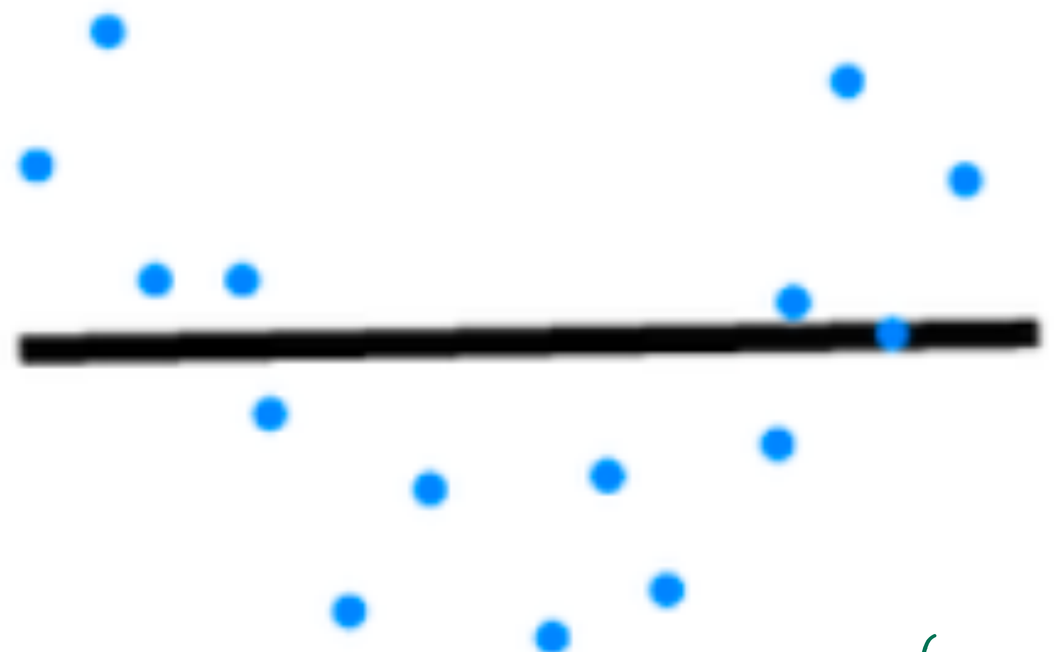
Underfitting
Overfitting

Image credit: hackernoon.com

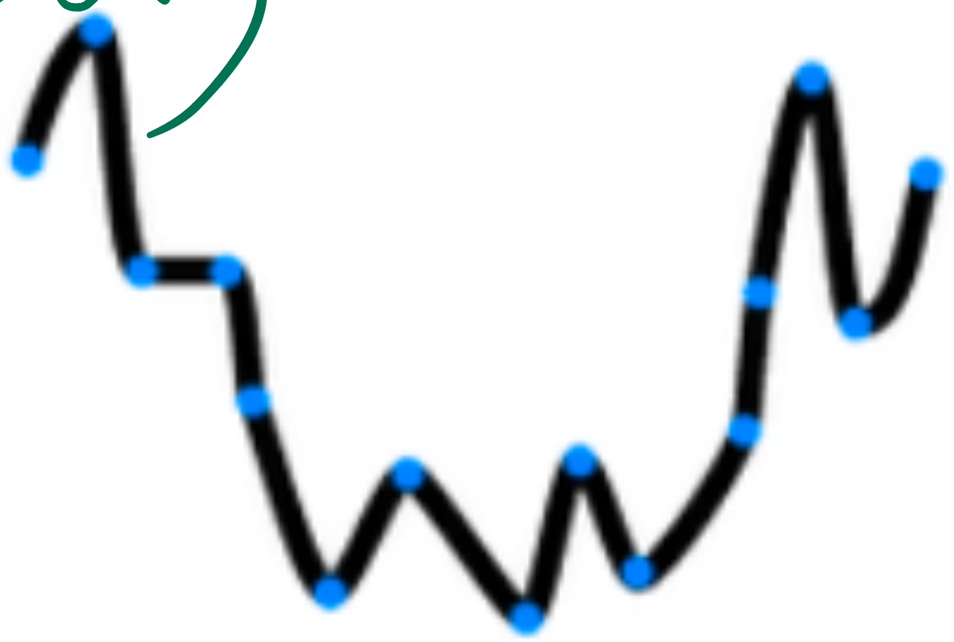
Model Capacity

high capacity model ⟺ model flexible.
DNN
fit very complicated data.

- The ability to fit variety of functions
- Low capacity models struggle to fit training set
  - Underfitting
- High capacity models can memorize the training set
  - Overfitting



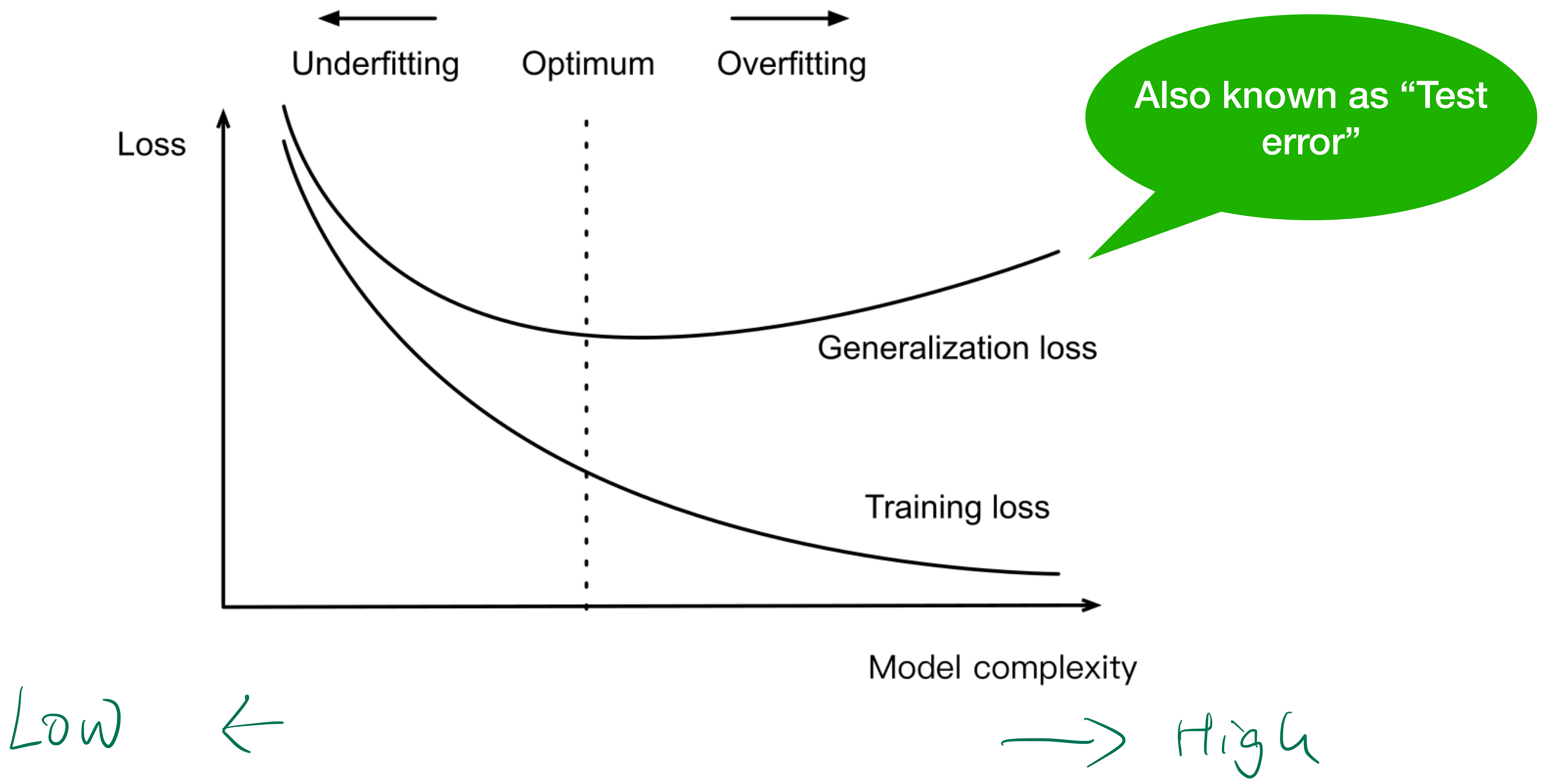
low capacity model ⟺ model too simple
(linear regression)
inflexible.



Underfitting and Overfitting

	Data complexity	ImageNet.
Model capacity	Simple	Complex
Low	Normal	Underfitting
High	Overfitting	Normal

Influence of Model Complexity



Estimate Neural Network Capacity

kNN $k=10$

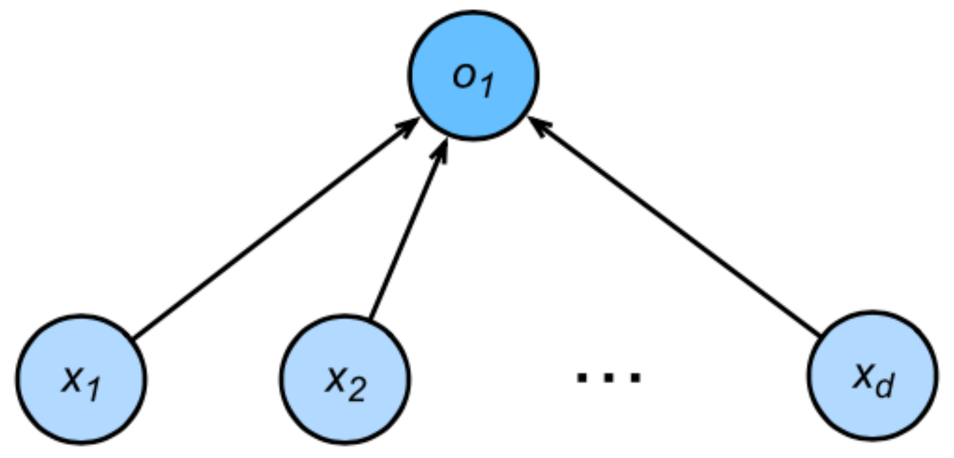
- It is hard to compare complexity between different algorithm families
  - e.g. kNN vs neural networks

NN $h=10$

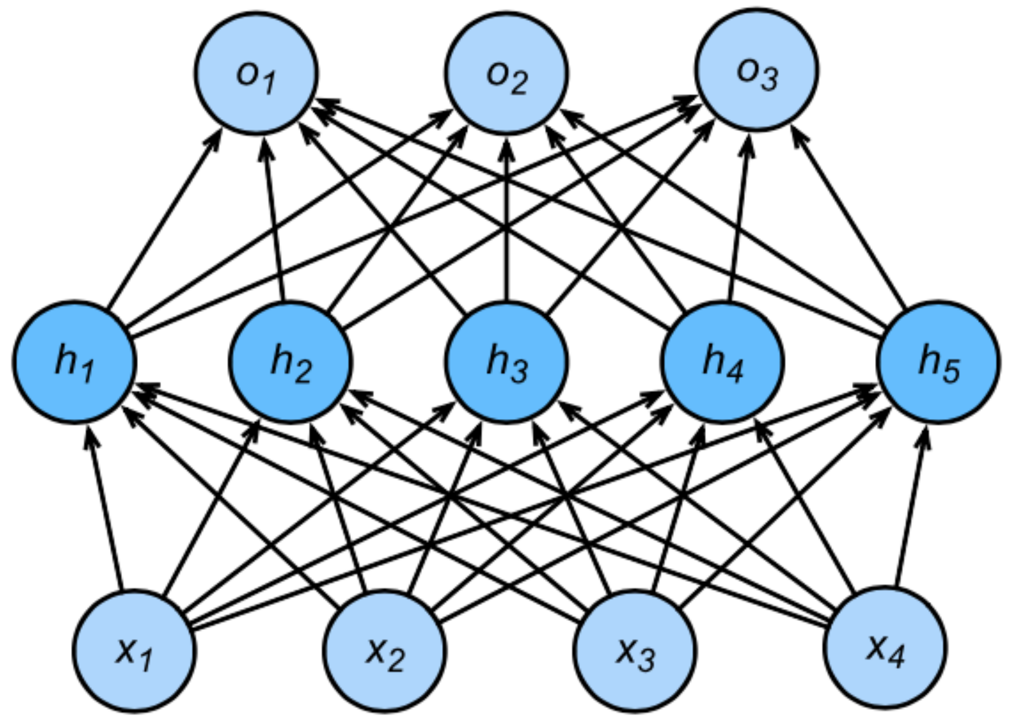
Estimate Neural Network Capacity

- It is hard to compare complexity between different algorithm families
  - e.g. kNN vs neural networks
- Given an algorithm family, two main factors matter:
  - The number of parameters
  - The values taken by each parameter

$d + 1$



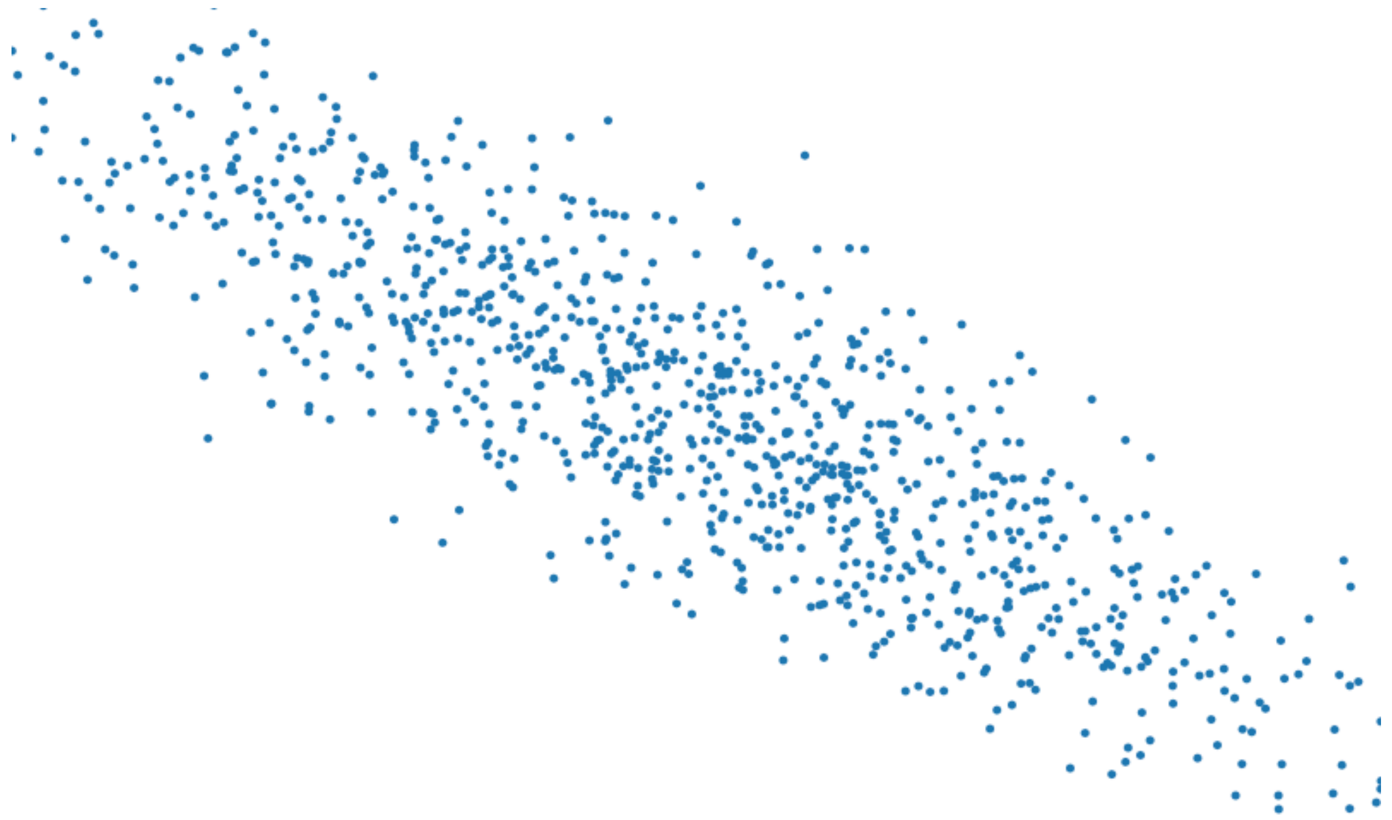
$(d + 1)m + (m + 1)k$

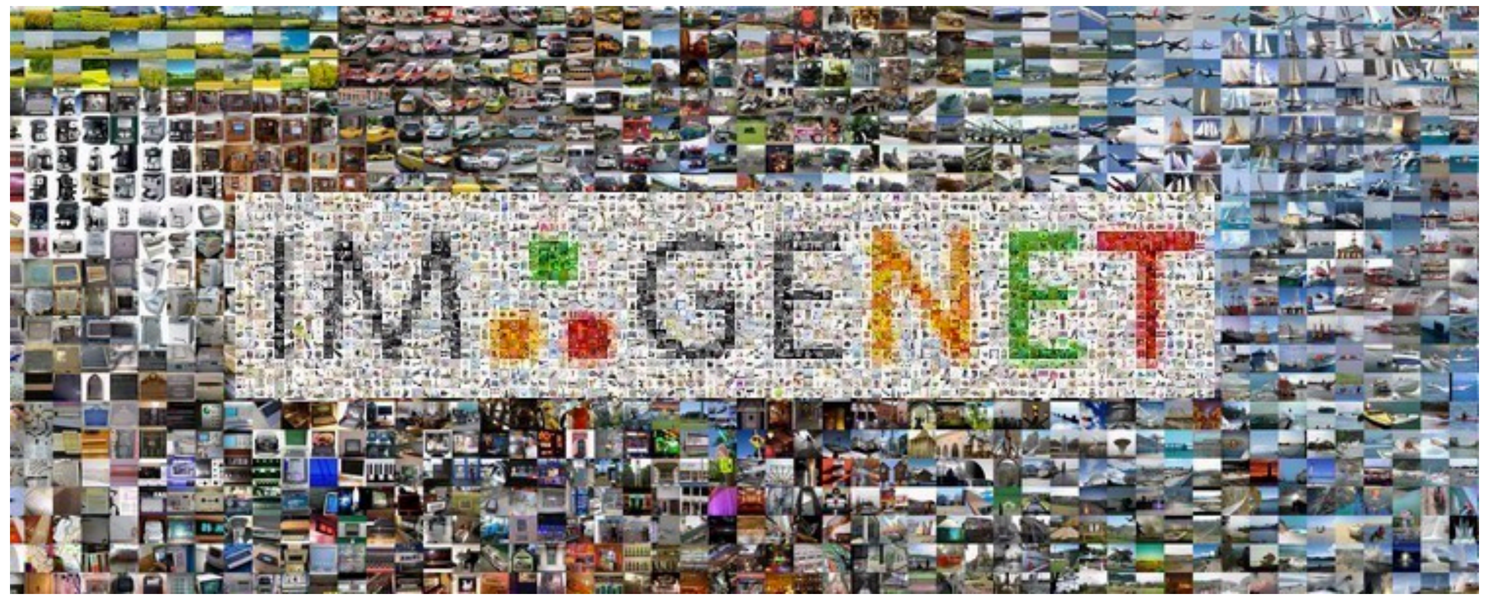


$W \in [-100, 100]$ $W \in [-10, +10]$

Data Complexity

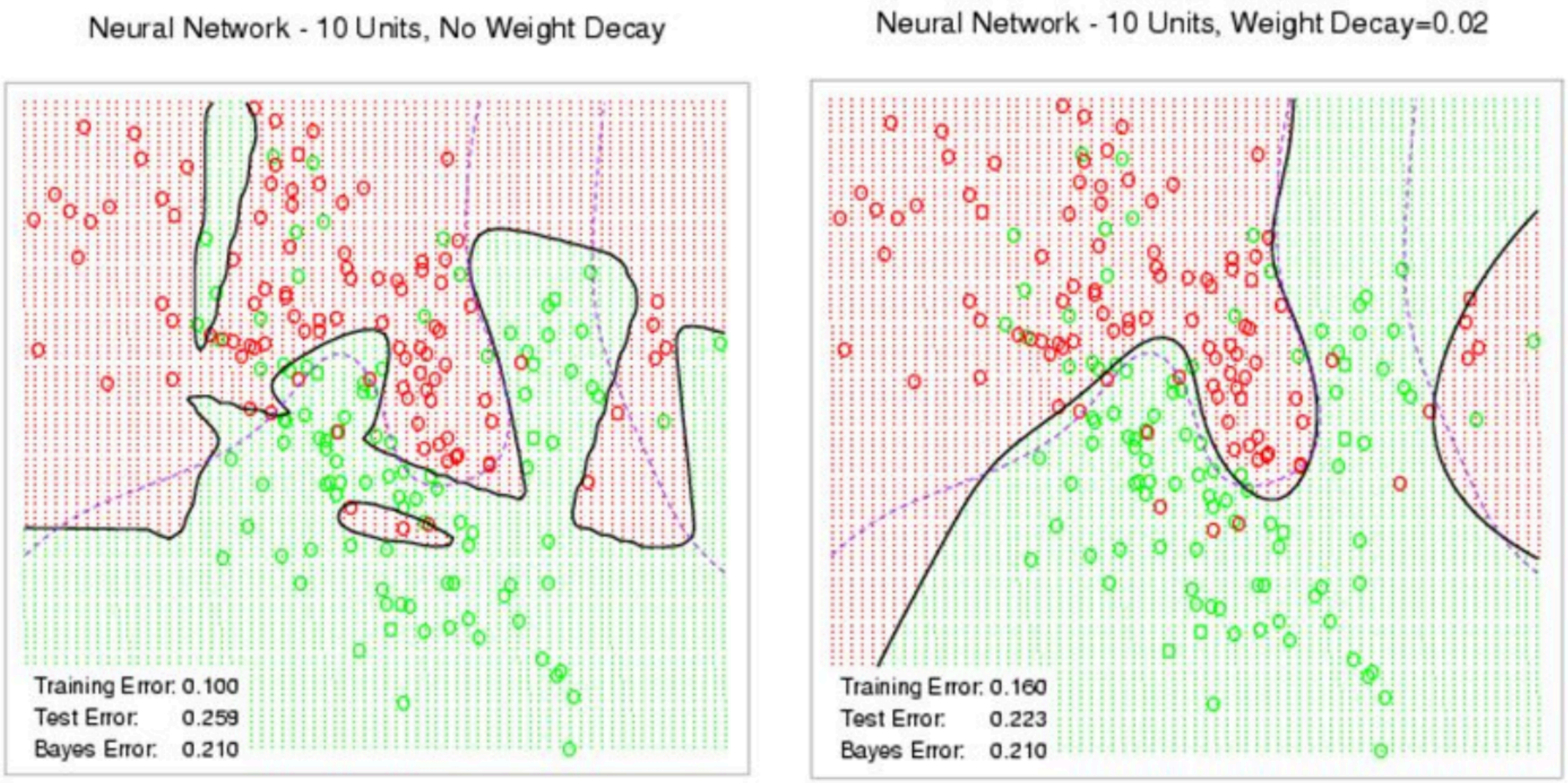
- Multiple factors matters
  - # of data points
  - # of features in each data point
  - time/space structure
  - # of classes





How to regularize the model for better generalization?

Weight Decay

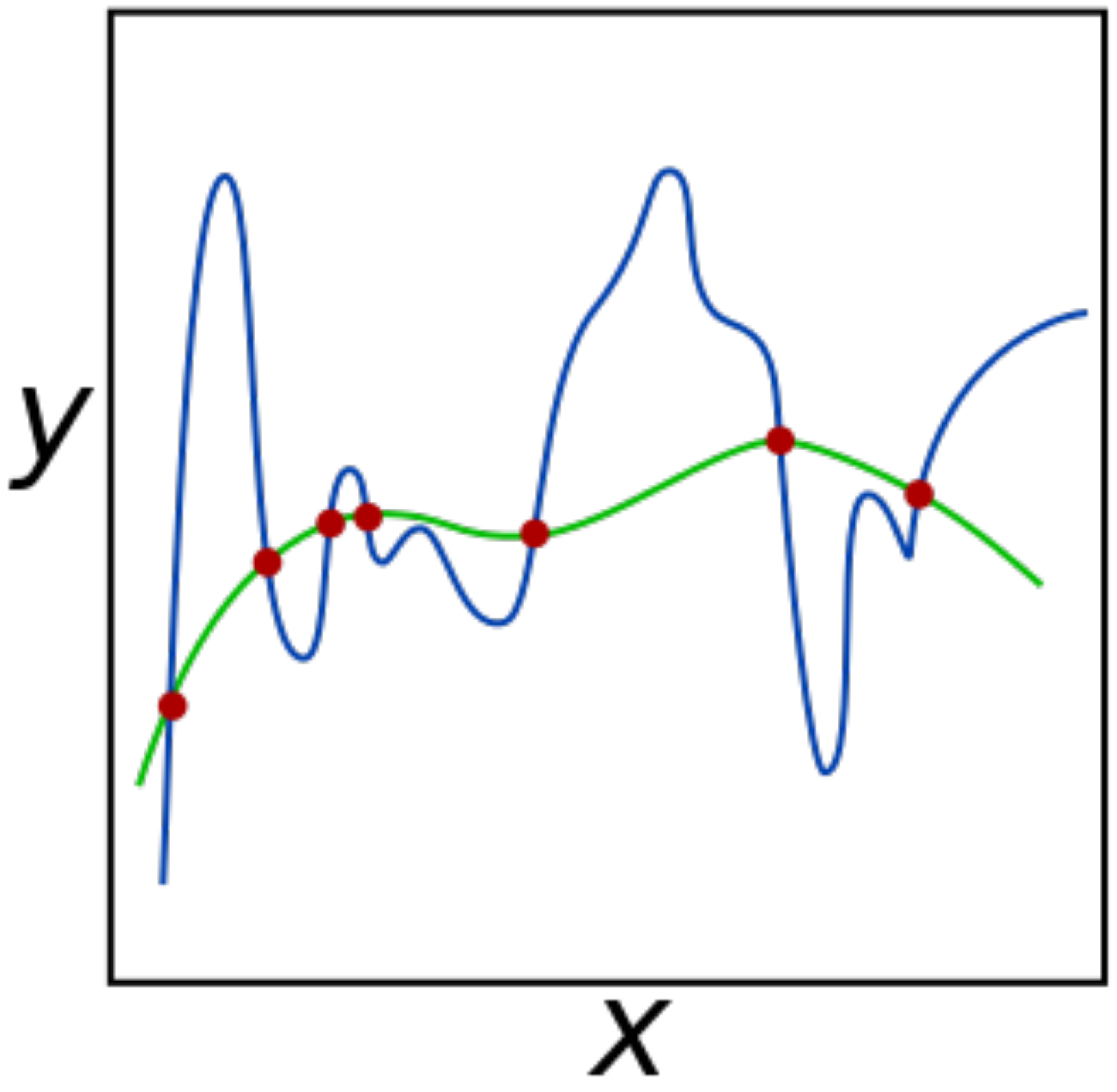


Squared Norm Regularization as Hard Constraint

- Reduce model complexity by limiting value range of weights

$$\min \ \ell(\mathbf{w}, b) \quad \text{subject to} \quad \boxed{\|\mathbf{w}\|^2 \leq \theta}$$

- Often do not regularize bias *b*
  - Doing or not doing has little difference in practice
- A small θ means more regularization



Squared Norm Regularization as Soft Constraint

- We can rewrite the hard constraint version as

$$\min \ \ell(\mathbf{w}, b) + \frac{\lambda}{2}\|\mathbf{w}\|^2$$

penality / regularization
term.

Squared Norm Regularization as Soft Constraint

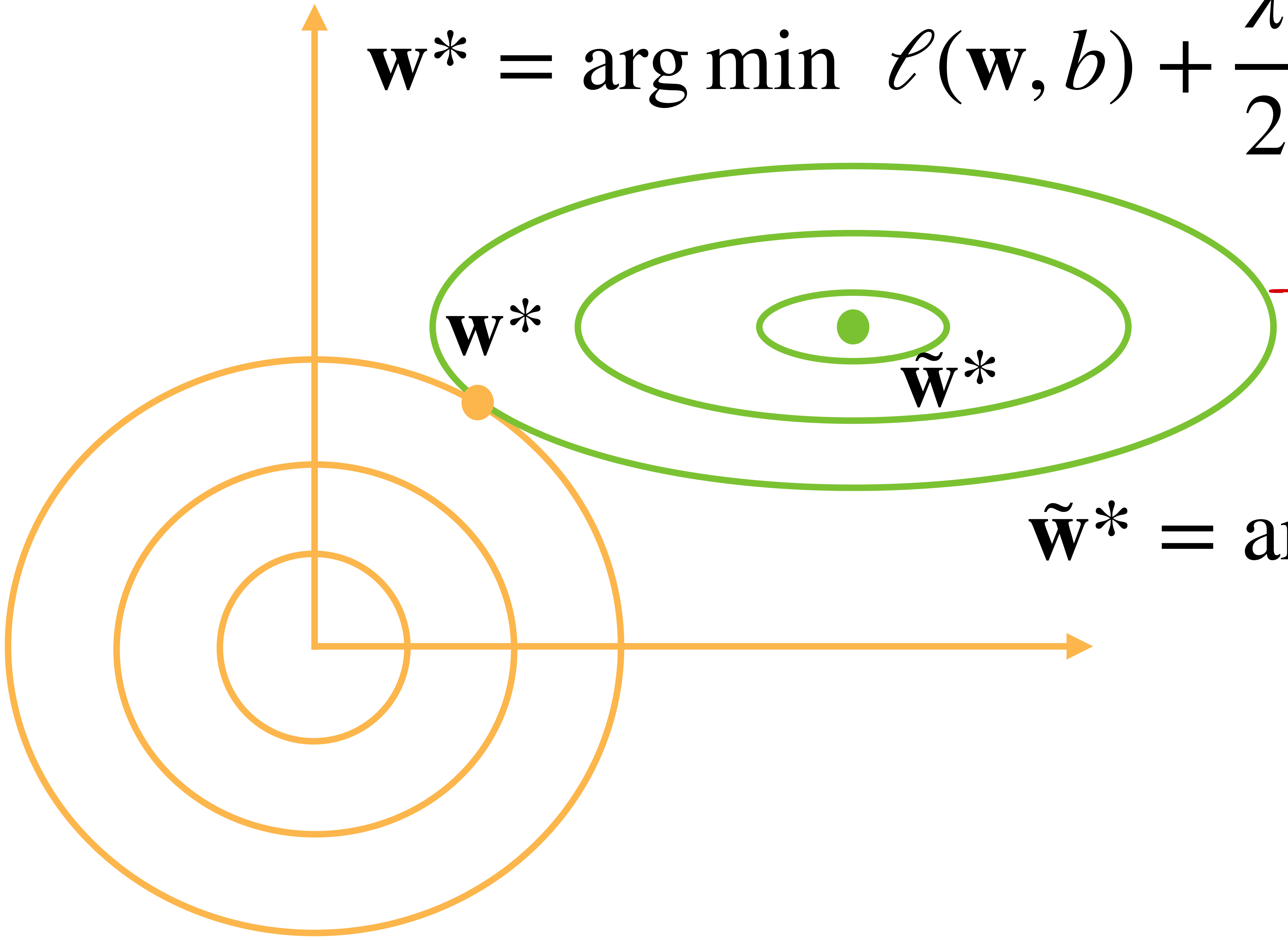
- We can rewrite the hard constraint version as

$$\min \ \ell(\mathbf{w}, b) + \frac{\lambda}{2}\|\mathbf{w}\|^2$$

  - Hyper-parameter λ controls regularization importance
  - $\lambda = 0$: no effect
  - $\lambda \to \infty, \mathbf{w}^* \to \mathbf{0}$

Illustrate the Effect on Optimal Solutions

$$\mathbf{w}^* = \arg\min \ \ell(\mathbf{w}, b) + \frac{\lambda}{2}\|\mathbf{w}\|^2$$



→ contour for $\ell(w, b)$.

$$\tilde{\mathbf{w}}^* = \arg\min \ \ell(\mathbf{w}, b)$$

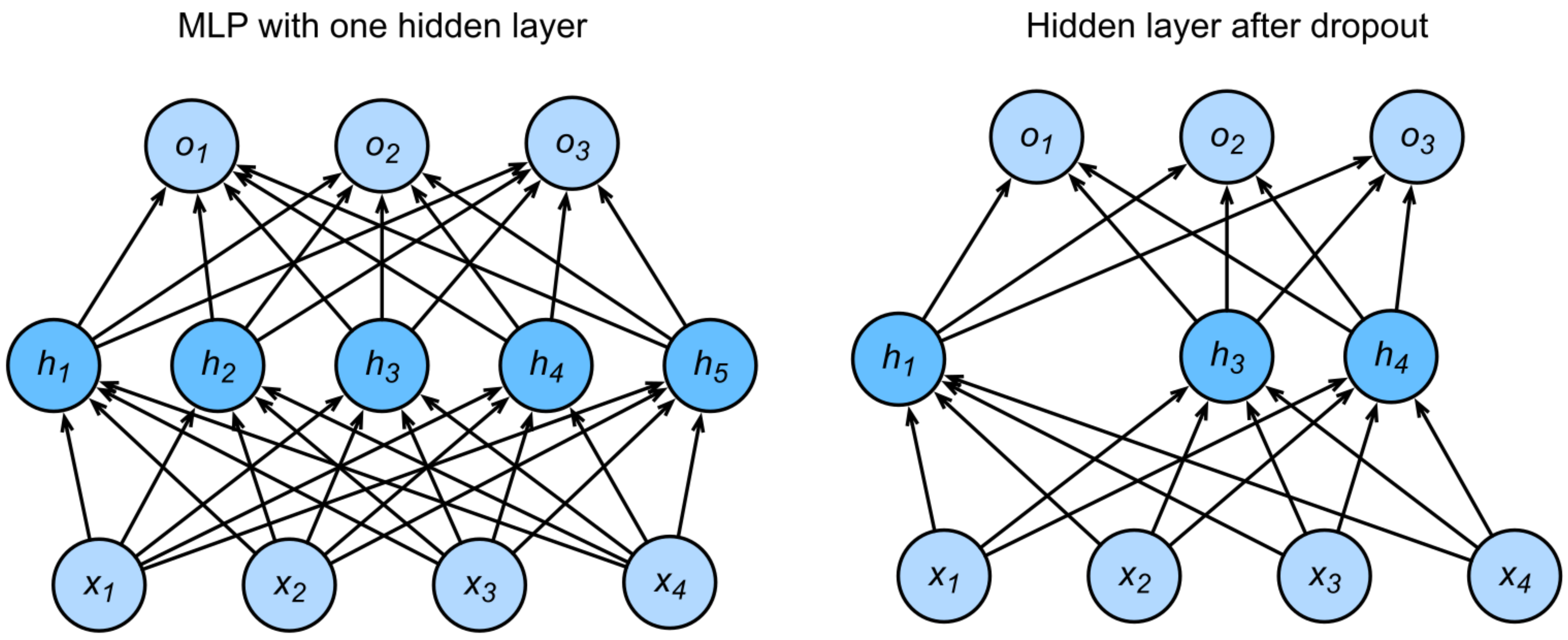
Dropout

Hinton et al.

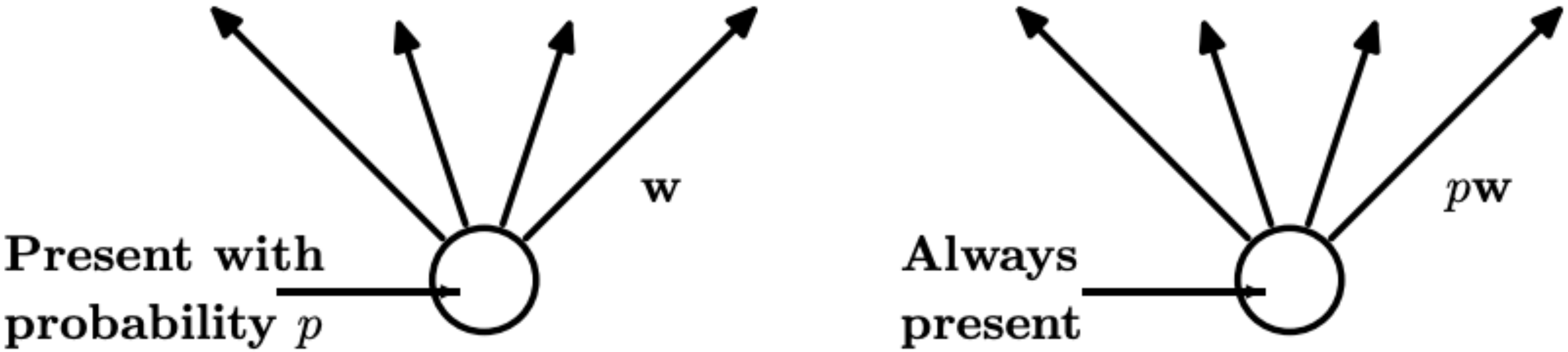
Apply Dropout

- Often apply dropout on the output of hidden fully-connected layers

$$\mathbf{h} = \sigma(\mathbf{W}_1\mathbf{x} + \mathbf{b}_1)$$
$$\mathbf{h}' = \text{dropout}(\mathbf{h})$$
$$\mathbf{o} = \mathbf{W}_2\mathbf{h}' + \mathbf{b}_2$$
$$\mathbf{y} = \text{softmax}(o)$$



Dropout



(a) At training time (b) At test time

Figure 2: **Left**: A unit at training time that is present with probability p and is connected to units in the next layer with weights $\mathbf{w}$. **Right**: At test time, the unit is always present and the weights are multiplied by p. The output at test time is same as the expected output at training time.

Dropout

Hinton et al.

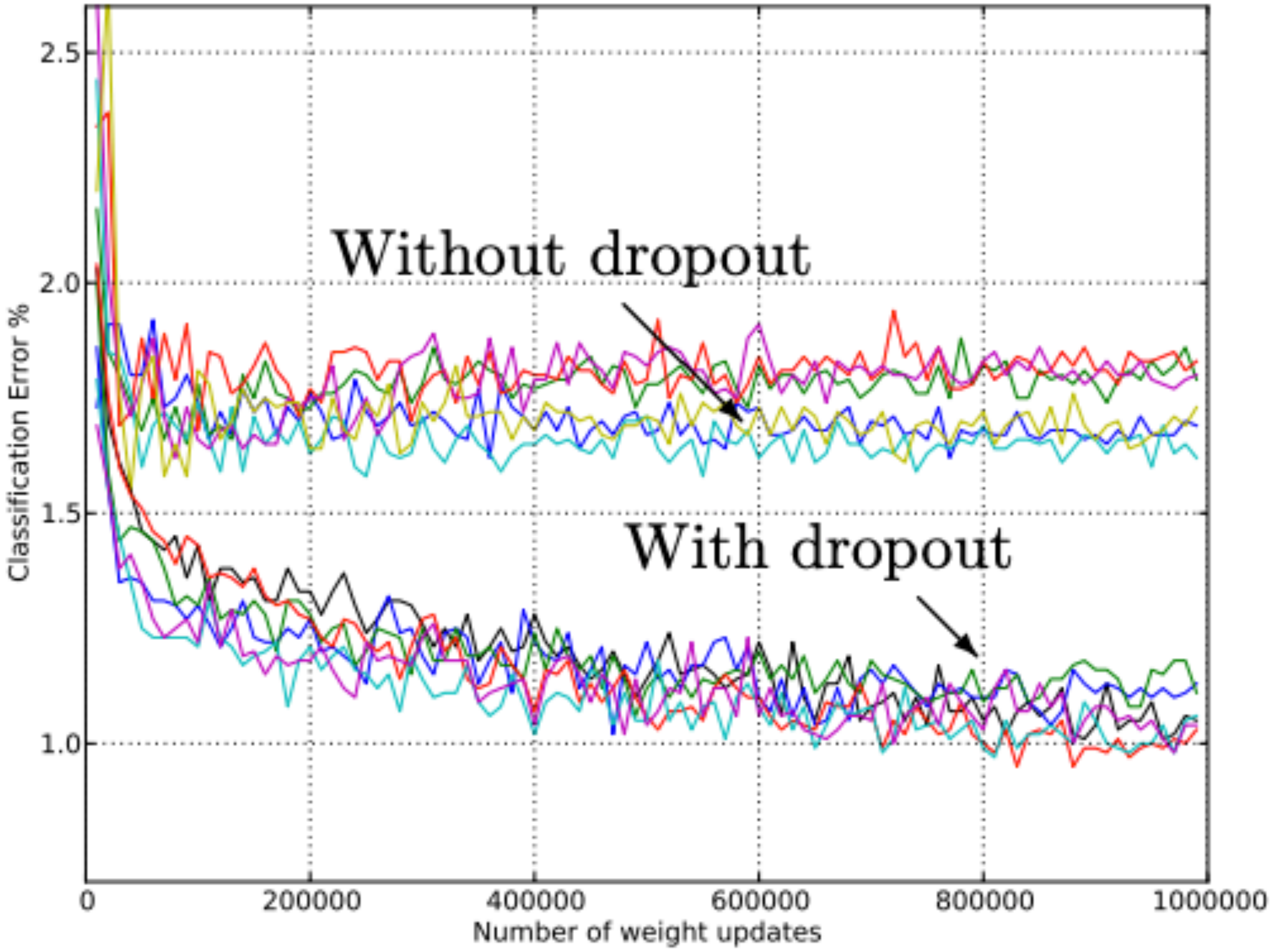


Figure 4: Test error for different architectures with and without dropout. The networks have 2 to 4 hidden layers each with 1024 to 2048 units.

What we've learned today…

- Deep neural networks
  - Computational graph (forward and backward propagation)
- Numerical stability in training
  - Gradient vanishing/exploding
- Generalization and regularization
  - Overfitting, underfitting
  - Weight decay and dropout

Thanks!

Based on slides from Sharon Li, Xiaojin (Jerry) Zhu, Yingyu Liang, Yin Li (CS540@UW-Madison) and Alex Smola: https://courses.d2l.ai/berkeley-stat-157/units/mlp.html