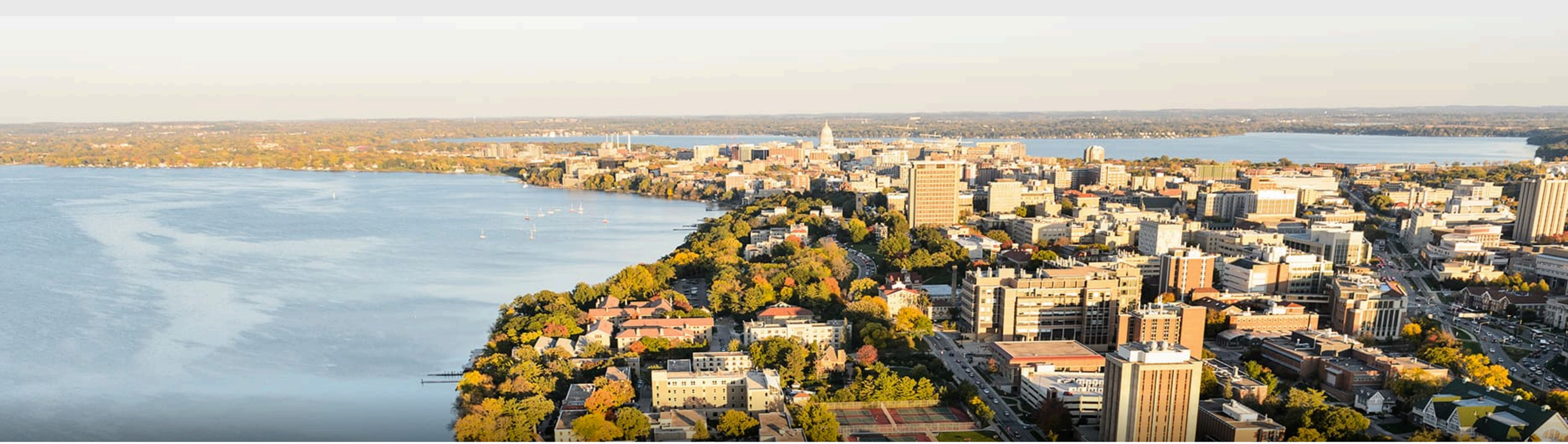


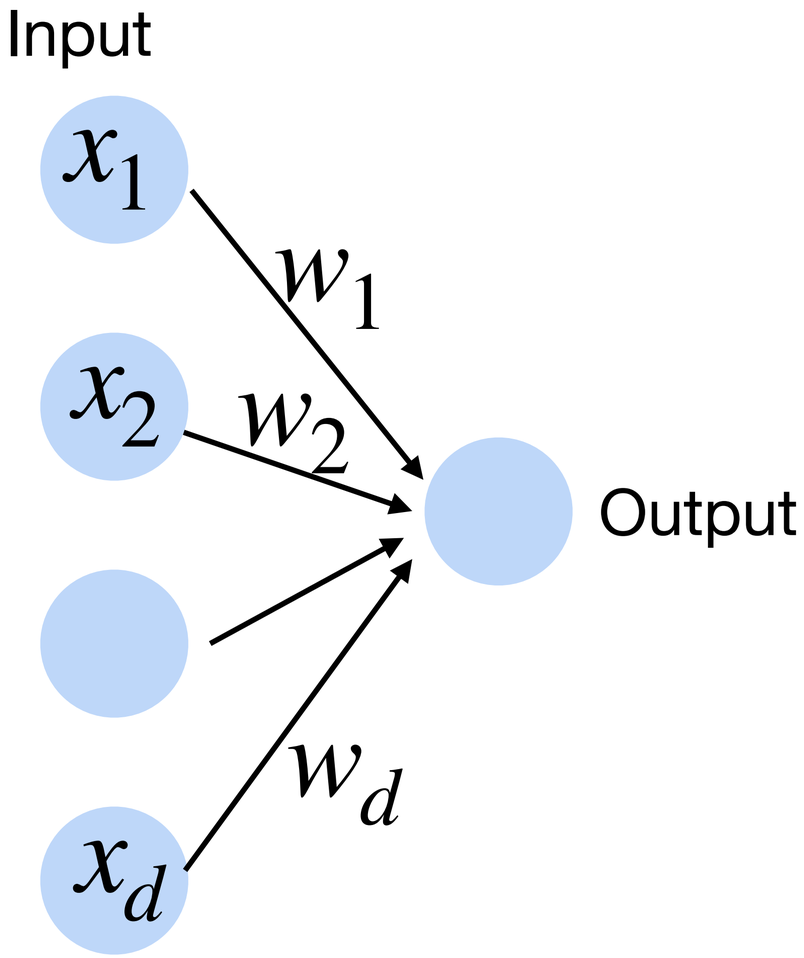
CS 540 Introduction to Artificial Intelligence
Neural Networks (III)

Sharon Yixuan Li
University of Wisconsin-Madison

March 11, 2021

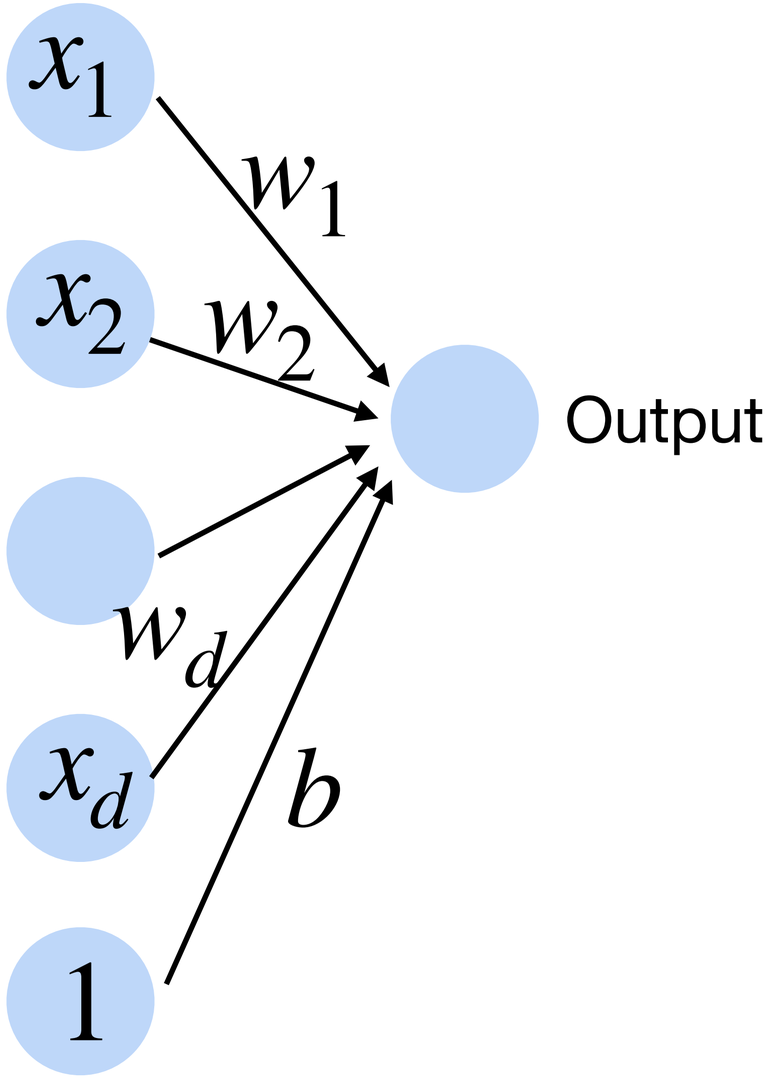
HW5 common confusion

$$f = \mathbf{w}^T\mathbf{x} + b$$



$$f = \mathbf{w}_{new}^T\mathbf{x}_{new}$$

$$\mathbf{x}_{new} = [\mathbf{x}, 1]$$



Logistics for the midterm exam

Logistics for the midterm exam:

1. The exam will be on Canvas. You will have a 24 hour period to start the midterm, but once you start, you will have a total of 1:45 to complete it. This is 90 minutes for the test plus a 15 minute grace period to handle uploads or any other tech issues (explained below).

2. You may only use the course materials from this semester's website (slides, quizzes, homework assignments) during the midterm. You may not use any other resources. You may not use search engines, collaboration tools or software, or anything else of that nature.

3. You may use calculators.

4. You may not share or communicate the midterm questions during or after the exam. This policy is strict.

5. Question types: the questions involve either multiple choice (some easy, some hard) and written questions. You can expect to see around 10 multiple choice and up to 4-5 of the written questions. The written questions will allow you to show your work for partial credit. The multiple choice questions are answered in Canvas directly; there will be an upload box for you to upload your solutions to the written questions.

Please use this dummy quiz to make sure it works. Also, make sure you test your ability to scan and send in the written answers. Do this early!

6. The sample questions are attached below; the answers will be released on Friday to encourage you to try them on your own. These questions will give you a sense of how the midterm questions will be, but are not representative of the topics. Note that the topics are in the earlier post: https://piazza.com/class/kk1k70vbawp3ts?cid=458

7. If you have questions or concerns, get in touch with the instructors.

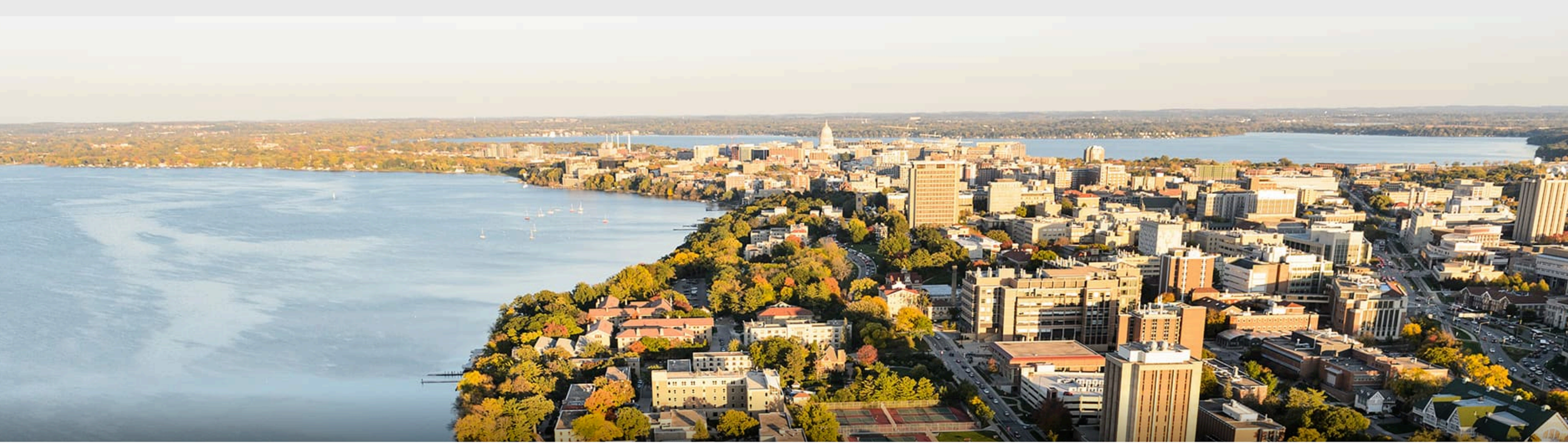
Midterm_Sample_Questions.pdf

#pin

announcements

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



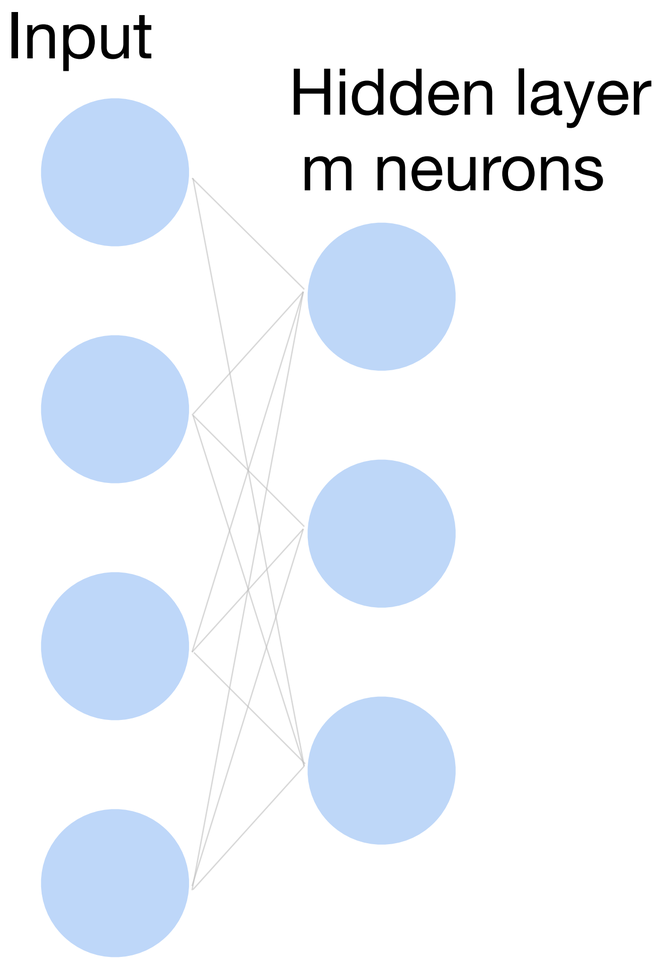
Part I: Neural Networks as a Computational Graph

Review: neural networks with one hidden layer

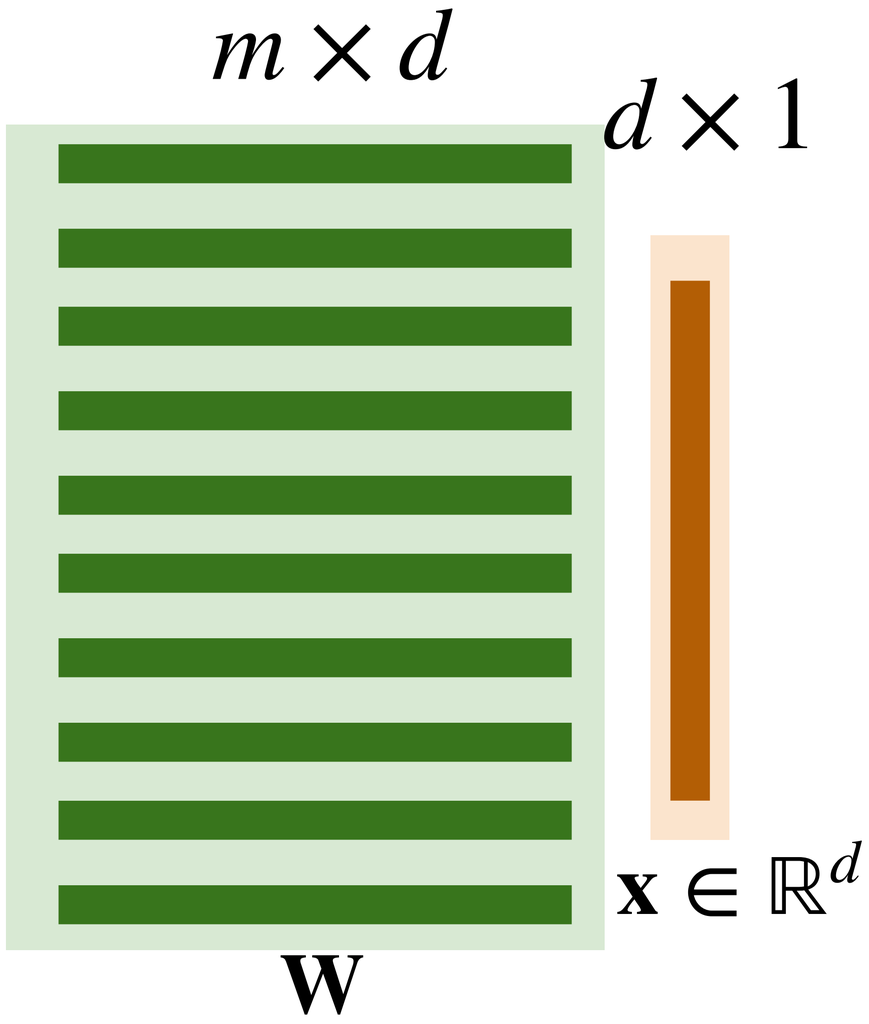
- Input $\mathbf{x} \in \mathbb{R}^d$
- Hidden $\mathbf{W}^{(1)} \in \mathbb{R}^{m \times d}, \mathbf{b} \in \mathbb{R}^m$
- Intermediate output

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

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

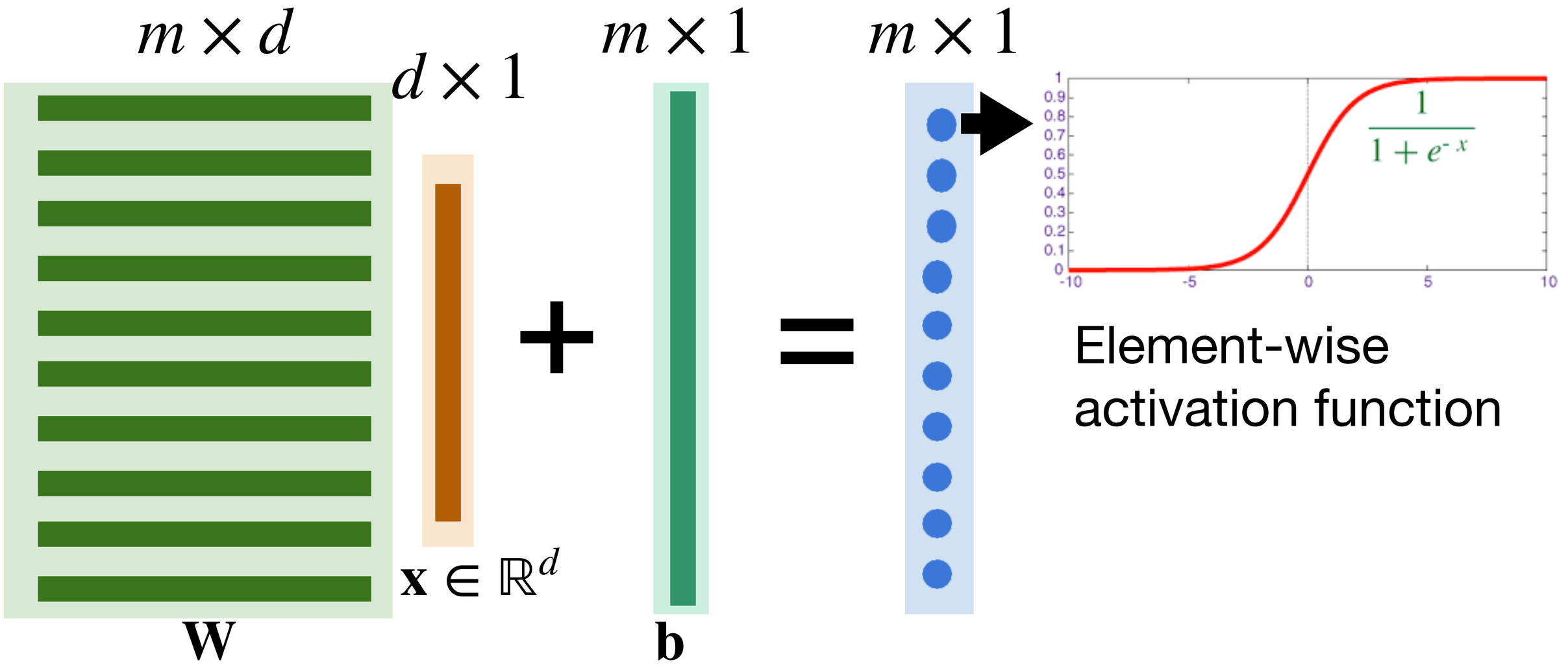


Review: neural networks with one hidden layer



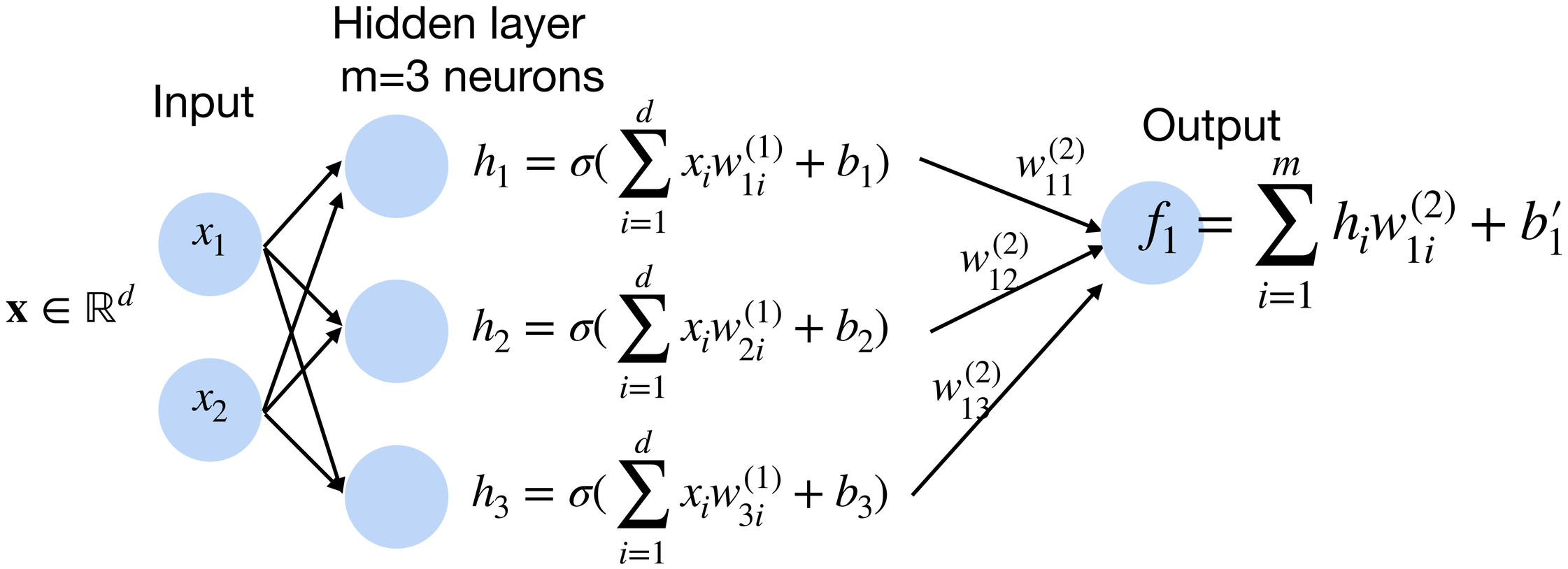
Review: neural networks with one hidden layer

Key elements: linear operations + Nonlinear activations



Review: Neural network for k-way classification

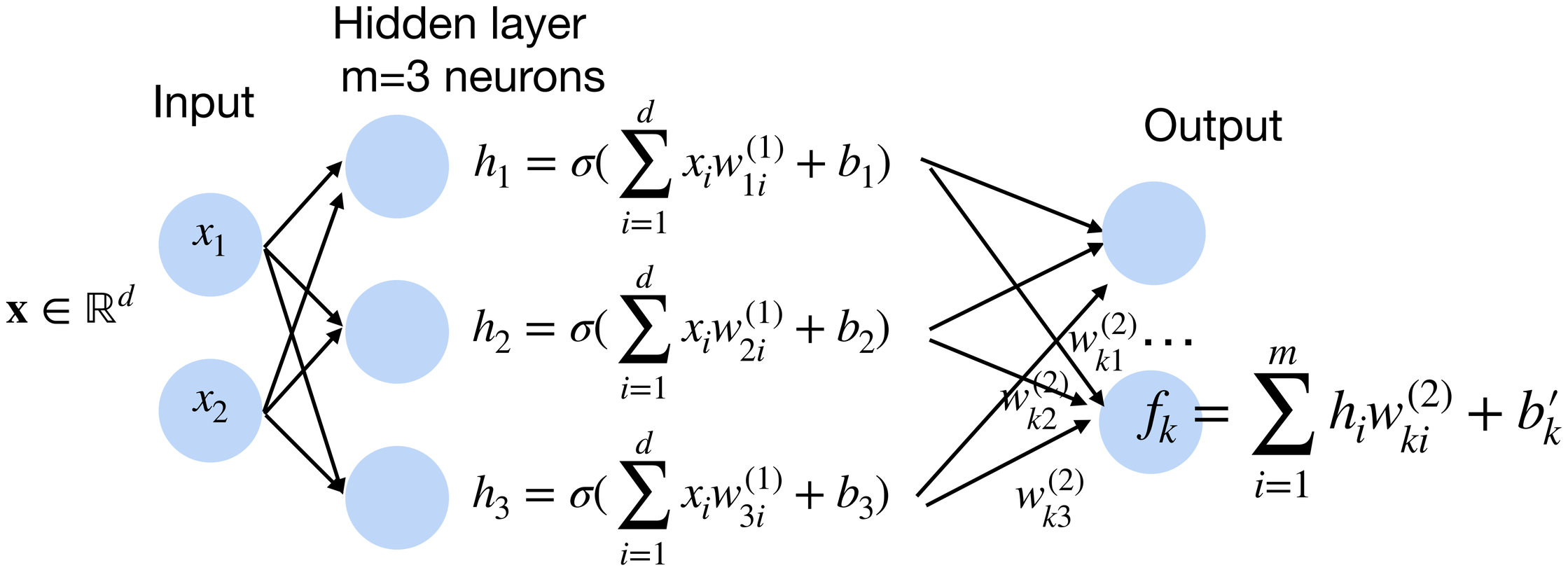
- K outputs in the final layer



Review: Neural network for k-way classification

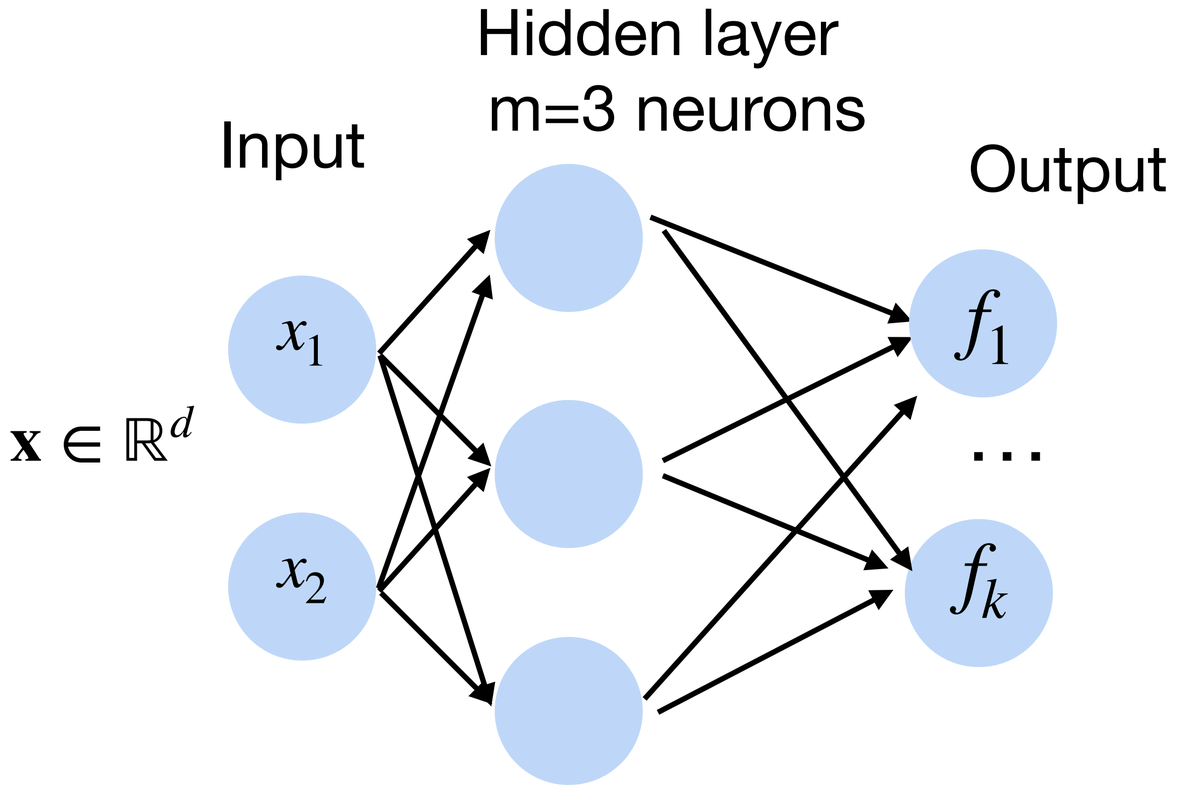
- K outputs units in the final layer

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



Review: Softmax

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

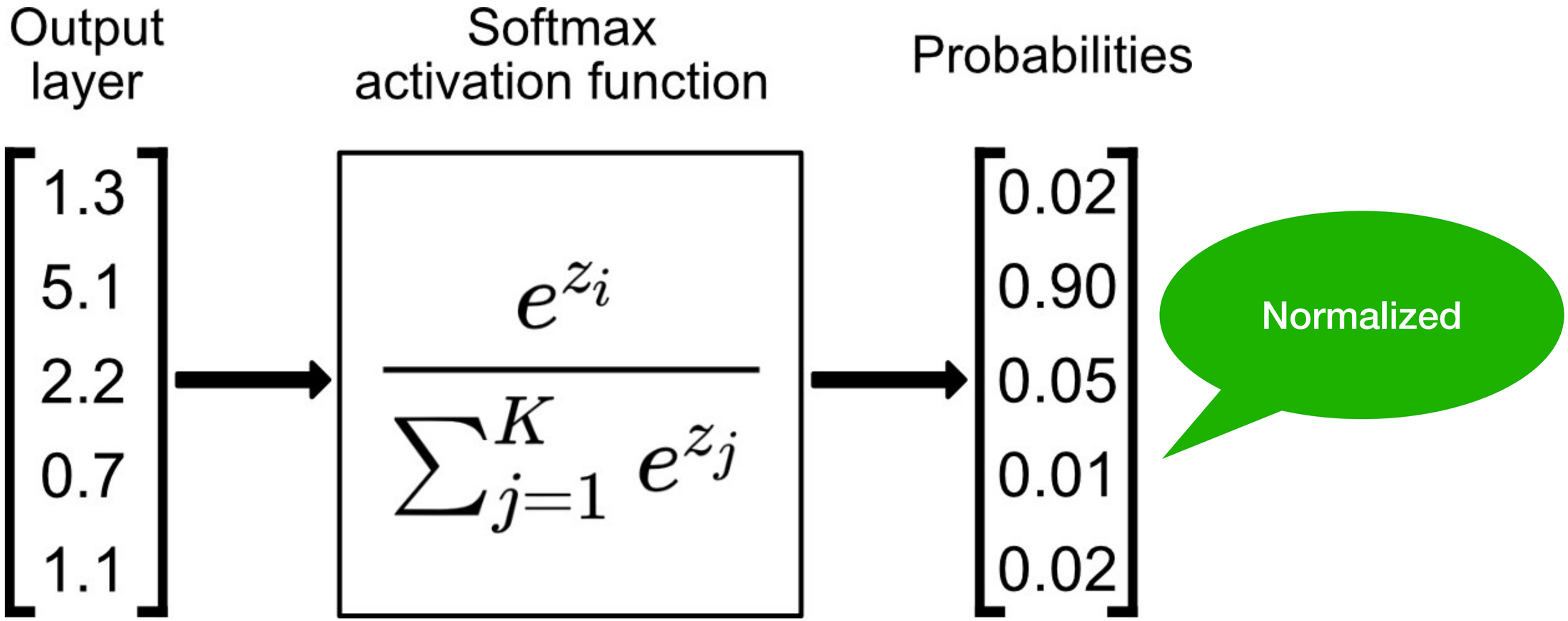


$$p(y \mid \mathbf{x}) = \text{softmax}(f)$$

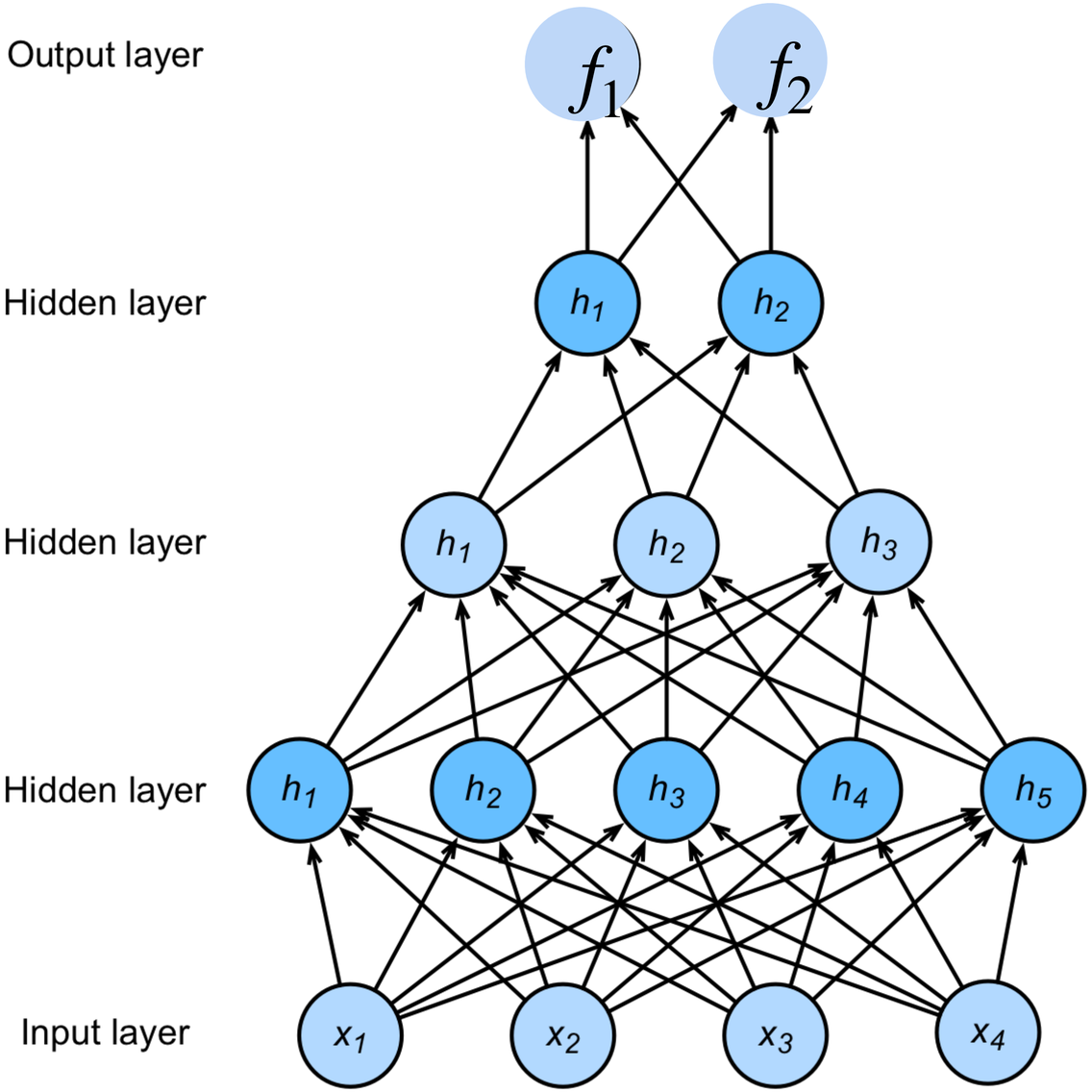
$$= \frac{\exp f_y(x)}{\sum_i^k \exp f_i(x)}$$

Softmax

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



Deep neural networks (DNNs)



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

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

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

$$\mathbf{f} = \mathbf{W}_4\mathbf{h}_3 + \mathbf{b}_4$$

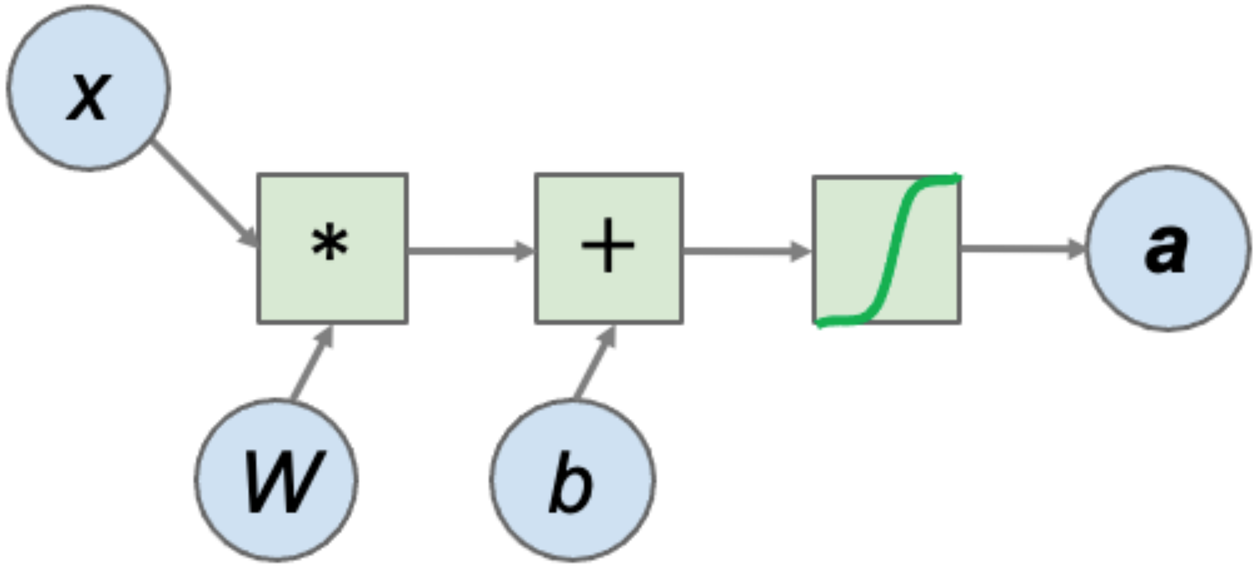
$$\mathbf{y} = \text{softmax}(\mathbf{f})$$

NNs are composition of nonlinear functions

Neural networks as variables + operations

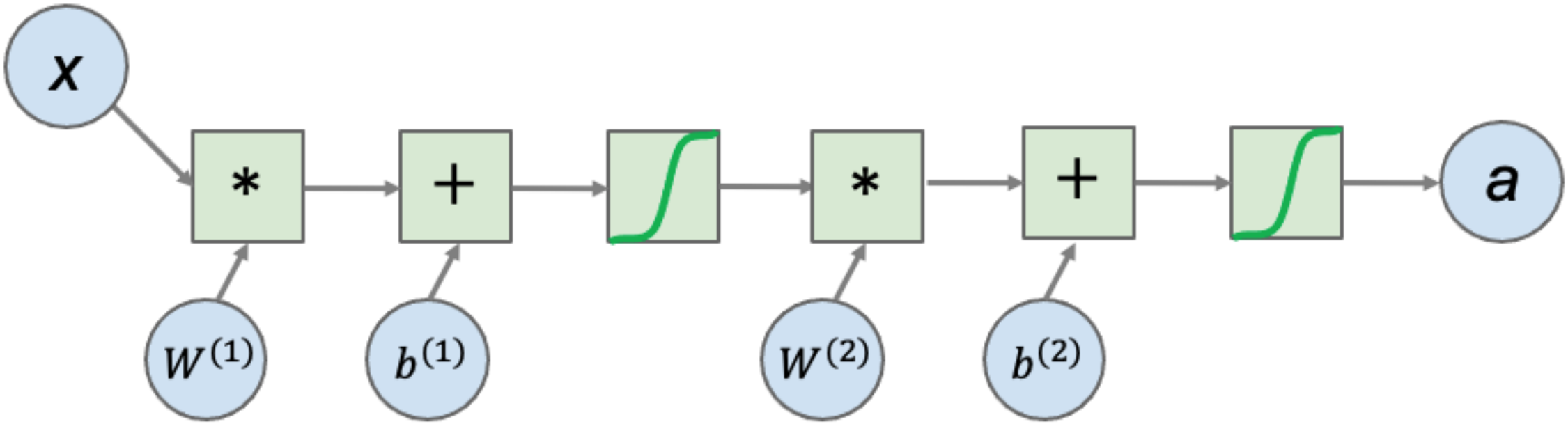
$$\mathbf{a} = \text{sigmoid}(\mathbf{W}\mathbf{x} + \mathbf{b})$$

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



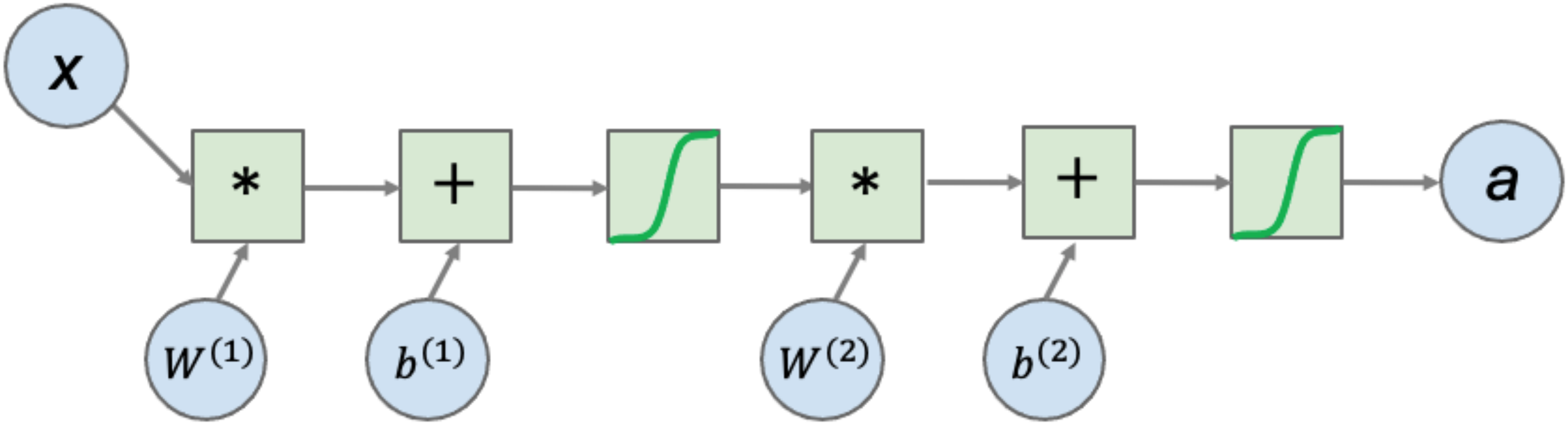
Neural networks as a computational graph

- A two-layer neural network



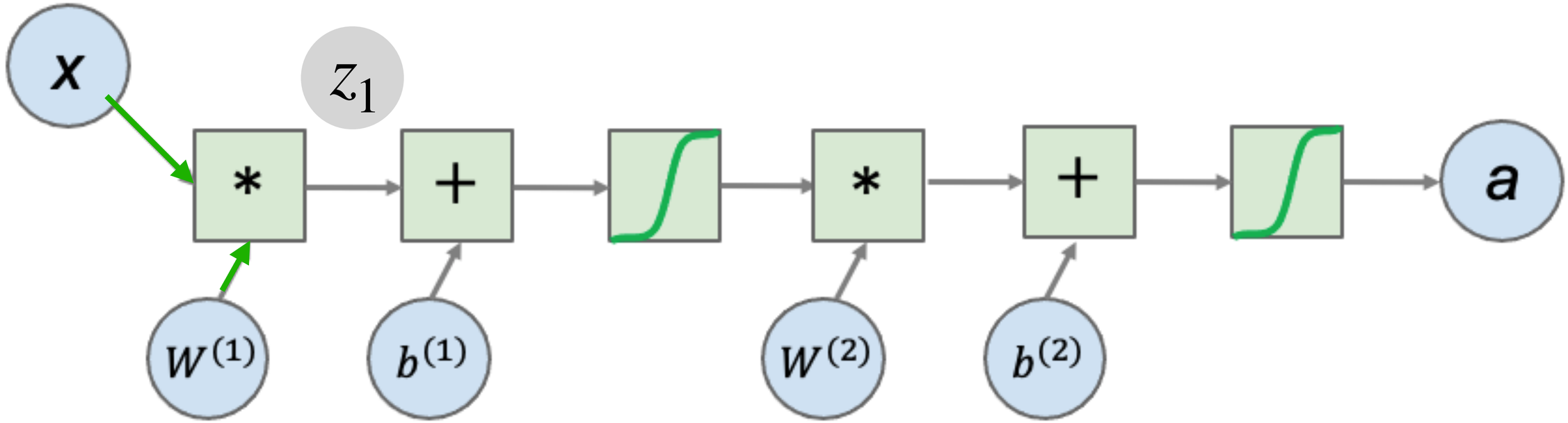
Neural networks as a computational graph

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



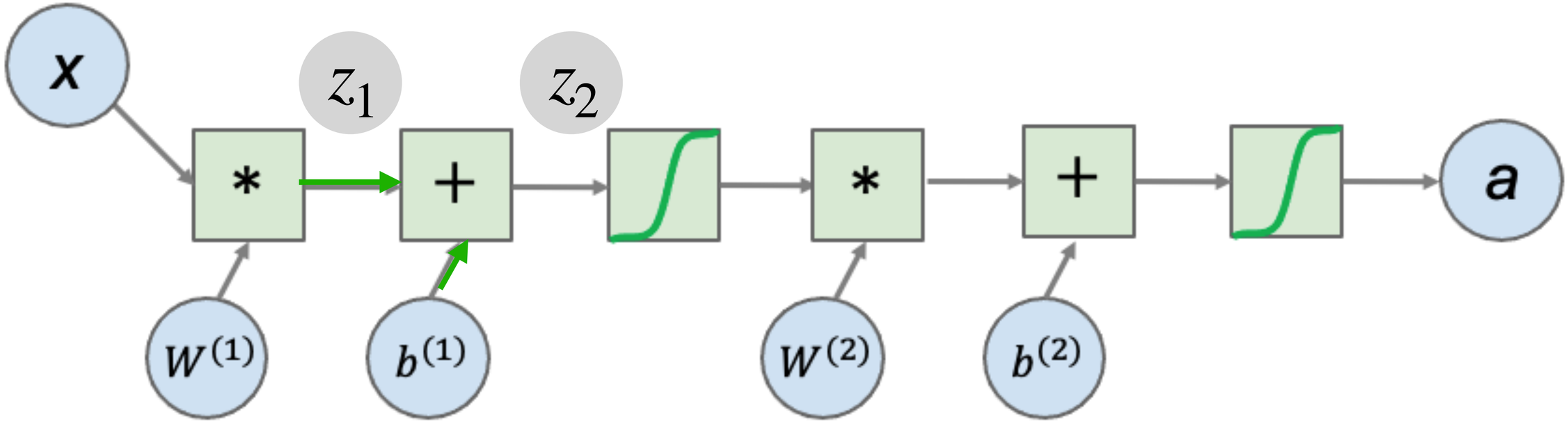
Neural networks: forward propagation

- A two-layer neural network
- Intermediate variables Z



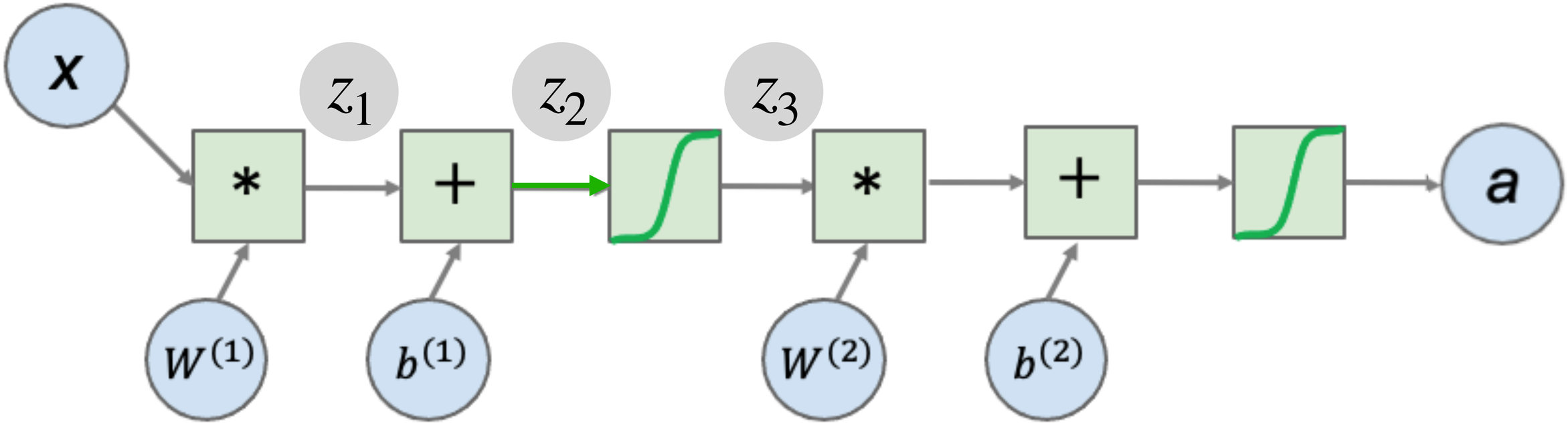
Neural networks: forward propagation

- A two-layer neural network
- Intermediate variables Z



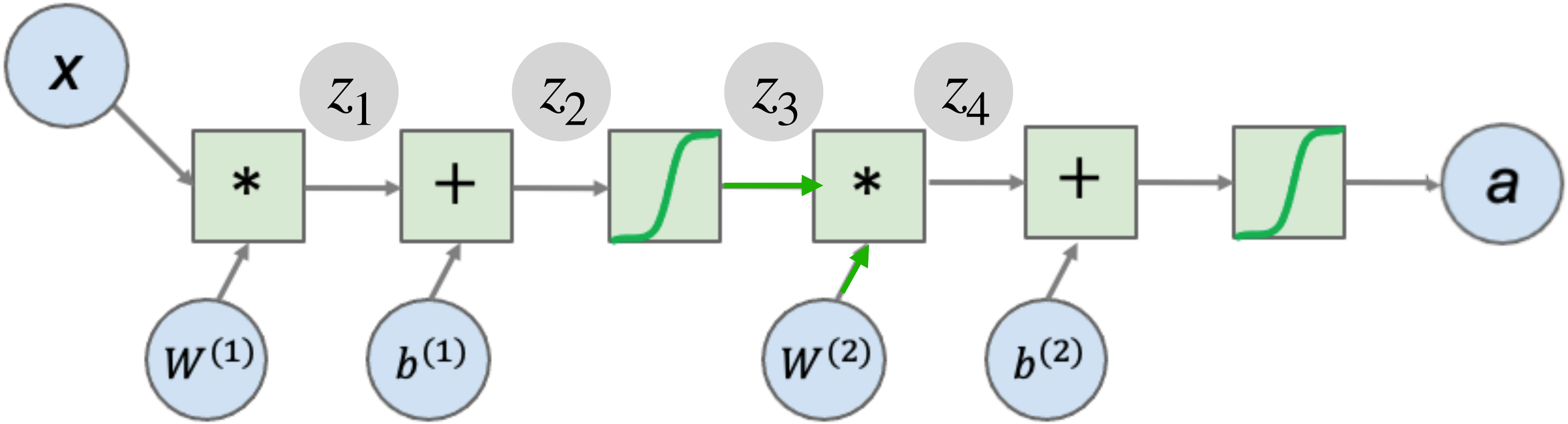
Neural networks: forward propagation

- A two-layer neural network
- Intermediate variables Z



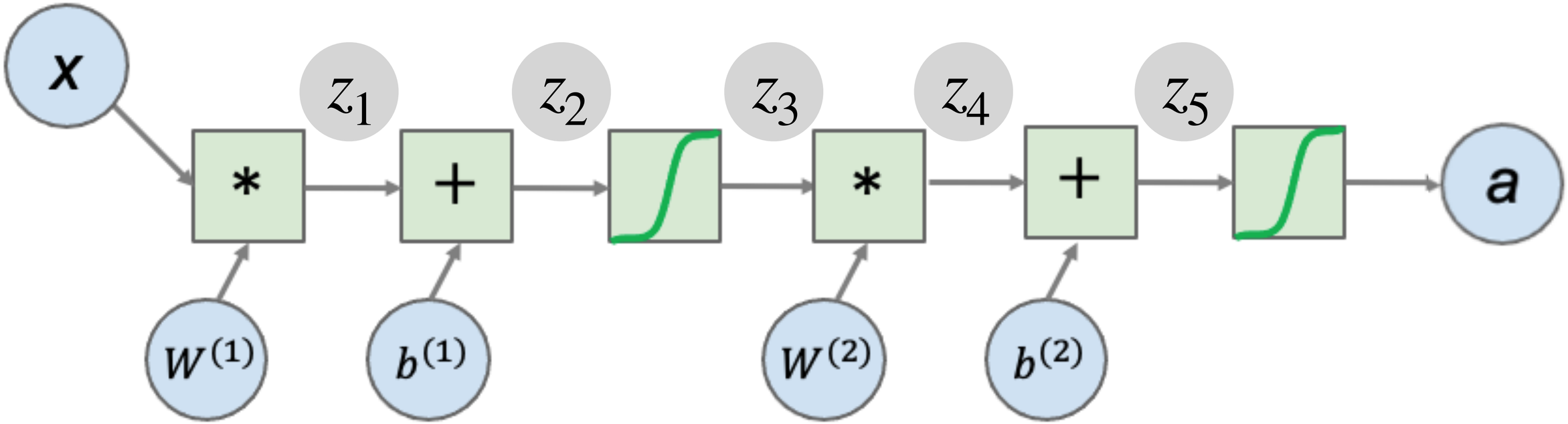
Neural networks: forward propagation

- A two-layer neural network
- Intermediate variables Z



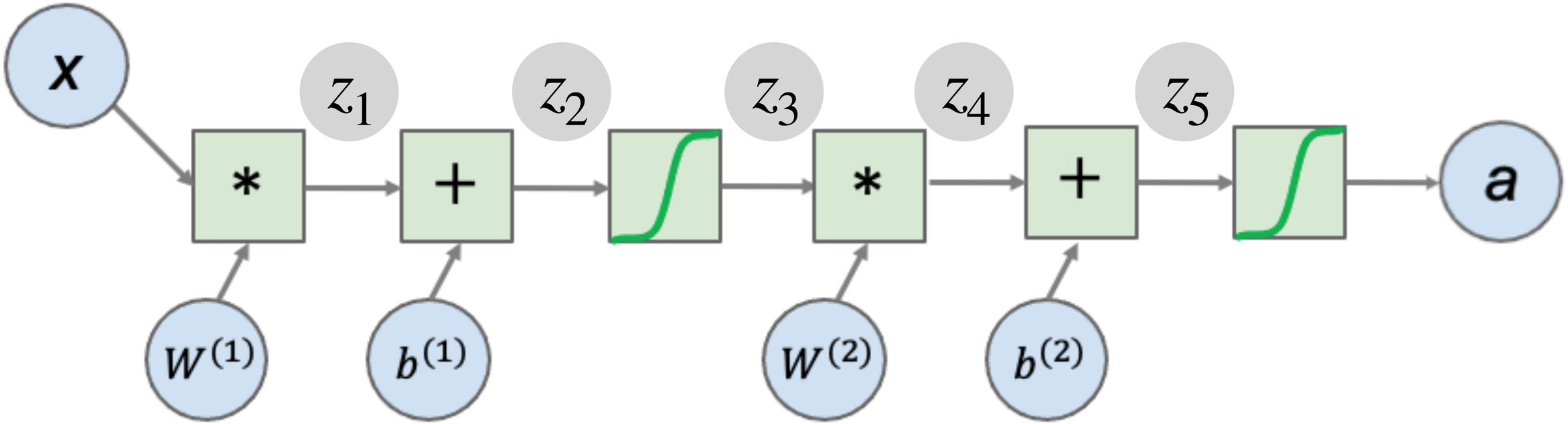
Neural networks: forward propagation

- A two-layer neural network
- Intermediate variables Z



Neural networks: backward propagation

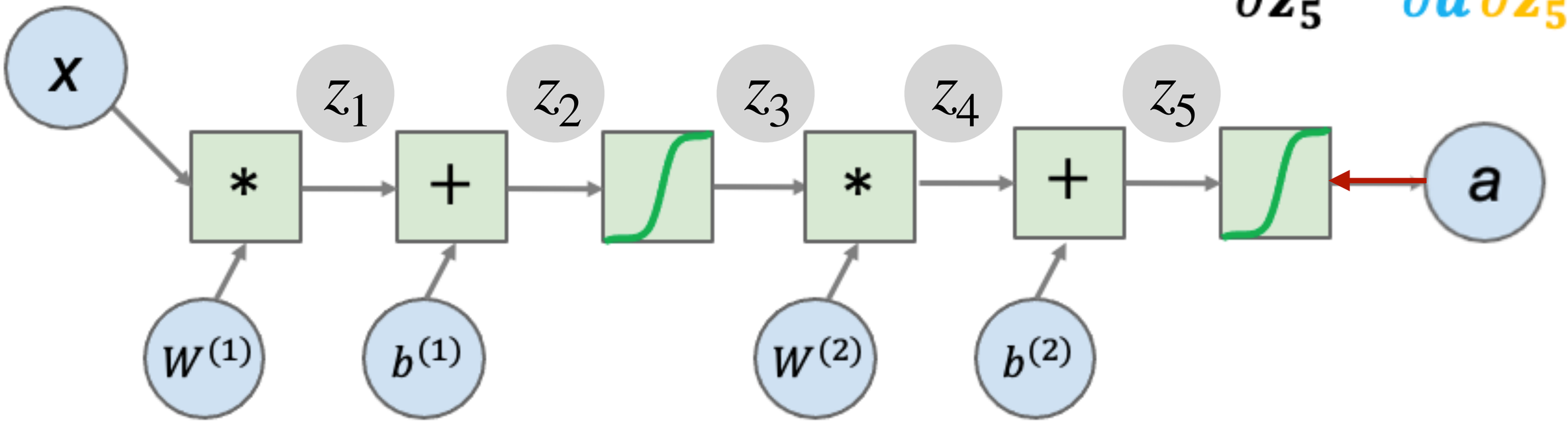
- A two-layer neural network
- Assuming forward propagation is done
- Minimize a **loss function** L



Neural networks: backward propagation

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

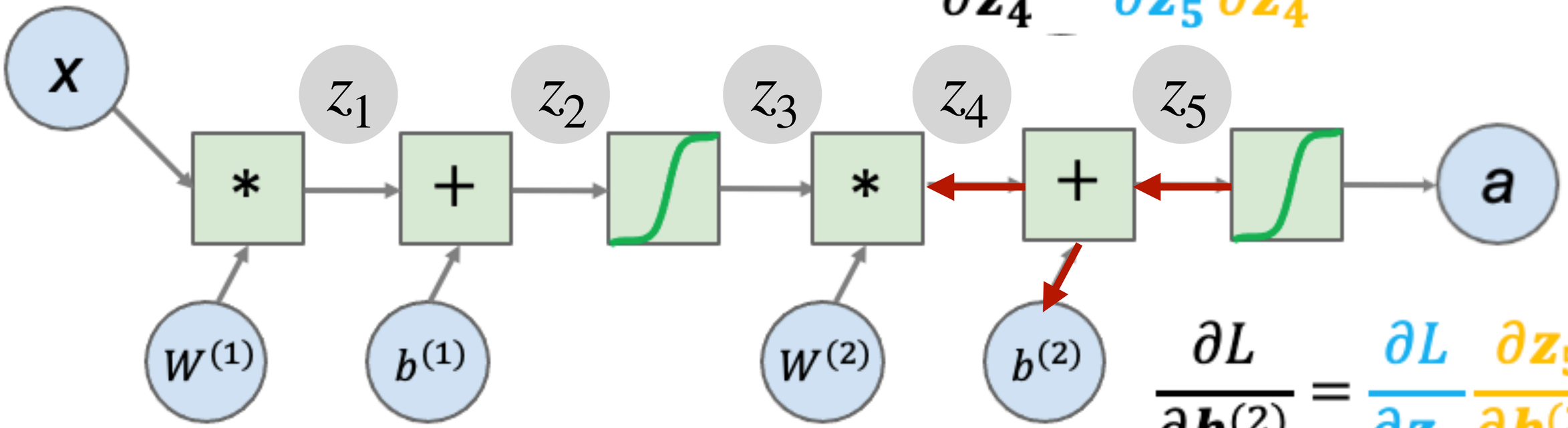
$$\frac{\partial L}{\partial z_5} = \frac{\partial L}{\partial a}\frac{\partial a}{\partial z_5}$$



Neural networks: backward propagation

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

$$\frac{\partial L}{\partial z_4} = \frac{\partial L}{\partial z_5}\frac{\partial z_5}{\partial z_4}$$

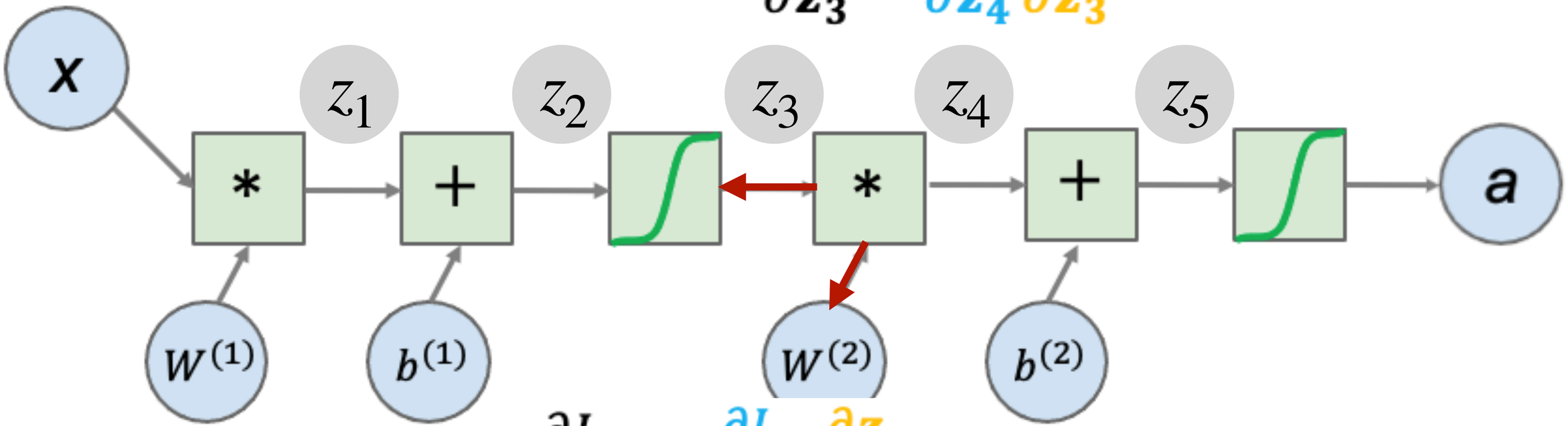


$$\frac{\partial L}{\partial b^{(2)}} = \frac{\partial L}{\partial z_5}\frac{\partial z_5}{\partial b^{(2)}}$$

Neural networks: backward propagation

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

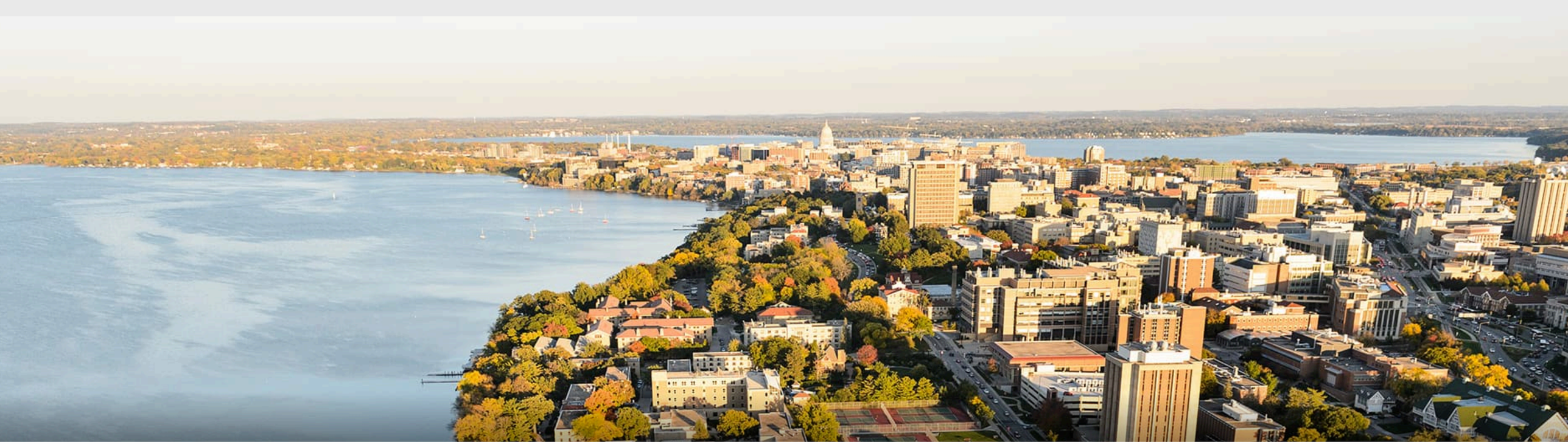
$$\frac{\partial L}{\partial z_3} = \frac{\partial L}{\partial z_4}\frac{\partial z_4}{\partial z_3}$$



$$\frac{\partial L}{\partial W^{(2)}} = \frac{\partial L}{\partial z_4}\frac{\partial z_4}{\partial W^{(2)}}$$

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**



Part II: Numerical Stability

Gradients for Neural Networks

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

$$\frac{\partial \ell}{\partial \mathbf{W}^t} = \frac{\partial \ell}{\partial \mathbf{h}^d} \underbrace{\frac{\partial \mathbf{h}^d}{\partial \mathbf{h}^{d-1}} \cdots \frac{\partial \mathbf{h}^{t+1}}{\partial \mathbf{h}^t}} \frac{\partial \mathbf{h}^t}{\partial \mathbf{W}^t}$$

Multiplication of *many* matrices

Wikipedia

Two Issues for Deep Neural Networks

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

Gradient Exploding

$$1.5^{100} \approx 4 \times 10^{17}$$

Gradient Vanishing

$$0.8^{100} \approx 2 \times 10^{-10}$$

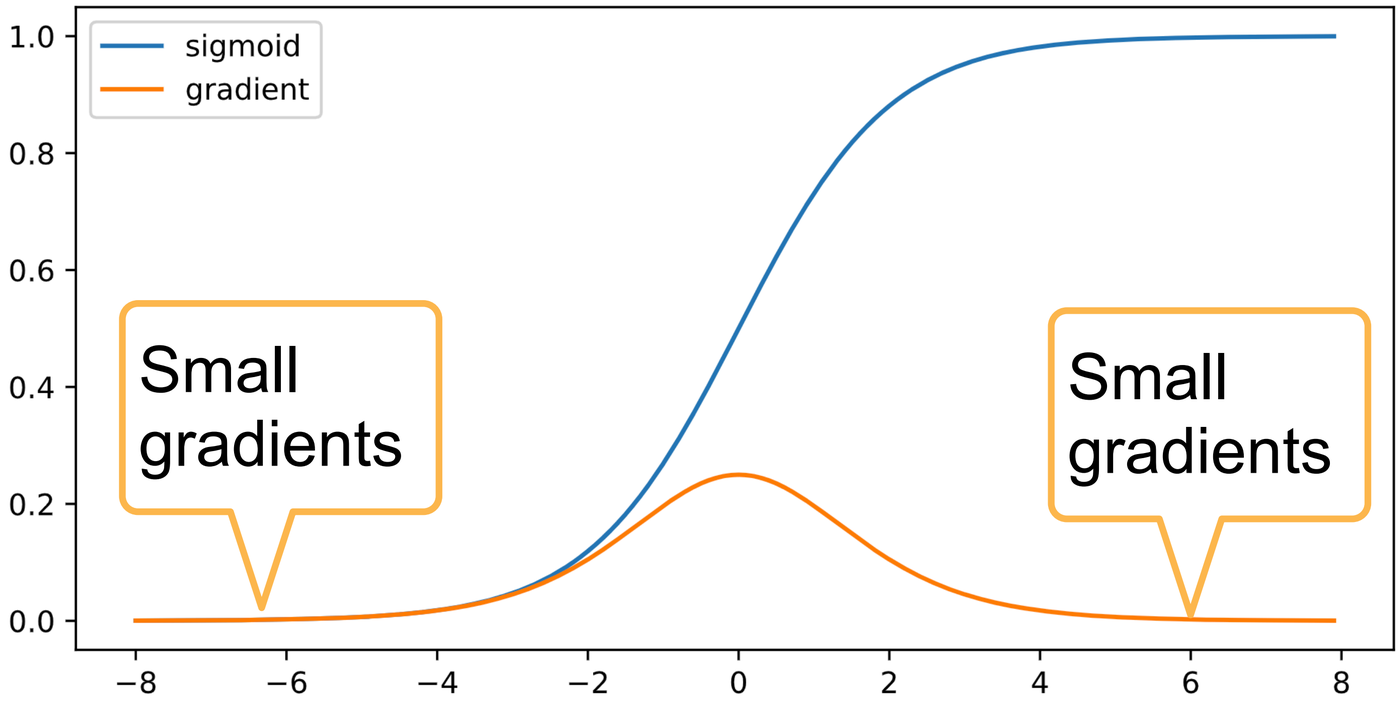
Issues with Gradient Exploding

- 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

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



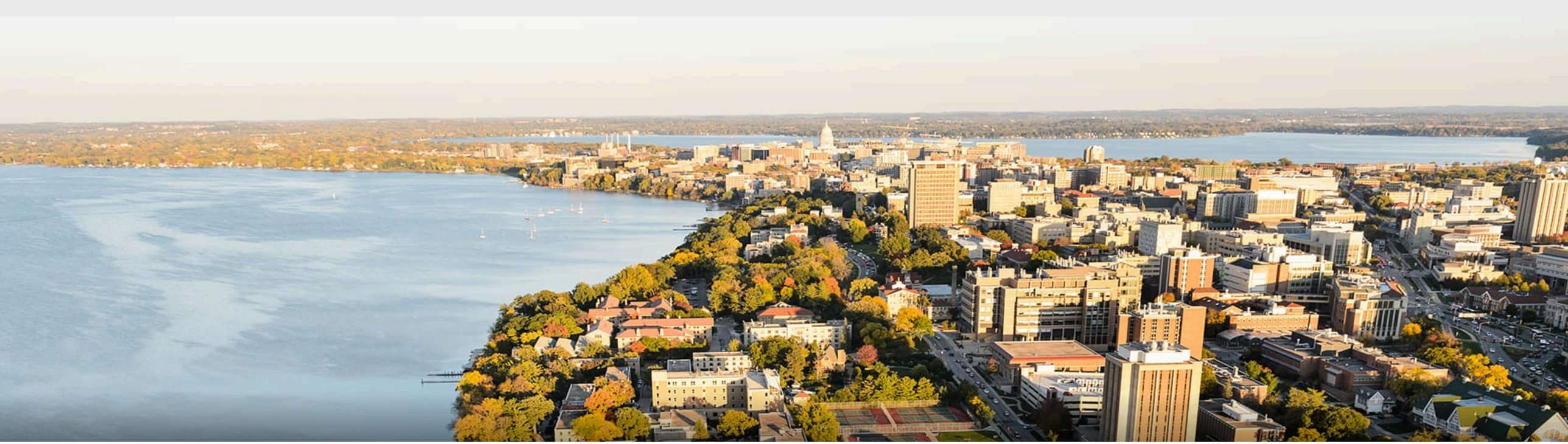
Issues with Gradient Vanishing

- 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

- Goal: make sure gradient values are in a proper range
  - E.g. in [1e-6, 1e3]
- Multiplication -> plus
  - Architecture change (e.g., ResNet)
- Normalize
  - Batch Normalization, Gradient clipping
- Proper activation functions



Part III: Generalization & Regularization

How good are the models?

Training Error and Generalization Error

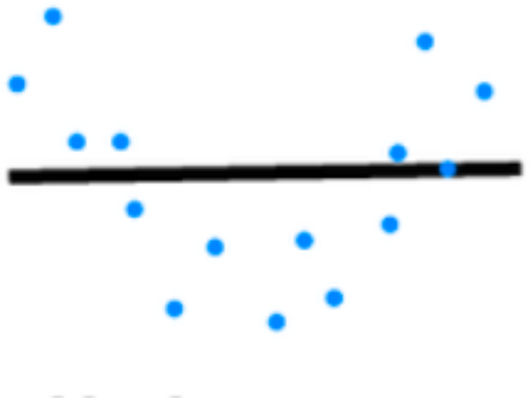
- Training error: model error on the training data
- **Generalization error**: model error on new data
- Example: practice a future exam with past exams
  - Doing well on past exams (training error) doesn’t guarantee a good score on the future exam (generalization error)

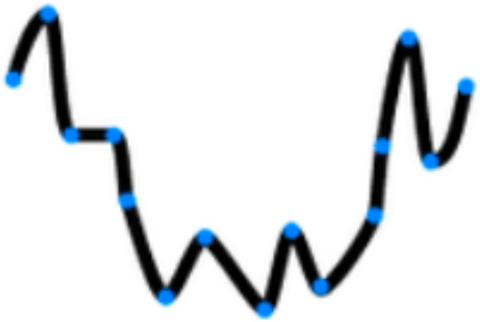
Underfitting
Overfitting

Image credit: hackernoon.com

Model Capacity

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

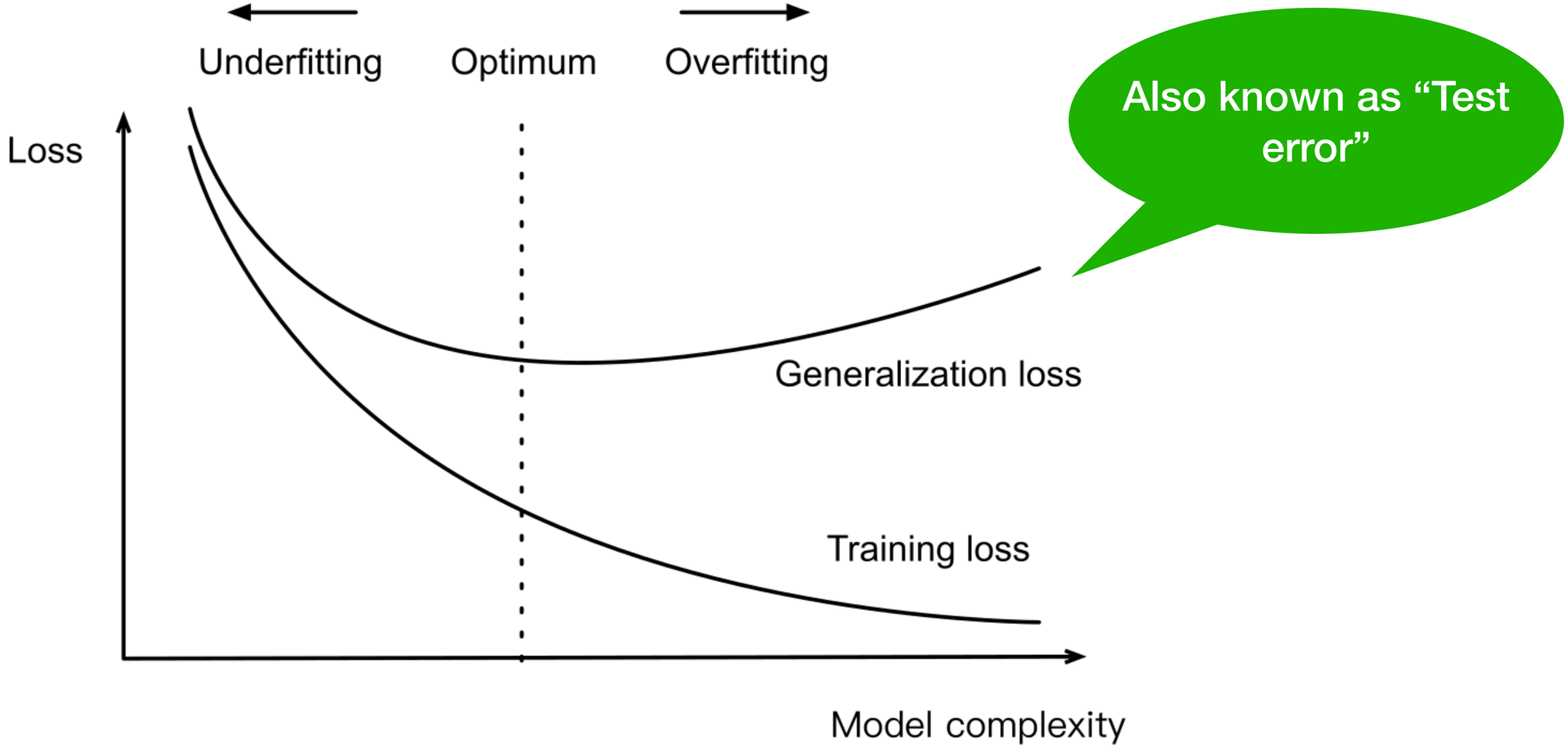




Underfitting and Overfitting

	Data complexity: Simple	Data complexity: Complex
Model capacity: Low	Normal	Underfitting
Model capacity: High	Overfitting	Normal

Influence of Model Complexity



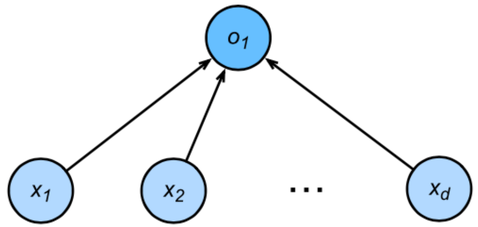
Estimate Neural Network Capacity

- It’s hard to compare complexity between different algorithms
  - e.g. tree vs neural network

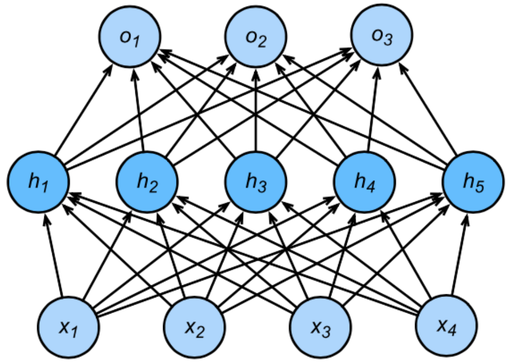
Estimate Neural Network Capacity

- It's hard to compare complexity between different algorithms
  - e.g. tree vs neural network
- Given an algorithm family, two main factors matter:
  - The number of parameters
  - The values taken by each parameter

$d + 1$



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

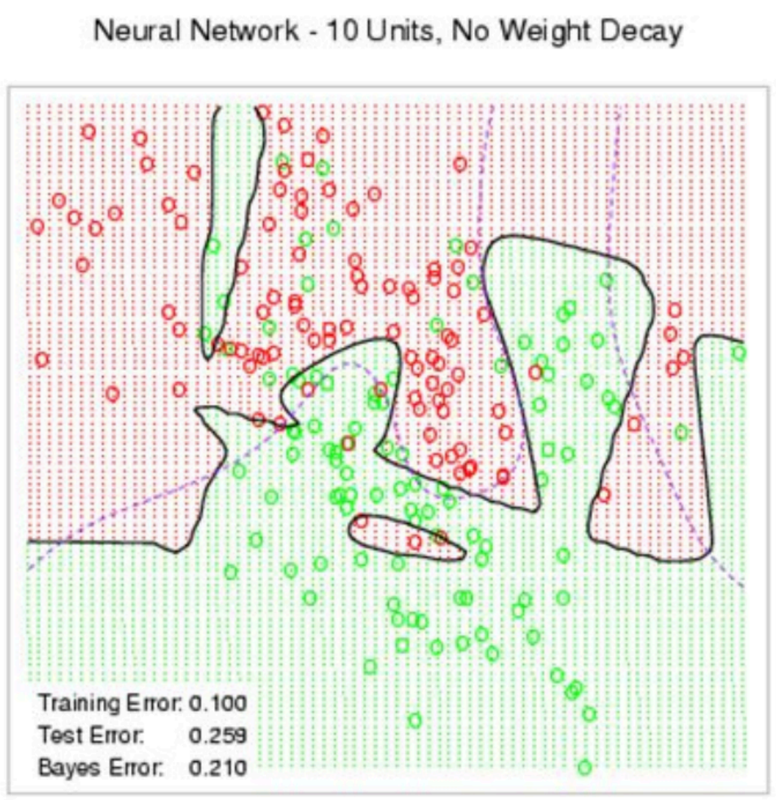


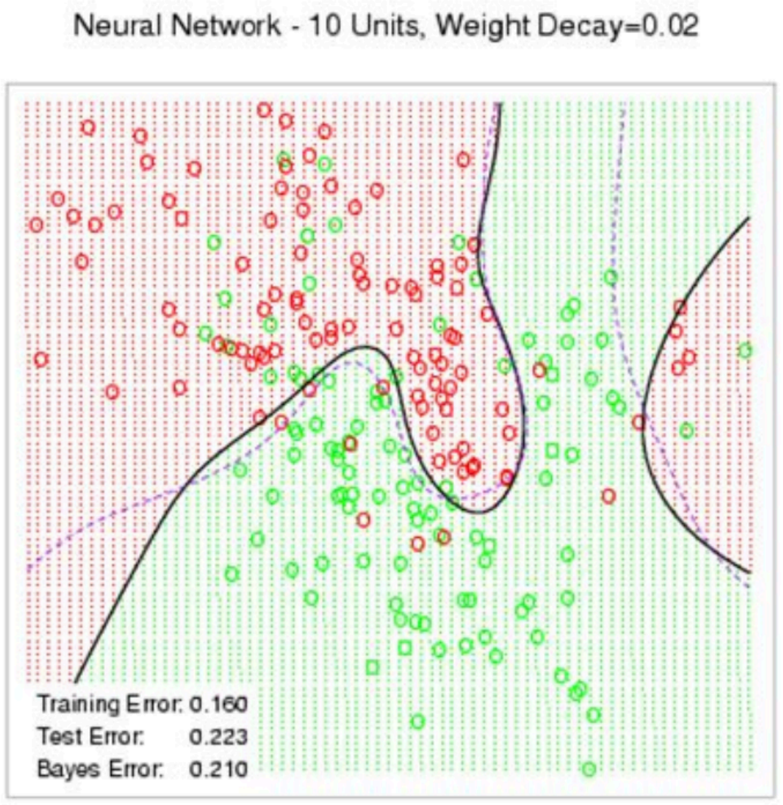
Data Complexity

- Multiple factors matters
  - # of examples
  - # of features in each example
  - time/space structure
  - # of labels

How to regularize the model for better generalization?

Weight Decay



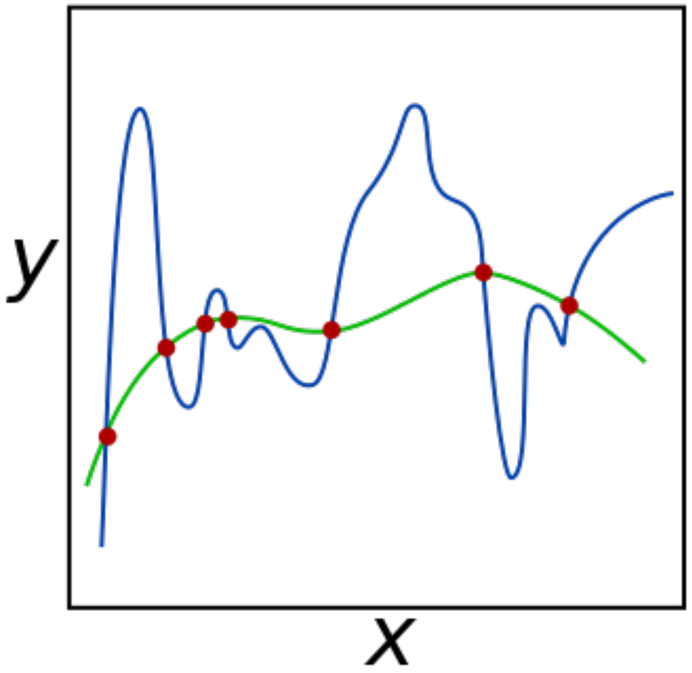


Squared Norm Regularization as Hard Constraint

- Reduce model complexity by limiting value range

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

  - 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

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

Squared Norm Regularization as Soft Constraint

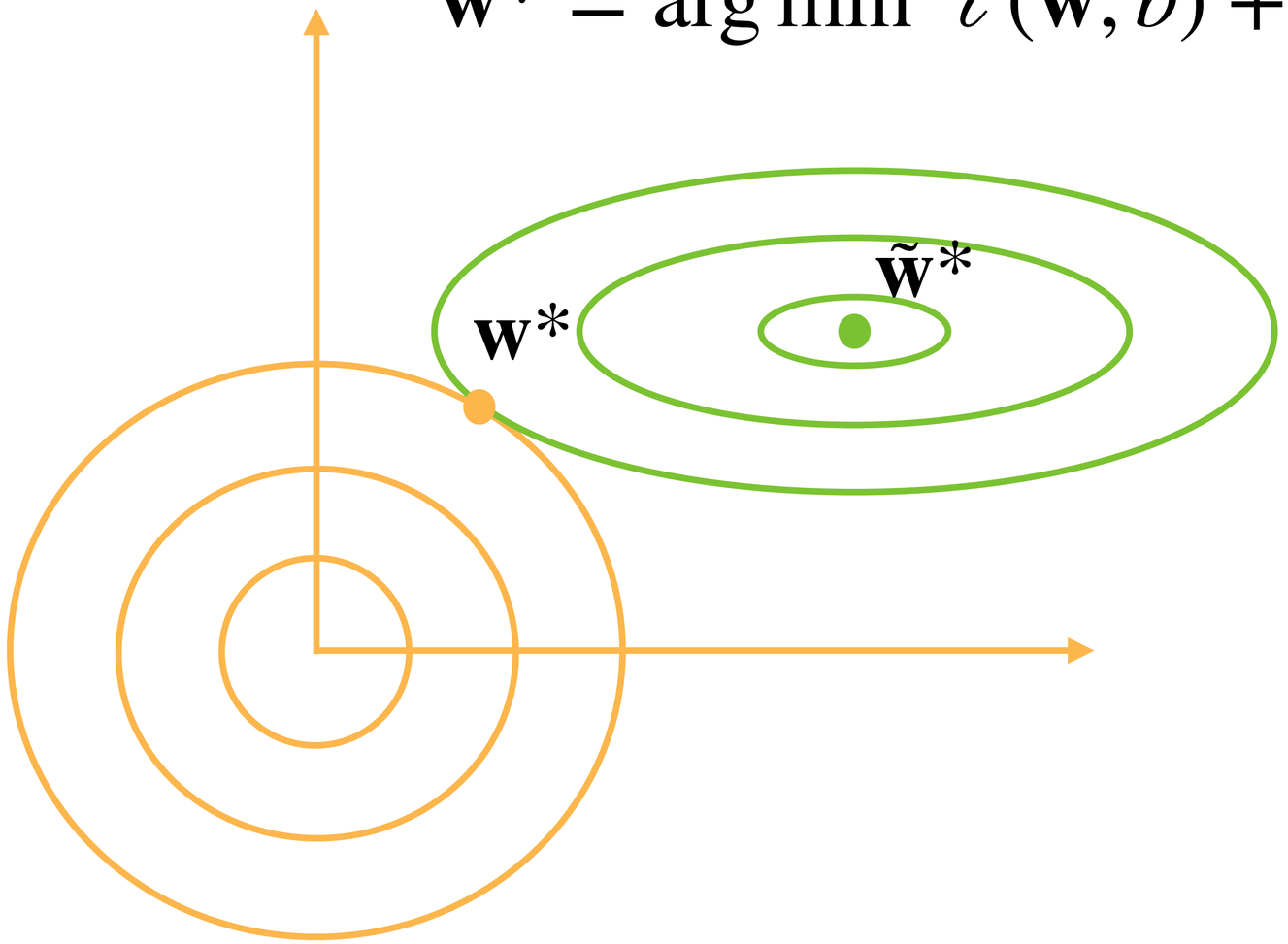
- We can rewrite the hard constraint version as

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

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

Illustrate the Effect on Optimal Solutions

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



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

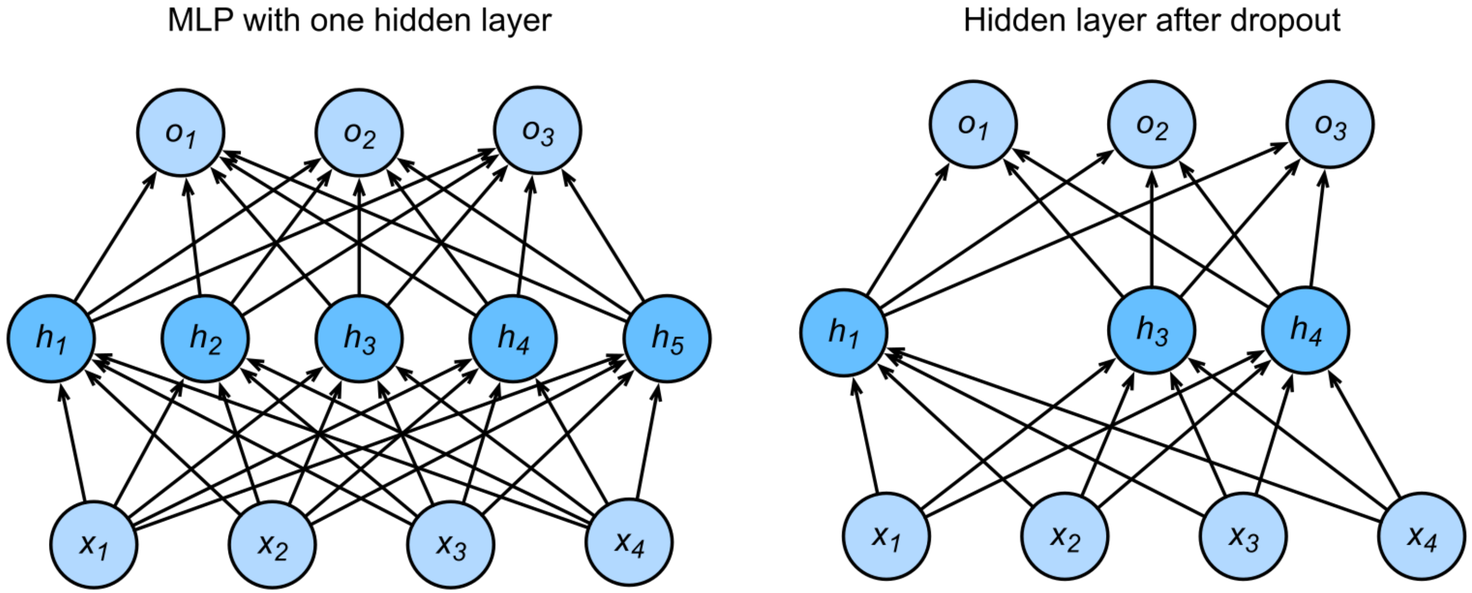
Dropout

Hinton et al.

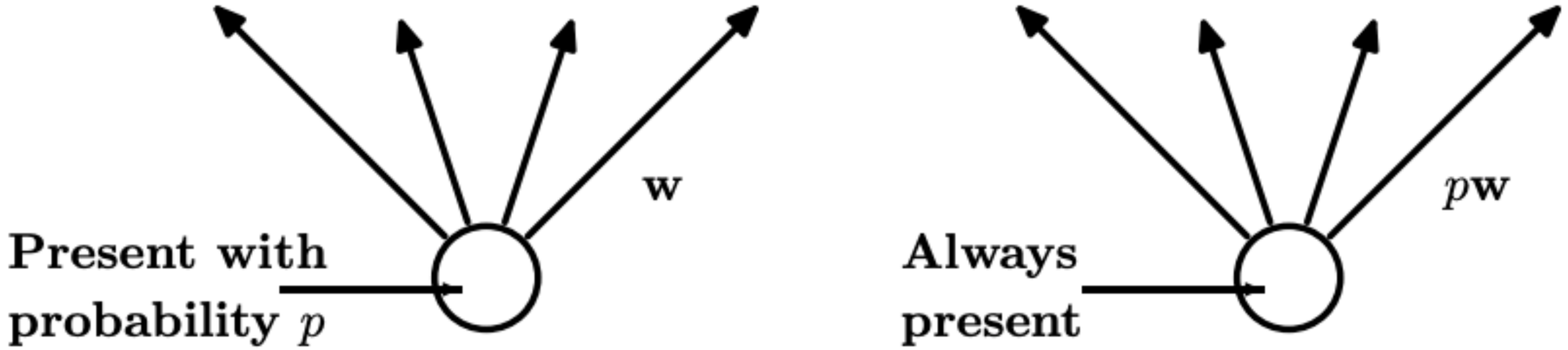
Apply Dropout

- 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.

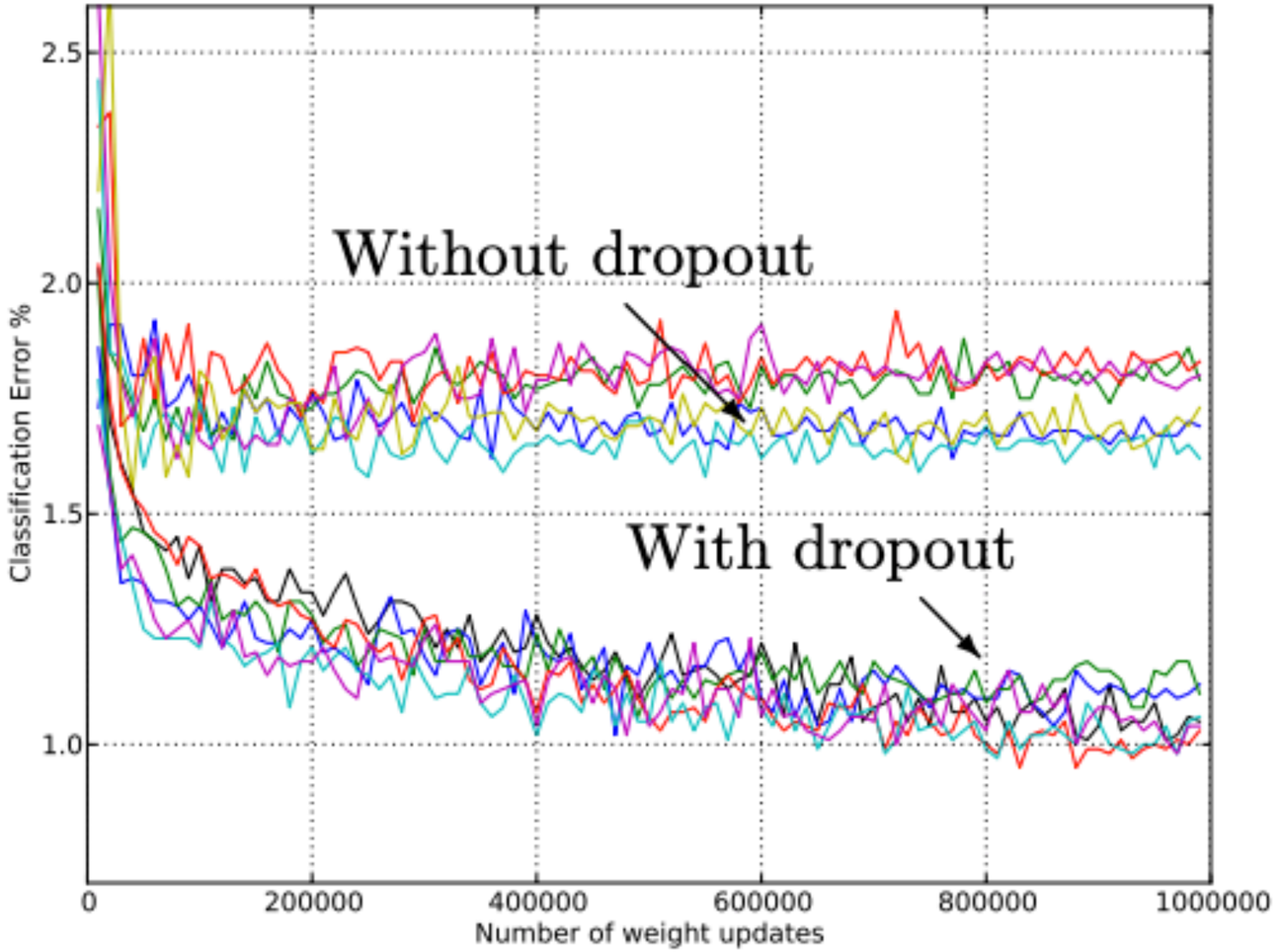
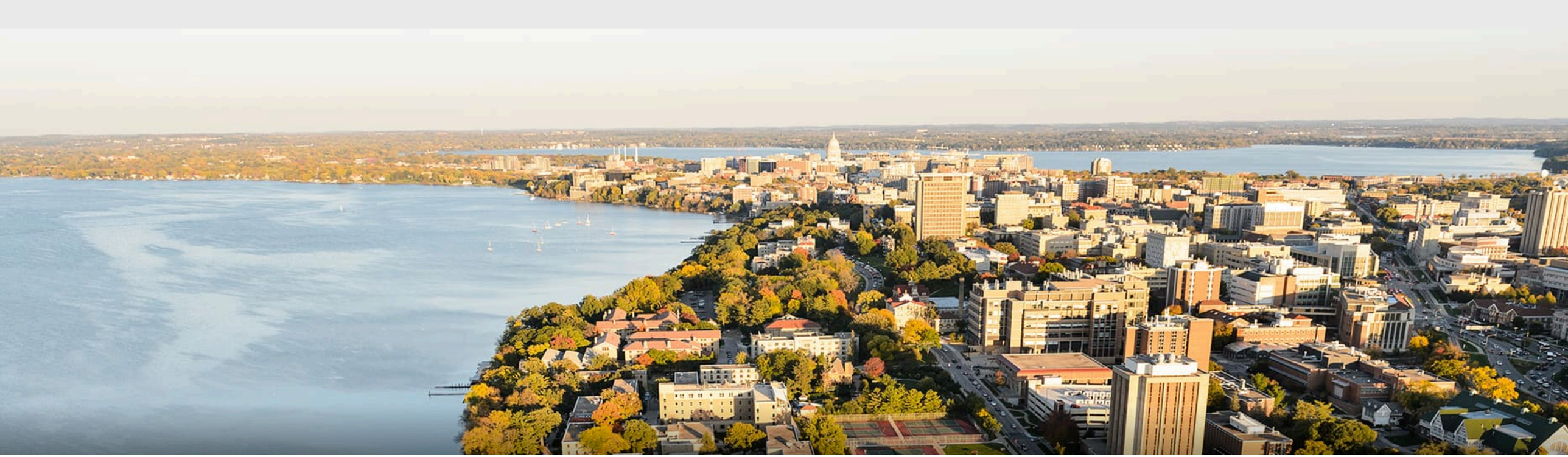


Figure 4: Test error for different architectures with and without dropout. The networks have 2 to 4 hidden layers each with 1024 to 2048 units.

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