CS 540 Introduction to Artificial Intelligence

Neural Networks (III)

University of Wisconsin-Madison

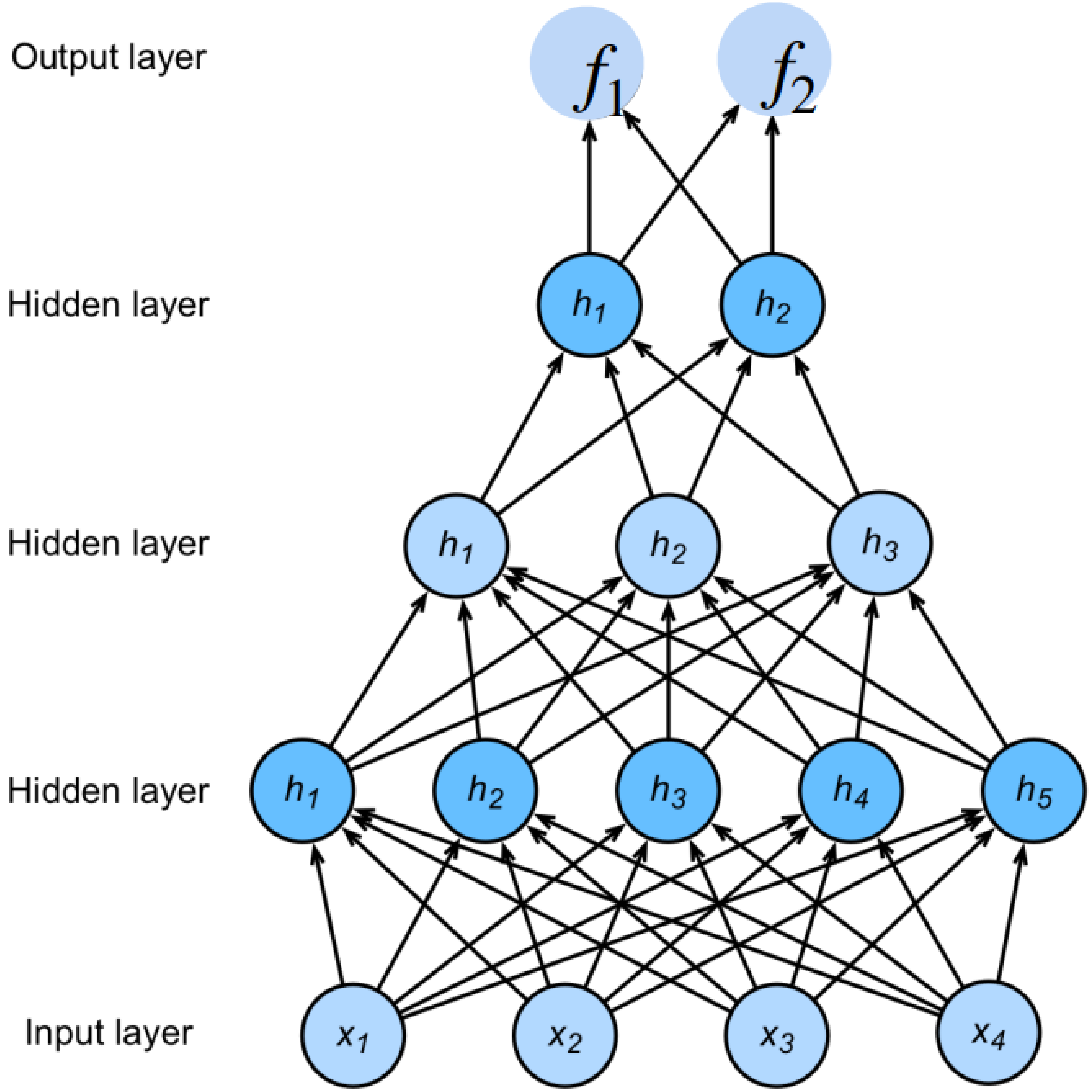
Spring 2024

Today's goals

- Understand deep neural networks as **computational graphs**.
  - Forward propagation of inputs to outputs.
  - Backward propagation of loss gradients to weights and biases.
- Understand **numerical stability** issues in training neural networks.
  - Vanishing or exploding gradients.
- Review **generalization** and understand how to use **regularization** for better generalization.
  - Overfitting, underfitting
  - Weight decay and dropout

Part I: Neural Networks as a Computational Graph

Deep neural networks (DNNs)



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

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

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

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

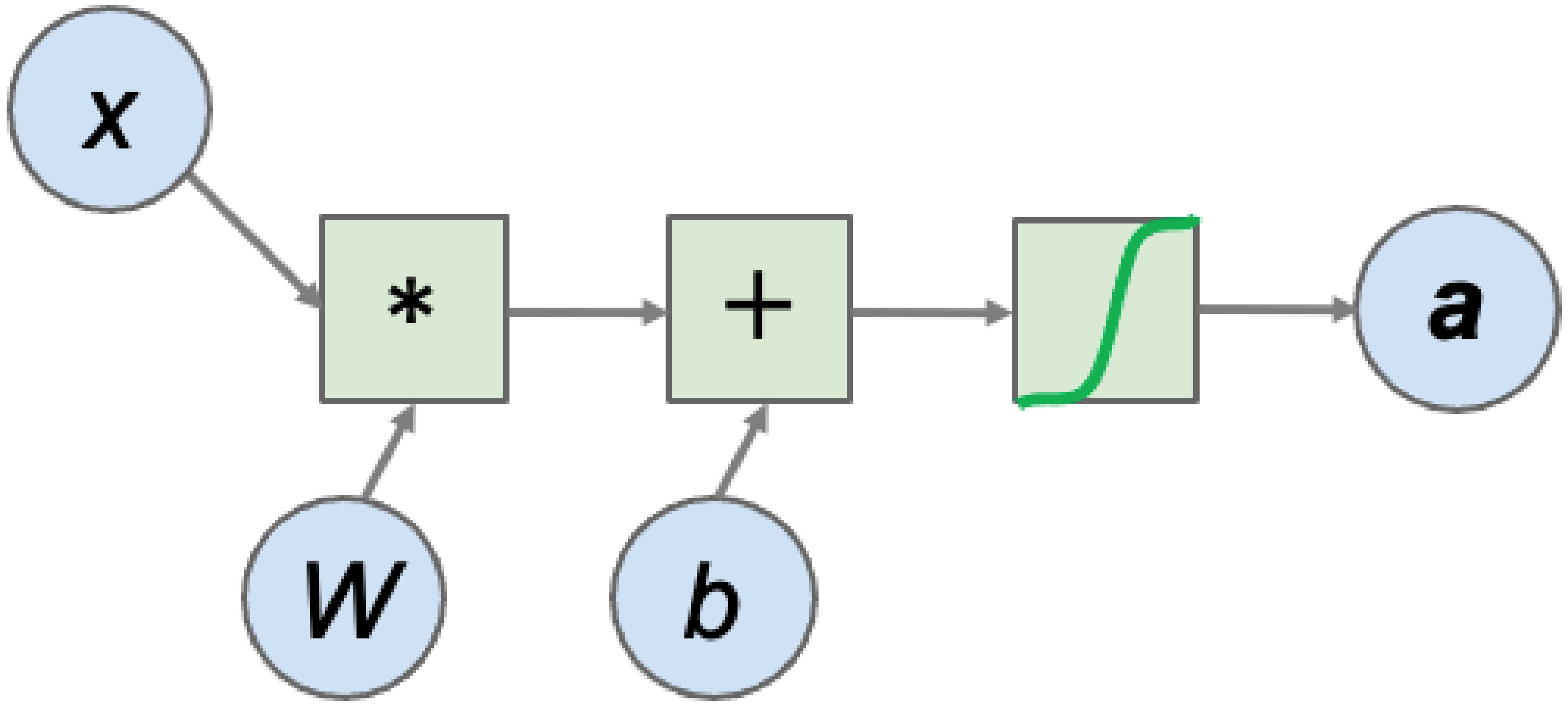
$$\mathbf{p} = \text{softmax}(\mathbf{f})$$

NNs are composition of nonlinear functions

Neural networks as variables + operations

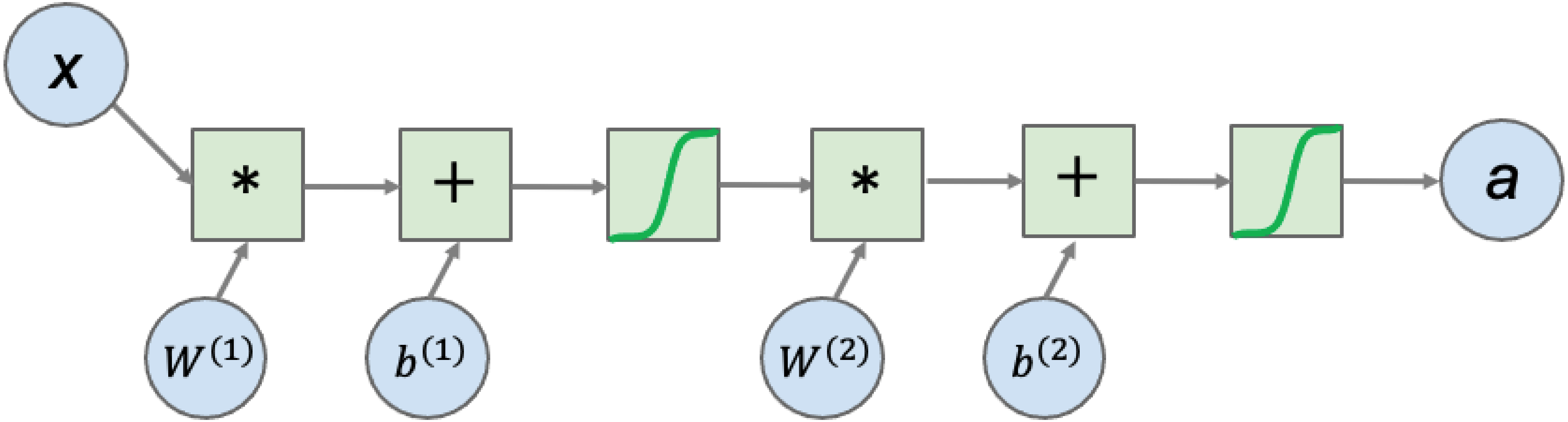
$$\mathbf{a} = sigmoid(\mathbf{Wx} + \mathbf{b})$$

- Can describe with a **computational graph**
- Decompose functions into atomic operations
- Separate data (variables) and computing (operations)



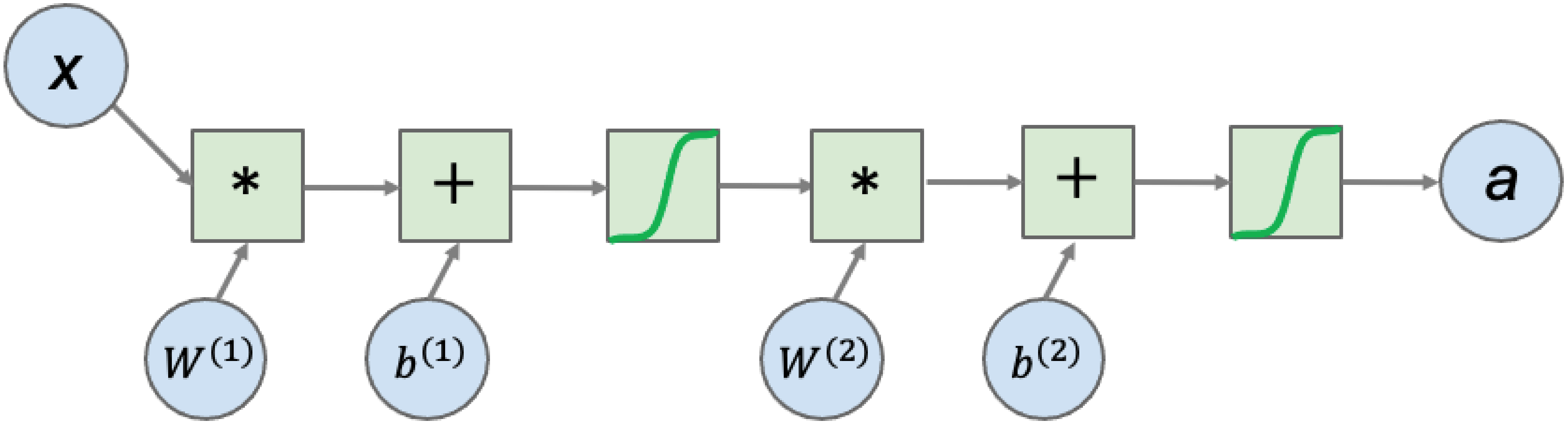
Neural networks as a computational graph

- A two-layer neural network



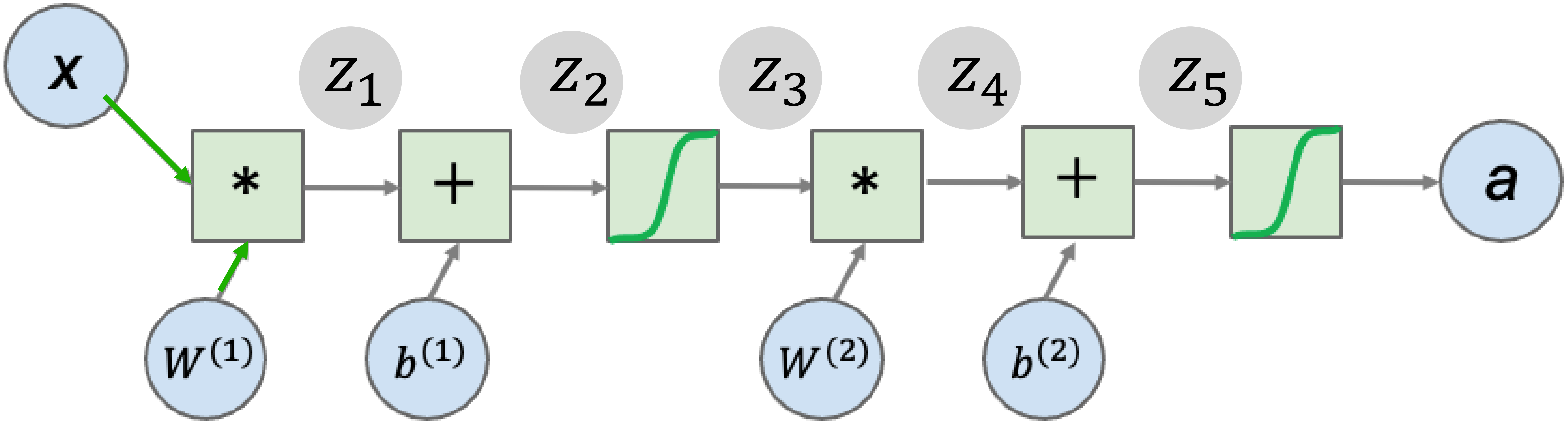
Neural networks as a computational graph

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



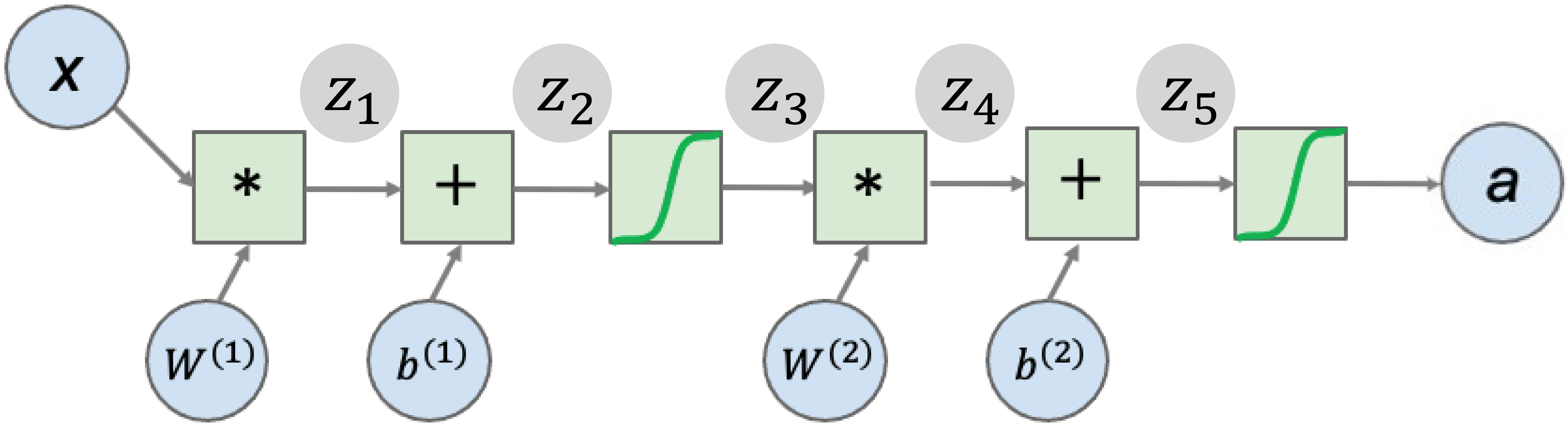
Neural networks: forward propagation

- A two-layer neural network
- Intermediate variables Z



Neural networks: backward propagation

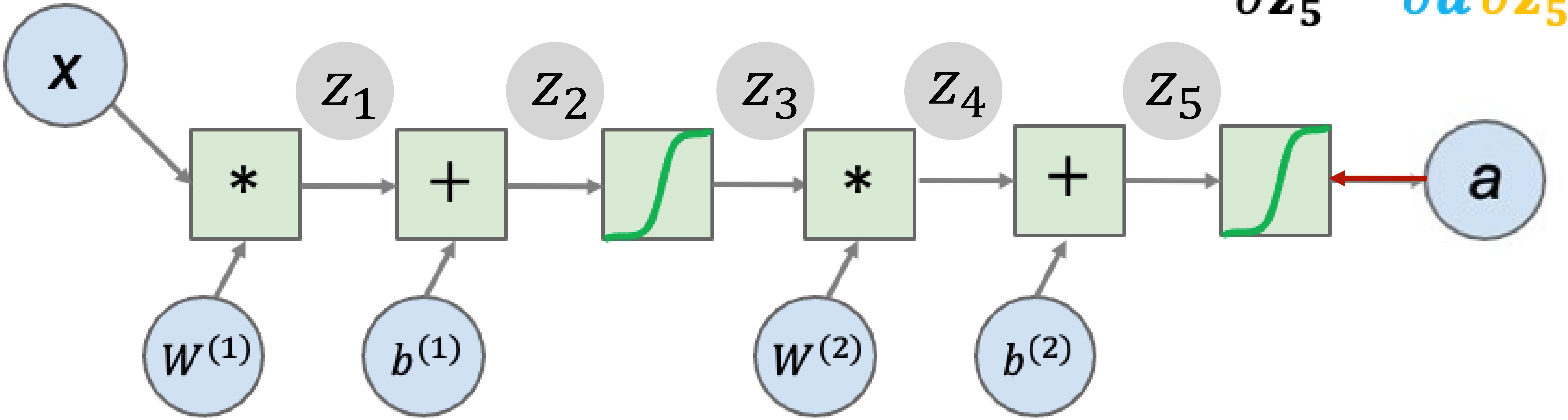
- A two-layer neural network
- Assuming forward propagation is done
- Minimize a **loss function** L



Neural networks: backward propagation

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

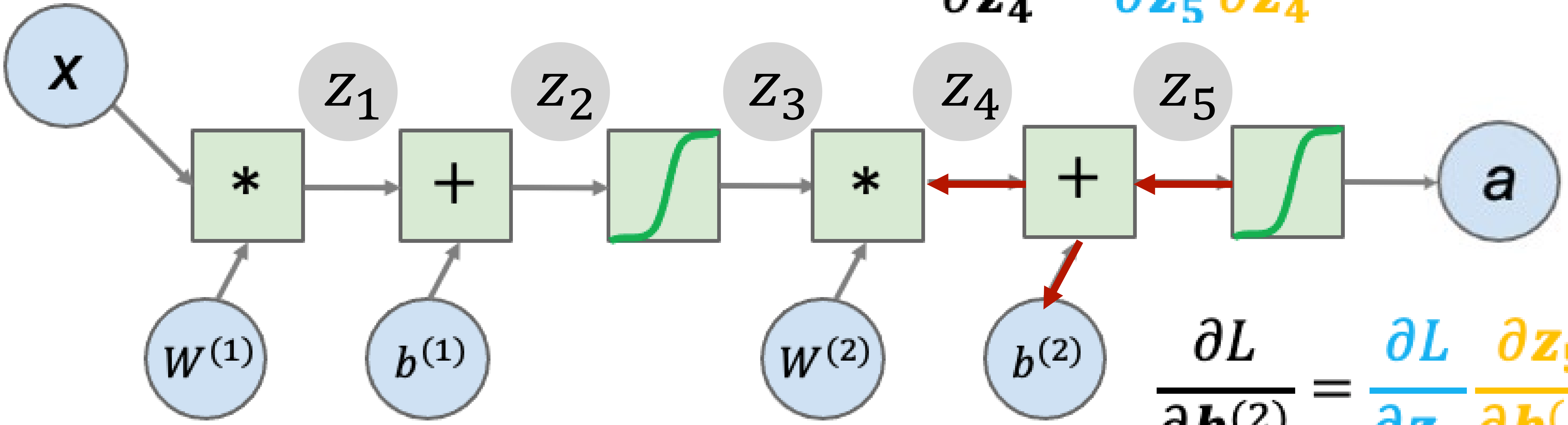
$$\frac{\partial L}{\partial z_5} = \frac{\partial L}{\partial a}\frac{\partial a}{\partial z_5}$$



Neural networks: backward propagation

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

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

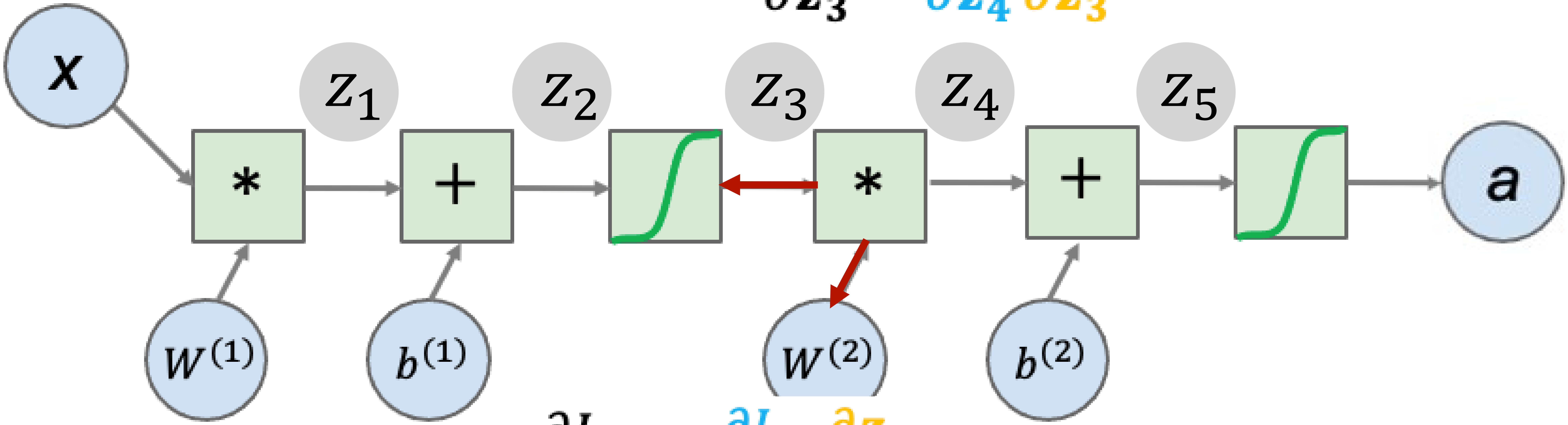


$$\frac{\partial L}{\partial \boldsymbol{b}^{(2)}} = \frac{\partial L}{\partial \boldsymbol{z}_5} \frac{\partial \boldsymbol{z}_5}{\partial \boldsymbol{b}^{(2)}}$$

Neural networks: backward propagation

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

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



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

Backward propagation: A modern treatment

- First, define a neural network as a computational graph
  - Nodes are variables and operations.
- Must be a directed graph
- All operations must be differentiable.
- Backpropagation computes partial derivatives starting from the loss and then working backwards through the graph.

Backward propagation: PyTorch

```
for t in range(2000):

    # Forward pass: compute predicted y by passing x to the
    # override the __call__ operator so you can call them l
    # doing so you pass a Tensor of input data to the Modul
    # a Tensor of output data.
    y_pred = model(xx)

    # Compute and print loss. We pass Tensors containing th
    # values of y, and the loss function returns a Tensor c
    # loss.
    loss = loss_fn(y_pred, y)
    if t % 100 == 99:
        print(t, loss.item())

    # Zero the gradients before running the backward pass.
    model.zero_grad()

    # Backward pass: compute gradient of the loss with resp
    # parameters of the model. Internally, the parameters c
    # in Tensors with requires_grad=True, so this call will
    # all learnable parameters in the model.
    loss.backward()

    # Update the weights using gradient descent. Each param
    # we can access its gradients like we did before.
    with torch.no_grad():
        for param in model.parameters():
            param -= learning_rate * param.grad
```

Forward propagation

Backward propagation

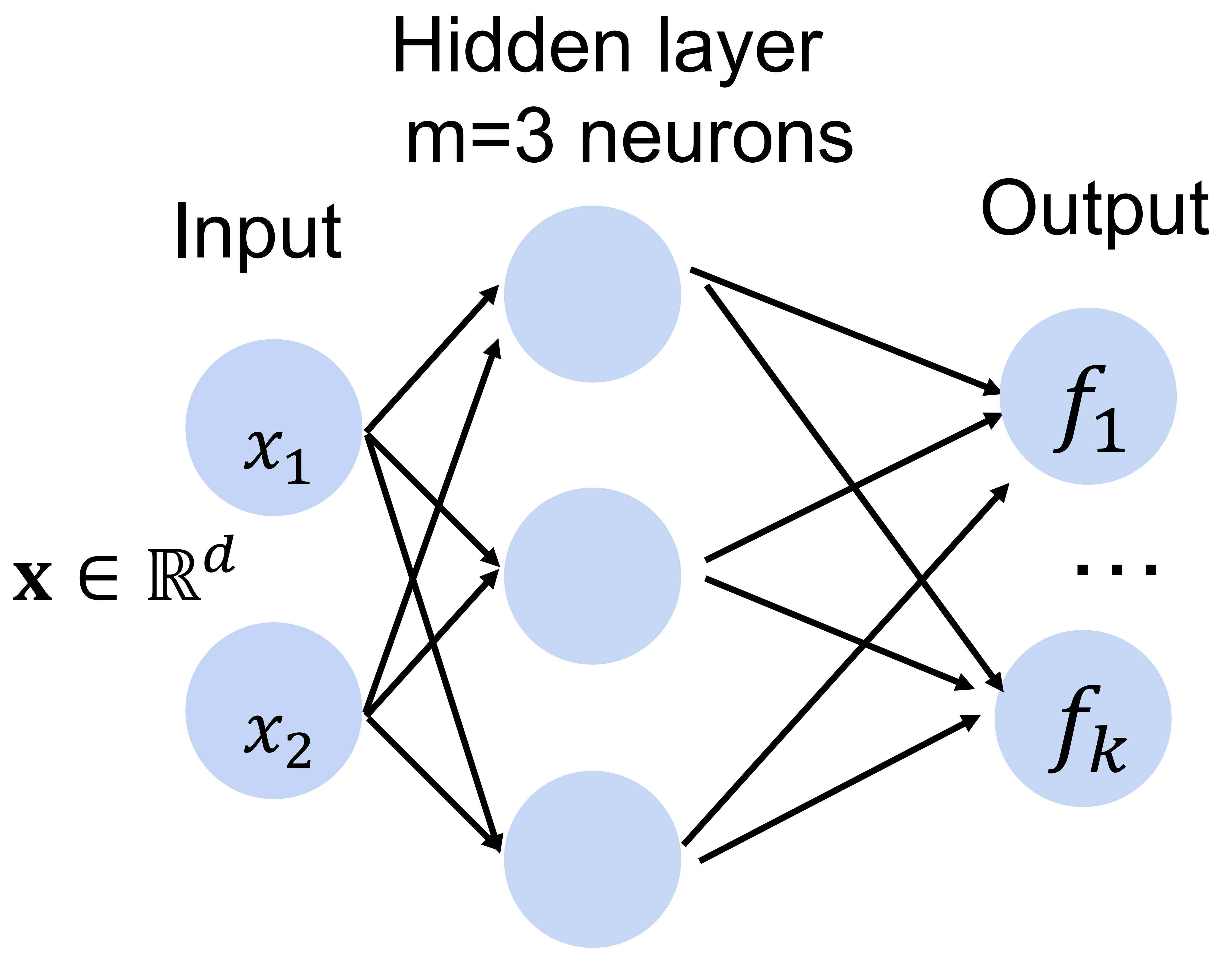
Gradient Descent

Q1.1 Suppose we want to solve the following k-class classification problem with cross entropy loss $\ell(\mathbf{y}, \hat{\mathbf{y}}) = -\sum_{j=1}^{k} y_j \log \hat{\mathbf{y}}_j$, where the ground truth and predicted probabilities $\mathbf{y}, \hat{\mathbf{y}} \in \mathbb{R}^k$. Recall that the softmax function turns output into probabilities: $\hat{y}_j = \frac{\exp f_j(x)}{\sum_i^k \exp f_i(x)}$. What is the partial derivative $\partial_{f_j} \ell(\mathbf{y}, \hat{\mathbf{y}})$?

$A.\, \hat{y}_j - y_j$

B. $\exp(y_j) - y_j$

$C.\, y_j - \hat{y}_j$



Q1.1 Suppose we want to solve the following k-class classification problem with cross entropy loss $\ell(\mathbf{y}, \hat{\mathbf{y}}) = -\sum_{j=1}^{k} y_j \log \hat{\mathbf{y}}_j$, where the ground truth and predicted probabilities $\mathbf{y}, \hat{\mathbf{y}} \in \mathbb{R}^k$. Recall that the softmax function turns output into probabilities: $\hat{y}_j = \frac{\exp f_j(x)}{\sum_i^k \exp f_i(x)}$. What is the partial derivative $\partial_{f_j} \ell(\mathbf{y}, \hat{\mathbf{y}})$?

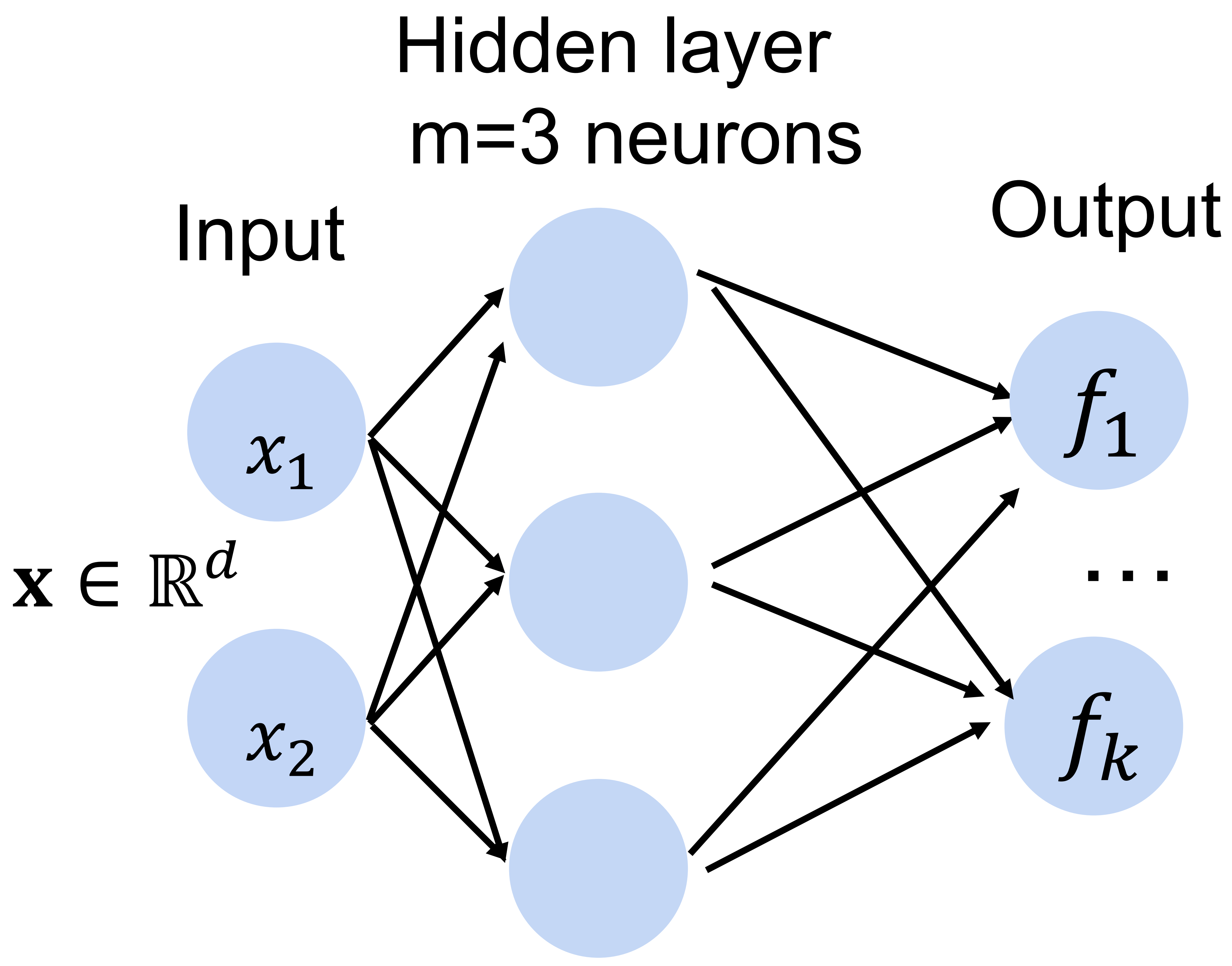
A. $\hat{y}_j - y_j$

B. $\exp(y_j) - y_j$

C. $y_j - \hat{y}_j$

Rewrite
$$\ell(\mathbf{y}, \hat{\mathbf{y}}) = -\sum_{j=1}^{k} y_j \log \frac{\exp(f_j)}{\sum_{i=1}^{k} \exp(f_i)}$$
$$= \sum_{j=1}^{k} y_j \log \sum_{i=1}^{k} \exp(f_i) - \sum_{j=1}^{k} y_j f_j$$
$$= \log \sum_{i=1}^{k} \exp(f_i) - \sum_{j=1}^{k} y_j f_j.$$

We have
$$\partial_{f_j} \ell(\mathbf{y}, \hat{\mathbf{y}}) = \frac{\exp(f_j)}{\sum_{i=1}^{k} \exp(f_k)} - y_j = \hat{y}_j - y_j$$



Part II: Numerical Stability

Gradients for Neural Networks

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

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

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

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

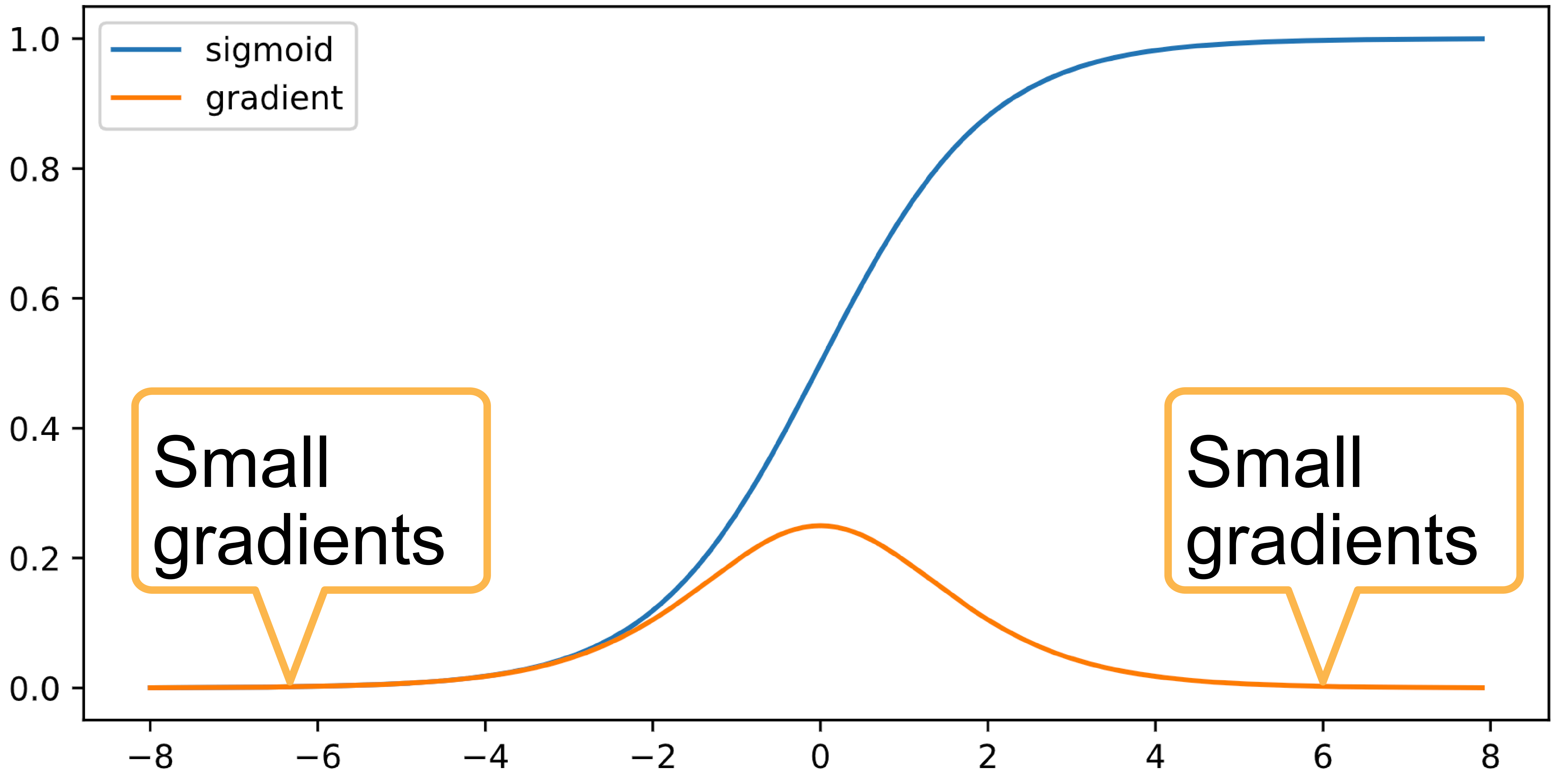
Issues with Gradient Exploding

- 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

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



Issues with Gradient Vanishing

- Gradients with value 0
- No progress in training
  - No matter how to choose learning rate
- Severe with bottom layers (those near the input)
  - Only top layers (near output) 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

Quiz. Which of the following are TRUE about the vanishing gradient problem in neural networks? Multiple answers are possible.

A. Deeper neural networks tend to be more susceptible to vanishing gradients.

B. Using the ReLU function can reduce this problem.

C. If a network has the vanishing gradient problem for one training point due to the sigmoid function, it will also have a vanishing gradient for every other training point.

D. Networks with sigmoid functions don't suffer from the vanishing gradient problem if trained with the cross-entropy loss.

Quiz. Which of the following are TRUE about the vanishing gradient problem in neural networks? Multiple answers are possible?

A. Deeper neural networks tend to be more susceptible to vanishing gradients.

B. Using the ReLU function can reduce this problem.

C. If a network has the vanishing gradient problem for one training point due to the sigmoid function, it will also have a vanishing gradient for every other training point.

D. Networks with sigmoid functions don’t suffer from the vanishing gradient problem if trained with the cross-entropy loss.

Quiz. Let’s compare sigmoid with rectified linear unit (ReLU). Which of the following statement is NOT true?

A. Sigmoid function is more expensive to compute

B. ReLU has non-zero gradient everywhere

C. The gradient of Sigmoid is always less than 0.3

D. The gradient of ReLU is constant for positive input

Quiz. Let’s compare sigmoid with rectified linear unit (ReLU). Which of the following statement is NOT true?

A. Sigmoid function is more expensive to compute

B. ReLU has non-zero gradient everywhere

C. The gradient of Sigmoid is always less than 0.3

D. The gradient of ReLU is constant for positive input

Q5. A Leaky ReLU is defined as $f(x)=max(0.1x, x)$. Let f’(0)=1. Does it have non-zero gradient everywhere??

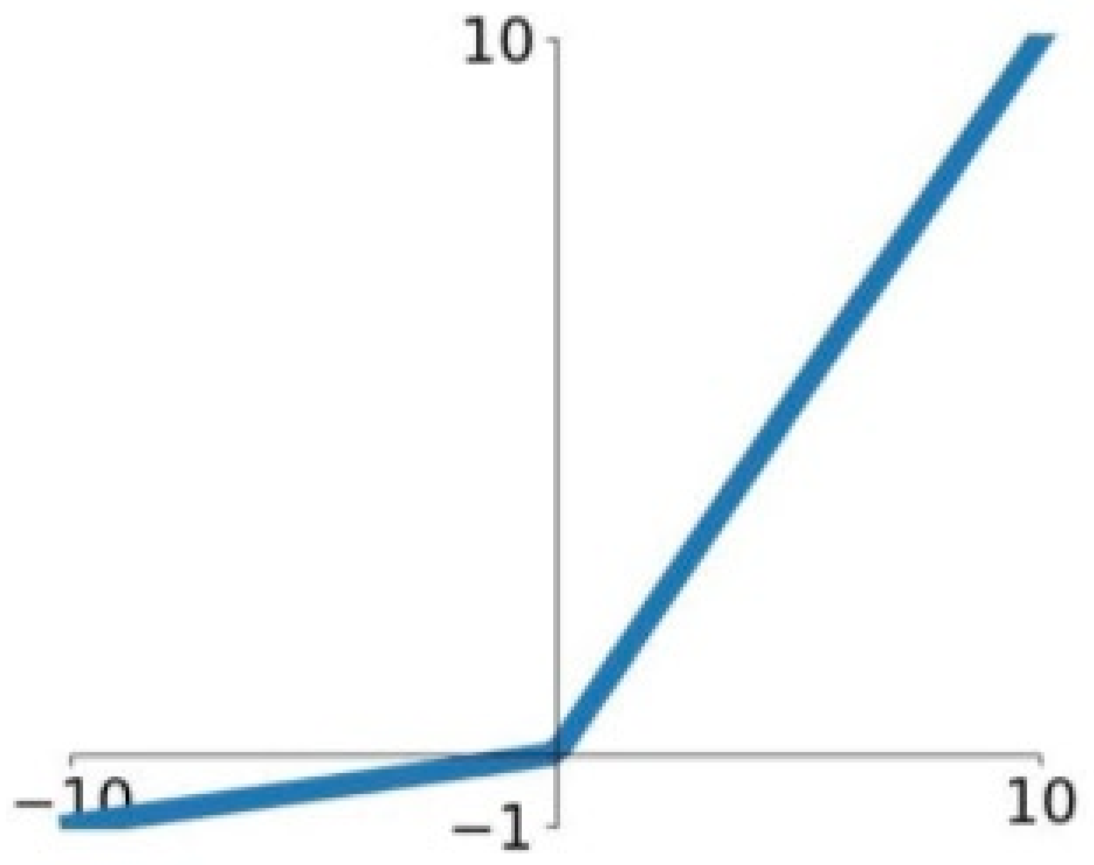
A. Yes

B. No

Q5. A Leaky ReLU is defined as *f(x)=max(0.1x, x).* Let f’(0)=1. Does it have non-zero gradient everywhere??

A. Yes

B. No



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

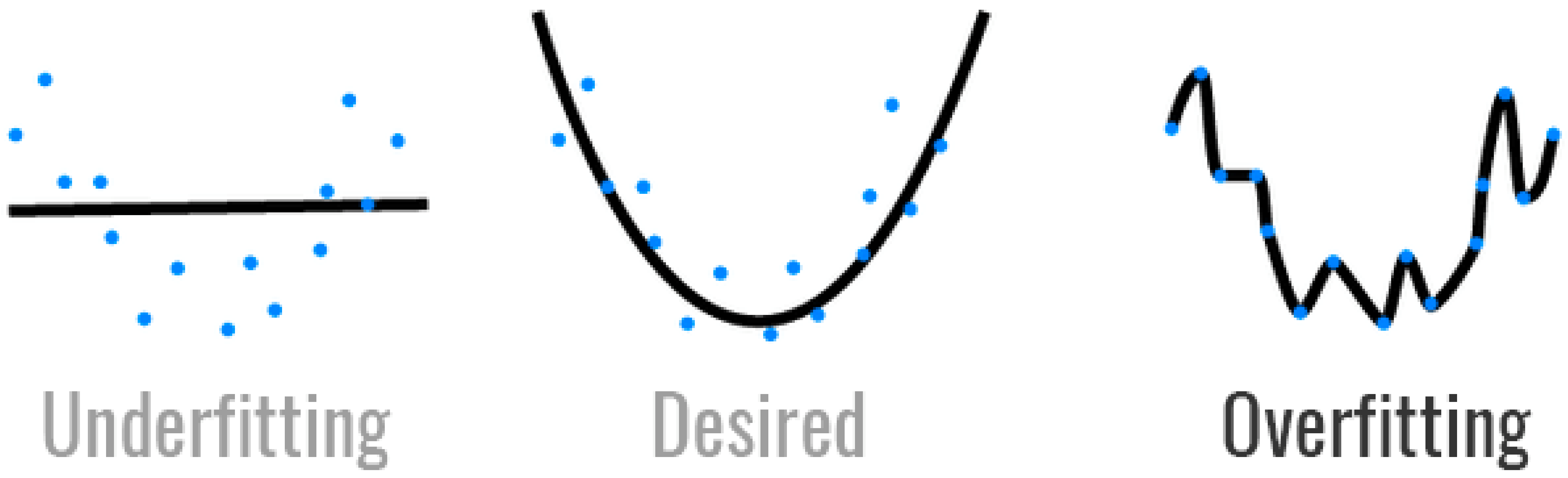
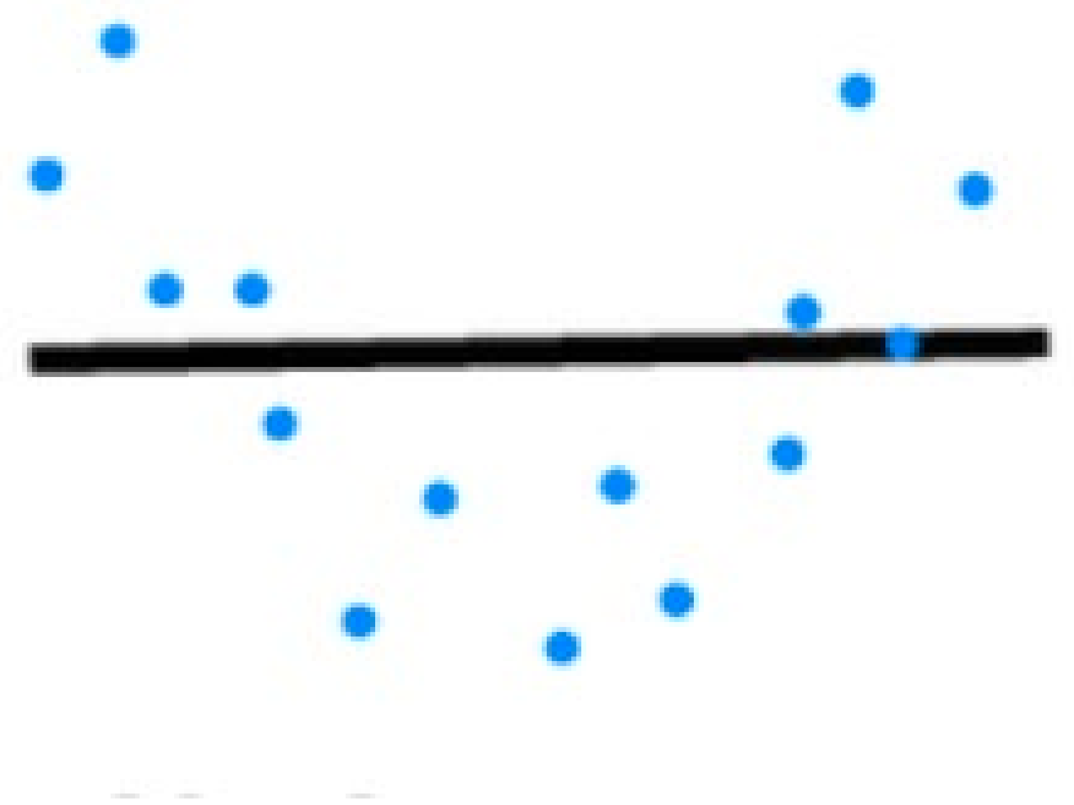


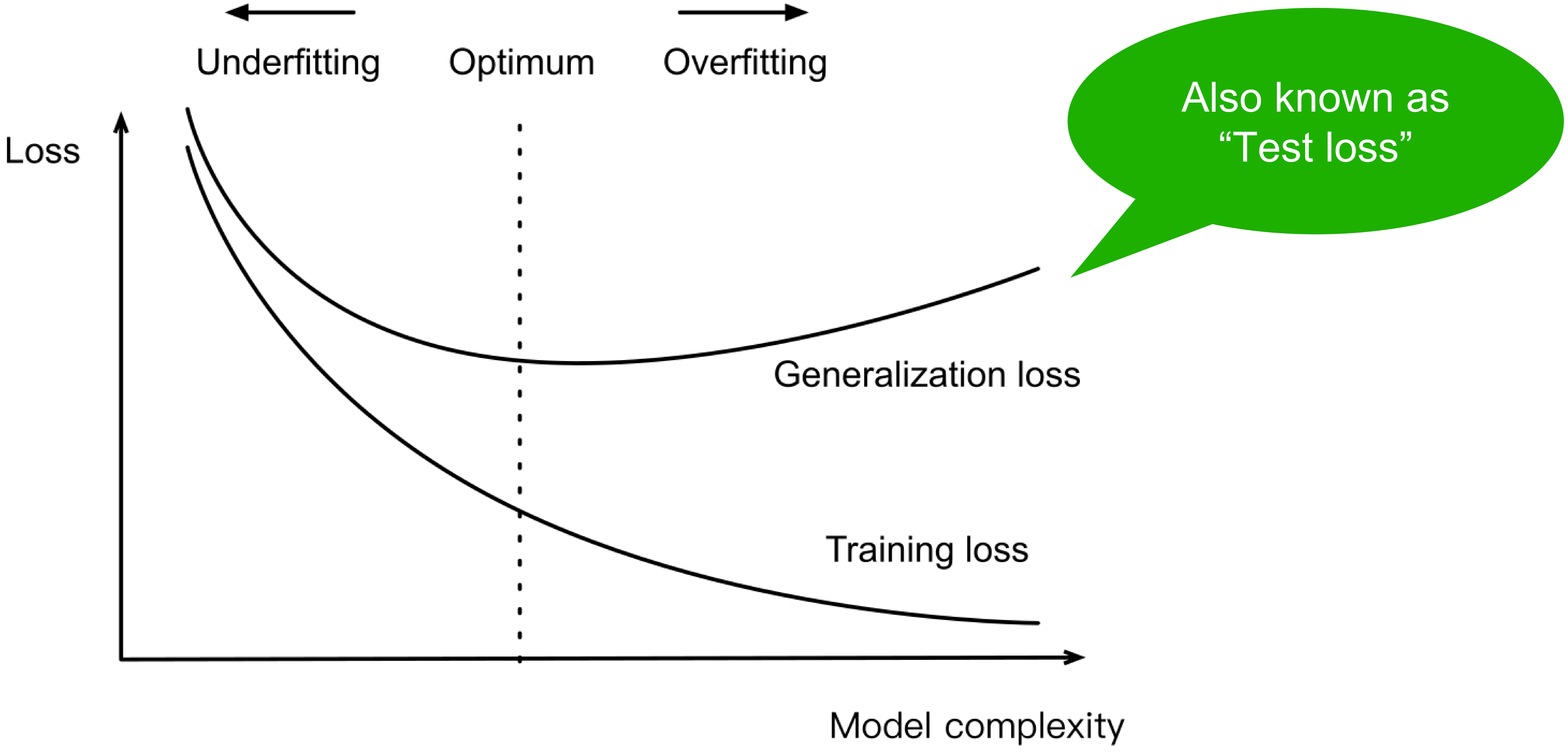
Image credit: hackernoon.com

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



Influence of Model Complexity

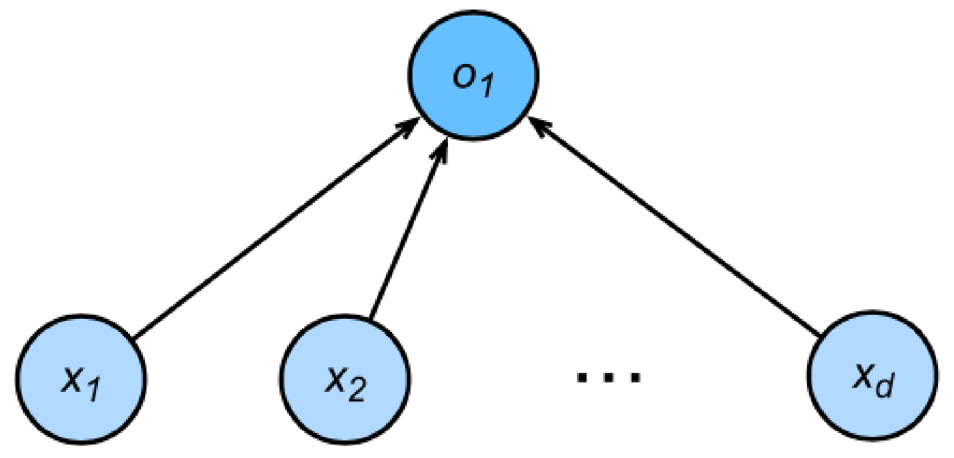


* Recent research has challenged this view for some types of models.

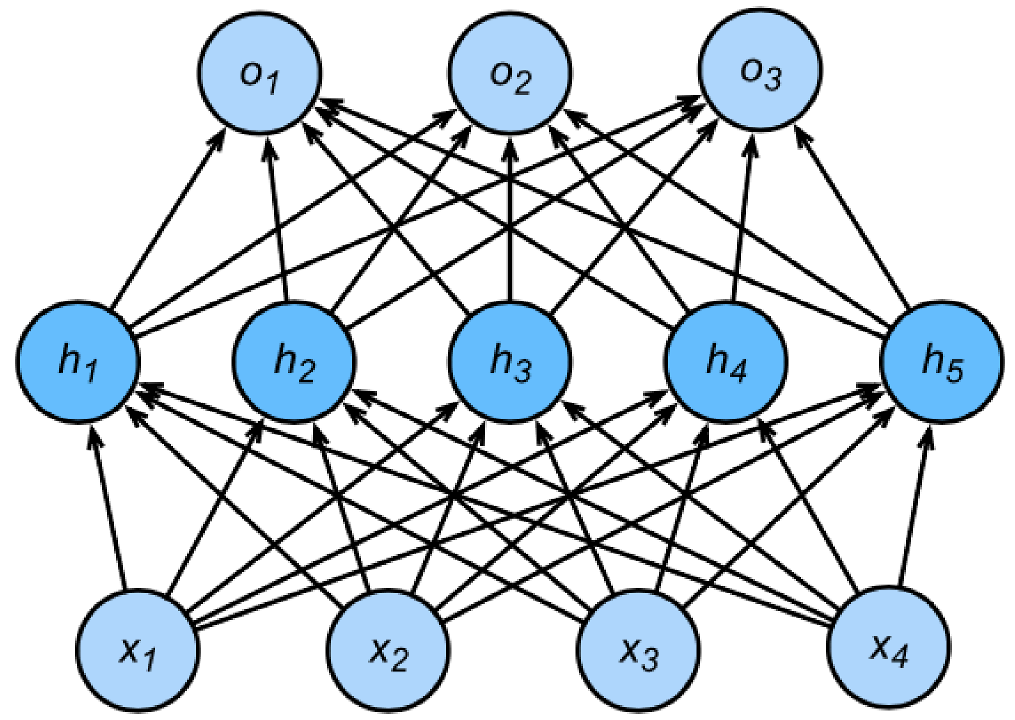
Estimate Neural Network Capacity

- It's hard to compare complexity between different families of models.
  - e.g. K-NN vs neural networks
- Given a model family, two main factors matter:
  - The number of parameters
  - The values taken by each parameter

$$d + 1$$

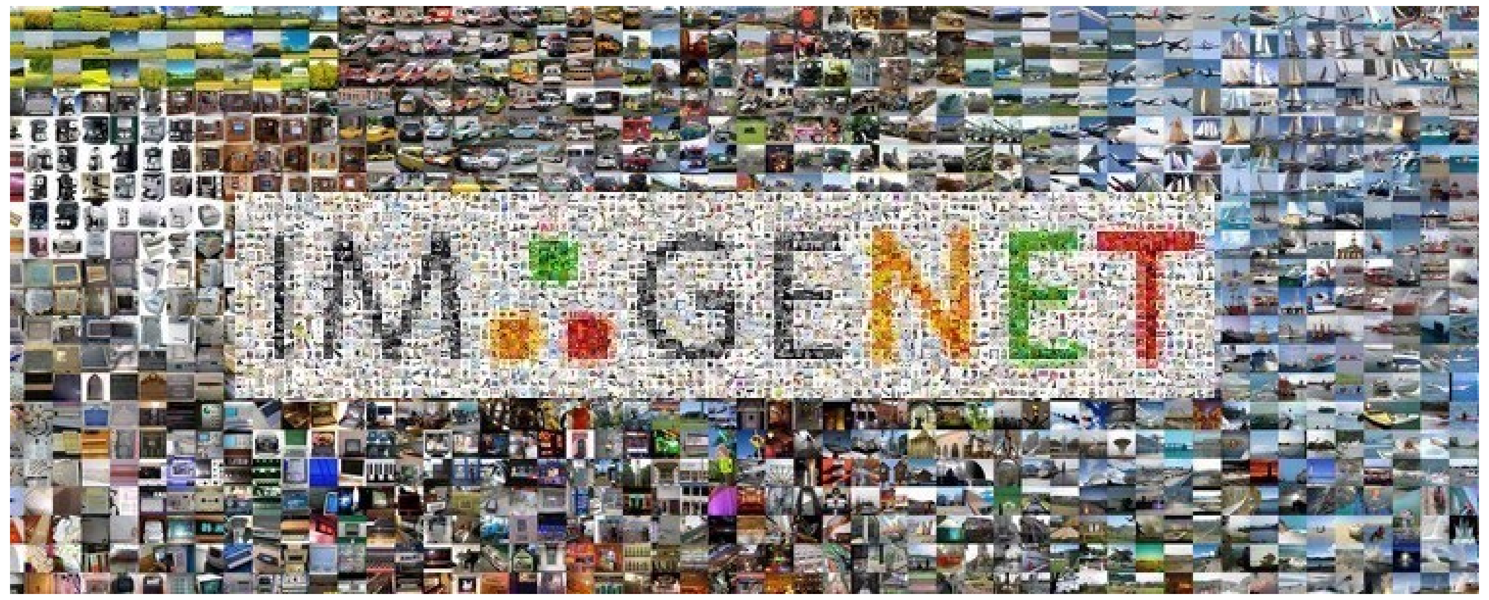


$$(d + 1)m + (m + 1)k$$



Data Complexity

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



Quiz Break: When training a neural network, which one below indicates that the network has overfit the training data?

A. Training loss is low and generalization loss is high.
B. Training loss is low and generalization loss is low.
C. Training loss is high and generalization loss is high.
D. Training loss is high and generalization loss is low.
E. None of these.

Quiz Break: When training a neural network, which one below indicates that the network has overfit the training data?

A. Training loss is low and generalization loss is high.
B. Training loss is low and generalization loss is low.
C. Training loss is high and generalization loss is high.
D. Training loss is high and generalization loss is low.
E. None of these.

Quiz Break: Adding more layers to a multi-layer perceptron may cause _______.

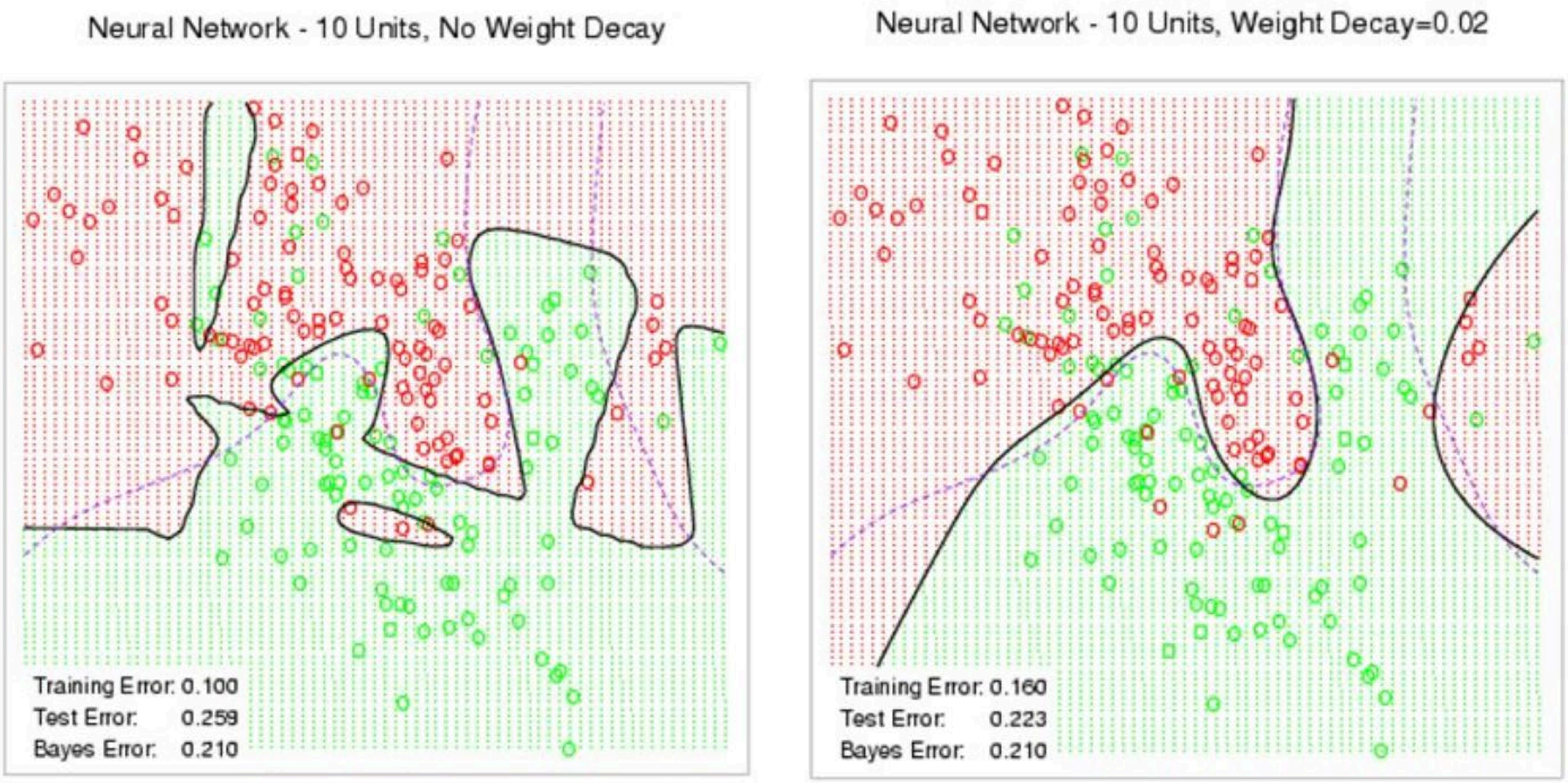
A. Vanishing gradients during back propagation.
B. A more complex decision boundary.
C. Underfitting.
D. Higher test loss.
E. None of these.

Quiz Break: Adding more layers to a multi-layer perceptron may cause _______. (Multiple answers)

A. Vanishing gradients during back propagation.
B. A more complex decision boundary.
C. Underfitting.
D. Higher test loss.
E. None of these.

How to regularize the model for better generalization?

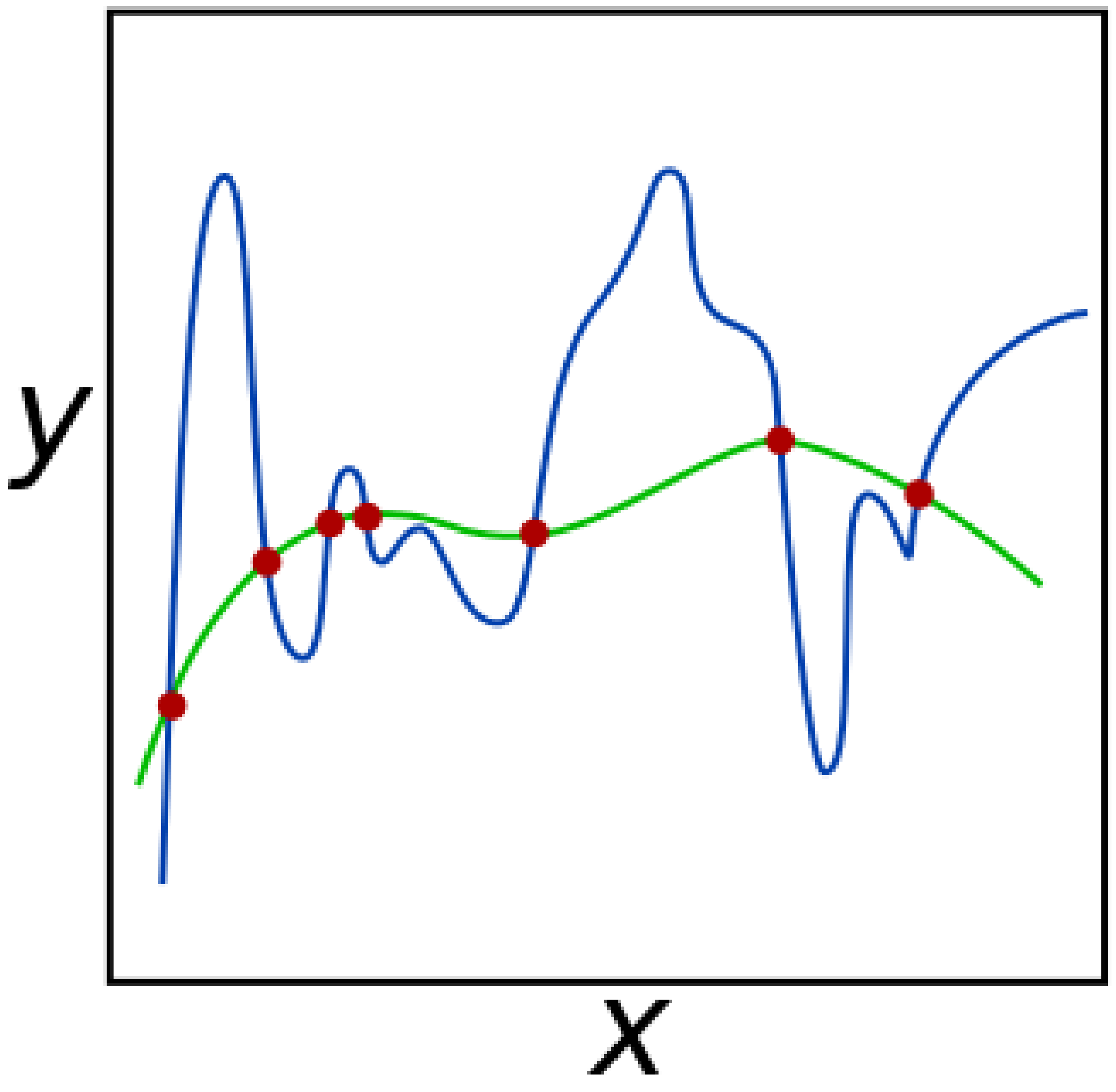
Weight Decay



Squared Norm Regularization as Hard Constraint

- Reduce model complexity by limiting value range

$$\min L(\mathbf{w}, b) \; subject \; to \; \| \mathbf{w} \|^2 \leq B$$



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

Squared Norm Regularization as Soft Constraint

- We can rewrite the hard constraint version as

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

Squared Norm Regularization as Soft Constraint

- We can rewrite the hard constraint version as

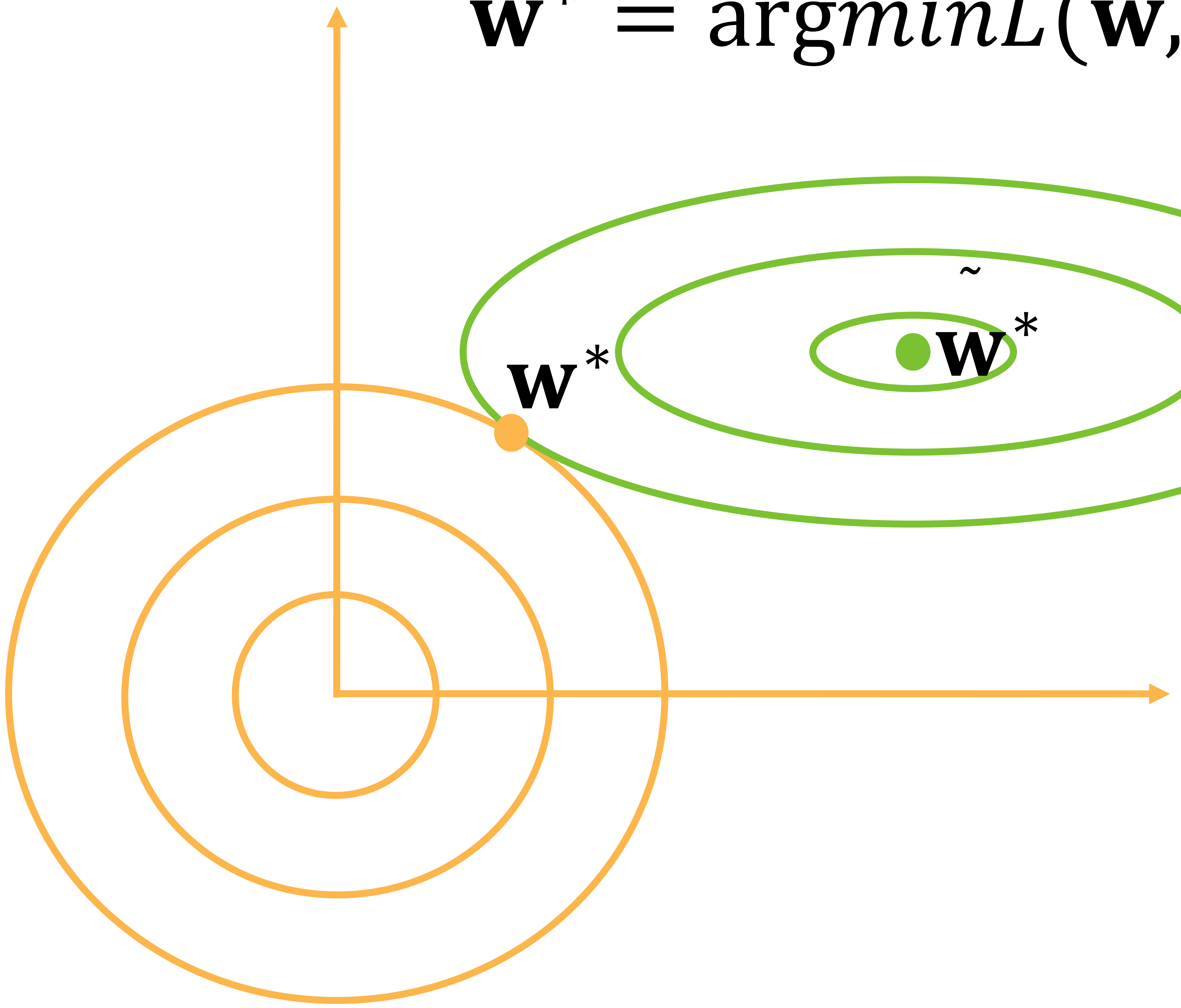
$$min L(\mathbf{w}, b) + \frac{\lambda}{2} \| \mathbf{w} \|^2$$

  - Hyper-parameter λ controls regularization importance
  - $\lambda = 0$:   no effect

    $\lambda \to \infty, \mathbf{w}^* \to \mathbf{0}$

Illustrate the Effect on Optimal Solutions

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



$$\tilde{\mathbf{w}}^* = \mathrm{arg}minL(\mathbf{w}, b)$$

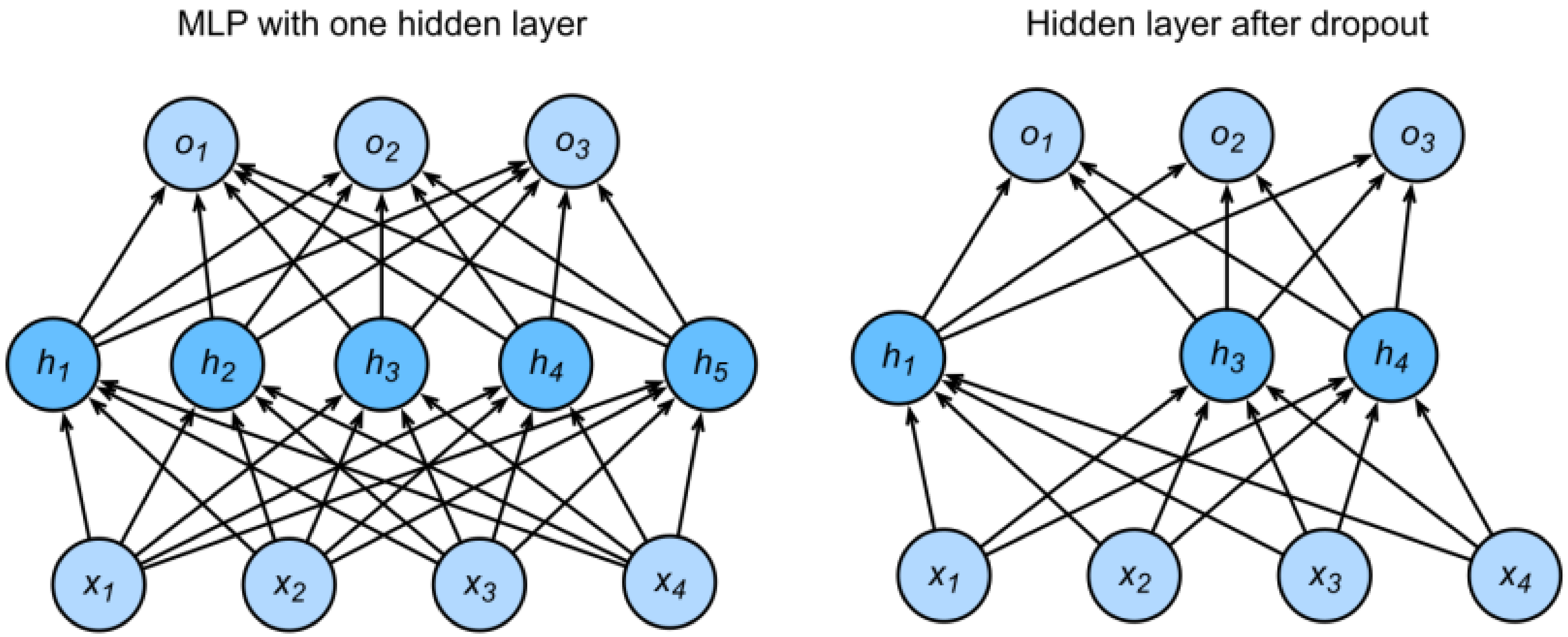
Dropout

Hinton et al.

Apply Dropout

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

$$\mathbf{h} = \sigma(\mathbf{W}^{(1)}\mathbf{x} + \mathbf{b}^{(1)})$$
$$\mathbf{h}' = \text{dropout}(\mathbf{h})$$
$$\mathbf{o} = \mathbf{W}^{(2)}\mathbf{h}' + \mathbf{b}^{(2)}$$
$$\mathbf{p} = \text{softmax}(\mathbf{o})$$



Dropout

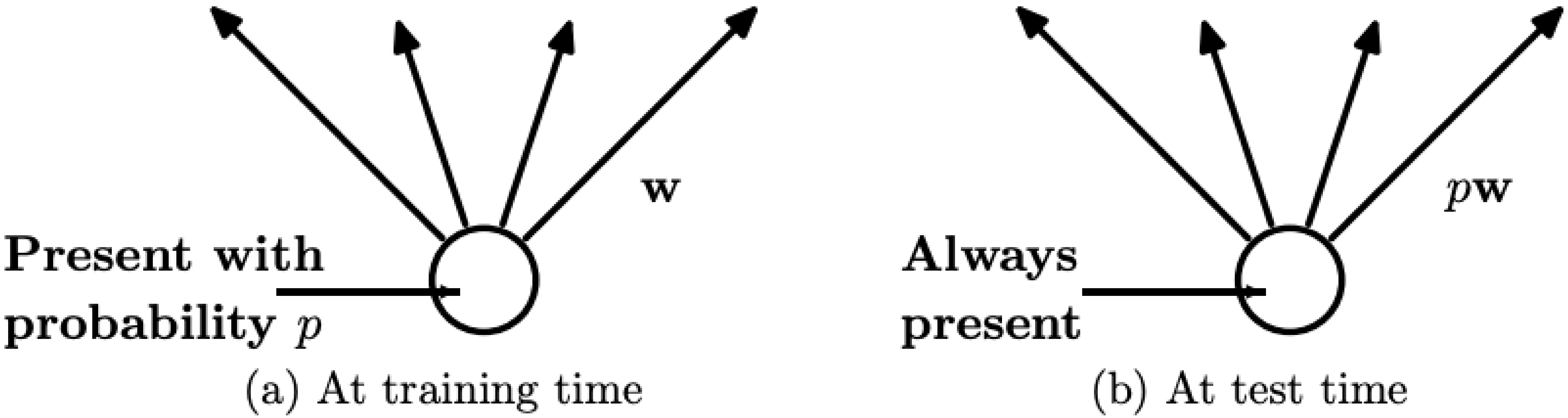


Figure 2: **Left**: A unit at training time that is present with probability p and is connected to units in the next layer with weights $\mathbf{w}$. **Right**: At test time, the unit is always present and the weights are multiplied by p. The output at test time is same as the expected output at training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

Dropout

Hinton et al.

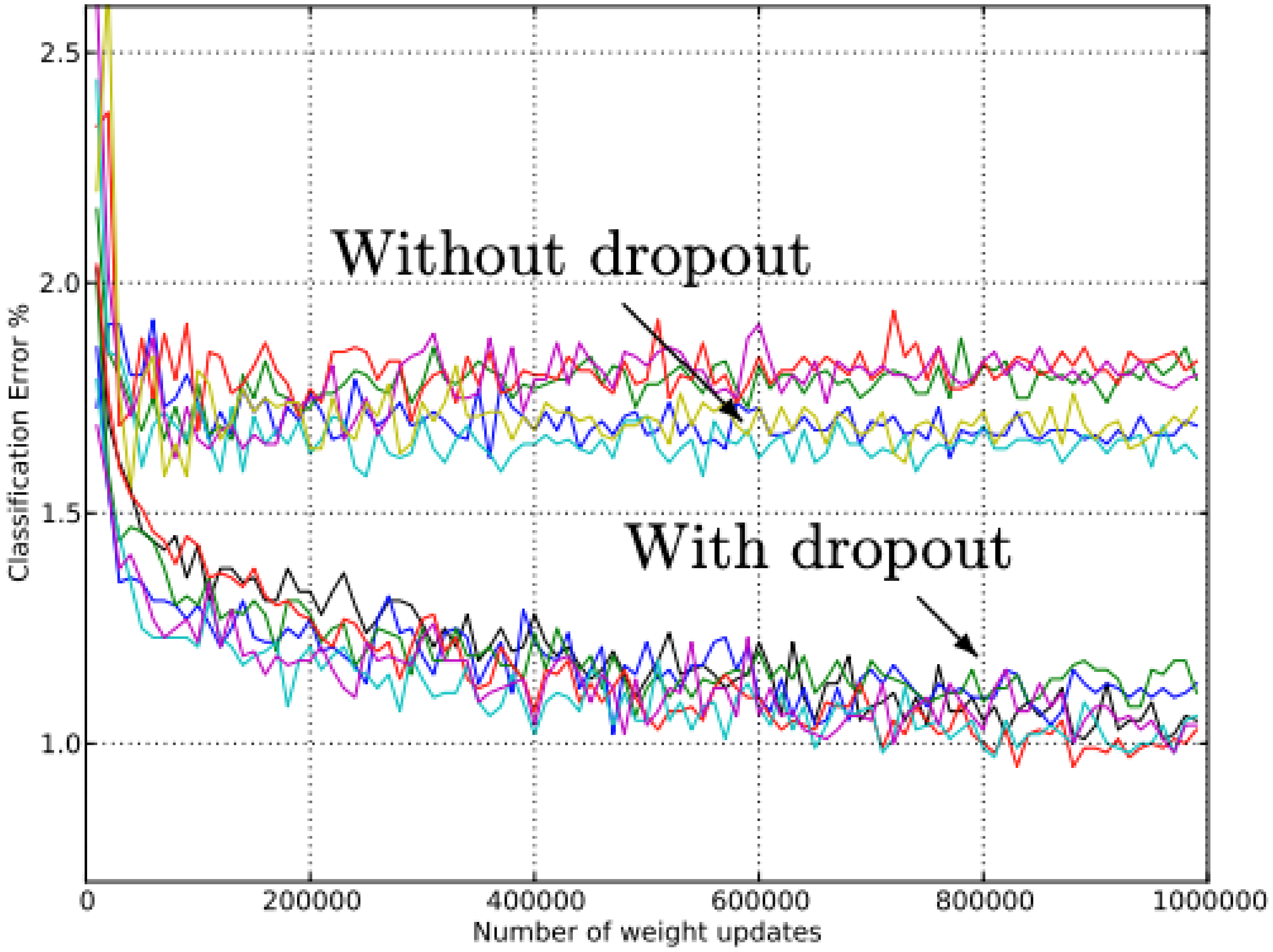


Figure 4: Test error for different architectures with and without dropout. The networks have 2 to 4 hidden layers each with 1024 to 2048 units.

Quiz Break, Q4.1:

In standard dropout regularization, with dropout probability p, each intermediate activation h is replaced by a random variable h' as: $h' = \begin{cases} 0 & \text{with probability } p \\ ? & \text{otherwise} \end{cases}.$

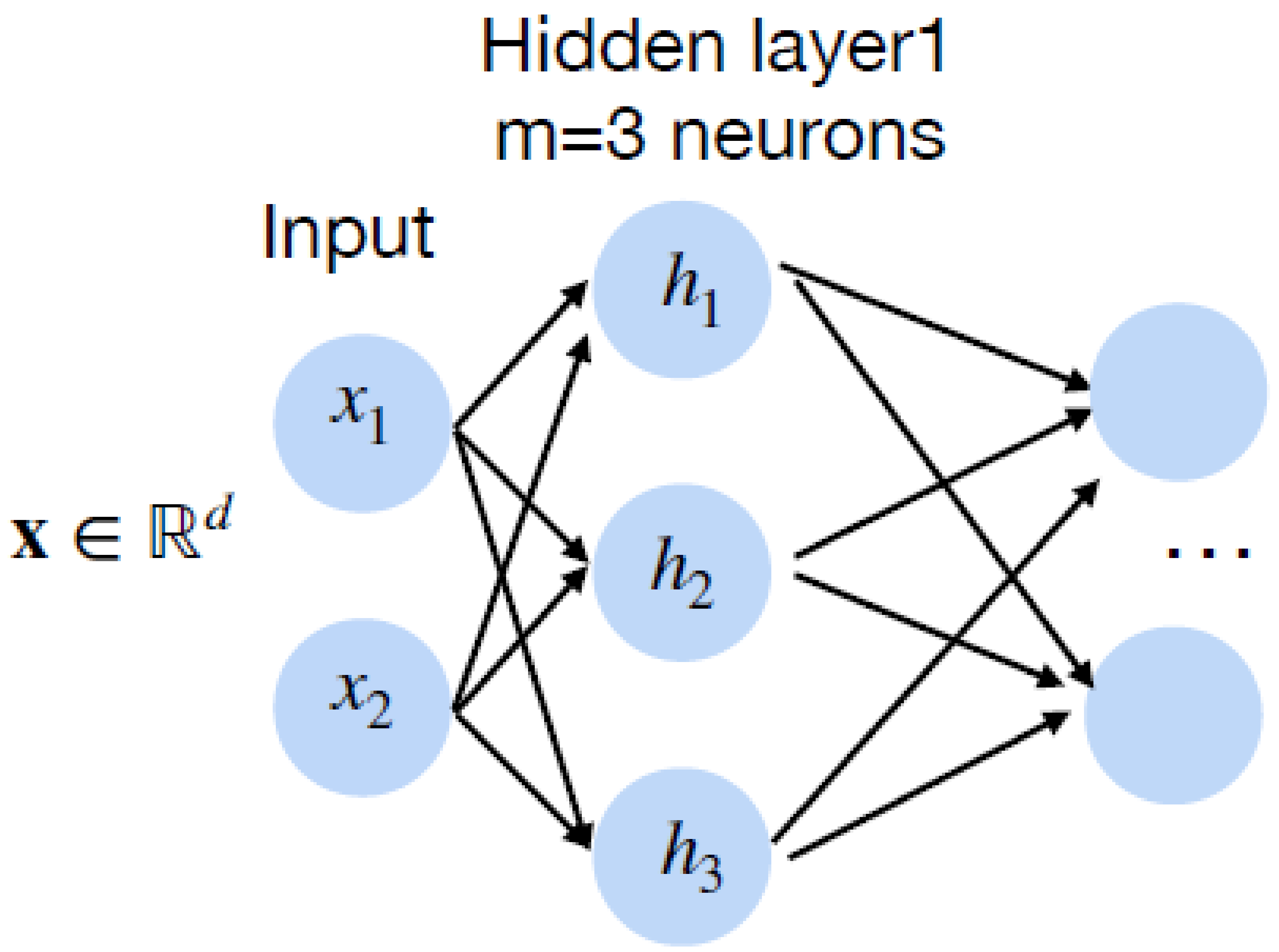
To make *E[h']* = *h*. What is "?" ?

A. h

B. h/p

C. h/(1-p)

D. h(1-p)



Quiz Break, Q4.1:

In standard dropout regularization, with dropout probability p, each intermediate activation h is replaced by a random variable h' as: $h' = \begin{cases} 0 & \text{with probability } p \\ ? & \text{otherwise} \end{cases}.$

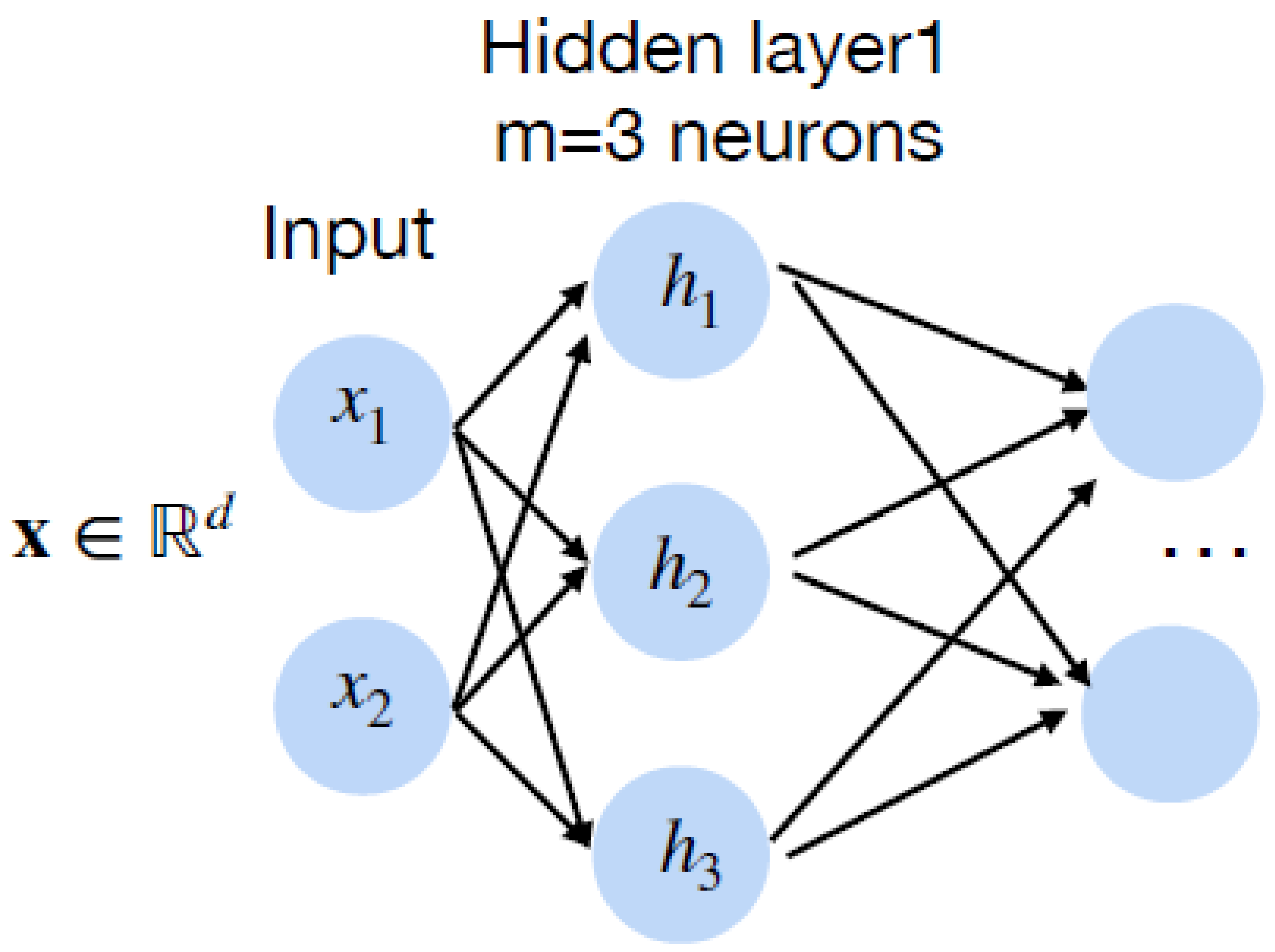
To make *E[h']* = *h*. What is "?" ?

A. h

B. h/p

C. h/(1-p)

D. h(1-p)



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