

Announcements

- No homework due to midterms
- If you believe you are taking an alternate or accommodated exam, you should have received an email with information; if you haven't, please reach out immediately via email

- Class roadmap:

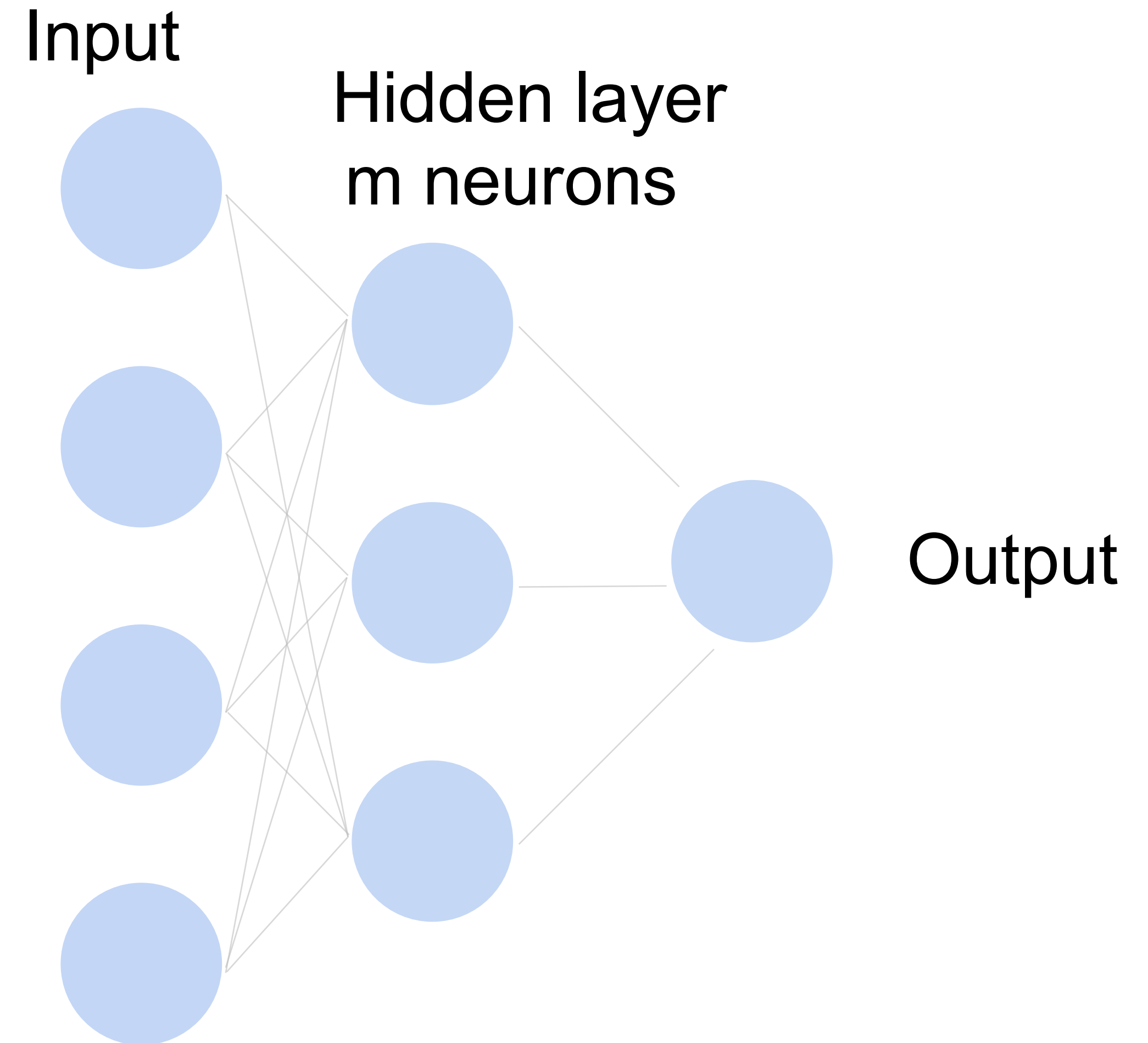
Neural Networks III
Deep Learning I
Deep Learning II

How to train a neural network?

Update the weights W to minimize the loss function

$$L = \frac{1}{|D|} \sum_{(\mathbf{x}, y) \in D} \ell(\mathbf{x}, y)$$

Use gradient descent!



Gradient Descent

- Choose a learning rate $\eta > 0$
- Initialize the model parameters $\mathbf{w}_0$
- For $t = 1, 2, \dots$

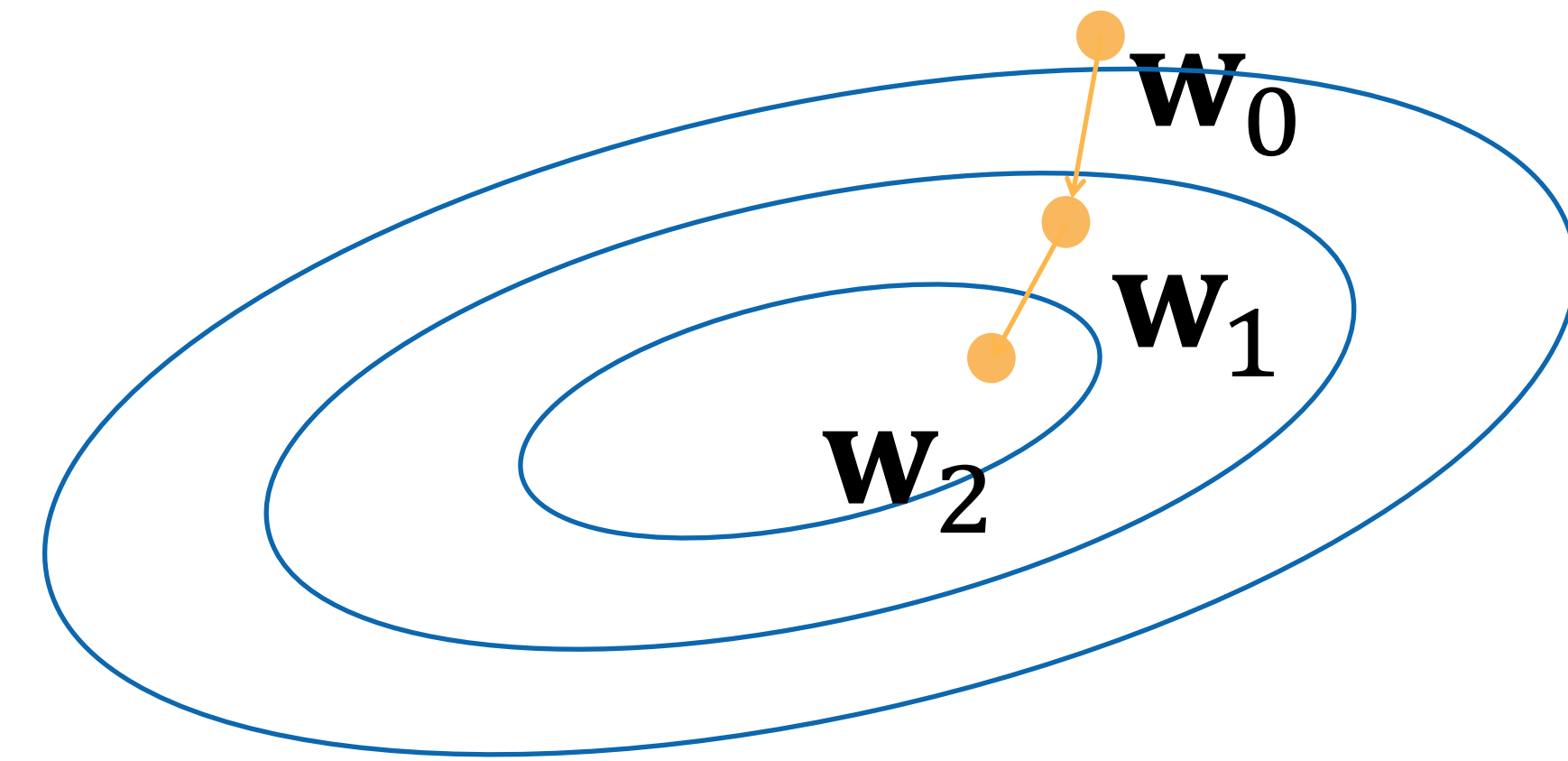
- Update parameters:

$$\begin{aligned}\mathbf{w}_t &= \mathbf{w}_{t-1} - \eta \frac{\partial L}{\partial \mathbf{w}_{t-1}} \\ &= \mathbf{w}_{t-1} - \eta \frac{1}{|D|} \sum_{(\mathbf{x}, y) \in D} \frac{\partial \ell(\mathbf{x}, y)}{\partial \mathbf{w}_{t-1}}\end{aligned}$$

D can be very large. Expensive per iteration

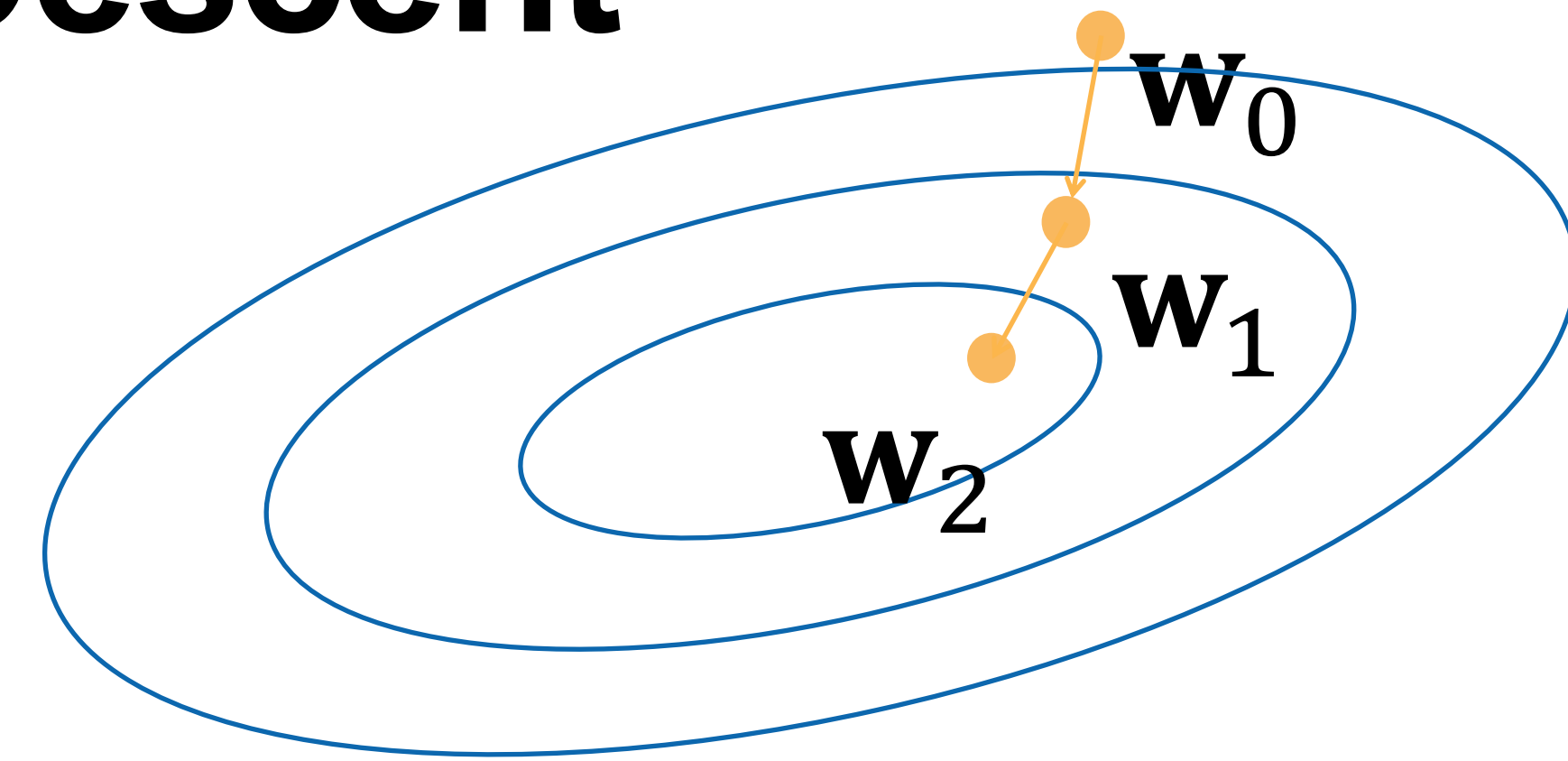
The gradient w.r.t. all parameters is obtained by concatenating the partial derivatives w.r.t. each parameter

- Repeat until convergence



Minibatch Stochastic Gradient Descent

- Choose a learning rate $\eta > 0$
- Initialize the model parameters $\mathbf{w}_0$
- For $t = 1, 2, \dots$

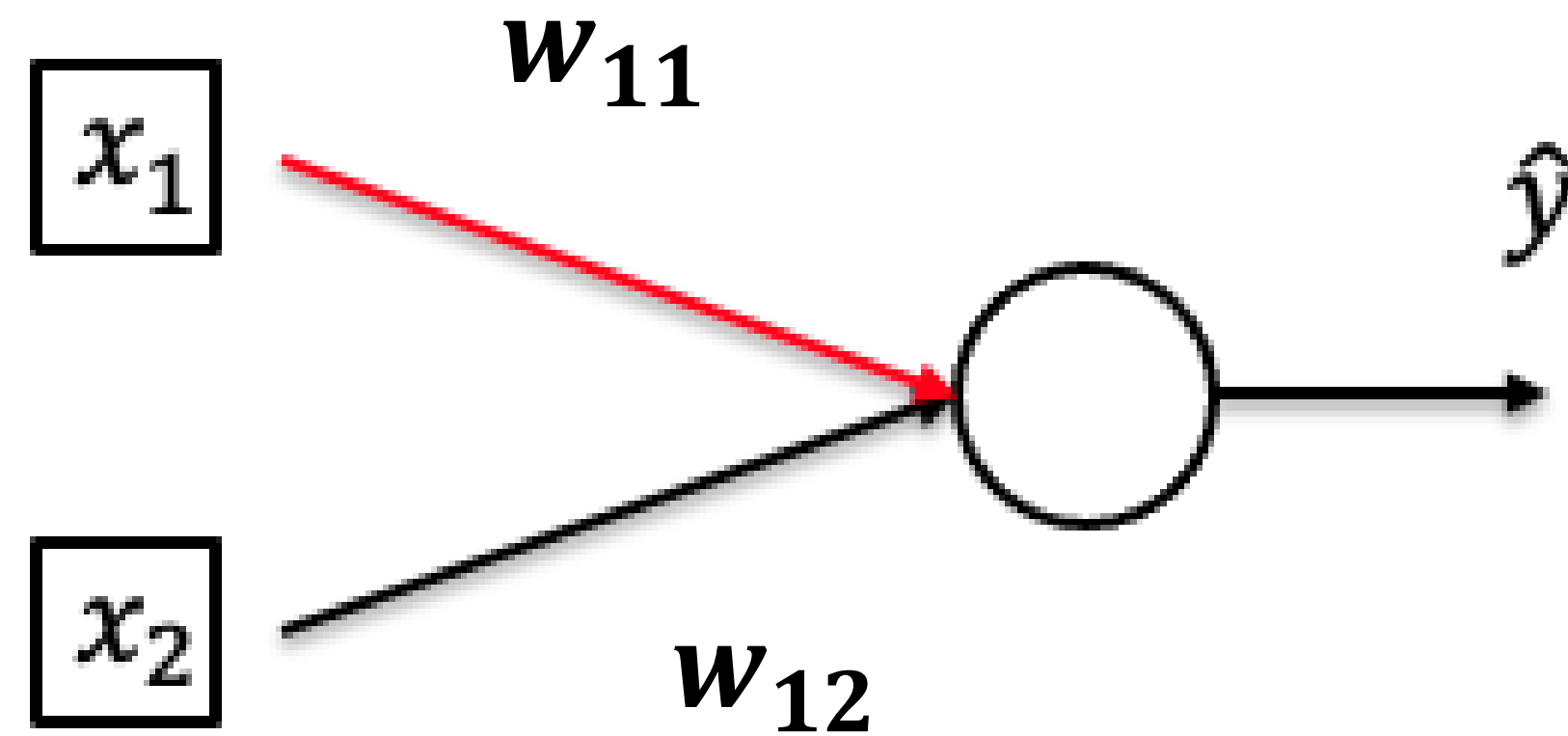


- Randomly sample a subset (mini-batch) $B \subset D$
- Update parameters:

$$\mathbf{w}_t = \mathbf{w}_{t-1} - \eta \frac{1}{|B|} \sum_{(\mathbf{x}, y) \in B} \frac{\partial \ell(\mathbf{x}, y)}{\partial \mathbf{w}_{t-1}}$$

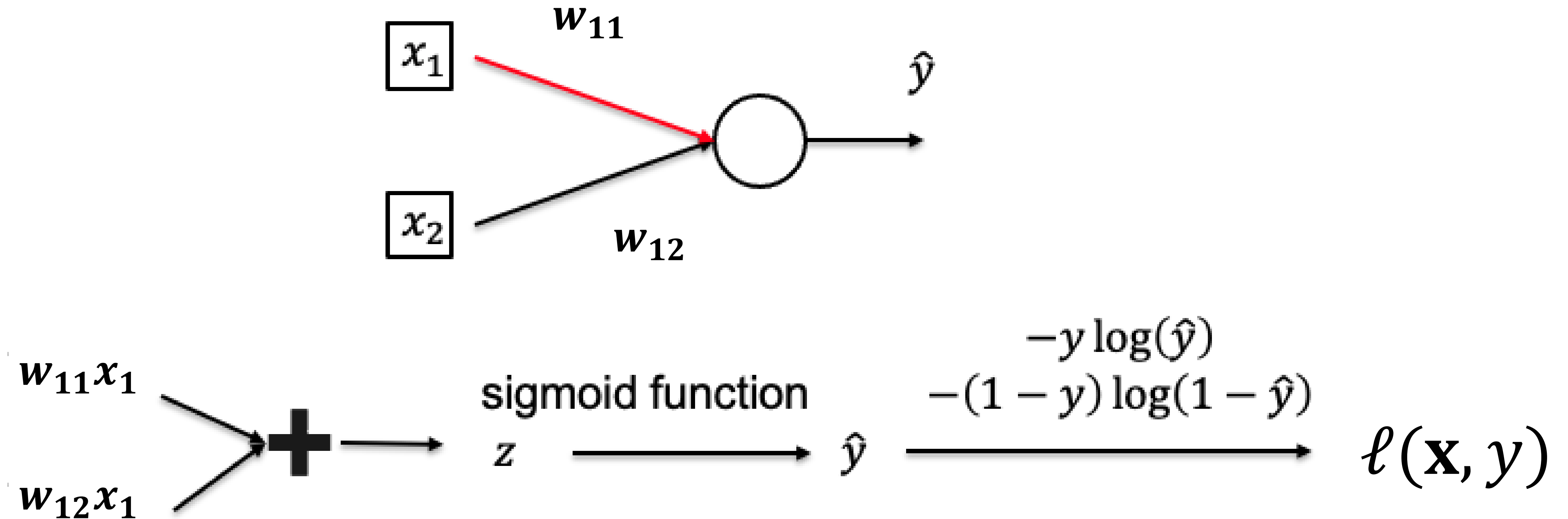
- Repeat until convergence

Calculate Gradient (on one data point)



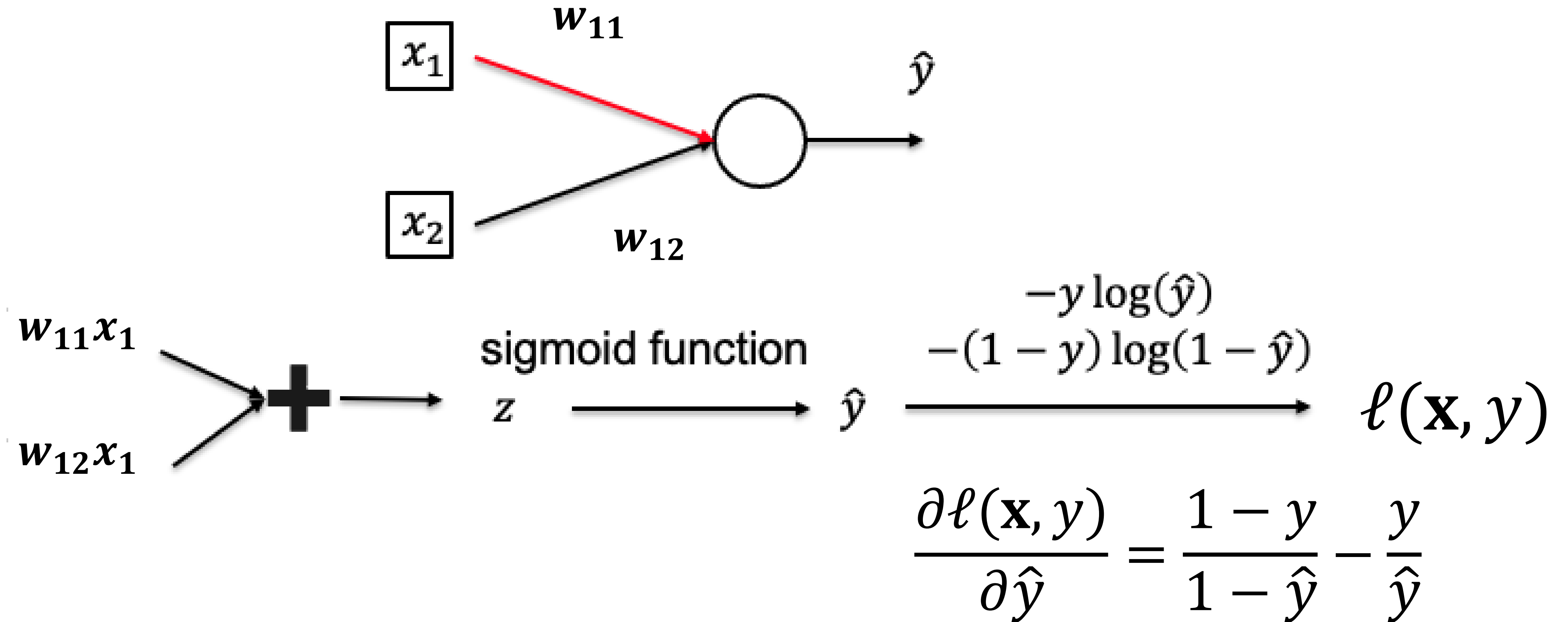
- Want to compute $\frac{\partial \ell(\mathbf{x}, y)}{\partial w_{11}}$
- Data point: $((x_1, x_2), y)$

Calculate Gradient (on one data point)



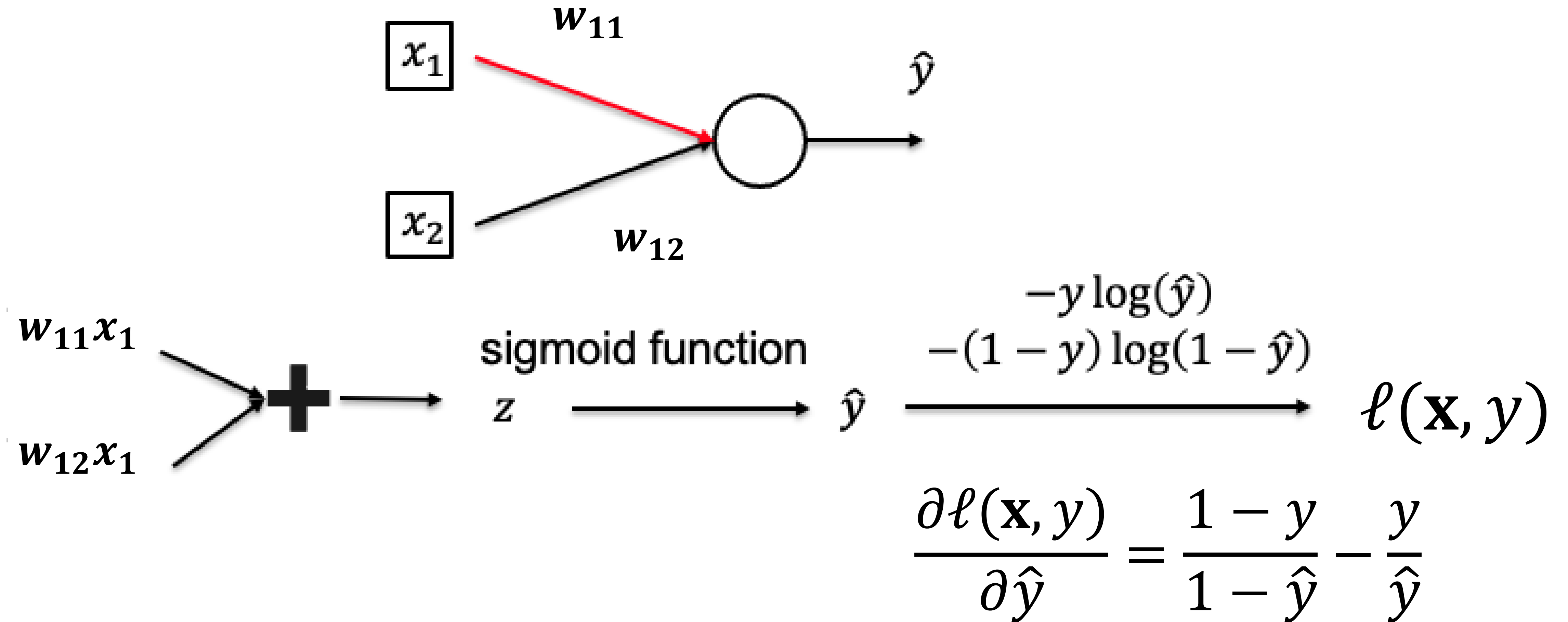
Use chain rule!

Calculate Gradient (on one data point)



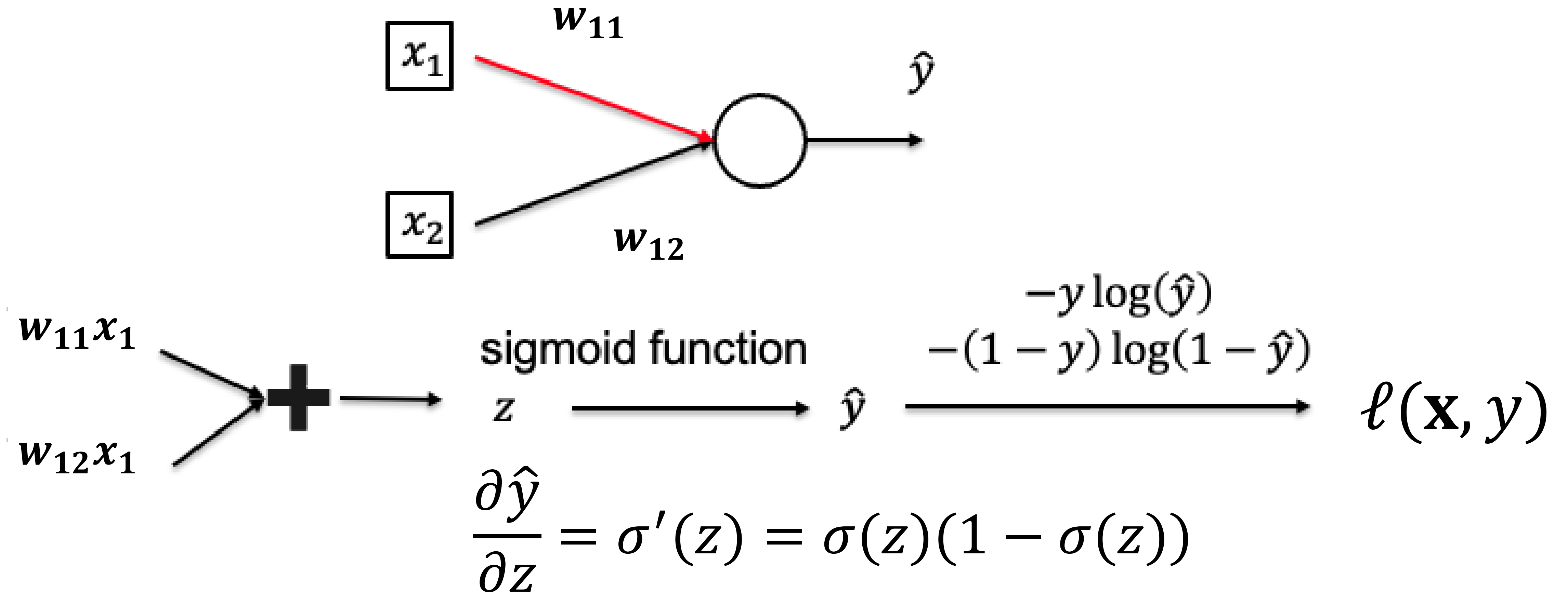
- By chain rule:
$$\frac{\partial l}{\partial w_{11}} = \frac{\partial l}{\partial \hat{y}} \frac{\partial \hat{y}}{\partial z} \frac{\partial z}{\partial w_{11}}$$

Calculate Gradient (on one data point)



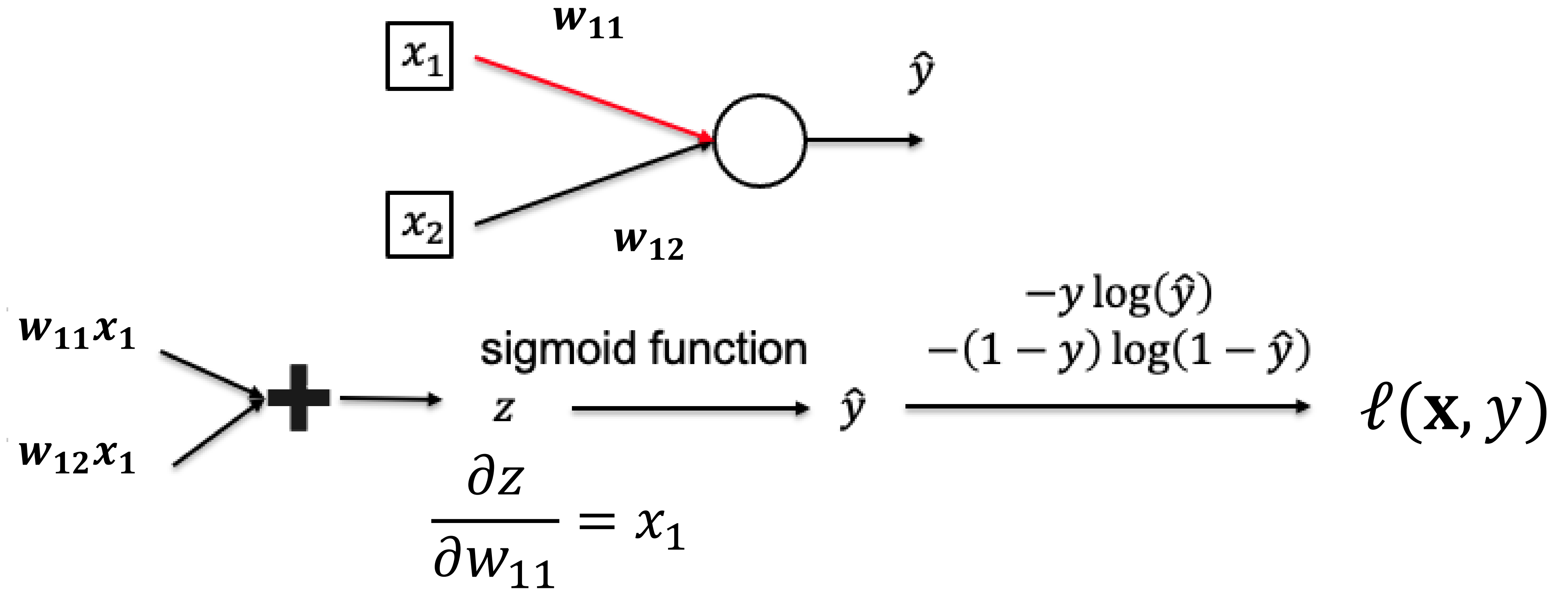
- By chain rule:
$$\frac{\partial \ell}{\partial w_{11}} = \left(\frac{1-y}{1-\hat{y}} - \frac{y}{\hat{y}} \right) \frac{\partial \hat{y}}{\partial z} \frac{\partial z}{\partial w_{11}}$$

Calculate Gradient (on one data point)



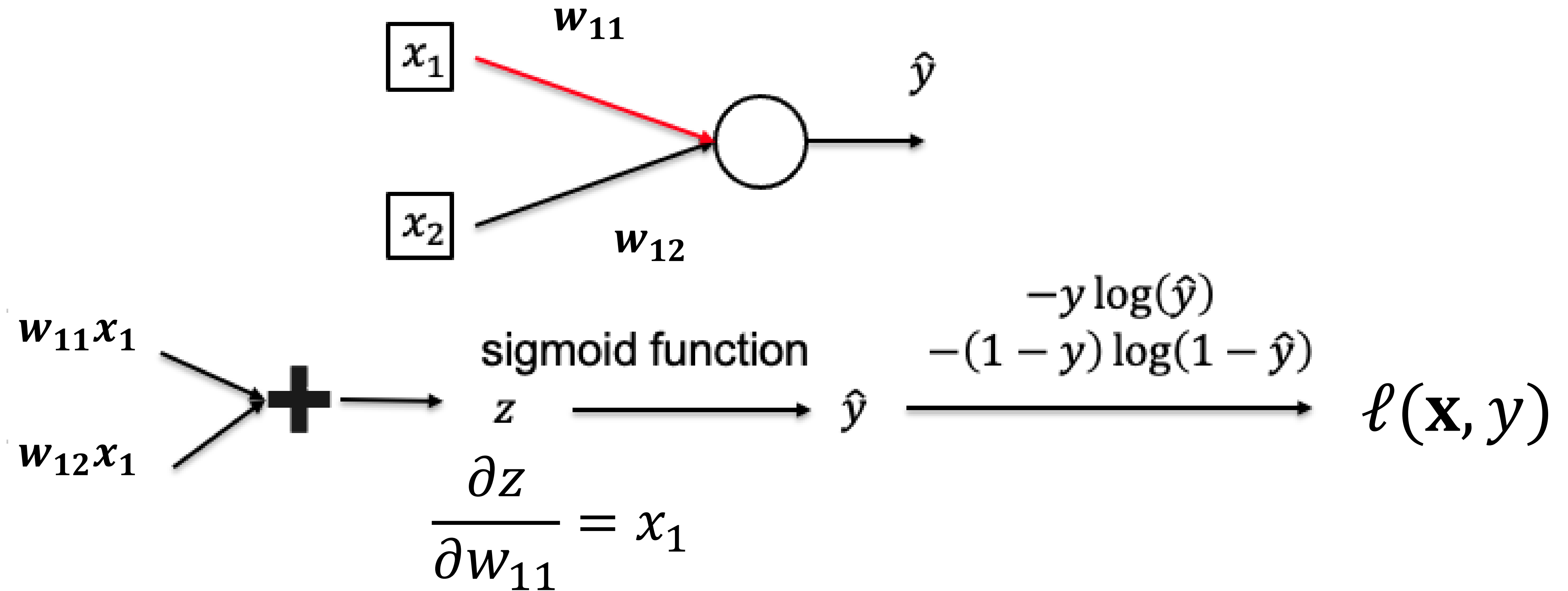
- By chain rule:
$$\frac{\partial \ell}{\partial w_{11}} = \left(\frac{1-y}{1-\hat{y}} - \frac{y}{\hat{y}} \right) (1 - \hat{y}) \frac{\partial z}{\partial w_{11}}$$

Calculate Gradient (on one data point)



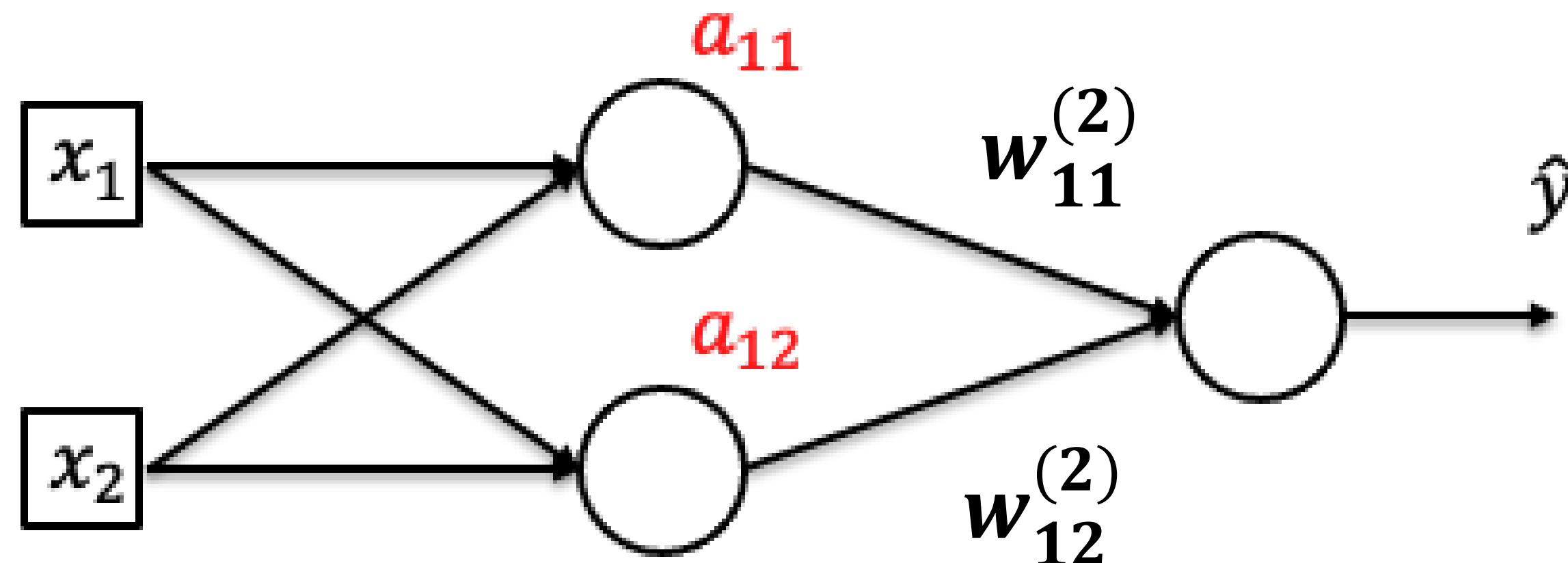
- By chain rule: $\frac{\partial \ell}{\partial w_{11}} = \left(\frac{1-y}{1-\hat{y}} - \frac{y}{\hat{y}} \right) (1-\hat{y})x_1$

Calculate Gradient (on one data point)

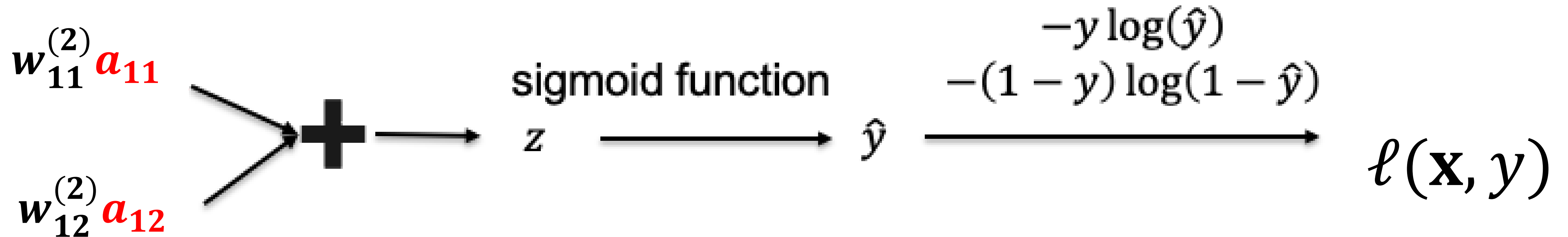


- By chain rule: $\frac{\partial l}{\partial w_{11}} = (\hat{y} - y)x_1$

Calculate Gradient (on one data point)

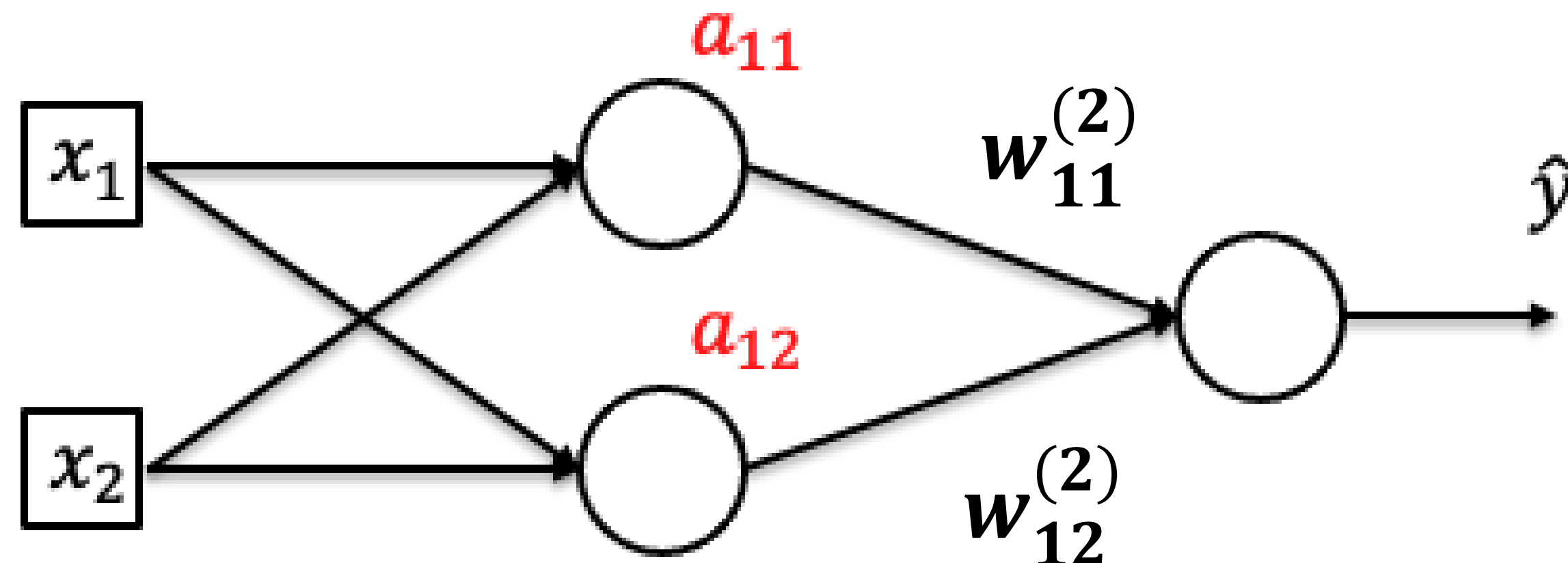


Make it deeper

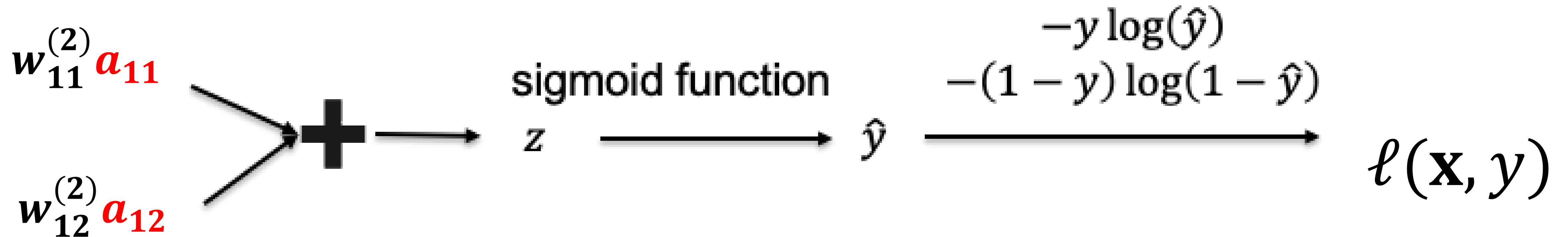


- By chain rule: $\frac{\partial l}{\partial a_{11}} = \frac{\partial l}{\partial \hat{y}} \frac{\partial \hat{y}}{\partial z} \frac{\partial z}{\partial a_{11}} = (\hat{y} - y) w_{11}^{(2)}$

Calculate Gradient (on one data point)

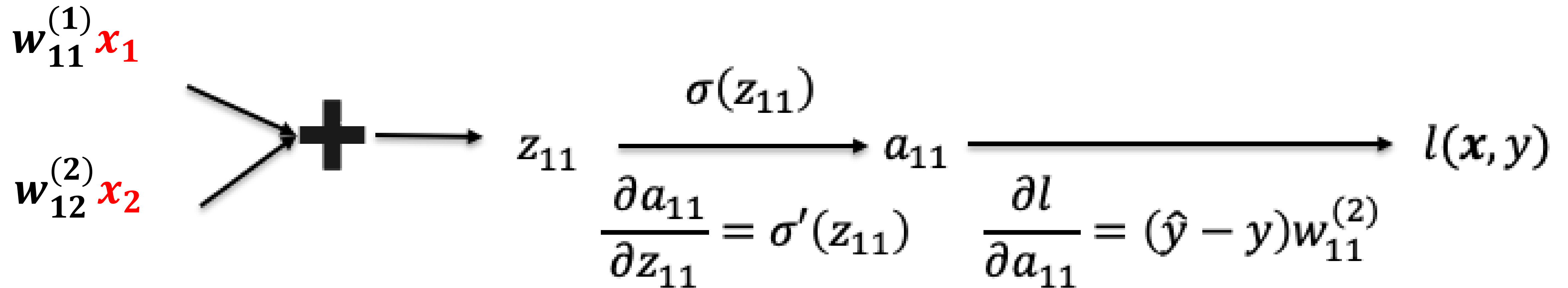
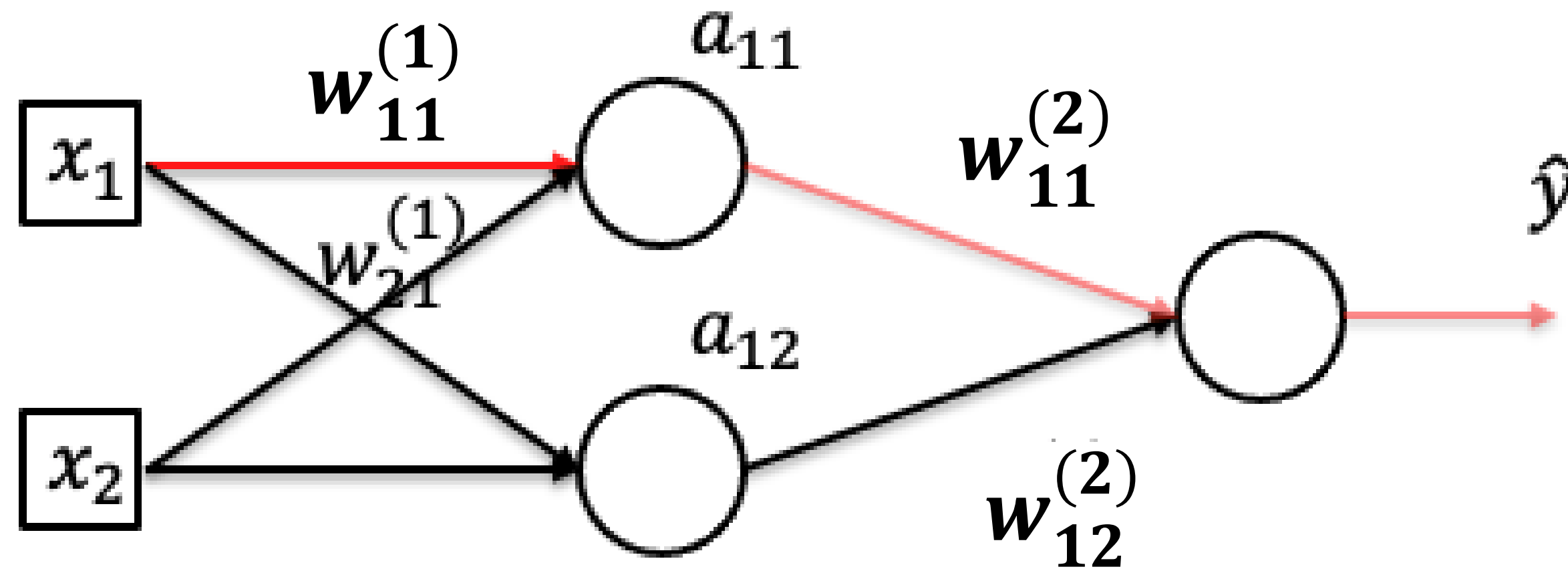


Make it deeper



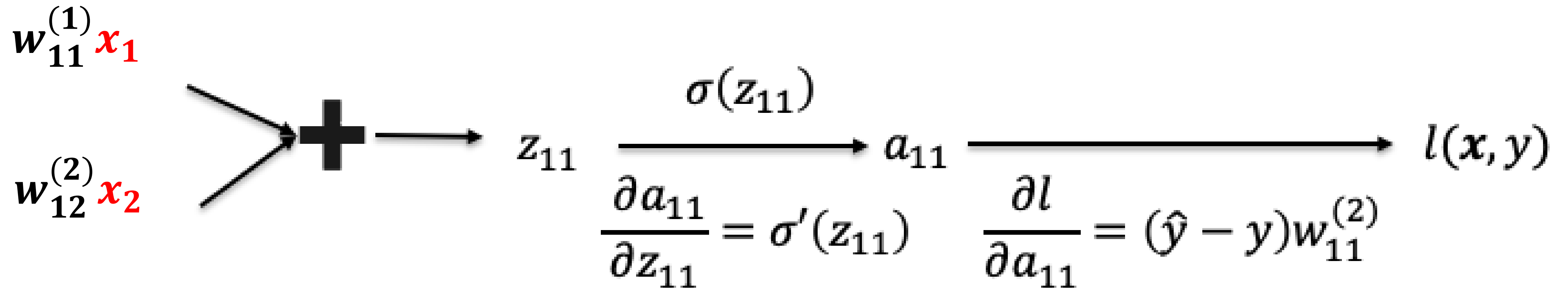
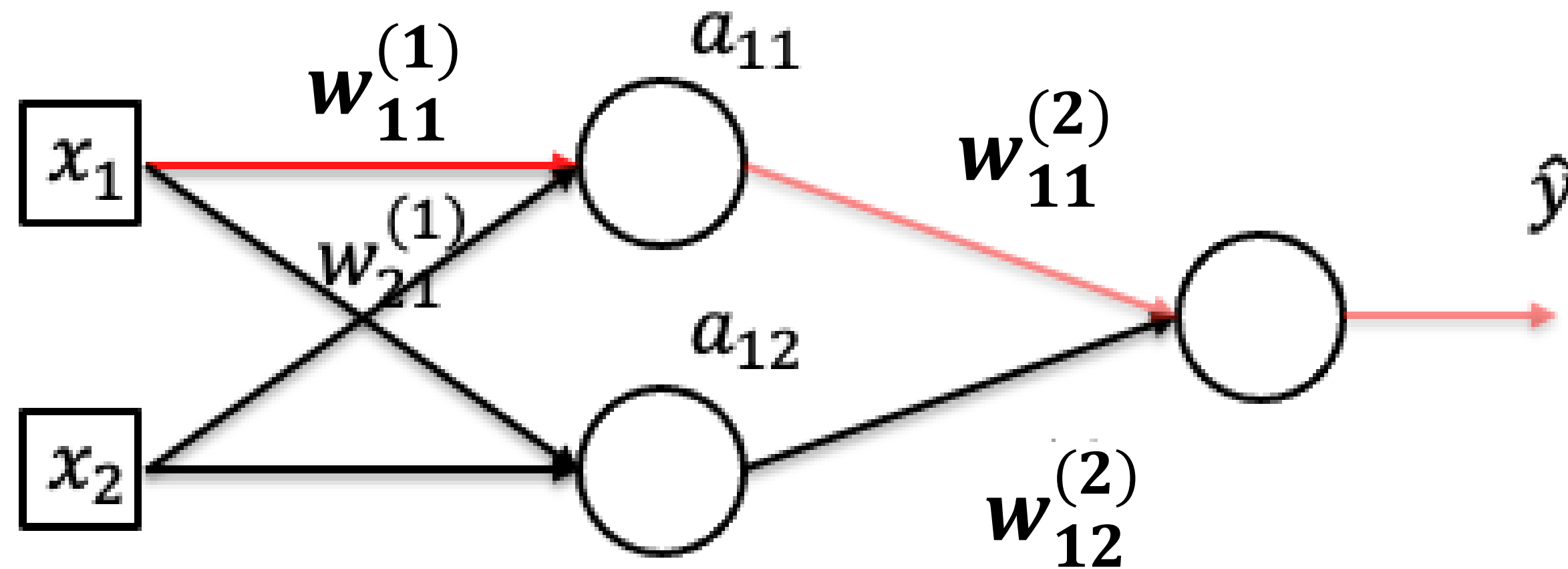
- By chain rule: $\frac{\partial l}{\partial a_{12}} = \frac{\partial l}{\partial \hat{y}} \frac{\partial \hat{y}}{\partial z} \frac{\partial z}{\partial a_{12}} = (\hat{y} - y) w_{12}^{(2)}$

Calculate Gradient (on one data point)



- By chain rule:
$$\frac{\partial l}{\partial w_{11}^{(1)}} = \frac{\partial l}{\partial a_{11}} \frac{\partial a_{11}}{\partial z_{11}} \frac{\partial z_{11}}{\partial w_{11}^{(1)}} = (\hat{y} - y)w_{11}^{(2)} a_{11}(1 - a_{11}) \frac{\partial z_{11}}{\partial w_{11}^{(1)}}$$

Calculate Gradient (on one data point)

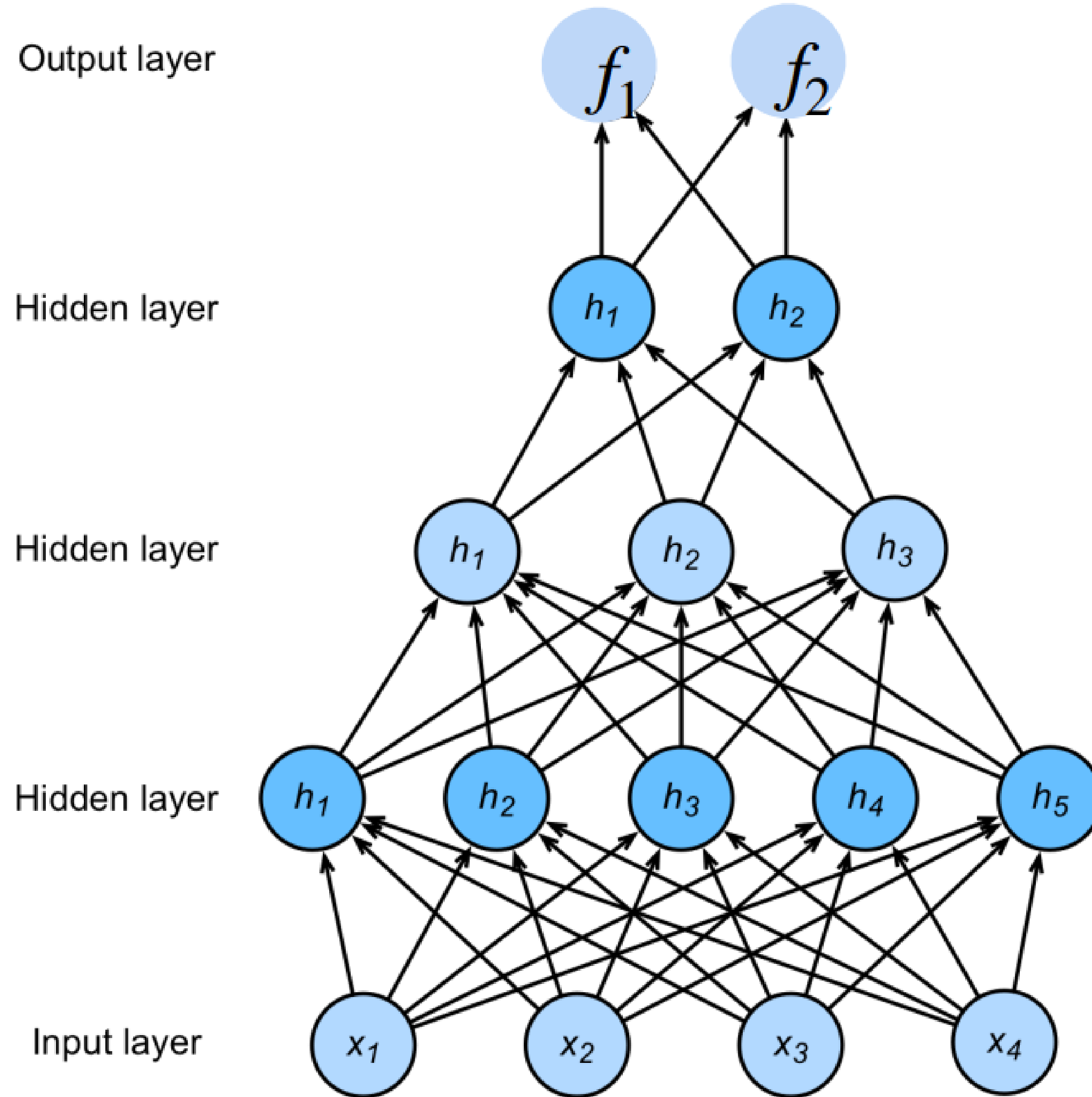


- By chain rule:
$$\frac{\partial l}{\partial w_{11}^{(1)}} = \frac{\partial l}{\partial a_{11}} \frac{\partial a_{11}}{\partial z_{11}} \frac{\partial z_{11}}{\partial w_{11}^{(1)}} = (\hat{y} - y)w_{11}^{(2)}a_{11}(1 - a_{11})x_1$$



Neural Networks as a Computational Graph

Deep neural networks (DNNs)



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

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

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

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

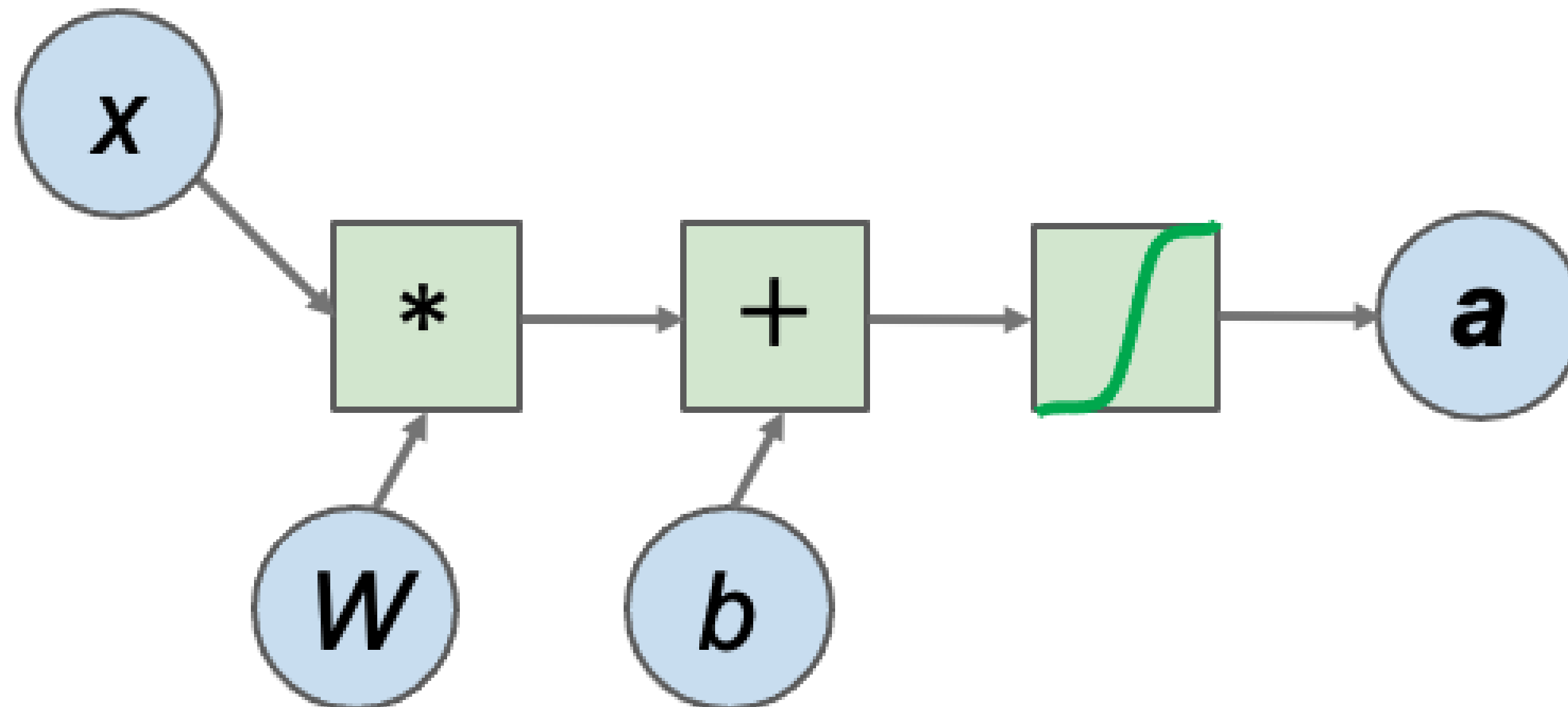
$$\mathbf{p} = \text{softmax}(\mathbf{f})$$

**NNs are composition
of nonlinear
functions**

Neural networks as variables + operations

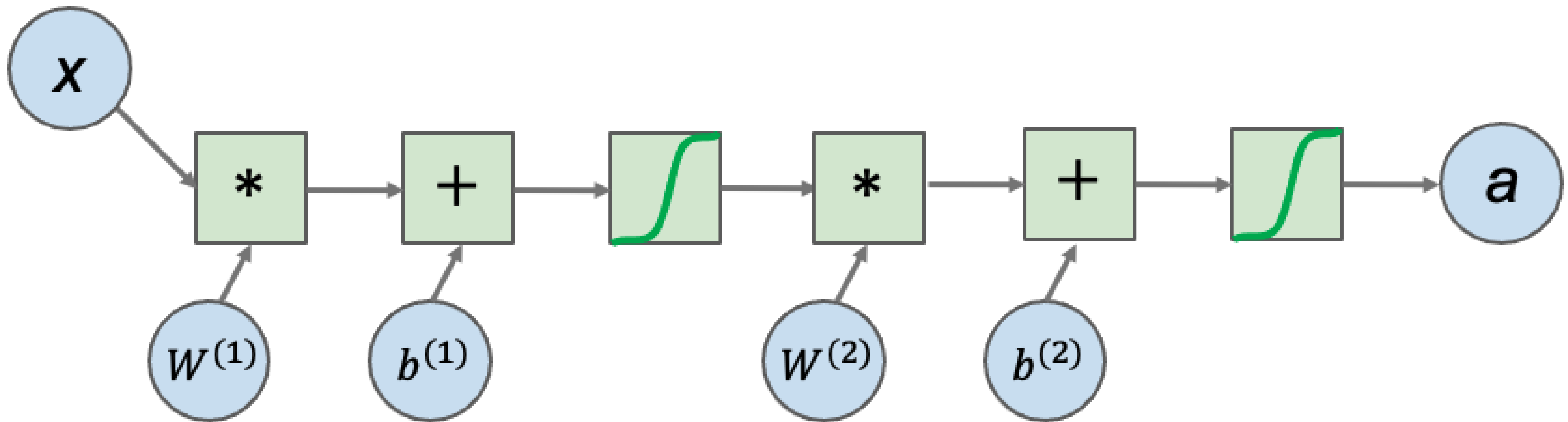
$$\mathbf{a} = \text{sigmoid}(\mathbf{W}\mathbf{x} + \mathbf{b})$$

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



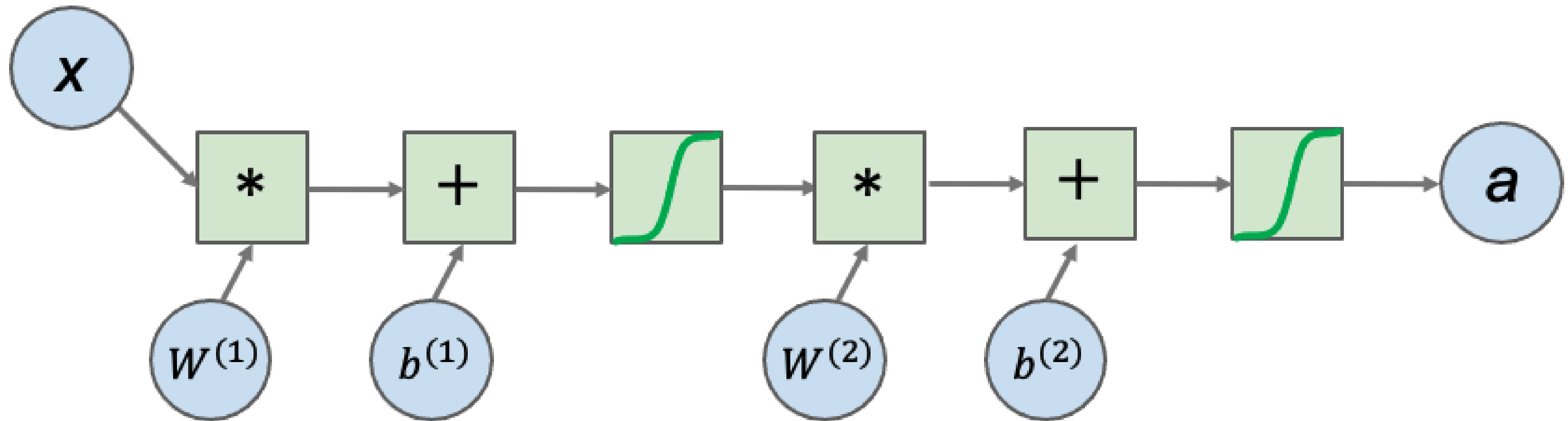
Neural networks as a computational graph

- A two-layer neural network



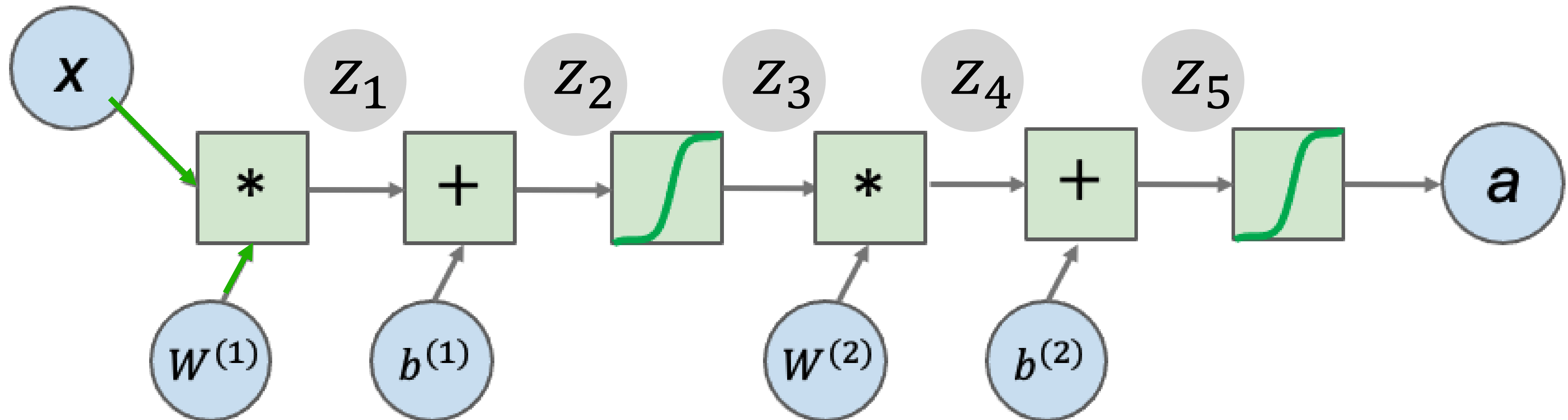
Neural networks as a computational graph

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



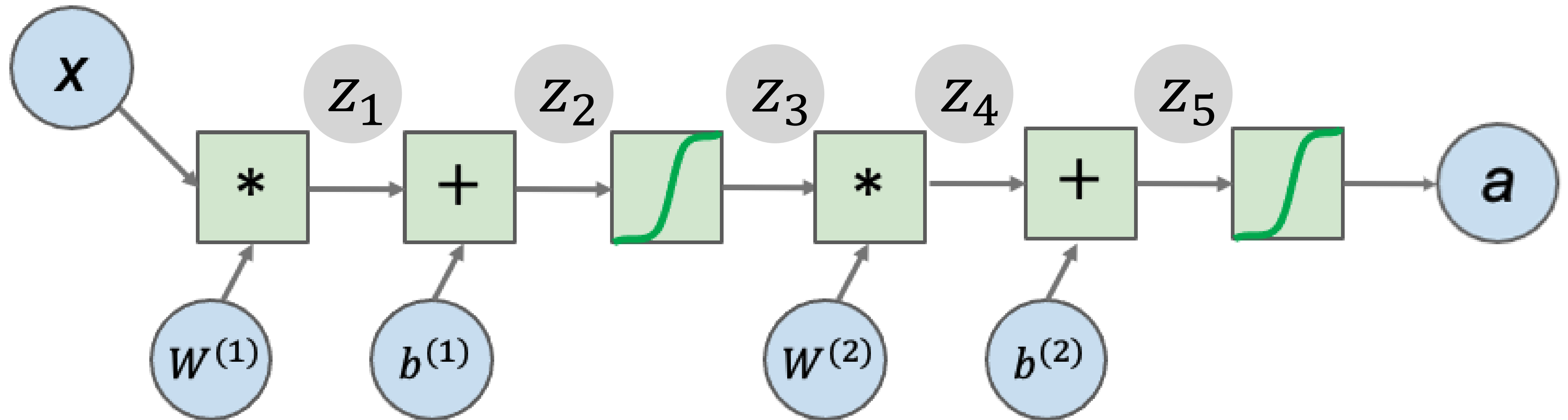
Neural networks: forward propagation

- A two-layer neural network
- Intermediate variables Z



Neural networks: backward propagation

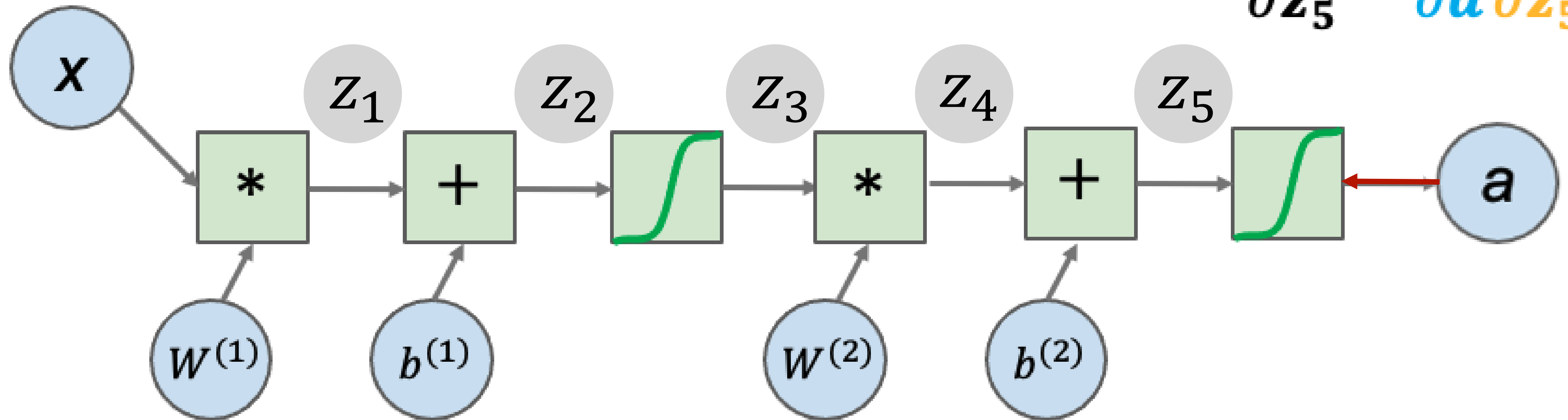
- A two-layer neural network
- Assuming forward propagation is done
- Minimize a **loss function** L



Neural networks: backward propagation

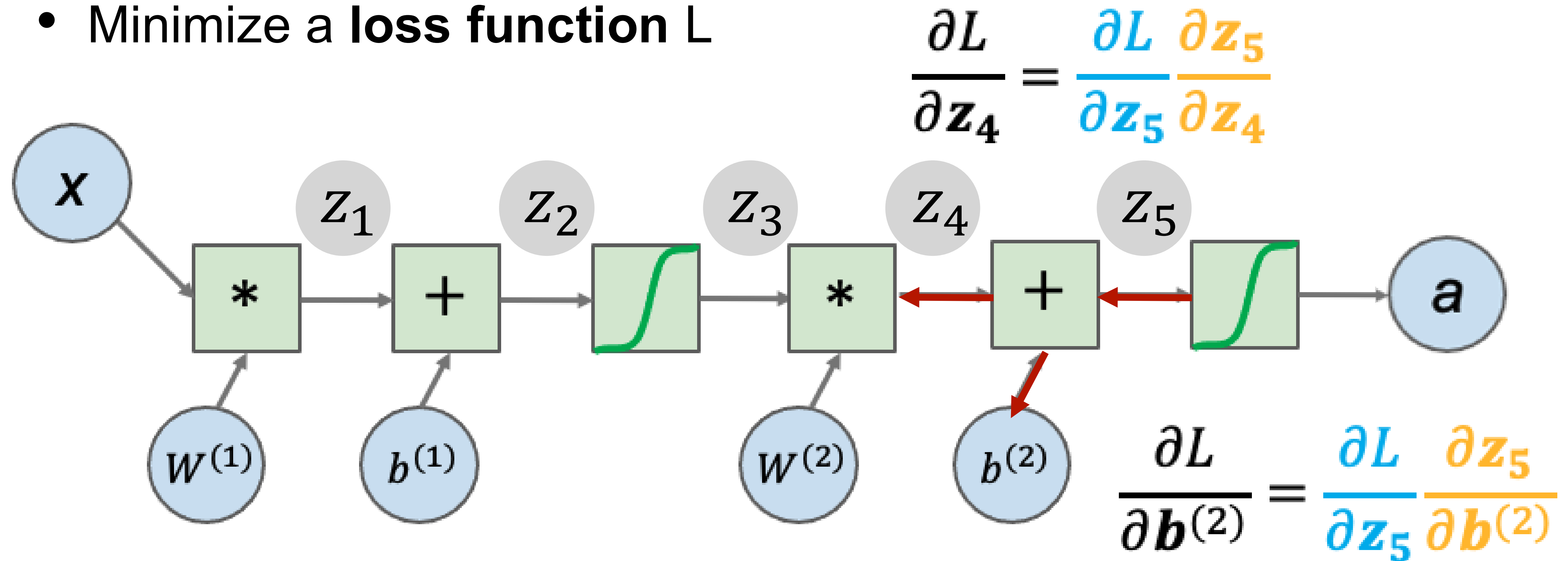
- A two-layer neural network
- Assuming forward propagation is done
- Minimize a **loss function** L

$$\frac{\partial L}{\partial \mathbf{z}_5} = \frac{\partial L}{\partial \mathbf{a}} \frac{\partial \mathbf{a}}{\partial \mathbf{z}_5}$$



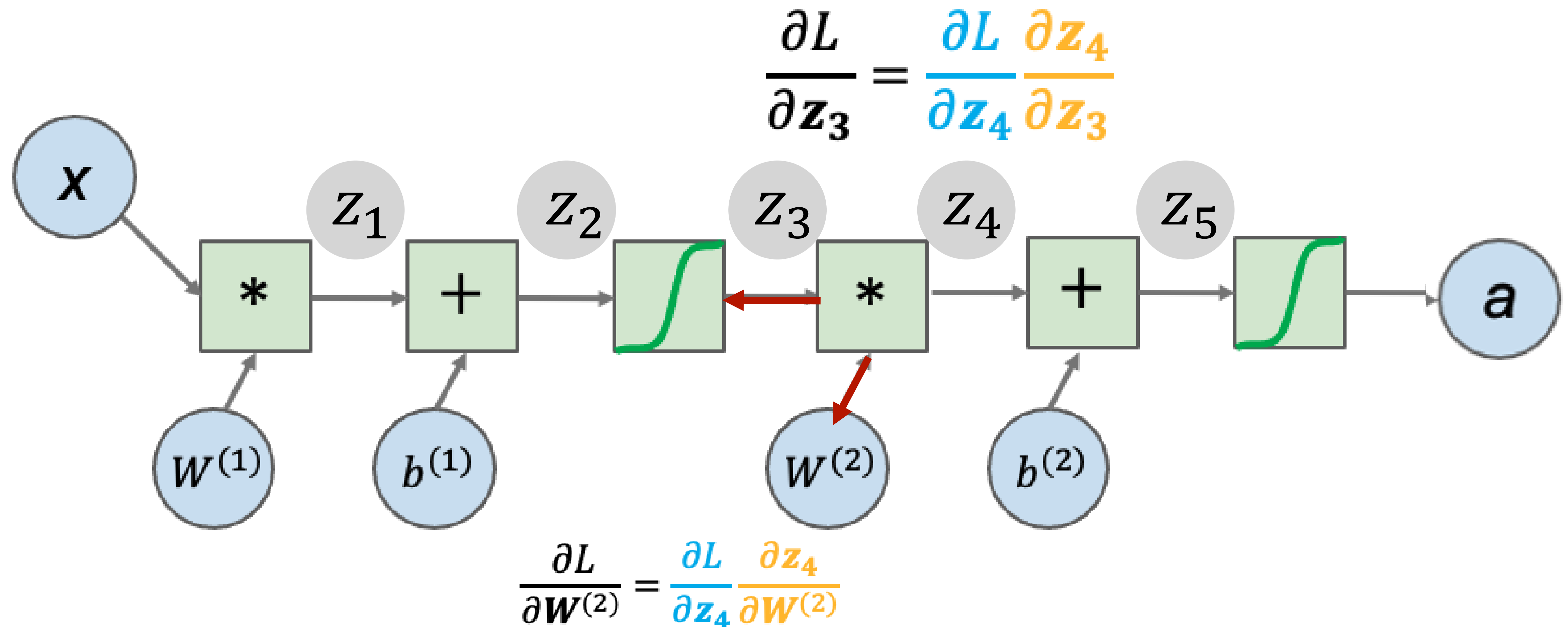
Neural networks: backward propagation

- 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



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 like
    # doing so you pass a Tensor of input data to the Module
    # a Tensor of output data.
    y_pred = model(xx)

    # Compute and print loss. We pass Tensors containing the
    # values of y, and the loss function returns a Tensor of
    # 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 respect to
    # parameters of the model. Internally, the parameters of the
    # 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 parameter
    # 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



Numerical Stability

Gradients for Neural Networks

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

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

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

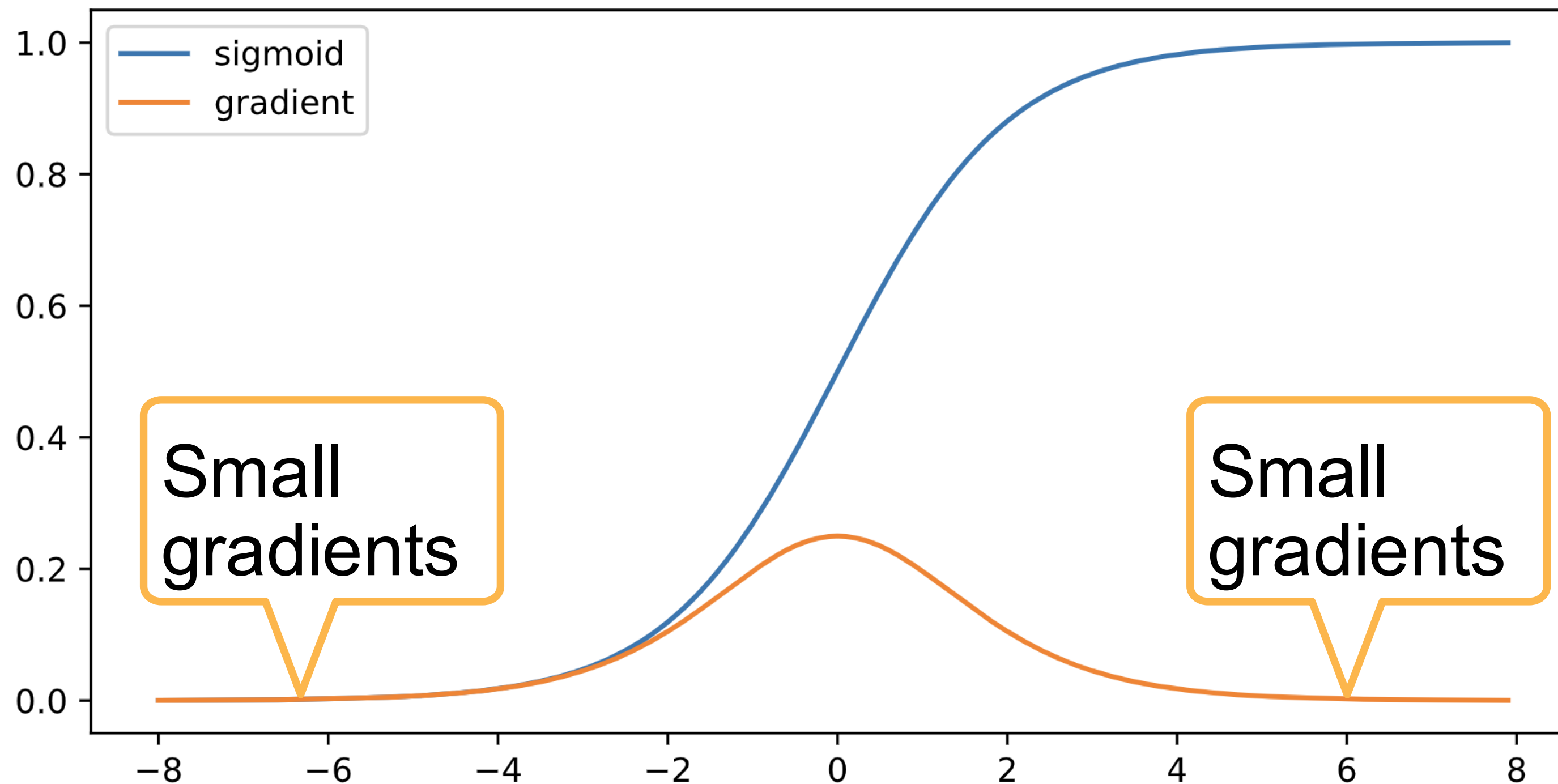


$$0.8^{100} \approx 2 \times 10^{-10}$$

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 we 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 $[1e-6, 1e3]$
- Different architectures (e.g., ResNet)
- Normalize
 - Batch Normalization, Gradient clipping
- Proper activation functions



Generalization & Regularization

Underfitting Overfitting

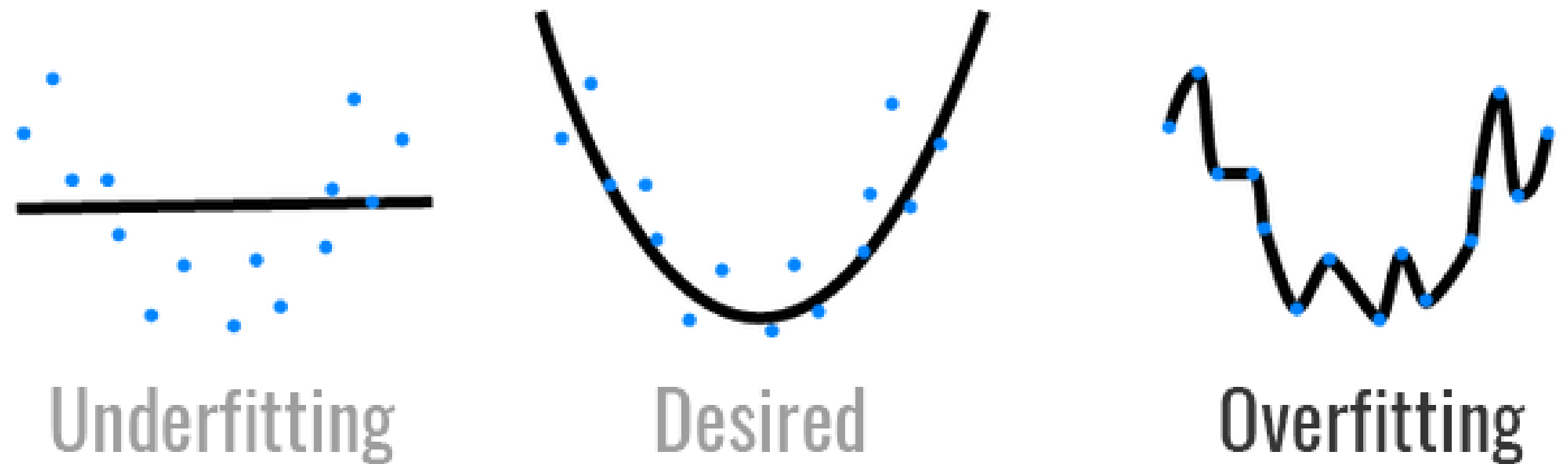
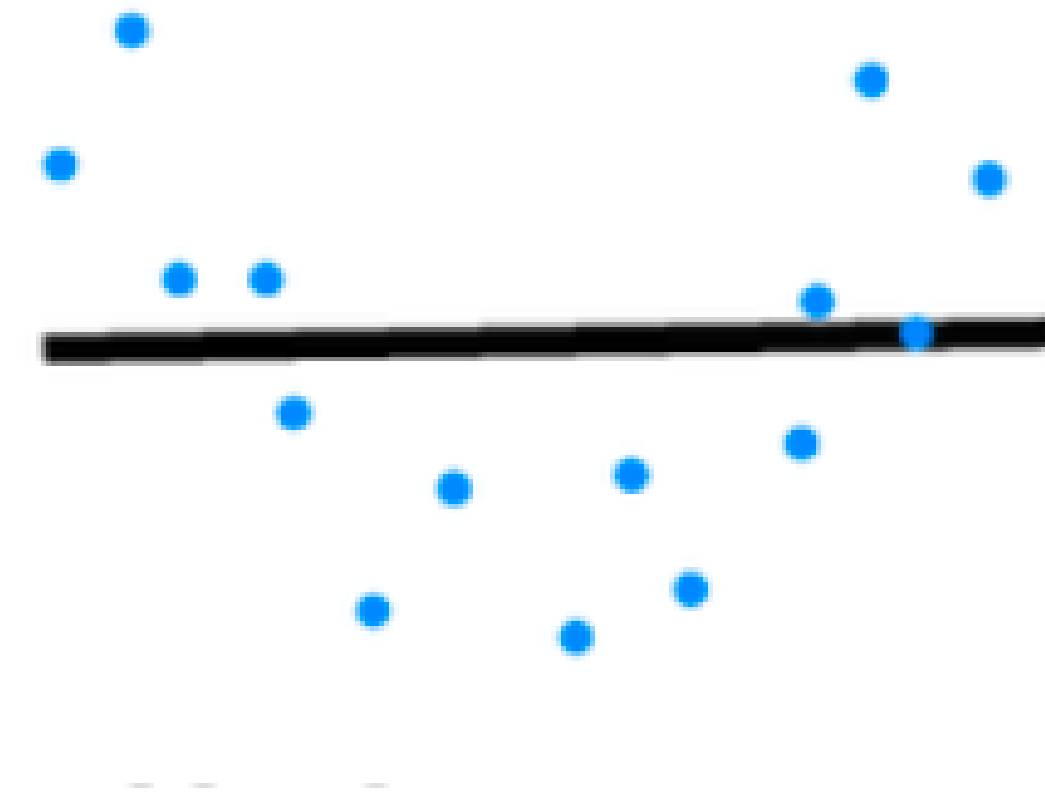


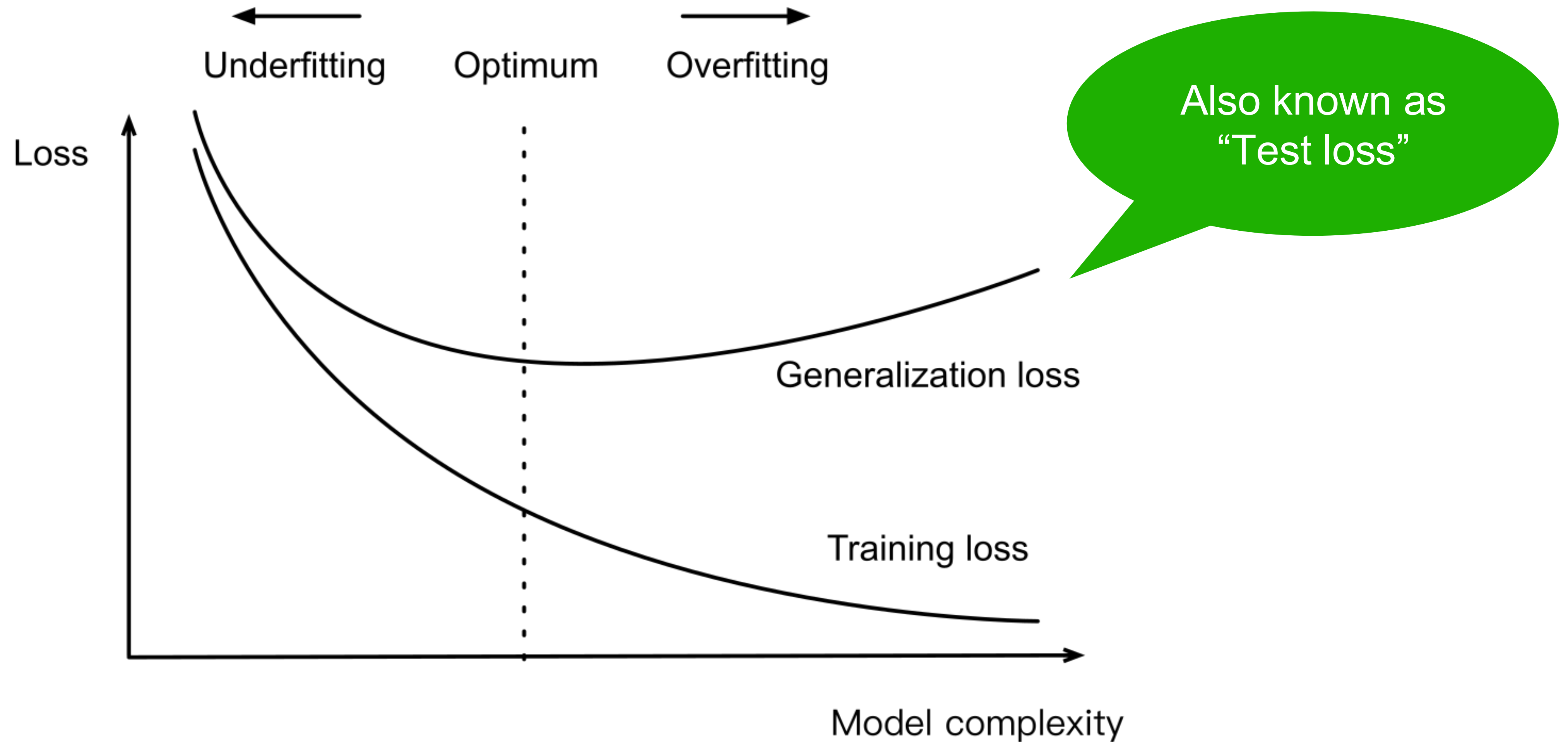
Image credit: hackernoon.com

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

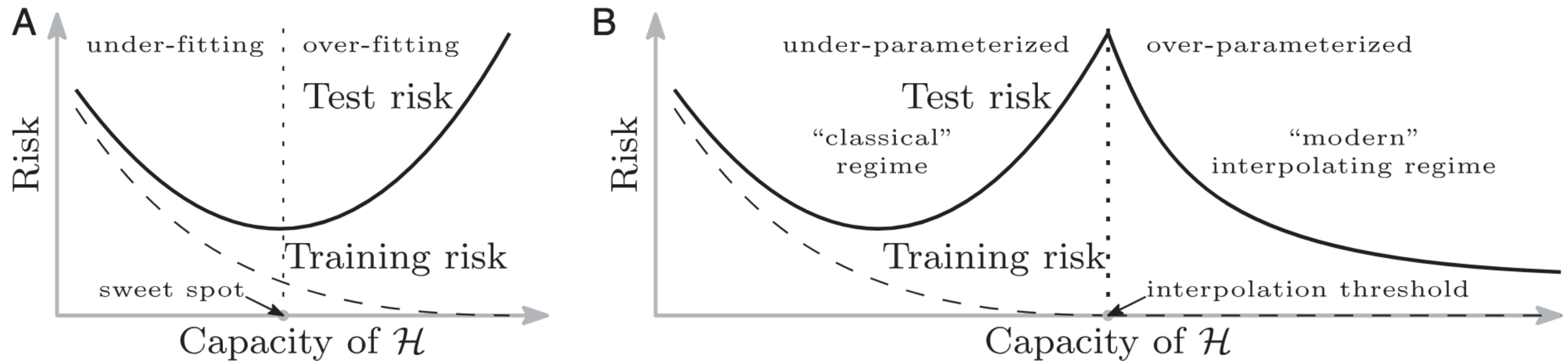


Classical View of Model Complexity



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

A Modern View: Double Descent

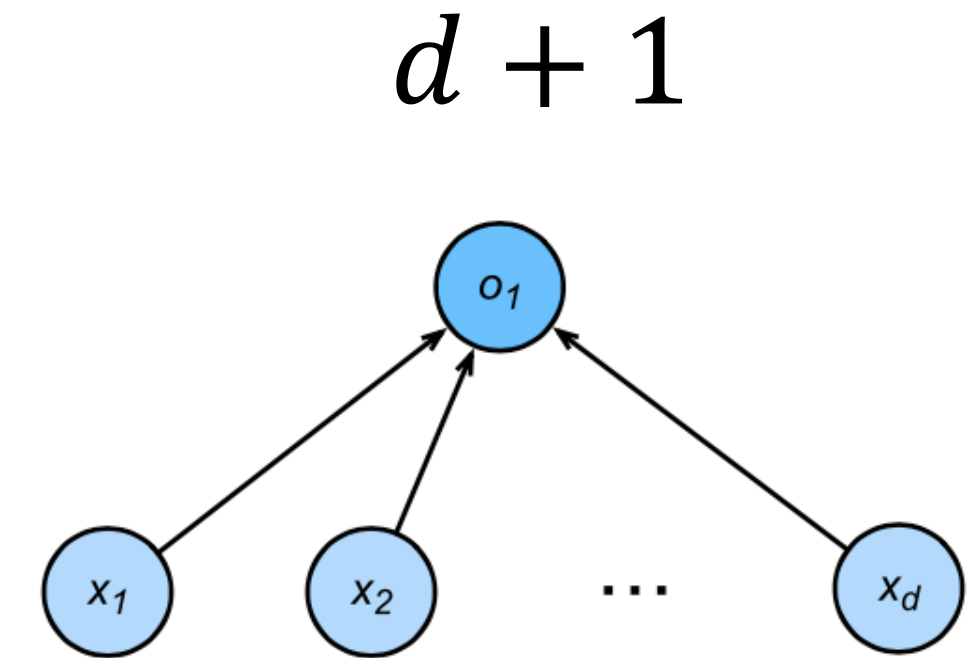


Belkin, Hsu, Ma, Mandal 2019

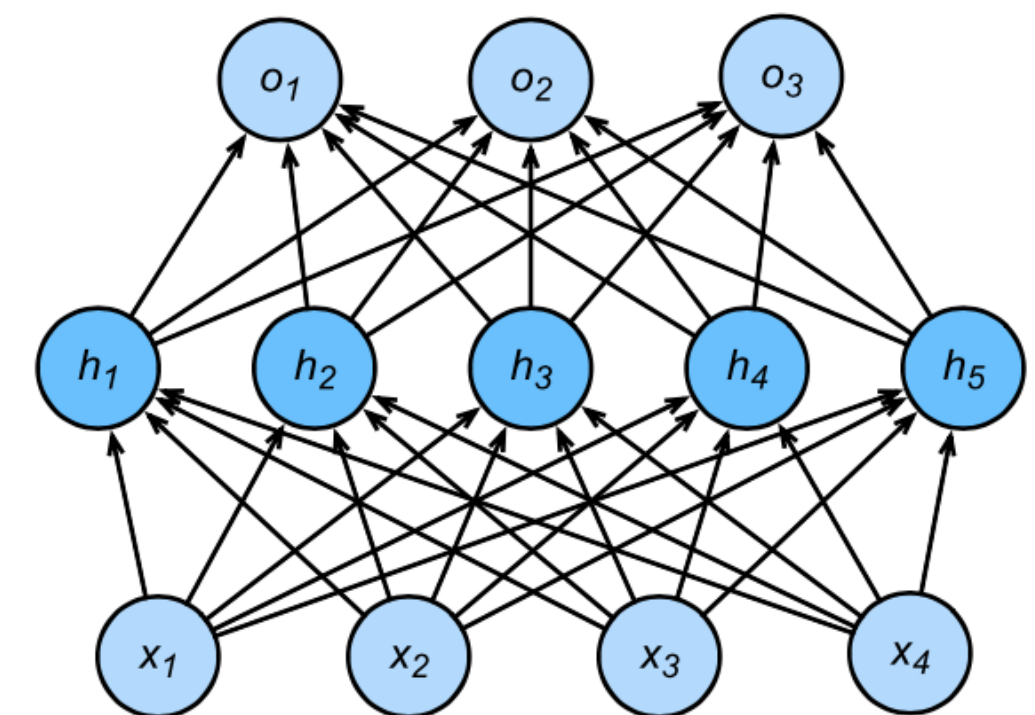
The ultimate measure is how well the network performs on new data

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

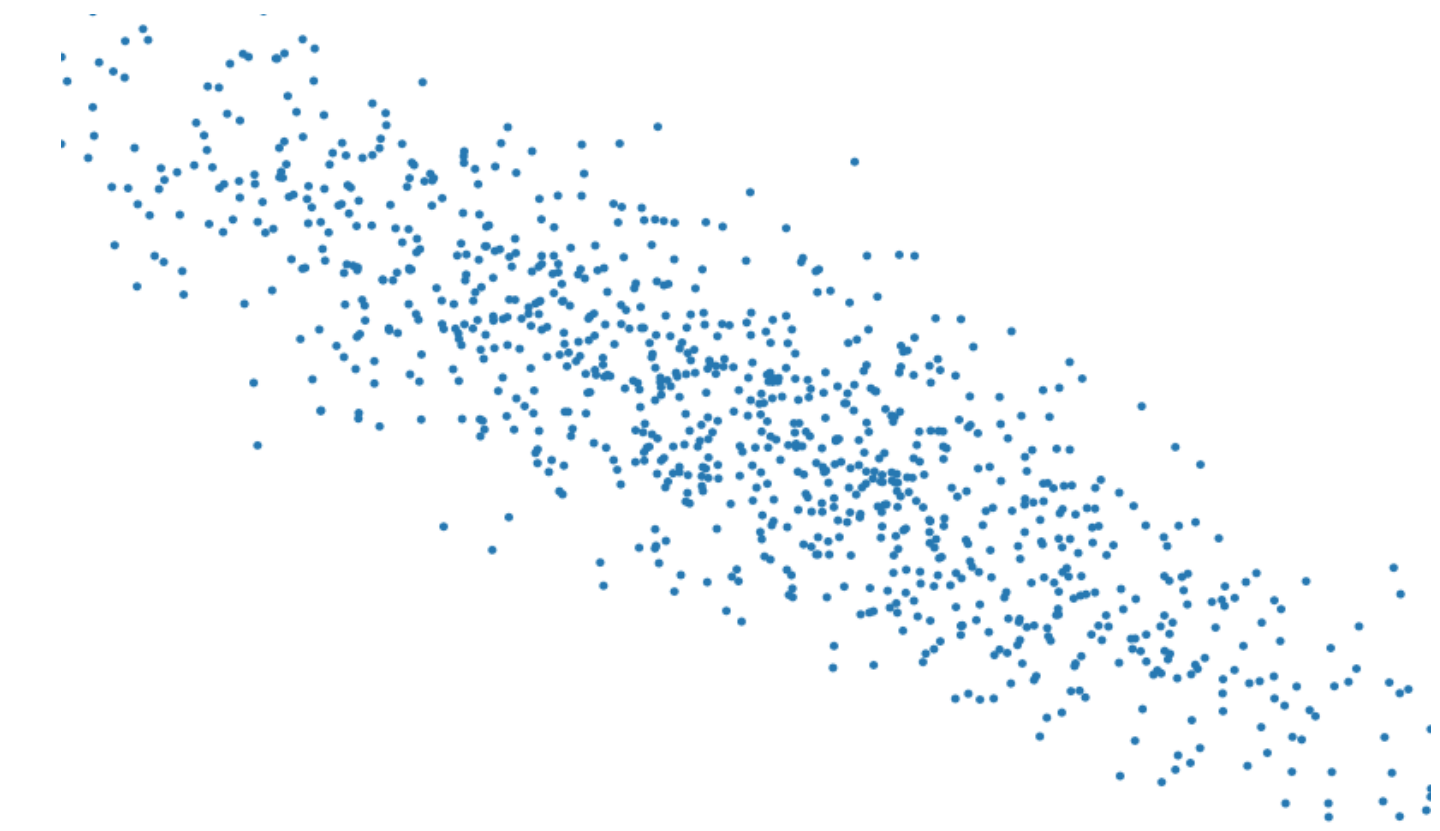


$$(d + 1)m + (m + 1)k$$



Data Complexity

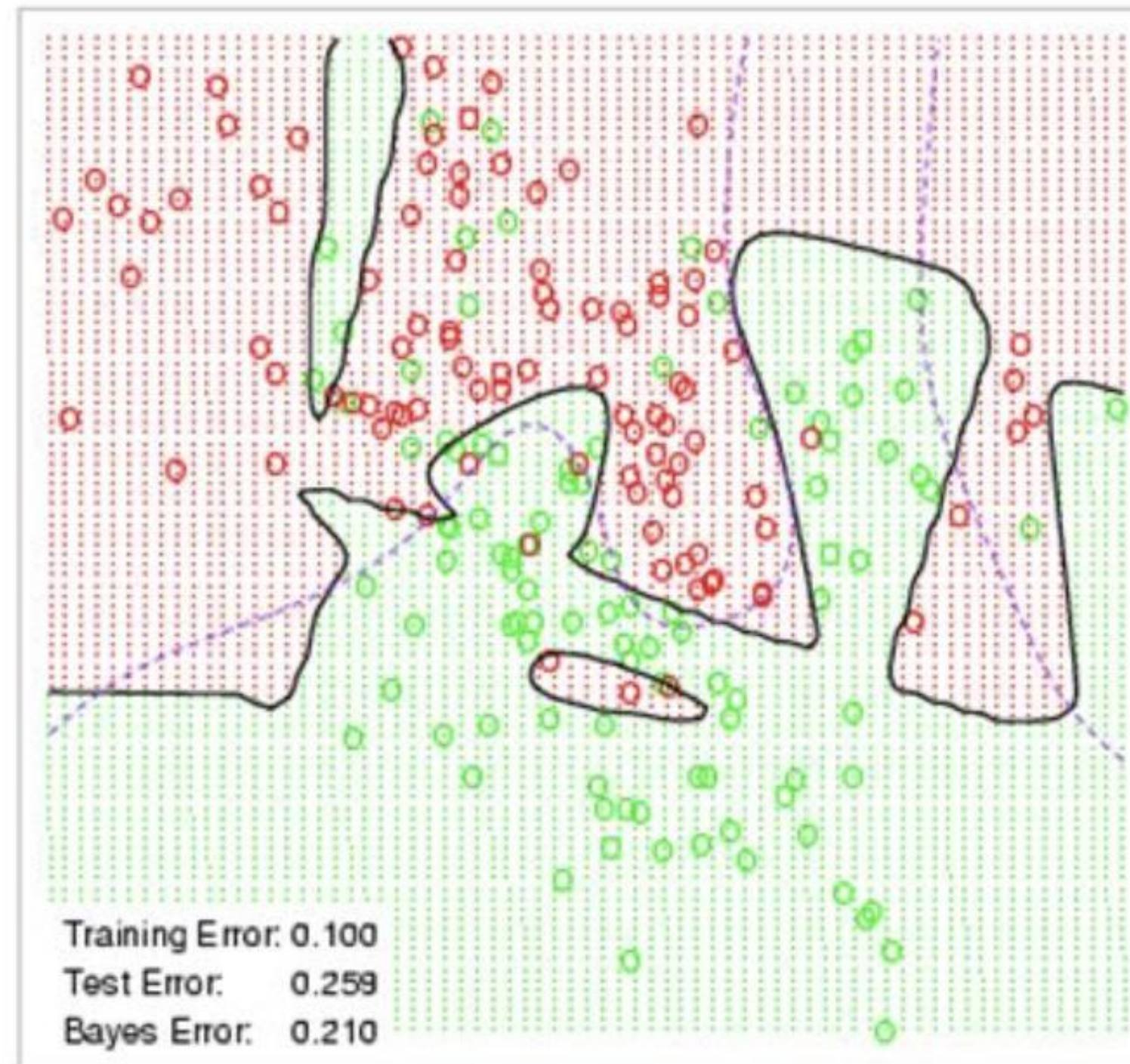
- Multiple factors matters
 - # of examples
 - # of features in each example
 - time/space structure
 - # of labels



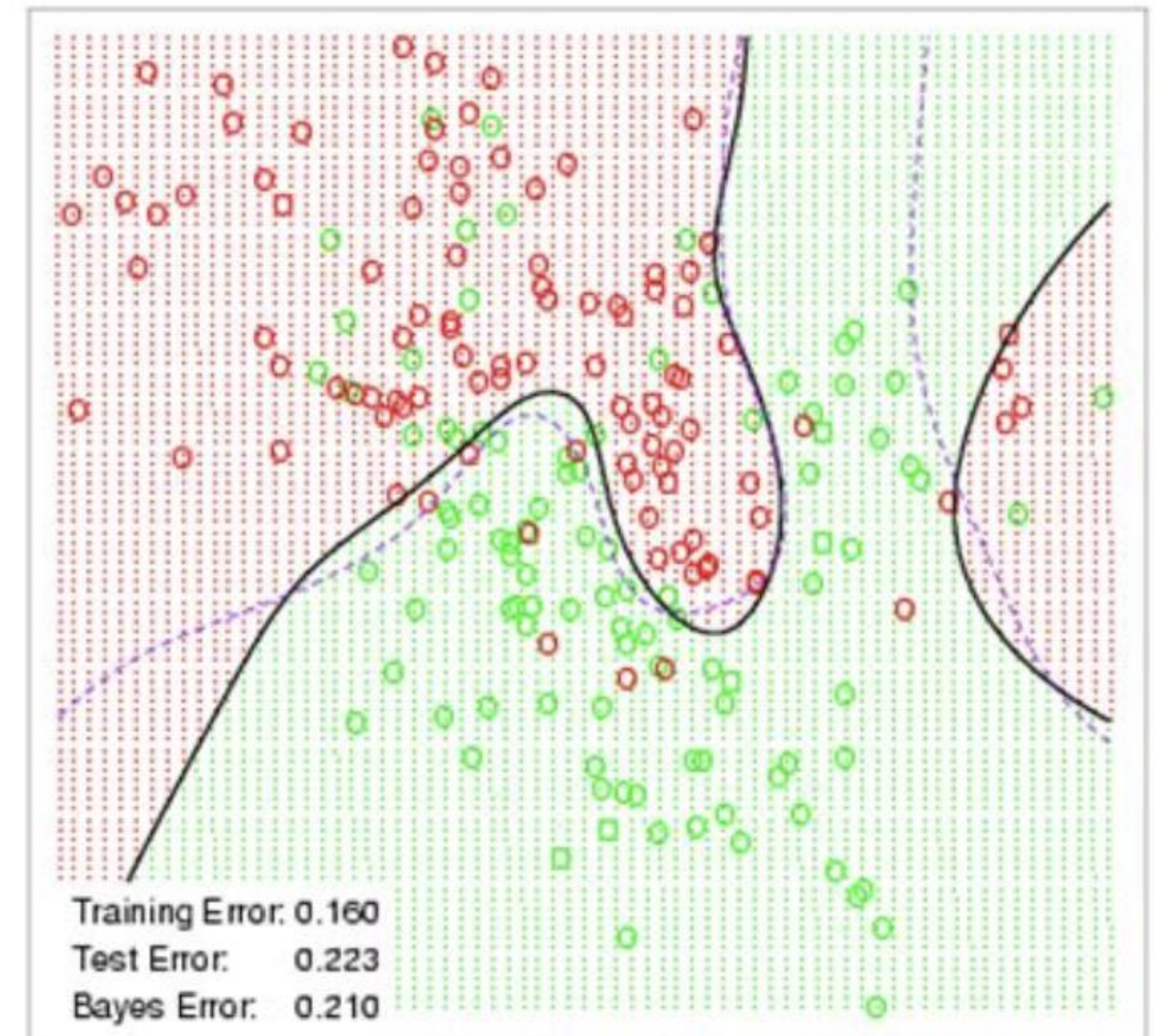
**How to regularize the model for
better generalization?**

Weight Decay

Neural Network - 10 Units, No Weight Decay



Neural Network - 10 Units, Weight Decay=0.02

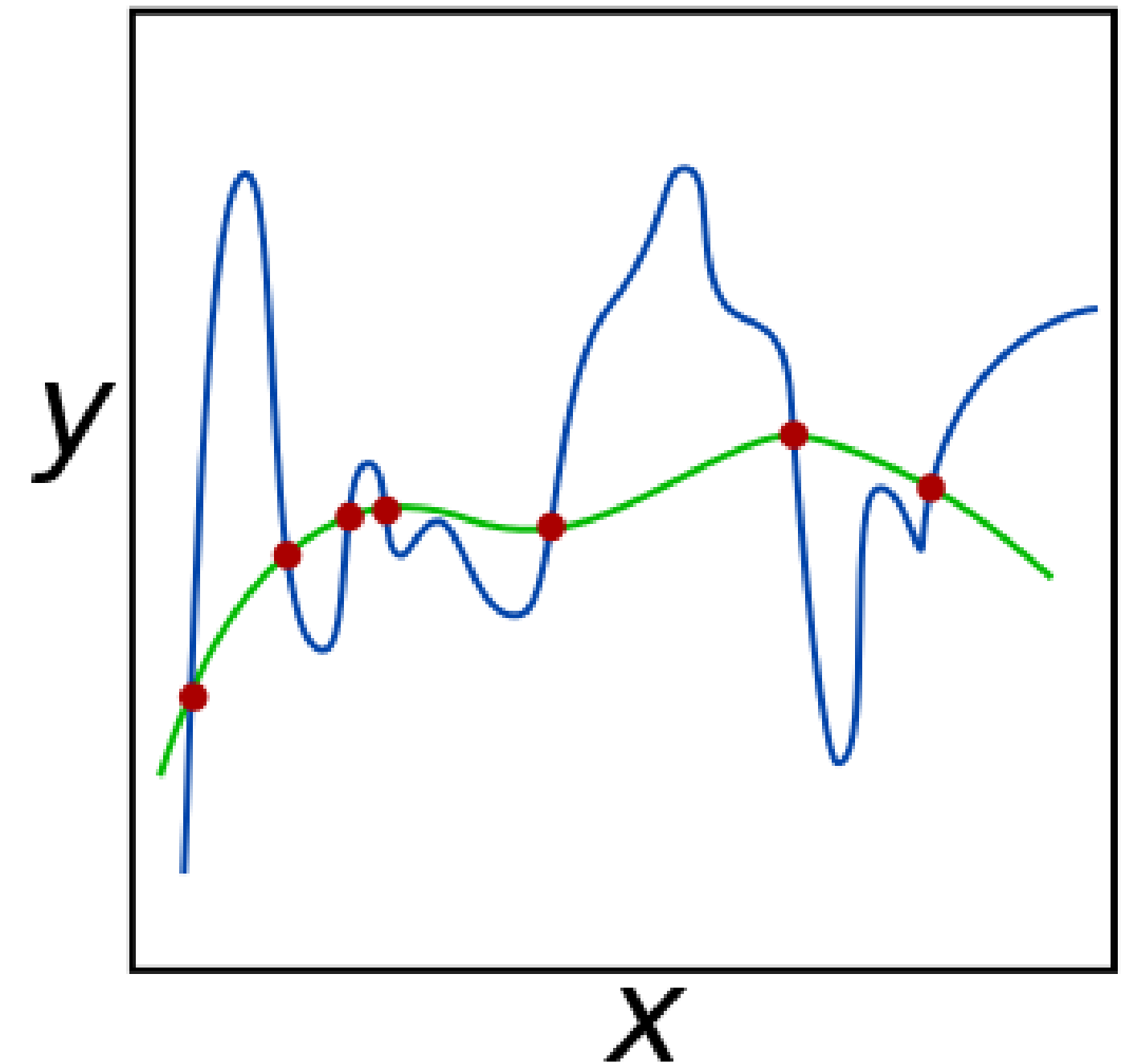


Squared Norm Regularization as Hard Constraint

- Reduce model complexity by limiting value range

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

- 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

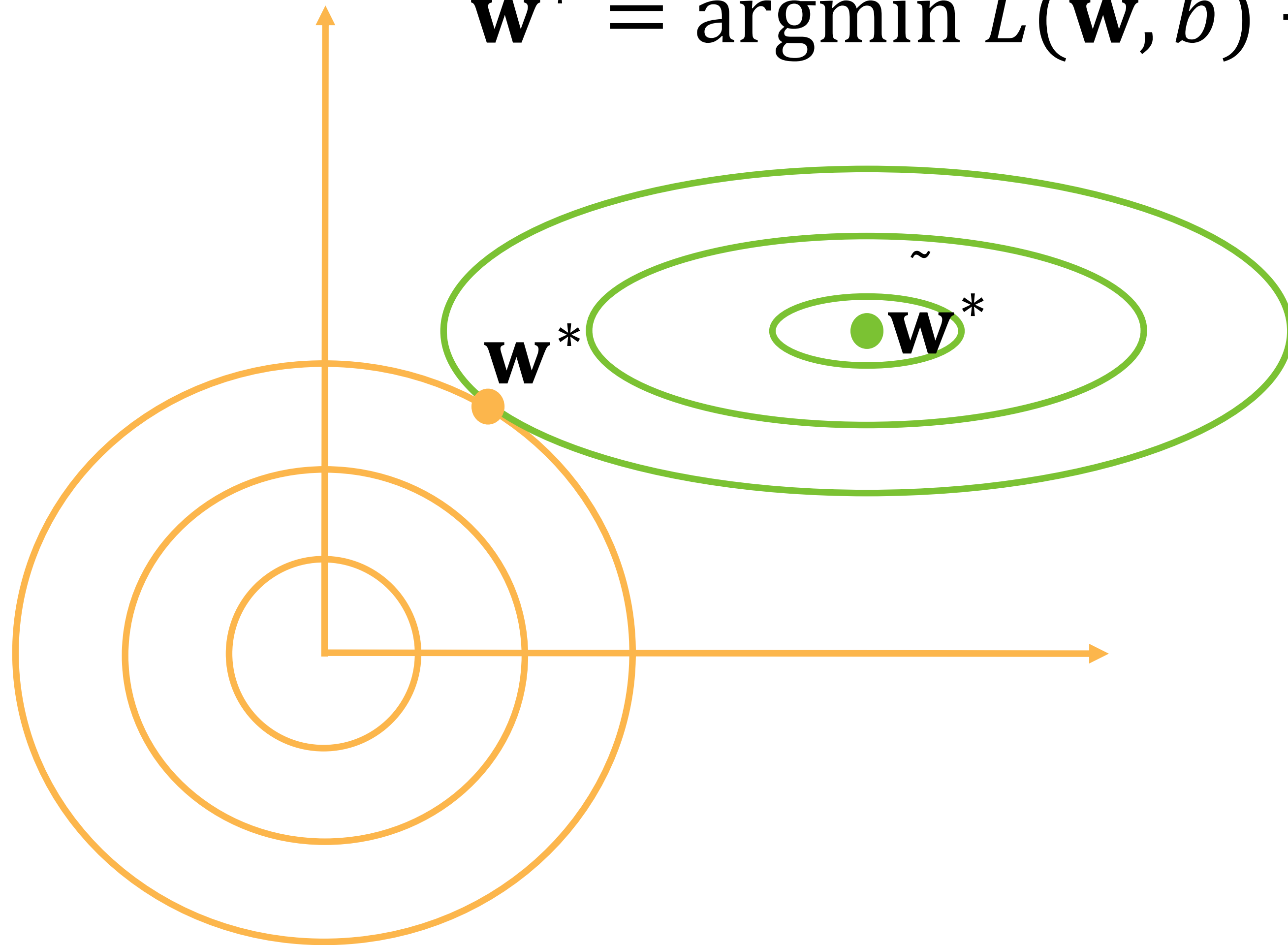
- We can rewrite the hard constraint version as

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

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

Illustrate the Effect on Optimal Solutions

$$\mathbf{w}^* = \operatorname{argmin} L(\mathbf{w}, b) + \frac{\lambda}{2} \|\mathbf{w}\|^2$$



$$\tilde{\mathbf{w}}^* = \operatorname{argmin} L(\mathbf{w}, b)$$

Dropout

Hinton et al.



Apply Dropout

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

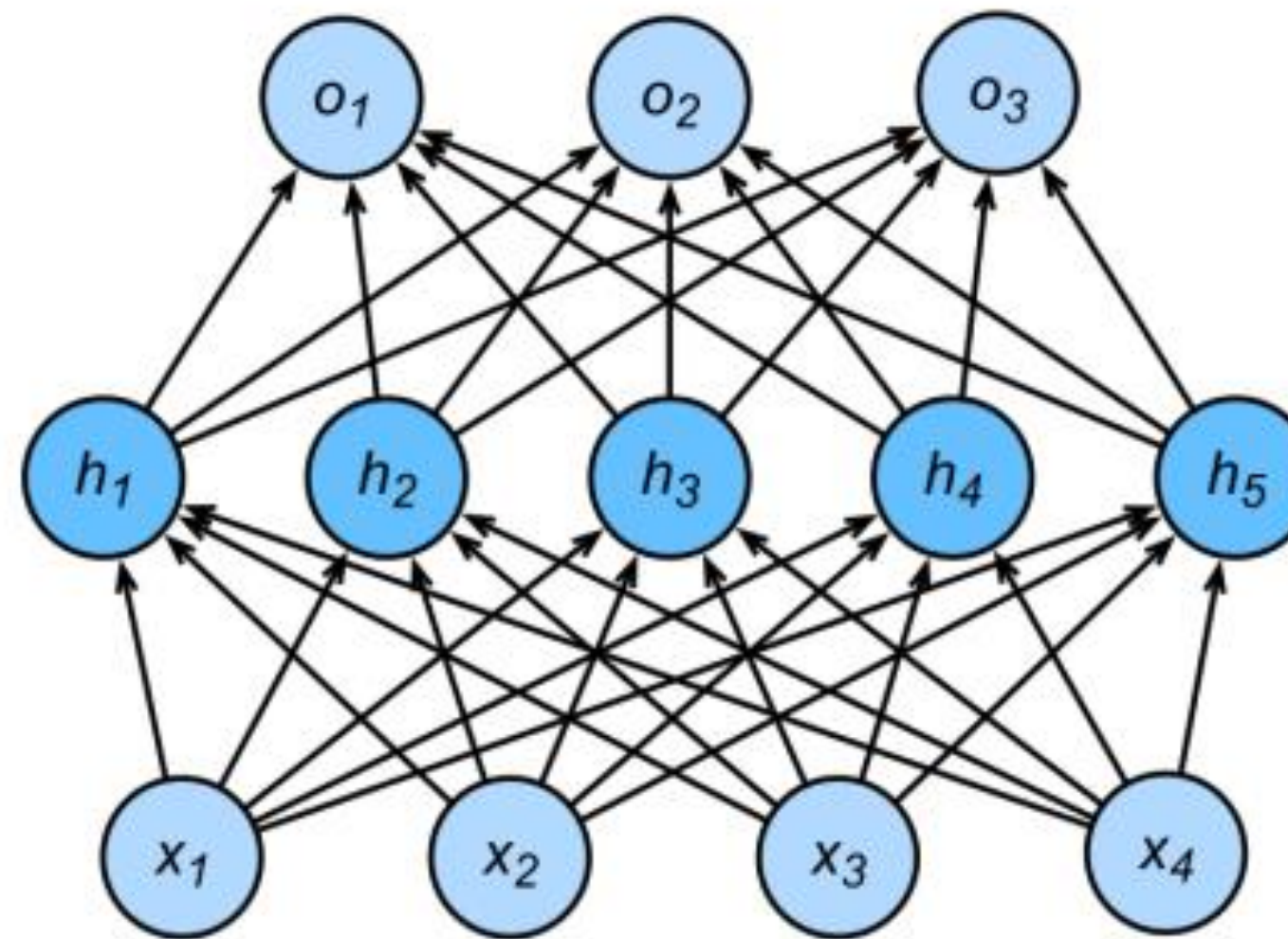
$$\mathbf{h} = \sigma(\mathbf{W}^{(1)}\mathbf{x} + \mathbf{b}^{(1)})$$

$$\mathbf{h}' = \text{dropout}(\mathbf{h})$$

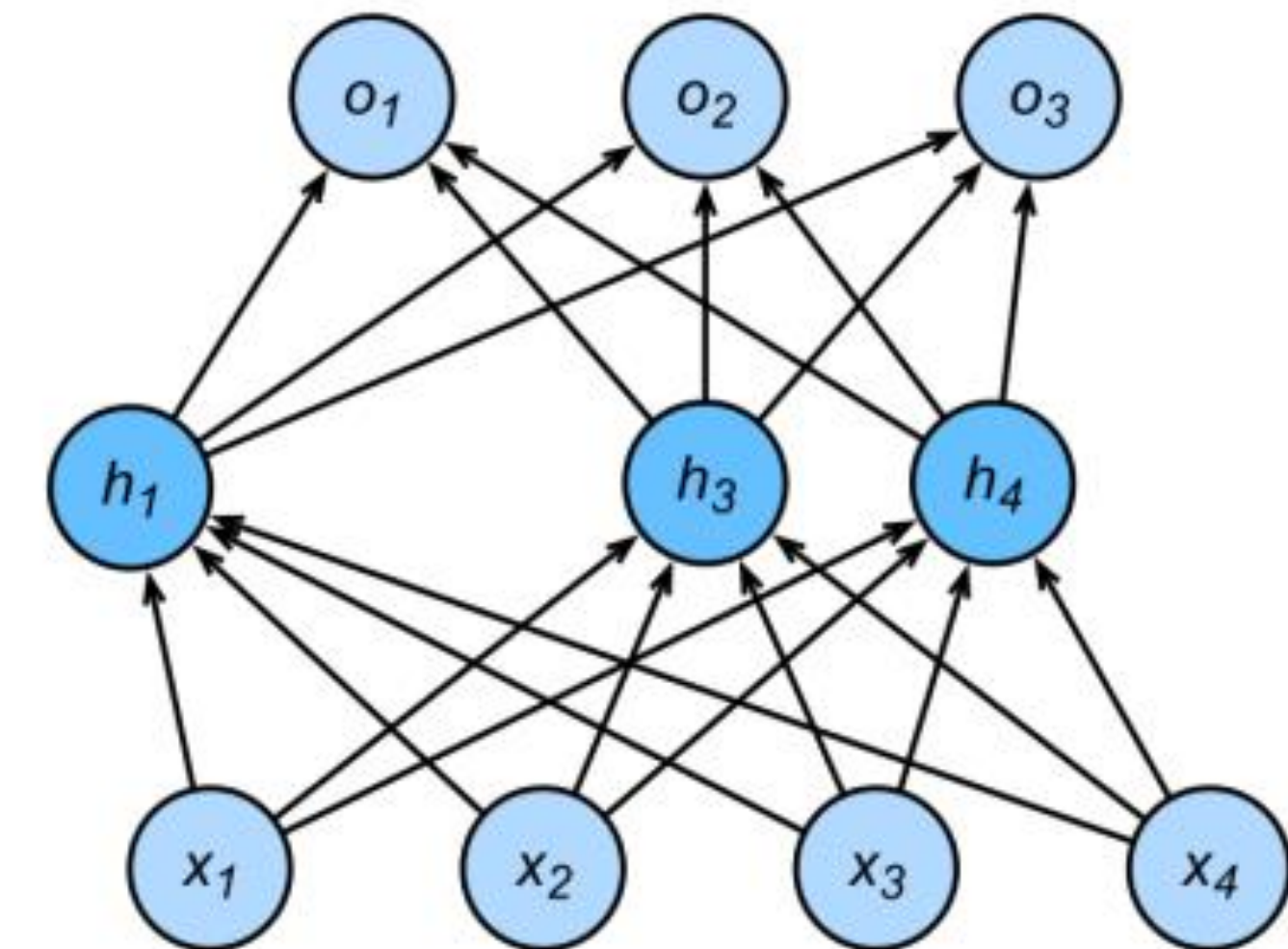
$$\mathbf{o} = \mathbf{W}^{(2)}\mathbf{h}' + \mathbf{b}^{(2)}$$

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

MLP with one hidden layer



Hidden layer after dropout



Dropout

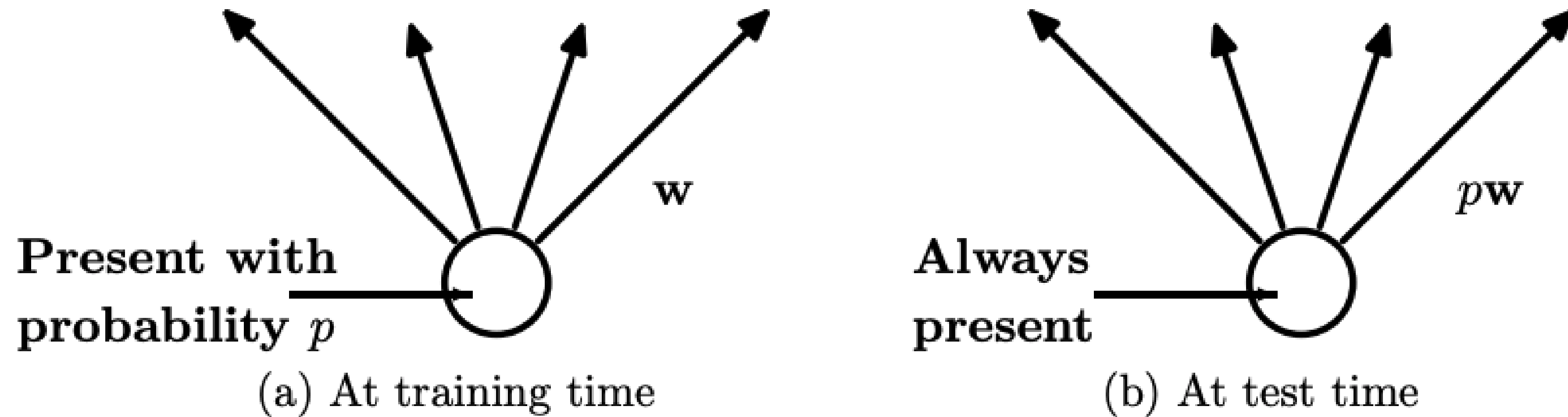


Figure 2: **Left:** A unit at training time that is present with probability p and is connected to units in the next layer with weights $\mathbf{w}$. **Right:** At test time, the unit is always present and the weights are multiplied by p . The output at test time is same as the expected output at training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

Dropout

Hinton et al.

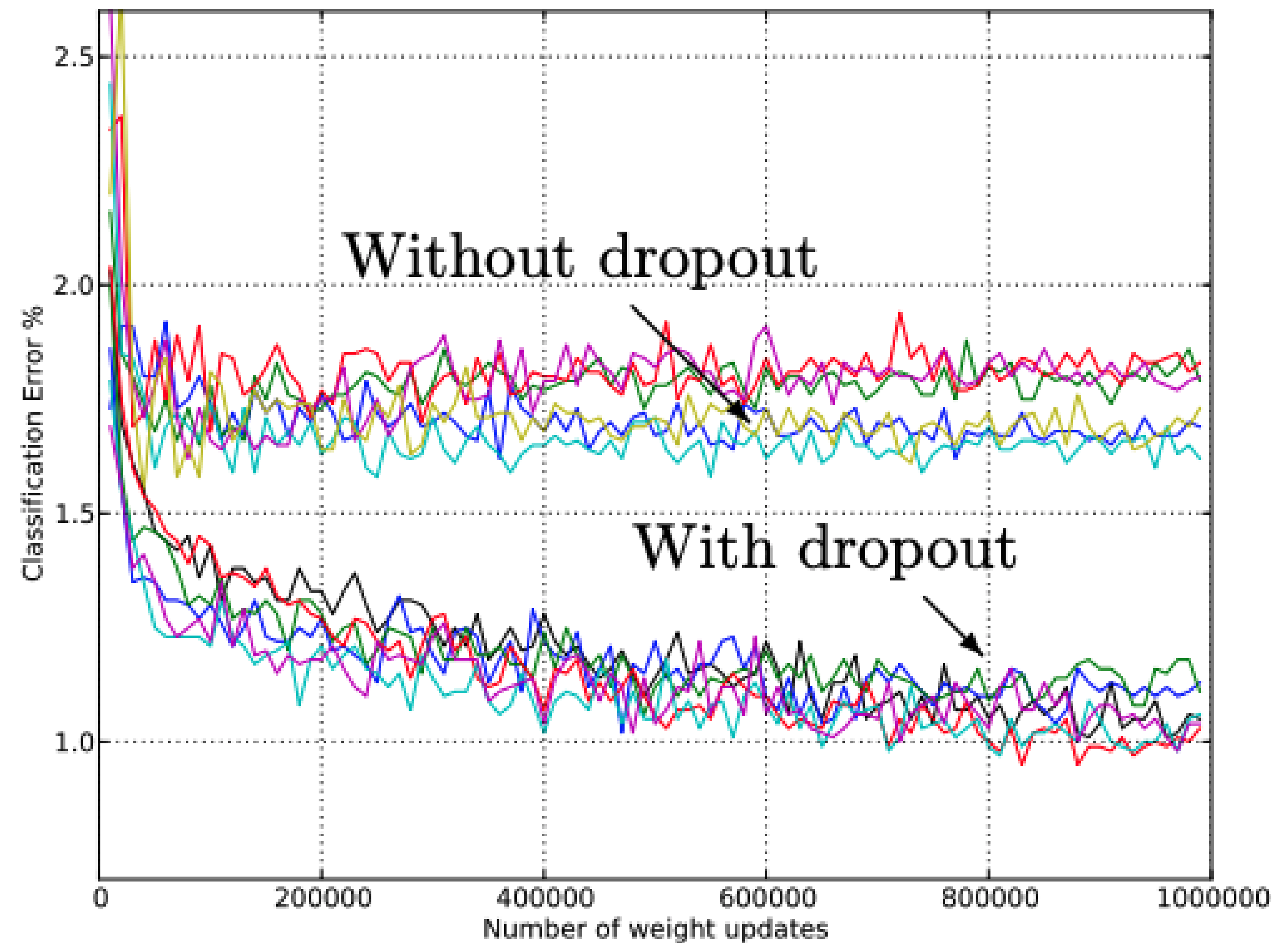


Figure 4: Test error for different architectures with and without dropout. The networks have 2 to 4 hidden layers each with 1024 to 2048 units.

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

Suggested reading

- Textbook: Artificial Intelligence: A Modern Approach (4th edition). Stuart Russell and Peter Norvig. Pearson, 2020.
 - Sections 21 Introduction, 21.1, 21.2, 21.4 ,21.5