

INF721

2024/2



Deep Learning

L6: Backpropagation

Logistics

Announcements

- ▶ PA1: Logistic Regression is due this monday (30/09)!
- ▶ We don't have class this monday. It's a holiday!

Last Lecture

- ▶ Non-linearly Separable Problems
- ▶ Multilayer Perceptron
 - ▶ Forward Pass
 - ▶ Hypothesis Space (Composite Functions)
- ▶ Categorical Cross-Entropy Loss

Lecture Outline

- ▶ Gradient Descent for Neural Networks
- ▶ Computational Graph
- ▶ Backpropagation
- ▶ Examples:
 - ▶ Logistic Regression
 - ▶ Multilayer Perceptron

Gradient Descent for Neural Networks

```
def optimize(x, y, lr, n_iter):  
    # Init weights with rand. vals. close to 0  
    W_1, b_1, W_2, b_2 = init_weights_rand()  
  
    for t in range(n_iter):  
        # Predict x labels  
        y_hat = forward(W_1, b_1, W_2, b_2)  
  
        # Compute gradients  
        dw_1 = ?, dw_2 = ?  
        db_1 = ?, db_2 = ?  
  
        # Update weights  
        W_1 = W_1 - lr * dw_1  
        b_1 = b_1 - lr * db_1  
        ...  
  
    return W_1, b_1, W_2, b_2
```

MLP (2 Layers)

$$\mathbf{z}^{[1]} = \mathbf{W}^{[1]}\mathbf{x} + \mathbf{b}^{[1]}$$

$$\mathbf{a}^{[1]} = g^{[1]}(\mathbf{z}^{[1]})$$

$$\mathbf{z}^{[2]} = \mathbf{W}^{[2]}\mathbf{a}^{[1]} + \mathbf{b}^{[2]}$$

$$\hat{y} = \sigma(\mathbf{z}^{[2]})$$

BCE Loss Function (Binary Classification)

$$L(h) = -\frac{1}{n} \sum_{i=1}^n (y_i \log \hat{y}_i + (1 - y_i) \log (1 - \hat{y}_i))$$

Gradients

$$\frac{\partial L}{\partial W_1} = ?$$

$$\frac{\partial L}{\partial W_2} = ?$$

$$\frac{\partial L}{\partial b_1} = ?$$

$$\frac{\partial L}{\partial b_2} = ?$$

Computing the gradients of a Neural Network

Linear models are simple enough so we can compute gradients by hand:

- ▶ Linear Regression: $\frac{\partial L}{\partial w} = (\hat{y} - y)x$, $\frac{\partial L}{\partial b} = (\hat{y} - y)$
- ▶ Logistic Regression: $\frac{\partial L}{\partial w} = (\hat{y} - y)x$, $\frac{\partial L}{\partial b} = (\hat{y} - y)$

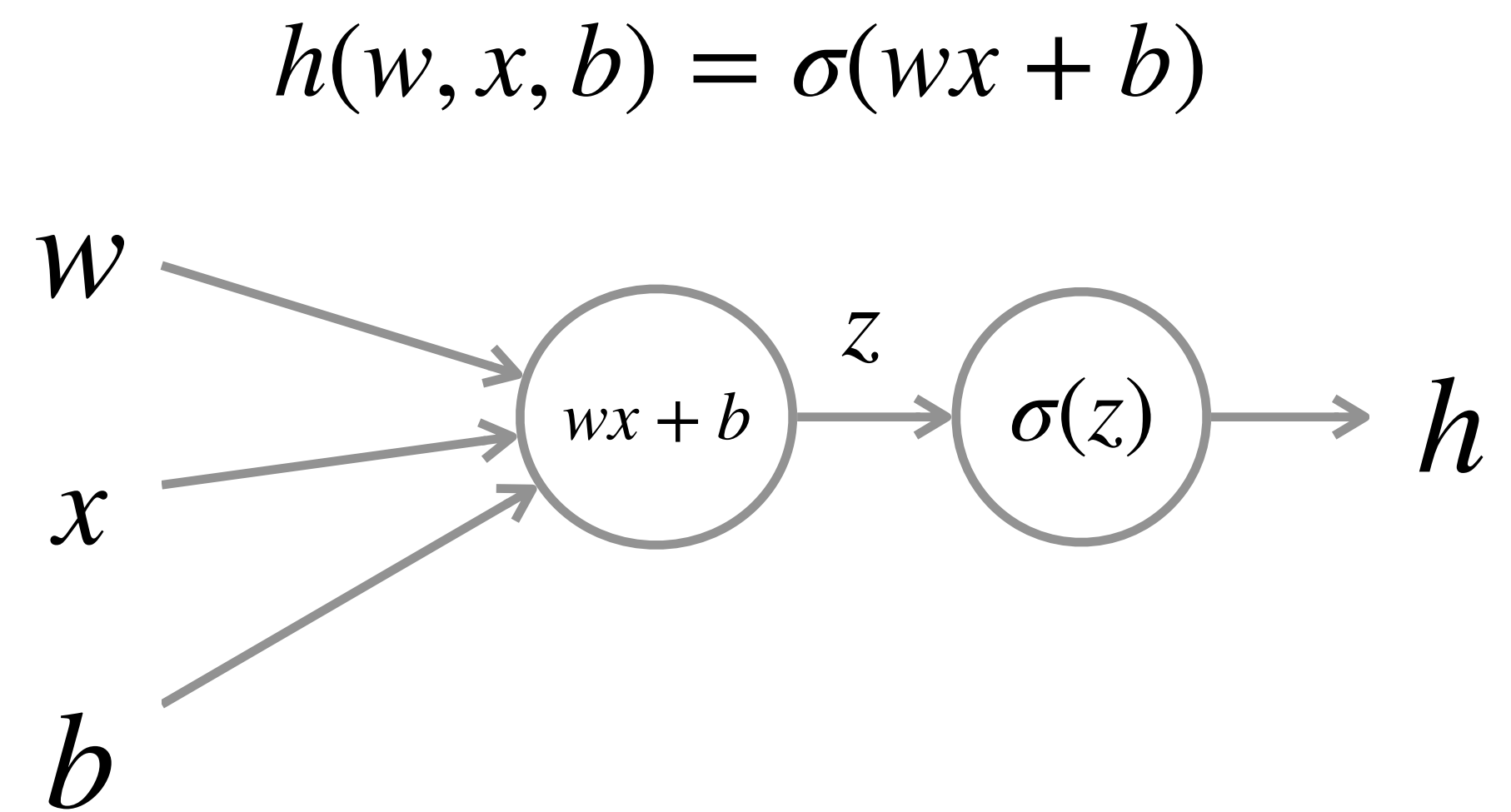
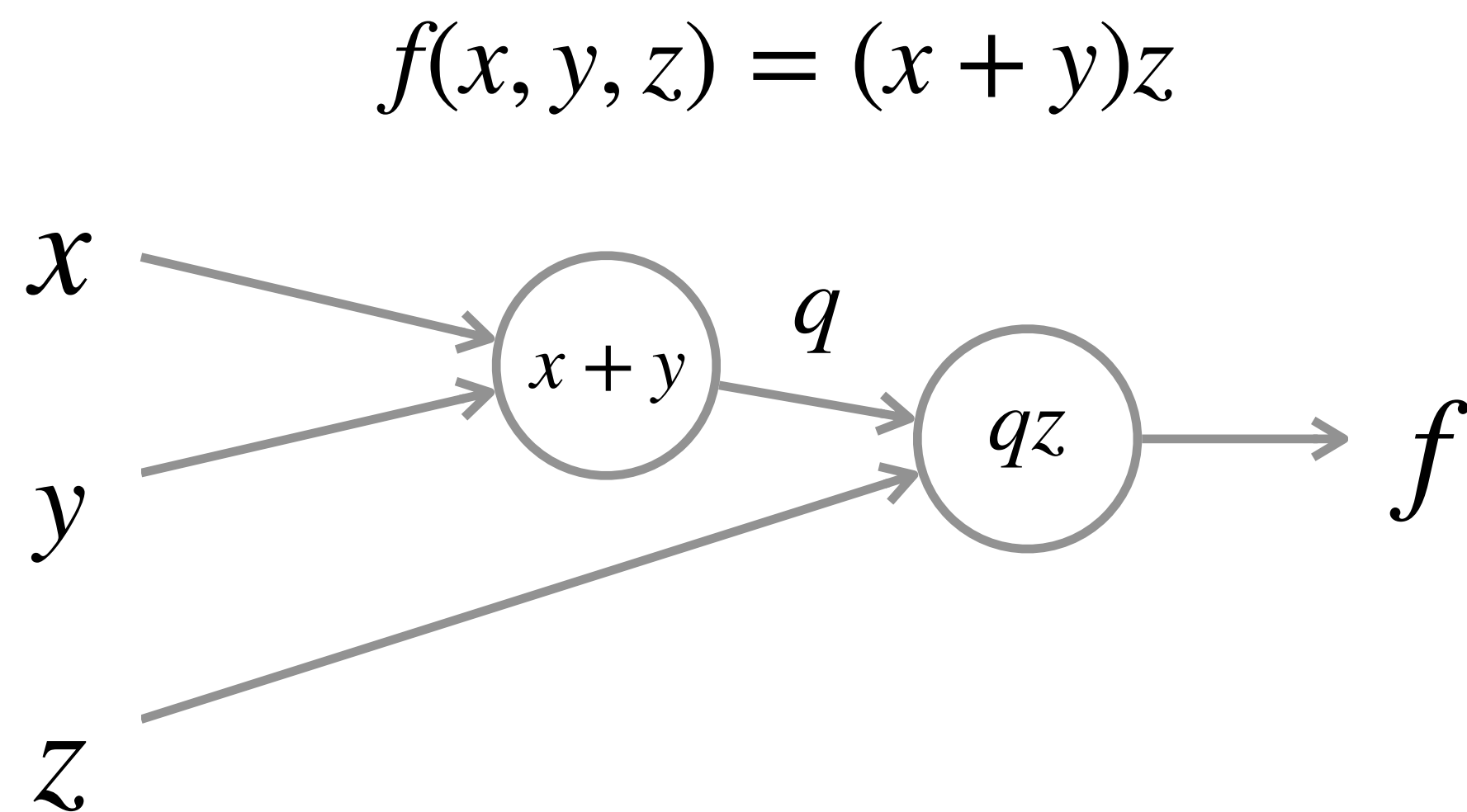
However, as the size of our models grows, this becomes impractical:

- ▶ It's easy to make mistakes
- ▶ It's not flexible — if we change the model or loss function, we have to recompute the gradients!
- ▶ **Solution: backpropagation!**

Computational Graph

A **computational graph** is a directed graph that represents mathematical operations:

- ▶ A node is a function of its inputs
- ▶ An edge represents a function argument



Backpropagation

Chain rule: $\frac{d}{dx}f(g(x)) = \frac{df}{dg} \cdot \frac{dg}{dx}$

Backpropagation is an algorithm that uses a computational graph and the chain rule to compute the gradient of a given function $f(x_1, x_2, \dots, x_n)$ with respect to its inputs x_1, x_2, \dots, x_n .

1. **Forward pass**

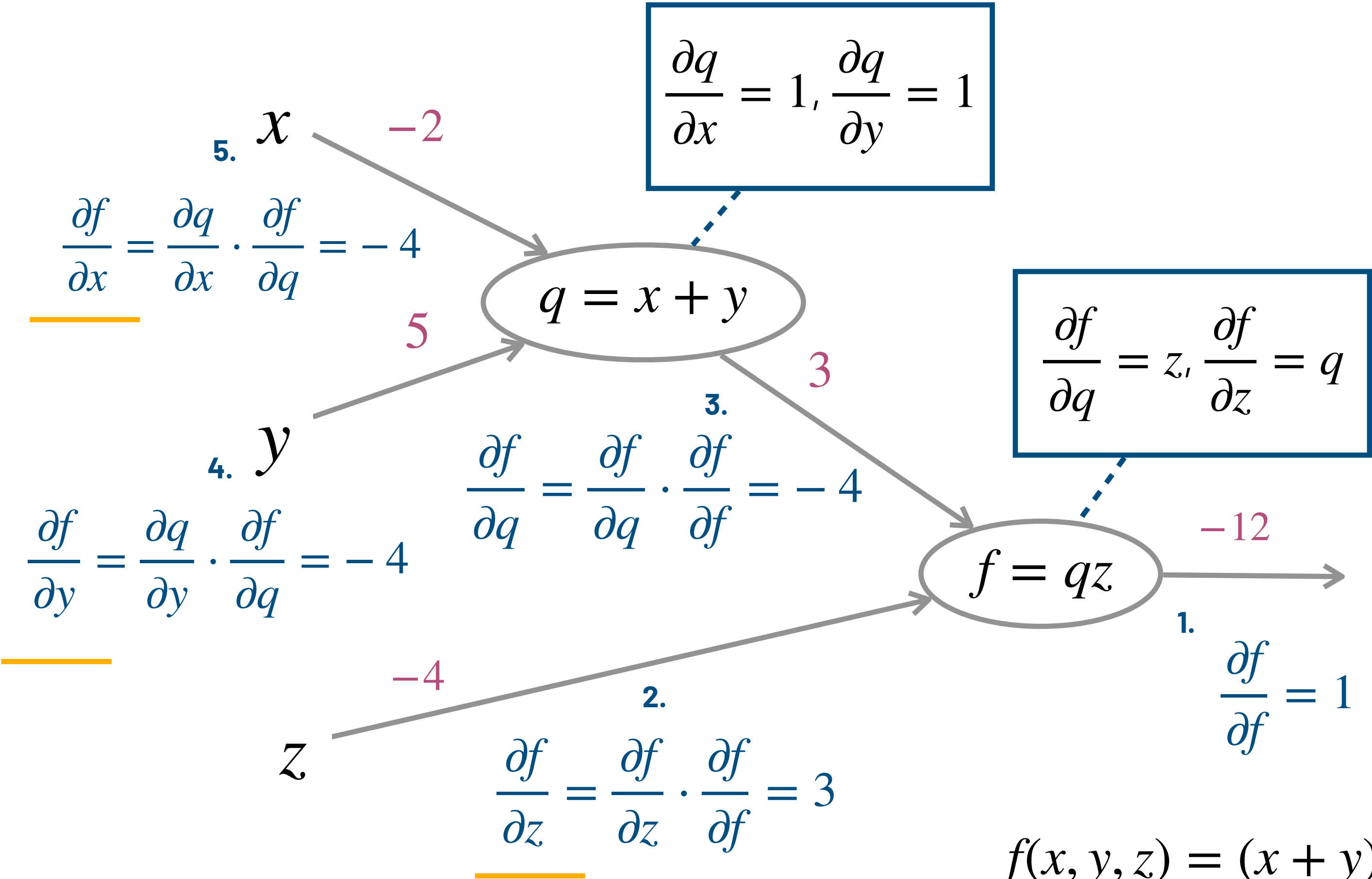
- Compute the output of f
- ▶ Nodes store partial results

2. **Backward pass**

Compute the gradient of the output with respect to each input:

$$\frac{\partial f}{\partial x}, \frac{\partial f}{\partial y}, \frac{\partial f}{\partial z}$$

- ▶ Nodes know how to compute **local gradients**
- ▶ Apply the chain rule backwards (depth-first traversal: **1. 2. 3. 4. 5.**)

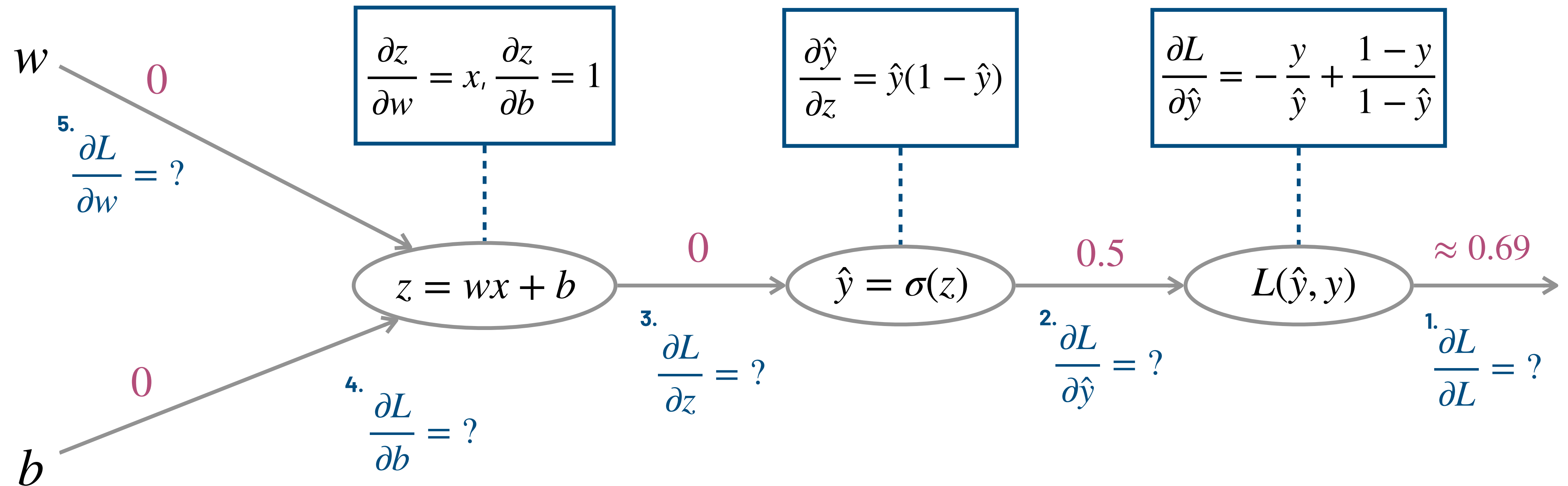


Backpropagation for Logistic Regression

We typically use backpropagation to compute the gradients of a loss function with respect to weights of a neural network

Logistic Regression: $\hat{y} = h(x) = \frac{1}{1 + e^{-(wx+b)}}$

BCE Loss: $L(\hat{y}, y) = -y \log \hat{y} + (1 - y) \log (1 - \hat{y})$

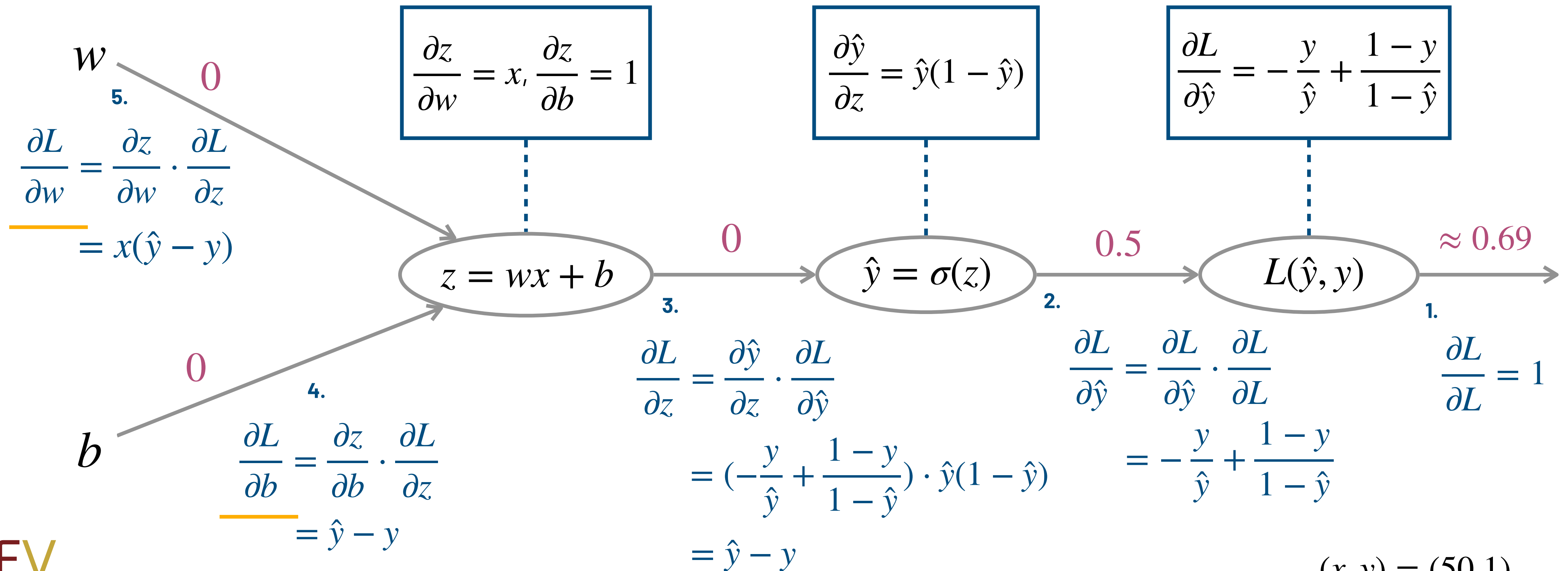


Backpropagation for Logistic Regression

We typically use backpropagation to compute the gradients of a loss function with respect to weights of a neural network

Logistic Regression: $\hat{y} = h(x) = \frac{1}{1 + e^{-(wx+b)}}$

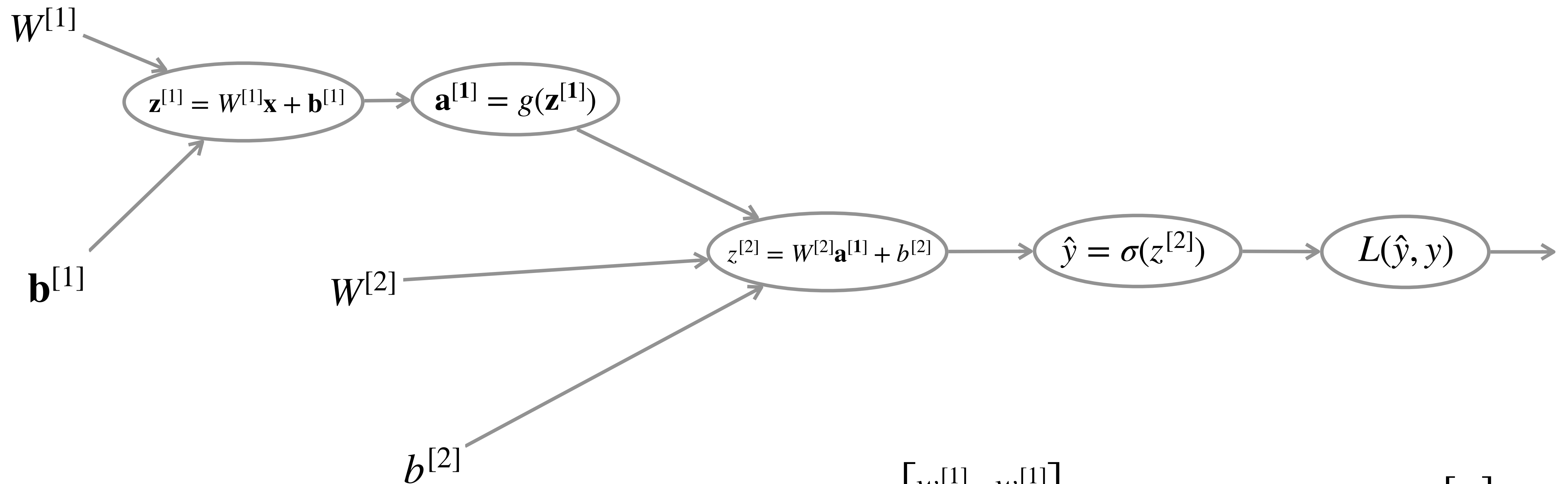
BCE Loss: $L(\hat{y}, y) = -y \log \hat{y} + (1 - y) \log (1 - \hat{y})$



Backpropagation for 2-Layer MLP

MLP: $\mathbf{z}^{[1]} = W^{[1]}\mathbf{x} + \mathbf{b}^{[1]}$ $z^{[2]} = W^{[2]}\mathbf{a}^{[1]} + b^{[2]}$
 $\mathbf{a}^{[1]} = g^{[1]}(\mathbf{z}^{[1]})$ $\hat{y} = \sigma(z^{[2]})$

BCE Loss: $L(\hat{y}, y) = -y \log \hat{y} + (1 - y) \log (1 - \hat{y})$

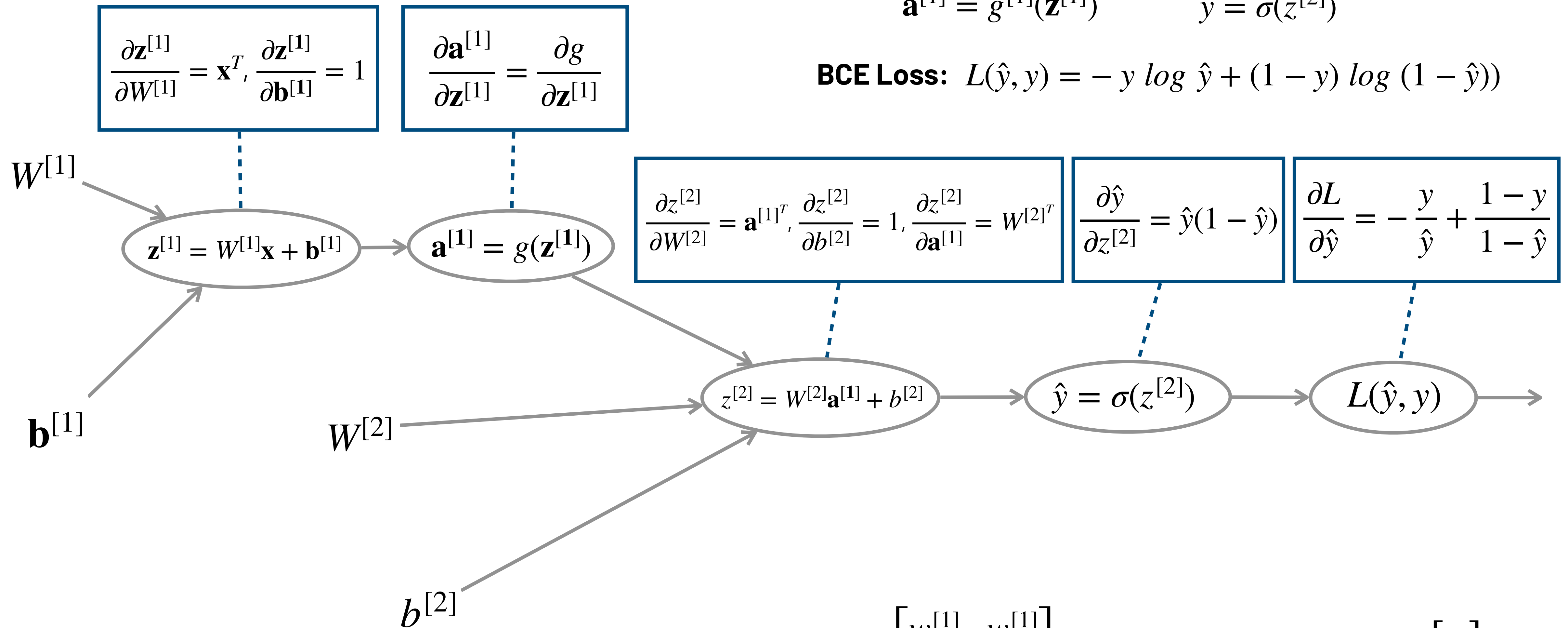


$$W^{[1]} = \begin{bmatrix} w_{11}^{[1]} & w_{21}^{[1]} \\ w_{12}^{[1]} & w_{22}^{[1]} \end{bmatrix} \quad W^{[2]} = \begin{bmatrix} w_{11}^{[2]} & w_{21}^{[2]} \end{bmatrix} \quad \mathbf{x} = \begin{bmatrix} x_1 \\ x_2 \end{bmatrix}$$

Backpropagation for 2-Layer MLP

MLP: $\mathbf{z}^{[1]} = W^{[1]}\mathbf{x} + \mathbf{b}^{[1]}$ $z^{[2]} = W^{[2]}\mathbf{a}^{[1]} + b^{[2]}$
 $\mathbf{a}^{[1]} = g^{[1]}(\mathbf{z}^{[1]})$ $\hat{y} = \sigma(z^{[2]})$

BCE Loss: $L(\hat{y}, y) = -y \log \hat{y} + (1 - y) \log (1 - \hat{y})$

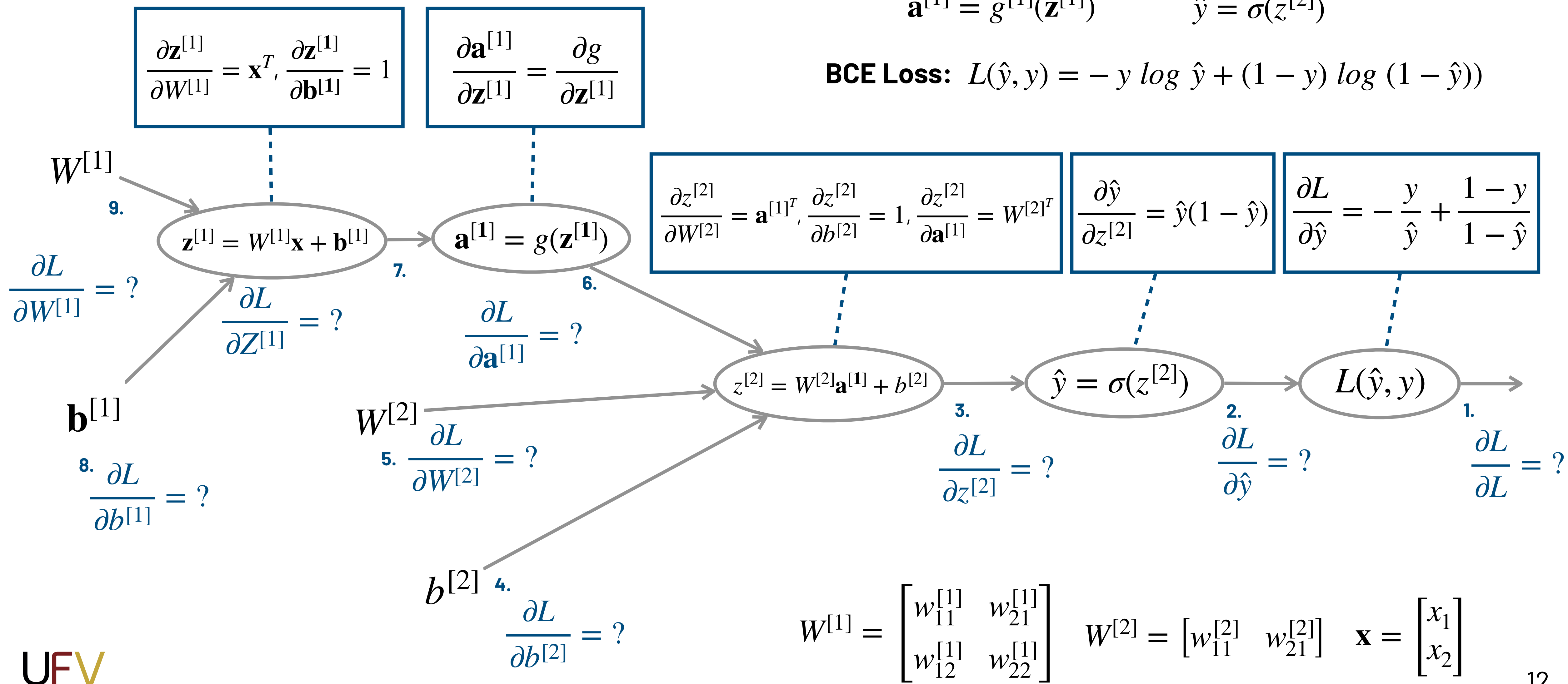


$$W^{[1]} = \begin{bmatrix} w_{11}^{[1]} & w_{21}^{[1]} \\ w_{12}^{[1]} & w_{22}^{[1]} \end{bmatrix} \quad W^{[2]} = \begin{bmatrix} w_{11}^{[2]} & w_{21}^{[2]} \end{bmatrix} \quad \mathbf{x} = \begin{bmatrix} x_1 \\ x_2 \end{bmatrix}$$

Backpropagation for 2-Layer MLP

MLP: $\mathbf{z}^{[1]} = W^{[1]}\mathbf{x} + \mathbf{b}^{[1]}$ $z^{[2]} = W^{[2]}\mathbf{a}^{[1]} + b^{[2]}$
 $\mathbf{a}^{[1]} = g^{[1]}(\mathbf{z}^{[1]})$ $\hat{y} = \sigma(z^{[2]})$

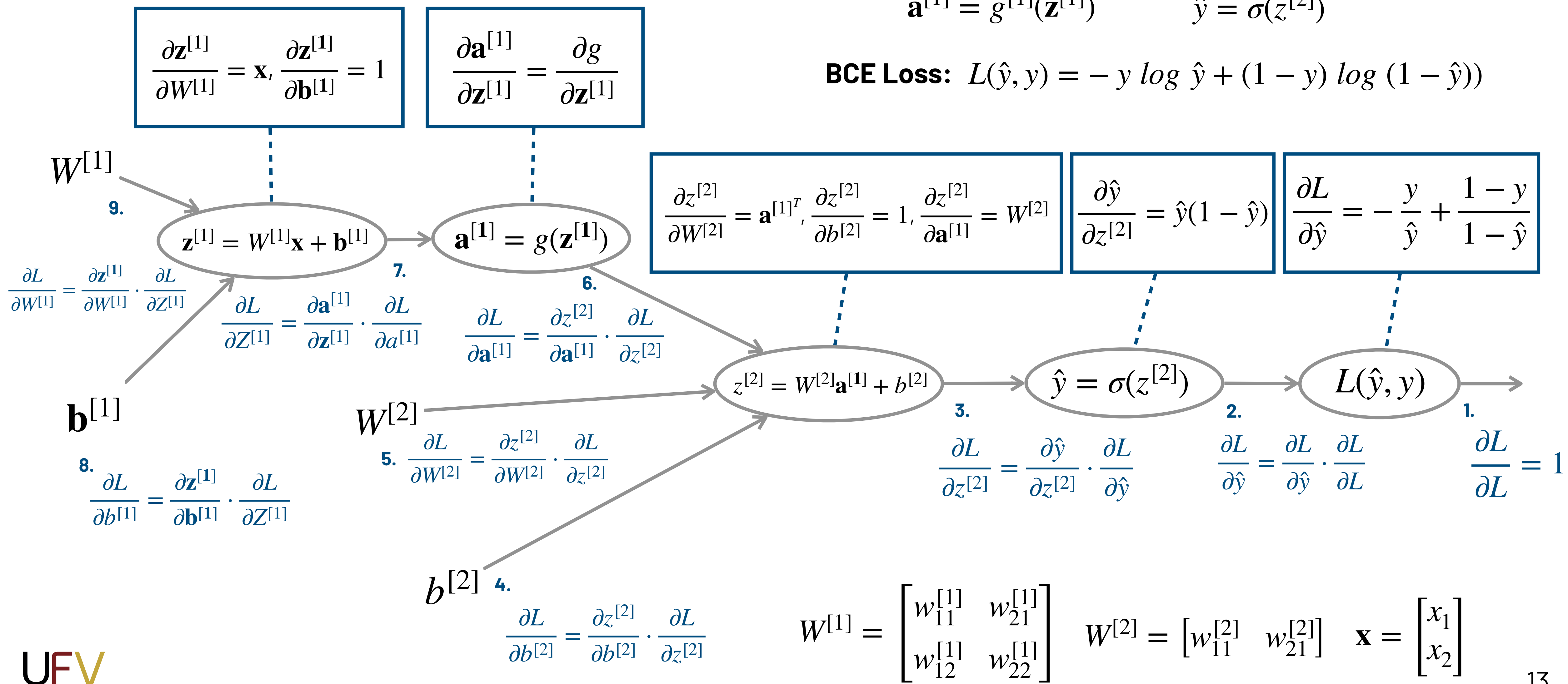
BCE Loss: $L(\hat{y}, y) = -y \log \hat{y} + (1 - y) \log (1 - \hat{y})$



Backpropagation for 2-Layer MLP

MLP: $\mathbf{z}^{[1]} = W^{[1]}\mathbf{x} + \mathbf{b}^{[1]}$ $z^{[2]} = W^{[2]}\mathbf{a}^{[1]} + b^{[2]}$
 $\mathbf{a}^{[1]} = g^{[1]}(\mathbf{z}^{[1]})$ $\hat{y} = \sigma(z^{[2]})$

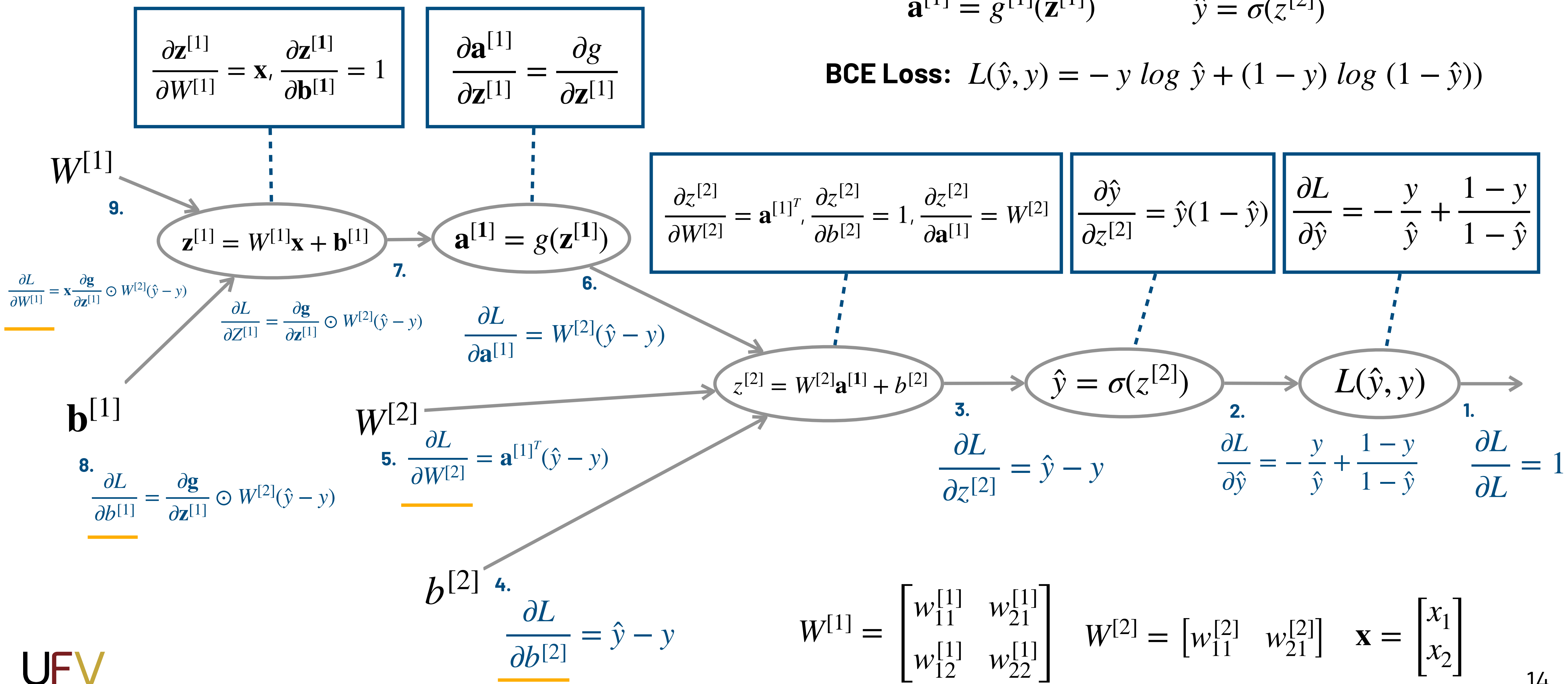
BCE Loss: $L(\hat{y}, y) = -y \log \hat{y} + (1 - y) \log (1 - \hat{y})$



Backpropagation for 2-Layer MLP

MLP: $\mathbf{z}^{[1]} = W^{[1]}\mathbf{x} + \mathbf{b}^{[1]}$ $z^{[2]} = W^{[2]}\mathbf{a}^{[1]} + b^{[2]}$
 $\mathbf{a}^{[1]} = g^{[1]}(\mathbf{z}^{[1]})$ $\hat{y} = \sigma(z^{[2]})$

BCE Loss: $L(\hat{y}, y) = -y \log \hat{y} + (1 - y) \log (1 - \hat{y})$



Gradient Descent for Neural Networks

```
def optimize(x, y, lr, n_iter):  
    # Init weights with rand. vals. close to 0  
    W_1, b_1, W_2, b_2 = init_weights_rand()  
  
    for t in range(n_iter):  
        # Predict x labels  
        y_hat = forward(W_1, b_1, W_2, b_2)  
  
        # Compute gradients  
        dw_1, db_1, dw_2, db_2 = backward()  
  
        # Update weights  
        W_1 = W_1 - lr * dw_1  
        b_1 = b_1 - lr * db_1  
        ...  
  
    return W_1, b_1, W_2, b_2
```

MLP (2 Layers)

$$\mathbf{z}^{[1]} = W^{[1]}\mathbf{x} + \mathbf{b}^{[1]}$$

$$\mathbf{a}^{[1]} = g^{[1]}(\mathbf{z}^{[1]})$$

$$\mathbf{z}^{[2]} = W^{[2]}\mathbf{a}^{[1]} + \mathbf{b}^{[2]}$$

$$\hat{y} = \sigma(\mathbf{z}^{[2]})$$

BCE Loss Function (Binary Classification)

$$L(h) = -\frac{1}{n} \sum_{i=1}^n (y_i \log \hat{y}_i + (1 - y_i) \log (1 - \hat{y}_i))$$

Gradients

$$\frac{\partial L}{\partial W_1} = \mathbf{x}^T \frac{\partial \mathbf{g}}{\partial \mathbf{z}^{[1]}} \odot W^{[2]T}(\hat{y} - y) \quad \frac{\partial L}{\partial W_2} = \mathbf{a}^{[1]}(\hat{y} - y)$$

$$\frac{\partial L}{\partial b_1} = \frac{\partial \mathbf{g}}{\partial \mathbf{z}^{[1]}} \odot W^{[2]T}(\hat{y} - y) \quad \frac{\partial L}{\partial b^{[2]}} = \hat{y} - y$$

Next Lecture

L7: Evaluating Deep Learning Models

Metrics for evaluating the generalization deep learning models

- ▶ Accuracy/Error
- ▶ Learning Curve
- ▶ Cross-validation
- ▶ Confusion Matrix