# CS 540 Introduction to Artificial Intelligence
# **Neural Networks: Backpropagation**

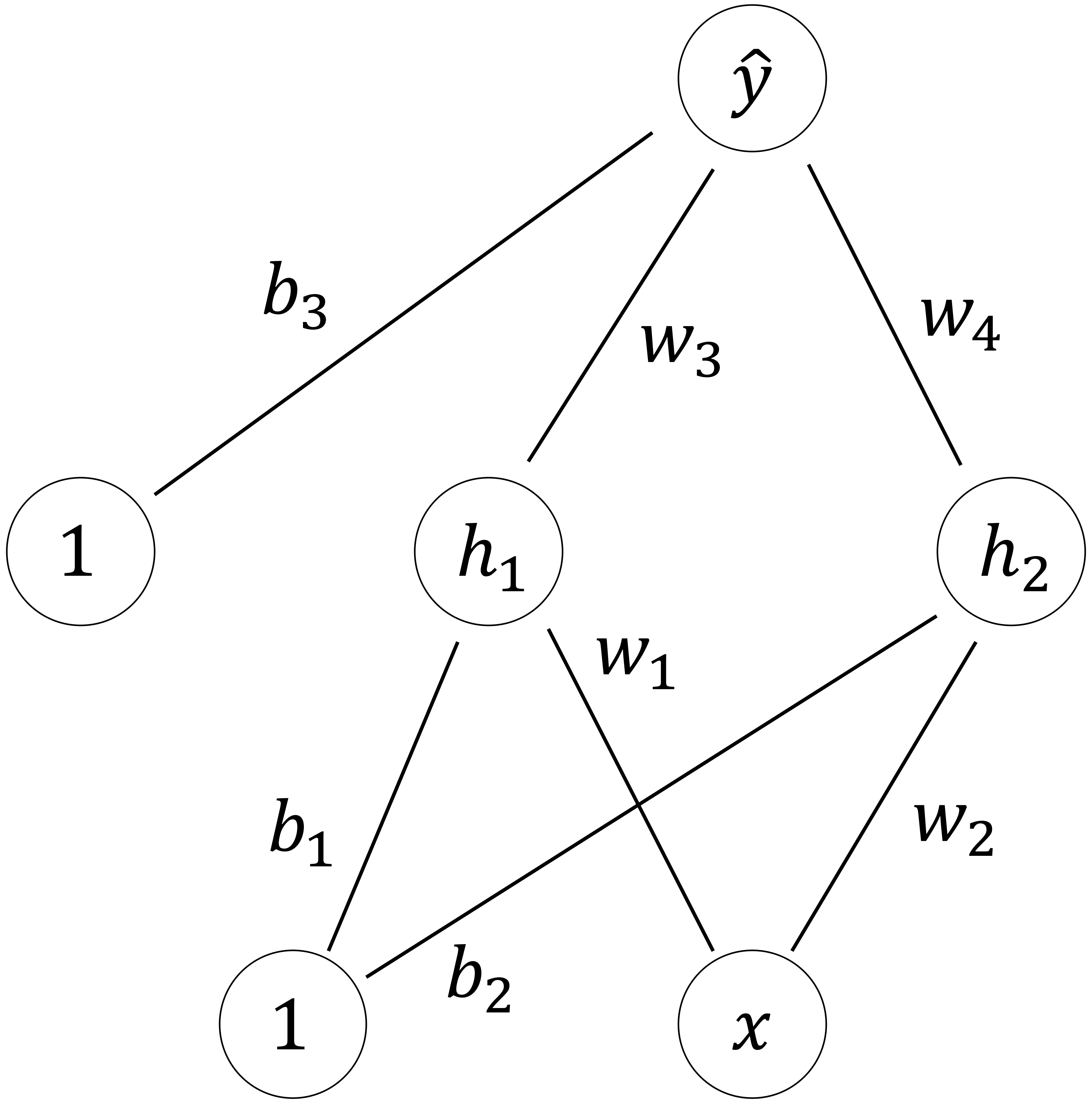
University of Wisconsin–Madison
Fall 2025, Section 3
October 13, 2025

# Announcements

- HW5 due Friday 10/6 at 11:59 pm
- Midterm exam:

Thursday 10/23
7:30 pm to 9:00 pm
Humanities Building, Room 3650

# A Simple Multilayer Perceptron



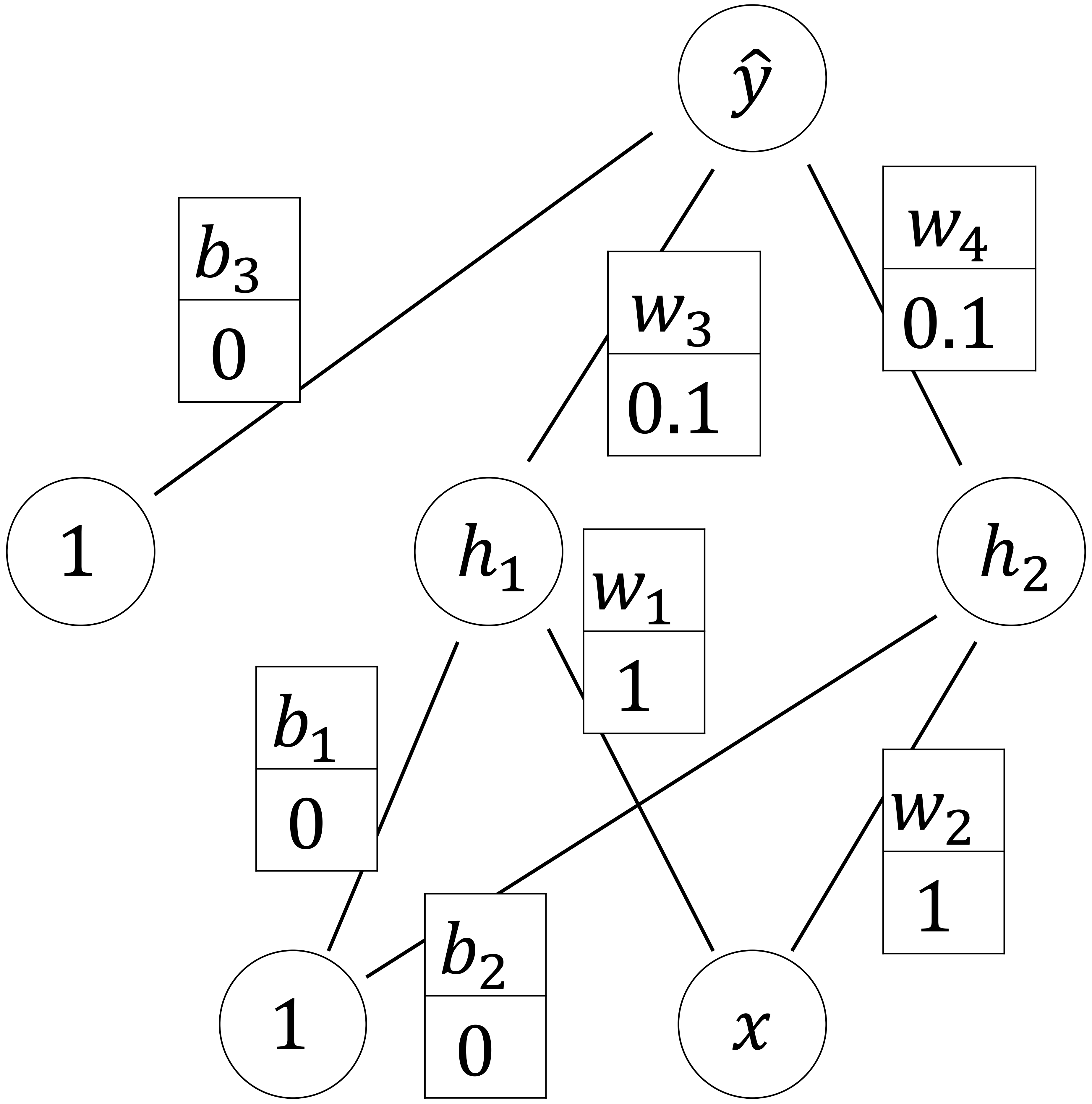

Hidden node: ReLU activations

$$h_1 = \max\{0, w_1 x + b_1\}$$

Output node: no activation

$$\hat{y} = w_3 h_1 + w_4 h_2 + b_3$$

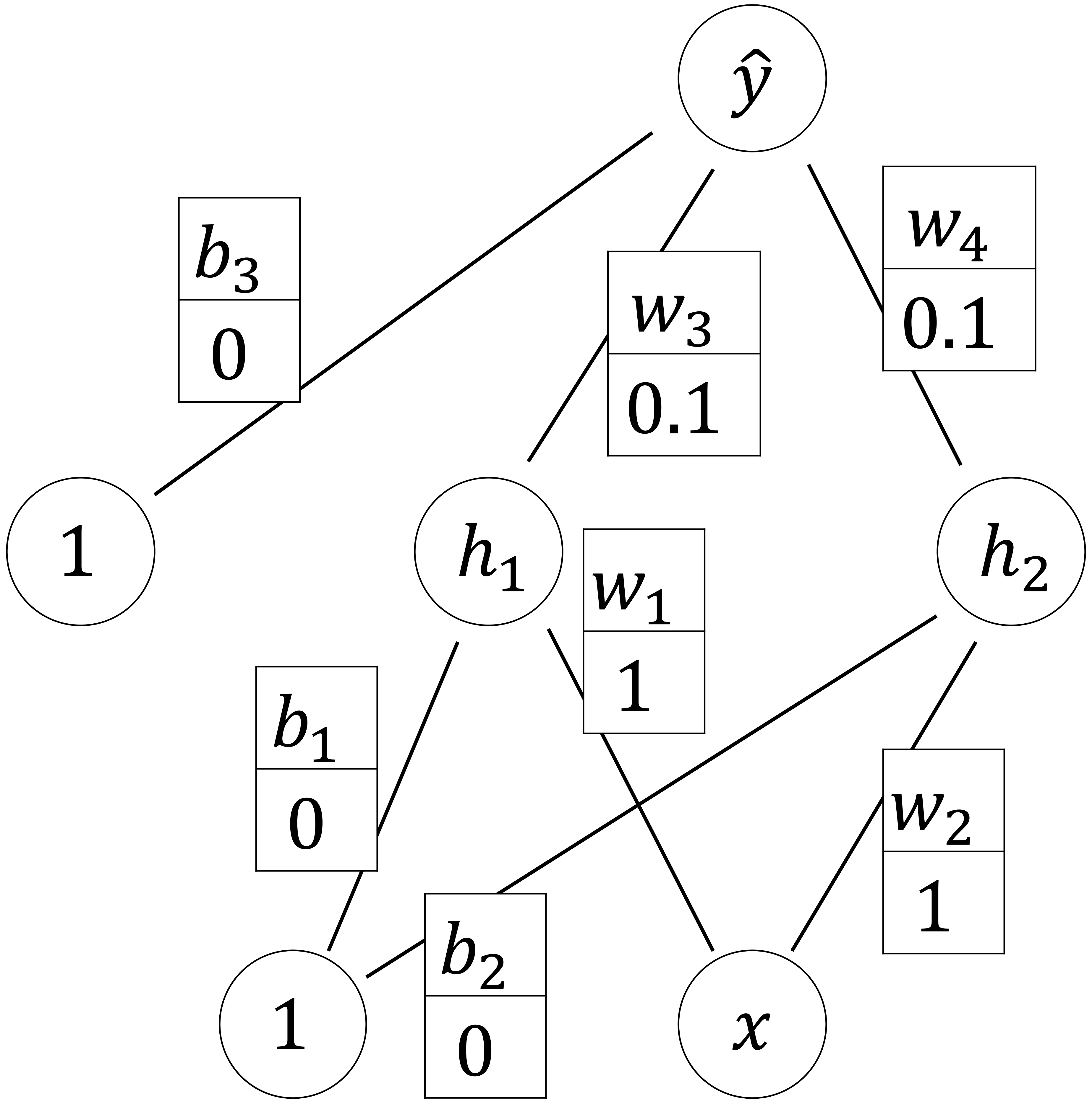
# The Same Network, With Numbers




Compute: $\hat{y}$ on input $x = 2$

This is a “forward pass” through the network

# Making Predictions Match the Labels

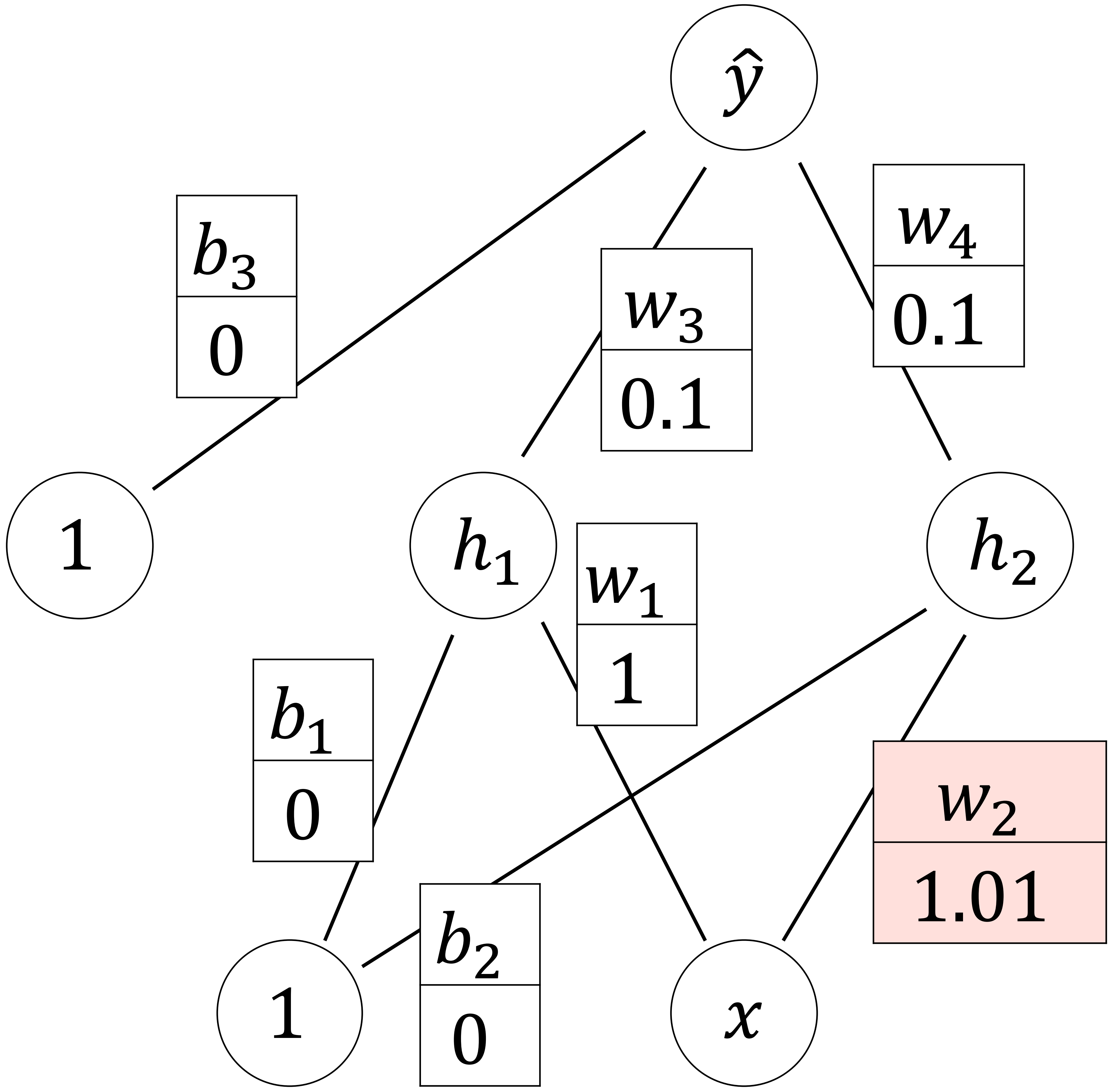



Suppose we have one labeled example

$$(x, y) = (2,1)$$

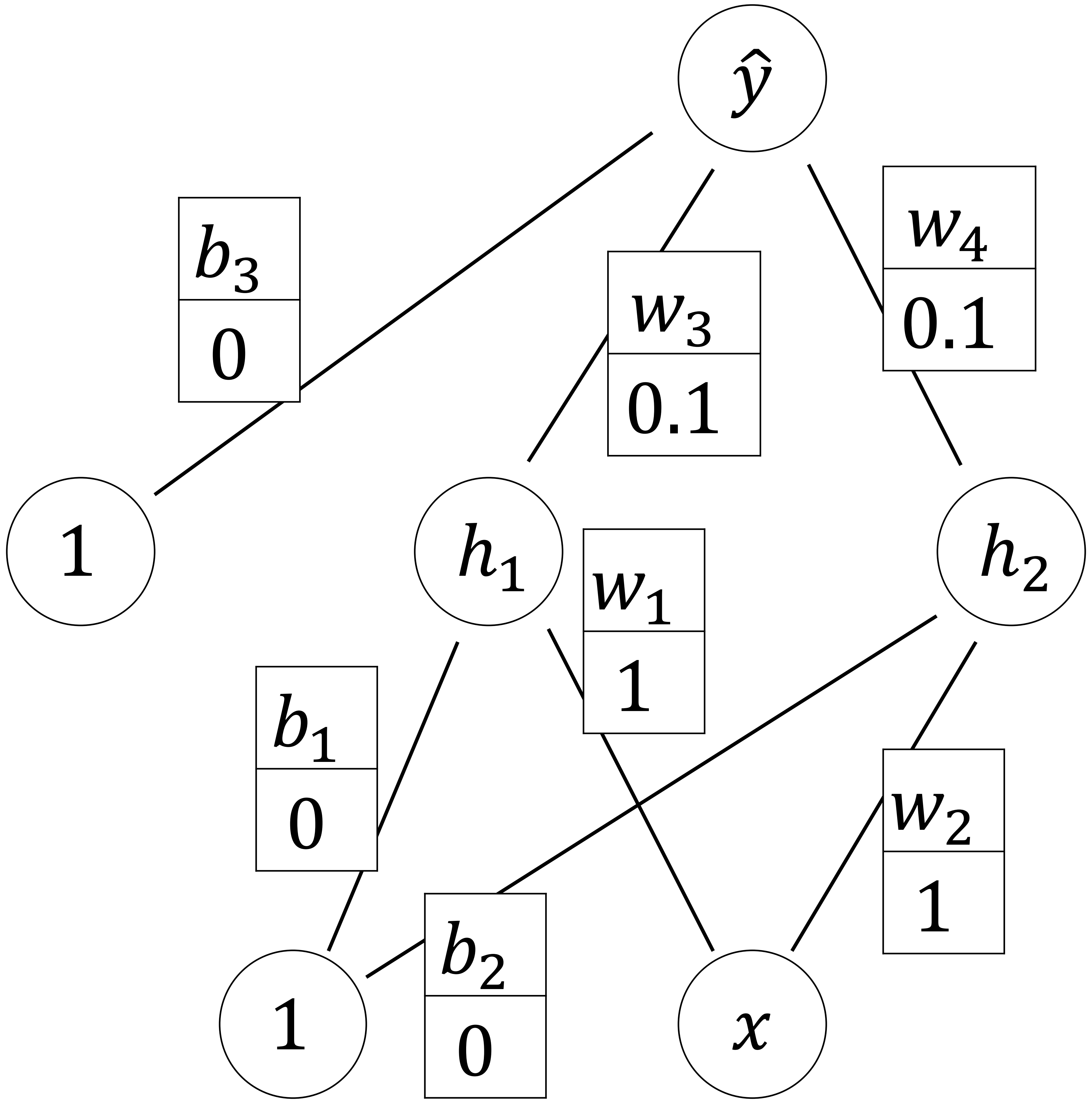
If we increase $w_2$ a little, will our prediction get better or worse?

# Change Weights, Change Prediction




Compute: $\hat{y}$ on input $x = 2$

# Formalize the Intuition with Calculus




Use the squared loss:

$$\ell(y, \hat{y}) = (y - \hat{y})^2$$

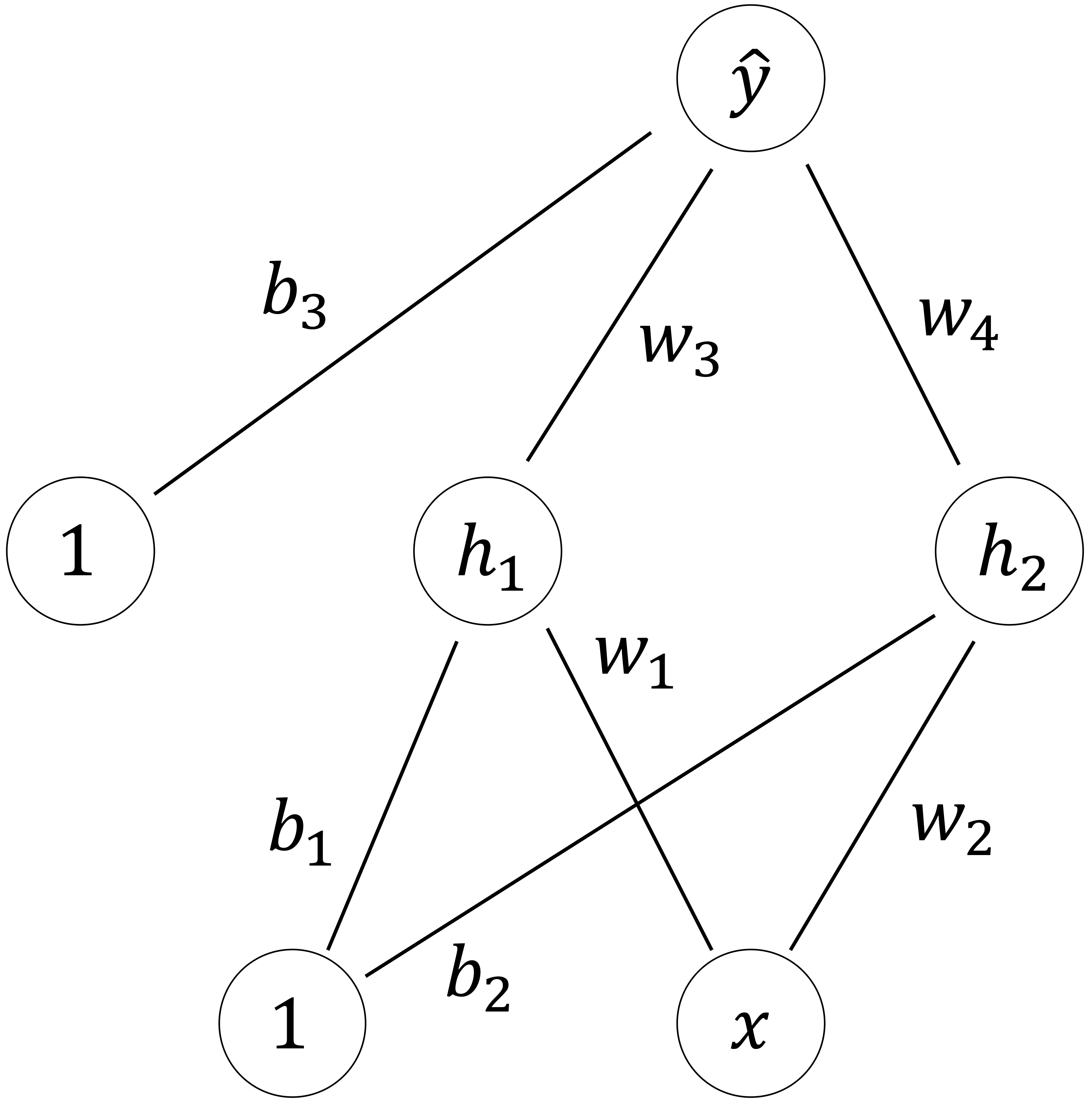
Partial derivative: $\frac{\partial \ell}{\partial w_2}$

How does loss change when $w_2$ changes?

For these weights, with $(x, y) = (2,1)$

$$\frac{\partial \ell}{\partial w_2} = -0.24$$

# Notation for all the parameters at once




Collect all parameters into a single vector $\theta$

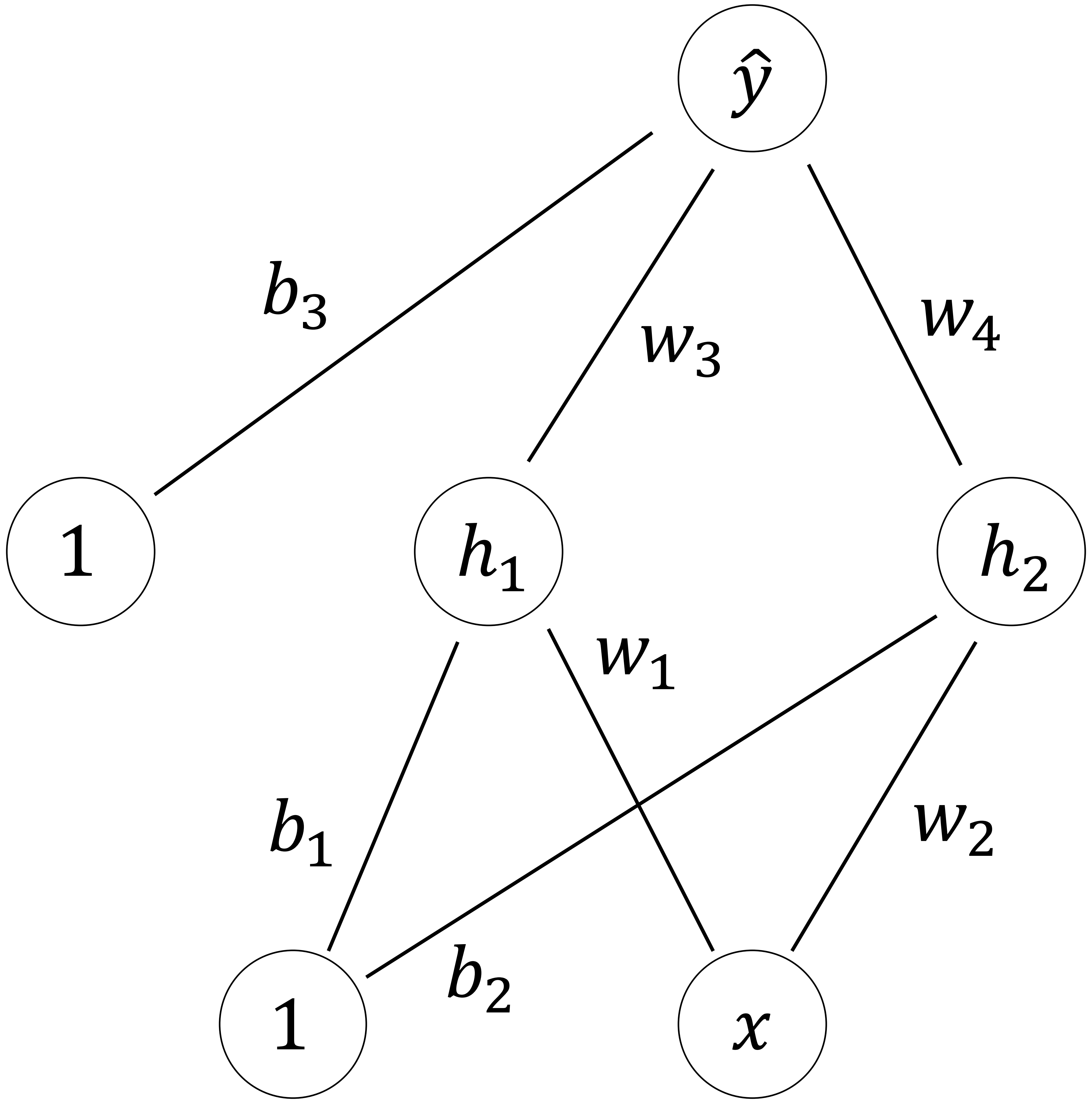
$$\theta = \begin{bmatrix} w_1 \\ w_2 \\ w_3 \\ w_4 \\ b_1 \\ b_2 \\ b_3 \end{bmatrix}$$

Write prediction function: $\hat{y} = f(x;\theta)$

(Squared) loss on a dataset:

$$L(\theta; X, y) = \frac{1}{n}\sum_{i=1}^{n} \ell(\theta; x_i, y_i) = \frac{1}{n}\sum_{i=1}^{n} (f(x;\theta) - y)^2$$

# Notation for all the parameters at once




Collect all parameters into a single vector $\theta$

$$\theta = \begin{bmatrix} w_1 \\ w_2 \\ w_3 \\ w_4 \\ b_1 \\ b_2 \\ b_3 \end{bmatrix}$$

Write all partial derivatives as one vector: the **gradient**

$$\nabla_\theta L(\theta; X, y) = \begin{bmatrix} \frac{\partial L}{\partial w_1} \\ \frac{\partial L}{\partial w_2} \\ \vdots \\ \frac{\partial L}{\partial b_3} \end{bmatrix}$$

# Dissecting our Gradient Notation

- When $f$ maps vector $x \in \mathbb{R}^d$ to scalar $f(x) \in \mathbb{R}$

$$\nabla f(x) \in \mathbb{R}^d$$

- We will work with functions of many vectors (and matrices!)

$$L(\theta; X, y)$$

- Write gradient with respect to $\theta$

$$\nabla_\theta L(\theta_{t-1}; X, y)$$

| **Gradient Descent** |
| --- |
| Input: dataset $(\mathrm{X}, \mathrm{y})$, loss function $L$, number of steps $T$, step size $\eta$ |
| 1. Initialize $\theta_0$<br>2. For $t = 1,2, \dots, T$<br>3. Calculate $g_t = \nabla_\theta L(\theta_{t-1}; X, y)$<br>4. Update $\theta_t \leftarrow \theta_{t-1} - \eta g_t$<br>5. Return $\theta_T$ |

# (Minibatch) Stochastic Gradient Descent

- Gradient descent uses loss on entire dataset every step:

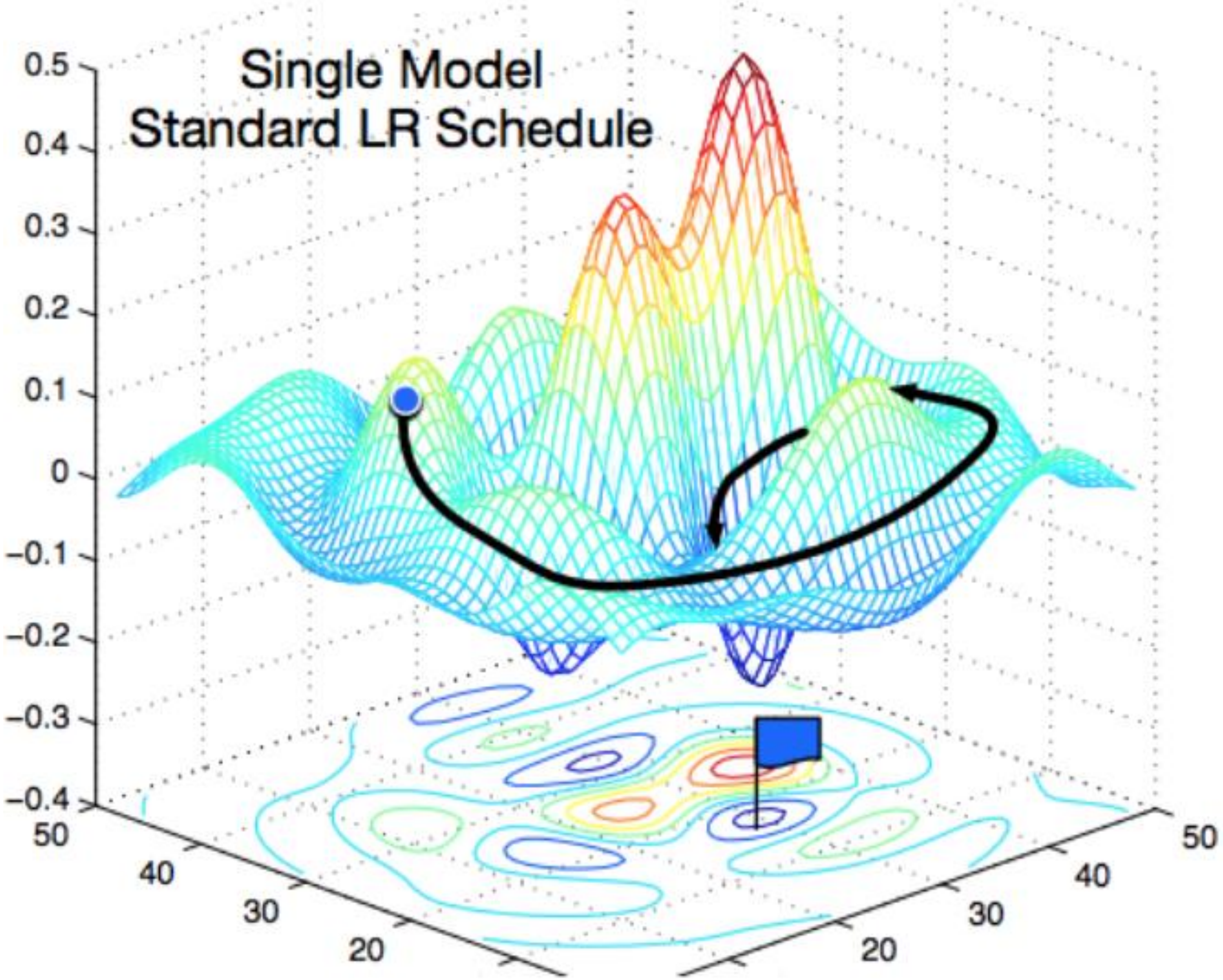
$$\nabla L(\theta; X, y) = \sum_{i=1}^{n} \nabla \ell(\theta; x_i, y_i)$$

- This is extremely inefficient!

- On big datasets, better to update based on a few examples

| **Stochastic Gradient Descent** |
|---|
| <u>Input:</u> dataset $(\mathrm{X}, \mathrm{y})$, loss function $L$, number of steps $T$, step size $\eta$, batch size $m$ |
| 1. Initialize $\theta_0$<br>2. For $t = 1,2, \dots, T$<br>3. Select random $(x_1, y_1), \dots, (x_m, y_m)$<br>4. Calculate $g_t = \sum_{i=1}^{m} \nabla_\theta \ell(\theta_{t-1}; x_i, y_i)$<br>5. Update $\theta_t \leftarrow \theta_{t-1} - \eta g_t$<br>6. Return $\theta_T$ |

# How do we get the gradient?

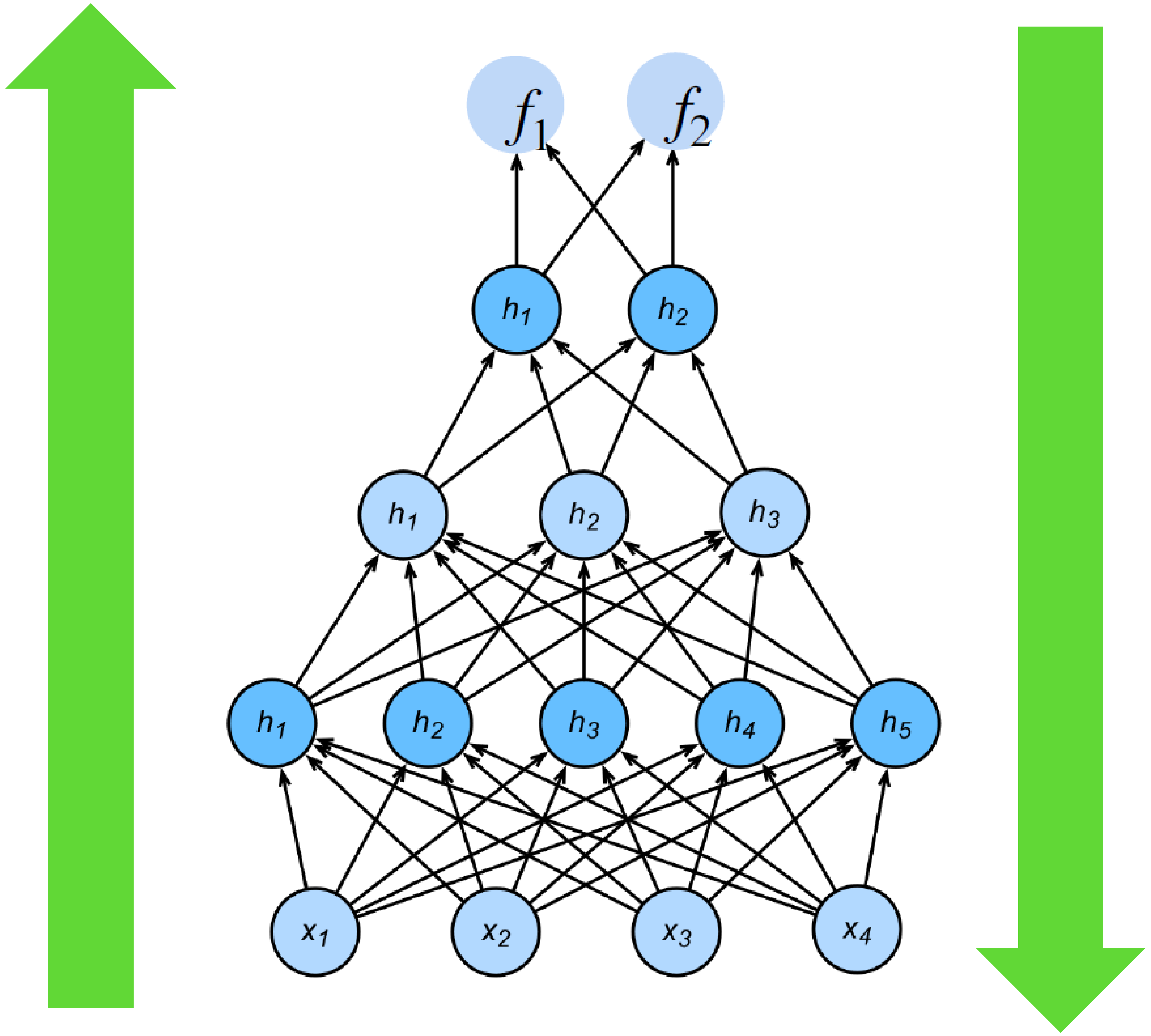



[Gao and Li et al., 2018]

# Backpropagation: An Efficient Algorithm for Gradients in Neural Networks

Forward pass:

Start with input layer, compute all hidden nodes and outputs layer-by-layer




Backward pass:

Start with output layer, compute all partial derivatives layer-by-layer

# Derivatives of functions of single variables

- For many functions $f(x)$ we have simple rules to find the derivative.
- If $f(x) = cx^2$ then $\frac{df}{dx} = 2cx$
  - Or more generally, if $f(x) = cx^p$ then $\frac{df}{dx} = cpx^{p-1}$
- If $f(x) = \log x$ then $\frac{df}{dx} = \frac{1}{x}$
- If $f(x) = c$ then $\frac{df}{dx} = 0$

https://www.storyofmathematics.com/derivative-calculus/

# More Complex Functions

- Derivation rules can be applied hierarchically to find derivatives of more complex functions.
- **Sum rule:** If $f(x) = h(x) + g(x)$ then $\frac{df}{dx} = \frac{dh}{dx} + \frac{dg}{dx}$
- **Product rule:** If $f(x) = h(x) \cdot g(x)$ then $\frac{df}{dx} = h(x)\frac{dg}{dx} + g(x)\frac{dh}{dx}$
- **Chain rule:** If $f(x) = h(g(x))$ then $\frac{df}{dx} = \frac{dh}{dg}\frac{dg}{dx}$
- **More complex chain rule:** if $f(x) = f_1(f_2(\cdots f_n(x)\cdots))$ then $\frac{df}{dx} = \frac{df_1}{df_2}\frac{df_2}{df_3}\cdots\frac{df_n}{dx}$

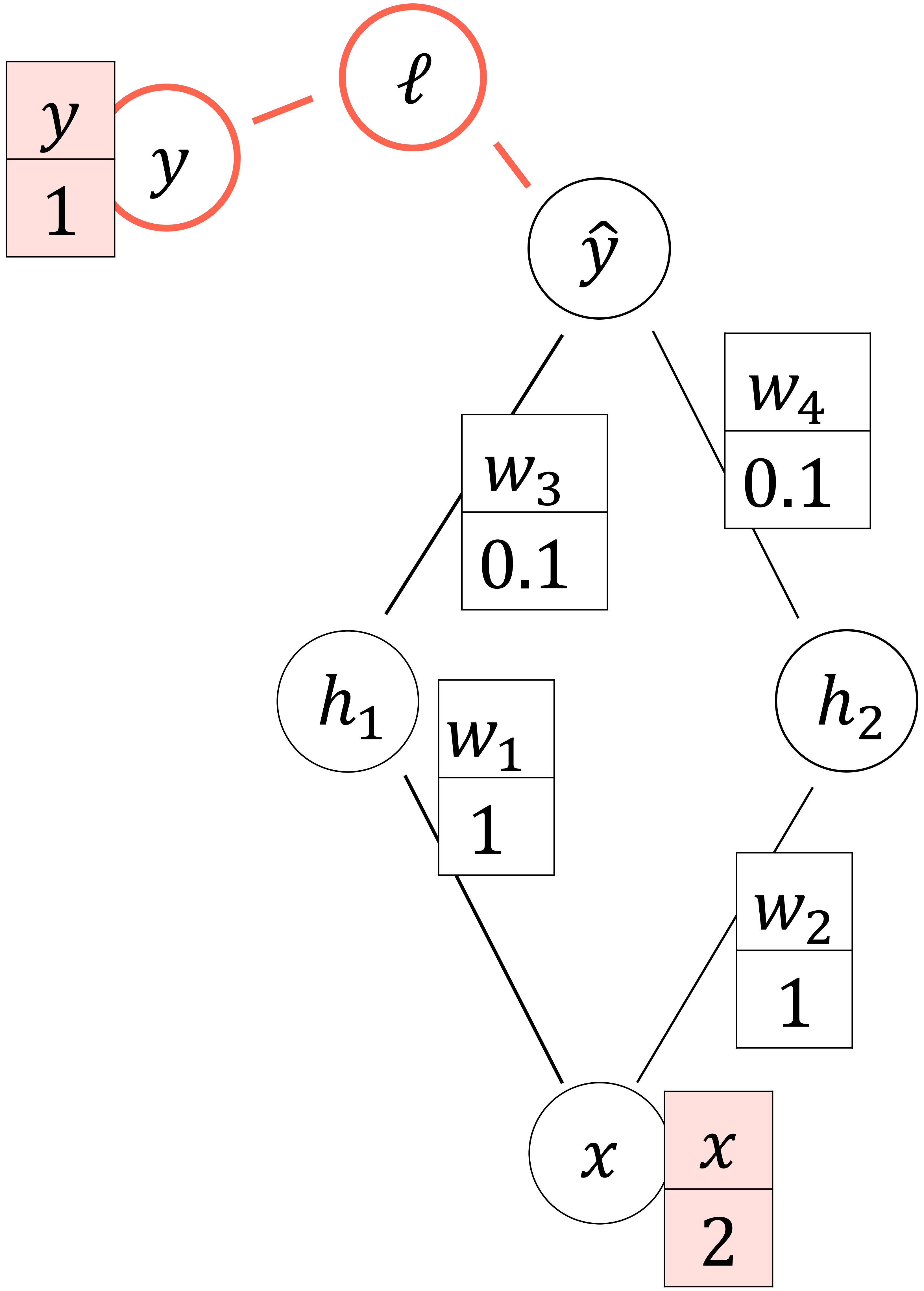
https://www.storyofmathematics.com/derivative-calculus/

# Computing Partial Derivatives

- Rules from single-variable calculus directly extend to multivariate calculus with one small change.
  - When computing $\frac{\partial f}{\partial x_i}$, treat all other variables as constants.
- Examples:
  - If $f(x_1, x_2) = \log x_1 + \log x_2$ then $\frac{\partial f}{\partial x_1} = \frac{1}{x_1}$
  - If $f(x_1, x_2) = x_1(x_1 + x_2)$ then $\frac{\partial f}{\partial x_2} = x_1$

https://www.storyofmathematics.com/derivative-calculus/

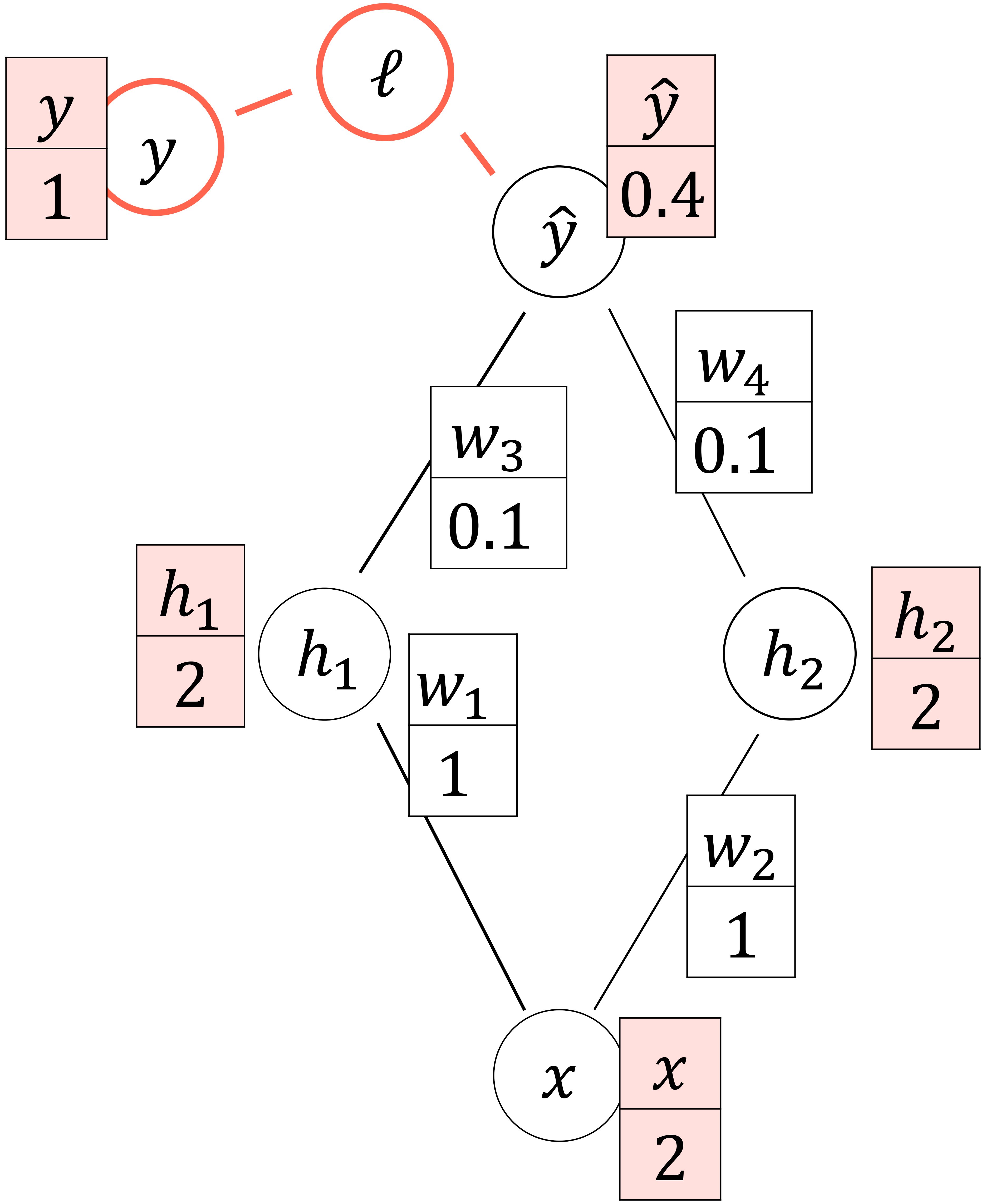
# Back to Our Simple Example




Drop bias terms to simplify the picture

Add in notation for loss, and data point $(x, y) = (2,1)$

# Back to Our Simple Example

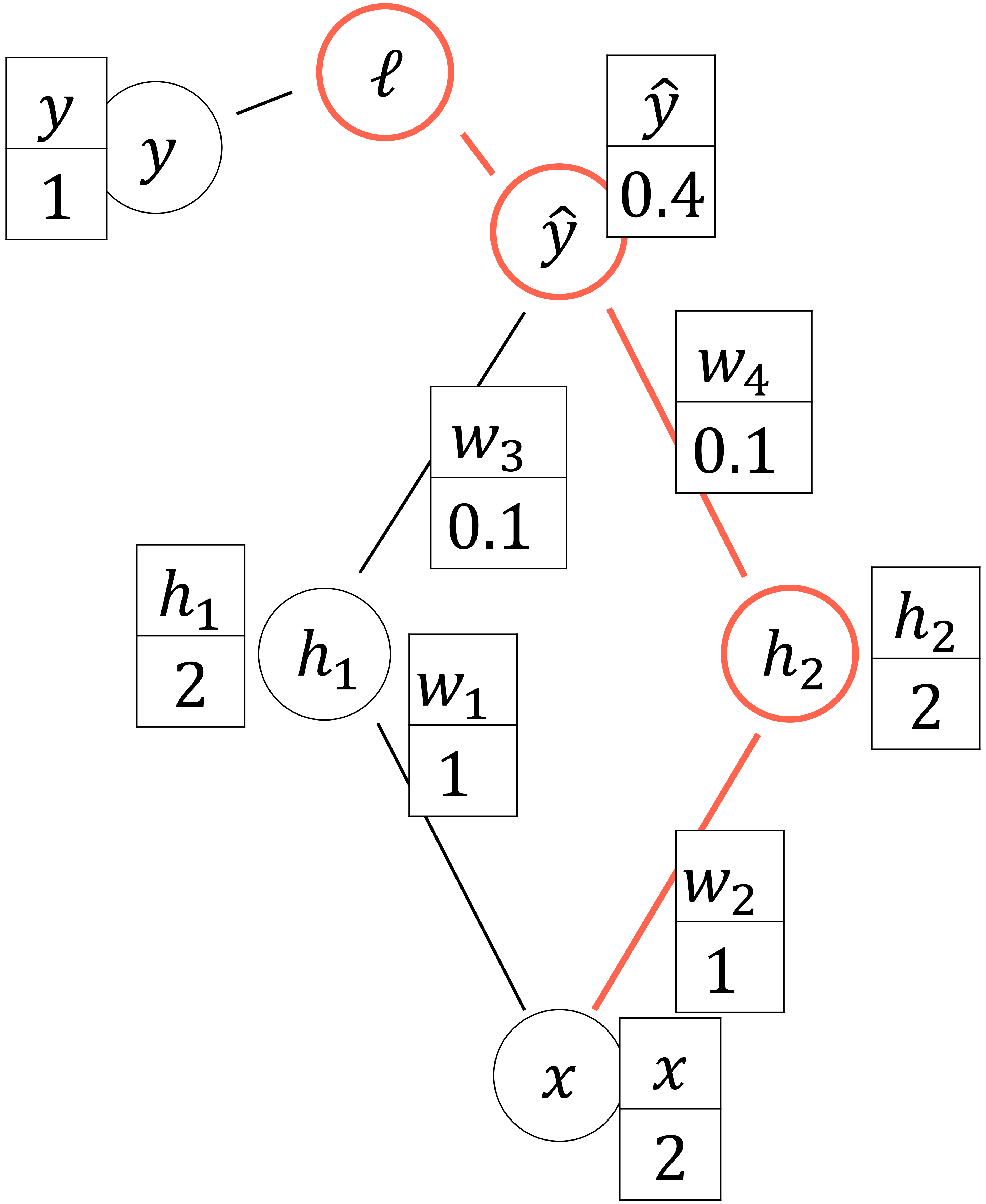



Drop bias terms to simplify the picture

Add in notation for loss, and data point $(x, y) = (2,1)$

We already performed forward pass, our algorithm stored output, intermediate hidden node values

# Back to Our Simple Example



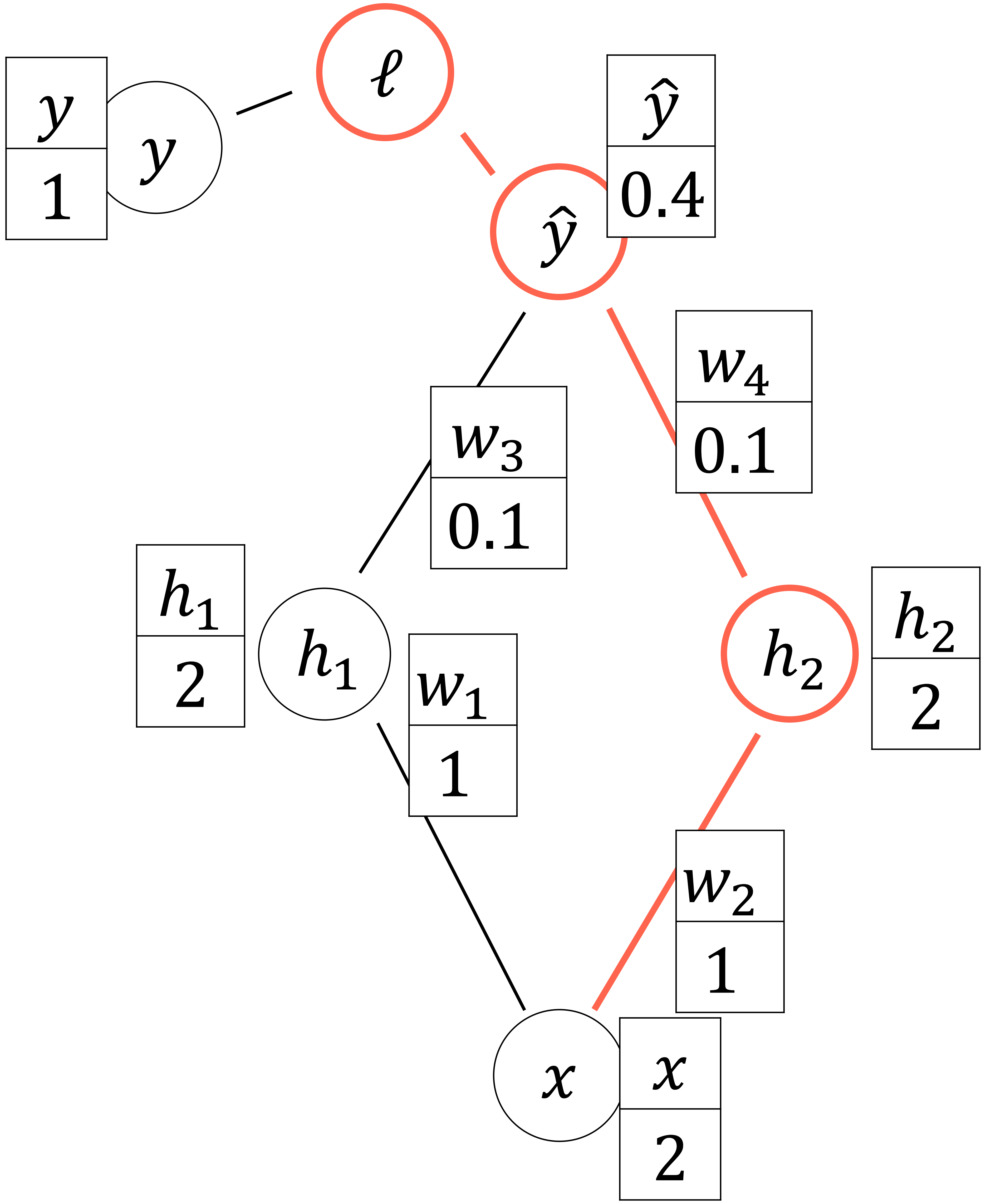

Use the squared loss: $\ell(y, \hat{y}) = (\hat{y} - y)^2$

Let's compute $\frac{\partial \ell}{\partial w_2}$

Orange lines: single path from $w_2$ to loss

Chain rule: $\frac{\partial \ell}{\partial w_2} = \frac{\partial \ell}{\partial \hat{y}} \times \frac{\partial \hat{y}}{\partial h_2} \times \frac{\partial h_2}{\partial w_2}$

# Back to Our Simple Example



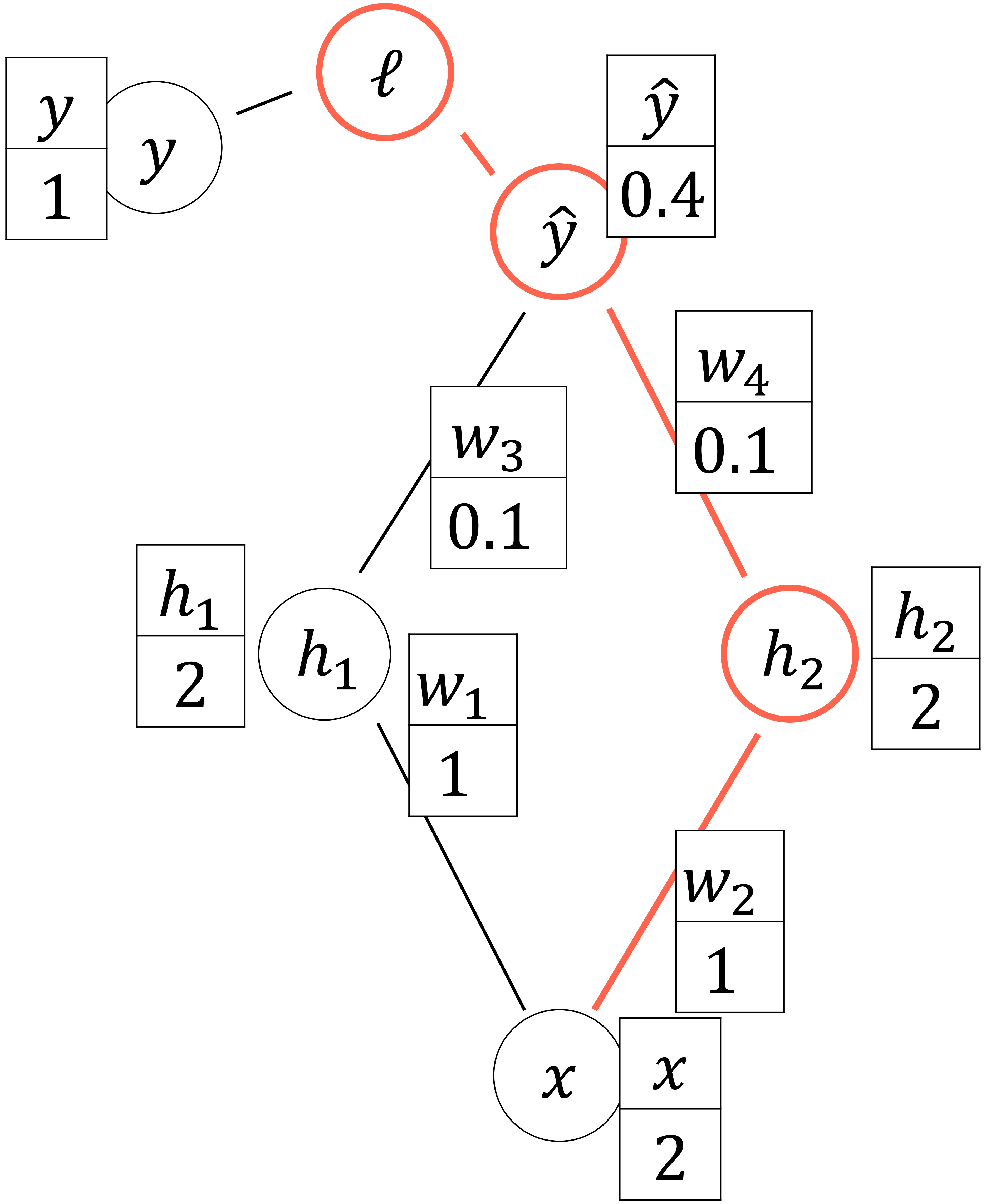

Chain rule: $\frac{\partial \ell}{\partial w_2} = \frac{\partial \ell}{\partial \hat{y}} \times \frac{\partial \hat{y}}{\partial h_2} \times \frac{\partial h_2}{\partial w_2}$

First term:

$$\ell = (\hat{y} - y)^2$$

$$\frac{\partial \ell}{\partial \hat{y}} = 2(\hat{y} - y) = -1.2$$

# Back to Our Simple Example



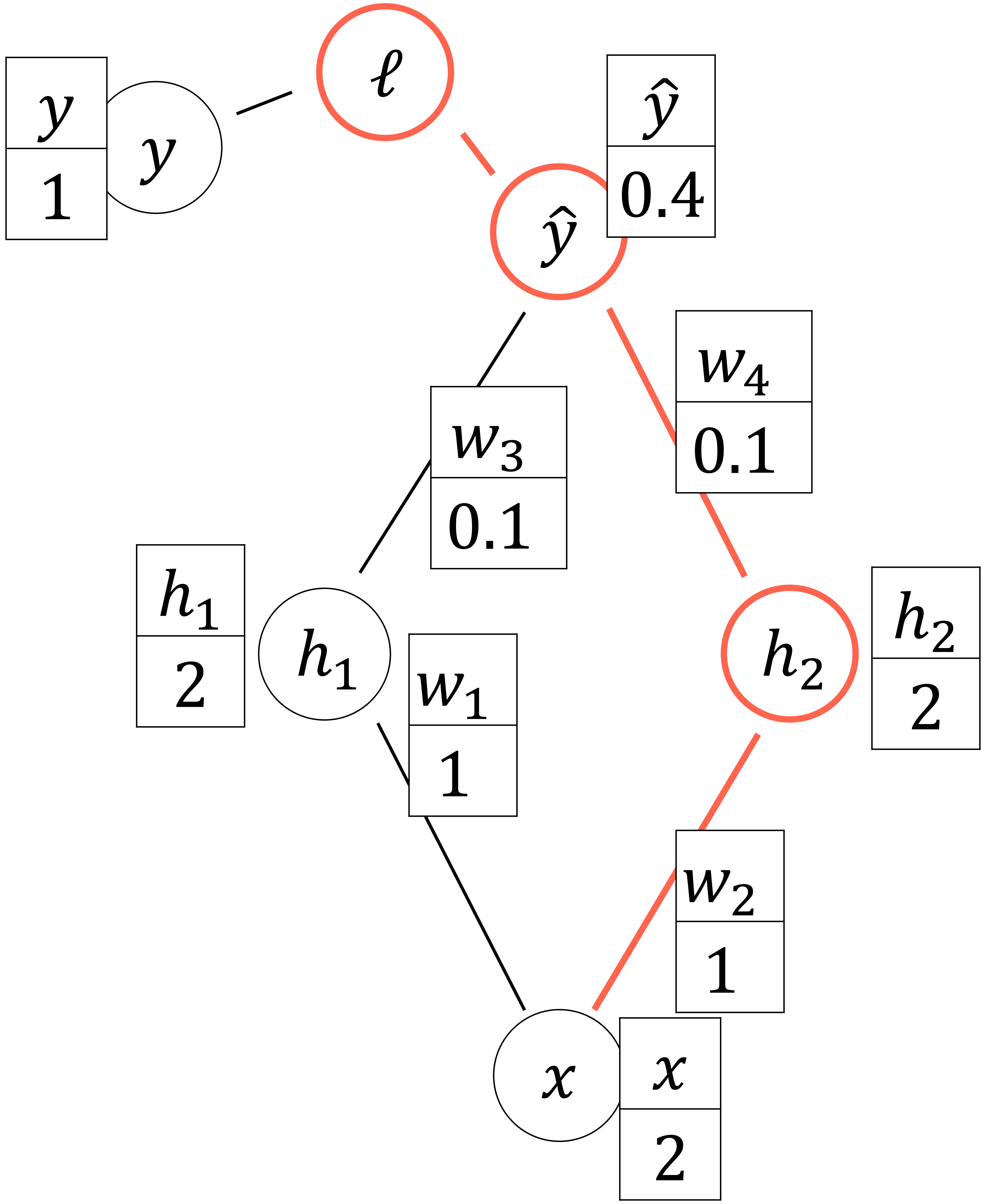

Chain rule: $\frac{\partial \ell}{\partial w_2} = \frac{\partial \ell}{\partial \hat{y}} \times \frac{\partial \hat{y}}{\partial h_2} \times \frac{\partial h_2}{\partial w_2}$

Second term:

$$\hat{y} = w_3 h_1 + w_4 h_2$$

$$\frac{\partial \hat{y}}{\partial h_2} = w_4 = 0.1$$

# Back to Our Simple Example




Chain rule: $\frac{\partial \ell}{\partial w_2} = \frac{\partial \ell}{\partial \hat{y}} \times \frac{\partial \hat{y}}{\partial h_2} \times \frac{\partial h_2}{\partial w_2}$

Putting it all together:

$$\frac{\partial \ell}{\partial w_2} = (-1.2) \times (0.1) \times (2) = -.24$$

# The Backpropagation Algorithm: Layer-by-Layer, Backwards

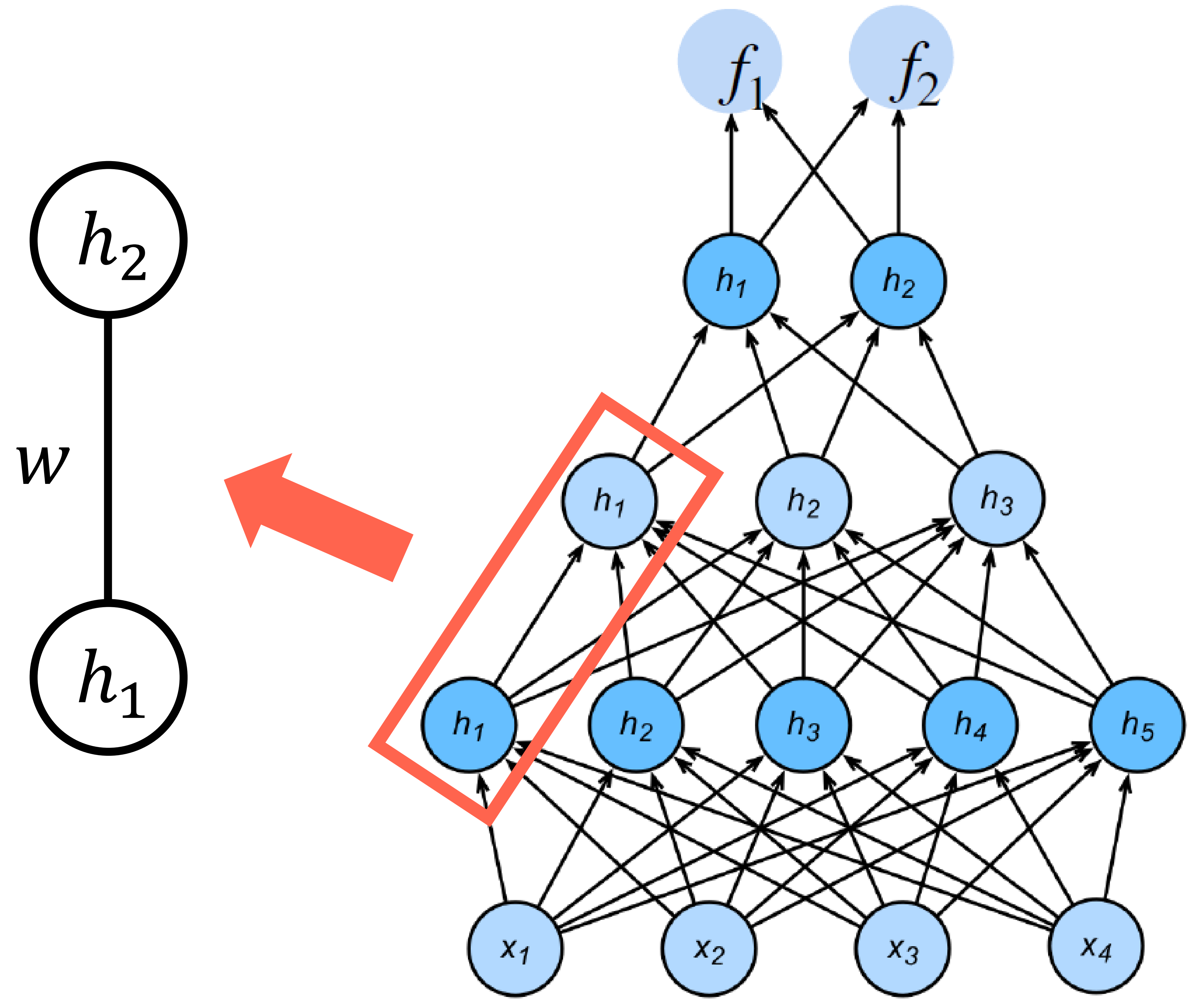
Want to find partial derivative for weight $w$ in middle of network

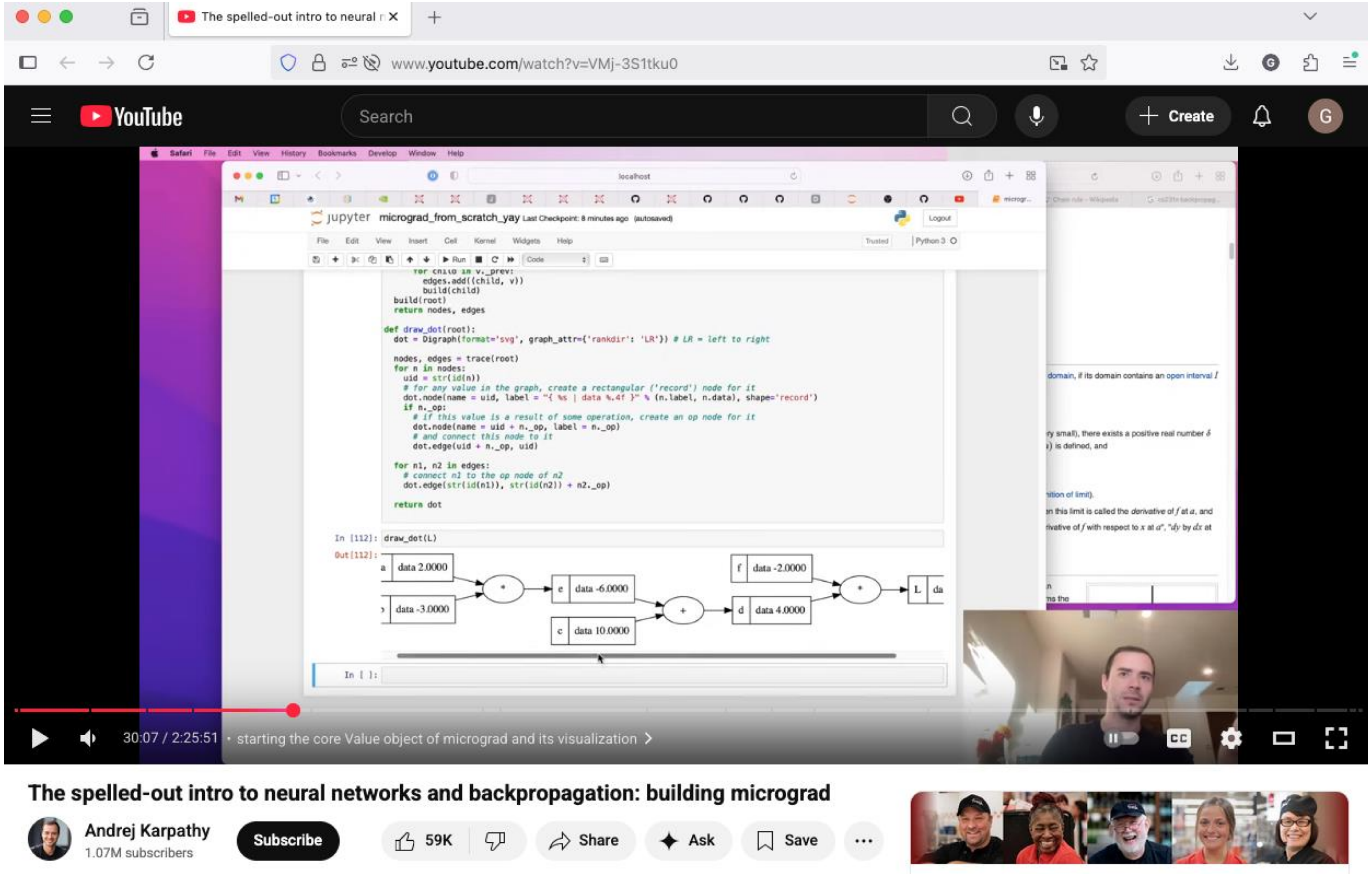
Connects from $h_1$ to $h_2$

Chain rule: $\frac{\partial \ell}{\partial w} = \frac{\partial \ell}{\partial h_2} \times \frac{\partial h_2}{\partial w}$

We already computed this

This is simple to compute



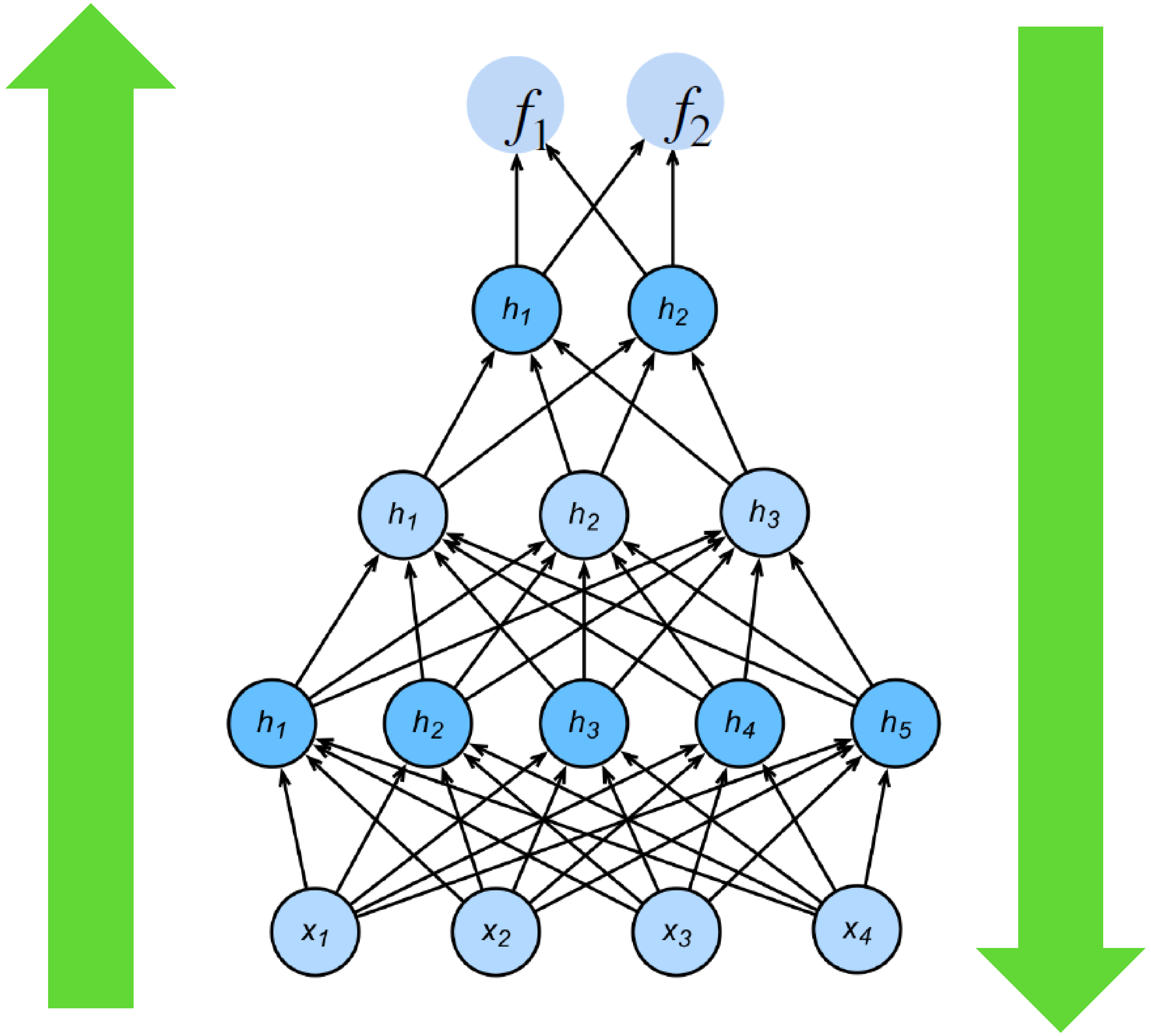



Highly recommended: https://karpathy.ai/zero-to-hero.html

# Backpropagation: An Efficient Algorithm for Gradients in Neural Networks

Forward pass:

Start with input layer, compute all hidden nodes and outputs layer-by-layer

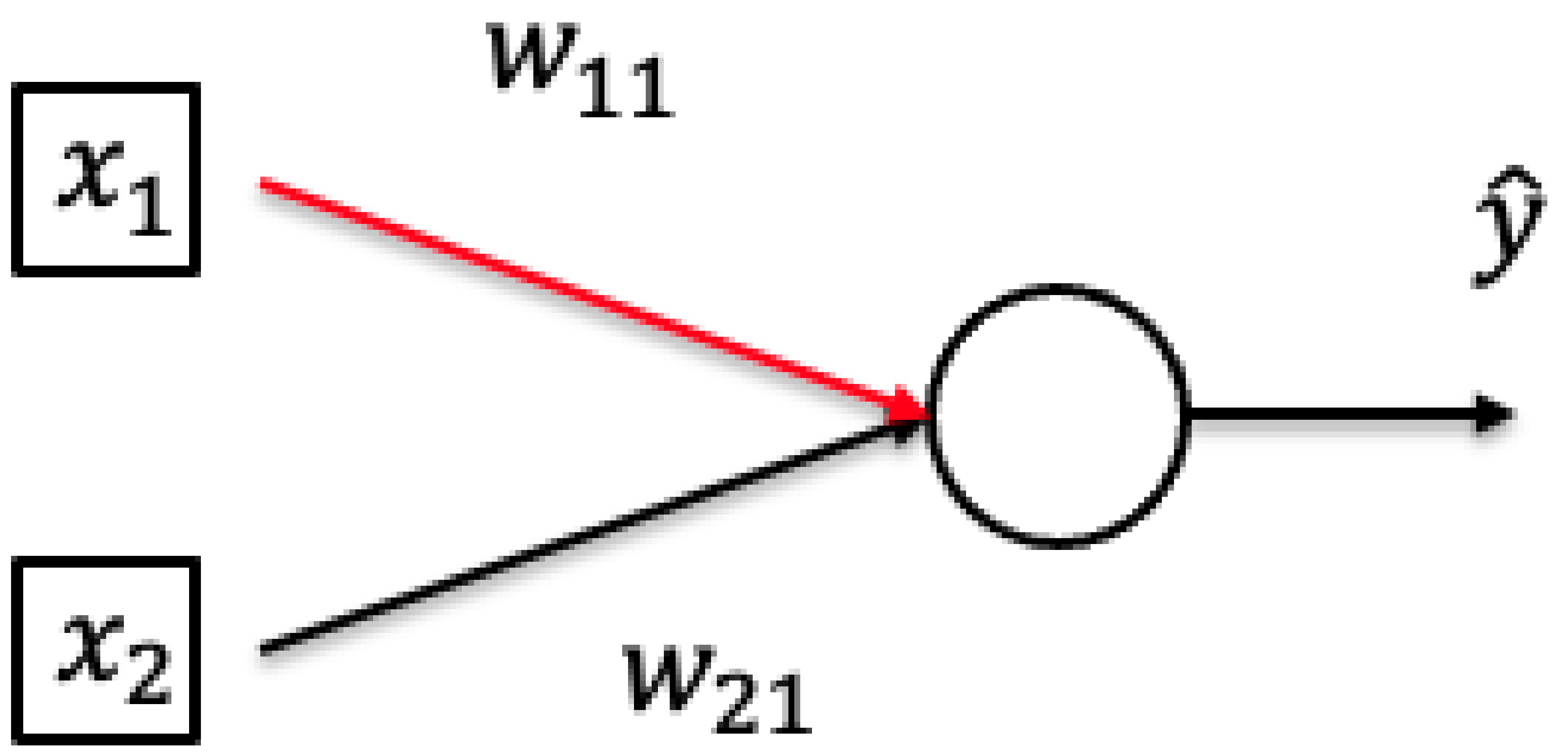



Backward pass:

Start with output layer, compute all partial derivatives layer-by-layer

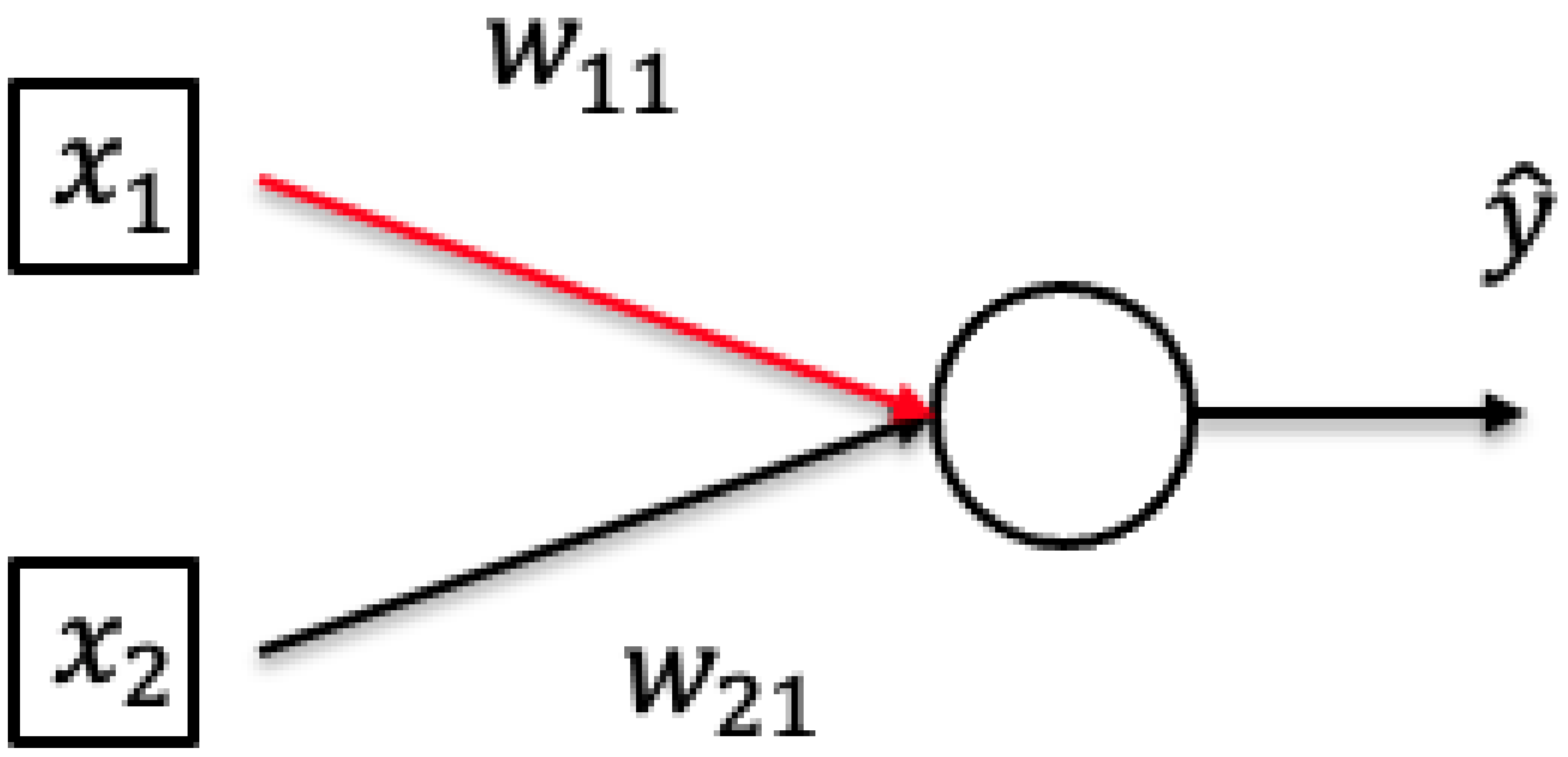
# Bonus: More Backprop Math

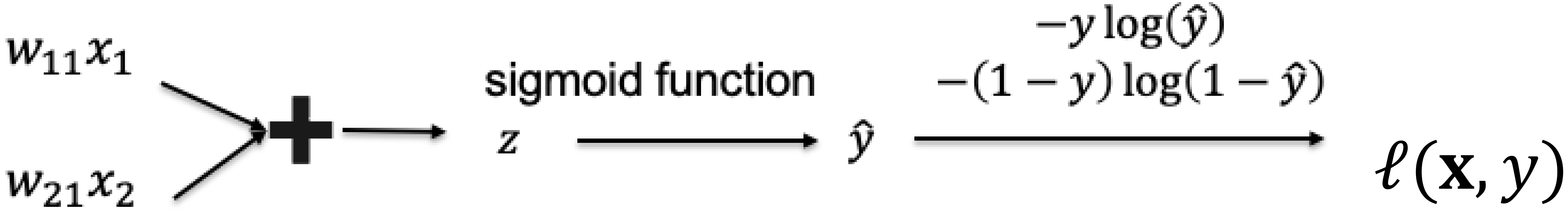
# Calculate Gradient (on one data point)




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

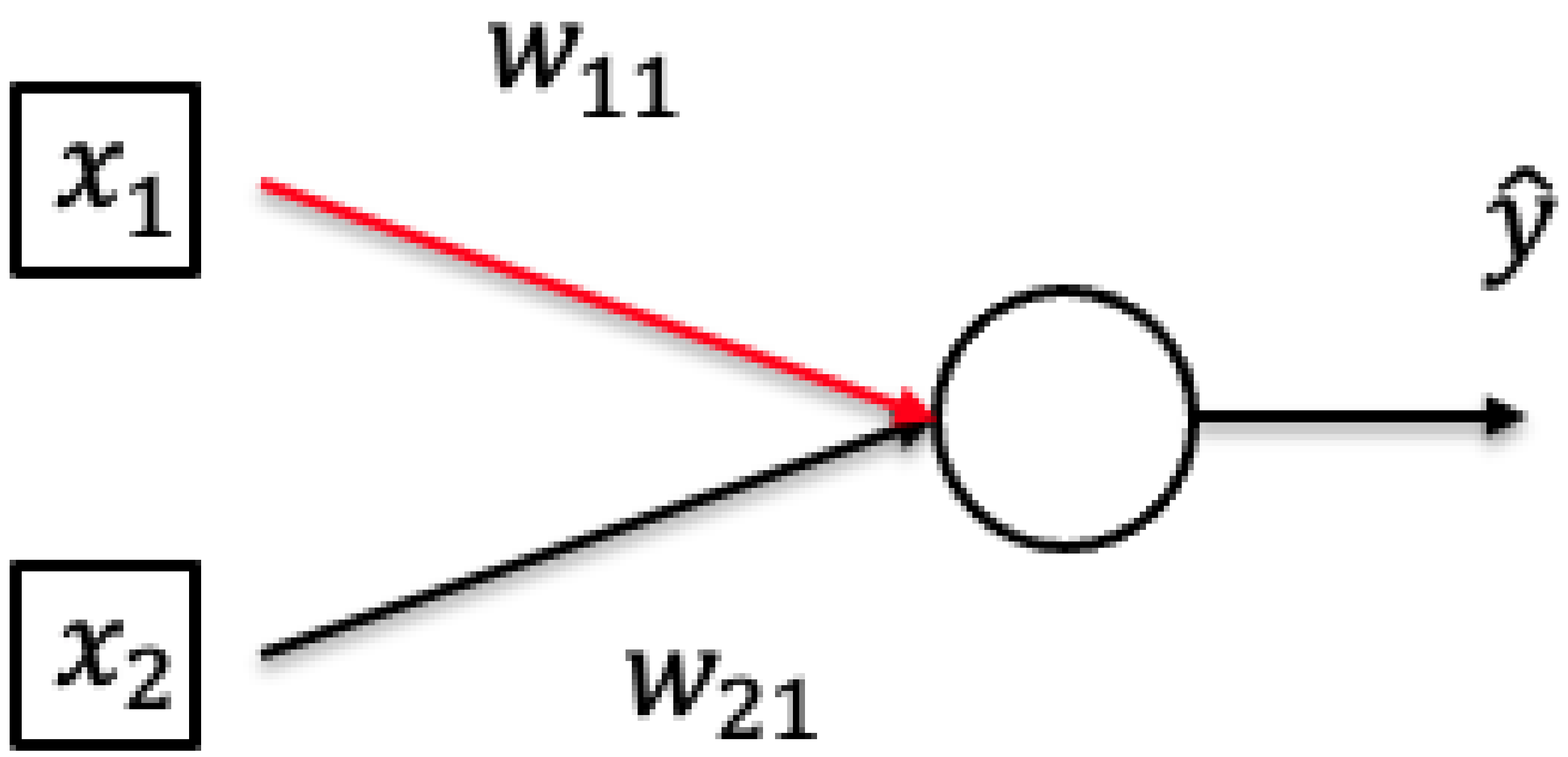
# Calculate Gradient (on one data point)




Use chain rule!

# Calculate Gradient (on one data point)

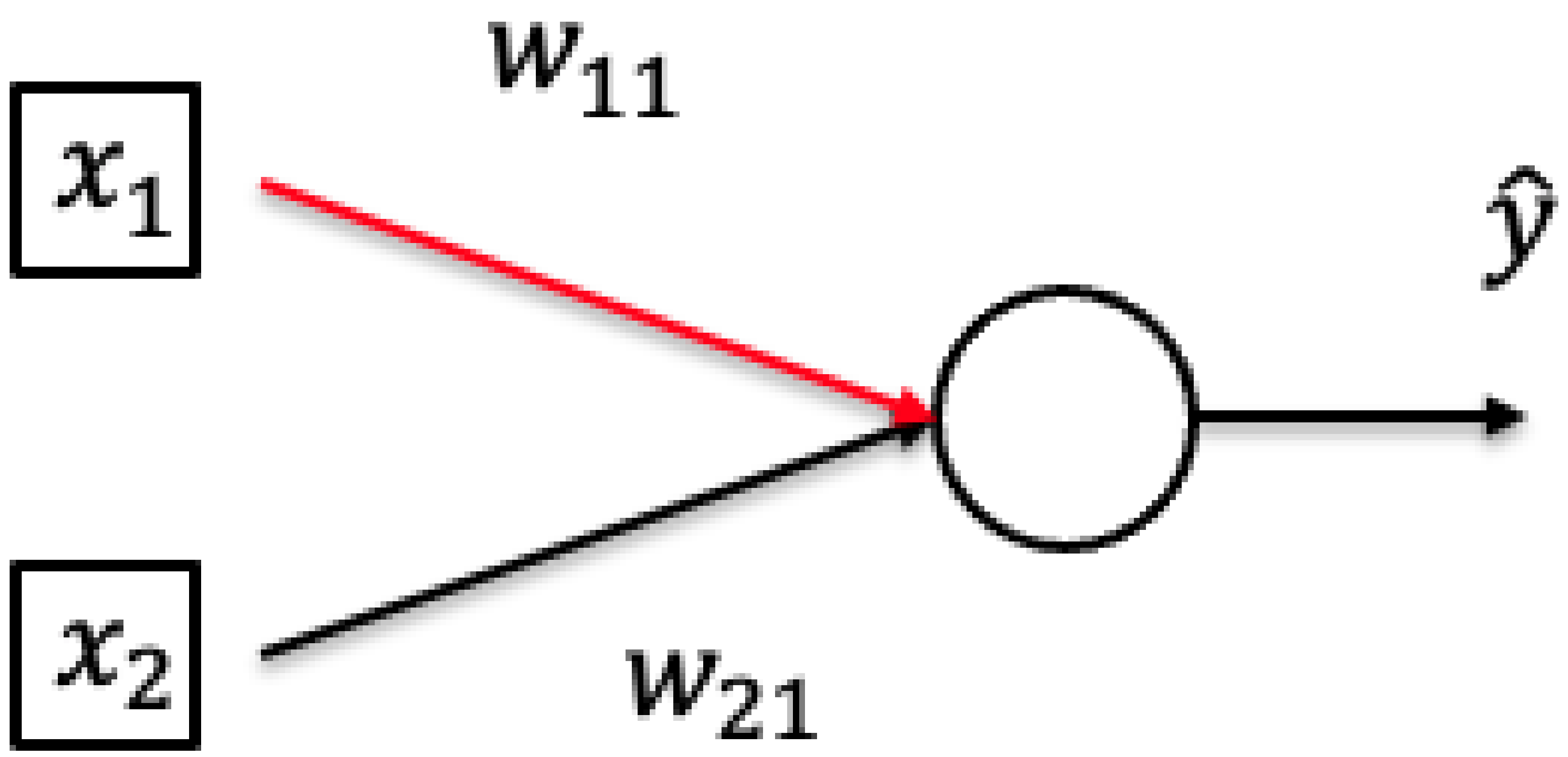


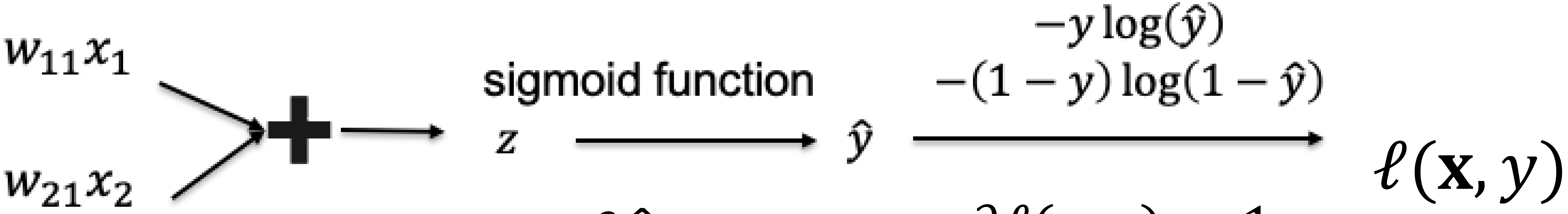

$$\left.\begin{matrix} w_{11}x_1 \\ w_{21}x_2 \end{matrix}\right\} + \longrightarrow z \xrightarrow{\text{sigmoid function}} \hat{y} \xrightarrow{\begin{matrix}-y\log(\hat{y}) \\ -(1-y)\log(1-\hat{y})\end{matrix}} \ell(\mathbf{x}, y)$$

$$\frac{\partial \hat{y}}{\partial z} = \sigma'(z) \qquad \frac{\partial \ell(\mathbf{x}, y)}{\partial \hat{y}} = \frac{1-y}{1-\hat{y}} - \frac{y}{\hat{y}}$$

- 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)

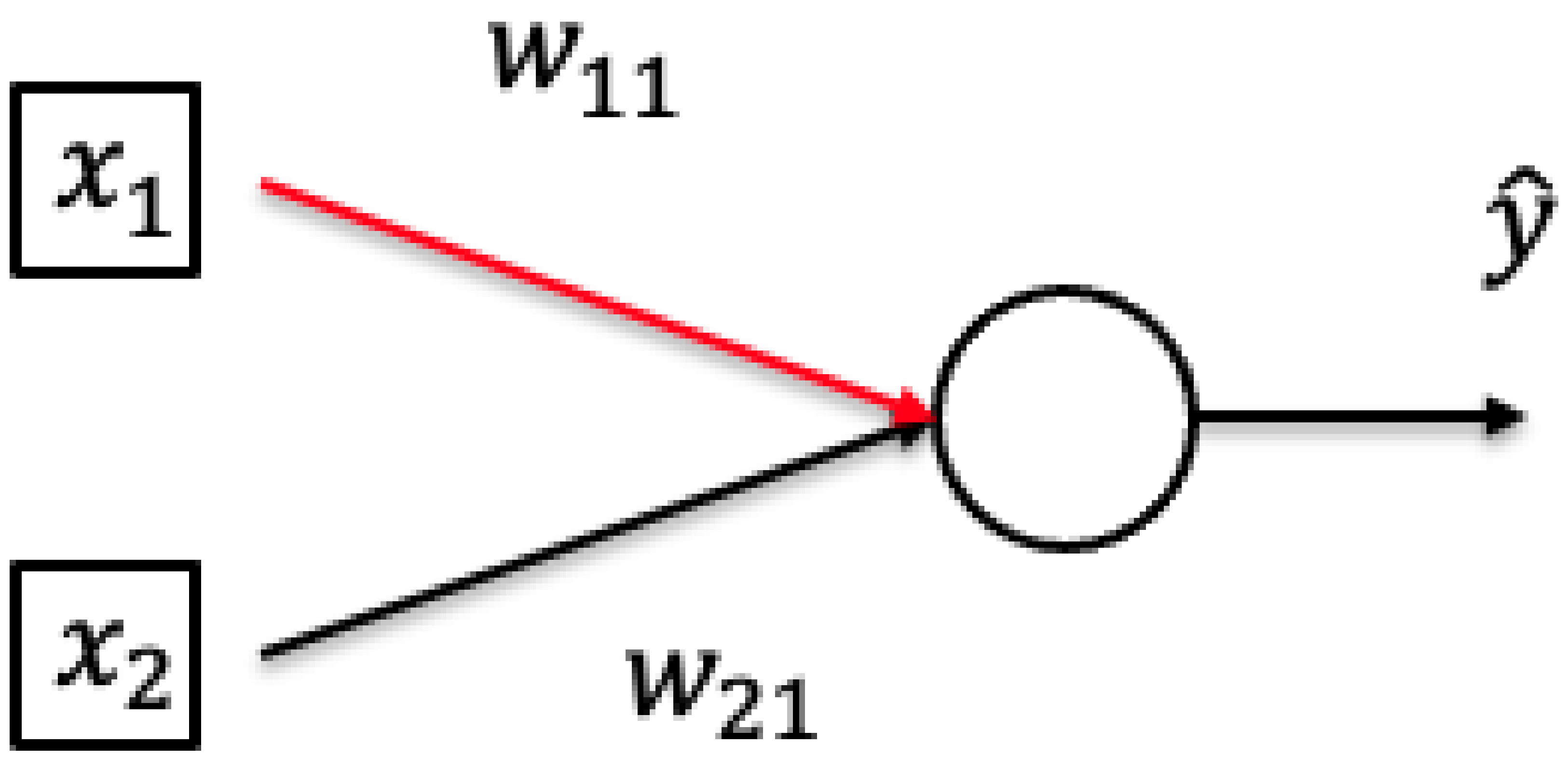


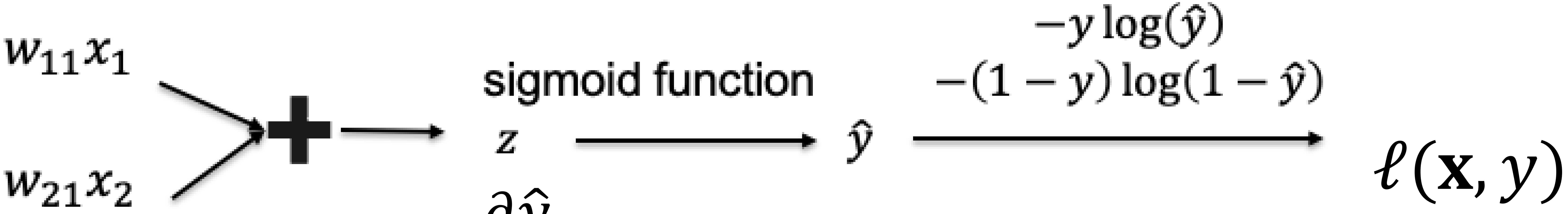




$$\frac{\partial \hat{y}}{\partial z} = \sigma'(z) \qquad \frac{\partial \ell(\mathbf{x}, y)}{\partial \hat{y}} = \frac{1-y}{1-\hat{y}} - \frac{y}{\hat{y}}$$

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

# Calculate Gradient (on one data point)

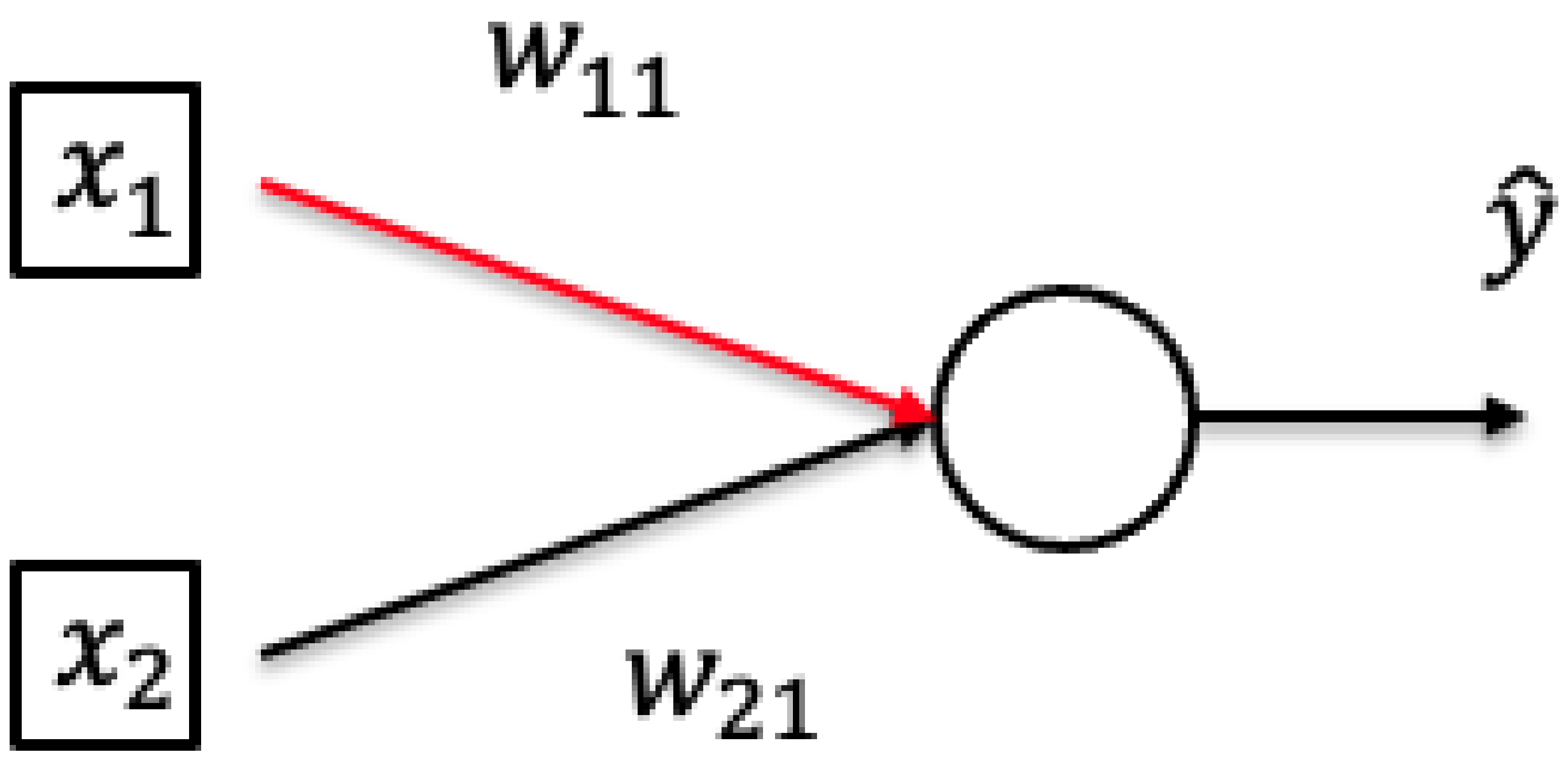


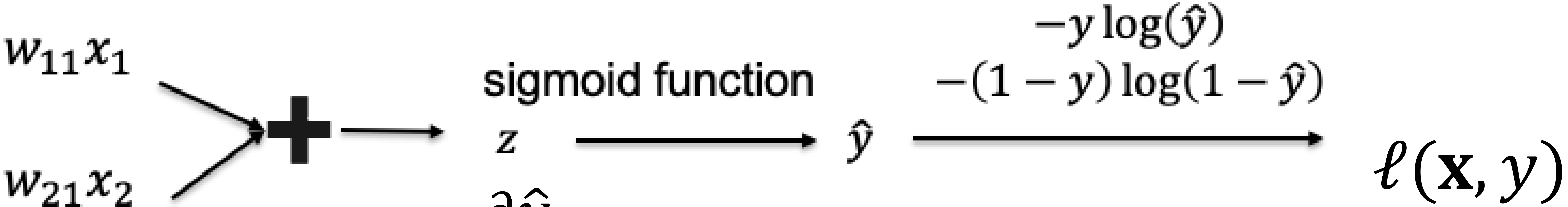




$$\frac{\partial \hat{y}}{\partial z} = \sigma'(z) = \sigma(z)(1-\sigma(z))$$

- By chain rule: $\dfrac{\partial l}{\partial w_{11}} = \dfrac{\partial l}{\partial \hat{y}} \; \hat{y}(1-\hat{y})x_1$

# Calculate Gradient (on one data point)

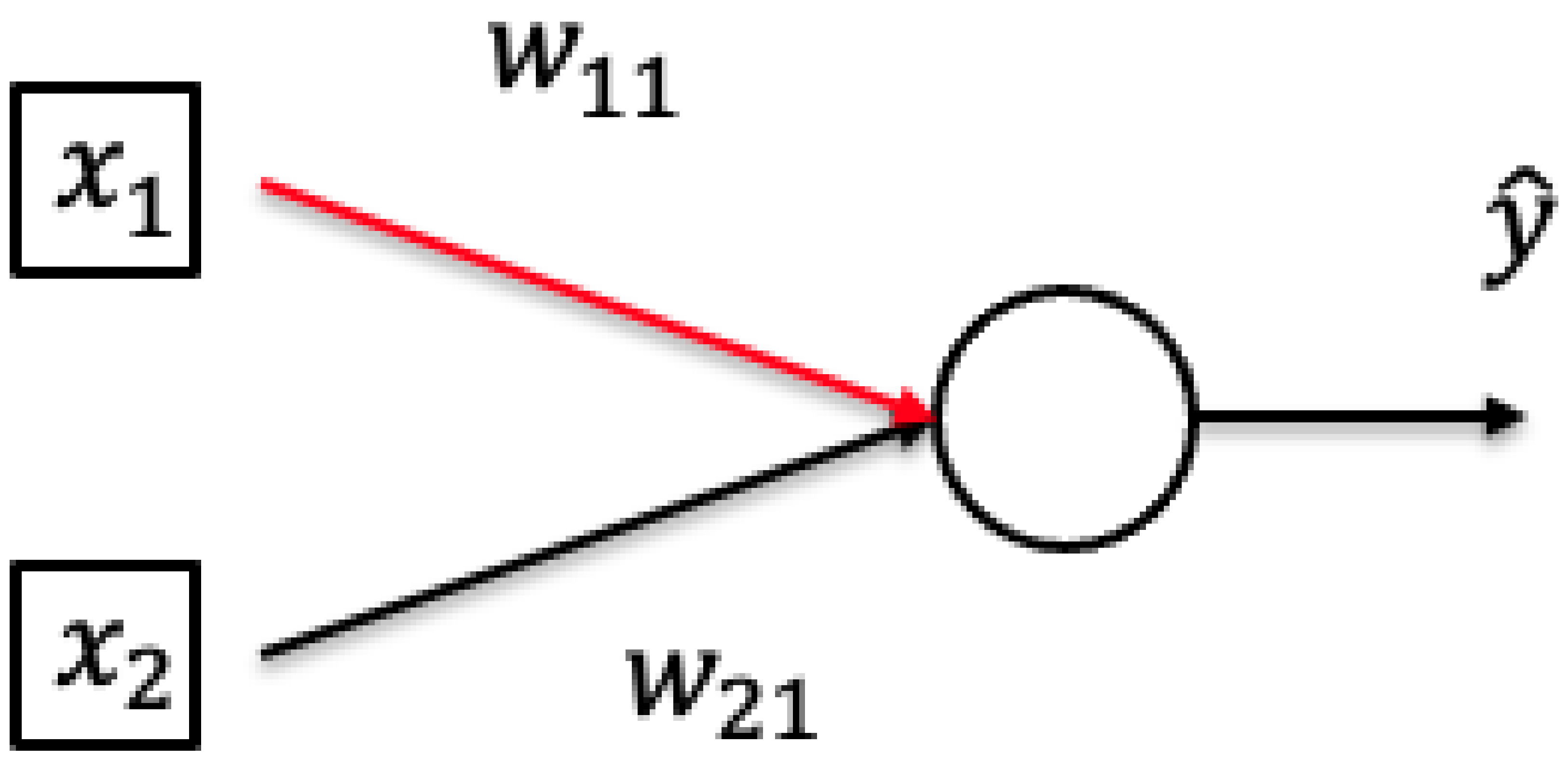


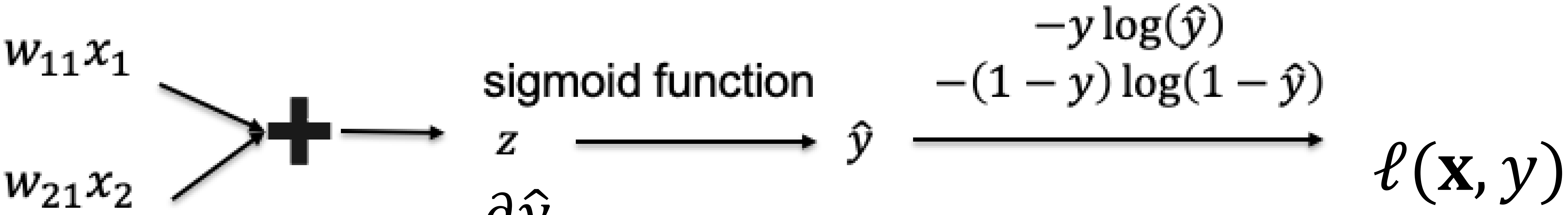




$$\frac{\partial \hat{y}}{\partial z} = \sigma'(z) = \sigma(z)(1-\sigma(z))$$

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

# Calculate Gradient (on one data point)

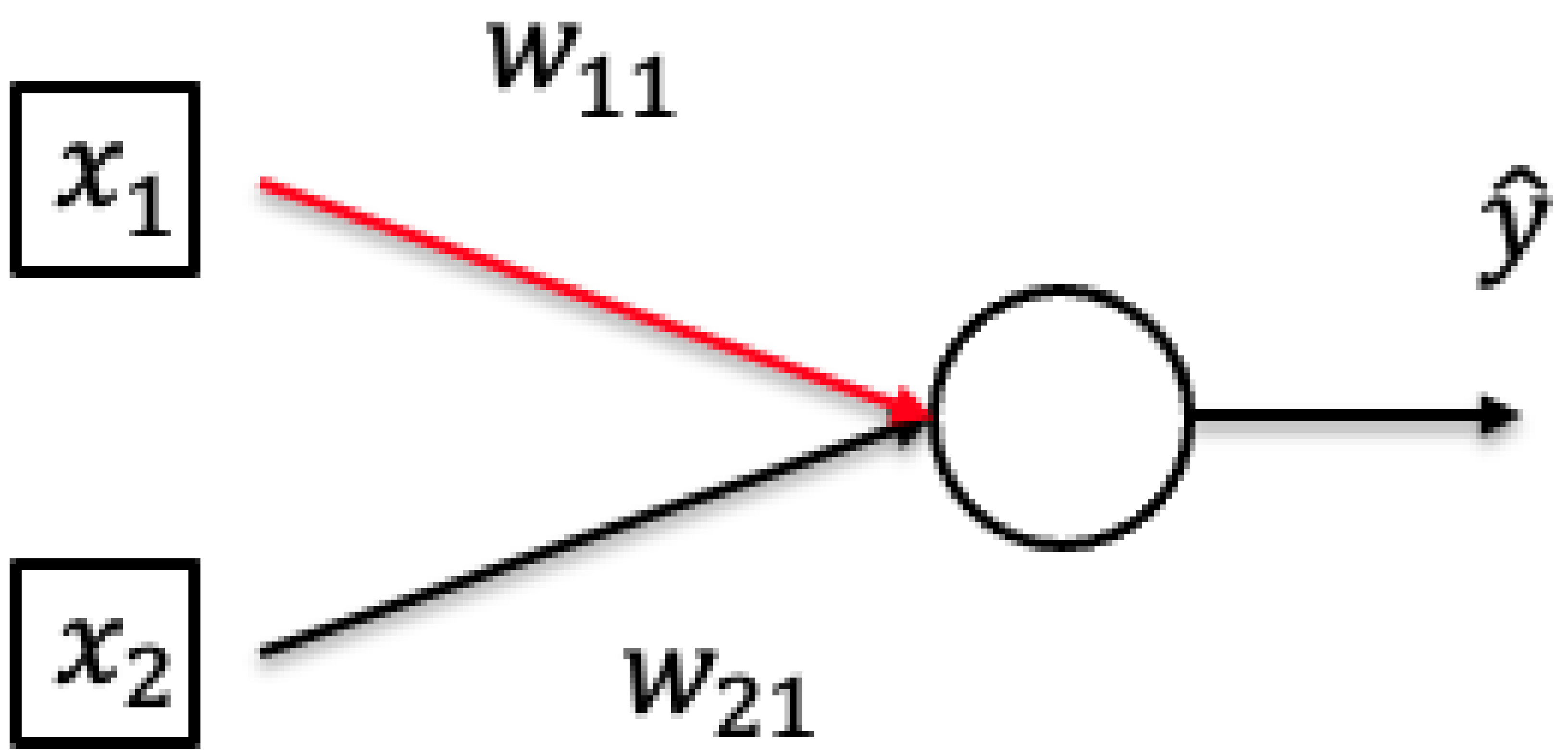


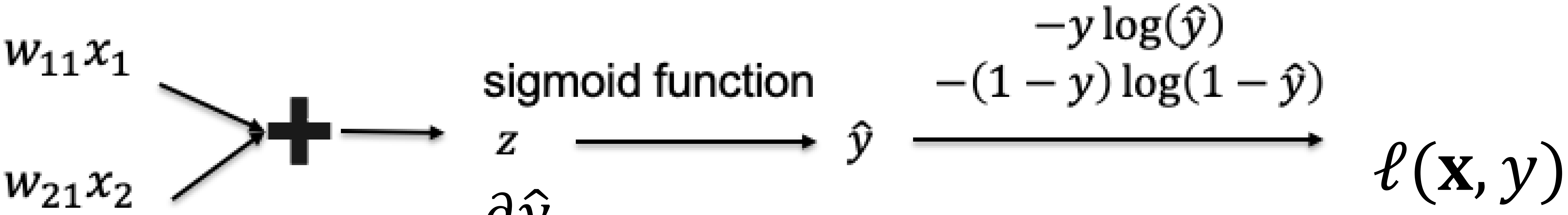




$$\frac{\partial \hat{y}}{\partial z} = \sigma'(z) = \sigma(z)(1 - \sigma(z))$$

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

# Calculate Gradient (on one data point)

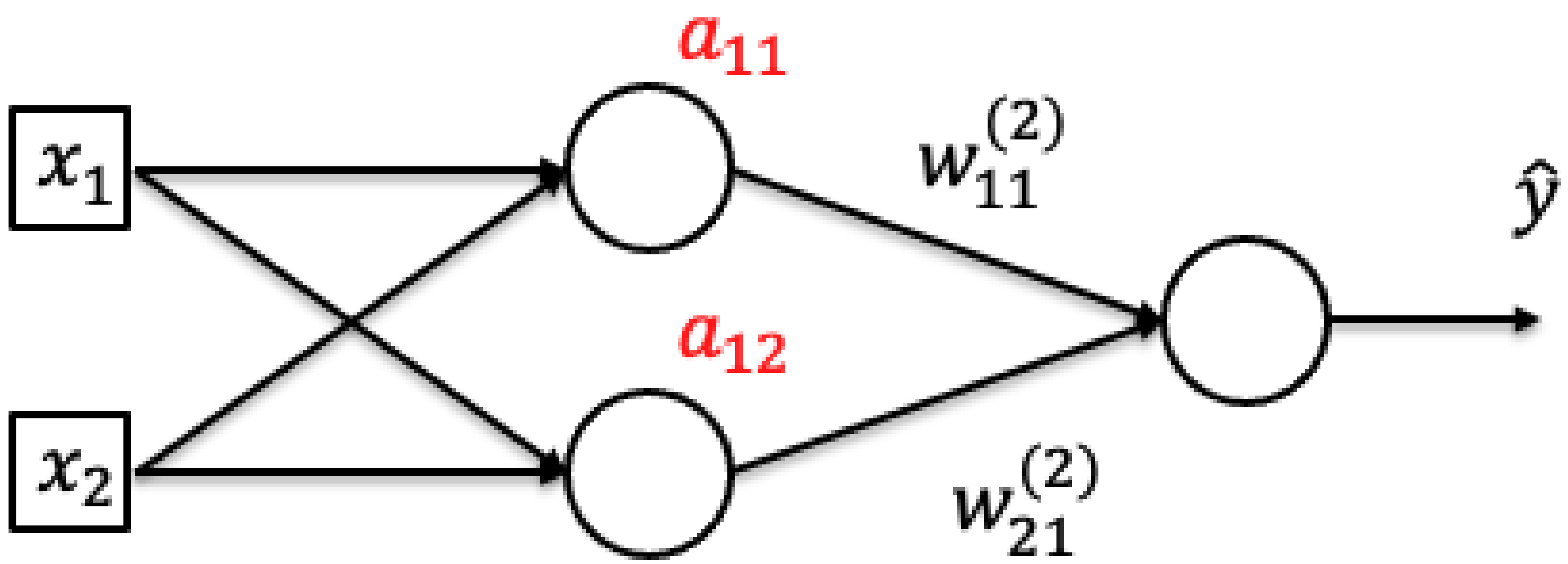


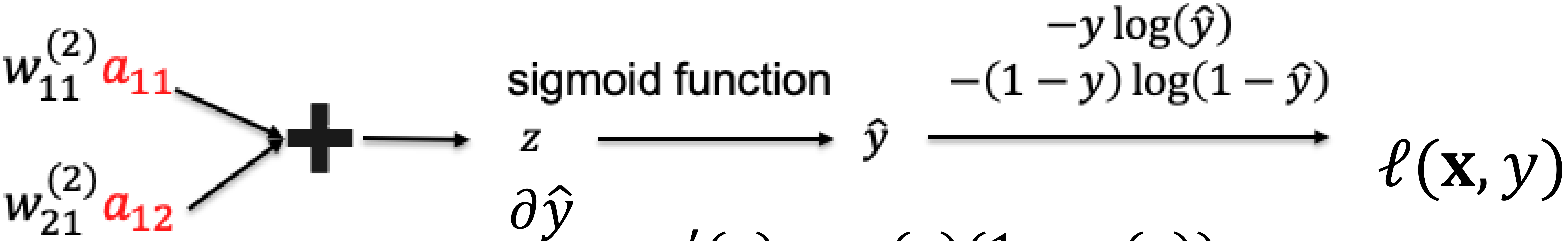

$$\frac{\partial \hat{y}}{\partial z} = \sigma'(z) = \sigma(z)(1 - \sigma(z))$$

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

# Calculate Gradient (on one data point)



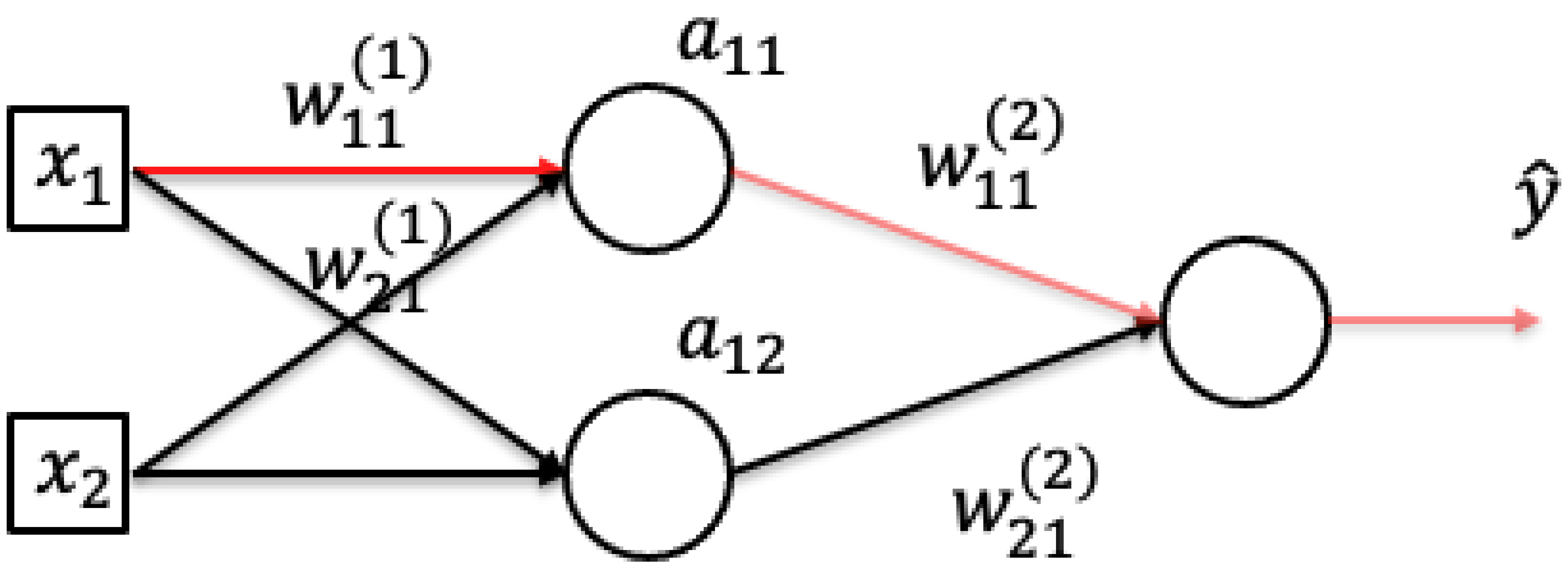

Make it deeper




$$\frac{\partial \hat{y}}{\partial z} = \sigma'(z) = \sigma(z)(1-\sigma(z))$$

- By chain rule: $\frac{\partial l}{\partial a_{11}} = (\hat{y} - y) w_{11}^{(2)}, \ \frac{\partial l}{\partial a_{12}} = (\hat{y} - y) w_{21}^{(2)}$

# Calculate Gradient (on one data point)

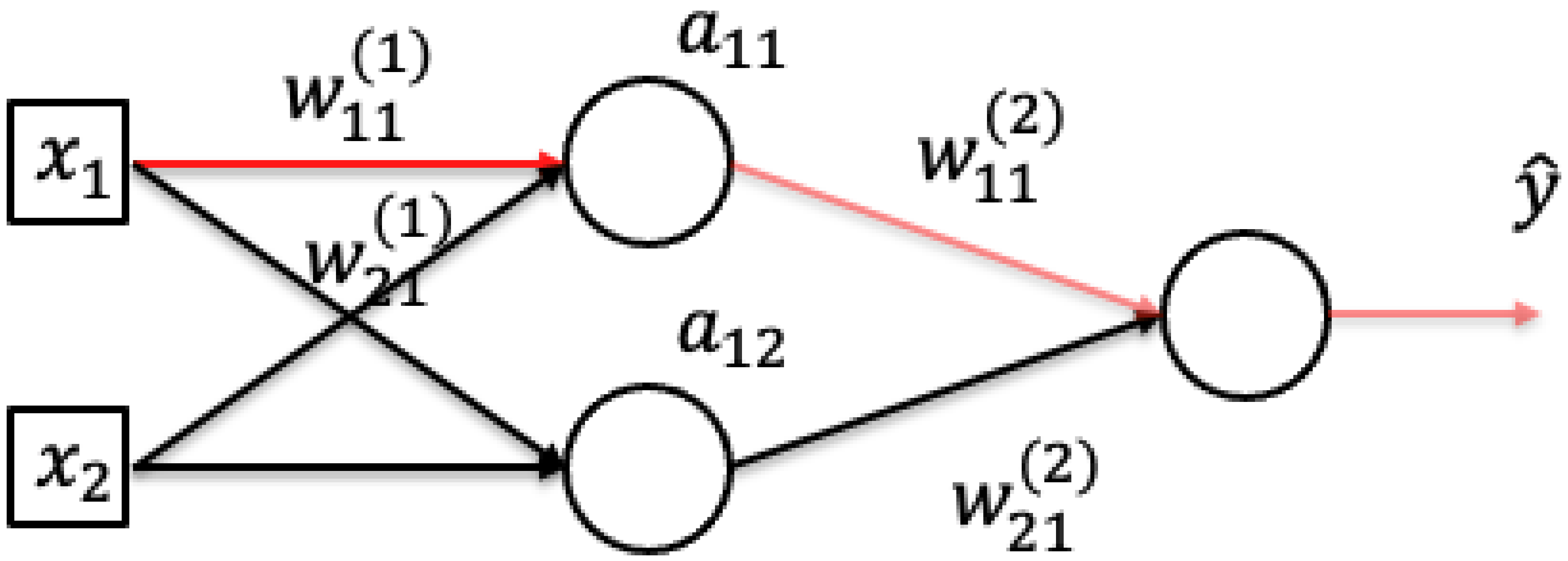


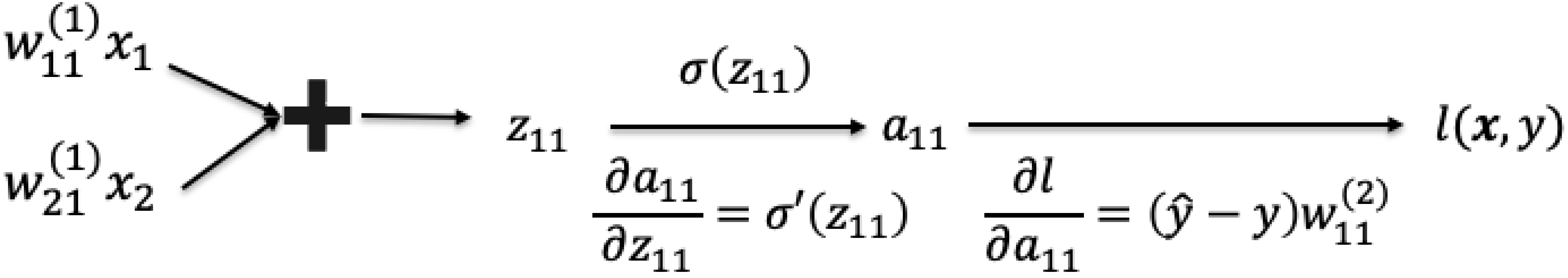

$$w_{11}^{(1)}x_1 + w_{21}^{(1)}x_2 \longrightarrow z_{11} \xrightarrow{\sigma(z_{11})} a_{11} \longrightarrow l(x, y)$$

$$\frac{\partial a_{11}}{\partial z_{11}} = \sigma'(z_{11}) \qquad \frac{\partial l}{\partial a_{11}} = (\hat{y} - y)w_{11}^{(2)}$$

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

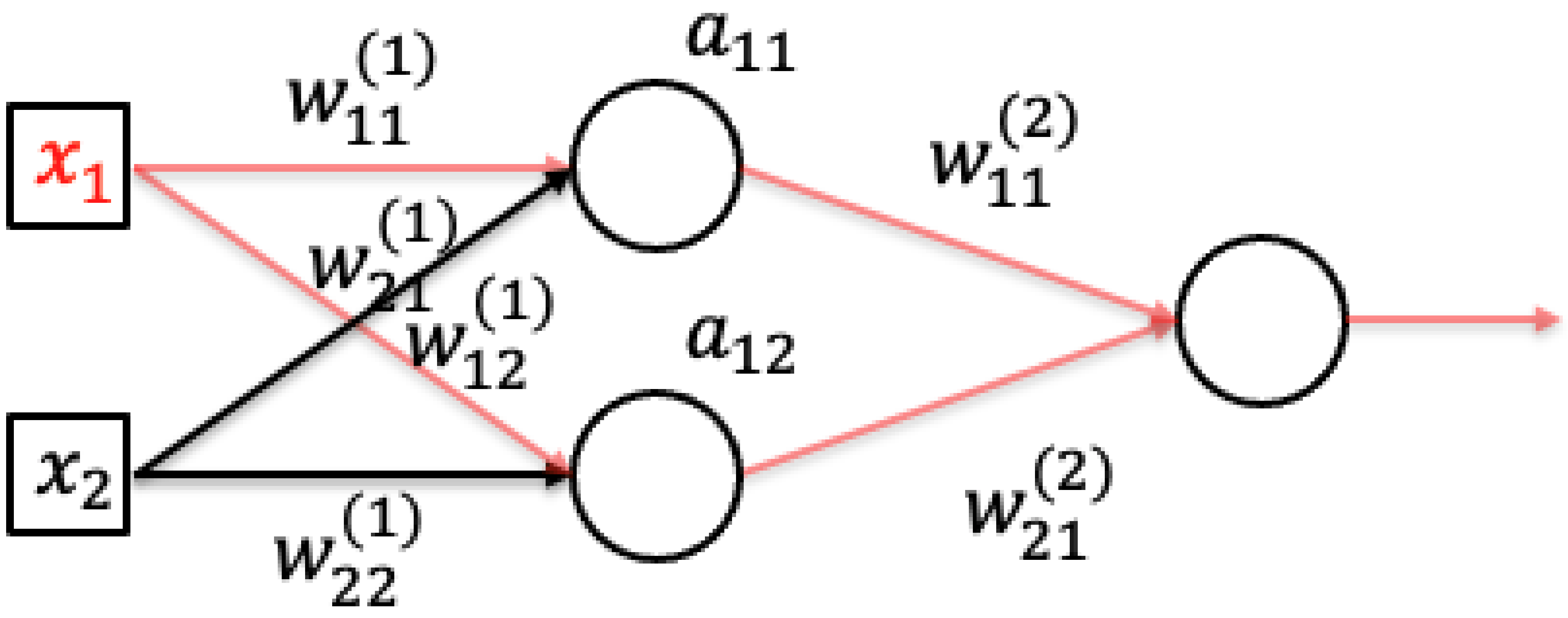
# Calculate Gradient (on one data point)




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

# Calculate Gradient (on one data point)




$w_{11}^{(1)}x_1$, $w_{21}^{(1)}x_2$ $\longrightarrow$ $+$ $\longrightarrow$ $z_{11} \xrightarrow{\sigma(z_{11})} a_{11} \longrightarrow l(x,y)$

$$\frac{\partial a_{11}}{\partial z_{11}} = \sigma'(z_{11}) \qquad \frac{\partial l}{\partial a_{11}} = (\hat{y} - y)w_{11}^{(2)}$$

- By chain rule: $$\frac{\partial l}{\partial x_1} = \frac{\partial l}{\partial a_{11}}\frac{\partial a_{11}}{\partial x_1} + \frac{\partial l}{\partial a_{12}}\frac{\partial a_{12}}{\partial x_1}$$