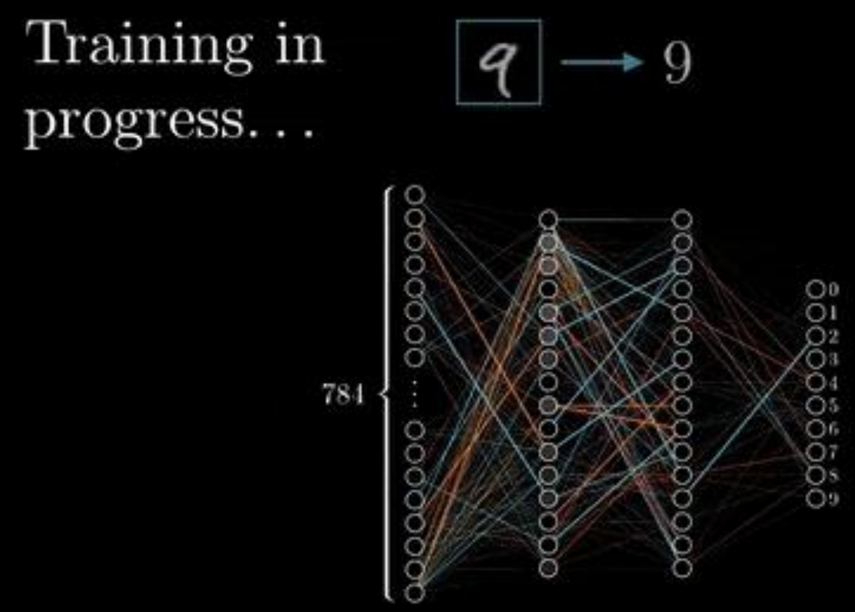
Backpropagation



Backpropagation ⇔ Chain rule

- The neural network loss is a composite function of modules
- We want the gradient w.r.t. to the parameters of the l layer

$$\frac{d\mathcal{L}}{dw_l} = \frac{d\mathcal{L}}{dh_L} \cdot \frac{dh_L}{dh_{L-1}} \cdot \ldots \cdot \frac{dh_l}{dw_l} \quad \Rightarrow \quad \frac{d\mathcal{L}}{dw_l} = \frac{d\mathcal{L}}{dh_l} \cdot \frac{dh_l}{dw_l}$$

Gradient of loss w.r.t. the module output ($\frac{d\mathcal{L}}{dh_l}$) — Gradient of a module w.r.t. its parameters ($\frac{dh_l}{dw_l}$)

- Backpropagation is the "algorithmic manifestation" of the chain rule

Backpropagation ⟺ Chain rule!!!

- Backpropagating gradients means repeating computation of 2 quantities

$$\frac{d\mathcal{L}}{dw_l} = \frac{d\mathcal{L}}{dh_l} \cdot \frac{dh_l}{dw_l}$$

- For $\frac{dh_l}{dw_l}$ just compute the Jacobian of the l-th module w.r.t. to its parameters w_l
- Very local rule → "every module looks for its own"
- Since computations can be very local, this means that
  - graphs can be complex
  - modules can be complex if differentiable

Backpropagation ⟺ Chain rule!!!

- Backpropagating gradients means repeating computation of 2 quantities

$$\frac{d\mathcal{L}}{dw_l} = \frac{d\mathcal{L}}{dh_l} \cdot \frac{dh_l}{dw_l}$$

- For $\frac{d\mathcal{L}}{dh_l}$ we apply chain rule again to recursively reuse computations

$$\frac{d\mathcal{L}}{dh^l} = \frac{d\mathcal{L}}{dh_{l+1}} \cdot \frac{dh_{l+1}}{dh_l}$$

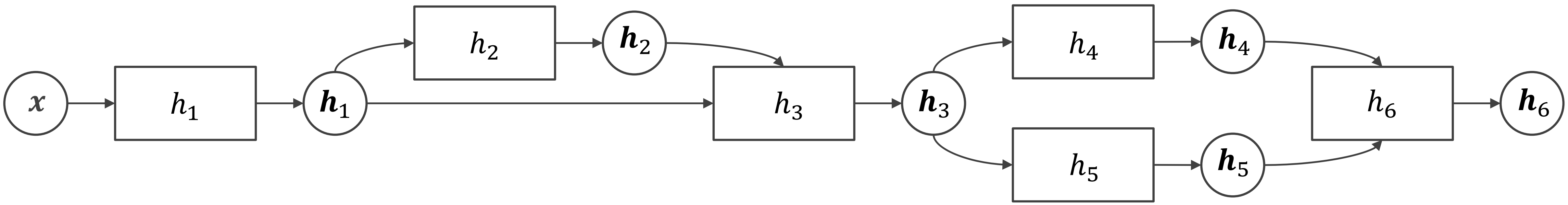
Recursive rule → computation-friendly

Gradient of module w.r.t. its module input

- Remember, the output of a module is the input for the next one: a_l=x_{l+1}

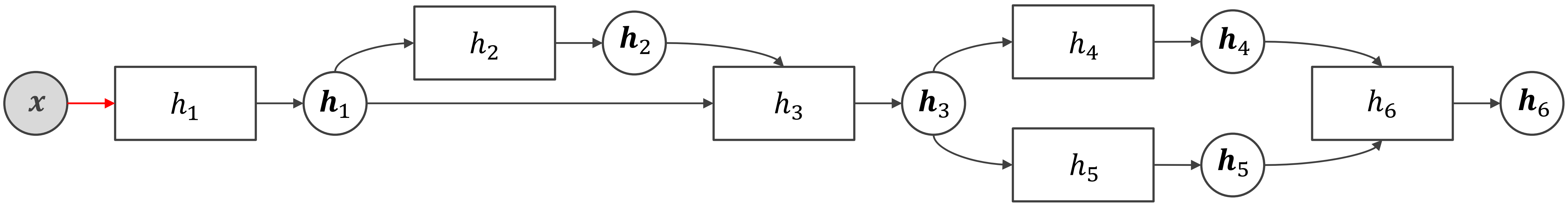
Computational graphs: Forward graph

- Compute the activation of each module in the network $\boldsymbol{h}_l = h_l(\boldsymbol{w}; \boldsymbol{x}_l)$
- Then, set $x_{l+1} := h_l$
- Store intermediate variables h_l
  - will be needed for the backpropagation and saves time at the cost of memory
- Then, repeat recursively and in the right order



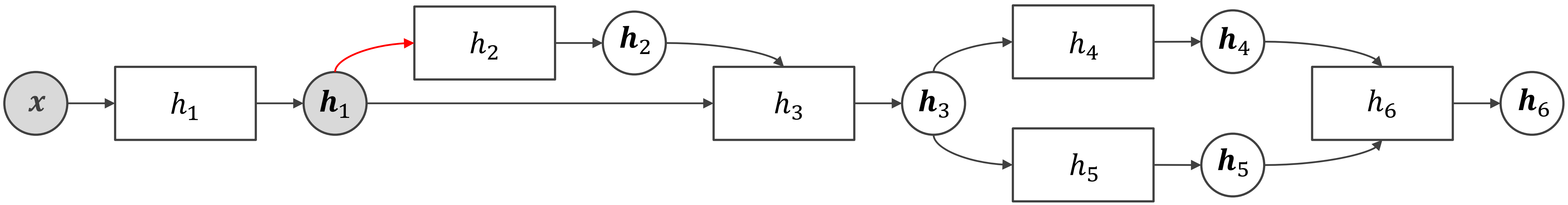
Computational graphs: Forward graph

- Compute the activation of each module in the network $\boldsymbol{h}_l = h_l(\boldsymbol{w}; \boldsymbol{x}_l)$
- Then, set $x_{l+1} := h_l$
- Store intermediate variables h_l
  - will be needed for the backpropagation and saves time at the cost of memory
- Then, repeat recursively and in the right order



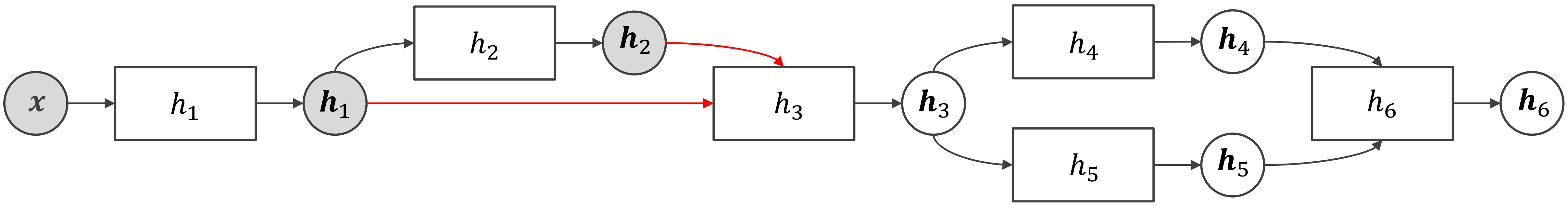
Computational graphs: Forward graph

- Compute the activation of each module in the network $\boldsymbol{h}_l = h_l(\boldsymbol{w}; \boldsymbol{x}_l)$
- Then, set $x_{l+1} := h_l$
- Store intermediate variables h_l
  - will be needed for the backpropagation and saves time at the cost of memory
- Then, repeat recursively and in the right order



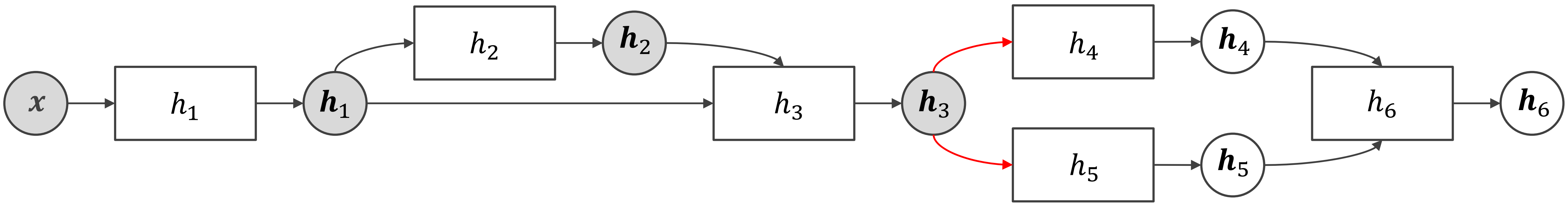
Computational graphs: Forward graph

- Compute the activation of each module in the network $\boldsymbol{h}_l = h_l(\boldsymbol{w}; \boldsymbol{x}_l)$
- Then, set $x_{l+1} := h_l$
- Store intermediate variables h_l
  - will be needed for the backpropagation and saves time at the cost of memory
- Then, repeat recursively and in <u>the right order</u>



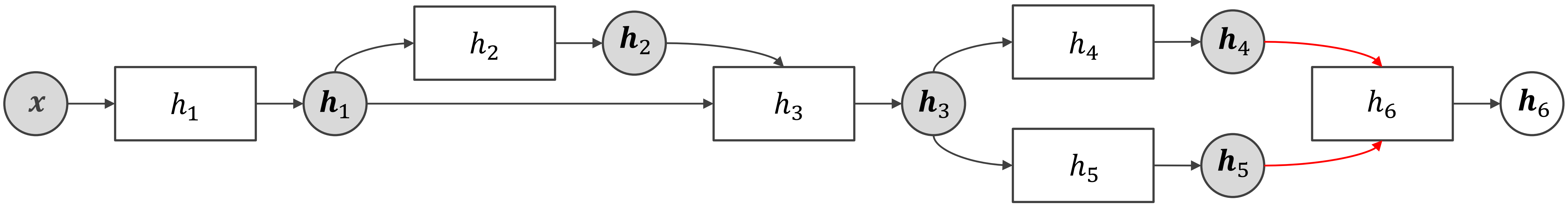
Computational graphs: Forward graph

- Compute the activation of each module in the network $\boldsymbol{h}_l = h_l(\boldsymbol{w}; \boldsymbol{x}_l)$
- Then, set $x_{l+1} := h_l$
- Store intermediate variables h_l
  - will be needed for the backpropagation and saves time at the cost of memory
- Then, repeat recursively and in the right order



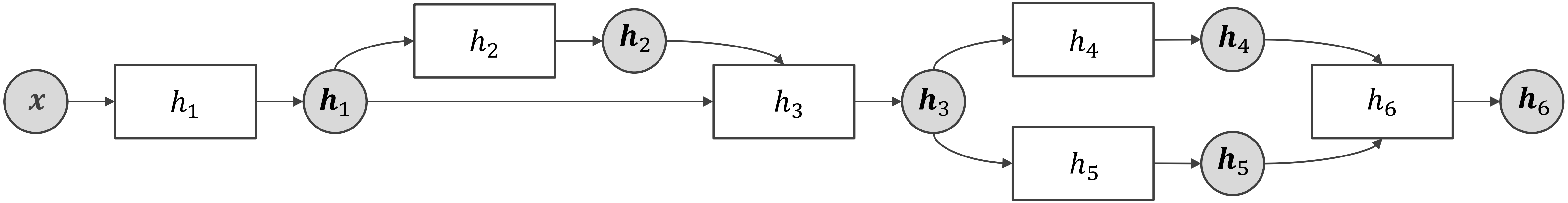
Computational graphs: Forward graph

- Compute the activation of each module in the network $\boldsymbol{h}_l = h_l(\boldsymbol{w}; \boldsymbol{x}_l)$
- Then, set $x_{l+1} := h_l$
- Store intermediate variables h_l
  - will be needed for the backpropagation and saves time at the cost of memory
- Then, repeat recursively and in <u>the right order</u>



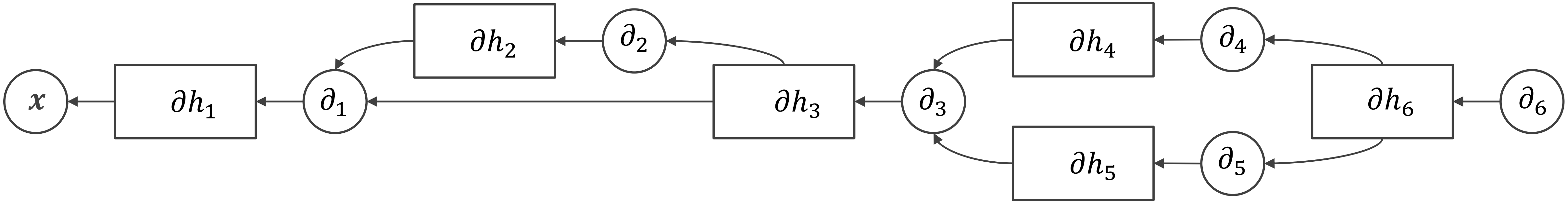
Computational graphs: Forward graph

- Compute the activation of each module in the network $\boldsymbol{h}_l = h_l(\boldsymbol{w}; \boldsymbol{x}_l)$
- Then, set $x_{l+1} := h_l$
- Store intermediate variables h_l
  - will be needed for the backpropagation and saves time at the cost of memory
- Then, repeat recursively and in the right order



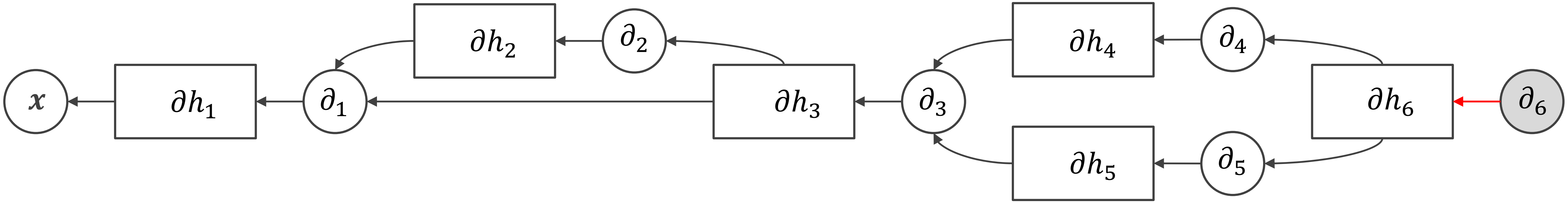
Computational graphs: Reverse graph

- Go backwards and use gradient functions instead of activations
  - Must have the gradient functions $\frac{\partial h_l}{\partial w_l}, \frac{\partial h^l}{\partial h^{l-1}}$ w.r.t. to x_l & w_l implemented
- The gradients will need activations from forward propagation, better save them
  - Sum all gradients from all samples in mini-batch
- Process also known as reverse-mode automatic differentiation
  - Because the flow of computations is reverse to data flow



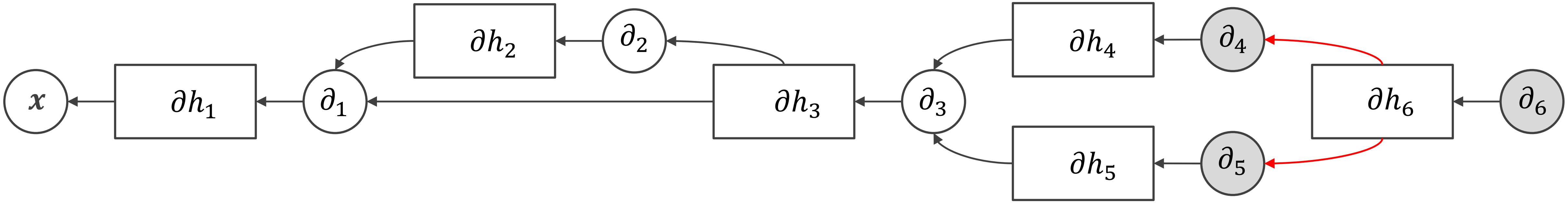
Computational graphs: Reverse graph

- Go backwards and use gradient functions instead of activations
  - Must have the gradient functions $\frac{\partial h_l}{\partial w_l}, \frac{\partial h^l}{\partial h^{l-1}}$ w.r.t. to x_l & w_l implemented
- The gradients will need activations from forward propagation, better save them
  - Sum all gradients from all samples in mini-batch
- Process also known as reverse-mode automatic differentiation
  - Because the flow of computations is reverse to data flow



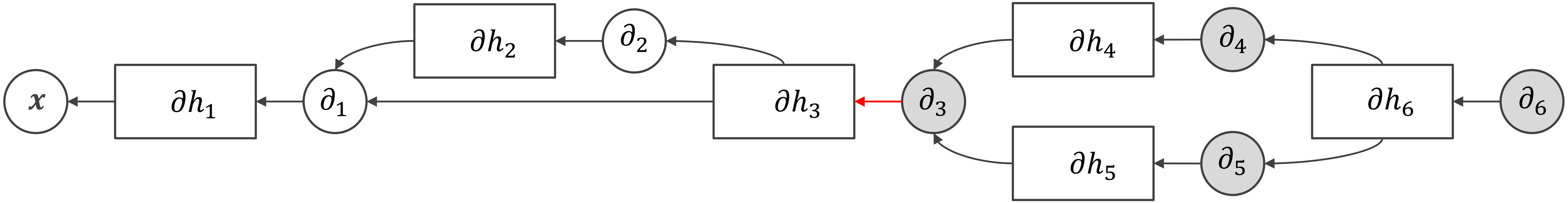
Computational graphs: Reverse graph

- Go backwards and use gradient functions instead of activations
  - Must have the gradient functions $\frac{\partial h_l}{\partial w_l}, \frac{\partial h^l}{\partial h^{l-1}}$ w.r.t. to x_l & w_l implemented
- The gradients will need activations from forward propagation, better save them
  - Sum all gradients from all samples in mini-batch
- Process also known as reverse-mode automatic differentiation
  - Because the flow of computations is reverse to data flow



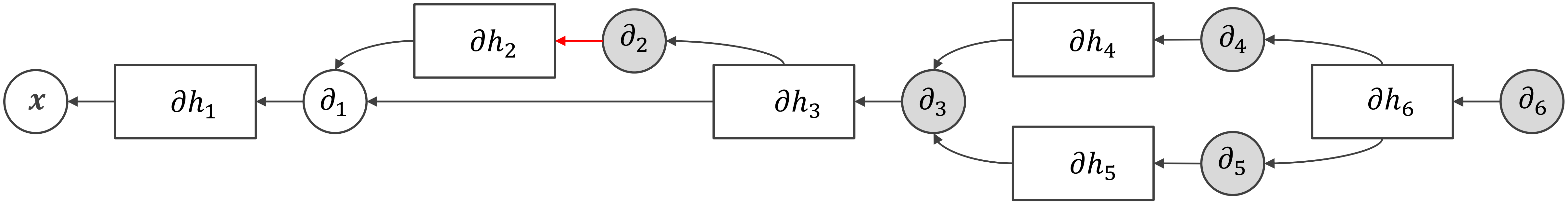
Computational graphs: Reverse graph

- Go backwards and use gradient functions instead of activations
  - Must have the gradient functions $\frac{\partial h_l}{\partial w_l}, \frac{\partial h^l}{\partial h^{l-1}}$ w.r.t. to x_l & w_l implemented
- The gradients will need activations from forward propagation, better save them
  - Sum all gradients from all samples in mini-batch
- Process also known as reverse-mode automatic differentiation
  - Because the flow of computations is reverse to data flow



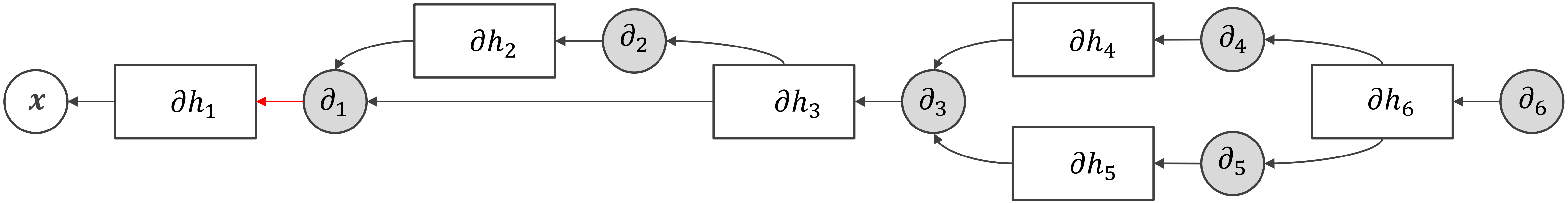
Computational graphs: Reverse graph

- Go backwards and use gradient functions instead of activations
  - Must have the gradient functions $\frac{\partial h_l}{\partial w_l}, \frac{\partial h^l}{\partial h^{l-1}}$ w.r.t. to x_l & w_l implemented
- The gradients will need activations from forward propagation, better save them
  - Sum all gradients from all samples in mini-batch
- Process also known as reverse-mode automatic differentiation
  - Because the flow of computations is reverse to data flow



Computational graphs: Reverse graph

- Go backwards and use gradient functions instead of activations
  - Must have the gradient functions $\frac{\partial h_l}{\partial w_l}, \frac{\partial h_l}{\partial h_{l-1}}$ w.r.t. to x_l & w_l implemented
- The gradients will need activations from forward propagation, better save them
  - Sum all gradients from all samples in mini-batch
- Process also known as reverse-mode automatic differentiation
  - Because the flow of computations is reverse to data flow



Higher-order derivatives

- Computing higher-order derivatives is similar

$$\text{Hessian} = H = \frac{d^2 h}{d\boldsymbol{x}^2} = \begin{bmatrix} \frac{\partial h}{\partial x_1 x_1} & \cdots & \frac{\partial h}{\partial x_1 x_{\mathrm{M}}} \\ \vdots & \ddots & \vdots \\ \frac{\partial h}{\partial x_{\mathrm{M}} x_{\mathrm{M}}} & \cdots & \frac{\partial h}{\partial x_{\mathrm{M}} x_{\mathrm{M}}} \end{bmatrix}$$

- Basically, it is the gradient of the gradient
- Per first-order partial derivative $\frac{\partial h}{\partial x_i}$, auto-differentiate once more
- In practice, computing the second-order gradient is very expensive

Backprogation in summary

- **Step 1.** Compute forward propagations for all layers recursively

$$h_l = h_l(x_l) \text{ and } x_{l+1} = h_l$$

- **Step 2.** Once done with forward propagation, follow the reverse path.
  - Start from the last layer and for each new layer compute the gradients
  - Cache computations when possible to avoid redundant operations

$$\frac{d\mathcal{L}}{dw_l} = \frac{d\mathcal{L}}{dh_l} \cdot \frac{dh_l}{dw_l} \qquad \frac{d\mathcal{L}}{dh_l} = \frac{d\mathcal{L}}{dh_{l+1}} \cdot \frac{dh_{l+1}}{dh_l}$$

- **Step 3.** Use the gradients $\frac{d\mathcal{L}}{dw^l}$ with Stochastic Gradient Descend to train

Backpropagation visualization

Forward propagation

$h_0 = x$

$h_1 = \sigma(w_1 h_0)$ → Store h_1 . Remember that $\partial_x \sigma = \sigma \cdot (1 - \sigma)$

$h_2 = \sigma(w_2 h_1)$ → Store h_2

$\mathcal{L} = 0.5 \cdot \|l - h_2\|^2$

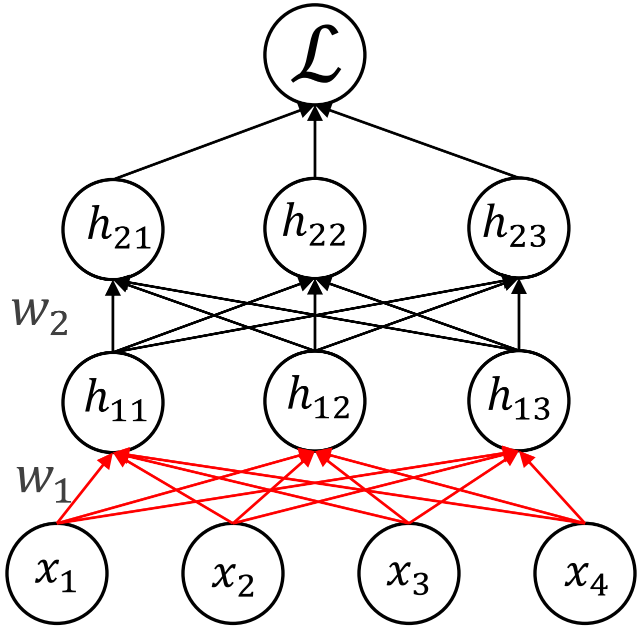
Backward propagation

$$\frac{d\mathcal{L}}{dh_2} = -(y^* - h_2)$$

$$\frac{d\mathcal{L}}{dw_2} = \frac{d\mathcal{L}}{dh_2}\frac{dh_2}{dw_2} = \frac{d\mathcal{L}}{dh_2} h_1 \sigma(w_2 h_1)\big(1 - \sigma(w_2 h_1)\big) = \frac{d\mathcal{L}}{da_2} h_1 h_2 (1 - h_2)$$

$$\frac{d\mathcal{L}}{dh_1} = \frac{d\mathcal{L}}{dh_2}\frac{dh_2}{dh_1} = \frac{d\mathcal{L}}{dh_2} w_2 \sigma(w_2 h_1)\big(1 - \sigma(w_2 h_1)\big) = \frac{d\mathcal{L}}{dh_2} w_2 h_2 (1 - h_2)$$

$$\frac{d\mathcal{L}}{dw_1} = \frac{d\mathcal{L}}{dh_1}\frac{dh_1}{dw_1} = \frac{d\mathcal{L}}{dh_1} h_0 \sigma(w_1 h_0)\big(1 - \sigma(w_1 h_0)\big) = \frac{d\mathcal{L}}{dh_1} h_0 h_1 (1 - h_1)$$



Backpropagation visualization

Forward propagation

$h_0 = x$

$h_1 = \sigma(w_1 h_0)$ → Store h_1 . Remember that $\partial_x \sigma = \sigma \cdot (1 - \sigma)$

$h_2 = \sigma(w_2 h_1)$ → Store h_2

$\mathcal{L} = 0.5 \cdot \|l - h_2\|^2$

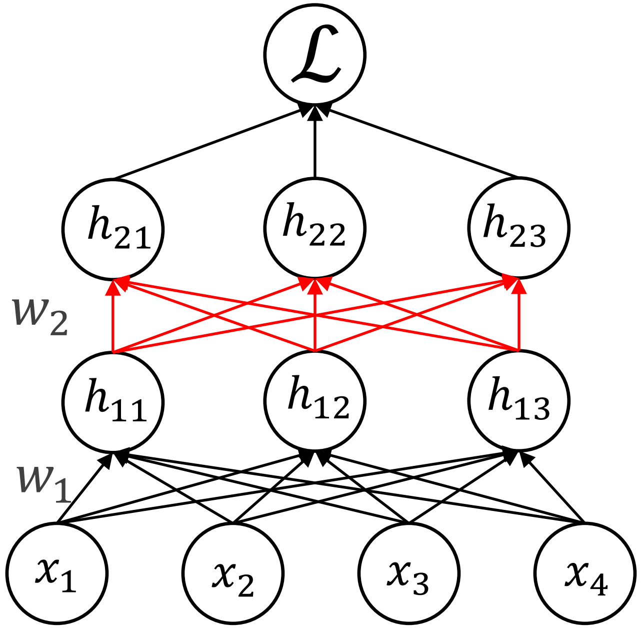
Backward propagation

$$\frac{d\mathcal{L}}{dh_2} = -(y^* - h_2)$$

$$\frac{d\mathcal{L}}{dw_2} = \frac{d\mathcal{L}}{dh_2}\frac{dh_2}{dw_2} = \frac{d\mathcal{L}}{dh_2} h_1 \sigma(w_2 h_1)\big(1 - \sigma(w_2 h_1)\big) = \frac{d\mathcal{L}}{da_2} h_1 h_2 (1 - h_2)$$

$$\frac{d\mathcal{L}}{dh_1} = \frac{d\mathcal{L}}{dh_2}\frac{dh_2}{dh_1} = \frac{d\mathcal{L}}{dh_2} w_2 \sigma(w_2 h_1)\big(1 - \sigma(w_2 h_1)\big) = \frac{d\mathcal{L}}{dh_2} w_2 h_2 (1 - h_2)$$

$$\frac{d\mathcal{L}}{dw_1} = \frac{d\mathcal{L}}{dh_1}\frac{dh_1}{dw_1} = \frac{d\mathcal{L}}{dh_1} h_0 \sigma(w_1 h_0)\big(1 - \sigma(w_1 h_0)\big) = \frac{d\mathcal{L}}{dh_1} h_0 h_1 (1 - h_1)$$



Backpropagation visualization

Forward propagation

$h_0 = x$

$h_1 = \sigma(w_1 h_0)$ → Store h_1 . Remember that $\partial_x \sigma = \sigma \cdot (1 - \sigma)$

$h_2 = \sigma(w_2 h_1)$ → Store h_2

→ $\mathcal{L} = 0.5 \cdot \|l - h_2\|^2$

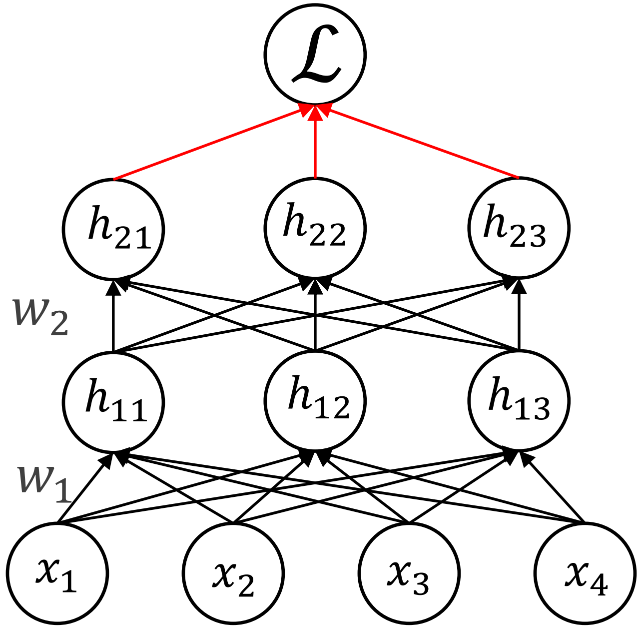
Backward propagation

$$\frac{d\mathcal{L}}{dh_2} = -(y^* - h_2)$$

$$\frac{d\mathcal{L}}{dw_2} = \frac{d\mathcal{L}}{dh_2}\frac{dh_2}{dw_2} = \frac{d\mathcal{L}}{dh_2} h_1 \sigma(w_2 h_1)\big(1 - \sigma(w_2 h_1)\big) = \frac{d\mathcal{L}}{da_2} h_1 h_2 (1 - h_2)$$

$$\frac{d\mathcal{L}}{dh_1} = \frac{d\mathcal{L}}{dh_2}\frac{dh_2}{dh_1} = \frac{d\mathcal{L}}{dh_2} w_2 \sigma(w_2 h_1)\big(1 - \sigma(w_2 h_1)\big) = \frac{d\mathcal{L}}{dh_2} w_2 h_2 (1 - h_2)$$

$$\frac{d\mathcal{L}}{dw_1} = \frac{d\mathcal{L}}{dh_1}\frac{dh_1}{dw_1} = \frac{d\mathcal{L}}{dh_1} h_0 \sigma(w_1 h_0)\big(1 - \sigma(w_1 h_0)\big) = \frac{d\mathcal{L}}{dh_1} h_0 h_1 (1 - h_1)$$



Backpropagation visualization

Forward propagation

$h_0 = x$

$h_1 = \sigma(w_1 h_0)$ → Store h_1 . Remember that $\partial_x \sigma = \sigma \cdot (1 - \sigma)$

$h_2 = \sigma(w_2 h_1)$ → Store h_2

$\mathcal{L} = 0.5 \cdot \|l - h_2\|^2$

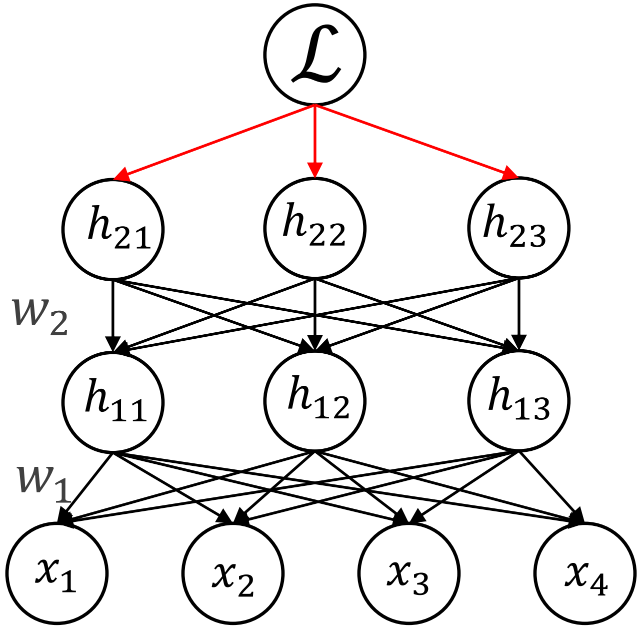
Backward propagation

$$\frac{d\mathcal{L}}{dh_2} = -(y^* - h_2)$$

$$\frac{d\mathcal{L}}{dw_2} = \frac{d\mathcal{L}}{dh_2}\frac{dh_2}{dw_2} = \frac{d\mathcal{L}}{dh_2} h_1 \sigma(w_2 h_1)\big(1 - \sigma(w_2 h_1)\big) = \frac{d\mathcal{L}}{da_2} h_1 h_2 (1 - h_2)$$

$$\frac{d\mathcal{L}}{dh_1} = \frac{d\mathcal{L}}{dh_2}\frac{dh_2}{dh_1} = \frac{d\mathcal{L}}{dh_2} w_2 \sigma(w_2 h_1)\big(1 - \sigma(w_2 h_1)\big) = \frac{d\mathcal{L}}{dh_2} w_2 h_2 (1 - h_2)$$

$$\frac{d\mathcal{L}}{dw_1} = \frac{d\mathcal{L}}{dh_1}\frac{dh_1}{dw_1} = \frac{d\mathcal{L}}{dh_1} h_0 \sigma(w_1 h_0)\big(1 - \sigma(w_1 h_0)\big) = \frac{d\mathcal{L}}{dh_1} h_0 h_1 (1 - h_1)$$



Backpropagation visualization

Forward propagation

$h_0 = x$

$h_1 = \sigma(w_1 h_0)$ $\rightarrow$ Store h_1 . Remember that $\partial_x \sigma = \sigma \cdot (1 - \sigma)$

$h_2 = \sigma(w_2 h_1)$ $\rightarrow$ Store h_2

$\mathcal{L} = 0.5 \cdot \|l - h_2\|^2$

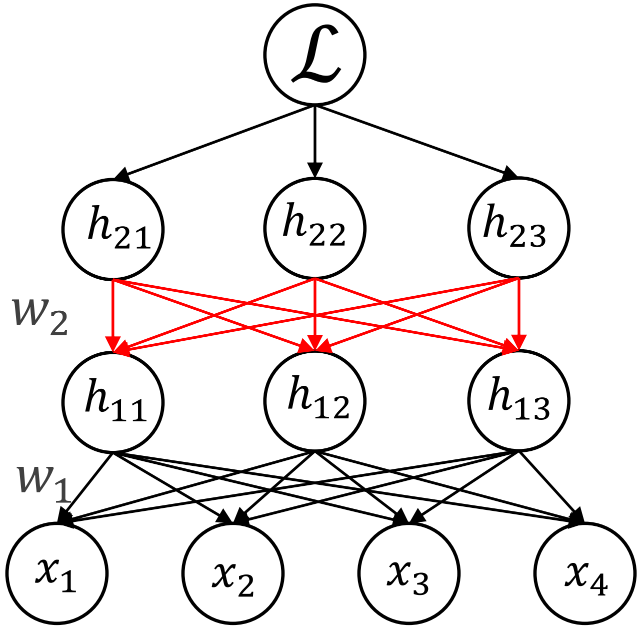
Backward propagation

$$\frac{d\mathcal{L}}{dh_2} = -(y^* - h_2)$$

$$\frac{d\mathcal{L}}{dw_2} = \frac{d\mathcal{L}}{dh_2}\frac{dh_2}{dw_2} = \frac{d\mathcal{L}}{dh_2} h_1 \sigma(w_2 h_1)\big(1 - \sigma(w_2 h_1)\big) = \frac{d\mathcal{L}}{da_2} h_1 h_2 (1 - h_2)$$

$$\frac{d\mathcal{L}}{dh_1} = \frac{d\mathcal{L}}{dh_2}\frac{dh_2}{dh_1} = \frac{d\mathcal{L}}{dh_2} w_2 \sigma(w_2 h_1)\big(1 - \sigma(w_2 h_1)\big) = \frac{d\mathcal{L}}{dh_2} w_2 h_2 (1 - h_2)$$

$$\frac{d\mathcal{L}}{dw_1} = \frac{d\mathcal{L}}{dh_1}\frac{dh_1}{dw_1} = \frac{d\mathcal{L}}{dh_1} h_0 \sigma(w_1 h_0)\big(1 - \sigma(w_1 h_0)\big) = \frac{d\mathcal{L}}{dh_1} h_0 h_1 (1 - h_1)$$



Backpropagation visualization

Forward propagation

$h_0 = x$

$h_1 = \sigma(w_1 h_0)$ $\rightarrow$ Store h_1 . Remember that $\partial_x \sigma = \sigma \cdot (1 - \sigma)$

$h_2 = \sigma(w_2 h_1)$ $\rightarrow$ Store h_2

$\mathcal{L} = 0.5 \cdot \|l - h_2\|^2$

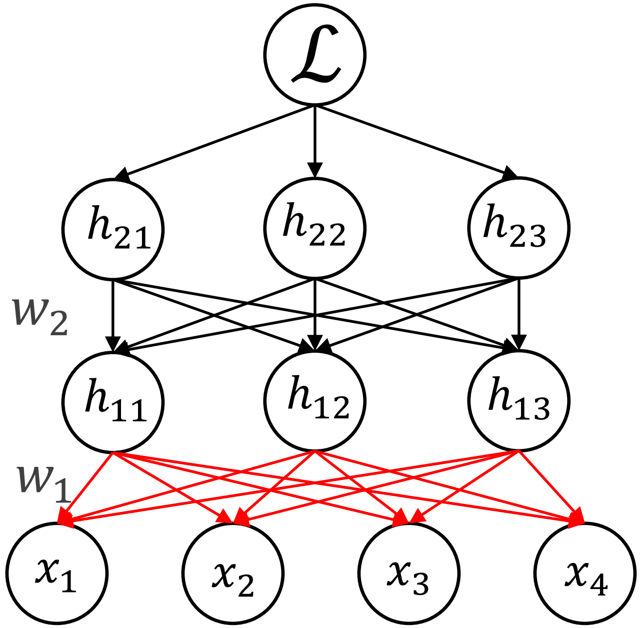
Backward propagation

$$\frac{d\mathcal{L}}{dh_2} = -(y^* - h_2)$$

$$\frac{d\mathcal{L}}{dw_2} = \frac{d\mathcal{L}}{dh_2}\frac{dh_2}{dw_2} = \frac{d\mathcal{L}}{dh_2} h_1 \sigma(w_2 h_1)\big(1 - \sigma(w_2 h_1)\big) = \frac{d\mathcal{L}}{da_2} h_1 h_2 (1 - h_2)$$

$$\frac{d\mathcal{L}}{dh_1} = \frac{d\mathcal{L}}{dh_2}\frac{dh_2}{dh_1} = \frac{d\mathcal{L}}{dh_2} w_2 \sigma(w_2 h_1)\big(1 - \sigma(w_2 h_1)\big) = \frac{d\mathcal{L}}{dh_2} w_2 h_2 (1 - h_2)$$

$$\frac{d\mathcal{L}}{dw_1} = \frac{d\mathcal{L}}{dh_1}\frac{dh_1}{dw_1} = \frac{d\mathcal{L}}{dh_1} h_0 \sigma(w_1 h_0)\big(1 - \sigma(w_1 h_0)\big) = \frac{d\mathcal{L}}{dh_1} h_0 h_1 (1 - h_1)$$



What’s the big deal?

- Backpropagation is as simple as it is complicated
- Mathematically, just the chain rule
- That simple, that we can even automate it (“reverse-mode differentiation”)

- However, algorithmically the devil is in the details to make it efficient
- And, theoretically, why does it even work given the strong non-convexity?

Summary

- Modularity in Neural Networks
- Neural Network Modules
- Neural Network Cheatsheet
- Backpropagation

Reading material

- Chapter 6
- Efficient Backprop, LeCun et al., 1998