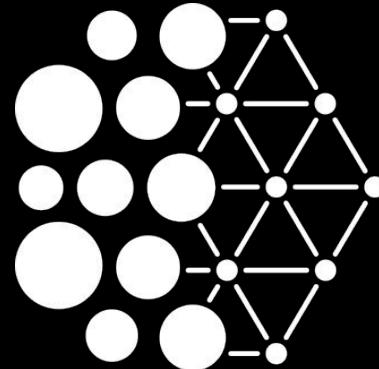


Quebec
Artificial
Intelligence
Institute



Mila

Computational graph & backpropagation

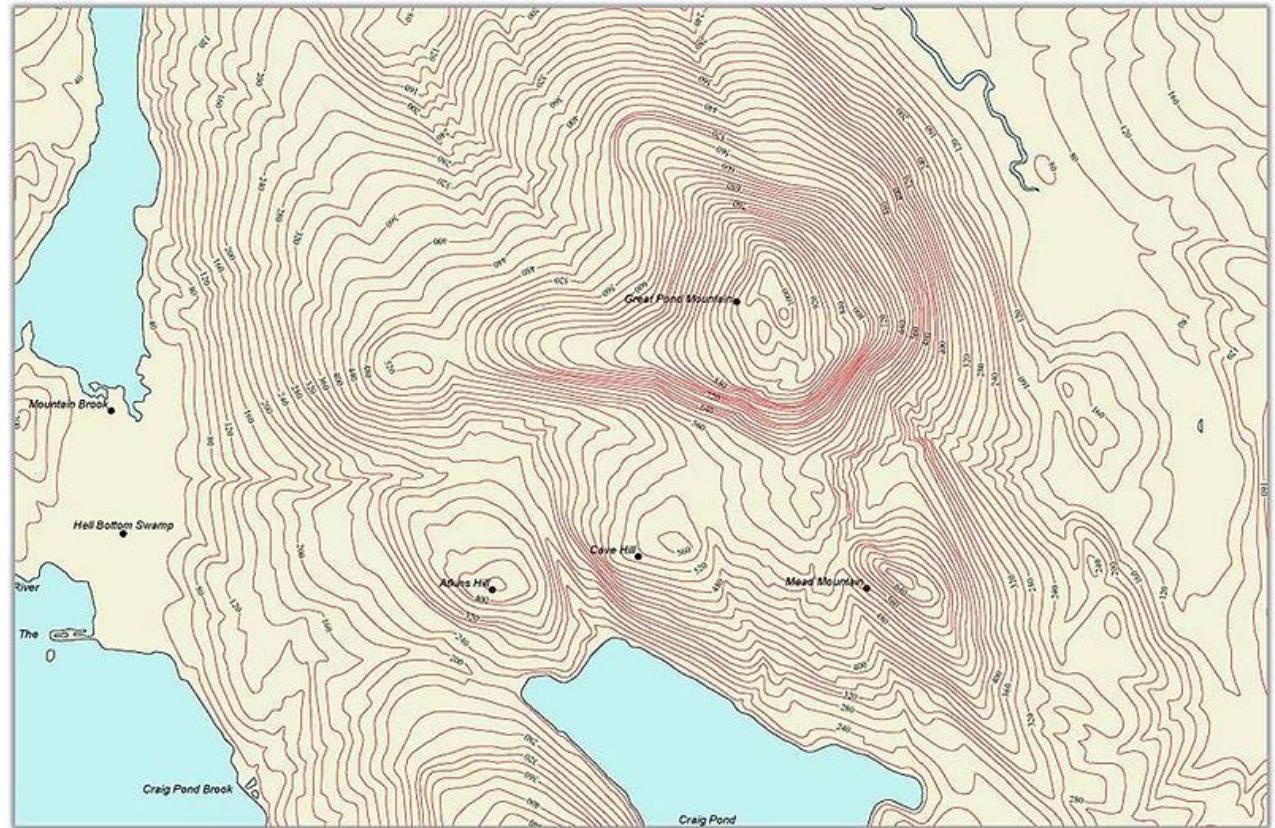
Gaétan Marceau Caron
Applied research scientist, Mila
gaetan.marceau.caron@mila.quebec

Ingredients for supervised deep learning

- A task definition
- An evaluation metric
- A large amount of high-quality labeled data (>100 000)
- A learning algorithm:
 - **End-to-end differentiable computational graph**
 - **A gradient calculation algorithm: backpropagation**
 - A parameter optimizer

Loss function

- Give feedback on the model response for every single example (iid).
- We average the loss over all examples.
- The averaged loss defines contour lines in the parameter space.

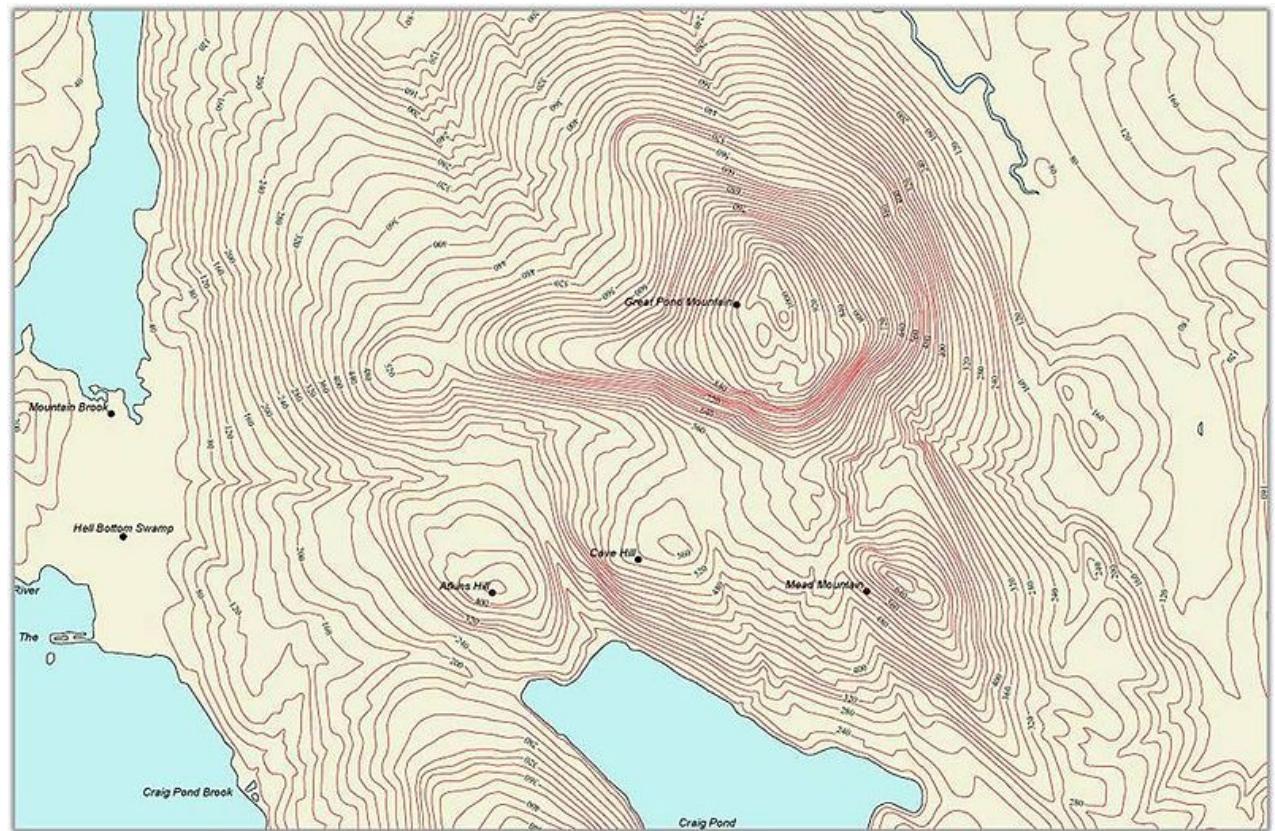


Source: Wikimedia commons

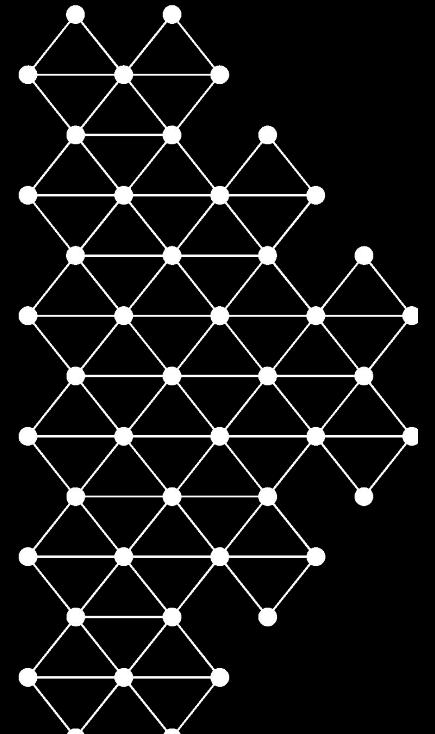
A walk in a million dimensions

- Gradient: backpropagation
- Displacement: optimizer

Optimization	Mountaineering
Parameter	Coordinate
Loss	Altitude
Gradient	Slope
Update	Displacement

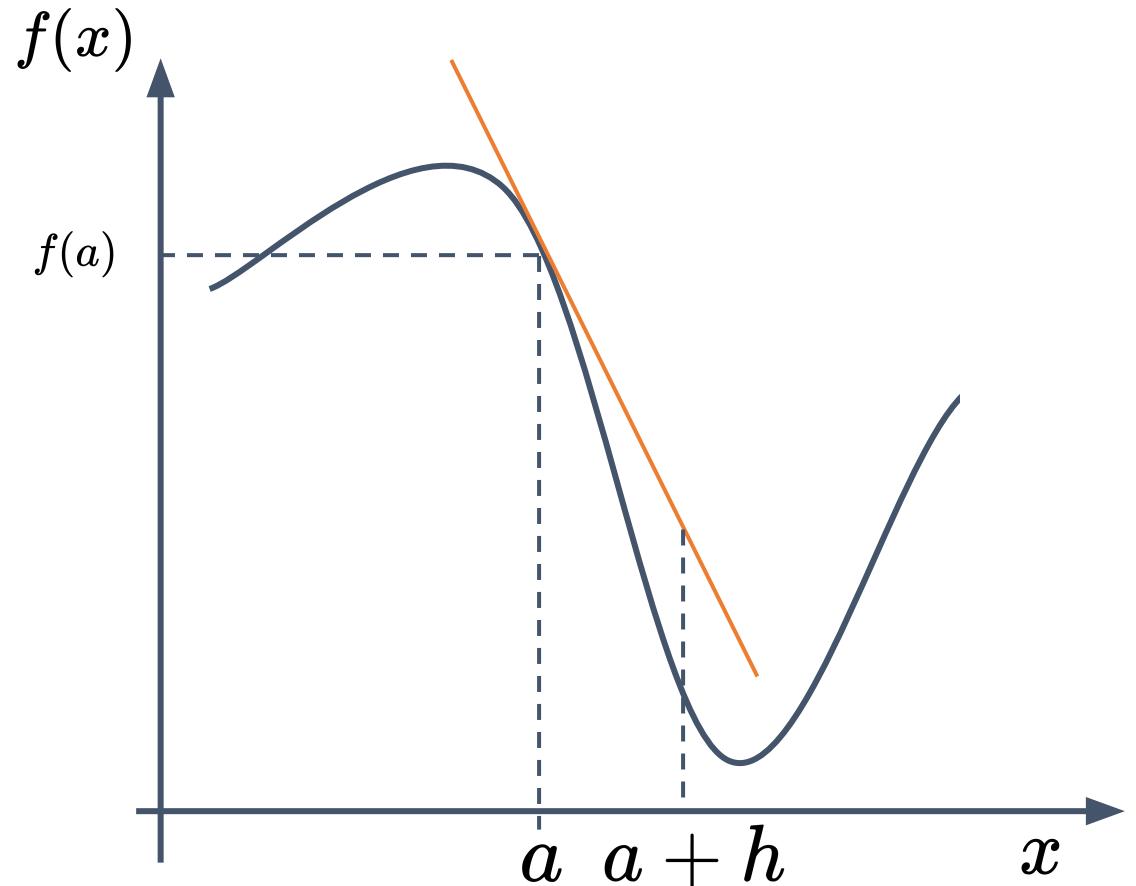


Source: Wikimedia commons



Review of calculus

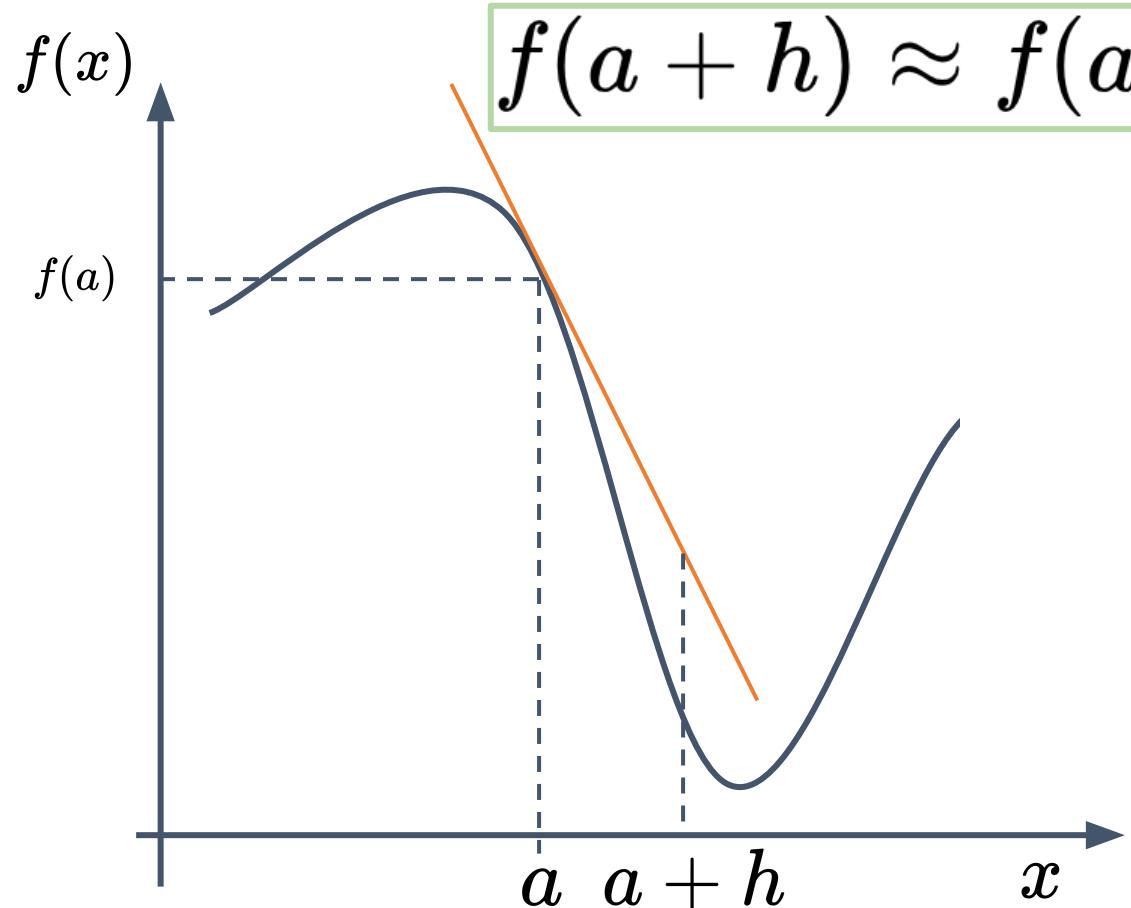
Function approximation



$$f : \mathbb{R} \rightarrow \mathbb{R}$$

$$f'(x) = \lim_{h \rightarrow 0} \frac{f(x+h) - f(x)}{h}$$

Function approximation



$$f(a + h) \approx f(a) + \langle f'(a), h \rangle$$

$$f : \mathbb{R} \rightarrow \mathbb{R}$$

$$f'(x) = \lim_{h \rightarrow 0} \frac{f(x+h) - f(x)}{h}$$

Multivariate function

n inputs - 1 output $f : \mathbb{R}^n \rightarrow \mathbb{R}$

Partial derivative

$$\partial_i f(x) = \lim_{h \rightarrow 0} \frac{f(x + h e_i) - f(x)}{h}$$

Gradient

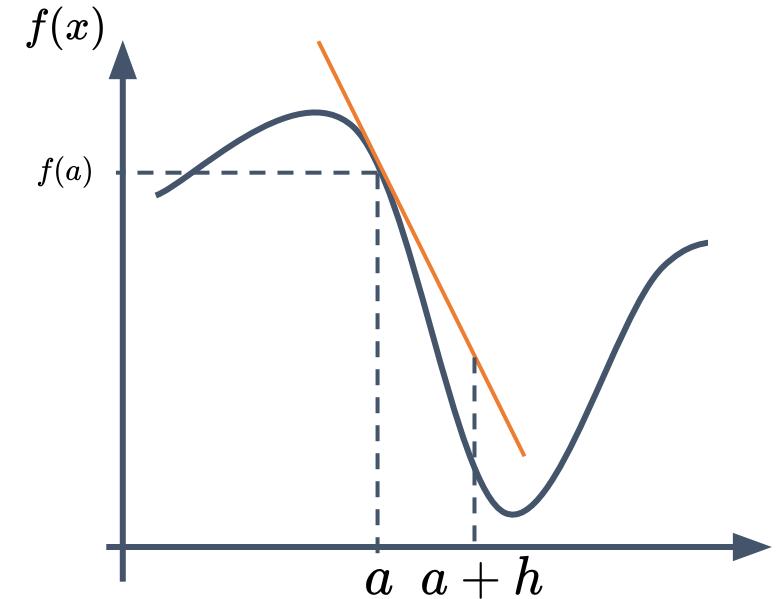
$$\nabla f(x) := f'(x) := \begin{bmatrix} \dots \\ \partial_i f(x) \\ \dots \end{bmatrix}$$

Differentiable function at x

If $\lim_{\|h\| \rightarrow 0} \frac{f(x+h) - f(x) - \langle f'(x), h \rangle}{\|h\|} = 0$

Then $f(x + h) \approx f(x) + \langle f'(x), h \rangle$

If we don't deviate too far from x.



Examples of derivatives

Linear

$$g(x) = Wx + b$$



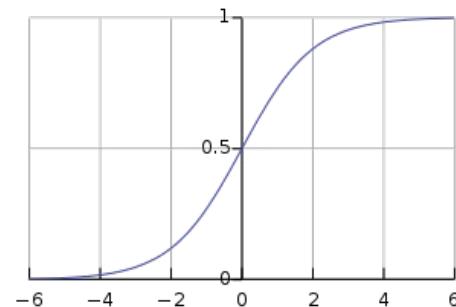
$$g'(x) = W$$

Sigmoid

$$g(x) = \frac{1}{1+e^{-x}}$$



$$g'(x) = g(x)(1 - g(x))$$

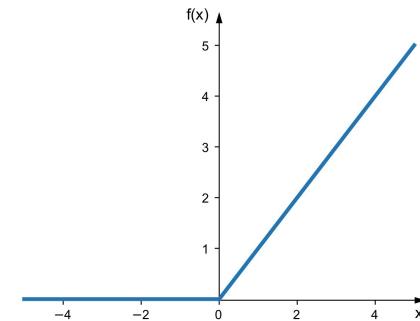


ReLU

$$g(x) = \max(x, 0)$$



$$g'(x) = \begin{cases} 1 & \text{if } x > 0 \\ 0 & \text{otherwise} \end{cases}$$



Calculus 101

Composition of functions

$$g \circ f(x) = g(f(x))$$



$$(g \circ f)'(x) = g'(f(x))f'(x)$$

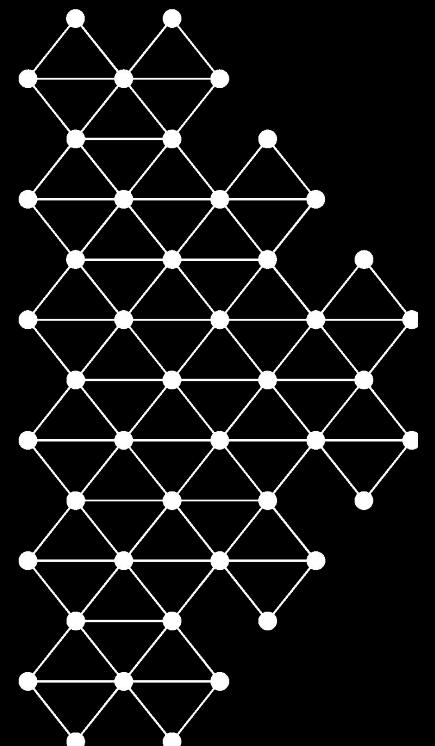
Total derivative (example)

$$\frac{dg(x(t), y(t))}{dt} = \partial_1 g(x(t), y(t))x'(t) + \partial_2 g(x(t), y(t))y'(t)$$

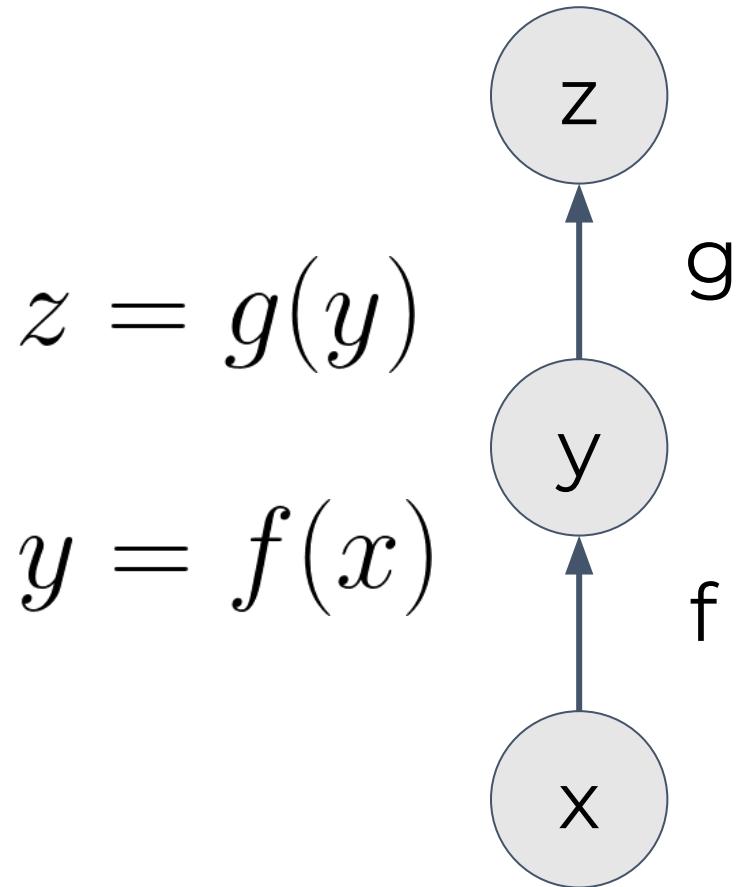
$$\frac{dg}{dt} = \frac{dg}{dx} \frac{dx}{dt} + \frac{dg}{dy} \frac{dy}{dt}$$

$$\frac{dg(x_1(t), \dots, x_n(t))}{dt} = \sum_{i=1}^n \partial_i g(x_1(t), \dots, x_n(t))x'_i(t)$$

Computational Graph

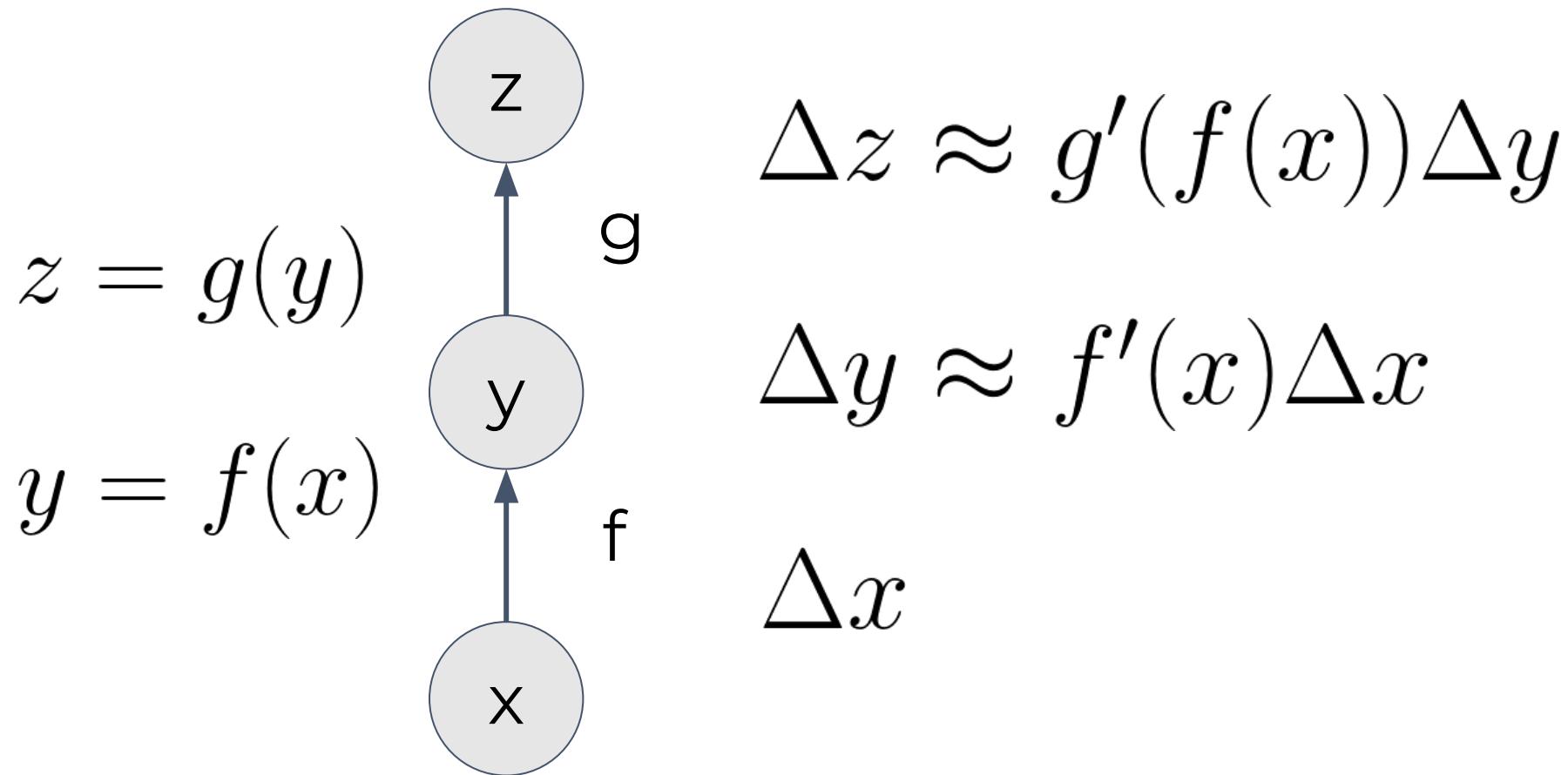


Computational graph: Ex. 1

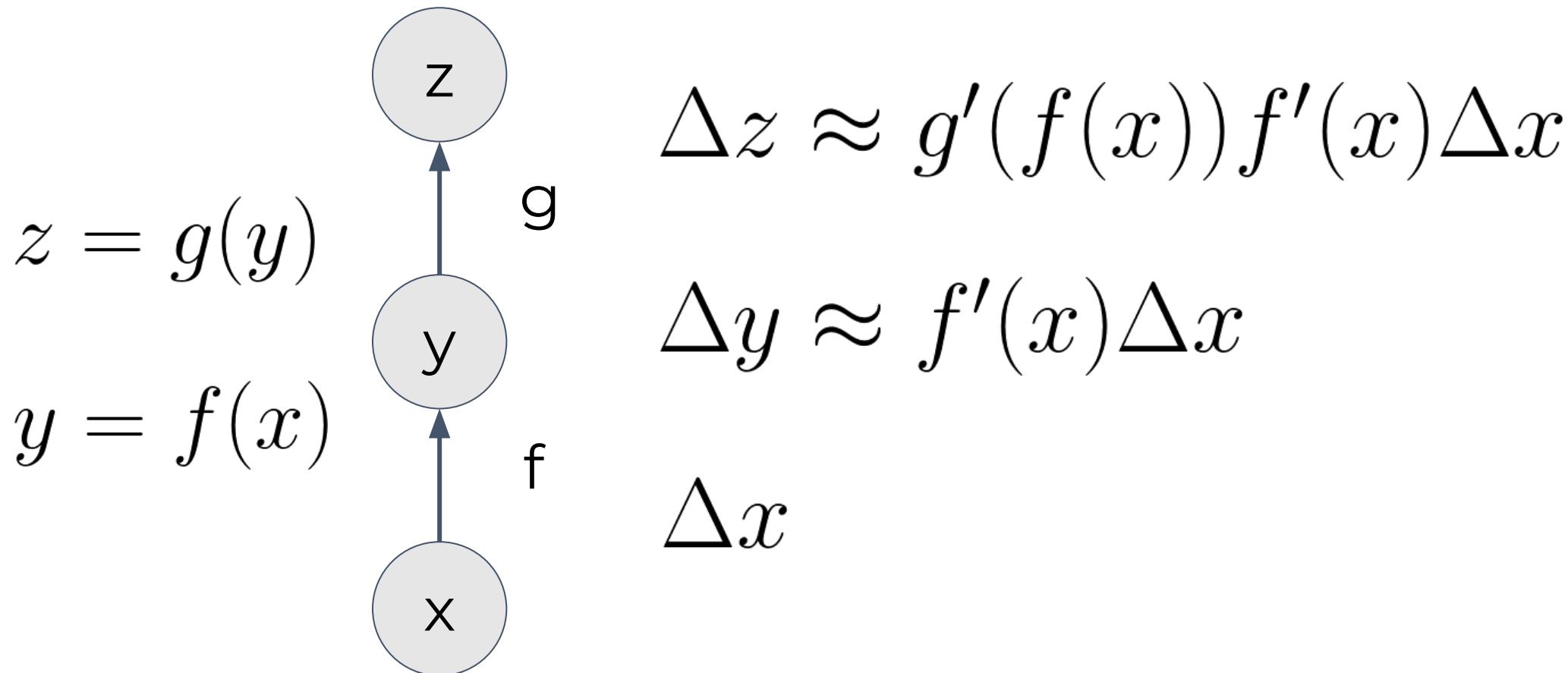


```
def g(y):  
    ... let's do something  
    return z  
  
def f(x):  
    ... let's do something else  
    return y  
  
x = 3.1415  
z = g(f(x))
```

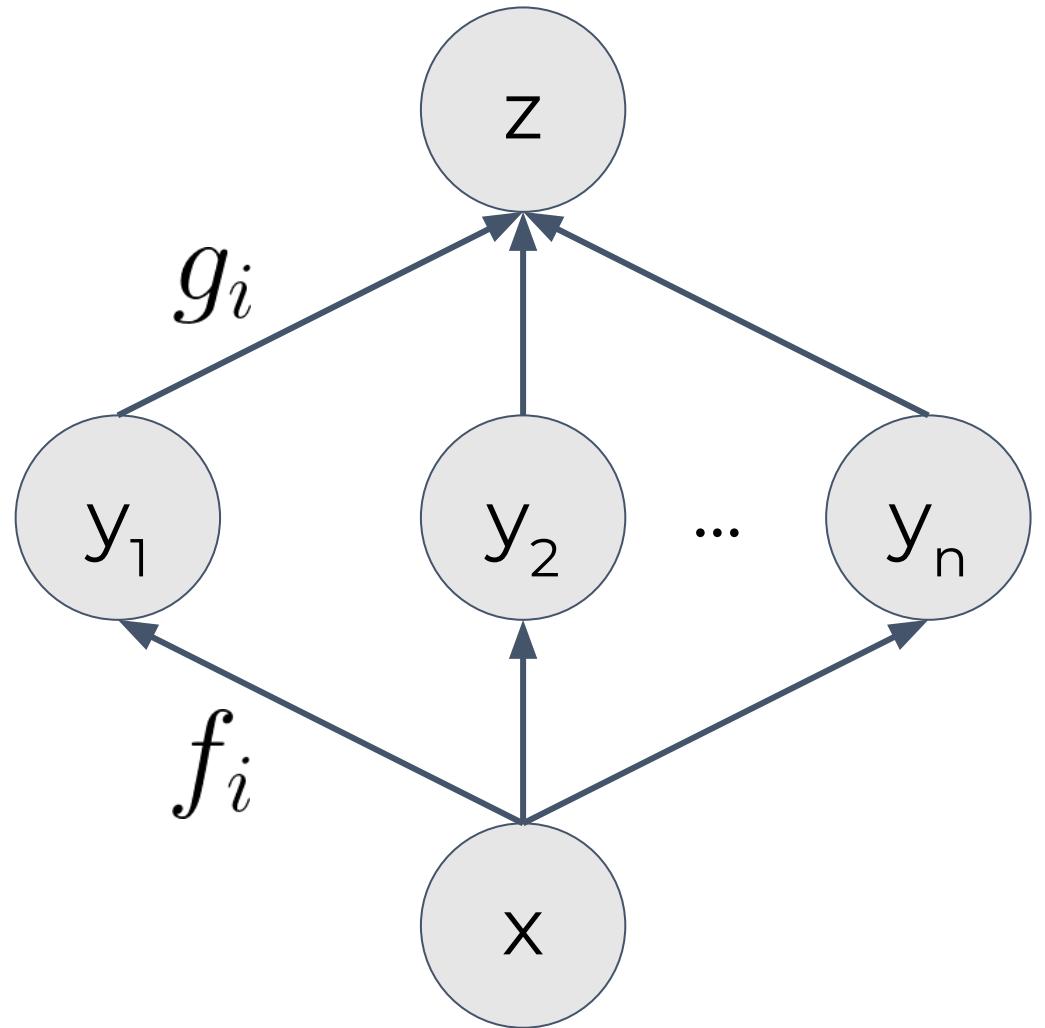
Computational graph: Ex. 1



Computational graph: Ex. 1



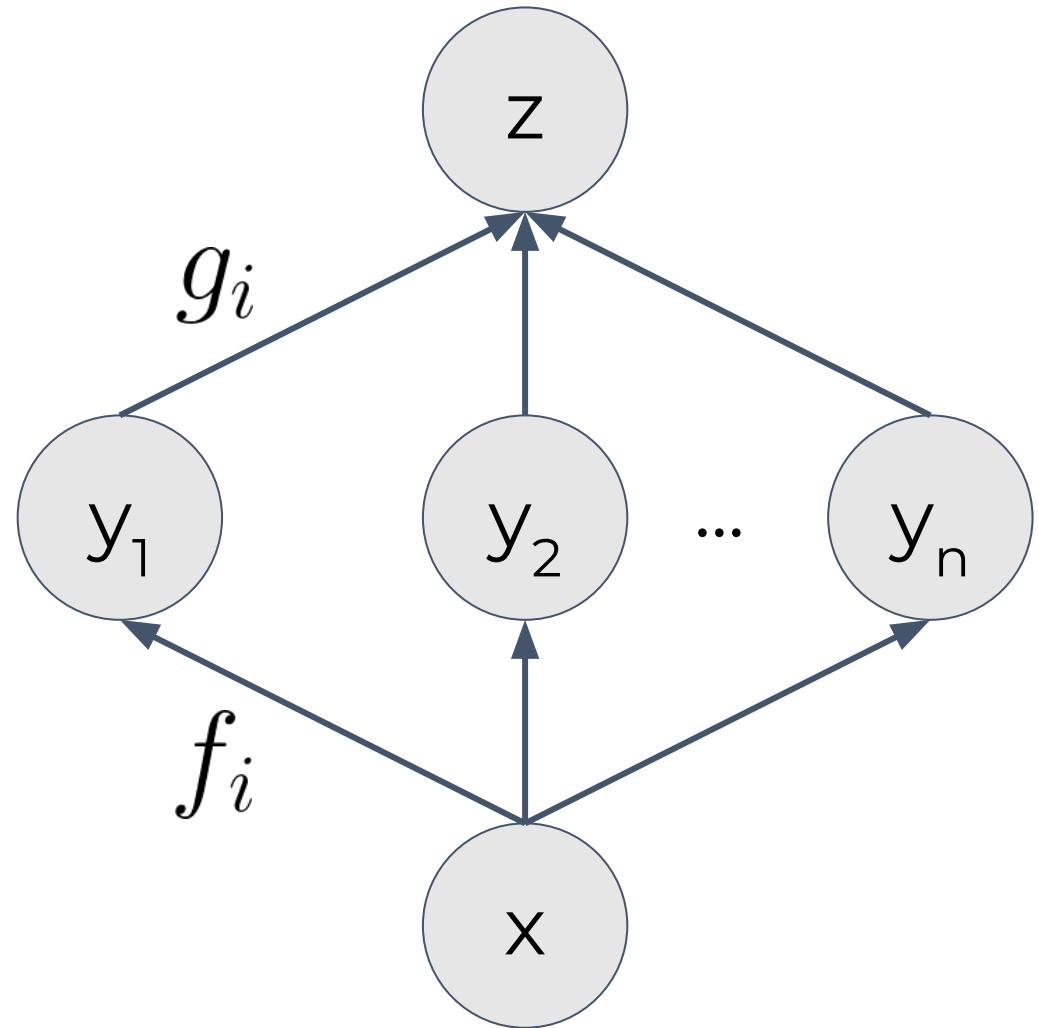
Computational graph: Ex. 2



$$z = \sum_i g_i(y_i)$$

$$y_i = f_i(x)$$

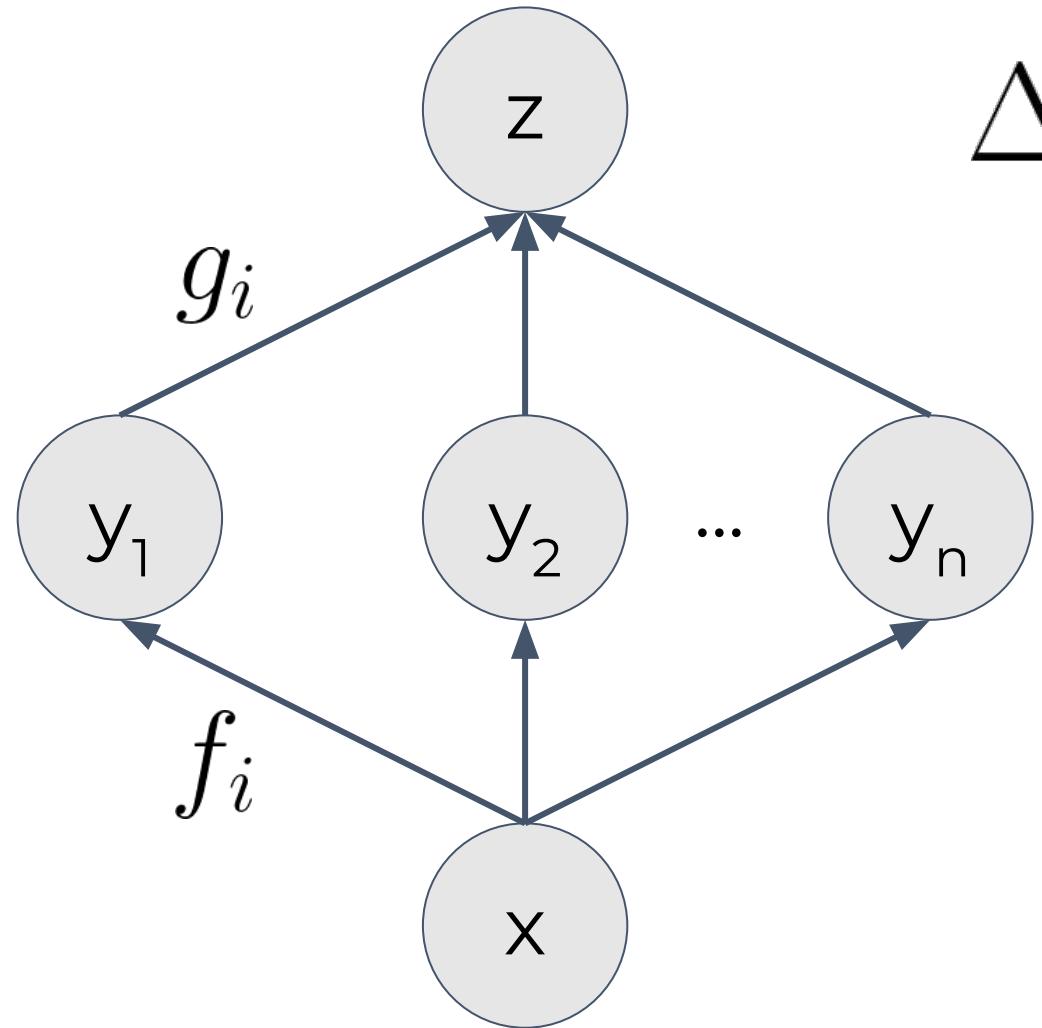
Computational graph: Ex. 2



$$\Delta z \approx \sum_i g'_i(f_i(x)) \Delta y_i$$

$$\Delta y_i \approx f'_i(x) \Delta x$$

Computational graph: Ex. 2

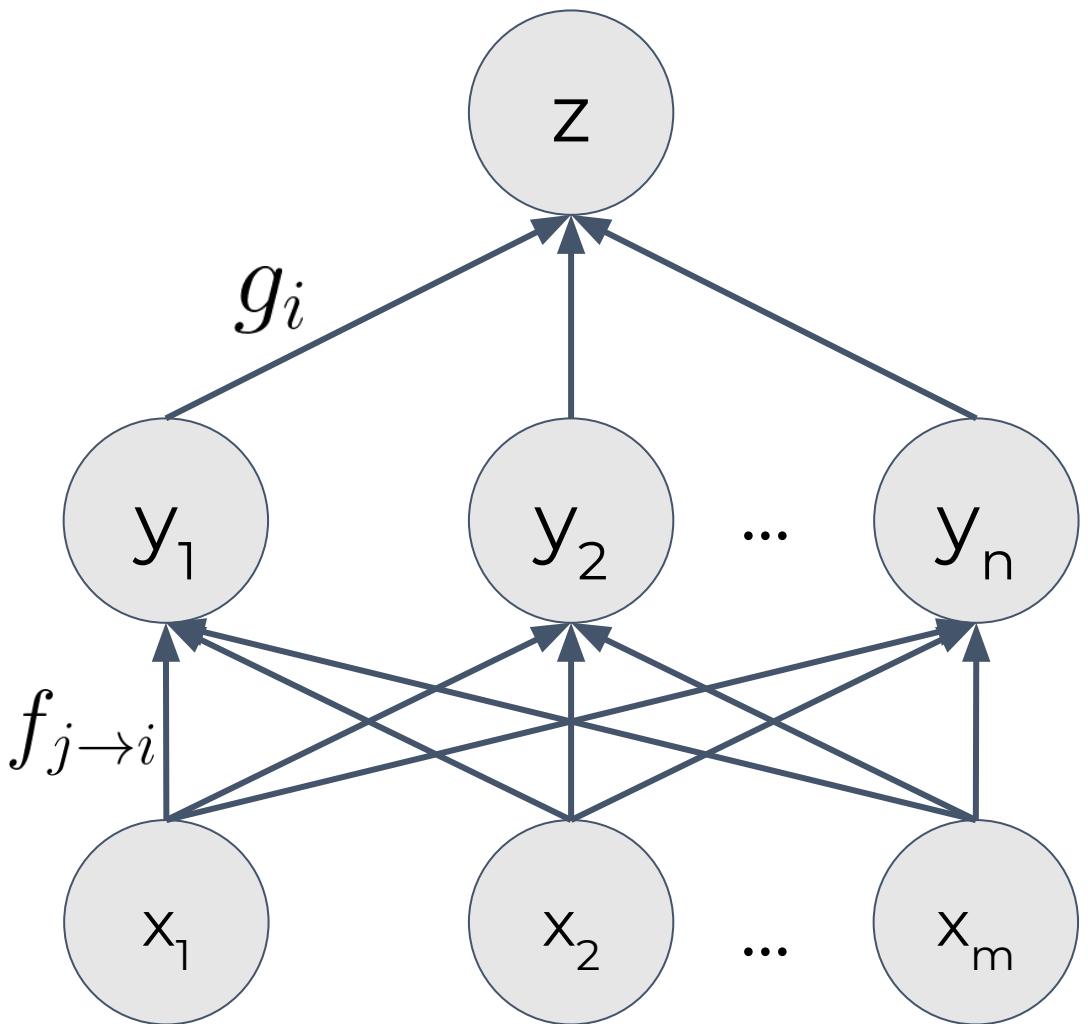


$$\Delta z \approx \sum_i g'_i(f_i(x)) f'_i(x) \Delta x$$

$$\Delta y_i \approx f'_i(x) \Delta x$$

$$\Delta x$$

Computational graph: Ex. 3

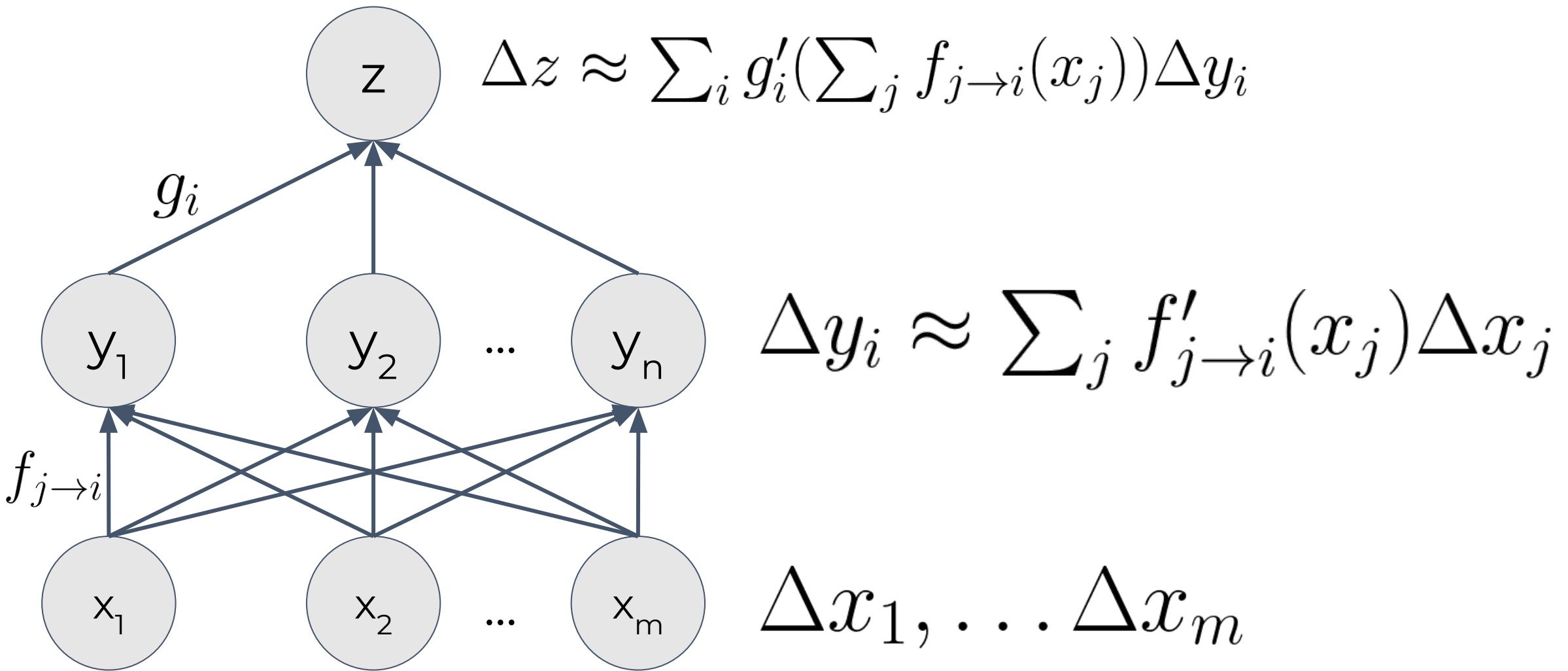


$$z = \sum_i g_i(y_i)$$

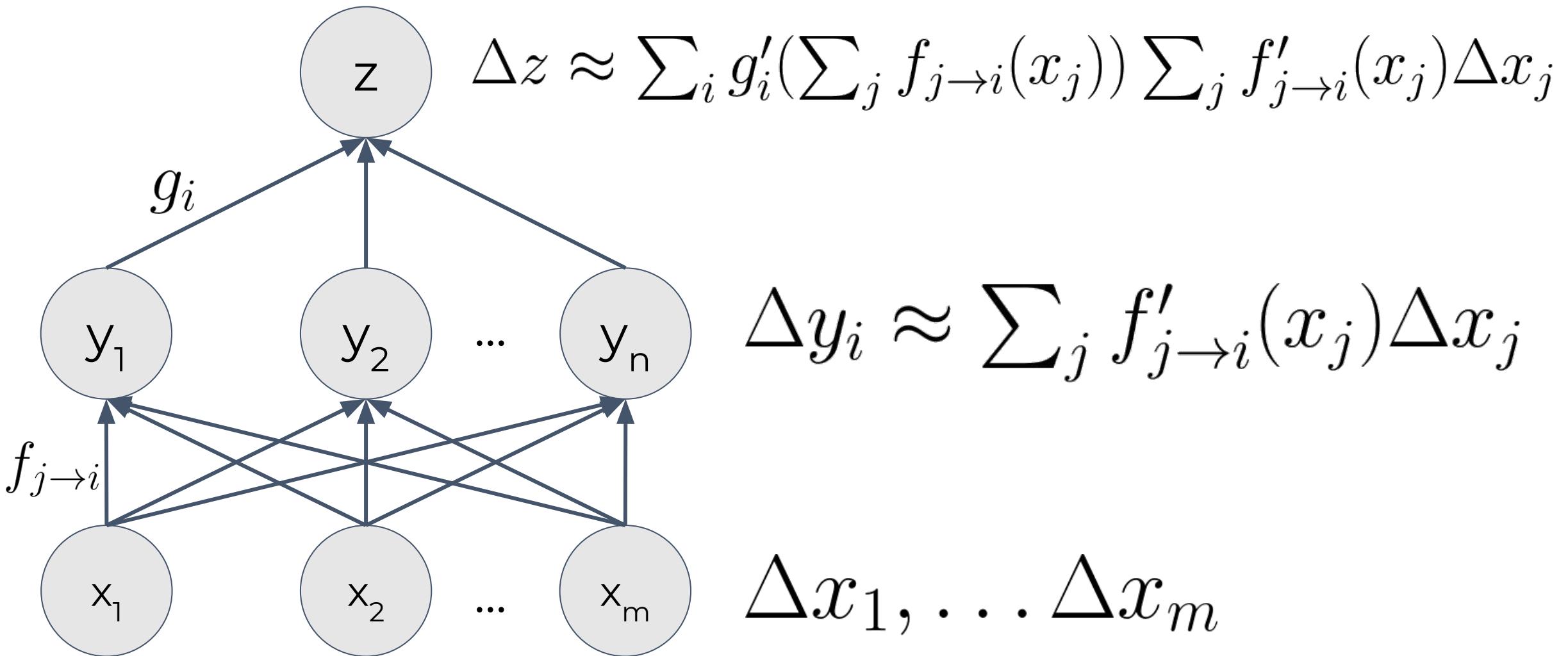
$$y_i = \sum_j f_{j \rightarrow i}(x_j)$$

$$\Delta x_1, \dots, \Delta x_m$$

Computational graph: Ex. 3



Computational graph: Ex. 3



Computational graph: Ex. 3

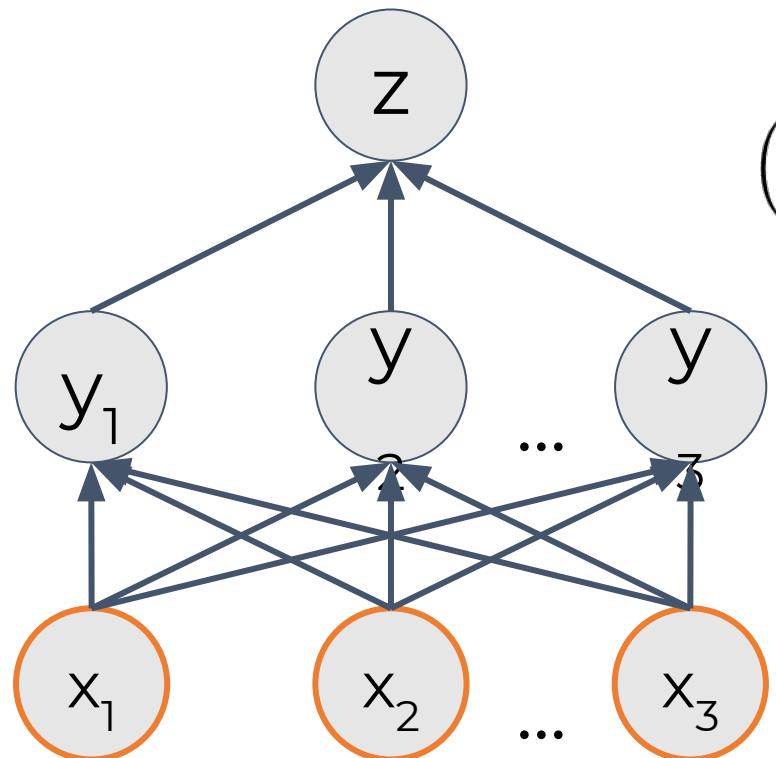
$$\Delta z = \sum_i g'_i(\sum_j f_{j \rightarrow i}(x_j)) \sum_j f'_{j \rightarrow i}(x_j) \Delta x_j$$

$$= \sum_j \sum_i g'_i(\sum_j f_{j \rightarrow i}(x_j)) f'_{j \rightarrow i}(x_j) \Delta x_j$$

$$= \langle \nabla z, \Delta x \rangle$$

$$(\nabla z)_j = \sum_i g'_i(\sum_j f_{j \rightarrow i}(x_j)) f'_{j \rightarrow i}(x_j)$$

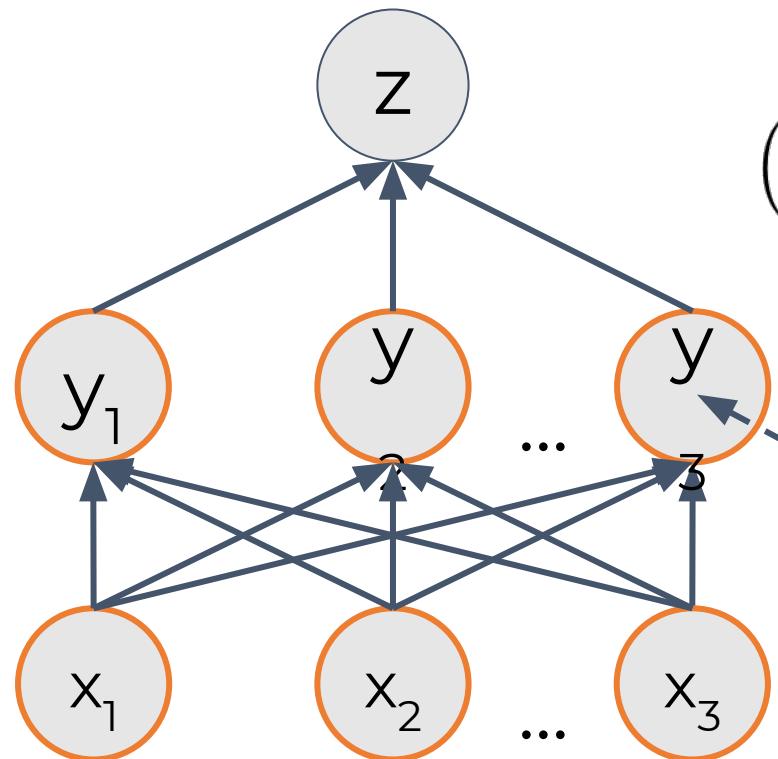
Backpropagation



$$(\nabla z)_j = \sum_i g'_i \left(\sum_j f_{j \rightarrow i}(x_j) \right) f'_{j \rightarrow i}(x_j)$$

Forward

Backpropagation



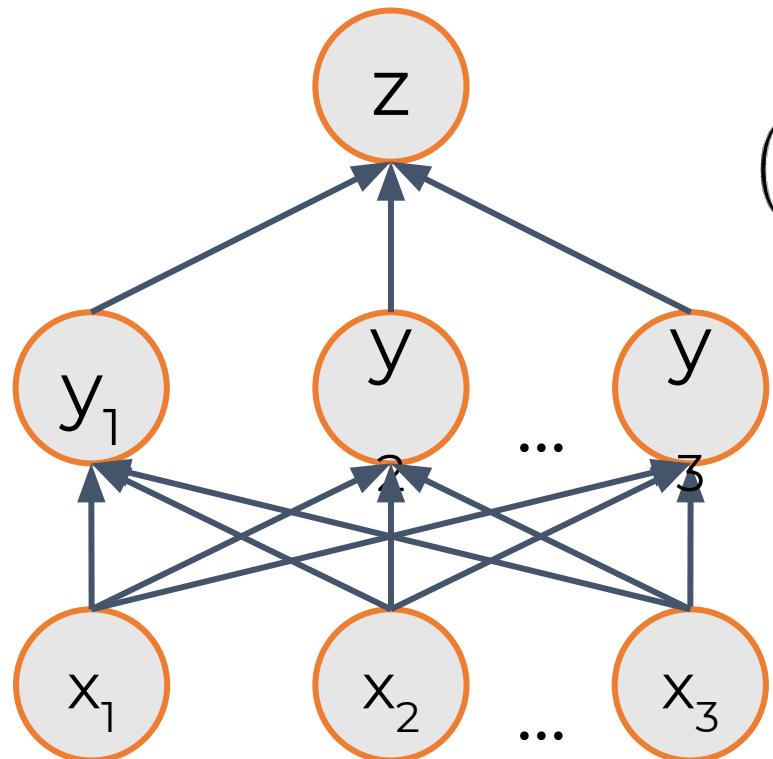
$$(\nabla z)_j = \sum_i g'_i \left(\sum_j f_{j \rightarrow i}(x_j) \right) f'_{j \rightarrow i}(x_j)$$

Forward

$$\sum_j f_{j \rightarrow i}(x_j)$$

Stores the intermediate results

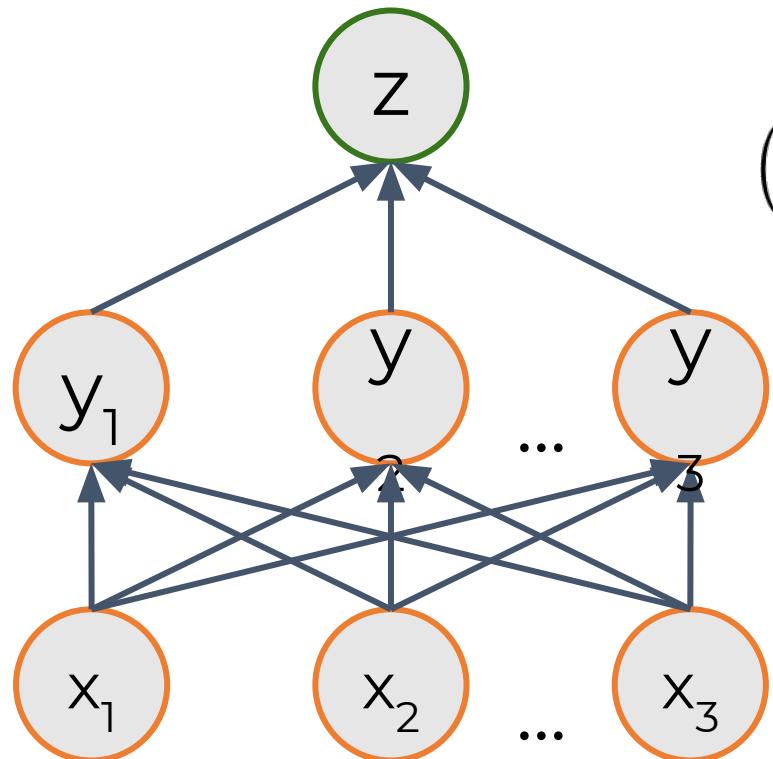
Backpropagation



$$(\nabla z)_j = \sum_i g'_i \left(\sum_j f_{j \rightarrow i}(x_j) \right) f'_{j \rightarrow i}(x_j)$$

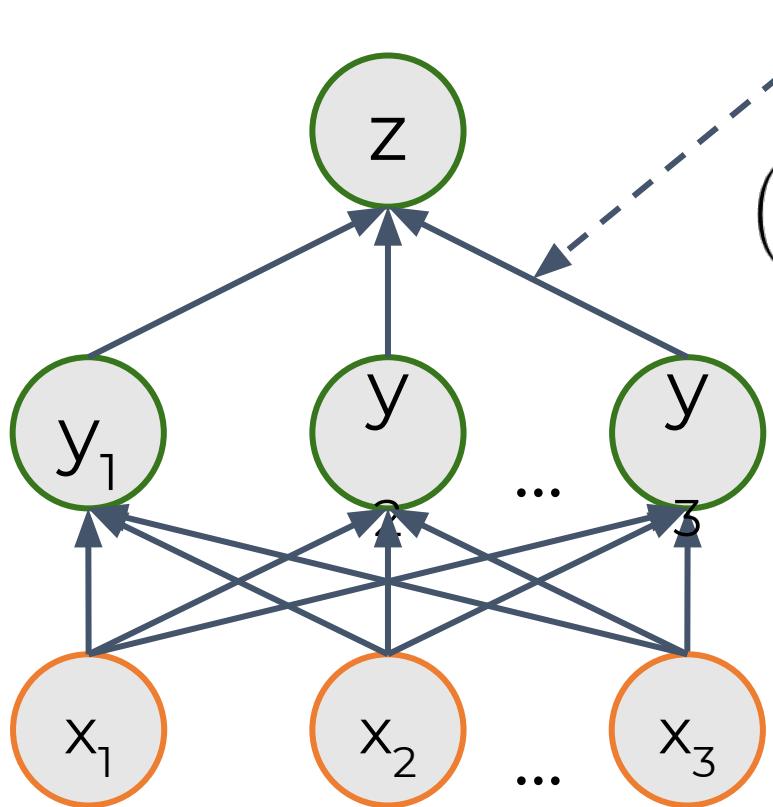
Forward

Backpropagation



$$(\nabla z)_j = \sum_i g'_i(\sum_j f_{j \rightarrow i}(x_j))f'_{j \rightarrow i}(x_j)$$

Backpropagation

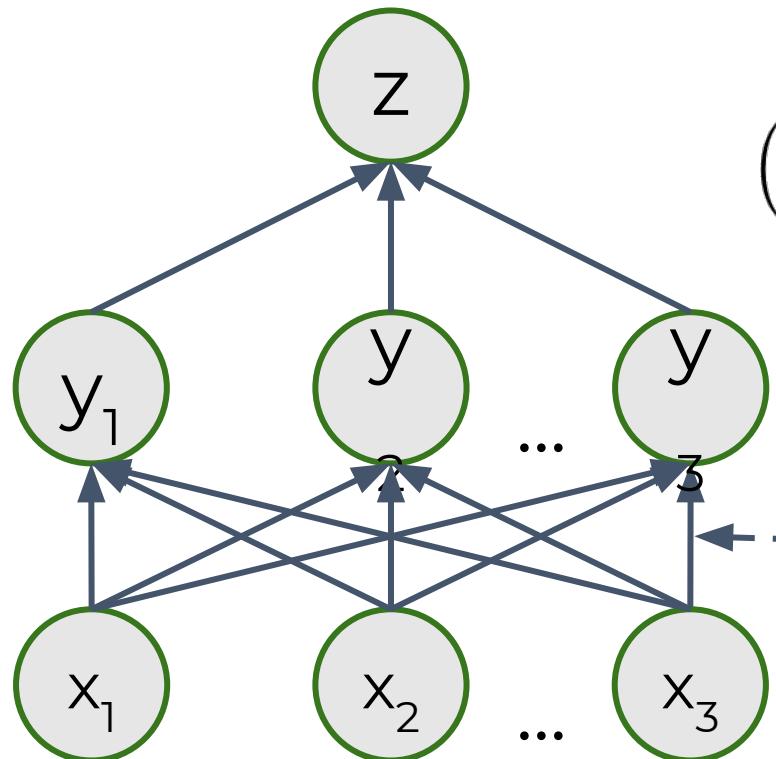


$$g'_i\left(\sum_j f_{j \rightarrow i}(x_j)\right)$$

Stores the intermediate results

$$(\nabla z)_j = \underbrace{\sum_i g'_i\left(\sum_j f_{j \rightarrow i}(x_j)\right)}_{\text{Backward}} f'_{j \rightarrow i}(x_j)$$

Backpropagation

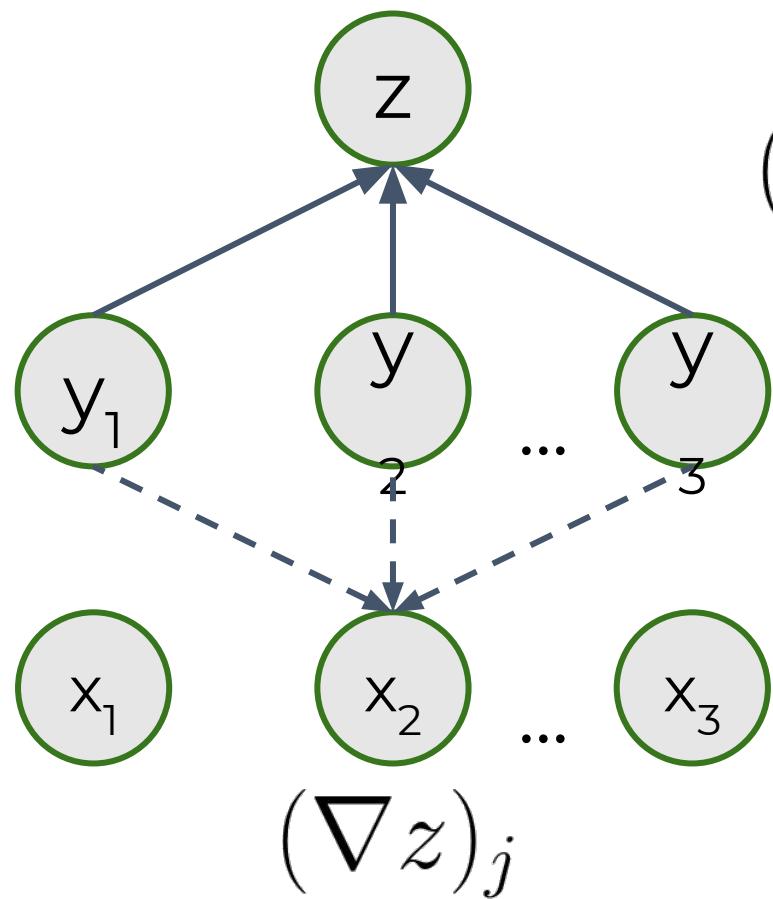


$$(\nabla z)_j = \sum_i g'_i(\sum_j f_{j \rightarrow i}(x_j))f'_{j \rightarrow i}(x_j)$$

$$f'_{j \rightarrow i}(x_j)$$

Stores the intermediate results

Backpropagation

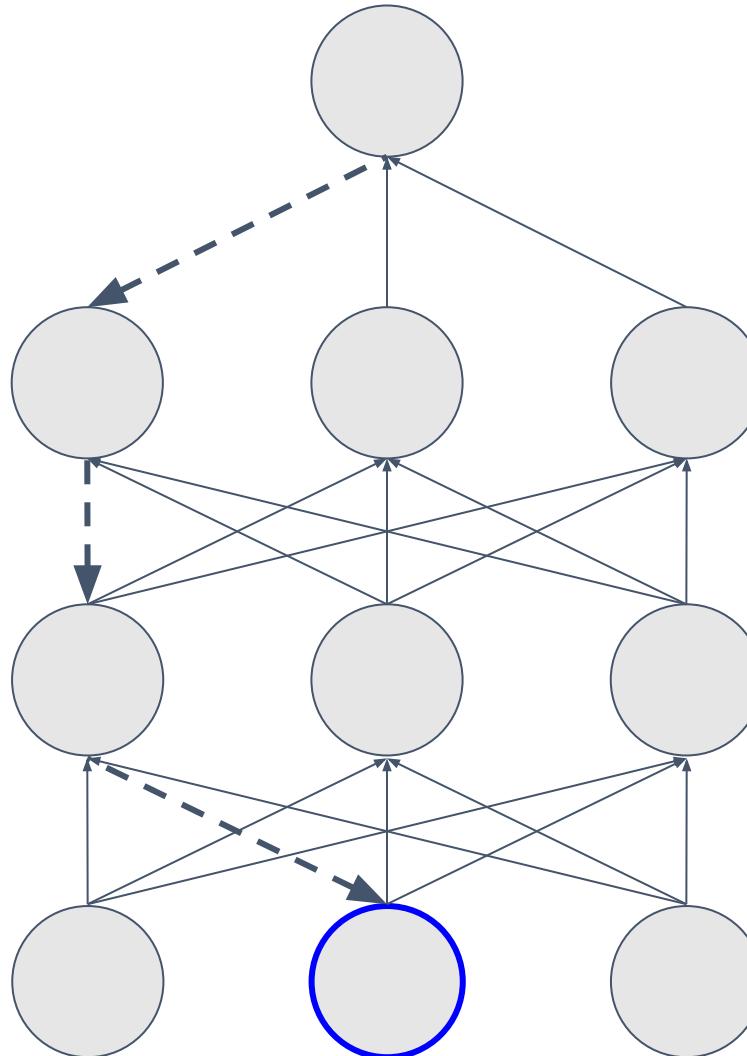


$$(\nabla z)_j = \underbrace{\sum_i g'_i(\sum_j f_{j \rightarrow i}(x_j))f'_{j \rightarrow i}(x_j)}_{\text{Backward}}$$

We reverse the order of propagation!

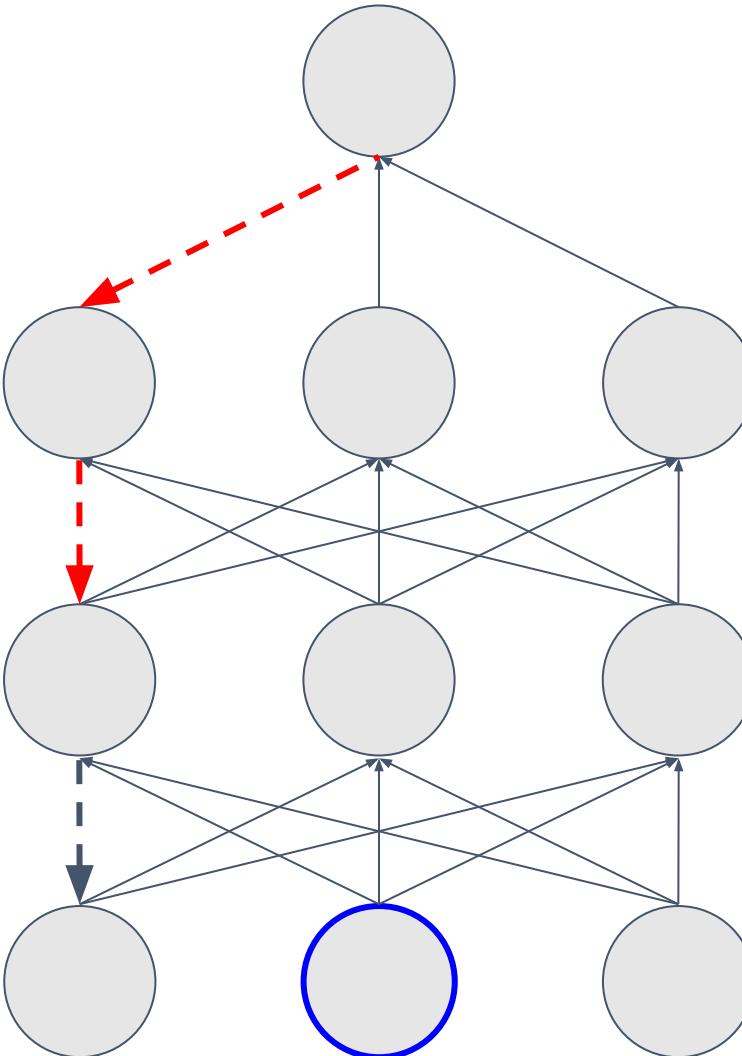
Backpropagation

Graph traversal
problem



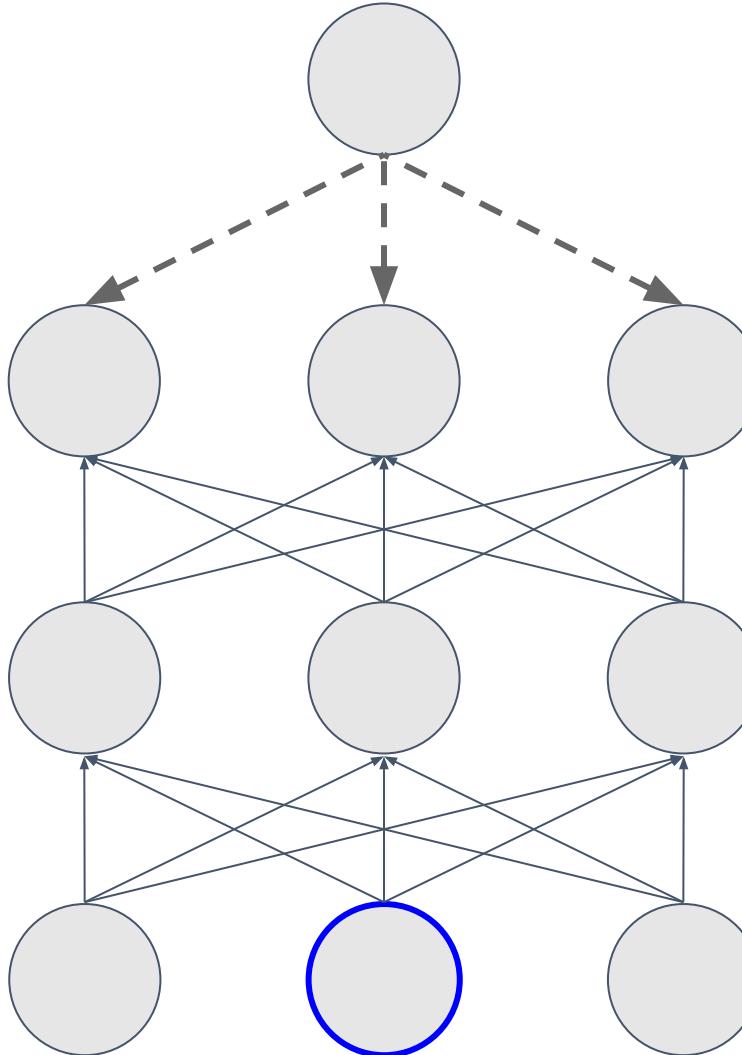
Backpropagation

We have already
computed the product
on this partial path!



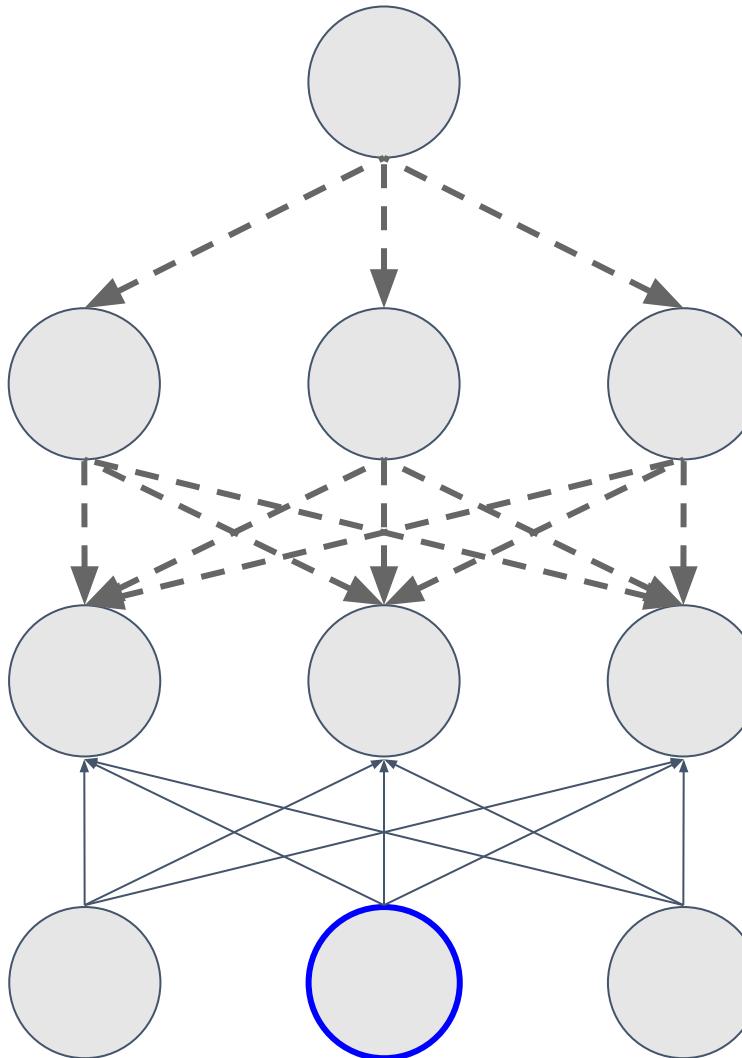
Backpropagation

Dynamic
programming



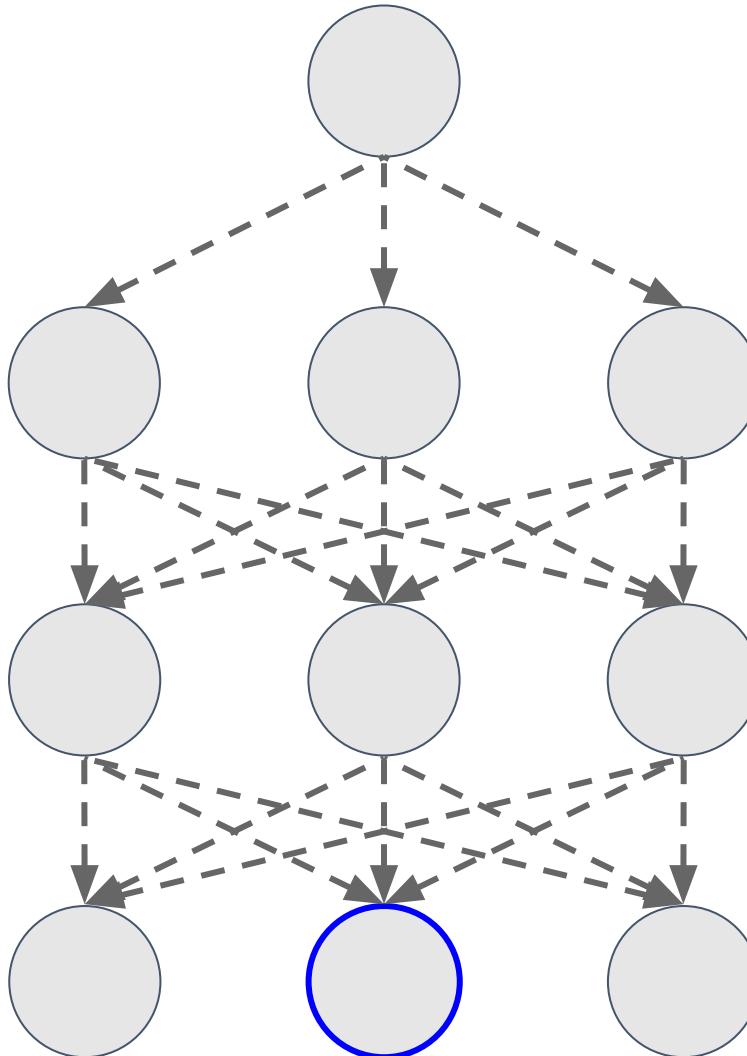
Backpropagation

Dynamic
programming

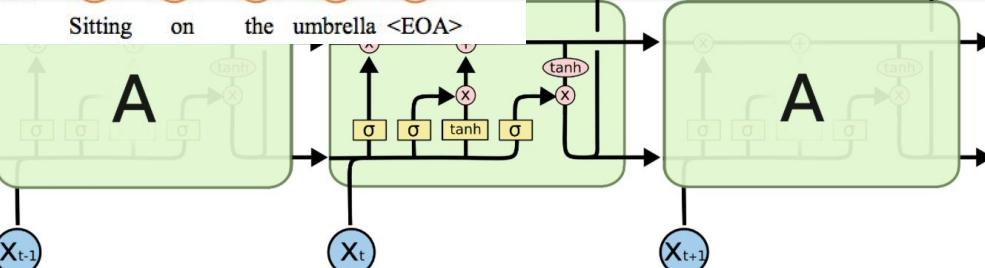
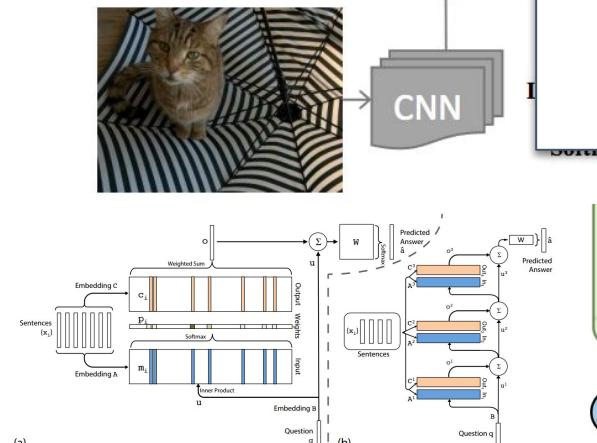
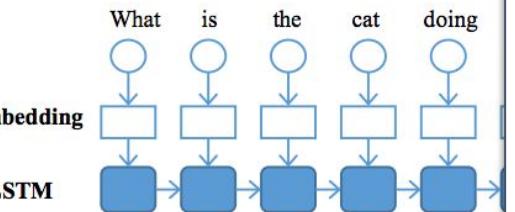
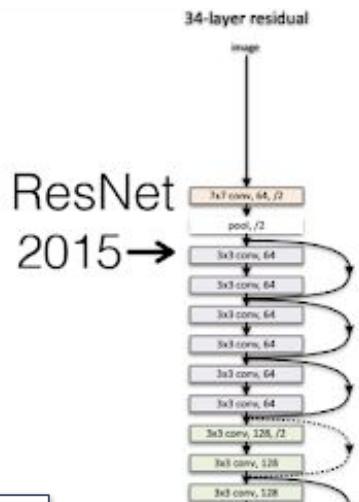
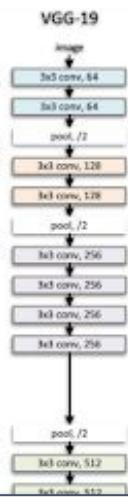
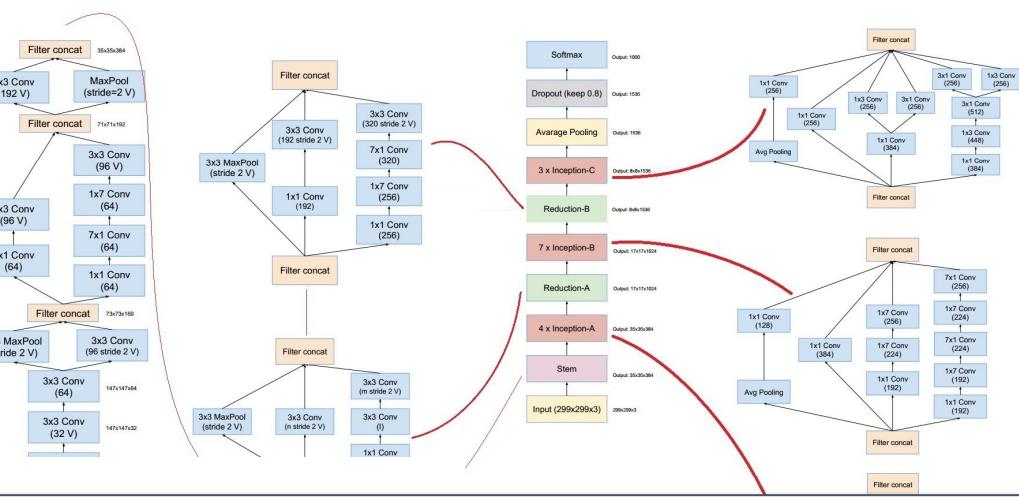
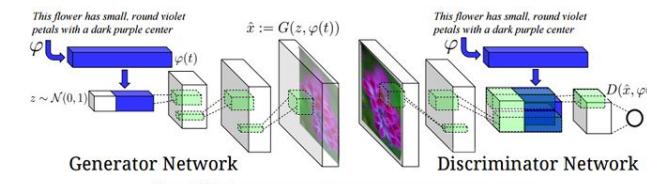


Backpropagation

Dynamic
programming



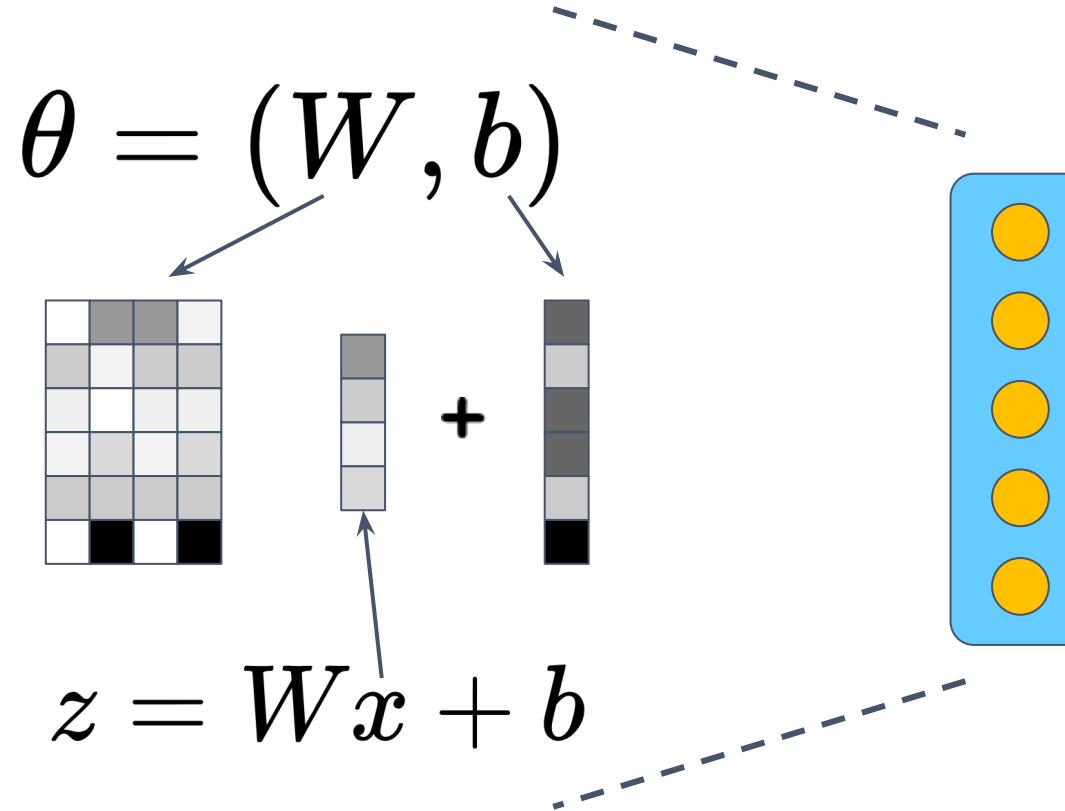
Seppe Linnainmaa (1970): automatic differentiation
Rumelhart et al., (1986): neural networks



The repeating module in an LSTM contains four interacting layers.

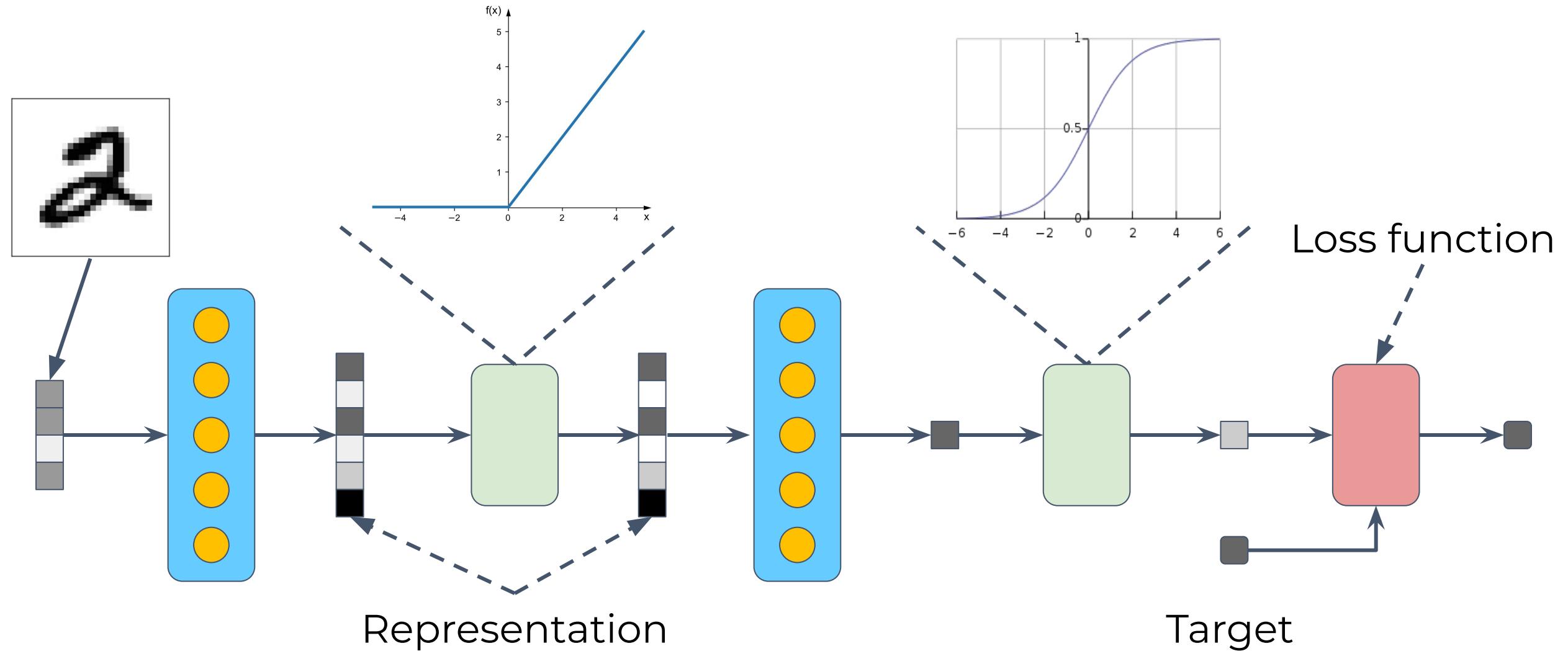
Module definition: dense module

Parameters



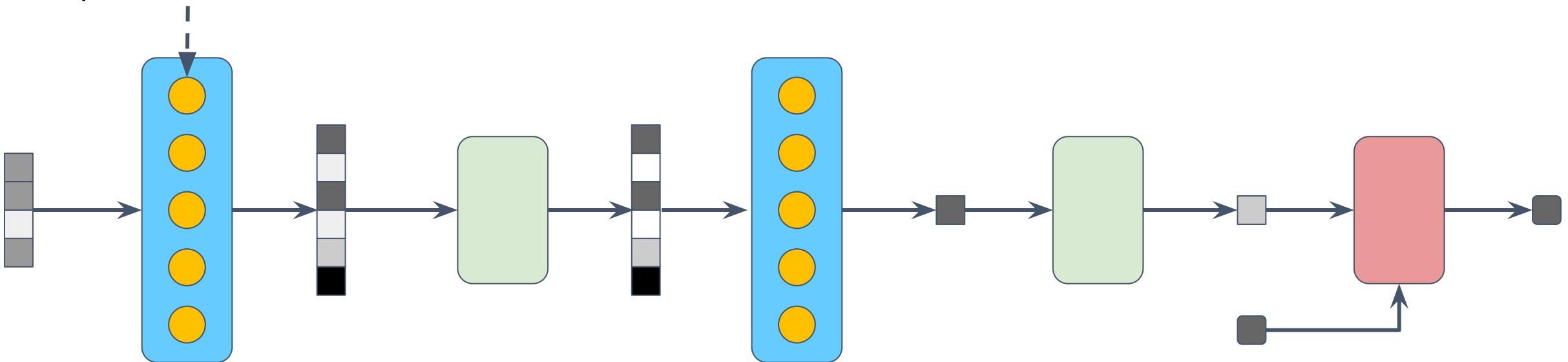
Transformation

Example



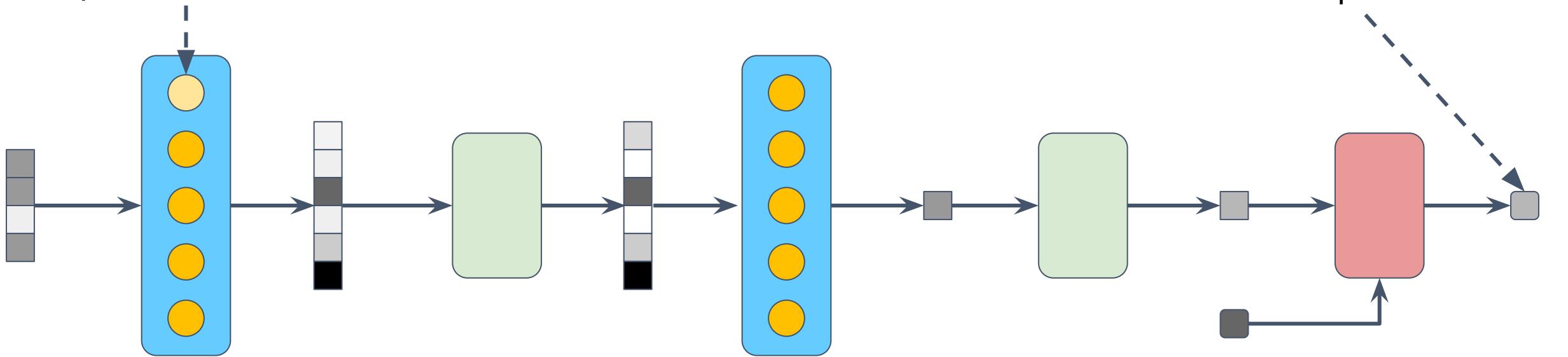
Impact of a parameter

Change of a parameter



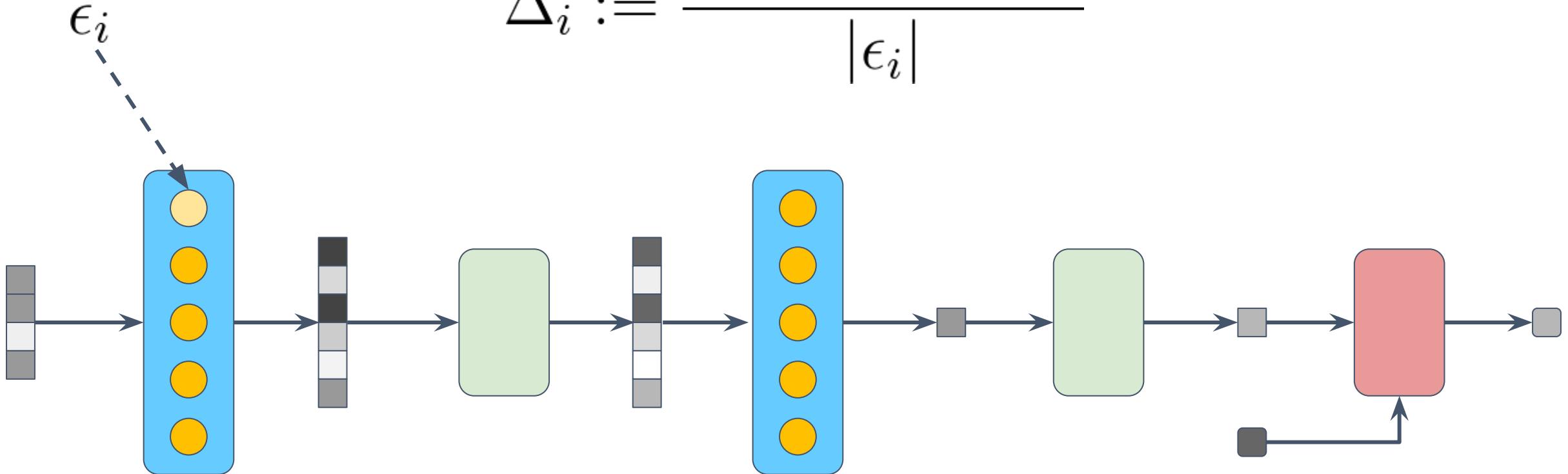
Impact of a parameter

Change of a parameter



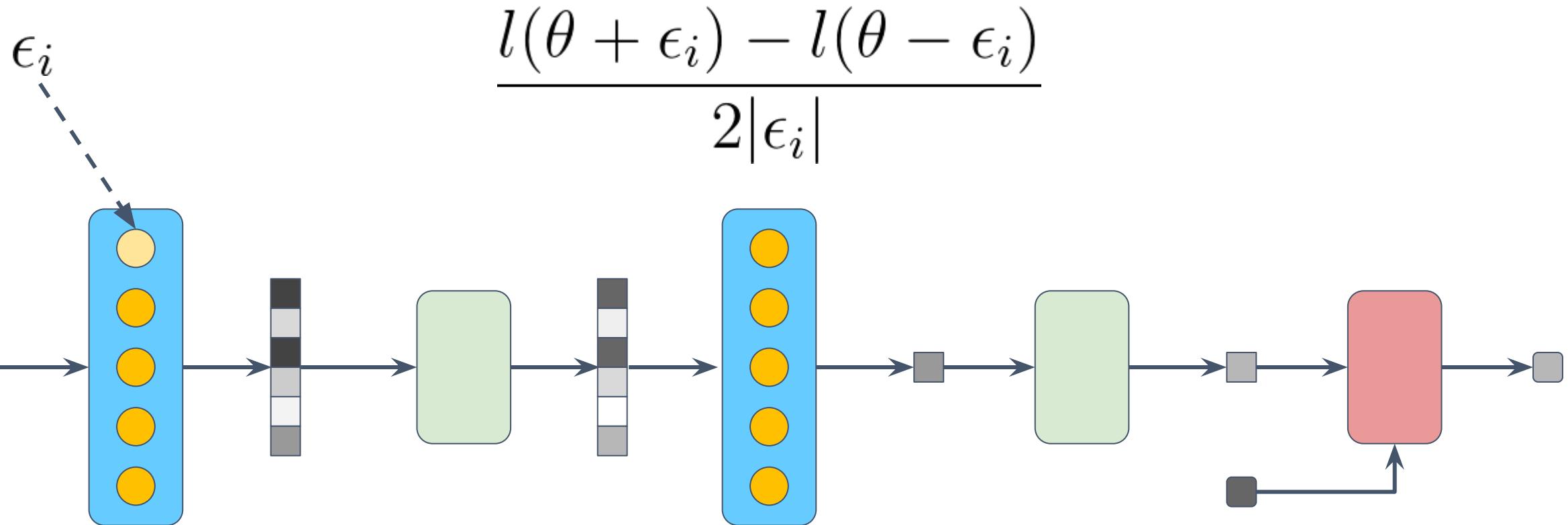
Impact of a parameter

$$\Delta_i := \frac{l(\theta + \epsilon_i) - l(\theta)}{|\epsilon_i|}$$



Impact of a parameter

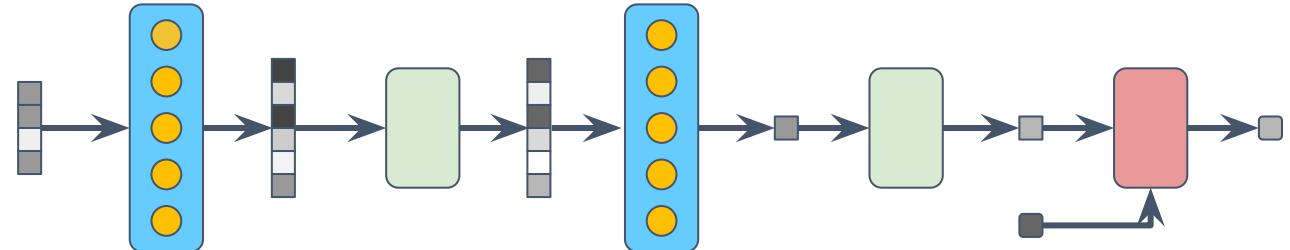
Finite difference method



Impact of all parameters

Complexity of
the finite difference method:
 $2*n$ propagations

n : Number of parameters



Implementation of a module

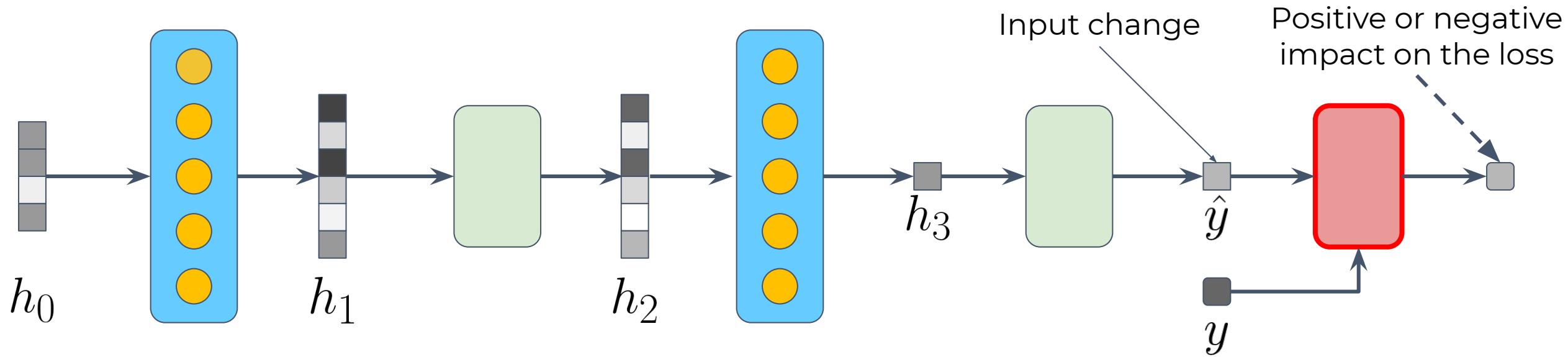
- Forward(x): data transformation
- Backward(g): gradient transformation (output->input)
- Update(g): calculation of the parameter gradient

Example

Forward

$$l(\hat{y}, y) = -y \ln \hat{y} - (1 - y) \ln(1 - \hat{y})$$

Binary Cross-Entropy



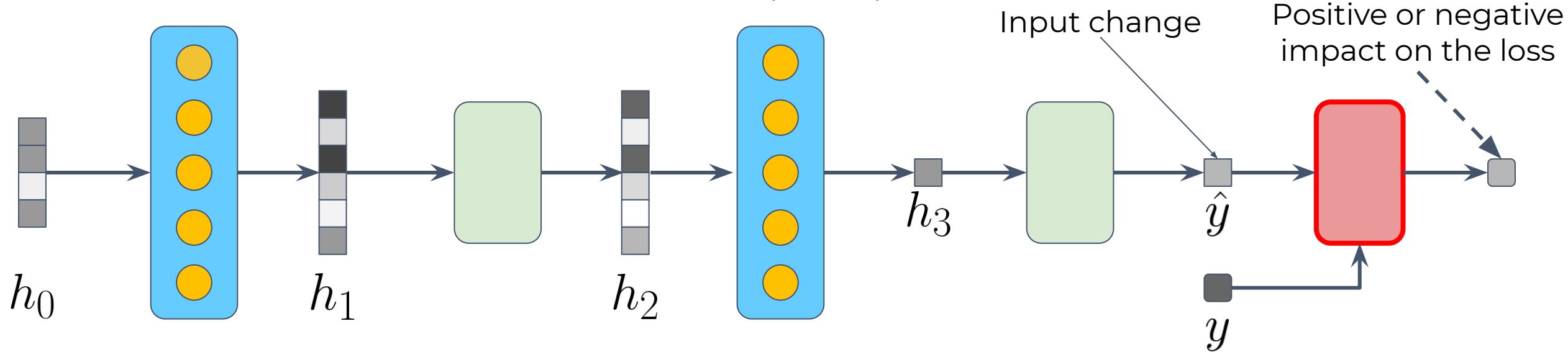
Example

Forward

$$l(\hat{y}, y) = -y \ln \hat{y} - (1 - y) \ln(1 - \hat{y})$$

Backward

$$\partial_{\hat{y}} l(\hat{y}, y) = \frac{\hat{y} - y}{\hat{y}(1 - \hat{y})}$$



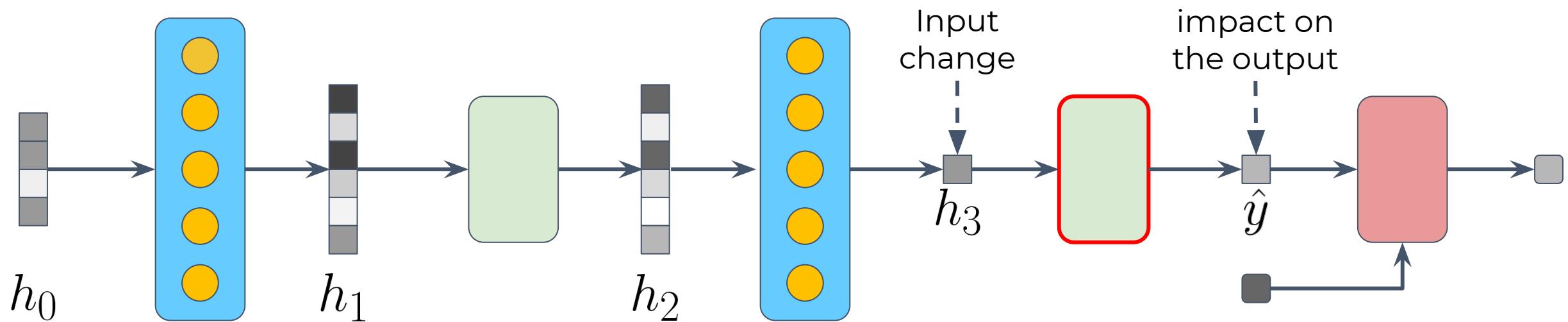
Example

Forward

Backward

Sigmoid

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



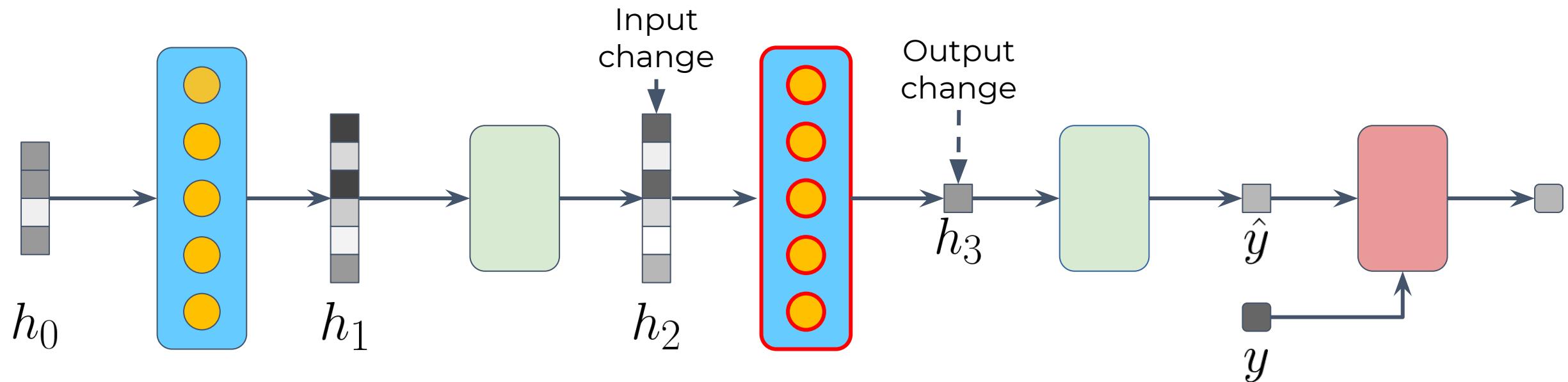
Example

Forward $\underbrace{g(x; W, b) = Wx + b}_{\text{Linear}}$

Backward $\partial_x g(x; W, b) = W$

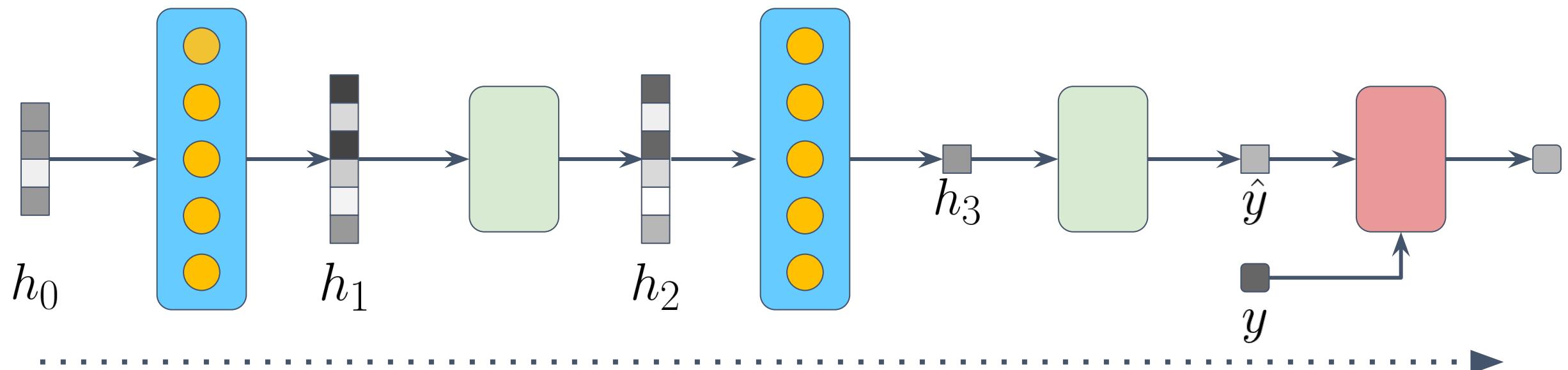
Update $\partial_W g(x; W, b) = x^\top$

$\partial_b g(x; W, b) = \mathbb{I}$



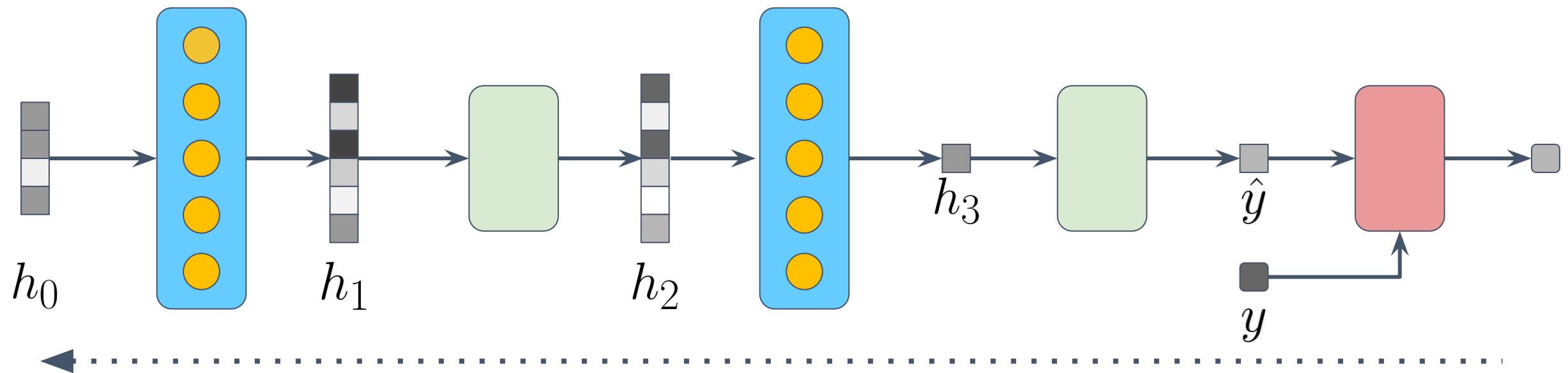
Forward

Evaluation of the differential at the current point in the parameter space.



Backward

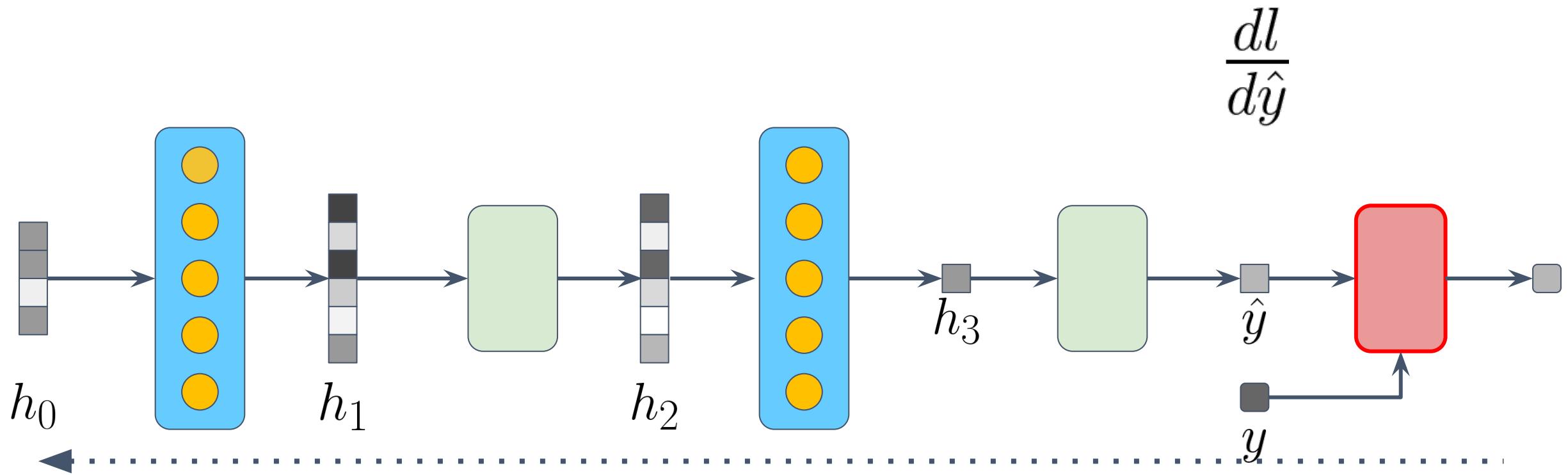
- Efficient calculation of the chain rule (matrix version)
- matrix multiplication: Jacobian*gradient



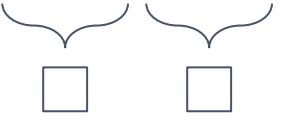
Backward

$$\frac{dh_1}{dh_0} \frac{dh_2}{dh_1} \frac{dh_3}{dh_2} \frac{d\hat{y}}{dh_3} \frac{dl}{d\hat{y}}$$

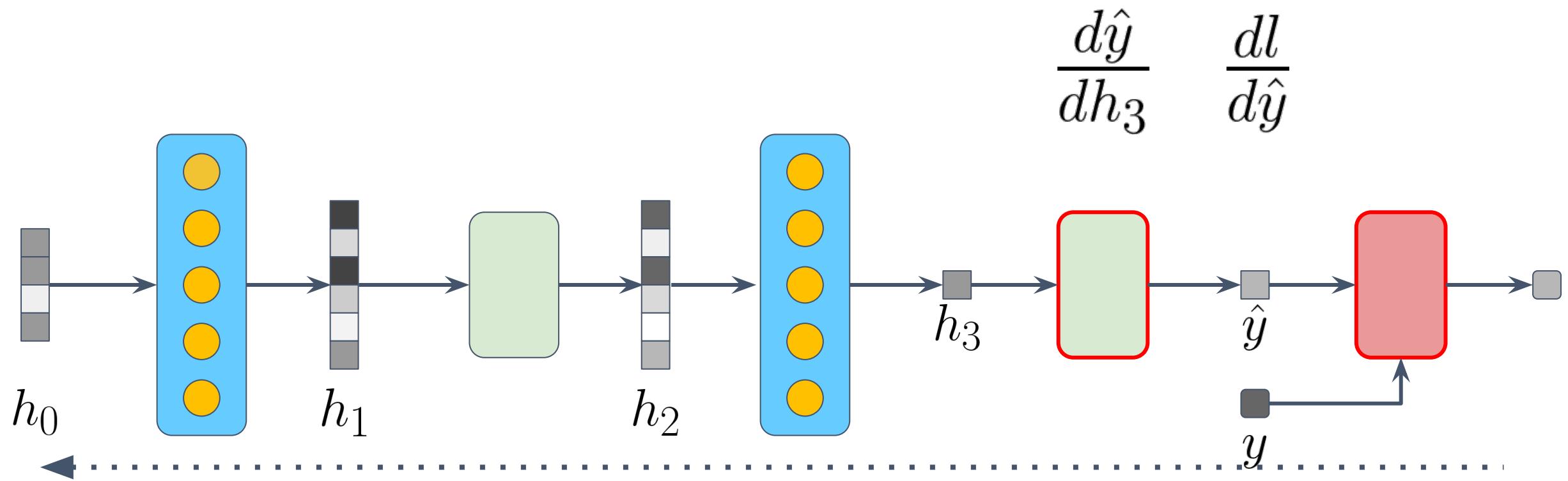
□



Backward

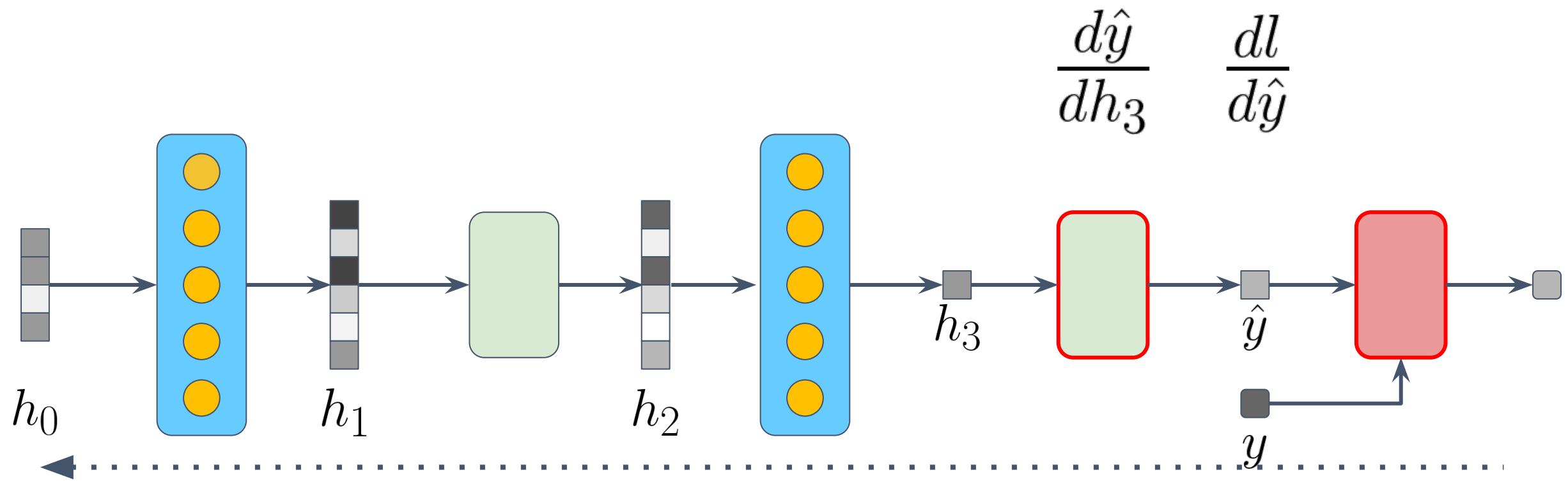
$$\frac{dh_1}{dh_0} \frac{dh_2}{dh_1} \frac{dh_3}{dh_2} \frac{d\hat{y}}{dh_3} \frac{dl}{d\hat{y}}$$


□ □



Backward

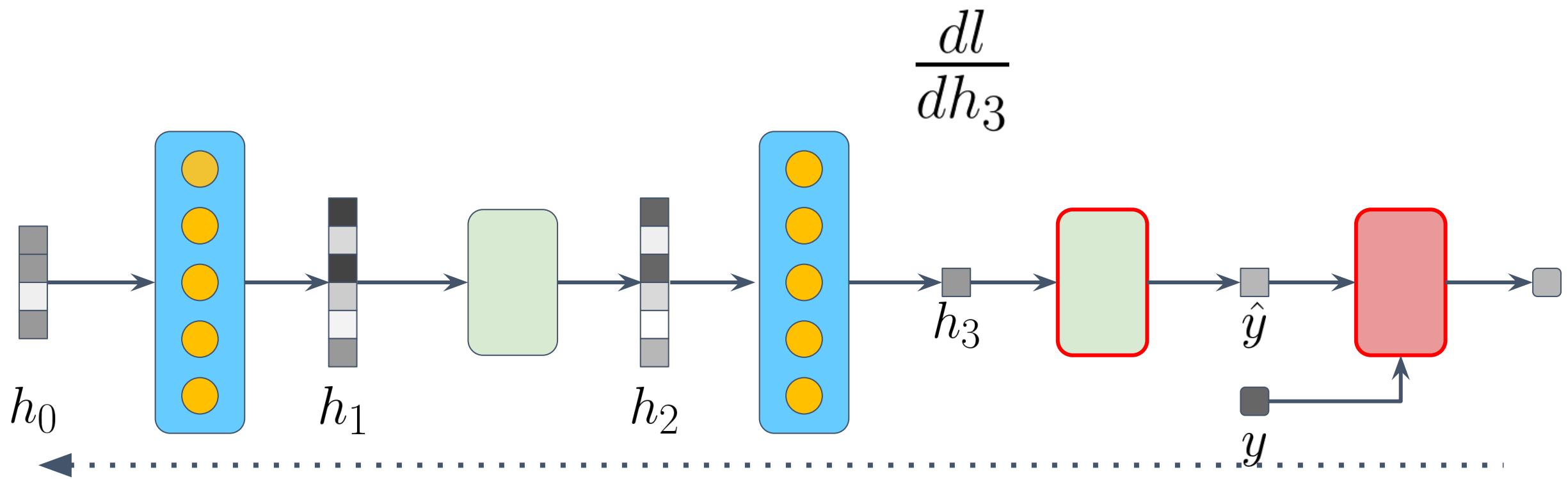
$$\frac{dh_1}{dh_0} \frac{dh_2}{dh_1} \frac{dh_3}{dh_2} \underbrace{\frac{d\hat{y}}{dh_3} \frac{dl}{d\hat{y}}}_{\square}$$



Backward

$$\frac{dh_1}{dh_0} \frac{dh_2}{dh_1} \frac{dh_3}{dh_2} \frac{dl}{dh_3}$$

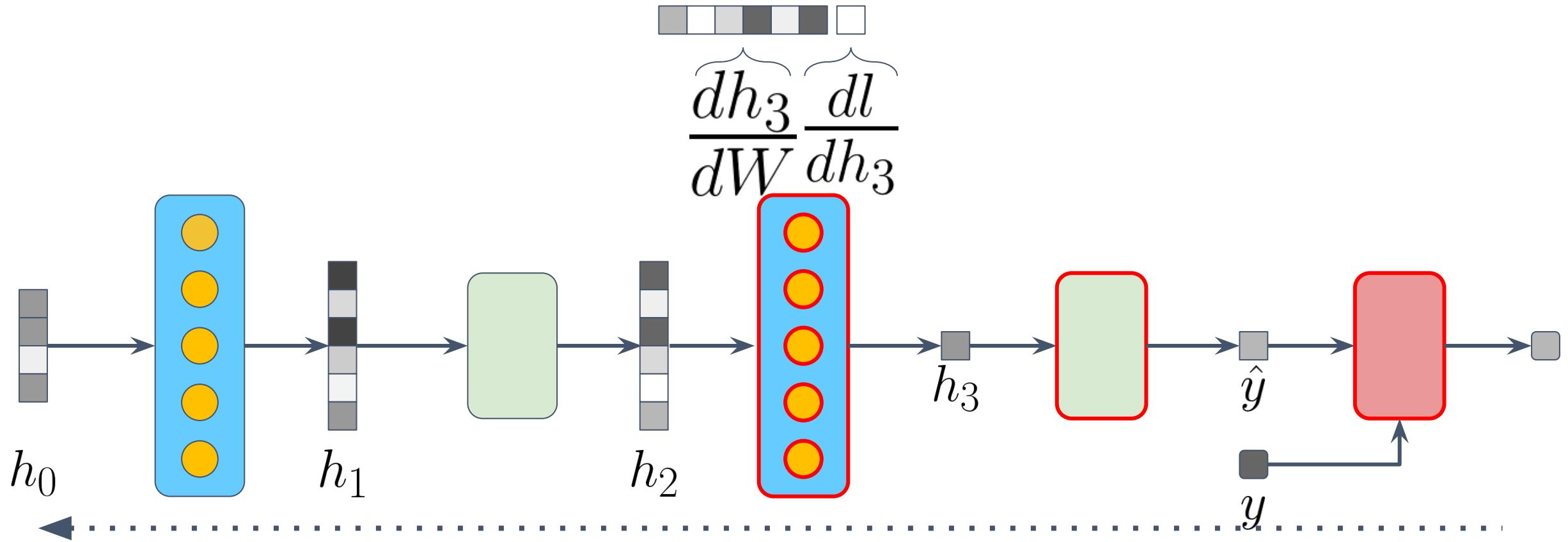
□



Update

$$\frac{dh_1}{dh_0} \frac{dh_2}{dh_1} \frac{dh_3}{dh_2} \frac{dl}{dh_3}$$

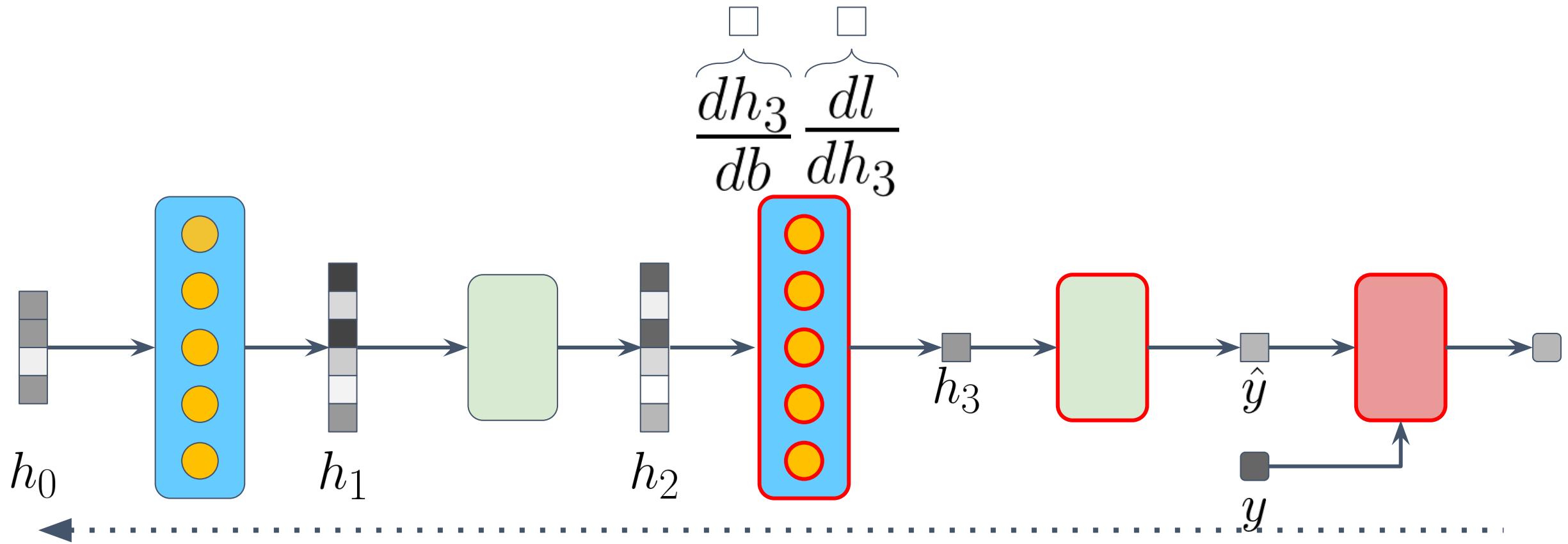
□



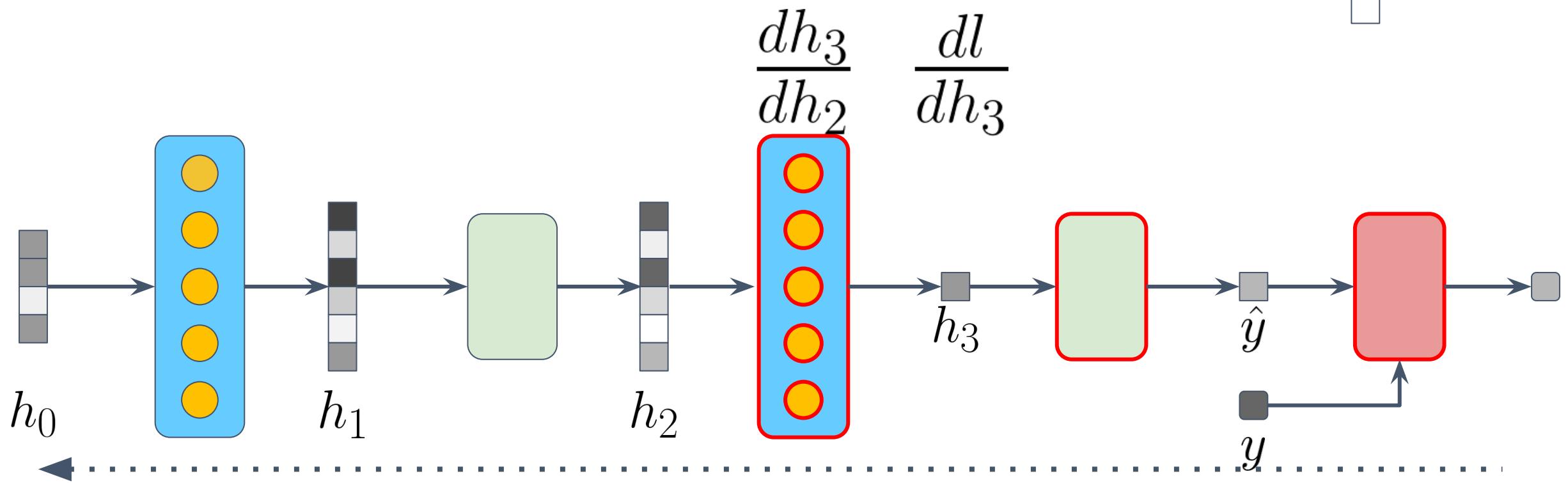
Update

$$\frac{dh_1}{dh_0} \frac{dh_2}{dh_1} \frac{dh_3}{dh_2} \frac{dl}{dh_3}$$

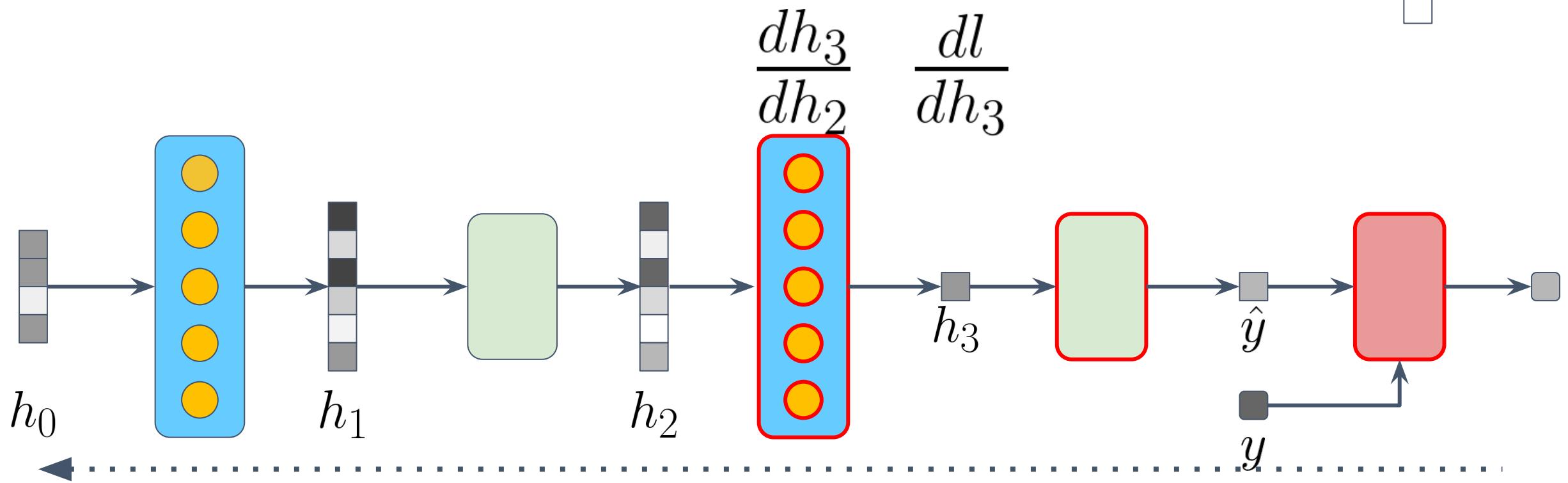
□



Backward



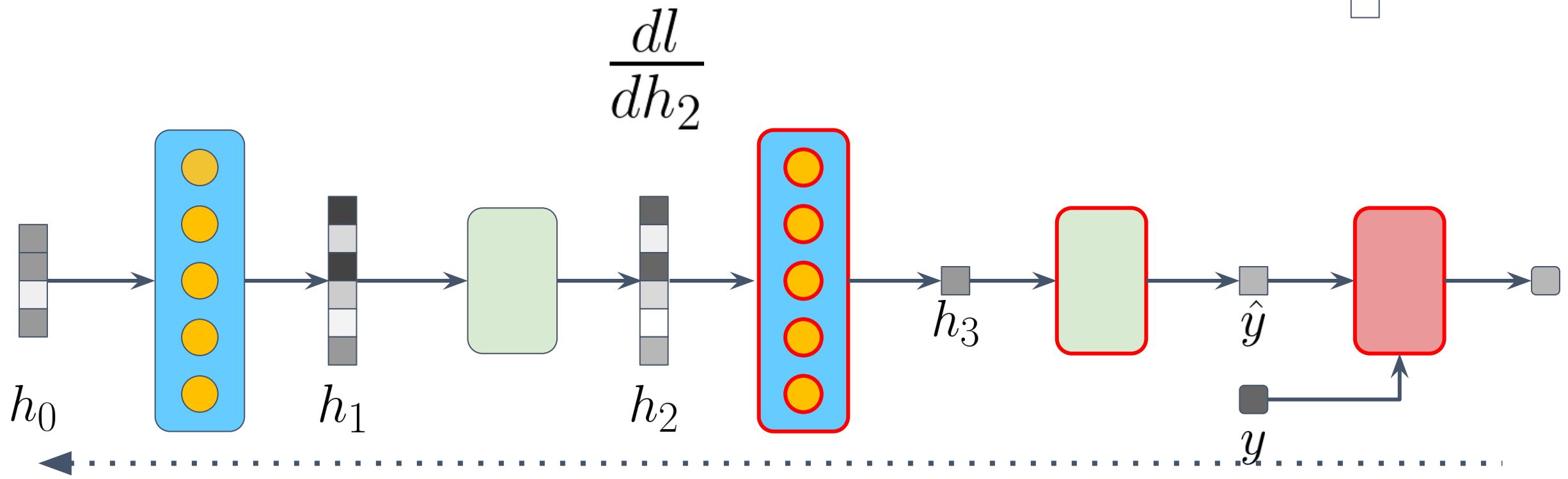
Backward



Backward

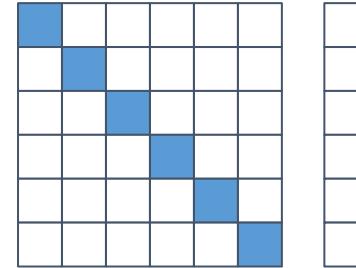
$$\frac{dh_1}{dh_0} \frac{dh_2}{dh_1} \frac{dl}{dh_2}$$

$$\frac{dl}{dh_2}$$

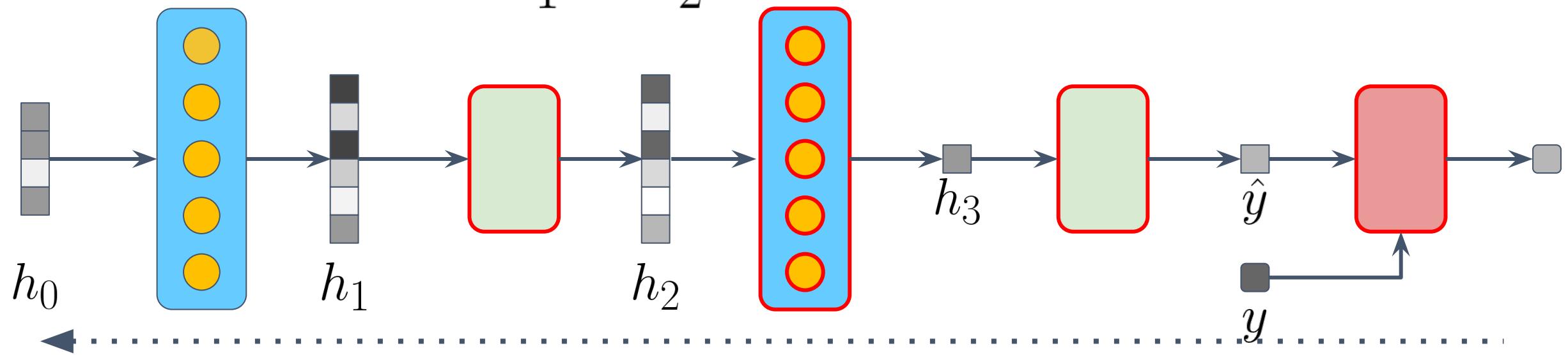


Backward

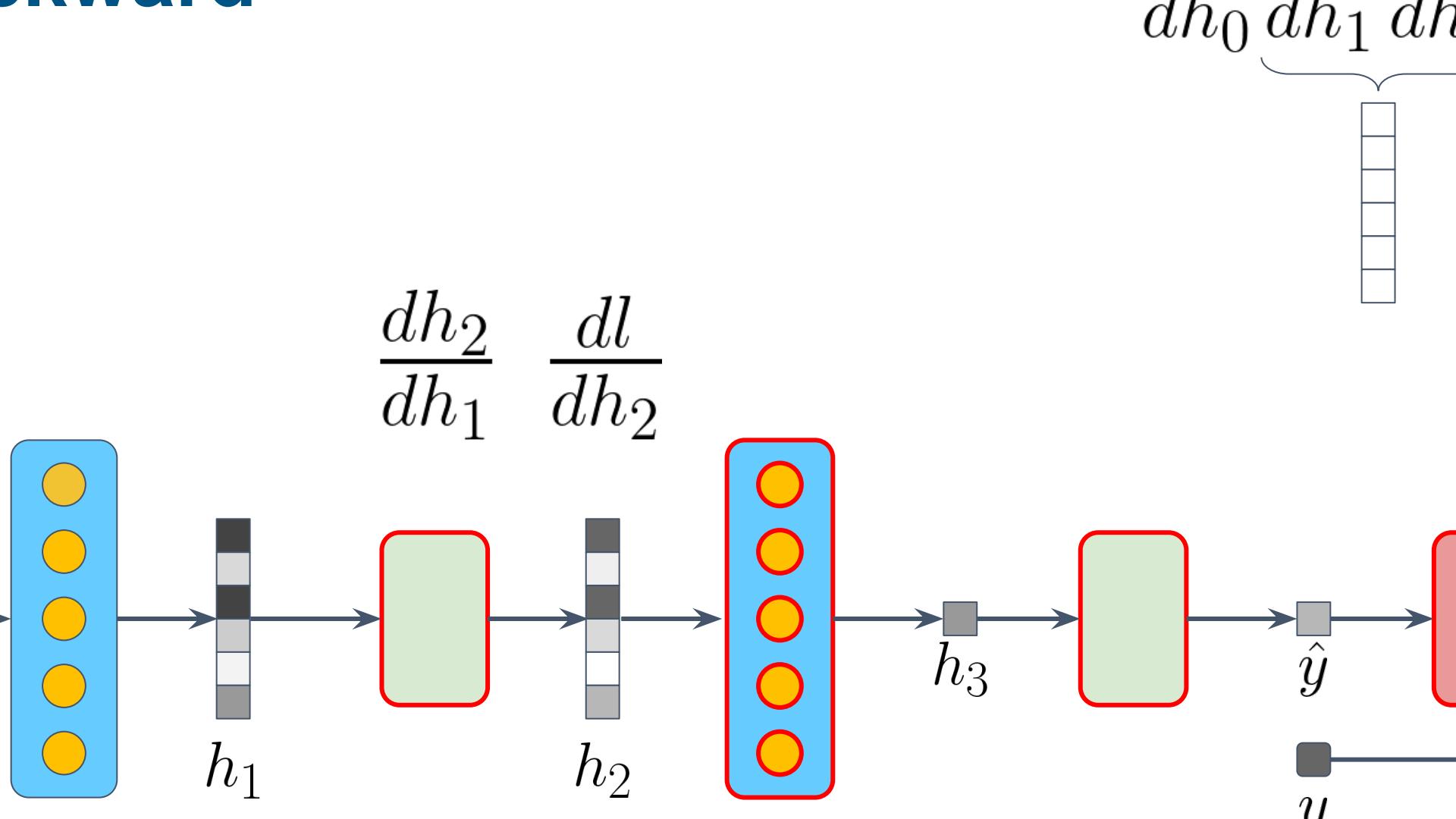
$$\frac{dh_1}{dh_0} \frac{dh_2}{dh_1} \frac{dl}{dh_2}$$



$$\frac{dh_2}{dh_1} \quad \frac{dl}{dh_2}$$

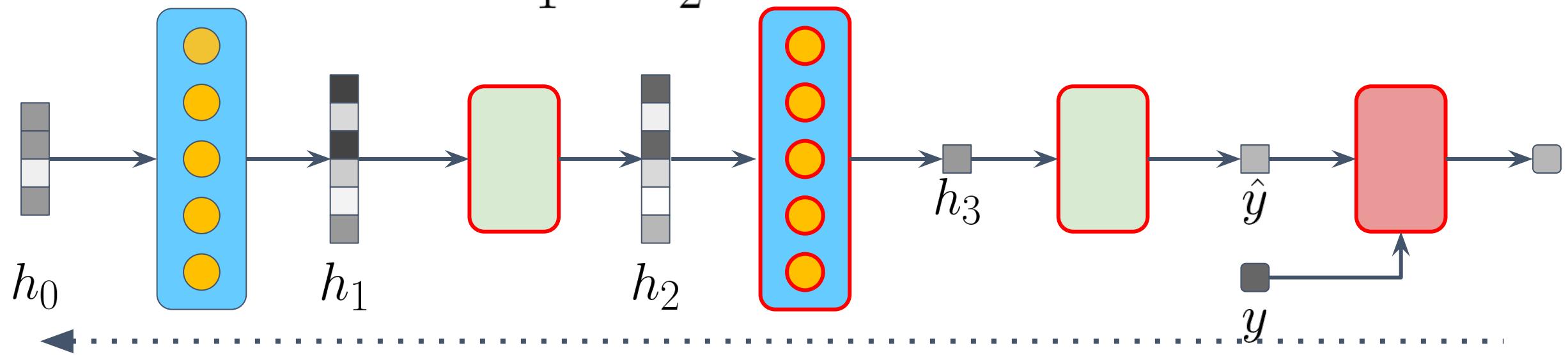


Backward

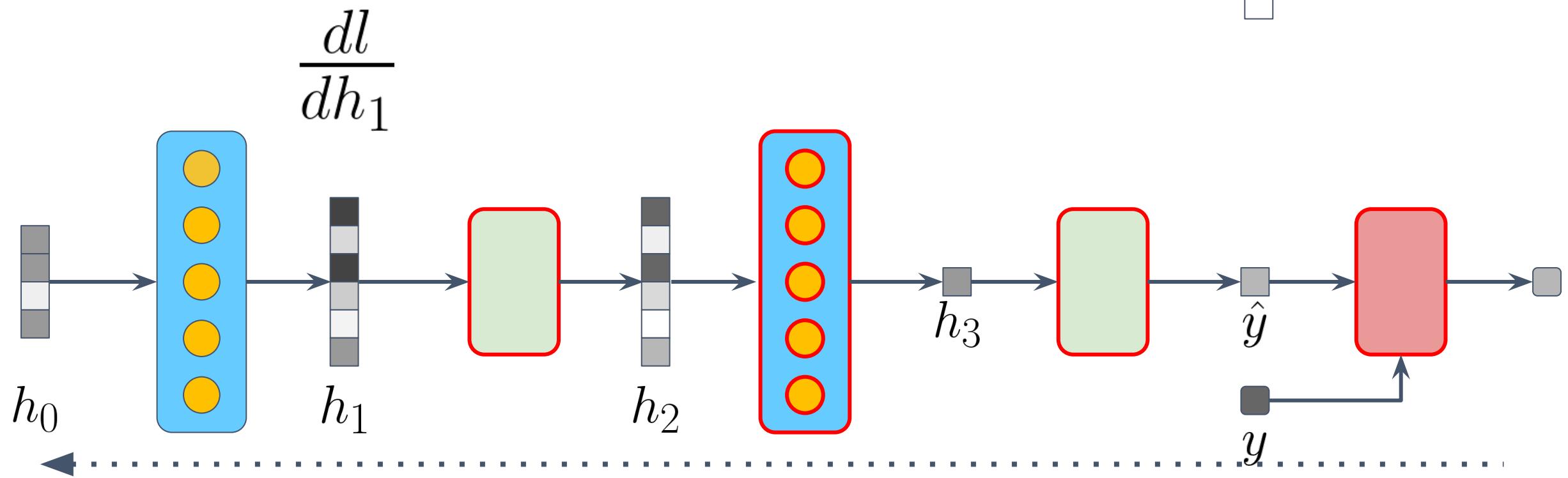
$$\frac{dh_1}{dh_0} \frac{dh_2}{dh_1} \frac{dl}{dh_2}$$


A diagram illustrating the backward pass through a neural network. The forward pass consists of layers h_0 , h_1 , h_2 , h_3 , \hat{y} , and y . The input h_0 is a vector of gray blocks. Layer h_1 is a blue rectangle containing five yellow circles. Layer h_2 is a green rectangle. Layer h_3 is a blue rectangle containing four yellow circles. The output \hat{y} is a red rectangle, and the target y is a gray square. The backward pass involves calculating gradients: $\frac{dh_2}{dh_1}$ (gradient of h_2 w.r.t h_1) and $\frac{dl}{dh_2}$ (gradient of loss l w.r.t h_2). These gradients are multiplied by the forward pass gradients $\frac{dh_1}{dh_0}$ (gradient of h_1 w.r.t h_0) and $\frac{dh_2}{dh_1}$ (gradient of h_2 w.r.t h_1), which are represented by vertical rectangles.

$$\frac{dh_2}{dh_1} \quad \frac{dl}{dh_2}$$

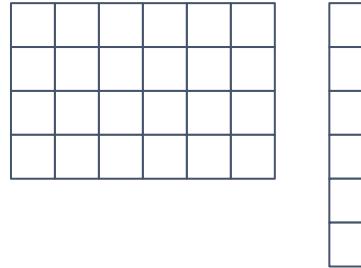


Backward

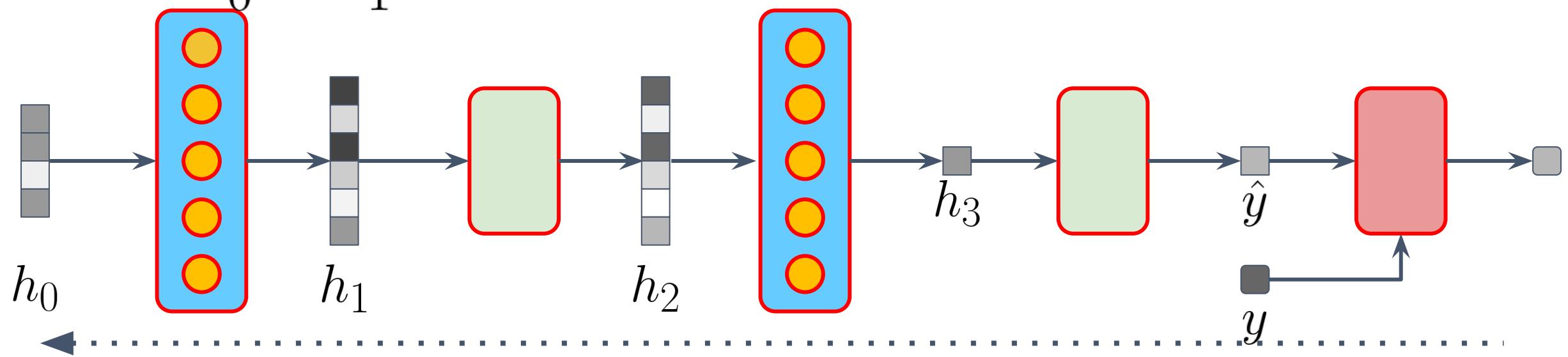


Backward

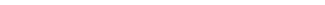
$$\frac{dh_1}{dh_0} \frac{dl}{dh_1}$$

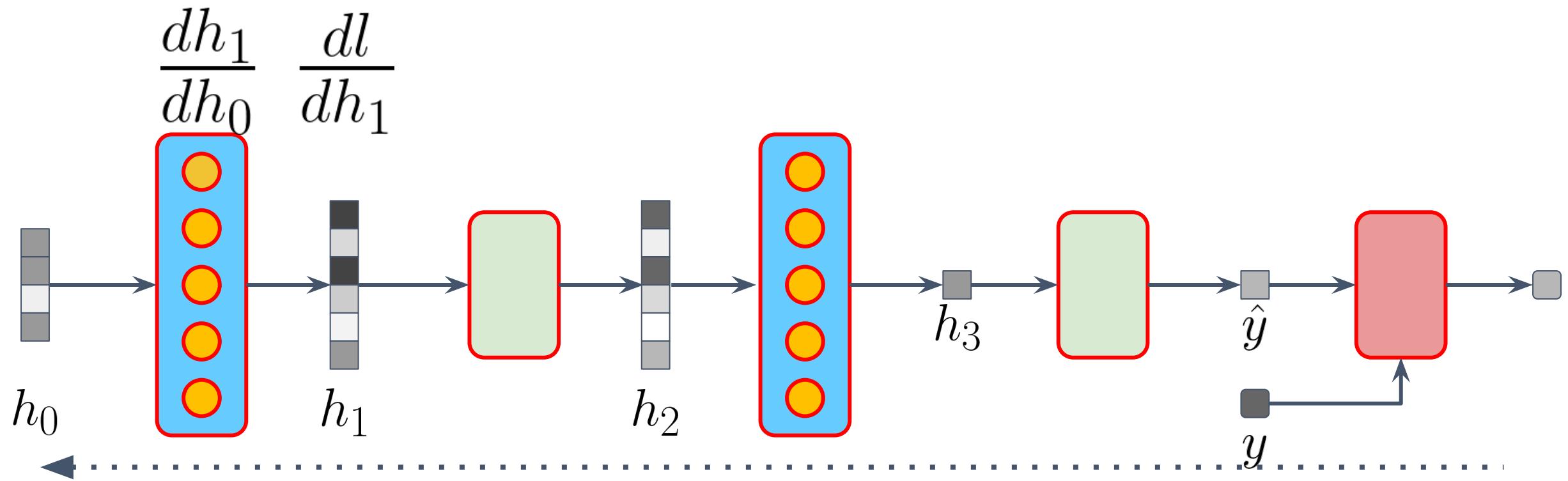


$$\frac{dh_1}{dh_0} \quad \frac{dl}{dh_1}$$

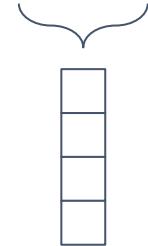


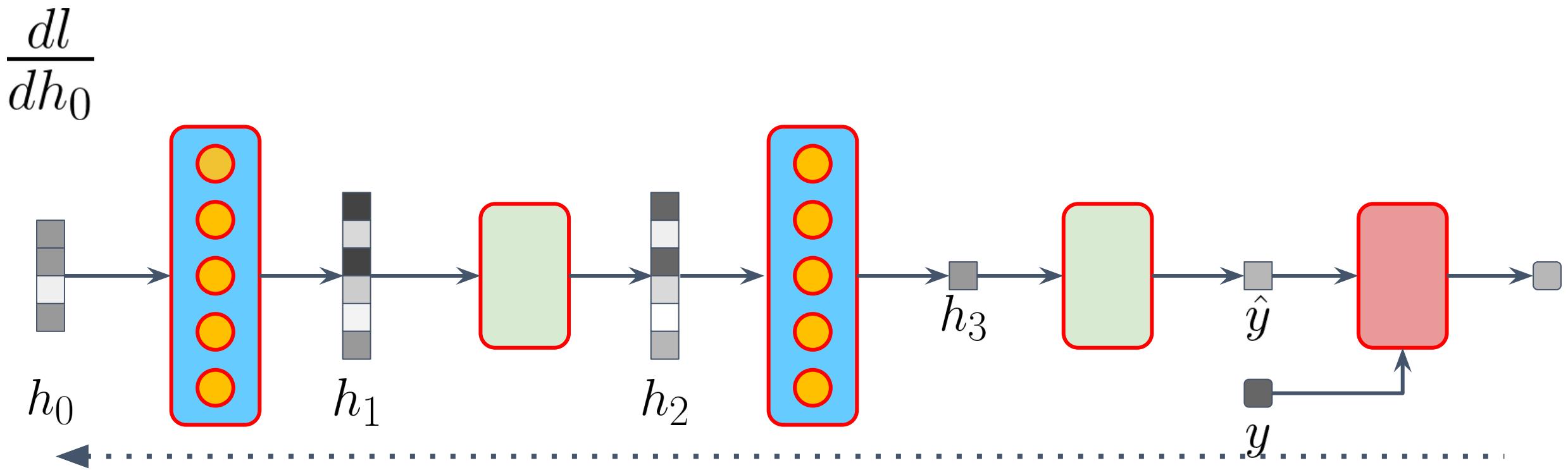
Backward

$$\frac{dh_1}{dh_0} \frac{dl}{dh_1}$$




Backward

$$\frac{dl}{dh_0}$$




Backpropagation: efficient computation

- Backward: efficient calculation of the chain rule (matrix version)
- Optimization: Jacobian*gradients
- Several examples can be stacked

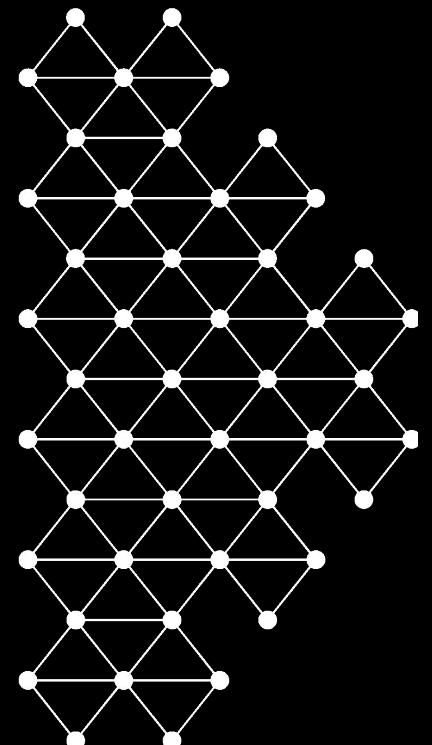
Implementation

- Forward(x): data transformation
- Backward(g): gradient transformation (output \rightarrow input)
- Update(g): calculation of the parameter gradient

FR: $R = \frac{3}{8\pi} H^2 \approx 10^{-123}$ $r = \sum r_i$ $\frac{\partial r}{\partial p_m} = \lambda^m [H(\cos \frac{p_m}{\lambda})] = \left[\frac{3}{2} \cos \frac{p_m}{\lambda} \right] \lambda^{-1} \approx \lambda^{-1}$
 $\rho = \frac{E}{V} = \frac{ER}{V} \approx \frac{1}{4\pi}$ $\frac{\partial \rho}{\partial p_m} = \lambda^{-1} \frac{\partial E}{\partial p_m} = \lambda^{-1} \frac{\partial (R \sin \frac{p_m}{\lambda})}{\partial p_m} = \lambda^{-1} R \cos \frac{p_m}{\lambda}$
 $\rho = \frac{E}{V} = \frac{ER}{V} = \frac{3R}{8\pi R^2} = \frac{3}{8\pi R^2} = \frac{3}{8\pi} H^2$ $\frac{\partial \rho}{\partial H} = \frac{3}{8\pi} \frac{2H}{R^2} = \frac{3H}{8\pi R^2} = \frac{3H}{V} = \frac{3}{V}$

Automatic differentiation





Applications of backpropagation

Understanding parameter initialization

$$z = Wx + b$$

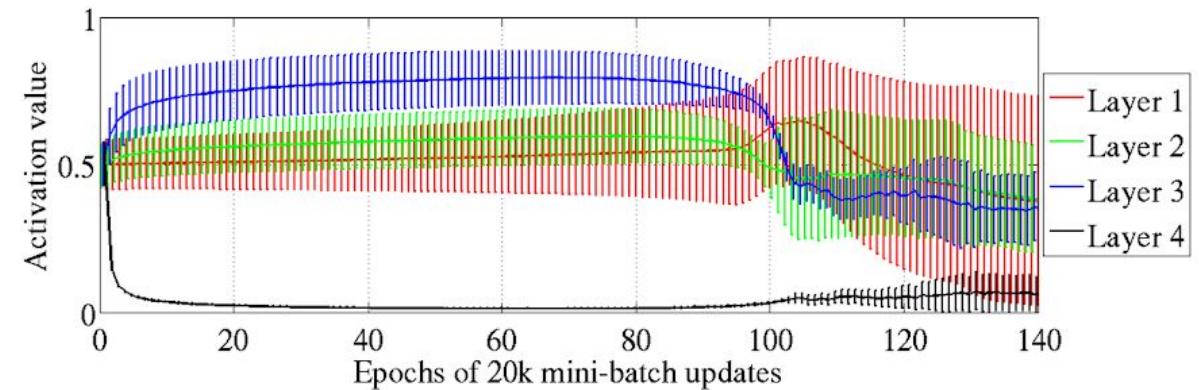
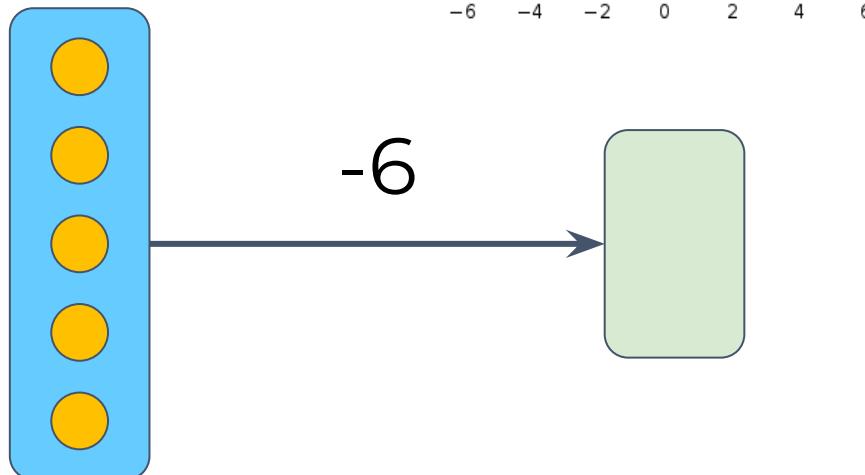


Figure 2: Mean and standard deviation (vertical bars) of the activation values (output of the sigmoid) during supervised learning, for the different hidden layers of a deep architecture. The top hidden layer quickly saturates at 0 (slowing down all learning), but then slowly desaturates around epoch 100.

Glorot, Xavier, and Yoshua Bengio. "Understanding the difficulty of training deep feedforward neural networks." Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics. 2010.

Adversarial example



x
“panda”
57.7% confidence

$$+ .007 \times$$



$\text{sign}(\nabla_x J(\theta, x, y))$
“nematode”
8.2% confidence

=



$x +$
 $\epsilon \text{sign}(\nabla_x J(\theta, x, y))$
“gibbon”
99.3 % confidence

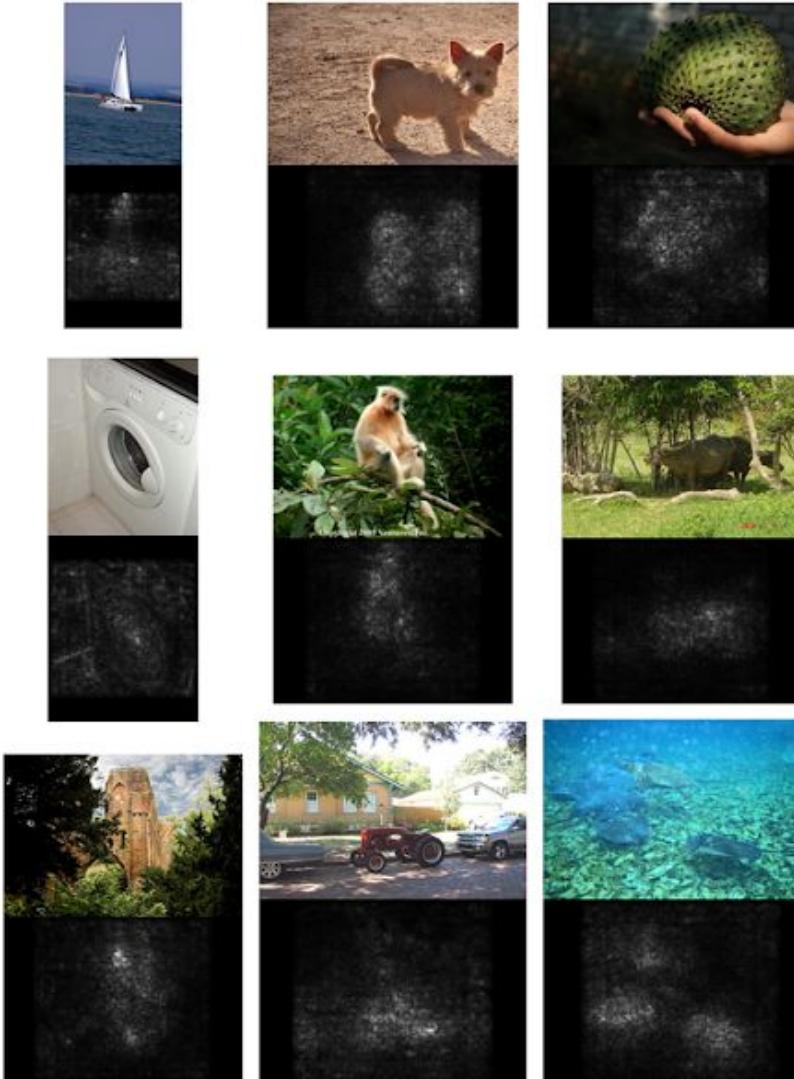
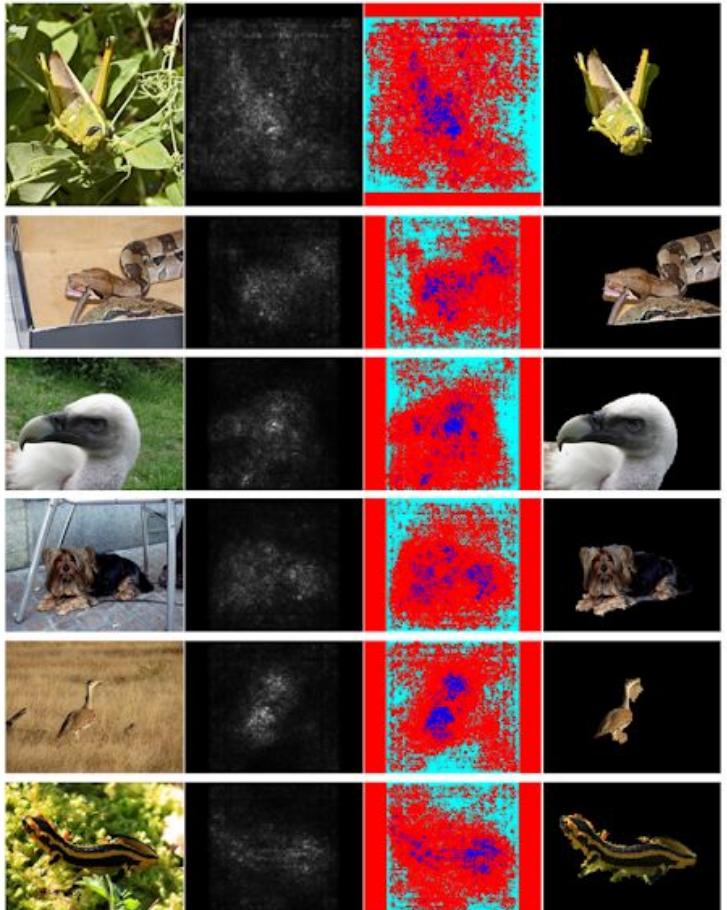
Goodfellow, Ian J., Jonathon Shlens, and Christian Szegedy. "Explaining and harnessing adversarial examples (2014)." arXiv :1412.6572.

Deepdream



<https://research.googleblog.com/2015/06/inceptionism-going-deeper-into-neural.html>

Saliency map



Simonyan, Karen, Andrea Vedaldi, and Andrew Zisserman. "Deep inside convolutional networks: Visualising image classification models and saliency maps." arXiv:1312.6034 (2013).

Erhan, Dumitru, et al. "Visualizing higher-layer features of a deep network." University of Montreal 1341.3 (2009).

Take-home message

- We can compute the linear approximation of a function represented as a computational graph.
- Backpropagation is a dynamic programming algorithm that computes the gradient efficiently.
- Backpropagation performs a forward and a backward pass.

References

- [Hugo Larochelle courses](#)
- [Colah's blog on calculus on Computational graph](#)
- [Michael Nielsen's chapter on backpropagation](#)
- [Feature Visualization in Distill Journal](#)
- [3BLUE1BROWN SERIES S3-E3 What is backpropagation really doing?](#)