

CS551: Introduction to Deep Learning

Backpropagation



Arijit Mondal

Dept. of Computer Science & Engineering

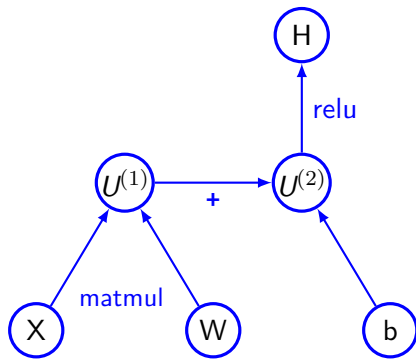
Indian Institute of Technology Patna

arijit@iitp.ac.in

Back propagation

- In a feedforward network, an input x is read and produces an output \hat{y}
 - This is forward propagation
- During training forward propagation continues until it produces cost $J(\theta)$
- Back-propagation algorithm allows the information to flow backward in the network to compute the gradient
- Computation of analytical expression for gradient is easy
- We need to find out gradient of the cost function with respect to the parameters ie. $\nabla_{\theta} J(\theta)$

Computational graph

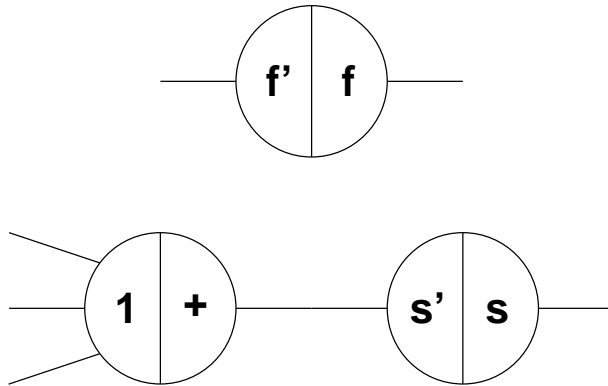


Chain rule of calculus

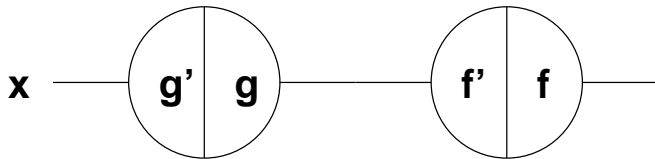
- Back-propagation algorithm heavily depends on it
- Let x be a real number and $y = g(x)$ and $z = f(g(x)) = f(y)$
- Chain rule says $\frac{dz}{dx} = \frac{dz}{dy} \frac{dy}{dx}$
- This can be generalized: Let $x \in \mathbb{R}^m$, $y \in \mathbb{R}^n$, $g: \mathbb{R}^m \rightarrow \mathbb{R}^n$ and $f: \mathbb{R}^n \rightarrow \mathbb{R}$ and $y = g(x)$ and $z = f(y)$ then $\frac{\partial z}{\partial x_i} = \sum_j \frac{\partial z}{\partial y_j} \frac{\partial y_j}{\partial x_i}$
- In vector notation it will be where $\frac{\partial y}{\partial x}$ is the $n \times m$ Jacobian matrix of g

$$\nabla_x z = \left(\frac{\partial y}{\partial x} \right)^T \nabla_y z$$

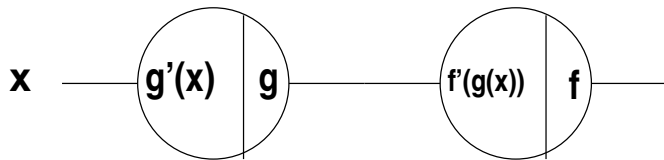
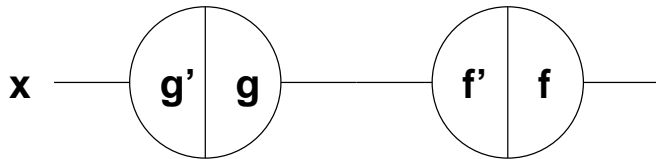
Back propagation - 1



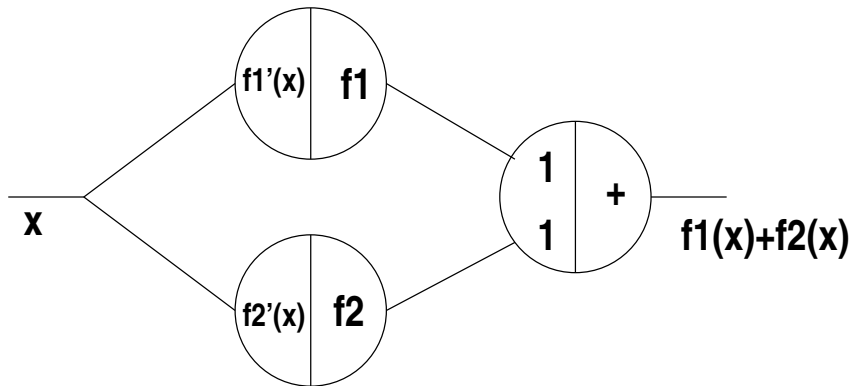
Back propagation - 2



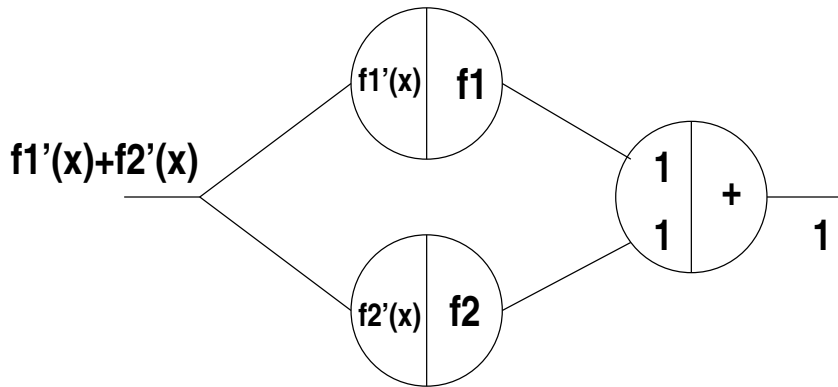
Back propagation - 3



Back propagation - 4



Back propagation - 5

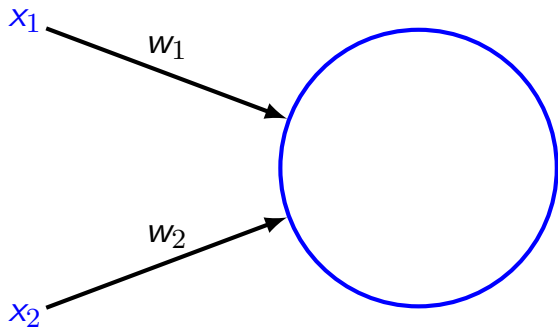


Back propagation - 6

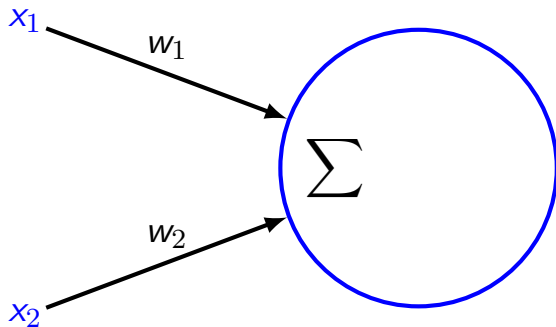
x_1

x_2

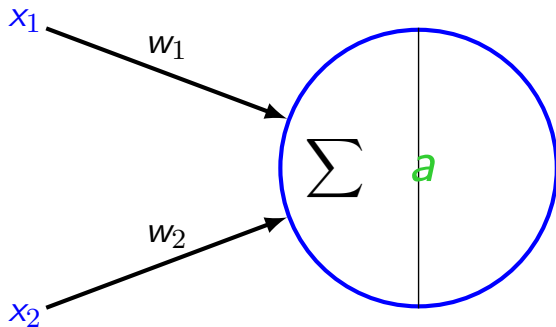
Back propagation - 6



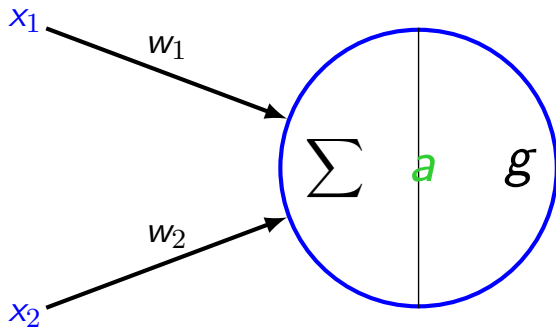
Back propagation - 6



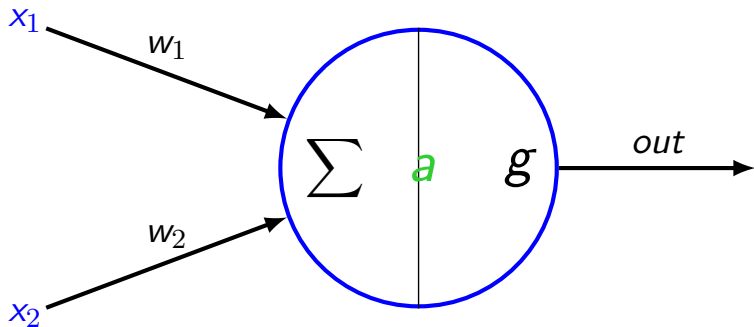
Back propagation - 6



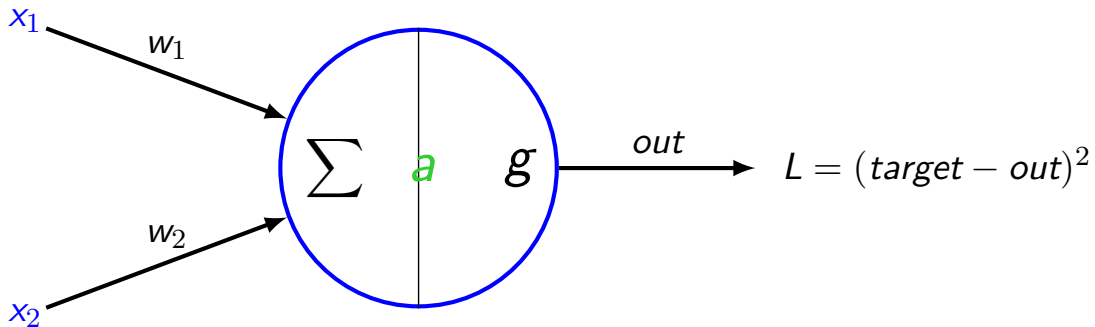
Back propagation - 6



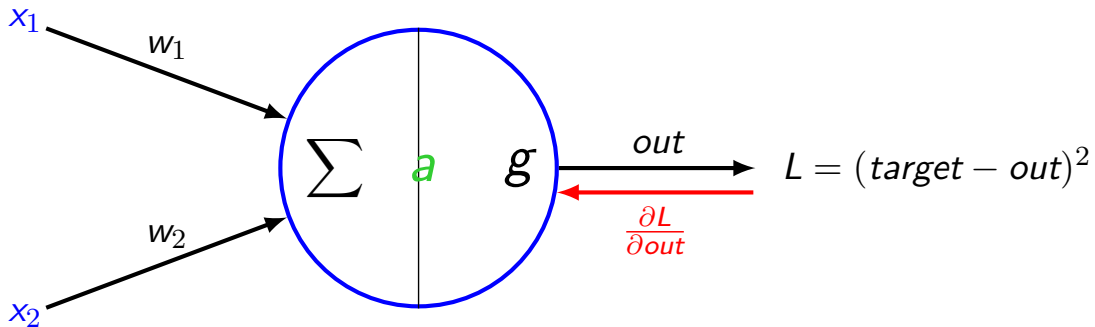
Back propagation - 6



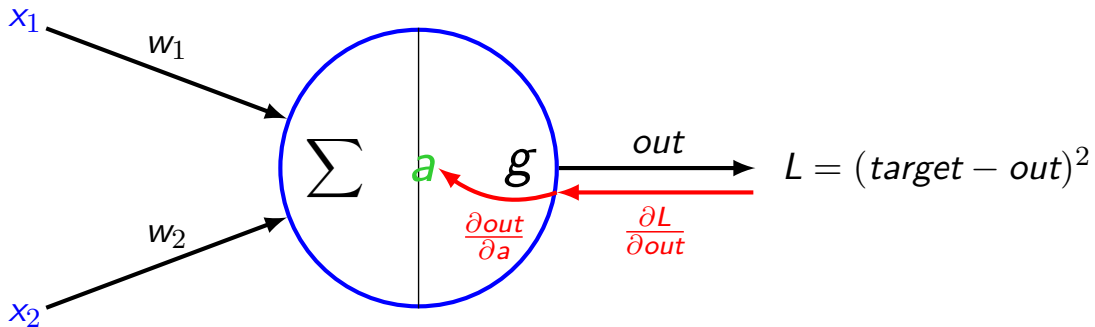
Back propagation - 6



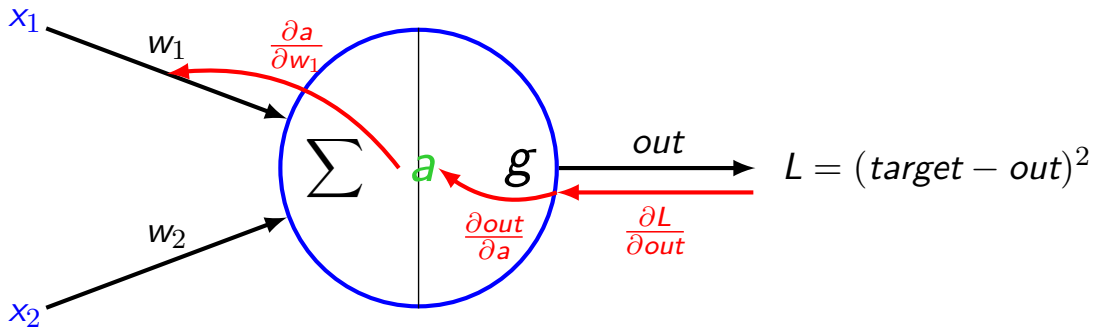
Back propagation - 6



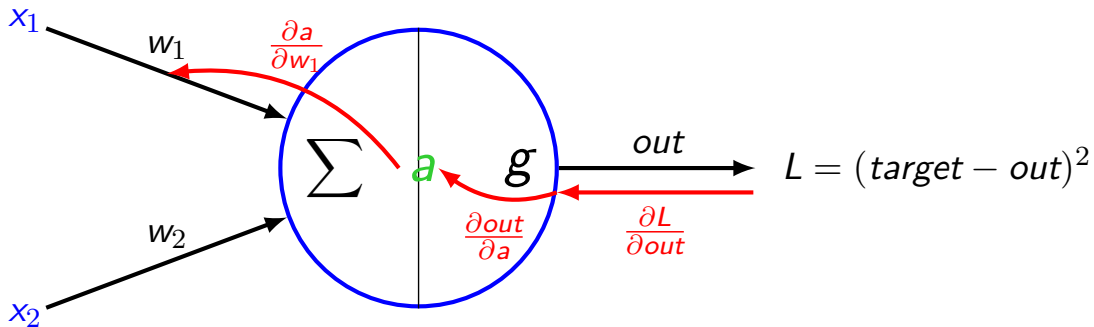
Back propagation - 6



Back propagation - 6



Back propagation - 6

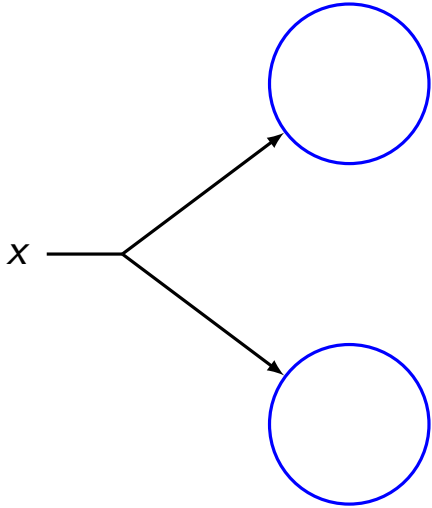


$$\frac{\partial L}{\partial w_1} = \frac{\partial L}{\partial out} \frac{\partial out}{\partial a} \frac{\partial a}{\partial w_1}$$

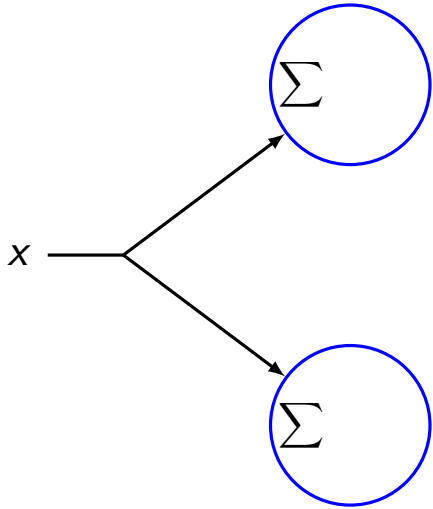
Back propagation - 7

X —

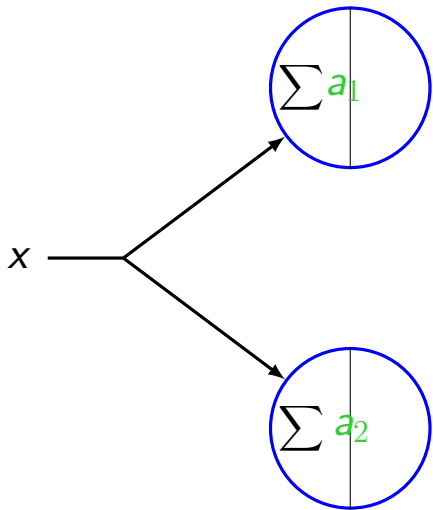
Back propagation - 7



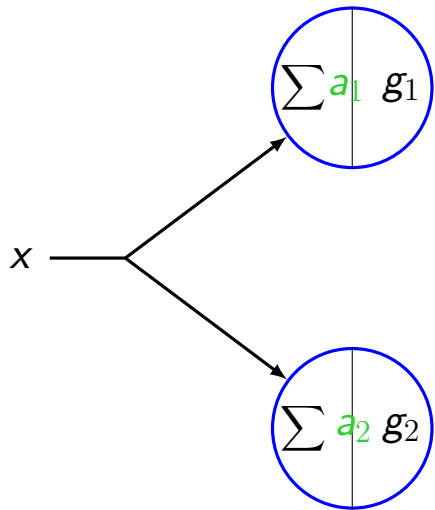
Back propagation - 7



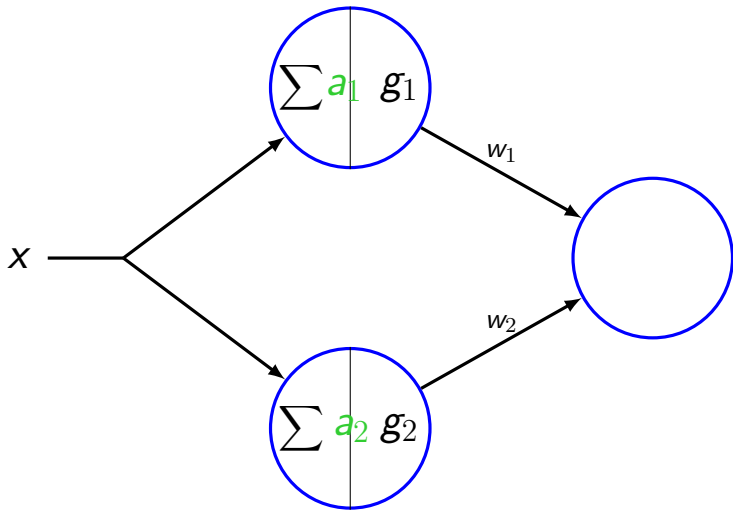
Back propagation - 7



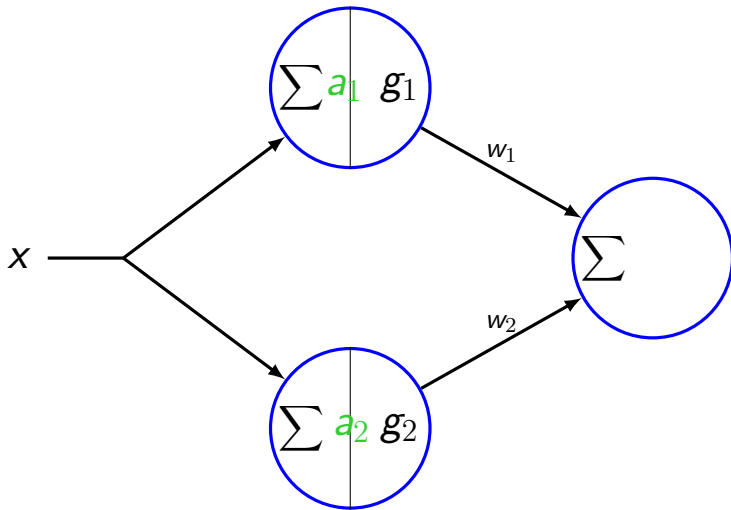
Back propagation - 7



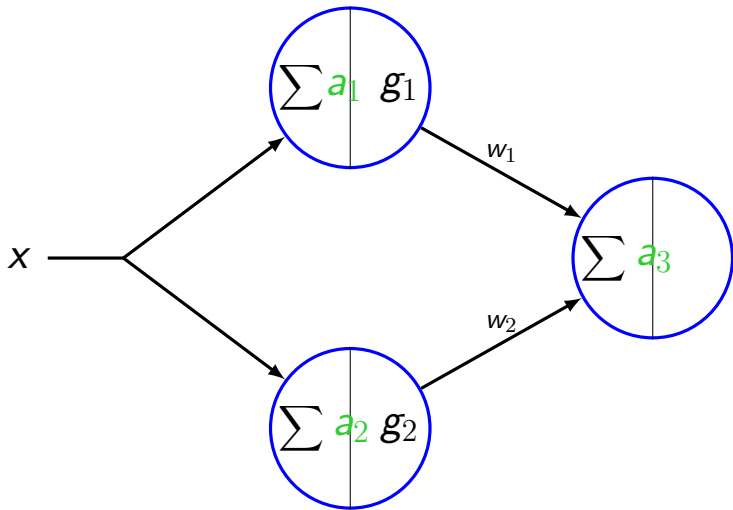
Back propagation - 7



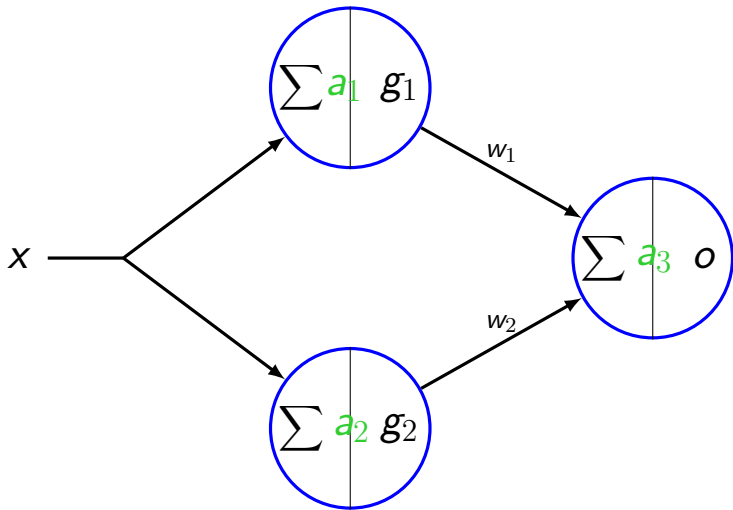
Back propagation - 7



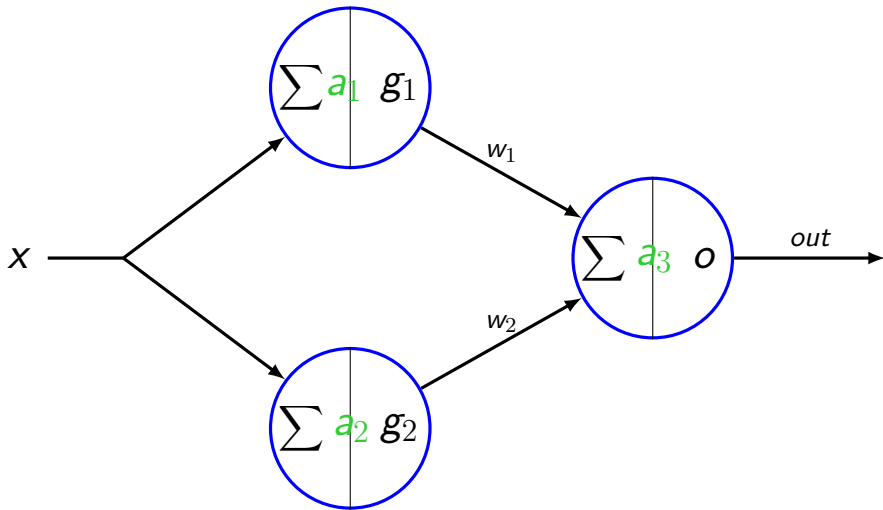
Back propagation - 7



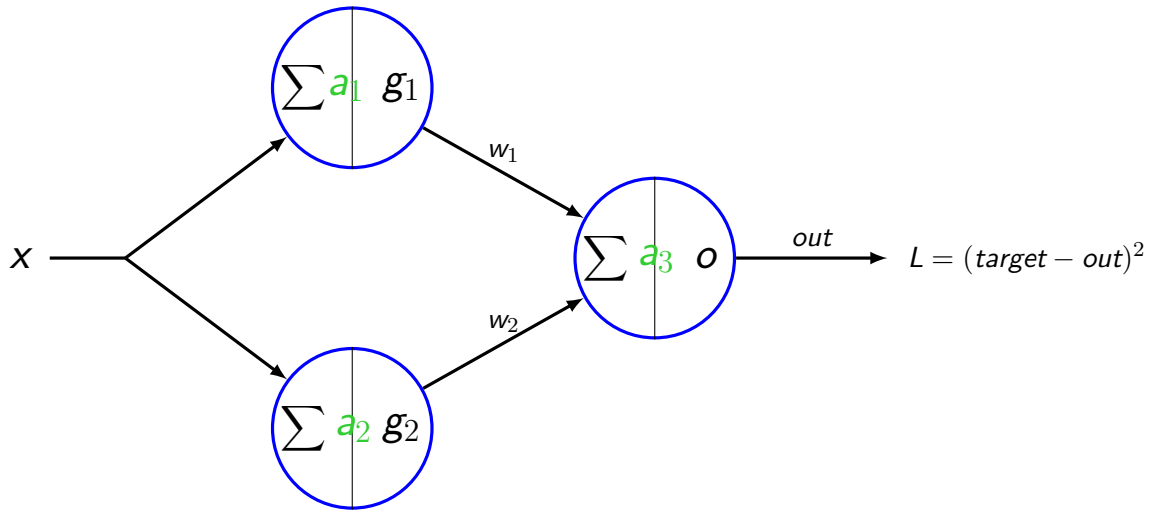
Back propagation - 7



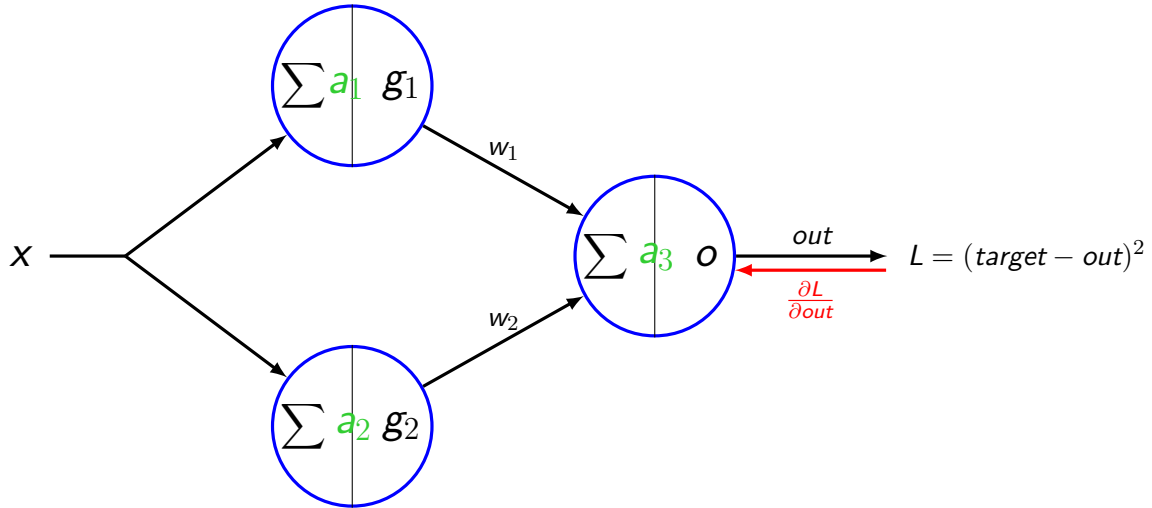
Back propagation - 7



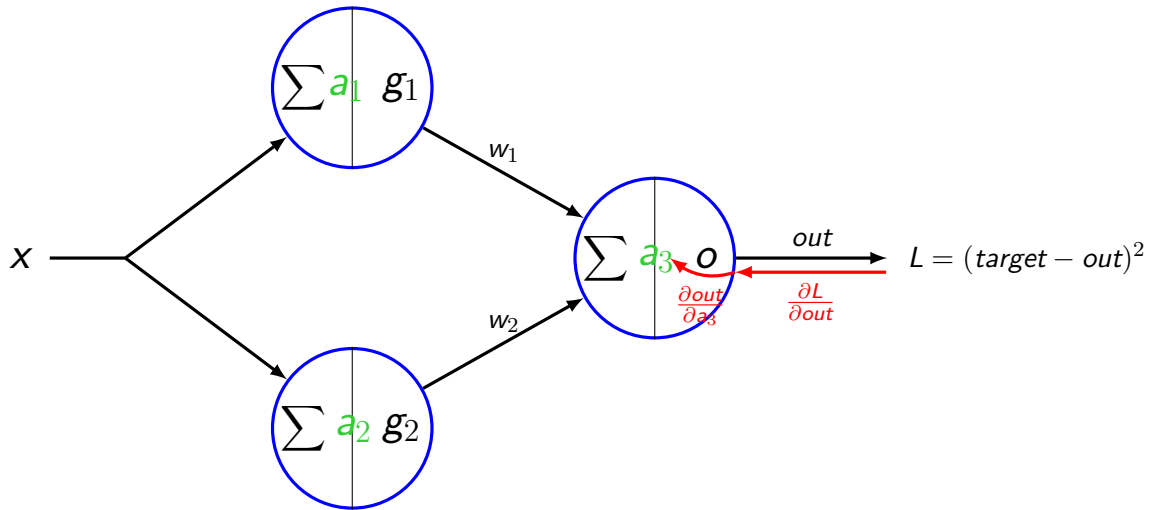
Back propagation - 7



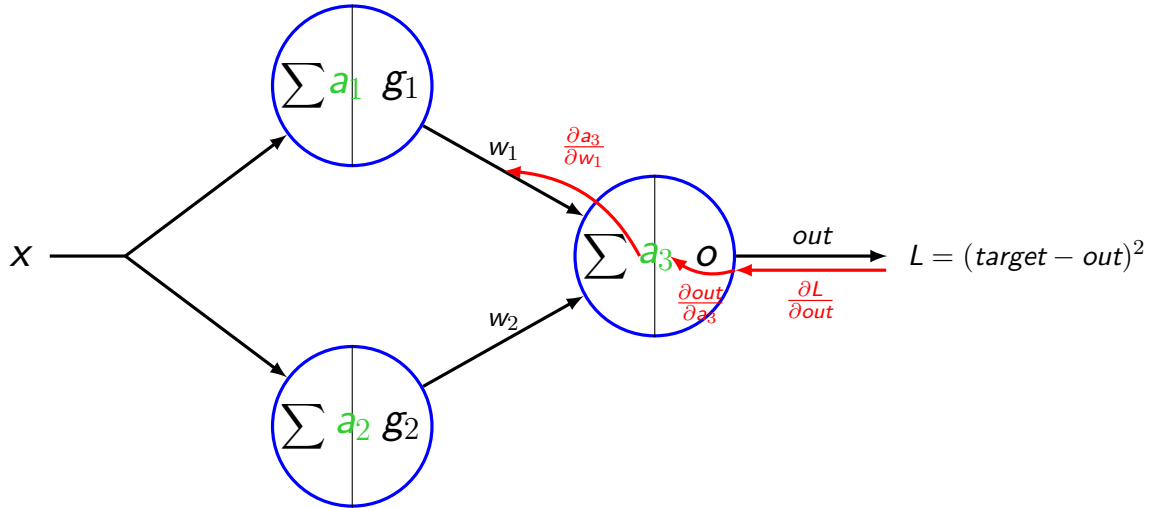
Back propagation - 7



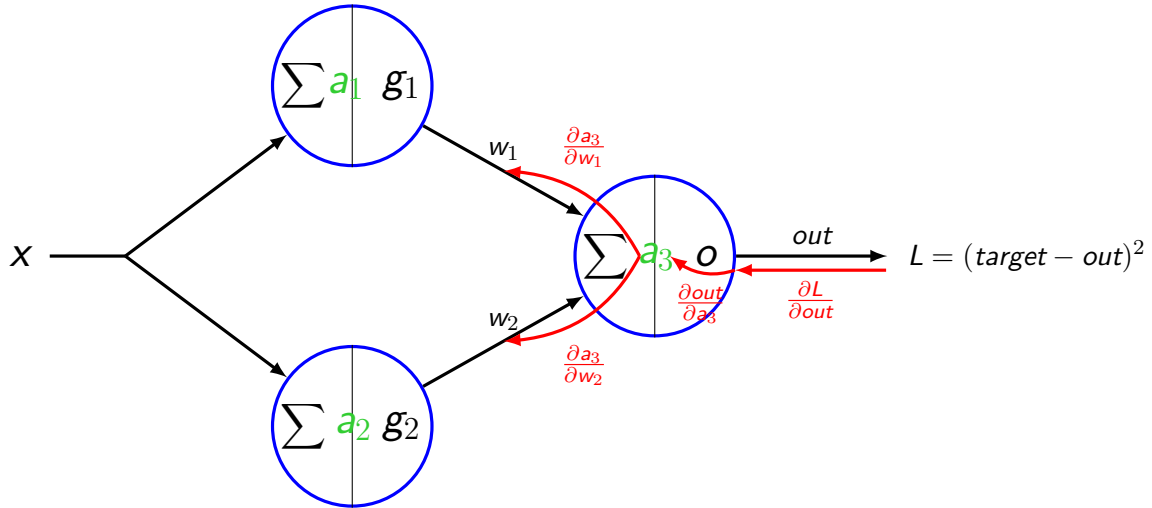
Back propagation - 7



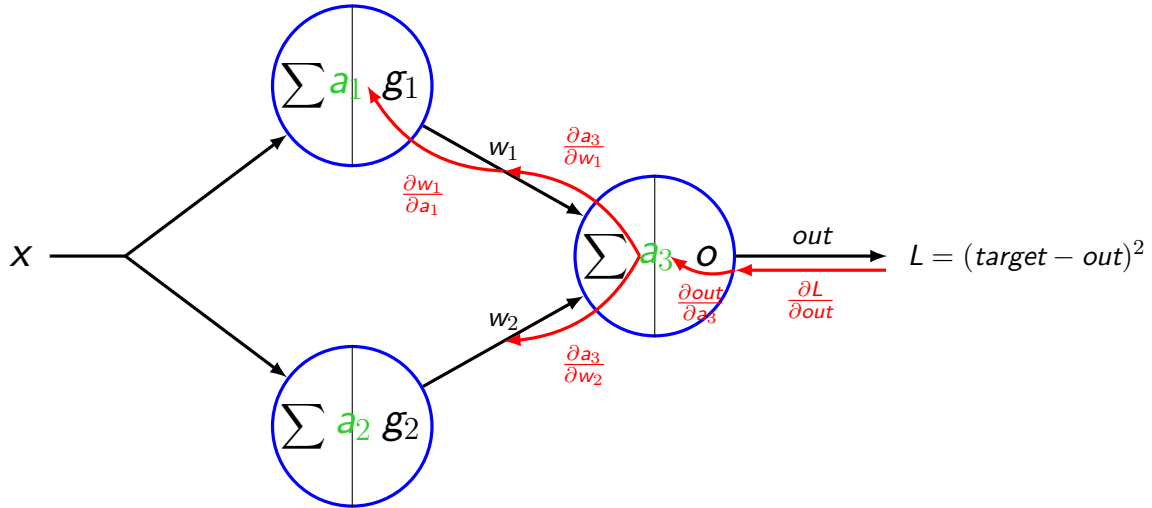
Back propagation - 7



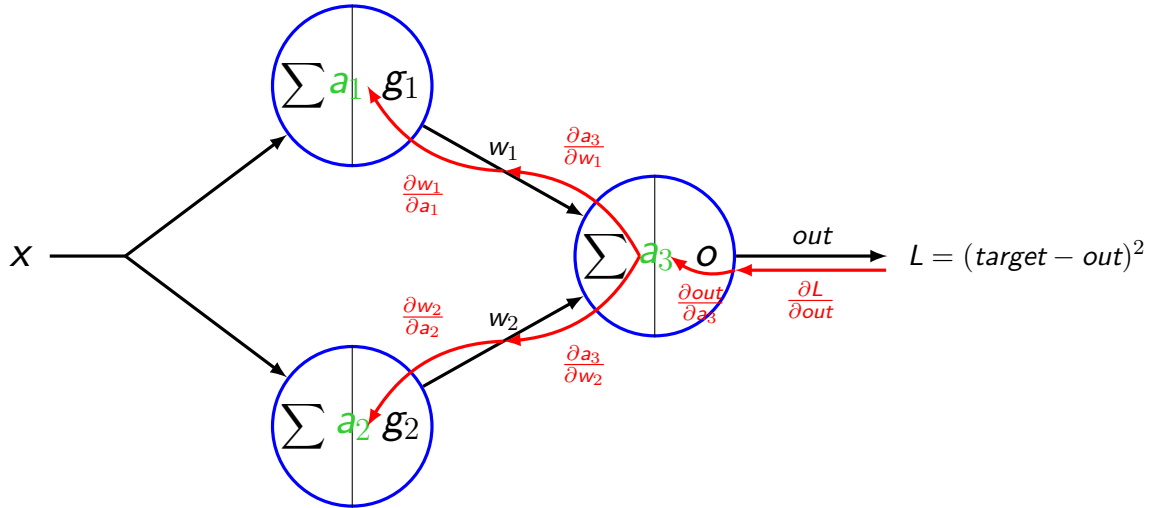
Back propagation - 7



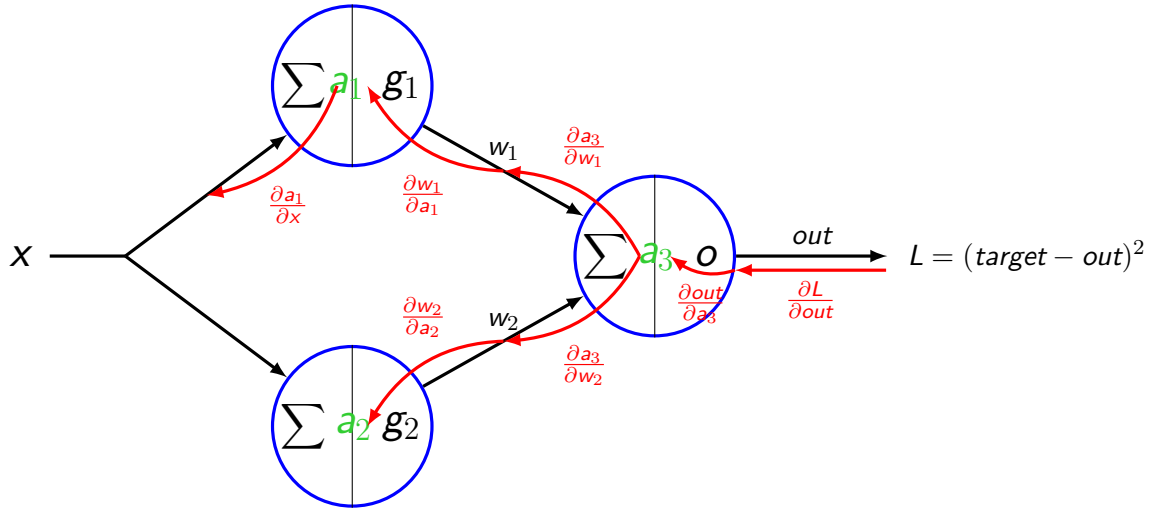
Back propagation - 7



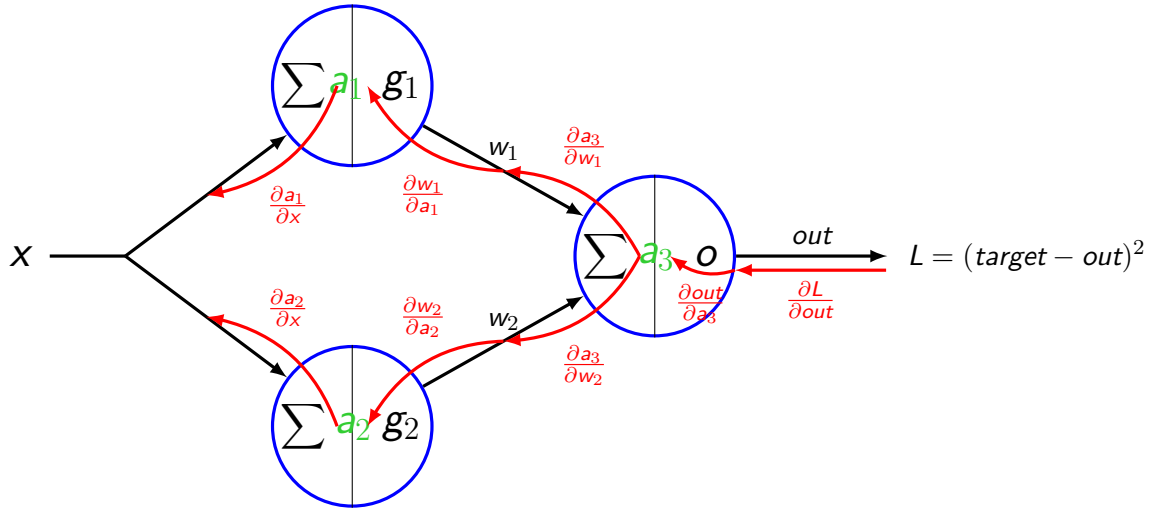
Back propagation - 7



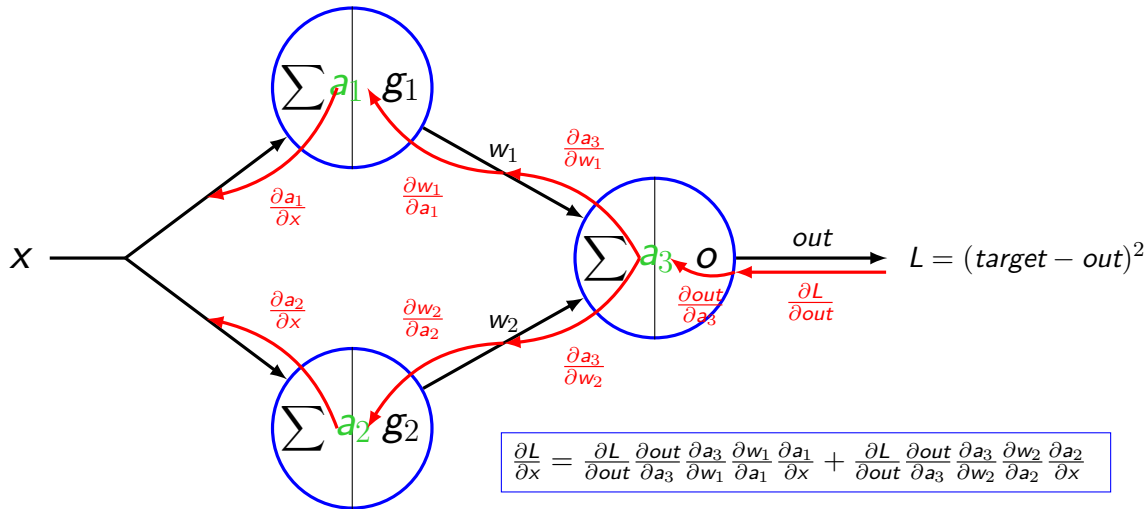
Back propagation - 7



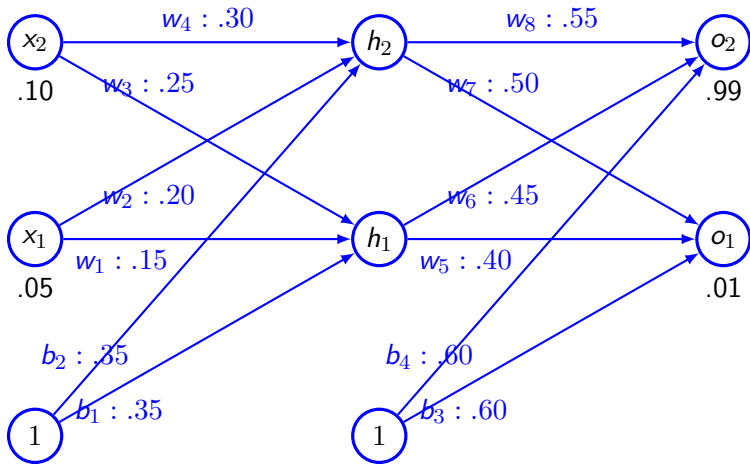
Back propagation - 7



Back propagation - 7

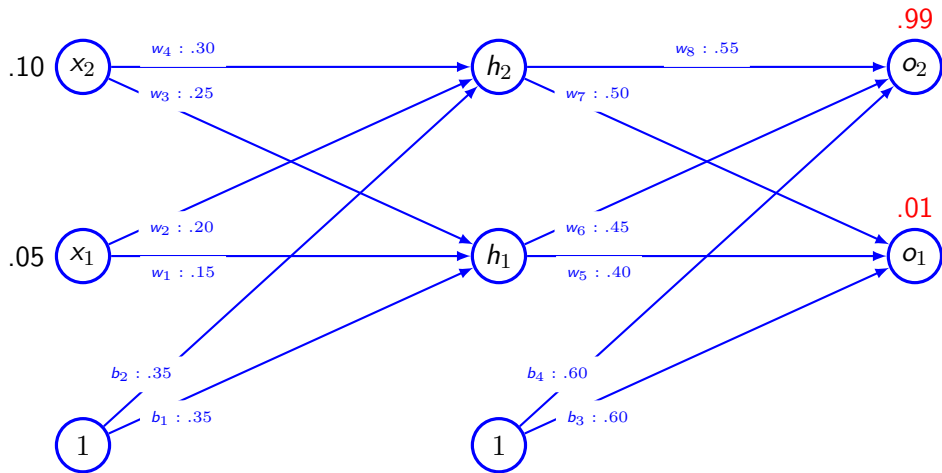


Example

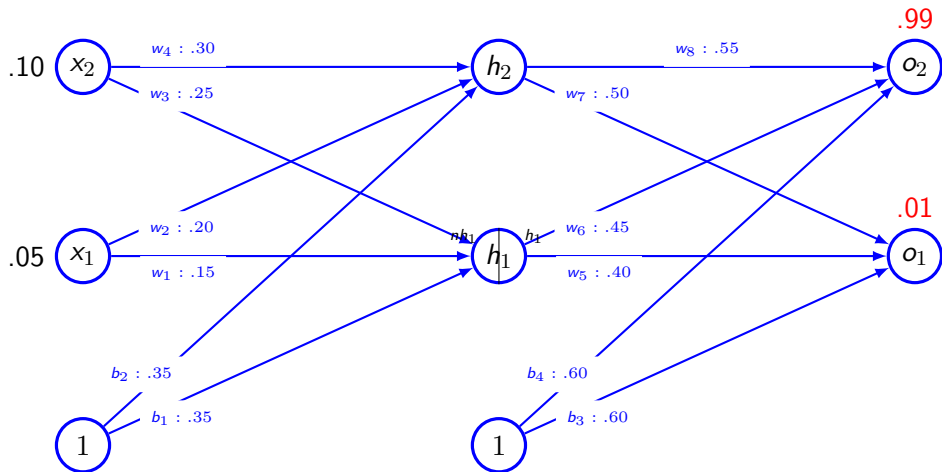


Hidden and output layer have sigmoid activation function. Loss function - MSE.

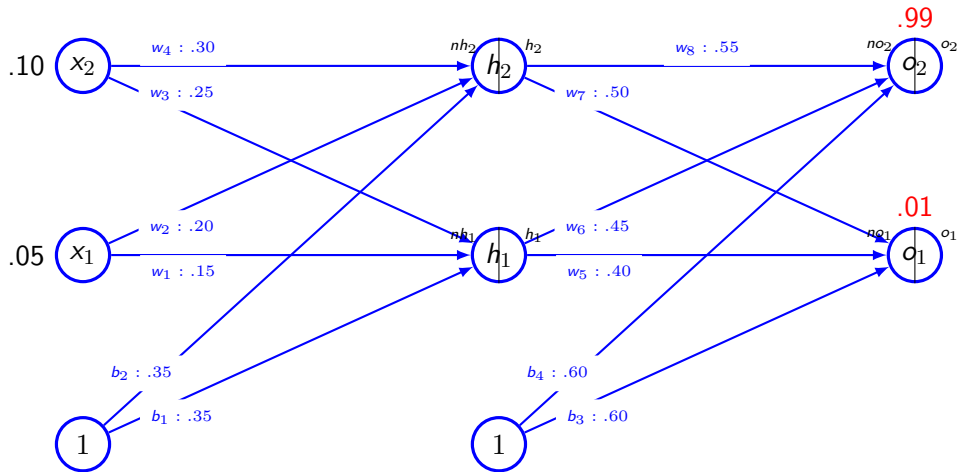
Example



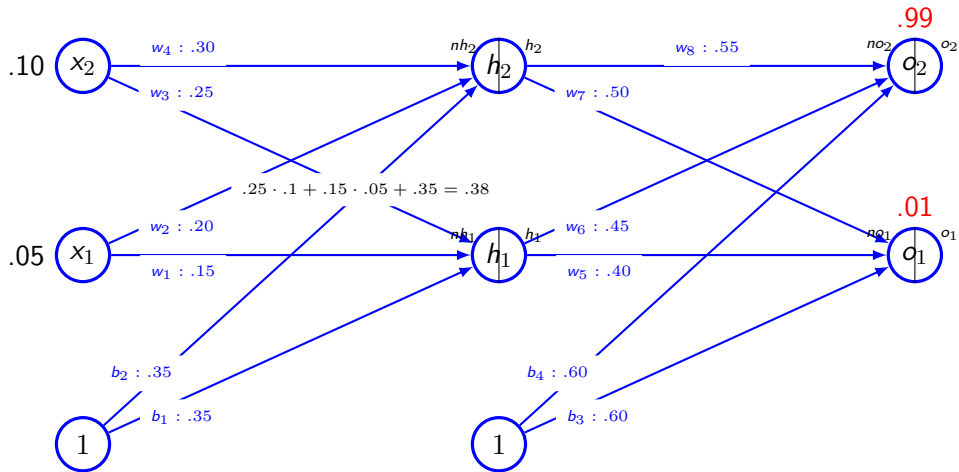
Example



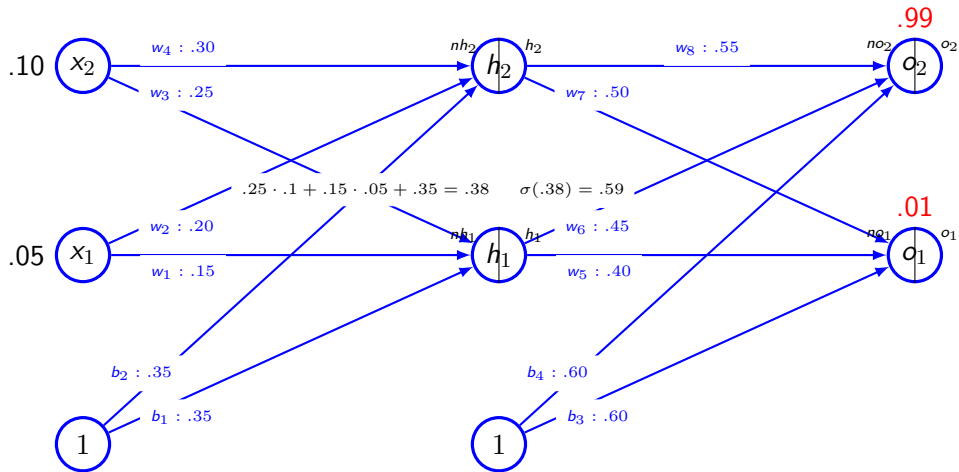
Example



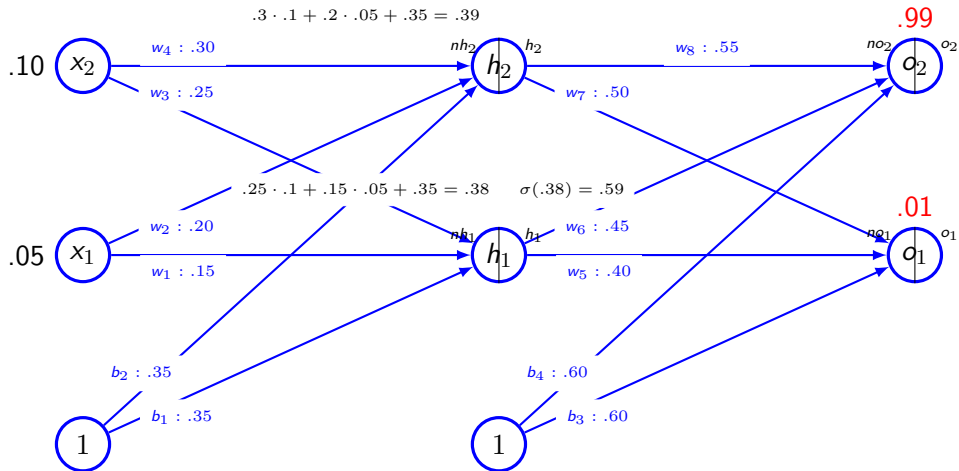
Example



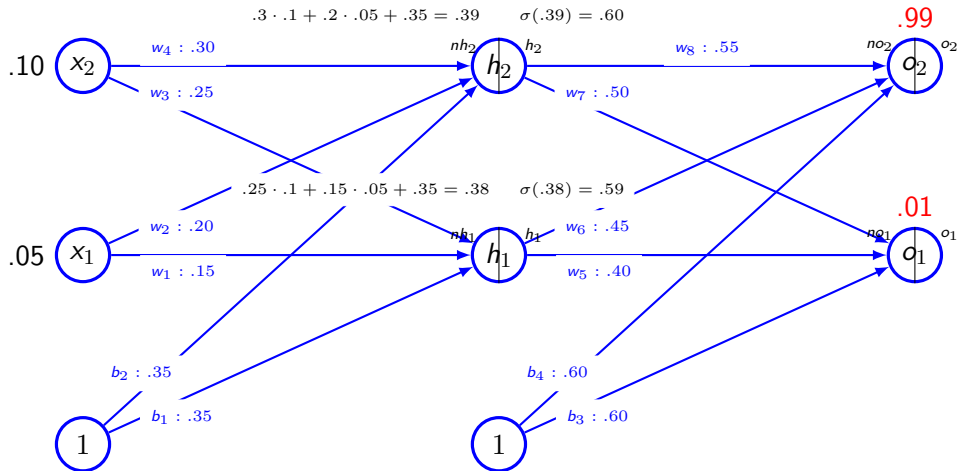
Example



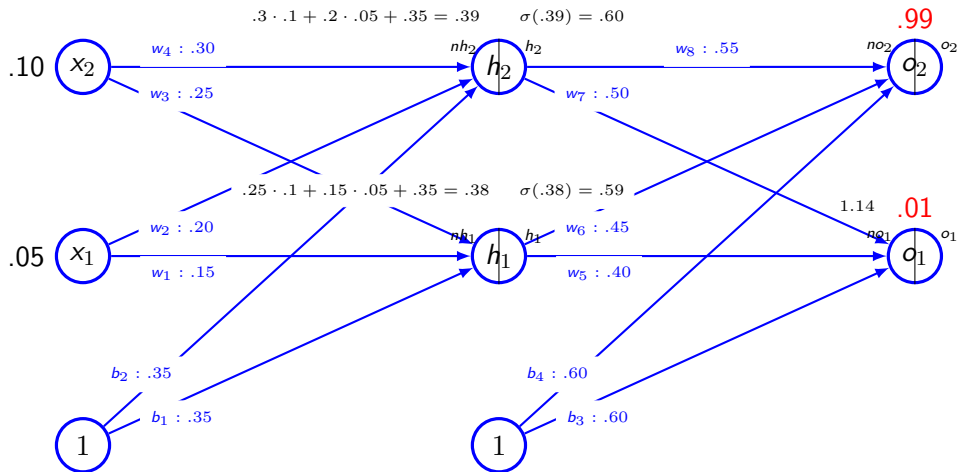
Example



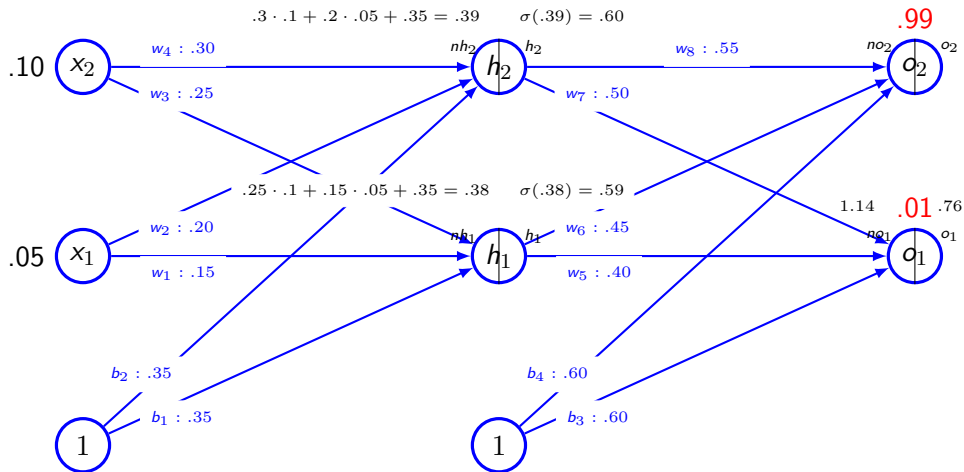
Example



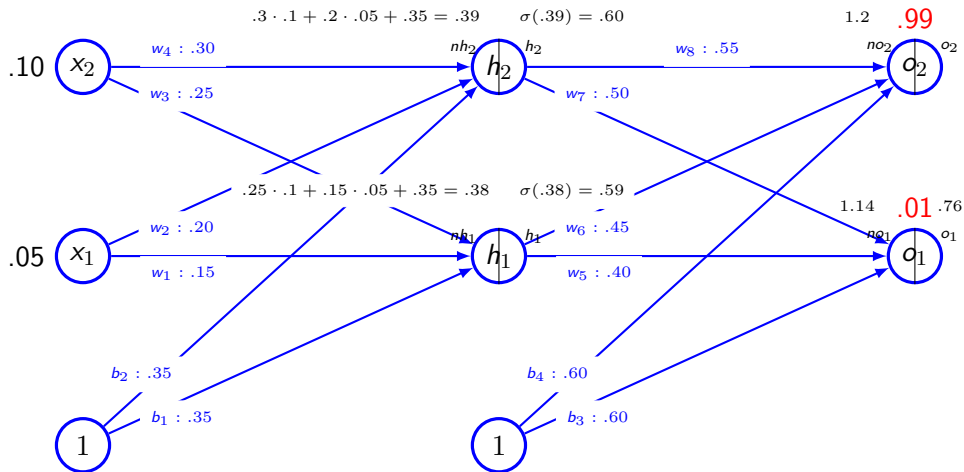
Example



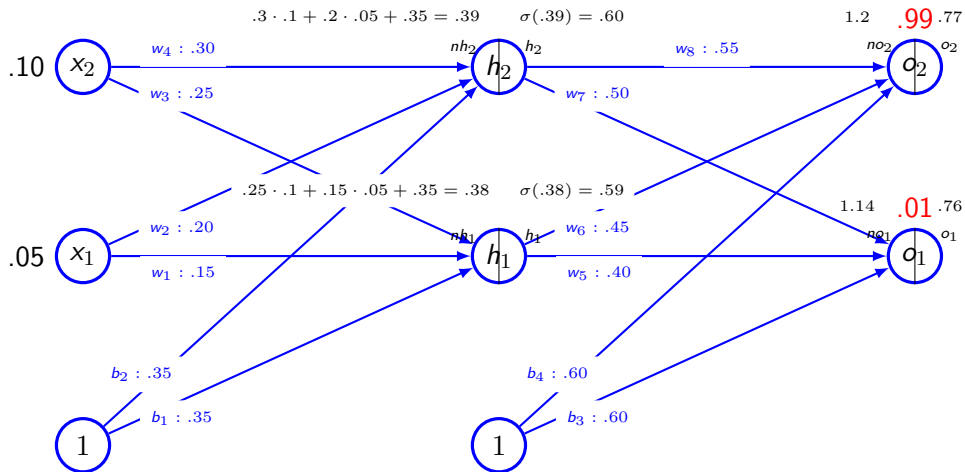
Example



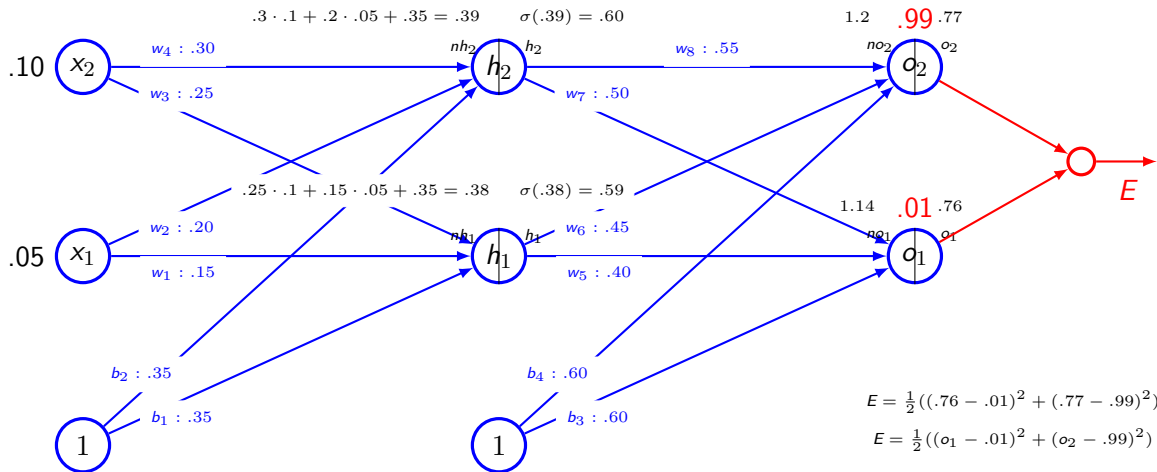
Example



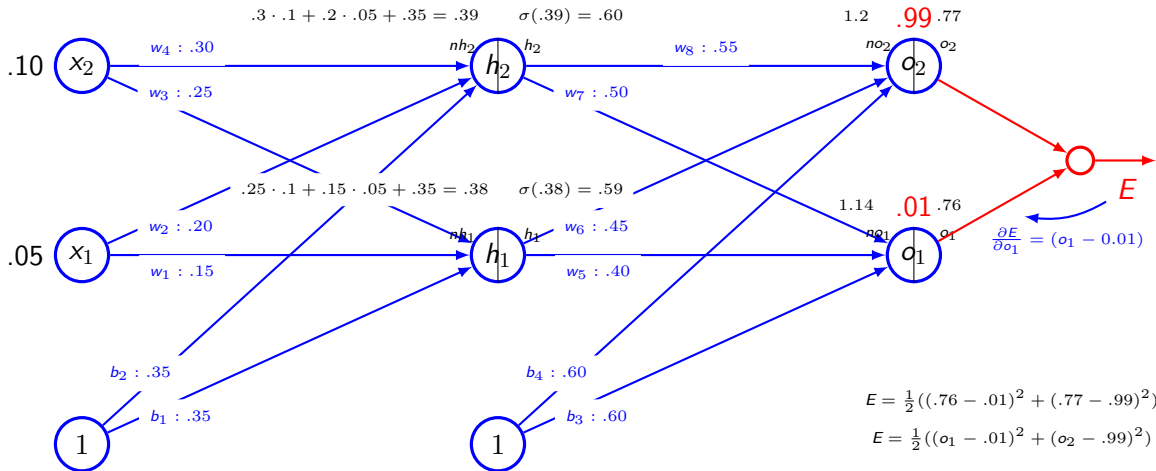
Example



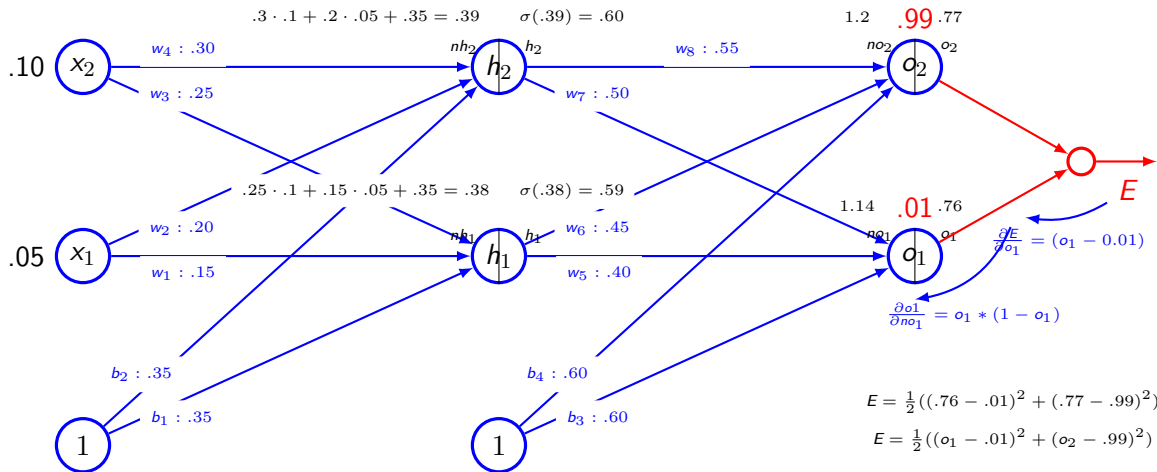
Example



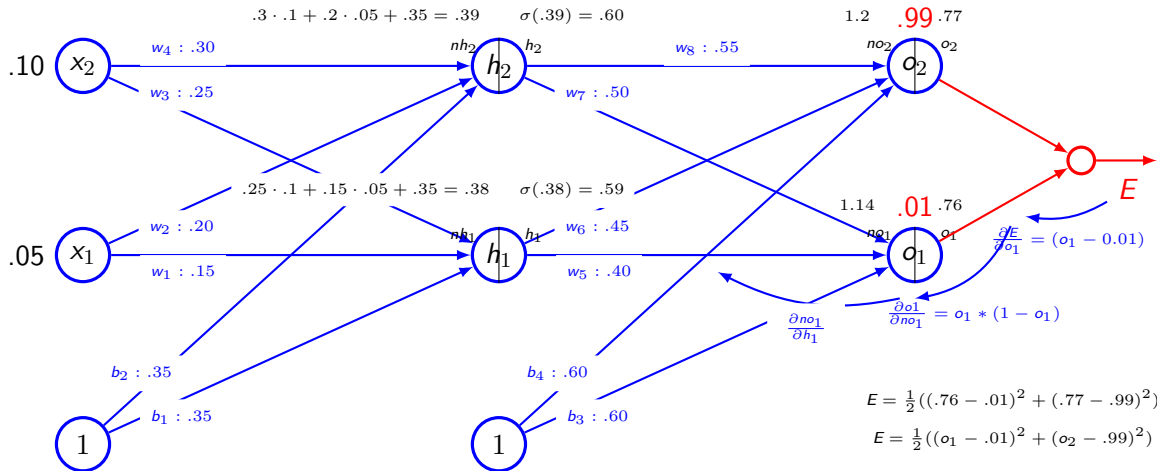
Example



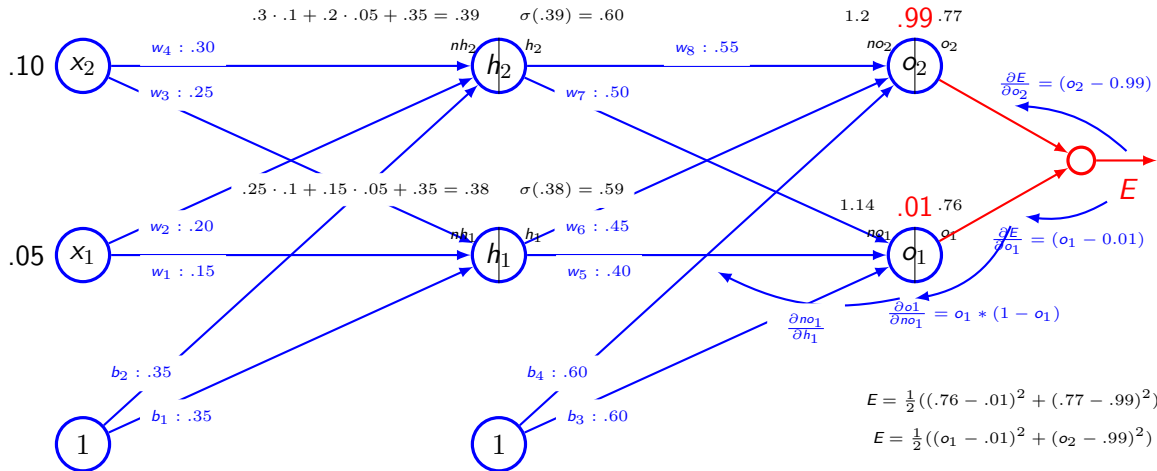
Example



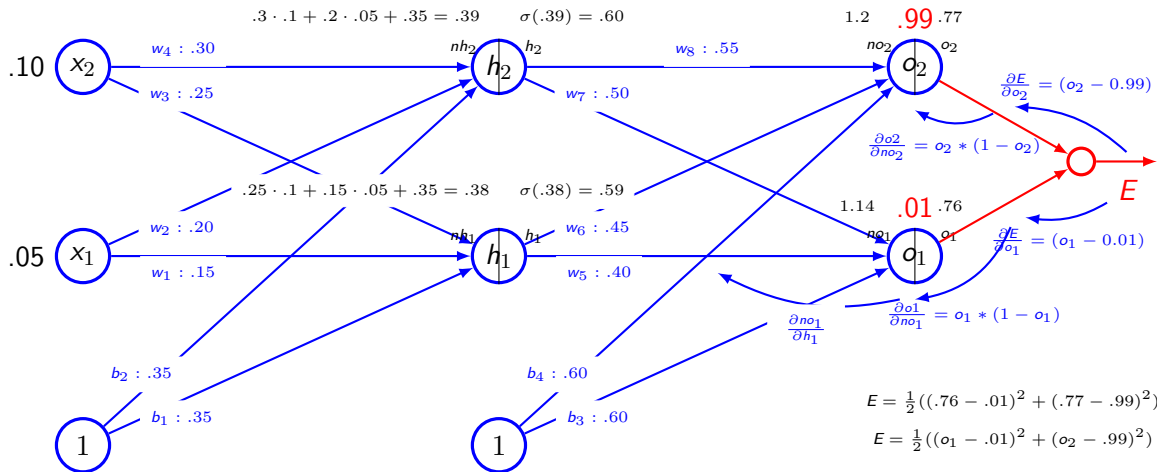
Example



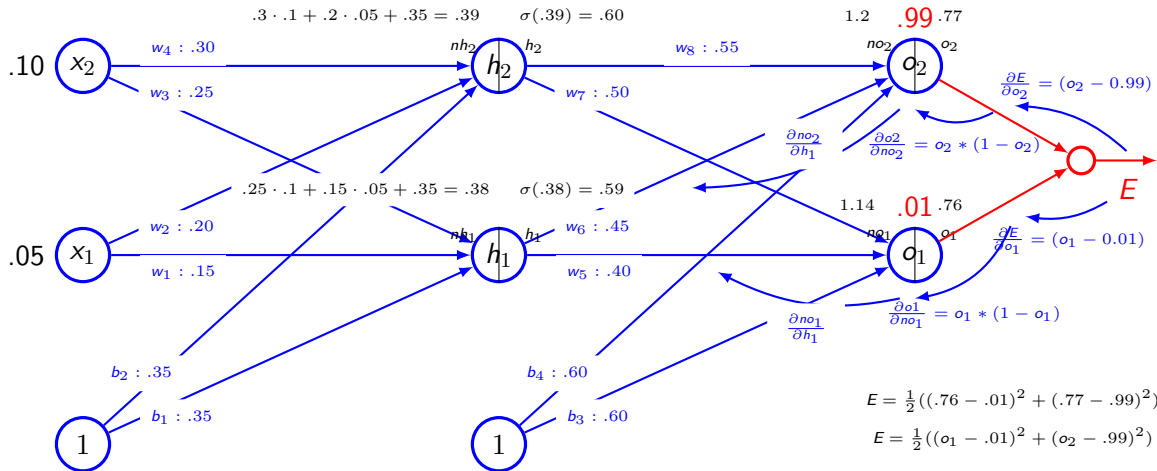
Example



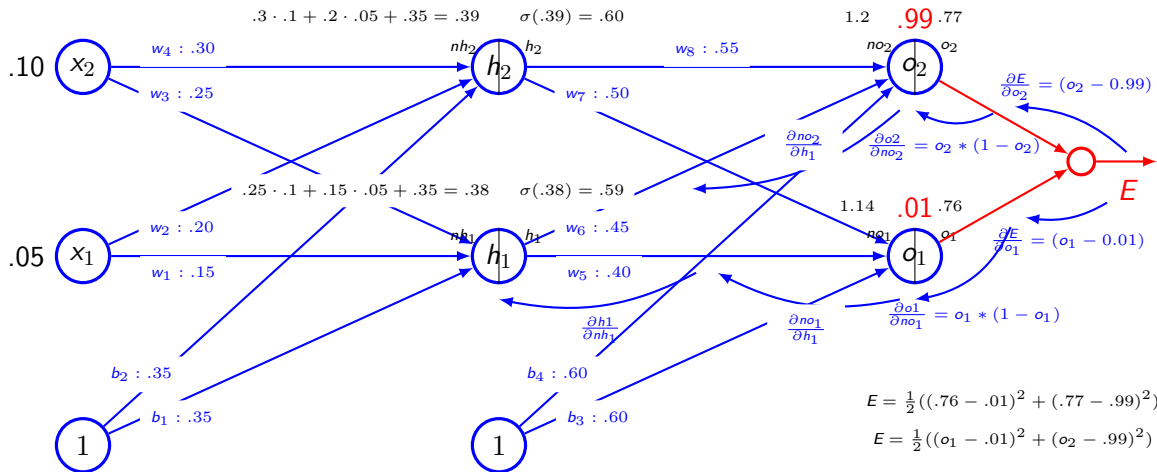
Example



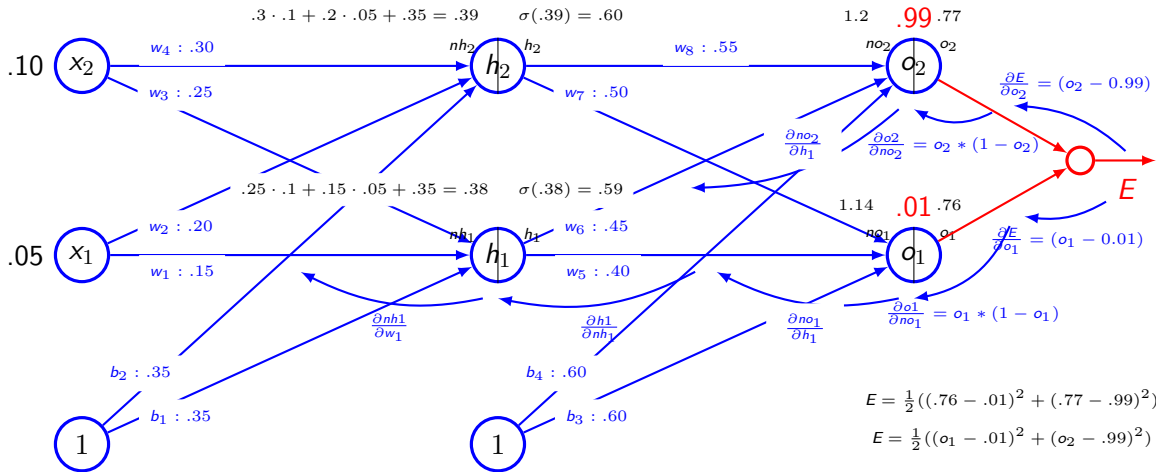
Example



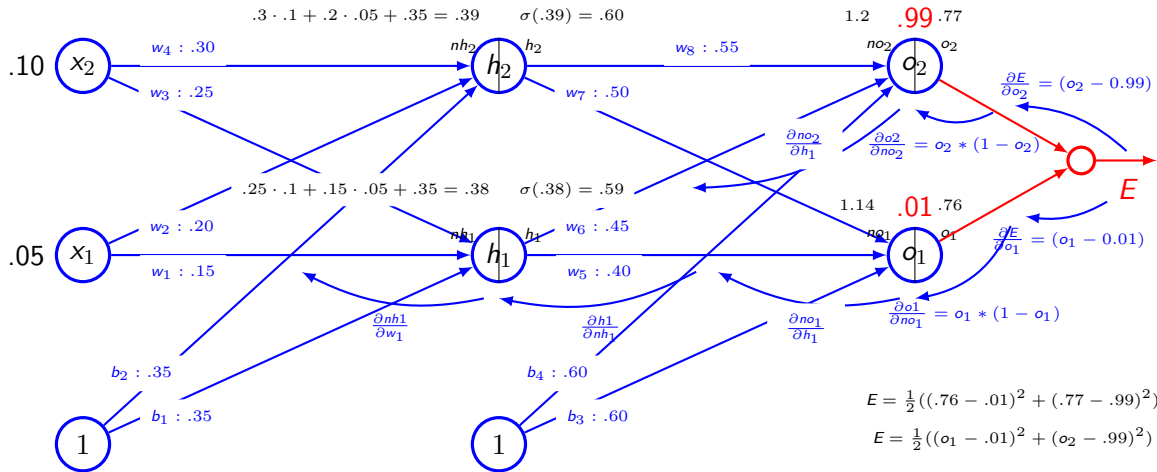
Example



Example



Example



$$\frac{\partial E}{\partial w_1} = \left(\frac{\partial E}{\partial o_1} \times \frac{\partial o_1}{\partial no_1} \times \frac{\partial no_1}{\partial h_1} + \frac{\partial E}{\partial o_2} \times \frac{\partial o_2}{\partial no_2} \times \frac{\partial no_2}{\partial h_1} \right) \times \frac{\partial h_1}{\partial nh_1} \times \frac{\partial nh_1}{\partial w_1}$$

Application of chain rule

- Let us consider $u^{(n)}$ be the loss quantity. Need to find out the gradient for this.
- Let $u^{(1)}$ to $u^{(n_i)}$ are the inputs
- Therefore, we wish to compute $\frac{\partial u^{(n)}}{\partial u^{(i)}}$ where $i = 1, 2, \dots, n_i$
- Let us assume the nodes are ordered so that we can compute one after another
- Each $u^{(i)}$ is associated with an operation $f^{(i)}$ ie. $u^{(i)} = f(\mathbb{A}^{(i)})$

Algorithm for forward pass

for $i = 1, \dots, n_i$ **do**

$$u^{(i)} \leftarrow x_i$$

end for

for $i = n_i + 1, \dots, n$ **do**

$$\mathbb{A}^{(i)} \leftarrow \{u^{(j)} \mid j \in Pa(u^{(i)})\}$$

$$u^{(i)} \leftarrow f^{(i)}(\mathbb{A}^{(i)})$$

end for

return $u^{(n)}$

Algorithm for backward pass

grad_table[$u^{(n)}$] \leftarrow 1

for $j = n - 1$ down to 1 **do**

$$\text{grad_table}[u^{(j)}] \leftarrow \sum_{i: j \in \text{Pa}(u^{(i)})} \text{grad_table}[u^{(i)}] \frac{\partial u^{(i)}}{\partial u^{(j)}}$$

end for

return grad_table

Computational graph & subexpression

- We have $x = f(w)$, $y = f(x)$, $z = f(y)$

$$\frac{\partial z}{\partial w}$$

$$= \frac{\partial z}{\partial y} \frac{\partial y}{\partial x} \frac{\partial x}{\partial w}$$

$$= f'(y) f'(x) f'(w)$$

$$= f'(f(f(w))) f'(f(w)) f'(w)$$



Forward propagation in MLP

- Input
 - $\mathbf{h}^{(0)} = \mathbf{x}$
- Computation for each layer $k = 1, \dots, l$
 - $\mathbf{a}^{(k)} = \mathbf{b}^{(k)} + \mathbf{W}^{(k)}\mathbf{h}^{(k-1)}$
 - $\mathbf{h}^{(k)} = f(\mathbf{a}^{(k)})$
- Computation of output and loss function
 - $\hat{\mathbf{y}} = \mathbf{h}^{(l)}$
 - $J = L(\hat{\mathbf{y}}, \mathbf{y}) + \lambda\Omega(\theta)$

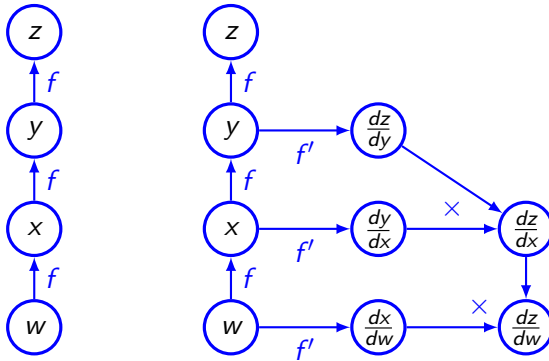
Backward computation in MLP

- Compute gradient at the output
 - $\mathbf{g} \leftarrow \nabla_{\hat{y}} J = \nabla_{\hat{y}} L(\hat{y}, y)$
- Convert the gradient at output layer into gradient of pre-activation
 - $\mathbf{g} \leftarrow \nabla_{\mathbf{a}^{(k)}} J = \mathbf{g} \odot f'(\mathbf{a}^{(k)})$
- Compute gradient on weights and biases
 - $\nabla_{\mathbf{b}^{(k)}} J = \mathbf{g} + \lambda \nabla_{\mathbf{b}^{(k)}} \Omega(\theta)$
 - $\nabla_{\mathbf{W}^{(k)}} J = \mathbf{g} \mathbf{h}^{(k-1)T} + \lambda \nabla_{\mathbf{W}^{(k)}} \Omega(\theta)$
- Propagate the gradients wrt the next lower level activation
 - $\mathbf{g} \leftarrow \nabla_{\mathbf{h}^{(k-1)}} J = \mathbf{W}^{(k)T} \mathbf{g}$

Computation of derivatives

- Takes a computational graph and a set of numerical values for the inputs, then return a set of numerical values
 - Symbol-to-number differentiation
 - Torch, Caffe
- Takes computational graph and add additional nodes to the graph that provide symbolic description of derivative
 - Symbol-to-symbol derivative
 - Theano, TensorFlow

Example



Summary

- Writing gradient for each parameter is difficult
- Recursive application of chain rule along the computational graph help to compute the gradients
- Forward pass - compute the value of the operations and store the necessary information
- Backward pass - uses the loss function, computes the gradient, updates the parameters.