ECBM E4040
Neural Networks and Deep Learning

Back-Propagation

Zoran Kostić
Columbia University
Electrical Engineering Department &
Data Sciences Institute

COLUMBIA | ENGINEERING
The Fu Foundation School of Engineering and Applied Science

References and Acknowledgments

- Deep Learning by Ian Goodfellow, Yoshua Bengio and Aaron Courville, http://www.deeplearningbook.org/ , chapter 6.
- Rumelhart, David E.; Hinton, Geoffrey E.; Williams, Ronald J. (8 October 1986). "Learning representations by back-propagating errors". Nature. 323 (6088): 533–536.
- Y. LeCun, et al., "Efficient BackProp", http://yann.lecun.com/exdb/publis/pdf/lecun-98b.pdf
- Lecture material by bionet group / Prof. Aurel Lazar (http://www.bionet.ee.columbia.edu/).

© ZK

Back-Propagation

Back-propagation is used to update coefficients of multi-layer neural networks.

coefficients = parameters = weights

Back-propagation is a method for computing gradients for multi-layer neural networks.

The method can easily be generalized to arbitrary families of parametrized functions and their corresponding computational graph.

© ZK

COLUMBIA | ENGINEERING
The Fu Foundation School of Engineering and Applied Science

Back-Propagation is Used for Gradient Computation

The term back-propagation is often misunderstood as meaning the whole learning algorithm for multi-layer neural networks.

Actually, back-propagation refers only to the method for computing the gradient,

while another algorithm, such as stochastic gradient descent, is used to perform learning using this gradient.

Furthermore, back-propagation is often misunderstood as being specific to multilayer neural networks,

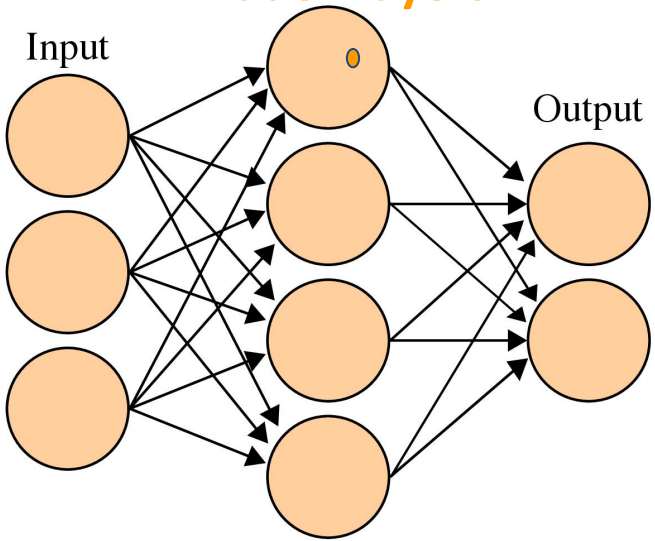
but in principle it can compute derivatives of any function.

© ZK

Deep Feedforward Networks or Multilayer Perceptrons

sigmoid is one possible activation function

hidden layers



$\boldsymbol{x} \in \mathbb{R}^{n_i}$ $\mathbf{h} \in \mathbb{R}^{n_h}$ $\hat{\mathbf{y}} \in \mathbb{R}^{n_o}$

$\mathbf{W} \in \mathbb{R}^{n_h \times n_i}$ $\mathbf{V} \in \mathbb{R}^{n_o \times n_h}$

The hidden unit vector is $\mathbf{h}$

$$\mathbf{h} = \text{sigmoid}(\mathbf{c} + \mathbf{W}\boldsymbol{x})$$

with weight matrix $\mathbf{W}$ and offset vector $\mathbf{c} \in \mathbb{R}^{n_h}$.
The output vector is obtained via the learned affine transformation

$$\hat{\mathbf{y}} = \mathbf{b} + \mathbf{V}\mathbf{h},$$

with weight matrix $\mathbf{V}$ and output offset vector $\mathbf{b} \in \mathbb{R}^{n_0}$.

© ZK
COLUMBIA | ENGINEERING
The Fu Foundation School of Engineering and Applied Science

Multi-Layer Neural Network
A Complete Description

or other activation function

$$\mathbf{f}(\boldsymbol{x};\boldsymbol{\theta}) = \mathbf{b} + \mathbf{V}\text{sigmoid}(\mathbf{c} + \mathbf{W}\boldsymbol{x}),$$

The parameters are the flattened vectorized version of the tuple

$$\boldsymbol{\theta} = (\mathbf{b}, \mathbf{c}, \mathbf{V}, \mathbf{W}).$$

parameters (coefficients) need to be (iteratively) updated

© ZK
COLUMBIA | ENGINEERING
The Fu Foundation School of Engineering and Applied Science

Multi-Layer Neural Network
Updating the Coefficients

Minimize the loss function

$$L(\hat{\mathbf{y}}, \mathbf{y})$$

Or cost function

$$J(\boldsymbol{\theta})$$

Example

$$L(\hat{\mathbf{y}}, \mathbf{y}) = ||\hat{\mathbf{y}} - \mathbf{y}||^2$$

© ZK

COLUMBIA | ENGINEERING
The Fu Foundation School of Engineering and Applied Science

Cost Minimization -> Iterative Weight Update/Training

Define the weights $\boldsymbol{\omega}$ as the concatenated elements of matrices $\mathbf{W}$ and $\mathbf{V}$ and $||\boldsymbol{\omega}||^2 = \sum_{i,j} W_{ij}^2 + \sum_{ki} V_{ki}^2$. The regularized cost function typically considered

$$J(\boldsymbol{\theta}) = \lambda||\boldsymbol{\omega}||^2 + \frac{1}{n}\sum_{t=1}^{n} ||\mathbf{y}^t - (\mathbf{b} + \mathbf{V}\text{sigmoid}(\mathbf{c} + \mathbf{W}\boldsymbol{x}^t))||^2,$$

where $(\boldsymbol{x}^t, \mathbf{y}^t)$ is the t-th training (input, target) pair.
The classical training procedure iteratively updates $\boldsymbol{\theta}$ as in

$$\boldsymbol{\omega} \leftarrow \boldsymbol{\omega} - \epsilon(2\lambda\boldsymbol{\omega} + \nabla_{\boldsymbol{\omega}} L(\mathbf{f}(\boldsymbol{x}^t; \boldsymbol{\theta}), \mathbf{y}^t))$$
$$\boldsymbol{\beta} \leftarrow \boldsymbol{\beta} - \epsilon\nabla_{\boldsymbol{\beta}} L(\mathbf{f}(\boldsymbol{x}^t; \boldsymbol{\theta}), \mathbf{y}^t),$$

where $\boldsymbol{\beta} = (\mathbf{b}, \mathbf{c})$ contains the offset parameters, ϵ is the learning rate and t is incremented after each training example, modulo n.

COLUMBIA | ENGINEERING
The Fu Foundation School of Engineering and Applied Science

Back-Propagation
Minimizing the Cost Function

In machine learning the output of the function to differentiate (e.g., the training criterion J) is a scalar. We are interested in its derivative with respect to a set of parameters (considered to be the elements of a vector $\boldsymbol{\theta}$), or equivalently, a set of inputs.

The partial derivative of J with respect to $\boldsymbol{\theta}$ (called the gradient) tells us whether $\boldsymbol{\theta}$ should be increased or decreased in order to decrease J, and is a crucial tool in optimizing the training objective.

© ZK
COLUMBIA | ENGINEERING
The Fu Foundation School of Engineering and Applied Science

Back-Propagation
Gradient Computation

It can be readily proven that the back-propagation algorithm for computing gradients has optimal computational complexity in the sense that there is no algorithm that can compute the gradient faster (in the O sense, i.e., up to an additive and multiplicative constant).

The basic idea of the back-propagation algorithm is that the partial derivative of the cost J with respect to parameters $\boldsymbol{\theta}$ can be decomposed recursively by taking into consideration the composition of functions that relate $\boldsymbol{\theta}$ to J, via intermediate quantities that mediate that influence, e.g., the activations of hidden units in a deep neural network.

Back-Propagation
Chain Rule for Taking Derivatives - Math Review

Let $z = z(y(x))$ be a composite function. To compute the derivative, we use the chain rule:

$$\frac{dz}{dx} = \frac{dz}{dy}\frac{dy}{dx} \quad \text{or} \quad dz = (\frac{dz}{dy})(\frac{dy}{dx})dx.$$

If $z = z(y_1(x), y_2(x), ..., y_n(x))$ then

$$\frac{dz}{dx} = \sum_{i=1}^{n} \frac{\partial z}{\partial y_i}\frac{dy_i}{dx}.$$

If $z = z(y(x_1, x_2, ..., x_n))$ then

$$\nabla z = \frac{dz}{dy}(\frac{\partial y}{\partial x_1}, \frac{\partial y}{\partial x_2}, ..., \frac{\partial y}{\partial x_n})^T,$$

© ZK
COLUMBIA | ENGINEERING
The Fu Foundation School of Engineering and Applied Science

Back-Propagation
Chain Rule Applied to Machine Learning

The "output" cost or objective function is $J = J(g(\boldsymbol{\theta}))$ and we are interested in the gradient with respect to the parameters $\boldsymbol{\theta}$. The intermediate quantities $g(\boldsymbol{\theta})$ correspond to neural network activations. The gradient of the cost function amounts to

$$\nabla_{\boldsymbol{\theta}} J(g(\boldsymbol{\theta})) = \nabla_{g(\boldsymbol{\theta})} J(g(\boldsymbol{\theta})) \frac{\partial g(\boldsymbol{\theta})}{\partial \boldsymbol{\theta}}$$

which works also when J, g or $\boldsymbol{\theta}$ are vectors rather than scalars (in which case the corresponding partial derivatives are understood as Jacobian matrices of the appropriate dimensions).
If g is a vector, we can rewrite the above as follows:

$$\nabla_{\boldsymbol{\theta}} J(g(\boldsymbol{\theta})) = \sum_i \frac{\partial J(g(\boldsymbol{\theta}))}{\partial g_i(\boldsymbol{\theta})} \frac{\partial g_i(\boldsymbol{\theta})}{\partial \boldsymbol{\theta}}.$$

© ZK
COLUMBIA | ENGINEERING
The Fu Foundation School of Engineering and Applied Science

Artificial Neural Networks
Iterative Computational Process

Feed forward computation

uses
given coefficient values.

Back-Propagation

is used for
updating the coefficient values.

© ZK
COLUMBIA | ENGINEERING
The Fu Foundation School of Engineering and Applied Science

Artificial Neural Networks
Forward Computation

We consider an MLP with affine layers composed with an arbitrary elementwise differentiable (almost everywhere) non-linearity f.

There are l layers, each mapping their vector-valued input $\mathbf{h}^{k-1}$ to a pre-activation vector $\mathbf{a}^k$ via a weight matrix $\mathbf{W}^k$ which is then transformed via f into $\mathbf{h}^k$.

The input vector $\boldsymbol{x}$ corresponds to $\mathbf{h}^0$ and the predicted outputs $\hat{\mathbf{y}}$ corresponds to $\mathbf{h}^l$.

The loss $L(\hat{\mathbf{y}}, \mathbf{y})$ depends on the output $\hat{\mathbf{y}}$ and on a target $\mathbf{y}$. The loss may be added to a regularizer Ω to obtain the cost function J.

© ZK
COLUMBIA | ENGINEERING
The Fu Foundation School of Engineering and Applied Science

Artificial Neural Networks
Forward Computation Algorithm

Algorithm 1 Forward propagation computation in matrix form for a multi-layer neural network with l affine layers and a non-linearity.

$\mathbf{h}^0 = x$
for $i = 1, ..., l$ **do**
  $\mathbf{a}^k = \mathbf{b}^k + \mathbf{W}^k \mathbf{h}^{k-1}$,
  $\mathbf{h}^k = f(\mathbf{a}^k)$.
end for
$\hat{\mathbf{y}} = \mathbf{h}^l$
$J = L(\hat{\mathbf{y}}, \mathbf{y}) + \lambda\Omega$

Each unit at layer k computes an output h_i^k as follows:

$$a_i^k = b_i^k + \sum_j W_{ij}^k h_j^{k-1} \quad \text{and} \quad h_i^k = f(a_i^k).$$

Note that the forward computation uses the input $\mathbf{x}$.

© ZK
COLUMBIA | ENGINEERING
The Fu Foundation School of Engineering and Applied Science

Artificial Neural Networks
Back-Propagation in MLP

The algorithm proceeds by first computing the gradient of the cost J with respect to output units, and these are used to compute the gradient of J with respect to the top hidden layer activations, which directly influence the outputs. We can then continue computing the gradients of lower level hidden units one at a time in the same way.

The gradients on hidden and output units can be used to compute the gradient of J with respect to the parameters. In a typical network divided into many layers with each layer parameterized by a weight matrix and a vector of biases, the gradient on the weights and the biases is determined by the input to the layer and the gradient on the output of the layer.

© ZK
COLUMBIA | ENGINEERING
The Fu Foundation School of Engineering and Applied Science

Artificial Neural Networks
Back-Propagation Intuition

We do not know what the hidden units ought to do, but we can compute how fast the error changes as we change a hidden activity.

- Instead of using desired activities to train the hidden units, use error derivatives w.r.t. hidden activities;
- Each hidden activity can affect many output units and can therefore have many separate effects on the error. These effects must be combined.

We can compute error derivatives for all the hidden units efficiently at the same time.

- Once we have the error derivatives for the hidden activities, its easy to get the error derivatives for the weights going into a hidden unit.

© ZK
COLUMBIA | ENGINEERING
The Fu Foundation School of Engineering and Applied Science

Artificial Neural Networks
Back-Propagation Algorithm Intuition

First convert the discrepancy between each output and its target value into an error derivative:

$$\frac{\partial J}{\partial a_j^l} = \frac{\partial J}{\partial h_j^l} \cdot \frac{\partial h_j^l}{\partial a_j^l}.$$

Here $\frac{\partial J}{\partial h_j^l}$ is the derivative of the cost function with respect to the output. The second term on the RHS amounts to f'.

Then compute error derivatives in each hidden layer from error derivatives in the layer above, i.e., compute

$$\frac{\partial J}{\partial h_i^k} = \sum_j \frac{\partial J}{\partial a_j^l} \frac{\partial a_j^l}{\partial h_i^k} = \sum_j W_{ij}^l \frac{\partial J}{\partial a_j^l}.$$

© ZK
COLUMBIA | ENGINEERING
The Fu Foundation School of Engineering and Applied Science

Artificial Neural Networks
Back-Propagation Algorithm Intuition

Then use error derivatives w.r.t. activities to get error derivatives w.r.t. the incoming weights:

$$\frac{\partial J}{\partial W_{ij}^k} = \frac{\partial J}{\partial a_j^l} \frac{\partial a_j^l}{\partial W_{ij}^k} = h_i^k \frac{\partial J}{\partial a_j^l}.$$

Note that since the l's is the output layer, we used the notation $k = l - 1$ for the hidden layer just before the output layer.

© ∠K

COLUMBIA | ENGINEERING
The Fu Foundation School of Engineering and Applied Science

Artificial Neural Networks
Back-Propagation Algorithm

Algorithm 2 Backward computation of the gradients of the cost function with respect to the parameters for each layer k.

Compute the gradient on the output layer

$\mathbf{g} \leftarrow \nabla_{\hat{\mathbf{y}}} J = \nabla_{\hat{\mathbf{y}}} L(\hat{\mathbf{y}}, \mathbf{y}) + \lambda \nabla_{\hat{\mathbf{y}}} \Omega$

for $k = l, l-1, ..., 1$ **do**

Convert the gradient on the layers output into a gradient into the pre-nonlinearity activation

$\mathbf{g} \leftarrow \nabla_{\mathbf{a}^k} J = \mathbf{g} \odot f'(\mathbf{a}^k)$

Compute gradients on weights and biases

$\nabla_{\mathbf{b}^k} J = \mathbf{g} + \lambda \nabla_{\mathbf{b}^k} \Omega$

$\nabla_{\mathbf{W}^k} J = \mathbf{g}\, \mathbf{h}^{(k-1)T} + \lambda \nabla_{\mathbf{W}^k} \Omega$

Propagate gradients w.r.t. the next lower-level hidden layers activations

$\mathbf{g} \leftarrow \nabla_{\mathbf{h}^{k-1}} J = \mathbf{W}^{(k)T} \mathbf{g}$

end for

COLUMBIA | ENGINEERING
The Fu Foundation School of Engineering and Applied Science

Computational Graph (Another look at... Forward Propagation and Back-Propagation)

Here we model the evaluation of the cost function J on a graph called the computational graph. Each node u_i of the graph denotes a numerical quantity that is obtained by performing a computation requiring the values u_j of other nodes, with $j < i$. The nodes satisfy a partial order which dictates in what order the computation can proceed.

We will use the generic notation $u_i = f_i(\mathbf{a}^i)$, where $\mathbf{a}^i$ is a list of arguments for the application of f_i to the values u_j of the parents of i in the graph: $\mathbf{a}^i = (u_j)_{j \in \text{parents}(i)}$.

COLUMBIA | ENGINEERING
The Fu Foundation School of Engineering and Applied Science

Computational Graph: Definitions

A node in the graph is used to indicate a variable. The variable may be a scalar, vector, matrix, tensor, or even a variable of another type.

An operation is a simple function of one or more variables.

If a variable y is computed by applying an operation to a variable x, then we draw a directed edge from x to y.

© ZK

COLUMBIA | ENGINEERING
The Fu Foundation School of Engineering and Applied Science

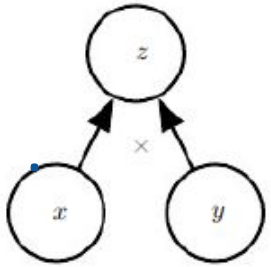
Computational Graph Examples

Scalars, vectors and matrices.

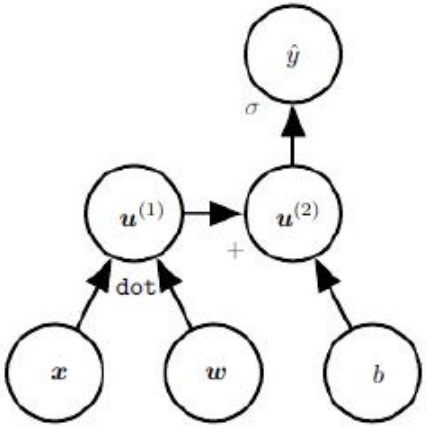
(b) logistic regression prediction

(c) H = max{0, XW + b}, which computes a design matrix of rectified linear unit activations H given a design matrix containing a minibatch of inputs X

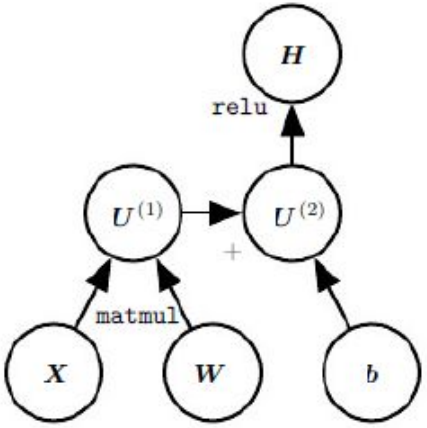
(d) Computing both the prediction and the weight decay penalty.



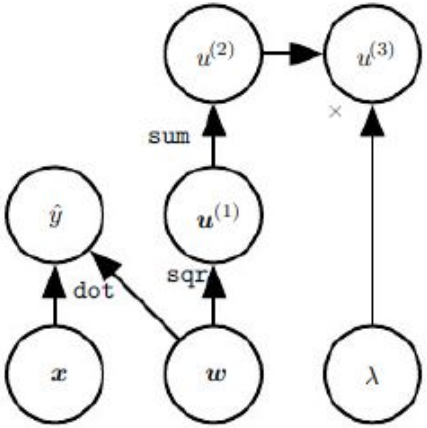
(a)



(b)



(c)



(d)

Computational Graph
Forward Computation

Algorithm 3 A computational graph describing forward propagation. Each node computes numerical value u_i by applying a function f_i to its argument list $\mathbf{a}^i$ that comprises the values of previous nodes $u_j, j < i$, with $j \in \text{parents}(i)$. The input to the computational graph is the vector $\boldsymbol{x}$, and is set into the first m nodes u_1 to u_m. The output of the computational graph is read off the last node u_n.

for $i = 1, ..., m$ **do**
  $u_i \leftarrow x_i,$
end for
for $i = m + 1, ..., n$ **do**
  $\mathbf{a}^i \leftarrow (u_j)_{j \in \text{parents}(i)}$
  $u_i \leftarrow f_i(\mathbf{a}^i)$
end for
return u_n

COLUMBIA | ENGINEERING
The Fu Foundation School of Engineering and Applied Science

Computational Graph
Direct & Indirect Effects in a Computational Graph

We consider $f_3(a_{31}, a_{32}) = e^{a_{31}+a_{32}}$ and $f_2(a_{21}) = a_{21}^2$, while $u_3 = f_3(u_2, u_1)$ and $u_2 = f_2(u_1)$. The direct derivative of f_3 with respect to its argument a_{32} (keeping a_{31} fixed) is $\frac{\partial f_3}{\partial a_{32}} = e^{a_{31}+a_{32}}$.

On the other hand, if we consider the variables u_3 and u_1 to which these arguments correspond, there are two paths from u_1 to u_3, and we obtain as the total derivative the sum of partial derivatives over these two paths, $\frac{\partial u_3}{\partial u_1} = e^{u_1+u_2}(1 + 2u_1)$.

The results are different because $\frac{\partial u_3}{\partial u_1}$ involves not just the direct dependency of u_3 on u_1 but also the indirect dependency through u_2.

© ZK

Computational Graph
Back-Propagation

Algorithm 4 $\pi(i,j)$ is the index of u_j as an argument to f_i. The back-propagation algorithm efficiently computes $\frac{\partial u_n}{\partial u_i}$ for all is (traversing the graph backwards), and in particular we are interested in the derivatives of the output node u_n with respect to the inputs $u_1, ..., u_m$ (which could be the parameters, in a learning setup). The cost of the overall computation is proportional to the number of arcs in the graph, assuming that the partial derivative associated with each arc requires a constant time.

$\frac{\partial u_n}{\partial u_n} \leftarrow 1$
for $j = n-1$ down to 1 **do**
  $\frac{\partial u_n}{\partial u_j} \leftarrow \sum_{i:j \in \text{parents}(i)} \frac{\partial u_n}{\partial u_i} \frac{\partial f_i(a_i)}{\partial a_{i,\pi(i,j)}}$
end for
return $(\frac{\partial u_n}{\partial u_i})_{i=1}^m$

COLUMBIA | ENGINEERING
The Fu Foundation School of Engineering and Applied Science

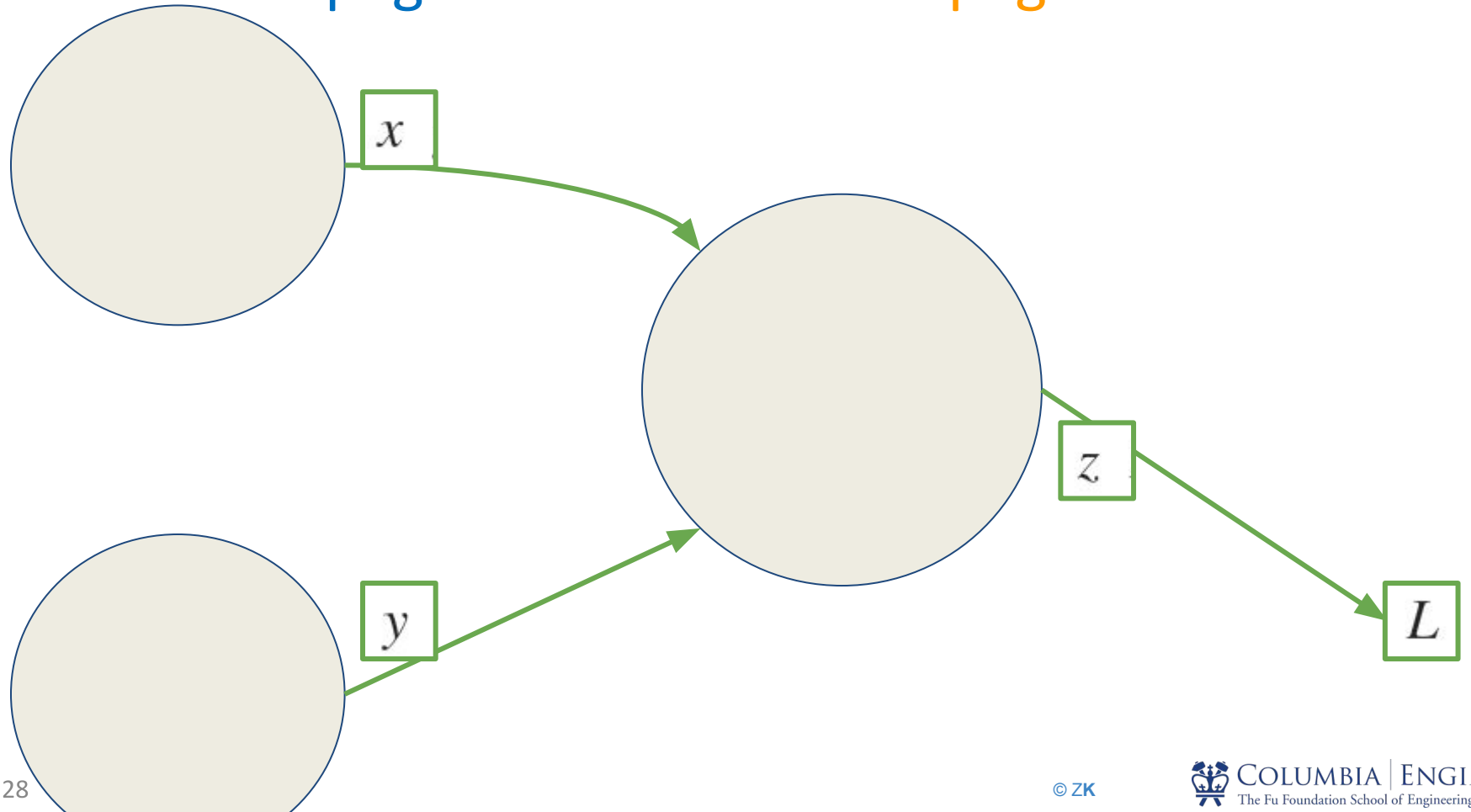
Back-Propagation Examples

Partial Derivatives, Node Propagation, Network Propagation

© ZK

COLUMBIA | ENGINEERING
The Fu Foundation School of Engineering and Applied Science

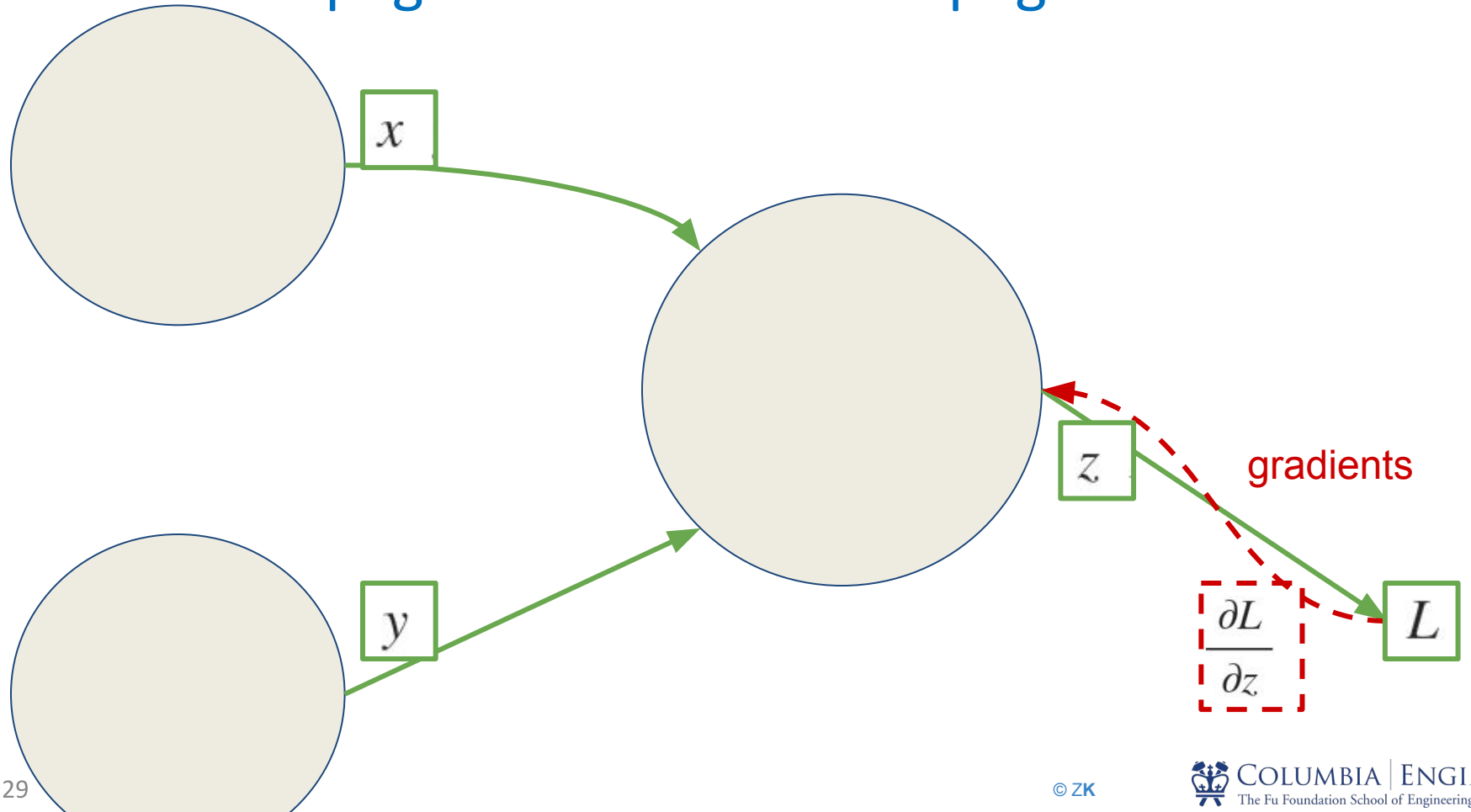
Back-Propagation: Gradient Propagation



© ZK

COLUMBIA | ENGINEERING
The Fu Foundation School of Engineering and Applied Science

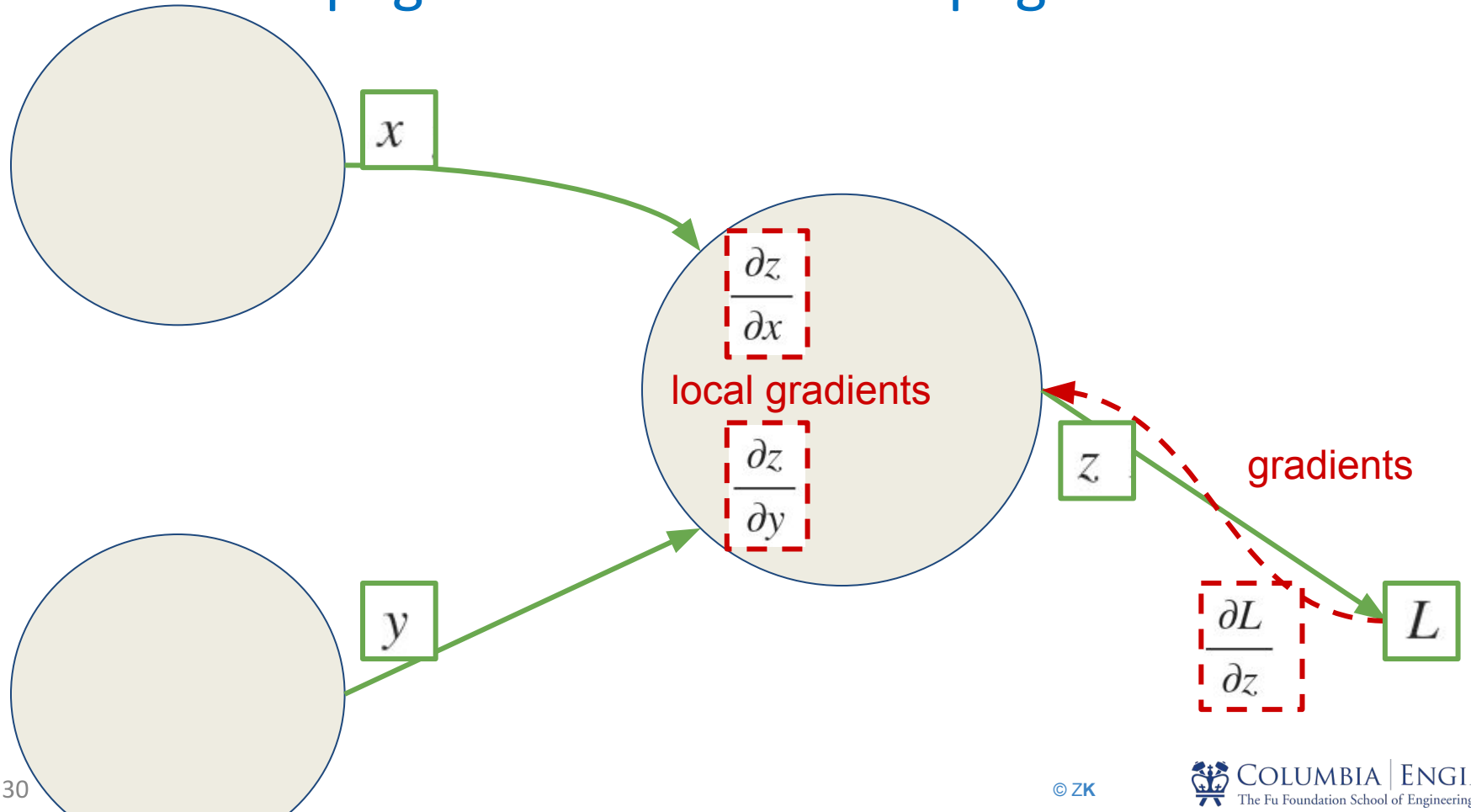
Back-Propagation: Gradient Propagation



© ZK

COLUMBIA | ENGINEERING
The Fu Foundation School of Engineering and Applied Science

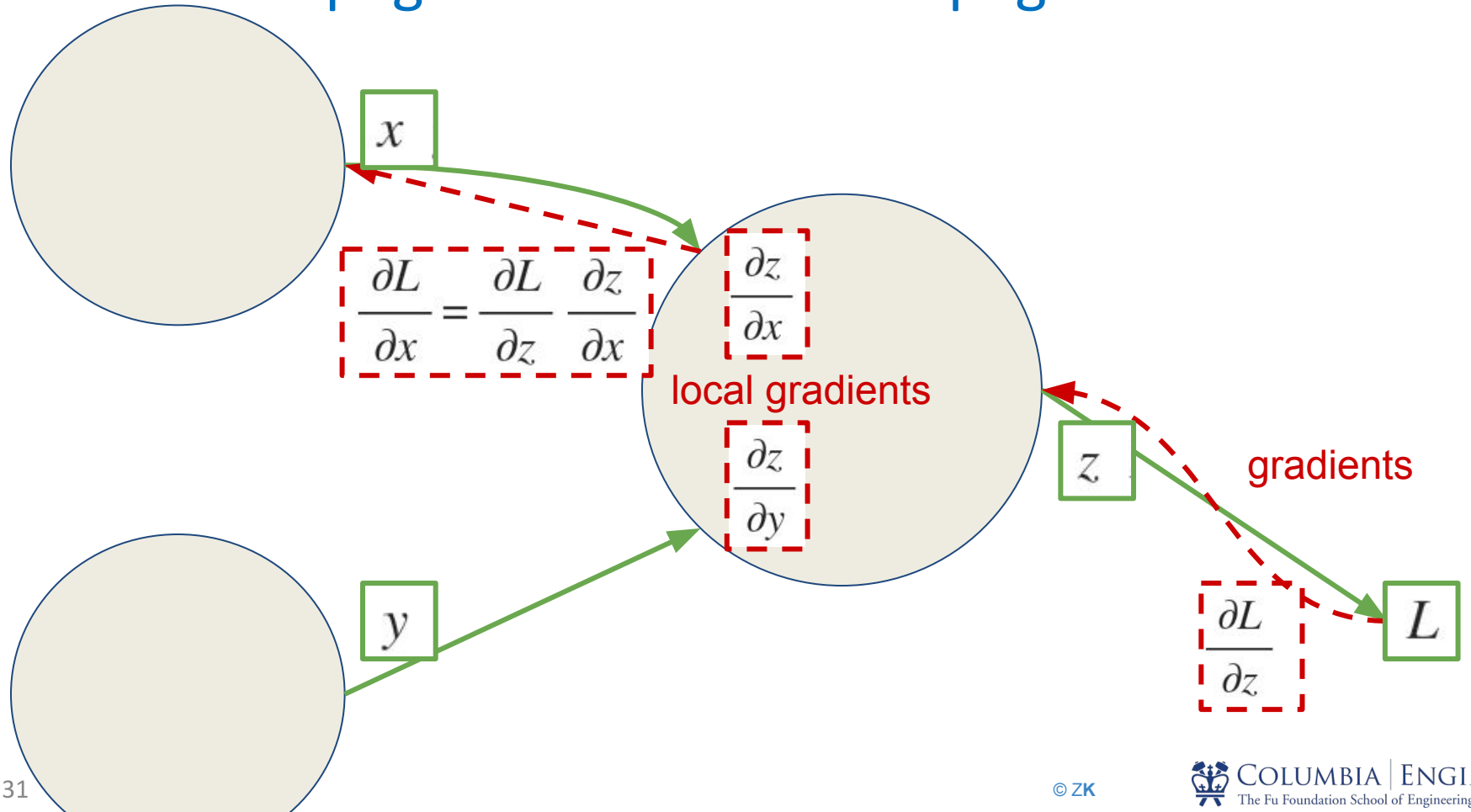
Back-Propagation: Gradient Propagation



© ZK

COLUMBIA | ENGINEERING
The Fu Foundation School of Engineering and Applied Science

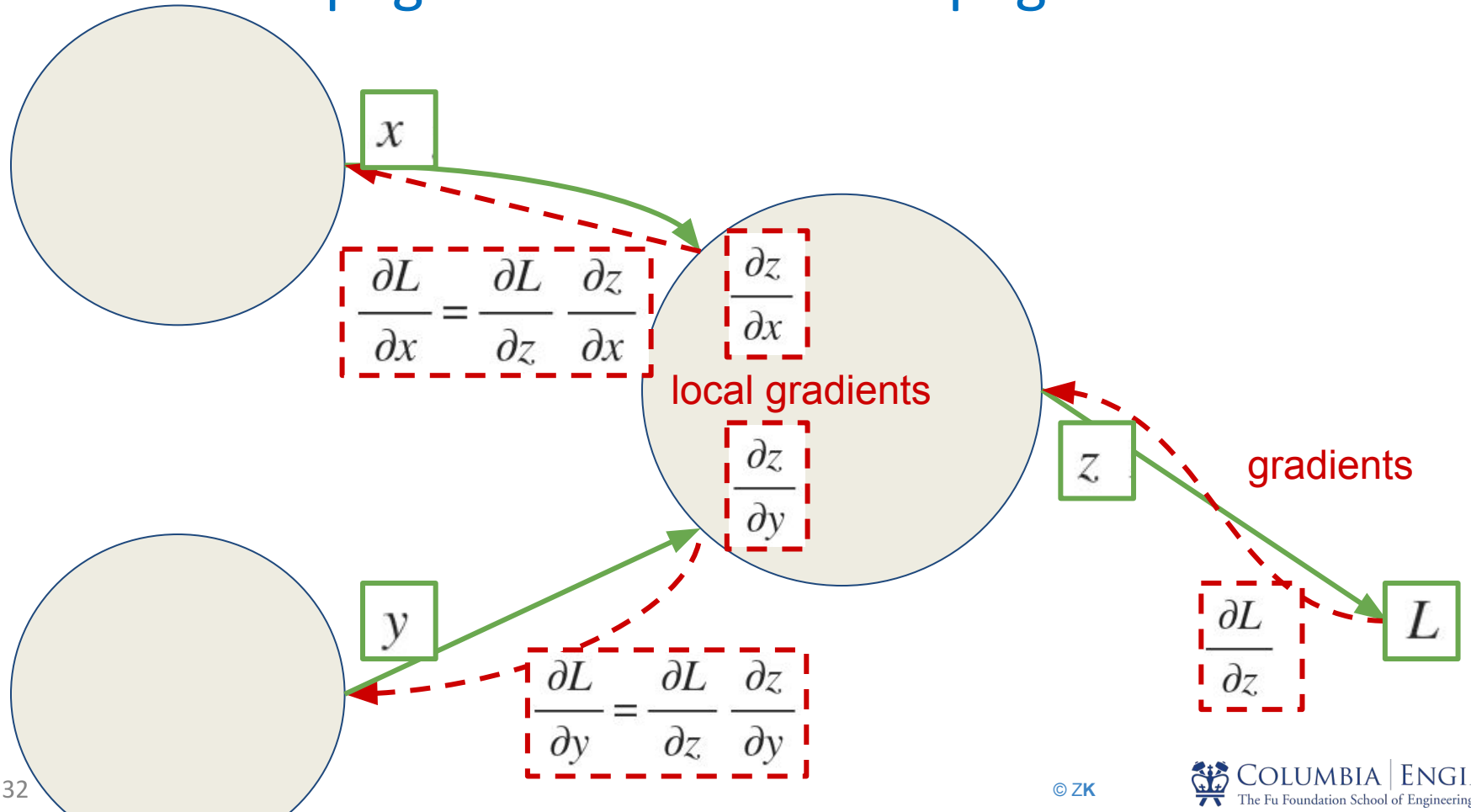
Back-Propagation: Gradient Propagation



© ZK

COLUMBIA | ENGINEERING
The Fu Foundation School of Engineering and Applied Science

Back-Propagation: Gradient Propagation



© ZK

COLUMBIA | ENGINEERING
The Fu Foundation School of Engineering and Applied Science

Back-Propagation Example 1
Computational Graph

$$f(x,y,z)=x*y+z$$

© ZK

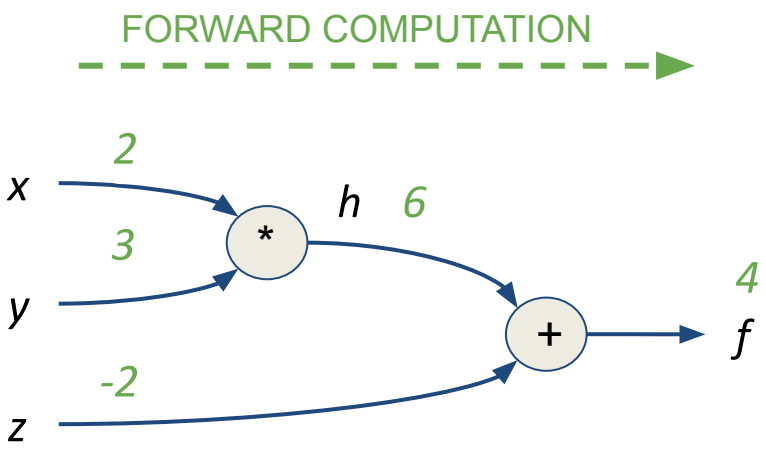
COLUMBIA | ENGINEERING
The Fu Foundation School of Engineering and Applied Science

Back-Propagation Example 1
Computational Graph

$f(x,y,z)=x*y+z$

$e.g.\ \ x=2,\ \ y=3,\ \ z=-2$

$We\ \ need:\ \ \frac{\partial f}{\partial x},\ \ \frac{\partial f}{\partial y},\ \ \frac{\partial f}{\partial z}$



© ZK

COLUMBIA | ENGINEERING
The Fu Foundation School of Engineering and Applied Science

Back-Propagation Example 1
Computational Graph

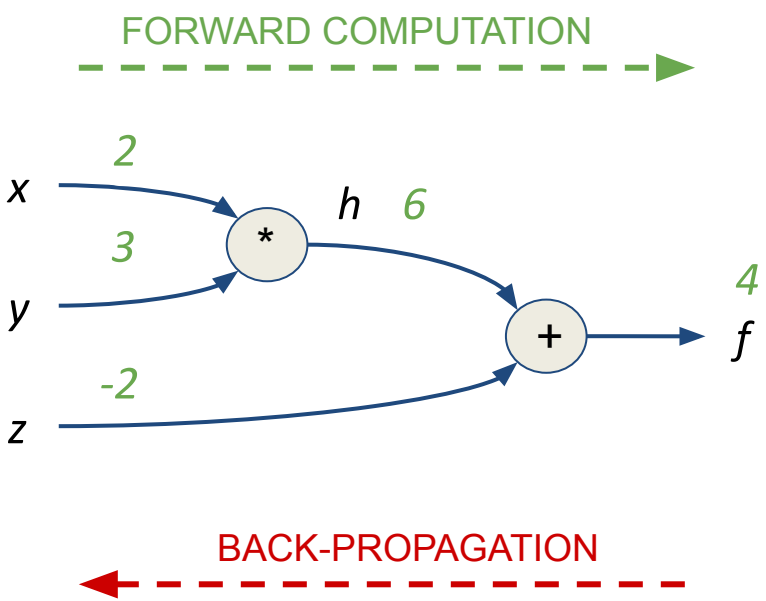
$f(x,y,z)=x*y+z$

$e.g.\ \ x=2,\ \ y=3,\ \ z=-2$

$We\ \ need:\ \ \frac{\partial f}{\partial x},\ \ \frac{\partial f}{\partial y},\ \ \frac{\partial f}{\partial z}$

$h=x*y \Rightarrow \frac{\partial h}{\partial x}=y,\ \ \frac{\partial h}{\partial y}=x$

$f=h+z \Rightarrow \frac{\partial f}{\partial h}=1,\ \ \frac{\partial f}{\partial z}=1$



© ZK

COLUMBIA | ENGINEERING
The Fu Foundation School of Engineering and Applied Science

Back-Propagation Example 1
Computational Graph

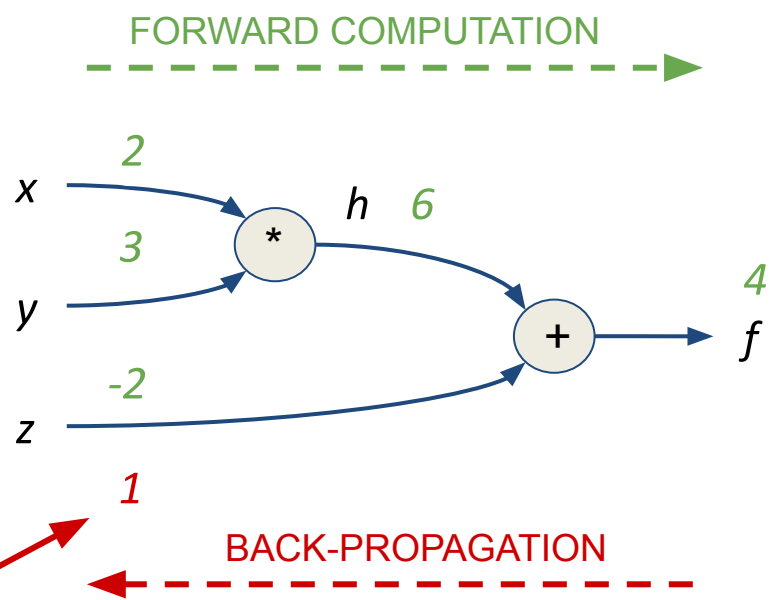
$f(x,y,z)=x*y+z$

$e.g.\ \ x=2,\ \ y=3,\ \ z=-2$

$We\ \ need:\ \ \frac{\partial f}{\partial x},\ \ \frac{\partial f}{\partial y},\ \ \frac{\partial f}{\partial z}$

$h=x*y \Rightarrow \frac{\partial h}{\partial x}=y,\ \ \frac{\partial h}{\partial y}=x$

$f=h+z \Rightarrow \frac{\partial f}{\partial h}=1,\ \ \frac{\partial f}{\partial z}=1$



© ZK

COLUMBIA | ENGINEERING
The Fu Foundation School of Engineering and Applied Science

Back-Propagation Example 1
Computational Graph

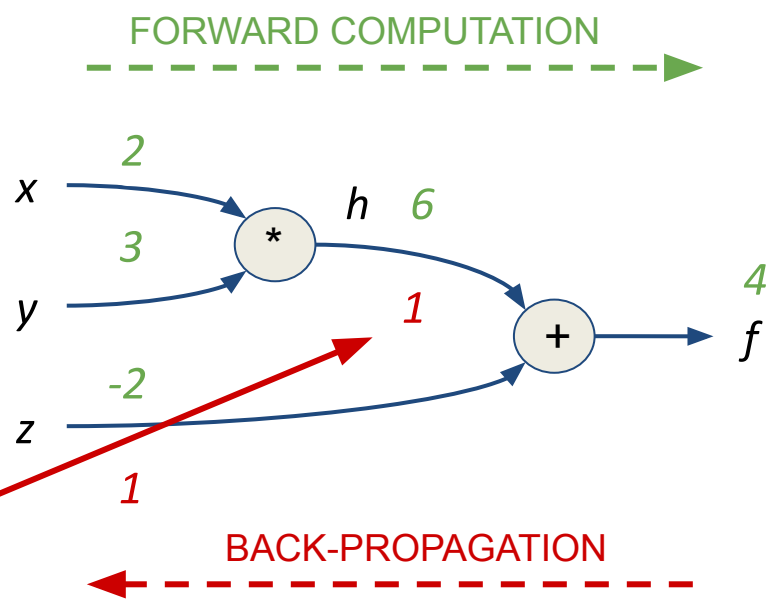
$f(x,y,z)=x*y+z$

$e.g.\ \ x=2,\ \ y=3,\ \ z=-2$

$We\ \ need:\ \ \frac{\partial f}{\partial x},\ \ \frac{\partial f}{\partial y},\ \ \frac{\partial f}{\partial z}$

$h=x*y \Rightarrow \frac{\partial h}{\partial x}=y,\ \ \frac{\partial h}{\partial y}=x$

$f=h+z \Rightarrow \frac{\partial f}{\partial h}=1,\ \ \frac{\partial f}{\partial z}=1$



© ZK

COLUMBIA | ENGINEERING
The Fu Foundation School of Engineering and Applied Science

Back-Propagation Example 1
Computational Graph

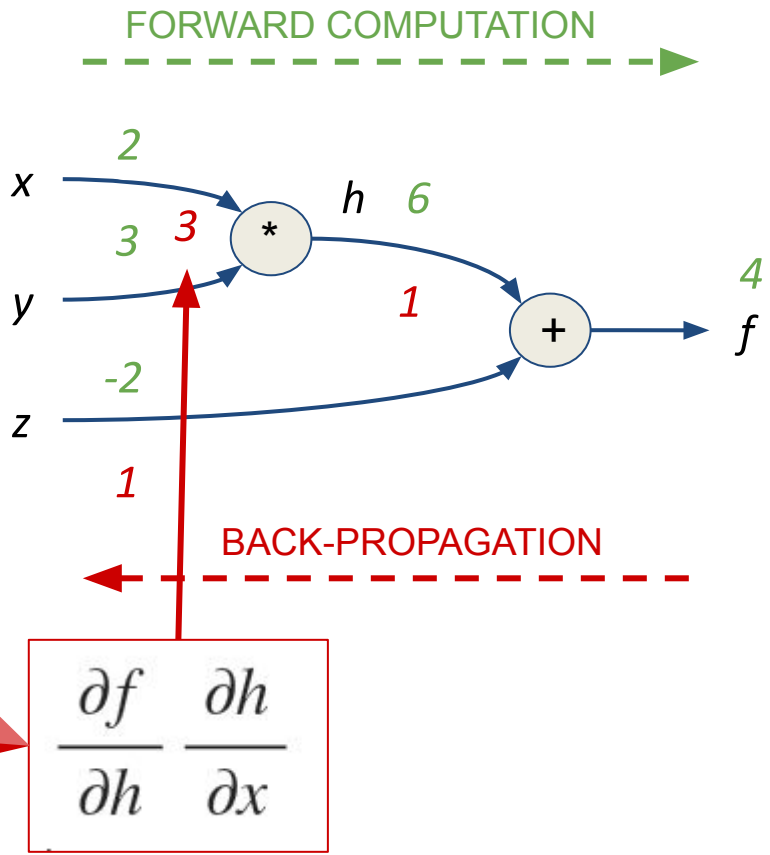
$f(x,y,z)=x*y+z$

e.g. $x=2,\ y=3,\ z=-2$

We need: $\frac{\partial f}{\partial x}, \frac{\partial f}{\partial y}, \frac{\partial f}{\partial z}$

$h=x*y \Rightarrow \frac{\partial h}{\partial x}=y, \frac{\partial h}{\partial y}=x$

$f=h+z \Rightarrow \frac{\partial f}{\partial h}=1, \frac{\partial f}{\partial z}=1$



© ZK

COLUMBIA | ENGINEERING
The Fu Foundation School of Engineering and Applied Science

Back-Propagation Example 1
Computational Graph

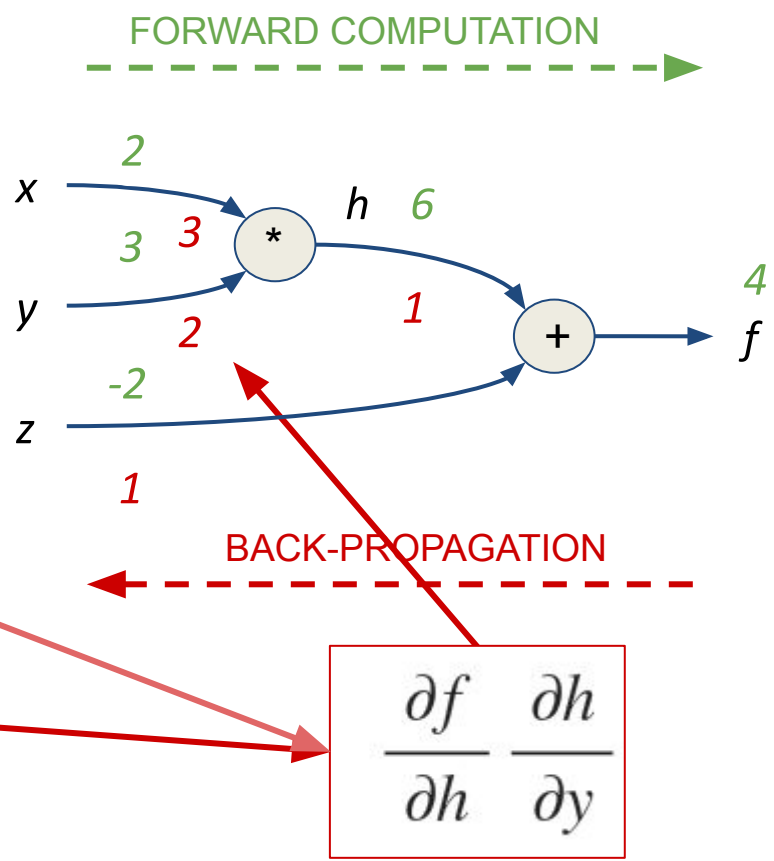
$f(x,y,z)=x*y+z$

e.g. $x=2,\ y=3,\ z=-2$

We need: $\frac{\partial f}{\partial x}, \frac{\partial f}{\partial y}, \frac{\partial f}{\partial z}$

$h=x*y \Rightarrow \frac{\partial h}{\partial x}=y, \frac{\partial h}{\partial y}=x$

$f=h+z \Rightarrow \frac{\partial f}{\partial h}=1, \frac{\partial f}{\partial z}=1$



© ZK

COLUMBIA | ENGINEERING
The Fu Foundation School of Engineering and Applied Science

Back-Propagation Example 2

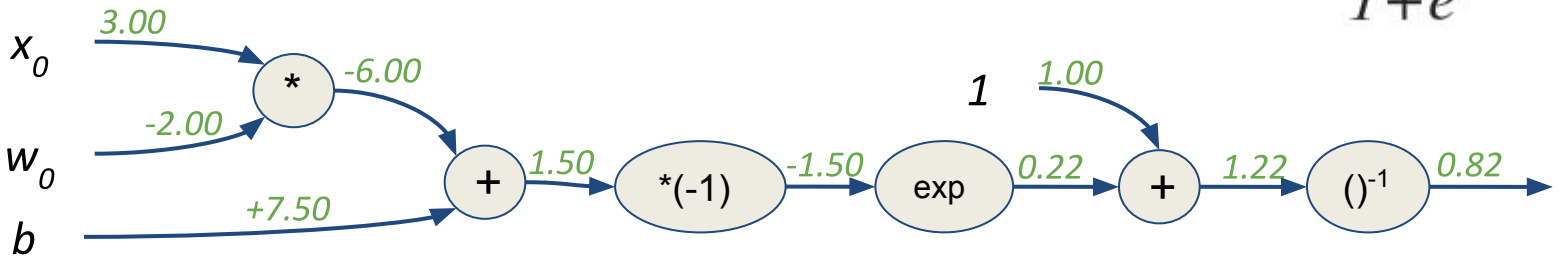
$$f(w,x) = \frac{1}{1+e^{-(w_0 x_0 + b)}}$$

© ZK

COLUMBIA | ENGINEERING
The Fu Foundation School of Engineering and Applied Science

Back-Propagation Example 2

$$f(w,x) = \frac{1}{1+e^{-(w_0 x_0 + b)}}$$



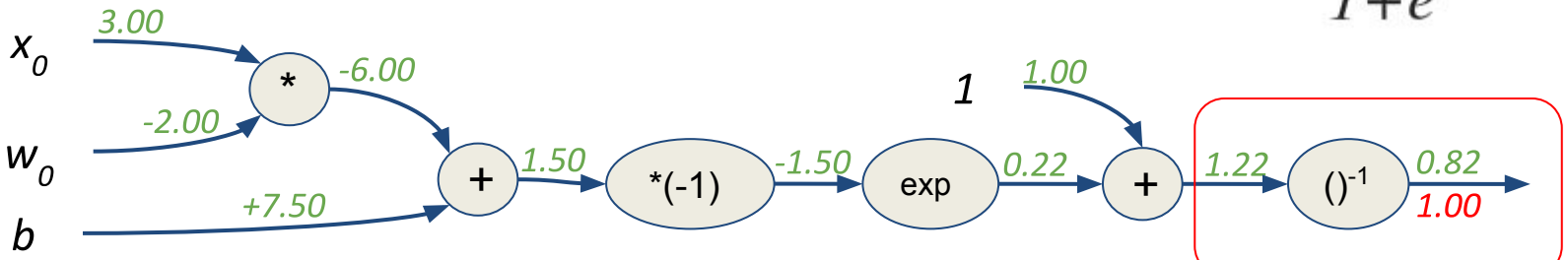
$f(x)$	$\frac{df(x)}{dx}$
e^x	e^x
ax	a
$1/x$	$-1/x^2$
$x+C$	1

© ZK

COLUMBIA | ENGINEERING
The Fu Foundation School of Engineering and Applied Science

Back-Propagation Example 2

$$f(w,x) = \frac{1}{1+e^{-(w_0 x_0 + b)}}$$



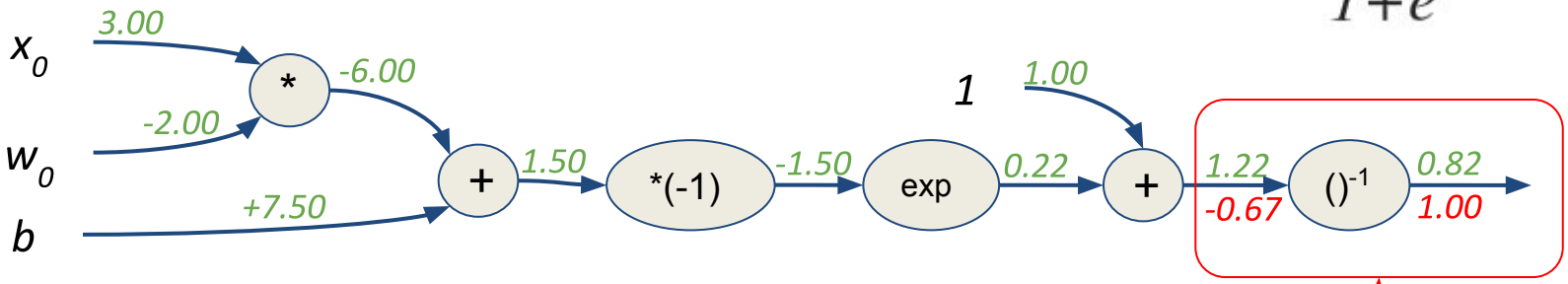
$f(x)$	$\frac{df(x)}{dx}$
e^x	e^x
ax	a
$1/x$	$-1/x^2$
$x+C$	1

© ZK

COLUMBIA | ENGINEERING
The Fu Foundation School of Engineering and Applied Science

Back-Propagation Example 2

$$f(w,x)=\frac{1}{1+e^{-(w_0x_0+b)}}$$



$f(x)$	$\frac{df(x)}{dx}$
e^x	e^x
ax	a
$1/x$	$-1/x^2$
$x+C$	1

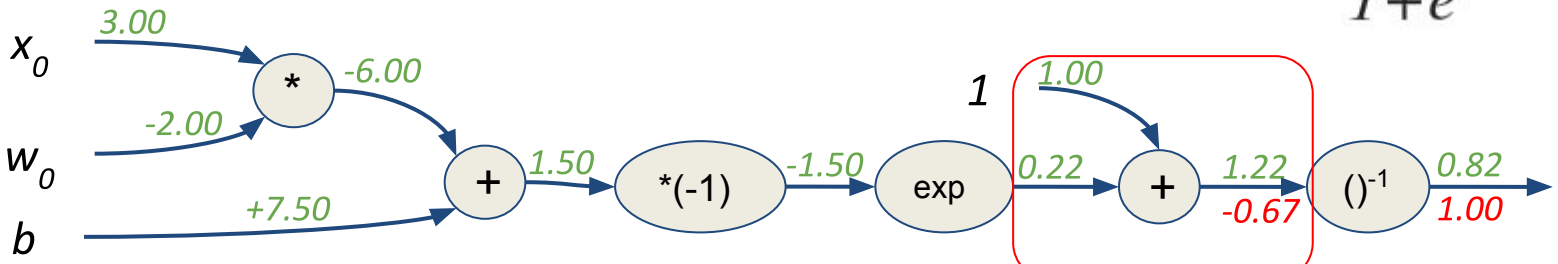
$$-\frac{1}{1.22^2}(1.00)=-0.67$$

© ZK

COLUMBIA | ENGINEERING
The Fu Foundation School of Engineering and Applied Science

Back-Propagation Example 2

$$f(w,x) = \frac{1}{1+e^{-(w_0 x_0 + b)}}$$



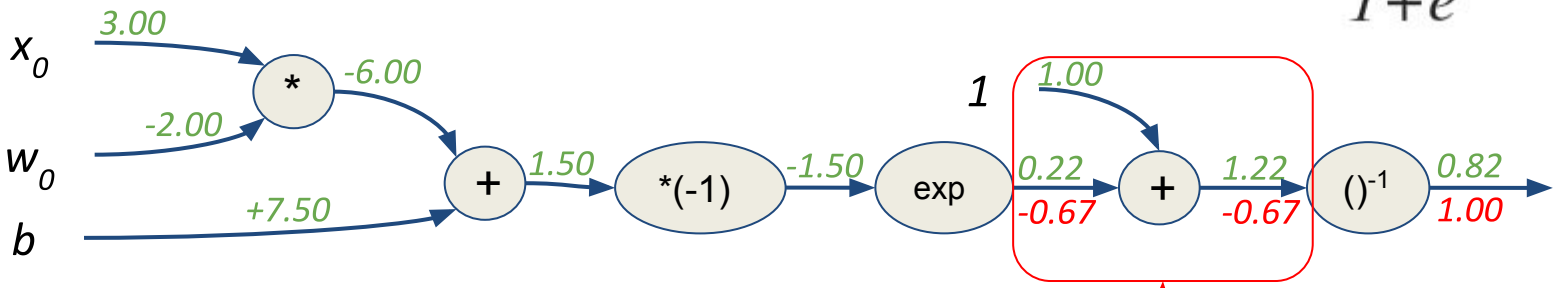
$f(x)$	$\frac{df(x)}{dx}$
e^x	e^x
ax	a
$1/x$	$-1/x^2$
$x+C$	1

© ZK

COLUMBIA | ENGINEERING
The Fu Foundation School of Engineering and Applied Science

Back-Propagation Example 2

$$f(w,x)=\frac{1}{1+e^{-(w_0x_0+b)}}$$



$f(x)$	$\frac{df(x)}{dx}$
e^x	e^x
ax	a
$1/x$	$-1/x^2$
$x+C$	1

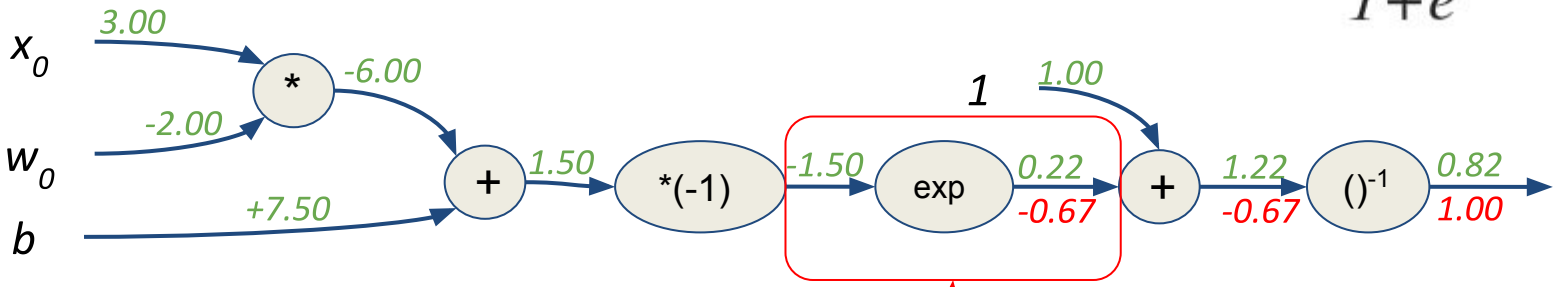
$$(1)(-0.67)=-0.67$$

© ZK

COLUMBIA | ENGINEERING
The Fu Foundation School of Engineering and Applied Science

Back-Propagation Example 2

$$f(w,x)=\frac{1}{1+e^{-(w_0x_0+b)}}$$



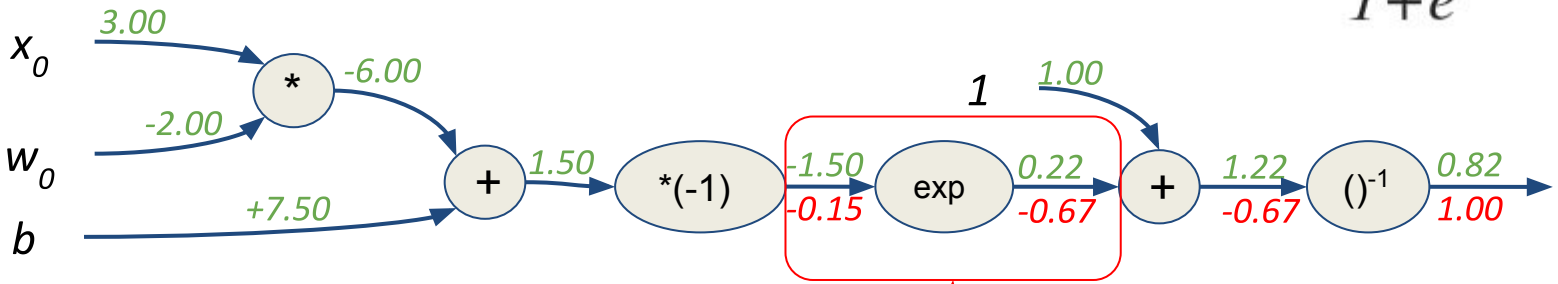
$f(x)$	$\frac{df(x)}{dx}$
e^x	e^x
ax	a
$1/x$	$-1/x^2$
$x+C$	1

© ZK

COLUMBIA | ENGINEERING
The Fu Foundation School of Engineering and Applied Science

Back-Propagation Example 2

$$f(w,x)=\frac{1}{1+e^{-(w_0x_0+b)}}$$



$f(x)$	$\frac{df(x)}{dx}$
e^x	e^x
ax	a
$1/x$	$-1/x^2$
$x+C$	1

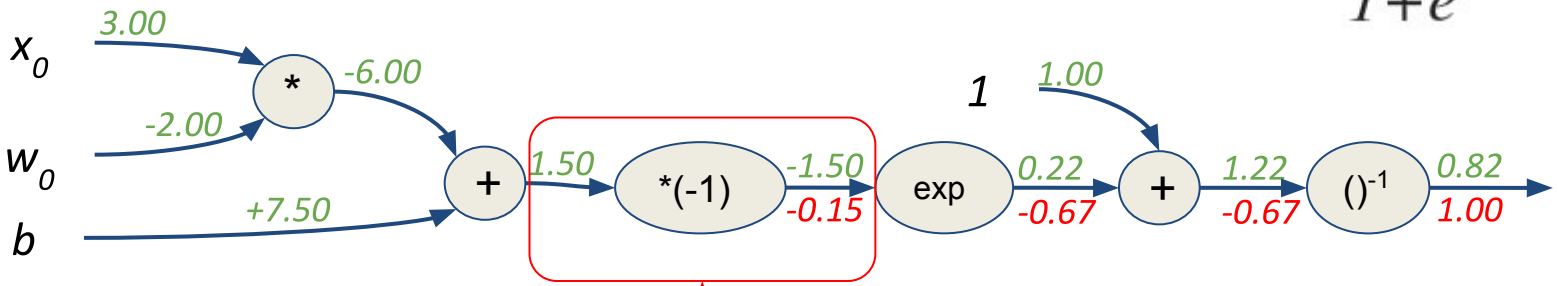
$$e^{-1.50}(-0.67)=-0.15$$

© ZK

COLUMBIA | ENGINEERING
The Fu Foundation School of Engineering and Applied Science

Back-Propagation Example 2

$$f(w,x) = \frac{1}{1+e^{-(w_0 x_0 + b)}}$$



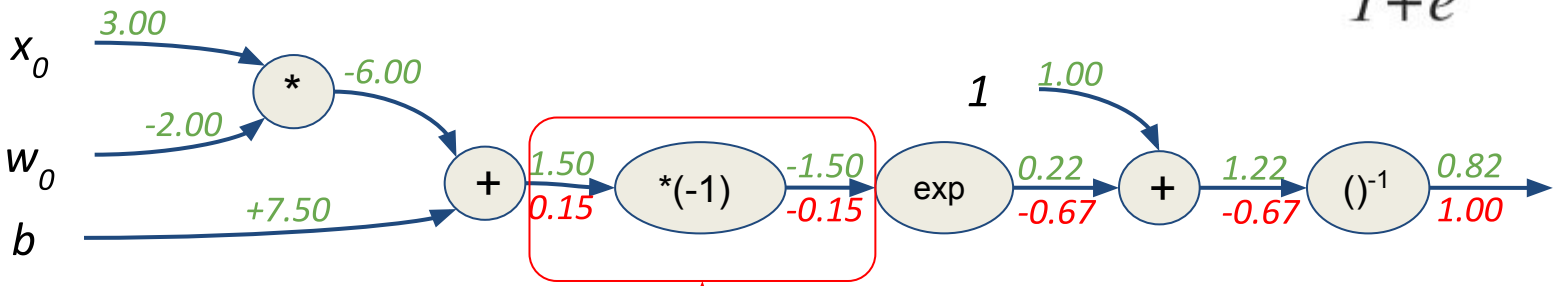
$f(x)$	$\frac{df(x)}{dx}$
e^x	e^x
ax	a
$1/x$	$-1/x^2$
$x+C$	1

© ZK

COLUMBIA | ENGINEERING
The Fu Foundation School of Engineering and Applied Science

Back-Propagation Example 2

$$f(w,x)=\frac{1}{1+e^{-(w_0x_0+b)}}$$



$f(x)$	$\frac{df(x)}{dx}$
e^x	e^x
ax	a
$1/x$	$-1/x^2$
$x+C$	1

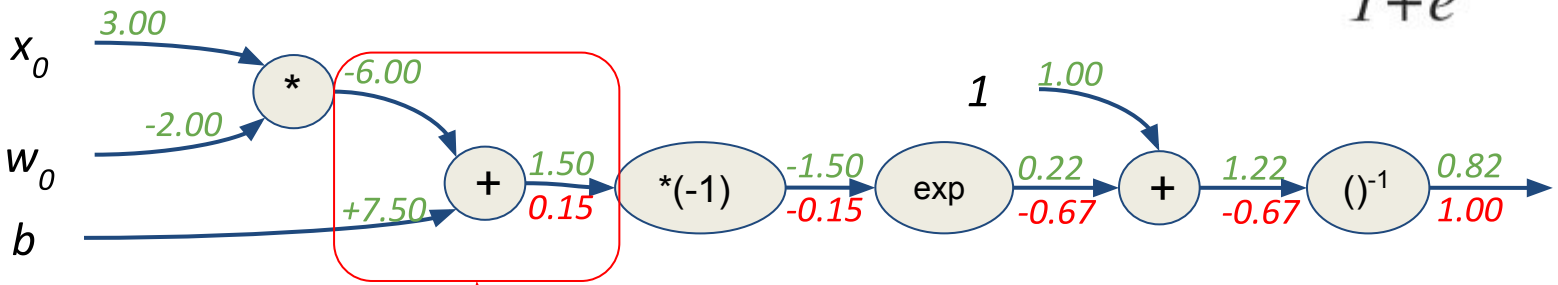
$(-1)(-0.15)=0.15$

© ZK

COLUMBIA | ENGINEERING
The Fu Foundation School of Engineering and Applied Science

Back-Propagation Example 2

$$f(w,x)=\frac{1}{1+e^{-(w_0x_0+b)}}$$



$localGradient * upstreamGradient$

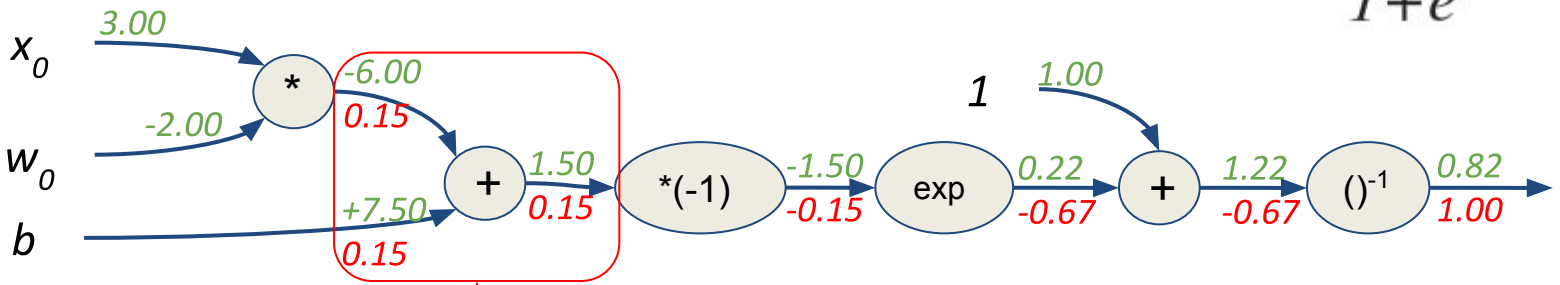
$f(x)$	$\frac{df(x)}{dx}$
e^x	e^x
ax	a
$1/x$	$-1/x^2$
$x+C$	1

© ZK

COLUMBIA | ENGINEERING
The Fu Foundation School of Engineering and Applied Science

Back-Propagation Example 2

$$f(w,x) = \frac{1}{1+e^{-(w_0 x_0 + b)}}$$



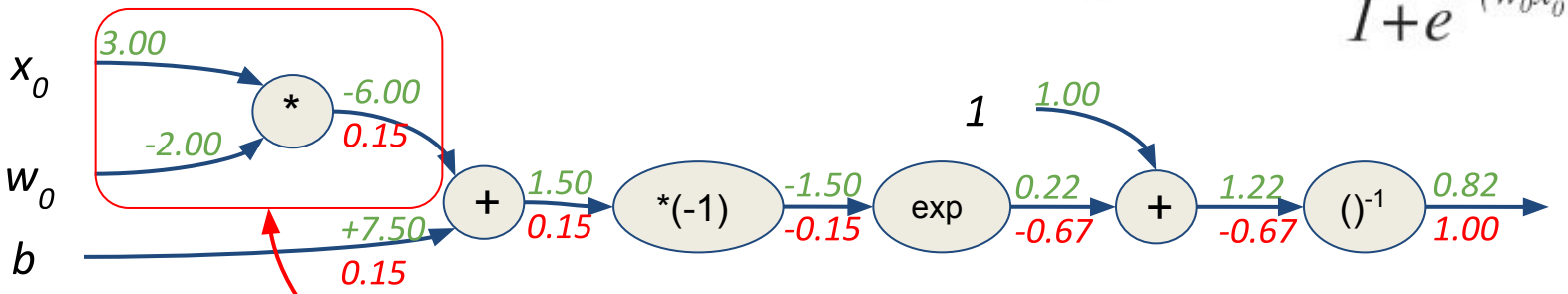
localGradient∗upstreamGradient
1*0.15

$f(x)$	$\frac{df(x)}{dx}$
e^x	e^x
ax	a
$1/x$	$-1/x^2$
$x+C$	1

© ZK
COLUMBIA | ENGINEERING
The Fu Foundation School of Engineering and Applied Science

Back-Propagation Example 2

$$f(w,x) = \frac{1}{1+e^{-(w_0 x_0 + b)}}$$



localGradient∗upstreamGradient

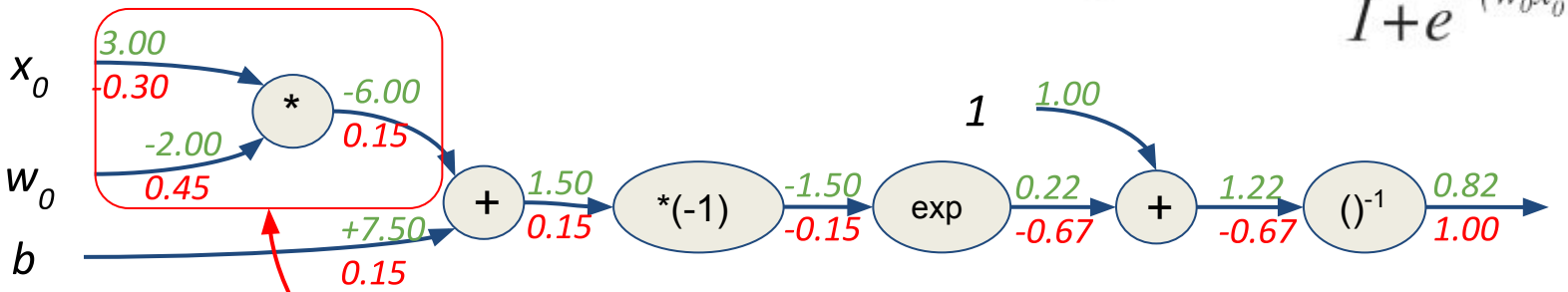
$f(x)$	$\frac{df(x)}{dx}$
e^x	e^x
ax	a
$1/x$	$-1/x^2$
$x+C$	1

© ZK

COLUMBIA | ENGINEERING
The Fu Foundation School of Engineering and Applied Science

Back-Propagation Example 2

$$f(w,x)=\frac{1}{1+e^{-(w_0x_0+b)}}$$



$f(x)$	$\frac{df(x)}{dx}$
e^x	e^x
ax	a
$1/x$	$-1/x^2$
$x+C$	1

$localGradient * upstreamGradient$

x_0: $(-2)*0.15=-0.30$

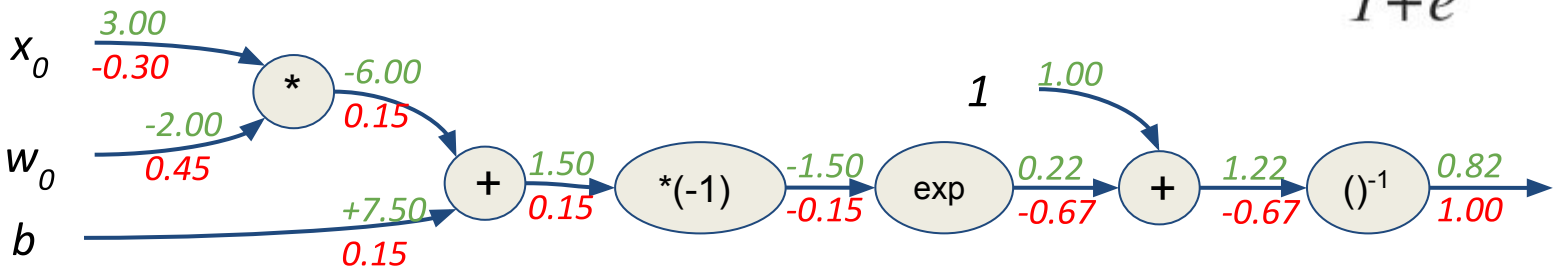
w_0: $(3)*0.15=0.45$

© ZK

COLUMBIA | ENGINEERING
The Fu Foundation School of Engineering and Applied Science

Back-Propagation Example 2

$$f(w,x) = \frac{1}{1+e^{-(w_0 x_0 + b)}}$$



$f(x)$	$\frac{df(x)}{dx}$
e^x	e^x
ax	a
$1/x$	$-1/x^2$
$x+C$	1

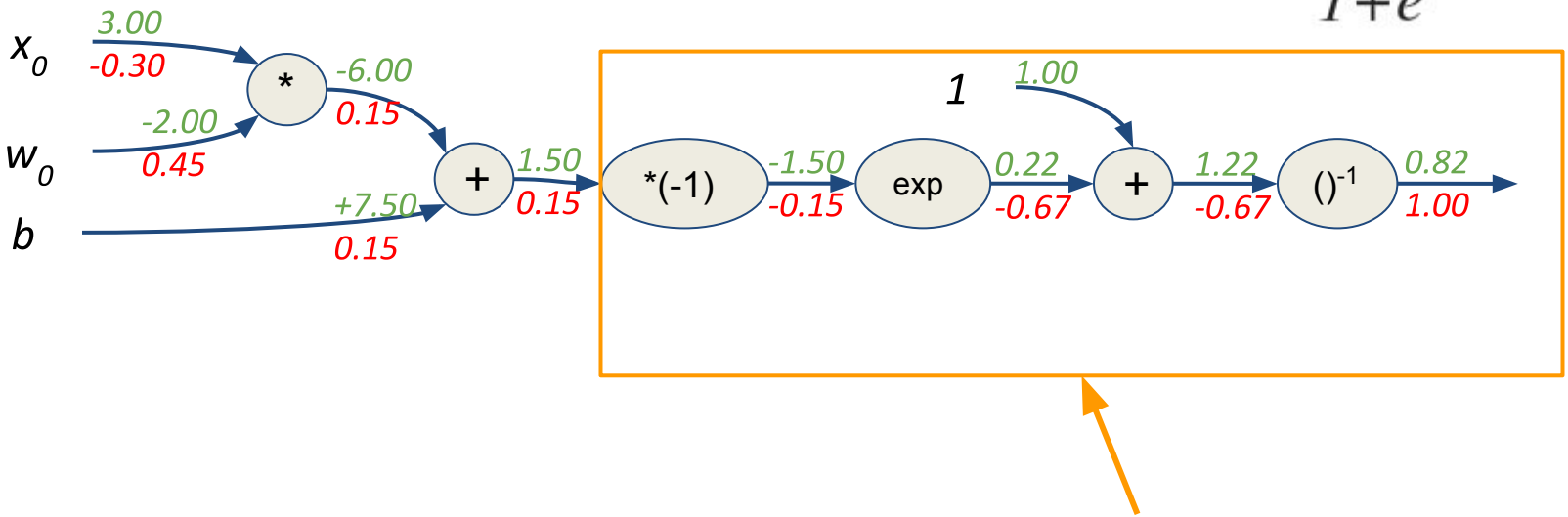
Ready to go!

© ZK

COLUMBIA | ENGINEERING
The Fu Foundation School of Engineering and Applied Science

Back-Propagation Sigmoid

$$f(w,x) = \frac{1}{1+e^{-(w_0 x_0 + b)}}$$



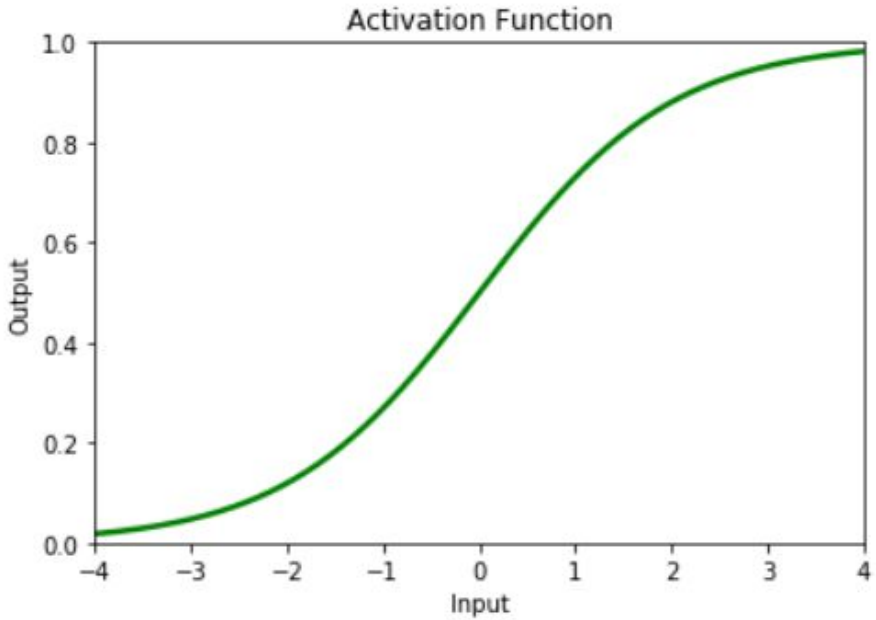
This is the sigmoid activation function.

© ZK

COLUMBIA | ENGINEERING
The Fu Foundation School of Engineering and Applied Science

Activation Function- Sigmoid

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



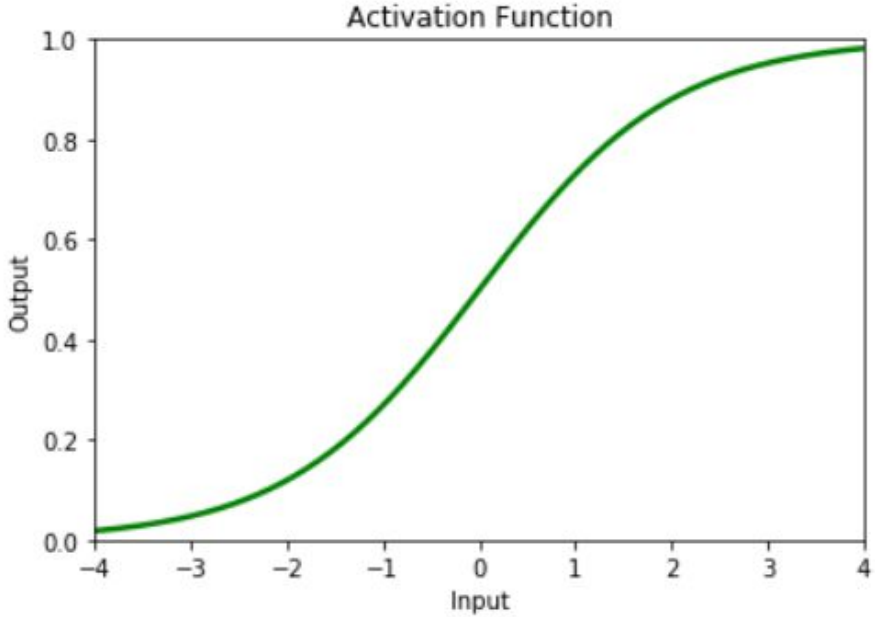
$$\frac{df(x)}{dx} = \frac{e^{-x}}{(1+e^{-x})^2}$$
$$= \left(\frac{1+e^{-x}-1}{1+e^{-x}}\right)\left(\frac{1}{1+e^{-x}}\right)$$
$$= f(x)(1-f(x))$$

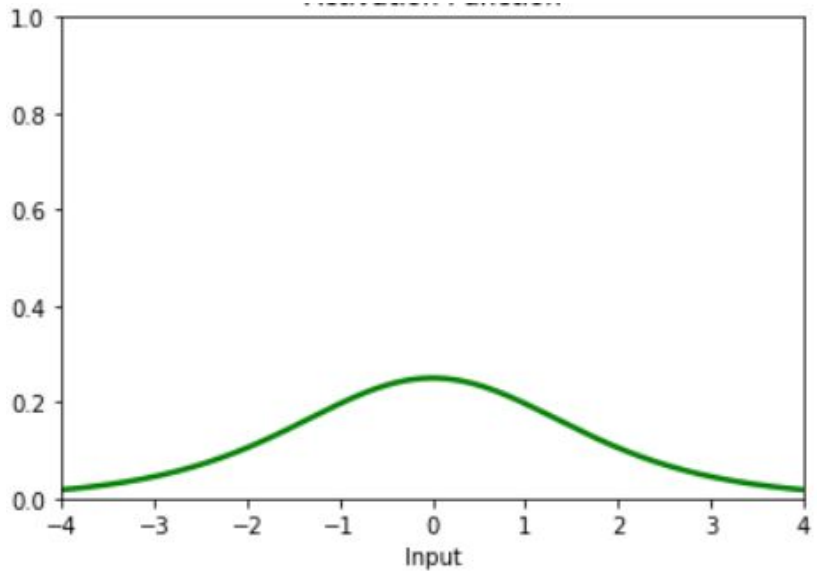
© ZK
COLUMBIA | ENGINEERING
The Fu Foundation School of Engineering and Applied Science

Activation Function- Sigmoid

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

$$f'(x)=f(x)(1-f(x)) \quad =\frac{e^{-x}}{(1+e^{-x})^2}$$



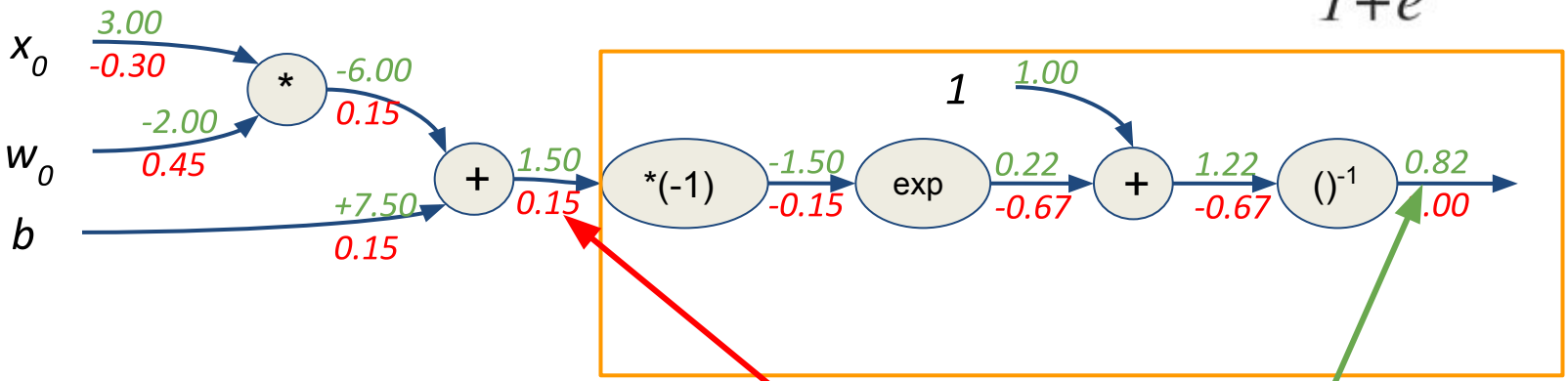


© ZK

COLUMBIA | ENGINEERING
The Fu Foundation School of Engineering and Applied Science

Back-Propagation - Sigmoid

$$f(w,x) = \frac{1}{1+e^{-(w_0 x_0 + b)}}$$



0.1476 ~0.15 = 0.82 * (1-0.82)

$$f(x)(1 - f(x))$$

© ZK

COLUMBIA | ENGINEERING
The Fu Foundation School of Engineering and Applied Science

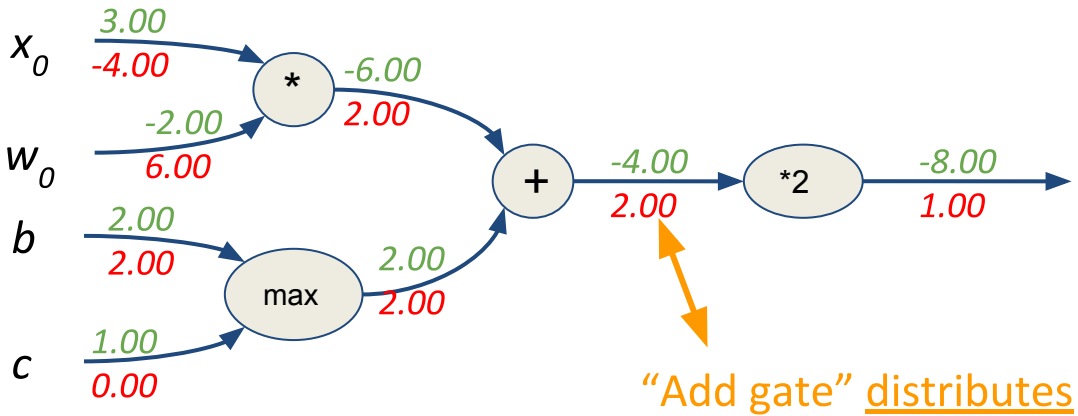
Back-Propagation

Gate Actions: Summary

© ZK

COLUMBIA | ENGINEERING
The Fu Foundation School of Engineering and Applied Science

Back-Propagation - Gate Action

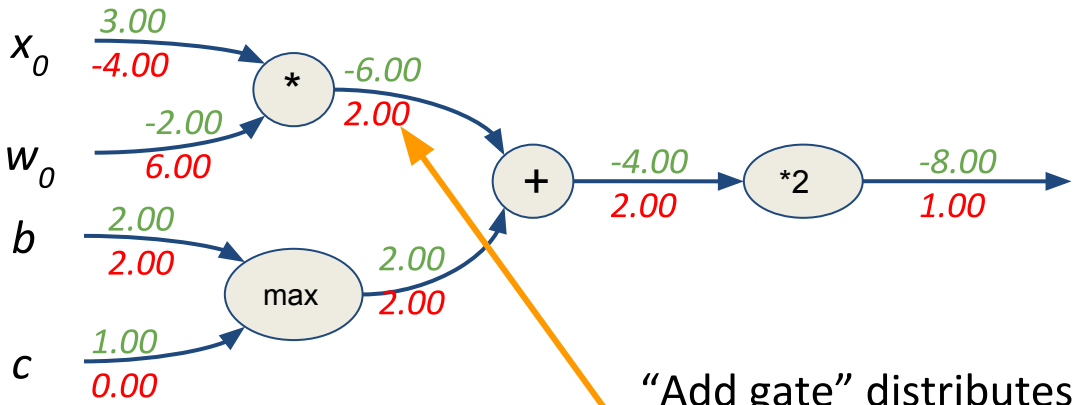


"Add gate" distributes (the same) gradient backwards

© ZK

COLUMBIA | ENGINEERING
The Fu Foundation School of Engineering and Applied Science

Back-Propagation - Gate Action



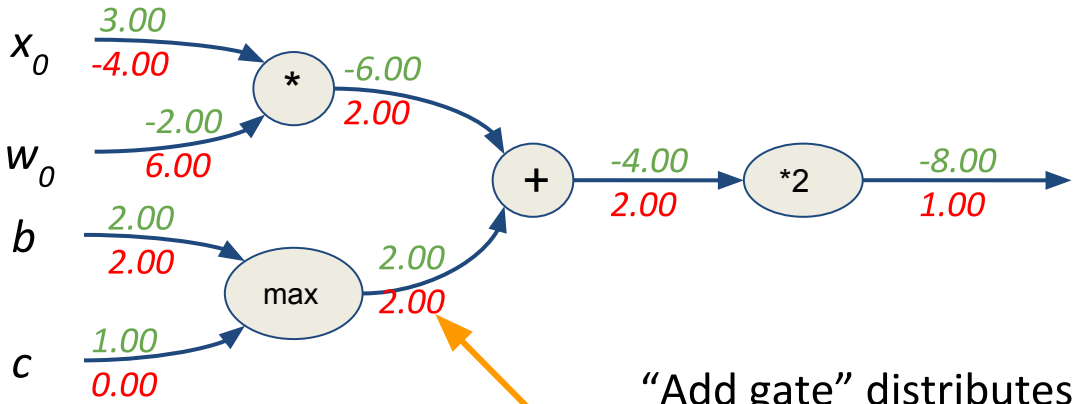
"Add gate" distributes (the same) gradient backwards

"Multiplier gate" (cross-) switches the gradient

© ZK

COLUMBIA | ENGINEERING
The Fu Foundation School of Engineering and Applied Science

Back-Propagation: Gate Action on Gradients



"Add gate" distributes (the same) gradient

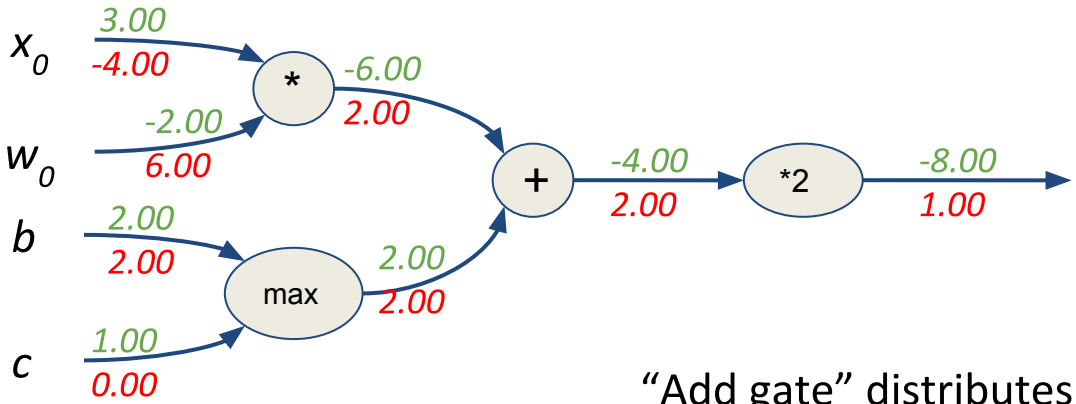
"Multiplier gate" (cross-) switches the gradient

"Max gate" routes the gradient (towards the max input)

© ZK

COLUMBIA | ENGINEERING
The Fu Foundation School of Engineering and Applied Science

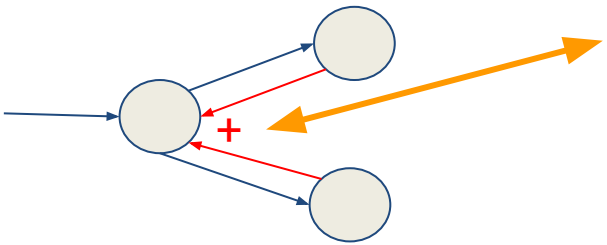
Back-Propagation: Gate Action on Gradients



"Add gate" distributes (the same) gradient

"Multiplier gate" (cross-) switches the gradient

"Max gate" routes the gradient (towards the max input)

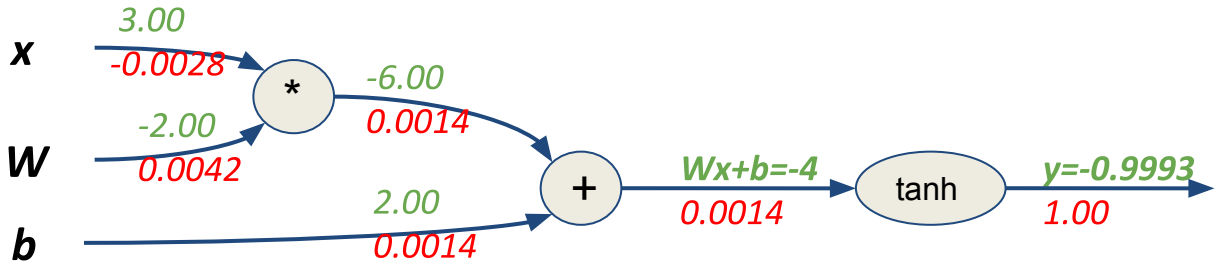


At "branches", gradients add

© ZK

COLUMBIA | ENGINEERING
The Fu Foundation School of Engineering and Applied Science

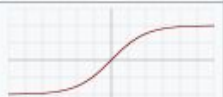
Back-Propagation: Gate Action on Gradients
tanh activation function



$1 - 0.9993^2 = 0.00139951 \sim 0.0014$

"Add gate" <u>distributes</u> (the same) gradient

$$f(z) = \tanh(z) = \frac{e^z - e^{-z}}{e^z + e^{-z}},$$

TanH		$f(x) = \tanh(x) = \frac{2}{1+e^{-2x}} - 1$	$f'(x) = 1 - f(x)^2$	$(-1, 1)$

© ZK
COLUMBIA | ENGINEERING
The Fu Foundation School of Engineering and Applied Science

(Another) Backprop Example

https://mattmazur.com/2015/03/17/a-step-by-step-backpropagation-example/

© ZK

COLUMBIA | ENGINEERING
The Fu Foundation School of Engineering and Applied Science

Vectorized Notation

Forward and Backward Computational Examples

© ZK

© ZK

Vectorized Operations

Inputs and outputs become

vectors and matrices

Partial derivatives become

Jacobians

© ZK

COLUMBIA | ENGINEERING
The Fu Foundation School of Engineering and Applied Science

J versus *J*

Cost Function *J*

$$J(\theta) = \lambda||\omega||^2 + \frac{1}{n}\sum_{t=1}^{n}||\mathbf{y}^t - (\mathbf{b} + \mathbf{V}\text{sigmoid}(\mathbf{c} + \mathbf{W}\boldsymbol{x}^t))||^2$$

is different from

© ZK

COLUMBIA | ENGINEERING
The Fu Foundation School of Engineering and Applied Science

J versus *J*

Cost Function *J*

$$J(\theta) = \lambda||\omega||^2 + \frac{1}{n}\sum_{t=1}^{n}||\mathrm{y}^t - (\mathrm{b} + \mathbf{V}\,\mathrm{sigmoid}(\mathrm{c} + \mathbf{W}x^t))||^2$$

is different from the **Jacobian matrix *J***

$$\mathbf{J} = \begin{bmatrix} \frac{\partial \mathbf{f}}{\partial x_1} & \cdots & \frac{\partial \mathbf{f}}{\partial x_n} \end{bmatrix} = \begin{bmatrix} \frac{\partial f_1}{\partial x_1} & \cdots & \frac{\partial f_1}{\partial x_n} \\ \vdots & \ddots & \vdots \\ \frac{\partial f_m}{\partial x_1} & \cdots & \frac{\partial f_m}{\partial x_n} \end{bmatrix}$$

or, component-wise:

$$\mathbf{J}_{ij} = \frac{\partial f_i}{\partial x_j}.$$

© ZK

COLUMBIA | ENGINEERING
The Fu Foundation School of Engineering and Applied Science

Vectorized Backpropagation Example
Square Norm for Matrix-Vector Product

$$\mathbf{x} = \begin{bmatrix} 0.1 \\ 0.2 \end{bmatrix}, \mathbf{W} = \begin{bmatrix} 0.3 & 0.4 \\ 0.5 & 0.6 \end{bmatrix}$$

$$f(\mathbf{x}, \mathbf{W}) = ||\mathbf{W}\mathbf{x}||^2 = \sum_{i=1}^{n} (\mathbf{W}\mathbf{x})_i^2$$

$$\mathbf{m} = \mathbf{W}\mathbf{x} = \begin{bmatrix} W_{1,1}x_1 + ... + W_{1,n}x_n \\ ... \\ W_{n,1}x_1 + ... + W_{n,n}x_n \end{bmatrix}$$

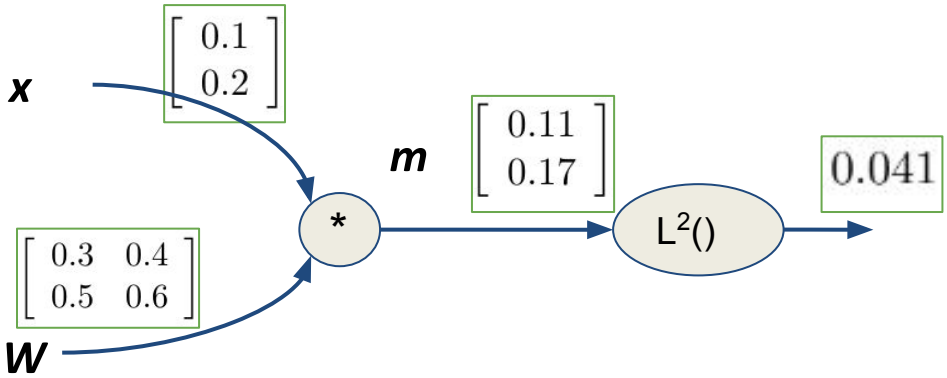
$$L_2(\mathbf{m}) = f(\mathbf{m}) = ||\mathbf{m}||^2 = m_1^2 + ... + m_n^2$$

$$L_2(\mathbf{m}) = \left|\left| \begin{bmatrix} 0.11 \\ 0.17 \end{bmatrix} \right|\right|^2 = 0.11^2 + 0.17^2 = 0.041$$

© ZK

COLUMBIA | ENGINEERING
The Fu Foundation School of Engineering and Applied Science

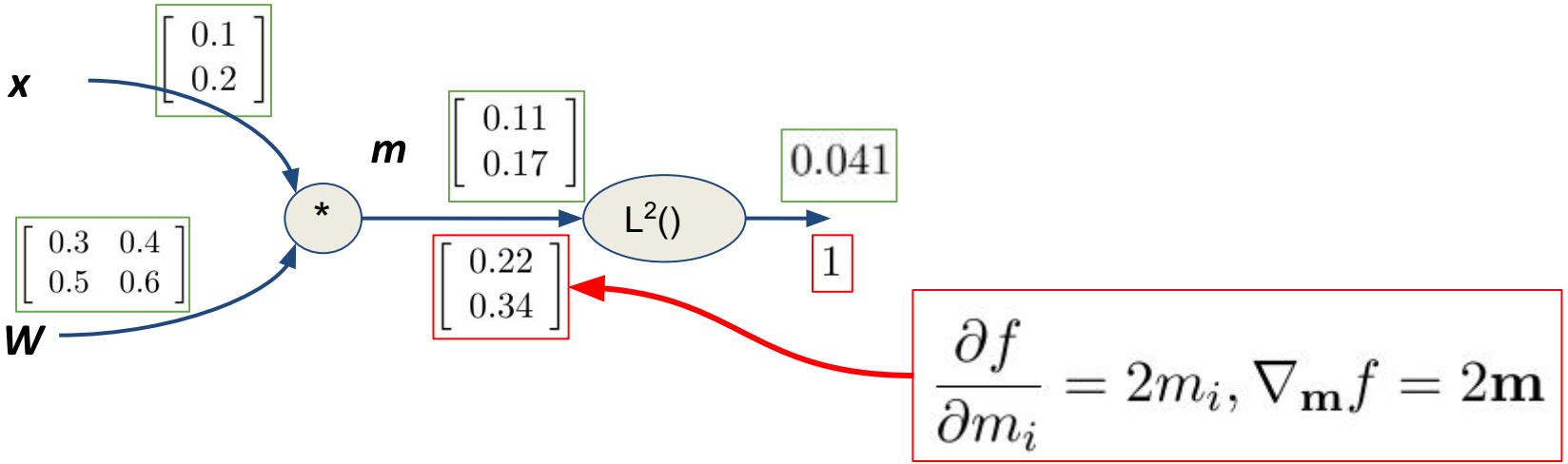
Vectorized Back-Propagation Example



© ZK

COLUMBIA | ENGINEERING
The Fu Foundation School of Engineering and Applied Science

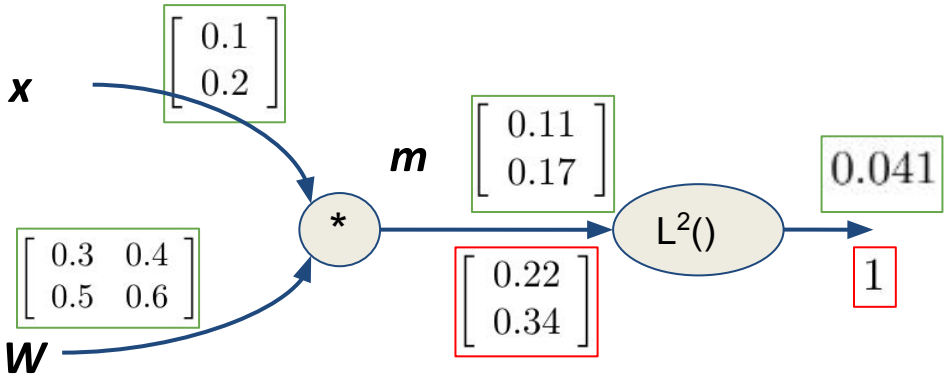
Vectorized Back-Propagation Example



© ZK

COLUMBIA | ENGINEERING
The Fu Foundation School of Engineering and Applied Science

Vectorized Back-Propagation Example



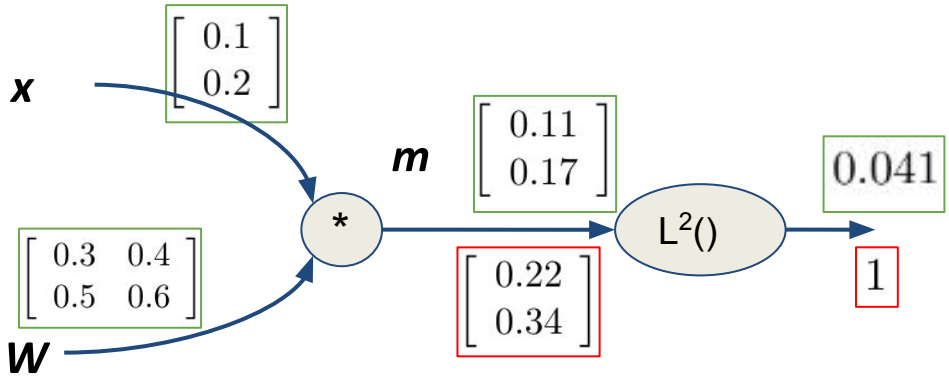
$$\frac{\partial f}{\partial m_i} = 2m_i, \nabla_{\mathbf{m}} f = 2\mathbf{m}$$

$$\frac{\partial m_k}{\partial W_{i,j}} = \mathbf{1}_{k=i} x_j$$

$$\frac{\partial f}{\partial W_{i,j}} = \sum_k \frac{\partial f}{\partial m_k} \frac{\partial m_k}{\partial W_{i,j}} = \sum_k (2m_k)(\mathbf{1}_{k=i} x_j) = 2m_i x_j$$

COLUMBIA | ENGINEERING
The Fu Foundation School of Engineering and Applied Science

Vectorized Back-Propagation Example



$$\nabla_{\mathbf{W}} f = 2\mathbf{m} \cdot \mathbf{x}^T$$

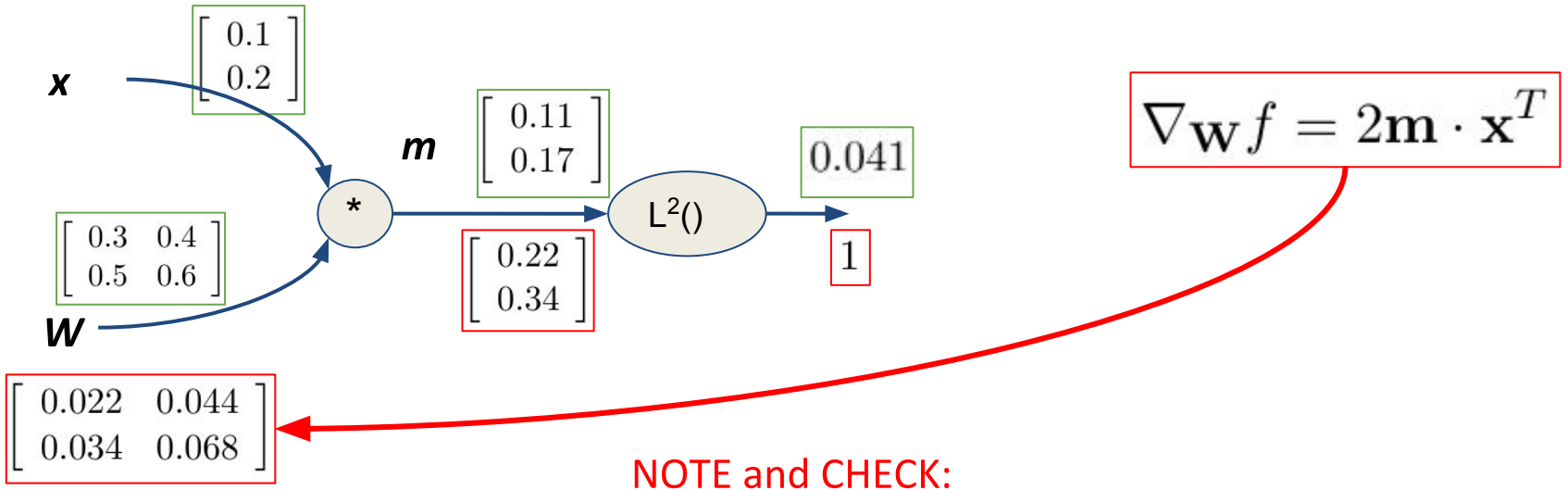
$$\begin{bmatrix} 0.022 & 0.044 \\ 0.034 & 0.068 \end{bmatrix}$$

$$\frac{\partial m_k}{\partial W_{i,j}} = \mathbf{1}_{k=i} x_j$$

$$\frac{\partial f}{\partial W_{i,j}} = \sum_k \frac{\partial f}{\partial m_k} \frac{\partial m_k}{\partial W_{i,j}} = \sum_k (2m_k)(\mathbf{1}_{k=i} x_j) = 2m_i x_j$$

COLUMBIA | ENGINEERING
The Fu Foundation School of Engineering and Applied Science

Vectorized Back-Propagation Example



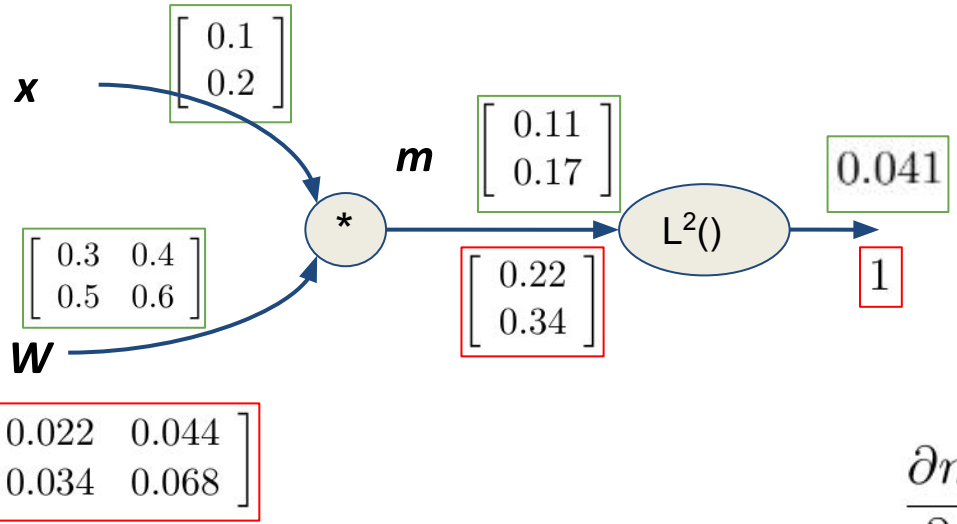
NOTE and CHECK:

Dimensions of the variable and the gradient of the variable have to be the same!

© ZK

COLUMBIA | ENGINEERING
The Fu Foundation School of Engineering and Applied Science

Vectorized Back-Propagation Example



$$\mathbf{m} = \mathbf{W}\mathbf{x} = \begin{bmatrix} W_{1,1}x_1 + ... + W_{1,n}x_n \\ ... \\ W_{n,1}x_1 + ... + W_{n,n}x_n \end{bmatrix}$$

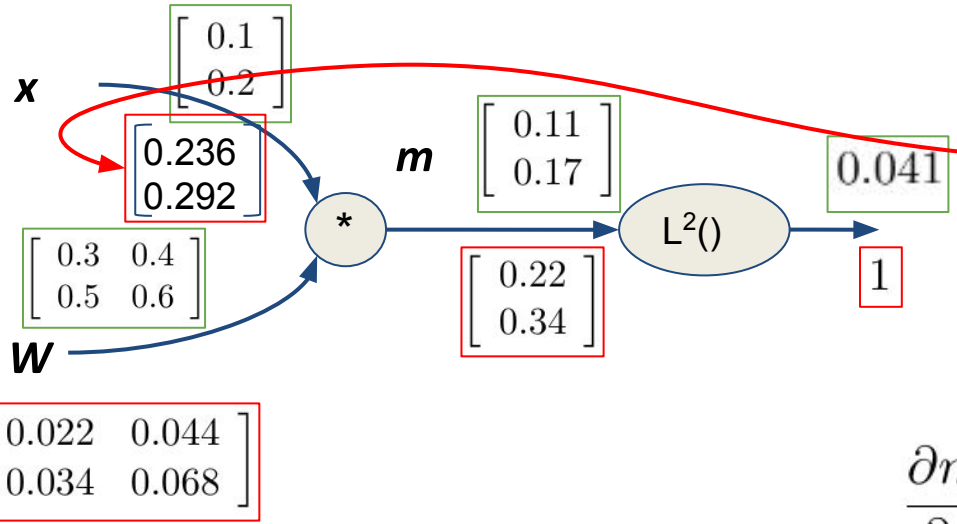
$$\frac{\partial m_k}{\partial x_i} = W_{k,i}$$

$$\frac{\partial f}{\partial x_i} = \sum_k \frac{\partial f}{\partial m_k}\frac{\partial m_k}{\partial x_i} = \sum_k (2m_k)(W_{k,i})$$

© ZK

COLUMBIA | ENGINEERING
The Fu Foundation School of Engineering and Applied Science

Vectorized Back-Propagation Example



$$\nabla_{\mathbf{x}} f = 2\mathbf{W}^T \cdot \mathbf{m}$$

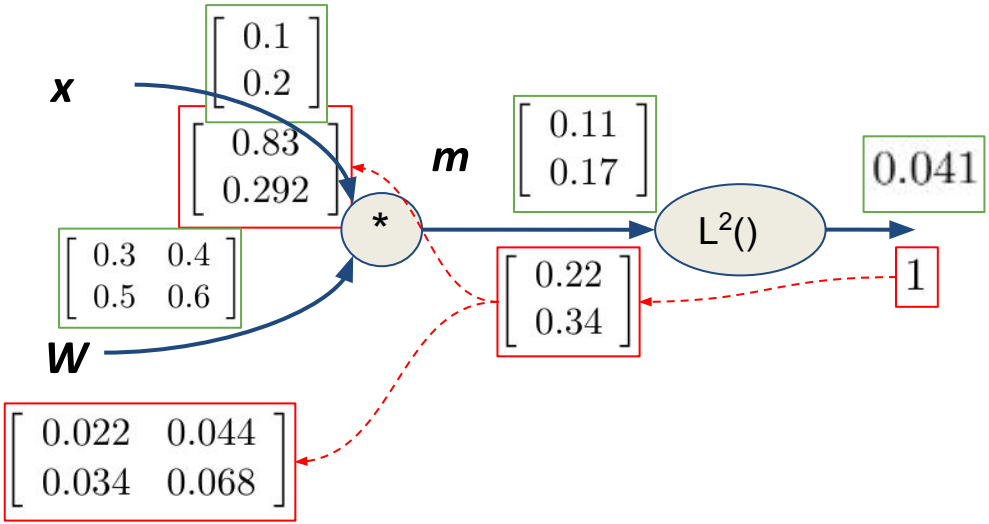
0.3*0.22 + 0.5*0.34 = 0.066 + 0.17 = 0.236
0.4*0.22 + 0.6*0.34 = 0.088 + 0.204 = 0.292

$$\frac{\partial m_k}{\partial x_i} = W_{k,i}$$

$$\frac{\partial f}{\partial x_i} = \sum_k \frac{\partial f}{\partial m_k}\frac{\partial m_k}{\partial x_i} = \sum_k (2m_k)(W_{k,i})$$

© ZK
COLUMBIA | ENGINEERING
The Fu Foundation School of Engineering and Applied Science

Vectorized Back-Propagation Example



© ZK

COLUMBIA | ENGINEERING
The Fu Foundation School of Engineering and Applied Science

Backup Slides

Various

© ZK